DOCTOR OF PHILOSOPHY

VLSI implementation of a spectral estimator for use with pulsed ultrasonic blood flow detectors.

Bellis, Stephen John

Award date:
1996

Awarding institution:
Bangor University

VLSI Implementation of a Spectral Estimator for use with Pulsed Ultrasonic Blood Flow Detectors

Stephen John Bellis

Thesis Submitted in Candidature for the Degree of

Doctor of Philosophy

July 1996

School of Electronic Engineering and Computer Systems

University of Wales, Bangor

United Kingdom

To my family

Acknowledgements

I would like to express my sincere thanks to Mr. Peter Fish for his supervision and motivation throughout the course. I am also deeply indebted to Dr. Liam Marnane for his support, invaluable advice, motivation and guidance despite the distance. Thanks Liam and Angela for your friendship and hospitality upon my visits to Cork.

Thank you to Mr. David Everett for initially stimulating my interest in the area of computer arithmetic hardware and all of his help during my time in Bangor. I would also like to thank all the other staff members of S.E.E.C.S. with whom I have had contact and to the other members of the Medical Physics, Computer and Control Systems groups for their helpful advice.

I am very grateful to Mum, Dad, Sandra, David, Cecelia, Nain and the rest of my family for their love, support and encouragement throughout.

Cheers to all my friends and sports colleagues who have made Bangor so enjoyable.

Τέλος, θα ήθελα να ευχαριστήσω όλους τους φίλους μου από την Ελλάδα. Χαίρομαι πολύ που είχα την ευκαιρία να σας γνωρίσω.

Summary

The focus of this thesis is on the design and selection of systolic architectures for ASIC implementation of the real-time digital signal processing task of Modified Covariance spectral estimation. When used with pulsed Doppler ultrasound blood flow detectors, the Modified Covariance spectral estimator offers increased sensitivity in the detection of arterial disease over conventional Fourier transform based methods.

The systolic model of computation is considered because through pipelining and parallel processing high levels of concurrency can be achieved to attain the necessary throughput for real-time operation. Systolic arrays of simple processing units are also well suited for implementation on VLSI. The versatility of the design of systolic arrays using the rigorous data dependence graph methodology is demonstrated throughout the thesis by application to all sections of the spectral estimator design at both word and bit levels.

Systolic array design for the model order 4 Modified Covariance spectral estimator, known to offer accurate estimation of blood flow mean velocity and disturbance at an acceptable computational burden, is initially discussed. A variety of problem size dependent systolic arrays for real-time implementation of the fixed model order spectral estimator are designed using data dependence graph mapping methods. Optimal designs are chosen by comparison of hardware, communication and control costs, as well as efficiency, timing, data flow and accuracy considerations. A cost/benefit analysis, based on results from structural simulation of the arrays, allows the most suitable word-lengths to be chosen.

Problem size independent systolic arrays are then discussed as means of coping with the huge increases in computational burden for a Modified Covariance spectral estimator which is programmable up to high model orders. This type of array can be used to reduce the number of PEs and increase efficiency when compared to the problem size dependent arrays and the research culminates in the proposal of a novel spiral systolic array for Cholesky decomposition.

Contents

Chapter 1

Introduction

1.1 Overview

This thesis investigates the feasibility of hardware implementation of a spectral estimator for use with instruments which utilise ultrasound and the Doppler effect for the assessment of blood flow.

Pulsed Doppler ultrasound detectors [1] are in wide use as non-invasive instruments for the diagnosis of arterial disease [2], providing quantitative monitoring of arterial blood-flow within a specific resolution cell. Backscattered ultrasound, from red blood cells moving within the resolution cell, contains Doppler shift frequencies which are detected and encoded into the envelope of a Doppler time signal by the pulsed Doppler instrumentation. The Doppler signal is random with a periodically time-varying power spectrum in the audio frequency range. The power spectrum gives an indication of the range of Doppler frequencies and is related to the profile of velocities within the resolution cell of the Doppler instrument [2]. The Doppler signal is non-stationary as the blood velocities are constantly changing due to the rhythmic pumping by the heart and it is also cyclo-stationary since arterial blood flow is pulsatile over the cardiac cycle, providing the cardiac period remains steady. However, the signal may be considered

as wide-sense-stationary over short periods of time, usually 2-20ms, where its power spectrum remains approximately constant [2].

The power spectral density of the Doppler signal contains useful diagnostic information as, over the periods when wide-sense-stationarity is assumed, the spectral mean frequency and bandwidth are respectively proportional to the spatial blood flow mean velocity and the range of velocities of the red blood cells in the resolution cell. In the region of a stenosis, mean flow velocity can increase and flow is disturbed, causing the range of flow velocities to widen. Therefore, correspondent increases in Doppler power spectrum mean frequency and the degree of spectral broadening allow the severity of arterial disease to be detected from the Doppler signal output of a pulsed ultrasonic blow flow instrument [3].

The process of obtaining the power spectral density (PSD) from the Doppler signal is termed spectral estimation. The conventional method of spectral estimation is to apply the short term fast Fourier transform (STFFT) [4] on sequential or overlapping windows of data producing a series of individual power spectra over time. Display of the spectra is usually in the form of a real-time spectrogram an example of which is shown in figure 1.1. Over the three cardiac cycles shown in the figure, narrow vertical sections represent individual spectra where the signal is assumed to be wide-sense-stationary over short time durations. The darkness of the grey scale is used to show the power of the Doppler frequency components.

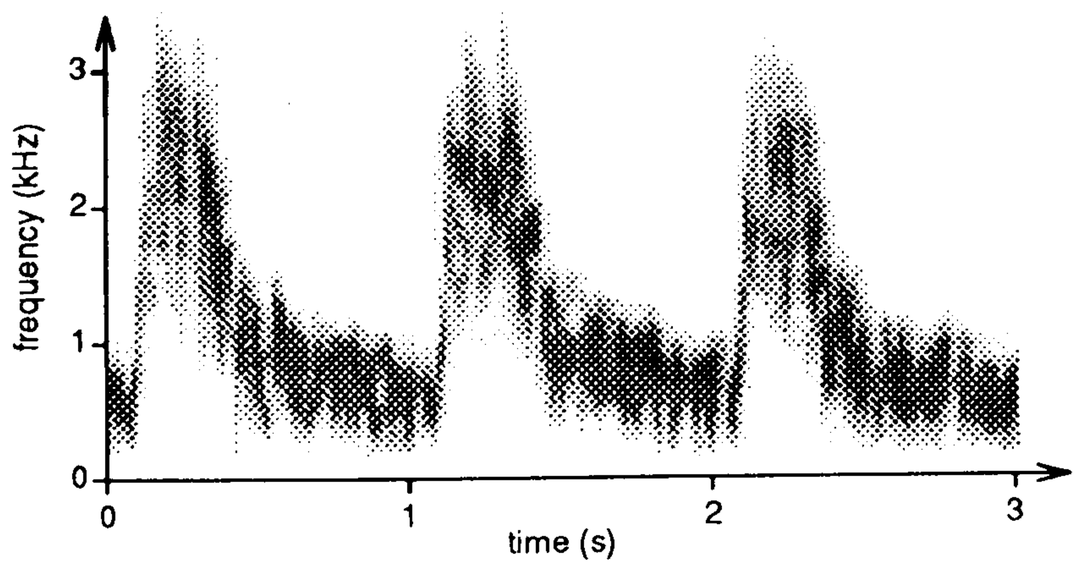


Figure 1.1: Spectrogram of the Doppler signals from a PDU instrument used on the common carotid artery.

A shortcoming of the of the STFFT method is that signal properties outside the data window are not taken into account leading to poor spectral estimates. The signal is assumed to be zero outside the window of interest, the sudden cut-off leading to spectral leakage [5] which is displayed as spurious peaks in the power spectrum. In attempts to suppress this effect the data can be multiplied by weighted window functions to gradually increase and decrease the sample amplitudes at the beginning and end of the window to smooth out the spurious peaks but this has the disadvantage of reducing resolution [4][5][6]. The frequency resolution of the STFFT is approximately the reciprocal of the data window length [7] and so longer windows can be used to improve the resolution. However, as the length of the data window is increased the assumption that data is stationary over the window duration is more likely to become invalid [8].

The disadvantages associated with the conventional method led to the investigation of parametric methods which makes use of a priori knowledge of the signal characteristics outside the region of interest [9]. There are three main branches of parametric spectral estimation, autoregressive (AR), moving average (MA) and autoregressive moving average (ARMA). All these methods entail the derivation of a set of filter parameters, such that a model of the Doppler signal is produced when the filter is driven by a white noise source. The PSD of the Doppler signal is given by the response of the filter, obtained from the Fourier transform of the filter parameters.

Many different models exist within each branch of parametric spectral estimation and various criteria can be used for selection of the best method [9][10][11][12][13]. Selection is further complicated as an appropriate model order must be chosen for the estimator. The cost/benefit selection procedure proposed by Ruano and Fish [10], where benefit is inversely proportional to estimation error and cost is computational complexity, led to the selection of the model order 4 AR Modified Covariance method of spectral estimation when estimating mean frequency and

bandwidth, which involves the zeroth, first and second spectral moments.

The price paid for improved estimation accuracy when using parametric methods is the increased computational complexity of such a scheme, the computational burden increasing with model order. However, the cost/benefit selection criterion [10] deems that the improvement in estimation accuracy of the relatively low model order 4 Modified Covariance estimator is justifiable over the use of the less expensive STFFT used with both Boxcar and Hanning windows.

Parallel processing techniques can be employed to attain the throughput necessary for the required real-time results display [14]. With the use of these parallel partitioning schemes the spectral estimator can be implemented in a transputer based system [15]. This thesis takes the implementation a step further considering the feasibility of an application specific integrated circuit (ASIC) approach.

Systolic arrays [16] are identified as suitable architectures for the development of a hardware solution of the Modified Covariance spectral estimator. They are used in many real-time digital signal processing applications where high computational throughput is a major requirement [17][18][19][20][21][22] and have a rigorous design methodology [23]. Systolic arrays consist of highly parallel arrays of pipelined processing elements which are regularly interconnected and whose function is usually a simple arithmetic operation such as multiply and accumulate. They exploit localised communication where, upon a global clock cycle, data is transferred to a neighbouring processor. The way in which data is transmitted through these arrays can be compared to the pumping of blood which pulses through the body caused by the systole, which is the rhythmic beating of the heart - hence the name systolic arrays [17]. In very large scale integration (VLSI) devices it is desirable to restrict communication as much as possible as this takes up large area, increases power supply load and slows down operation [16]. The localised communication and regularity of systolic arrays therefore makes them very amenable to VLSI implementation [16][24].

1.2 Goals

- Application of the systolic model of computation to the real-time, model order 4, AR Modified Covariance method.

- Selection of the most efficient architectures with a view to ASIC implementation on a VLSI platform, thereby increasing the portability of such a system and minimising its power requirements.

- To derive a methodology for the design and selection of optimally partitioned programmable systolic arrays for the Modified Covariance estimator for model order 4, used for measurement of mean frequency and bandwidth, and also for higher model orders up to 30 which are more suitable for analysis of other blood flow properties.

1.3 Outline of Thesis

The background of the project is initially presented, describing the use of pulsed Doppler ultrasound instrumentation for non-invasive detection of blood flow, the formation of the Doppler signal and the affects of arterial stenosis on the PSD of this signal [2]. Conventional and modern spectral estimation methods are then introduced [25] and the research which led to the selection of the Modified Covariance spectral estimator with model order 4 for estimation of spectral mean frequency and bandwidth, is reviewed [10]. To gain maximum speedup and increase portability, use of the systolic model of computation with a VLSI implementation platform [16], over MIMD models, previously used in the parallelisation of the spectral estimator onto a transputer platform [26], is proposed. The data dependence graph (DDG) method [16], used for the design of systolic architectures, is introduced with regard to the simple white noise variance calculation which is involved with the Modified Covariance method and previously proposed systolic arrays pertinent to this overall application are discussed.

Expanding on the DDG design methodology, the mapping of the fixed model order 4 Modified Covariance spectral estimator onto problem size dependent systolic architectures is explored. Two main sections of the estimator are considered. The first involves a convolution type calculation for the derivation of covariance matrix elements and four systolic arrays formed from different DDGs are compared with regard their cost, communication, control and efficiency in finding an optimal solution. The second section considers decomposition techniques [27] which are used in the calculation of the Modified Covariance filter parameters. Two optimal systolic arrays, one of which requires the undesirable square-root operation, are selected from a range, this time produced by considering different projections of the DDGs from several algorithms rather than a variety of different DDGs for a single algorithm as in the covariance matrix element calculation. A cost/benefit analysis is developed to select the best method and its word-length.

The design of problem size independent systolic arrays [28] for mapping higher model order Modified Covariance spectral estimators, which may be required for monitoring different properties of blood flow requiring an estimation of higher spectral moments, is considered in the latter part of the thesis. The array selected for the model order 4 covariance matrix calculation is developed for use as a programmable model order device and a range of solutions result from varying partition size. DDG partitioning and reindexing techniques [22] are again employed to produce a series of spiral systolic arrays for the LU decomposition problem, demonstrating the superiority of this mapping method for design optimisation by comparison with another systolic array solution designed using an alternative scheme [28]. This design procedure is modified to produce a novel array for Cholesky decomposition. For each of the three types of programmable array guidelines are provided for selection of the best partitions for the desired range of programmability when the processor performance is known.

The thesis concludes by considering the feasibility of systolic array implementation and suggestions for further research possibilities are made.

1.4 Contribution of Research

The MIMD approach for transputer implementation of the Modified Covariance spectral estimator fails to satisfy the computational demands required for real-time operation under certain conditions. This thesis demonstrates that the systolic model of computation implemented on a VLSI platform, can be used to design hardware which does provide the necessary real-time computational throughput. The following contributions are resultant from the research.

- Design of a number of efficient systolic architectures, which can be implemented very easily on a VLSI platform, for the calculation of the covariance matrix elements - the most computationally demanding aspect of the model order $p = 4$ spectral estimator [29]. A comparison results in the selection of a systolic array which partitions the multiplications from the accumulations.

- A comparison of a range of systolic arrays, for use in the calculation of the fixed model order $p = 4$ Modified Covariance filter parameters, formed by projecting DDG's for square-root-Cholesky, LU and LDL^T decomposition [27] in different directions. This results in some new systolic arrays and the other projections produce architectures which are topologically equivalent to previously proposed designs. Modifications to existing systolic arrays for Cholesky and LU decomposition, leading to reduced hardware cost and to facilitate on-the-fly data retrieval, are proposed.

- A cost/benefit analysis is used to make an in depth comparison of the square root $[1, 1, 1]$ Cholesky and non-square-root $[1, 0, 0]$ LU decomposition arrays, weighing the cost of ASIC implementation to the results accuracy for a range of word-lengths in the model order $p = 4$ Modified Covariance spectral estimator [30]. The cost function considers both hardware and communication for systolic array implementation strategies based upon fixed point arithmetic schemes, which utilise scale factors to gain an increased dynamic range. Benefit is treated as the inverse of the estimation error

which is caused by finite word-length rounding and averaged over a large set of typical signals. The results of the cost/benefit analysis demonstrate that the Cholesky decomposition systolic array with a word-length of 12 bits is optimal when estimating spectral mean frequency and bandwidth in the Modified Covariance model order $p = 4$ spectral estimator.

- Development of the partitioned multiply accumulate systolic array, previously selected for the fixed model order $p = 4$ matrix element calculation, for use in a programmable model order Modified Covariance spectral estimator. The number of PEs in this problem size independent systolic array design can be varied according to the problem specification, that is the range of model orders required and input data sampling frequencies, in order to utilise the full performance potential of the PEs. A method for optimisation of systolic array size for model orders up to $p = 30$, given the timing constraints imposed by system and PE specification, is presented.

- A spiral systolic array for arbitrary dimension LU decomposition is designed with the use of DDG methodology which clearly shows partitioning of the standard DDG, subproblem reindexing and chaining, thereby allowing the precise scheduling and allocation of all tasks. The DDG design demonstrates superiority over the non-graphical matrix transformation methods described by Navarro et al. [28] in which the subproblem interconnectivity is only given a vague treatment. The advantages of the DDG design are evident when comparing the array produced, which uses shift registers to reschedule data between subproblems, with that proposed by Navarro et al. where only the spiral data re-circulation within subproblems is considered. Data dependence requirements motivate a modified reindexing scheme for Cholesky decomposition which results in proposal of a novel spiral systolic array, which, when compared to the LU spiral systolic array, approximately halves the number of clock cycles required. Selection of optimal array sizes, based on the range of programmable model order, is also presented.

Chapter 2

Background

2.1 Introduction

This chapter further expands on the background information introduced in the first chapter, presenting a review of previous research material to provide firstly a clear understanding of the motivation behind the project and secondly an insight into digital signal processing architectures which could be used to implement real-time spectral estimation hardware.

Initially, pulsed Doppler ultrasound detectors and their role in the non-invasive measurement of blood flow is discussed. The formation of the Doppler signal, its power spectrum and the effects of arterial stenosis are described.

The second section, reviews conventional and modern methods of spectral estimation, in particular the research which led to the selection of the Modified Covariance method and its MIMD transputer implementations.

The last section introduces systolic arrays as a viable approach to hardware implementation of the Modified Covariance method. An overview of design approaches and previously proposed systolic array designs which are applicable to this area of real-time digital signal processing is provided.

2.2 Doppler Ultrasound

2.2.1 The Doppler Effect

The Austrian physicist Johann Christian Doppler (1803-1853) postulated that if an observer is moving relative to a wave source, then the frequency detected by the observer is different to that emitted from the source. When moving closer to the source the observed frequency is higher than that of the source while a lower frequency is detected when moving away. The early research which leads to the discovery of this Doppler effect is reviewed by White [31] and Jonkman [32]. Formally the Doppler shift frequency f_d is given by:

$$f_d = f_r - f_t \tag{2.1}$$

which is the difference between the received frequency f_r and transmitted frequency f_t. If the relative speed between the receiver and the source is v_d (note that either the source or the receiver could be moving for the Doppler effect to be observed) then for $v_d \ll c_u$:

$$f_d \approx \frac{v_d}{c_u}.f_t \tag{2.2}$$

where c_u is the speed of ultrasound.

2.2.2 Clinical Use of the Doppler Effect in Ultrasound

Use of the Doppler effect with ultrasound is made in clinical applications for the measurement of moving structures, most notably blood cells. Use of Doppler ultrasound is now common and widespread in flowmetry and imaging systems which can be used to detect disease in arteries such as the carotid plus those of the lower limb and renal regions, with applications in other fields including cardiology, obstetrics and tumour diagnosis [2][33][34][35]. Measurement of changes in blood flow velocity waveform properties caused by abnormalities which occur in the common carotid artery is of particular interest in this study.

The popularity of the use of Doppler ultrasound in clinical analysis stems from the safety of the method, provided that power recommendation standards are adhered to [36], in comparison with that of X-rays for example where the risk of exposing the patient during the scan must be weighed against the possible diagnostic benefit. Also, the Doppler ultrasound methods are non-invasive, meaning that there is no need to introduce any foreign substances or objects into the body thus eliminating any surgical puncturing, simplifying the scanning procedures to reduce consultation time and avoiding any further distress to the patient.

2.2.3 Doppler Ultrasound Reflection

The Doppler theory can be used to estimate the speed of blood cells within arterial flow by considering the shift frequencies in ultrasound reflected off these moving targets. A targeted blood cell initially acts as a moving receiver, but also upon reflection of the ultrasound beam back to a stationary detecting transducer, the blood cell also acts as travelling transmitter doubling the Doppler effect. If the ultrasound is reflected straight back along its transmission propagation path then the Doppler difference frequency f_d (2.1) detected back at the receiver is:

$$f_d \approx 2.\frac{v_d.cos\theta}{c_u}.f_t \tag{2.3}$$

where $v_d.cos\theta$ is the component the blood cell velocity in the direction of the ultrasound beam. The validity of expression (2.3) for the Doppler shift frequency holds only under ideal conditions where a monochromatic, infinitely wide ultrasound beam is uniformly transmitted to and reflected from a plane surface moving at a constant velocity. However, it is not realistic to assume that a blood cell produces a single shift frequency and in practice spectral broadening is observed due to the finite beam width causing amplitude fluctuations in the reflected echo, the variation in target cell velocity and the band of ultrasound frequencies transmitted [37]. Nevertheless, there is a strong relationship between the shift frequency and the target blood cell velocity, justifying the use of Doppler methods.

2.2.4 Ultrasonic Doppler Instrumentation

Continuous wave ultrasonic Doppler blood flow instruments, first demonstrated for detection of blood flow [38] in 1961, are limited as the range at which movement occurs cannot be distinguished making it difficult to separate signals from arteries and veins.

For analysis of blood flow in a blood vessel it is desirable to target a specific volume in the vicinity of the suspected occlusion and the introduction of the pulsed ultrasonic blood flow detector [1], the general block layout of which is shown in figure 2.1, makes this possible. In these systems a periodically transmitted ultrasound pulse is reflected from targets at various depths within the tissue and to select a particular region of the blood flow the received signal is gated at a specific time delay after initial transmission. The depth of the target volume can be controlled by the time delay assuming that the speed of the ultrasound through the tissue between the transducer and the target is known. The gated return echo signal is then demodulated by comparing its phase with that of a

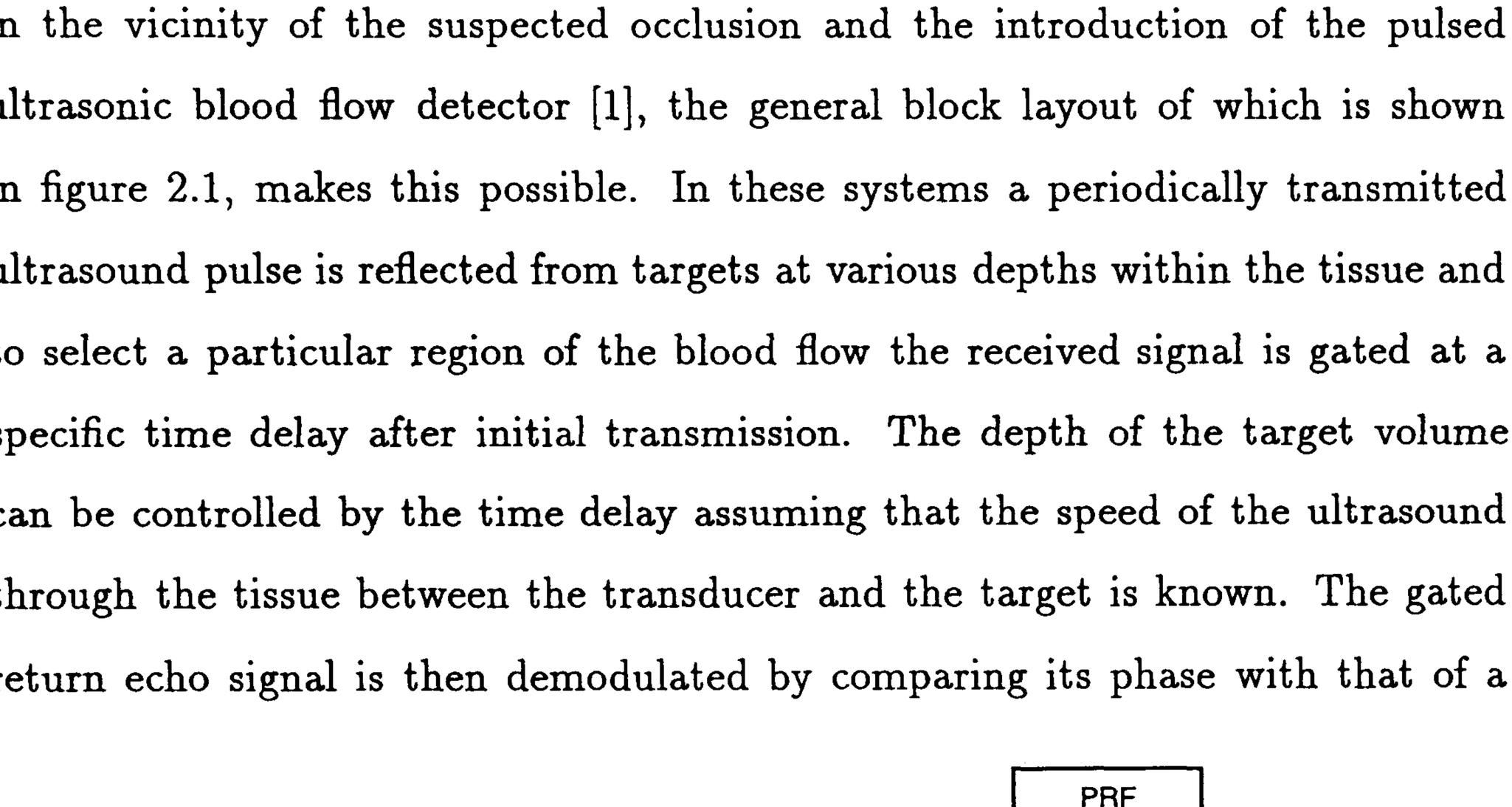


Figure 2.1: Functional block diagram of a PDU blood flowmeter.

signal from the master oscillator from which the transmission pulses are initially derived. Stationary objects at the selected depth return echoes whose phase difference remains constant with successive pulses producing a constant output. Moving targets give rise to echoes whose phase changes with time and after this coherent demodulation process the sample and hold circuitry produces a time series of pulses varying in amplitude at the Doppler shift frequencies. The output signal is referred to as the Doppler signal.

Since the amplitude of the return echoes decreases with target depth, there comes a point when the return echoes become masked by the electronic noise of the system. The depth of penetration is the maximum depth at which the return echo is just distinguishable above the noise threshold [39]. The time between successive transmission pulses must be greater than the transit time of the transmitted pulse to the maximum depth of penetration and back to the transducer, to avoid confusion as to which echo relates to which transmission pulse [39] The maximum pulse repetition frequency (PRF) is therefore constrained and, due to the Nyquist sampling criterion [40], the maximum Doppler shift frequency that can be detected is limited to half of the PRF.

The ultrasonic frequencies used for most clinical applications lie in the range from 2 to 10MHz and by a fortunate coincidence the Doppler frequencies arising from blood flow measurement lie in the audio frequency range. A physician with a trained ear listening to the Doppler signal may be able to draw some simple qualitative indications of a patients condition.

2.2.5 The Power Spectrum of the Doppler Signal

Listening to changes in the Doppler audio signal is adequate for initial assessment of whether or not a problem exists but even the most experienced practitioner would find it difficult to relay accurate information about the degree of a stenosis. A more reliable, quantitative method of Doppler signal analysis is necessary and the spectrum of the Doppler signal, which bears a direct relationship with the velocity profile (2.3), contains useful diagnostic information [41].

The process of obtaining the power spectral density of the Doppler signal is termed spectral estimation. Spectral estimation is not a straightforward procedure since the range of blood flow velocities in the sample volume varies over the cardiac cycle due to the systole action which is rhythmic pumping of blood through the body by the heart. The blood flow is therefore a non-stationary process and consequently the shape of the Doppler power spectrum is constantly changing. The Doppler signal can however be considered to be stationary with the blood velocity remaining approximately constant over short periods of time, usually from around 2 to 20ms depending on the phase of the cardiac cycle, and is accordingly classed as wide-sense-short-term-stationary.

The conventional method of estimating the spectrum is to use the short term fast Fourier transform (STFFT) [5] on these short windows of quantised quasi-stationary data and the results are commonly displayed in the form of a sonogram, an example of which is shown in figure 1.1. The sonogram can be split into columns, each the width of the data window, to show a series of Doppler spectra in real-time. The power of each frequency component is then indicated by the grey scale intensity, in this case white showing that a frequency component has no strength and black representing the components with maximum amplitude.

2.2.6 Disease Detection in the Common Carotid Artery

Atherosclerotic disease in the common carotid artery can be brought on by hypertension, smoking and diabetes. Plaque can build up on the wall of the artery and the pulsatile flow causes particles of the plaque to be shed off. These particles embolise, blocking finer blood vessels downstream of flow and strokes can occur as a result of this. The aim of this work is to reduce the cost and increase portability of a system which offers high sensitivity for detection of arterial disease.

An arterial stenosis blocks and narrows the channel of the blood flow causing a change in its velocity profile which is reflected in the Doppler power spectrum of the Doppler signal detected by a PDU instrument [3]. The compressibility of the blood is negligible and the volume flow rate, defined as the product of cross-sectional area and the mean velocity of the blood cells [42] passing through that area, in a normal section of the carotid artery must be maintained through the diseased section. A stenosis causes a narrowing of the blood vessel and to maintain the volume flow through the diseased region the velocity of blood passing the stenosis must increase. If A_n and A_s are the arterial cross sectional areas in normal and stenosed regions and v_{m_n} and v_{m_s} are the mean velocities of the blood cells through these areas (figure 2.2) then if the volume flow rates are equal:

$$A_n.v_{m_n} = A_s.v_{m_s} \tag{2.4}$$

which can be rearranged to:

$$\frac{v_{m_s}}{v_{m_n}} = \frac{A_n}{A_s} \tag{2.5}$$

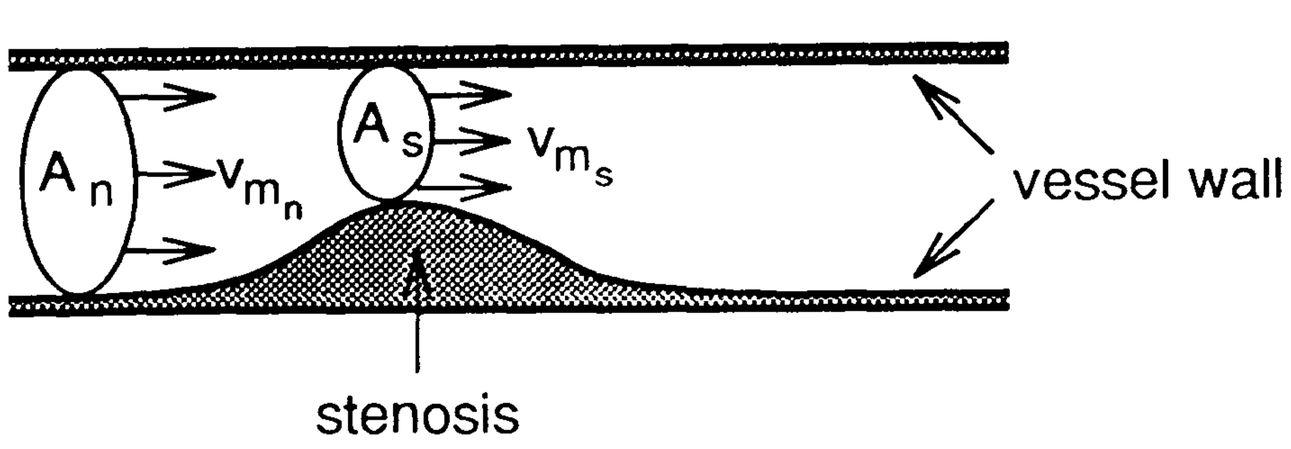


Figure 2.2: Flow velocity and cross-sectional area in normal and stenosed regions of the common carotid artery.

Since the mean Doppler frequency is proportional to the mean velocity of the blood flow (2.3):

$$\frac{f_{m_s}}{f_{m_n}} = \frac{A_n}{A_s} \tag{2.6}$$

enabling the severity of stenosis to be determined from the ratio of the mean Doppler shift frequencies in the proximity of and prior to a stenosed region.

Post-stenotic blood flow tends to be disturbed [43], the strength of turbulence depending on the degree of stenosis. The velocity range of blood cells within the disturbed region is widened and may even be reversed, translating to an increase in the bandwidth of the Doppler power spectrum [42][44].

The effects of change in mean frequency and bandwidth first become noticeable in the decelerative phase of systole and so spectra estimated from data segments around this region are used for analysis. Typical shapes of spectrograms and their PSDs on the decelerative phase are shown in figure 2.3.

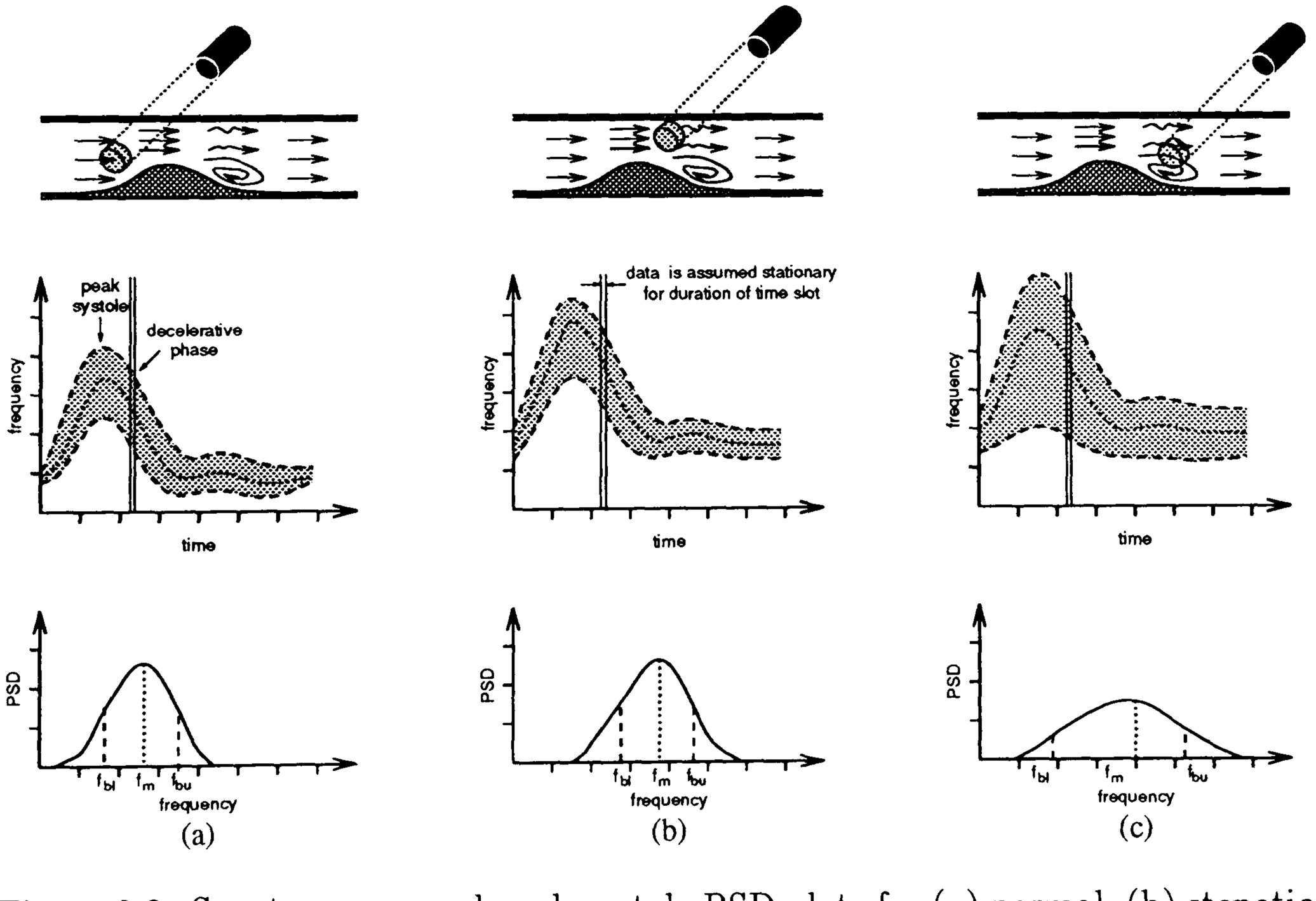


Figure 2.3: Spectrograms and peak systole PSD plots for (a) normal, (b) stenotic and (c) post-stenotic blood flow regions of the common carotid artery. Mean frequency f_m $(\cdots)$ is increased in the vicinity of the stenosis while half peak power bandwidth $f_b = f_{bu} - f_{bl}$ $(--)$ is widened in the post-stenotic region.

2.3 Spectral Estimation

This section discusses conventional and modern methods of spectral estimation [9] which may be used to extract the power spectral density from the Doppler time signal. The primary aim of this thesis is to design hardware for a real-time spectral estimator which offers high sensitivity to atherosclerotic disease detection at an acceptable cost, hence, research leading to selection of an optimal method for estimation of mean frequency and bandwidth [45] is reviewed here.

2.3.1 Modelling the Doppler Ultrasound Blood Flow

A model is needed to characterise the random blood flow signal resulting from measurement on a pulsed Doppler ultrasonic instrument in order to estimate its spectral characteristics [46]. Rather than considering the Doppler shift of a single blood cell as in (2.3) which assumes ideal conditions, the contributions from all the red blood cells within the resolution sample volume of the PDU instrument should be taken into account. Brody and Meindl [47], Mo and Cobbold [48] define models for Doppler ultrasound backscatter from a single blood cell movement and assume the total signal returned to be the sum of the contributions of each of the individual scatters. Shung et al. [49] however postulate that red blood cells, whose volume concentration in blood is approximately 45%, do not act as independent scatterers as they are likely to interact with one another due to their close packing. Angelsen [50] considers the blood cell interactions by treating the blood flow within the sample volume in a different manner, considering the blood cells to act as a continuum with scattering arising from fluctuations in the density and compressibility of the blood, caused by variations in red blood cell concentration. Instrumentation can also effect the characteristics of the Doppler signal [37] as for example the geometry of a narrow beam-width results in a broadening of the Doppler power spectrum [51][52]. The joint conclusion drawn

from the research is that the Doppler signal is a zero-mean random variable with a Gaussian probability density function and a periodically time-varying spectrum.

The digitised Doppler signal sampled at frequency f_s is a random variable characterised by the stochastic process $\{\mathrm{x}[n]\}$ [53]:

$$\{\mathrm{x}[n]\} = ..., \mathrm{x}[-1], \mathrm{x}[0], \mathrm{x}[1], \mathrm{x}[2], ... \tag{2.7}$$

Over the cardiac cycle the velocity profile of the blood flow changes and thus the spectrum of Doppler shift frequencies, encoded into the envelope of the Doppler time signal [1], varies in sympathy. This non-stationarity in the frequency spectrum makes characterisation difficult as the parameters to be estimated are continuously changing. However, by considering short periods of time, the velocity profile and consequently the Doppler frequency spectrum remain approximately constant, implying that Doppler signal is quasi-stationary and its autocorrelation function [53][54]:

$$r_{\mathrm{xx}}[l] = E\{\mathrm{x}[n].\mathrm{x}[n-l]\} = E\{\mathrm{x}[n+l].\mathrm{x}[n]\} \tag{2.8}$$

is dependent only on the time lag l for all discrete times n and lags l (note E denotes the expectation operator).

Since $\{\mathrm{x[n]}\}$ is stationary in the first two moments and has finite variance σ^2_{x}, which is equal to $r_{\mathrm{xx}}[0]$, the Doppler signal is classed as wide-sense stationary [46], a characteristic which simplifies the estimation process [53].

This wide-sense stationary stochastic process $\{\mathrm{x}[n]\}$ contains infinite energy and hence its Fourier transform does not exist due to collapse of Dirichlet's conditions [55][56]. This leads to consideration of the Doppler signals power spectral density (PSD) function which is a plot of time averaged energy distribution with frequency. The PSD may be obtained using the Wiener-Kintchine theorem [57]

by computing the Fourier transform of the autocorrelation sequence:

$$P(f) = \sum_{l=-\infty}^{\infty} r_{\mathbf{xx}}[l].e^{-2\pi lf/f_s} \qquad (2.9)$$

which is a real valued positive semidefinite function [27]. The PSD shows the distribution of Doppler shifts and is therefore used to estimate the velocity profiles.

The random time series $\{\mathbf{x}[n]\}$ represents the ensemble of all the possible Doppler time series realizations from which the ensemble statistics are drawn. In practice the statistical ensemble averages need to be replaced by averages derived from the N sample short data segments of observed Doppler signal realisations $\{x[n]\}$ (figure 2.4), which is only possible if the process is ergodic [58][59]. In addition to being wide-sense stationary over these short periods, the Doppler time signal can is also cyclo-stationary [8] over the cardiac cycle since the rhythmic pumping by the heart periodically produces similar blood flow velocity profiles (figure 1.1). Averaging reduces the variance of spectra estimated from similar points on consecutive cardiac cycles, allowing the Doppler signal to be classed as approximately ergodic and the ensemble averages to be estimated with sufficient accuracy providing the averages are taken over enough cycles [4][60].

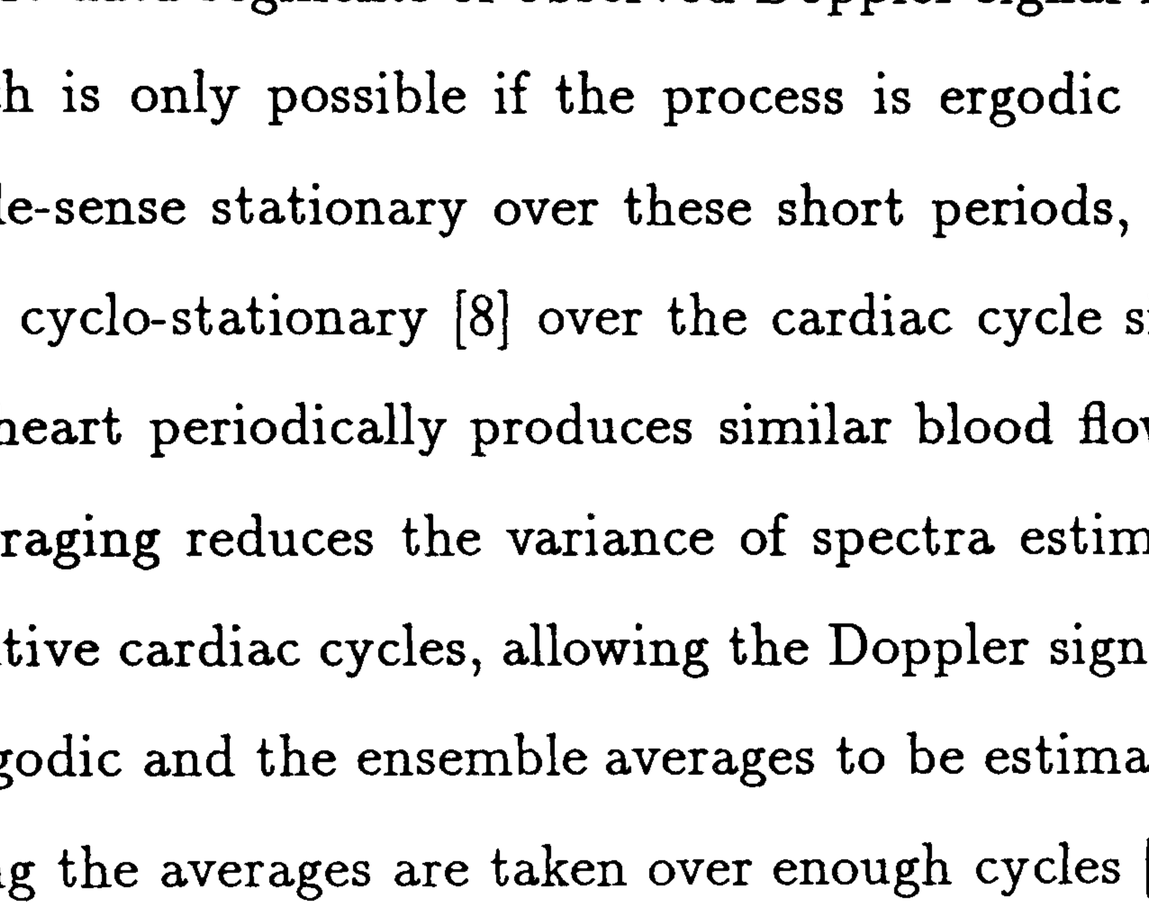


Figure 2.4: Example of the random output signal from a PDU detector. The signal, sampled at a frequency f_s=12.8kHz, consists of 2048 samples in the first 80ms of the cardiac cycle and is quantised into 10 bit words (1024 levels).

2.3.2 Overview of Spectral Estimation Methods

Figure 2.5 shows a tree structure of different spectral estimation methods which may be used to measure the blood flow from the Doppler time signal produced by pulsed ultrasonic Doppler detectors. Time/frequency resolution limitations associated with the Fourier transform based conventional methods led to consideration of the modern methods in which these problems were addressed [25]. The parametric (model based) modern methods are divided into three categories namely autoregressive (AR), moving average (MA) and autoregressive moving average (ARMA) from which a variety of estimators can be derived by using different mathematical interpretations [9]. Finally, the minimum variance modern method is based around measuring the power output of a bank of narrow-band filters [61].

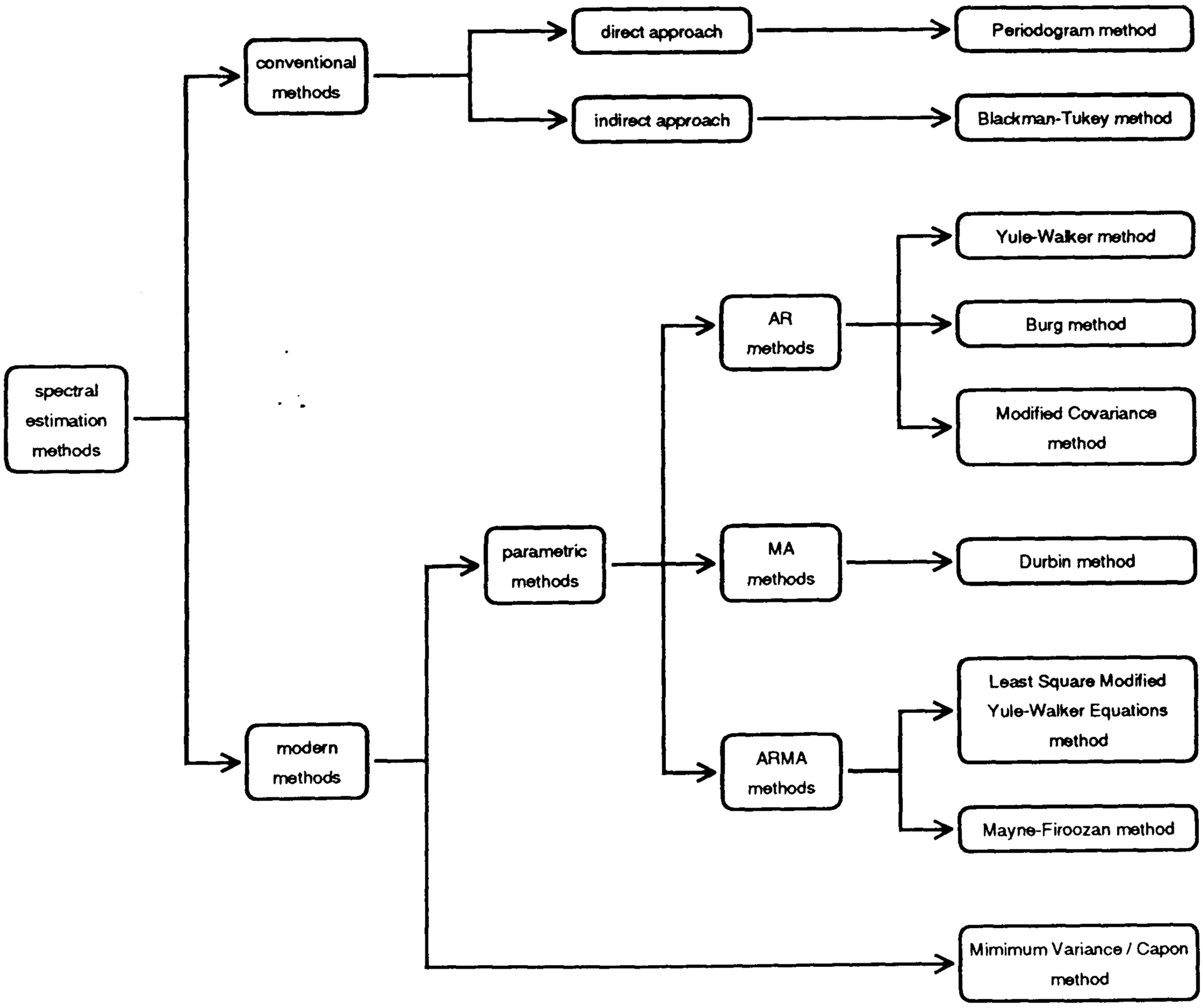


Figure 2.5: Overview of spectral estimation branches and methods.

2.3.3 Conventional Methods of Spectral Estimation

Conventional estimation of the PSD of the discretely sampled Doppler signal $x[n]$ is based on the use of the fast Fourier transform (FFT) using either direct or indirect methods described as follows.

(A) Direct Methods

The direct method of spectral estimation based on the periodogram, involves taking squared magnitude of the Fourier transform of the N data samples [62]:

$$\hat{P}_{PER}[m] = \frac{1}{N} \left| \sum_{n=0}^{N-1} x[n].e^{-j2\pi mn/N} \right|^2 \tag{2.10}$$

where m represents the PSD frequency index such that $f[m] = m.f_s/N$ Hz is limited to half of the sampling frequency f_s $(0 \leq m \leq N-1)$ and $\hat{}$ indicates an estimate.

(B) Indirect Methods

The indirect methods due to Blackman and Tukey [63] compute the PSD by taking the FFT of a time averaged autocorrelation sequence:

$$\hat{P}_{BT}(f) = \sum_{l=-L}^{l=L} \hat{r}_{xx}[l] e^{-2j\pi l.f/f_s} \tag{2.11}$$

where the magnitude of f is also defined for the same frequency interval as in the direct method and the autocorrelation lag estimates $\hat{r}_{xx}[l]$ are forced to zero outside the range of $-L$ to L $(L \leq N-1)$. Several different autocorrelation estimates can be used [13][58] with Jenkins and Watts [64] for example arguing the use of the biased estimate:

$$\hat{r}_{xx}[l] = \frac{1}{N} \sum_{n=0}^{N-l-1} x[n+l].x[n] \tag{2.12}$$

for $l = 0, 1, .., M$ and given that the data is wide-sense stationary:

$$\hat{r}_{xx}[-l] = \hat{r}_{xx}[l] \tag{2.13}$$

The estimate displays low mean-square error and is positive definite [25].

(C) Evaluation of Conventional Methods

High density, fast speed VLSI hardware coupled with the computationally efficiency of the FFT algorithm [65] make conventional methods economically feasible for real-time estimation [66] and with continuing technological advances such devices are becoming more commonly used due to their increasing affordability. In general, the PSD estimates P_{PER} and P_{BT} differ, but can be used to provide matching results if the number of autocorrelation lags in (2.12) is maximised for the number of data samples ($L = N-1$) [25]. Comparing the complexity of direct and indirect methods then for production of identical PSD estimation results the direct method is the preferred approach since it does not require the calculation of an autocorrelation sequence estimate prior to the FFT.

However, accurate estimation is compromised in the conventional methods due to the finite length of the data sequence [12]. Similar spectral resolutions and distorting effects are produced by the zeroing of data outside the observation window in the periodogram approach and by truncation of the autocorrelation sequence in the indirect method [25]. Concentrating on the periodogram approach, the use of a finite length data sequence can be considered to be the result of windowing an infinite length data sequence with a boxcar function [5]. In the frequency domain this windowing translates to a convolution of the desired transform with the sinc function which results in distorted estimate. The main lobe width of the sinc function causes spectral broadening and its side lobes produce spectral leakage (Gibbs Phenomenon [55]) which can mask the weaker frequency components of the true PSD.

A longer data sequence reduces the PSD distortion as the signal becomes a closer representation to the infinite data sequence and also improves spectral resolution which is proportional to f_s/N [25]. However, the window time length must be kept short for the wide-sense stationary assumption, upon which conventional estimation is based, to be valid, thereby putting a limitation on the resolution that can be achieved for a given sampling frequency f_s.

In attempts to improve the PSD estimation using conventional methods window functions other than the boxcar, Bartlett or Hanning for example, can be used to reduce side-lobe levels [5] but these lower the resolution of the spectral estimate by broadening the main-lobe of the window transform [25].

2.3.4 Modern Parametric Methods of Spectral Estimation

Parametric or model based methods of spectral estimation can alternatively be used, offering potentially better frequency resolution than the conventional approaches but with the cost of significantly greater computational burden [9]. In these methods the unrealistic assumption that data is zero outside the wide-sense stationary data segment is dropped by using knowledge of the underlying process or by making more reasonable assumptions about the unobserved data behaviour. There are three steps to the parametric estimation procedure [45]:

- The first is to select a suitable time series model with an appropriate model order making use of a priori knowledge of the underlying process.
- Secondly the parameters of the model need to be estimated from the observed data segment or the estimated autocorrelation sequence derived from this data.
- The Doppler signal spectral estimate is derived by substituting the model parameters estimated in the second step into a theoretical PSD expression for the model selected in the first step.

(A) Background of Parametric Estimation

A brief introduction to the parametric modelling is detailed here, for detailed analysis of these methods [9][13] should be referred to. The derivation of a linear system model, whose transfer function is such that when driven by a white noise signal $\{u[t]\}$ the output signal $\{x[t]\}$ has characteristics resembling those of the Doppler signal, is proposed. In general a linear system can be modelled in terms of a filter linear difference equation:

$$x[n] = -\sum_{k=1}^{p} a[k].x[n-k] + \sum_{k=0}^{q} b[k].u[n-k] \tag{2.14}$$

where $a[k]$ are the model order p autoregressive parameters and $b[k]$ are the model order q moving average parameters. The rational system transfer function, expressed in terms of the z transform [58] is:

$$H(z) = \frac{X(z)}{U(z)} = \frac{B(z)}{A(z)} \tag{2.15}$$

$$H(z) = \frac{\sum_{k=0}^{q} b[k]z^{-k}}{1 + \sum_{k=1}^{p} a[k].z^{-k}} \tag{2.16}$$

Equation (2.16) specifies the process as autoregressive moving average with p poles and q zeroes which can abbreviated to ARMA(p,q). Two other types of parametric model are also feasible. The autoregressive all pole model is formed by removing the zeroes (q=0):

$$H(z) = \frac{1}{1 + \sum_{k=1}^{p} a[k].z^{-k}} \tag{2.17}$$

assuming that $b[0] = 1$ and can be referred to as AR(p). In contrast, the all zero moving average model, MA(q), sets $p = 0$ to eliminate the poles:

$$H(z) = \sum_{k=0}^{q} b[k]z^{-k} \tag{2.18}$$

(B) AR, MA and ARMA Parametric Methods

Within each of the three branches of parametric spectral estimation a number of methods can be derived be using different mathematical approaches. The Yule-Walker equations [9] relate the AR parameters to the autocorrelation function of the Doppler signal and are solved to reveal the model parameters by the computationally efficient Levinson algorithm [67] as the autocorrelation matrix formed is Toeplitz. Burg's AR method [68][69] uses linear prediction theory to extrapolate the autocorrelation sequence, calculated from the N data samples, for unknown lags and is named as the Maximum Entropy spectral estimator because it maximises the randomness of the unknown time series to impose the fewest constraints [25]. The Modified Covariance AR method [9] is based on solution of a set of linear modified covariance equations whose solution determines a set of parameters which minimise forward and backward prediction error power.

One of the key features of a MA(q) process is that for lags with magnitude greater than q the theoretical autocorrelation values are zero by model definition, instead of being forced to zero as in the Blackman Tukey method. Durbin's MA method [9] uses Wold's theorem [25] to treat the non-linear MA process as a high model order linear AR process (e.g. Yule-Walker), to simplify MA computation.

The Least Square Modified Yule-Walker Equations method, a maximum likelihood ARMA(p,q) process [9], involves a least squares computation [27] for the calculation of AR parameters from a system of equations based around the estimated autocorrelation sequence. Once the AR parameters are estimated they are used as coefficients of an all zero filter whose output approximates the MA(q) process. Use of the Durbin method then enables the MA parameters to be estimated. Another ARMA method known as the Mayne-Firoozan [9] or three-stage least square method exploits the association between the output data sequence $x[n]$ and the unknown input $u[n]$, avoiding the non-linear systems of equations encountered in ARMA maximum likelihood estimators.

Once the parameters of the process are estimated then they can be substituted in the following expression for the computation of the estimated ARMA PSD:

$$P(f) = \sigma_u^2 \left| \frac{B(z)}{A(z)} \right|^2_{z=e^{j2\pi f/f_s}} = \sigma_u^2 . \frac{\left| \sum_{k=0}^{q} b[k].e^{-j2\pi kf/f_s} \right|^2}{\left| \sum_{k=0}^{p} a[k].e^{-j2\pi kf/f_s} \right|^2} \tag{2.19}$$

where σ_u^2 is the variance of the input white noise source. The PSDs of the AR and MA processes can also be calculated from (2.19) by setting either $B(z)$ or $A(z)$ to unity as appropriate.

(C) Selection of a Spectral Estimation Method

Selection of the branch of parametric estimation is aided if knowledge of the spectral shape exists a priori. AR methods model spectra with sharp peaks but no deep nulls well, while MA methods are suitable for spectra showing the opposite characteristics of deep nulls and no sharp peaks. The ARMA methods are recommended when the spectral shape has both sharp peaks and deep nulls.

Parametric methods also require the selection of an appropriate model order given that too low an order leads to poor spectral resolution while too high an order can produce spurious spectral peaks. Choice of model order is further complicated as the best order for one method may not be optimal for a different method. Various criteria, such as the Final Prediction Error (FPE) [70][71], Akaike Information Criterion (AIC) [72] and Criterion Autoregressive Transfer function (CAT) [73], can be used to aid in selection of the model order by analysing the prediction error power. However, this study is concerned with the accurate estimation of mean frequency and bandwidth (see section 2.2.6) for which the recognised criteria are ill-suited resulting in selection of model orders which are higher than necessary [10]. A selection criterion which specifically weights accuracy in estimation of the spectral mean frequency and bandwidth of a set of typical Doppler signals against the cost of different methods over a range of model orders is therefore required.

(D) Cost/Benefit Selection of Spectral Estimation Methods

Ruano and Fish [10][11] proposed a scheme specifically aimed at selection of optimal pulsed Doppler ultrasound spectral mean frequency and bandwidth estimators and covers both conventional and modern methods. The cost/benefit criterion:

$$c(m,k,p,q) = \frac{cost(m,N(k),p,q)}{benefit(W,B_{f_m},S_{f_m},B_{f_b},S_{f_b})} \tag{2.20}$$

weighs computational cost of different spectral estimation methods against the benefit or accuracy which they provide. The criterion, expressed in terms of the indices m - type of estimation method, p - AR model order, q - MA model order (for a non-parametric method $p = q = 0$), is minimised over a set of signals k which have Doppler signal characteristics, that is signals covering a range of typical mean frequency f_m and half power bandwidth f_b. The benefit of the particular estimator being considered is viewed as the inverse of the error it produces, in terms of the bias B and standard deviation S in the estimated values of f_m and f_b, with different weights applied to these statistics depending on whether f_m or f_b or both f_m and f_b are of interest.

Cost/benefit analysis of a range of the spectral estimators overviewed in figure 2.5, including the periodogram windowed by boxcar and Hanning windows, results in selection of the Modified Covariance spectral estimator with model order $p = 4$ when both f_m and f_b are to be estimated. Estimation of f_m alone for detection of average flow velocity results in selection of periodogram method with a boxcar window while the $p = 4$ AR Modified Covariance method is once again chosen when determining just f_b to measure flow disturbance alone.

Another requirement is for estimation of spectral properties other than f_m and f_b involving the higher moments for which it is necessary to use large model orders. Model orders up to $p = 30$, presenting a very large computational burden, are envisaged.

(E) Modified Covariance Method

The Modified Covariance method is described in this section due to its suitability for determining f_m and f_b. The Modified Covariance spectral estimator models the signal whose spectrum is required by autoregressive (AR) filtered white noise. The core of the estimator depends on the solution of the matrix equation:

$$C.\hat{A} = -B \tag{2.21}$$

for filter parameters $\hat{a}[i]$ from the vector $\hat{A}$ where:

$$\hat{A} = \begin{bmatrix} \hat{a}[1] \\ \hat{a}[2] \\ \vdots \\ \hat{a}[p] \end{bmatrix} \tag{2.22}$$

The covariance matrix C:

$$C = \begin{bmatrix} c[1,1] & c[1,2] & \dots & c[1,p] \\ c[2,1] & c[2,2] & \dots & c[2,p] \\ \vdots & \vdots & \ddots & \vdots \\ c[p,1] & c[p,2] & \dots & c[p,p] \end{bmatrix} \tag{2.23}$$

and RHS vector B:

$$B = \begin{bmatrix} c[1,0] \\ c[2,0] \\ \vdots \\ c[p,0] \end{bmatrix} \tag{2.24}$$

contain elements $c[j,k]$ given by:

$$c[j,k] = \frac{1}{2(N-p)} \cdot (c_1[j,k] + c_2[j,k]) \tag{2.25}$$

where

$$c_1[j,k] = \sum_{n=p}^{N-1} x[n-j].x[n-k] \tag{2.26}$$

and

$$c_2[j,k] = \sum_{n=0}^{N-1-p} x[n+j].x[n+k] \tag{2.27}$$

N is the number of samples in the time segment, p is the filter model order and $x[n]$ is the signal sample at time nT where $n = 0, 1, 2, ..., N-1$ and T is the sampling period.

The white noise variance parameter is determined by:

$$\hat{\sigma}_u^2 \;=\; c[0,0] \;+\; \sum_{k=1}^{p} \hat{a}[k] \;.\; c[0,k] \tag{2.28}$$

and this is then used in the calculation of the power spectral density (PSD):

$$\hat{P}_{AR}(f_n) \;=\; \frac{\hat{\sigma}_u^2}{|\,1 \;+\; \hat{a}[1] \;.\; e^{-j2\pi f_n} \;+\; \ldots \;+\; \hat{a}[p] \;.\; e^{-j2\pi f_n p}\,|^2} \tag{2.29}$$

where f_n is a frequency index.

The four stages of the Modified Covariance method, the covariance/RHS matrix element calculation (2.25) to (2.27), the solution of the set of linear equations (2.21), the white noise variance calculation (2.28) and the calculation of the PSD (2.29), are summarised by the block diagram shown in figure 2.6.

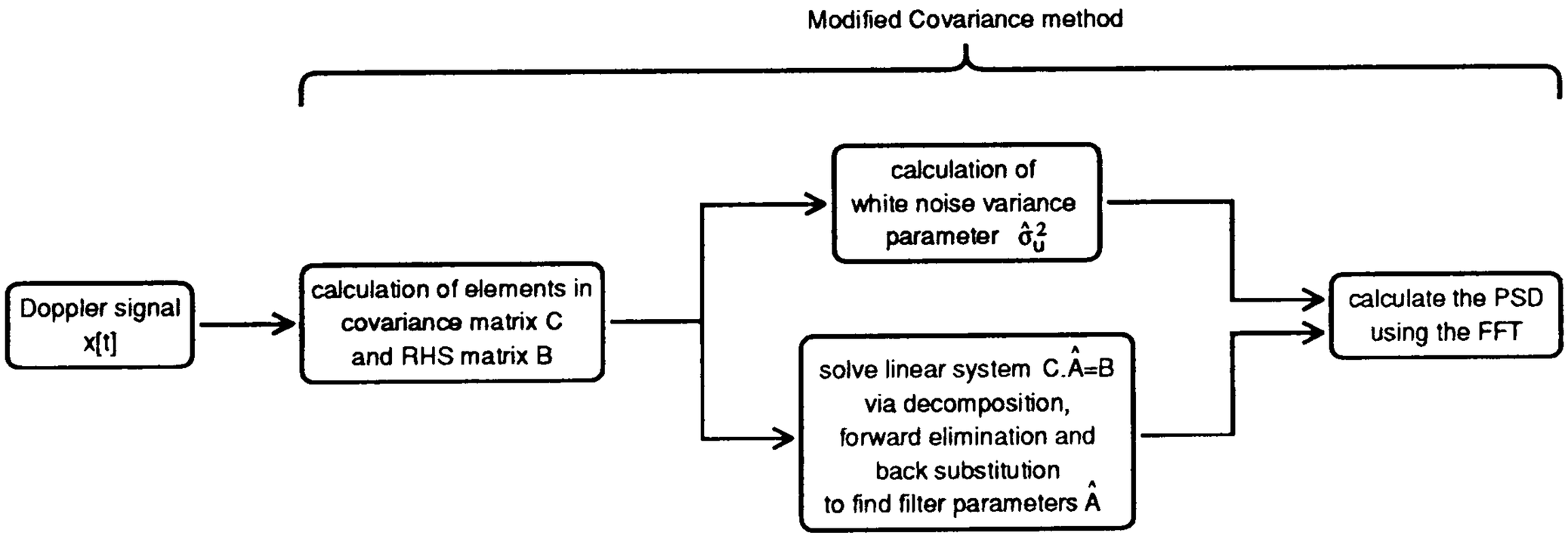


Figure 2.6: Block diagram showing the four stages of the Modified Covariance spectral estimation method.

(F) Implementation of the Modified Covariance Spectral Estimator with the use of MIMD Model Topologies on a Transputer Platform

Previous research has shown that parallel processing techniques need to be employed in order to gain the throughput necessary for real-time computation of the Modified Covariance method [14] where the feasibility of modelling the estimator as a multiple instruction stream multiple data stream (MIMD) process [74] was investigated. Transputers [75], configured to various MIMD topologies (figure 2.7), were used as an implementation platform [14][15][26] and programmed with the use of the OCCAM language [76].

The Modified Covariance algorithms had to be partitioned before they were mapped onto the standard MIMD topologies. The partitioning involved the selection of a level of granularity, e.g. in a medium grain partitioning scheme a processor performed the entire computation of a matrix element as opposed to a fine grain scheme where this computation was shared amongst a number of processors. Tasks defined by the level of granularity were then allocated to specific processors for execution in a certain time frame. Partitioning of the algorithms concerned with individual blocks of the Modified Covariance method shown in figure 2.6 (apart from the white noise variance which was computed sequentially) onto the MIMD model topologies was initially considered [14]. The results of the investigation showed that the calculation of the covariance matrix elements (2.25) to (2.27) and the PSD calculation (2.29) could be efficiently performed on

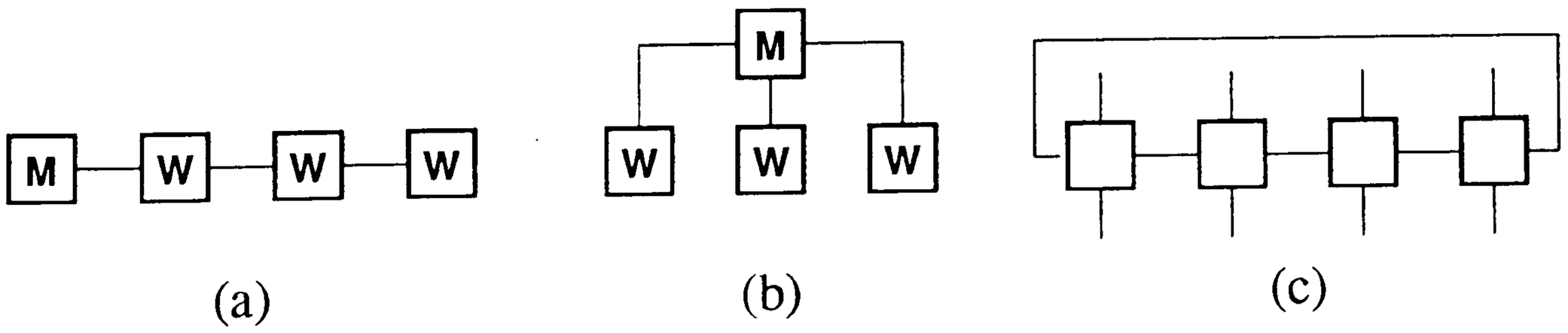


Figure 2.7: Transputer based topologies (a) processor farm linear, (b) processor farm tree - depth 2 (M denotes a master processor which schedules tasks to the worker processors indicated by W), and (c) ring topology.

depth 2 tree topologies. Despite this however the execution of the DFT part of the PSD calculation was too slow to meet the real-time requirement for certain signal cases when it was programmed onto the transputer based platform. Solution of the system of equations (2.21) was attempted using the Cholesky decomposition technique with forward elimination and back substitution [27]. However, it was found difficult to gain any useful speedup using the MIMD ring topology over sequential execution.

A second approach, which considered spectral estimation on a number of Doppler data segments simultaneously, proved more successful, as this strategy did not require intercommunication between master and worker processors during execution of the algorithm [26]. A processor farm tree topology was again used with communication only necessary to distribute input data time segments and to retrieve the computed spectral estimates. However, the transputer based execution time results shown were only for a fixed data length of 10ms and estimation using data segments as short as 2ms could have led to difficulty. The range of model orders considered was limited to a maximum of $p = 10$ for which real-time operation was just achieved for the case when $N = 256$. Estimation up to model order $p = 30$ would however have been desirable and this transputer implementation would not have been able to produce the required throughput to cope with the hugely increased burden of this high model order in certain conditions.

In summary the MIMD topologies imposed restriction, in terms of the available communication resources and the number of processors, which constrained the partitioning of the algorithms. Finding the strategy which offered maximum speedup and minimum execution time, proved to be difficult as demonstrated by the inadequacy of the MIMD transputer implementations under certain conditions. The system also imposed fairly high costs, that cost of the host computer, the transputers, the transputer platform and the software. There was also a problem in that the large physical size made equipment difficult to transport.

2.4 Systolic Arrays

2.4.1 The Systolic Model of Computation

As an alternative to the MIMD model the systolic model of computation [23] is considered for the real-time implementation of the Modified Covariance method in this thesis. The systolic model also utilises concurrent processing techniques to speed up computation but the constraints imposed by partitioning onto a predetermined topology are removed. Systolic architectures are designed to fit the algorithm by considering the communication and processing requirements presented by the data dependencies between the operations involved. This can be compared to trying to fit the algorithms onto different known MIMD topologies to determine which offers the best parallelism. The systolic design methodology is rigorous, and systolic architectures are suitable for implementation of a wide range of algorithms. They employ parallel and pipeline processing to exploit the inherent parallelism and recursive properties of the algorithms to a high degree, enabling the computationally demanding problems, such as those encountered in the Modified Covariance estimator, to be performed in real-time.

2.4.2 What is the Systolic Model of Computation?

The concept of the systolic systems was defined by Kung and Leiserson [17]:

"*A systolic system is a network of processors which rhythmically compute and pass data through the system. Physiologists use the word 'systole' to refer to the rhythmically recurrent contraction of the heart and arteries which pulses blood through the body. In a systolic computing system, the function of a processor is analogous to that of the heart. Every processor regularly pumps data in and out, each time performing some short computation, so that a regular flow of data is kept up in the network.*"

2.4.3 Systolic Architecture Implementation on VLSI

Systolic architectures are used to implement special purpose digital signal processing (DSP) hardware in real-time [23]. The driving factors in the design of the application specific hardware considered here are to provide cost effectiveness, increase portability and improve computational throughput [24]. The advantages of systolic architecture implementation of application specific hardware on a VLSI platform, as opposed to the transputer platform, are discussed here.

Systolic architectures consist of a regular array of low complexity processing elements (PEs) which as the result of a very fine grain partitioning execute single assignment tasks such as multiply accumulate, square-root or division. In an ASIC solution PE's are designed to just execute these specific tasks leading to better cost effectiveness when compared with implementation on the more general purpose transputer system. The modular systolic nature allows easy extension, by adding extra PEs to cope with larger problems. Special purpose systolic systems can therefore be designed to match the specified performance requirements, presenting improved efficiency over a general purpose system. The repeated use of simple building blocks which are regularly located, simplifies the fabrication of systolic arrays in VLSI because of the relatively low cost of replicating an existing PE, thus reducing the design cost. Low design cost coupled with the availability of VLSI technology and the efficiency in utilising resources justify the limited use of the application specific device.

System portability can be achieved by utilisation of the full potential of VLSI technology which allows large highly functional designs to be cheaply mounted on a small number of packages. However, as the number of transistors that can be mounted on a chip continually increases with technological advances, interconnection delay and area pose a more significant constraint on the overall integrated circuit (IC) performance and speed [16]. Hence, communication accounts for a

major proportion of the cost in a VLSI device and there is good reason to restrict the number and length of interconnections. Systolic arrays are therefore cost effective in this sense as communication is localised so that only neighbouring PEs are interconnected, keeping track length as low as possible.

There are two ways in which the designer can attempt to meet the high computational throughput which is necessary in order to achieve real-time operation. The first is to use a highly specified processor capable of running at very fast speed and the second is to use a number of PEs operating concurrently [16]. The huge advances in the transistor packing densities over past decades are not matched by correspondent increases in device speed. IC speed growth rate is in continual decline so exploitation of any concurrency which exists in the algorithms is more effective than trying to reduce processor latency. VLSI technology allows relatively large numbers of the low complexity PEs to be implemented allowing systolic arrays to take full advantage of this concurrency. Systolic arrays employ data pipelining so that throughput rates become dependent on the lag associated with a single PE rather than a group of PEs and high levels of parallelism to spread the computational load over a relatively large number of simple PEs.

Another feature of systolic arrays is simple control, data is transferred between processors on the systolic clock cycle and control signals may be needed to switch PE function, provide resetting or to schedule input/output data. Thus, control software such as OCCAM previously used in the transputer implementation is not needed. The control signals can easily be produced from on board hardware removing the need for a host PC to run the control software.

Hence, the key features of low PE complexity, localised communication and regularity make systolic array processors very amenable to VLSI implementation. Systolic array implementation of the real-time Modified Covariance spectral estimator on a VLSI platform should allow improved cost effectiveness and increased portability over a more general MIMD transputer based system.

2.4.4 Systolic Array Design Methods

A clear methodical architectural design strategy is needed in order to produce functionally correct, localised systolic arrays. To map the algorithm into a systolic array it is usual to first express the algorithm in some type of intermediate format to help facilitate this. The alternative algorithm representation may be in graphical, matrix or computational language format and follows directly from a recursive algorithm description, resulting in a variety of approaches to systolic array design.

The high level language ALPHA is used to express algorithms so that transformations can be applied to the computational description to automatically synthesise systolic arrays [77]. Dense to band matrix transformation and triangular block partitioning techniques are two methods used in conjunction with each other to efficiently partition algorithms onto arrays of known dimension [28]. Snapshots allow the operation of arrays to be viewed on a series of frames which show current processor states and data communications [17]. Systolic mapping procedures based on mathematical transformations of index sets representing PE operations and data dependencies between these operations are also published [78]. Perhaps the most easily understandable and manipulative method is that developed by Kung [23] in which the algorithm is initially expressed in the form of a data dependence graph (DDG) that may be projected in a number of directions to produce a variety of systolic array solutions. The DDG method is adopted in this study.

The DDG method is used here to illustrate the key aspects of recurrence relationships, data dependence, dependence graph scheduling and projection in the mapping procedure from algorithm to systolic array. Systolic arrays for the simple white noise variance calculation in the Modified Covariance method are considered here in order to introduce the design method and show how the key principle of pipelining is used to achieve concurrent processing.

(A) Algorithmic Problem Definition

The white noise variance calculation (2.28) may be rearranged slightly to incorporate $c[0,0]$ within the summation assuming that $a[0] = 1$:

$$\hat{\sigma}_u^2 = \sum_{k=0}^{p} \hat{a}[k] \,.\, c[0,k] \tag{2.30}$$

(B) Recurrence Equations

Most DSP algorithms can be written in the form of a set of recurrence equations consisting of a finite number of separate tasks, commonly referred to as single assignments in the literature. Expressing the summation (2.30) in terms of a recursive variable $r^{(k)}$, where (k) $(0 \leq k \leq p)$ is the recursion step:

$$r^{(k+1)} = \hat{a}[k].c[0,k] + r^{(k)} \tag{2.31}$$

such that $r^{(0)} = 0$, then the white noise variance is given for $k = p$, i.e. $\hat{\sigma}_u^2 = r^{(p+1)}$.

(C) Data Dependence Graph

The recursion in (2.31) can be expressed in graphical format as shown by the node in figure 2.8(a). On the node there are 3 input arcs and 1 output arc, data communication flow directions are indicated by the arrow head and each arc carries a data word. A plot of these nodes on the $[j,k]$ plane $(j = 0)$ as in figure 2.8(b) shows the dependence between the single assignment operations in the calculation of the white noise variance (2.30) and is accordingly called a data dependence graph (DDG). Throughout the thesis nodes are represented as circular symbols and are referenced with respect to their graph coordinates, e.g. $node[j,k]$. The order in which the nodes must be processed is defined by the direction of the horizontal interconnection arcs, which are referred to as local

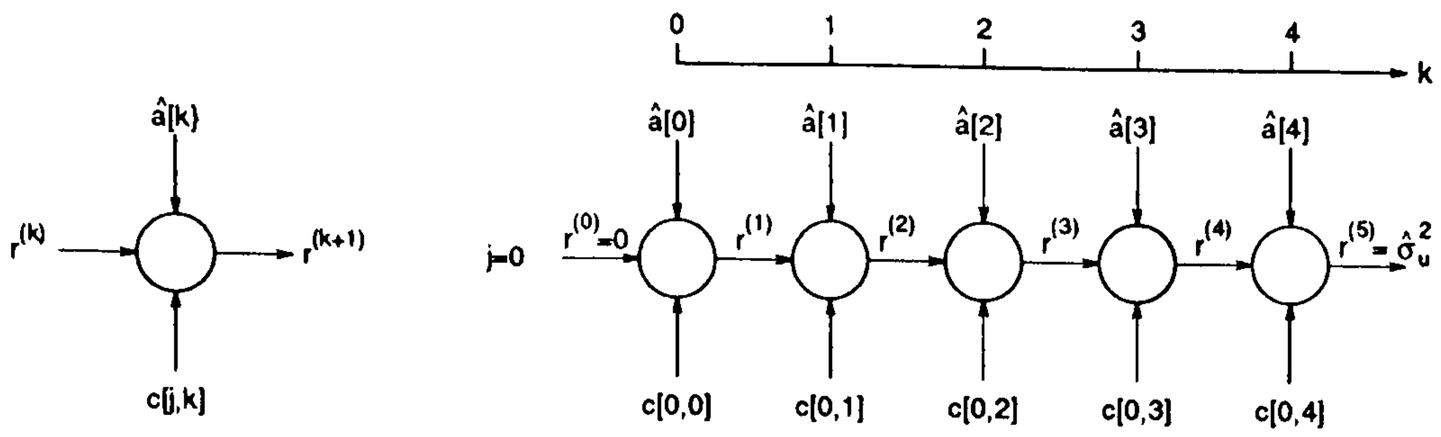


Figure 2.8: (a) Node representing the assignment in (2.31), (b) DDG for general matrix element calculation in a model order $p = 4$ estimator $\hat{\sigma}_u^2 = r^{(5)}$.

data dependencies since the output of one node forms the input to the next. The DDG is fully localised, a condition which allows a purely systolic array to be formed.

(D) Use of Timing and Allocation Functions in Systolization

Different timing and allocation functions can be used to produce a variety of systolic array solutions from the DDG. The timing allocation dictates the time t in clock cycles that a node assignment is performed. Single assignment tasks must execute in the order defined by the directions of the data dependence arcs so for the localised DDG in figure 2.8(b) the linear schedule $t = k$ is valid. Figure 2.9(a) indicates this timing function by the internal node numbering which shows that each node is executed on successive clock cycles.

The allocation function dictates which node operations are performed by a particular PE and also determines the size and shape of the systolic array. The most straightforward allocation is to have a PE representing each node of the DDG by projecting in the j direction (i.e. any direction orthogonal to the k axis since $j = 0$ for all nodes) such that $node[j, k]$ is mapped into $pe[k]$, producing the systolic array in figure 2.9(b) whose internal PE schematic is shown in (d). On each of the input buses to the PE there are systolic delay registers so that all data entering the PE is clocked in synchronously. It is important to set the convention

for input/output timing of data at this early stage to avoid confusion further on when more complex arrays considered. The timings t shown for the input data denote the positive clock edges on which data is clocked into the PEs while for the output t indicates when data should be clocked into external memory so that the result here is available for storage at $t = 5$. Hardware can be reduced by nearly a fifth if the DDG is projected along the k direction into a single systolic PE as shown in figure 2.9(c). The internal PE configuration remains the same but a feedback loop is introduced so that products can be continuously updated and a multiplexor is required to provide initialisation.

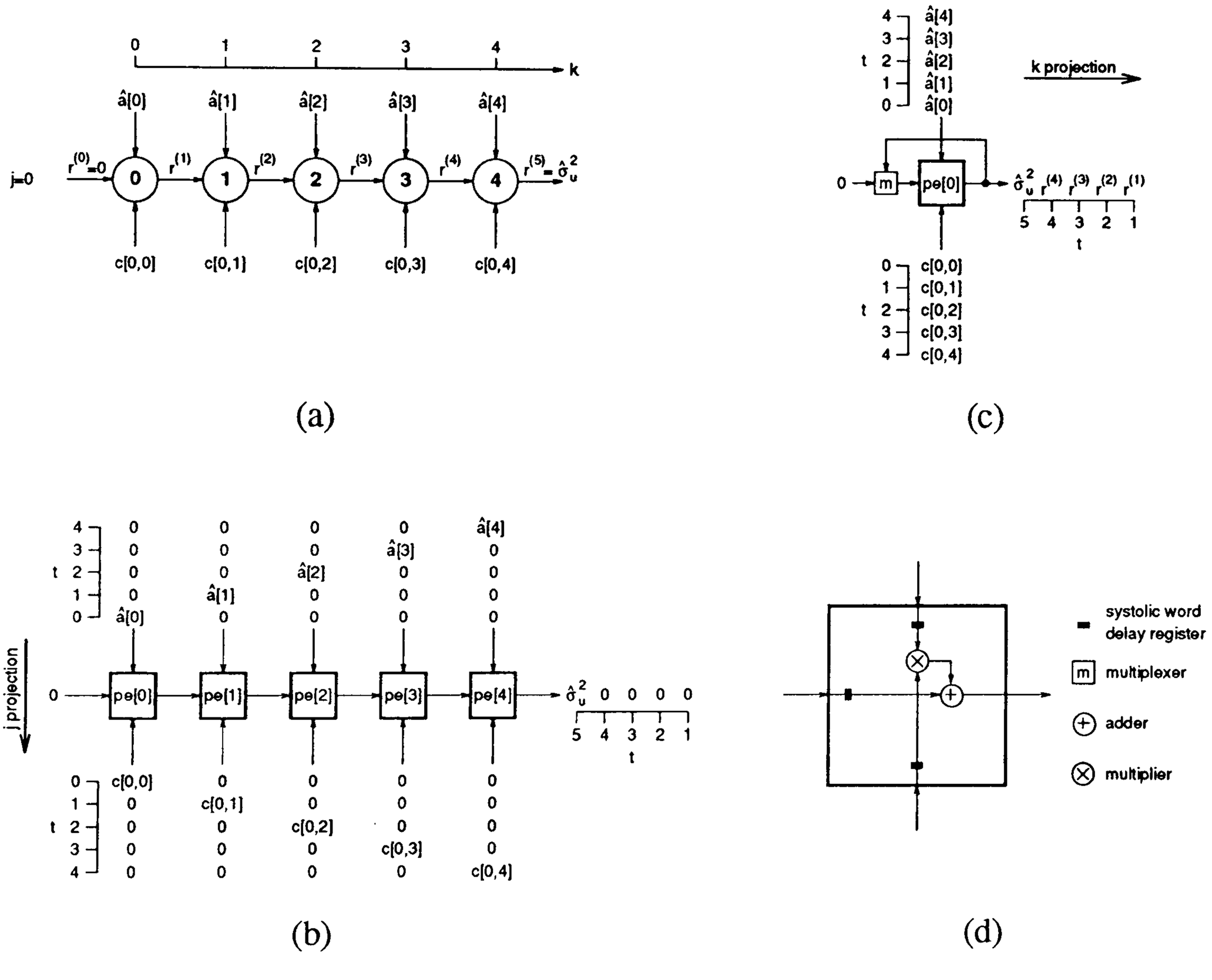


Figure 2.9: (a) Localised DDG showing timing function $t = k$ which is projected in the (b) j direction to form a systolic array of 5 PEs and (c) k direction into a single systolic PE. Part (d) shows a key to the PE function for both of the arrays.

(E) Pipeline and Parallel Processing

The systolic arrays in figure 2.9 are fully pipelined since there is no data broadcasting. The significance of pipelining can be demonstrated by removing the clocking from the array in figure 2.9(b) so that intermediate multiply-accumulate results ripple through. If each inner step product calculation takes time T_{ma} then the overall operation time of the ripple through array becomes $5.T_{ma}$ which is the minimum time between successive white noise variance computations or the block pipeline period. The minimum clock period of the systolic array is T_{ma} leading to the same overall operation time of $5.T_{ma}$. However the block pipeline period of the systolic array is just T_{ma} so that 5 variance computations can be calculated on the array simultaneously or in other words the systolic array provides 5 times the throughput rate compared to the ripple through array. Despite the hardware cost saving of the single systolic PE solution in figure 2.9(c) there is disadvantage in that its continuous problem throughput is also a fifth that of the 5 PE array in (b). However, the single systolic PE offers the same performance as the ripple through array at a fifth of the cost, displaying the importance of pipelining in the design for high throughput applications.

Parallel combines with pipeline processing to provide massive concurrency in other systolic arrays discussed in the thesis such those proposed for the matrix element calculation where a number of PEs similar to that in figure 2.9(b) perform internally pipelined inner-product calculations in parallel with one another.

(F) Design Methodology

The DDG design methodology used is adapted from Kung's cut-set method [16]. Kung's intermediate signal flow graph stage is eliminated here and arc-to-bus mappings are thought of in terms of node-to-PE allocation function with systolic delays carried on these buses derived from the timing function.

2.4.5 Systolic Arrays for the Modified Covariance Spectral Estimator

This section reviews previously proposed systolic arrays relevant to the Modified Covariance method. These systolic arrays fall into two distinct classes, that is problem size dependent and problem size independent.

(A) Review of Problem Size Dependent Systolic Arrays

Problem size dependent systolic arrays are specifically aimed at computing fixed dimension, or fixed model order in the case of the Modified Covariance method, algorithms. Often these systolic arrays can be used in problems of smaller model order but not for larger model order problems.

The calculation of the covariance matrix elements (2.25) to (2.27) is a demanding compute bound algorithm, as the total number of operations involved is greater than the total number of inputs. Consequently the systolic approach represents an inexpensive method for achieving computational speedup [24]. The algorithm, although being somewhat more complicated, bears resemblance to the convolution operation and for this reason Kung's convolution linear systolic arrays [24] provide a useful guide to implementation of the matrix element calculation.

The usual method of solving the system of linear equations for the filter parameters (2.21) is to decompose the problem into the form of triangular systems of equations which can be easily solved by forward elimination and back substitution. One of the first designs to be proposed is Kung and Leiserson's hexagonally connected LU decomposition systolic array [17]. The limiting factor in the attainable throughput from this array tends to be the lag associated with the division PE, a slower operation than multiplication which creates a communication bottleneck at the top of the array. Consideration of alternative mapping directions [23][79] leads to a variety of LU systolic solutions.

One of the features of the Covariance Matrix is that it has symmetrical properties, allowing the Cholesky decomposition and LDL^T algorithms to alternatively be used to reduce the number of single assignment operations. Brent and Luk illustrate arrays for computing these decompositions [80] based on a development of the hexagonal LU array. These two arrays contain $\frac{1}{2}(p^2+p)$ PEs, approximately half the number in the LU array but the Cholesky array has the disadvantage of requiring square-root operation while the LDL^T array, not requiring square root operation, presents greater PE interconnection burden.

QR decomposition [27] can also be used in matrix triangularisation, and hence in the solution of linear equations. With this method applying Given's rotations to the covariance/RHS matrix cuts out the need for the forward elimination stage. However, an initial comparison with the other decomposition methods leads to exclusion of QR decomposition since its overall hardware cost is much greater due to a typical QR systolic array PE needing to perform 4 multiplications [81][82]. This method is more widely used for least square computation [20][83][84] and could be used in a versatile hardware system to compute AR-Modified Covariance and ARMA-Least Square Modified Yule Walker Equations spectral estimation.

The COordinate Rotation DIgital Computer (CORDIC) proposed by Volder [85] then later unified by Walther [86] can be used to compute mathematical operations by performing a series of shift, addition and subtraction operations. Ahmed et al. [87] show the use of CORDIC PEs (see [88][89] for possible PE implementations) in 2-dimensional systolic arrays for Given's orthogonal QR decomposition and for a variation of the Cholesky method which triangularises using hyperbolic rotations [90]. Despite the use of simple PE units, the scheme suffers in that it imposes a limited domain on the input data and variable scale factors [86] need to be removed before PE output. Attempts to overcome these limitations [91][92] add to PE/control complexity and as a consequence of its shortcomings, CORDIC implementation is not pursued in this study.

When considering the hardware implementation of problem size dependent decomposition arrays the precision of data to give sufficient accuracy in the filter parameters needs to be determined. Errors arise due to quantisation of the Doppler signal [93] and finite length rounding effects during computations. If the covariance matrix is ill-conditioned the propagation of these errors through the decomposition array can lead to poor filter parameter estimates [27]. Wilkinson presents error bounds for Cholesky and LU decomposition fixed point arithmetic schemes [94][95][96]. However, the analysis does not consider any error already present in the matrix to be decomposed or the likelihood of overflow, both of which could severely affect the accuracy of the filter parameter results. Computation of relative error bounds [97] on the output of a single assignment resulting from the relative errors on the input operands is also an unsuitable method of determining the filter parameter accuracy. This is because the bounds through all the successive computations grow very large and the probability of reaching such an error becomes extremely remote.

(B) Problem Size Independent Systolic Arrays

The disadvantage with problem size dependent systolic arrays is that when dealing with large dimensioned problems the number of PEs can become too large, leading to high cost and implementation difficulties. Partitioning of algorithms, so that large dimensioned problems can be mapped onto small systolic arrays, is therefore recommended and is considered in [23][28][98]. Of particular interest is the work of Navarro, Llaberia and Valero [28][99][100] detailing partitioned systolic arrays for the solution of triangular linear systems of equations and LU decomposition. In the case of the LU decomposition the algorithm is divided into a number of sub-algorithms which represent LU decomposition, triangular system and inner product step matrix problems, all of smaller dimension than the original algorithm. Triangular block and dense to band matrix partitioning

techniques (DBT) are used to separately map each of the three different types of subproblem onto efficient s by s PE spiral systolic array processors. All of the subproblems can be sequentially chained together for solution of the LU decomposition on a single spiral systolic array.

An unattractive feature of this array is that bi-directional ports are needed on the north and west PEs of the array for the input/output of the U and L matrix elements respectively. This adds to PE complexity as buffering is necessary to avoid logical contention. Similarly buffering is required on the input/output of the memory devices which are used to store the L and U matrices. The subject of storage and retrieval of this data is not discussed in the reference and so it is assumed that they intend to use an addressing system where each of the matrix elements are stored in a specific location. Another problem encountered in their design is that it is not clear how to chain the subproblems together.

(C) Computation of the PSD

The computation of the PSD (2.29) requires use of a Fourier transform technique, and the two main choices are the discrete Fourier transform (DFT) and the fast Fourier transform (FFT) [23]. In implementing these algorithms on VLSI architectures such as systolic arrays there is a trade off between computational burden and communication when considering whether to use the DFT or FFT [16]. Kung demonstrates that the N-point DFT can be mapped onto a locally interconnected linear systolic array of length N PEs which needs to perform N^2 computations [16]. Kung compares this to a systolic array for the N-point FFT computation, requiring a global perfect-shuffle interconnection network which is more expensive in terms of VLSI implementation, but the array reduces the number of computations to $N.log_2N$ effectively reducing operation time [16]. In both arrays however the hardware cost is dependent upon N and since N is quite large ($N = 64$, 128 or 256 for example) the cost would be excessive.

The N-point DFT $\hat{A}[n]$ of $\hat{a}[n]$ can be expressed as the product of a weight matrix W and the filter parameters $\hat{a}[n]$ [101].

$$\begin{bmatrix} \hat{A}[0] \\ \hat{A}[1] \\ \hat{A}[2] \\ \vdots \\ \hat{A}[N-1] \end{bmatrix} = \frac{1}{N} \begin{bmatrix} W^0 & W^0 & W^0 & \dots & W^0 \\ W^0 & W^1 & W^2 & \dots & W^{N-1} \\ W^0 & W^2 & W^4 & \dots & W^{2(N-2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ W^0 & W^{N-1} & W^{2(N-2)} & \dots & W^{(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} \hat{a}[0] \\ \hat{a}[1] \\ \hat{a}[2] \\ \vdots \\ \hat{a}[N-1] \end{bmatrix} \tag{2.32}$$

It is usual to make the number of points N in the DFT/FFT the same as the Doppler signal data sequence length as opposed to a $p+1$ point DFT/FFT (2.29), in order to obtain a smoothed estimate and to do this requires padding with zeroes. If $N \gg p$ (e.g. $p = 4$ and $N = 128$) then the DFT computation can be greatly simplified. The padding by zeroes sets $\hat{a}[n] = 0$ for $p < n < N$ and the DFT for the $p = 4$ problem can be rewritten as:

$$\begin{bmatrix} \hat{A}[0] \\ \hat{A}[1] \\ \hat{A}[2] \\ \hat{A}[3] \\ \hat{A}[4] \\ \vdots \\ \hat{A}[N-1] \end{bmatrix} = \frac{1}{N} \begin{bmatrix} W^0 & W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 & W^4 \\ W^0 & W^2 & W^4 & W^6 & W^8 \\ W^0 & W^3 & W^6 & W^9 & W^{12} \\ W^0 & W^4 & W^8 & W^{12} & W^{16} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ W^0 & W^{N-1} & W^{2(N-2)} & W^{3(N-2)} & W^{4(N-1)} \end{bmatrix} \begin{bmatrix} \hat{a}[0] \\ \hat{a}[1] \\ \hat{a}[2] \\ \hat{a}[3] \\ \hat{a}[4] \end{bmatrix} \tag{2.33}$$

The number of multiply-accumulate computations in (2.33) is greatly reduced to $N(p+1)$ from N^2 in (2.32). Representing the matrix-vector multiplication (2.33) on a DDG is also a straightforward as shown in figure 2.10(a) [102]. Each row of the DDG is similar to the DDG for the white noise variance calculation (figure 2.8) but now with the products $W^{jk}.\hat{a}[k]$ being accumulated along the horizontal to form $\hat{A}[j]$. The $\hat{a}[k]$ inputs are broadcast in the j direction to all nodes in a column as represented by the directionless vertical interconnection arcs. Figure 2.10(b) shows a localised version of the DDG with the timing function $t = j + k$ denoted by the internal node numbering.

Projection of the DDG in the k direction leads to an N PE linear systolic array similar to Kung's [16] but the number of PE's can be significantly reduced by projection in the j direction ($node[j,k] \rightarrow pe[k]$) into $p+1 = 5$ PEs as illustrated in figure 2.11. This and other matrix-vector product arrays formed by other

DDG projections are discussed in Megson [102].The W weight factors can be retrieved from memory and the array also has the advantage that the results are piped out rather than being stored within the PEs thus eliminating parallel access difficulties. The PSD calculation is then the reciprocal of the squared magnitude of the array outputs.

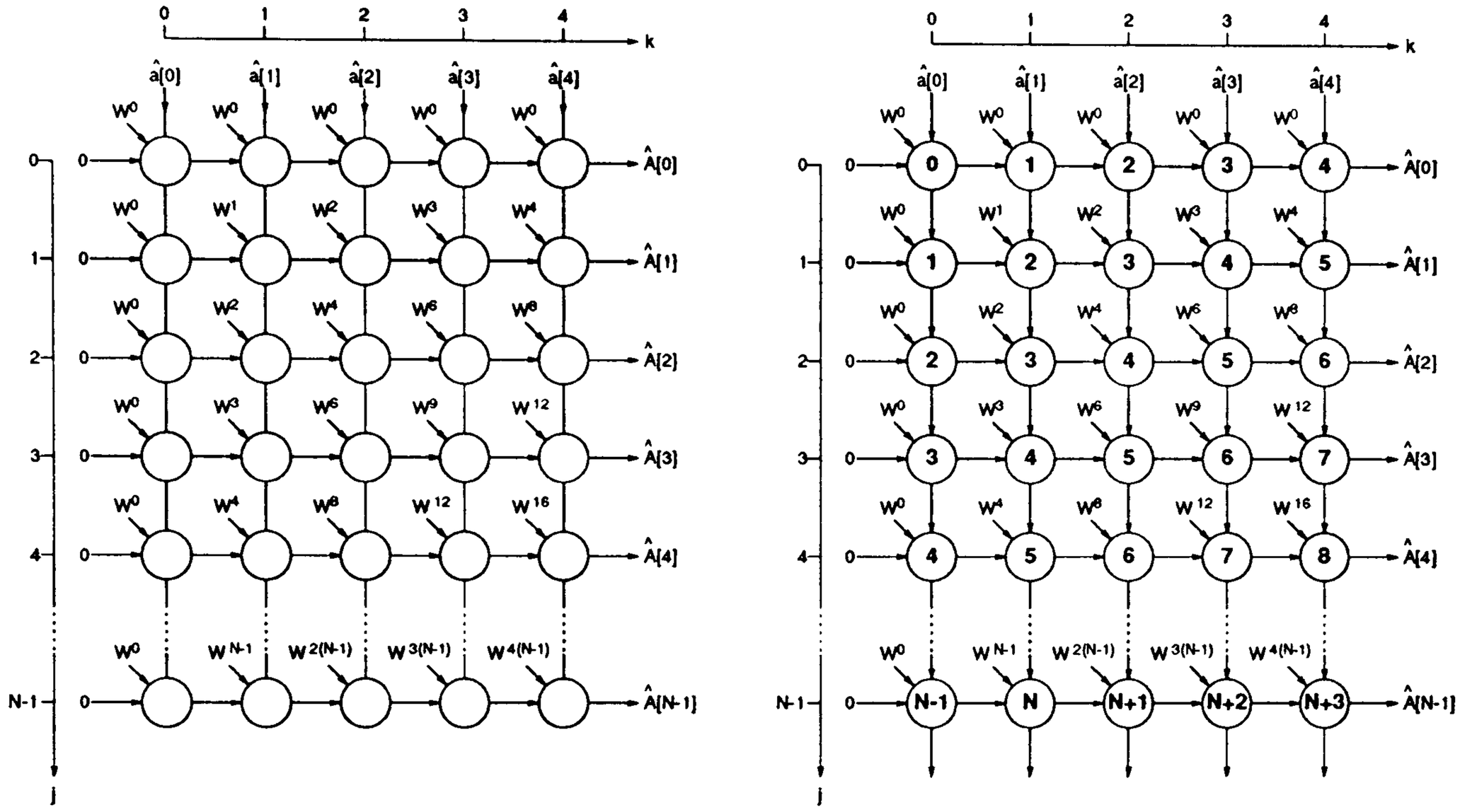


Figure 2.10: (a) DDG representation of (2.33) with global communication for $\hat{a}[n]$ (b) fully localised DDG.

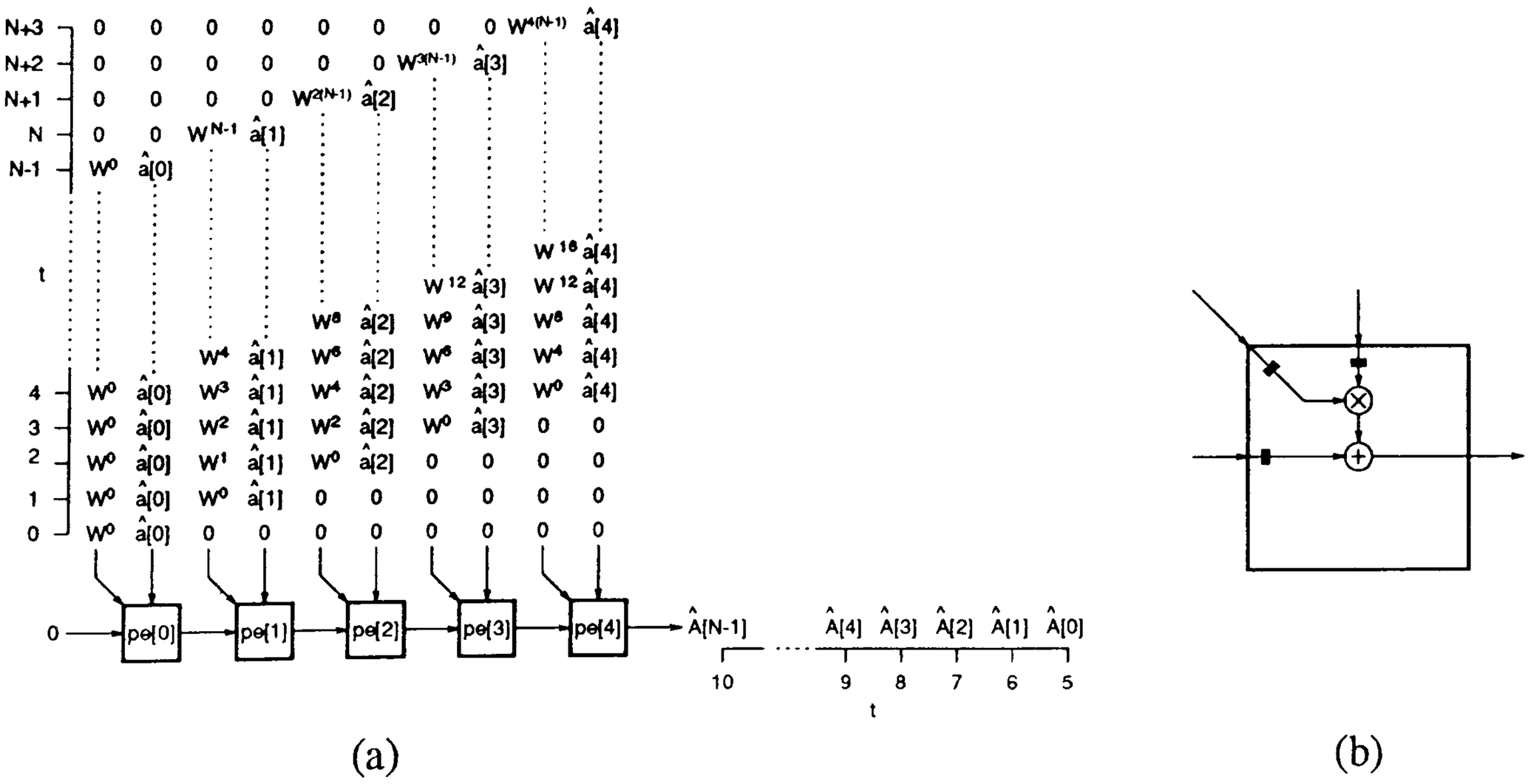


Figure 2.11: (a) systolic array formed by j direction projection (b) PE function.

Other linear DFT arrays are also proposed in [103][104][105][106] but these have the disadvantage that the number of PEs is proportional to N which would lead to high cost for this application. Higher dimensioned arrays are used to improve throughput [107][108][109][110] however the linear array proposed here provides enough concurrency for the blood-flow estimation problem based on typical times for bit-serial arithmetic processing [111][112]. Attempts have been made to localise the FFT computation [19][113][114][115][116] for example Willey et al. [117] develop a systolic elevator concept to replace the global communications with a number pipelined shifts. However complex control requirements in the FFT arrays lead to preference of the array shown in figure 2.11.

(D) Use of Background Research in this Study

The first half of the thesis is devoted to the design of problem size dependent systolic arrays for the matrix element, filter parameter and PSD computations within the Modified Covariance method. Of particular relevance are the previously proposed decomposition systolic arrays, some of which are modified, to reduce cost or to improve throughput, for comparison with systolic arrays formed by alternative DDG mappings. Due to the disadvantages with the error analysis schemes reviewed, an error analysis of a fixed-point scheme, which utilises scale factors to improve dynamic range, is used in the selection of optimal word-length data in the spectral estimator.

The rest of the thesis discusses problem size independent systolic arrays. Further partitioning the chosen systolic array for the matrix element calculation is used in the design of a programmable model order systolic array. The problems highlighted with the spiral systolic array presented by Navarro et al. [28] are addressed by using DDG methods, as opposed to triangular block and dense to band matrix methods, to design a superior systolic array. A novel spiral-type systolic array for Cholesky decomposition is then designed using a similar partitioning but a different reindexing scheme to that used for the LU array.

Chapter 3

Fixed Model Order - Covariance Matrix Element Calculation

3.1 Introduction

Section 2.3.4(E) reviews the Modified Covariance method and shows the partitioning of the calculation into a number of distinct sections. This chapter describes the design of four systolic arrays which may be used for the real-time implementation of the first and most computationally demanding part of the model order $p = 4$ estimator in which the covariance matrix elements are computed.

The classic data dependence graph (DDG) mapping technique developed by Kung [16] is used to produce the systolic arrays. The DDG technique involves translation of the initial algorithm into diagrammatical format where the individual nodes on the graph represent single assignment operations which occur in the algorithm. Multiply accumulate is one such single assignment operation which forms the basis of the computation considered here. Nodes are interconnected by arcs which represent data communication. In the example of a series of multiply accumulations the result of an operation affects that of the next. This is shown on the localised graph as a directed arc between the output of one node and the input of the adjacent node. After pipelining the communication on the DDG a

space and time indexing function can be applied to the DDG in order to produce a systolic array. This involves selection of a projection direction such that a number of nodal operations along a certain channel of the graph are assigned to a particular processor. The timing function schedules the order in which the nodal operations are performed by their associated processor and is constrained by the data dependencies.

The systolic arrays are compared with regard to their efficiency and suitability to VLSI implementation. This involves consideration of hardware cost estimate, control complexity, communication burden and timing constraints.

3.2 Computational Burden

Section 2.3.4(E) details the equations for the matrix element calculation in the Modified Covariance method. Equations (2.23) and (2.24) show that p^2+p matrix elements need to be calculated for the covariance matrix C (2.23) and RHS vector B (2.24). The calculation of elements $c[j,k]$ (2.25) in matrix C is split into the sum of two other elements $c_1[j,k]$ (2.26) and $c_2[j,k]$ (2.27). The calculations of c_1 and c_2 each require $N-p$ multiply accumulate operations. Hence for each window of data input into the spectral estimator a total number of M multiply accumulates would be required to calculate all of the covariance matrix elements, where:

$$M = 2(p^2+p)(N-p) \tag{3.1}$$

N can range up to 256 samples for a window length of 5ms, corresponding to a maximum sampling frequency f_s of 51.2kHz, leading to a maximum of just over two million multiply accumulate operations per second for this calculation when model order $p=4$ is considered. This represents a substantial processing task which can be partitioned onto systolic arrays of simple processing elements. The computational burden can also be reduced further by considering symmetries which are present in C and this is discussed in the design of the systolic arrays.

3.3 Systolic Array Design

This section describes the DDG design of four systolic arrays for the calculation of the covariance matrix elements in the Modified Covariance estimator of model order $p = 4$. The first design presented is used to introduce in further detail the DDG mapping methodology and results in a systolic array with a two dimensional architecture. The attributes of this array are studied and the design of the DDG is modified in order to improve performance. This results in the formation of a tri-linear systolic array. Further optimisation is then considered by separating the calculation into two pipelined sections. This produces a bi-linear systolic array which is used to feed data into a sorting network. A fourth design attempts to exploit the best features of the tri-linear and bi-linear arrays resulting in a partitioned multiply accumulate array.

3.3.1 Two Dimensional Systolic Array Design

(A) Data Dependence Graph Design

When model order $p = 4$ is considered a square 5 by 5 matrix C', which contains all the required $c[j,k]$ elements for the determination of the covariance matrix C (2.23) and RHS vector B (2.24), can be formed:

$$C' = \begin{bmatrix} c[0,0] & c[0,1] & c[0,2] & c[0,3] & c[0,4] \\ c[1,0] & c[1,1] & c[1,2] & c[1,3] & c[1,4] \\ c[2,0] & c[2,1] & c[2,2] & c[2,3] & c[2,4] \\ c[3,0] & c[3,1] & c[3,2] & c[3,3] & c[3,4] \\ c[4,0] & c[4,1] & c[4,2] & c[4,3] & c[4,4] \end{bmatrix} \tag{3.2}$$

Similar matrices C_1 and C_2 containing elements $c_1[j,k]$ (2.26) and $c_2[j,k]$ (2.27) respectively may be formed. The matrix C_1 can be viewed in terms of a recurrence variable $C_1^{(n)}$ comprised of elements $c_1^{(n)}[j,k]$ defined for $(0 \leq j,k \leq 4)$ by:

$$c_1^{(n)}[j,k] = \begin{cases} 0 & n = 3 \\ c_1^{(n-1)}[j,k] + x[n-j].x[n-k] & 4 \leq n \leq N-1 \end{cases} \tag{3.3}$$

leading to the assignment:

$$C_1 = C_1^{(N-1)} \tag{3.4}$$

By considering a square j versus k plane of nodes as shown in figure 3.1(a) a DDG for the calculation of $C_1^{(n)}$ from $C_1^{(n-1)}$ can be derived. Since the index $[n-j]$ is independent of k, then, for a particular value of j, the input samples $x[n-j]$ will be common to all nodes across row j of the graph. This is known as a global broadcast of $x[n-j]$ and is shown on the graph by the horizontal lines connecting the nodes. For example $x[n]$ is input to the graph and is transmitted simultaneously to all nodes in the top row where $j = 0$. Similarly $x[n-k]$ is globally transmitted along column k of the graph. Each node of the graph therefore has an $x[n-j]$ and an $x[n-k]$ input whose product is added to the recursion input $c_1^{(n-1)}[j,k]$ to produce $c_1^{(n)}[j,k]$ as defined in equation 3.3 ($4 \leq n \leq N\text{-}1$). The node function is denoted in figure 3.1(b).

A DDG for the calculation of C_1 can be formed by connecting together the $C_1^{(n)}$ graphs with substitution of n from 4 through to $N-1$. The three dimensional DDG shown in figure 3.2 therefore contains $N-4$ layers, nearest plane has $n = 4$ and n is incremented for consecutive planes going into the paper up until the last

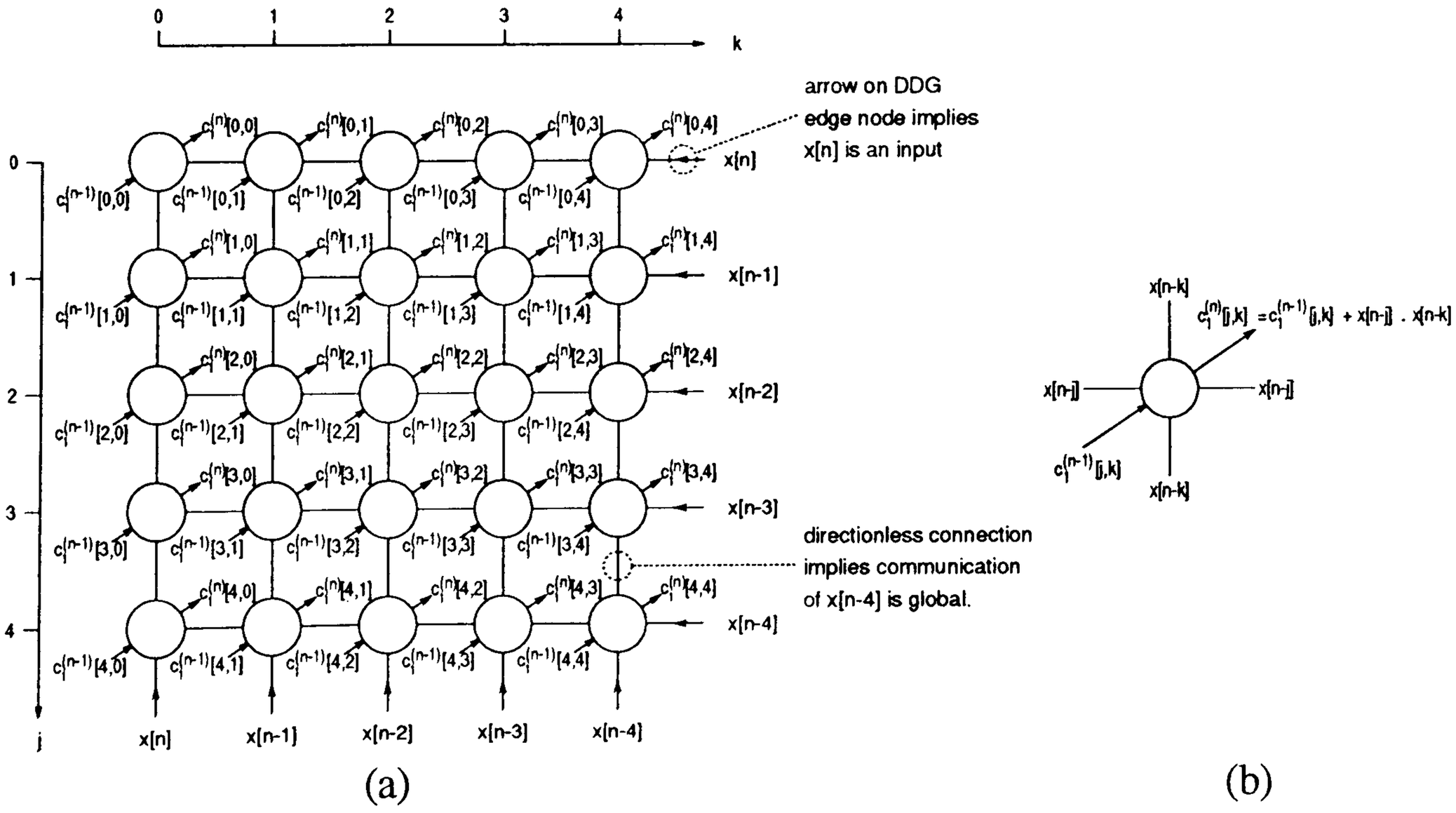


Figure 3.1: (a) DDG for $C_1^{(n)}$ $(p \leq n \leq N-1)$, (b) node function.

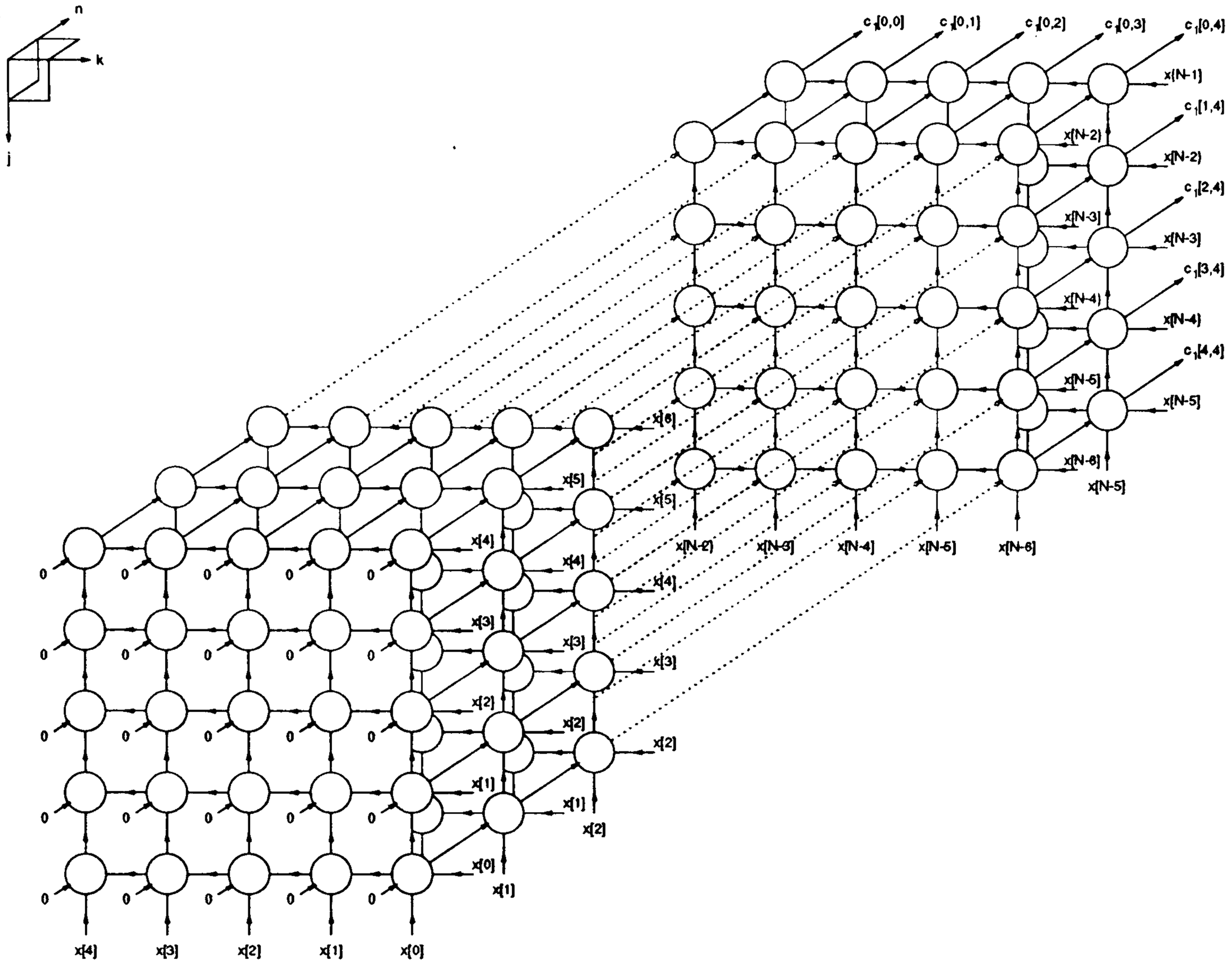


Figure 3.2: Localised DDG for calculation of C_1.

layer where $n = N-1$. When the layers are connected together in such a manner pipelines representing the recursive updating of the elements $c_1^{(n)}[j,k]$ are set up in the n direction. This means that the n direction output from $node[j,k,n]$ will be input to $node[j,k,n+1]$. The pipelines are initialised by the input of zeroes, equivalent to $C_1^{(3)}$ for $n = 3$ (3.3), to the nearest plane and the matrix C_1 is output from the last layer as in the assignment (3.4).

(B) Data Dependence Graph Localisation

Localised communication where a processor only communicates data with its neighbours is one of the key features which make systolic arrays very amenable to VLSI implementation [16]. It is therefore necessary to localise or pipeline all of the communication in the DDG before formation of the systolic array since global DDG links would result in a semi-systolic array [82].

The global communication of the $x[n-j]$ and $x[n-k]$ inputs in figure 3.1 therefore needs to be pipelined. The pipelining of this data is shown by the directional arcs on the horizontal and vertical connections between the nodes in figure 3.2. The directions of the pipelines are chosen by consideration of input data ordering as $x[0]$ is available before $x[1]$ etc. Formally $x[n-j]$ data is passed leftward from $node[j,k,n]$ to $node[j,k-1,n]$ and $x[n-k]$ data is transmitted upwards from $node[j,k,n]$ to $node[j-1,k,n]$. The DDG is now fully localised.

(C) Data Dependence Graph Timing Function

The timing function sets up the input/output scheduling and determines the order in which the systolic processors perform tasks defined by nodes of the DDG. The function is limited by pipelining restrictions within the array which define the ordering of a set of tasks. Data is pumped through a systolic array on a global clock cycle. On each successive clock cycle a PE performs a task defined by the next node pipelined in the projection direction. The execution of any DDG node computation must therefore be complete within the clock period. In the text array timing functions are denoted in terms of clock cycles.

The timing function used in the DDG is $t[j,k,n] = 4-j-k+n$, which can also be represented in the form of a scalar and vector as $4+[-1,-1,1]$. The scheduling of the node operations, illustrated by the internal node numbering in figure 3.3, inserts single pipeline delays in each of the arcs of the DDG. The $-j$ term of the timing function arises from the pipelining direction of the $x[n-k]$ inputs where data is pumped up the columns of DDG nodes in the $-j$ direction. So for example due to the inclusion of an offset of 4 in the timing function then at $t=0$ $x[0]$ is input to $node[4,4,4]$ and this word is passed onto $node[3,4,4]$ at $t=1$. Similarly the $-k$ part of the expression stems from the pipelining of $x[n-j]$. The $+n$ term means that a $node[j,k,n]$ is executed a clock cycle before $node[j,k,n+1]$ as defined by the pipelining of $c_1^{(n)}[j,k]$.

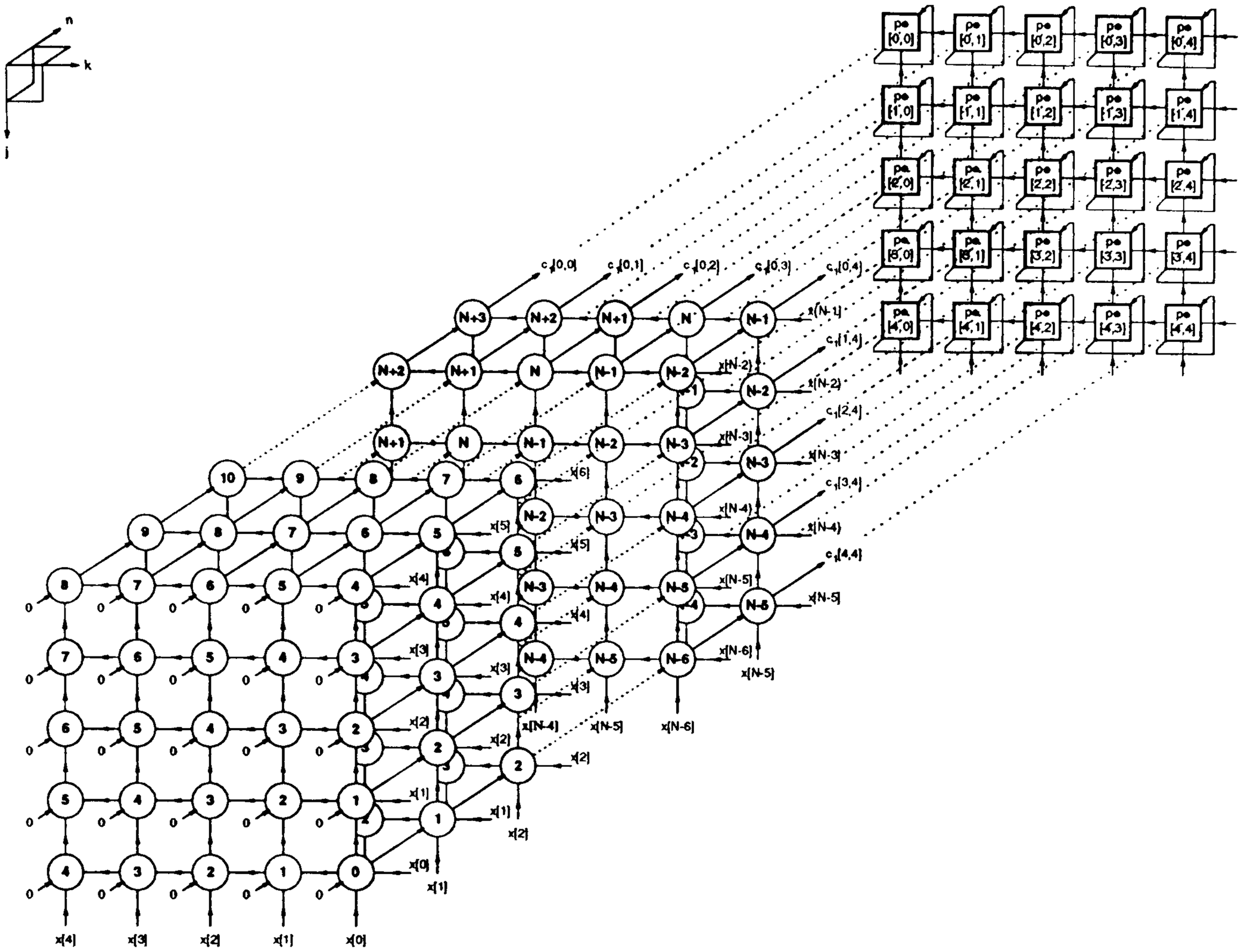


Figure 3.3: Projection of time indexed C_1 DDG in the n direction.

(D) Data Dependence Graph Allocation Function

The allocation function determines which nodal operations are assigned to a particular processor, providing a spatial mapping of tasks from the DDG into the systolic array. The allocation function can be applied by considering linear projection of the DDG. There are many options available to the designer when choosing a projection direction and so a variety of systolic arrays can be formed from a single DDG. The main factors governing the choice of a projection direction are the hardware cost and speed of the resulting systolic array. For example a highly parallel array may result in fast operation in comparison with an array containing fewer processors.

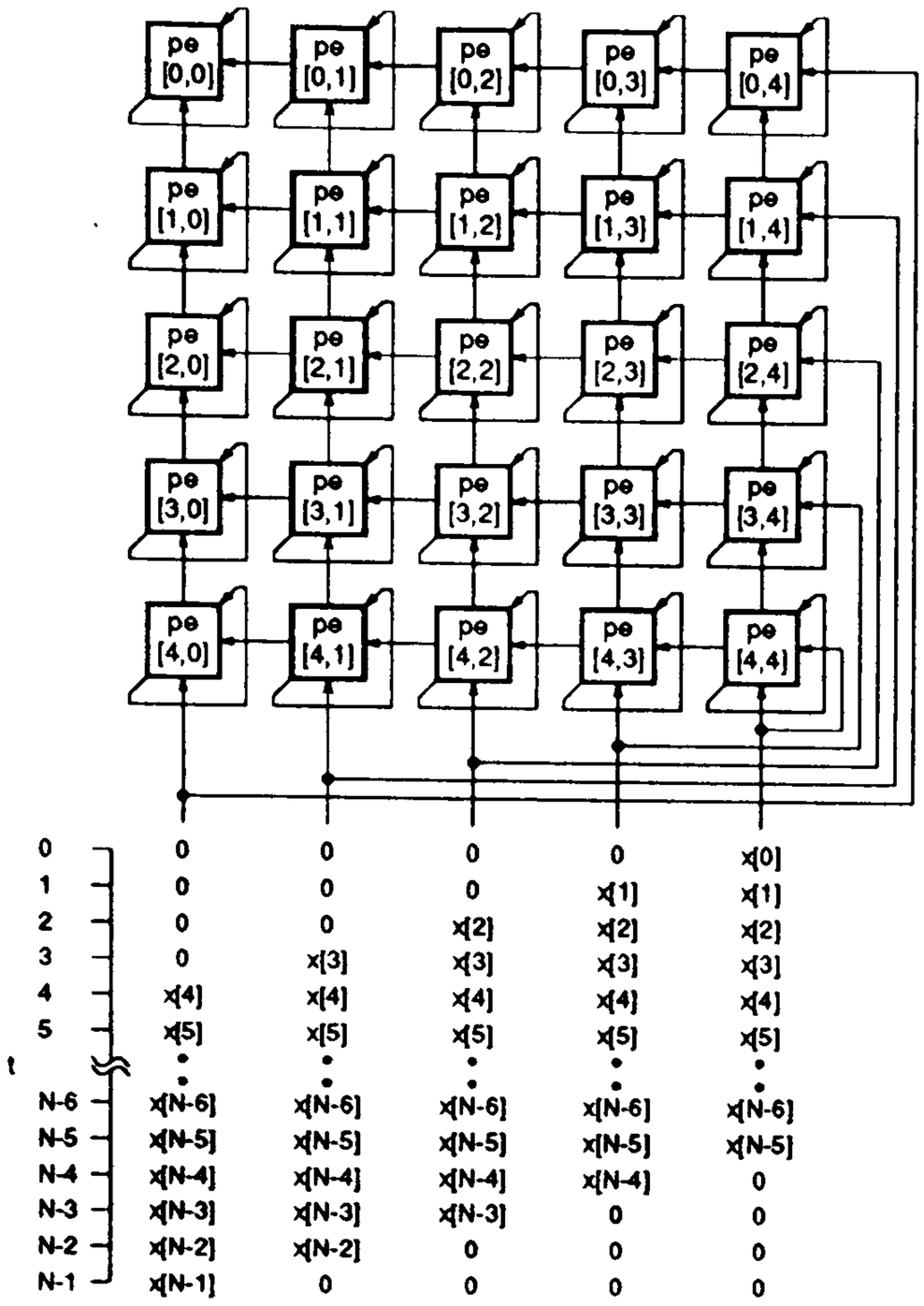
Given that the number of samples in a data window N is likely to be of the order

of 64 to 256 it is fairly obvious that the DDG should be projected along the n direction ($[0,0,1]$ vector) into a square systolic array containing 25 PEs. If the graph is projected along any other direction then the number of PEs would be dependent on N leading to huge arrays. For example systolic arrays produced by projection in either the j or k directions would contain $5(N-4)$ processors. Figure 3.3 shows the n direction projection of the DDG. The processors of the systolic array are shown as square elements as opposed to the circular nodes of the DDG. Each processor is given a reference name $pe[j,k]$ according to their coordinate position so that for example any $node[0,0,n]$ $(4 \leq n \leq N-1)$ is mapped into $pe[0,0]$.

The communication arcs are also mapped into the systolic array. Since the horizontal and vertical arcs are orthogonal to the projection direction then these communications are represented by respective horizontal and vertical data buses in the systolic array. The n direction arcs however point in the same direction as the projection vector. The result of the mapping is that these links are shown as feedback data buses around the PEs of the systolic array.

(E) Systolic Array for Calculation of C_1

The systolic array for the calculation of C_1 with its input scheduling is shown in figure 3.4(a). The input scheduling is derived by looking at the operation times of nodes on the bottom and right hand faces of the DDG in figure 3.3. An advantage of the pipelining directions which were chosen for $x[n-j]$ and $x[n-k]$ now becomes apparent since the lower border inputs to the systolic array are mirrored along the right hand edge and so array input burden can be reduced by a factor of two. Also with the use of multiplexors on the inputs to the lower edge then all of the five different data sequences can be derived from a common stream. The source of this stream is an analogue to digital converter which outputs an ordered time sequence of sampled data starting with $x[0]$ and ending with $x[N-1]$.



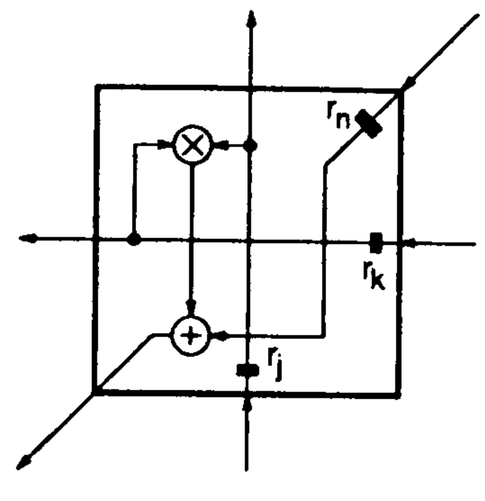


Figure 3.4: (a) Systolic array for calculation of C_1, (b) processor function.

The processor construction is displayed in figure 3.4(b). It contains a multiplier, an adder and three systolic delay registers. The registers r_j, r_k and r_n are connected to the PE inputs and have common clock and reset signals which have not been included on the diagram. On the active clock edge the $x[n-j]$ input on the horizontal bus is stored in r_k and the vertical bus input $x[n-k]$ is transferred to r_j. The product of this data is then added with the word stored in register r_n. Due to the feedback around the PEs the content of r_n is the result of the addition on the previous clock cycle. On successive clock cycles the products are accumulated and the data stored in r_j and r_k is passed onto neighbouring PEs to the right and above respectively.

A similar systolic array can be designed for the calculation of the matrix C_2. There is a difference however in that the pipelining direction for the $x[n+j]$ input is in the $+k$ direction and that of the $x[n+k]$ input is in the $+j$ direction. The inputs here can also be routed from the input data streams of the C_1 array.

(F) Systolic Array for Calculation of C'

The systolic arrays for C_1 and C_2 can be merged so that each PE requires two multiply accumulate units plus an extra adder which is used to add the final values of C_1 and C_2 together to form C' [29]. The matrix C' has symmetrical properties that can be used to reduce the computational burden of the problem. Matrix element values are reflected across the main diagonals which are drawn from $c[0,0]$ to $c[4,4]$ and through $c[4,0]$ to $c[0,4]$ separating C' into quadrants. For the diagonal running from $c[0,0]$ to $c[4,4]$ it follows from (2.26) and (2.27) that for elements lying either side of the diagonal, $c[j,k] = c[k,j]$, whilst for the other diagonal from $c[4,0]$ to $c[0,4]$ it is observed for elements not lying on this diagonal that $c[j,k] = c[4-k, 4-j]$. For example the 9 elements in the lower quadrant, $c[4,0]$, $c[4,1]$, $c[4,2]$, $c[4,3]$, $c[4,4]$, $c[3,1]$, $c[3,2]$, $c[3,3]$ and $c[2,2]$, only need to be calculated. Due to the matrix symmetry the number of PEs can be reduced down from 25 to 9 as shown in figure 3.5. Note that certain pipeline delays from the PEs that are removed from the array must be kept to preserve the correct scheduling of input data.

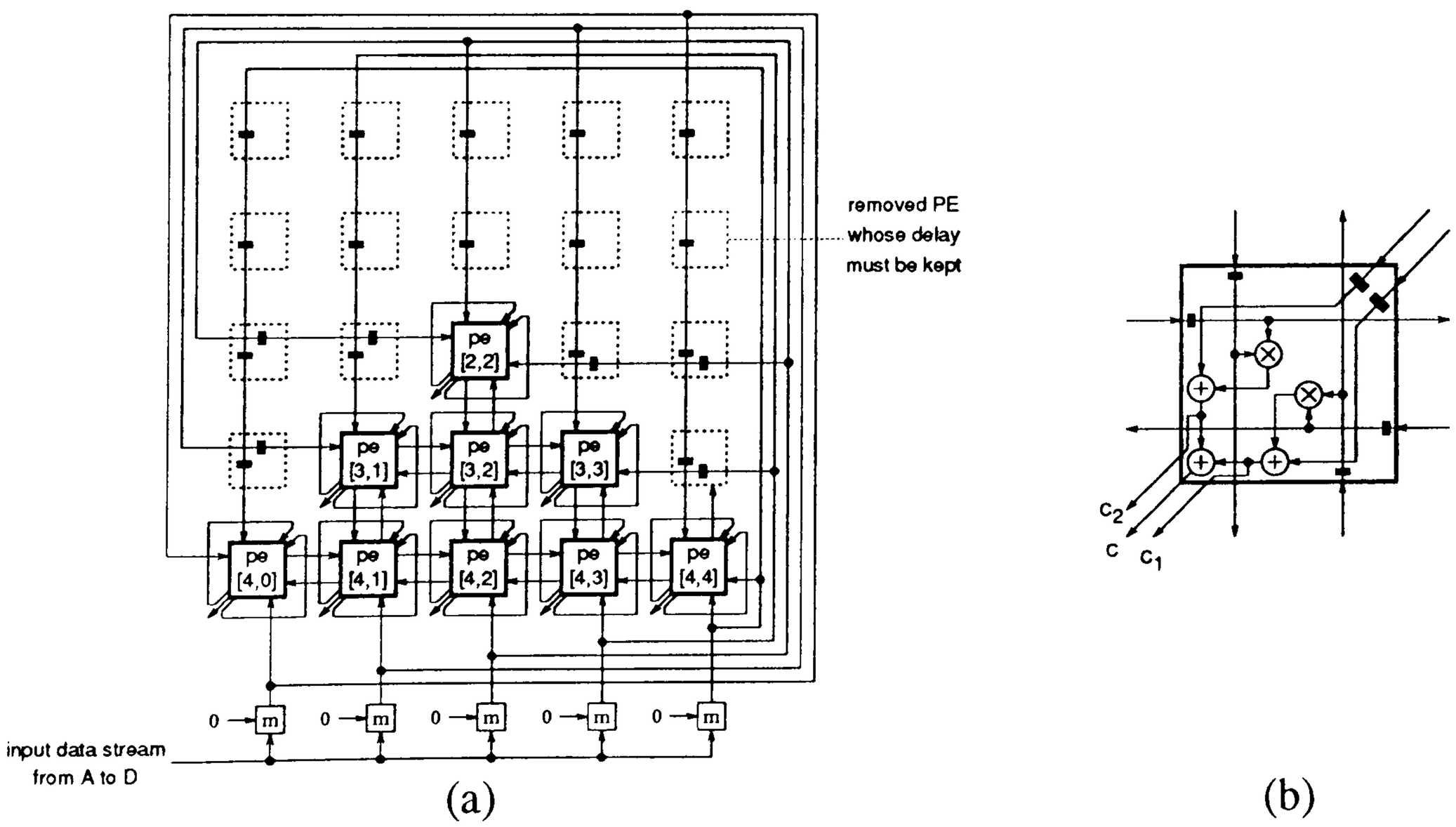


Figure 3.5: (a) Systolic array for calculation of C', (b) processor function.

The two dimensional systolic array described in section 3.3.1 suffers from several problems. Its main drawback is hardware cost. Each PE contains two multiply accumulate units, one for calculation of c_1 (2.26) and the other for c_2 (2.27). Study reveals that the majority of multiplications performed in each of these two algorithms are common for $N \gg p$ as is the case when $p = 4$. As a consequence of this the use of two separate multipliers is very inefficient. Another problem with the array is that the two dimensional style of the architecture makes data retrieval of the computed matrix elements $c[j, k]$ difficult, as these are left in their respective PEs at the end of the calculation. The addition of extra data buses from each PE to an edge of the array would lead to an untidy architecture. The output data buses would crossover existing PE interconnections increasing the number of circuit board layers required and this would not conform to the localised nature of a systolic architecture. A solution to this is to multiplex the output data onto existing localised connections but this has the drawback of increasing hardware cost and extra clock pulses would be necessary.

3.3.2 Tri-Linear Systolic Array

With the inadequacies of the two dimensional array in mind, an alternative DDG is derived in this section leading to the formation of a tri-linear systolic array.

(A) Data Dependence Graph Design

A different method of partitioning the matrix element calculation is explored where the simplifications arising from the symmetry of C' (3.2) are taken into account at the start of the design process. As stated previously only 9 of the 25 elements in C' need to be calculated and correspondingly 9 of the matrix elements in each of C_1 and C_2 are required. Considering C_1, the elements $c_1[j, k]$ within its left hand quadrant can be split into three groups with $k = 0$ $(0 \leq j \leq 4)$, $k = 1$ $(1 \leq j \leq 3)$ and $k = 2$ $(j = 2)$.

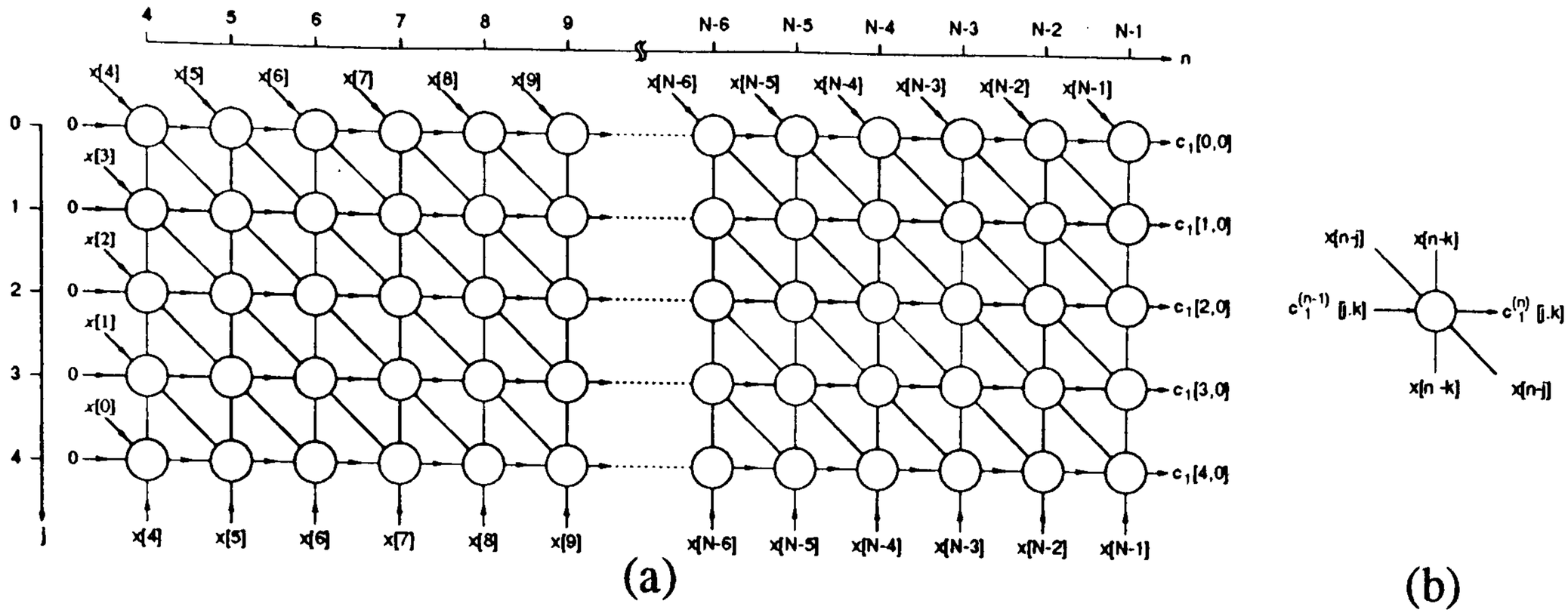


Figure 3.6: (a) DDG for $c_1[j,k]$ for $0 \leq j \leq 4$, $k = 0$, (b) node function.

Separate DDGs can be designed for each group by relating the matrix elements $c_1[j,k]$ to a recurrence variable $c_1^{(n)}[j,k]$ as before (3.3). Firstly consider the situation when $k = 0$. Since k is held constant the data dependencies from the calculations of $c_1[j,0]$ $(0 \leq j \leq 4)$ can be plotted on the j versus n plane as illustrated in figure 3.6. In order to form $c_1[j,0]$ the product of $x[n-j]$ and $x[n-k]$ must be accumulated over n ranging from 4 to $N-1$. Input data $x[n-j]$ is globally transmitted to the diagonals while there is a global communication of $x[n-k]$ across the columns of the graph. The products of the $x[n-j]$ and $x[n-k]$ inputs are accumulated along each of the rows of the DDG and this is indicated by the horizontal local interconnects. Figure 3.7 shows the DDG for $c_2[j,0]$ on the j versus n plane. The calculation here involves the accumulation

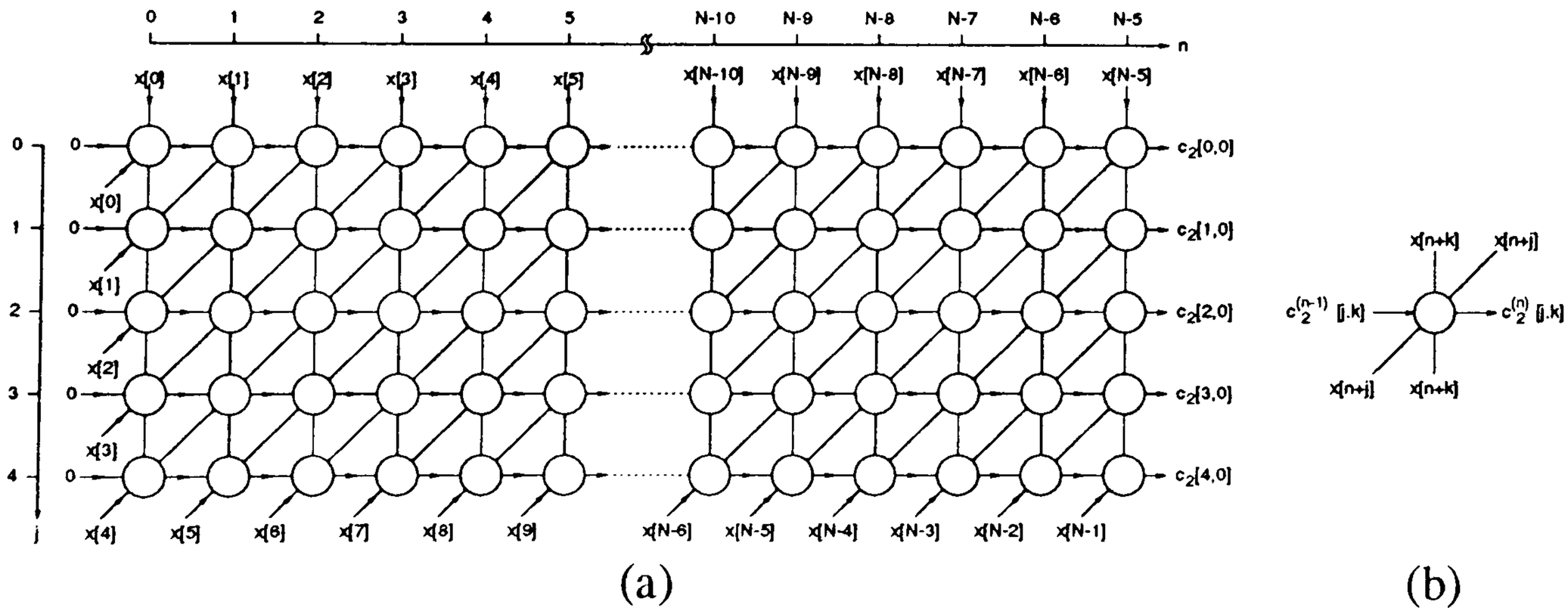


Figure 3.7: (a) DDG for $c_2[j,k]$ for $0 \leq j \leq 4$, $k = 0$, (b) node function.

of the products of $x[n+j]$ and $x[n+k]$. The graph is very similar to that for $c_1[j,0]$ (figure 3.6) except for the range of n from 0 to $N-5$ and the $x[n+j]$ data is broadcast along the diagonal which is perpendicular to the diagonal on which $x[n-j]$ is broadcast.

Study of the DDGs in figures 3.6 and 3.7 reveals that some products calculated in one graph are repeated in the other. The DDGs for $c_1[j,0]$ and $c_2[j,0]$ may be merged together to see if the number of multiplications can be reduced. If nodes from each graph which produce the same product are coincidental the merged node function can be implemented by just one multiplication and a binary shift. Overlapping the graphs as they stand results in a complicated graph as two diagonal communications for $x[n-j]$ and $x[n+j]$ are required. However the vertical $x[n]$ $(4 \leq n \leq N-5)$ data buses can be combined but similar products only coincide in the $c[0,0]$ computation.

The graph in figure 3.7 can be transformed to that shown in figure 3.8 by expressing $c_2[j,0]$ in terms of products of $x[n]$ and $x[n-j]$ as opposed to $x[n+j]$ and $x[n]$ since:

$$c_2[j,0] = \sum_{n=0}^{N-5} x[n+j].x[n] = \sum_{n=j}^{N-5+j} x[n].x[n-j] \tag{3.5}$$

The altered graph is effectively a skewed version of the old one with row j of nodes shifted by j positions along the n-axis.

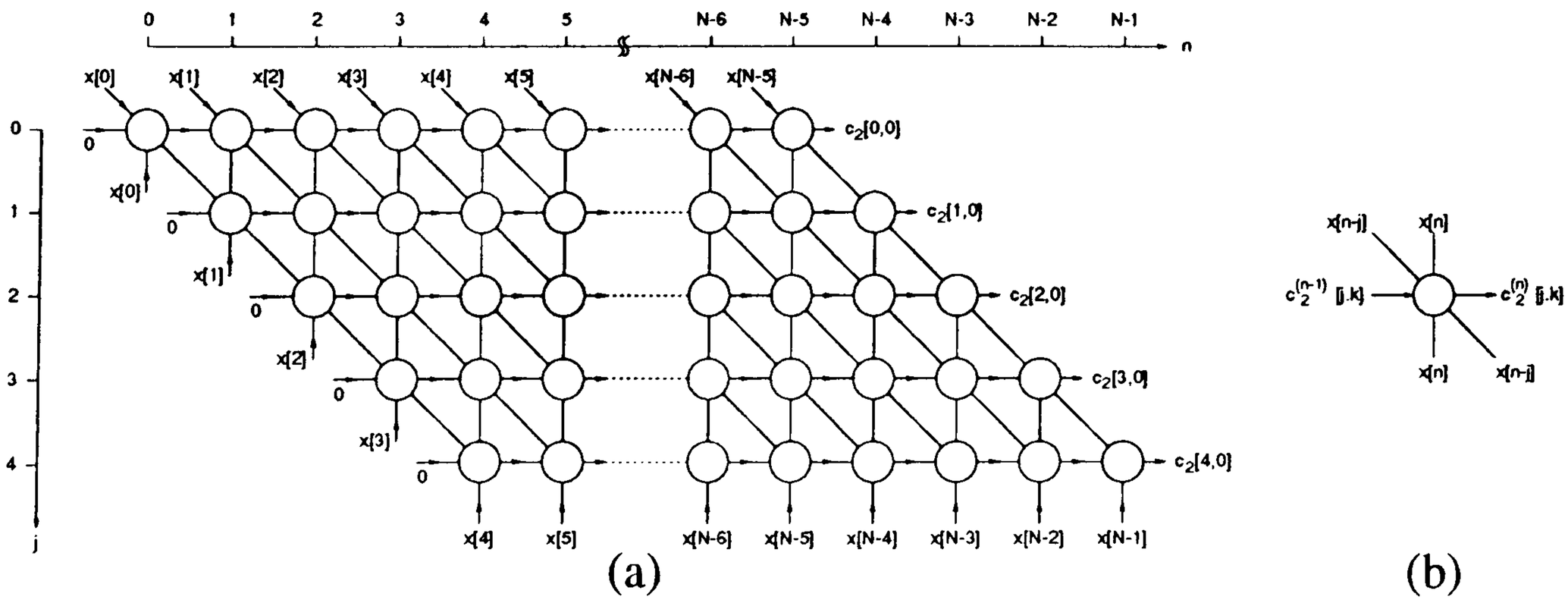
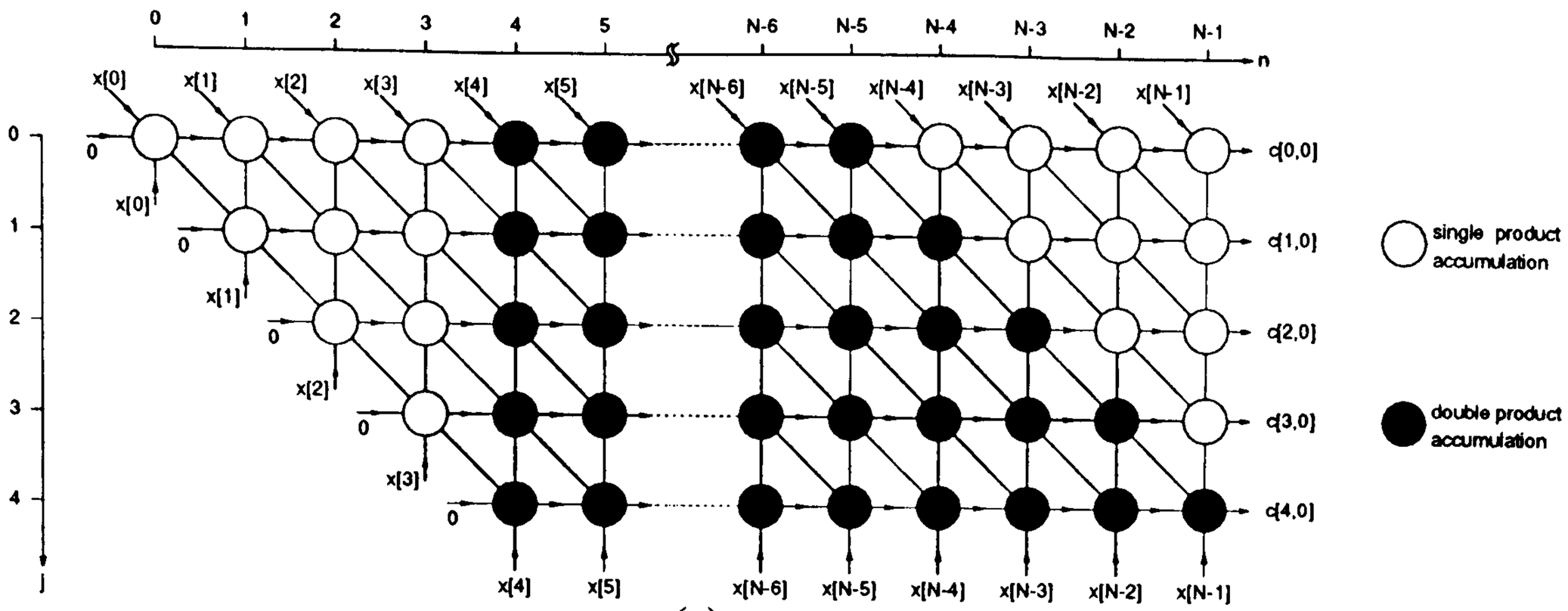


Figure 3.8: (a) Skewed DDG for $c_2[j,0]$, (b) node function.



(a) (b)

Figure 3.9: (a) Merging of DDGs in figures 3.6 and 3.8, (b) node functions.

Now combining the DDGs of figures 3.6 and 3.8 produces a much neater merged graph for the computation of $c[j,0]$ as shown in figure 3.9. All the global communication lines which carry the same input data may now be merged. The shaded cells represent the regions of overlap for the two arrays and due to (2.25) the overlapping nodes are adapted to form twice the product. DDGs for $c[j,1]$ $(1 \leq j \leq 3)$ and $c[2,2]$ can be derived using the same methodology with the resulting DDGs being three rows and one row in height respectively.

(B) DDG Localisation, Scheduling and Projection

Before timing and allocation function assignments are made it is necessary to pipeline the communication of $x[n-j]$ and $x[n]$. The pipelining directions are chosen bearing in mind that the DDG should be projected in the n direction due the large value of N as in figure 3.10. In order to respect the time sequence ordering of the input data it is desirable to start accumulating from the left hand edge of the DDG and so the selected direction for $x[n-j]$ is from $node[j,k,n]$ to $node[j+1,k,n+1]$. For $x[n]$ the $-j$ pipeline direction is arbitrarily chosen creating a bi-direction data flow within the systolic array. The timing function illustrated by the internal node numbering is given by:

$$t = 4 - j - k + 2n \tag{3.6}$$

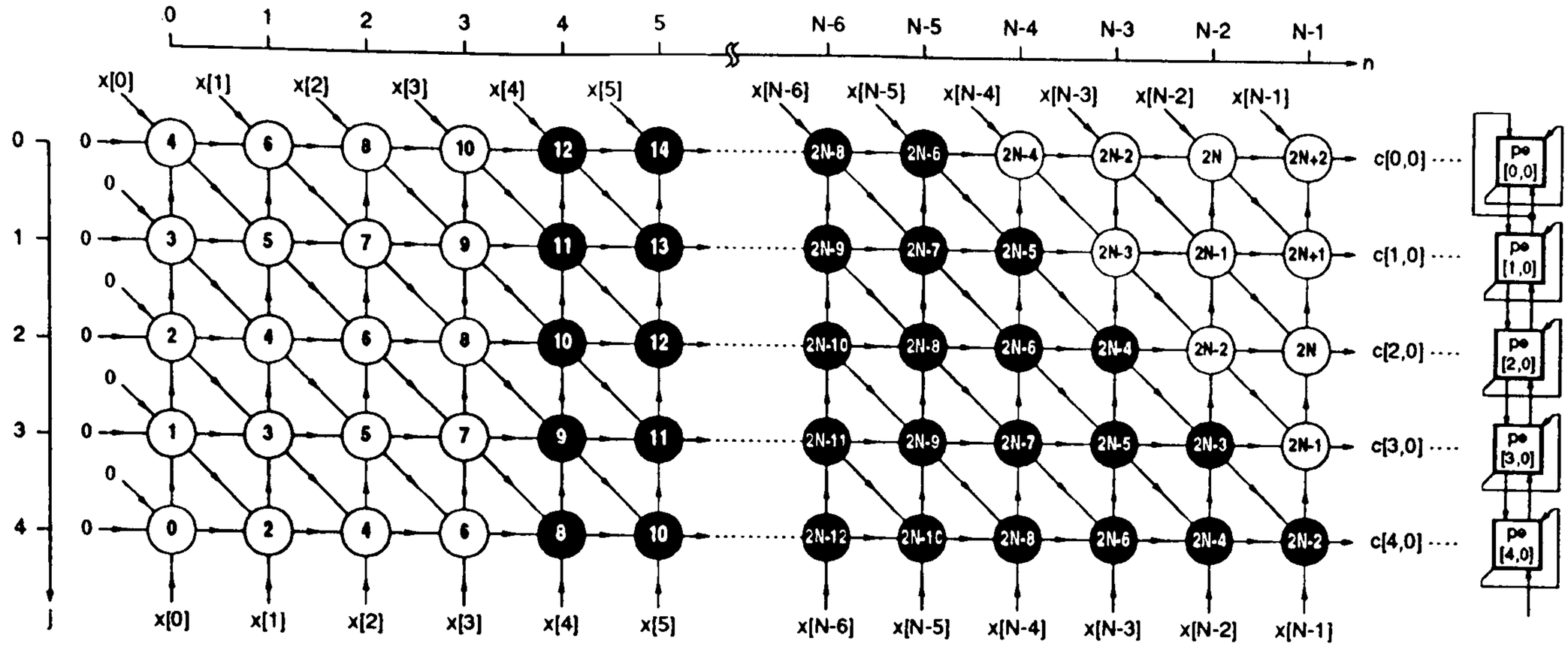


Figure 3.10: Projection of localised DDG for $c[j,0]$ into a linear systolic array.

The $[-1,-1,2]$ schedule vector places single delay registers in the $x[n]$ and $x[n-j]$ pipelines whilst each PE performs a new accumulation every two time steps. Extra nodes are included in the bottom left hand corner of the DDG to set up $x[n]$ data within the systolic array. Since their $x[n-j]$ inputs are held at zero then their function is to simply act as pipeline delays. The same timing and projection can be applied to the $k=1$ and $k=2$ DDGs to produce two more linear systolic arrays.

(C) Systolic Array

The complete tri-linear systolic array is shown in figure 3.11. Data enters the array once in every two clock cycles so that each systolic PE is active for only

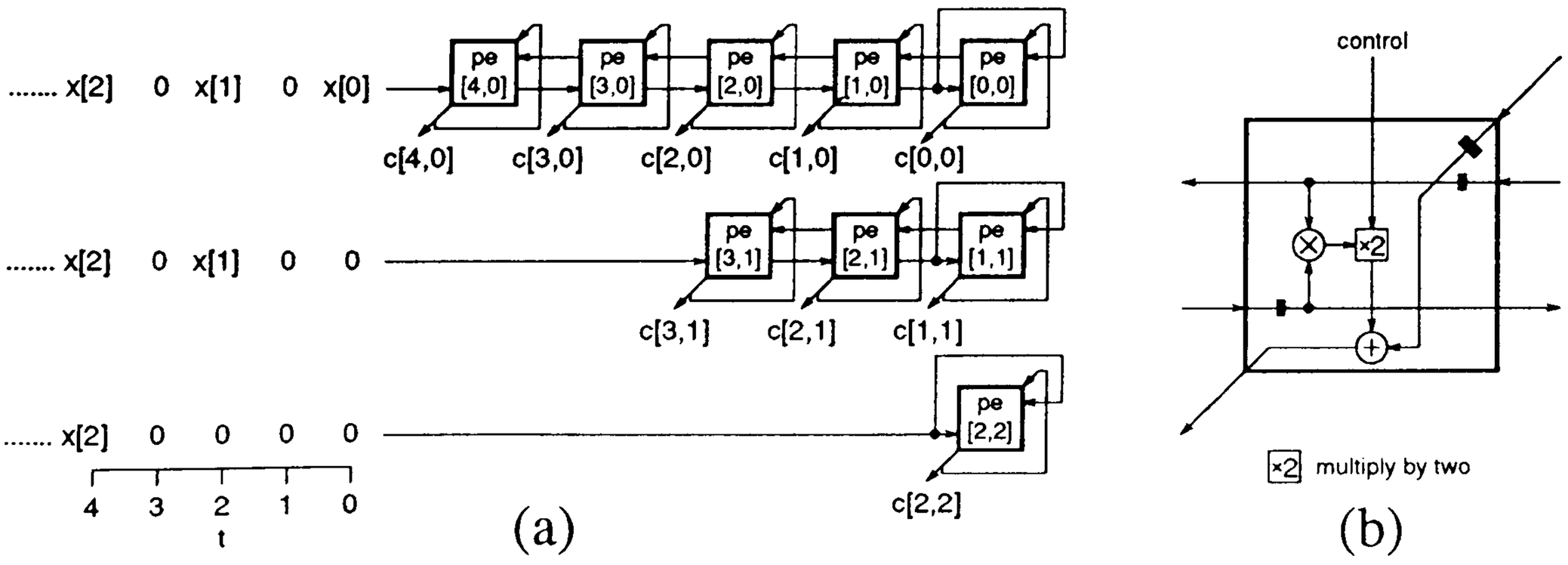


Figure 3.11: (a) Tri-linear systolic array (b) processor diagram.

50% of the time. Each PE accumulates single or double products depending on the state of an input control line. The doubling operation is just a matter of a shift operation in fixed point arithmetic. If zeroes are interleaved between the existent input data then there is only need for a single delay in the accumulator feedback link as the accumulation of the product of two zeroes will not change the stored value. The input data streams $pe[3,1]$ and $pe22$ can both be derived from the $pe[0,0]$ input with the use of multiplexors. At the end of the calculation $c[j,k]$ can be picked off from the feedback loop of $pe[j,k]$.

3.3.3 Bi-Linear Systolic Array

One of the shortcomings with the systolic arrays shown in figures 3.5 and 3.11 is that multiplications are repeated when calculating different matrix elements. For example, all of the multiplications required to calculate $c[2,2]$ are used in the formation of $c[1,1]$. In general it can be said that the majority of products formed in the calculation of $c[j,k]$ overlap with those for $c[j+1,k+1]$. A different way of attacking the problem would be to design a systolic array that accumulates all the desired products without any repetition, enabling the matrix elements to be calculated by selectively picking off the array outputs at appropriate times then adding and subtracting as necessary. Relaxing the adherence to the systolic model to allow a less regular DDG representation of the algorithm it is found that upon examination the number of PEs can be further reduced, resulting in the design of a bi-linear systolic array.

(A) Data Dependence Graph Optimisation

A DDG of similar to that of figure 3.10 but with the double product nodes replaced single product nodes could be used to represent all the necessary permutations of multiplication operations required for the calculation of C' (3.2). For the left hand quadrant of C' considered in the previous section it can be said

that element $c[j,k]$ may be calculated from the accumulated products in row $j-k$ of the graph. Therefore the new DDG is plotted on an axis of $j-k$ versus n. One of the problems with the DDG in figure 3.10 is that once projected the systolic processors are active for only 50% of the time. To alleviate this the graph can be split into two separate graphs the first containing the rows with the even $j-k$ index and the second with the two other rows as shown in figure 3.12. With the use of schedule vector $[-\frac{1}{2}, \frac{1}{2}, 1]$, a new timing function:

$$t = \frac{p}{2} - \frac{j-k}{2} + n \tag{3.7}$$

is applied to the DDG in figure 3.12(a) whilst

$$t = \frac{p}{2} - \frac{j-k+1}{2} + n \tag{3.8}$$

is used in the graph in part (b). The effect of the partitioning and new timing function is to allow accumulation on successive clock steps whilst still maintaining

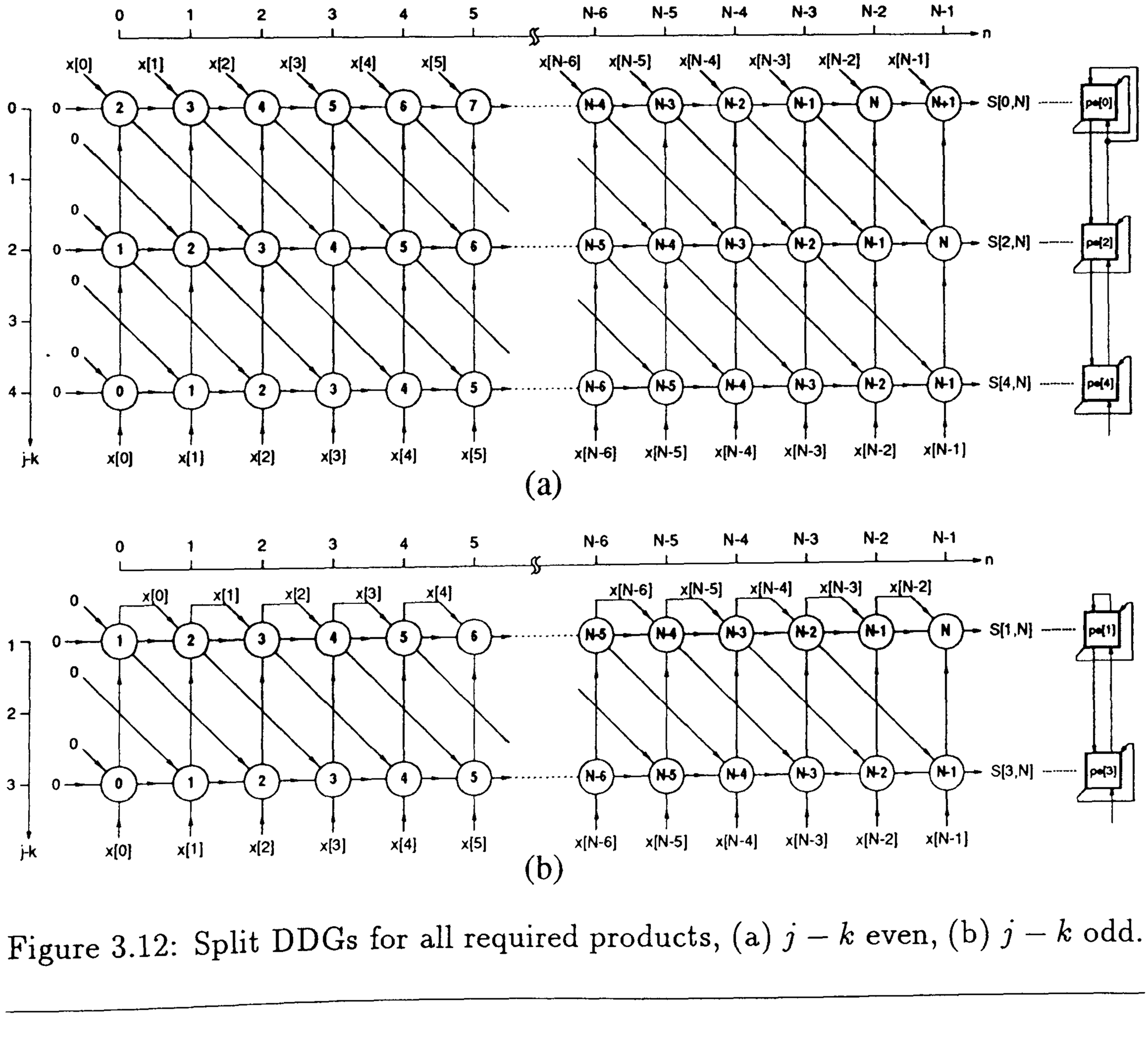


Figure 3.12: Split DDGs for all required products, (a) $j-k$ even, (b) $j-k$ odd.

unit delays in the input data pipelines. There is no longer a need to interleave the input data with zeroes and consequently the efficiency of the array is doubled. The accumulation input to $node[j-k,n]$ is labelled $S[j-k,n]$ for future reference.

(B) Optimised Systolic Array

Figure 3.13 shows the two linear systolic arrays of length two and three PEs formed by projecting the graphs in figure 3.12 along the n direction. The internal PE structure in this array is basically the same as that in the tri-linear array of figure 3.11 except that there is no longer need for the product doubling function. The input data is clocked into the array on clock cycle t and the PE outputs at that time are denoted by $S[j-k,t]$.

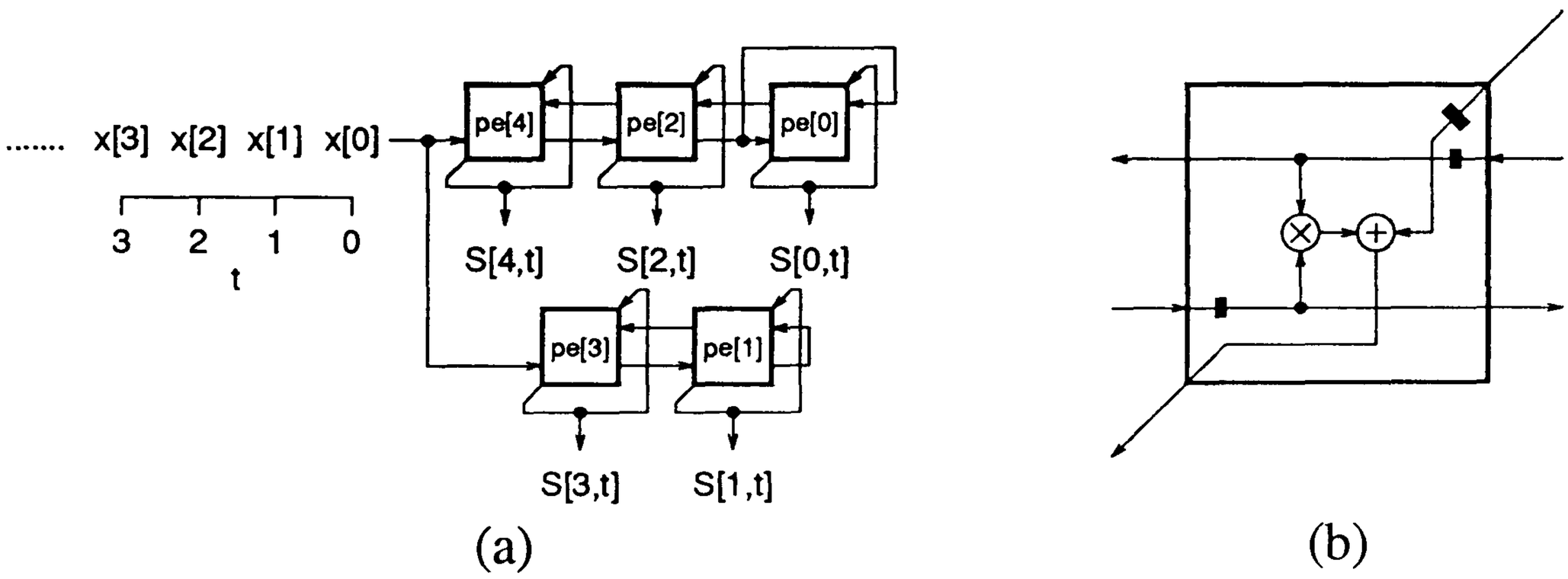


Figure 3.13: (a) Bi-linear systolic array (b) PE

(C) Matrix Element Calculation Using Systolic Array Outputs

Referring back to the graphs in figure 3.12 when dealing the left hand quadrant of C' then $c[j,k]$ may be calculated from $S[j-k,m]$ by addition and subtraction for various values m lying on the n axis. The task is therefore to determine the values of m $(0 \leq m \leq N)$ at which the accumulated product needs to be picked off. $S[j-k,m]$ can be defined by analysis of the graphs in figure 3.12 bearing in mind that $S[j-k,m]$ is defined as the accumulative sum going into $node[j-k,m]$.

The summation sequence can be written out in a general format as follows:

$$S[j-k,m] = \begin{cases} 0 & m \leq j-k \\ \sum_{n=0}^{m-1-j+k} x[n].x[n+j-k] & m > j-k \end{cases} \tag{3.9}$$

In order to find the required values of m for c_1 (2.26) the summation in (3.9) for $m > j-k$ needs to be expressed in terms of the product $(x[n-j].x[n-k])$. This can be achieved by subtracting j from the x indices in (3.9) providing the limits of summation are adjusted appropriately. The definition then becomes:

$$S[j-k,m] = \sum_{n=j}^{m-1+k} x[n-j].x[n-k] \tag{3.10}$$

The lower limit of summation in (3.10) is $n = j$ whilst c_1 is summed from $n = p = 4$. In order to counteract this problem equation (2.26) can be expressed in terms the difference of two separate summations:

$$c_1[j,k] = \sum_{n=j}^{N-1} x[n-j].x[n-k] \; - \; \sum_{n=j}^{p-1} x[n-j].x[n-k] \tag{3.11}$$

To find the values of m, say m_0 and m_1, such that $S[j-k,m]$ accurately represents the calculation of $c_1[j,k]$ the upper summation limit in (3.10) may be equated with the upper limits of the two summations in (3.11):

$$m_0 - 1 + k = N - 1 \qquad m_0 = N - k \tag{3.12}$$

$$m_1 - 1 + k = p - 1 \qquad m_1 = p - k \tag{3.13}$$

Therefore for the general case:

$$c_1[j,k] = S[j-k, \; m_0 \;] - S[j-k, \; m_1 \;] \tag{3.14}$$

$$= S[j-k, \; N-k] - S[j-k, \; p-k] \tag{3.15}$$

For the calculation of $c_2[j,k]$ the product in (3.9) needs to be expressed in terms of $x[n+j]$ and $x[n+k]$. An analysis similar to that above for $c_1[j,k]$ can be used to find an expression for $c_2[j,k]$ in terms of $S[j-k,n]$:

$$c_2[j,k] = S[j-k, \; N-p+j] - S[j-k, \; j \;] \tag{3.16}$$

Equations (3.15) and (3.16) are added together to give $c[j,k]$:

$$\begin{aligned} c[j,k] &= S[j-k, \quad N-k \quad] - S[j-k, p-k] \\ &+ S[j-k, \; N-p+j] - S[j-k, \quad j \quad] \end{aligned} \tag{3.17}$$

The calculation of each matrix element from the systolic array shown in figure 3.13 has therefore been reduced to the addition and subtraction of four values of the accumulative product PE outputs at appropriate times. To find the times at which the specific values m of n are output from the systolic array involves substitution into the timing functions t used for the graphs in figure 3.12. $S[j-k,t]$ refers to the word being clocked into $node[j-k,n]$ at time t. Equation(3.17) can therefore be expressed in terms of time by substitution of (3.7) and (3.8). So for even values of $j-k$:

$$\begin{aligned} c[j,k] &= S[j-k, N+\tfrac{p}{2}-l] - S[j-k, \tfrac{3p}{2}-l] \\ &+ S[j-k, N-\tfrac{p}{2}+l] - S[j-k, \tfrac{p}{2}+l] \end{aligned} \tag{3.18}$$

where:

$$l = \frac{j+k}{2} \tag{3.19}$$

and for odd values of $j-k$:

$$\begin{aligned} c[j,k] &= S[j-k, N+\tfrac{p}{2}-\tfrac{1}{2}-l] - S[j-k, \tfrac{3p}{2}-\tfrac{1}{2}-l] \\ &+ S[j-k, N-\tfrac{p}{2}-\tfrac{1}{2}+l] - S[j-k, \tfrac{p}{2}-\tfrac{1}{2}+l] \end{aligned} \tag{3.20}$$

The matrix elements are therefore calculated by substitution of $p=4$ and the appropriate $[j,k]$ coordinates into (3.18) and (3.20). For example $c[0,0]$, $c[1,1]$ and $c[2,2]$ are derived from output $S[0,t]$ as follows:

$$c[0,0] = S[0,N+2] - S[0,6] + S[0,N-2] - S[0,2] \tag{3.21}$$

$$c[1,1] = S[0,N+1] - S[0,5] + S[0,N-1] - S[0,3] \tag{3.22}$$

$$c[2,2] = S[0,N] \quad - S[0,4] + S[0,N] \quad - S[0,4] \tag{3.23}$$

(D) Implementation of the Sorting PE

Hardware implementation of (3.18) and (3.20) for each of the nine matrix elements in the left hand quadrant of C' is required. Figure 3.14 shows the architecture of a PE that may be used to calculate matrix elements from the output stream of a systolic PE from the array shown in figure 3.13.

The operation of the sorting PE can be demonstrated with reference to (3.21) to (3.23). The first in first out (FIFO) S register bank initially stores the systolic array data output $S[0,t]$ on successive clock cycles from $t = 2$ to $t = 6$. This data is held until $t = N - 2$ when $S[0,2]$ is subtracted from $S[0,N-2]$ to form $c_2[0,0]$ which is stored in the c_2 register. On the next cycle $S[0,N-1]$ is input to the sorter PE, the S register is shifted so that $S[0,3]$ is output and $c_2[1,1]$ is formed from their difference then stored. The procedure is repeated on the following clock pulses and ends when $c_1[0,0]$ is stored in the c_1 register. The c_1 and c_2 registers allow parallel access to their data for selective addition to form the matrix elements $c[0,0]$, $c[1,1]$ and $c[2,2]$.

The same structure may be used for the calculation of the other matrix elements by altering the timing of the register clock enables appropriately. Although, since the storage requirement for the other systolic array outputs is less, this leads to some redundant registers. Hardware could be reduced by using fewer registers, however in VLSI design repeated structures of identical units are desirable leading to a tradeoff between an irregular structure and register redundancy.

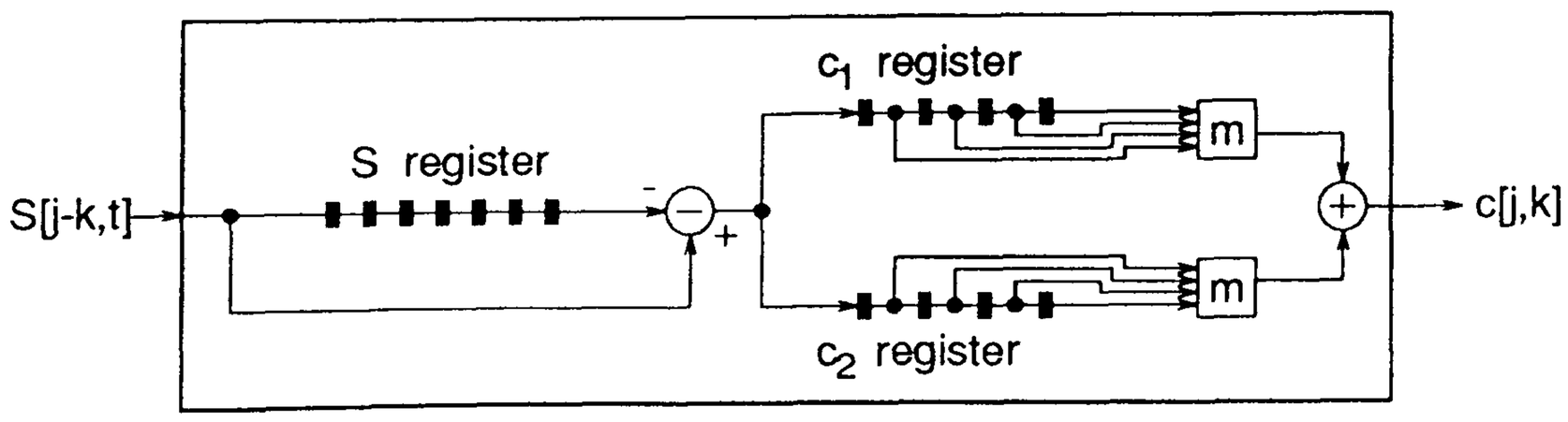


Figure 3.14: Architecture of a sorting PE.

3.3.4 Partitioned Multiply Accumulate (PMA) Array

The tri-linear array in figure 3.11 has the advantage of a basic multiplier accumulator PE configuration but suffers because a relatively large number of these PEs are required. The bi-linear array in figure 3.13 shows improved efficiency over the tri-linear array since repetition of multiplications is eliminated by reducing the number of PEs. However, the design of the sorting section of the bi-linear PE (figure 3.14) is much more complex creating a disadvantage in terms of storage capacity and control signal generation. A systolic array solution which combines the simplicity of the tri-linear approach while attempting to meet the efficiency (in terms of non repetition of multiplications) of the bi-linear design would be superior. The aim here is to produce an array which reduces the sorting required to a minimum while avoiding repeated calculations.

An alternative design approach is to consider multiplications and accumulations on separate DDGs. Once hardware solutions for these two subproblems are derived the merging of the two processes can be considered.

(A) Accumulation DDGs

The accumulations necessary to form the matrix elements can be drawn on a number of DDGs. DDG representations of the accumulations for the calculation of $c[0,0]$, $c[1,1]$ and $c[2,2]$ share the same data inputs $x[n].x[n]$ $(0 \leq n \leq N-1)$ and are therefore illustrated on a common graph in figure 3.15(a). The x product inputs are assumed to be available from a single PE to avoid the repeated multiplications. A similar graph for the calculation of $c[1,0]$ and $c[2,1]$ is presented in part (b) of the figure as consisting of three rows to maintain PE uniformity after projection.

The x product inputs are globally transmitted down the columns of the graphs without transformation and the accumulations are made along the rows of the graphs in the n direction to output the matrix elements from the right hand border. The nodes work in three different modes. In *mode* 0 the nodes perform a straightforward accumulate, *mode* 1 is a data transfer without product addition and in *mode* 2 twice the input product is accumulated to allow for the overlap which arises from the calculations of c_1 (2.26) and c_2 (2.27). In all, five such DDGs are created for the $p = 4$ estimator.

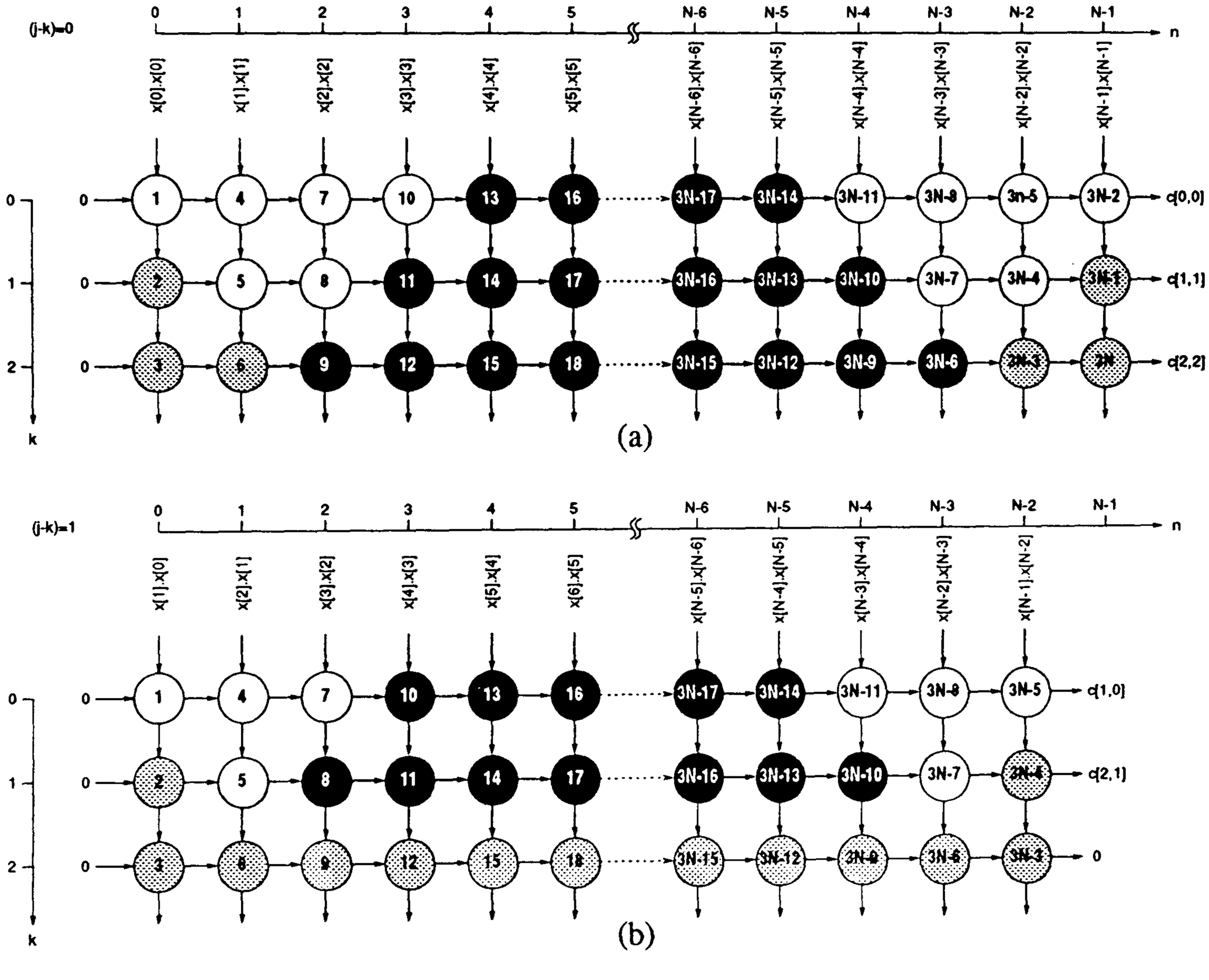


Figure 3.15: Accumulation DDGs for calculation of (a) $c[0,0]$, $c[1,1]$ and $c[2,2]$, (b) $c[1,0]$ and $c[2,1]$, (c) key to node functions.

(B) Multi-projection of the Accumulation DDGs

With the use of multi-projection [23] each of the $p+1$ accumulation DDGs can be projected into a single processor. Since the aim is to map all of the tasks of a DDG into a single PE, each node must be processed in its own exclusive time slot. The timing function:

$$t = k + 3n + 1 \tag{3.24}$$

allows this, creating single delay k direction arcs while the arcs along the direction of n need three pipeline delays. Projection of the DDG in figure 3.15(a) in the direction of k results in the array of figure 3.16(a) where the operation of $node[k,n]$ is processed in $pe[n]$ and whose processor layout is detailed in figure 3.16(b). There are a total of three systolic word delay registers interconnecting each of the PEs. Instead of implementing a wrap around feedback bus for the mapping of the k direction arcs a single input port is sufficient if the input data to these ports is clocked in, once in every three clock cycles.

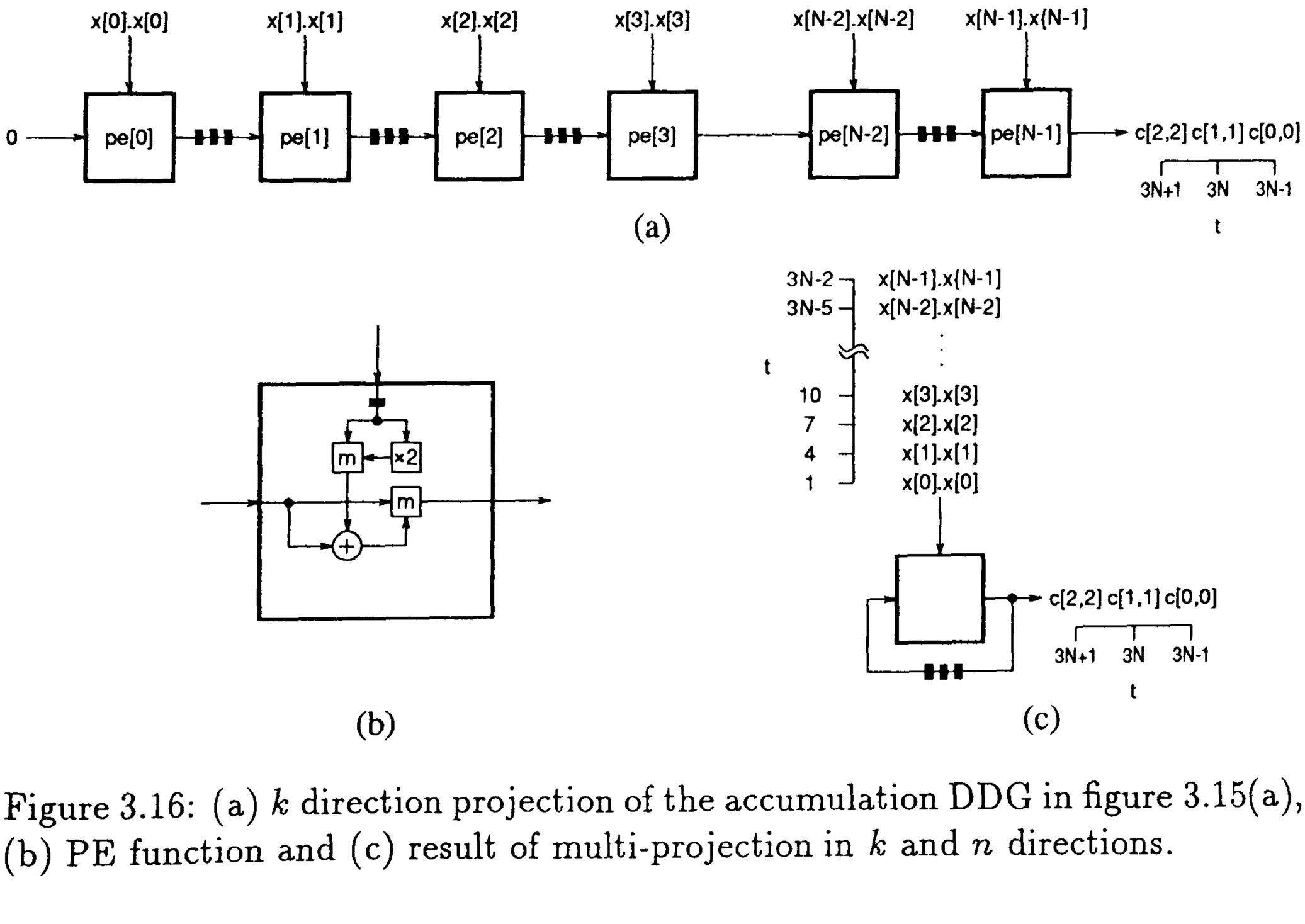


Figure 3.16: (a) k direction projection of the accumulation DDG in figure 3.15(a), (b) PE function and (c) result of multi-projection in k and n directions.

Neighbouring PEs of the array produced by the k projection are active on consecutive blocks of three clock cycles. Projection of this array in the n direction is therefore a possibility as the timing function used saves any of the processor operations from clashing. The result of the multi-projection is the single systolic PE shown in figure 3.16(c), the internal layout of which is still the same as in figure 3.16(b). The PE interconnection buses of the previous array become feedback buses carrying four systolic delays whose contents at any time instant represent the current state of the accumulation of $c[0,0]$, $c[1,1]$ and $c[2,2]$. The initialisation state of these registers is provided by using their reset function.

The same arrangement as that of figure 3.16(c) can be used for the calculation of the other matrix elements but the timing of the control signals and the input products need to be varied accordingly.

(C) Multiplication DDGs

The split DDG approach of section 3.3.3(A) can be used to perform efficiently the multiplications required for supply of the accumulation DDGs. Figure 3.17(a) and (b) are the DDGs which cover all permutations of $x[n].x[n-(j-k)]$ for $(j-k)$ even and odd respectively. The timing allocations of the original array, (3.7) and (3.8) are scaled 3 times as each consecutive product is required to be updated every three clock cycles for input to the accumulation PEs. Usage of the allocation function where $node[(j-k),n]$ is mapped into $pe[(j-k)]$ results in the systolic arrays shown on the right of the DDGs. The timing allocation suggests three register delays on each of the communication buses between PEs. A much simpler and less expensive solution is to opt for a clock signal whose frequency is a third of that on the accumulation feedback registers as with such a signal only single systolic delays are required. The output products of these PEs are fed directly to the accumulation PEs for complete calculation of the matrix elements from the sampled Doppler signal.

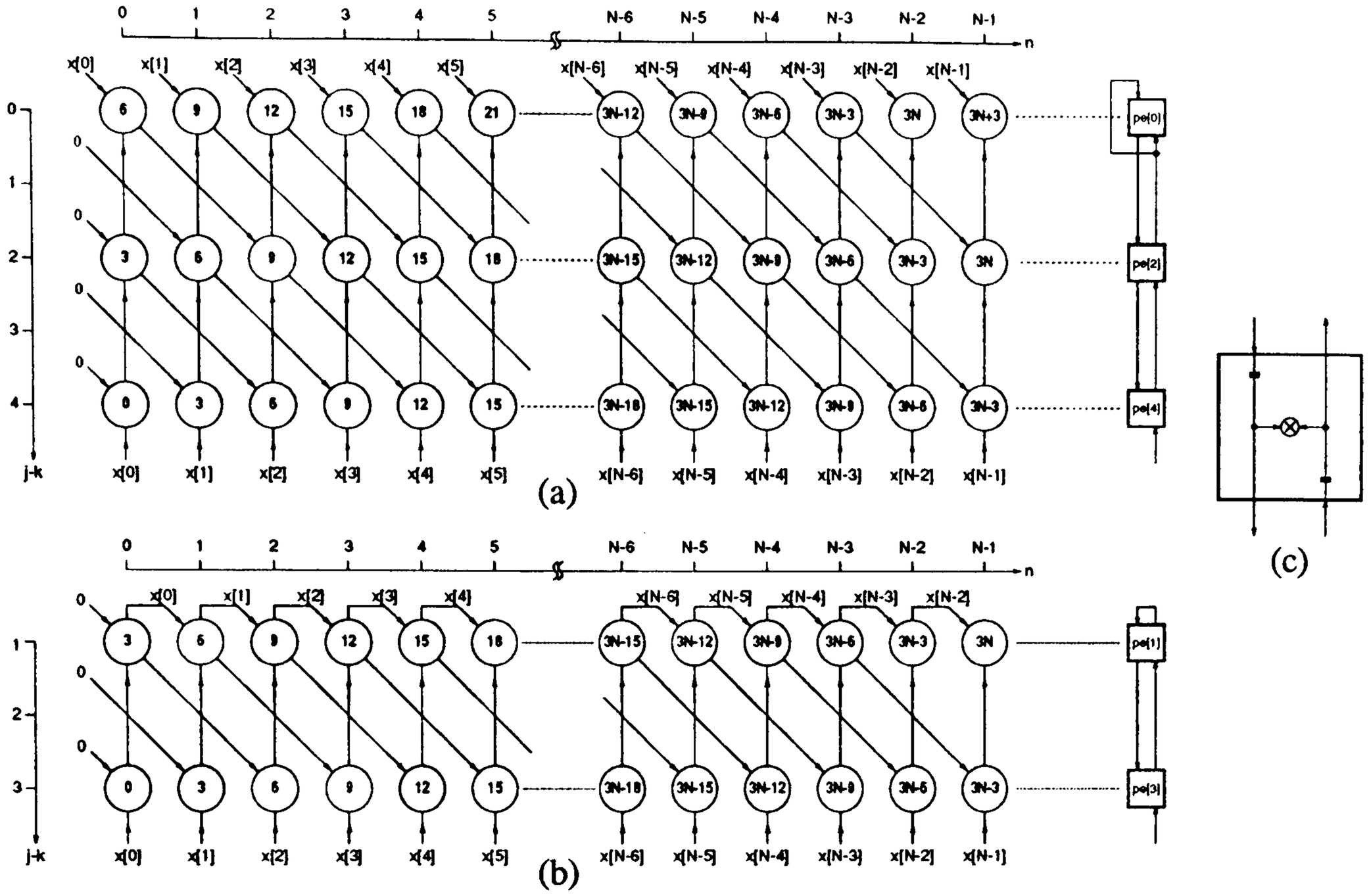


Figure 3.17: Split DDG and systolic mapping for calculation of products $x[n].x[n-(j-k)]$ where $(j-k)$ is (a) even, (b) odd. (c) systolic PE.

(D) Merging the Multiplication and Accumulation Systolic Arrays

The PMA systolic array is shown in figure 3.18(a) with the configuration of the merged multiply and accumulate PE shown in part (b) of the figure. Systolic delay registers within the PE are clocked at input data rate f_s and those in the feedback loop are clocked at three times this frequency.

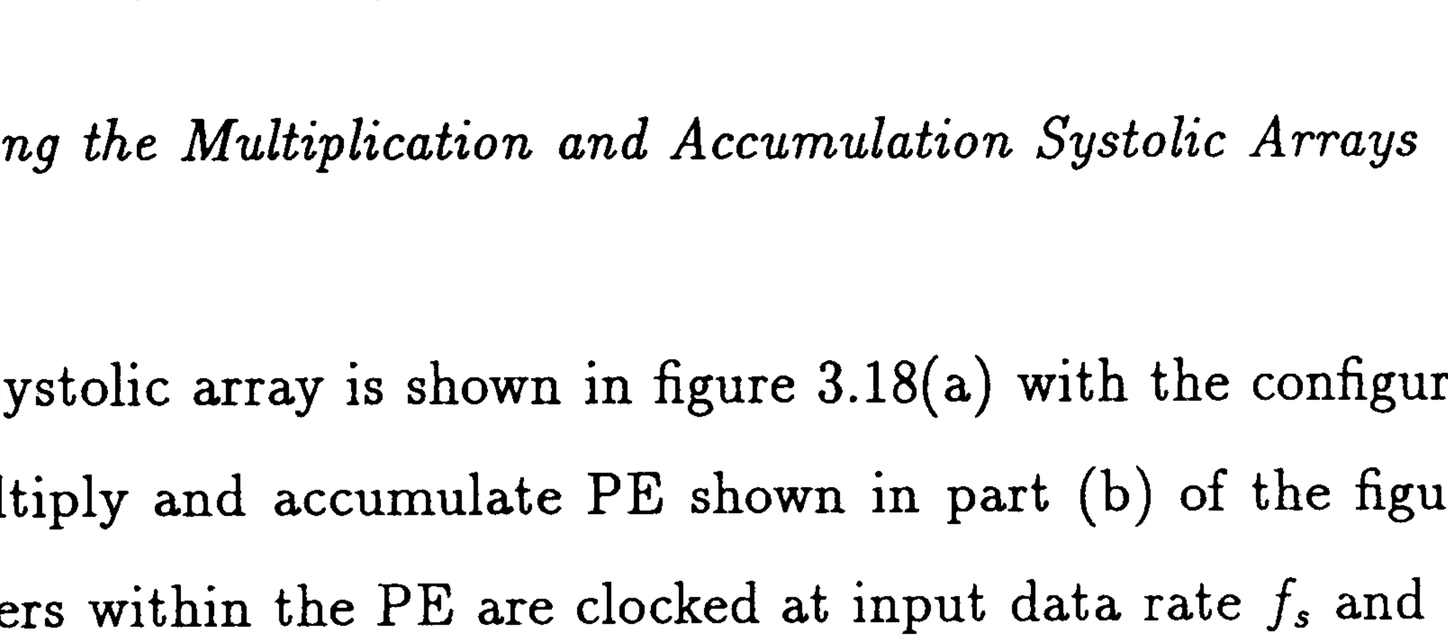


Figure 3.18: (a) PMA systolic array and (b) merged multiply accumulation PE.

3.4 Comparison of Systolic Arrays

In the design of systolic arrays it is important to always bear in mind their feasibility to hardware implementation. This is often overlooked by designers who propose arrays without any thoughts to the constraints imposed by the available VLSI technology. A comparison of the four systolic arrays for the calculation of the matrix elements is made here bearing in mind their suitability to hardware implementation. The main scope for comparison is cost. Hardware cost may be divided between that for arithmetic, register and control modules. The cost is also affected by data communication and control signal requirements between the processors of the array. The efficiency of each of the systolic arrays in terms of the processor utilisation and block pipelining period, needs to be considered also. There is no need to compare the accuracy of the different arrays since the results produced by each should be closely matched if the same rounding methods and precision are used.

3.4.1 Cost Estimate

Ultimately the cost for implementation of a systolic array is decided by the route which is used to fabricate the VLSI device be it full custom, semi custom or programmable logic. The relative cost, not the final cost, of the four arrays is of interest here and therefore the functions used to make the comparison of designs are viewed as being independent of the production technology. The hardware cost analysis presented is viewed in terms of the approximate number of NAND gates needed to construct the systolic arrays. This enables the arithmetic, register and control costs to be fairly summed when calculating the total cost of each design. Communication requirement can be simply viewed as the number of wires which make up the data buses. The costs are affected by several factors, namely type of approach, data format and word-length.

(A) Type of Approach

Word-parallel and bit-serial approaches are considered. The type of approach will affect the hardware cost, processor intercommunication requirements and speed of individual processor operation. In word-parallel arithmetic all bits in the data words are clocked into modules simultaneously. The communication links between PEs will therefore be in the form of data buses. Bit-serial modules tend to use less hardware and communication requirements can be substantially reduced as data buses are replaced with single communication lines making this an economical approach to systolic array implementation. However bit-serial modules need to be clocked within the systolic cycle. Although throughput can be improved by using high levels of pipelining the bit-serial modules tend to be slower and control circuitry may also be more complex.

(B) Data Format

The leading choices for type of arithmetic are fixed point and floating point [94]. In floating point the data is expressed in terms of a mantissa and exponent. The scheme offers an increased dynamical range over fixed point [88] but at the cost of complex extra circuitry for normalisation and exponent calculation. The fixed point approach is pursued in the matrix element calculation due to the hardware cost saving when a number of PEs are utilised.

(C) Word-Length

Fixed point data can be represented by w bit binary numbers. Bus width in a word-parallel approach, arithmetical precision and logic gate usage are increased with greater word-length. A range of word-lengths from 8 to 20 bits is considered in the cost estimation.

(D) Hardware Cost Estimation

The hardware cost estimates detailed here are based on two's complement fixed point word-parallel and bit-serial implementation of the four systolic arrays. In order to compare the hardware requirement of each of the designs in terms of NAND gates it is necessary to consider the gate count of the individual arithmetic, memory and control modules. The modules used in the matrix element calculation systolic arrays may be constructed from lower level cells such as full adders, and d-type flip flops. The relative sizes for NAND gate implementations of the cells that are used are shown in table 3.1.

type of cell		NAND gate usage
full adder	(FA)	12
gated full adder	(GFA)	13
d-type flip flop	(DFF)	6
2 to 1 line multiplexor	(MUX)	4
exclusive OR gate	(XOR)	4

Table 3.1: NAND gate usage of low level cells.

w bit module	word-parallel	bit-serial
addition	FA.w	FA+DFF
subtraction	GFA.w	GFA+DFF
multiplication	GFA.$(w^2 + w)$	(GFA+3.DFF+XOR).$(w + 1)$
register	DFF.w	DFF.w
multiplex	MUX.w	MUX

Table 3.2: Cellular construction cost of arithmetic, memory and control modules for word-parallel and bit-serial approaches.

Using the information in table 3.2 hardware cost for bit-serial and word-parallel versions of the various arithmetic, memory and control modules can be deduced in terms of a word-length w. The word-parallel addition and subtraction unit cost[1] is based on ripple carry linear arrays whilst their bit-serial versions are reduced to single cell units with carry feedback via d-flip-flops.

[1]Note that word-parallel costs for addition, subtraction, multiplex and register units, double when manipulating double precision products.

The DDG concept can be extended to the bit-level in order to design word-parallel and bit-serial multiplication devices as shown by McCanny et al. [118], where the multiplication of positive binary numbers is explored. Moving to two's complement representation, the $w = 3$ bit DDG in figure 3.19(a) is based on the use of an adjustment constant which is input to the right hand nodes for addition with partial products [119]. The adjustment is needed to account for negatively weighted bit products (i.e. those formed in the shaded nodes) which are logically inverted then treated as positive bit products in the partial product summations [111]. This DDG is highly regular and there are only two different types of node whose operation is shown in figure 3.19(b). The regularity of the DDG is reflected in the multiplier arrays resulting from various projections and this property together with the low number of PE operation modes eases VLSI implementation and results in simple control. Direct mapping such that each node is mapped into a single PE ($node[j,k] \rightarrow pe[j,k]$) results in a ripple through word-parallel array multiplier of $w^2 + w$ GFAs. This design is preferred over, a design

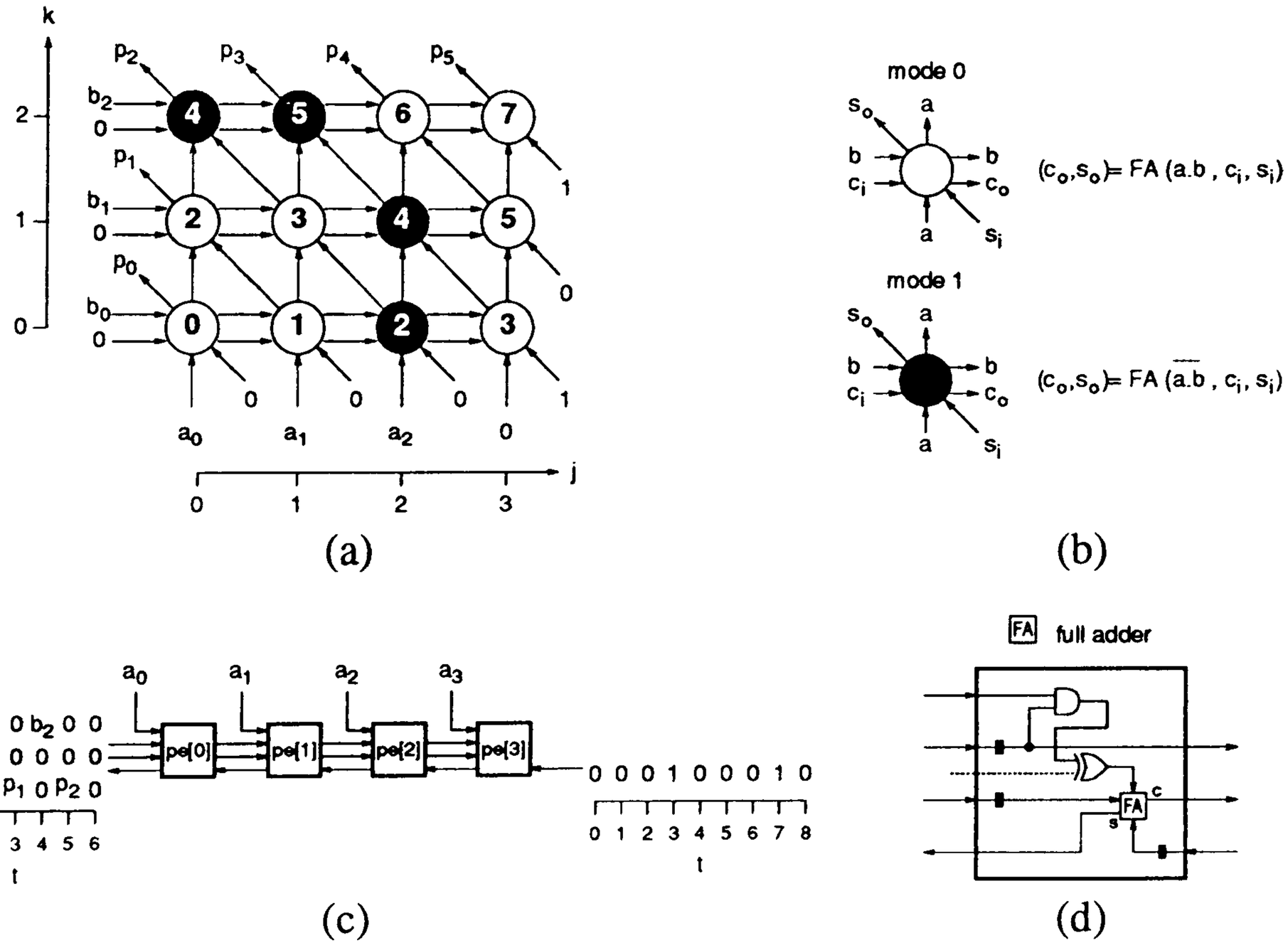


Figure 3.19: (a) Bit level DDG for two's complement multiplication using an adjustment scheme (b) key to node operations (c) systolic bit-serial/word-parallel multiplier produced by projecting the DDG in the k direction and (d) PE layout.

such as the Pezaris multiplier array [88] for example which, despite containing $2w$ fewer PEs, is difficult to implement in VLSI due to a less regular architecture and double the number of PE types. Lower cost, but slower, pipelined bit-serial multipliers can be designed by applying timing and allocation functions to the DDG. At first glance j projection into a systolic array of w PEs seems to produce the most cost efficient bit-serial multiplier. However, a more detailed inspection reveals that projection in the j direction, although resulting in a $w+1$ PE systolic array (figure 3.19(c)), has a lower cost for $w \geq 8$ since the each of its PEs (figure 3.19(d)) require one less register.

The systolic delay registers contain the same number of d-type flip flops in the word-parallel and bit serial approaches since the number of bits that must be stored are the same in both cases. The difference occurs in the way in which the registers are configured. The word-parallel type are PIPO and the bit-serial registers are SISO. Bit-serial multiplexing is carried out one bit at a time and so requires just one MUX cell but in the word-parallel equivalent this hardware is increased by a factor of w.

Finally the hardware usage of the systolic arrays can be described by the number of arithmetic, memory and control modules incorporated each design. This information is detailed in table 3.3 enabling the NAND gate usage to be compared by using the information presented in this and the previous two tables.

w bit module	number of modules used in the systolic arrays			
	2-dimensional	tri-linear	bi-linear	PMA
addition (d)	27	9	10	5
subtraction (d)	0	0	5	0
multiplication	18	9	5	5
register	58	18	10	10
register (d)	18	9	60	20
multiplex	5	2	0	0
multiplex(d)	0	9	20	10

Table 3.3: Hardware usage for systolic arrays in terms of module usage where (d) indicates a double precision module.

(E) Hardware Cost Estimation Results

Using the information from tables 3.1 to 3.3 the total number of NAND gates in each of the four systolic arrays may be approximated. The results of cost in terms of the total number of NAND gates in each design are plotted against the word-length w in the graphs shown in figure 3.20 for word-parallel and bit-serial approaches. As expected the word-parallel approach exhibits the greater hardware costs. The graphs show that the 2-dimensional array uses significantly more hardware than the other arrays in both approaches for all word-lengths which is mainly due to the large number of multipliers used. The bit-serial multipliers are smaller in size compared to those of the word-parallel scheme. Hence, between approaches the relative costs for all multipliers in an array against the systolic registers changes because storage requirements remain constant whatever the approach. This effect is reflected in the results from the tri-linear and bi-linear arrays. Both of these arrays show similar costs in the word-parallel approach but the cost of the tri-linear array is significantly lower in the bit-serial approach due to the cost saving on a larger number of multipliers relative to the bi-linear arrays large memory requirement in both approaches. In both approaches the PMA array is the least expensive to implement due to its low multiplier and register combined cost.

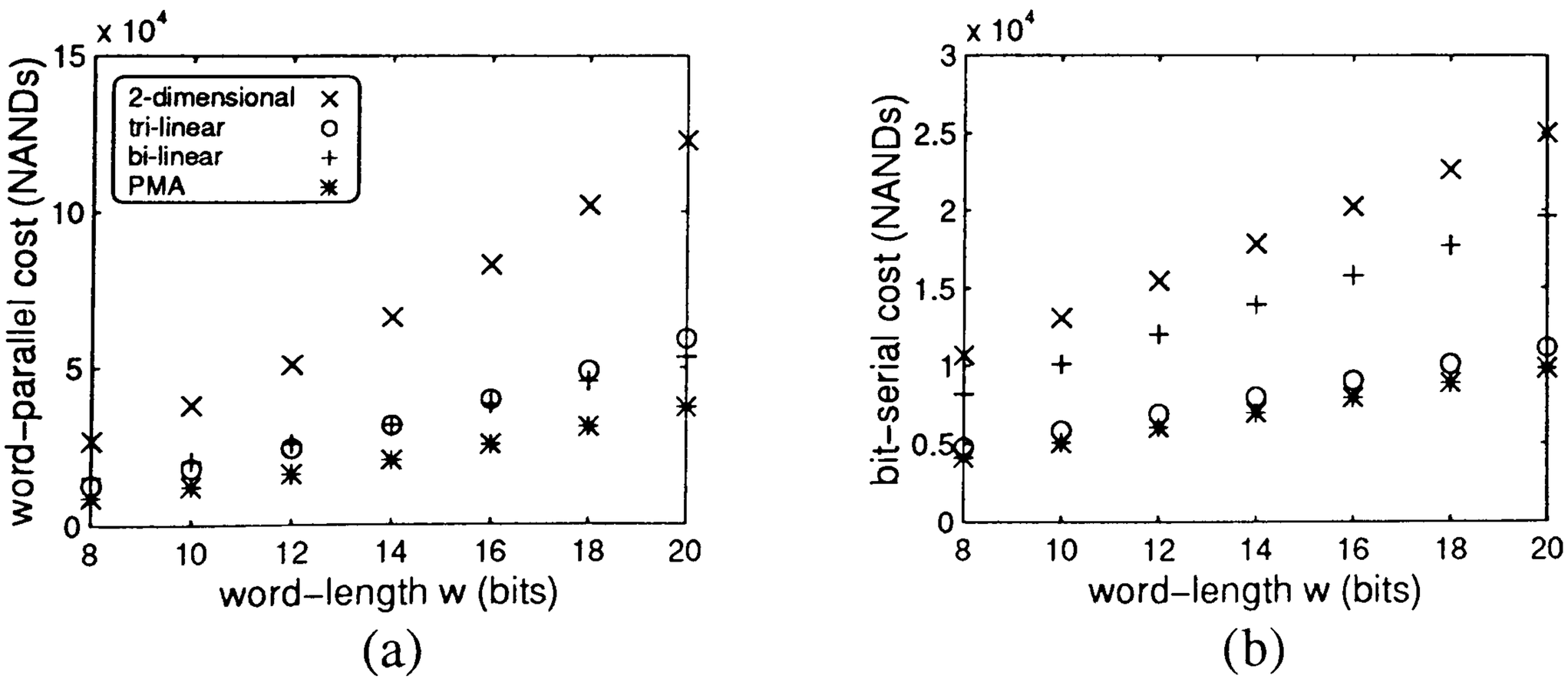


Figure 3.20: Cost for (a) word-parallel and (b) bit-serial approaches.

(F) Data Communication Cost

The data communication cost covers the number of data communication lines between the processing elements and the number of feedback links around the PEs. The communication cost relates to printed circuit board track area, routeing resources on integrated circuits and the number of input/output pins around a device. Communication cost should be as low as possible to keep within physical limitations imposed by the hardware technology and to facilitate the ease of implementation. The communication cost analysis is expressed in terms of the number of data buses. For the word-parallel approach bus-width bw is equal to the word-length w. In the bit-serial approach $bw = 1$ and is therefore independent of word-length, demonstrating the communication cost saving advantage of this approach when dealing with large systolic arrays and high precision data.

The data communication costs in the four systolic arrays are detailed in table 3.4. The results of the cost analysis show that the 2-dimensional array uses by far the most resources. Its problems are further exasperated by the global communications around the array, which provide the input routeing, and the difficulty in data retrieval. The bi-linear and PMA arrays display lowest costs due to their smaller PE count. However there is greater communication burden within each PE of the bi-linear array since each consists of a multiplier accumulator and a sorting network. Internal PE communication is not reflected in the results table and so the advantage of the bi-linear array over the tri-linear design is not so great as the figures suggest.

communication type	number data buses			
	2-dimensional	tri-linear	bi-linear	PMA
PE interconnection	40	15	8	8
localised feedback	18	9	5	5
array input	16	3	2	2
array output	9	9	5	5

Table 3.4: Data communication cost.

(G) Control Signal Cost

All the arrays have clock and reset signals which control activation and initialisation. The bi-linear array requires nine extra control lines for multiply-by-two selection at appropriate times plus two more controls on the multiplexor input selection lines. The sorting networks of the bi-linear design prove to have the biggest control burden. Each of the three registers contained within its sorting processors require clock enable generation and two more multiplexor select control lines are needed. Of the four arrays considered the 2-dimensional design is the only one that requires no extra internal PE control during the calculation of the matrix elements but it does have five multiplexors which require input selection signals. The tri-linear array needs 9 control lines for double product selection while 5 control lines are required for this function in the PMA array. Additionally there are five multiplexor select signals, for switching to accumulation *mode* 1, and an extra clock running at $3f_s$ in the PMA array.

3.4.2 Efficiency

The allowable operation time of the array PEs T_{PE}, and consequently, of the modules contained within the PEs, can be derived from the clock rate f_{clk} which is necessary for the real-time systolic array processing of data sampled at a rate of f_s.

$$T_{PE} < \frac{1}{f_{clk}} \tag{3.25}$$

The allowable processing time will influence the choice of whether a word-parallel or bit-serial approach is to be followed. Once the approach is decided upon arithmetic modules can be designed to meet the specification imposed by the input data sampling rate. The most efficient systolic arrays have the longest processing time and allow arithmetic modules to be designed to less stringent specification than would be the case for an inefficient array. The efficiency of the designs may be looked at in terms of processor and block pipelining periods.

(A) Processor Pipelining Period

The processor pipelining period α gives an indication of how busy each of the processors are and is thus related to their efficiency [23]. It is formally given by the time interval (in clock periods) between two successive computations and in general is the same for all the processors in the array. Ideally $\alpha = 1$. The processor pipeline period can be calculated from the product of the row schedule vector $\bar{S}$ and the transpose of the row projection vector $\bar{P}^T$:

$$\alpha = \bar{S}.\bar{P}^T \tag{3.26}$$

For example in the bi-linear systolic array where $\bar{S} = [-1, -1, 2]$ (3.6) and $\bar{P} = [0, 0, 1]$ represents projection in the n direction then:

$$\alpha = [-1, -1, 2]. \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = 2 \tag{3.27}$$

so that each processor is active in one in every two clock cycles.

(B) Block Pipelining Period

The block pipelining β period refers to how soon a new set of data can be processed in the systolic array after the previous set of data has been input. In terms of the matrix element calculation array, this is the shortest allowable time from array input of the $x[0]$ data samples in two consecutive windows. The block pipeline period can be calculated from the product of the processor pipeline period and the maximum number of node operations performed by any processor in the array. So again for example considering the bi-linear array, then from the DDG in figure 3.10 each PE is required to perform N operations and therefore:

$$\beta = \alpha.N = 2.N \ . \tag{3.28}$$

(C) Results of Efficiency Analysis

The pipelining periods noted in table 3.5 are derived by consideration of the timing functions and DDGs associated with each design. The 2-dimensional, bi-linear and PMA arrays show maximum processor utilisation with $\alpha = 1$ and therefore their processors are active on consecutive clock cycles. In the tri-linear array $\alpha = 2$ so its processors are only 50% efficient. The clock rate of this array therefore needs to be twice that of the others and its arithmetic units need to be designed to a higher specification.

type of pipelining period		pipelining periods for systolic arrays (clock pulses)			
		2-dimensional	tri-linear	bi-linear	PMA
processor	α	1	2	1	1
block	β	$N+4$	$2N$	N	N

Table 3.5: Processor and block pipelining periods.

For the real-time processing of input data sampled at f_s then:

$$f_{clk} = \alpha . f_s \tag{3.29}$$

The highest sampling frequency needed for the incoming data is 51.2kHz in the Modified Covariance application. This gives approximate maximum allowable PE latency T_{clk} of $\approx 20\mu$s for the 2-dimensional, bi-linear and PMA arrays compared to the more stringent $\approx 10\mu$s for the tri-linear array. The PE latency is determined by the design of the internal arithmetic units, of which the multipliers display the greatest lag, and the VLSI implementation technology used (e.g. full custom or FPGAs) which affects communication delays and the propagation delays of logic gates. In the bi-linear and PMA arrays β is the same length as the data window and so these arrays have the ability to continuously process consecutive sets of data. For the tri-linear array however β is double this but in terms of real time it is the same since the tri-linear array is being clocked at twice the rate.

3.5 Concluding Remarks

This chapter has presented four different designs for the real-time calculation of the covariance matrix elements in a fixed model order Modified Covariance spectral estimator. The designs were based on implementation for model order $p = 4$ as this has been previously determined as optimal for when both mean frequency and half bandwidth are to be calculated.

The real-time calculation was shown to be computationally intensive and for this reason the algorithms involved were partitioned onto systolic array processors using data dependence graph methods. This chapter has shown different ways of capturing the matrix element calculation on DDGs, demonstrating the importance of the DDG representation on the design of the systolic array which results from space-time mapping. Initially the formal DDG design method was strictly adhered to, to produce a two dimensional systolic array containing 9 processors, with each processor containing two multiply accumulate units and an adder. To reduce cost, a second array was designed from a number of independent DDGs resulting in the matrix element calculation being partitioned into three separate linear systolic arrays of length 5, 3 and 1 processors. Each PE contained only a controlled multiply accumulate unit but it was observed that processor utilisation was only 50% and that certain products were formed more than once in different PEs. In an attempt to optimise the tri-linear design the calculation was partitioned into two pipelined sections resulting in a less regular design. The first section was a bi-linear array which formed and accumulated all the required products just once with maximum efficiency. The other section contained sorting networks to form the matrix elements but the greater storage requirement of this module led to high bit-serial bi-linear array costs. A fourth systolic array was designed by partitioning the multiplications from the accumulations. This PMA array matched the efficiency, while eliminating the large memory and control burdens imposed by the bi-linear design.

The cost in terms of hardware usage of the arrays was estimated in order to compare between the three arrays for word parallel and bit-serial implementation approaches at different word-lengths. The PMA array was chosen as the optimal solution since it displayed the lowest cost for all the word-lengths in both approaches, it also showed low communication burden and high efficiency with the only drawback for the requirement of two clocks running at f_s and $3f_s$.

The choice on whether bit-serial or word-parallel arithmetic units are to be used can be based upon the allowable lag of a PE operation given the sampling frequency of the input data. Multiplication of two w bit numbers is in the order of w times more computationally complex than their addition and so multiplication execution time tends to be longer. In the PMA array the maximum allowable PE lag is $\approx 20\mu$s (see section 3.4.2(C)) and thus multiplication execution time must be less than 20μs assuming the systolic register transfer delays are relatively small. Typically, a two's complement, bit-serial multiplication module implemented in CMOS technology can perform a complete multiplication in ≈ 1 to 2μs for word-length ranging from 8 to 20 bits [111][112]. Hence, the bit-serial approach is deemed to meet the specification imposed by the real-time systolic processing restrictions and is capable of achieving a factor of 10 times the necessary throughput due to the high computational efficiency of the systolic array. This justifies the use of a bit-serial scheme over the more highly specified but expensive to implement word-parallel approach.

The computationally efficient bit-serial PMA systolic array design requires ≈ 4100 to 9800 NAND gates for word-lengths from 8 to 20 bits respectively. The next stage in the design of the PMA array is to select the word-length which is required to obtain sufficient precision in the covariance matrix input data to the next stage of the estimator in which the filter parameters are computed. Following a review in chapter 4 of decomposition systolic arrays for use in the filter parameter computation chapter 5 goes on to consider the word-length requirements.

Chapter 4

Fixed Model Order - Calculation of Filter Parameters

4.1 Introduction

Chapter 3 considers the calculation of the covariance matrix elements in the Modified Covariance spectral estimator of fixed model order. This chapter considers systolic array designs that could be used for the next main section of the fixed model order estimator in which the autoregressive filter parameters are calculated by solving the modified covariance system of equations (2.21).

The main part of the chapter is devoted to consideration of systolic array implementations of decomposition techniques which may be used to efficiently solve the system of equations. Using these techniques the set of covariance equations can be rewritten in the form of lower and upper triangular systems of equations which may be respectively solved using the forward elimination and back substitution algorithms. Systolic array realizations of the three most applicable decomposition algorithms, Cholesky, LU and LDL^T, are reviewed. Data dependence graph representations for each of these methods are presented. From these graphs a wide variety of systolic arrays are derived by choosing different projection directions.

4.1.1 Overview

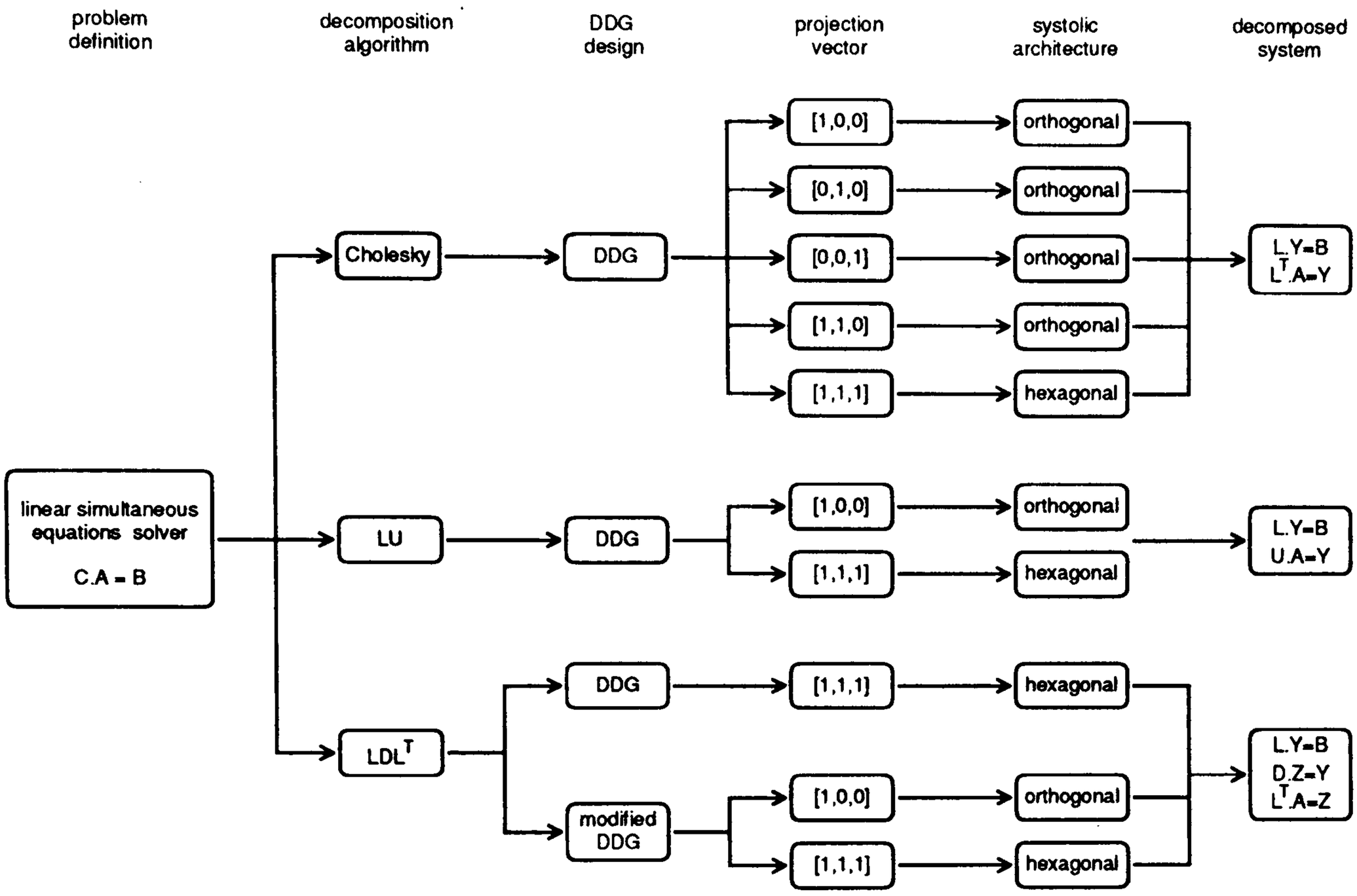


Figure 4.1: Overview of the decomposition systolic arrays described.

Figure 4.1 overviews the decomposition systolic arrays which are discussed in this chapter with regard the calculation of the filter parameters[1]. The Cholesky algorithm is considered as it displays good numerical stability [27] and is suited to solution of a symmetrical positive definite system of equations such as that in the Modified Covariance method. The dimension of the Cholesky decomposition DDG for the model order $p = 4$ spectral estimator is no greater than 4 nodes in either the j, k or n directions, thus a wide choice of projection directions are feasible. The selection of the projection vector leading to the optimal decomposition systolic array design is therefore not as clear cut as in the case of the matrix element calculation DDGs discussed in the previous chapter, whose length restricted the projection to along the $n-$axis in order to avoid excessively long systolic arrays. The design of the various systolic arrays in chapter 3 is based

[1]It should be noted that, despite the symmetrical properties of the Modified Covariance matrix, it is not Toeplitz. The Modified Covariance equations cannot therefore be solved by the computationally superior Levinson algorithm.

around projection of different DDG representations of the matrix element algorithm. Using the same DDG design methodology, this chapter provides a tutorial overview of the design of five Cholesky decomposition systolic arrays resulting from the different projections of a single DDG (figure 4.1). Brent & Luk's array, topologically equivalent to a $[1, 1, 1]$ mapping is compared with those formed by the alternative mappings to prove that it is in fact the optimal design in order to justify its selection.

On the comparison of the systolic arrays it is difficult to make a decision as to what makes a design optimal. Many designers would just make the choice by looking for the design which uses the least amount of PEs, but which array should be chosen if they all contain the same number of PEs? The in depth comparison of the arrays used here is concerned with the relative costs involved when actually implementing the designs on a VLSI device. Such factors as hardware cost in terms of NAND gates, communication burden and control complexity are considered providing detailed comparison.

The square-root algorithm, which is required for the Cholesky decomposition, is not well suited to VLSI implementation [120]. The DDG representation of the square-root algorithm is irregular and its nodes are required to work in 4 different modes [88]. Therefore pipelined square-root arrays have complex control requirements. An intrinsic dependence among the iteration steps also makes square-root devices considerably slower than multipliers [121] and this results in a bottleneck which limits the maximum clock frequency of the systolic array [80]. The problems associated with the square-root VLSI implementation lead to consideration of non-square-root decomposition algorithms, such as the LU and LDL^T, respectively used to decompose non-symmetrical and symmetrical matrices.

Once again using the DDG design method, the $[1,0,0]$ and $[1,1,1]$ projection vectors, which are found to produce the two optimal systolic arrays for square-root Cholesky decomposition, are applied to DDGs for the non-square-root decomposition methods. The $[1,0,0]$ projection of the LU DDG produces the array proposed by Kung [23] and analysis of Kung's design shows that no provision is made for retrieval of the upper triangular matrix from the array PEs. A new DDG is presented which when projected by the $[1,0,0]$ vector produces a systolic array with a novel on-the-fly data retrieval network at no extra cost. This array is compared with $[1,1,1]$ projections of LU and LDL^T decomposition which respectively produce systolic arrays topologically equivalent to those proposed by Kung & Leiserson [17] and Brent & Luk [80]. Inefficiencies with Brent and Luk's $[1,1,1]$ LDL^T design [80] are recognised and after re-examination of the recurrence equations a modified DDG which re-represents the data flow is presented. A new LDL^T systolic array, produced by the $[1,1,1]$ projections, is proposed and is found to show significant cost reduction over Brent & Luk's. A new array produced by $[1,0,0]$ projection of the modified LDL^T DDG is also presented and compared in detail in terms of VLSI implementation cost with rest of non-square-root decomposition systolic arrays.

4.2 Computational Burden

When considering model order $p = 4$ this part of the problem is less computationally intensive than the matrix element calculation. The Cholesky decomposition algorithm for example requires $\approx \frac{p^3}{6}$ operations. Assuming that one decomposition needs to be performed within one data window of 5ms length then this leads to a computational rate of approximately two thousand operations per second which is much less demanding than the burden imposed by the matrix element calculation as described in section 3.2. Slower clock rates may be used for this section of the estimator and it seems sensible to follow a bit-serial approach.

4.3 Triangular Systems

A common feature of the decomposition methods which are described is that they all produce triangular systems of simultaneous equations. Cholesky, LU and LDL^T decompositions all produce lower and upper triangular systems which may be solved by the forward elimination and back substitution processes respectively.

4.3.1 Forward Elimination and Back Substitution Algorithms

Forward elimination is used to solve the lower triangular system of equations:

$$L.Y = B \tag{4.1}$$

which, for model order $p = 4$ is written as:

$$\begin{bmatrix} l[1,1] & 0 & 0 & 0 \\ l[2,1] & l[2,2] & 0 & 0 \\ l[3,1] & l[3,2] & l[3,3] & 0 \\ l[4,1] & l[4,2] & l[4,3] & l[4,4] \end{bmatrix} \begin{bmatrix} y[1] \\ y[2] \\ y[3] \\ y[4] \end{bmatrix} = \begin{bmatrix} b[1] \\ b[2] \\ b[3] \\ b[4] \end{bmatrix} \tag{4.2}$$

The forward elimination recurrences to calculate Y, described in [27], show that the algorithm requires $\frac{p^2}{2}$ operations. The single assignment format of the forward elimination algorithm:

$$r^{(k+1)}[j] = \begin{cases} 0 & k = 0 \\ r^{(k)}[j] + l[j,k].y[k] & 1 \leq k < j \end{cases} \tag{4.3}$$

$$y[j] = (b[j] - r^{(j)}[j])/l[j,j] \qquad 1 \leq j \leq p \tag{4.4}$$

defines the computation, firstly of element $y[1]$ through to element $y[4]$. Back substitution is used for the solution of an upper triangular system of equations:

$$U.A = Y \tag{4.5}$$

and for model order 4:

$$\begin{bmatrix} u[1,1] & u[1,2] & u[1,3] & u[1,4] \\ 0 & u[2,2] & u[2,3] & u[2,4] \\ 0 & 0 & u[3,3] & u[3,4] \\ 0 & 0 & 0 & u[4,4] \end{bmatrix} \begin{bmatrix} a[1] \\ a[2] \\ a[3] \\ a[4] \end{bmatrix} = \begin{bmatrix} y[1] \\ y[2] \\ y[3] \\ y[4] \end{bmatrix} \tag{4.6}$$

The back substitution recurrences:

$$r^{(p+2-k)}[j] = \begin{cases} 0 & k = p+1 \\ r^{(p+1-k)}[j] + u[j,k].a[k] & j < k \leq p \end{cases} \quad (4.7)$$

$$a[j] = (y[j] - r^{(p+1-j)}[j])/u[j,j] \qquad 1 \leq j \leq p \quad (4.8)$$

are similar to those for forward elimination except that the vector A is calculated $a[4]$ first then $a[3]$ etc.

4.3.2 Forward Elimination and Back Substitution Systolic Arrays

The forward elimination and back substitution recurrences can be mapped into the systolic arrays shown in figures 4.2(a) and (b) respectively which were first presented by Kung & Leiserson [17][122]. The systolic array contains two types of processor $PE0$ which calculates (4.4) and (4.8) and $PE1$ which handles (4.3) and (4.7). Of the four processors in each of the arrays $pe[0]$ is an instance of type $PE0$ and $pe[1]$ to $pe[3]$ are of type $PE1$.

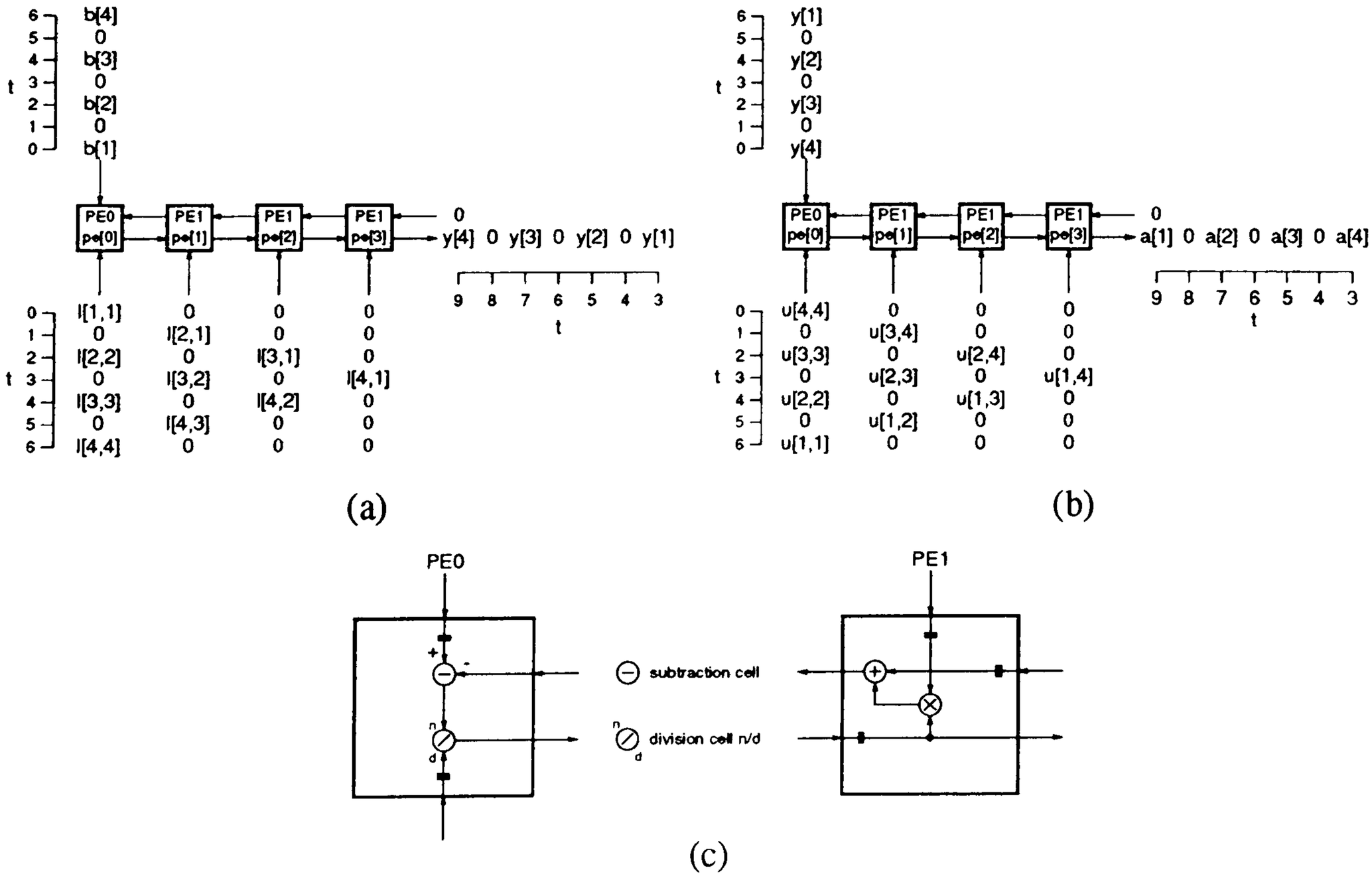


Figure 4.2: Systolic arrays for (a) forward elimination, (b) back substitution and (c) PE function.

4.4 Square-Root Cholesky Decomposition

4.4.1 Cholesky Decomposition Algorithm

In the Cholesky decomposition method the covariance matrix is decomposed into the product of a lower triangular matrix and its transpose:

$$C = L.L^T \tag{4.9}$$

where for model order 4:

$$L = \begin{bmatrix} l[1,1] & 0 & 0 & 0 \\ l[2,1] & l[2,2] & 0 & 0 \\ l[3,1] & l[3,2] & l[3,3] & 0 \\ l[4,1] & l[4,2] & l[4,3] & l[4,4] \end{bmatrix} \tag{4.10}$$

This method takes the symmetry about the northwest to southeast diagonal of the positive definite covariance matrix into account and therefore only the lower triangular part of the covariance matrix is required in the evaluation of L. The purpose of forming the decomposed system is to produce two triangular sets of simultaneous equations which can be easily solved. Substitution of (4.9) into (2.21) leads to the following equation:

$$L.L^T.\hat{A} = B \tag{4.11}$$

The product of the p by p matrix L with the length p vector $\hat{A}$ can be said to equal another length p vector Y:

$$L^T.\hat{A} = Y \tag{4.12}$$

Equation (4.12) has the same format as the upper triangular system given in (4.5). $\hat{A}$ may therefore be calculated using back substitution once Y has been determined. Substitution of (4.12) into (4.11) yields the lower triangular system:

$$L.Y = B \tag{4.13}$$

which has the same style as (4.1) for solution by forward elimination.

The algorithm for the calculation of the Cholesky decomposition is described in terms of a *for* loop algorithm in Golub and Van Loan [27] who state the computational complexity as $\frac{p^3}{6}$ flops. It can be viewed in terms of a series of iterative updates on the matrix C which is being decomposed. The following recurrence equations show the Cholesky Decomposition algorithm written in single assignment format from which the DDG can be derived:

$$c^{(1)}[j,k] = \begin{cases} 0 & j < k \\ c[j,k] & k \leq j \end{cases} \tag{4.14}$$

$$c^{(n+1)}[j,k] = \begin{cases} 0 & j < k \\ 0 & j \leq n \\ c^{(n)}[j,k] - l[j,n].l[k,n] & n < k \leq j \end{cases} \tag{4.15}$$

$$l[j,k] = \begin{cases} 0 & j < k \\ \sqrt{c^{(k)}[j,k]} & j = k \\ c^{(k)}[j,k]/l[k,k] & k < j \end{cases} \tag{4.16}$$

4.4.2 Cholesky Decomposition Data Dependence Graph

To aid in the derivation of the DDG for Cholesky decomposition a series of matrices $C^{(n+1)}$ with elements $c^{(n+1)}[j,k]$ (4.14) (4.15) can be formed. The initialisation of matrix of $C^{(n+1)}$ for $n = 0$ (4.14) to the lower triangular part of the original covariance matrix C is shown below:

$$C^{(1)} = \begin{bmatrix} c[1,1] & 0 & 0 & 0 \\ c[2,1] & c[2,2] & 0 & 0 \\ c[3,1] & c[3,2] & c[3,3] & 0 \\ c[4,1] & c[4,2] & c[4,3] & c[4,4] \end{bmatrix} \tag{4.17}$$

Assignment (4.15) describes how the matrix $C^{(1)}$ (4.17) is iteratively updated as the recursion index n is incremented. The sequence of iterations for calculation of the Cholesky decomposition, matrices (4.18) to (4.20), show that as $C^{(n+1)}$ is updated the dimensions of the lower triangular matrix are reduced because the elements in column n are replaced with zeros as defined by (4.15) for $j \leq n$:

$$C^{(2)} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & c^{(1)}[2,2] - (l[2,1])^2 & 0 & 0 \\ 0 & c^{(1)}[3,2] - l[2,1].l[3,1] & c^{(1)}[3,3] - (l[3,1])^2 & 0 \\ 0 & c^{(1)}[4,2] - l[2,1].l[4,1] & c^{(1)}[4,3] - l[3,1].l[4,1] & c^{(1)}[4,4] - (l[4,1])^2 \end{bmatrix} \quad (4.18)$$

$$C^{(3)} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & c^{(2)}[3,3] - (l[3,2])^2 & 0 \\ 0 & 0 & c^{(2)}[4,3] - l[3,2].l[4,2] & c^{(2)}[4,4] - (l[4,2])^2 \end{bmatrix} \quad (4.19)$$

$$C^{(4)} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & c^{(3)}[4,4] - (l[4,3])^2 \end{bmatrix} \quad (4.20)$$

The elements $l[j,k]$ in column $k = n+1$ of the lower triangular matrix L can be calculated from the elements of the same column in matrix $C^{(n+1)}$ by the assignments detailed in (4.16). For example when $n = 1$, $l[2,2]$ is calculated from the square root of element $c^{(2)}[2,2]$ in matrix $C^{(2)}$ (4.18). Element $l[3,2]$ can then be calculated from the quotient of $c^{(2)}[3,2]$ with $l[2,2]$ and similarly for $l[4,2]$.

Figure 4.3(a) shows the Cholesky decomposition DDG. The three dimensional graph consists of four planes of nodes, each lower triangular shaped plane corresponding to a particular value of n. The nodes of the graph operate in either *mode* 0, *mode* 1, *mode* 2 or *mode* 3 (figure 4.3(b)), each mode corresponding to a specific task respectively defined by the recurrences (4.16) $j = k$, (4.16) $j > k$, (4.15) $j = k$ and (4.15) $j > k$. The orthogonal input to layer n of the graph is matrix $C^{(n)}$, each element $c^{(n)}[j,k]$ is input to $node[j,k,n]$ and matrix $C^{(n+1)}$ is the output. Therefore in the nearest layer where $n = 1$, elements $c^{(1)}[j,k]$ of matrix $C^{(1)}$ (4.17) are input in the n direction to $node[j,k,1]$ (nodes are referenced as $node[j,k,n]$) and $C^{(2)}$ (4.18) is produced by this plane. Elements of the matrix L computed in the *mode* 0 and *mode* 1 nodes are globally transmitted in the j and k directions.

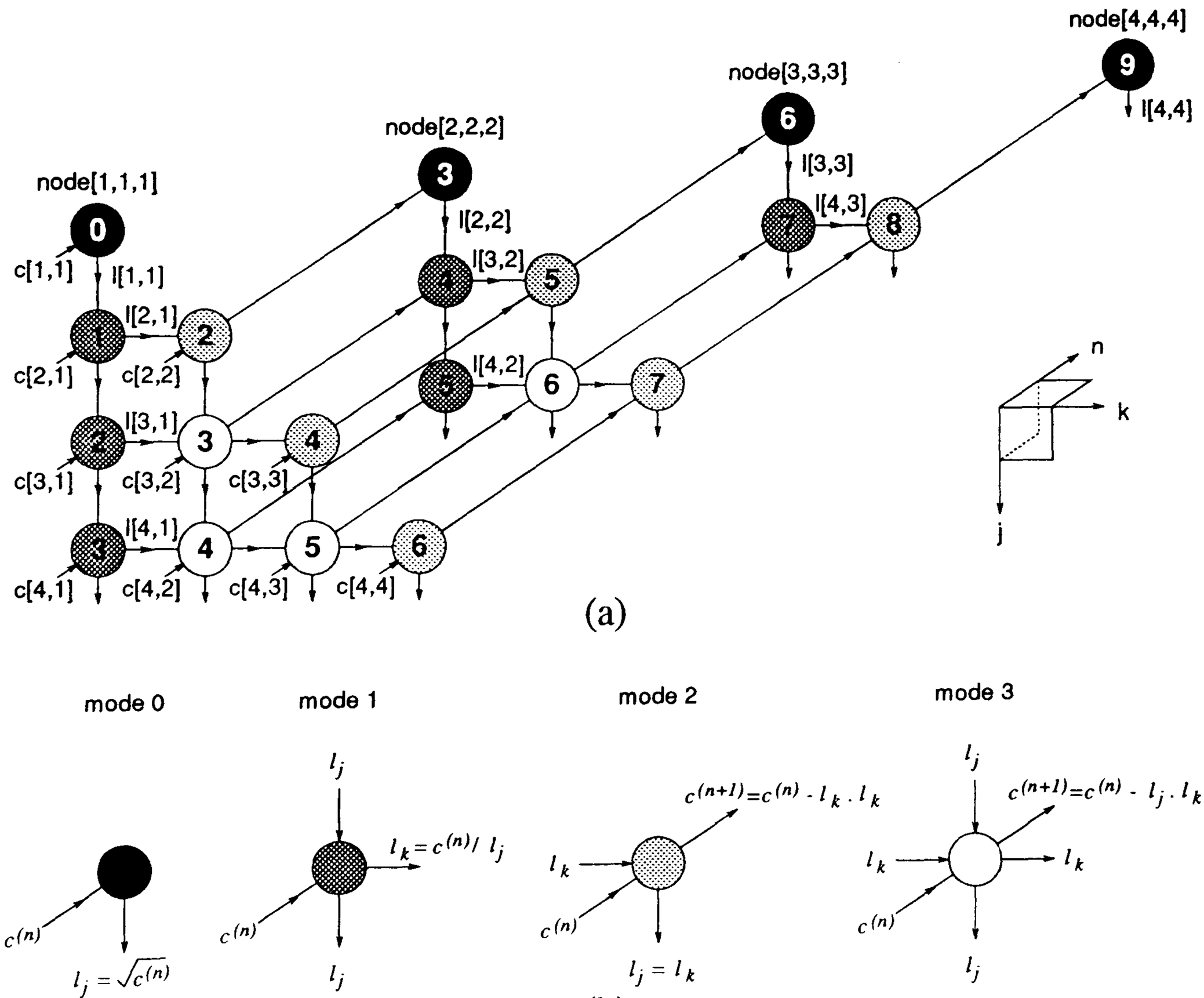


Figure 4.3: (a) DDG for Cholesky decomposition, (b) key to symbols.

4.4.3 Cholesky Decomposition DDG Localisation

The pipelining directions of the data in the Cholesky decomposition DDG are also shown in figure 4.3 by the node interconnection arcs. The pipeline of $l[1,1]$ from $node[1,1,1]$ to $node[4,1,1]$ in the j direction defines the ordering of the computations as $l[2,1]$, $l[3,1]$ then $l[4,1]$. Although this ordering is not critical it is necessary to maintain localised communication throughout the DDG. Similarly, pipelines can be set up in the second, third planes for $l[2,2]$ and $l[3,3]$ respectively. In the n layers elements $l[j,n]$ $(n < j \leq 4)$ are also localised in the k direction along rows j and in the j direction along column j. Localisation for the updating of the $C^{(n+1)}$ matrices in the n direction respects the imposed data dependencies.

4.4.4 Cholesky Decomposition DDG Timing Function

A timing function can be derived for the Cholesky decomposition DDG based on the localisation. Insertion of unit delay into each of the node interconnection arcs leads to a timing function:

$$t[j,k,n] = j + k + n + o \tag{4.21}$$

where t is given in clock cycles and o is an offset which defines the start time of the computation at $node[1,1,1]$. This start time may vary depending upon the allocation function chosen since some of the systolic arrays derived in the following section need several clock cycles for data to be loaded into the array before calculation can begin. The timing function also determines the overall operation time of the array. For example, the timing function is illustrated by the internal node numbering in figure 4.3 is for an array whose computation in $node[1,1,1]$ begins at $t = 0$. For this timing function $o = -3$ and operation is complete after 10 clock cycles when $node[4,4,4]$ executes at $t = 9$. Processor and block pipeline periods are determined partly by the timing function but are also related to the chosen allocation function (see section 3.4.2).

4.4.5 Cholesky Decomposition DDG Projection

A variety of systolic arrays can be formed by projecting the Cholesky decomposition DDG in different directions. There are many more viable projections for this DDG compared with the matrix element calculation DDGs whose options for projection are limited due to the long graph lengths of approximately N nodes in the n direction. The length of the Cholesky DDG however only extends to four nodes along the n axis and so projection in directions other than n is feasible. Projection directions are represented by vector $[j, k, n]$ which is the line drawn from the origin to $node[j, k, n]$. Five different projection directions are considered. The $[1, 0, 0]$, $[0, 1, 0]$ and $[0, 0, 1]$ vectors project along the directions of the j, k and n axis respectively. The $[1, 1, 0]$ and $[1, 1, 1]$ vectors are also considered.

4.4.6 Cholesky Decomposition Systolic Arrays

Of the five Cholesky decomposition systolic arrays discussed those produced by the $[1, 0, 0]$, $[0, 1, 0]$, $[0, 0, 1]$ and $[1, 1, 0]$ projection vectors are new. The $[1, 0, 0]$ systolic array bears slight similarity, in terms of its orthogonally connected architecture and input data ordering, with an array reported by Jain et al. [123]. However the referenced array contains extra multiply accumulate PEs along its diagonal border which are unnecessary, increasing hardware cost and computational delays. The faults with their array are no doubt the result of an uncoordinated algorithm to systolic array mapping strategy and comparison with the $[1, 0, 0]$ Cholesky systolic array presented here clearly demonstrates the superiority of the DDG methodology in the design of efficient systolic arrays. The final Cholesky array to be discussed is that produced by the $[1, 1, 1]$ mapping, initially proposed by Brent & Luk [80].

(A) [1,0,0] Vector Projection

Figure 4.4(a) shows the orthogonally connected systolic array produced by the $[1,0,0]$ projection vector. The j direction arcs are represented by the feedback buses around each of the PEs. Part (b) of the figure shows the architecture of the two different types of processor that are required. Each processor handles two different modes and the circuitry for each of the different states is detailed in figure 4.4(c). Switches between the different modes are made by enabling the clocks on the rl registers and by multiplexor selection. The timing assignment $t = j + k + n - 3$ is used to determine the input scheduling by considering the execution times of nodes in the $n = 1$ plane of the DDG of figure 4.3. Output is retrieved from feedback loops of PEs along the diagonal edge with the use of a multiplexor to route the k direction $l[j,k]$ $(j > k)$ elements onto this bus. A pipelined control strategy is indicated in figure 4.4 for this example but for reasons of clarity control lines are not shown in the rest of the arrays discussed.

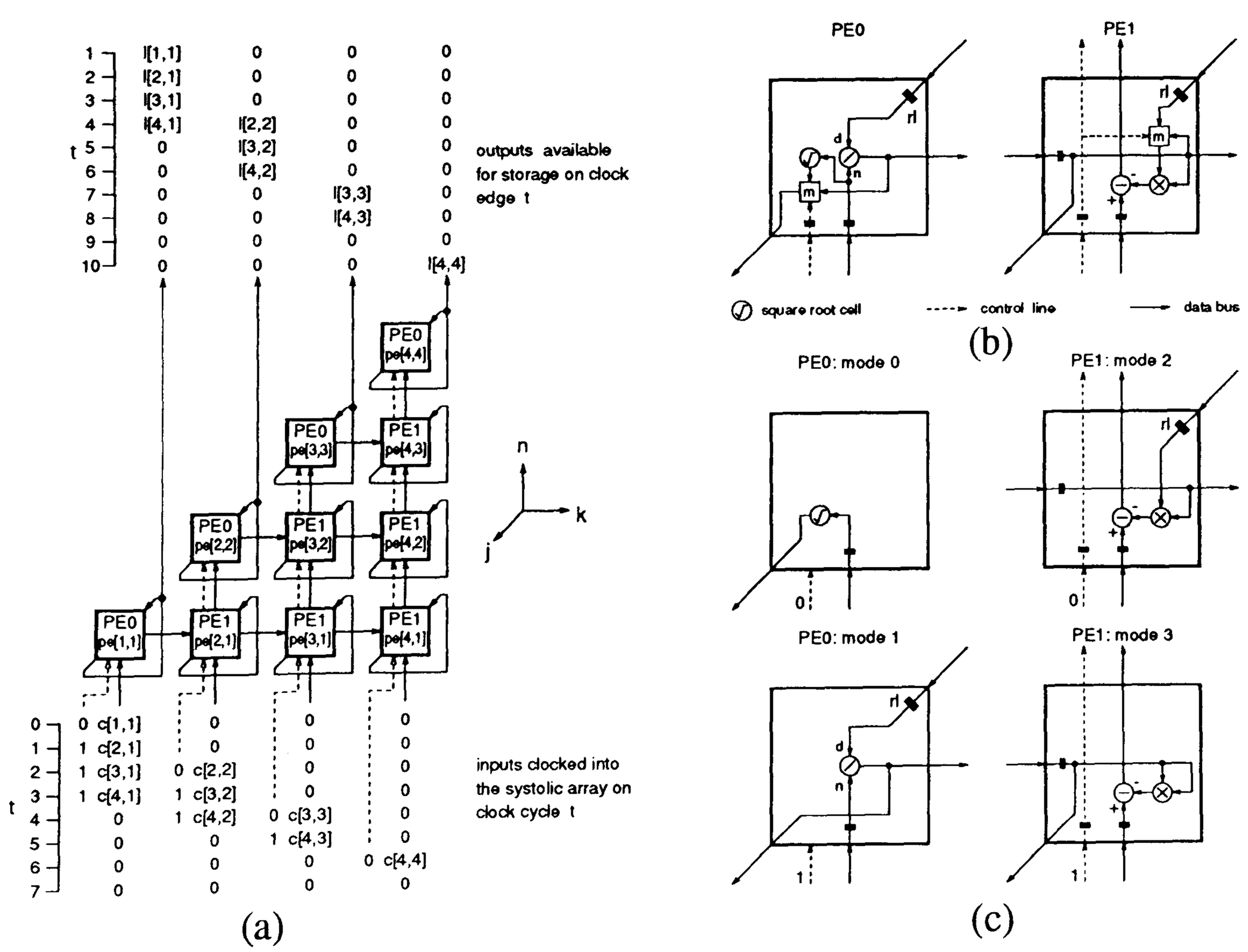


Figure 4.4: (a) Systolic array produced by $[1,0,0]$ projection vector, (b) PEs, (c) PEs configured in different modes.

(B) [0,1,0] Vector Projection

When the DDG is systolized using the $[0,1,0]$ vector $node[j,k,n]$ is mapped into $pe[j,n]$. This vector projects the DDG in the k direction resulting in the orthogonally systolic array of figure 4.5(a). This array has a very similar architecture to that of the $[1,0,0]$ mapping in figure 4.4(a) except that PE feedback links are not required on the type $PE0$ processors. Again there are two different types of PE (figure 4.5(b)) but here the type $PE0$ processors are only required to operate in *mode* 0 whilst the processors of type $PE1$ must handle the other three modes. The same timing function $t = j + k + n - 3$ is used again so that the operation of the array is complete at $t = 10$. One advantage of this array is that data input and output is straightforward eliminating the need for extra multiplexing functions thus alleviating the control signal complexity.

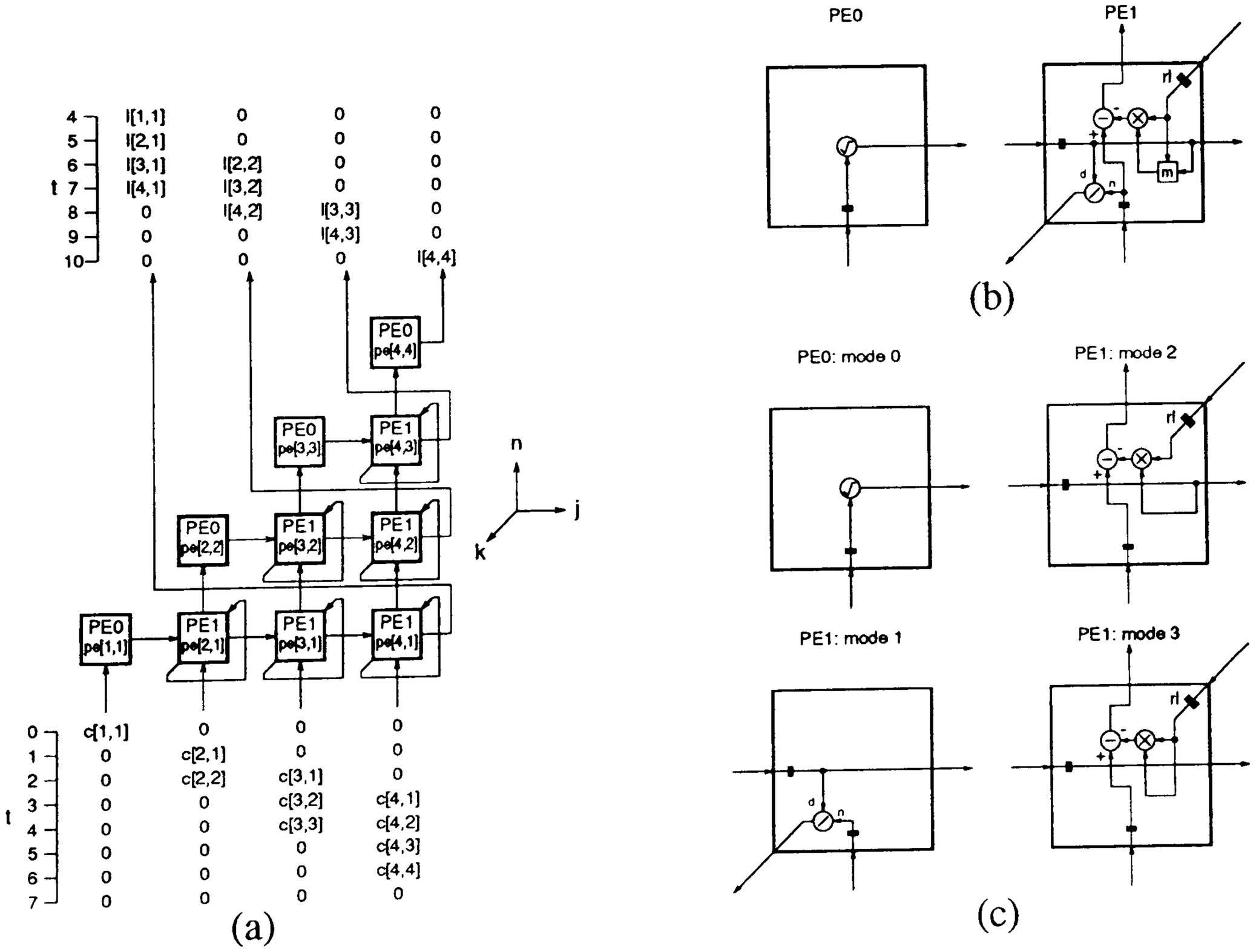
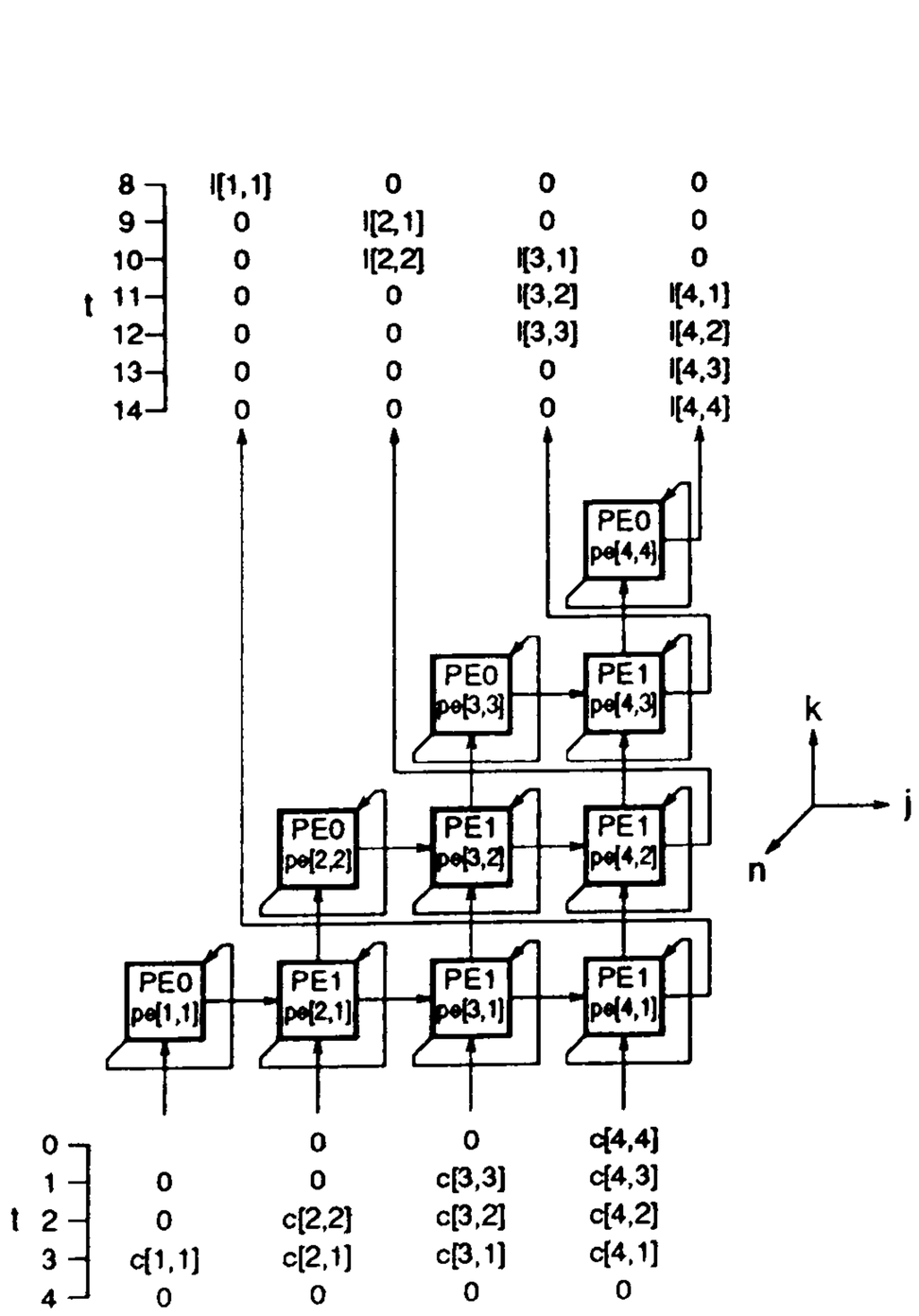


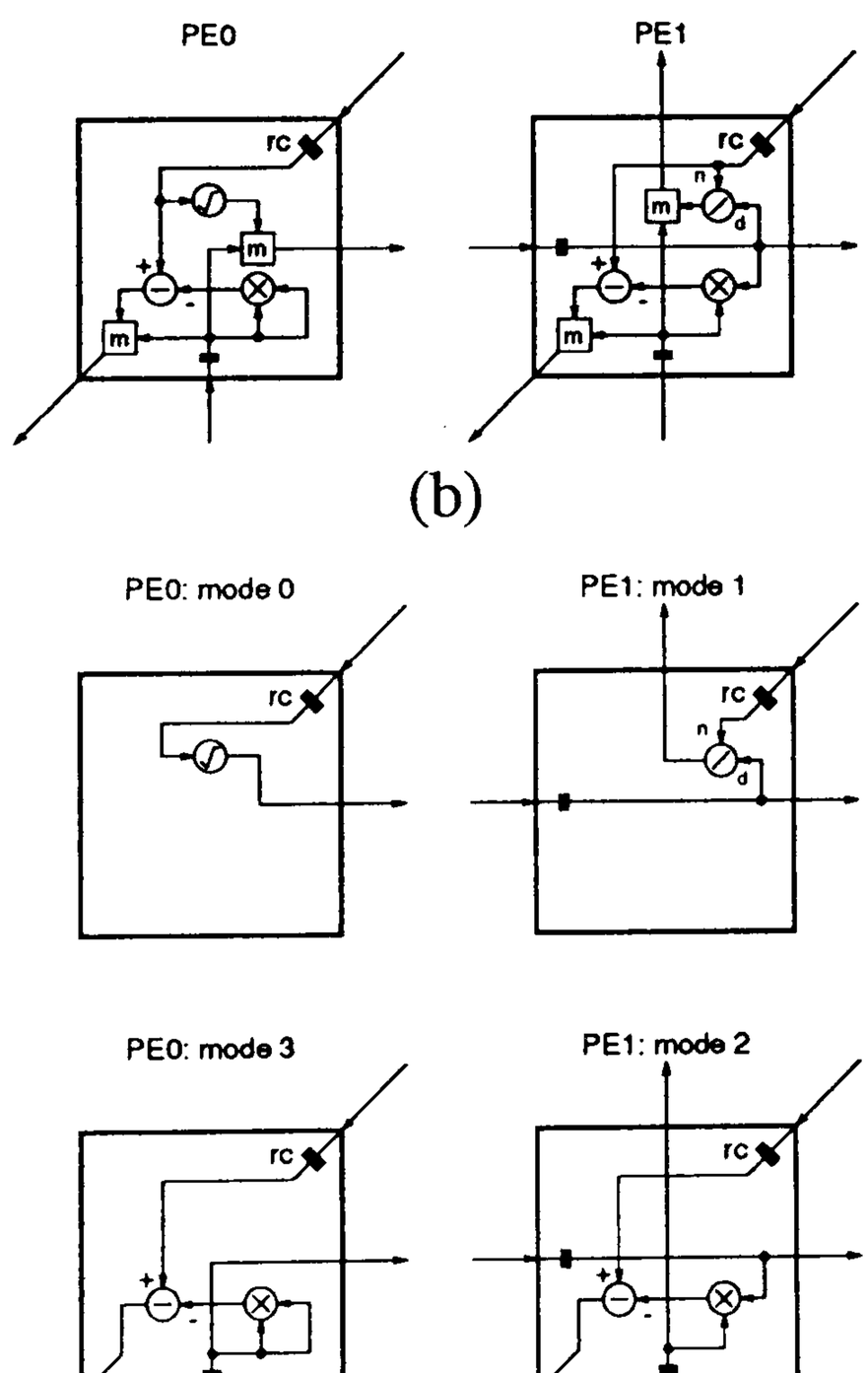
Figure 4.5: (a) Systolic array produced by $[0,1,0]$ projection vector, (b) PEs, (c) PEs configured in different modes.

(C) [0,0,1] Vector Projection

The $[0,0,1]$ vector maps the DDG into the systolic array shown in figure 4.6(a). Once again the PEs are orthogonally connected with feedback links around the PEs. The two types of processor illustrated in 4.6(b) each are required to compute in two different modes. The type $PE0$ processors work in *mode* 0 and *mode* 3 leaving the *mode* 1 and *mode* 2 operations for the type $PE1$ processors (figure 4.6(c)). Projection in the direction n leads to a problem in that the matrix $C^{(1)}$ (4.17) must be loaded into the rc registers of each PE before computation may commence. An extra multiplexor is included in each PE to configure the array for loading using the k direction buses. The DDG timing function becomes $t = j + k + n + 1$ and so operation does not begin until $t = 4$ and ends at $t = 14$. This procedure adds to hardware cost and control complexity.



(a)



(b)

(c)

Figure 4.6: (a) Systolic array produced by $[0,0,1]$ projection vector, (b) PEs, (c) PEs configured in different modes.

(D) [1,1,0] Vector Projection

The allocation function that maps $node[j,k,n]$ into $pe[j-k+1,n]$ corresponds to the projection vector $[1,1,0]$ and is used to derive the systolic array shown in figure 4.7(a). An orthogonal architecture is still displayed but here the PE feedback is replaced by a bidirectional data flow. The advantage of the new architecture is that it eliminates the need for clock enable signals which were previously required in the *rl* feedback loop registers of the $[1,0,0]$ and $[0,1,0]$ projections. The two different types of processor $PE0$ and $PE1$ shown in figure 4.7(b) each handle two modes, the former operating in *mode* 0 and *mode* 3, the latter handling *mode* 1 and *mode* 2. The configurations modes of these PEs are illustrated in figure 4.7(c). The optimal timing function $t = j + k + n - 3$ is applied to the DDG and operation is completed at $t = 10$.

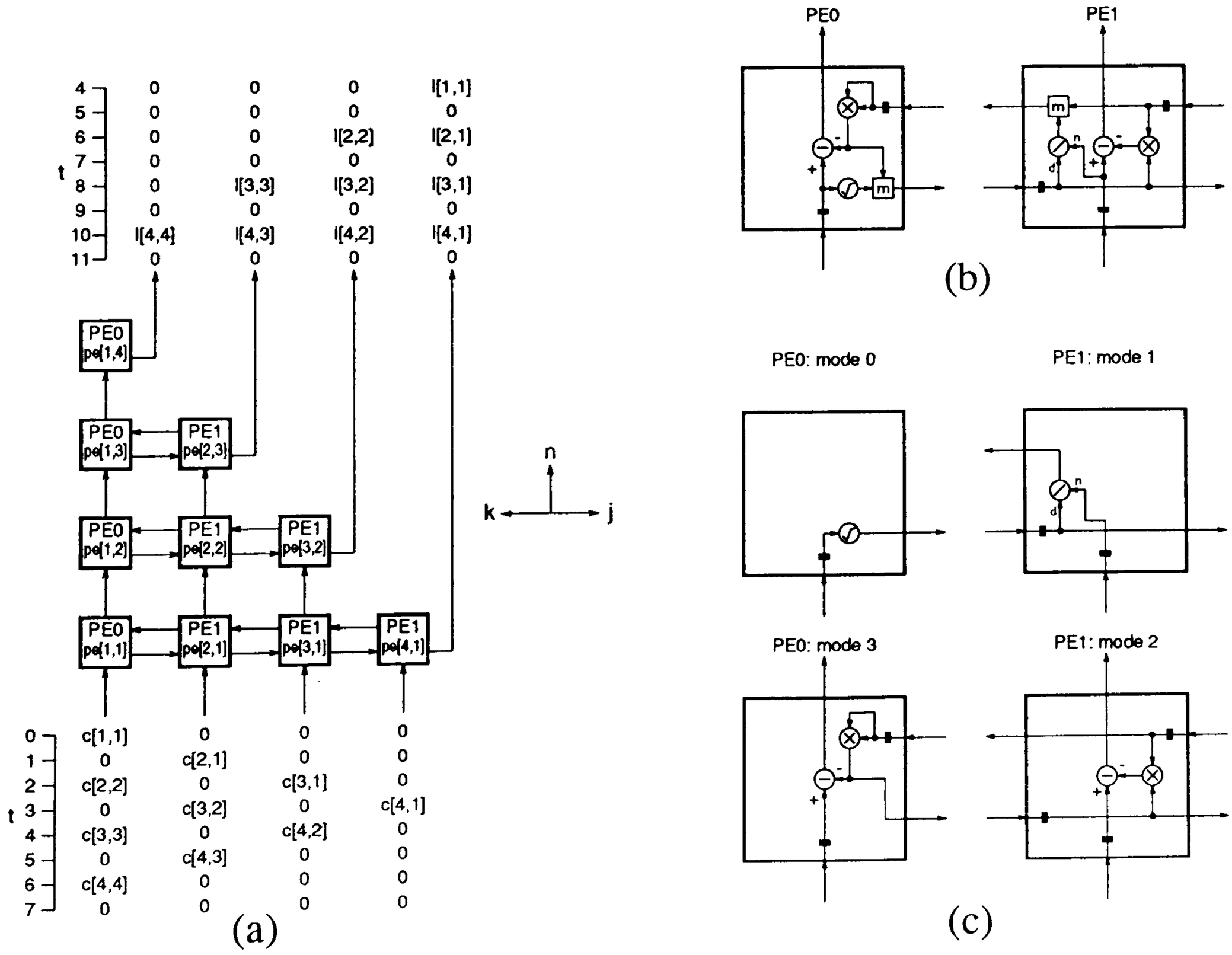


Figure 4.7: (a) Systolic array produced by $[1,1,0]$ projection vector, (b) PEs, (c) PEs configured in different modes.

(E) [1,1,1] Vector Projection

The $[1,1,1]$ projection vector, which maps the operation of $node[j,k,n]$ into $pe[j-n+1, k-n+1]$, produces the systolic array shown in figure 4.8(a). This hexagonally interconnected architecture is topologically equivalent to that proposed by Brent and Luk [80]. There are four different types of processor in the array which has a distinct advantage since each type is only required to work in a single mode, eliminating the multiplexing functions which are needed to switch between modes in the other mappings, making control simple. The timing function $t = j + k + n$ is applied to the DDG as three clock cycles are required to set up the input data. The array does differ slightly from Brent & Luk's design [80] in that their $PE0$ performs square-root/reciprocal while the PEs of type $PE1$ are multiplication. Here the type $PE0$ processor performs just square-root and to compensate the $PE1$ processors compute division approximately doubling the maximum allowable systolic clock frequency.

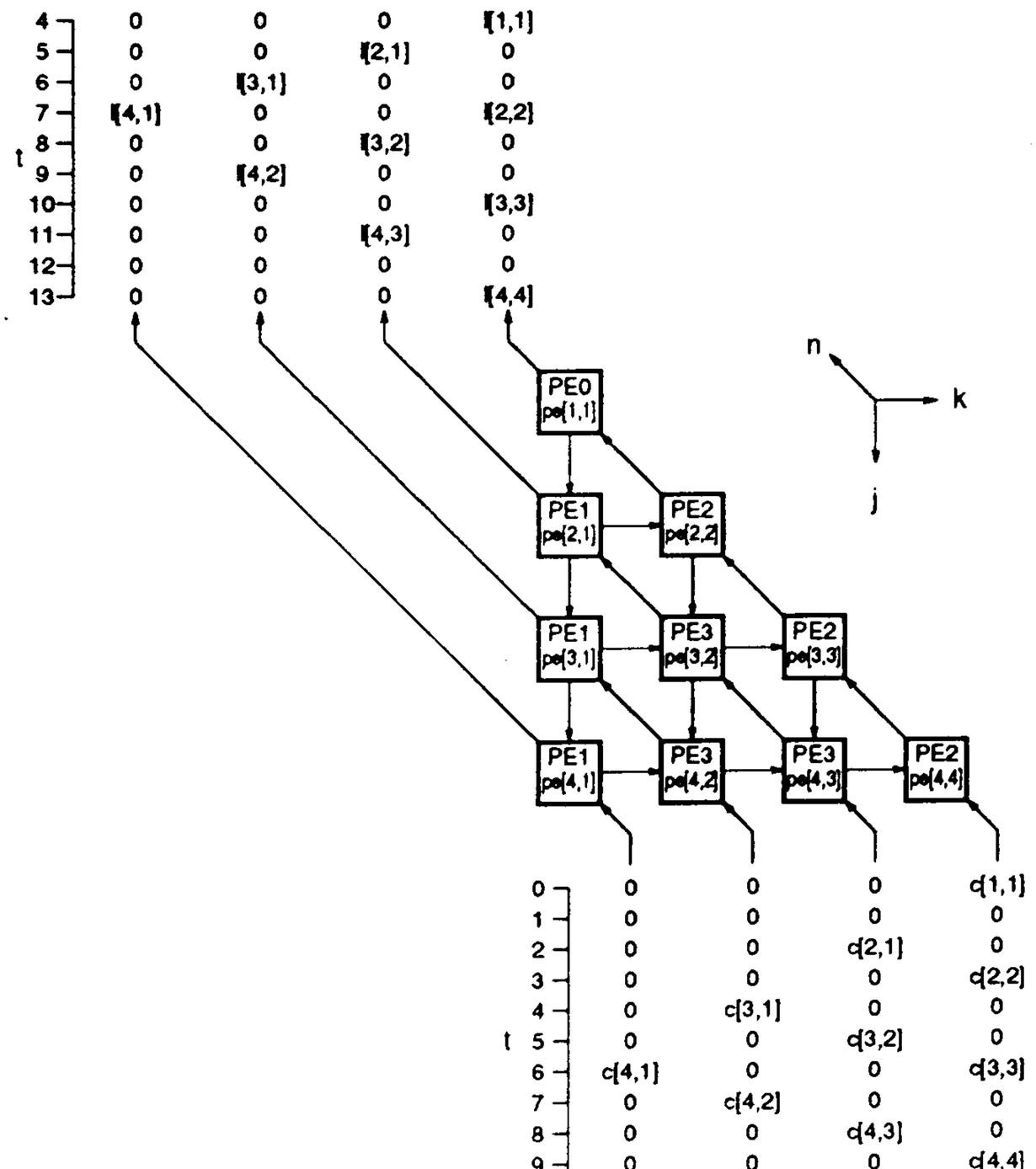


Figure 4.8: (a) Systolic array produced by $[1,1,1]$ projection vector, (b) PEs.

4.4.7 Comparison of Cholesky Decomposition Arrays

The five systolic arrays represent a good cross section of solutions for implementation of the Cholesky decomposition algorithm. The arrays presented use the most obvious mapping directions and it is likely that the systolic arrays produced are in an optimal set. Other, not so obvious mappings are possible but these are likely to result in systolic arrays with more complicated communication requirements, greater processor complexity and bigger pipelining delays. A similar analysis to that used in section 3.4 for comparison of the matrix element calculation systolic arrays is used for selection of the optimal mapping. For completeness both word-parallel and bit-serial approaches are considered even though the latter is most likely to be selected due to the lower computational burden of this part of the estimator.

4.4.8 Cost Estimation

The design of division and square-root hardware units needs to be considered before the cost of the Cholesky decomposition arrays can be estimated.

(A) Division Unit Cost Estimation

The bit-level DDG for division (3 bit output) shown in figure 4.9(a) is based on the non-restoring array divider proposed by Guild [88][124]. Each row of the DDG performs either addition or subtraction of the divisor to or from the partial remainder dividend, that is the word formed on the s_i inputs to a row. The operation mode depends on the state of the controlled-add-subtract (CAS) cell *ctr* input (figure 4.9(b)) which is set by the result quotient bit produced from the output carry of the previous iteration.

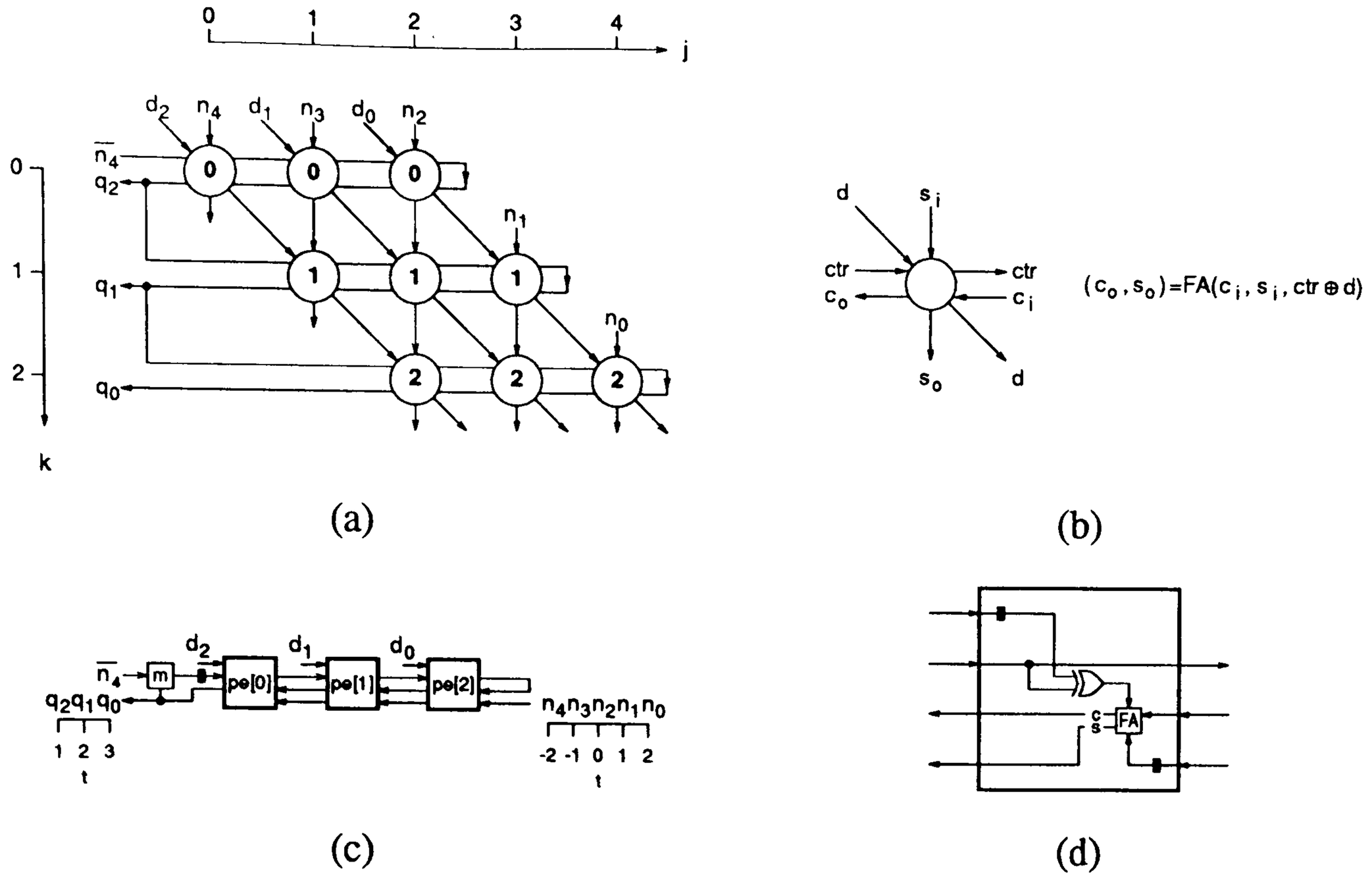


Figure 4.9: (a) DDG for the non-restoring division $Q = N/D$ where quotient $Q = q_2q_1q_0$, numerator $N = n_4n_3n_2n_1n_0$, divisor $D = d_2d_1d_0$, (b) CAS node function, (c) bit-serial semi-pipelined arithmetic unit and (d) CAS division cell.

A word-parallel ripple through divider is formed from a direct node to PE mapping and would therefore require w^2 PEs. Operation is however fairly slow as carries must ripple through each row of the DDG before a quotient bit can be calculated, the result of which determines the CAS mode operation for the next row of nodes. Alternatively a bit-serial array can be produced by assigning a timing and allocation function to the graph. The timing allocation $t = k$, such that a bit of the quotient is calculated on each clock cycle, and the projection by the [1,1] vector, which allocates $node[j, k]$ to $pe[j - k]$, results in the semi-systolic array shown in figure 4.9(c) whose CAS PE function is given in part (d). The bit-serial array is semi-systolic as while the sum input line is pipelined the carry is left to ripple through. Full pipelining of the carry is possible but this would increase the number of delays required on the sum pipeline to w leading to higher cost. The bit-serial divider is an MSB first device, unlike the adders and multipliers which are LSB first (figure 3.19), and therefore some data reordering is required between PEs.

(B) Square Root Unit Cost Estimation

Square root can also be calculated using non-restoring arithmetic. The DDG in figure 4.10(a) is based on the schematic for a non-restoring square-root extractor built with CAS cells. However the communication strategy demonstrated in the DDG in figure 4.10(a) is more regular and is localised throughout. The localised variations in the communication of the square-root unit proposed in [88] are transformed here to particular modes of the nodes. There are 4 modes in total (figure 4.10(b)) and therefore a general purpose PE design would require 2 control lines for mode selection. The cells lying in the shaded backdrop area directly correspond to those for a 6 bit version of the non-restoring array and are all that is required for the word-parallel ripple through version. Extra *mode* 0 nodes extend the array to a more regular shape for projection by the $[1, 1]$ vector which results in the semi-pipelined bit-serial array shown in figure 4.10(c).

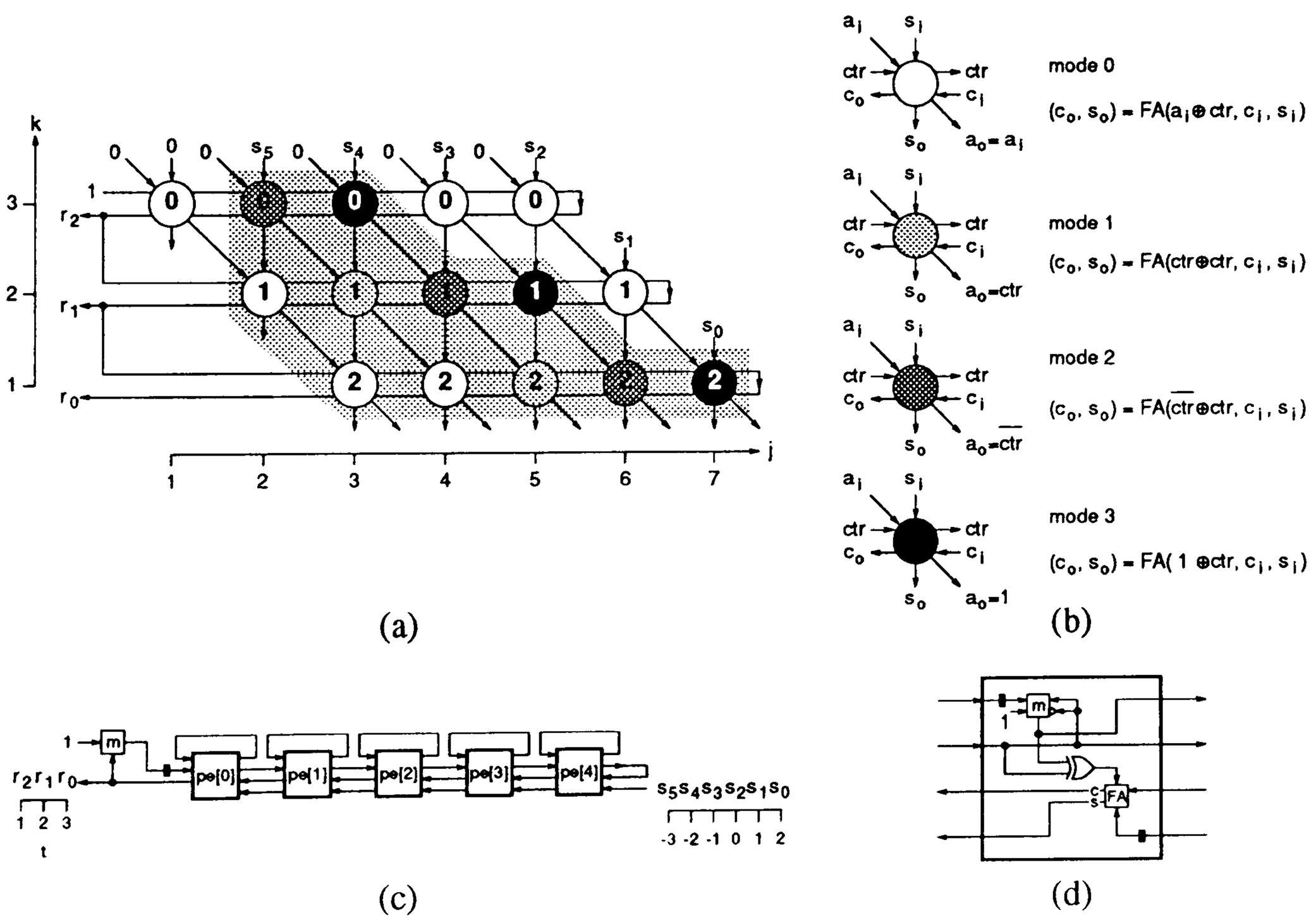


Figure 4.10: (a) DDG for non-restoring square-root, (b) CAS node function, (c) bit-serial semi-pipelined arithmetic unit and (d) CAS bit-serial array PE.

(C) Summary of Division and Square Root Unit Cost

Table 4.1 details approximate NAND gate costs for these modules in terms of word-length w. The word parallel costs are based on non-restoring arrays shown in figures 4.9 and 4.10. The cost of a CAS cell is given by CAS=FA+XOR which is the equivalent of 16 NAND gates.

w bit module	approach	hardware cost (NANDs)
division	word-parallel	CAS.w^2
	bit-serial	(CAS+2.DFF).w+MUX+DFF
square-root	word-parallel	(CAS+3.MUX+1).$(w^2 + 5w - 2)/2$
	bit-serial	(CAS+2.DFF+3.MUX+1).(w+2)+MUX+DFF

Table 4.1: Cellular construction cost of square root and division modules.

(D) Modular Hardware Cost for the Cholesky Decomposition Systolic Arrays

The hardware usage for each of the five systolic arrays formed by projection of the Cholesky decomposition DDG is demonstrated in terms of the number of modules in table 4.2. This can be used with the information in tables 3.1, 3.2 and 4.1 to estimate the total NAND gate requirement of each mapping.

w bit module	number of modules used in the arrays				
	[1,0,0]	[0,1,0]	[0,0,1]	[1,1,0]	[1,1,1]
addition	0	0	0	0	0
subtraction	6	6	10	10	6
multiplication	6	6	10	10	6
division	4	6	6	6	3
square root	4	4	4	4	1
register	22	22	26	26	18
multiplex	10	6	10	10	0

Table 4.2: Hardware usage for Cholesky systolic arrays in terms of module usage.

(E) Hardware Cost Estimation Results

The results of the hardware cost estimation for the Cholesky decomposition systolic arrays can be seen in figure 4.11. As expected the bit-serial costs are lower than the word-parallel. However, with regard to relative hardware usage in both approaches the results are very similar. The $[1,1,1]$ shows the least hardware usage and the $[1,0,0]$ projection also displays good results. The $[0,0,1]$ and $[1,1,0]$ arrays display the largest NAND gate usage for all bus widths.

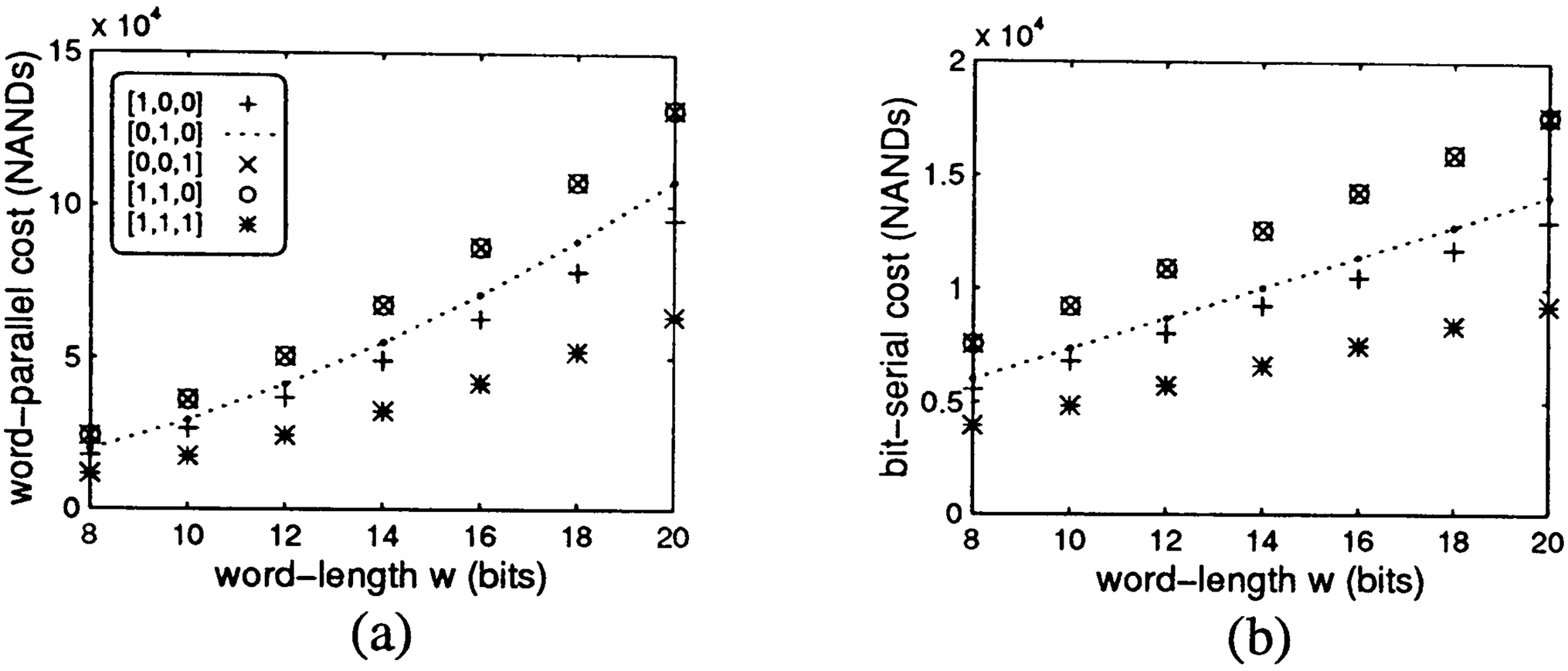


Figure 4.11: Cost for (a) word parallel and (b) bit-serial approaches.

(F) Data Communication Cost

The various communication costs incurred in each of the 5 Cholesky decomposition systolic arrays are summarised in table 4.3. The diagonal PE interconnections are weighted by $\sqrt{2}$ relative to the orthogonal interconnections to allow for their longer physical length, approximated using simple geometry. The communication cost is similar for all the arrays considered. The $[1,1,1]$ and $[1,1,0]$ systolic arrays require the greatest amount of PE interconnection but do not need any feedback links around the PEs. The j, k and n projections form systolic arrays that require less PE interconnections but these do need feedback links around the

PEs. The PE interconnection links burden pin resources if it is not feasible to mount the whole systolic array on a single integrated circuit while the feedback links are made internally on the IC due to their extreme locality. This therefore leads to preference of the j, k and n arrays when considering the communication factor. All of the arrays share the same input/output burden.

communication type	number of data buses				
	$[1,0,0]$	$[0,1,0]$	$[0,0,1]$	$[1,1,0]$	$[1,1,1]$
PE interconnection	12	12	12	18	$12+6\sqrt{2}$
localised feedback	10	6	10	0	0
array input	4	4	4	4	4
array output	4	4	4	4	4

Table 4.3: Data communication cost.

(G) Control Cost

Clock, reset and module level control signals such as an MSB indicator bit in a two's complement multiplier are common to all of the designs and so do not serve for comparison. Extra control signals must be generated in order to switch multiplexors and enable clocks for processor mode selection and input/output data retrieval and each mapping has its own requirements. The $[1,1,1]$ array needs no extra control since each PE works in single mode during calculation. All the other arrays require mode switching signals; the $[1,0,0]$, $[0,0,1]$ and $[1,1,0]$ mappings each contain two different types of PE which need to be switched between two different modes while the $[0,1,0]$ array uses two PE types one of which works in three of the modes, the other in fixed mode. The $[0,0,1]$ array also requires extra control signals to load data into the array for initialisation.

4.4.9 Efficiency

Table 4.4 summarises computation time in terms of number of clock cycles N_{clk}, processor pipelining period α and block pipelining period β for each of the five different Cholesky decomposition arrays discussed.

timing factor	number of clock cycles				
	$[1,0,0]$	$[0,1,0]$	$[0,0,1]$	$[1,1,0]$	$[1,1,1]$
N_{clk}	10	10	14	10	13
α	1	1	1	2	3
β	4	4	10	8	12

Table 4.4: Processor/block pipelining periods and block latency.

(A) Block latency

The $[1,1,1]$ and the $[0,0,1]$ arrays come off worst in terms of the number of clock cycles from input of the first data word to output of the last since the input data has to loaded into these arrays to before the calculation can begin. The other arrays all require a total of ten clock edges.

(B) Processor/Block Pipelining Period

The PEs of the $[1,1,0]$ and $[1,1,1]$ arrays are active on one in every 2 and 3 clock cycles respectively and so their PEs display at best 50% and 33% efficiency. The block pipelining period of the hexagonal array is determined by the time that its processor $pe[0]$ is active and the results show that a new decomposition may be processed every 12 clock pulses compared to 8 for the $[1,1,0]$ array. However the pipelining periods of these two arrays can be improved as the zeroes which are input between the existing data may be replaced with new data sets. The $[1,1,0]$ and $[1,1,1]$ mappings could respectively work on 2 and 3 sets of

data simultaneously leading to higher PE utilisation and average block pipelines of 4 clock signals matching the performance of the $[1,0,0]$ and $[0,1,0]$ arrays. Although displaying good PE utilisation the $[0,0,1]$ array can only process a new set of data after 10 clock pulses due to the input data loading restrictions.

4.4.10 Selection of the Optimal Cholesky Decomposition Array

Of the five different arrays for Cholesky decomposition produced by mapping its DDG in different directions, the hexagonal systolic array formed by the $[1,1,1]$ projection vector displays the lowest hardware cost. Analysis shows that this systolic array also matches the best performance of any of the other Cholesky decomposition arrays in terms of processor and block pipelining when a data interleave strategy is adopted. Hence there is no degradation of throughput when using this array even though the total operation time for one complete block of data is greater than most of the other designs by three clock cycles. None of the PEs require any mode switching and for this reason the $[1,1,1]$ array is the easiest to control. Its only drawback comes from its greater processor interconnection burden but the advantages of the array become over-riding when considering the bit-serial approach where communication cost becomes less significant.

The $[1,0,0]$ systolic array presents low communication burden, compared with the $[1,1,1]$ array. The $[1,0,0]$ array is also very efficient as a processor pipeline period of $\alpha = 1$ can be achieved without having to interleave data sets from successive windows. Despite these advantages however its use cannot be justified as the hardware cost of the $[1,0,0]$ array, although being lower in comparison with most of the Cholesky decomposition arrays formed by different mappings, is in fact higher than the cost of the $[1,1,1]$ array, and control is significantly more complex as processor modes are required to be switched.

4.5 Non-Square-Root Decomposition Methods

Study of the non-restoring word-parallel and bit-serial designs presented in figure 4.10 demonstrates the problems associated with VLSI implementation of square-root. Firstly the PE cost is high, each cell being required to perform a CAS type function with a 4 to 1 line multiplexor required for mode selection. Despite the square-root design having a lower number of PEs in the word-parallel approach when compared to an equivalent word-length multiplier, the square-root design presents higher overall hardware cost and the irregular shape of the array makes efficient floor-planning difficult. The bit-serial square-root design also presents relatively high hardware cost and control signal generation cost to switch each PE between the four modes. Due to the problems encountered in the VLSI implementation of the square-root, this section studies non-square-root LU and LDL^T decomposition using the results from the square-root Cholesky decomposition example to narrow down the field of projection vectors considered.

4.5.1 LU Decomposition Algorithm

In the LU decomposition the covariance matrix is split into the product of a lower triangular matrix L and an upper triangular matrix U, as is the case for Cholesky decomposition, except that here L now has ones along its diagonal and U is no longer the transpose of L. Therefore the decomposition of the matrix C is defined as:

$$C = L.U \tag{4.22}$$

where for the model order 4 system:

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ l[2,1] & 1 & 0 & 0 \\ l[3,1] & l[3,2] & 1 & 0 \\ l[4,1] & l[4,2] & l[4,3] & 1 \end{bmatrix} \tag{4.23}$$

$$U = \begin{bmatrix} u[1,1] & u[1,2] & u[1,3] & u[1,4] \\ 0 & u[2,2] & u[2,3] & u[2,4] \\ 0 & 0 & u[3,3] & u[3,4] \\ 0 & 0 & 0 & u[4,4] \end{bmatrix} \tag{4.24}$$

This method is more general than Cholesky because the matrix that is being decomposed does not have to be symmetrical. This does however have the disadvantage of approximately doubling the number of computations required. As with the Cholesky decomposition, substitution allows two triangular sets of simultaneous equations to be formed. The lower triangular system:

$$L.Y = B \tag{4.25}$$

is initially solved by forward elimination using the systolic array shown in figure 4.2. This array can however be simplified when used with LU decomposition because the ones along the leading diagonal of L allow the dividers in the type $PE0$ processors of the forward elimination array to be omitted. The filter parameters may then be derived by solution of the upper triangular system:

$$U.A = Y \tag{4.26}$$

using the back substitution systolic architecture shown in figure 4.2(b).

Golub and Van Loan [27] details the LU decomposition algorithm which may be converted into the form of the following set of single assignments.

$$c^{(1)}[j,k] = c[j,k] \tag{4.27}$$

$$c^{(n+1)}[j,k] = \begin{cases} 0 & j \leq n \\ 0 & k \leq n \\ c^{(n)}[j,k] - l[j,n].u[n,k] & j > n, k > n \end{cases} \tag{4.28}$$

$$l[j,k] = \begin{cases} 0 & j < k \\ 1 & j = k \\ c^{(k)}[j,k]/u[k,k] & j > k \end{cases} \tag{4.29}$$

$$u[j,k] = \begin{cases} 0 & j < k \\ c^{(j)}[j,k] & j \geq k \end{cases} \tag{4.30}$$

Since the symmetry of the covariance matrix is not utilised then each iteration of the matrix $C^{(n+1)}$ (4.28) contains square sections of non-zero elements $c^{(n+1)}[j,k]$ $(j > n, k > n)$. The difference between the Cholesky and LU decompositions can be seen by comparing (4.16) with (4.29). In the Cholesky case the assignment of $l[j,k]$ for the $j = k$ condition results in a square-root operation while in the LU decomposition these elements are simply assigned to unity and the added computation of U is no more than a set of straightforward assignments (4.30).

4.5.2 LU Decomposition Data Dependence Graph

The DDG for the LU decomposition shown in figure 4.12(a) is derived using a similar procedure to that used for the Cholesky decomposition DDG in section 4.4.2. Only three types of node are required for the LU DDG and their functions are noted in figure 4.12(b).

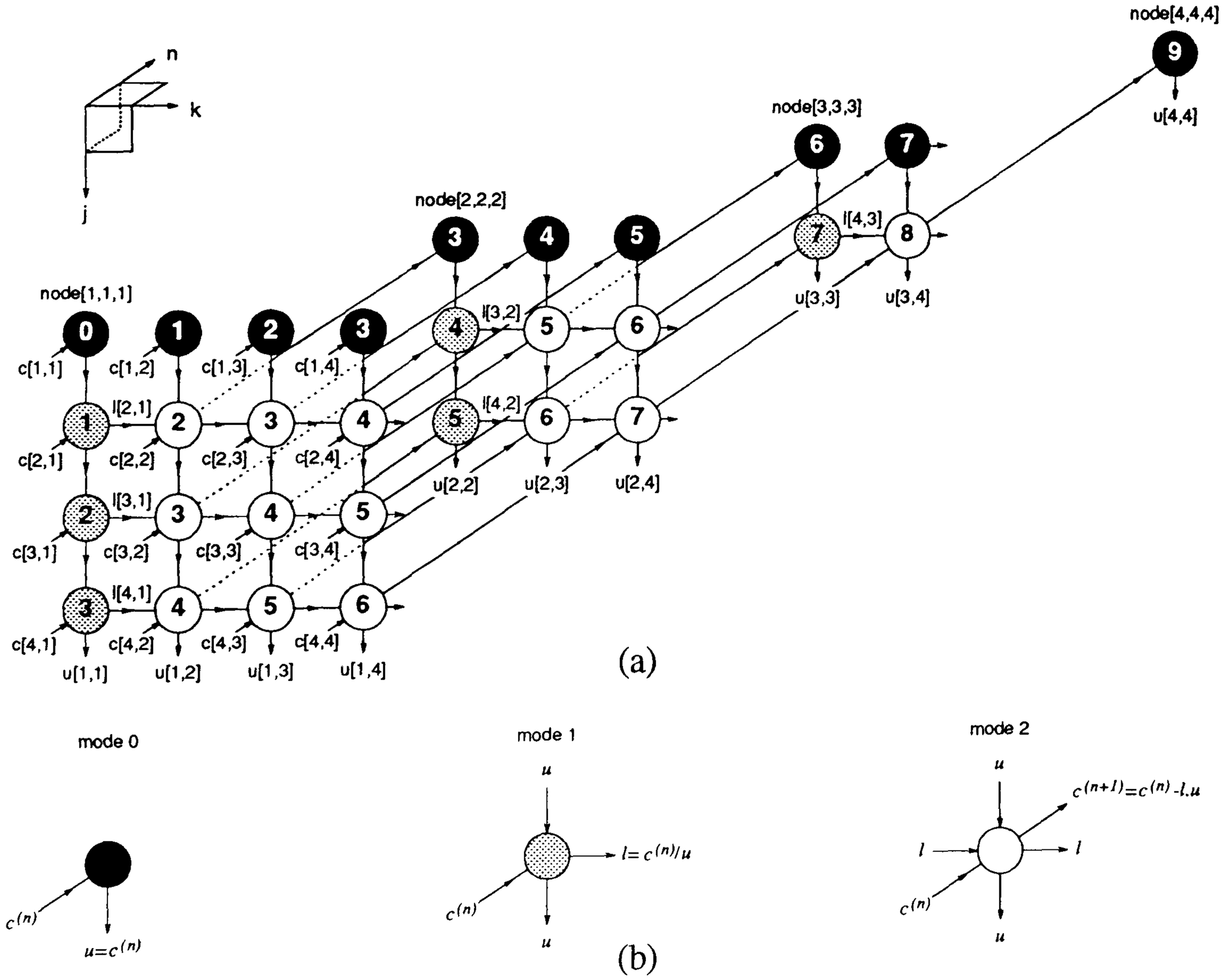


Figure 4.12: (a) DDG for LU decomposition, (b) key to symbols.

4.5.3 LU Decomposition DDG Localisation and Timing

The communication in the *LU* DDG of figure 4.12 is localised with unit pipeline delays along j, k and n direction communication paths. The same schedule (4.21) as for the Cholesky decomposition DDG from figure 4.3 is indicated by the internal node numbering on the graph in figure 4.12. Despite the fact that the graph contains more nodes (representing a greater number of computations) than the Cholesky DDG, operation is still complete after the same number of clock cycles as both graphs operate on the same timing function which gives the order in which computations take place for parallel processing.

4.5.4 LU Decomposition DDG Projection

The previous section identified the advantages and disadvantages for hardware implementation of five arrays formed by various projections of the Cholesky DDG. Complex control for array initialisation and poor block pipelining rate led to rejection of the $[0, 0, 1]$ mapping while high hardware and communication cost resulted in exclusion of the $[1, 1, 0]$ projection. The $[1, 1, 1]$ projection produced a hexagonally interconnected systolic array which had the lowest hardware cost at the price of a large communication burden. The $[1, 0, 0]$ and $[0, 1, 0]$ mappings displayed similar results to each other in terms of efficient hardware usage and low communication burden but did however require control signals to change the operating modes of the PEs.

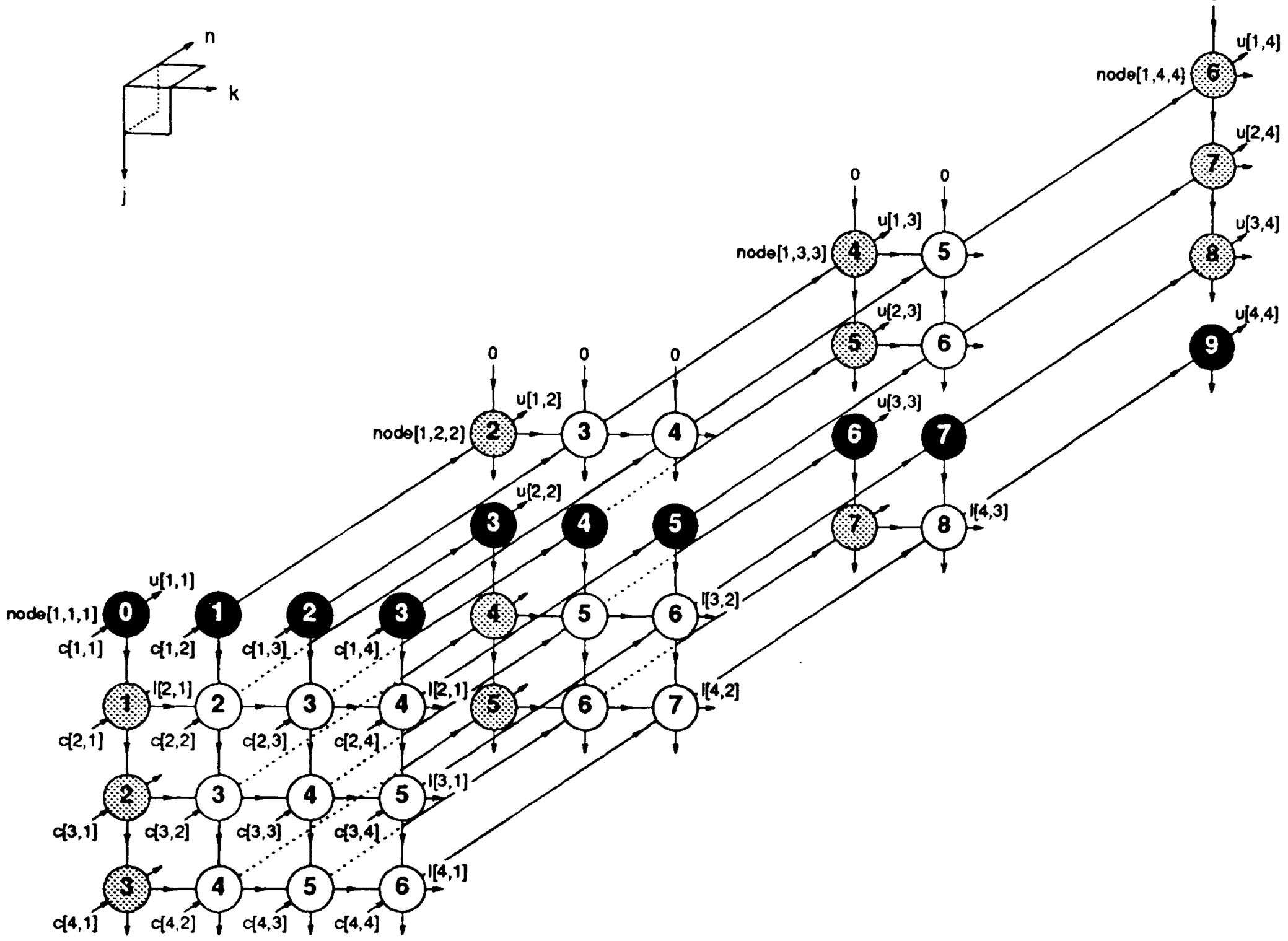
Due to the similarities between the *LU* and Cholesky DDGs then it is reasonable to assume that arrays produced by the same projection vectors share some similar traits, e.g. the *LU* $[0, 0, 1]$ mapping would also need an input data loading procedure. *LU* arrays produced by the $[1, 0, 0]$ and $[1, 1, 1]$ projection judged to be the best projection vectors for Cholesky are discussed and a new array for on-the-fly data retrieval when using the $[1, 0, 0]$ mapping is presented.

4.5.5 LU Decomposition Systolic Arrays

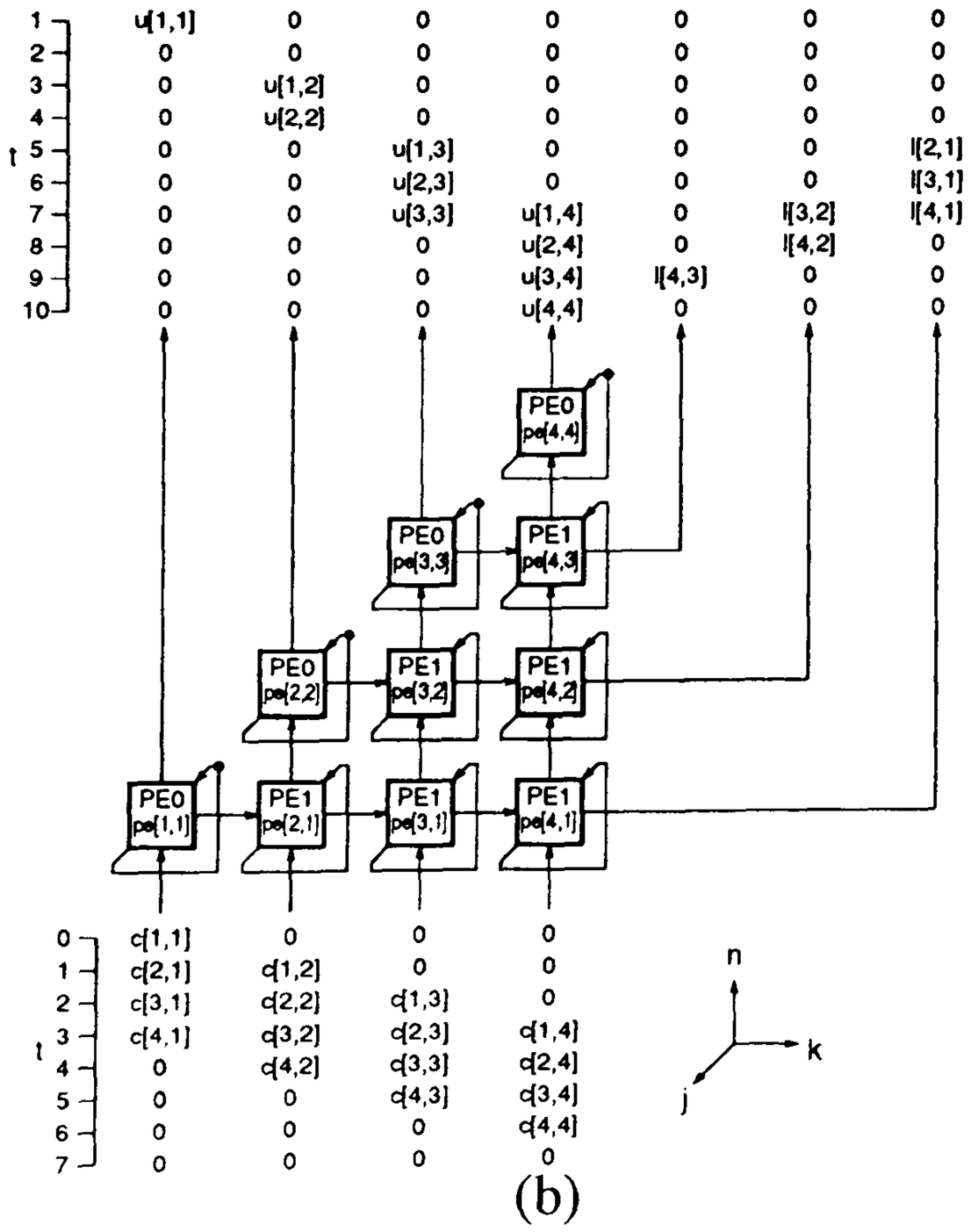
An array equivalent to that produced by the $[1,1,1]$ projection vector, first presented by Kung and Leiserson [17][122], is also discussed by Rajopadhye [79] and Megson [102]. Kung [23] briefly compares this array with that produced by the $[1,0,0]$ mapping. His conclusions are that the hexagonal array produced by the $[1,1,1]$ mapping has the advantage that its processors do not need reprogramming while the $[1,0,0]$ projection is attractive due to the fact that diagonal connections are not required and its PE utilisation is more efficient since it yields the same throughput with approximately 50% of the PEs.

(A) [1,0,0] Vector Projection

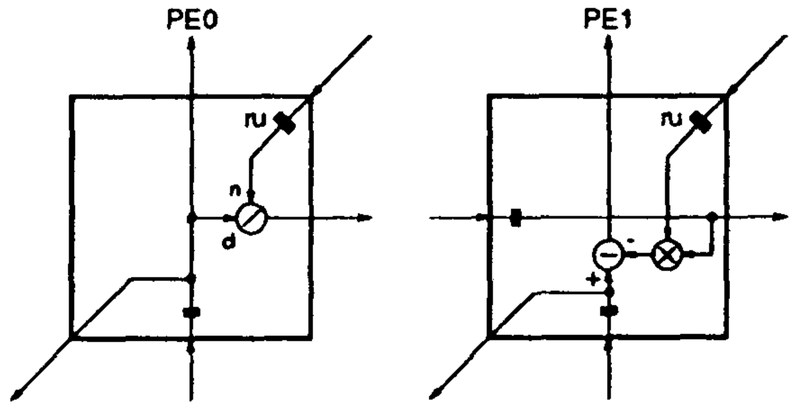
In the $[1,0,0]$ mapping the DDG is mapped in the j direction and so the elements $u[j,k]$ which are also pipelined in this direction get stored within the PE feedback loops at the end of the calculation. One method of data retrieval would be to multiplex the values onto the k direction buses at the end of the calculation after L has been output, but this however would increase the block pipelining period. A modification is proposed by re-representing the data flows in the form of the DDG of figure 4.13(a) to enable on-the-fly data output. The *mode* 0 nodes now have n direction outputs which are routed straight to the node input. Otherwise the modes are the same as for figure 4.12(b). Extra nodes are added to extend all the layers of the DDG up to the same height facilitating output of $u[j,k]$ elements from the n direction border of the graph without increasing the block pipelining period. The systolic array produced, shown in figure 4.13(b), is orthogonally connected with PE feedback links. A total of seven output buses are required, four for elements of U output from the diagonal border and three for elements of L available from the right hand edge PEs. PE descriptions are given in figure 4.13(c) with their mode configurations in (d). Both PE types are required to operate in *mode* 0 and the mode is changed by appropriately controlling the clock enable signal of the *ru* registers. Operation is complete by $t = 9$.



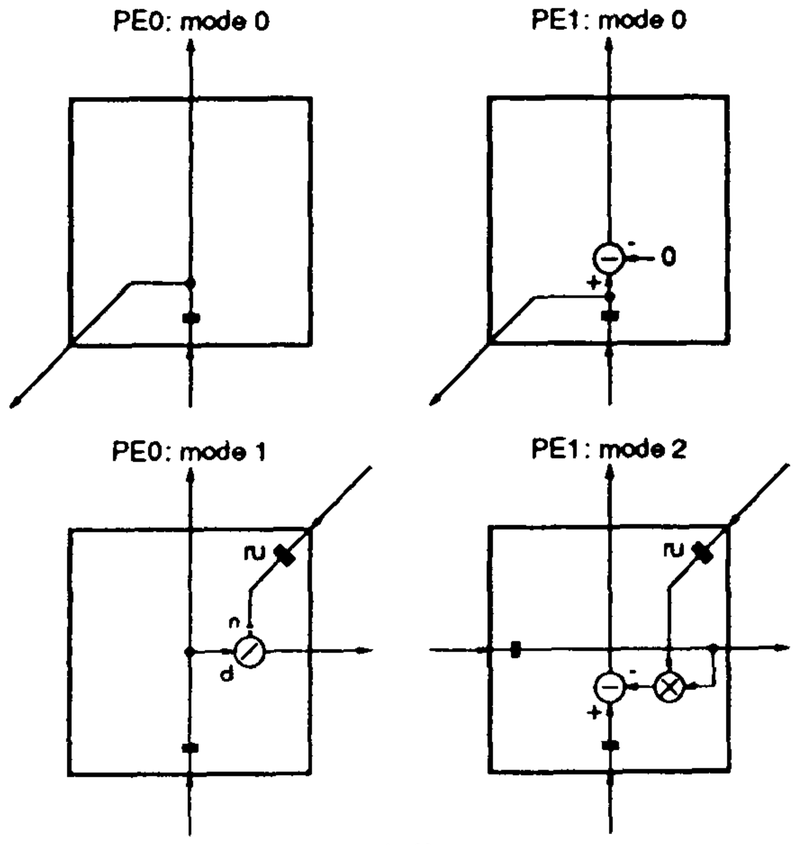
(a)



(b)



(c)



(d)

Figure 4.13: (a) Modified DDG for $[1, 0, 0]$ mapping, (b) systolic array produced by $[1, 0, 0]$ projection vector, (c) PEs, (d) PEs configured in different modes.

(B) [1,1,1] Vector Projection

This mapping results in Kung and Leiserson's array, a representation of which is shown in figure 4.14(a). The allocation function is the same as for the equivalent Cholesky mapping and therefore $node[j,k,n]$ of the original DDG in figure 4.12 is processed by $pe[j-n+1, k-n+1]$. The square shape of the n plane however results in the formation of a square shaped systolic array containing 16 PEs. There are three different types of processor in the array each corresponding to a particular mode as detailed in part (b) of the figure. The array has the hexagonal communication structure and does not require processor feedback as a consequence of this. The input/output burden is relatively large as there are seven input buses and seven output buses. The timing function used $t = j+k+n$ results in the first operation being carried out at $t = 3$ by $pe[1,1]$ and this processor also completes the decomposition on the $t = 12$ cycle.

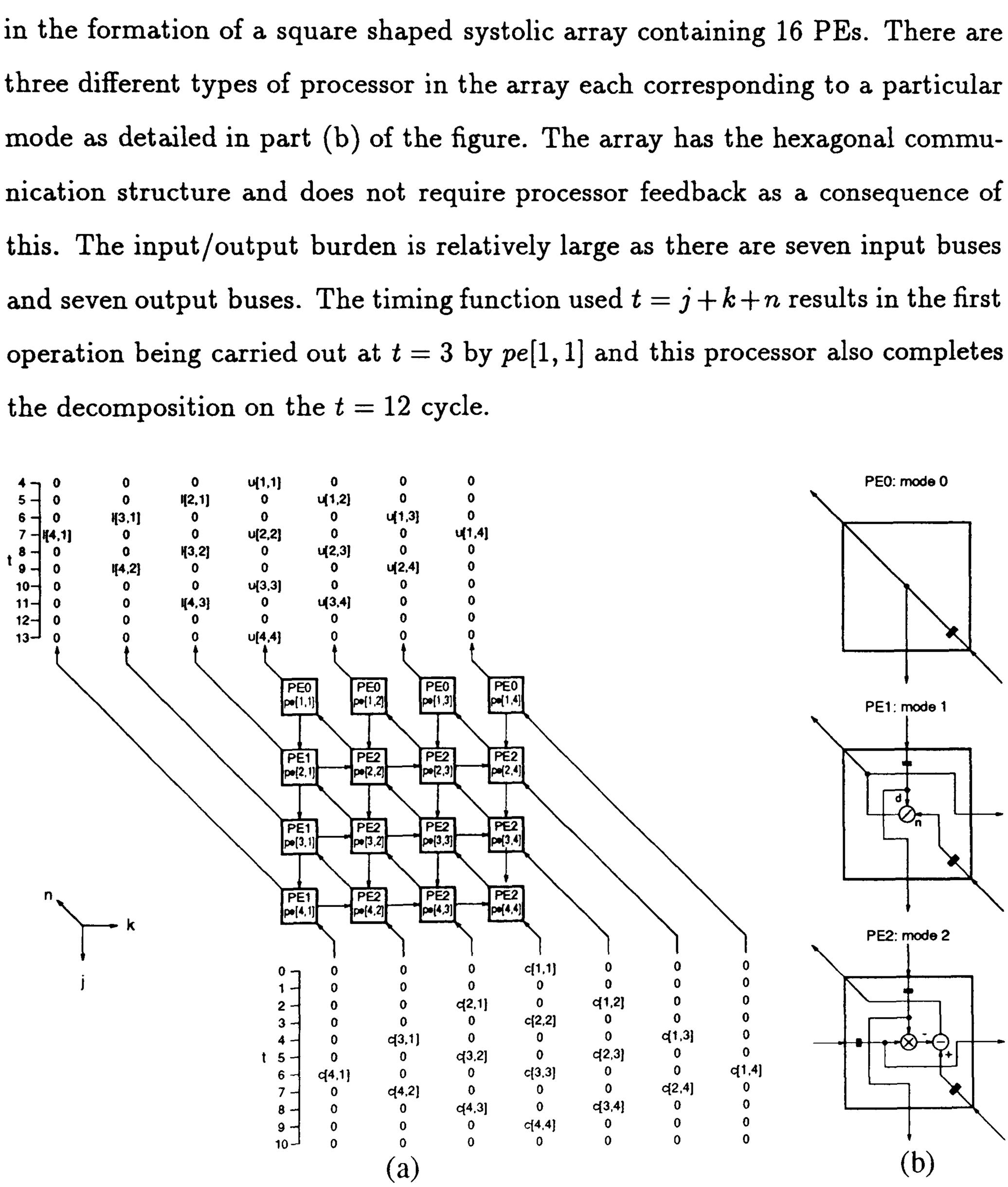


Figure 4.14: (a) Systolic array produced by $[1,1,1]$ projection vector, (b) PEs.

4.5.6 LDL^T Decomposition Algorithm

The LU decomposition algorithm may be refined for use with symmetric matrices as in LDL^T decomposition. The computational burden of this algorithm is therefore lower than the LU and similar to the Cholesky algorithm in terms of number of operations. In the LDL^T algorithm the covariance matrix is decomposed into the product of a lower triangular matrix L, a diagonal matrix D and an upper triangular matrix L^T which is the transpose of the lower triangular matrix:

$$C = L.D.L^T \tag{4.31}$$

When considering the model order 4 system L is defined the same as that in the LU decomposition (4.23) which has ones along its leading diagonal. The product of $D.L^T$ is equivalent to U (4.24) where D takes the form:

$$D = \begin{bmatrix} d[1] & 0 & 0 & 0 \\ 0 & d[2] & 0 & 0 \\ 0 & 0 & d[3] & 0 \\ 0 & 0 & 0 & d[4] \end{bmatrix} \tag{4.32}$$

Substitution of (4.31) into (2.21) leads to:

$$L.D.L^T.A = B \tag{4.33}$$

This can be separated into different groups of equations which can be easily solved. Firstly Y can be defined as:

$$Y = D.L^T.A \tag{4.34}$$

so that the lower triangular system of equations:

$$L.Y = B \tag{4.35}$$

can be formed and solved by forward elimination again using the systolic array shown in figure 4.2 but with the omission of the dividers in $pe[0]$ as suggested for the LU decomposition example in section 4.5.1.

Another new vector Z can be defined such that:

$$Z = L^T.A \tag{4.36}$$

which allows the solution of the original set of equations to be calculated using back substitution once Z has been determined from:

$$D.Z = Y \tag{4.37}$$

Elements $z[j]$ $(1 \leq j \leq 4)$ of Z can therefore be calculated with division operation:

$$z[j] = \frac{y[j]}{d[j]} \tag{4.38}$$

This could be achieved by using a divider on the input to $pe[0]$ of the back substitution array (figure 4.2(b)). Processor $pe[0]$ can also be modified similarly to the forward elimination array since L^T also has ones along its leading diagonal.

The $L.D.L^T$ decomposition is defined according to the following single assignments:

$$c^{(1)}[j,k] = \begin{cases} 0 & j < k \\ c[j,k] & j \geq k \end{cases} \tag{4.39}$$

$$c^{(n+1)}[j,k] = \begin{cases} 0 & j \leq n \\ 0 & j < k \\ c^{(n)}[j,k] - \frac{(c^{(n)}[j,n])^2}{d[n]} & j = k \\ c^{(n)}[j,k] - c^{(n)}[j,n].l[k,n] & j > k \end{cases} \tag{4.40}$$

$$d[n] = c^{(n)}[n,n] \tag{4.41}$$

$$l[j,n] = \begin{cases} 0 & j < n \\ \frac{c^{(n)}[j,n]}{d[n]} & j \geq n \end{cases} \tag{4.42}$$

4.5.7 LDL^T Decomposition Localised DDG

The DDG for the LDL^T decomposition is detailed in figure 4.15(a). Apart from the interconnections between the nodes on the diagonal border of the graph the structure is the same as the Cholesky decomposition graph of figure 4.3 and the same timing function may be applied. However, the node functions shown in part (b) of the figure differ considerably from the Cholesky example. Those nodes operating in *mode* 0 perform (4.41) to assign values to $d[n]$ whose reciprocal is transmitted in the j direction so that $l[j,k]$ (4.42) can be calculated in the *mode* 1 nodes from a multiplication. The *mode* 2 nodes perform the calculation of $c^{(n+1)}$ (4.40) for $j = k$ and the *mode* 3 nodes cope with the $j > k$ situation. The Cholesky timing function (4.21) is applied and that shown in the figure is offset for $[1,1,1]$ mapping where data must be piped in before commencement.

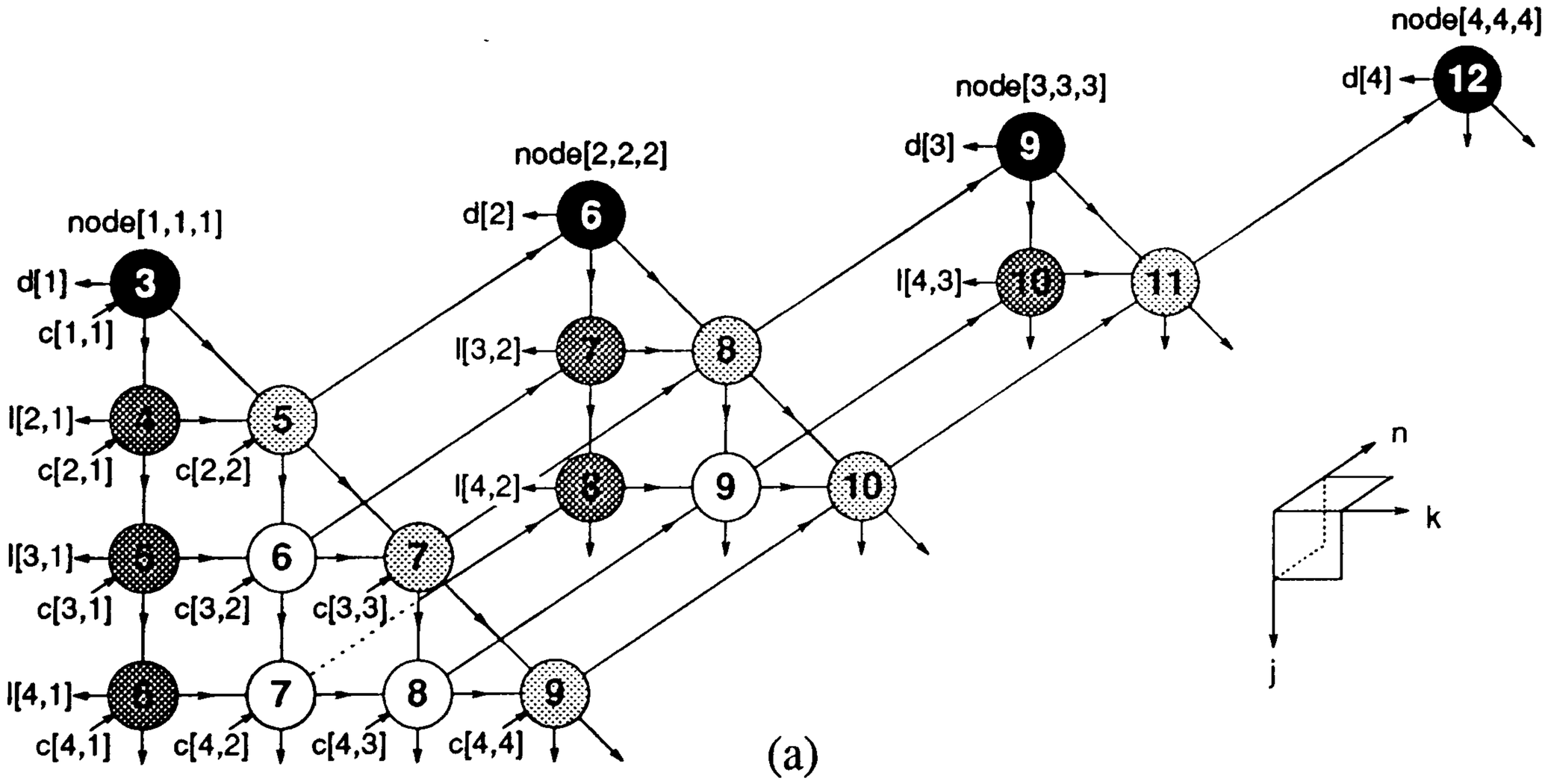


Figure 4.15: (a) DDG for LDL^T decomposition, (b) key to symbols.

4.5.8 LDL^T Decomposition Systolic Array

(A) [1,1,1] Vector Projection

In an effort to avoid the square root bottleneck operation of the hexagonally connected Cholesky Decomposition array Brent and Luk [80] developed a hexagonally connected array for LDL^T decomposition which is topologically equivalent to the array shown in figure 4.16(a). This array is produced by $[1,1,1]$ mapping of the DDG in figure 4.15. The array structure is similar to that for the $[1,1,1]$ Cholesky projection except that along the diagonal border of the array there are extra buses whose localised connections have a delay of two. Study of this array reveals inefficiencies because the multiplications carried out by the type $PE1$ processors to form $l[j,k]$ are repeated in the type $PE2$ processors. This prompts re-representation of the data flow to form a modified LDL^T decomposition DDG.

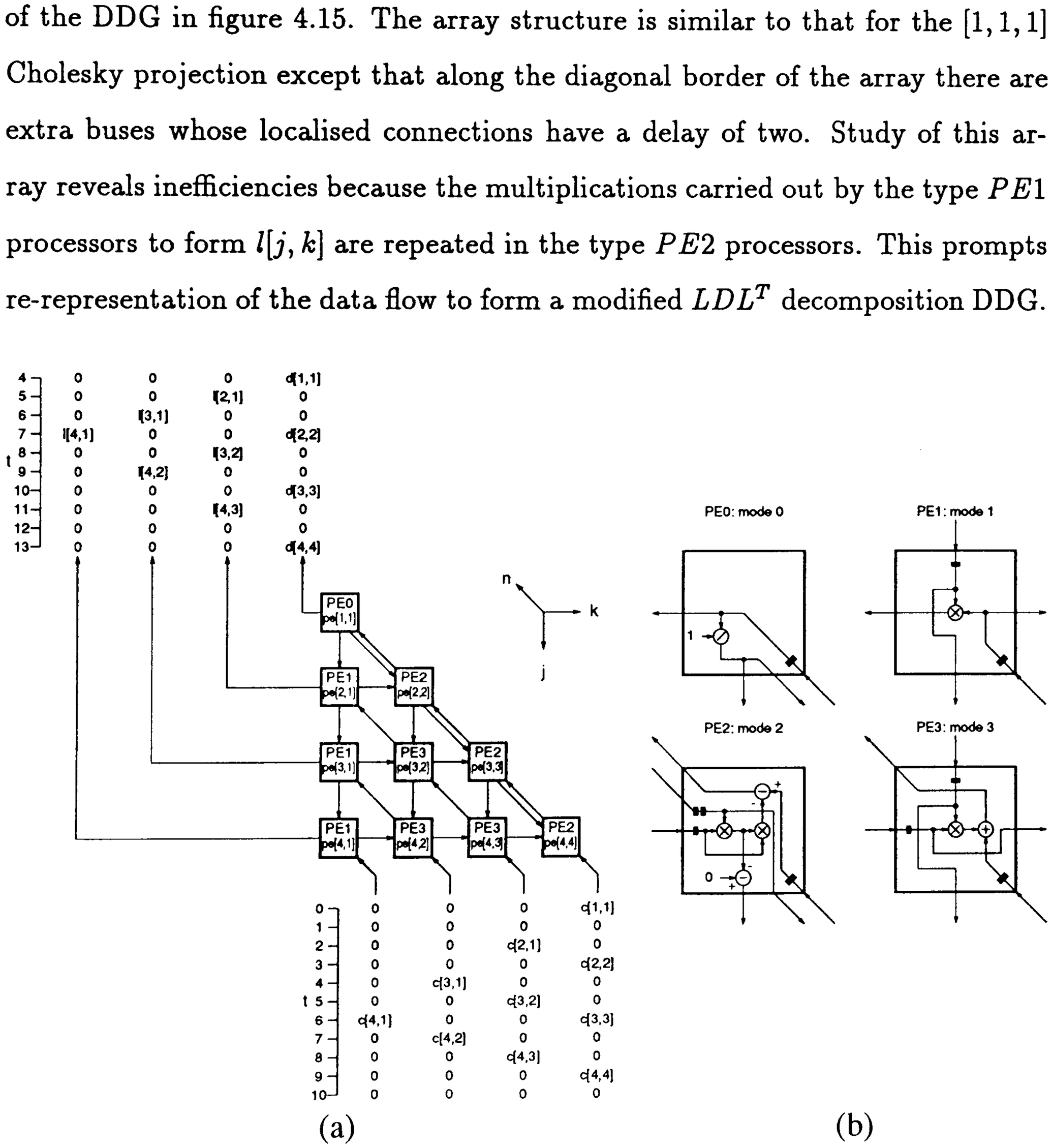


Figure 4.16: (a) Systolic array produced by $[1,1,1]$ projection vector, (b) PEs.

4.5.9 Modified LDL^T Decomposition DDG

The assignment for $c^{(n+1)}[j,k]$ in (4.40) may be expressed in a slightly more concise format as follows:

$$c^{(n+1)}[j,k] = \begin{cases} 0 & j \leq n \\ 0 & j < k \\ c^{(n)}[j,k] - c^{(n)}[j,n].l[k,n] & j \geq k \end{cases} \quad (4.43)$$

This simplifies the $j = k$ condition squaring and division computation to just a multiplication and the modified recurrence (4.43) can be incorporated into a new DDG for LDL^T decomposition illustrated in figure 4.17(a). The diagonal communications for the reciprocal of $d[n]$ are no longer necessary but now extra k direction arcs are needed to broadcast $l[j,n]$.

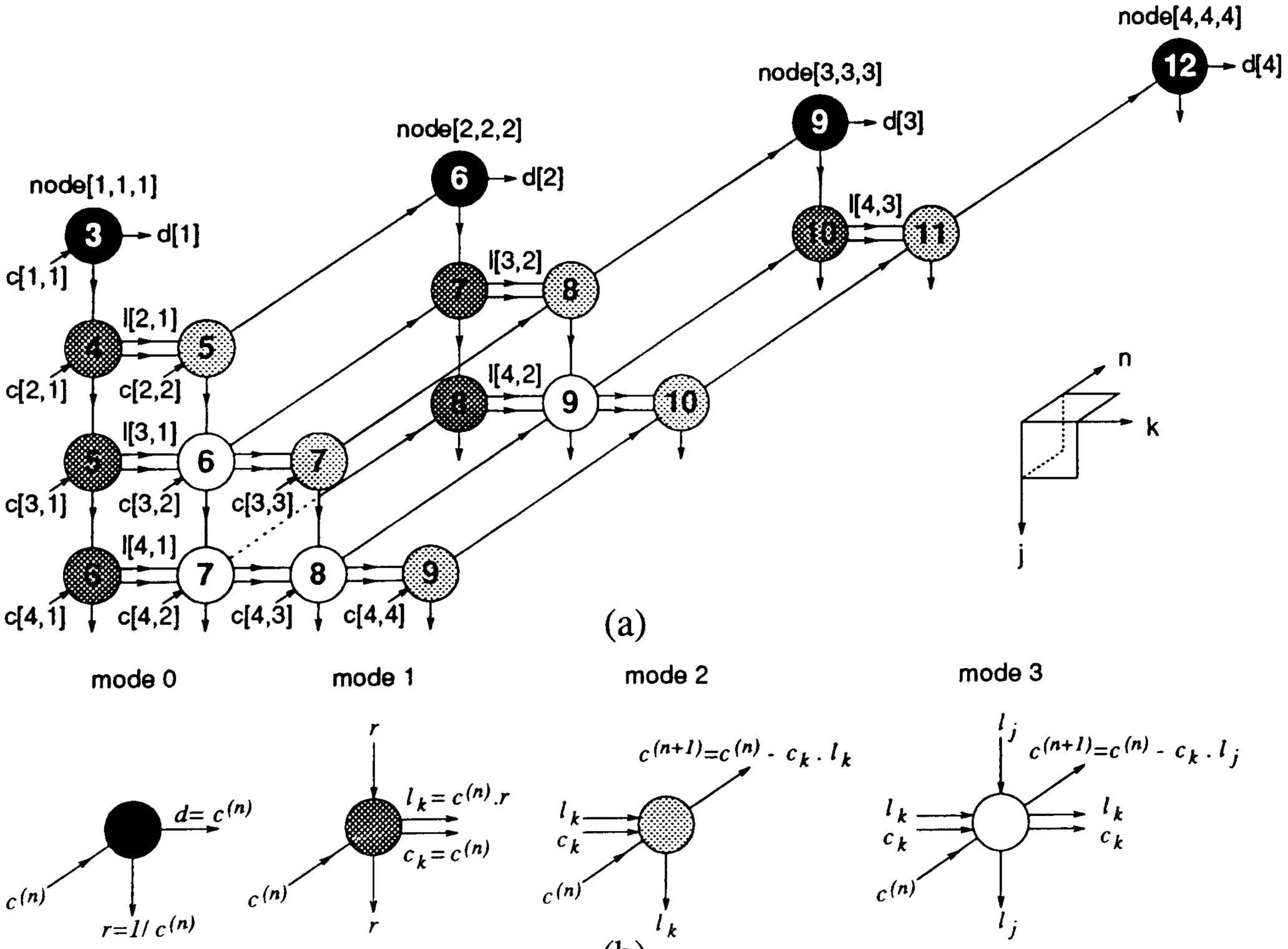


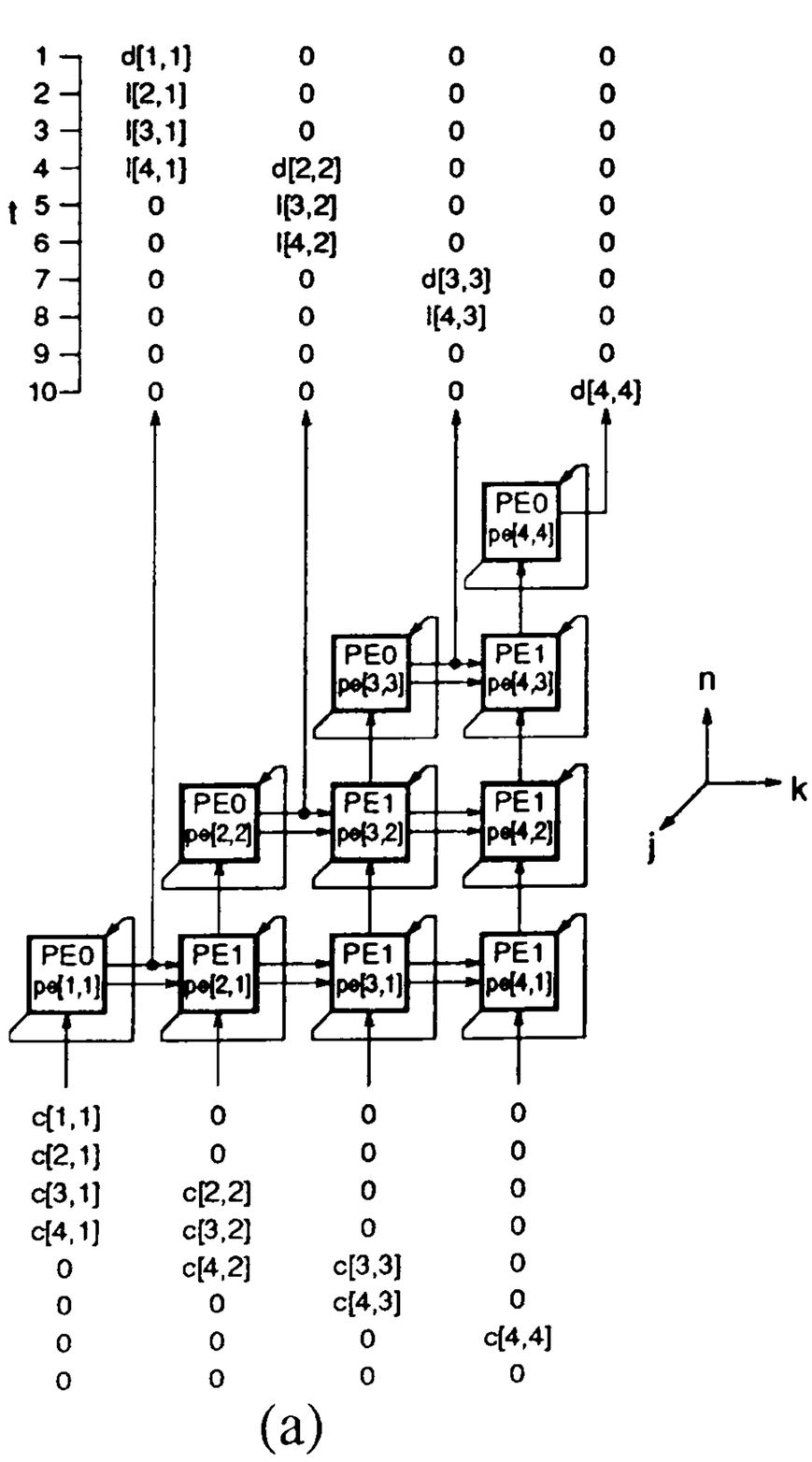
Figure 4.17: (a) Modified DDG for LDL^T decomposition, (b) key to symbols.

4.5.10 Modified LDL^T Decomposition Systolic Arrays

Section 4.5.4 discussed selection of projection directions of the LU decomposition DDG based on the results of the optimal Cholesky decomposition mapping. Once again, due to the similarity of the LDL^T DDG with that of the Cholesky, the $[1,0,0]$ and $[1,1,1]$ mappings are investigated.

(A) [1,0,0] Vector Projection

The j direction mapping results in the orthogonally connected array shown in figure 4.18(a). The array contains two types of PE whose construction is detailed in figure 4.18(b) and the configuration of each in the different modes is shown in section (c) of the diagram. To save hardware the function of the type $PE0$ processors is changed slightly so that the actual value rather than the reciprocal



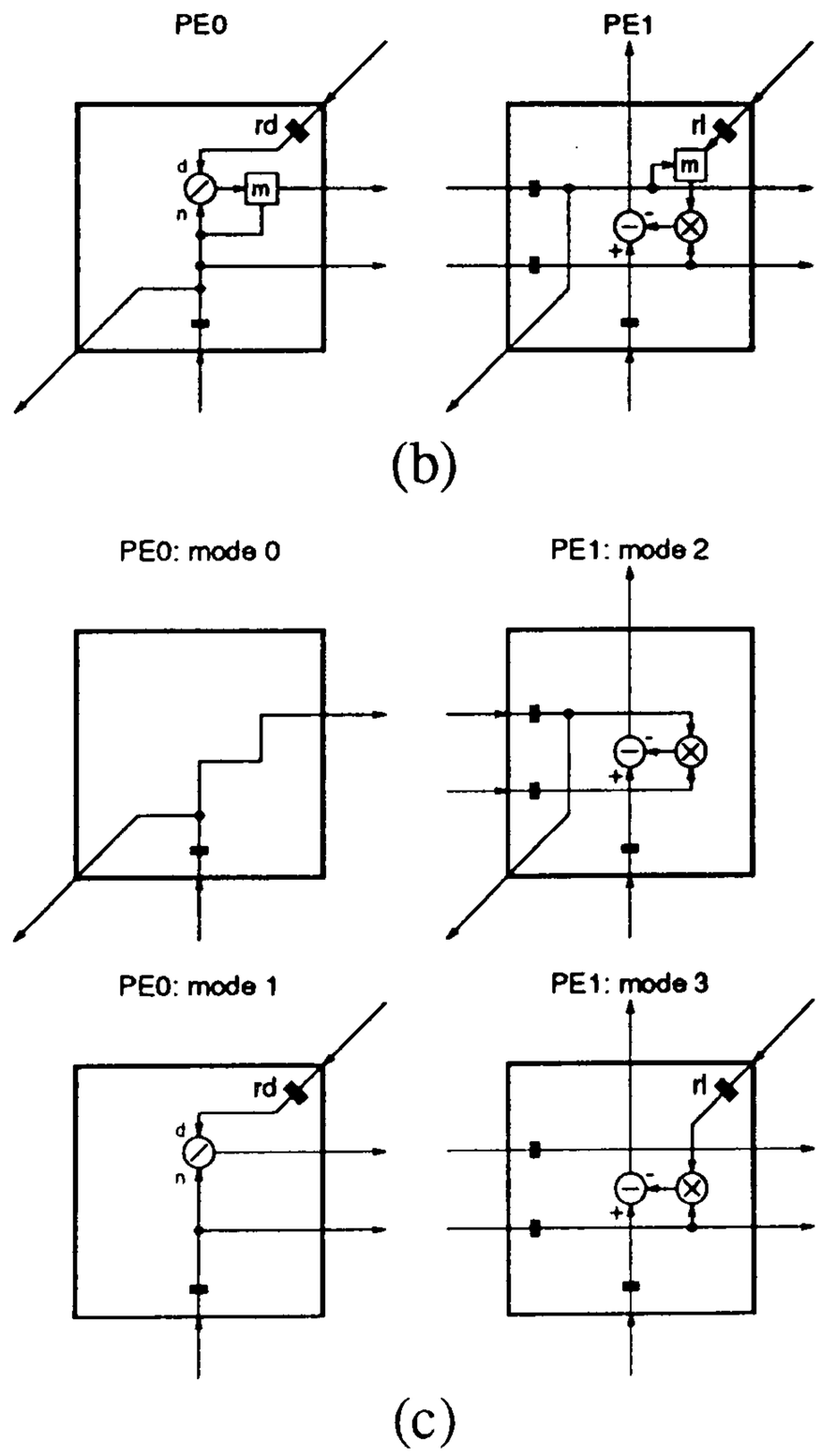


Figure 4.18: (a) Systolic array produced by $[1,0,0]$ projection vector, (b) PEs, (c) guide to PE functions.

of $d[n]$ gets stored in register rd. Elements $l[j,k]$ are subsequently calculated by a division operation as opposed to multiplication by the reciprocal when these PEs are switched to *mode 1*. Therefore rather than having a divider and a multiplier in each of the type $PE0$ processors only a single divider is required.

(B) [1,1,1] Vector Projection

Application of the $[1,1,1]$ projection vector to the modified LDL^T DDG of figure 4.17 results in the systolic array detailed in figure 4.19(a) whose PE functions are shown in part (b). As with the [1,1,1] projections on the other DDGs considered each type of PE operates in a single mode. This design can be directly compared with Brent and Luk's [80] (figure 4.16). Overall the communication burden is greater in the modified design but it does however have fewer diagonal buses. The advantage of the modification can be seen when the type $PE2$ processors are compared as the modified version uses approximately half the hardware.

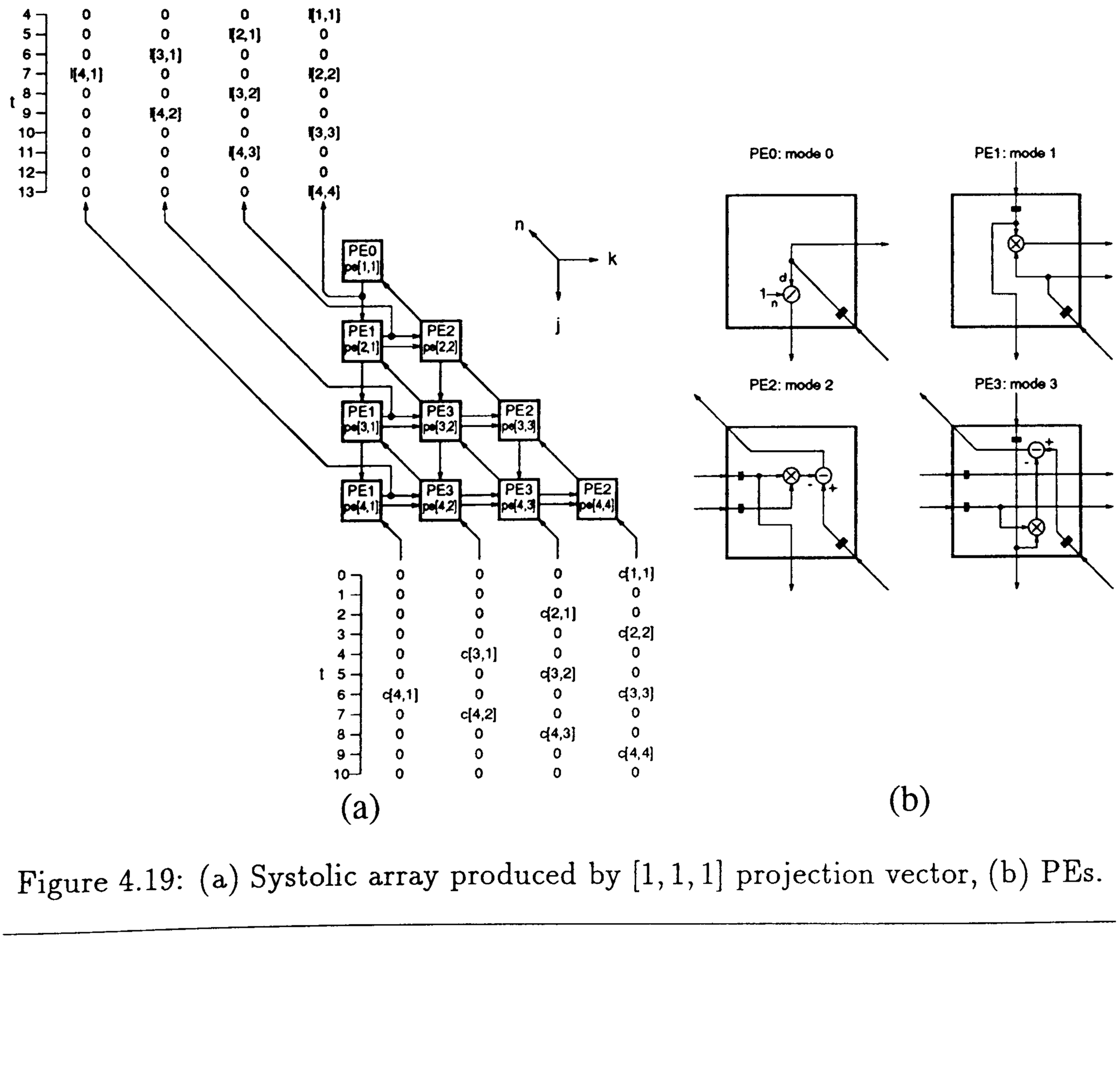


Figure 4.19: (a) Systolic array produced by $[1,1,1]$ projection vector, (b) PEs.

4.5.11 Comparison of LU & LDL^T Decomposition Arrays

The advantages and disadvantages of the five arrays for LU and LDL^T decomposition non-square root methods are compared in the following sections.

4.5.12 Cost Estimation

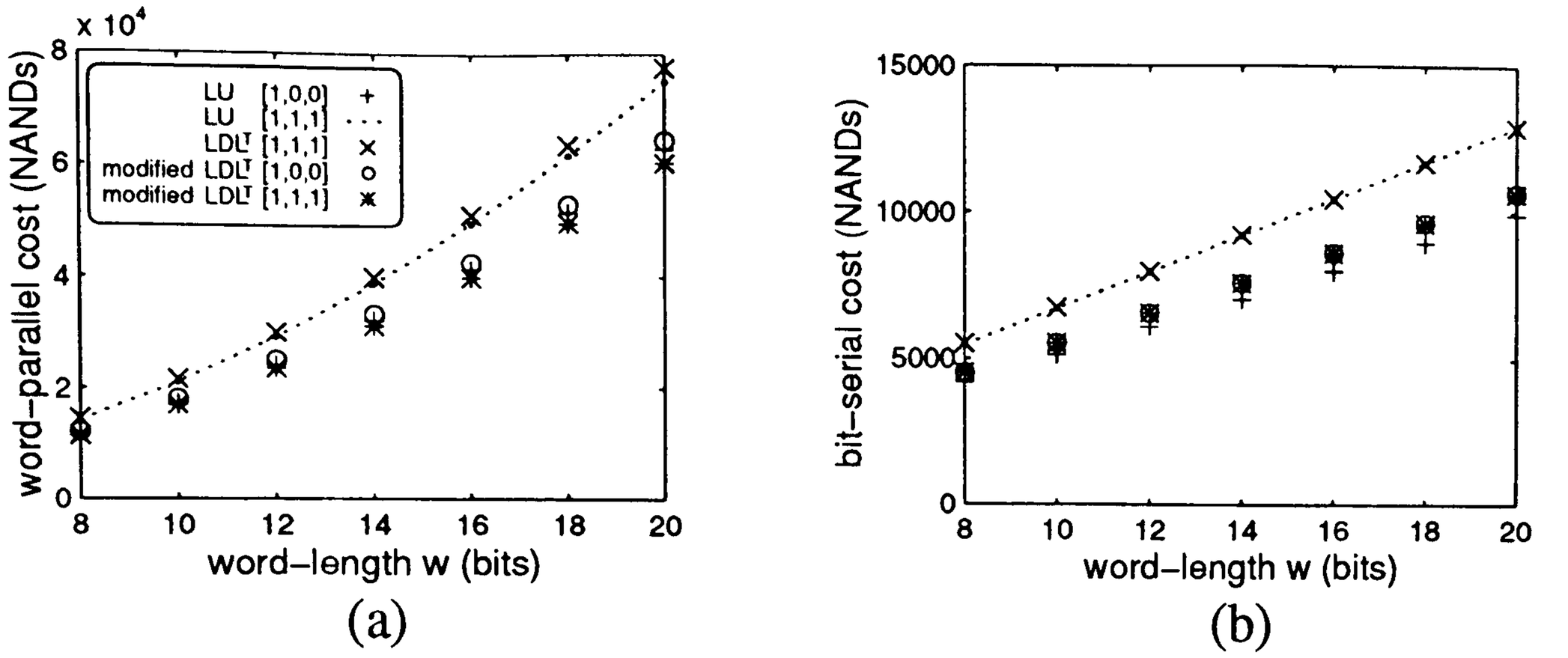
(A) Hardware Cost Estimation

w bit module	number of modules used in the arrays				
	LU		LDL^T	modified LDL^T	
	$[1,0,0]$	$[1,1,1]$	$[1,1,1]$	$[1,0,0]$	$[1,1,1]$
addition	0	0	3	0	0
subtraction	6	9	6	6	6
multiplication	6	9	12	6	9
division	4	3	1	4	1
register	26	37	28	32	28
multiplex	0	0	0	10	0

Table 4.5: Hardware usage for LU and LDL^T systolic arrays in terms of modules.

Table 4.5 details the cost of the non-square-root decomposition systolic arrays in terms of arithmetic, control and register modules. This information, in conjunction with that forwarded in tables 3.1, 3.2 and 4.1, is used to estimate the total NAND gate usage for the non-square-root decomposition arrays and the results are presented in figure 4.20 for both word-parallel and bit-serial approaches.

Brent & Luk's LDL^T $[1,1,1]$ array [80] displays highest hardware cost for both word-parallel and bit-serial approaches. The two new arrays for LDL^T decomposition, derived from $[1,1,1]$ and $[1,0,0]$ projections of a modified DDG (a DDG in which data communication flows are represented in a different way to avoid repeated calculations) both show significant hardware cost reductions over Brent & Luk's design. In the bit-serial approach the cost savings of the modified LDL^T arrays are as much as 17% of Brent & Luk's array, and in the word-parallel ap-



(a) (b)

Figure 4.20: Cost for (a) word parallel and (b) bit-serial approaches.

proach the $[1,1,1]$ modified LDL^T array presents the lowest cost of all 5 of the arrays.

In contrast to the results seen for the Cholesky decomposition array the $[1,1,1]$ LU decomposition array presents a much higher cost than the LU array produced by $[1,0,0]$ projection. This is due to the involvement of the full matrix in the LU decomposition calculation, (rather than the symmetrical matrix in the $[1,0,0]$ array) which causes array sizes, in terms of number of PEs, to be large when n features as a component of the projection vector.

The bit-serial approach is judged to provide sufficient throughput for decomposition in the model order $p = 4$ Modified Covariance method (section 4.2). Therefore the results shown in figure 4.20(b) are applicable when choosing the array with the lowest hardware cost leading to selection of the LU $[1,0,0]$ mapping.

(B) Data Communication Cost

Table 4.6, which summarises the communication costs incurred in each of the non-square-root decomposition arrays, shows that the LU $[1,0,0]$ mapping requires minimum PE interconnection whilst the LU $[1,1,1]$ array needs the maximum. The disadvantage of the $[1,1,1]$ modified LDL^T array over Brent & Luk's version

can be seen in that it requires a slightly greater PE interconnection resource and it presents over twice the interconnection burden of the $[1,1,1]$ LU array. The modified LDL^T $[1,0,0]$ array performs better than the modified LDL^T $[1,1,1]$ in this respect but still presents higher PE interconnection cost than the LU $[1,0,0]$ array. Both of the $[1,0,0]$ LU and LDL^T arrays however require the localised feedback costs but as discussed in section 4.4.8(F) these are likely to present less of a burden relative to PE interconnection. An advantage of the LDL^T arrays is that they have a lower input/output burden compared to the LU arrays. However, even though the LDL^T arrays only require 4 output buses as opposed to 7 in the LU decomposition, there is still need for connection to both forward elimination and back substitution systolic arrays, so that the input burden to the next two stages of the estimator is equal for LU and LDL^T.

communication type	number of data buses				
	LU		LDL^T	modified LDL^T	
	$[1,0,0]$	$[1,1,1]$	$[1,1,1]$	$[1,0,0]$	$[1,1,1]$
PE interconnection	12	$21+9\sqrt{2}$	$12+9\sqrt{2}$	18	$18+6\sqrt{2}$
localised feedback	10	0	0	10	0
array input	4	7	4	4	4
array output	7	7	4	4	4

Table 4.6: Data communication cost.

(C) Control Signal Cost

Both the LU and LDL^T $[1,0,0]$ mappings require extra control signals for PE mode switching. These signals are in the form of clock enables in the LU case and as well as these enables the LDL^T array needs multiplexor control. These control signals can of course though be pipelined using a similar methodology to that shown for Cholesky decomposition $[1,0,0]$ array figure 4.4.

4.5.13 Efficiency

The processor α and block β pipelining periods shown in table 4.7 reflect application of the interleaving method discussed in section 4.4.9 for the $[1,1,1]$ mappings. In terms of α and β all arrays share the same efficiency, however the block latency for both $[1,0,0]$ arrays is slightly less as no array initialisation is required.

timing factor	number of clock cycles				
	LU		LDL^T	modified LDL^T	
	$[1,0,0]$	$[1,1,1]$	$[1,1,1]$	$[1,0,0]$	$[1,1,1]$
N_{clk}	10	13	13	10	13
α	1	1	1	1	1
β	4	4	4	4	4

Table 4.7: Processor/block pipelining periods and block latency.

4.5.14 Selection of the Optimal LU/LDL^T Decomposition Array

The choice of the optimal systolic array from the two LU mappings and the three arrays for the LDL^T decomposition is not quite as clear cut as that for the Cholesky decomposition, but basically falls between the $[1,0,0]$ LU array and the $[1,1,1]$ modified LDL^T array. The latter displays slightly lower overall hardware cost in the word-parallel approach but in the bit-serial approach which is more relevant to the $p = 4$ Modified Covariance spectral estimation problem the $[1,0,0]$ LU array displays the lowest cost. The $[1,0,0]$ LU array also requires less than 50% of the processor interconnection links needed in the $[1,1,1]$ modified LDL^T. Both arrays can be set up to produce equivalent processor/block pipelining but interleaving of covariance matrix data from consecutive windows is necessary in the LDL^T array which also presents a lag of 3 extra clock pulses. The LU array does require control signals to provide internal register clock enabling and this is not necessary for the other array. The LDL^T array is mapped onto a hexagonal type array which can be compared to the orthogonal LU architecture. Despite its higher control burden the LU decomposition array is chosen in preference to the LDL^T array due to its lower cost, simpler orthogonal architecture,

which has a greatly reduced PE interconnection burden, and also because of the uncomplicated nature of its internal PE circuitry.

4.6 Concluding Remarks

This chapter has dealt with hardware implementation for real-time calculation of filter parameters by solution of a system of linear equations in the Modified Covariance spectral estimator with fixed model order $p = 4$. Decomposition techniques were considered in order to transform the covariance equations into triangular form enabling efficient solution by forward elimination and back substitution. The DDGs for Cholesky, LU and LDL^T decomposition were presented, and clearly showed the relationships with the initial algorithms.

Different projections of the Cholesky decomposition DDG resulted in five systolic array designs. Projection by the $[1,1,1]$ vector resulted in an array that was hexagonally connected and topologically equivalent to an array described in a prior article by Brent & Luk [80] and the other projections considered produced a set of newly proposed orthogonally connected systolic arrays. In a comparison of the arrays the hexagonal array was selected as optimal due to its efficient use of hardware despite its disadvantage in PE interconnection and computational time lag. It was also shown how the maximum systolic clock frequency of Brent & Luk's $[1,1,1]$ array could be improved by a factor of two by some simple changes to the assignments of certain nodes.

Arrays for LU and LDL^T decomposition were considered as an alternative to the Cholesky method, as these designs eliminate the square-root operation with its associated control and irregularity problems. Analysis of Kung's $[1,0,0]$ LU array [23] showed that data was left stored in PEs at the end of calculation. A new array, which allowed on-the-fly retrieval of these matrix elements without increasing the block pipeline period, was formed by $[1,0,0]$ projection of a modified DDG in

which the data flow of these words was re-represented. Inefficiencies of Brent and Luk's LDL^T array [80] led to re-representation of the data flow in the DDG for this method and two new arrays which showed significant hardware reduction were produced by projecting the modified graph in the $[1,0,0]$ and $[1,1,1]$ directions. The $[1,0,0]$ LU array was however the preferred choice due its lower bit-serial implementation cost, the much reduced interconnection burden of its orthogonal architecture and the simplicity of its internal PE construction.

This chapter has presented a method for selection of optimal square-root and non-square-root decomposition arrays, on a basis of hardware cost, communication burden, control complexity and efficiency. One question which however has not been addressed is which design is most suitable for connection to the systolic arrays in the other parts of the estimator, that is the PMA matrix element calculation array and the triangular system solvers. Chapter 5 compares the cost of integrating both of the selected decomposition array designs into the system, incorporating this cost into a function which also takes into account the relative effects of communication. The next chapter also presents an error analysis which is required in order to select a suitable word-length. Too short a word length can lead to lack of precision and subsequently large error in the output data. Too long a word-length, while giving good result accuracy can lead to unnecessarily high hardware cost. Rounding error in the input data and during the computations within the Cholesky $[1,1,1]$ and LU $[1,0,0]$ arrays therefore needs to be carefully studied as to its effect on the accuracy of estimated parameters. The performances of the $[1,1,1]$ Cholesky and $[1,0,0]$ LU decomposition arrays can then be compared using a cost/benefit analysis [30]. Over a range of word-length the new cost function is weighted against benefit which is treated as the inverse of estimation error. This then enables selection of an optimal decomposition systolic array and word-length, in turn determining the word-length requirements of the other systolic arrays used in the estimator.

Chapter 5

Cost/Benefit Analysis

5.1 Introduction

The two previous chapters presented systolic array solutions for the first two sections of the Modified Covariance spectral estimator. Chapter 3 considers designs for the computation of the matrix elements resulting in selection of a systolic array formed by mapping accumulation and multiplication DDGs separately. Chapter 4 examines the next section of the estimator in which two systolic arrays, one for Cholesky and one for LU decomposition which avoids square root calculation at the cost of higher computational burden, are selected. This chapter considers the combination of these systolic arrays to form an optimal system.

A cost/benefit criterion is developed to aid in the selection of the best decomposition method with its most appropriate precision. The cost estimates developed in the previous chapter are weighted against each other and the cost of integrating the decomposition array into the estimator for use in the analysis. For the benefit calculation the errors which occur due to rounding in finite word-length arithmetic are estimated using hardware description language simulations. The results of the cost/benefit can then be used to select the most appropriate word-length for quantised Doppler signal input to the PMA array which is used to calculate the covariance matrix elements to the required accuracy.

5.2 Simulation

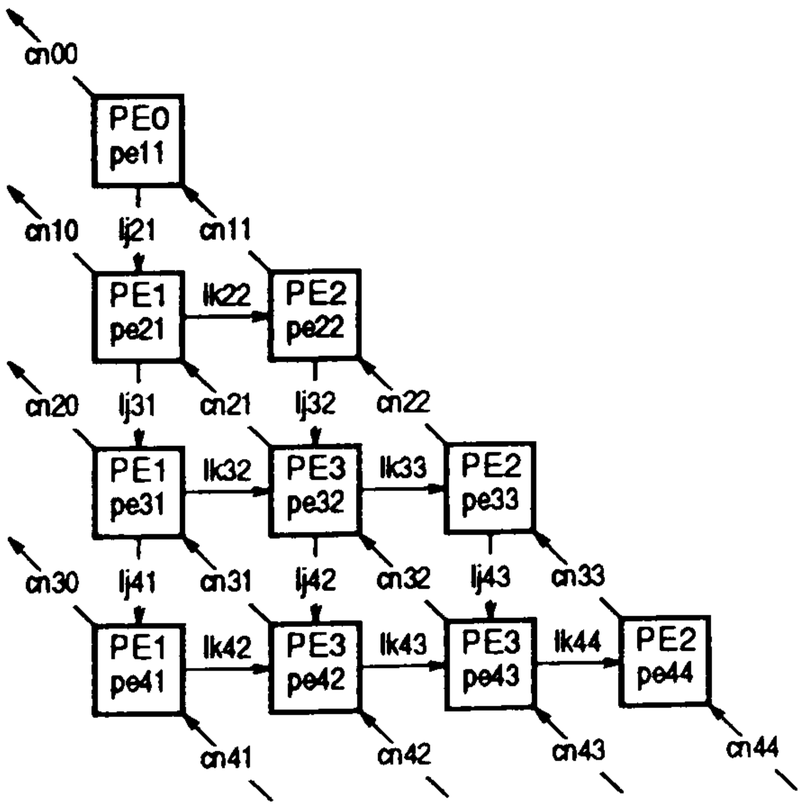
The systolic arrays presented in the previous chapters can be modelled using Hardware Description Language (HDL). The modelling serves to check the operation of the arrays, making sure that the designs are functionally correct. The models are also used to analyse the errors which occur due to finite word-length rounding in the arithmetic.

Verilog HDL [125] models containing both behavioural and structural constructs can be used to check the PE operation, architectural communication and timing of data flow in the systolic arrays. A structural HDL description of the processor interconnections in the systolic array can be written by defining module port inputs and outputs on instances of the processors. Since the circuitry behind the registers, arithmetic units and multiplexors is not of interest at this point these units can be looked upon as the lowest level modules of the hierarchy and should be described using behavioural constructs. Describing processor operation using these behavioural modules makes simulation much simpler because the arithmetic operations can be implemented using $+$, $-$, / and $*$ rather than considering detailed structural descriptions of full adder based arithmetic arrays.

When verifying the systolic array operation it is desirable to work with real numbers since when using finite word-length arithmetic rounding errors in the result can make it difficult to determine whether or not it is exactly correct. There are two function calls in the Verilog software, called *$realtobits* and *$bitstoreal*, which allow real data to be passed across ports by conversion into 64 bit net format. Usage of these functions to communicate between processor modules implies that a word-parallel approach is most appropriate for verification purposes so that all of the PEs are interconnected by 64 bit nets. The processors are also defined by structural 64 bit net interconnection of their behavioural base modules. The large bus widths are tolerated for the purpose of initial array verification as these

widths can be reduced at a later stage of the simulation when the numerical format for data representation, structures of the arithmetic units and required precision are considered.

A bottom-up approach is proposed which builds up the hierarchical design starting with behaviourally based components. Referring to the $[1,1,1]$ mapping of the Cholesky decomposition array (figure 4.8), the base level components of the processors, that is the register, multiplication, division, subtraction and square root units are described as behavioural modules. The *$bitstoreal* and *$realtobits* functions are used at this level to exploit the behavioural operator set. However a function based on a recursive approximation [126] has to be written for square root as this is not included in the basic set of operators. The four PEs of the array are then described structurally by connecting together instances of the base level modules. The systolic array can then be given a structural definition as shown in figure 5.1.



(a)

```
//module definition of Cholesky decomposition array
module  CHOLESKY_SA (cn30 , cn20, cn10, cn00, cn41, cn42, cn43, cn44, clk , rst);

//input ports
input     cn41, cn42, cn43, cn44, clk , rst;

//output ports
output    cn30, cn20, cn10, cn00;

//declaration of bus widths
wire['bw:1]    cn00, cn10, cn20, cn30, cn11, cn21, cn31, cn41, cn22,
               cn32, cn42, cn33, cn43, cn44, lj21  , lj41  , lj32  , lj42  ,
               lj43  , lk22 , lk32 , lk33 , lk42 , lk43 , lk44 ;

//instances of processors used in the Cholesky decomposition array
CHOLESKY_PE0   pe11 ( lj21 , cn00, cn11, clk , rst );

CHOLESKY_PE1   pe21 ( lj31 , lk22 , cn10, lj21  , cn21, clk , rst ),
               pe31 ( lj41 , lk32 , cn20, lj31  , cn31, clk , rst ),
               pe41 (      , lk42 , cn30, lj41  , cn41, clk , rst );

CHOLESKY_PE2   pe22 ( lj32 , cn11, lk22 , cn22, clk , rst ),
               pe33 ( lj43 , cn22, lk33 , cn33, clk , rst ),
               pe44 (      , cn33, lk44 , cn44, clk , rst );

CHOLESKY_PE3   pe32 ( lj42 , lk33 , cn21, lj32  , lk32 , cn32, clk , rst ),
               pe42 (      , lk43 , cn31, lj42  , lk42 , cn42, clk , rst ),
               pe43 (      , lk44 , cn32, lj43  , lk43 , cn43, clk , rst );

endmodule
```

(b)

Figure 5.1: (a) Systolic array with PE and bus labelling, (b) Verilog description of the Cholesky decomposition $[1,1,1]$ mapping.

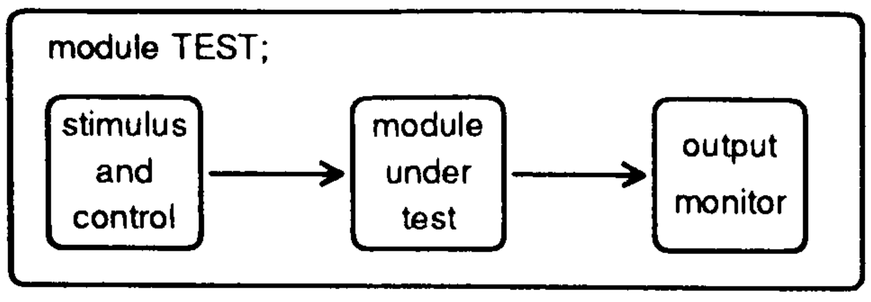


Figure 5.2: Simulation environment for general a test module.

Testing is emphasised at at each level of the hierarchy. There are three main sections to a general test module as shown in figure 5.2. Stimulus and control provides the data input and control waveforms which can be behaviourally modelled in Verilog. The control waveforms include clock, reset and multiplexor select signals. The stimulus and control waveforms are applied to an instance of the module under test whose response is viewed from the last block which monitors the output data in a convenient real format.

Individual test environments can be set up for each of the behavioural arithmetic modules and for the four different types of PE used in the Cholesky decomposition array. A program written in Matlab [127], which generates a set of data typical to that output from a pulsed Doppler ultrasound transducer when measuring blood flow, is used to provide test data. Another Matlab program, which performs the Modified Covariance method, produces real covariance matrices for presentation to the Verilog stimulus module before conversion into 64 bit net format for input into an instance of the systolic array. The Matlab software has built in decomposition functions for cross checking with the results from the Verilog simulation.

The tri-linear (figure 3.11) and bi-linear (figure 3.13) matrix element calculation, $[1,1,1]$ mapping of the Cholesky decomposition array (figure 4.8), $[1,0,0]$ mapping of the LU decomposition (figure 4.13), $[1,1,1]$ mapping of the modified LDL^T decomposition (figure 4.19), forward elimination (figure 4.2(a)) and back substitution (figure 4.2(b)) systolic arrays can similarly be simulated using HDL.

5.2.1 Bit Level Simulation

Once the systolic array architectures are proven to be functionally correct, the simulation can be taken a stage further where the format of the data in the real system is considered. As stated in chapter 3 a fixed point scheme is to be pursued rather than floating point due to the cost saving when a large number of PEs are considered. The problem with using a fixed point scheme for simulation is that the dynamic range is a severe limitation when considering applications such as decomposition. For example a 10 bit, two's complement, fixed point scheme allows a range from -512 to 511. If this data format is used to represent the full range of the input covariance matrix elements in the Cholesky decomposition example then when element $l[1,1]$ is calculated its maximum value is $\sqrt{511} \approx 23$. This result could be represented in the same two's complement format by just 6 bits. All the other values of $l[j,k]$ output on the same bus of the systolic array can only have smaller magnitude than $l[1,1]$ and therefore 4 bits of this data bus are redundant. Resolution could be improved considerably if these 4 bits are utilised. For this reason scale factors are introduced to the systolic array so that for the example given, $l[1,1]$ would be represented by 10 bits in the range from -32 to 31.9375 which is obtained by scaling the 10 bits by a factor of 2^{-4}.

(A) Calculation Of Scale Factors for Decomposition Arrays

To aid in calculation of the scale factors an ensemble of the simulated stationary pulsed Doppler signals can be generated using the Matlab programs. Previous analysis [45] of Gaussian spectral profiles which are derived from the signal simulation sets postulates that accurate statistical conclusions can only be drawn from estimated spectra constituted from a sufficient number of frequency bins. Eleven cases are chosen from spectra containing a minimum of 9 bins. The characteristics of these signal simulation sets are shown in table 5.1, the table gives

the mean frequency f_m, RMS bandwidth f_b and sampling frequency f_s of the chosen simulations.

f_m (kHz)	f_b (kHz)	f_s (kHz)
1.0	100	6.4
	200	
2.0	100	12.8
	200	
	400	
4.0	200	25.6
	400	
	800	
8.0	400	51.2
	800	
	1600	

Table 5.1: Characteristics of the pulsed Doppler ultrasound signals.

Each of the sets consist of 51200 data samples, normalised to give peak values of $\pm$511mV, 4 times the RMS signal level and equivalent to a 10 bit range when quantised in 1mV intervals. This data is segmented into long windows which contain 512 samples as this allows better estimates to be made when dealing with stationary signals. This is then used as input data to the Matlab Modified Covariance estimator which in turn produces a set of covariance matrices for input to the Verilog HDL behavioural/structural descriptions of the decomposition methods. Monitoring and storage of the values transmitted along each of the data buses allows the distributions data words produced within the arrays to be studied with the high accuracy of the floating point arithmetic used by the behavioural operator set. The scale factors can then be deduced to cover with the ranges of values appearing on the buses. An individual scale factor could be associated with each bus in the array but here the approach followed assigns the same scale factor to buses carrying similar values (e.g. all the covariance matrix input buses). This may lead to some bits being redundant but design is made simpler if there are fewer scale factors.

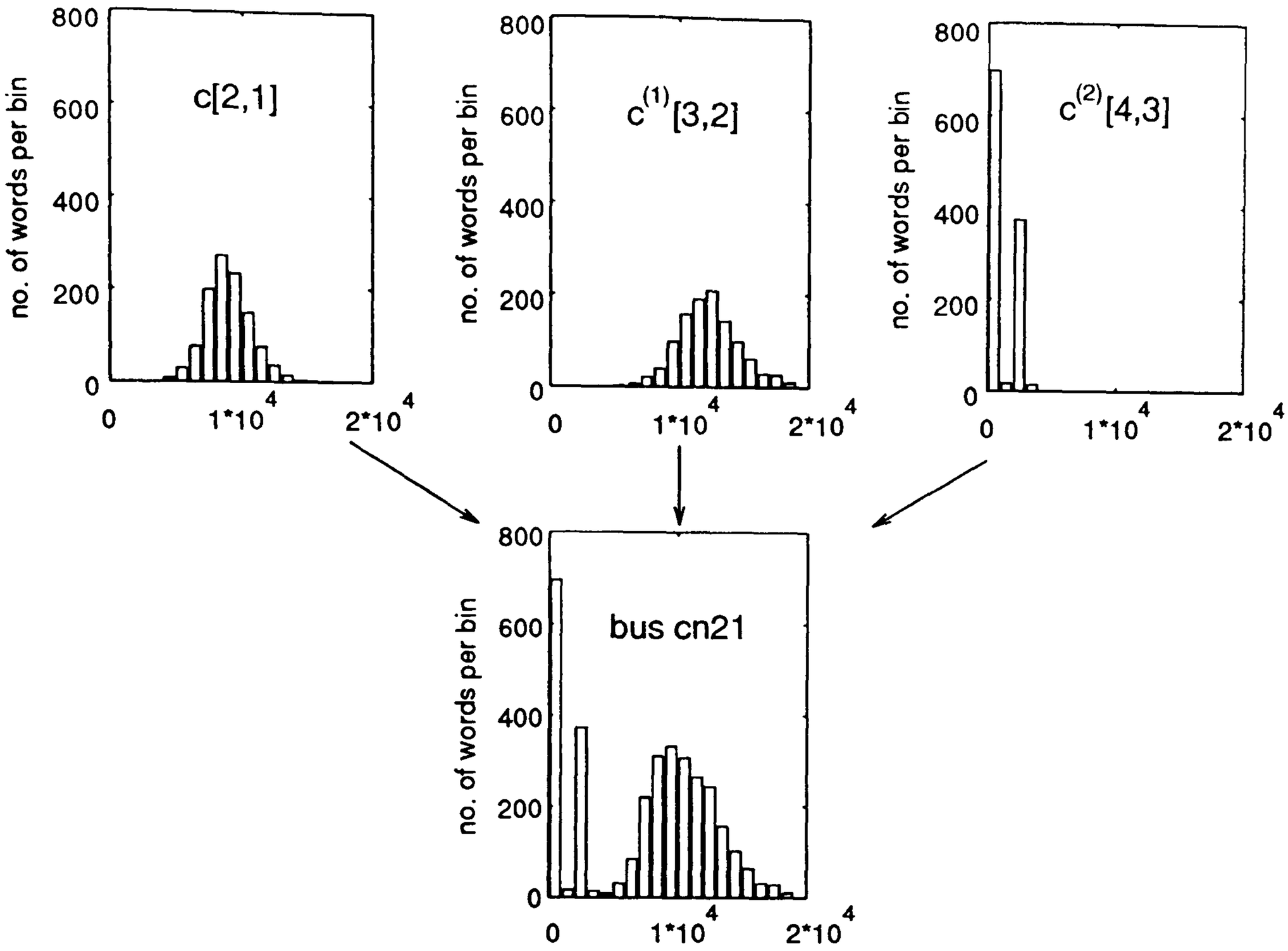
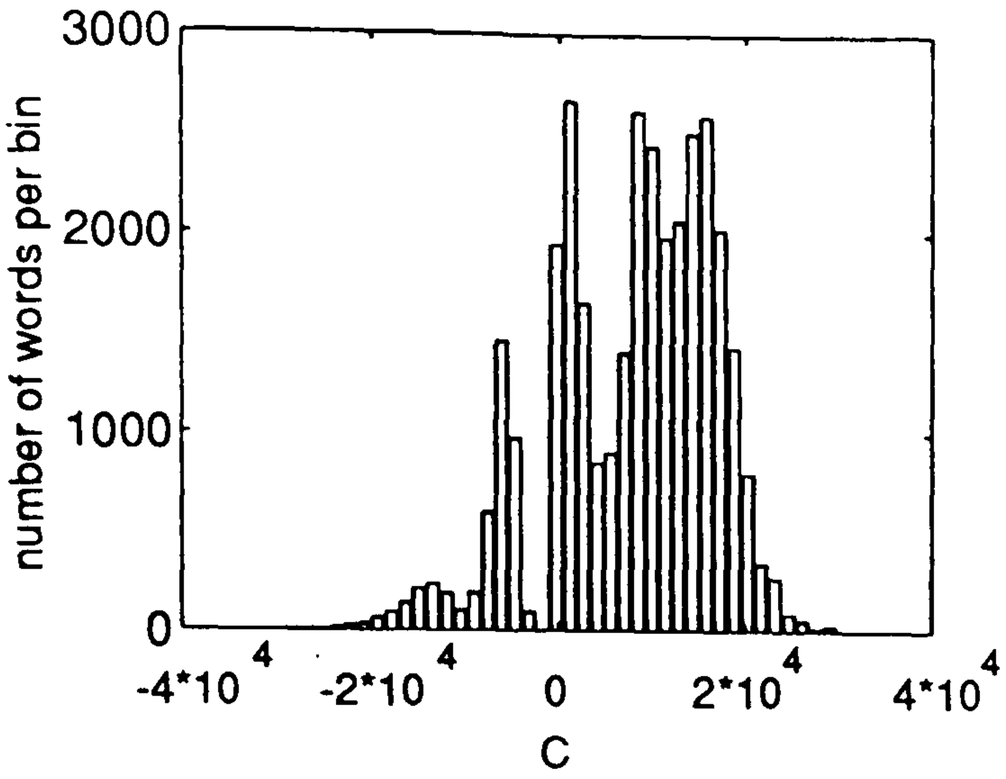


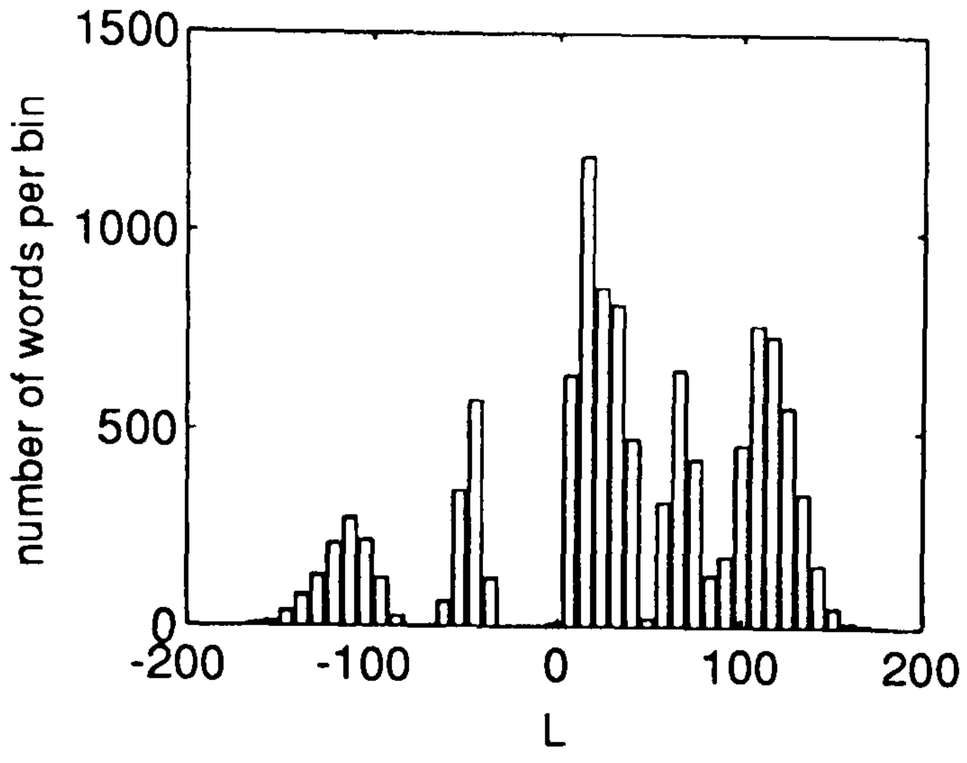
Figure 5.3: Combining the distributions of values on data bus $cn21$ of the Cholesky decomposition systolic array shown in figure 5.1.

For example when the calculation of the filter parameters using the Cholesky decomposition, forward elimination and back substitution processes is considered, four different scale factors can be applied to various data nets in the arrays. These are associated with the iterations of the input covariance matrices C, the output lower triangular matrix L, the vector output of the forward elimination Y and the filter parameters A.

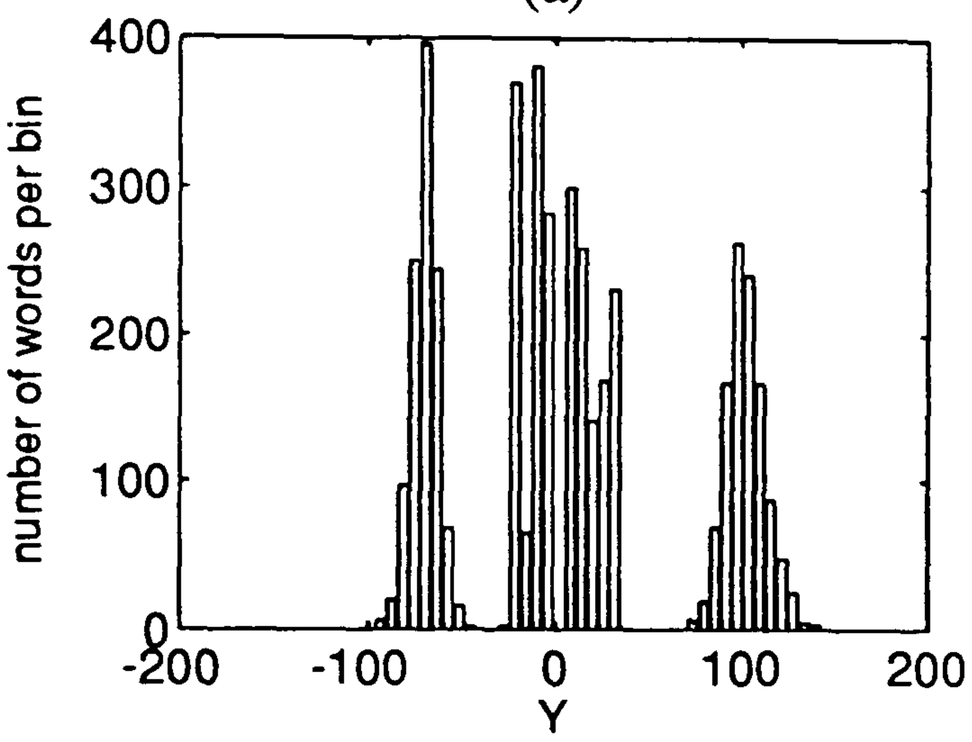
Figure 5.3 shows histograms of the $c[2,1]$, $c^{(1)}[3,2]$ and $c^{(2)}[4,3]$ data words for all the 1100 covariance matrix inputs. These data words all appear on the $cn21$ data bus which is indicated in figure 5.1 and the figure also shows how the three histograms are combined to produce a histogram for the distribution of values on the $cn21$ data bus.



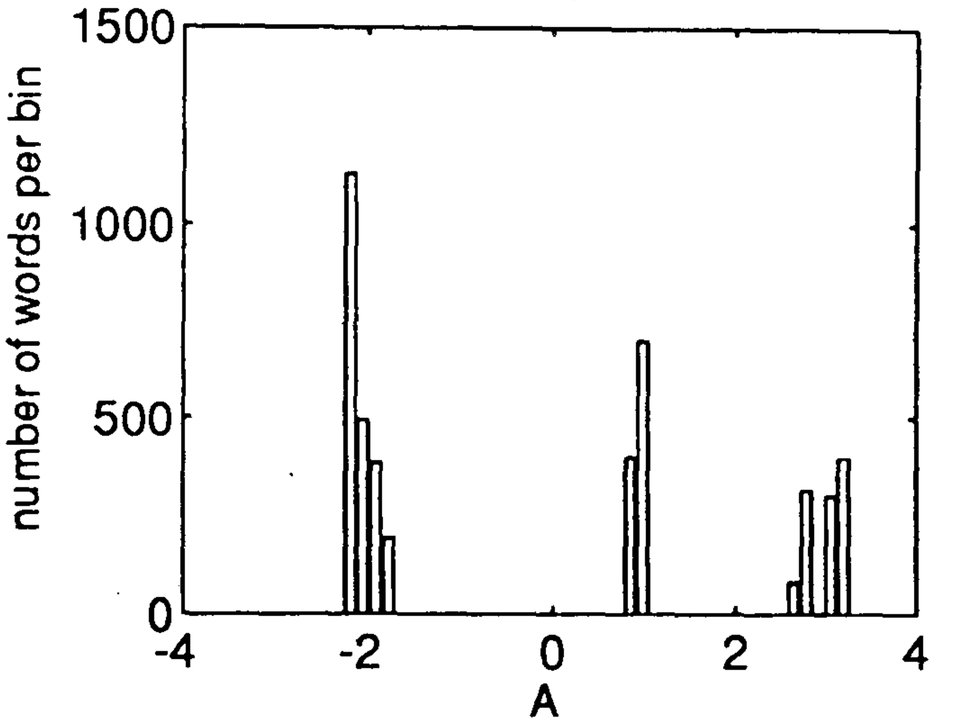
(a)



(b)



(c)



(d)

Figure 5.4: Histograms showing the distributions of (a) covariance matrix and its iterations - C, (b) lower triangular matrix - L, (c) forward elimination output vector - Y and (d) the filter parameter values - A of the Cholesky decomposition systolic array shown in figure 5.1.

Combining the distributions for all of the cn data buses, excluding those which carry the output data, produces the histogram shown in figure 5.4(a). The distributions for the Cholesky decomposition array output L, forward elimination output Y and back substitution output A are also shown in parts (b), (c) and (d) respectively of the figure. The histograms shown take into account all one hundred data window inputs to the estimator. No appreciable decrease to the width of these distributions is observed when only fifty samples from each set are used and therefore it is assumed that the sample set taken adequately describes the general ranges. So there is a high probability for any new random input generated to produce values within the ranges indicated in figure 5.4.

	type of matrix			
	C	L	Y	A
maximum	2.9776e+04	172.3544	142.6815	3.2646
minimum	-2.8787e+04	-167.3384	-96.4427	-2.2620
scale factor	2^{16-w}	2^{9-w}	2^{9-w}	2^{3-w}

Table 5.2: Data ranges and scale factors for the Cholesky decomposition array.

Scale factors are deduced such that the observed data range can be optimally encompassed by a word-length w. The maximum and minimum ranges of C, L, Y and A produced by the simulations are noted in table 5.2. The input covariance matrix elements range in value from approximately ± 30000 whilst the output L matrix elements range only from ± 170. Usage of 2's complement 10 bit words with a scale factor of 2^6 gives the range $(-512 \rightarrow 511)2^6$ or $-32768 \rightarrow 32704$, adequate for the representation of C. If standard fixed point is used then this scaling factor would be used throughout the rest of the process. $2^6 = 64$ is however comparable in magnitude to the elements L and would cause large relative error if used in its representation. Therefore it is far better to introduce a new scale factor for L of 2^{-1} which gives a range of $-256 \rightarrow 255.5$. The maximum absolute error in L is reduced to 0.5 as opposed to 64 for the standard fixed point case and the full ranges are still accommodated. The same method may also be applied to the systolic array produced by the $[1, 0, 0]$ projection of the LU decomposition problem resulting in the scale factors shown in table 5.3. Note that the scale factors for L and Y differ to those for the Cholesky decomposition.

	type of matrix				
	C	L	U	Y	A
maximum	2.9776e+04	1.7052	2.9706e+04	2.0432e+04	3.2646
minimum	-2.8787e+04	-0.9840	-2.8787e+04	-1.6622e+04	-2.2620
scale factor	2^{16-w}	2^{1-w}	2^{16-w}	2^{16-w}	2^{3-w}

Table 5.3: Data ranges and scale factors for the LU decomposition array.

(B) Affect of Scale Factors on Arithmetic operations

The scale factors directly affect the choice of bits from double precision products. The analysis here assumes that single precision products are used throughout the arrays otherwise word-lengths could grow quite large. The type $PE2$ and $PE3$ processors of the Cholesky decomposition array are both required to form the product of two elements of matrix L to form a covariance matrix element iterative update $c^{(n)}[j,k]$. Using the word-length of 10 bits for the purpose of illustration then with reference to table 5.2 and the scale factor used for L is 2^{-1}. The scale factor associated with the double precision product is calculated by the product of the scale factors on the multiplier and multiplicand and is therefore 2^{-2}. The least significant bit (LSB) p_o of this double precision product therefore has weight 2^{-2}. In order to keep ten bit buses through the array the appropriate 10 bits must be selected from the 20 bit product. When the single precision word is selected from these 20 bits its LSB must have the weight of 2^6, which is the scale factor associated with the covariance matrix C (table 5.2). Therefore bits $p_{17} \rightarrow p_8$ should be selected since p_8 has a weight of $2^8.2^{-2} = 2^6$. Figure 5.5 reviews the bit selection from a double precision product.

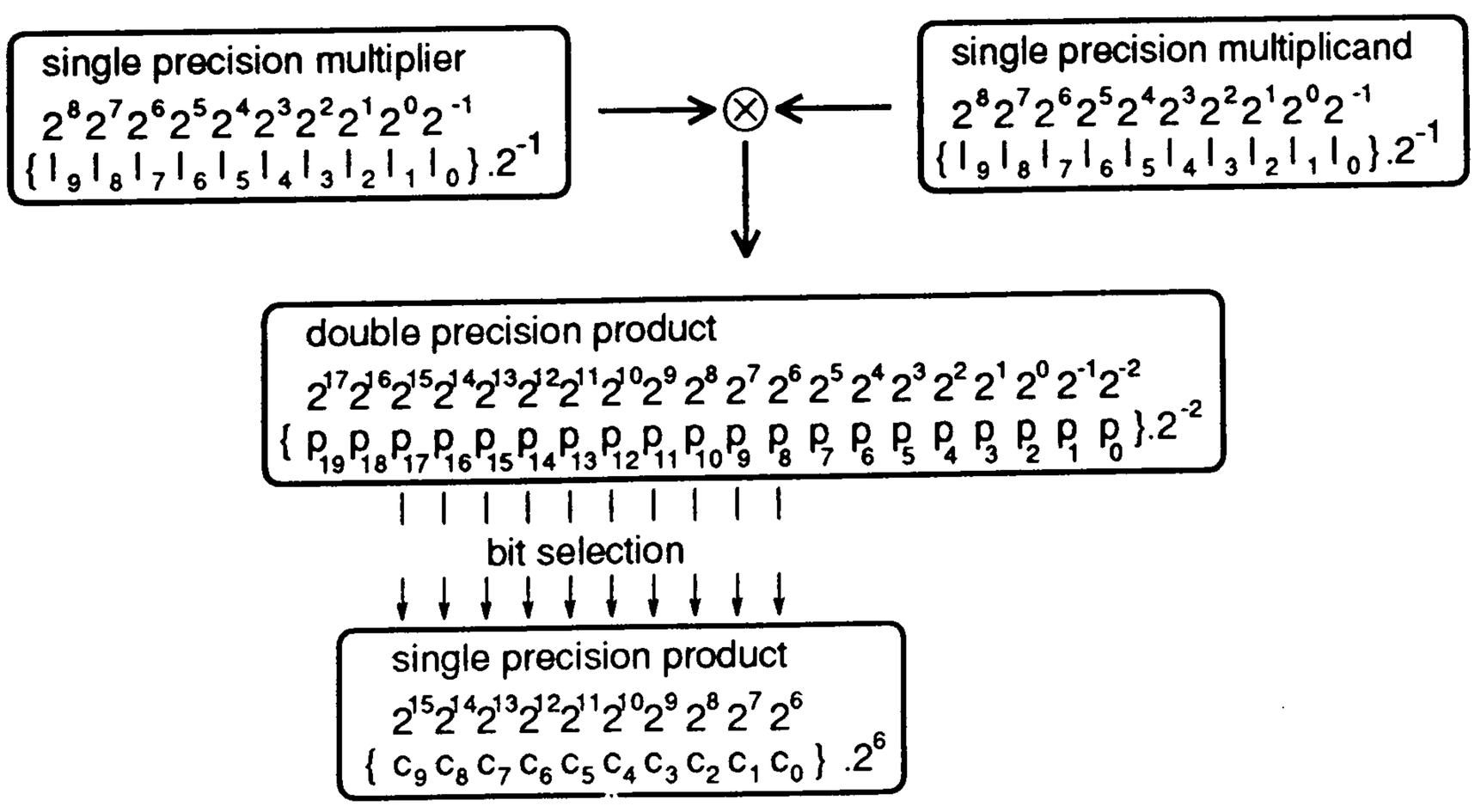


Figure 5.5: Bit selection from the double precision product in a type $PE2$ Cholesky decomposition $[1,1,1]$ array PE (figure 4.8) using a word-length $w = 10$ bits.

The type $PE0$ element in the Cholesky decomposition array requires a square root to be calculated. The output of the square root unit is an element of L and is therefore scaled by 2^{-1} when $w = 10$. To produce a 10 bit output from the square root unit the converse situation to the product formation calculation is encountered and the LSB input to the square root unit should have a weight of 2^{-2}. Since the input to word to this PE is weighted by 2^6 then 8 least significant bit inputs to the square root unit should be zero. A similar situation to the square root is encountered in the division process of the type $PE1$ processors. The dividend is scaled by 2^6, the divider by 2^{-1} and the quotient produced is scaled by 2^{-1}. So the LSB dividend input to the multiplier should have weight 2^{-2} in order to scale the output by 2^{-1} which again means padding out by multiplying the numerator input by 2^8.

For addition and subtraction both the inputs must be scaled the same as the output. The above analysis can also be applied to the LU decomposition example.

(C) Time Dependent Scale Factors

An alternative way to design the array is to have scale factors which change with time. This can be used to reduce the overall error associated with the decomposition by improving the precision of certain data words. By way of example consider the data bus $cn11$ in the Cholesky decomposition array (figure 5.1). The data words $c^{(1)}[1,1]$, $c^{(1)}[2,2]$, $c^{(1)}[3,3]$ and $c^{(1)}[4,4]$ are all transmitted along this data bus and figure 5.6 shows the maximum and minimum values of these data words for the full signal simulation set mentioned earlier. There is a noticeable decrease in the range of the data words on the bus with time. The scale factors on the words with smaller ranges can be decreased thereby increasing the precision with which these words are represented. This increases the precision of the $c^{(3)}[3,3]$ and $c^{(4)}[4,4]$ words by 3 and 4 bits (table 5.4) which could increase result accuracy significantly when dealing with the smaller word-lengths.

	type of matrix			
	$c^{(1)}[1,1]$	$c^{(2)}[2,2]$	$c^{(3)}[3,3]$	$c^{(4)}[4,4]$
maximum	2.9706e+04	2.0506e+04	0.2165e+04	0.1635e+04
minimum	0.7402e+04	0.5105e+04	0.0046e+04	0.0031e+04
scale factor	2^{16-w}	2^{16-w}	2^{13-w}	2^{12-w}

Table 5.4: Scale factors for bus $cn11$ of the Cholesky decomposition array.

This scheme has certain implications when its hardware implementation is considered. For a word-parallel approach changing the scale factor on bus $cn11$ with time means that a multiplexer is required to route different bits from the double precision product to the output bus. The scale of the words on this bus also affects the operation of the square root module in $pe11$ as a different number of zeroes would be required in the least significant positions. Also the scale factors on the $cn22$ and $lk22$ input buses to $pe22$ may change for different data words as time passes and so careful design is needed to make sure that the correct bits are chosen. In the bit-serial approach bit-selection is slightly simpler as clock enable signals are all that is required to pick out the correctly scaled data word from the bit-stream data word. Using a strategy such as this therefore requires a complex control scheme and detailed analysis of probable data distribution before design can commence. In fact with the extra cost of having so many different scale

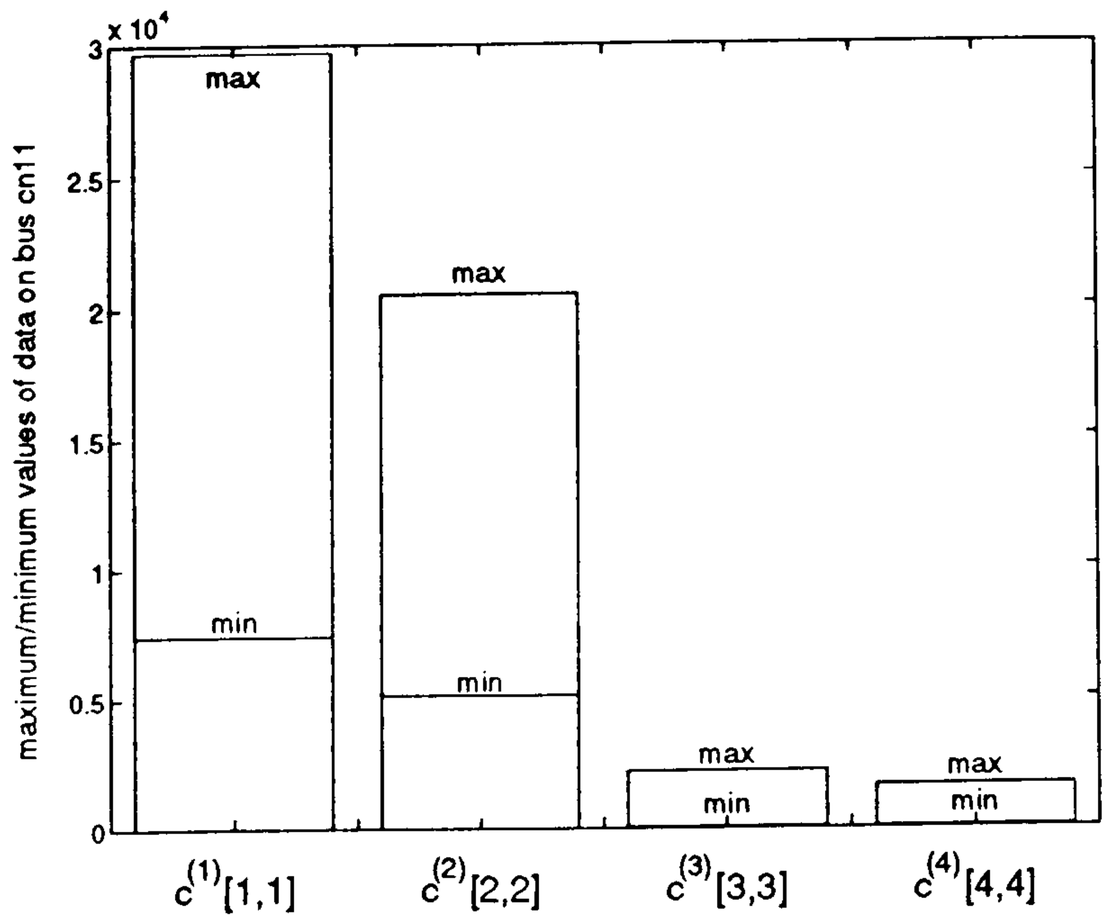


Figure 5.6: Maximum and minimum values of data words communicated along bus $cn11$ of the Cholesky decomposition systolic array shown in figure 5.1.

factors may not be justified when comparing to the more versatile floating point format. For this reason the earlier scheme which proposes associating the same scaling to groups of buses with the carrying the same type of data is preferred.

(D) Behavioural/Structural Simulation

Two's complement finite word-length data Verilog simulations of the Cholesky $[1,1,1]$ and LU $[1,0,0]$ arrays can be developed utilising the scaling schemes suggested in tables 5.2 and 5.3. Behavioural models for the arithmetic units are introduced as resorting to full adder based descriptions of these modules would result in long simulation times and possibly system memory problems. When selecting bits from the double precision products a truncation scheme is followed this allows the required bits to be simply picked off. If rounding is considered extra adders would be needed on each multiplier output to add the double precision product bit whose weight is half that of the scale factor, to the w selected bits but the upper bound of the rounding error is half that of truncation. Even bus widths from 8 to 20 bits are simulated.

5.3 Filter Parameter Computation Cost/Benefit Analysis

A cost/benefit analysis is used to compare the performance of the estimator using the Cholesky $[1,1,1]$ and LU $[1,0,0]$ decomposition methods and a range of word-lengths. The cost function is concerned with the amount of hardware usage, data communication requirement, complexity of the control circuitry and the cost of interfacing the decomposition arrays to the other systolic arrays in the estimator. This is weighed against a benefit which is calculated from the accuracy of each method. The analysis is carried out on a range of word-lengths in order to find an optimal combination of decomposition method and precision.

5.3.1 Hardware Cost

A simple method of selection could be based upon the number of PEs in the array, however this would be of little use since both types of decomposition use the same number of PEs to calculate $\hat{A}$ and this does not account for all the different PE sizes. It would be far better to base the results on the actual number of logic gates that would be required for each array and so the results of the hardware cost comparisons in chapter 4 can be used here. The cost of the forward elimination and back substitution arrays should also be in included since these are slightly different due to the omission of the divider in the LU decomposition forward elimination array as indicated in section 4.5.1.

The cost analysis in chapter 4 looked at comparison of different decomposition arrays and resulted in selection of two quite different designs each with their own input/output data specification. The cost function described in this chapter also uses the costs explored in chapter 4 but also takes into account the cost of integrating each design into the Modified Covariance spectral estimator. That is the cost of connecting the decomposition arrays to the PMA matrix element array chosen in chapter 3 (figure 3.18) and the triangular system solvers (figure 4.2). There are several ways in which systolic array connectivity can be implemented. The first is to have a centralised memory bank from which all data inputs and outputs are fetched and stored depending on the state of the program counter. Such a method would however be likely to cause problems since several different words may be needed to be fetched from or stored in memory on the same clock cycle. An alternative is local memory storage where multiplexing can be used to reorder the data and the cost of this approach is considered here.

(A) Cholesky Decomposition Array System Integration Hardware Cost

The first cost which needs to be considered is the cost of connectivity from the PMA matrix element array to the $[1,1,1]$ Cholesky decomposition array. Study of figures 3.18 and 4.8 shows that the data output on a certain bus of the PMA array needs to be routed to a certain input of the Cholesky array. The PMA array outputs need to be captured as they are produced and then a multiplexor can be used to do the any necessary reordering. This can be done with the use of some registers and multiplexors as shown in figure 5.7. As the matrix elements become available from the PMA array they are piped into a SIPO register. The matrix elements may then be selected as required by control of the multiplexors for input to the Cholesky array. The reordering elements shown in figure 5.7 each have the capacity to store 3 words. This means there are redundant registers but since each element has the same design then the reordering can easily be incorporated into the design of the PMA array PE.

Similar PEs can be used for storage and reordering of data between the Cholesky decomposition array and the triangular system solvers. This time however an extra multiplexor and register would be required in the PE since the maximum number of elements that need to be stored goes up to 4. Another such PE is also required to reverse the order of the Y elements from the output of the forward elimination array before input to the back substitution array.

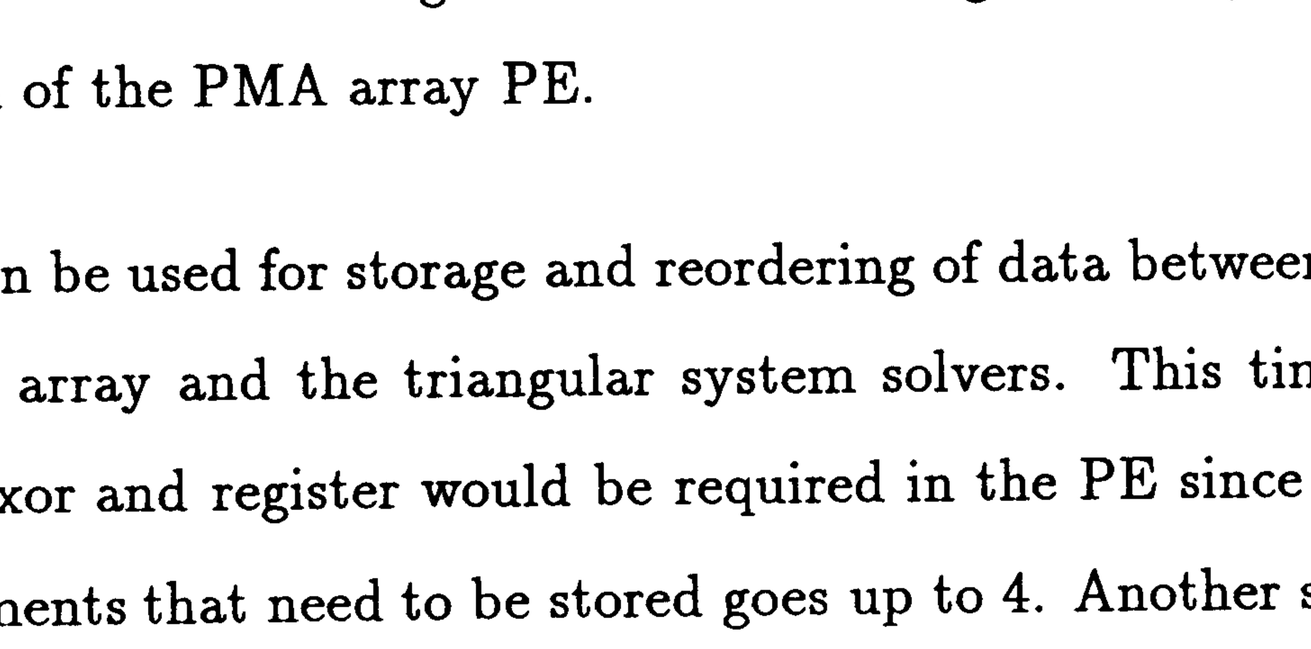


Figure 5.7: Reordering modules for connecting the PMA array to the Cholesky $[1,1,1]$ array.

The RHS matrix (2.24), consisting of elements $c[1,0]$, $c[2,0]$, $c[3,0]$ and $c[4,0]$, is needed for input to $pe[0]$ of the forward elimination array (figure 4.2(a)). These elements and also $c[0,0]$ are required in the white noise variance calculation (2.28). Grouping this data together presents a slightly more difficult problem since each of these elements appear on different outputs of the PMA array (figure 3.18). The reordering PEs on the outputs of the PMA array route elements $c[4,1]$, $c[4,2]$, $c[4,3]$ and $c[4,4]$, equal to $c[3,0]$, $c[2,0]$, $c[1,0]$ and $c[0,0]$, into the Cholesky decomposition array (figure 4.8) on input clock cycles $t = 6$, 7, 8 and 9 respectively. To follow the pattern another reordering PE can be used to output $c[4,0]$ at $t = 5$. The grouping of this data can be represented using the data dependence graph in figure 5.8 where the horizontal inputs to row j are the outputs from the reorder PE associated with $pe[j]$ of the PMA array. The *mode* 0 nodes (figure 5.8(b)) represent the capture of the matrix elements sourced from the reordering PE outputs at time t. The *mode* 0 nodes output the matrix elements along a diagonal path through the *mode* 1 nodes to the edge of the graph, maintaining a localised communication. Projection along the t axis results in the linear systolic array shown in part (c) of the figure, whose simple pipelined multiplexing function is shown in part (d). The output of $pe[4]$ is then connected to another of the reordering type PEs, similar to those in figure 5.7 but with the capacity to store 5 words.

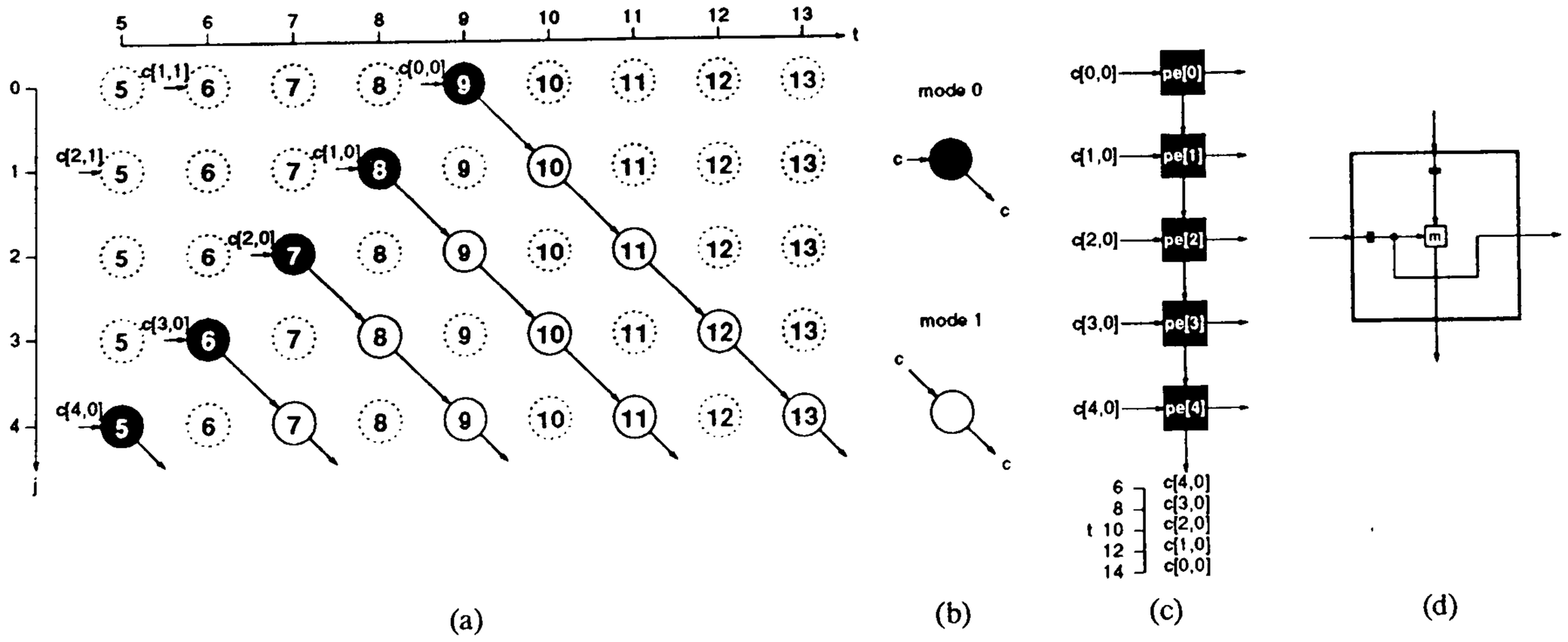


Figure 5.8: (a) Localised DDG for regrouping of elements $c[j,0]$ from the PMA array, (b) node functions, (c) systolic array from t projection and (d) PE function.

(B) LU Decomposition Array System Integration Hardware Cost

It is much more difficult to integrate the *LU* decomposition array into the system because of its input bus data flow requirements and the ordering of data produced on each output bus. The input data to a certain bus of the $[1,0,0]$ *LU* decomposition array must now be pulled off 3 or 4, depending on the input bus, of the PMA matrix element array outputs. Figure 5.9(a) shows a network of elements for performing the required regrouping and reordering. Each column of regroup PEs performs a similar function to the linear regrouping systolic array shown in figure 5.8. The matrix elements which have been collected together in the regrouping array are then stored in the reorder PEs (figure 5.7), which have a capacity to store 5 matrix elements, before connection to the *LU* decomposition array. There are some redundant PEs in this interconnection network which are judged to be worth including in order to maintain regularity when considering VLSI/FPGA implementation. Regroup and reorder networks are also required between the *LU* decomposition array and the triangular system solvers and suitable architectures to perform this are shown in figure 5.9(b) and (c). A reordering PE is also required between the forward elimination array output and the input to the back substitution array.

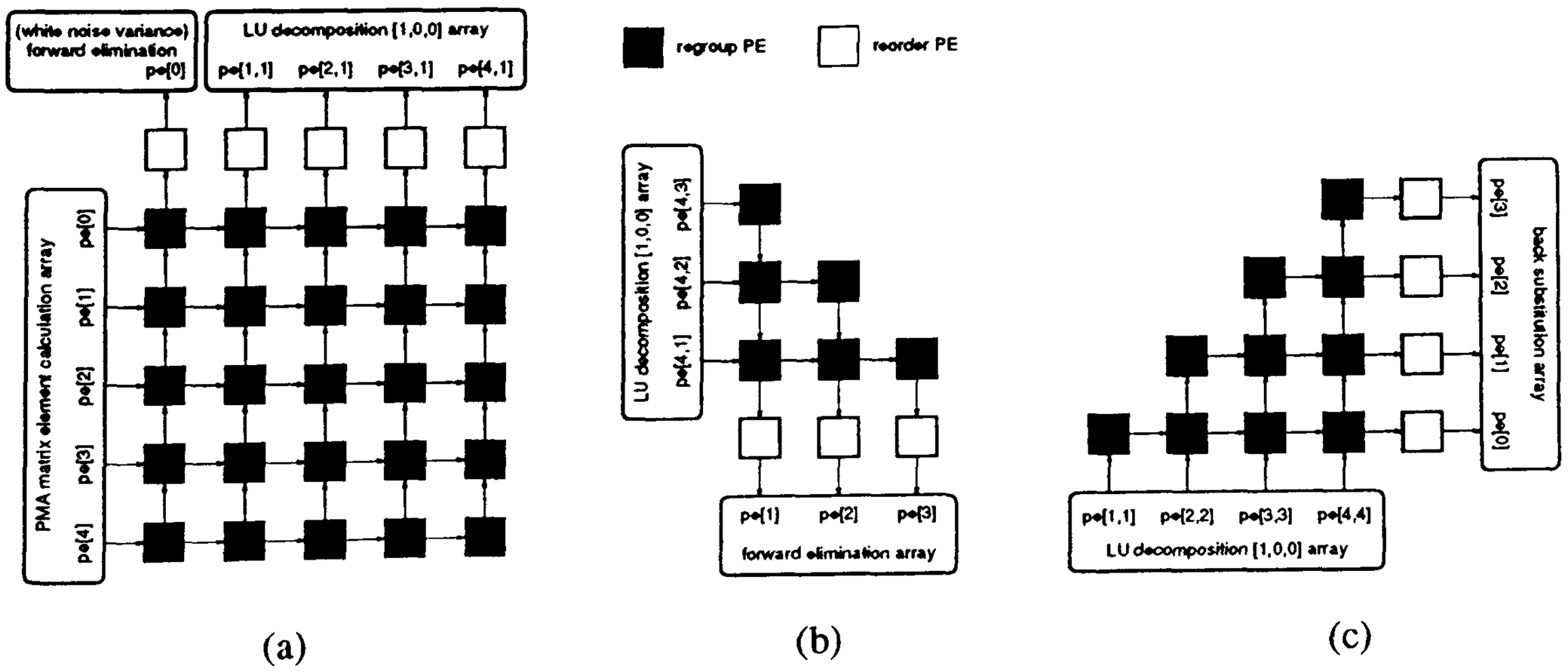


Figure 5.9: Interconnection networks for (a) PMA array to *LU* decomposition/forward elimination arrays and from the *LU* decomposition array to (b) forward elimination array (c) back substitution array.

The regrouping networks have the disadvantage of considerably increasing control burden and computational delay. An alternative way to approach the problem is to redesign the forward elimination arrays to accept the data groupings which are output from the LU decomposition array so that the networks shown in figure 5.9(b) and (c) can be replaced just by reorder PEs. The DDG for back substitution is shown in figure 5.10(a) and the mode functions associated with the nodes are indicated in part (b) of the figure. Kung and Leiserson's [17] array shown in figure 4.2 is produced by the $[0, 1]$ vector projection in the k direction. Projection in the $[1, 1]$ direction however, such that $node[j, k]$ is projected into $pe[k - j]$ of the systolic array shown in figure 5.10(c), results is much more convenient data flow. The disadvantage with this mapping however is that now there is only one type of processor which is required to work in two different modes as detailed in part (d) of the figure and so the array contains 4 dividers as opposed to one for the other mapping. Note that dividers can be omitted from forward elimination version of this array.

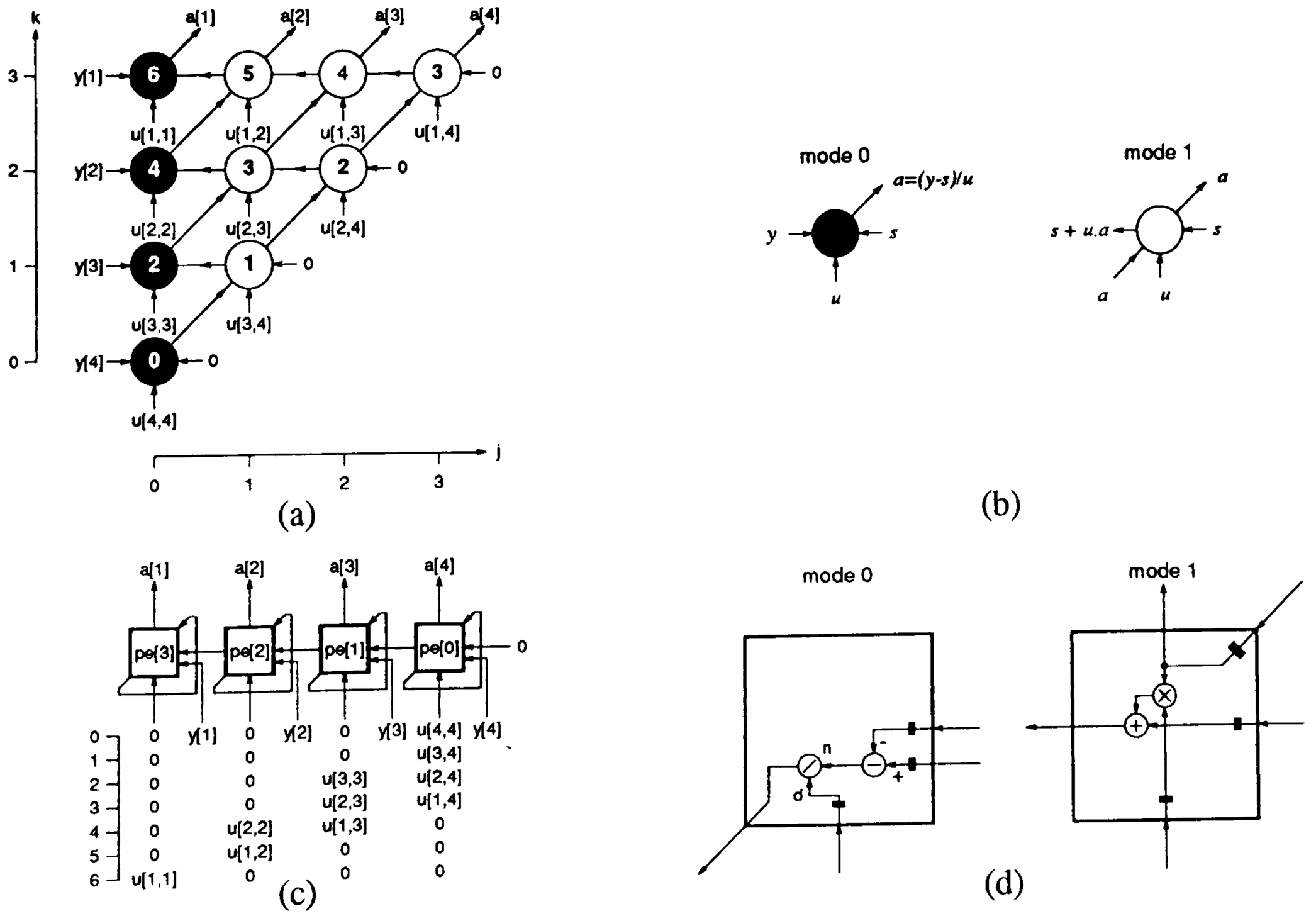


Figure 5.10: (a) DDG for back substitution, (b) modes of DDG nodes, (c) systolic array formed by $[1, 1]$ projection and (d) PE modes.

5.3.2 Overall Filter Parameter Calculation Hardware Cost

The total hardware cost of the filter parameter calculation, that is the cost of the decomposition arrays, triangular system solvers and the connectivity between the modules is discussed in this section.

It could be argued that the same hardware for the forward elimination array could be used for the back substitution array since the latter calculation cannot commence until $y[4]$ is output on the former computation. The reason that this is not arranged is due to differences in scale factors needed in the two processes which leads to different bits being selected from the double precision products of the multipliers and varying scales applied to the numerator inputs of the dividers. If one array was to be used for both triangular solvers then this would therefore lead to time varying scale factors as described in section 5.2.1(C).

The approximate total cost for the filter parameters computation when using the Cholesky decomposition $[1, 1, 1]$ array shown in figure 4.8 is shown in table 5.5 in terms of modules. These results are based upon the use of the forward elimination/back substitution arrays of figure 4.2 and the array interconnection costs discussed in section 5.3.1(A).

w bit module	number of modules used in the arrays			
	Cholesky decomposition	forward elimination	back substitution	array interconnection
addition	0	3	3	0
subtraction	6	1	1	0
multiplication	6	3	3	0
division	3	1	1	0
square root	1	0	0	0
register	18	11	11	47
multiplex	0	0	0	32

Table 5.5: Hardware usage for Cholesky systolic arrays in terms of module usage.

When considering the use of the LU $[1, 0, 0]$ array in the filter parameter computation, two different versions of triangular system solvers are considered. Table 5.6 details the costs for when the original triangular system solvers of figure 4.2 are used with the interconnect networks of figure 5.9. These costs can be compared to the costs in table 5.7 which consider the use of the redesigned triangular system solvers of figure 5.10, eliminating the need for the interconnection networks in figure 5.9(b) and (c).

w bit module	number of modules used in the arrays			
	LU decomposition	forward elimination	back substitution	array interconnection
addition	0	3	3	0
subtraction	6	1	1	0
multiplication	6	3	3	0
division	4	0	1	0
square root	0	0	0	0
register	26	11	11	136
multiplex	0	0	0	82

Table 5.6: Hardware usage for LU systolic arrays in terms of module usage when using the triangular system solvers of figure 4.2.

w bit module	number of modules used in the arrays			
	LU decomposition	forward elimination	back substitution	array interconnection
addition	0	4	4	0
subtraction	6	4	4	0
multiplication	6	4	4	0
division	4	0	4	0
square root	0	0	0	0
register	26	16	16	100
multiplex	0	0	0	63

Table 5.7: Hardware usage for LU systolic arrays in terms of module usage when using the modified triangular system solvers of figure 5.10.

The hardware cost, including the cost of system integration, for the filter parameter calculation in the Cholesky and two LU decomposition based systems plotted in figure 5.11 over word-length from 8 to 20 bits, is derived from the information forwarded in tables 3.1, 3.2, 4.1, 5.5, 5.6 and 5.7. The results show that the LU

decomposition methods are more expensive to implement in terms of the NAND gate cost for all the word-lengths considered. The LU method with the original triangular system solvers presents lower hardware cost than that displayed after the modification to the triangular system solvers when the word-parallel approach is considered. Conversely, in the bit-serial approach the modification to the triangular system solvers presents the lower cost of the two LU systems with the need for control signals to regroup the LU array outputs removed. Due to these two advantages and since the bit-serial approach is most appropriate to the Modified Covariance application then the LU system with the modified triangular system solvers is chosen out of the two LU methods. The LU method which uses the regroup networks to set up the L and U matrix data on the inputs to the original triangular system solvers is therefore not given any further consideration.

Systolic array interconnection strategy has a large effect on the results as is evident from comparison the bit-serial results from chapter 4 with those presented here. Figures 4.11(b) and 4.20 show the bit-serial costs of the Cholesky $[1,1,1]$ and LU $[1,0,0]$ arrays taken alone to be very similar. When these arrays are integrated into the filter parameter calculation system however the cost of the LU based system rises to between 1.5 to 1.75 times higher than the Cholesky system. This demonstrates the distinct cost advantage of simple data flow between the PMA and Cholesky decomposition array.

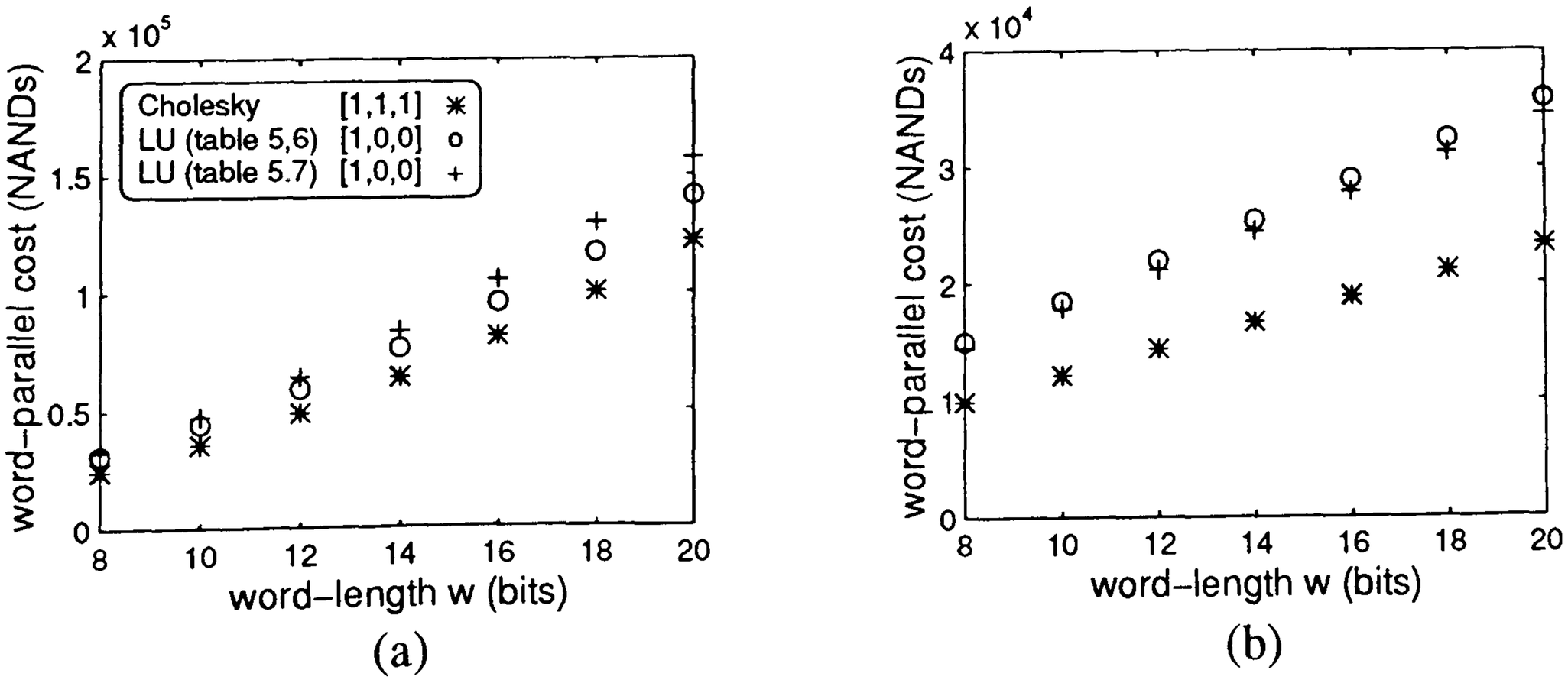


Figure 5.11: Hardware cost for (a) word parallel and (b) bit-serial approaches.

5.3.3 Data Communication Cost

The data communication cost in terms of number of buses is detailed for the two decomposition methods in table 5.8. The communication cost for the forward elimination and back substitution arrays is the same and so the cost for just one such array in each of the decomposition methods is presented in the table. The Cholesky decomposition shows a greater PE interconnection requirement burden, fewer input/output lines but does not have any localised feedback connections as in the LU decomposition. In the word-parallel approach the number of data lines can be expressed in terms of the word-length w as there are this many lines per bus and for the bit-serial approach it is assumed that the width of each data bus is one. Communication burden in the regrouping network is identified in the next section.

type of data bus	Cholesky [1, 1, 1]		LU [1, 1, 1]	
	decomposition	forward/back substitution	decomposition	forward/back substitution
PE interconnection	$12 + 6\sqrt{2}$	6	12	3
localised feedback	0	0	10	4
array input	4	6	4	9
array output	4	1	7	4

Table 5.8: Cholesky and LU decomposition data bus costs.

5.3.4 Weighting Hardware and Communication Cost

It is very difficult to weigh communication cost against hardware cost as this involves consideration of the floor planning and the technology on which the systolic arrays are to be mounted, be it full/semi-custom or field programmable gate arrays (FPGAs) for example. In order to see the overall effect of communication on the overall cost one of these methods of implementation must be considered. The cost analysis presented here is geared towards FPGA implementation, as these devices present a convenient platform for prototyping systolic arrays [128] and their logic cell array (LCA) specification presents a basis on which hardware and communication cost can be compared.

A Xilinx FPGA [129] contains an array of user configurable logic blocks (CLBs) surrounded by a number of pins. Each CLB contains a combinatorial function logic block, capable of being programmed to perform a single FA, GFA or CAS cell function equivalent to an average of $\approx$ 14 NANDs, and two D-type flip/flops, adding on 12 more NAND gates. This leads to an average gate count of $\approx$ 26 NANDs per CLB. The I/O blocks associated with each pin can be programmed to perform a D-type flip/flop operation giving a NAND gate count of 6. Using this information the the number of NAND gate equivalents for each device in the Xilinx XC3100 LCA family [129] is shown in table 5.9.

Bus connections can be made via the pins of the FPGA or internally using the programmable interconnect points (PIPs) of the LCA architecture and so the number of NANDs that a bus link is worth can be determined by estimating the NAND gate equivalence for a pin or internal interconnect. The value of a pin resource in terms of NAND gates can be calculated by taking the ratio of the number of NAND gate equivalents for the whole FPGA to the number of pins for that device as shown in the final column of table 5.9. The cost of an internal interconnection resource can be estimated by considering the number of input/output lines that a CLB can support. Each CLB of the array is capable of supporting 5 logic inputs, a direct input and 2 outputs a total of 8 data lines of which 5 or 6 could typically be programmed in an efficient design. Therefore the 26 NANDs associated with each of the CLBs can be weighted against 5 or 6 internal communication lines leading to a CLB input or output connection being equivalent to $\approx$ 5 NAND gates.

Device	No. of CLBs	No. of Pins	NANDs	NANDs/pin
XC3120	64 (8 x 8)	64	2000	31
XC3130	100 (10 x 10)	80	3100	39
XC3142	144 (12 x 12)	96	4300	45
XC3164	224 (16 x 14)	120	6600	55
XC3190	320 (16 x 20)	144	9200	64
XC3195	484 (22 x 22)	176	13600	77

Table 5.9: Pin to CLB utilisation in Xilinx XC3100 Series FPGAs.

The question which now arises is how many of these pins are utilised for bus connections compared to internal bus links. To answer this the number of PEs which can be implemented on a single FPGA must be studied. When considering implementation of the word-parallel approach the arithmetic units involved are not pipelined and therefore the D-type flip-flops contained within the CLBs are not used. The maximum utilisation is therefore not much greater than 60% of the total number of available gate equivalents when using the word parallel approach, assuming that some of the D-type flip-flops go towards providing the systolic delays. In figure 5.12 the Xilinx NAND gate equivalents for 60% utilisation (indicated by the dotted horizontal lines) are compared with the average cost of a systolic array arithmetic PE in the *LU* and Cholesky based systems. From these results it can be concluded that one word-parallel arithmetic PE can be successfully implemented on a single FPGA (e.g. the average size PE in the 14 bit Cholesky based system can be mounted on the XC3164). On implementing a single PE per FPGA all PE interconnections must be made via the pins and so each interconnection bus link taking up 2 pins can be weighted by choosing the most appropriately sized FPGA and referring to table 5.9. Systolic array input/output bus links each just count for one pin while localised feedback can be implemented within the FPGA from the output of one CLB to the input of another, each feedback link therefore counting towards 10 NAND equivalents.

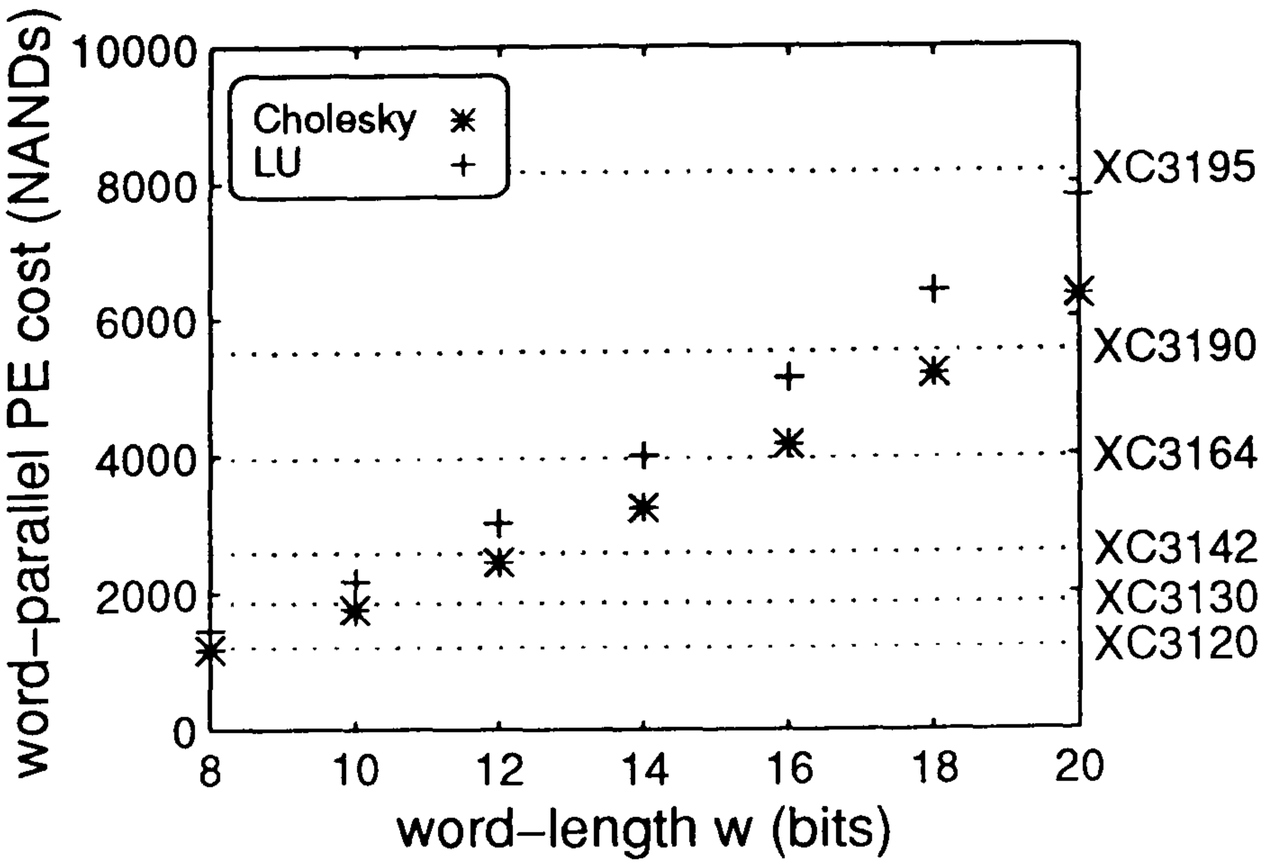


Figure 5.12: Average cost of word-parallel systolic array PEs from *LU* and Cholesky integrated systems (not including regroup and reorder PEs).

A much higher utilisation of up to 90% can be achieved for the bit-serial arithmetic modules [128] since the CLB flip-flops can be used to provide the pipeline delays required for implementation of the arithmetic modules. This high utilisation and the use of fewer NAND gates in bit-serial implementation allows whole systolic arrays to be mounted on single FPGAs. Figure 5.13 shows the hardware cost of the bit-serial Cholesky and LU decomposition systolic arrays for various word-length which can be compared to the number of NAND gate equivalents for each of the Xilinx XC3100 series devices at 90% utilisation. The combined cost of the forward elimination and back substitution arrays for each of the two decomposition systems is also shown and it can be seen that both of these arrays in each case may be mounted on a single FPGA. Pins are therefore only required for input/output bus connections of the decomposition systolic arrays and just one of triangular system solvers. PE interconnection, localised feedback, the connections between the forward elimination and back substitution arrays can all be made within the FPGAs.

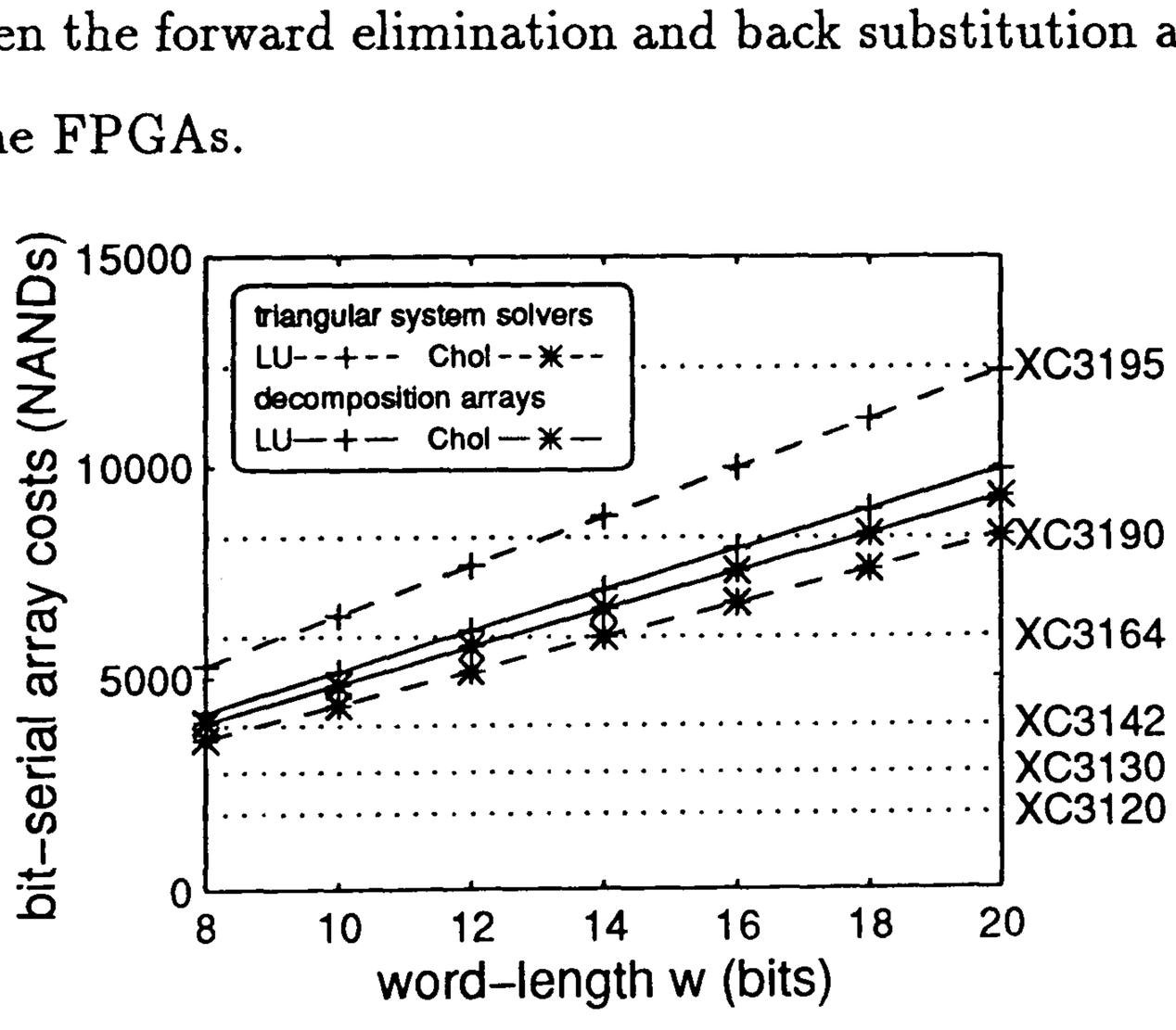


Figure 5.13: Costs of bit-serial Cholesky and LU decomposition systolic arrays and total costs of the forward elimination and back substitution arrays in each of the two systems.

In order to efficiently implement the regrouping network on a FPGA a combined bit-serial/word-parallel implementation is suggested independent of whether the arithmetic in the other systolic arrays is bit-serial or word-parallel. The reasons

for this are that if a purely word-parallel approach is taken then there would be a very large input/output connectivity burden compared to a small processing task, leading to very inefficient use of the FPGA. The problem with a purely bit serial approach is that this would make regrouping relatively slow and this would also make quite inefficient use of the FPGA architecture as only one multiplexer would be required for every $2w$ D-type flip-flops. A serial/parallel approach draws a compromise allowing regrouping to be performed at an acceptable rate, with lower I/O burden than the strictly word-parallel approach and a more efficient usage of CLBs. A possible arrangement for the serial/parallel regrouping PE is shown in figure 5.14. The communication burden resulting from use of this PE arrangement in the Cholesky and LU systems is shown in table 5.10 where the PE interconnections are routed on the FPGA layout and the array I/O connections[1] are made via pins. The most suitable Xilinx device can be selected given that w CLBs per regroup PE are required.

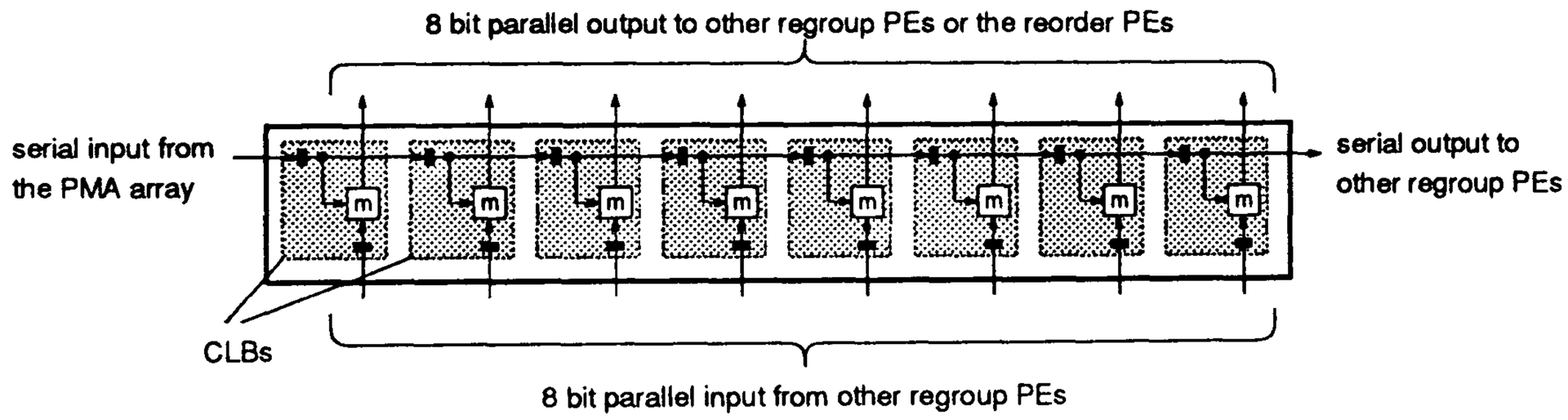


Figure 5.14: Serial/parallel layout of a $w = 8$ bit regroup PE suitable for implementation on w CLBs of a Xilinx FPGA . Note that serial input registers need to be clocked a factor of w times higher than the parallel input registers.

type of communication	Cholesky system	LU system
PE interconnection	$4w$	$20w + 20$
array input	5	5
array output	w	$5w$

Table 5.10: Communication burden in the regroup arrays.

[1]For the case when connecting to a bit-serial system the output communication burden can be reduced by a factor of w by incorporating parallel to serial conversion within the Xilinx FPGA.

5.3.5 Overall Hardware and Communication Cost

The methodology for expressing communication burden relative to hardware cost as described in the section 5.3.4 with regard to implementation on the Xilinx 3100 series of FPGAs is used to obtain the overall hardware and communication results plotted in figure 5.15. By comparison with the results for the hardware cost only in figure 5.11 it can be seen that the effect of communication is much more pronounced in the word-parallel approach accounting for 45% and 49% of the overall cost in the *LU* and Cholesky systems respectively. Communication has a much smaller burden in the *LU* & Cholesky bit-serial approaches where it respectively estimated to contribute to an average of 13% and 9% of the overall cost, with the large PE interconnection burden of the *LU* regrouping array pushing up the cost in the *LU* system. However the general trends when comparing the Cholesky and *LU* based systems are repeated since the Cholesky decomposition array displays considerably lower cost throughout the range of word-length in both word-parallel and bit-serial situations. These results further demonstrate the cost advantage of the bit-serial approach over word-parallel since in the case of the Cholesky based system, the overall bit-serial costs range from 25%, at $w = 8$ bits, to 10% at $w = 20$ bits, of the overall word-parallel costs.

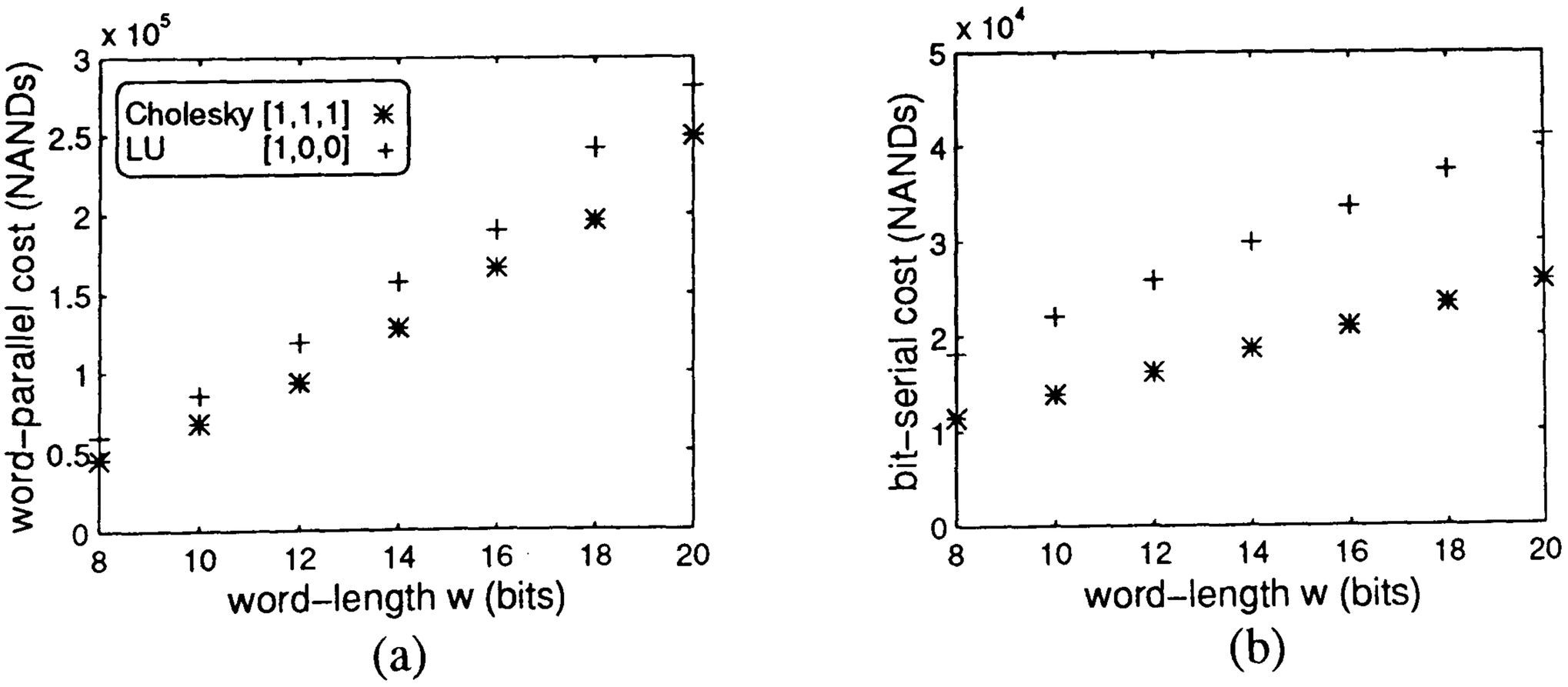


Figure 5.15: Combined hardware and communication cost estimates for (a) word parallel and (b) bit-serial approaches.

5.3.6 Control Cost

The control can be broken up into three broad sections, array level control, array connectivity control and arithmetic module control. Array level control covers signals such as the clock, reset and register clock enable signals which control the processor mode of operation. The array connectivity control determines when output data is to be stored and provides multiplexor selection signals to set up the input data streams to the systolic arrays. The arithmetic control signals are for the arithmetic modules and these need to be repeated within the systolic clock cycle at one in every α clock cycles, where α is the processor pipelining period.

(A) Array Level Control

As discussed in the previous chapter control of the Cholesky decomposition array at the array level is very simple due to the fact that in the $[1,1,1]$ mapping only nodes operating in the same mode are projected into the PEs of the array. This is also true for the forward elimination and back substitution arrays which are used with the Cholesky decomposition array and hence there is no need for any array level control.

The LU decomposition array requires a set of clock enable pulses to store the upper triangular matrix elements in the processor feedback registers. The flow of this clock enable signal can easily be pipelined throughout the array. For example the clock on the ru register of $pe[1,1]$ (figure 4.13) needs to be enabled on the $t = 0$ clock edge. The neighbouring $pe[2,1]$ needs $u[1,2]$ to be stored at $t = 1$ and so a single bit register delay from the $pe[1,1]$ enable signal could be used to provide the enable for $pe[2,1]$. A localised control network can then be set up throughout the array. A similar enabling signal is necessary in the triangular system solvers used with the LU decomposition array but with a pipeline of 2 delays between processors.

(B) Array Connectivity Control

Both decomposition methods need control signals to store the systolic array output data in the reorder PEs and to schedule this data for input to the neighbouring arrays. For each reorder cell these control signals are in the form of a register storage clock enable line and a multiplexor control bus, the width of which depends on the number of word registers in the reordering elements, so that the data stored can be selected by parallel access. Multiplexor control signals are also required in the regrouping networks to redirect the elements of the from the various PMA array outputs onto a number of buses. One multiplexor select signal is required for each regroup PE and the array connectivity control burden is summarised in table 5.11.

type of signal	Cholesky system	LU system
regroup - MUX control	5	25
reorder - MUX control	21	29
reorder - clock enable	10	12

Table 5.11: Array connectivity control burden in terms of number of control lines.

(C) Arithmetic Module Control

The advantage of using a word-parallel over a bit-serial approach becomes apparent when considering this type of control signal. The bit-serial modules need their own clocks operating within the systolic cycle, MSB indicator pulses, plus mode control pulses while the word-parallel equivalents require only data input on the systolic clock edge and results ripple through with little or no control signal requirement. The bit-serial Cholesky decomposition process has a bigger disadvantage here because the control of the square root unit demands 4 different modes of operation in each of its bit-level PEs. Both of the bit-serial decomposition methods need addition, subtraction, multiplication and division unit control signals.

(D) Control Results Summary

Its very difficult to quantify the control burden and match it against that of the hardware and communication cost. The *LU* decomposition process is at disadvantage due to its clock enabling requirements for storage of localised PE feedback words whilst in the Cholesky array mode changing of nodes is not required for any of its four PEs. When considering array interconnectivity, table 5.11 shows that the Cholesky decomposition array requires fewer control lines than in the *LU* system for both the reordering and regrouping of data. The only drawback with the Cholesky system is the control required for the bit-serial square-root computation but this disadvantage becomes insignificant in comparison with the control complexity in the required in the *LU* system.

5.3.7 Efficiency Comparison

Processor utilisation could also be used for comparison of the methods. Each PE in the Cholesky array is active on one in every three clock cycles whilst the PEs in the *LU* array are active on each consecutive clock cycle. The inefficiency of the Cholesky array may be looked upon as a disadvantage but since these arrays both require a similar number of clock cycles there is no real gain in using the *LU* method based on utilisation. Also, the data interleaving technique could be used to improve the processor/block pipelining periods of the Cholesky decomposition array to obtain a similar efficiency to that of the *LU* decomposition array. The latency of the individual PEs could be used as a factor in the comparison. The square-root PE in the Cholesky array causes a bottleneck since the iterative nature of these devices causes them to be substantially slower than multipliers. However, much the same problem is encountered with the division process and so the Cholesky decomposition is not at a particular disadvantage in this respect and efficiency is not brought into the cost/benefit analysis.

5.3.8 Method of Error Calculation and Benefit Analysis

Benefit is considered here to be a measure of spectral estimation accuracy and specifically estimation of those spectral parameters of particular interest in this application - mean frequency (f_m) and RMS bandwidth (f_b).

The set of stationary signal cases listed in table 5.1 for scale factor calculation can be used again to perform the error analysis here. A cost/benefit selection process [11], which also uses data with these characteristics, leads to the choice of the modified covariance method over a range of other spectral estimators. The stationary signals had Gaussian spectral shapes with mean frequency f_m, RMS bandwidth f_b.

The 1100 covariance matrices generated (see section 5.2.1(A)) can be applied to the Verilog descriptions of the two's complement decomposition modules. All cases use a data window of a fixed number ($N = 512$) samples rather than a fixed time duration so that the cases with the same percentage bandwidth consist of similar numbers of frequency bins and the long window length ensures that good estimates of f_m and f_b can be made. The filter parameters are retrieved from the output of the Verilog simulations, which are run for 8 to 20 bit word-lengths, and Matlab is again used to calculate the PSD from which f_m and f_b are estimated.

The different methods of decomposition may be compared on the basis of the RMS error in f_m and f_b caused by the limited precision and overflow error in the filter parameter outputs for different word-lengths. In addition a user defined Verilog behavioural function indicates overflow occurrence. Taking each decomposition process separately the f_m and f_b RMS percentage error versus word-length plots are similar for all signal cases with the same percentage bandwidth and so the average errors are taken over these cases.

The benefit function is defined as inverse of the observed error:

$$benefit = \frac{1}{(W_{f_m}.RMS(f_m)) + (W_{f_b}.RMS(f_b))} \tag{5.1}$$

where

$$RMS(f_m) = RMS_{5\%}(f_m) + RMS_{10\%}(f_m) + RMS_{20\%}(f_m) \tag{5.2}$$

$$RMS(f_b) = RMS_{5\%}(f_b) + RMS_{10\%}(f_b) + RMS_{20\%}(f_b) \tag{5.3}$$

and $RMS_{5\%}(f_b)$, for example, is the average RMS error in f_b over the 5% bandwidth cases. W_{f_m} and W_{f_b} are weights which may be set to reflect the relative importance of mean frequency and bandwidth estimation accuracy.

5.3.9 Results of Error Calculation and Benefit Analysis

The results of the error analysis in figure 5.16 show percentage RMS error in both the estimated values of f_m and f_b compared to the ideal values given in table 5.1 versus word-length w for the three different percentage bandwidths. The dotted line is an asymptote representing the error produced by the actual estimation process and this result is calculated using the Matlab software at high precision. As expected the RMS error curves converge to this line as the word-length is increased. The Cholesky process is the most accurate of the two with RMS error becoming most significant at 8 bits. With decreasing word-length the error becomes apparent at an earlier stage in the 5% half bandwidth case compared with the 10% case and similarly the error becomes more noticeable in the 10% case before it does in the 20% case. This is because as the percentage bandwidth of the input signal is decreased then the covariance matrix to be solved is more likely to become ill conditioned [27]. The condition of the covariance matrix is a measure of the sensitivity of the system of linear simultaneous equations to relative error in the covariance and RHS matrix. Using Matlab to evaluate the condition numbers (K) for the covariance matrices, K was found to average around 5000 for the 5% cases 1000 for the 10% cases and 200 for the 20% cases.

According to matrix theory the relative error in the solution of the simultaneous equations could be up to K times that of the input matrices [27] and this change in condition number with percentage bandwidth is consistent with the change in estimation error.

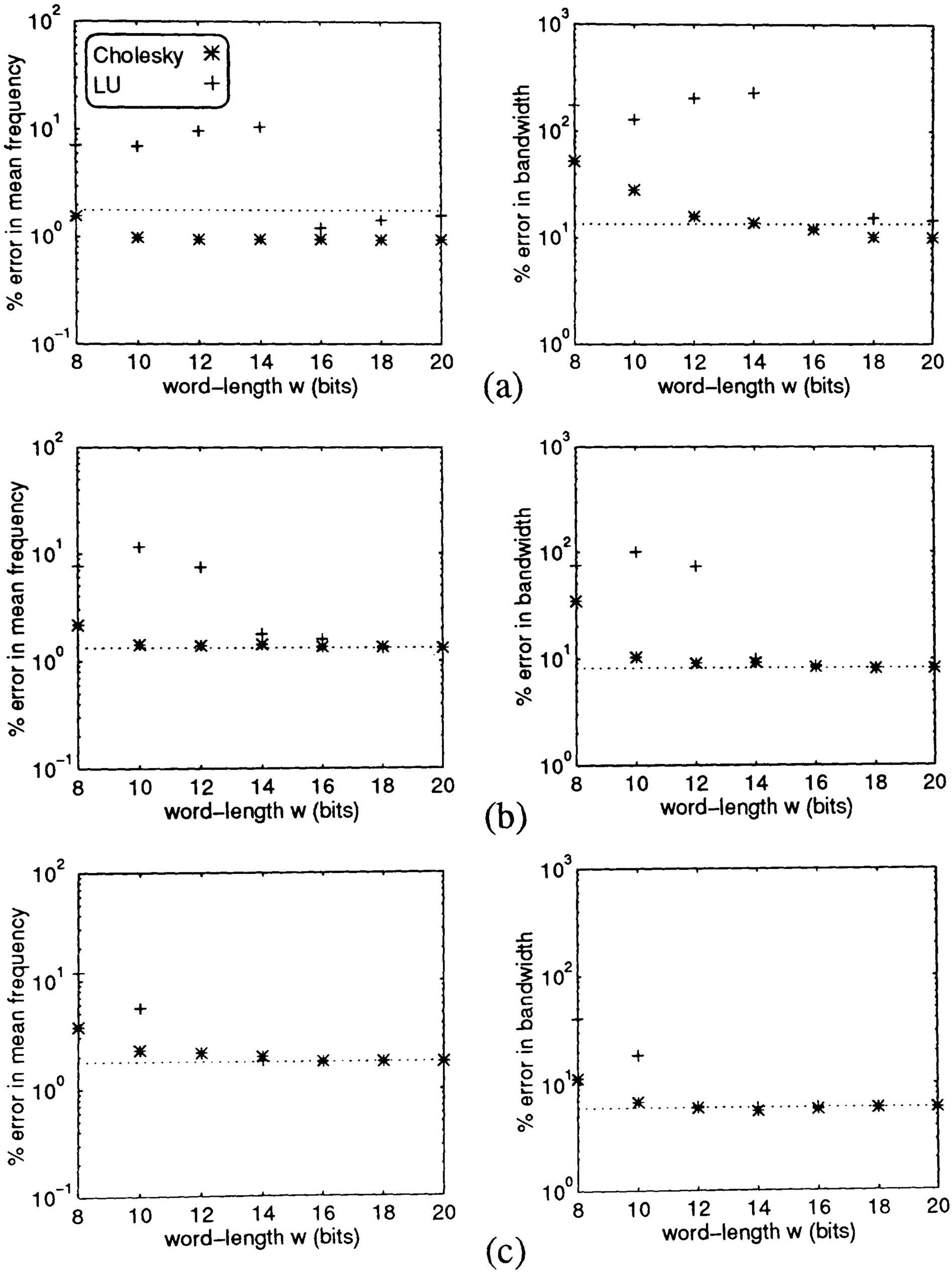


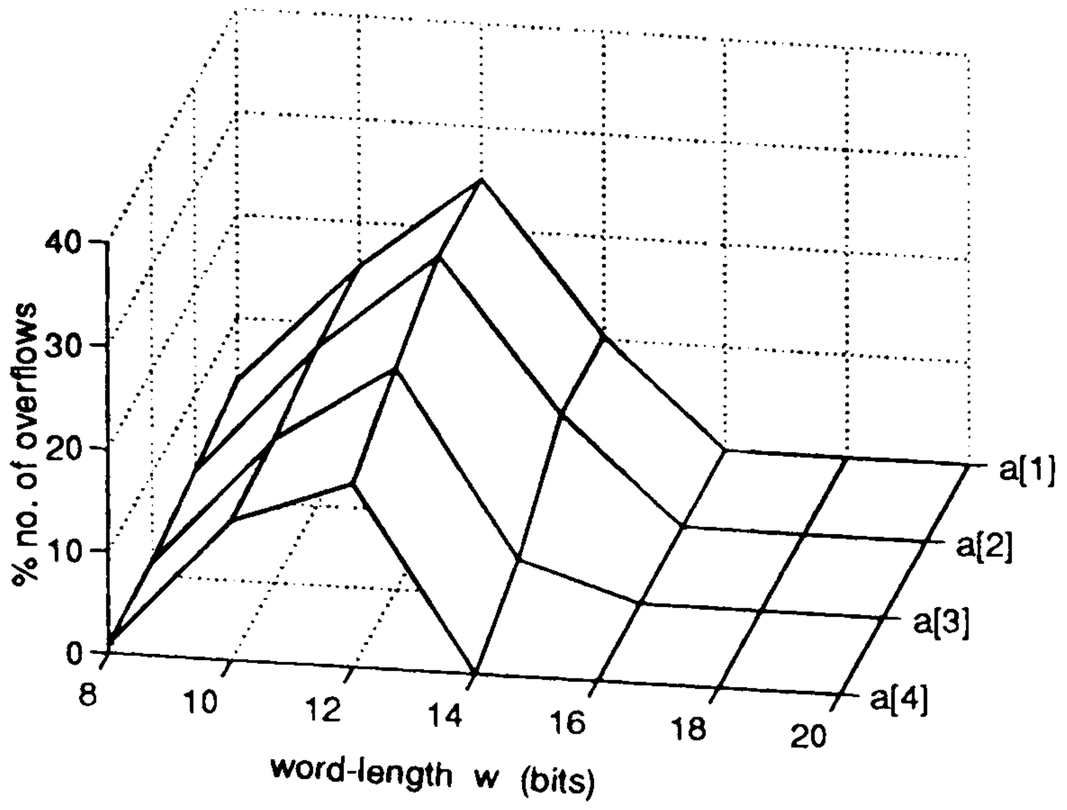
Figure 5.16: Percentage error in decomposition methods for (a) 5%, (b) 10% and (c) 20% mean frequency to bandwidth ratios.

Overflow errors [88] occur if the scale factor is not large enough when selecting the appropriate bits from a double precision product or when the two inputs to an adder are too large. They are in turn caused by rounding or truncation errors whose accumulation causes the multiplier and multiplicand or adder inputs to deviate from the values which would be calculated when using high precision floating point arithmetic.

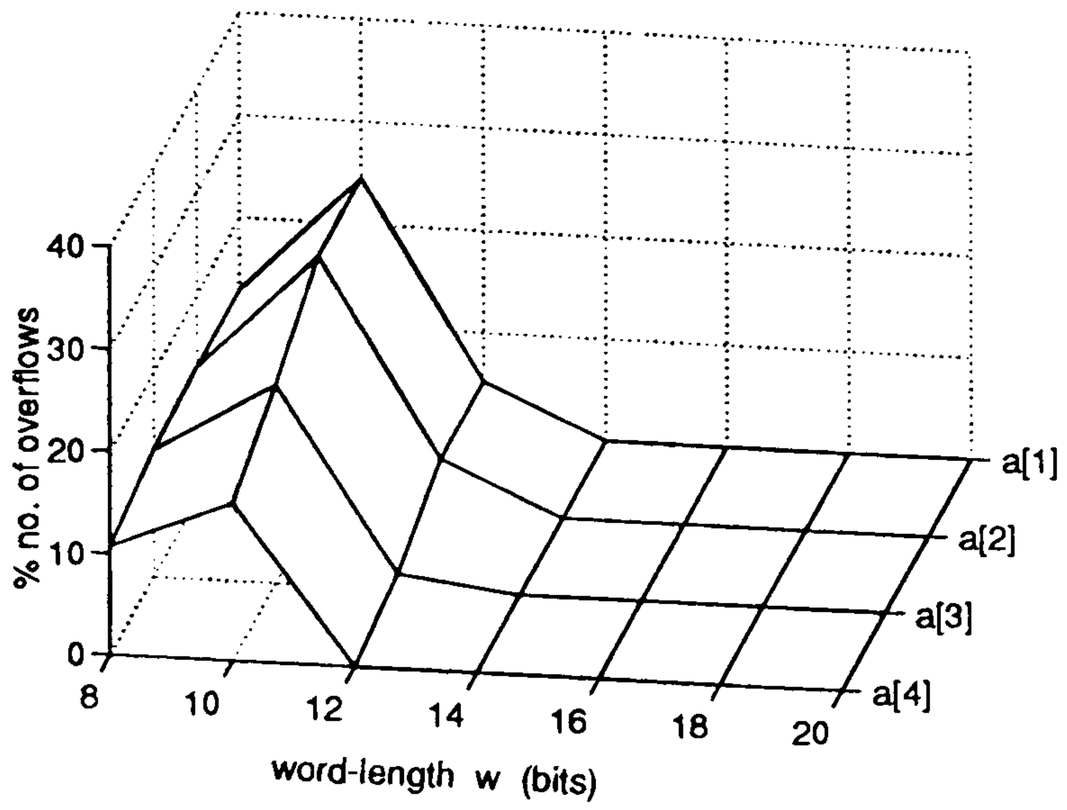
The Cholesky decomposition architecture was found to suffer from only a very small percentage of overflows at 8 bit word-length in the 5% bandwidth case and was robust for the rest of the range. However the LU array did not fare so well. Figure 5.17 shows the percentage of the 1100 cases of filter parameter results in error due to overflows. For each of the three percentage bandwidth cases the number of results suffering from overflow error as a percentage of the total number of simulations in that case is plotted for each of $\hat{a}[4] \rightarrow \hat{a}[1]$. On the LU plots most of the overflows are already apparent in the system before the $U.A = Y$ back substitution calculation. Investigation reveals that the main source of overflow is in the calculation of the $l[4,3]$ and $u[3,4]$ matrix elements. With reducing word-length it can be seen that the points at which overflows start to enter the systems are at the same word-lengths at which the dramatic increase in RMS error is observed, that is for example at 14 bits for the LU 5% bandwidth case shown in figure 5.16.

Another point of interest is that the number of overflows peaks at certain word-lengths rather than steadily increasing as the precision of the systolic array processors is decreased. This is attributed to the fact that as data word-lengths reduce, scaling factors increase, causing data to zero out. For example $y[4]$ can typically be a factor of over 100 less than $y[1]$. The scale factor is chosen for the maximum range of Y and this may result in $y[4]$ being evaluated as zero. If $y[4]$ is zero then $\hat{a}[4] = y[4]/u[4,4]$ will be zero. $\hat{a}[4]$ being zero makes the calculation of the rest of the $\hat{A}$ values dependent on just the first three columns of the upper

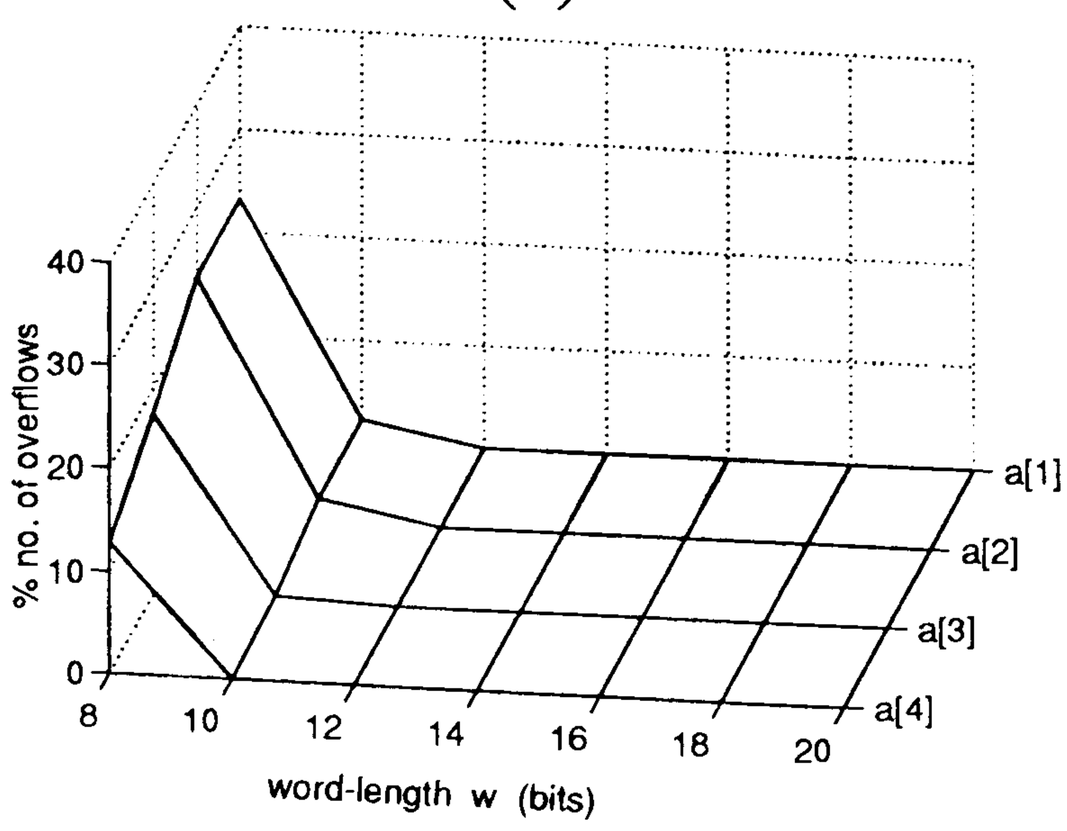
triangular matrix whose round off errors are likely to be less than that of the 4th column because they are calculated earlier in the process. Therefore the chance of an overflow error is decreased.



(a)



(b)



(c)

Figure 5.17: Percentage number of overflows in LU decomposition filter parameter results for (a) 5%, (b) 10%, (c) 20% mean frequency to bandwidth ratios.

Using the results of the error analysis (figure 5.16) the benefit curves derived from (5.1) are shown in figure 5.18 for different weights applied to W_{f_m} and W_{f_b}. The *LU* curve shows a sharp transition to increased benefit at the word-length of 16 bits at which overflow ceases. Parts (a) and (c) of the figures show very similar results demonstrating that the percentage error in f_b has the main effect on the benefit curve in (a) which puts equal weight on the determination of f_m and f_b.

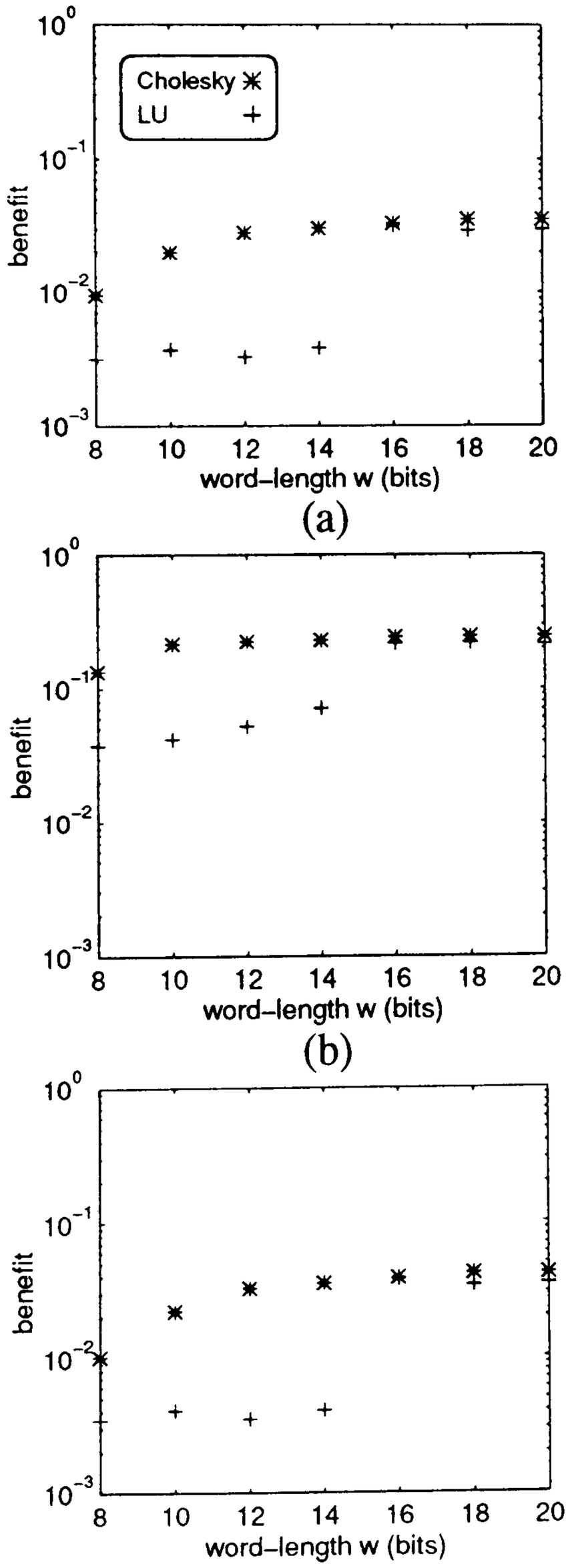


Figure 5.18: Benefit analysis for decomposition methods, (a) $W_{f_m} = 1$, $W_{f_b} = 1$, (b) $W_{f_m} = 1$, $W_{f_b} = 0$, (c) $W_{f_m} = 0$, $W_{f_b} = 1$.

5.3.10 Cost/Benefit Results

The cost/benefit criterion is simply defined as cost divided by benefit. Figure 5.19 shows cost/benefit functions run for different weights on the calculation of mean frequency (W_{f_m}) and bandwidth (W_{f_b}).

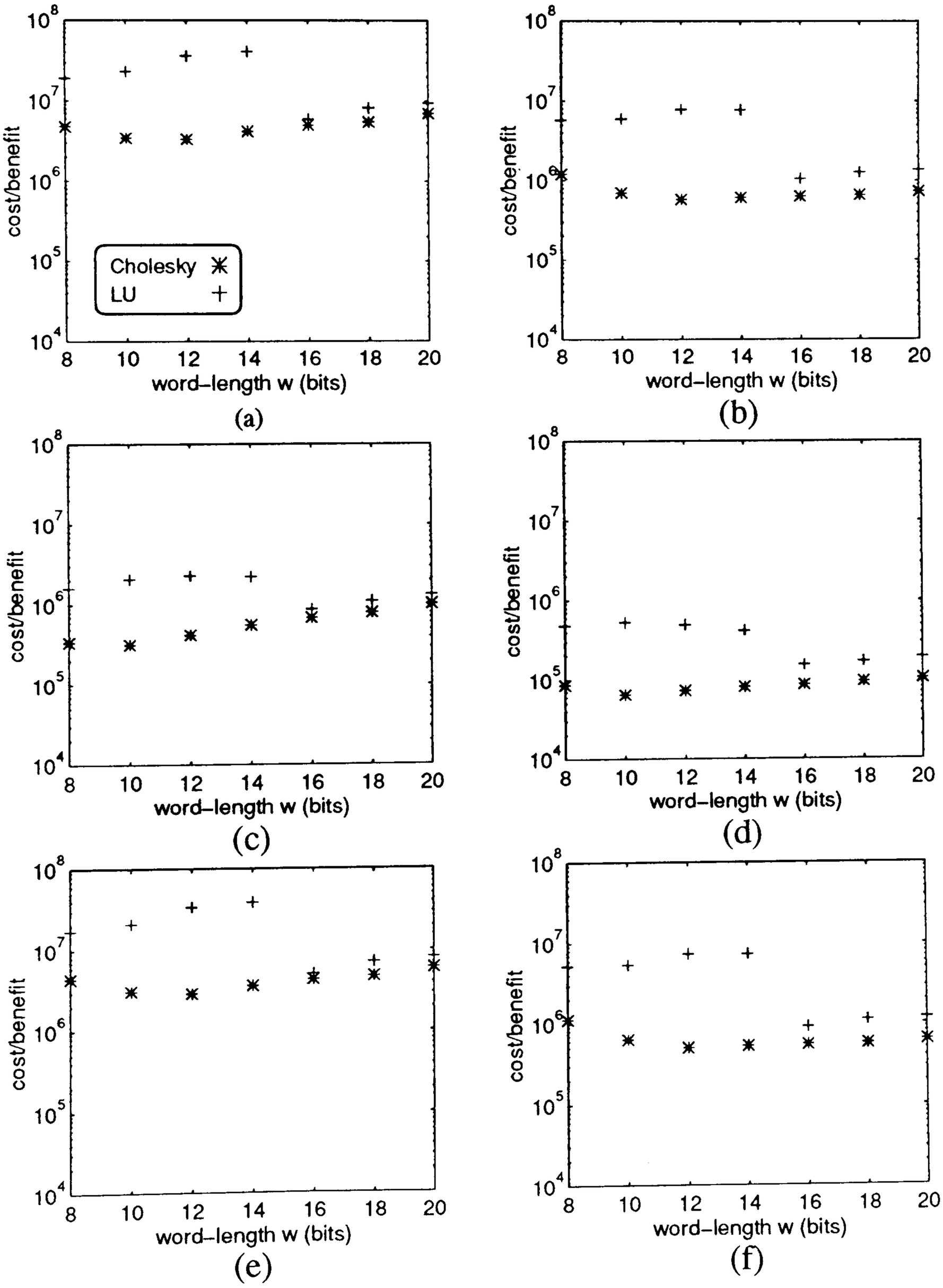


Figure 5.19: Cost/benefit analysis for decomposition methods, (a) $W_{f_m} = 1$, $W_{f_b} = 1$ word-parallel, (b) bit-serial, (c) $W_{f_m} = 1$, $W_{f_b} = 0$ word-parallel, (d) bit-serial, (e) $W_{f_m} = 0$, $W_{f_b} = 1$ word-parallel, (f) bit-serial.

The minimum values of the cost/benefit curves give optimal word-lengths. The Cholesky decomposition array is the best method since it shows the lowest minimum in each case. If f_m and f_b estimation accuracy are both considered equally important then the plots of figures 5.19(a) and (b) should be considered. The minimum of the Cholesky curve occurs at a word-length of 12 bits for both the word-parallel and bit-serial approaches. At greater word-lengths than this the computational complexity becomes so great that the cost of this outweighs the extra benefit in the increased accuracy of results. The opposite is true at word-lengths lower than 12 bits in that the reduced cost due to the decreased usage of silicon area is outweighed by the large errors apparent at these word-lengths. The LU decomposition curve follows the shape of the Cholesky curve from 16 bits upwards but at slightly higher cost/benefit levels due to the greater cost of this method. The LU's minimum occurs at 16 bits, the effect of overflow at lower word-lengths leads to rapid increase.

It may be desirable just to estimate f_m by itself. The cost/benefit curves for these cases are shown in figures 5.19(c) and (d) for word-parallel and bit-serial approaches respectively. The Cholesky method is again the best choice when the interest is in f_m only ($W_{f_m} = 1, W_{f_b} = 0$) but this time the minimum occurs at 10 bits for both approaches so less computational complexity is needed for accurate estimation of f_m alone.

The last of the plots in figures 5.19(e) and (f) are for estimation of only bandwidth ($W_{f_m} = 0$ and $W_{f_b} = 1$). Once again Cholesky is the method of choice. The curves have their minimum at 12 bits and are very similar in shape to the curves with the equal weighting in (a) and (b). Hence it can be deduced that the bandwidth calculation is the dominant factor there.

Cholesky decomposition with 12 bit word-length is therefore determined as optimal for most weight combinations with the bit-serial approach displaying the minimum cost/benefit results

5.4 Matrix Element Calculation Error Analysis

Chapter 3 concludes that the cost and performance of the PMA array, shown in figure 3.18, as optimal from the four designs proposed. Each of the matrix elements computed in a particular PE of the PMA array are sent to the same PE input port of the Cholesky decomposition array so any reordering required is a simple multiplexing task. By performing the multiply accumulate operations at double precision the accumulation error which arises from the truncation of individual products is kept as low as possible. For example a 10 bit quantised simulated Doppler signal would produce a 20 bit product. The cost/benefit analysis of the decomposition stage shows that the Cholesky process requires its input to be 10 or 12 bits and so the least significant 10 or 8 bits of the $c[j,k]$ values should be discarded.

5.4.1 Method of Error Analysis

Error analysis allows a suitable input word-length to be chosen for this estimator. The error analysis described in this section is carried out using Matlab. The input data used here is the same as that used for the decomposition error analysis. The distribution of the errors produced by 10 and 12 bit quantising of the simulated input Doppler signal in the elements of matrix C can be calculated by comparison with the high precision floating point case.

5.4.2 Results of Error Analysis

Histograms of the errors produced in the first five elements of C are shown in figure 5.20. The first 5 elements only are discussed because $c[1,1]$ and $c[2,2]$ give similar distributions to $c[0,0]$. This is also true for the pair of elements $c[2,1]$, $c[1,0]$ and also for $c[3,1]$ and $c[2,0]$. It can be deduced from this that the error

distribution is dependent upon the auto-correlation function of the covariance matrix. The histograms resemble normal distributions with zero mean. The standard deviation of each of the 12 bit distributions is approximately one quarter that of the corresponding 10 bit distributions. This pattern is repeated for every 2 bit increase in bus width.

The error in the output of the matrix element generator is ideally less than that associated with the truncation in the input words of the Cholesky module. This truncation error is equal to the scale factor introduced in the matrix element input and is either 16 or 64 depending on whether f_m or f_b estimation is required. The errors introduced into the covariance matrix elements by quantising the input Doppler signal to 10 or more bits are sufficiently low to avoid introducing significant extra error into the Cholesky decomposition stage, leading to the choice of 10 bit quantisation.

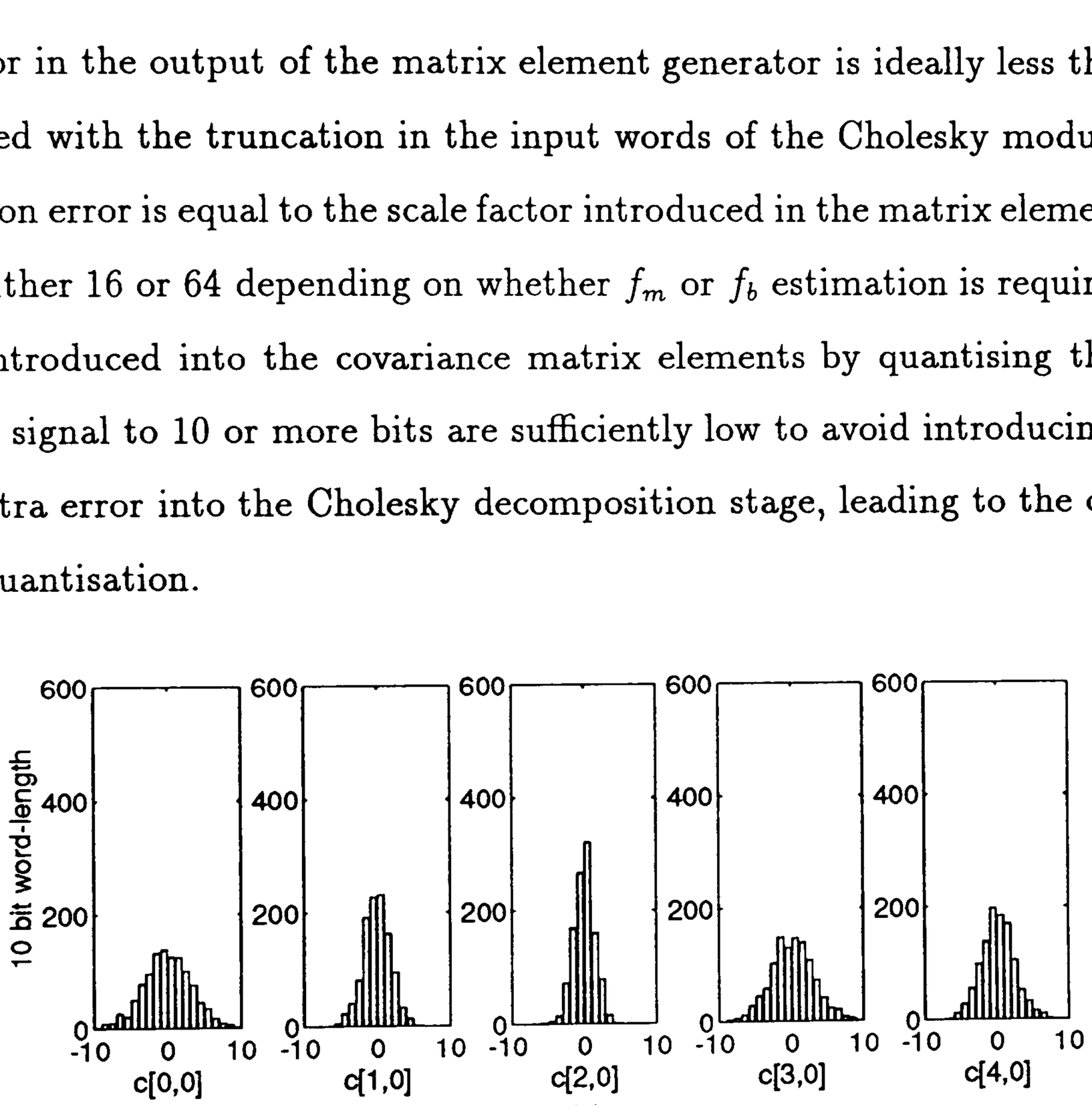


Figure 5.20: Histograms of error in matrix element calculation for (a) 10 bit and (b) 12 bit word-lengths.

5.5 Concluding Remarks

The Cholesky $[1,1,1]$ and LU $[1,0,0]$ decomposition designs for the solution of the set of linear equations were compared. For simulation and eventual implementation a pseudo floating point format scheme which introduced scale factors was developed for each of the systolic arrays. This was done to increase the dynamic range of the system over a standard fixed point scheme, without the extra cost involved in the normalisation circuitry of a floating point scheme. A cost/benefit analysis was developed to aid in the selection of the best process and its optimal word-length.

Unlike the cost analysis of the previous chapter where the hardware usage of the individual decomposition systolic arrays were compared to each other, this chapter extended the cost function to consider the effect of integrating the optimal Cholesky and LU decomposition systolic arrays into the overall system.

Data dependence graph methods were once again utilised to derive regrouping networks which were necessary to match the I/O data flows in the LU system to the output of the PMA array and input to the triangular solvers. The regrouping and reordering of data substantially increased the LU system cost above that of the integrated Cholesky system where only reordering was required. The triangular system solver arrays were redesigned to eliminate the regrouping required on the LU array output but the subsequent increase in the number of dividers in the modified back substitution array cancelled out this advantage.

Communication was also brought weighted into the cost function by considering implementation on FPGAs, but, despite the Cholesky array having a greater PE interconnection burden, the overall system cost in the LU system was still the highest in both bit-serial and word-parallel approaches.

Verilog HDL simulation was used to perform an error analysis which showed the occurrence of overflow errors in the LU architecture with bus widths up to 14 bits to have a very detrimental effect on its accuracy. The Cholesky decomposition process showed good numerical stability for all the signal cases with finite-word length rounding errors becoming significant when reducing the word-length to 8 bits.

Using the cost and error results a cost/benefit analysis was developed. Different weighting was applied to the cost/benefit criterion according to the application of the spectral estimator. For estimation of mean frequency alone a 10 bit bus width was found to be sufficient when using Cholesky decomposition. Increased precision was required for the bandwidth calculation, the optimal word-length for this calculation was determined to be 12 bits.

With regard to computation of the covariance matrix elements an error analysis based on double precision inner product accumulation was performed on the PMA array. The results of this determined that 10 bit quantisation of the sampled input Doppler signal was sufficient to supply a 10 or 12 bit word-length Cholesky systolic array without significantly affecting the previous decomposition cost/benefit analysis.

5.5.1 Overall Design of the p=4 Spectral Estimator

The systolic computation system for the fixed model order $p = 4$ spectral estimator which results from the design and analysis presented in chapters 3, 4 and 5 of this thesis is shown in figure 5.21. The entire bit-serial systolic array system hardware cost, excluding cost of control circuitry and weighted communication, is approximately 24000 NAND gates. An example for mounting the system onto four various sized Xilinx 3100 series FPGAs, given the high utilisation which is achievable when implementing bit-serial modules onto these devices, is also shown in the figure.

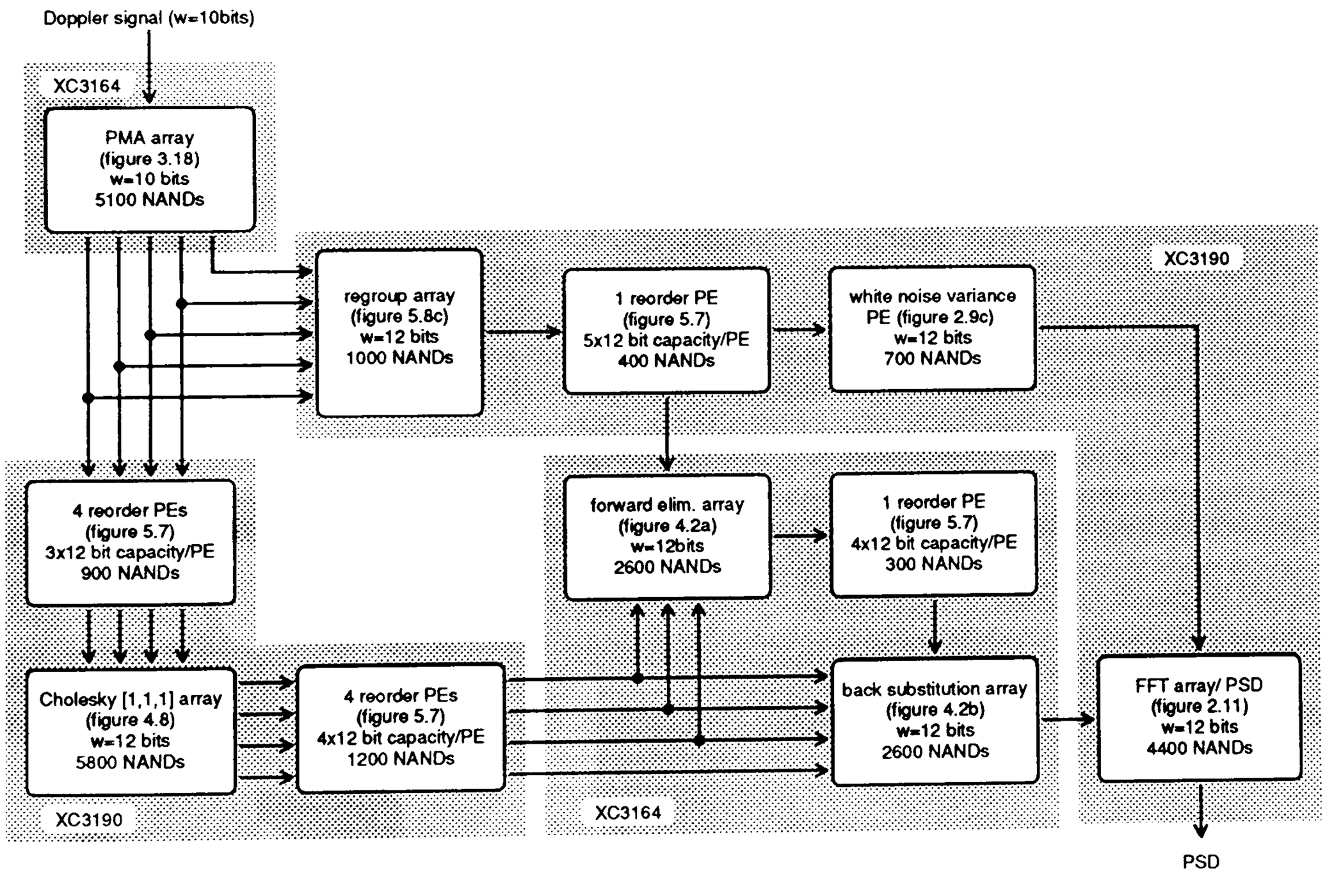


Figure 5.21: Overall systolic computation system for the fixed model order $p = 4$ Modified Covariance spectral estimator. An example for implementation on Xilinx FPGAs is given with modules lying on the same shaded backdrop to be mounted on a single device.

Chapter 6

Programmable Model Order - Matrix Element Calculation

6.1 Introduction

The optimum order of an autoregressive spectral estimator depends on the input signal and the application for which it is used. It is sometimes desirable to use adaptive spectral estimation for analysis of different properties of the blood flow other than mean frequency and bandwidth on non-stationary input signals which necessitates the need for selectable model order [130]. Too low an order gives poor spectral resolution and too high an order leads to spurious spectral peaks. A Modified Covariance spectral estimator for use with a range of input signals and for analysis of different characteristics derived from spectral shape should have a selectable model order up to $p = 30$.

The systolic arrays previously discussed are termed problem size dependent since the number of processors is dependent on the model order or dimension of the problem in hand, e.g. the PMA array of figure 3.18 requires $p+1$ PEs. Analysis, based on the input sampling frequency of the Doppler signal and typical PE bit-serial operational lags reveals that the model order PMA array, chosen as optimal for model order $p = 4$ estimation, exploits the inherent parallelism and recursiveness of the matrix element calculation to such a high extent that data

can be processed at 10 times the required maximum frequency (section 3.5). Maximum use of processing potential is therefore not being made and extension of the PMA array to $p+1$ PEs for higher model order problems, not only increases hardware costs, but also linearly increases the amount of unused processor time since, for equal word length, each PE is only required to do the same amount of work as a PE of the $p = 4$ array on each clock cycle.

Hence, there is scope to reuse the spare processor time when computing more complex problems, such as those encountered in higher model order spectral estimation, so that the size of the array in terms of the number of PEs comes down to being determined by the classic time versus area trade off. Such a route to the design of systolic arrays is termed the problem size independent approach where problems of various dimension can be mapped onto fixed sized systolic arrays. Partitioning of the algorithms for computation on systolic arrays smaller than those produced by straightforward DDG projection is essential to keep hardware costs at acceptable levels and to maximise processor potential [28][98].

This chapter builds on the design concepts introduced in the chapter 3, developing the DDGs using a partitioning technique to explore the design of feasible problem size independent systolic arrays for the matrix element calculation in a programmable model order Modified Covariance spectral estimator. This basis of the partitioning technique comes from the observation that data has to be interleaved with zeroes in arrays with bi-directional data flow, such as the tri-linear array (section 3.3.2). It is generally well known that this allows computations to be interleaved [82] and this concept is exploited here to effectively merge the operations of two or more PEs in a problem size dependent systolic array into a single PE. The problem size independent route taken in this chapter leads to the design of more efficient systolic arrays for the covariance matrix element calculation in comparison with those of the tri-linear and bi-linear systolic arrays, which remove the zero interleaving by DDG splitting (section 3.3.3).

6.2 A Comparison of Problem Size Dependent Systolic Array Costs for High Model Order Problems

In chapter 3 the PMA design was chosen as optimal for use in the matrix element calculation in the model order $p = 4$ spectral estimator. However the costs of implementing this and the other arrays presented at higher model order were not discussed.

The problem size dependent systolic array approach of chapter 3 can be extended to design fixed model order arrays for higher model order problems. This allows the costs of high model order problem size dependent systolic arrays to be compared and to reveal whether or not the PMA type array remains as optimal. The reason for making this comparison is that the DDG behind the most cost efficient high model order problem size dependent array is likely to yield the best problem size independent systolic array when further partitioning is discussed. The relative costs of the tri-linear, bi-linear and PMA type arrays derived in chapter 3 and used for the calculation of the matrix elements in high model order Modified Covariance estimation are therefore discussed in this section. The 2-dimensional array is not given any further consideration due to its relatively poor initial performance in the model order $p = 4$ problem.

6.2.1 Multi-linear Array Method

The ideas used in the design of the tri-linear systolic array can be extended to higher model orders. For a model order of p an equivalent DDG design process would result in $\frac{p}{2} + 1$ linear systolic arrays ranging in length from $p + 1$ PEs to a single PE. High model order systolic arrays designed using this method are therefore referred to as multi-linear and in total, $(\frac{p}{2}+1)^2$ multiplier PEs would be required. As the model order of such a system is increased the deficiencies associated with this type of design become more pronounced. The main problem with

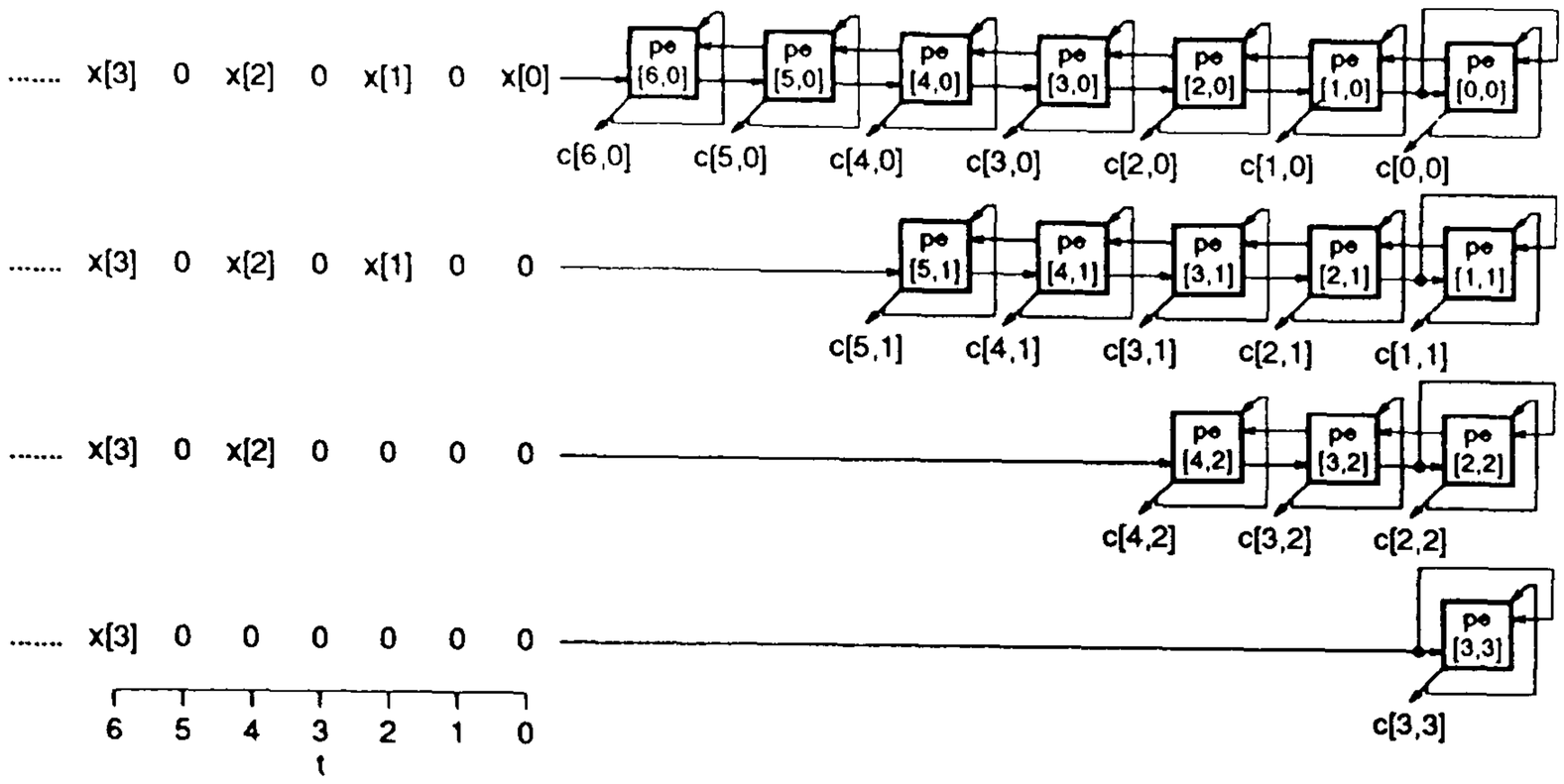


Figure 6.1: Problem size dependent systolic array for model order $p = 6$ multi-linear array based on the design of the $p = 4$ tri-linear array in section 3.3.2.

this design is that of repeated multiplications. As the model order is increased by two an extra linear array containing $p + 1$ PEs is added. The multiplications performed in this array are also computed in the shorter linear arrays and so efficiency is further decreased. Figure 6.1 illustrates the multi-linear systolic array for the $p = 6$ estimation problem. 7 extra PEs are required in comparison with the tri-linear array of figure 3.11, the overall design therefore requiring a total of 16 PEs. Multiplications performed in the 1, 3, and 5 element systolic arrays are all repeated in the 7 PE linear systolic array.

6.2.2 Bi-linear Array Method

Using the bi-linear design method for the general model order p still results in the formation of two linear systolic arrays. Their lengths are $\frac{p}{2} + 1$ and $\frac{p}{2}$ PEs giving a total of $p + 1$ PEs so two extra PEs are required for a $p = 6$ bi-linear design over the $p = 4$ version (figure 3.13). However, in this approach the size of the sorting PE is proportional to the model order as a greater number of words need to be stored, requiring $2p + 3$ double precision registers in total. For example the model order $p = 4$ sorter PE requires 11 registers (figure 3.14) compared to a total of 15 in the $p = 6$ sorter PE (figure 6.2). Since these $2p + 3$ registers are included in each of the $p+1$ PEs of the systolic array then a p^2 term is introduced

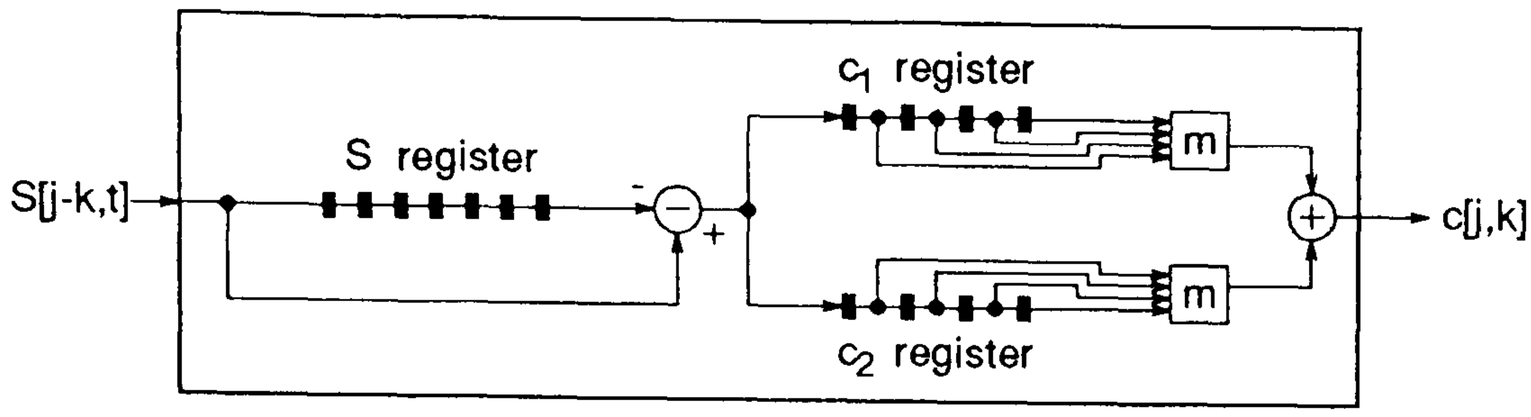


Figure 6.2: Sorting PE design for the $p = 6$ bi-linear array, where a total of $2p + 3 = 15$ double precision word registers are required.

into the cost function and dominates at higher model orders, again leading to a design which is expensive to implement.

6.2.3 PMA Array Method

Partitioning the high model order problem into separate multiplication and accumulation DDGs also results in two linear systolic arrays of length $\frac{p}{2} + 1$ and $\frac{p}{2}$ PEs similar to the bi-linear design. The processor architecture differs however in that number of double precision registers in the accumulator feedback loop is $\frac{p}{2} + 1$ and this also brings the p^2 term into the cost function. The clock speed required in the accumulator feedback registers also is dependent on the model order and is increased to $(\frac{p}{2} + 1).f_s$.

6.2.4 Cost Comparison

The relative costs of the multi-linear, bi-linear and PMA systolic array designs can be compared for various models orders while keeping the word-length constant. Table 6.1 details the cost of these arrays as a function of the model order p. The information gathered from the table is used in conjunction with the data collected in tables 3.1 and 3.2 to produce the plots of cost versus model order for a fixed word length of $w = 16$ bits presented in figure 6.3. A longer word-length is used as high model order systems require greater accuracy and longer word-length due to ill-conditioning tendencies of the decomposition processes.

w bit module	number of modules used in the systolic arrays		
	multi-linear	bi-linear	PMA
addition (d)	$(p/2+1)^2$	$2(p+1)$	$(p+1)$
subtraction (d)	0	$(p+1)$	0
multiplication	$(p/2+1)^2$	$(p+1)$	$(p+1)$
register	$2(p/2+1)^2$	$2(p+1)$	$2(p+1)$
register (d)	$(p/2+1)^2$	$(2p+4)(p+1)$	$(p/2+2)(p+1)$
multiplex	$p/2$	0	0
multiplex (d)	$(p/2+1)^2$	$p(p+1)$	$2(p+1)$

Table 6.1: Hardware usage for systolic arrays in terms of module usage.

The high costs incurred by the problem size dependent arrays (figure 6.3) may be further increased due to the requirement of longer word-length, if the large order covariance linear system of equations becomes ill-conditioned. Partitioning onto problem size independent systolic arrays can be used to lower the costs below those of the problem size dependent systolic arrays and thus improve the overall efficiency of the array. In the word-parallel approach the bi-linear design displays lower cost than the multi-linear design while the opposite is true in the bit-serial implementation. The problem size dependent PMA arrays display substantially lower costs in both bit-serial and word-parallel approaches than either of the other two types of array throughout the range of model order. It therefore seems reasonable to assume that further partitioning of the PMA DDGs produces the most effective problem size independent systolic arrays and optimisation of this design is described in the remainder of this chapter.

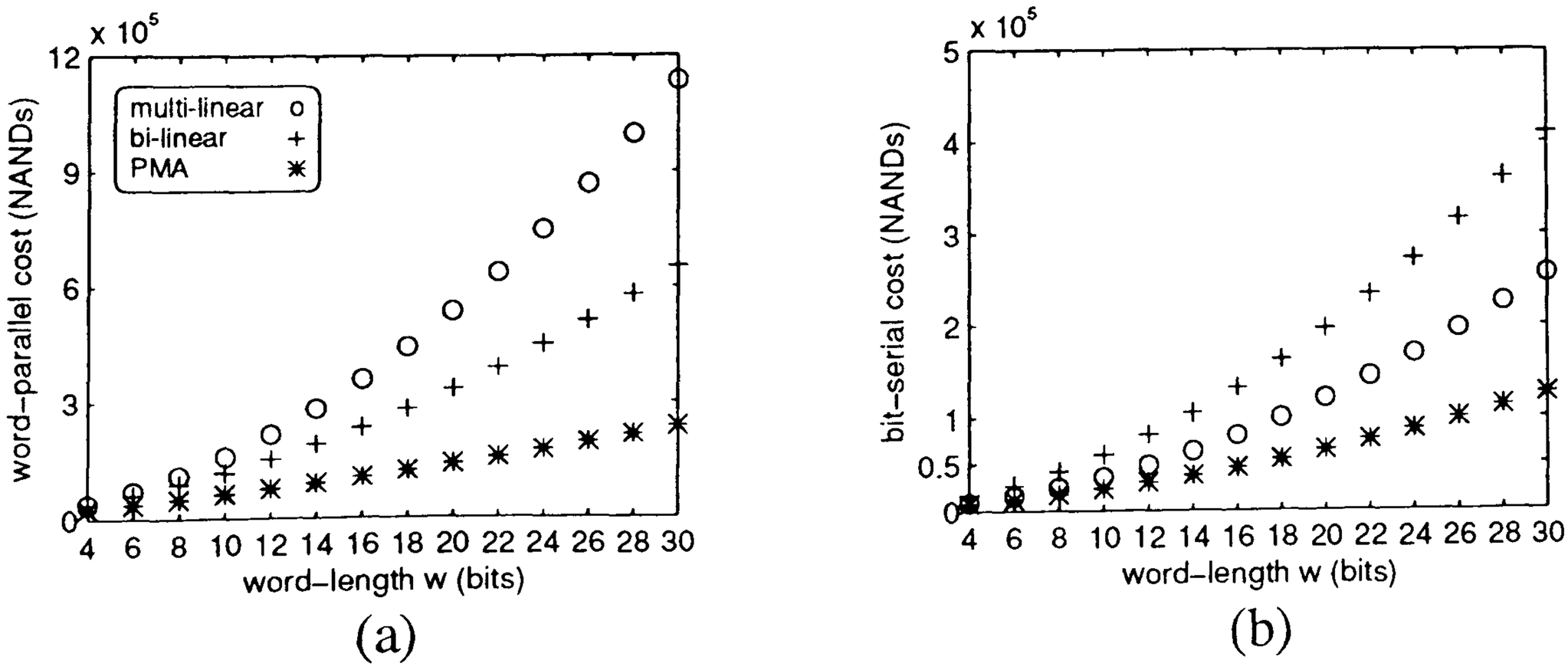


Figure 6.3: Cost for (a) word parallel and (b) bit-serial approaches in terms of model order p for a fixed word-length of $w = 16$ bits.

6.3 High Model Order PMA Array Redesign

The above discussion shows that it is not feasible to continue extending the size of the PMA type systolic array proposed for the matrix element calculation when considering the solution of high model order problems. A partitioning scheme is therefore necessary to maximise the potential of the array PEs.

6.3.1 Partitioning the $p = 6$ Multiplication DDG

When using the PMA method the multiplication processes are initially partitioned from the accumulations. Further partitioning of the multiplication DDG is considered in this section by looking at the $p = 6$ matrix element calculation.

The DDG drawn in figure 6.4 performs the multiplications necessary for the matrix element calculation in the $p = 6$ Modified Covariance spectral estimator. Input data $x[n]$ and $x[n-(j-k)]$ are respectively globally transmitted along columns and diagonals of the graph. Each node forms and stores internally the product of these two x inputs. The graph could be partitioned using the split DDG initially introduced for the design of the bi-linear array in section 3.3.3. Projection in the n direction would then produce 2 linear systolic arrays of length 3 and 4 PEs. A different approach to partitioning is considered here however.

The approach taken here is to group together rows of the DDG and then consider the mapping of each group of rows into a single PE. The DDG shown in figure 6.4 is partitioned into 4 groups, each containing 2 rows. The partitioning is indicated by the dashed horizontal lines across the graph and the size of the partition is referred to as s, i.e. $s = 2$. An extra row of nodes with dotted circumference are added to the bottom of the DDG in the $(j-k) = 7$ position to fill out the last partition. These are don't care operations and may be programmed as desired.

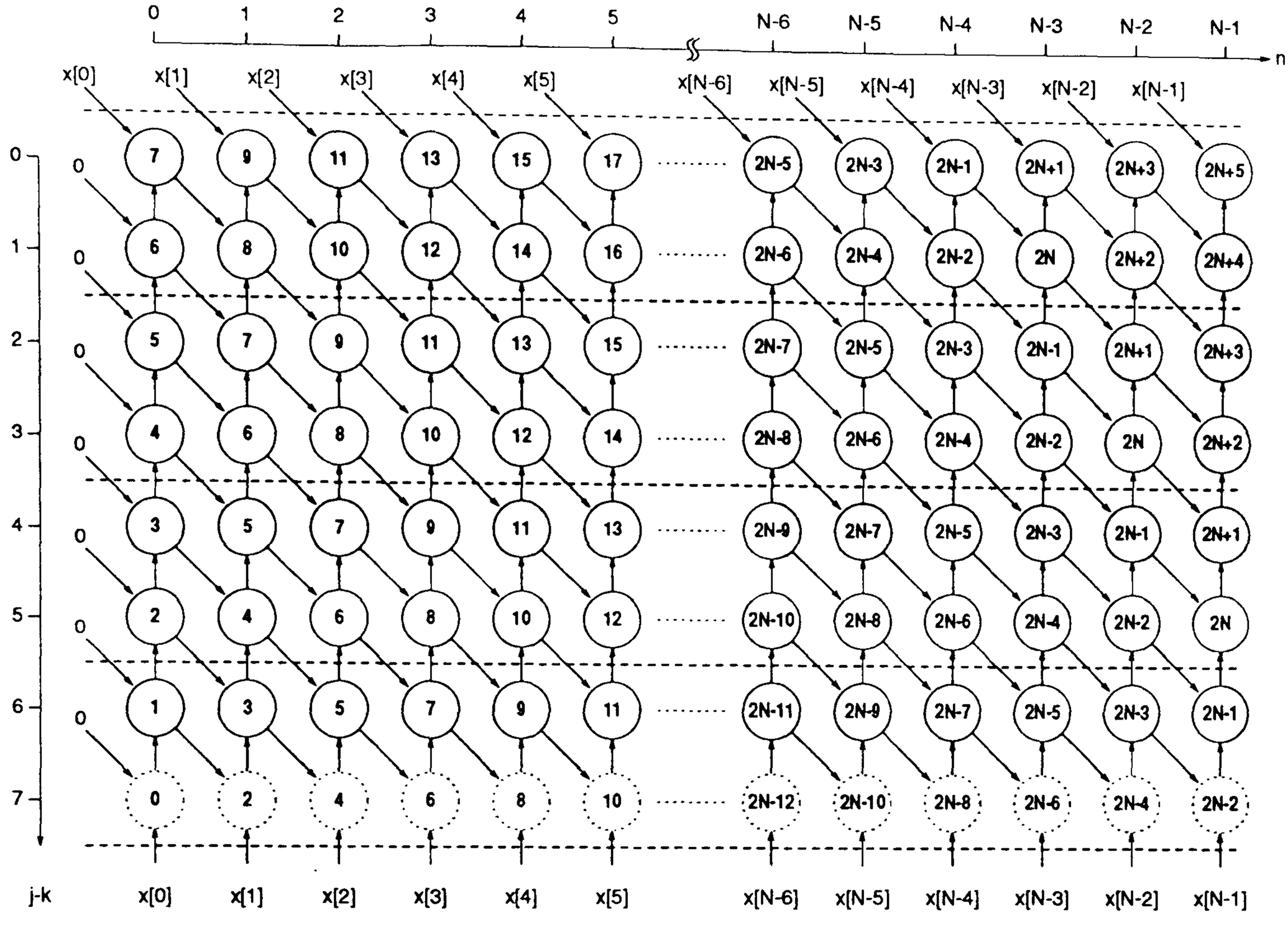


Figure 6.4: DDG for multiplications in the $p = 6$ matrix element calculation.

The timing function applied to the DDG must take into account that any node within a partition should be processed within its own exclusive clock cycle since all nodes within a specific partition are mapped into a single PE. The task scheduling, based on a systolic clock running at $2f_s$, indicated by the internal node numbering is:

$$t = p - (j - k) + s.n + s - rem\left(\frac{p+1}{s}\right) \tag{6.1}$$

where rem is the remainder function used in the calculation of the number of extra rows used to fill the last partition which in this case is 1. When considering the DDG projection into a systolic array the DDG of figure 6.4 can be separated into s separate graphs, the first made up from rows $(j - k) = 0,\ s,\ 2s, ...$, the second from rows $(j - k) = 1,\ 1 + s,\ 1 + 2s, ...$ and so on. The $s = 2$ partition therefore produces two graphs the first made from the even rows and the other graph the odd rows as illustrated in figure 6.5.

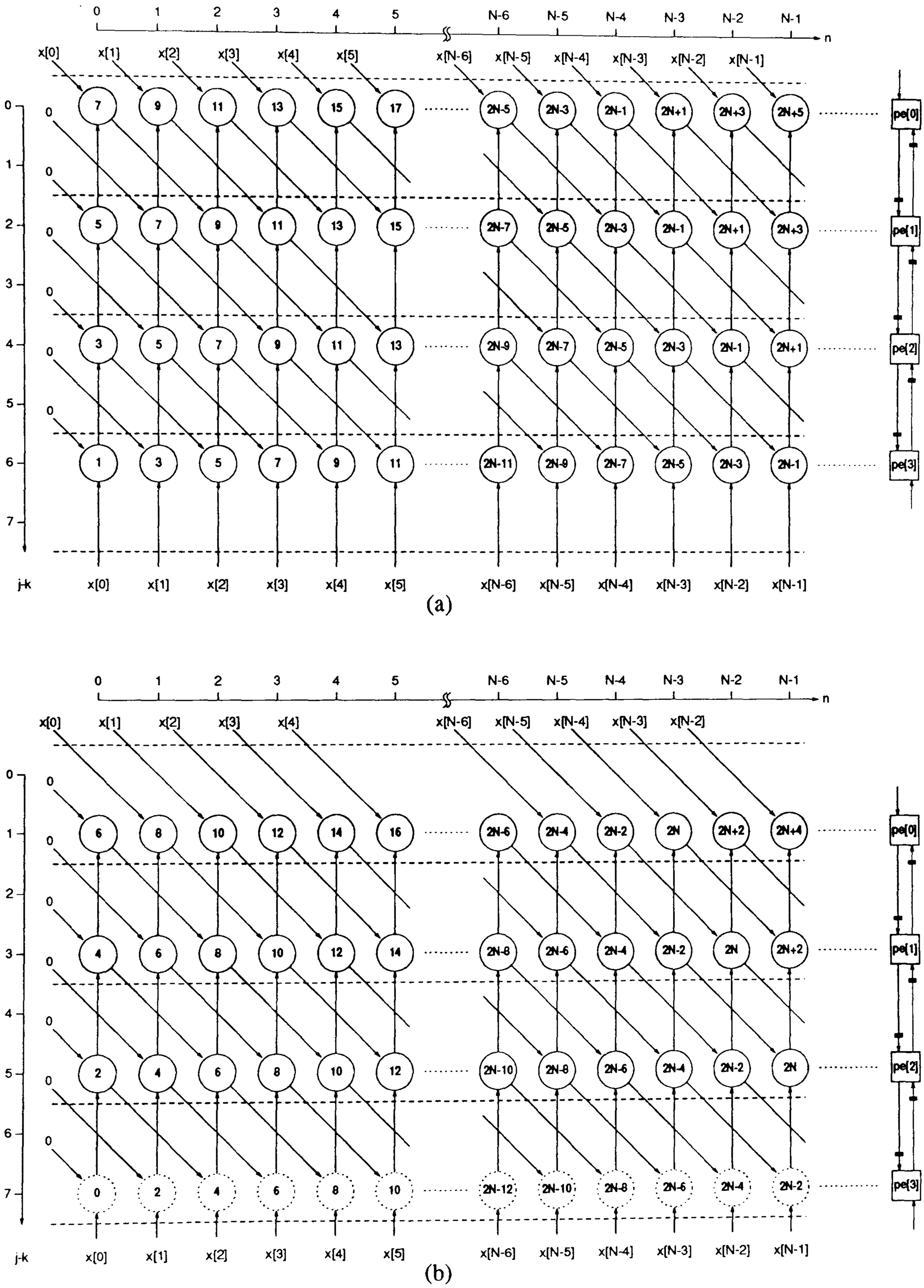


Figure 6.5: Partitions of the $p = 6$ DDG using $s = 2$ (a) (j-k) even, (b) (j-k) odd.

(A) Systolic Array Formed by Projection of the Partitioned Multiplication DDG

Projection of the separated DDGs in figure 6.5 results in linear systolic arrays of length 4 PEs. Due to the timing function two systolic delays are required on each PE interconnection bus. Since there are already unit systolic word delays internally on the PE inputs (figure 3.17 shows the PE layout) then one extra register is needed on each interconnection bus. The construction of each of the two arrays is exactly the same, the only difference is in the timing of the input data. The external inputs $x[n]$ to $pe[0]$ and $pe[3]$ are scheduled at $t = 2n + 7$ and $t = 2n + 1$ respectively for the array in part (a) of the figure. For the systolic array in figure 6.5(b) $x[n]$ is input to $pe[0]$ at $t = 2n + 8$ and to $pe[3]$ at $t = 2n$. Each array processes data on alterate clock cycles to the other and therefore they can be merged into a single array whose input scheduling is derived by combining the input timings of the constituent arrays. The clock frequency of the array is $s.f_s = 2f_s$ since the data input changes once in every two clock cycles, e.g. $x[0]$ is clocked into $pe[3]$ at $t = 0$, $t = 1$ and $x[1]$ is input on the next two cycles.

(B) Combining the Partitioned Multiplication and Accumulation Systolic Arrays

A total of $p + 1 = 7$ accumulation DDGs can be derived for the $p = 6$ matrix element using a similar procedure as in the formation of the DDGs for the $p = 4$ problem in figure 3.15. The DDG for the calculation of $c[0, 0]$, $c[1, 1]$, $c[2, 2]$ and $c[3, 3]$ is formed from 4 rows and therefore the multi-projection of this graph results in a systolic PE which has 4 word delays in the feedback loop. Now since each multiplication PE alternately outputs products relating to two of the accumulation DDGs then the number of word delays in the accumulator feedback loop can be extended to 8. The clock frequency of these feedback registers is $(\frac{p}{2} + 1).s.f_s = 8f_s$. The merged PMA array for the $p = 6$ matrix element calculation partitioned using $s = 2$ is shown in figure 6.6.

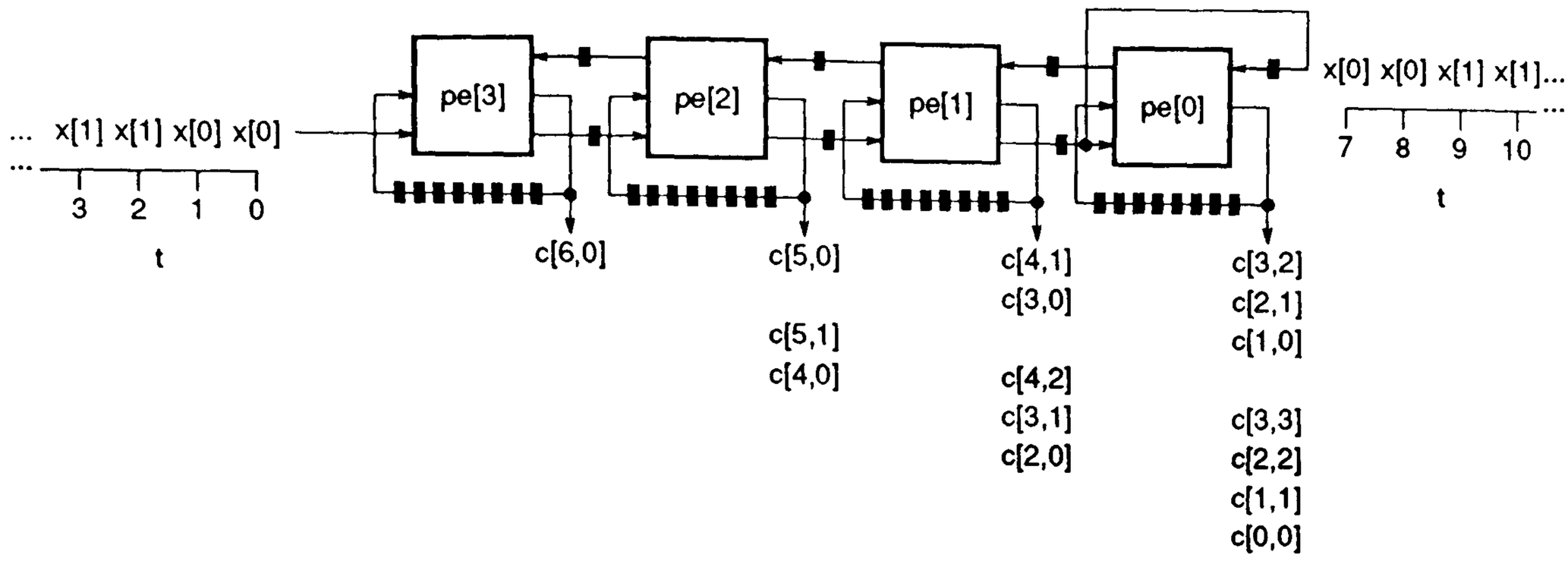


Figure 6.6: (a) PMA systolic array for computation of covariance matrix elements in the $p = 6$ estimator using partitions of size $s = 2$ (feedback registers clocked at $8f_s$, inter-connection and internal PE registers clocked at $2f_s$). For PE configuration see figure 3.18(b).

6.3.2 Partitioning the $p = 6$ Multiplication DDG using $s = 3$

Bigger partitions can be applied to the multiplication DDG to increase the level of pipelining in the systolic array while reducing the degree of parallelism. Partitioning the DDG into groups of $s = 3$ rows is shown in figure 6.7. The timing function derived from (6.1) allows each node within a particular partition to operate on an exclusive clock cycle. The DDG can be split into 3 separate graphs to indicate three phases of operation. The first graph contains rows with $(j - k)$ coordinates 0, 3, 6, the second has rows $(j - k)$ =1, 4, 7 and the last graph is constructed from rows $(j - k) = 2$, 5 and 8. Each can be projected into a systolic array with pipeline word delays of 3 in the inter-PE connections. The three arrays can be merged into one array whose overall input scheduling may be derived from the information given in the three separate DDGs. The clock frequency of this array is $3f_s$. The total number of systolic word registers in the accumulation feedback loop is increased to 12 and these registers must be clocked at $12f_s$. The systolic array produced by the $s = 3$ partitioning of the $p = 6$ matrix element calculation is shown in figure 6.8. The input stream to the right hand side can be derived from the $x[n]$ data travelling from left to right using a multiplexor to select between different register outputs of this bus.

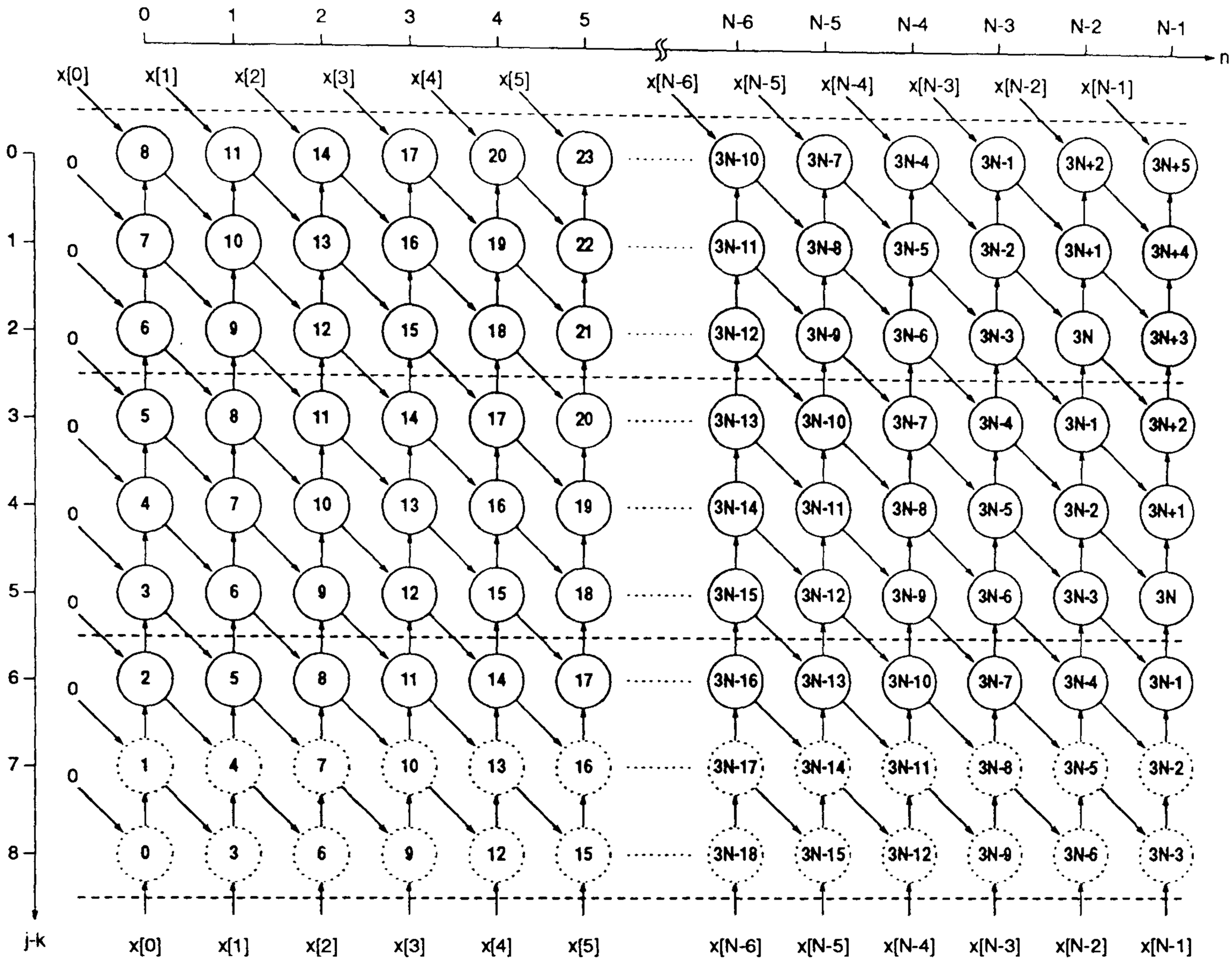


Figure 6.7: DDG for multiplications in the $p = 6$ matrix element calculation partitioned using $s = 3$ with its timing function derived from (6.1).

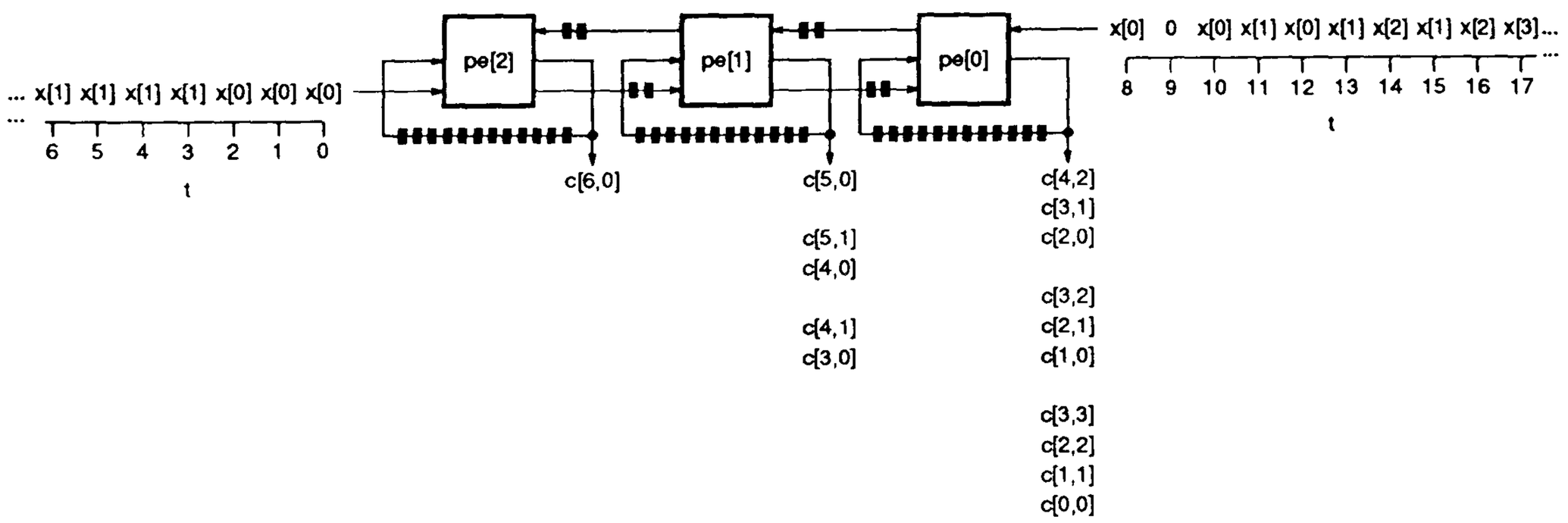


Figure 6.8: (a) PMA systolic array for computation of covariance matrix elements in the $p = 6$ estimator using partitions of size $s = 3$ (feedback registers clocked at $12f_s$, inter-connection and internal PE registers clocked at $3f_s$). For PE configuration see figure 3.18(b).

6.4 Selection of the Optimal Partition

6.4.1 Hardware Cost Analysis

The PMA array partitioning techniques demonstrated in the previous section can be applied to a range of different model order problems. The cost function in table 6.2 details the cost of the average PE in a model order p PMA type array with partition size s.

w bit module	number of modules used		
	accumulation PE	multiplication PE	Merged PE
addition (d)	1	0	1
multiplication	0	1	1
register	0	$2s$	$2s$
register (d)	$s.p/2 + s + 1$	0	$s.p/2 + s + 1$
multiplex (d)	2	0	2

Table 6.2: Hardware usage for PMA array PE in terms of module usage.

The total cost of the array in terms of modules is calculated from the product of the merged PE cost with the number of PEs in the array which is given by:

$$N_{PE} = ceil\left(\frac{p+1}{s}\right) \tag{6.2}$$

where the *ceil* function is used to round to the nearest integer towards infinity. The NAND gate usage is obtained in conjunction with the information used to derive the cost functions in chapter 3.

The results of the cost analysis versus partition size s are shown in figure 6.9 for model orders p from 4 to 30 in steps of 2 and for both word-parallel and bit-serial approaches. The results are based on a longer word-length of 16 bits which allows greater accuracy for the determination of the more ill-conditioned matrices likely to occur at high model order. At the small partition sizes the cost of the multiplication dominates in the word-parallel approach more so than in the bit-serial approach. This is due to the large number of PEs required from

mapping using small sized partitions. For example when $s = 1$ a total of 31 PEs are needed for a $p = 30$ problem. As the size of the partition increases then the number of PEs decreases and the dominant cost is that of the systolic word registers. Since the overall number of these registers is the same for both types of approach then the curves for word-parallel and bit-serial approaches converge to each other as the partition size is increased.

On each of the plots in figure 6.9 a number of localised maxima and minima occur. For example on both of the $p = 30$ plots minima occur at s =8, 11, 16 and 31 and maxima occur at s =10, 15, 30. Within the intervals from s =8 to 10, 11 to 15 and 16 to 30 the PMA array produced by the partitions consists of 4, 3 and 2 PEs respectively. The reason for increase in cost with s in these localised regions is due to processor redundancy. The DDGs in figures 6.4 and 6.7 show the use of extra rows of don't care nodes to fill out the partitions. To maximise efficiency when mapping into a certain number of PEs the number of extra rows of don't care nodes should be kept to minimum as in the case of the s =8, 11, 16 and 31 partitions of the $p = 30$ curves. To demonstrate this when mapping into 4 PEs, partitions of size s =8, 9 and 10 produce DDGs with 1, 5 and 9 rows of the don't care nodes, leading to selection of the s =8 partition. The larger partitions at

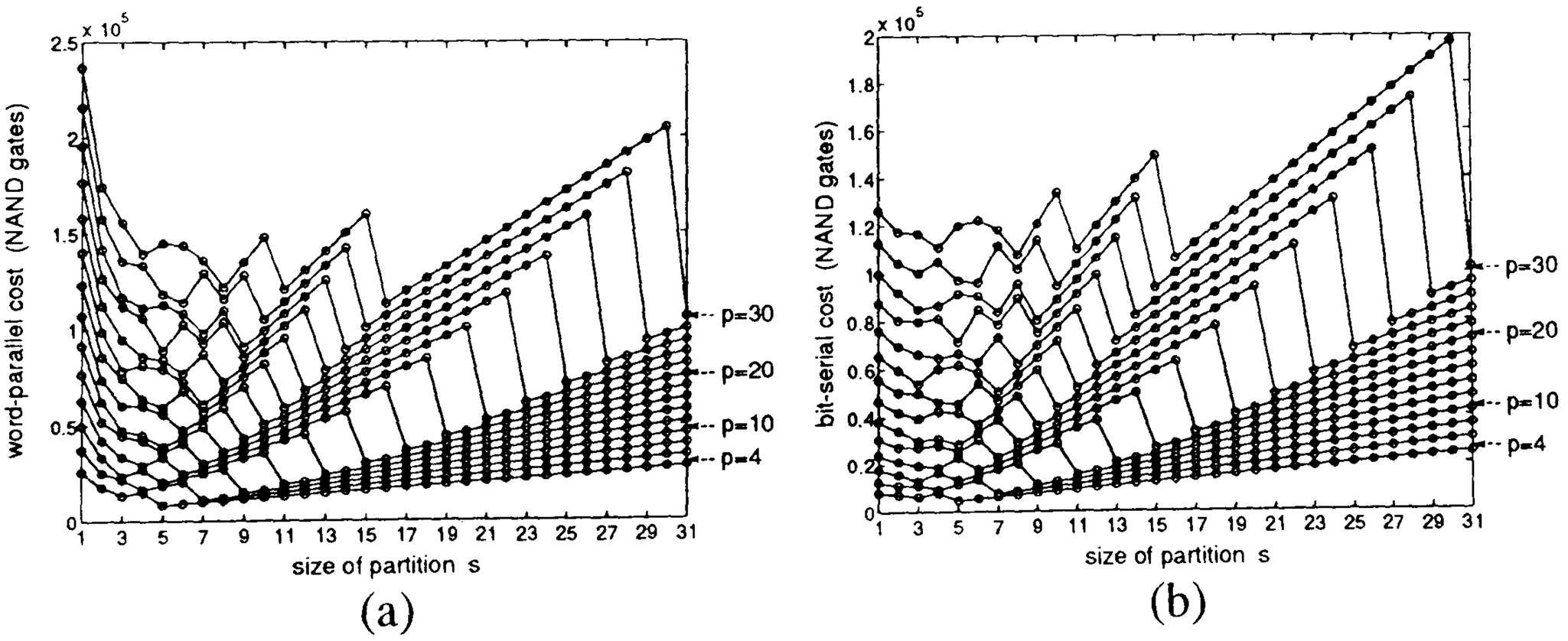


Figure 6.9: Cost for (a) word parallel and (b) bit-serial approaches versus partition size s for various model order p (lower curve corresponds to $p = 4$ and for each consecutive curve moving up the graphs p is increased by 2 until the top curve where $p = 30$) for a fixed word-length of $w = 16$ bits.

s =9 and 10, which display greater cost due to the number of PEs remaining the same and the fact that the number of systolic word registers is proportional to partition size as detailed in table 6.2, should therefore be avoided. The partition sizes from $s = 12$ to 15, 17 to 30 should likewise be disregarded. In the bit-serial approach when using a $s = 30$ partition in the $p = 30$ problem the total cost turns out to be much higher than the array cost without partitioning, i.e. when $s = 1$, and so there would be no point in perusing such a scheme. Note the sharp decreases in cost when moving from maximum to minimum points which are caused by reduction in the number of systolic arrays PEs.

The graphs of figure 6.9 with the inefficient partitions removed are shown in figure 6.10 to show just the optimal partition sizes when mapping into a number of PEs. In general the cost tends to decrease with bigger partition sizes, but there are some exceptions to this for a few of the model orders. For example, in the $p = 30$ implementations the number of systolic registers in the $s = 5$ partitioned array which contains 7 PEs is greater than for the 8 PE array produced by the $s = 4$ partition because there are 4 rows of don't care nodes in the DDG of the former as opposed to 1 row in that of the latter. The don't care nodes lead to the end PE of the array carrying redundant registers resulting in poorer cost efficiency.

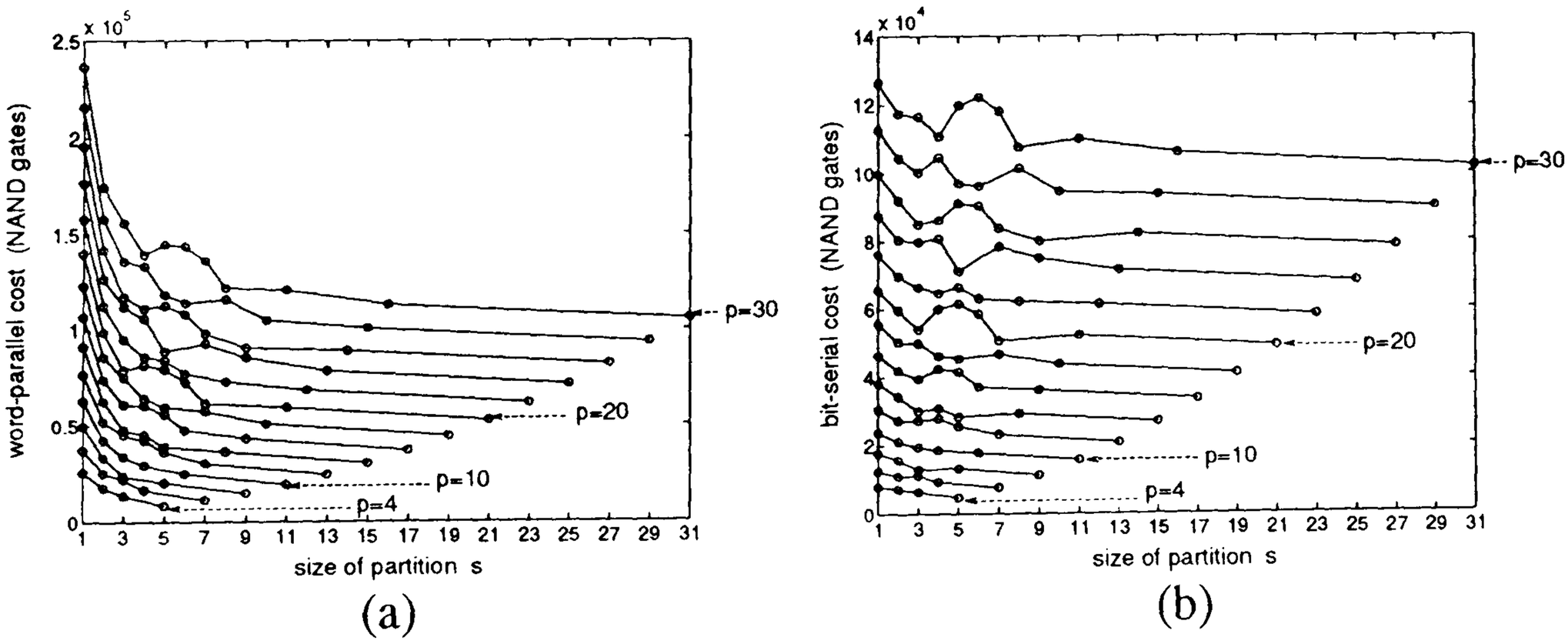


Figure 6.10: Cost for (a) word parallel and (b) bit-serial approaches versus partition size s for various model order p with conditions the same as in figure 6.9. Plot lines are used to connect between the optimal partition size points of problems with equal model order and do not refer to costs of the inefficient partitions.

A different way of viewing the cost is in terms of the number of PEs as shown in figure 6.11. The general trend of the curves shows a decrease in cost for the smaller arrays. The steeper gradients of the word-parallel curves show that partitioning is more beneficial in this approach than in the bit-serial equivalents where the effect is very subtle. The points on each of the curves indicate where efficient mapping into a certain number of PEs is possible. So for example, all of the model orders can be efficiently mapped into 1 or 2 PEs whereas an array containing 5 PEs can only efficiently process model orders $p = 4$, 8, 12, 14, 16, 18, 20, 22, 24, 26, 28 and 30 problems. However note that problems of model order $p = 6$ and 10 may still be performed on a PMA array of 5 PEs but with less cost effectiveness.

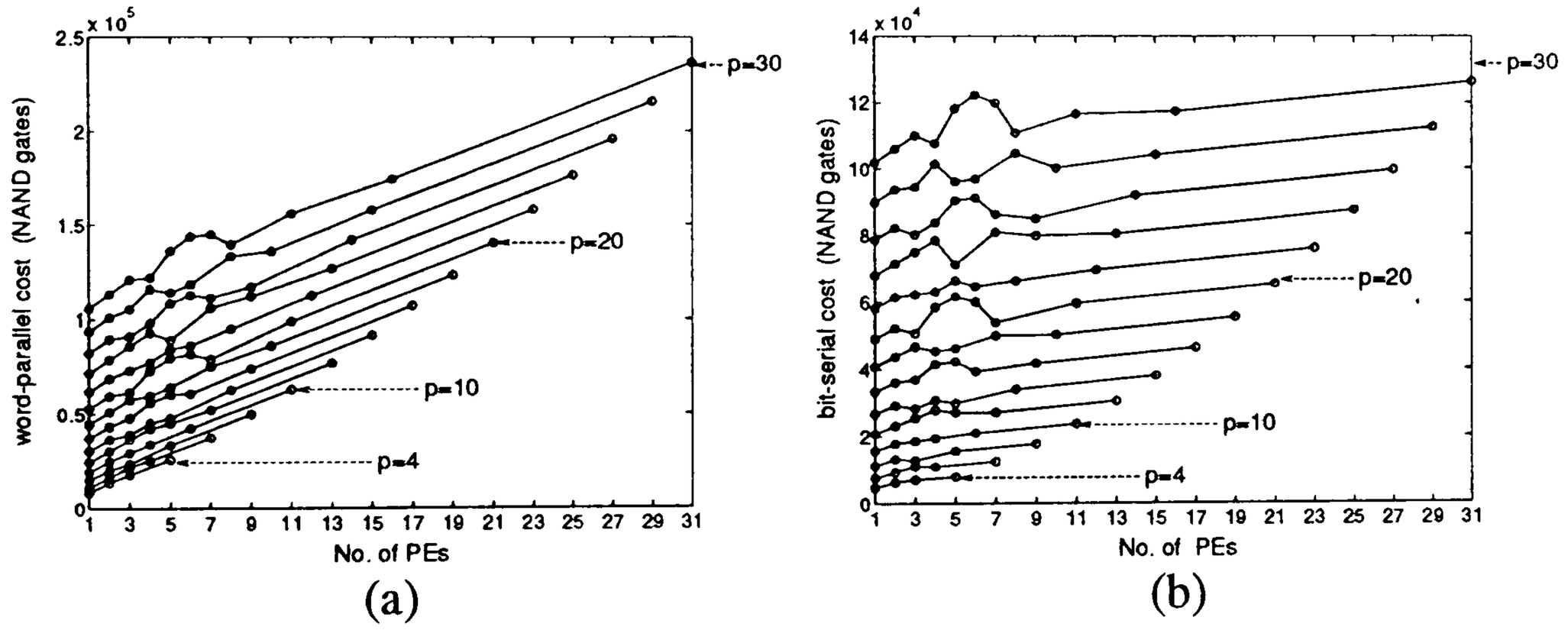


Figure 6.11: Cost for (a) word parallel and (b) bit-serial approaches versus number of PEs for various even model order p from 4 to 30. Only the optimal partition sizes, indicated by the points, are included as in figure 6.10

6.4.2 Timing Considerations

One of the effects of partitioning the PMA design is to increase the number of clock cycles required for complete solution. The total number of clock cycles for the multiplication operations can be derived with the use of (6.1) given that whatever the model order or size of partition the last multiplication takes place in the node with coordinates $(j - k) = 0$, $n = N - 1$ as can be seen in figure 6.4.

This leads to the timing function:

$$t_{total} = p + s.N - rem\left(\frac{p+1}{s}\right) \tag{6.3}$$

As with the hardware cost functions the execution time can be examined as a function of partition size as in figure 6.12. The input data window length used is $N = 256$ and all even model orders from $p = 4$ to 30 in steps of 2 are included. The product $s.N$ dominates the shape of the plots and overlap makes it difficult to distinguish one model order curve from another. Hence there is a close to linear relationship between the increase in the total number number of clock cycles and the size of partition when the product $s.N \gg p$ as is the case here.

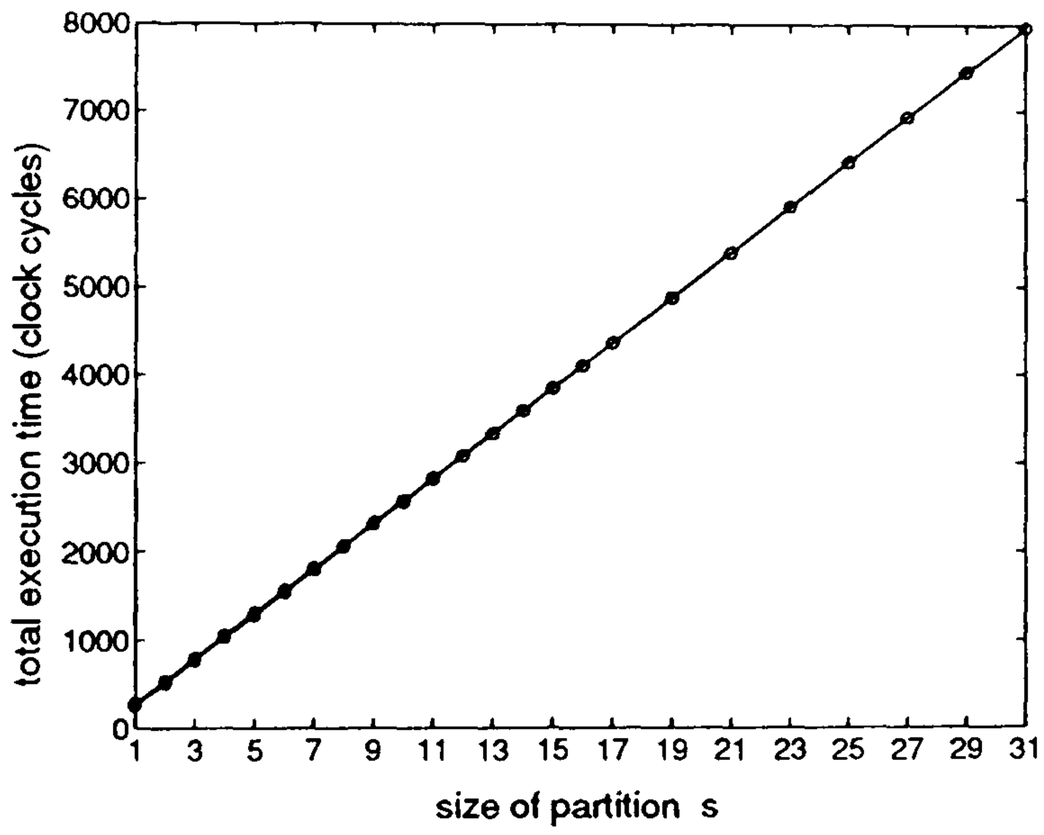


Figure 6.12: Total execution time for optimal PMA array mappings of even model orders from $p = 4$ to 30 versus partition size s.

As with the cost analysis, the results can also be viewed in terms of the number of PEs as illustrated by figure 6.13. The plot clearly demonstrates the disadvantage of mapping into small systolic arrays as massive increases in the number of clock cycles required are observed when reducing the size of the array to a few PEs. If the time length of the data window $\frac{N}{f_s}$ and model order p remain fixed when considering mapping into PMA arrays of various length the allowable PE latency becomes less for smaller arrays. Therefore smaller systolic arrays require highly specified PE arithmetic modules.

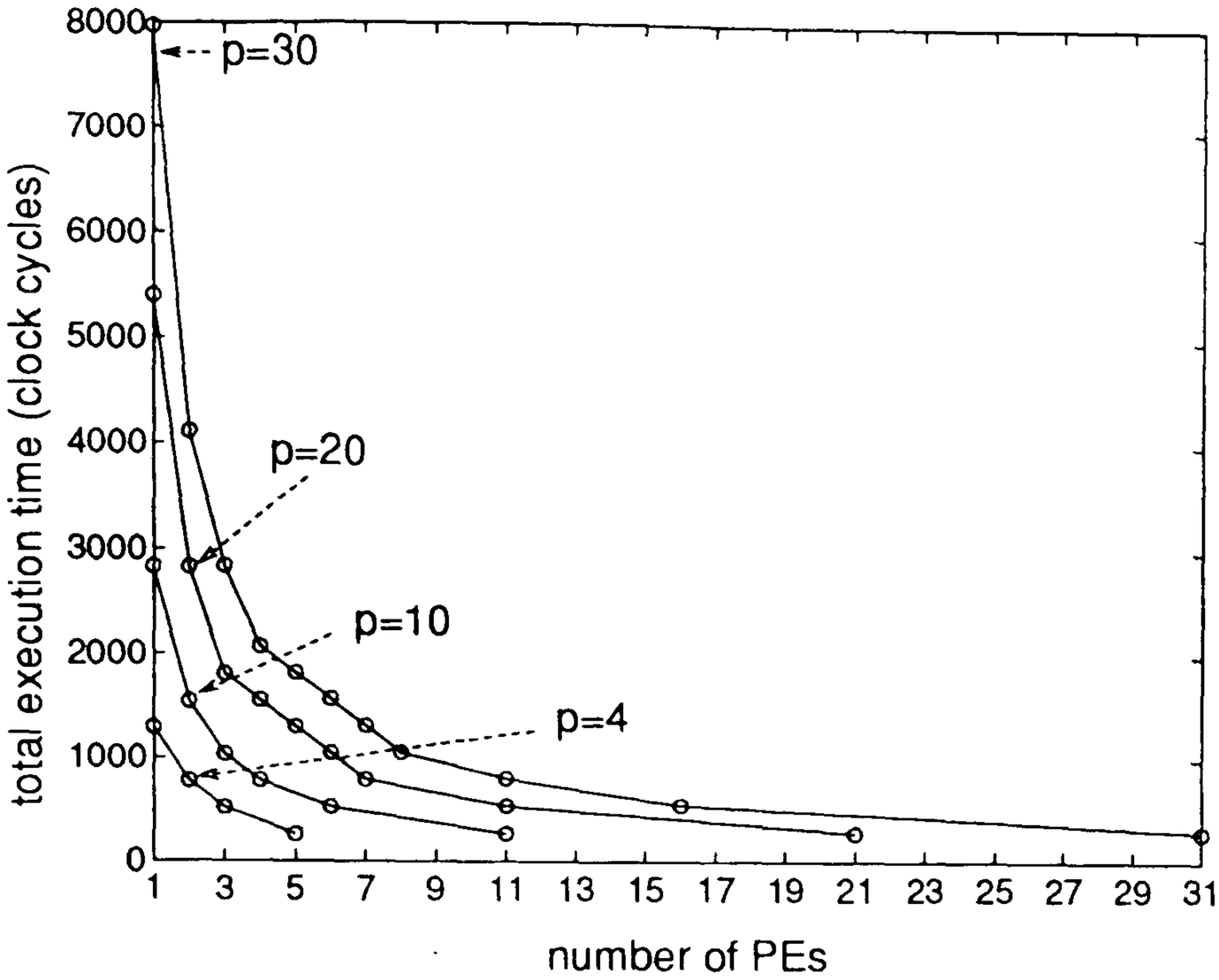


Figure 6.13: Total execution time for optimal PMA array mappings of model orders $p = 4$, 10, 20 and 30 versus number of PEs.

The performance requirement of a processor can be determined by considering the number of operations required within the sampling period of the input data $T_s = \frac{1}{f_s}$ for a given sized partition. The DDGs in figures 6.4 and 6.7 show that the number of multiplications required within T_s is equal to the partition size s when the sampled input data considered is the input $x[n]$ to the lower row of the graph. Hence if register propagation time is assumed negligible, multiplication operation within the PEs of the PMA array is limited to:

$$T_{mul} < \frac{T_s}{s} \tag{6.4}$$

or

$$T_{mul} < \frac{1}{s.f_s} \tag{6.5}$$

The accumulation operations are pipelined with the multiplication operation and need to be clocked at a higher rate than the multiplication array systolic frequency. Addition takes less time than multiplication but nevertheless the accumulation lag becomes the important factor when high model orders are considered due to a dependency upon p:

$$T_{add} < \frac{2}{(p+2).s.f_s} \tag{6.6}$$

6.4.3 Design of a Programmable Array

The previous sections show that the design of a programmable model order PMA type array is affected by the following inter-related factors:

- programmable range of model orders
- partition size
- number of available PEs
- maximum sampling frequency of the input data
- type of approach - word-parallel or bit-serial
- performance of underlying hardware

Many options are therefore available to the designer. Ultimately the size of the array is determined by highest model order problem which needs to be processed since this presents the largest cost. It is therefore important to concentrate initially on a cost efficient solution to this problem and then go on to consider the modifications which need to be performed in order to solve problems with lower model orders.

Say, for example, that model order $p = 30$ is the highest that needs to be processed. The size of partition s should then be chosen so that the $p = 30$ DDG can be efficiently mapped into a number of PEs. Efficient partitions are shown on the plots in figures 6.10 and 6.11. The partition size must be such that (6.5) and (6.6), which determine the maximum allowable processing delays of the PE arithmetic modules when input data is sampled at its maximum frequency, are both satisfied. These timing constraints are in turn dependent upon the type of approach followed, the design of the arithmetic modules and the technology used for hardware implementation.

Bit-serial arithmetic modules are likely to be slower than their word-parallel equivalents when implemented with the same technology, e.g. full custom CMOS or FPGA's. With the increased ill-conditioning of a model order $p = 30$ problem it may be necessary to perform the matrix element calculation at long word-lengths in order to keep the error in the output of the decomposition stage at an acceptable level. An error analysis such as that discussed in chapter 5 could be used to determine the necessary word-length.

The previous analysis shows that cost is reduced when the size of the partition is increased. The aim is therefore to map the $p = 30$ problem using the largest possible partition and thus the minimum number of PEs while bearing in mind the timing constraints (6.5) and (6.6). The large model order may lead to the partition size being constrained by accumulation (6.6) rather than multiplication (6.5). If the partition sizes resulting from the timing analysis are deemed to be inefficient, that is a point not lying on the $p = 30$ curve of figure 6.10, then the next lowest efficient partition lying on the curve should be chosen. Maximum partition sizes in the bit-serial approach are more limited than those of the word-parallel approach due to the longer latency of bit-serial arithmetic modules. This leads to a choice between a word-parallel array which may contain just a small number of PEs and a bit-serial arrays of perhaps more than double this size.

(A) Word-Parallel Xilinx Implementation Example

The implementation of the problem size independent systolic array is best demonstrated with an example. Continuing with the theme of chapter 5 which considered the costs of Xilinx FPGA implementation, the FPGA route is considered again here in order to obtain some typical performance timing characteristics of multiplication and addition modules. Figure 6.14 shows a possible Xilinx XC3164-5 layout for the word-parallel version of the 3 by 3 bit multiplier from figure 3.19 (note that in order to minimise the diagonal sum path delays the Xilinx layout

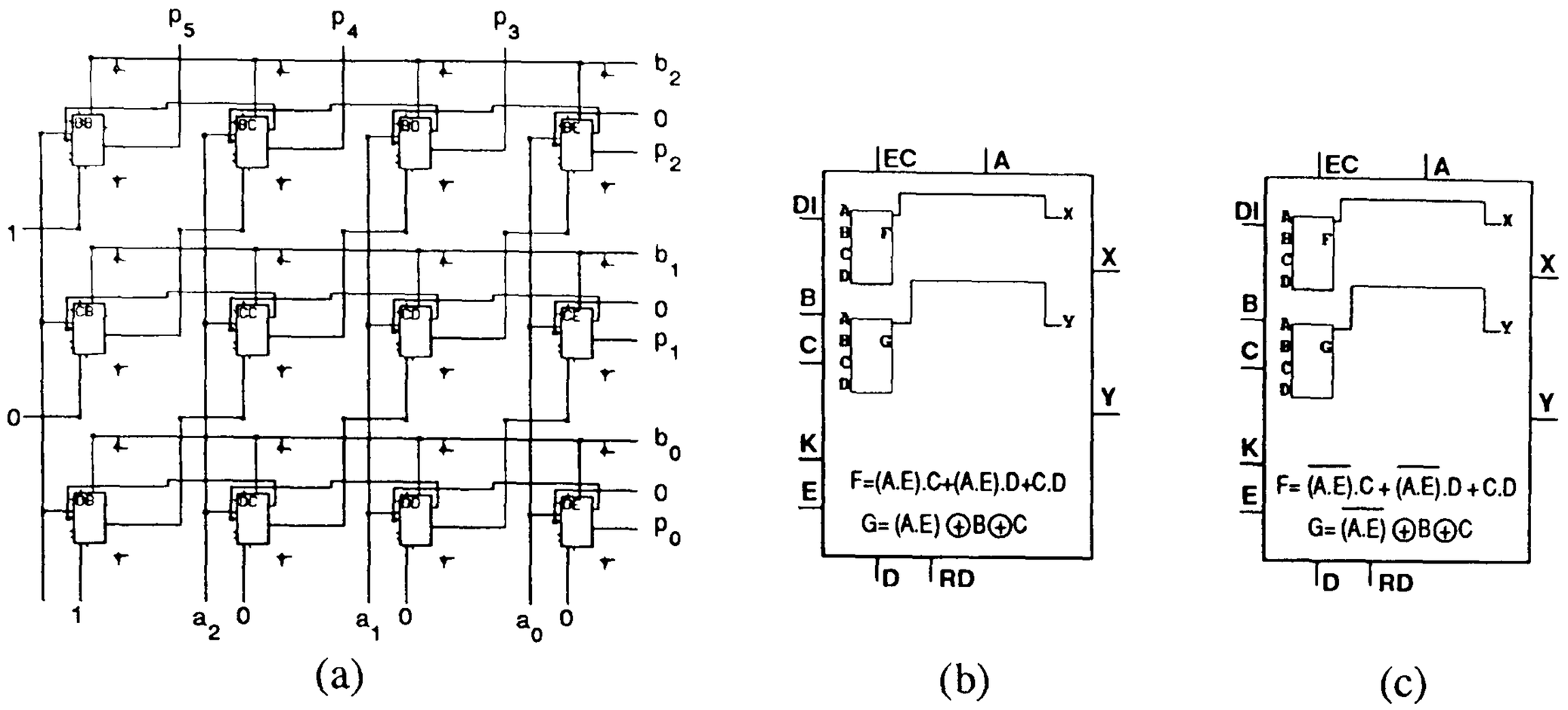


Figure 6.14: (a) Xilinx layout of the 3 bit two's complement word-parallel multiplier of figure 3.19 (b) *mode* 0 configuration of CLBs BB, BC, CB, CD, CE, DB, DD, DE, (c) *mode* 1 configuration of CLBs BD, BE, CC and DC.

is laterally inverted compared to the DDG representation so that FPGA CLBs BE, CE and DE correspond to $node[0,2]$, $node[0,1]$ and $node[0,0]$) which was directly configured using the XACT Design Editor (XDE) [131].

Use of the Xilinx XDelay timing analysis program shows that there the worst case ripple through path delay of the multiplier is from the inputs of CLB DE to the output p_5 of CLB BB and is equal to 38.0ns. The path can be broken up over 8 block levels as shown in table 6.3. The direct CLB interconnections, which produce 0.5ns delay, are utilised for the carry nets so that, given a CLB delay of 4.1ns, the CLB/carry net delays (e.g. from $DD.C \rightarrow DD.X \rightarrow DC.C$) total 4.6ns. Other indirect routeing resources must be utilised for the diagonal sum nets and these paths carry a longer 1.1ns delay so that the combined CLB/sum net delay (e.g. is $DB.C \rightarrow DB.Y \rightarrow CC.D$) is 5.2ns. These results can be used

From To	*DE.C* *DD.C*	*DD.C* *DC.C*	*DC.C* *DB.C*	*DB.C* *CC.D*	*CC.D* *CB.C*	*CB.C* *BC.D*	*BC.D* *BB.C*	*BB.C* *BB.Y*
Local Delay (ns)	4.6	4.6	4.6	5.2	4.6	5.2	4.6	4.6
Cumulative Delay (ns)	4.6	9.2	13.8	19.0	23.6	28.8	33.4	38.0

Table 6.3: Xilinx word-parallel 3 bit multiplier maximum delays.

to calculate the lag of a general input word-length multiplier since for the $w = 3$ bit version there are $2w = 6$ CLB/carry net delays and $w - 1 = 2$ CLB/sum net delays leading to a lag in ns of:

$$T_{mul} = 4.6(2w) + 5.2(w - 1) + 20 = 14.4w + 14.8 \tag{6.7}$$

for a w by w bit multiplier, where the extra 20ns allows for the setup time of the input data from the systolic registers and the time to store the product. Therefore when considering $w = 16$ bits as used for the results in figure 6.10 the multiplication time is estimated to be ≈ 250ns.

A similar approach can be used to calculate the total delay of the Xilinx ripple through adder circuit shown in figure 6.15. The total worst case delay in the adder is from the inputs of CLB BG to the output port $BB.X$ totals 27.6ns, that is 6 CLB/carry net (direct interconnect) delays of 4.6ns. Therefore the lag in ns of a double precision addition is:

$$T_{acc} = 4.6(2w) + 20 = 9.2w + 20 \tag{6.8}$$

where the extra 20 ns accounts for the setup delay and storage in the accumulation feedback registers directly connected to the output. A $w = 16$ bit double precision addition therefore takes ≈ 170ns.

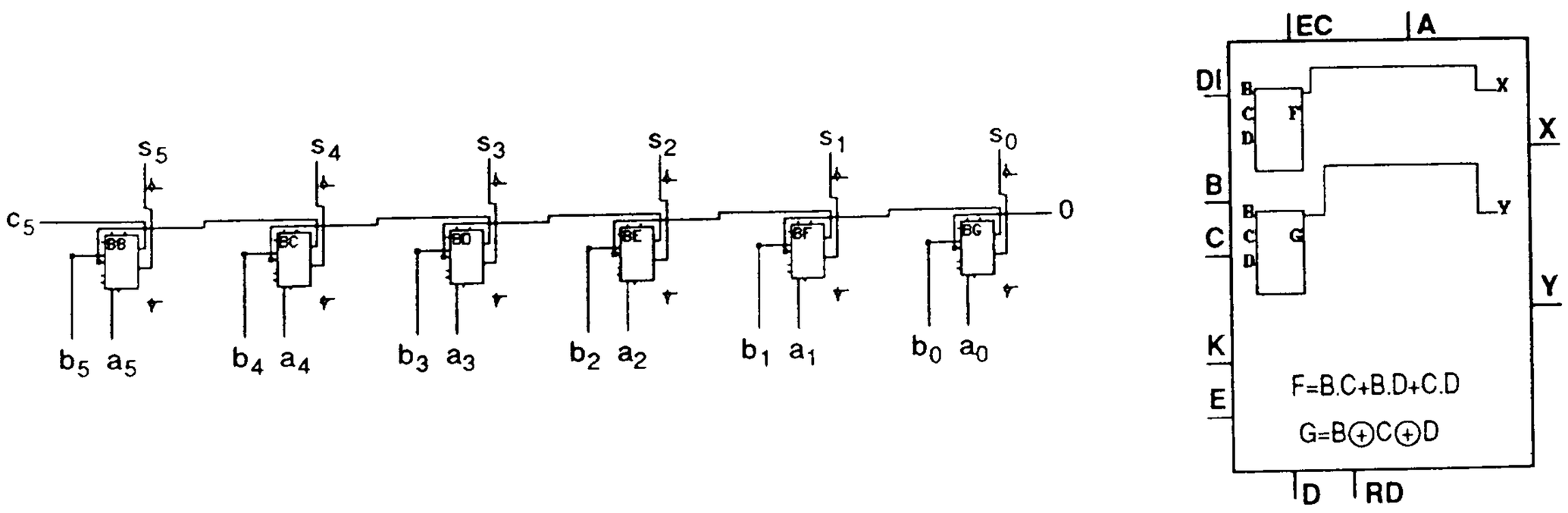
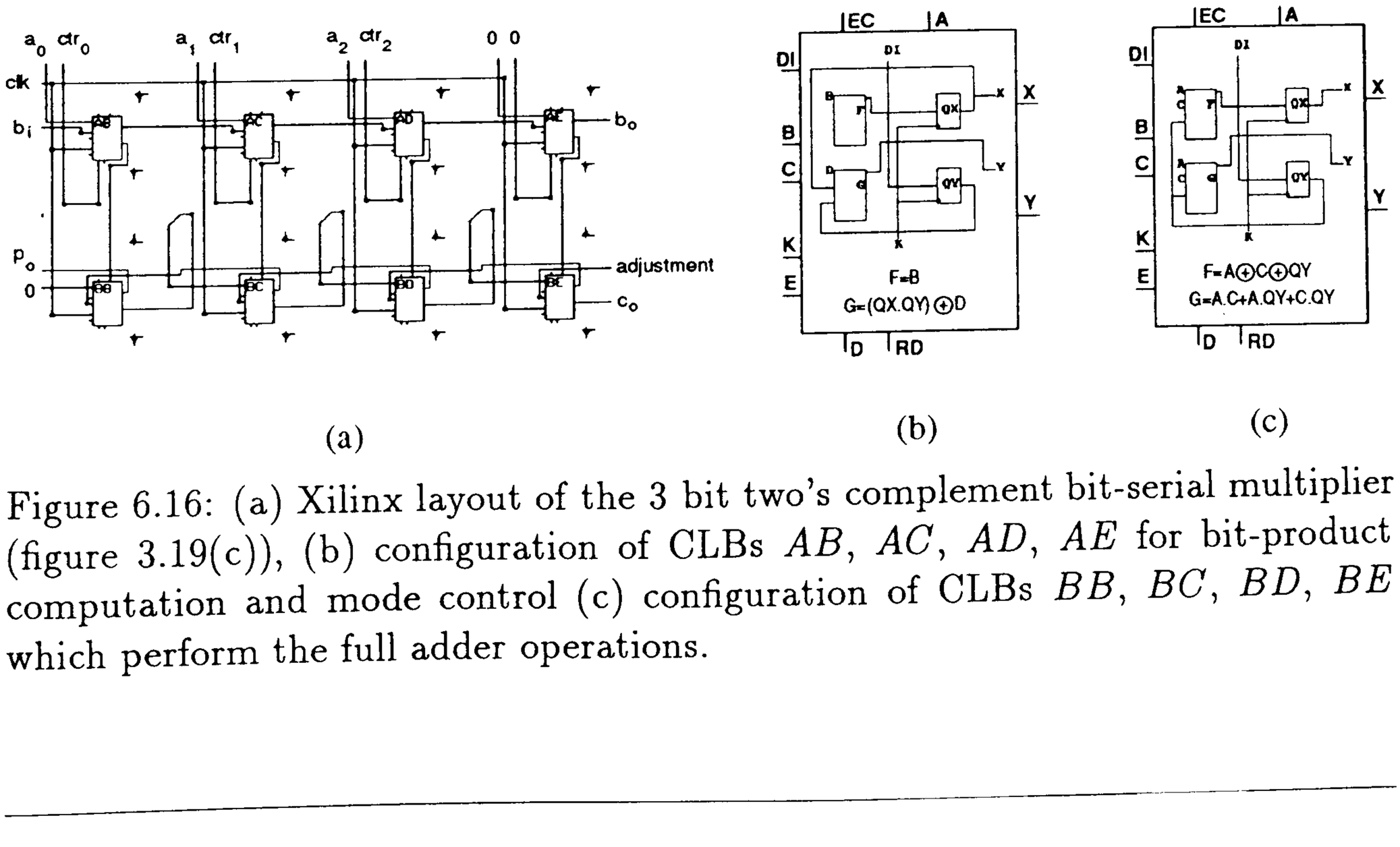


Figure 6.15: (a) Xilinx layout of a double precision $2w$ bit word-parallel adder (b) CLB full-adder arrangement.

Now given that the highest sampling frequency of the Doppler signal is f_s=51.2kHz then from (6.5) the constraint on the partition size imposed by the multiplication lag T_{mul} = 250ns is $s < 78.1$. Hence when considering a $p = 30$ estimation the maximum efficient partition size $s = 31$ (i.e. a single PE array) would be most efficient as can be seen by considering figure 6.10(a). However, at the high model order of $p = 30$, the accumulation sets a tighter constraint than multiplication as from (6.6) and figure 6.10(a) the maximum partition size is $s = 7$ given that a double precision addition takes 170ns. From figures 6.10(a) 6.11(a) the $s = 7$ partition gives a total array cost of $\approx$ 140000 NAND gates and requires 5 PEs.

(B) Bit-Serial Xilinx Implementation Example

The Xilinx LCA configuration for the bit-serial 3 bit multiplier of figure 3.19(c), where each PE is configured into two CLBs, is shown in figure 6.16. The top row of CLBs control the mode of the operation (i.e. invert the bit-product when changing to *mode* 1 operation) and produce the bit-product. The lower row CLBs perform the full adder operations and the result product bits are piped out from CLB BB with the final bit p_5 becoming available from this PE $4w - 1 = 11$ clock pulses after the start of operation.



(a) (b) (c)

Figure 6.16: (a) Xilinx layout of the 3 bit two's complement bit-serial multiplier (figure 3.19(c)), (b) configuration of CLBs AB, AC, AD, AE for bit-product computation and mode control (c) configuration of CLBs BB, BC, BD, BE which perform the full adder operations.

Using the XACT system timing analysis tool the lag associated with the maximum path delay in the Xilinx implementation of the bit-serial PE is 14.6ns which allows a maximum clock speed of 68.4MHz. The design can easily be extended for longer word-lengths w which should also run at around 68.4MHz since the PE pipeline delay remains the same. A w bit multiplication on the Xilinx bit-serial multiplier would therefore require a total operation time in ns of:

$$T_{mul} = 14.6(4w - 1) = 58.4w - 14.6 \tag{6.9}$$

and for $w = 16$ bits the operation time $\approx$ 920ns.

The bit-serial addition module for a general word-length w consists of a single full adder type PE with carry feedback, which can be viewed as a projection of the word-parallel adder (figure 6.15) along its carry propagation direction. This is easily implemented in Xilinx using a single CLB configured as shown in figure 6.17. The timing analysis tool reveals that the bit-serial adder has a lag of 12.1ns when locally connected within the FPGA and so can be clocked at frequencies up to 82.4MHz. The minimum time in ns of a double precision addition is therefore:

$$T_{add} = 12.1(2w) = 24.2w \tag{6.10}$$

and a 16 bit double precision bit-serial addition therefore takes $\approx$ 390ns. The bounds set by (6.5) and (6.6) for the model order $p = 30$ problem with $f_s = 51.2$MHz determine that the maximum partition size of $s = 3$, leading to an 11 PE array, is once again constrained by accumulation.

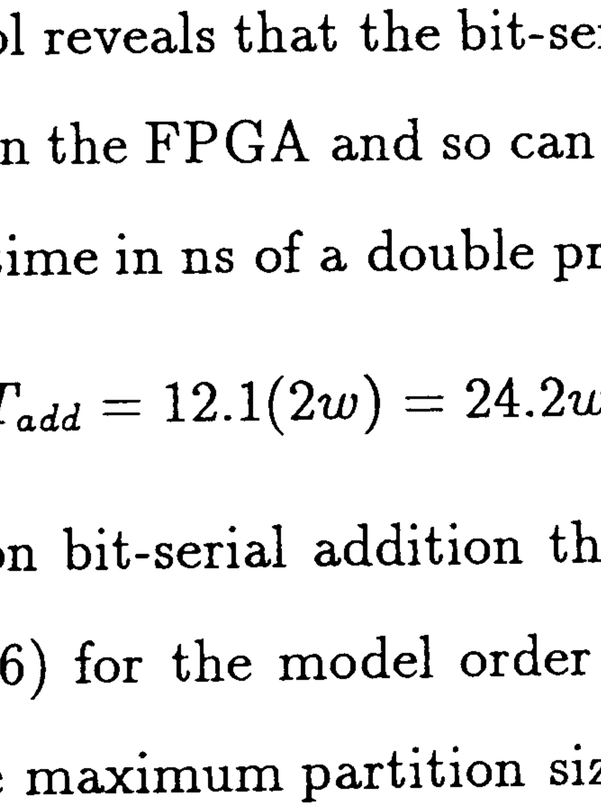


Figure 6.17: Xilinx layout of a bit-serial adder.

(C) Comparison of Xilinx Implementation Approaches

Figure 6.18 plots the partition size bounds on s set by (6.5), (6.6) and the propagation delays of the Xilinx word-parallel and bit-serial arithmetic modules for different estimators where p represents the maximum model order which needs to be computed. From these plots it is apparent that the limits set by accumulation determine the maximum possible partition size s as those set by multiplication which remain constant over the range of p are more lenient. This suggests a mismatch of resources in the arrays, resulting from the multiplication units not working as efficiently as they possibly could. Ideally the multiplication bound and addition bound would be equal, and this could be achieved by decreasing the speed of the multiplier or increasing the speed of the adder in each approach. A combined bit-serial/word-parallel approach could be taken where the bit-serial multiplier and word-parallel adder are used in the same system. Alternatively in the word-parallel implementation speed of word-parallel addition could be improved by using carry-lookahead logic [88] or with pipelining.

The optimal partition sizes for the Xilinx word-parallel and bit-serial implementations shown in figure 6.18 are determined by choosing the lowest cost optimal

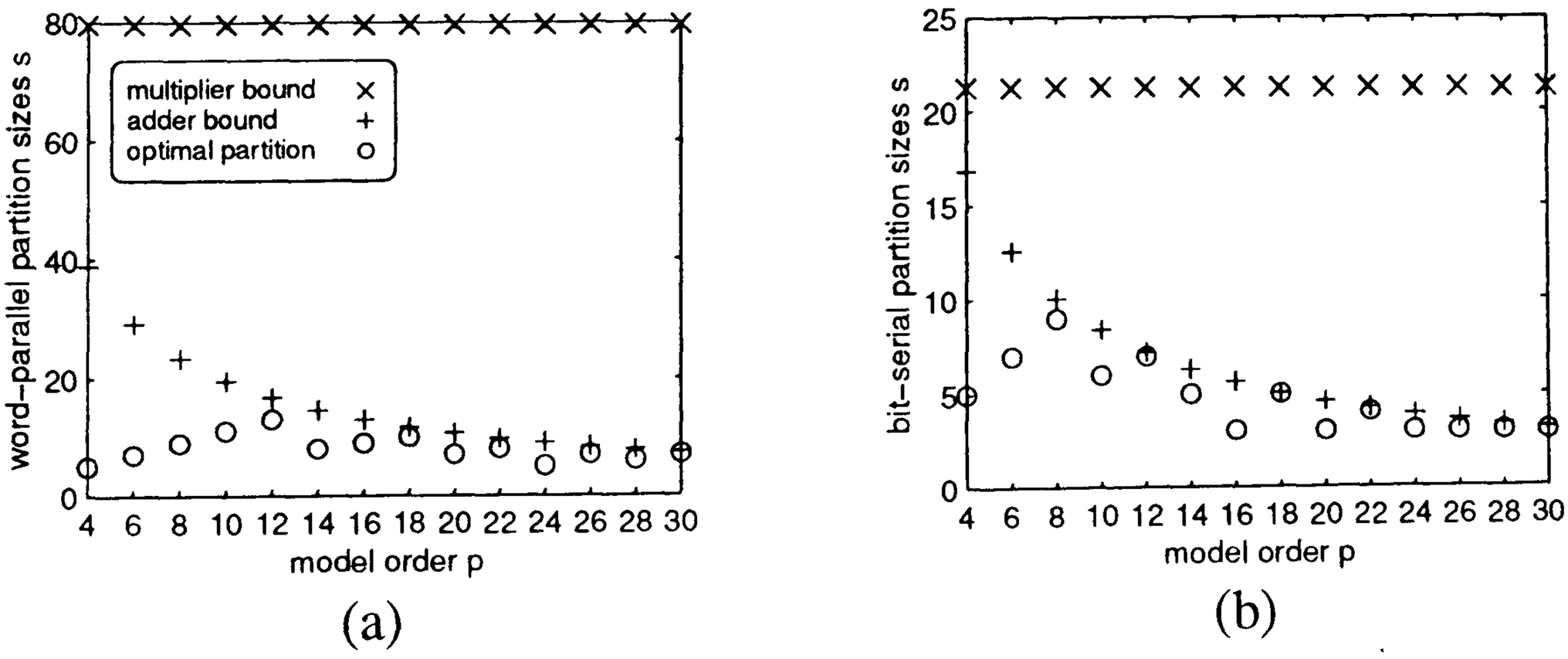


Figure 6.18: Maximum partition bounds set by multiplication/ addition and the optimal partition size which should be used when considering a number of different problems, where p represents the maximum model order of each problem, in (a) word-parallel and (b) bit-serial approaches.

partition sizes from in figure 6.10 which are below the bounds set by addition time. This gives the total costs of the word-parallel and bit-serial arrays for each model order shown in figure 6.19(a) and also the selection of s allows the number of PEs in each array (figure 6.19(b)) to be calculated using (6.2). The bit-serial approach, despite requiring greater numbers of PEs, presents slightly lower costs when compared with word-parallel. The bit-serial approach also has a lower communication cost and can utilise the resources of the FPGA more effectively (see section 5.3.4) when compared with the word-parallel approach whose only advantage lies in its simpler control circuitry. Once maximum model order p and optimal partition size s are known then the number of systolic registers and accumulation feedback registers can be calculated with reference to the general architecture in figure 6.20. Given the optimal array for the maximum model order, programmability to lower model orders is achieved by simply having a unique set of accumulation control lines for each of the different model order covariance matrices that need to be computed.

The costs of the problem size dependent systolic arrays (figure 6.3) can be compared with those of the problem size independent systolic arrays to show the advantage of the further partitioning proposed in this chapter. Figure 6.21 shows the costs of the problem size independent arrays relative to the costs of the prob-

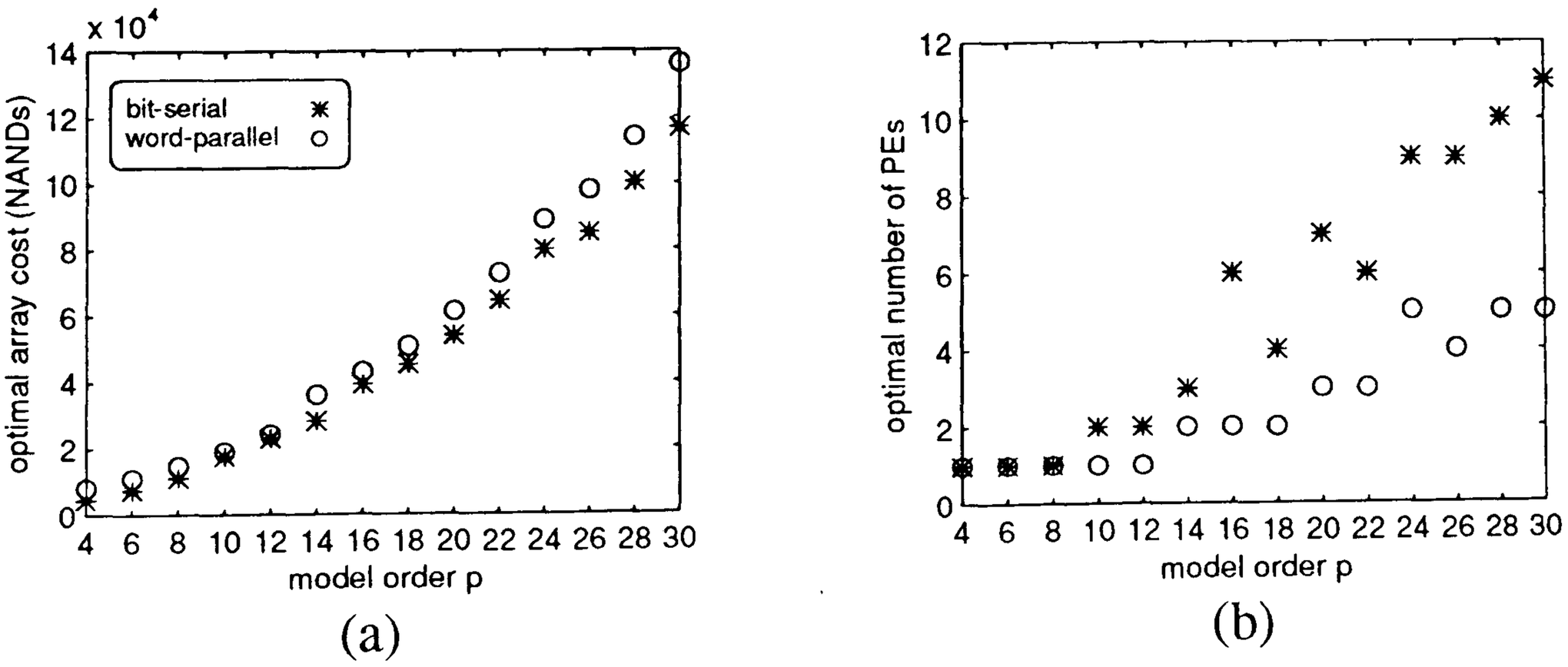


Figure 6.19: (a) Optimal hardware costs of the problem size independent systolic array designs for the covariance matrix elements for estimators with maximum model order p, (b) optimal number of PEs in each array.

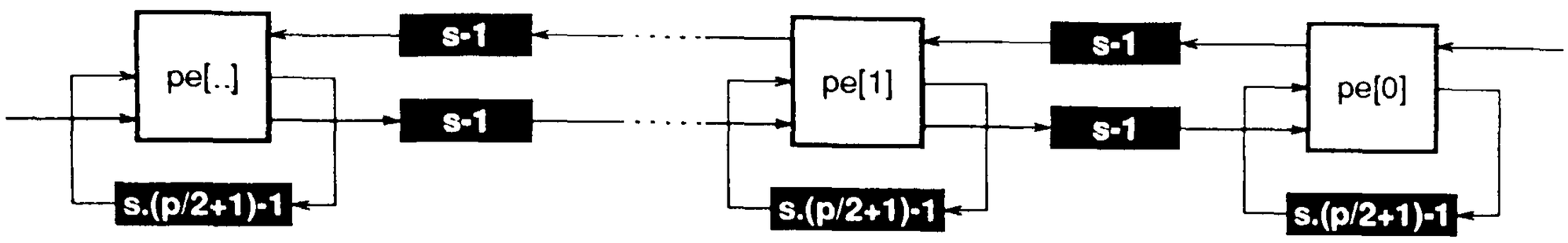


Figure 6.20: Generalised data architecture for the PMA array partitioned to size s for model order p. Black boxes indicate first-in first-out systolic register delays which are required to store the number of words indicated by the internal labelling. Note that the PE feedback registers need to store words at double precision and RAM could be used to replace the FIFO register for large partitions and high model order. The PE configuration is given in figure 3.18(b) and the number of PEs is required is determined by (6.2).

lem size dependent PMA arrays. The partitioning can be seen to be beneficial for all model orders in both approaches since the cost of all of the problem size independent arrays are lower than equivalent arrays in the problem size dependent approach. The word-parallel approach benefits the most as the new arrays are 30% to 60% respectively of the cost of the problem size dependent arrays from the lower to upper model orders comparing to 60% to 90% in the bit-serial approach. This is because the partitioning is used to reduce the number of arithmetic modules, which account for a higher proportion of the hardware cost compared to registers in the word-parallel approach than in the bit-serial approach and also because the faster speed of word-parallel arithmetic meant that the efficiency of the original problem size dependent arrays could be improved more.

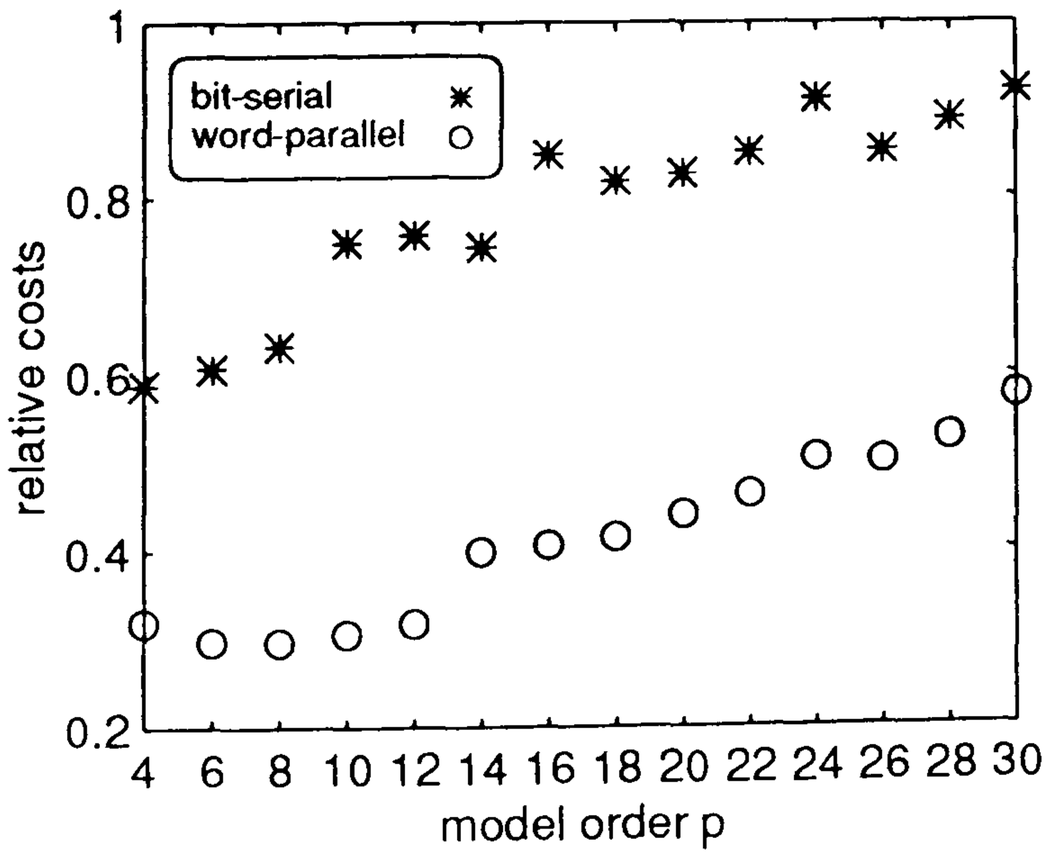


Figure 6.21: Relative costs of the problem size independent arrays compared to the problem size dependent arrays.

6.5 Concluding Remarks

This chapter has addressed the problem of mapping high model order matrix element calculations onto systolic array processors. Partitioning techniques were applied to the higher model order PMA type DDG designed systolic arrays introduced in chapter 3 in order to make the most efficient use of the processing potential of the PEs. Consequentially the partitioning was found to reduce the number of PEs required in the systolic array and although the number of systolic inter-PE and PE feedback registers was increased the effect of partitioning was shown to reduce the overall cost of the array.

It was shown how programmability could be achieved by first designing an efficient array for solution of the highest required model order problem. The computation of lower model order covariance matrices could then be implemented on the same systolic array by simply changing the accumulation control sequence.

In an example, which considered Xilinx FPGA implementation times of word-parallel and bit-serial arithmetic modules, the problem size independent design approach used in this chapter resulted in bit-serial arrays with slightly lower cost than the word parallel arrays of the same model order. However the partitioning was seen to be most beneficial in the word-parallel approach when comparing with arrays designed using the problem size dependent design methodology used in chapter 3. Cost effectiveness was also shown for the lower model order bit-serial arrays, as for example the gate usage for the 16 bit problem size dependent array was reduced from $\approx$ 7900 NANDs to $\approx$ 4700 NANDs in the problem size independent approach. In terms of implementation on Xilinx FPGAs this means that the XC3190 can be replaced with the smaller XC3164 (see figure 5.13).

Chapter 7

Programmable Model Order - Matrix Decomposition

7.1 Introduction

This chapter continues with the theme of the programmable model order Modified Covariance spectral estimator and considers problem size independent decomposition systolic arrays for use in the filter parameter computation hardware. An investigation into further partitioning of the Cholesky decomposition systolic array, thoroughly proven in the cost/benefit analysis of Chapter 5 to offer high cost effectiveness and numerical stability against the effects of finite word-length rounding in the model order $p = 4$ estimator, results in the proposal of a novel spiral systolic architecture for Cholesky decomposition.

The basis for the work described in this chapter is a spiral systolic array solution for LU decomposition designed using dense to band matrix and triangular block partitioning techniques by Navarro et al. [28]. This spiral systolic array and the design methodology behind it are first reviewed. A critical assessment reveals problems of subproblem chaining, bi-directional data flows and overall task/data scheduling in the spiral systolic LU array. DDG partitioning and reindexing methods are used to reconsider the design of an LU spiral systolic array showing

how the problems associated with Navarro's design can be overcome. By utilising redundant processing slots for array initialisation it is shown how consecutive subproblems can be merged to obtain minimal execution periods. It also demonstrates how first in, first out data register links around the array can be used for storage and rescheduling of L and U matrix elements, eliminating bi-direction data flows and the need for memory addressing during the computation.

A problem size independent Cholesky decomposition systolic array has not previously been proposed and the design of such an array is considered for the first time in this thesis. On partitioning the Cholesky decomposition DDG, analysis of the data dependencies within the subproblem DDGs formed shows that the LU decomposition spiral reindexing cannot be used in this case. It follows from this that the standard spiral architecture, used for the LU decomposition, is not suitable for problem size independent Cholesky computation. A new reindexing scheme, which does respect the data dependencies involved in the subproblems is used to show how arbitrary dimension Cholesky decomposition problems can be mapped into a novel spiral systolic architecture.

7.1.1 Problem Size Dependent Systolic Arrays for High Dimension Cholesky Decomposition

(A) Cost

The justification for the development of a problem size independent systolic architecture for various dimension Cholesky decomposition can be seen by looking at the hardware cost of bit-serial and word-parallel implementations of the problem size dependent systolic arrays. The design of the Cholesky decomposition $[1,1,1]$ array, shown in figure 4.8 for model order $p = 4$, is scalable for higher dimension problems. Referring to the array and its PE functions in figure 4.8 then for a model order p array $pe[1,1]$ is always type $PE0$, $pe[2,1]$ to $pe[p,1]$ along the left hand edge are all type $PE1$, $pe[2,2]$ to $pe[p,p]$ along the diagonal edge are type

PE type	number of PEs
$PE0$	1
$PE1$	$p-1$
$PE2$	$p-1$
$PE3$	$p^2/2-3p/2+1$

Table 7.1: Number of PEs used in the model order p problem size dependent Cholesky $[1,1,1]$ decomposition array (figure 4.8 for $p=4$).

w bit module	no. of modules per PE			
	$PE0$	$PE1$	$PE2$	$PE3$
subtraction	0	0	1	1
multiplication	0	0	1	1
division	0	1	0	0
square root	1	0	0	0
register	1	2	2	3

Table 7.2: Hardware module usage for each of the PEs in the Cholesky systolic arrays in terms of module usage.

$PE2$ and the remainder of the PEs filling the triangle between these two edges are type $PE3$. The PE type usage in the problem size dependent systolic array is summarised in table 7.1. Given the number of modules used in each of these PEs in table 7.2 with the information in tables 3.1, 3.2 and 4.1 then the NAND gate usage for model order p Cholesky decomposition word-parallel and bit-serial problem size dependent arrays may be estimated.

Figure 7.1 shows plots of the word-parallel and bit-serial Cholesky decomposition array costs in terms of the even model order p from 4 to 30. For example purposes the word-length used is $w=16$ bits for each model order, although this word-length may need extending when allowing for increased ill-conditioning at the higher model orders and the higher number of finite word-length rounding errors which affect the accuracy of the matrix elements in L. The costs in both approaches dramatically increase at higher model orders than $p=4$ due to the extension of the systolic array in 2 dimensions.

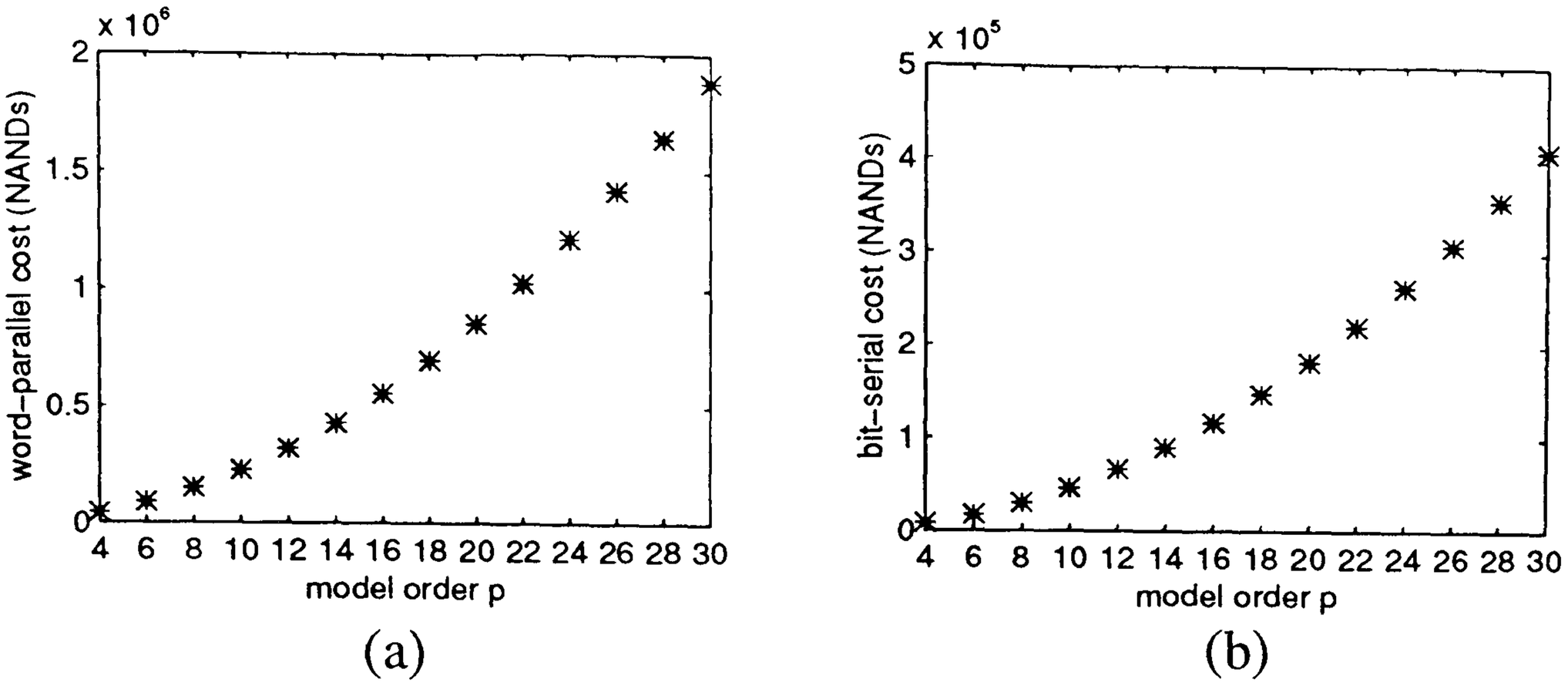


Figure 7.1: Cost for (a) word parallel and (b) bit-serial approaches in terms of model order p for a fixed word-length of $w = 16$ bits.

While the whole of the model order $p = 4$ bit-serial Cholesky decomposition could be easily be mounted on a single FPGA, the $w = 16$ bit $p = 30$ bit-serial array, at a cost of $\approx 4.1\text{x}10^5$ NAND gates, would require at least 35 XC3195 FPGAs which are the largest in the Xilinx XC3100 series (see section 5.3.4 and figure 5.13). Matters are even worse in the word-parallel array where the number of NANDs is $\approx 1.9\text{x}10^6$. A average sized $w = 16$ bit word-parallel PE of this array could be mounted on an XC3190 (see section 5.3.4 and figure 5.12) but a total of $\frac{p^2+p}{2} = 465$ such PEs would be required. This is clearly not practical when the aim is to design a portable system and although much larger densities could be achieved in full custom implementation, the cost is unnecessarily high as can be seen by considering the latency of arithmetic modules within the PEs.

(B) Timing Considerations

The majority of the PEs in the bit-serial and word-parallel arrays contain multiply accumulate devices. From the results of the Xilinx implementations discussed in chapter 6 sections 6.4.3(A) and (B) word-parallel $w = 16$ bit multiply accumulate takes in the order of $\approx$ 335ns (single precision addition) compared to less than 1μs for bit-serial (assuming the bit-serial addition unit processes the output product

bits as they become available). Given that the block pipeline period of the array $\beta = 3p$ (section 4.4.9) and assuming that the time length of the input data window, a minimum of 2ms, is available for processing the $p = 30$ Cholesky problem then the maximum allowable lag of a PE operation in order to maintain real-time processing on consecutive data segments is $\frac{2ms}{3x30} \approx 22\mu s$. The maximum clock frequency of the array is set by the division and square root bottleneck operations [80] and this limits the efficiency of the multipliers to a certain degree. Even so in the $p = 30$ problem size dependent array there is clearly a huge wastage of the processing potential throughout of the multiply accumulate devices in the array, especially in the word-parallel version which firmly justifies investigation of problem size independent arrays for high dimension Cholesky decomposition.

7.2 Review of Problem Size Independent Decomposition Arrays

Partitioning of algorithms, so that large dimensioned problems can be mapped onto small systolic arrays, is necessary to reduce cost and to use the arithmetic units within the array PEs more efficiently. Of particular interest here is the work of Navarro, Llaberia and Valero [28] which maps the LU decomposition onto a spiral systolic array architecture using triangular block partitioning and dense to band matrix partitioning methods. Their systolic array is reviewed in this section to introduce the concept of spiral architectures. The next section reconsiders LU spiral systolic array design in terms of the DDG methods used throughout this thesis, resulting in the formation of an improved array, and providing a basis for the design of a novel Cholesky decomposition spiral systolic array.

Navarro et al. consider the general problem of mapping the p by p dimension LU decomposition algorithm onto a two dimensional s by s systolic array [28][99][100]. To illustrate their scheme the problem of mapping the $p = 4$ LU decomposition onto a $s = 2$ array is reviewed because this represents the simplest example of

spiral LU decomposition array design. It should be noted however that the $s = 2$ partition size is not necessarily optimal for the all the values of p from 4 to 30. Initially the input matrix C is divided into square blocks whose dimensions are s by s.

$$C = \begin{bmatrix} c[1,1] & c[1,2] & \vdots & c[1,3] & c[1,4] \\ c[2,1] & c[2,2] & \vdots & c[2,3] & c[2,4] \\ \dots & \dots & & \dots & \dots \\ c[3,1] & c[3,2] & \vdots & c[3,3] & c[3,4] \\ c[4,1] & c[4,2] & \vdots & c[4,3] & c[4,4] \end{bmatrix} \tag{7.1}$$

Each block is referenced as $C_{j,k}$, so that for example $C_{1,2}$ refers to upper right sub-matrix of C and the L and U matrices are similarly split into their respective blocks. Navarro, Llaberia and Valero propose an algorithm for calculation of the subproblems formed by the above matrix partitioning. Applying this algorithm to the $s = 2$ example results in the following. Matrices $L_{1,1}$ and $U_{1,1}$ are initially computed from:

$$C_{1,1} = L_{1,1}.U_{1,1} \tag{7.2}$$

given that each of the elements on the leading diagonal of $L_{1,1}$ are 1. This is a $p = 2$ LU decomposition problem. Matrix $U_{1,2}$ is then calculated from:

$$C_{1,2} = L_{1,1}.U_{1,2} \tag{7.3}$$

which is followed by the calculation of $L_{2,1}$ given that:

$$C_{2,1} = L_{2,1}.U_{1,1} \tag{7.4}$$

Equations (7.3) and (7.4) are basically of the same type as each is the solution of a triangular system of equations. Matrix $C_{2,2}$ is then updated according to:

$$C_{2,2} \Leftarrow C_{2,2} - L_{2,1}.U_{1,2} \tag{7.5}$$

and the final step in the algorithm is to compute $L_{2,2}$ and $U_{2,2}$ from:

$$C_{2,2} = L_{2,2}.U_{2,2} \tag{7.6}$$

which is a similar problem to that described by (7.2).

With the aid of triangular block partitioning (TBP) and dense to band transformation (DBT) of matrices, Navarro et al. rearrange the matrix problems in (7.2) to (7.5) to fit onto a square s by s systolic array. The operation of the array and the TBP and DBT techniques used are illustrated here with respect the $C_{2,2}$ matrix updating (7.5) which involves a matrix by matrix operation. The iteration (7.5) can be rewritten as:

$$C_{2,2} \Leftarrow C_{2,2} - P_{2,2} \tag{7.7}$$

where $P_{2,2}$ in terms of the matrix elements is:

$$\begin{bmatrix} p[3,3] & p[3,4] \\ p[4,3] & p[4,4] \end{bmatrix} = \begin{bmatrix} l[3,1] & l[3,2] \\ l[4,1] & l[4,2] \end{bmatrix} \begin{bmatrix} u[1,3] & u[1,4] \\ u[2,3] & u[2,4] \end{bmatrix} \tag{7.8}$$

Using the TBP and DBT transformations the matrix product computation is re-represented as:

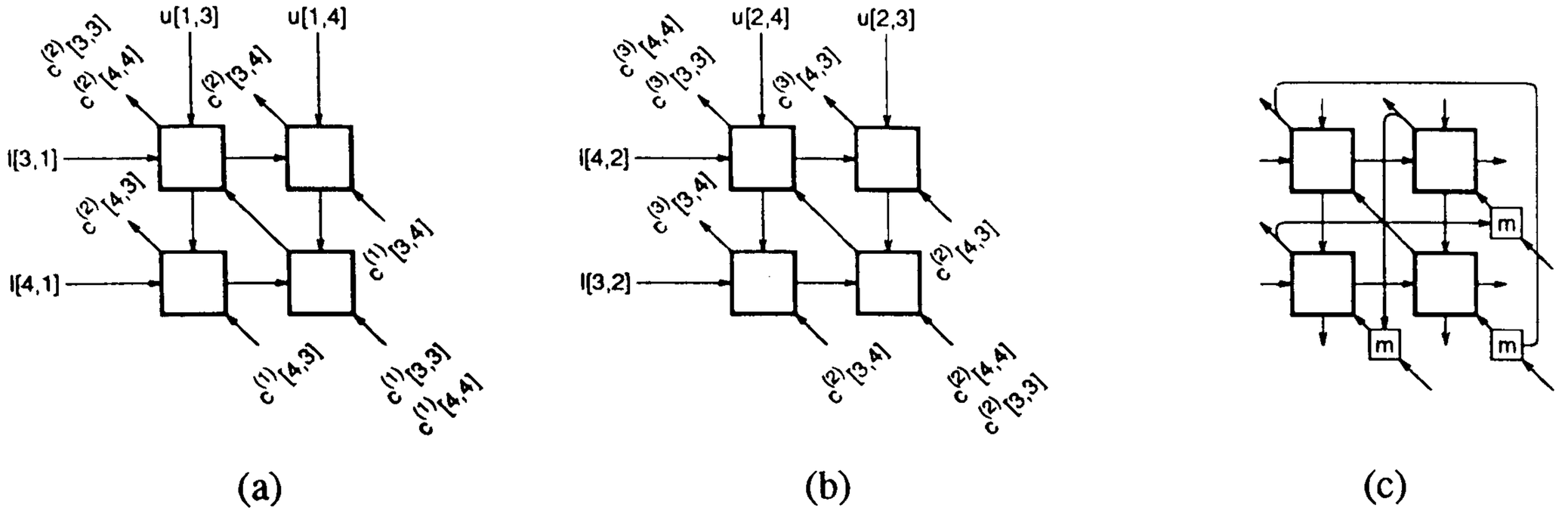
$$\begin{bmatrix} p^{(1)}[3,3] & p^{(1)}[3,4] & 0 \\ p^{(1)}[4,3] & p^{(1)}[4,4] + p^{(2)}[4,4] & p^{(2)}[4,3] \\ 0 & p^{(2)}[3,4] & p^{(2)}[3,3] \end{bmatrix} \tag{7.9}$$

$$= \begin{bmatrix} l[3,1] & 0 \\ l[4,1] & l[4,2] \\ 0 & l[3,2] \end{bmatrix} \begin{bmatrix} u[1,3] & u[1,4] & 0 \\ 0 & u[2,4] & u[2,3] \end{bmatrix} \tag{7.10}$$

where the product elements are then equal to:

$$p[j,k] = p^{(1)}[j,k] + p^{(2)}[j,k] \tag{7.11}$$

The purpose of performing this transformation is to express each of the matrices in (7.8) in band format such that the width of the bands correspond to the number of available input buses along an edge of the systolic array that the algorithm is to be performed on. The transformed $L_{2,1}$ and $U_{1,2}$ matrices, each have bands of width 2, which encompass 2 diagonals of non-zero elements representing 2 input streams of data into the systolic array. The $P_{2,2}$ matrix has a band of width 3 and so represents 3 streams of data calculated within the array.



(a) (b) (c)

Figure 7.2: Systolic array for calculation of (a) (7.12), (b) (7.13) and (c) spiral systolic architecture for computation of both iterations.

If matrix $C_{2,2} = C_{2,2}^{(1)}$ is updated to $C_{2,2}^{(3)}$ over two iterations then the first step:

$$\begin{bmatrix} c^{(2)}[3,3] & c^{(2)}[3,4] \\ c^{(2)}[4,3] & c^{(2)}[4,4] \end{bmatrix} \Leftarrow \begin{bmatrix} c^{(1)}[3,3] & c^{(1)}[3,4] \\ c^{(1)}[4,3] & c^{(1)}[4,4] \end{bmatrix} - \begin{bmatrix} p^{(1)}[3,3] & p^{(1)}[3,4] \\ p^{(1)}[4,3] & p^{(1)}[4,4] \end{bmatrix} \quad (7.12)$$

can be computed on the s by s systolic array in figure 7.2(a) where each of the PEs are of inner product step type (note that zeroes must be interleaved into the data streams). However on the second step the elements in each matrix are diagonally interchanged in order to respect the ordering of the input data streams defined by the $P_{2,2}$ band matrix:

$$\begin{bmatrix} c^{(3)}[4,4] & c^{(3)}[4,3] \\ c^{(3)}[3,4] & c^{(3)}[3,2] \end{bmatrix} \Leftarrow \begin{bmatrix} c^{(2)}[4,4] & c^{(2)}[4,3] \\ c^{(2)}[3,4] & c^{(2)}[3,2] \end{bmatrix} - \begin{bmatrix} p^{(2)}[4,4] & p^{(2)}[4,3] \\ p^{(2)}[3,4] & p^{(2)}[3,2] \end{bmatrix} \quad (7.13)$$

This can also be performed on an s by s systolic array as shown in figure 7.2(b). In order to consecutively execute (7.12) and (7.13) on the same array then spiral links must be used with multiplexers to recirculate the results from the first iteration back into the array as shown in figure 7.2(c). Due to the use of the spiral or global links the array is termed a spiral systolic array.

The application of the TBP and DBT techniques to all of the subproblems (7.2) to (7.6) results in the array configurations shown in figure 7.3. The systolic array shown in figure 7.3(a), used to compute the s by s LU decomposition subproblems (7.2) and (7.6), consists of four different types of PE whose functions are given in part (e) of the figure. The array outputs the matrices $L_{1,1}$ and $U_{1,1}$ of subproblem (7.2) from the west and north border PEs. $L_{2,2}$ and $U_{2,2}$ calculated in subproblem

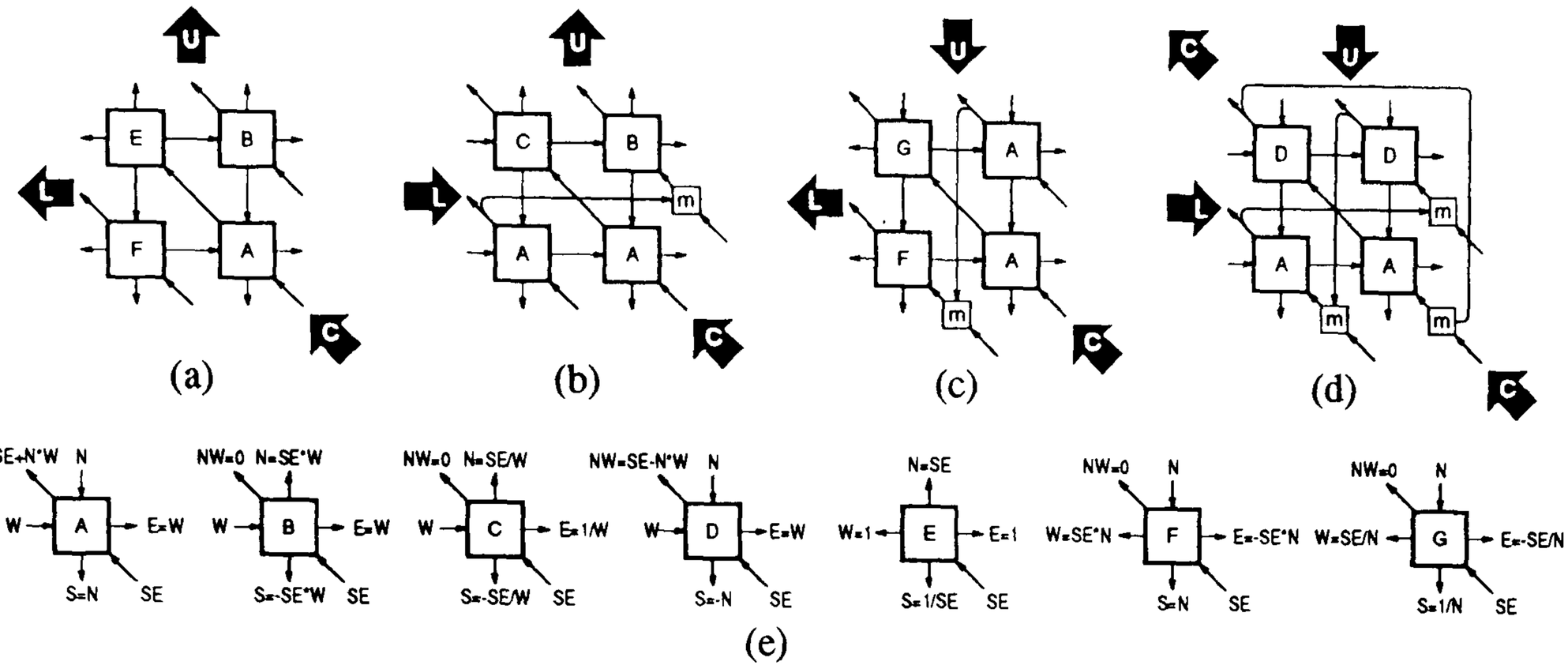


Figure 7.3: Navarro, Llaberia and Valero's systolic arrays for (a) LU decomposition ($s = 2$) - subproblems (7.2) (7.6), (b) solution of lower triangular matrix equations - subproblem (7.3), (c) solution of upper triangular matrix equations - subproblem (7.4), (d) matrix by matrix multiplication and accumulation -subproblem (7.5) and (e) key to processor operations.

(7.6) are similarly output. An architecture for solution of the subproblem (7.3) is given in figure 7.3(b). The array contains three types of PE, inputs $L_{1,1}$ to the west side PEs and outputs $U_{1,2}$ from the PEs on the north border. Subproblem (7.4) is the transpose of (7.3) and can be solved on the array given in figure 7.3(c). A systolic architecture for the last type of subproblem given by (7.5) already discussed is shown again in figure 7.3(d). This array requires two types of PE which perform inner product step functions.

All of the subproblems can be chained together for sequential solution of the $p = 4$ LU decomposition on a single $s = 2$ spiral systolic array. The systolic array can also be used to solve higher dimension problems such as $p = 6$ by applying the TBP and DBT schemes to the larger matrices creating a longer chain of ordered subproblems.

The following points arise in review of the LU spiral systolic array design by Navarro et al. [28]:

- The initial algorithm is partitioned into 4 different types of subproblem in a textual computational algorithm format.

- The subproblems must then be represented in matrix format and the way in which each type of subproblem is transformed using the TBP and DBT schemes differs between subproblems.

- The transformed subproblems are made to fit onto a pre-determined systolic architecture with a range of available PE functions.

- Although the partitioned matrices show the data flows into each input of the systolic array they do not relate information as to the scheduling and allocation of tasks in the array.

- There are a total number of seven different processor mode configurations.

- Bi-direction data ports are required for the input/output of L and U sub-matrices on the edge of the array.

- It is difficult to see how to chain subproblems, that is in determination of the time lag required between execution of two consecutive subproblems and the scheduling of input/output data flows.

7.3 LU decomposition Spiral Systolic Array DDG Design

This section reconsiders the design of the spiral systolic array in order to address the points made on the design of Navarro et al. [28], demonstrating,in terms of the DDG methodology, the whole of the LU spiral systolic array design process, from initial subproblem partitioning to the chaining of subproblems for implementation on a common architecture. It is shown how all subproblems can be reindexed using a common strategy as opposed to the various approaches of the TBP and DBT algorithms. This section also looks at the need for 7 different PE functions in Navarro's array, which strikes as being unnecessary since the original DDG representation before partitioning (figure 4.12) only requires 3 different modes of node operation. An unattractive feature of Navarro's array is that bi-directional

ports are needed on the north and west PEs of the array for the input/output of the L and U matrix elements respectively, the necessary buffering adding to PE complexity. Removal of this bidirectional data flow and the recirculation of data without resorting to storage and retrieval memory cycles are the prime objectives of the alternative DDG route described in this section. The timing and allocation of all tasks from each of the subproblem DDGs is described with the aim of minimising the time delay between successive subproblems.

7.3.1 LU Decomposition DDG Partitioning

The DDG partitioning of the $p = 4$ LU decomposition algorithm onto a s by s ($s = 2$) systolic array is considered to show direct comparisons with the method of Navarro et al. [28] and because this is the simplest example to present the methodology. It should be noted that the design methodology used can easily be extended to higher dimensioned problems or larger arrays and that the $s = 2$ partition is not necessarily the optimal size for the range of model orders considered. The aim is to partition the LU DDG into a number of subproblem DDGs which can be chained together and mapped onto a common systolic architecture. The systolic array produced is smaller than that which would be formed by the projection of the original DDG and utilises programmable function PEs to cope with the different types of subproblem.

Now consider partitioning the LU decomposition DDG shown in figure 4.12(a) into five cubed sections of side length $s = 2$ each containing a maximum of eight nodes as illustrated in figure 7.4. Each of the subproblems formed by the DDG partitioning directly corresponds to the partitioning proposed by Navarro et al. in (7.2) to (7.6) as indicated in the caption of figure 7.4. The additional nodes with the dotted circumference are effectively null processes, included to show the vertices of the cubes. These nodes are don't care processes and their functions can be programmed as desired to help simplify the design.

To map subproblems into a common systolic architecture the individual DDGs formed by the partitioning need to be expressed in the form of a common DDG arrangement. The DDG in figure 7.4(d) contains all the dependence arcs necessary for all the subproblems to be represented and its communication strategy can therefore be treated as a basis for all of the subproblem DDGs. The don't care nodes lead to some redundancy in subproblems (a), (b), (c) and (e) but all these subproblems could be expressed in terms of the structure of (d) with appropriate node function changes. Further on in the text it is discussed how the unused arcs and the redundant operations, which are created by expressing subproblems (a), (b), (c) and (e) on the common DDG structure, can be utilised in the chaining of the subproblem DDGs to minimise execution time. The systolic architecture produced by the projection of the common DDG structure is therefore able to handle all the subproblems, providing that the PEs are programmable.

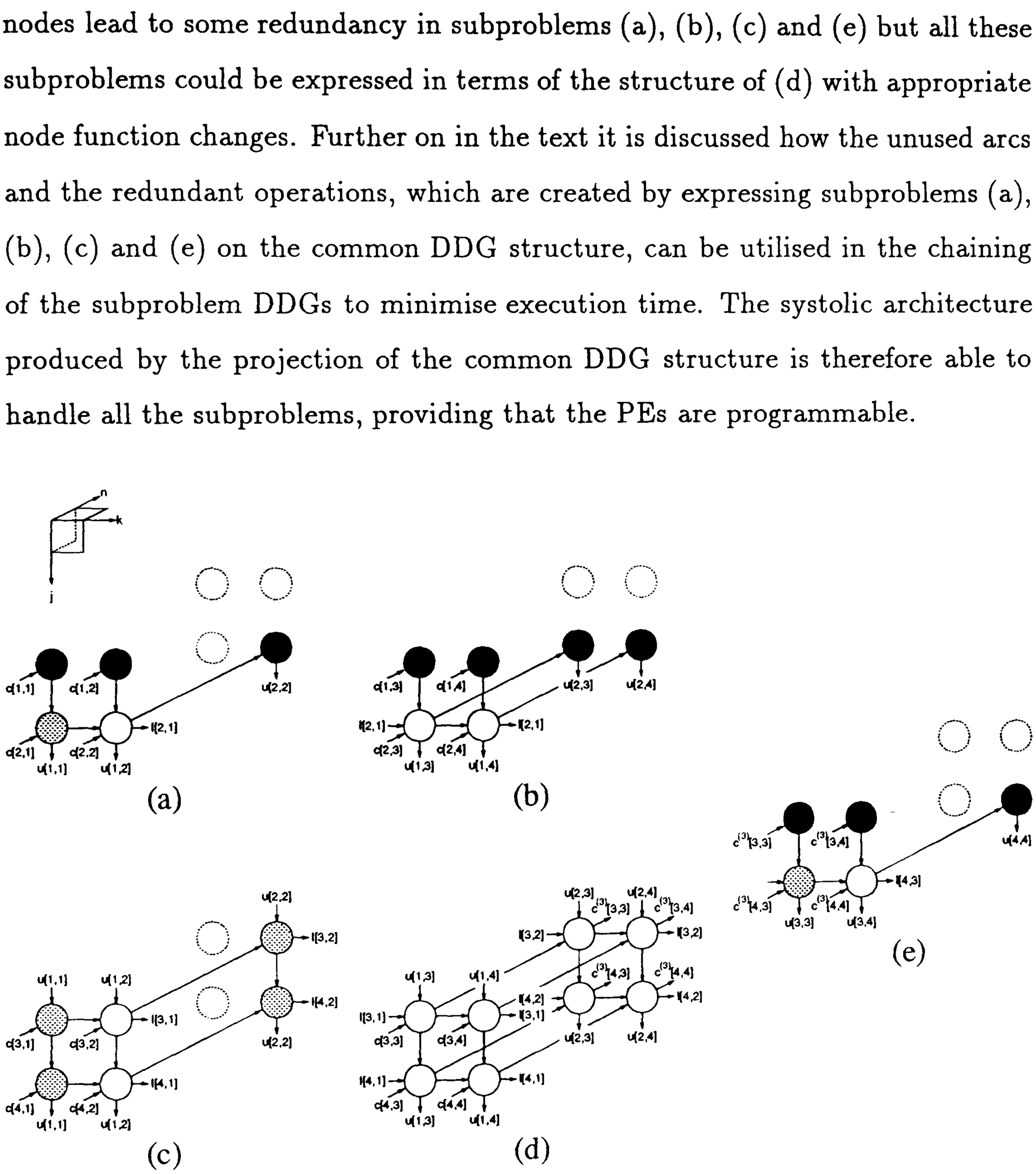


Figure 7.4: Partitioning of the model order $p = 4$ LU decomposition DDG into cubed subgroups of side length 2 nodes (a) $p = 2$ LU decomposition (7.2), (b) solution of lower triangular system (7.3), (c) solution of upper triangular system (7.4), (d) matrix by matrix computation (7.5) and (e) $p = 2$ LU decomposition (7.6).

7.3.2 Mapping the LU Common Subproblem DDG into an Array

Projection of the common DDG structure used for representation of all the subproblems produces a systolic architecture whose PEs can be programmed to perform the functions defined by any particular subproblem. The five subproblem DDGs can be chained together to form a long DDG from which, the overall timing and data dependencies between subproblems, can be derived. It is desirable to avoid projection in the exact direction of the arcs since this leads to internal processor storage, complicating control with clock enabling and data referencing. For the LU decomposition, subproblem DDG projection along the arc directions also results in systolic arrays whose PE functions are required to change within individual subproblem execution. The aim is therefore to have continuous data flows around the common architecture whose PE functions only need to change between different subproblems, ruling out the possibility of straightforward connection of the subproblem DDGs end on end with the projection direction orthogonal to the DDG interconnection planes.

The problem size dependent systolic arrays derived in [17][80] (chapter 4 figures 4.14 and 4.8 respectively) demonstrate continuous data flow since the $[1,1,1]$ projection vector used to produce these arrays cuts all of the DDG communication arcs. Projection of the partitioned DDG base section in the same direction, as illustrated in figure 7.5, produces a systolic array with similar characteristics. The

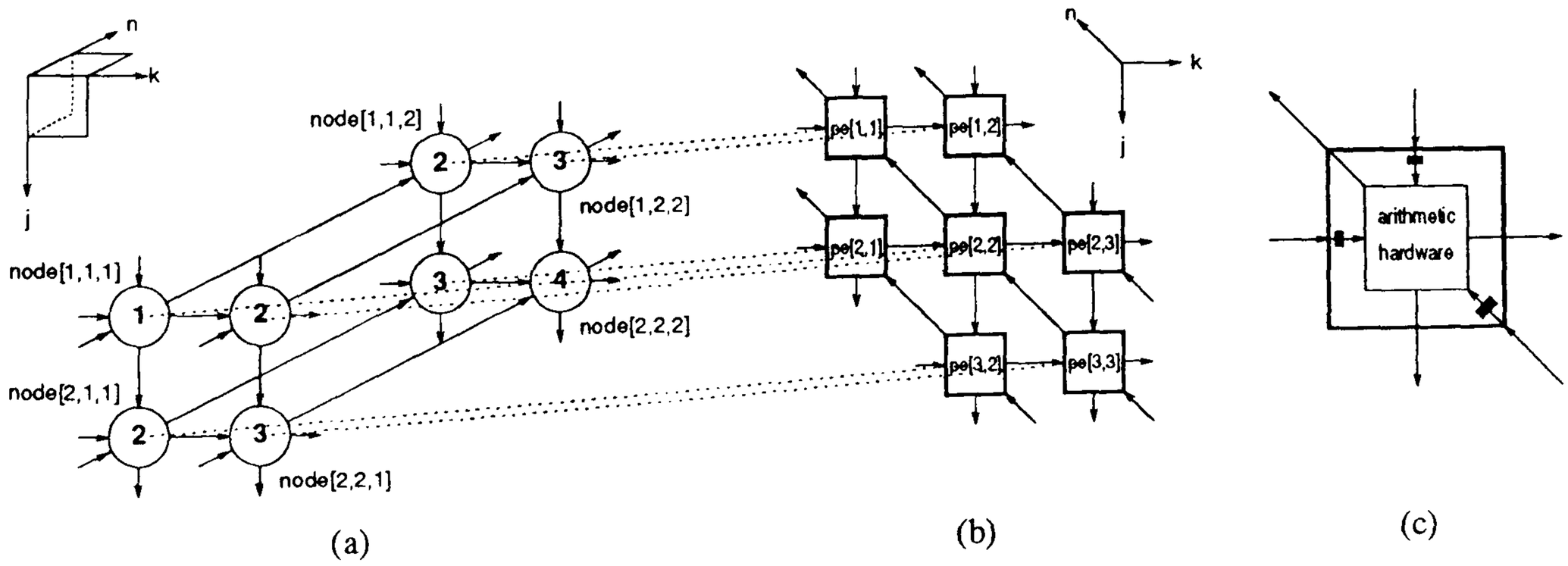


Figure 7.5: (a) Common subproblem DDG, (b) its $[1,1,1]$ projection into a hexagonally connected systolic architecture and (c) a general PE.

allocation function used to produce the array in figure 7.5(b) is for $node[j,k,n]$ to be mapped into $pe[j-n+2, k-n+2]$ and each node operation is processed at time $t = j+k+n-2$. The systolic array is however quite inefficient as whilst $pe[2,2]$ is active in one in every three clock cycles performing the tasks defined by $node[1,1,1]$ and $node[2,2,2]$, the other PEs only perform single node operations. This means that there is effectively almost one processor for each node of the graph. The inefficiency associated with the systolic array produced by the projection of the common DDG representation is ultimately reflected in the array used to solve the overall decomposition and so it worthwhile attempting to improve efficiency whilst maintaining the same projection vector.

7.3.3 Reindexing of the LU Common Subproblem DDG

(A) Node Reindexing for Projection into the s by s Array

The efficiency of the systolic array of figure 7.5(b) can be improved upon by rearrangement of the common subproblem DDG nodes so that the array produced from projection of the rearranged DDG contains fewer PEs. Kung uses a DDG reindexing scheme in the formation of a systolic array for the Warshall-Floyd algorithm [22]. The node reindexing is as follows for a partition of size s:

$$node[j,k,n] \rightarrow node[(j-n)_{\text{mod } s}+n, (k-n)_{\text{mod } s}+n, n] \tag{7.14}$$

and can be applied to the common subproblem DDG. The DDG produced by reindexing the common subproblem DDG with (7.14) is shown in figure 7.6. Only nodes of the second plane are reindexed, with the operations labelled **E**,**F** and **G** of $node[1,1,2]$, $node[1,2,2]$ and $node[2,1,2]$ respectively being moved to $node[3,3,2]$, $node[3,2,2]$ and $node[2,3,2]$. The $node[2,2,2]$ however is not affected by the reindexing and remains fixed in its position. The systolic array produced by $[1,1,1]$ vector projection (figure 7.6(c)) now contains four PEs, each performing two node operations for improved efficiency, compared to the seven PEs produced by the $[1,1,1]$ projection before reindexing (figure 7.5(b)).

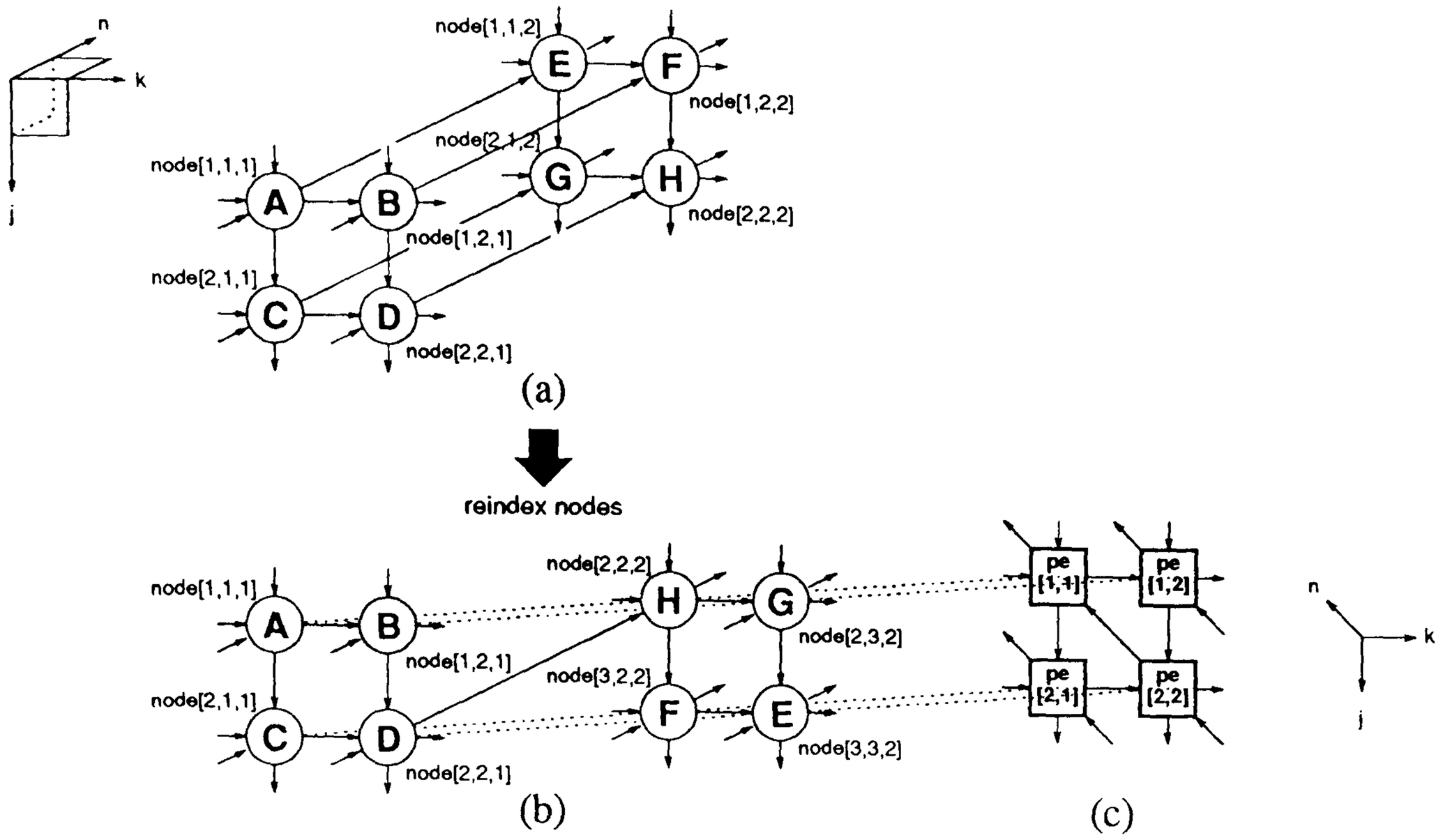


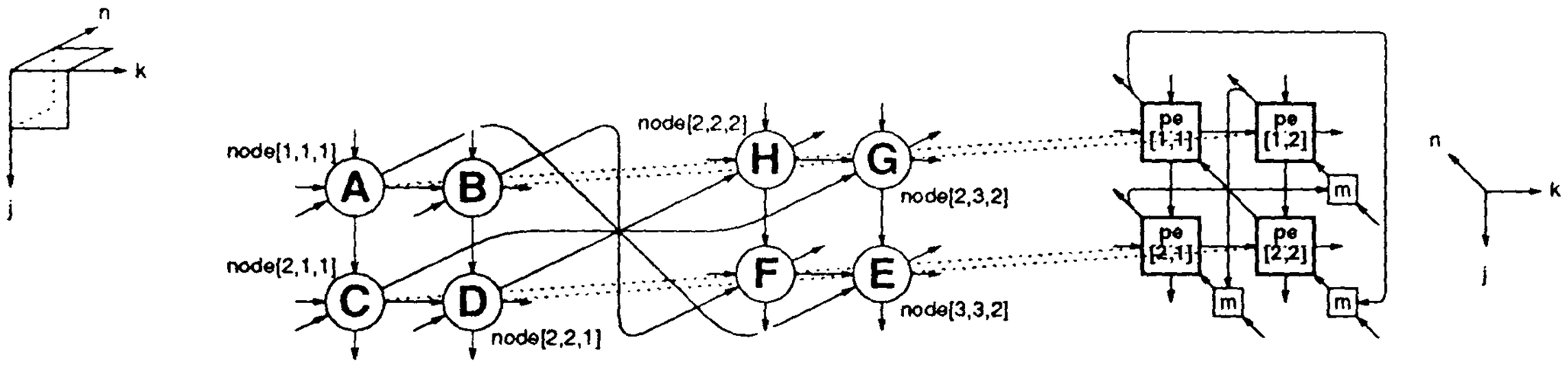
Figure 7.6: (a) The common subproblem DDG ($s = 2$) with each node operation labelled **A** to **H**, (b) effect of reindexing the common subproblem DDG ($s = 2$) using (7.14) and (b) $[1, 1, 1]$ projection of the reindexed nodes into a 2 by 2 systolic architecture.

The reason that this reindexing is permitted is that the order in which the nodes must execute is commutative in the j and k directions of the $n = 2$ planes in each of the subproblem DDGs shown in figure 7.4. The j and k direction arcs in the $n = 2$ plane of the reindexed common subproblem DDG in figure 7.6(b) can therefore remain localised in the same direction as before reindexing, instead of being swapped as would be the case with non-commutative tasks. Since the j and k arc directions are the same in the $n = 1$ and $n = 2$ planes of the reindexed DDG then they can therefore be mapped into single direction data buses in the systolic array, avoiding bi-directional flows.

(B) Spiral Data Dependence Arcs and their Systolic Array Representation

The data dependencies defined by the n direction arcs between the $n = 1$ and $n = 2$ planes in the original DDG of figure 7.6(a) (i.e. **A**→**E**, **B**→**F** and **C**→**G**) must also be represented in the reindexed graph (note re-representation not shown in figure 7.6(a)). This results in spiral or global dependence arcs from $node[1, 1, 1]$

to $node[3,3,2]$, $node[1,2,1]$ to $node[3,2,2]$ and $node[2,1,1]$ to $node[2,3,2]$ as shown in figure 7.7(a). The spiral arc communications in the DDG are mapped into three spiral data buses which connect across the array shown in figure 7.7(b) (e.g. the arc from **C** to **G** gets mapped into the spiral bus from $pe[2,1]$ to $pe[1,2]$ due to the respective allocation of these tasks to these two PEs). Three multiplexors are also required to switch between routeing external data into the array (i.e. n direction inputs to the $n = 1$ plane) and the spiral recirculation of data (i.e. n-direction inputs to the $n = 2$ plane). The spiral reindexing of the DDG is directly equivalent to the TBP and DBT methods (7.10) of Navarro et al. [28].

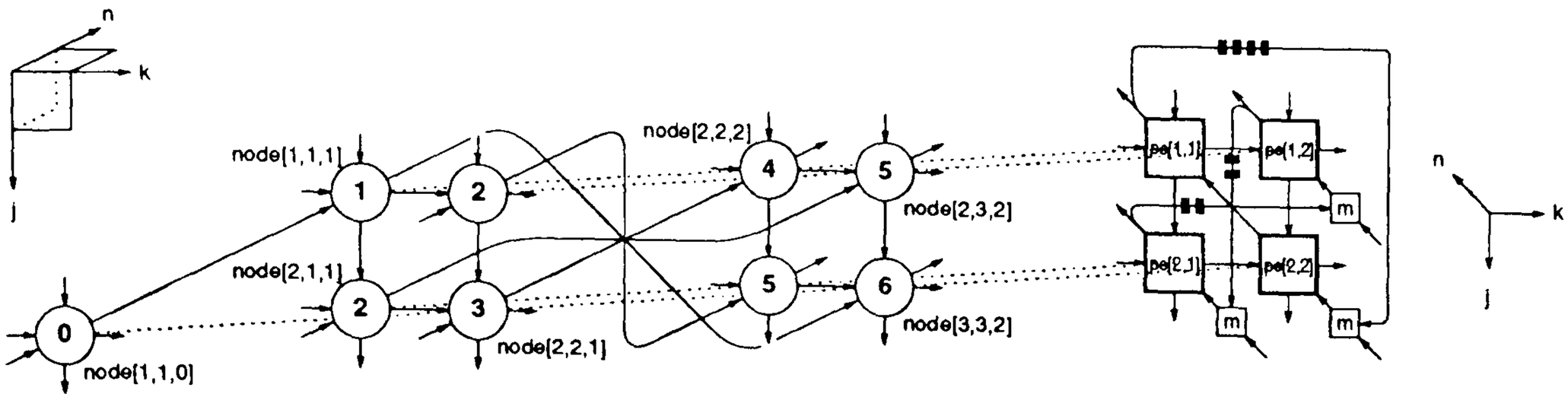


(a) (b)

Figure 7.7: (a) Inclusion of spiral interconnection arcs in the reindexed DDG and (b) $[1,1,1]$ projection of the spiral reindexed DDG into a s by s spiral systolic architecture.

(C) Timing Function

A possible timing function which respects the data dependencies imposed and does not allow a PE to execute more than one node operation on a single clock cycle is indicated by the internal node numbering in the DDG of figure 7.8. The node added to the DDG in position [1,1,0] is for the loading of data onto the n input of $pe[1,1]$ via $pe[2,2]$. The spiral buses carry extra systolic word registers to allow for the longer delay between the node processes connected by the spiral dependence arcs. For example $pe[2,1]$, performing the operation defined by $node[2,1,1]$ at $t = 2$, outputs a result from its n output port which is routed



(a) (b)

Figure 7.8: (a) Timing function for the reindexed common subproblem DDG and (b) extra systolic delay registers required on the spiral recirculation buses.

across the array to the n input of $pe[1,2]$ for the processing of the $node[2,3,2]$ operation at $t = 5$. This spiral bus therefore needs two extra systolic registers given that the third delay is provided by the register on the n input bus within $pe[1,2]$ (see figure 7.5(c) for a general PE configuration). Similarly, the spiral from $pe[1,2]$ to $pe[2,1]$ carries two extra registers but that from $pe[1,1]$ to $pe[2,2]$ has four extra registers.

7.3.4 Chaining the Reindexed LU Subproblem DDGs

The partitioned DDGs for the LU decomposition problem shown in figure 7.4 can be reindexed using (7.14) resulting in the DDGs of figure 7.9(a) to (e). In order to minimise mode switching, the null node processes indicated by the nodes with dotted circumference in figure 7.4, are set to the same mode as the corresponding $n = 1$ plane node in the line of projection of the reindexed DDGs. Therefore, when the graph is projected by the $[1,1,1]$ vector, each PE of the resulting systolic array needs to work only in one mode during execution of a particular subproblem.

All of these reindexed subproblem DDGs can be represented on the common reindexed DDG representation of figure 7.8(a) and therefore can all be mapped into the spiral systolic architecture. When considering the timing of the whole LU decomposition computation, the reindexed subproblem DDGs can be connected

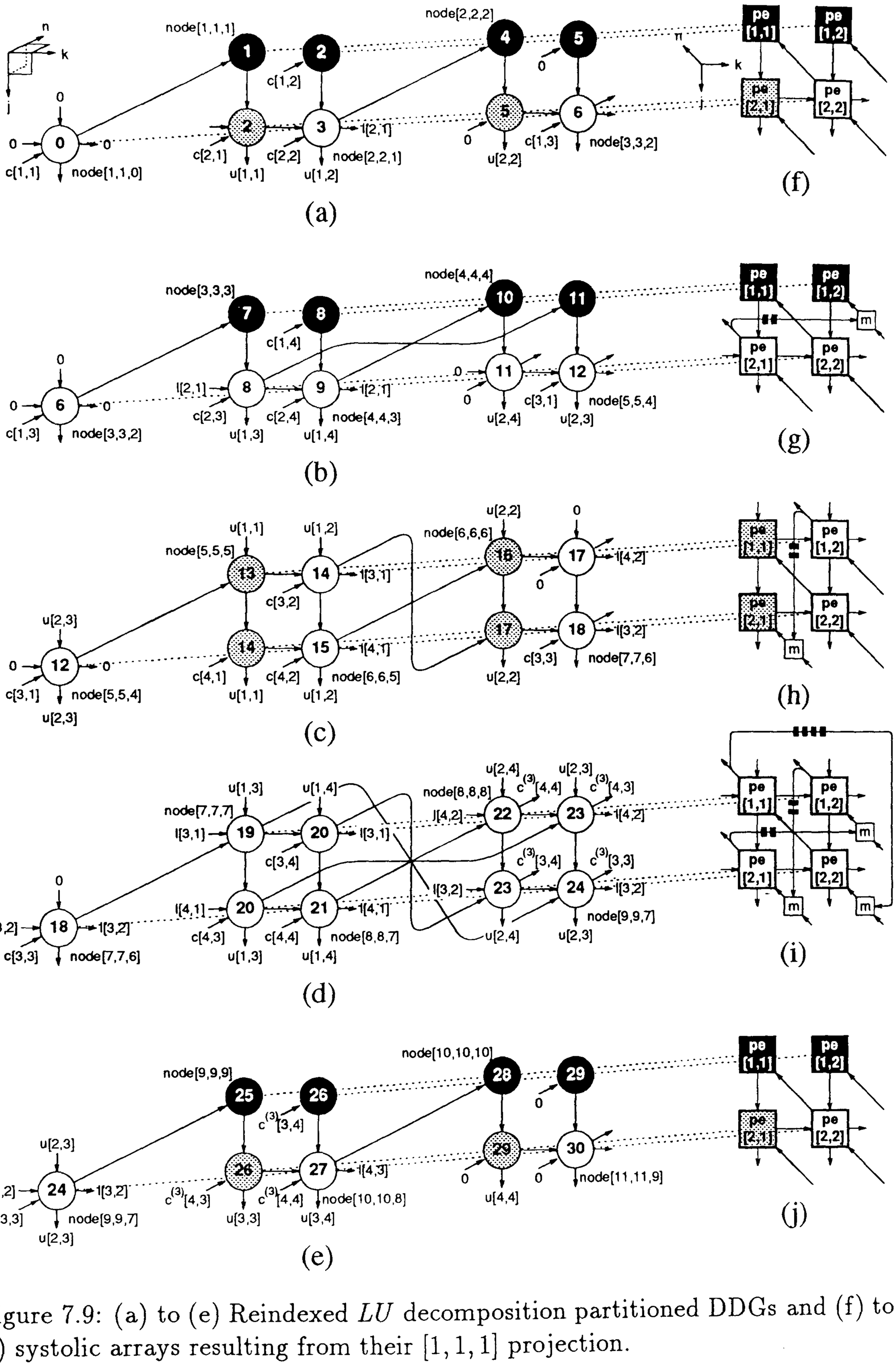


Figure 7.9: (a) to (e) Reindexed LU decomposition partitioned DDGs and (f) to (j) systolic arrays resulting from their $[1, 1, 1]$ projection.

together to form one long chained DDG. The $[1,1,1]$ projection of this DDG still results in the same spiral architecture but must have programmable function processing elements to cope with the different node functions in each subproblem. The order in which the subproblems are chained together must respect the data dependencies defined by the original LU decomposition DDG of figure 4.12. Therefore the graph in part (a) of figure 7.9 is the first part of the chain. The second and third sections are the DDGs (b) and (c) in either order. Here DDG (b) is executed before (c). The fourth section in the chain is the DDG in part (d) of the figure and the final section is part (e).

(A) Minimising Time Delay Between Successive Subproblems

The former null processes can be used to facilitate the data loading operation of $node[1,1,0]$ in the common subproblem DDG (figure 7.8(a)) in order to reduce the time delay between the execution of successive subproblems. The $node[3,3,2]$ in DDG (a) is one such null process which has been set to operate in *mode* 2. This node can coincide with the loading node of graph (b) because the zero j and k inputs to $node[3,3,2]$ cause transference of the data along its n direction input/output path. Data word $c[1,3]$ can therefore be clocked through $node[3,3,2]$ at $t = 6$. This procedure can be repeated all the way along the chain. Joining the graphs in such a manner causes the coordinates of nodes in consecutive sections to be offset by $[2,2,2]$. However, the n direction connections between the DDGs (d) and (e) are made in the systolic array via the spirals.

(B) Systolic Feedback of L and U Sub-matrices

The null processes are also used at the end of subproblems to transfer the calculated elements to output arcs of the DDG. For example, $u[2,2]$ in subproblem (a) is piped through $node[3,2,2]$. U and L sub-matrices can therefore be piped

out from the lower and right hand borders of the array respectively eliminating the need for the bi-directional data ports which in are necessary in the design by Navarro et al. [28] (figure 7.3).

Data j and k direction inputs to the edge of the array are easily derived by using delay feedback links around the array. The number of delays attached to these links is determined from consultation of the chained subproblem reindexed DDGs. In the DDG of figure 7.9(a) $l[2,1]$ gets output from $node[2,2,1]$ or $pe[2,2]$'s k output at $t = 3$ only to be input to $node[4,3,3]$ of the DDG in figure 7.9(b) or $pe[2,1]$'s k input at $t = 8$. Assuming that there are unit systolic delays on each of the PE inputs then this feedback link contains 4 extra systolic register word delays. Similarly $l[4,1]$ and $l[3,2]$ are recirculated around this feedback loop with the same delay of 5 clock pulses and another loop is required from $pe[1,2]$ to $pe[1,1]$. The two feedback loops (figure 7.10(a)) can be merged (figure 7.10(b)) because $pe[1,2]$ and $pe[2,2]$ output data on successive clock cycles with each processor active once in every three cycles. Note that to avoid logic contention, a multiplexor is required to select either the output of $pe[1,2]$ or that of $pe[2,2]$ but the control for this is easily implemented with a modulo three counter. Due to the ordering of the sections in the chain, 10 extra delays are needed in the feedback from the $pe[2,1]$ and $pe[2,2]$ j direction outputs to the $pe[1,1]$ and $pe[1,2]$ j input ports (figure 7.10(c)).

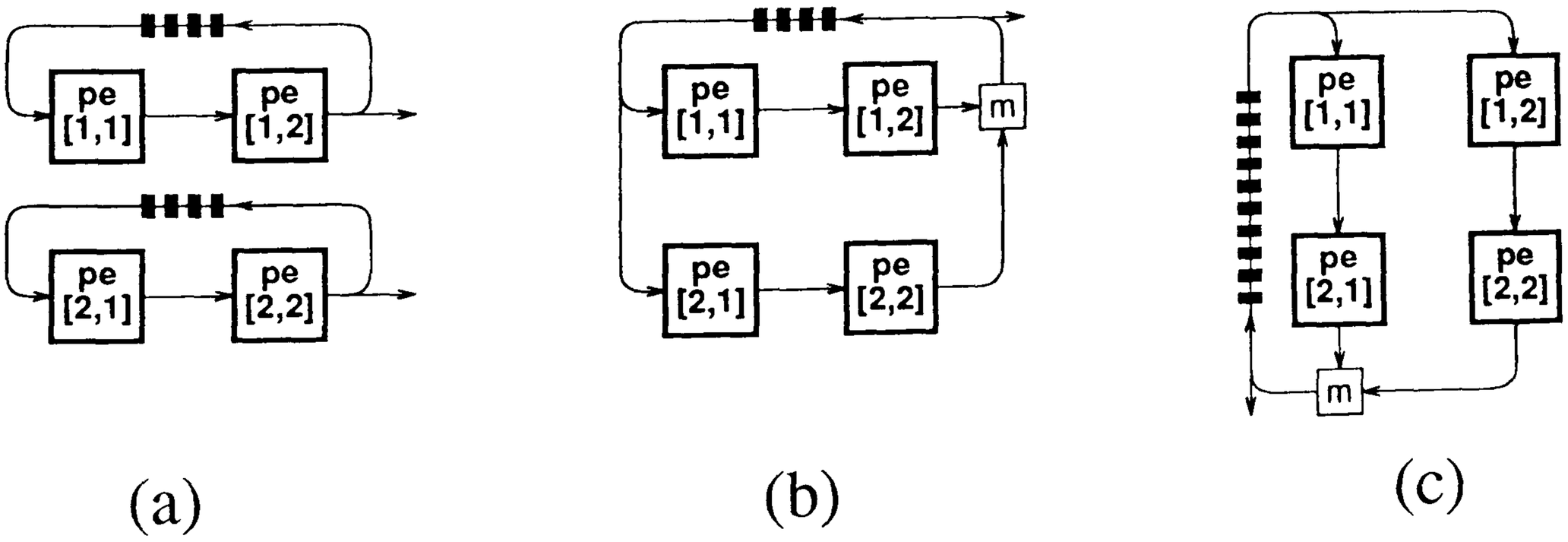


Figure 7.10: (a) Separate feedback loops for recirculation of L sub-matrices, (b) merged L sub-matrix feedback loop (c) merged U sub-matrix feedback loop.

7.3.5 The LU Decomposition Spiral Systolic Array

The configurations of the array for the various subproblems are shown in figure 7.9(f) to (j). There are 4 different types of subproblem as arrays (f) and (j) are effectively the same except that the input and output $[j, k]$ coordinates of data in graph (e) are increased by $[2, 2]$. From figures 7.9(f) to (j) it can deduced that $pe[1, 1]$ operates in all three modes, $pe[1, 2]$ in *mode* 0 and *mode* 2, $pe[2, 1]$ in *mode* 1 and *mode* 2 while $pe[2, 2]$ is just required to operate in *mode* 2.

The complete data communication network for the LU decomposition spiral systolic array is illustrated in figure 7.11. The input matrix C is input to the n direction ports of $pe[1, 2]$, $pe[2, 2]$ and $pe[2, 1]$. The lower triangular matrix L being made available from the right hand edge of the array and the upper triangular matrix U from the lower edge. The timing of input data scheduling, output data availability and mode control changes are easily derived from the internal node numbering (figure 7.9(a) to (e)). Higher dimensioned LU decomposition problems can be solved on this array by forming a longer chain of DDG subproblems and by switching longer delays into the L and U feedback links.

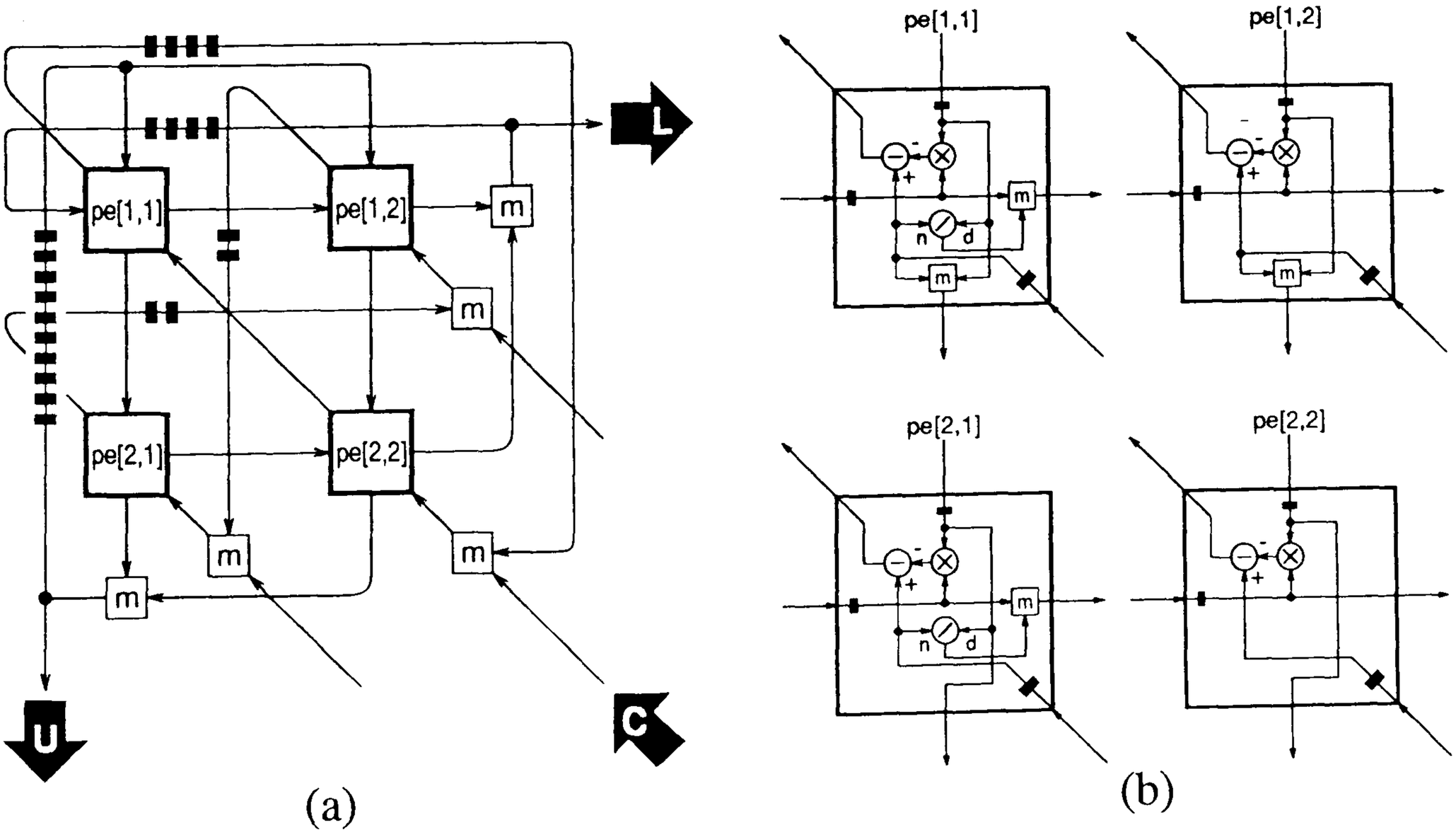


Figure 7.11: (a) Data communication network for LU spiral systolic array, (b) PE arithmetic layout.

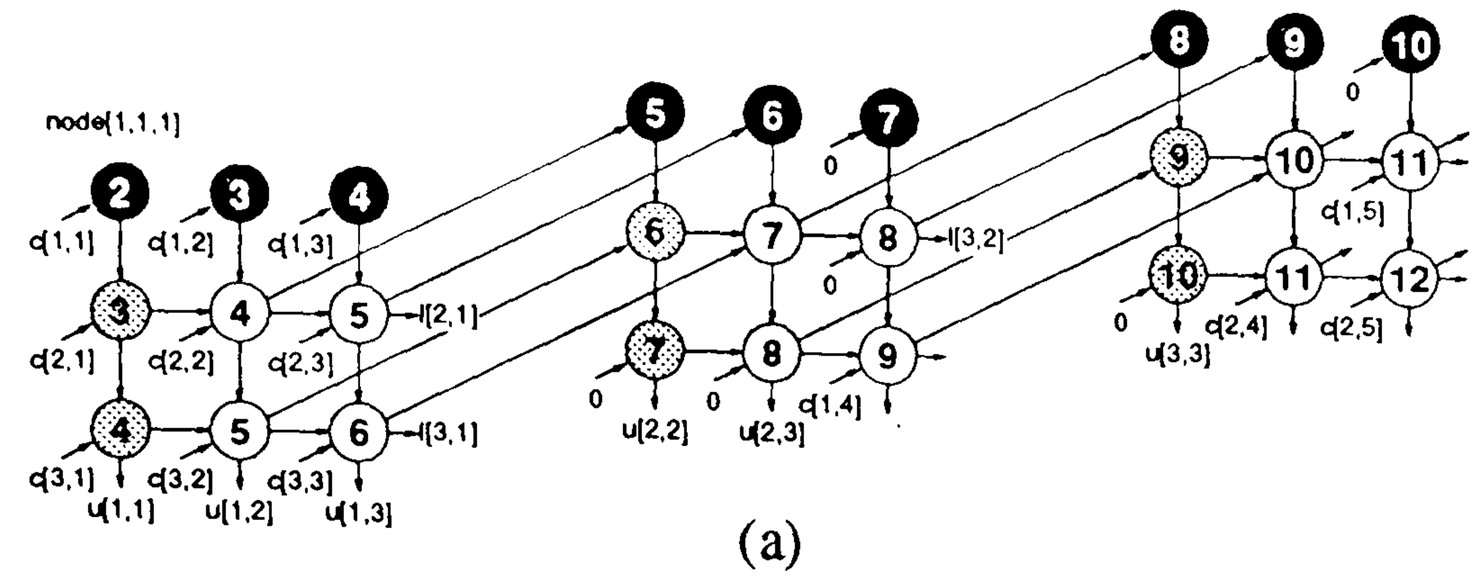
7.3.6 Extending the Spiral Systolic Array Dimension

The partitioning of the LU DDG using $s = 2$ results in a chain of pipelined subproblems. When large model orders are considered, the number of subproblems, and thus the length of the chain, are increased. This in turn leads to long execution time. Restrictions set by PE operating lag may dictate that more parallelism and less pipelining are required in order to achieve real-time processing capability. Higher degrees of parallelism are achieved by using bigger partitions such as $s = 3$ for example. Figure 7.12(a) to (e) shows the subproblems of the $p = 6$ LU decomposition problem reindexed according to (7.14) with the use of $s = 3$ partition. The graph timing function allows for the loading of inputs through two previous n planes and therefore operation in DDG(a) starts in $node[1,1,1]$ at $t = 2$. Figures 7.12(f) to (j) show the arrays produced by $[1,1,1]$ projection of the subproblems in (a) to (e). Chaining of the subproblems results in a total execution time of 50 clock cycles. DDG communication requirements dictate that spirals carrying an extra three delays are required from $pe[1,2]$ to $pe[3,1]$, $pe[1,3]$ to $pe[3,2]$, $pe[2,1]$ to $pe[1,3]$ and $pe[3,1]$ to $pe[2,3]$. Another spiral connection from $pe[1,1]$ to $pe[3,3]$ carries an extra six delays.

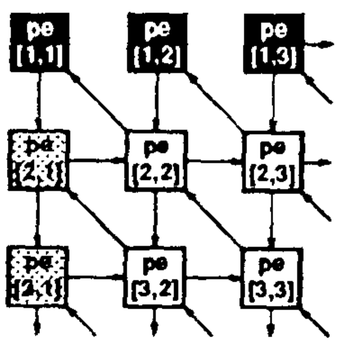
7.3.7 Comparison of LU Decomposition Methods

The design of the LU decomposition spiral systolic array has been approached using the DDG methodology in this section in order to address the points which were made after study of the LU spiral systolic array designed by Navarro et al. [28]. The result was a more coordinated design strategy made entirely on the basic DDG principles.

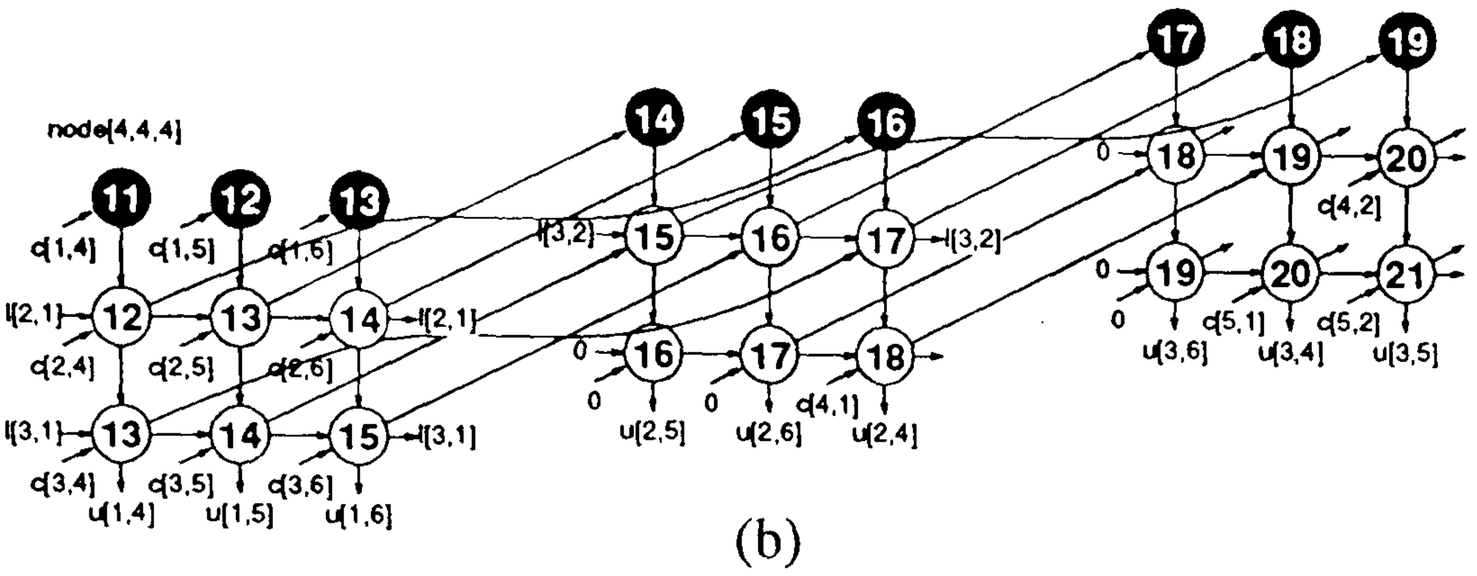
The starting point for the DDG design was to simply partition the LU decomposition $p = 4$ DDG into 5 subproblem DDGs (figure 7.4). This directly compared to the textual computation algorithms in Navarro's method (7.2) to (7.6).



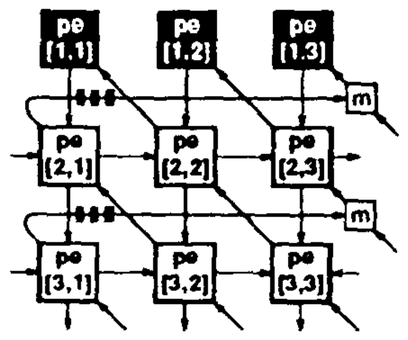
(a)



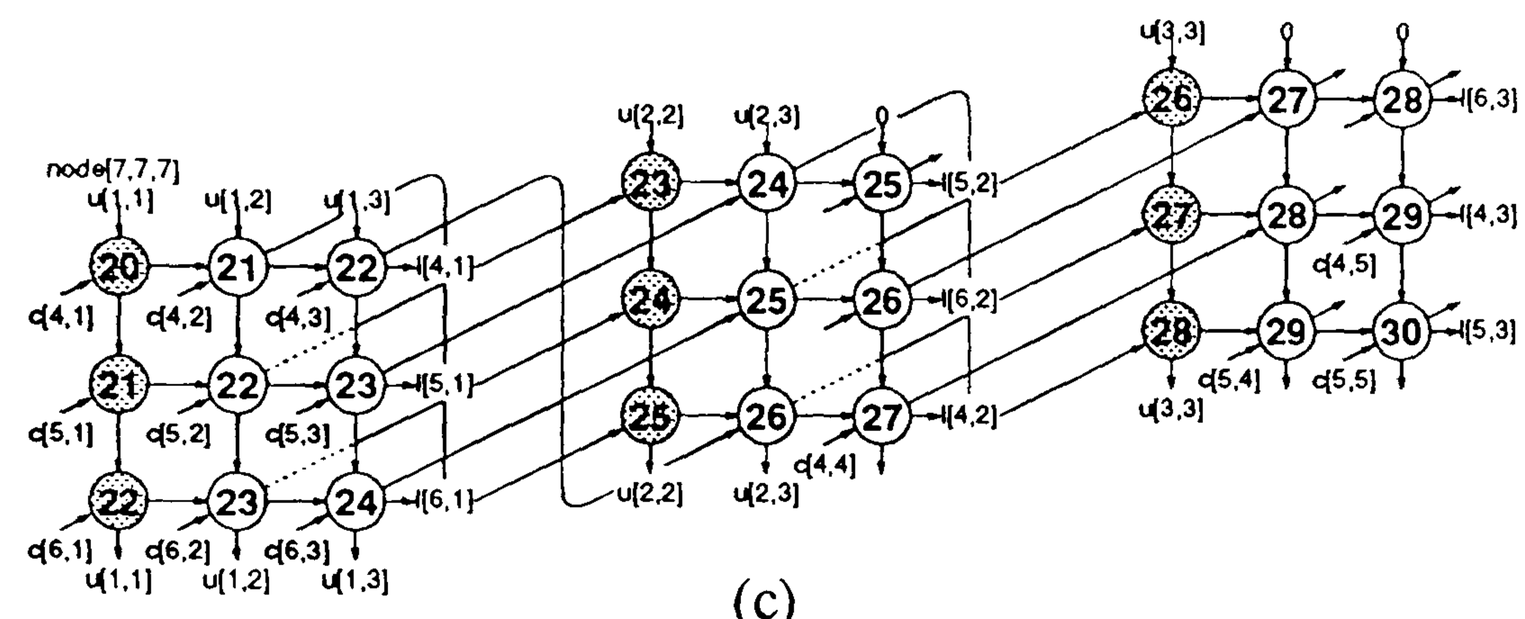
(f)



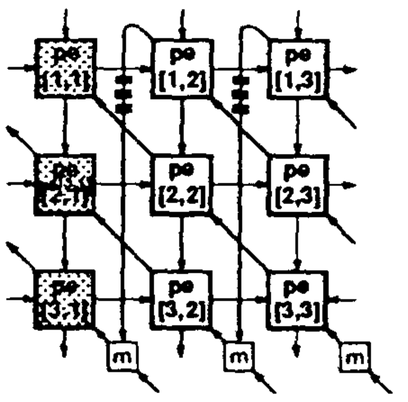
(b)



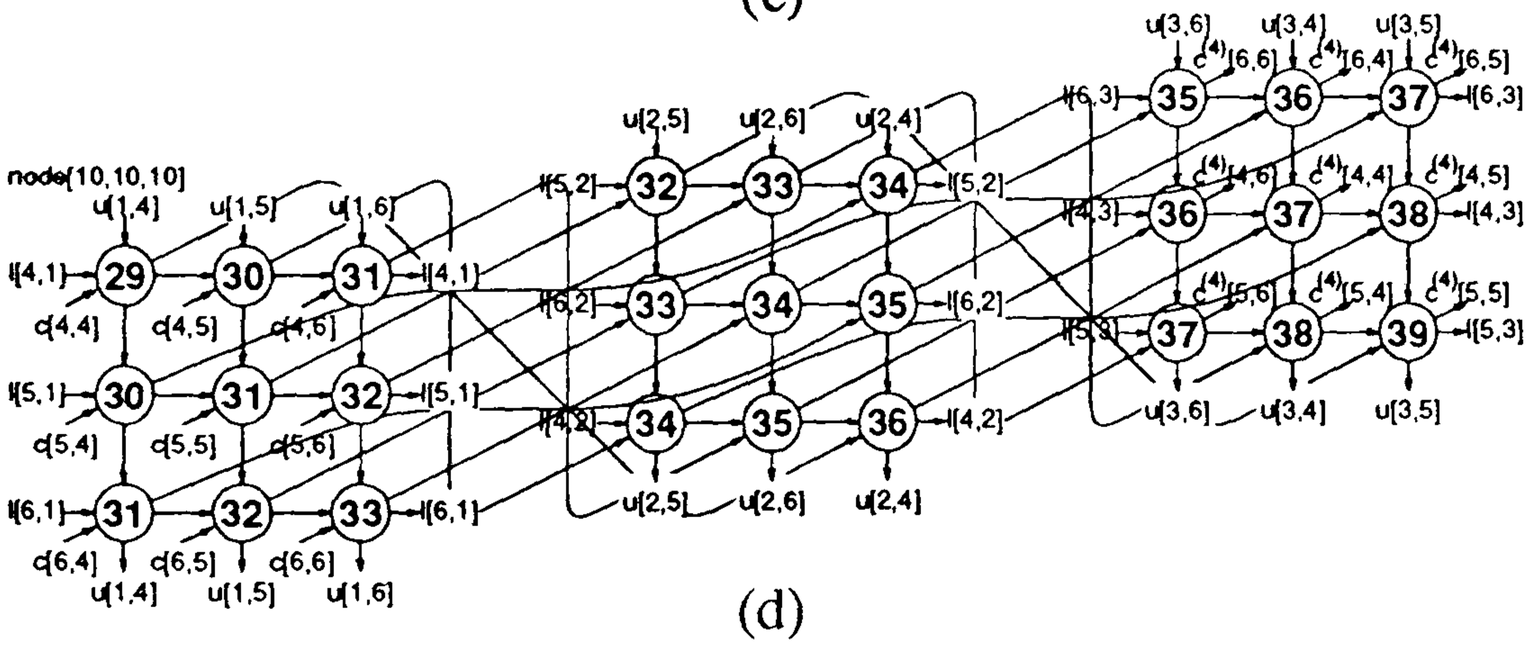
(g)



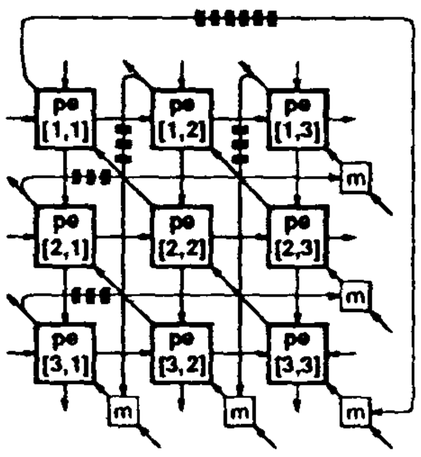
(c)



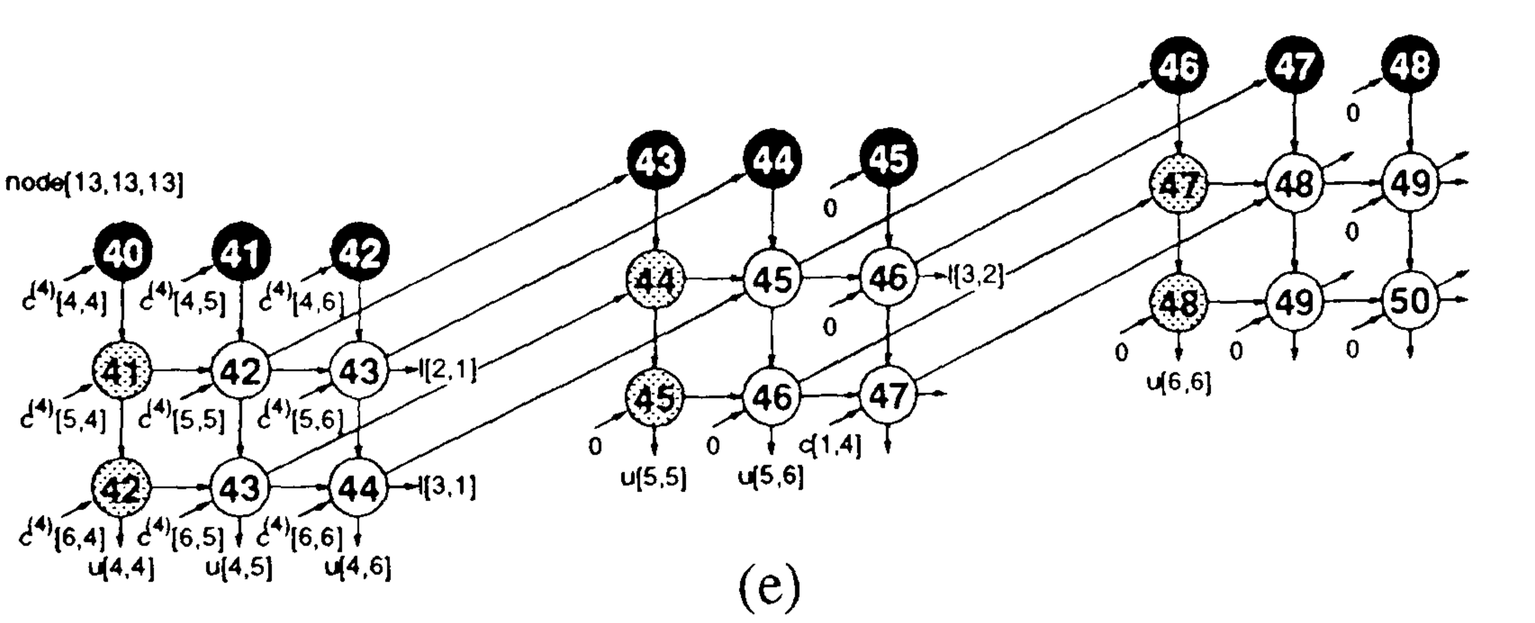
(h)



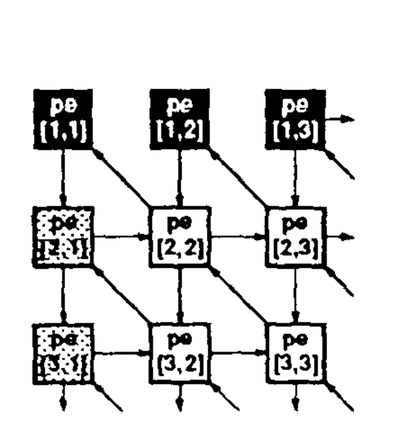
(d)



(i)



(e)



(j)

Figure 7.12: (a) to (e) Reindexed subproblem DDGs for the $p = 6$ Cholesky decomposition problem using $s = 3$ partition and (f) to (j) the spiral systolic arrays formed by the $[1, 1, 1]$ projection vector.

A common subproblem DDG upon which all of the subproblem DDGs could represented with appropriate node function changes was identified (figure 7.5(a)) and spiral reindexing of this common subproblem DDG (figure 7.7(a)) was shown to improve the efficiency of the systolic array produced by $[1, 1, 1]$ projection. The spiral reindexing of the common subproblem DDG was equivalent to the TBP and DBT partitioning methods (7.10) used in Navarro's approach [28]. This is where the advantage of the DDG approach first became apparent, as, since all DDG subproblems were able to be expressed on the same common subproblem DDG representation, then after reindexing, all subproblem DDGs were all able to be expressed on a common subproblem reindexed DDG. In comparison to this, the various TBP and DBT algorithms, were selected as appropriate to the type subproblem [28] and the design route did not therefore remain standard. The reindexed subproblem DDGs were then mapped into the same spiral architecture with programmable PE functions, using the established projection method. The TBP and DBT transformed matrices however only really provided information on the data flows into a pre-determined spiral architecture for which PE functions were selected from a standard set and it was not clear how to schedule data or indeed to which port data should be input. The end results were that the DDG method required only 3 different types of PE operation (7.9(f) to (j)) compared to 7 in Navarro's method (figure 7.3), and the need for bi-directional data ports was eliminated with use of the DDG method.

Chaining of subproblem DDGs was then used to obtain information pertaining to the timing of data flows for all subproblems together. Don't care node states were utilised in order to minimise the time delay between the execution of consecutive subproblems in the chain. Study of data output flows and input data scheduling between different subproblems revealed that it was possible to recirculate the values of L and U sub-matrix elements back into array with the use of the two systolic feedback links (figure 7.10(c)). This removed any need for memory addressing during the computation, as all that is required was a simple

multiplexing function to route one of two outputs into each feedback loop.

Differences in the PE arithmetic functions for certain subproblem modes arise from the definition of the original DDG in figure 4.12. This DDG takes into account the fact that elements on the leading diagonal of the L matrix are ones removing a number of multiplications from the top rows of each n plane. However, when the PEs of the spiral array which are affected by this simplification are programmed in different modes it turns out that multiplication units are required anyway and so there is only a cost saving in power consumption (figure 7.11(b)). The DDG designed array has the disadvantage of requiring two division units, one in $pe[2,1]$ the other in $pe[1,1]$ (figure 7.11(b)) compared to just one in $pe[1,1]$ in the referenced array (figure 7.3). This is easily remedied by changing the top left nodes of each n plane of the DDG in figure 4.12 to output the reciprocal of their input and by replacing the dividers in the *mode* 1 nodes with multipliers. This means that the reciprocal of the leading diagonal of U gets output from the spiral array but this actually is an advantage as it removes a divider from the back substitution systolic array to which these values are subsequently fed. The arithmetic unit PE cost from both of the methods therefore can be made approximately equal but control of the DDG designed array PEs is simpler due to the smaller number of PE mode configurations.

In conclusion, the spiral array produced using the DDG approach displays several distinct advantages over that designed by Navarro et al.[28] as summarised above. Translation of the DBT and TBP methods proposed by Navarro et al. [28] into the DDG design methodology has enabled a more structured approach to the complete design of a problem size independent LU decomposition array. Having considered the DDG design of the LU decomposition, problem size independent, systolic array, a platform has been established to lay out the route to the design of a small, spiral, systolic architecture for Cholesky decomposition,.

7.4 Cholesky Decomposition Spiral Systolic Array Design

Cholesky decomposition displays superior numerical stability over the LU decomposition when considering systolic array implementation as concluded in chapter 5 and this becomes an increasingly important advantage at higher model orders where the system of equations is likely to become more ill-conditioned [27]. For problems with high dimension the number of computations for the LU decomposition is approximately double the number required for the Cholesky decomposition, a factor which is very significant since the number of computations, dependent upon p^3 [27], is greatly increased over the lower model orders. Halving the number of computations effectively halves the number of subproblems formed by partitioning and consequently a problem size independent Cholesky decomposition could run in approximately half the time of the LU decomposition spiral array if the systolic clocking speeds for each are the same.

It is therefore important to use a problem size independent systolic array for Cholesky decomposition rather than LU decomposition when considering filter parameter computations with model orders up to $p = 30$. The design of a problem size independent systolic array for Cholesky decomposition, which has not been previously considered using either TBP/DBT or graphical partitioning methods, is investigated in this thesis for the first time. The DDG based design principles, laid out in the previous section, which led to the LU decomposition spiral systolic array shown in figure 7.11 are applied to the Cholesky decomposition algorithm in this section. It is shown that the standard spiral systolic architecture previously used is not suitable for Cholesky decomposition. A modified reindexing scheme is therefore proposed and this results in a novel, problem size independent, Cholesky decomposition systolic array.

7.4.1 Cholesky Decomposition DDG Partitioning

A similar DDG partitioning analysis to that used for the LU decomposition is applied to the $p = 6$ Cholesky decomposition algorithm in this section. The aim of this section is to show how the large dimensioned Cholesky decomposition algorithm can be mapped onto a small sized spiral architecture which utilises programmable PE functions.

The Cholesky decomposition model order $p = 6$ DDG may be simply derived from the $p = 4$ DDG in figure 4.3(a) by simple extension of two more j planes as shown in figure 7.13, giving an indication of the increase in computational burden for an increase in model order of 2. A problem size dependent array which can be derived from the $p = 6$ DDG by $[1, 1, 1]$ projection contains 21 PEs. When partitioning the Cholesky decomposition algorithm onto a spiral systolic array, to reduce the number of PEs, the question arises as to what is the optimal size of the array to suit the application. This is ultimately decided by the allowed real-time operational period and the speed of underlying hardware in the processors to give the precision of data necessary for obtaining results at an adequate accuracy.

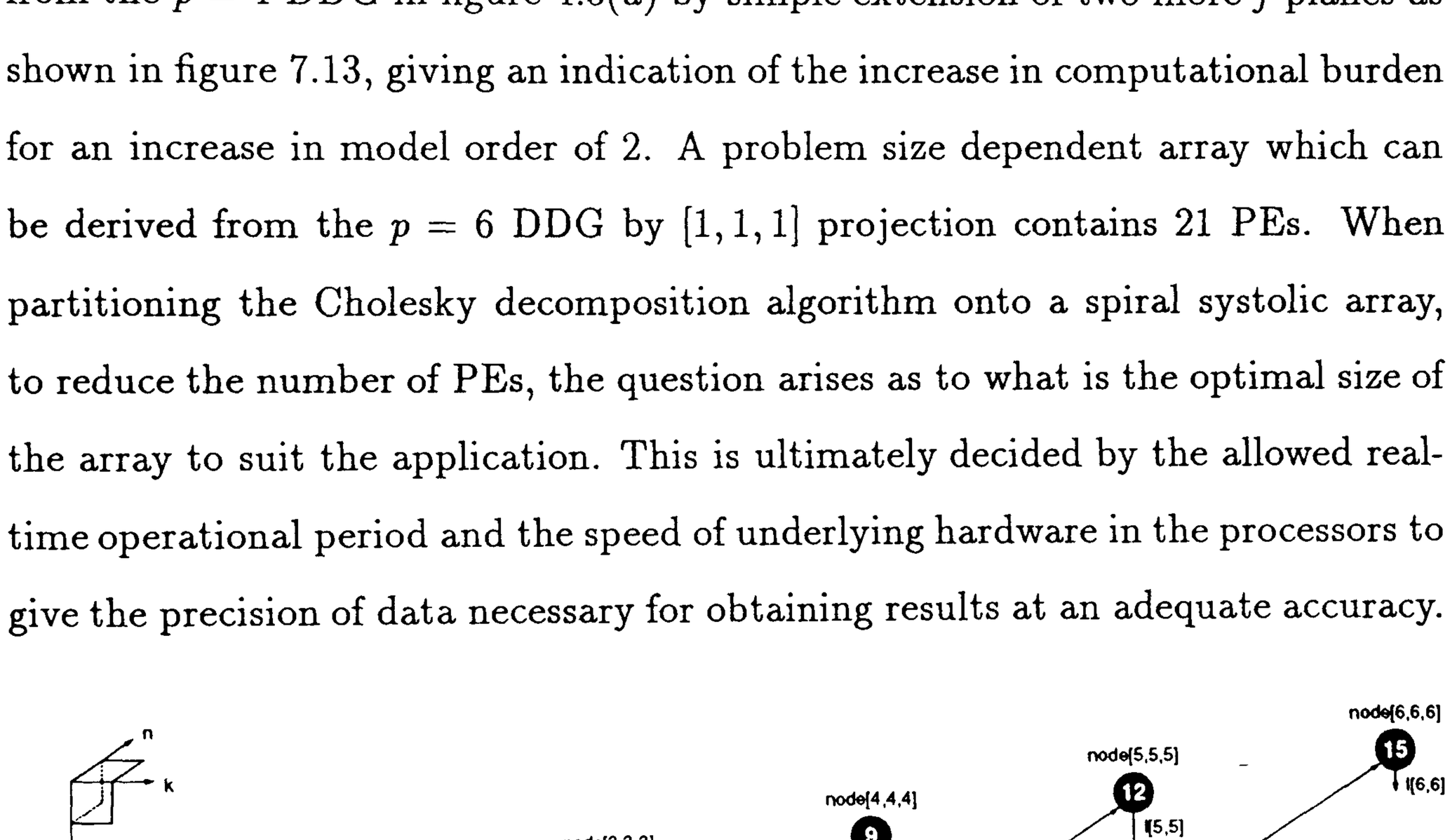


Figure 7.13: The Cholesky decomposition DDG for use the model order $p = 6$ computation, for node functions refer to figure 4.3(b).

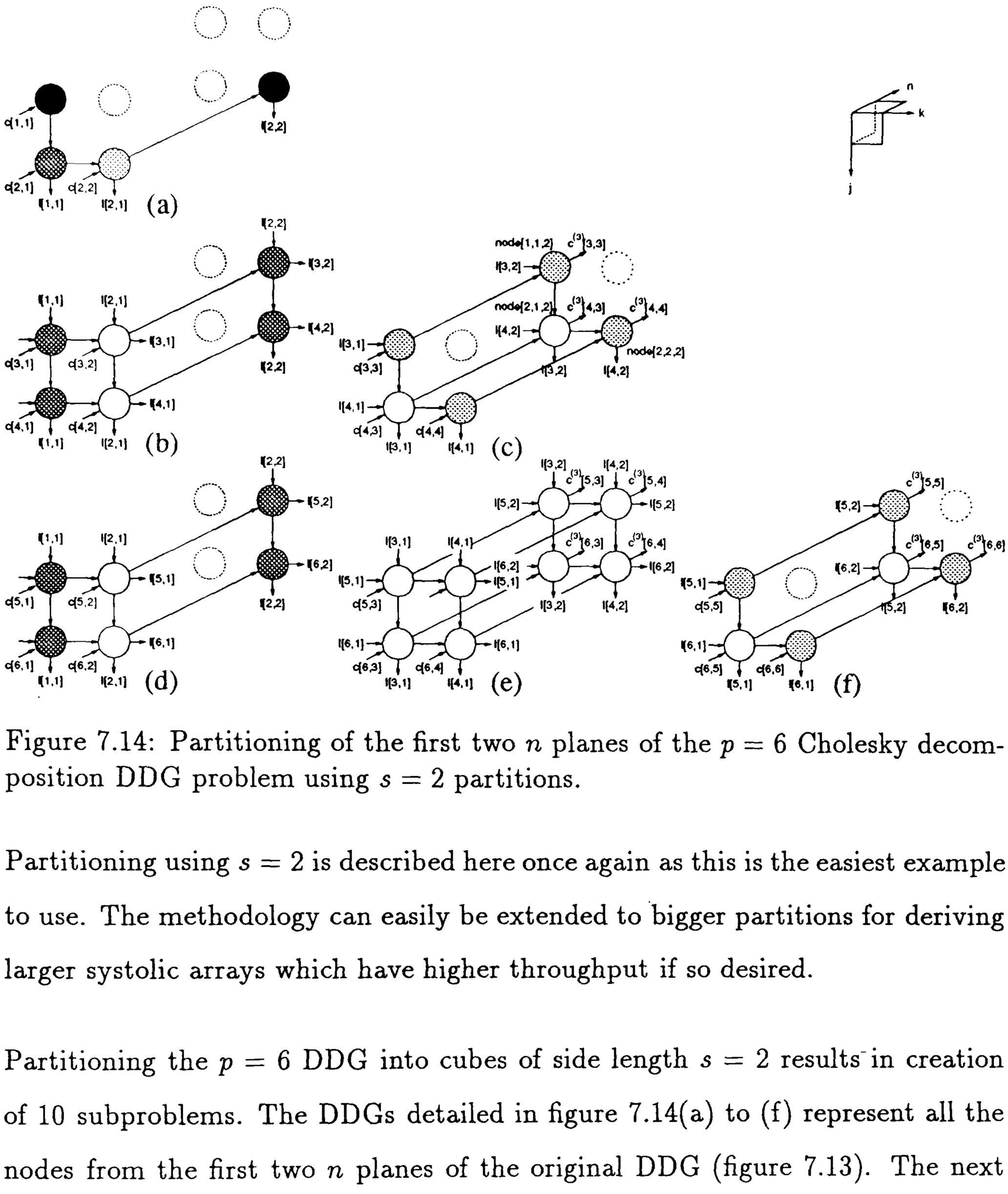


Figure 7.14: Partitioning of the first two n planes of the $p = 6$ Cholesky decomposition DDG problem using $s = 2$ partitions.

Partitioning using $s = 2$ is described here once again as this is the easiest example to use. The methodology can easily be extended to bigger partitions for deriving larger systolic arrays which have higher throughput if so desired.

Partitioning the $p = 6$ DDG into cubes of side length $s = 2$ results in creation of 10 subproblems. The DDGs detailed in figure 7.14(a) to (f) represent all the nodes from the first two n planes of the original DDG (figure 7.13). The next three subproblems, (g) to (i) (not shown in the figure), which cover the third and fourth n planes are a repeat of subproblems (a) to (c) but with the indices of the l and u elements increased by $[2, 2]$ and the n direction input derived from the n direction outputs of subproblems (c), (e) and (f) respectively. The last subproblem, (j), computes the last two recursions and is the same type as subproblem (a) taking its input from the results produced in subproblem (i). In all there are four different types of subproblem included in this set.

7.4.2 Spiral Reindexing of the Cholesky Common Subproblem DDG

A different spiral reindexing scheme needs to be introduced for this algorithm due to the data dependencies between nodes in certain types of subproblem. The spiral reindexing scheme defined by (7.14) is valid for the LU decomposition subproblems since the operating order of nodes the $n = 2$ plane of each of its subproblems is commutative in the j and k directions. However, for the Cholesky decomposition this does not hold for all of the subproblems as in subproblems of type (c) and (f), in figure 7.14 the node **E** labelled on the common subproblem DDG in figure 7.15(a) must operate before node **G** which also must operate before **H**. These types of operation are therefore not commutative in the j and k directions and therefore an alternative reindexing scheme to that described by (7.14) must be used. It follows from this that the standard spiral architecture used by Navarro et al. is not suitable for Cholesky decomposition.

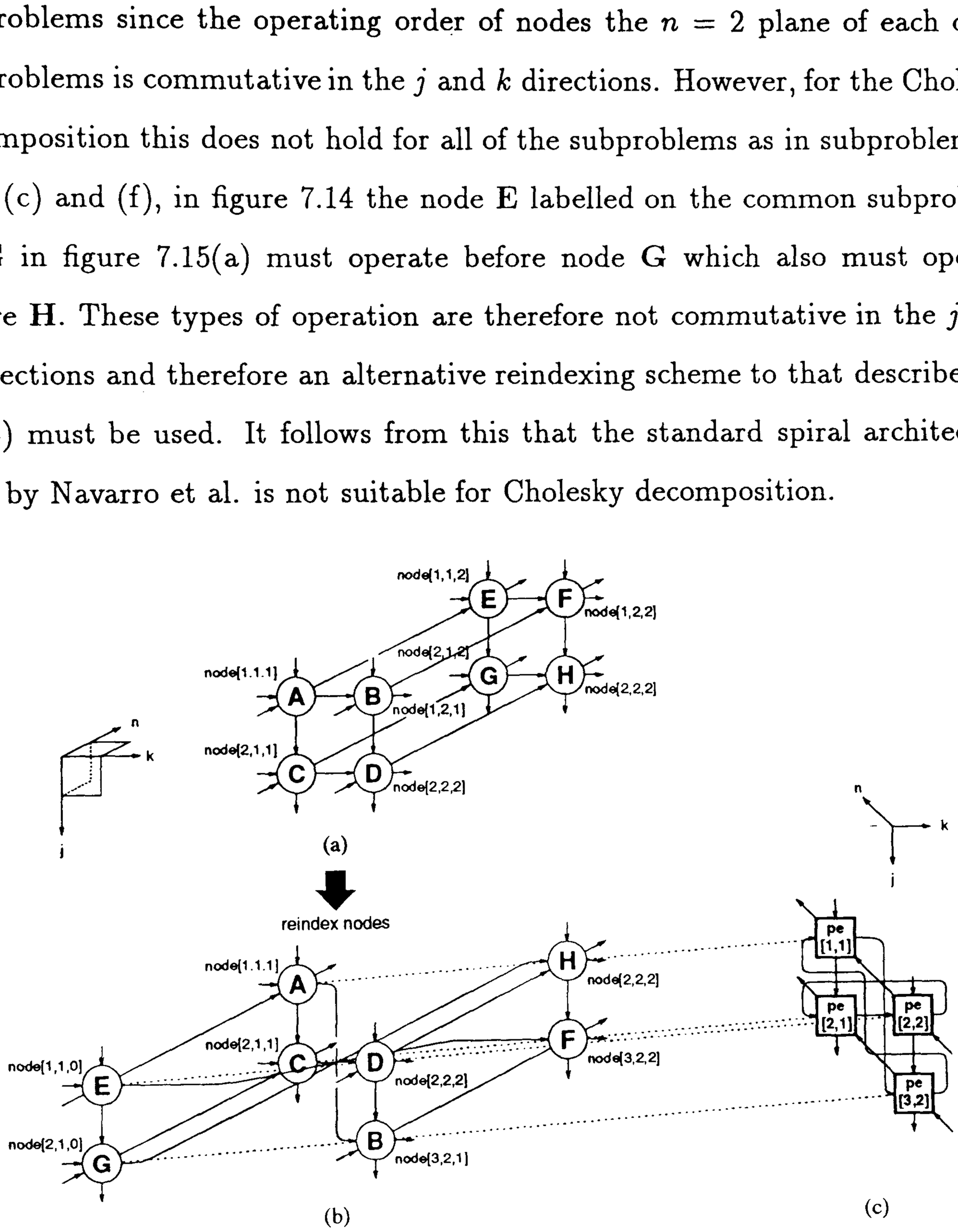


Figure 7.15: (a) Cholesky decomposition common subproblem DDG, (b) effect of reindexing the DDG using the scheme given in (7.15) and (c) systolic architecture produced by $[1, 1, 1]$ projection of the DDG.

The reindexing scheme applied to the Cholesky decomposition must firstly respect the data dependencies of each of type of Cholesky subproblem shown in figure 7.14 and must allow 2 nodes to be mapped into one of 4 PEs of the array formed by projection of the reindexed DDG in the [1, 1, 1] direction. Study shows that the operations of nodes in the planes orthogonal to the k direction are commutative in the j direction when $k = 2$, whilst the ordering is commutative in the n direction when $k = 1$. This is formally stated as

$$node[j, k, n] \rightarrow node[(j - k)_{\text{mod } s} + k, k, k - (k - n)_{\text{mod } s}] \qquad (7.15)$$

and applying this to the Cholesky decomposition common subproblem DDG in figure 7.15(a) produces the reindexed DDG in part (b) of the figure where the operations **B**, **E**, **F**, **G** move from node coordinates [1,2,1], [1,1,2], [1,2,2], [2,1,2] to new locations [3,2,1], [1,1,0], [3,2,2], [2,1,0] respectively. The transformation inserts spiral communications into three of the four k direction inner DDG communication arcs which can be compared to spiralling of the n direction paths in the LU decomposition reindexing scheme. The new spiral systolic architecture produced by [1,1,1] projection, shown in figure 7.15(c), contains 4 PEs each performing 2 node operations and there are three spiral buses..

7.4.3 Chaining and Reindexing of the Cholesky Subproblem DDGs

The DDG reindexing defined by (7.15) can be applied to the subproblems formed by the partitioning of the Cholesky decomposition. The transformed DDGs for the first five subproblems from figure 7.14(a) to (e) are illustrated in figure 7.16(a) to (e). A chained DDG is formed by interconnecting the DDGs end on end so that operation **E** (figure 7.15(b)) in the second subproblem DDG becomes $node[3, 3, 2]$ of the combined graph and so on. The internal node numbering indicates a possible timing strategy for calculation of the subproblems in their alphabetical ordering from figure 7.14.

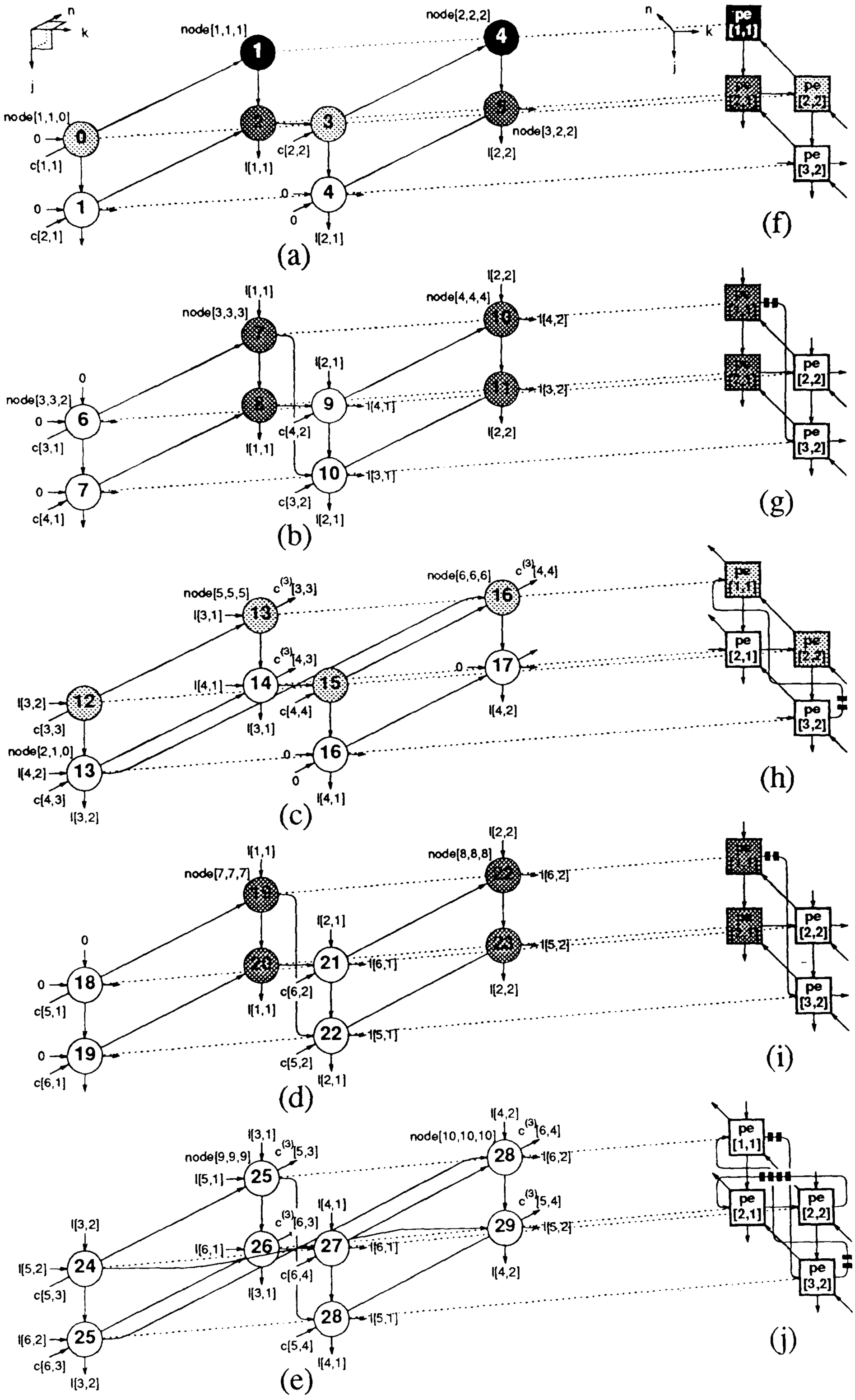


Figure 7.16: (a) to (e) Reindexing of DDG subproblems (a) to (e) respectively in figure 7.14 and (f) to (j) spiral systolic arrays formed by projection by the $[1,1,1]$ vector.

7.4.4 Projection of the Chained DDG into a Programmable Array

Projection of the reindexed subproblem DDGs in figure 7.16(a) to (e) by the $[1,1,1]$ vector results in the spiral systolic arrays shown in figure 7.16(f) to (j), and it can be seen that the arrays are no longer square shaped. A maximum of three spirals are introduced by the DDG reindexing and these are all present in the systolic array shown in part (j) of the figure and carry either 2 or 4 extra systolic registers as indicated. All the types of subproblem can therefore be calculated on such an architecture as that of part (j) providing that the PE functions are made programmable.

(A) Systolic Feedback of L and C Sub-matrices

Feedback links can be used in the j and n directions to reload data back into the array. Data is recirculated in the j direction between DDGs (a) and (b) of figure 7.16 with a total delay of five units so four extra delay elements are needed in the feedback line from $pe[2,1]$ to $pe[1,1]$ and from $pe[3,2]$ to $pe[2,2]$. From subproblems (b) to (d) and from (c) to (e) this delay is increased to eleven clock cycles and so an extra six registers would need to be switched into the feedback loop during this period. When recirculating data from the n direction ports it would be wise to use clock enabling on the feedback loop registers from $pe[1,1]$ to $pe[2,2]$ and from $pe[2,1]$ to $pe[3,2]$. This is because of the large delay of 23 clock cycles between output of $c^{(3)}[3,3]$, $c^{(3)}[4,3]$ and $c^{(3)}[4,4]$ in subproblem (c) to input of subproblem (g). Only ten data words need to be stored on this feedback line during this period so using 22 extra registers would be quite wasteful. Once stored, data in these feedback registers is naturally ordered for input to subproblems (g) to (i) which have the same control requirements as subproblems (a) to (c). Direction n input data to subproblem (j), which is also the same type as (a), can be fed back from subproblem (i).

7.4.5 The Cholesky Decomposition Spiral Systolic Array

The complete spiral array for model order $p = 6$ Cholesky decomposition problem is presented in figure 7.17. Two control lines are required to switch between different processor configuration modes. An additional two more control lines are needed for the multiplexors on the n direction inputs and one is needed to control the j direction input multiplexors. Lower dimension problems can be efficiently solved on this array by simply adjusting the input and control signals. Given that there are a total of 10 subproblems each requiring 6 clock pulses then the total number of clock cycles for complete operation is 60. With the addition of extra delays in the feedback loops, problems with dimension higher than $p = 6$ can also be decomposed.

The DDG design methods can also be used to design larger arrays for higher throughput. Figure 7.18 shows the DDG chain for a $s = 3$ partition array which reduces the execution time to 38 clock cycles for the $p = 6$ problem.

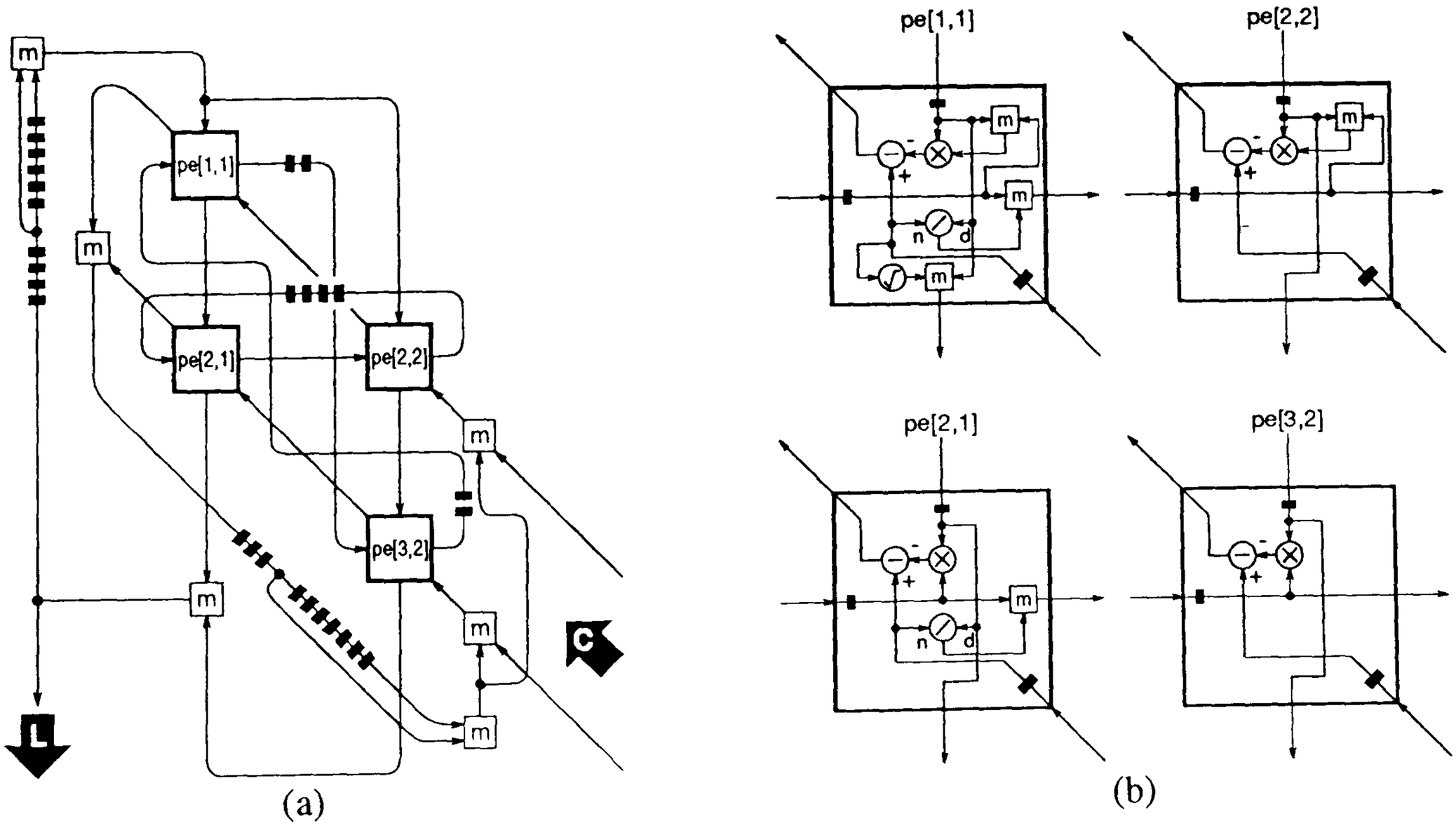


Figure 7.17: (a) Cholesky spiral systolic array, (b) PE arrangement.

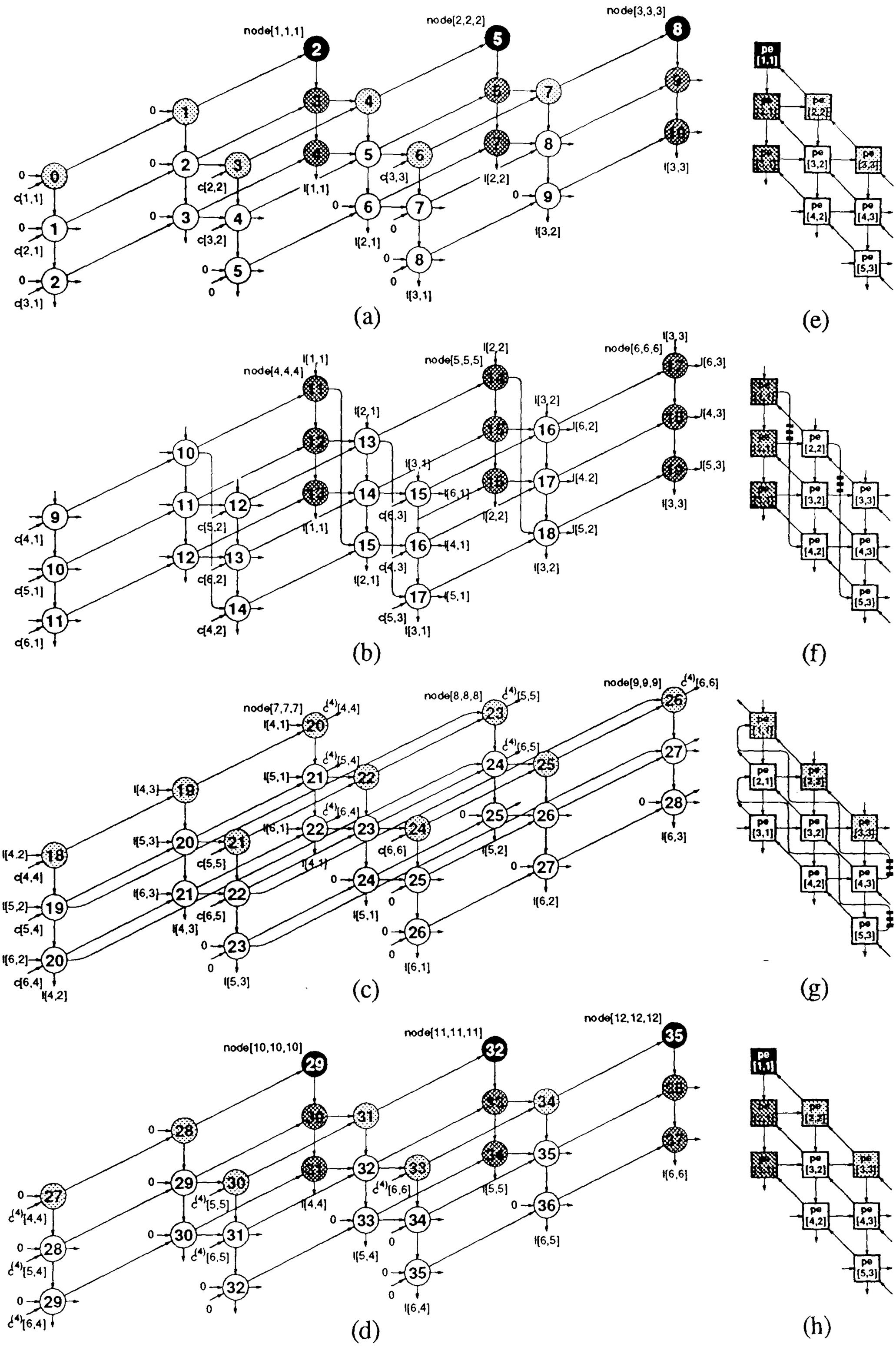


Figure 7.18: (a) to (d) Reindexed subproblem DDGs for the $p = 6$ Cholesky decomposition problem using $s = 3$ partition and (e) to (h) the spiral systolic arrays formed by the $[1, 1, 1]$ projection vector.

7.5 Comparison of Programmable Systolic Arrays

This section considers the cost performances of the Cholesky and LU spiral systolic arrays for various model order problems p and partition sizes s. The cost of the LU array is detailed as this design may be used in other applications where the matrix to be decomposed is not symmetrical. A timing analysis also allows the high model order computation time advantage, resulting from the much reduced number of clock cycles, of the Cholesky decomposition method to be shown.

7.5.1 Method of Cost Comparison

(A) PE Cost

Tables 7.3 and 7.4 give the costs of the programmable LU and Cholesky decomposition array PEs, shown in figures 7.11(b) and 7.17(b) respectively, in terms of their module usage.

w bit module	type of PE (figure 7.11(b))			
	$pe[0,0]$	$pe[1,2]$	$pe[2,1]$	$pe[2,2]$
subtraction	1	1	1	1
multiplication	1	1	1	1
division	1	0	1	0
register	3	3	3	3
multiplex	2	1	1	0

Table 7.3: Hardware usage for LU array PEs in terms of module usage.

w bit module	type of PE (figure 7.17(b))			
	$pe[0,0]$	$pe[2,1]$	$pe[2,2]$	$pe[3,2]$
subtraction	1	1	1	1
multiplication	1	1	1	1
division	1	1	0	0
square root	1	0	0	0
register	3	3	3	3
multiplex	3	1	1	0

Table 7.4: Hardware usage for Cholesky array PEs in terms of module usage.

When deriving the PE cost in an arbitrary sized spiral systolic array partitioned by s the same PE configurations as those described in tables 7.3 and 7.4 can be used. In general the LU and Cholesky designed arrays both contain s^2 PEs. Also for the arrays of both decomposition methods, $pe[1,1]$ is the same type of PE as $pe[1,1]$ in the $s = 2$ arrays and $pe[j,1]$ $(2 \leq j \leq s)$ are all of the type $pe[2,1]$ for each respective method. In the LU arrays $pe[1,k]$ $(2 \leq k \leq s)$ need to be same as $pe[1,2]$ in table 7.3 while in the Cholesky arrays $pe[j,j]$ $(2 \leq j \leq s)$ perform the function of $pe[2,2]$ in table 7.4. The $(s-1)^2$ remaining PEs of both decomposition arrays perform the inner product step function of $pe[2,2]$ in table 7.3 and $pe[3,2]$ in table 7.4. Note that for this type of partitioning method the number of PEs is independent of model order unlike the partitioned array for matrix element calculation described in the previous chapter (6.2).

(B) Spiral Bus Cost

The number of spiral buses and the number of delays per spiral bus is dependent on the partition size s. In all, for both Cholesky and LU methods an array has $2s-2$ spiral buses each carrying s extra delays plus one more bus which requires $2s$ systolic delays. This leads to a total of $2s^2$ systolic registers on the spiral buses for each decomposition method. In the LU array $2s-1$ multiplexors are also included to switch between the spiral and input C matrix buses.

(C) Feedback Bus Cost

The cost of the feedback buses is dependent on the model order as well as the size of partition. Due to the PE pipelining period $\alpha = 3$ in the arrays for both decomposition methods one feedback bus can be used to route from 3 consecutive outputs along an array edge to the corresponding inputs along the opposite edge.

The number of feedback buses required around a decomposition array is:

$$N_{fb\ bus} = 2.ceil\left(\frac{s}{3}\right) \tag{7.16}$$

and feedback bus utilisation is maximised when three is an exact factor of the partition size partition size e.g $s = 3$, $s = 6$. Also a number of multiplexors are required to route different outputs onto each bus. The total number of these for each array is:

$$N_{fb\ mux} = 2.trunc\left(\frac{2s}{3}\right) \tag{7.17}$$

where *trunc* rounds to the nearest integer towards zero.

The number of registers and multiplexors on each type of bus is specific to the decomposition method, model order, partition size and the order in which its subproblems are executed. Firstly consider the LU decomposition. To avoid interleaving with blocks of zeroes, the subproblem ordering restriction is for subproblems working on the same covariance matrix element updates to execute consecutively so that data is spirally recirculated in the n direction whenever possible. The ordering scheme employed here also gives precedence of subproblem execution in the k direction over the j direction. On each of the k direction feedback buses $q-1$ different systolic delays need to be switched between, where

$$q = ceil\left(\frac{p}{s}\right) \tag{7.18}$$

The lengths of each delay, that is the number of systolic word registers, range from $2s$, increasing in blocks of $3s$ to $2s + 3s(q-2)$. The switching may be implemented with $(q-2)$ multiplexors as shown in figure 7.4.

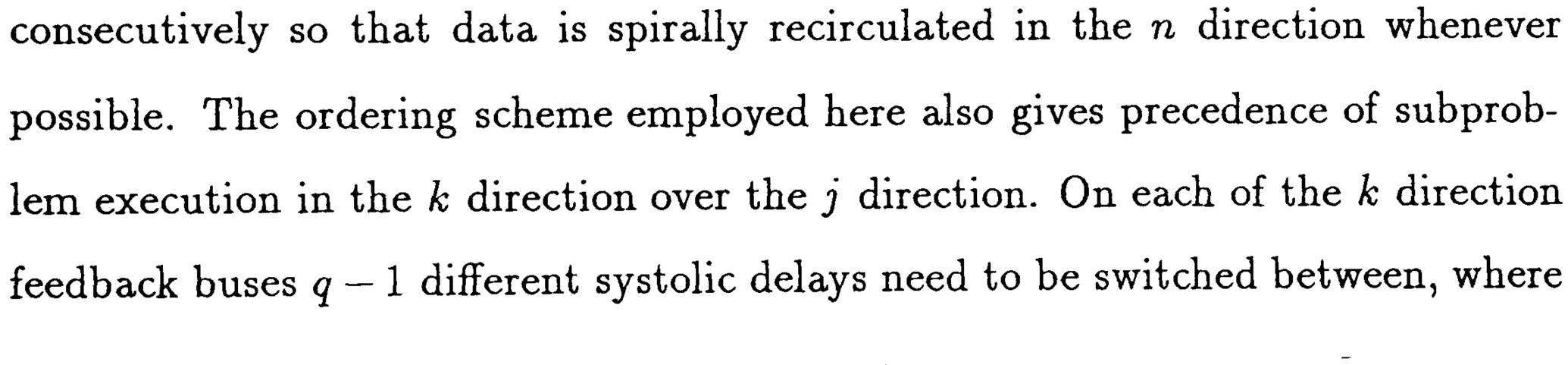


Figure 7.19: k direction feedback bus for LU spiral arrays.

The same type of configuration can be used for the j direction LU feedback buses but the delays are increased. The total delays on these buses range from $3s(q-1)-s$ followed by a total of $q^2/2-q/2-1$ FIFO blocks each of length $3s$ word registers and also $q^2/2-q/2-1$ multiplexors for switching delay time.

In the Cholesky decomposition the consecutive ordering of the subproblems in the k direction arises from the k direction spirals and the subproblem execution in the j direction is given priority over the n direction. The general configuration of the j direction feedback bus is the same as for that of the k direction in the LU method but with different control timing on the multiplexors. The n direction feedback buses range from $2s$ registers, adding on $q^2/2+q/2-3$ length $3s$ FIFOs and $q^2/2-q/2-1$ multiplexers are needed to select various delay outputs. This bus requires the same amount of hardware as the j direction feedback bus in the LU design but the delays at different points on the feedback bus are taken.

7.5.2 Results of Cost Analysis

Figure 7.20 plots word-parallel and bit-serial costs of the complete LU and Cholesky decomposition spiral systolic arrays versus partition size s (s^2 gives the number of PEs) for a range of model orders. For each model order curve, the Cholesky arrays show slightly greater hardware costs over the LU arrays due to

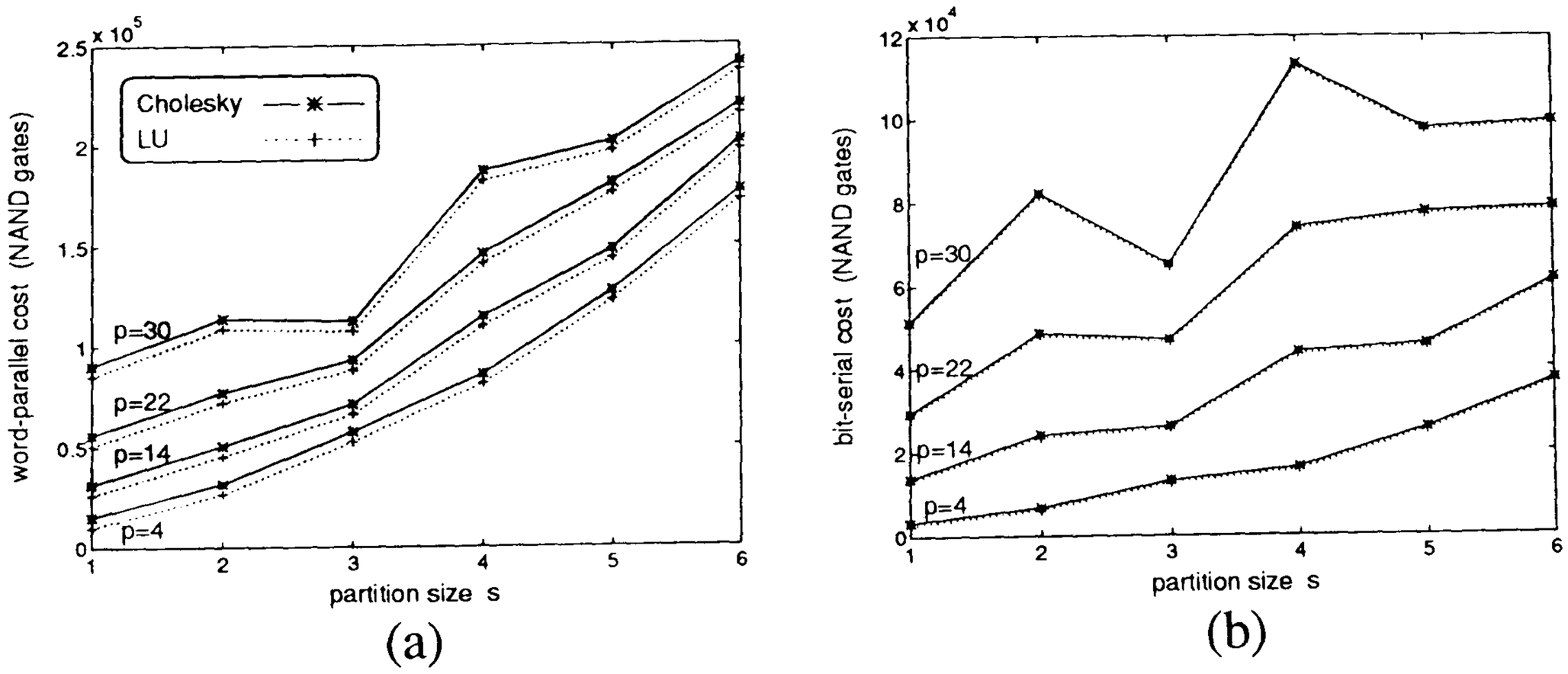


Figure 7.20: Cost for (a) word parallel, (b) bit-serial approaches versus partition size s for a word-length of $w = 16$ bits and model orders $p = 4$, 14, 22, 30.

the inclusion of the extra square root unit in the Cholesky which is not needed in the LU. The results for the single PE $s = 1$ partition show lowest cost. In determining of the results for the single PE array it is assumed that the PE is active on consecutive clock cycles. This is possible as communication with other PEs, which sets the processor pipeline period α to 3 in the larger arrays, is not necessary. The effect of this continual operation on the hardware is to reduce the number of systolic registers in the feedback loop by a factor of three, eliminating much redundant storage which arises if the single processor array is clocked 1 in every 3 cycles.

The PE cost difference between the word-parallel and bit-serial approaches reduces as the partition size and thus the number of PEs is decreased. Therefore the plots for both approaches tend to converge with smaller partitions since the hardware cost of the spiral and feedback buses is independent of approach. In both approaches the $s = 3$ partition, which combines efficient use of feedback in a reasonably sized array, displays low cost for the highest model order curves. For the lower model orders it becomes more cost efficient to use smaller arrays. Comparison of figure 7.21 with figure 7.1 shows the advantage of the spiral over problem size dependent systolic arrays with 90% word-parallel and 60% bit-serial cost reduction when partitioning high model order arrays with $s = 6$.

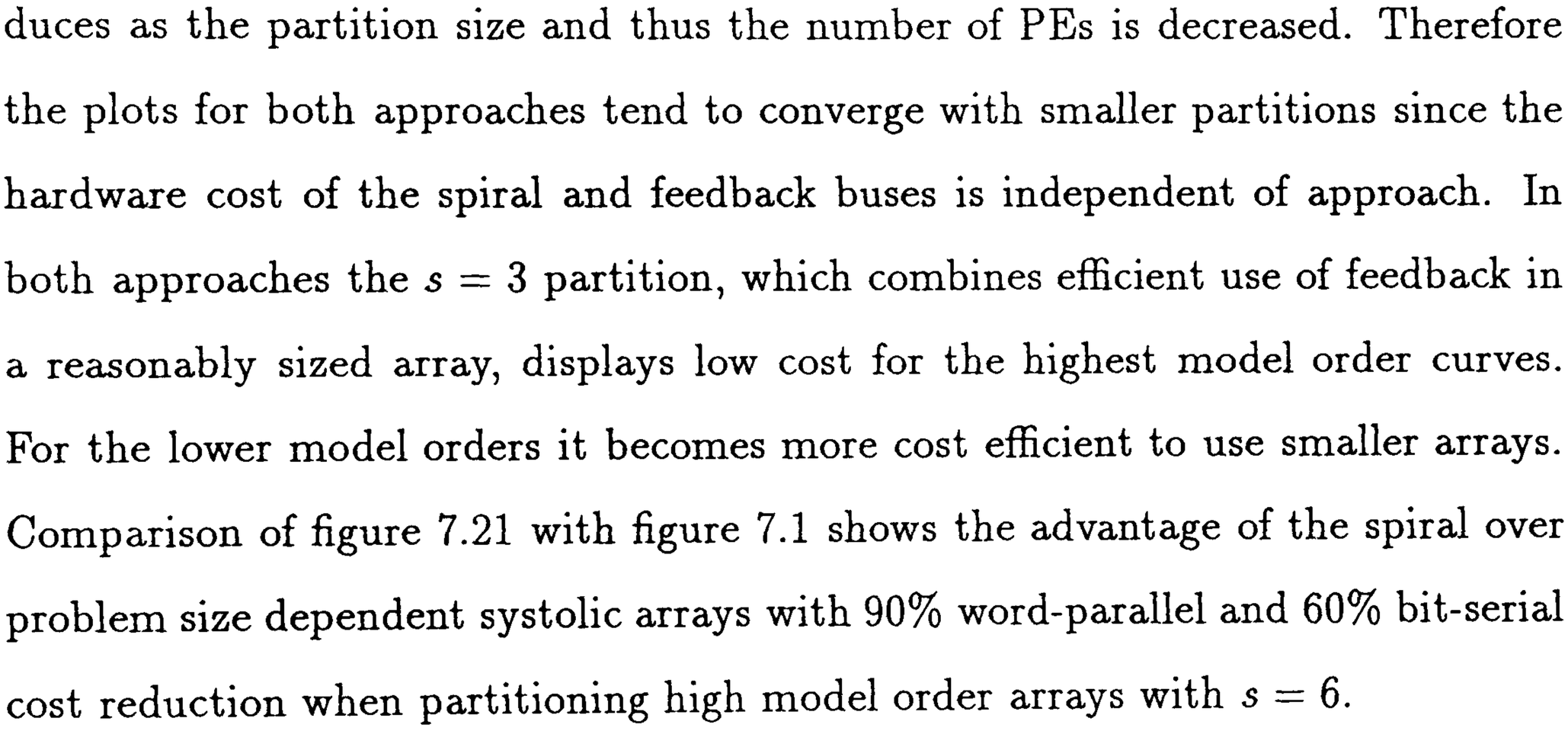


Figure 7.21: Cost for (a) word parallel and (b) bit-serial approaches versus model order p for a fixed word-length of $w = 16$ bits for partitions $s = 1, 3, 4, 6$.

7.5.3 Execution Time

The reduced cost of the spiral systolic arrays over the problem size dependent arrays is not without its drawbacks as the execution time in terms of clock cycles increases dramatically as the partition size decreases. Study of the geometry of decomposition DDG for Cholesky decomposition (figure 4.3(a)) shows that the number of subproblems DDGs is:

$$N_{sub}(Cholesky) = \frac{q^3}{6} + \frac{q^2}{2} + \frac{q}{3} \tag{7.19}$$

while partitioning the LU DDG (figure 4.12(a)) results in creation of:

$$N_{sub}(LU) = \frac{q^3}{3} + \frac{q^2}{2} + \frac{q}{6} \tag{7.20}$$

subproblems where q is defined by (7.18). The time period between execution of two consecutive subproblems is the product of the PE pipeline period $\alpha = 3$ and the partition size s enabling the total number of clock cycles for a specific array to be calculated. The timing results are shown in figure 7.22 as a function of the partition size. The Cholesky decomposition spiral array is superior to the LU array in that it requires approximately half the number of clock cycles when s and p are same in each case. Despite the continuous clocking of the single PE array the results show that $s = 1$ partition requires the greatest execution times.

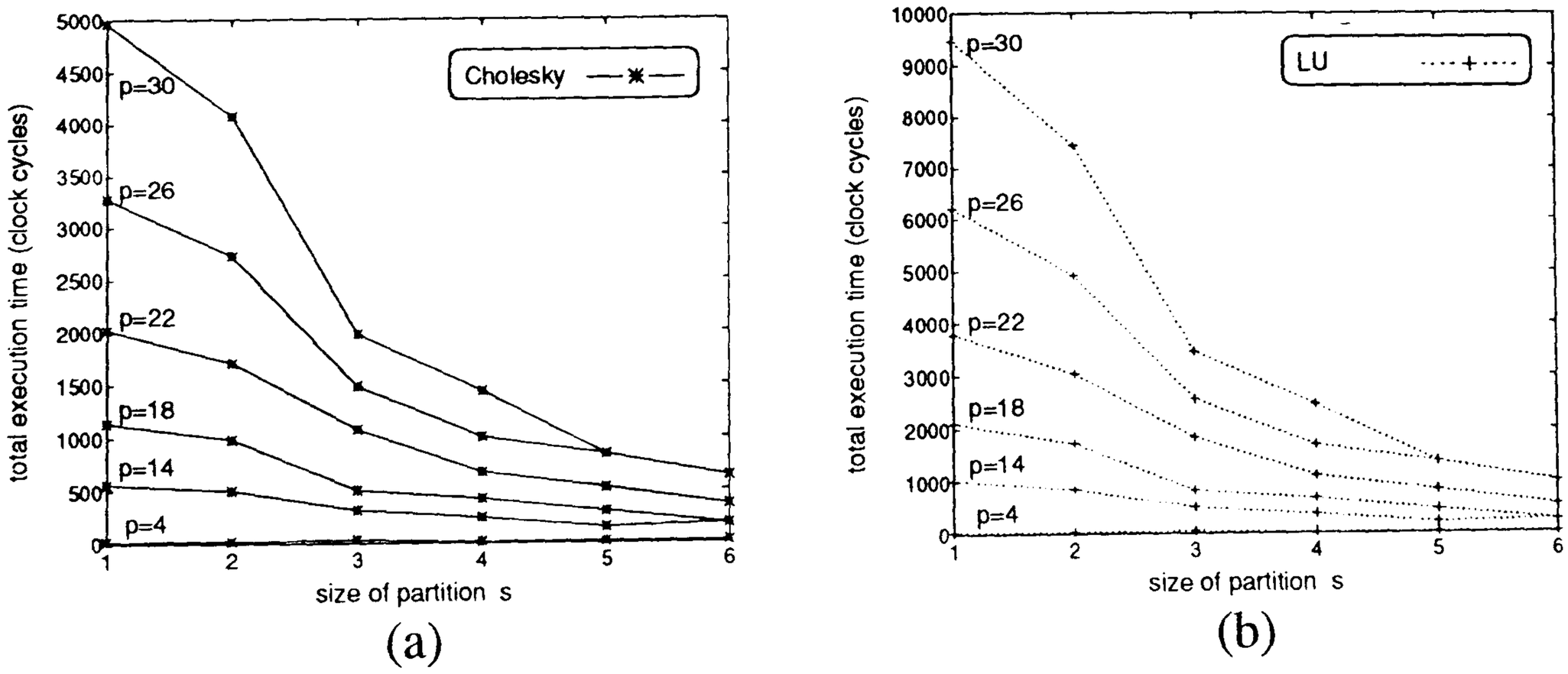


Figure 7.22: Systolic clock cycle execution times of (a) Cholesky and (b) LU decomposition spiral versus partition size s. A range of even model order curves are shown with the lowest curves corresponding to $p = 4$ and the uppermost representing $p = 30$.

7.5.4 Design of a Programmable Array

Since the hardware cost difference between the *LU* and *Cholesky* spiral arrays is very small the Cholesky decomposition should be chosen due to the superior execution time where the required number of clock cycles is almost halved. The Cholesky scheme also offers improved numerical stability over *LU* decomposition (chapter 5) so that shorter word-length can be used when solving equivalent problems - representing a substantial cost saving for the larger spiral arrays.

The selection of the partitioning size s and consequently the number of PEs s^2 in the Cholesky spiral systolic array should be based on the highest model order problem that needs to be calculated and the allowable execution time, given the performance of underlying arithmetic modules for the adopted approach. Programmability can then be achieved since lower model order problems can be treated as subproblems of the highest order solution.

A decomposition calculation can be performed in parallel with the matrix element calculation for the next window of data and therefore the allowable execution time becomes dependent upon the data window length. The minimum window length $T_{min}(win)$ for the highest model order problem sets the performance requirement of PEs in a spiral array, and is determined by the product of the shortest sampling period of the input data with the lowest number of data samples N_{min} within the window. In general arrays with fewer PEs present lower overall cost but have the disadvantage of requiring a greater number of clock cycles $N_{clk}(spiral)$ thereby limiting the maximum speed of operation to a greater extent than in a larger array. The minimum clock cycle length $T_{min}(spiral)$ of the spiral array is likely to dependent upon the lag associated with square-root and divide operations due to their iterative nature and therefore implementations of these devices need to be investigated before selecting partition size.

Formally array selection should be based upon selecting maximum $N_{clk}(spiral)$ such that the inequality:

$$\frac{T_{min}(win)}{N_{clk}(spiral)} > T_{min}(spiral) \tag{7.21}$$

is satisfied, unless a larger partition than that selected results in lower cost.

(A) Xilinx FPGA Implementation Example

FPGA implementation of arithmetic modules is used here to demonstrate the design constraints on problem size independent decomposition arrays to keep in line with the FPGA implementations used to consider the programmable matrix element computation in section 6.4.3. Figures 7.23 to 7.26 show possible word-parallel and bit-serial implementations of the square-root and division arithmetic units discussed in section 4.4.8 and shown in figures 4.9 and 4.10.

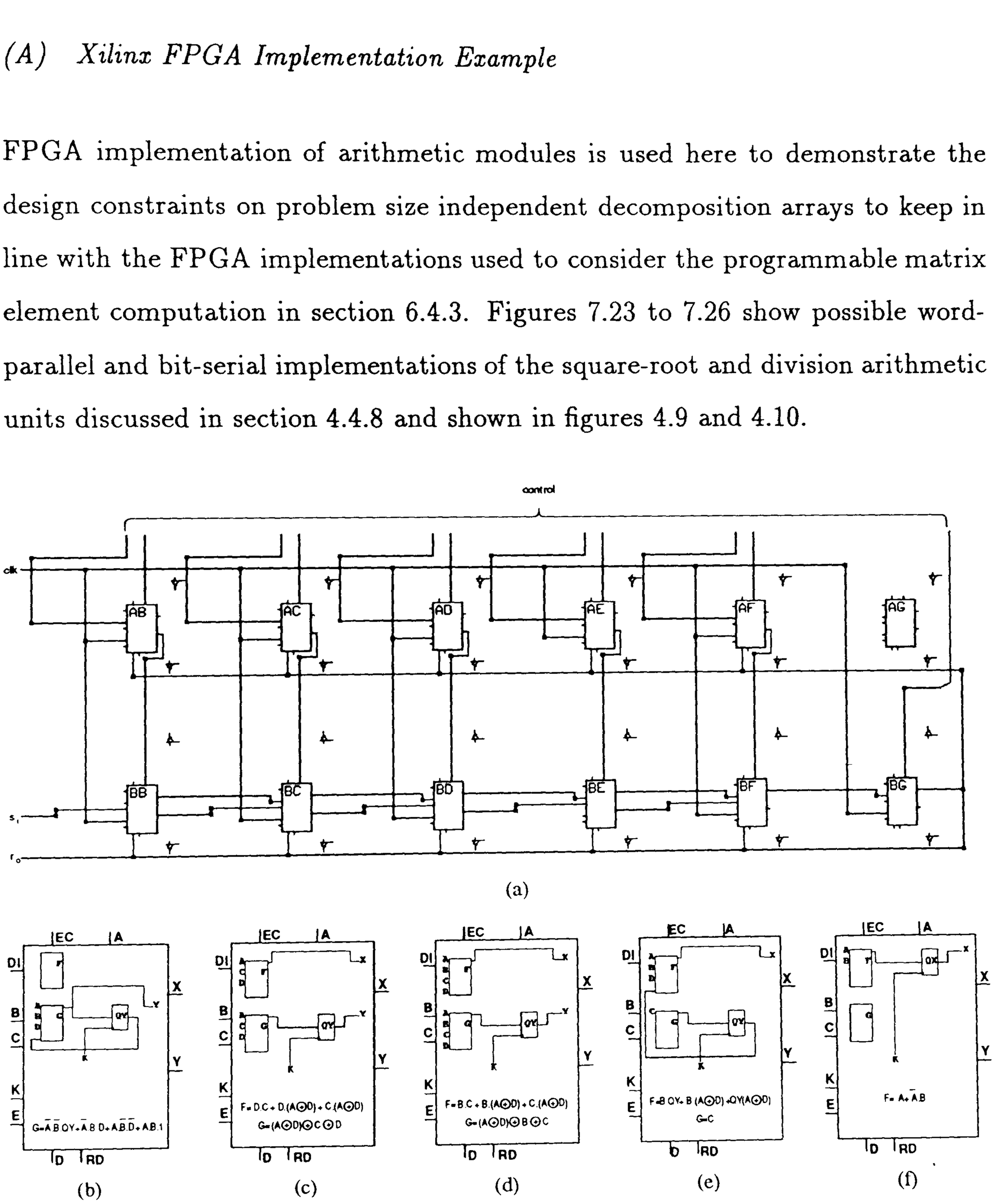
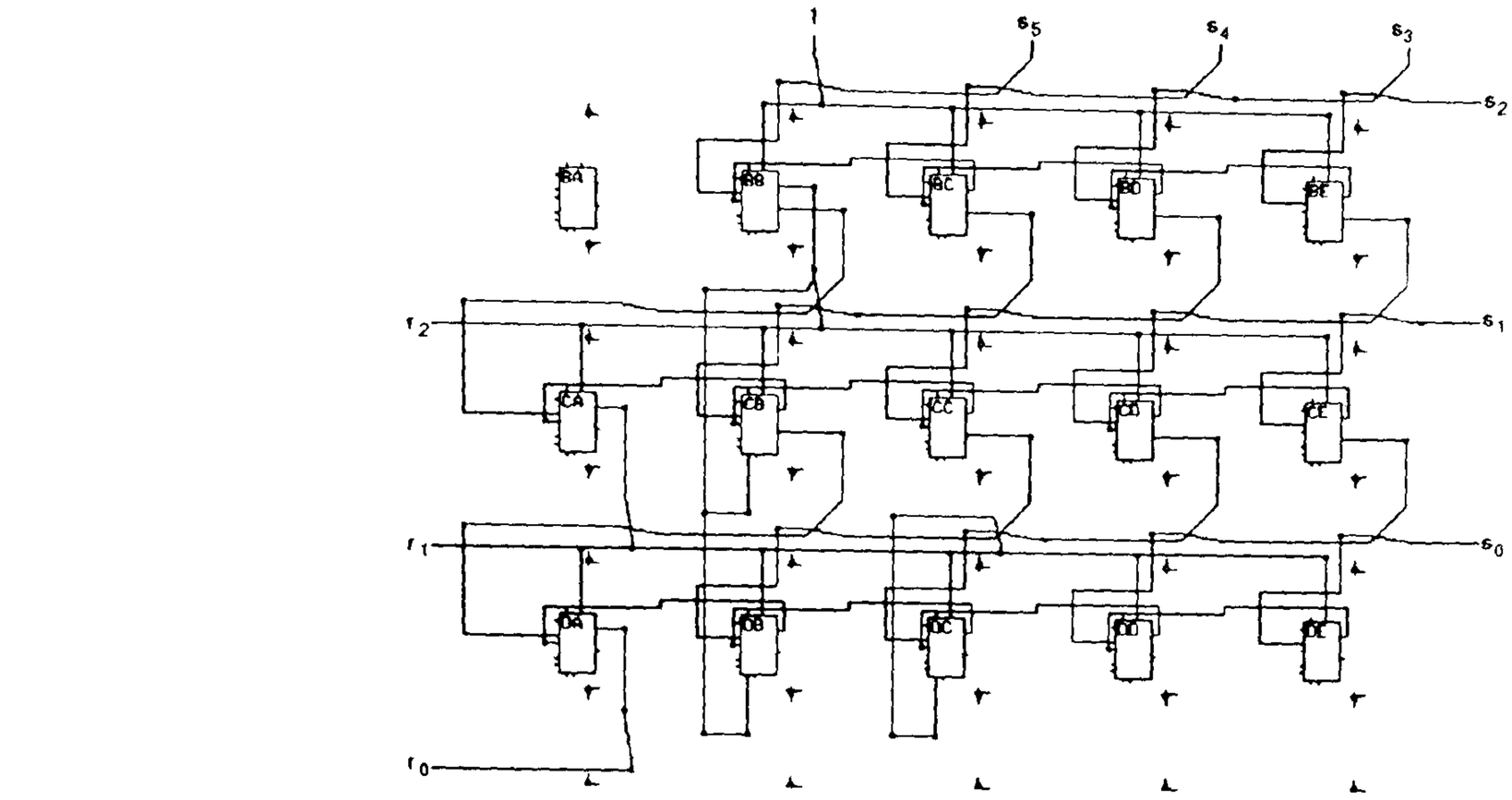
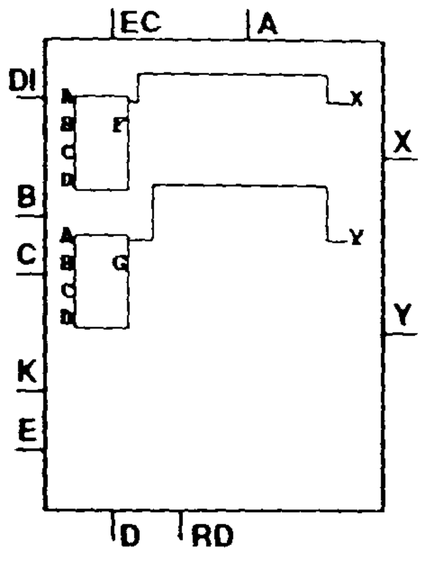


Figure 7.23: (a) Xilinx XC3130PC84-5 layout of the 3-bit non-restoring bit-serial square root (figure 4.10(c)) and CLB configurations of (b) multiplexer section AB to AF, (c) BB, (d) BC to BE (no delay required in BE), (e) BF, (f) BG.



(a)



(b)

MODE 0

CLBs: BD CA DA
F=B.C+B.(A⊕0)+C.(A⊕0)
G=(A⊕0)⊕B⊕C

CLBs: BE CE
F=B.A+B.(A⊕0)+A(A⊕0)
G=(A⊕0)⊕B⊕A

CLB: DB
F=B.C+B.(A⊕D)+C.(A⊕D)
G=(A⊕D)⊕B⊕C

MODE 1

CLBs: CB DC
F=B.C+B.(A⊕D)+C.(A⊕D)
G=(A⊕D)⊕B⊕C

MODE 2

CLBs: BB CC DD
F=B.C+B.(A⊕$\bar{A}$)+C.(A⊕$\bar{A}$)
G=(A⊕$\bar{A}$)⊕B⊕C

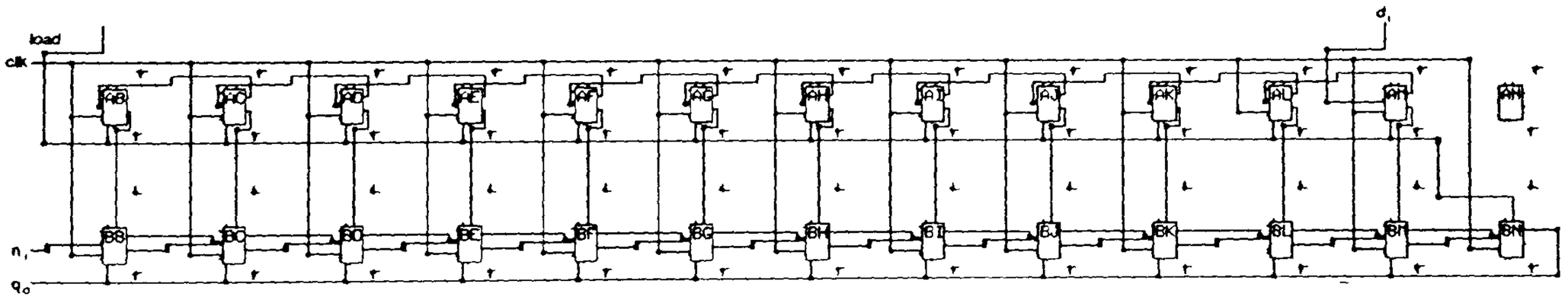
MODE 3

CLBs: BC CD
F=B.C+B.(A⊕1)+C.(A⊕1)
G=(A⊕1)⊕B⊕C

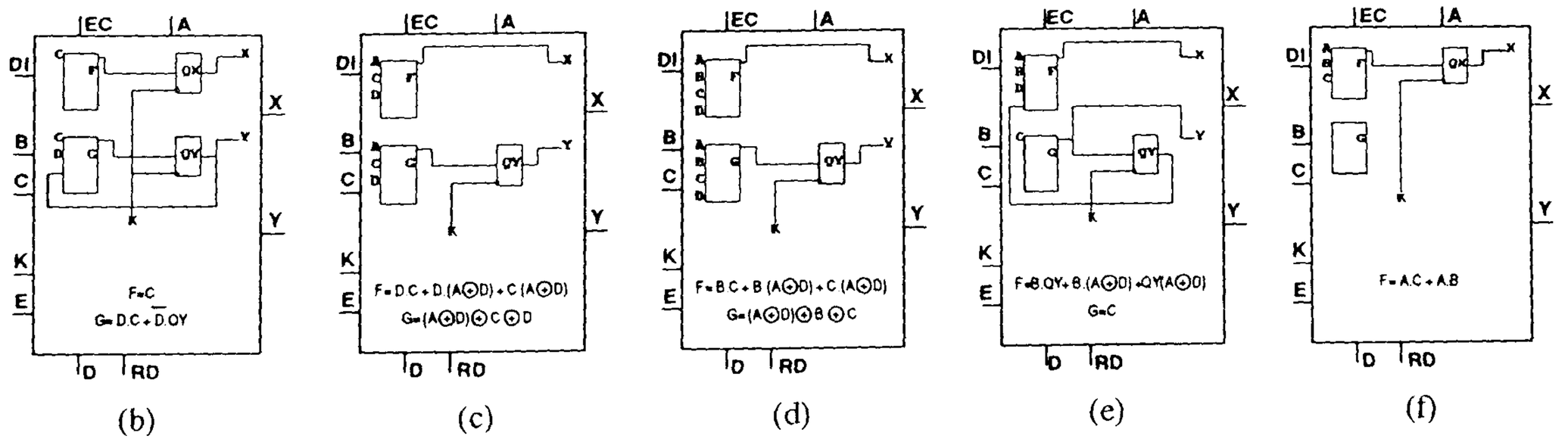
CLBs: BC CD
F=B.A+B.(A⊕1)+A(A⊕1)
G=(A⊕1)⊕B⊕A

(c)

Figure 7.24: (a) Xilinx XC3130PC84-5 layout of the 3-bit non-restoring word-parallel square root (figure 4.10(a)), (b) basic CLB configuration and (c) key to CLB logic functions.

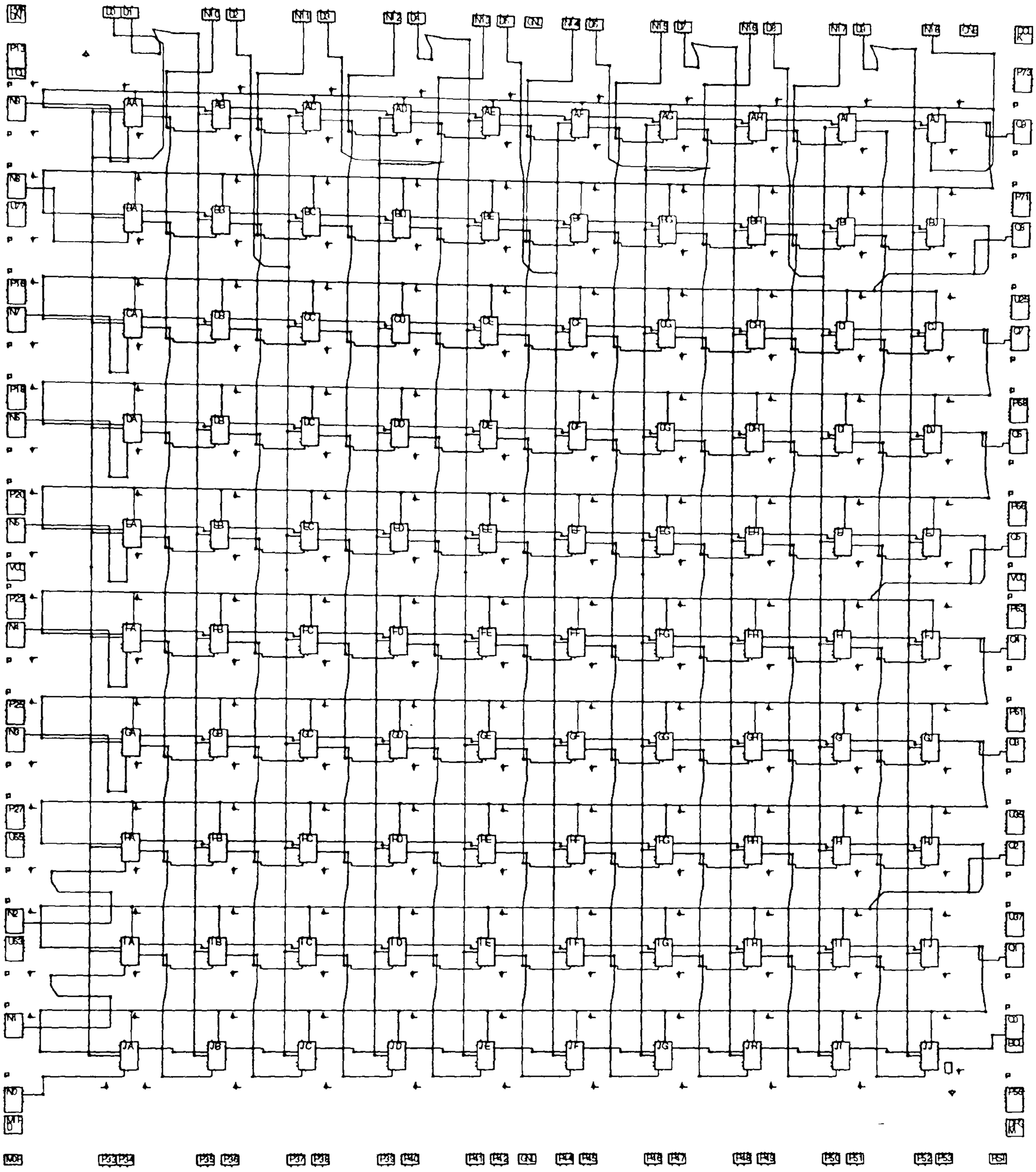


(a)

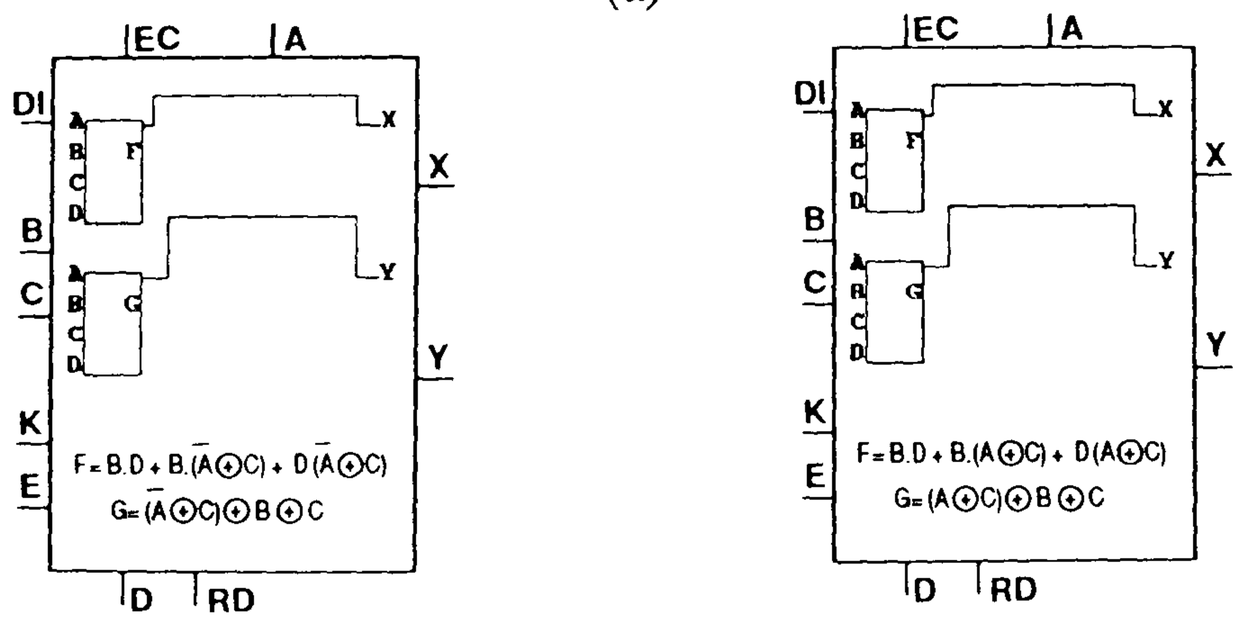


(b) (c) (d) (e) (f)

Figure 7.25: (a) Xilinx 3190PP175-5 layout of a 12-bit non-restoring bit-serial division unit (based on extending figure 4.9(c)) programmed using FRADL and CLB configurations of (b) AB to AM, (c) BB, (d) BC to BL (no delay required in BL), (e) BM, (f) BN.



(a)



(b)

(c)

Figure 7.26: (a) Xilinx XC3130PC84-5 layout of a 10-bit non-restoring word-parallel division unit (based on extending figure 4.9(a)) programmed using FRADL, (b) CLB configuration for top row of CLBs and (c) CLB configuration for the rest of the CLBs.

The top row of CLBs in the bit-serial divider unit (figure 7.25) performs a serial to parallel conversion for the denominator input data. The $w = 12$ bit-serial (figure 7.25) and $w = 10$ word-parallel (figure 7.26) versions of the divider units were programmed using FPGA regular array description language (FRADL) [128][132]. FRADL allows rapid prototyping of regular architectures such as the divider units by expressing the designs in terms of high level structural concepts. Compilation of a FRADL description substantially reduces design time and is especially useful for large arrays such as those in figures 7.26 and 7.26.

Using the XDE timing analysis [131] programs the propagation lags of the Xilinx square-root and divider implementations can be calculated for a word-length w as shown in table 7.5. Considering once again the 16 bit example then in the Cholesky decomposition case the square-root units have the longest lag in both word-parallel and bit-serial approaches when compared with propagation delays associated with division and multiplication units (section 6.4.3). Therefore the square root modules determine the maximum possible systolic clock frequency of the problem size independent Cholesky decomposition arrays. The LU array does not require square-root operation and its division units cause the maximum lag of any of its PEs to set the maximum clock speed. With use of maximum PE lags in each array the minimum partition size, that is the partition size leading to the smallest array which can maintain real-time operation, can be calculated from (7.19) to (7.21).

module	lag for specified word-length (ns)	
	w bits	16 bits
word-parallel square-root	$4.6(w^2 + w - 1) + 10(w) + 20$	1427
bit-serial square-root	$2w(4.6(w + 3) + 10)$	3117
word-parallel division	$4.6w^2 + 10w + 20$	1358
bit-serial division	$(2w - 1)(4.6w + 10)$	2592

Table 7.5: Propagation delays of square-root and division arithmetic units.

(B) Comparison of Xilinx Problem Size Independent Decomposition Arrays

The results of the minimum partition sizes in figure 7.27, show the partition sizes of the Cholesky array to be either less than or equal to the those of the LU array. It can be concluded from this that the advantage of the lower number of clock cycles required in the Cholesky arrays is more significant than the disadvantage caused by the greater lag associated with the square-root PEs of this array.

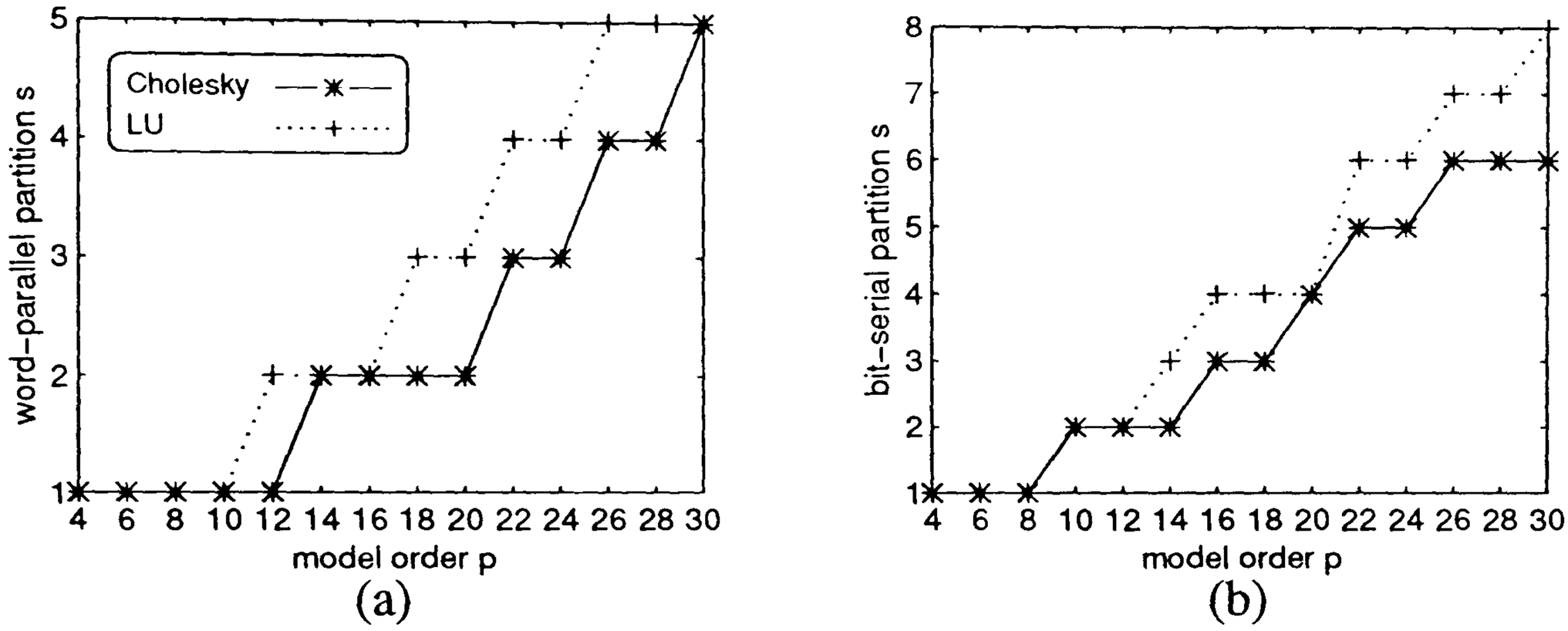


Figure 7.27: Minimum partition sizes s for even model orders p in the (a) word-parallel and (b) bit-serial Xilinx implementations of the Cholesky and LU decomposition arrays.

Study reveals that the minimum partition sizes also produce arrays with minimum cost. The optimal array costs associated with each model order problem are shown in figure 7.28. For the lowest model orders, where the partition sizes are equal (figure 7.27), the Cholesky array uses more equivalent NANDs than the LU due to the inclusion of the extra square-root unit. However, the Cholesky array does generally show significantly lower cost for the higher model order cases where the LU array requires a bigger partition size and thus more PEs. The bit-serial approach uses hardware resources more efficiently and can be implemented at a lower cost than word-parallel. To achieve better results in word-parallel effort should be concentrated on speeding up square-root and division, for example with the use of carry lookahead, to bring the speed of these modules more in line with that of the multipliers, thereby reducing partition size and the number of PEs.

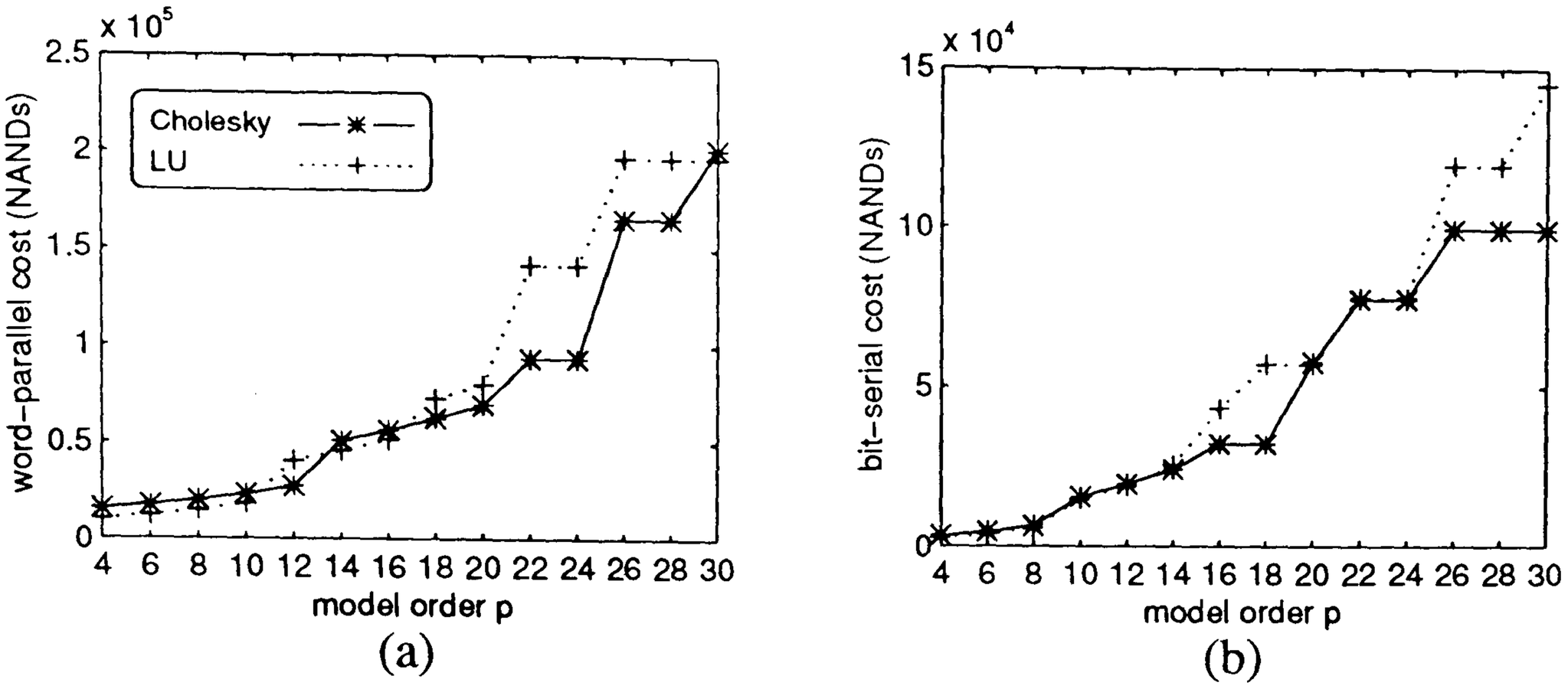


Figure 7.28: Optimal problem size independent array costs for even model orders p in the (a) word-parallel and (b) bit-serial Xilinx implementations of the Cholesky and LU decomposition arrays.

Finally the advantage of moving toward the problem size independent implementation from the problem size dependent approaches discussed in chapter 4 is illustrated in figure 7.29 in which the relative costs of these two methods are compared. In the word-parallel approaches the costs are reduced by 85 to 90% for model orders from 8 to 30 while in the bit-serial approach the cost reduction is in the order of 60 to 80%.

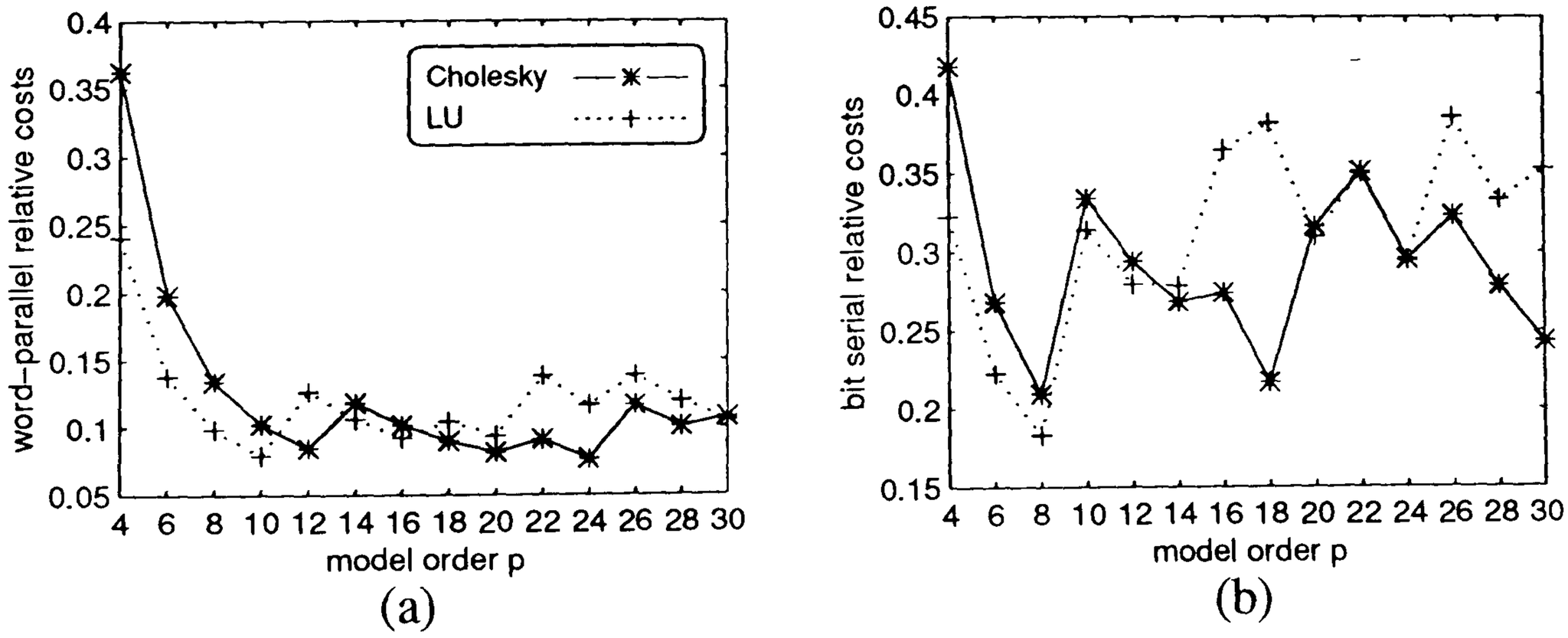


Figure 7.29: Relative cost of optimal problem size independent Cholesky and LU arrays respectively against the Cholesky [1,1,1] (figure 4.8) and LU [1,0,0] (figure 4.13) problem size dependent arrays for even model orders p in the (a) word-parallel and (b) bit-serial Xilinx implementations.

7.6 Concluding Remarks

The aim of this work was to design real-time DSP hardware to aid in calculation of filter parameters in the programmable model order Modified Covariance spectral estimator.

To limit the size of the systolic arrays for high model order problems the mapping of the algorithms into small systolic arrays using DDG partitioning and reindexing schemes was investigated. The research produced a spiral systolic array for the LU decomposition which utilised feedback links to provide simple rescheduling of intermediate results. The array was shown to offer significant advantages in respect of data rescheduling, input/output connectivity and operation time over a previously proposed spiral solution which was designed using alternative methods. In respect of the Cholesky decomposition algorithm the standard type of spiral architecture which was used for the LU decomposition was deemed to be unsuitable. This led to proposal of a different reindexing scheme resulting in a novel spiral systolic architecture for the Cholesky decomposition computation. The two arrays presented have been shown to be easily adaptable for use with higher model orders with the use of extra delays in the feedback loops. The versatility of DDG reindexing methods has been shown in that they were able to be applied to different types of decomposition problem and showed a clearer, more easily understandable mapping from algorithm to systolic array than was the case for the DBT method.

In a Xilinx FPGA implementation example the LU and Cholesky DDG designed spiral systolic arrays, both had similar control requirement, but the Cholesky method offered lower cost in general at higher model order. Bit-serial implementation was found to be most cost efficient and the reduction in cost of 75% at $p = 30$ in the bit-serial problem size independent Cholesky array compared to the problem size dependent $p = 30$ Cholesky [1,1,1] array meant the number of Xilinx XC3195-5 FPGAs required could be reduced from 35 to 9.

Chapter 8

Conclusion

8.1 General Review

This thesis details the design of hardware architectures for use in Modified Covariance spectral estimation with pulsed ultrasonic Doppler instruments in blood flow measurements. The motivation behind the research was to design application specific hardware which offered increased sensitivity in the detection of arterial disease by enabling the real-time implementation of the Modified Covariance method, which has superior performance to the conventional short term fast Fourier transform (STFFT).

Previous research has determined that the power frequency spectrum of the Doppler time signal output from pulsed ultrasonic blood flow detectors gives a good representation of the velocity profile of the scanned blood cells [48][50]. It has been shown that arterial stenosis widens the velocity range and increases the mean flow velocity of blood cells in the region of the occlusion, enabling the severity of disease to be diagnosed from subsequent changes in the width and mean frequency of the Doppler power spectrum. In a comparison of Doppler power spectrum mean frequency and bandwidth estimation methods, the model order 4 Modified Covariance spectral estimator had been previously identified to give the best accuracy when weighted against computational burden [10]. At-

tempts had been made to implement the Modified covariance spectral estimator on a transputer platform, but in certain cases it was found that such an approach could not provide the necessary throughput for real-time operation [14].

This thesis readdressed the problem of real-time implementation of the Modified Covariance spectral estimator and considered the feasibility of using VLSI array processors. VLSI technology has allowed increasingly large numbers of gates to be mounted on small integrated circuits thus promoting the portability and the accessibility to special purpose systems. Systolic architectures have been shown to be very amenable to VLSI fabrication due to their localised communication network and regular arrays of relatively simple PEs. They have found application in many areas of real-time digital signal processing, their high throughput achieved through the use of massive levels of concurrent processing arising from the parallel and pipelined nature of the arrays.

In the design of the ASIC device it was deemed necessary to have a coherent strategy for the derivation of systolic array architectures from the algorithms involved in the Modified Covariance method. This led to selection of the data dependence graph (DDG) design method [23] which was used throughout the thesis to provide clear representations of the single assignments and communication requirements in the algorithms performed in the Modified Covariance spectral estimator. By way of many examples the process of systolic array formation from DDG projection has been demonstrated. For certain algorithms it was shown how a number of different DDGs could be derived, each leading to a unique systolic array solution. It was also shown how a variety of systolic arrays could be designed by projecting DDGs in various directions. The versatility of the systolic model of computation was demonstrated by applying the DDG methodology to the design of word-level systolic arrays for all the algorithms required in the Modified Covariance method, data reordering networks and bit-level pipelined arithmetic units for use in the PEs of the word-level systolic arrays.

The question which then arose was, that when presented with a range of systolic array designs, then what makes a design optimal? This was a difficult choice to make given the variety of systolic architectures which were designed, each with their own specific PE circuitry, communication architectures and control requirements. Many researchers would make simple comparisons based on the number of PEs but more detailed comparisons were made in this thesis, using such metrics as array gate level hardware, communication and control cost. Costs of integrating systolic arrays into the system were considered and a cost/benefit analysis which weighted hardware and communication against estimation error allowed optimal word-length arrays to be selected. The aspects of operation time, processor and array efficiency were also recognised as important features in the comparison of systolic array performance to help make the difficult choice.

Two types of systolic array design were considered in the thesis. The first considered mapping algorithms of a fixed model order spectral estimator onto problem size dependent arrays. The second approach was slightly more flexible in that algorithms were mapped onto problem size independent systolic arrays which could handle various sized problems without having to vary the number of PEs.

8.1.1 The Problem Size Dependent Systolic Array Approach

The mapping onto problem size dependent systolic arrays was initially considered for the model order $p = 4$ Modified Covariance spectral estimator to derive simple architectures for estimation of Doppler mean frequency and bandwidth. The calculation of the covariance matrix elements and the filter parameters were of interest. The calculation of the $p = 4$ covariance matrix elements, although being a relatively simple algorithm consisting of a number of multiply accumulate operations, was shown to present a fairly large real-time computational burden.

Two 3-dimensional DDGs were derived to compute all of the necessary single assignment operations and the mapping of these graphs onto a 2-dimensional systolic array was shown with symmetrical matrix properties used to reduce the number of PEs. The resulting array however was expensive to implement since there were two multiply accumulators within each PE, it displayed high communication cost (requiring some global connections) and results were left within the PEs which made retrieval of data difficult and untidy.

Prompted by these disadvantages attempts were made to redesign the DDGs. The computations were partitioned into a total of three 2-dimensional DDGs and these were projected into linear arrays of length 5 PEs, 3 PEs and 1 PE, each PE containing a single multiply accumulator unit, with a control option for doubling the product. Although this tri-linear array offered substantial reductions in hardware, communication, and input connectivity, limitations were recognised in that each PE could only be active 50% of the time and multiplications were repeated from one linear array to another, combining to give an array with poor efficiency.

Further optimisation was proposed by deriving a DDG which computed all the necessary multiply accumulate operations just once and, by splitting the DDG the redundancy caused by bi-directional data flow was removed. Two 100% efficient linear arrays of length 2 and 3 PEs resulted from projection. However it was found that the sorting network required for the calculation of the covariance matrix elements from this bi-linear array's multiply accumulate outputs, added considerably to the register and control cost of the design, consequently having a detrimental effect on the advantage gained by the reduction of the number of PEs over the previous tri-linear systolic array.

The final problem size dependent systolic array design was aimed at reaching a compromise between the simplicity of the tri-linear array while attempting to meet the multiply accumulate efficiency of the bi-linear array. To do this the

bi-linear multiply accumulate DDG was treated as simply a DDG for the sole calculation of the products projected into 2 linear systolic arrays. Five separate DDGs for the accumulation were then designed to feed off the multiplication array outputs and multi-projection was used to map these accumulation DDGs into single PEs. In this partitioned multiply accumulate (PMA) design the two types of PE were later merged and it was found that hardware costs could be lowered considerably with a dual frequency systolic clocking strategy. Hardware cost, communication and control comparisons of all four matrix element systolic arrays led to selection of the PMA array and the bit-serial approach was deemed to provide sufficient throughput for real-time operation.

For the solution of the system of linear equations decomposition techniques were identified as efficient algorithms to perform the computation of the filter parameters. It has been shown how these algorithms may be used to decompose the equation into the format of triangular systems of equations which may then be easily solved using the forward elimination and back substitution algorithms [17]. Three different decomposition algorithms were considered, namely Cholesky, LU and LDL^T.

The Cholesky algorithm was investigated separately as it was noted that this algorithm required the square root operation which presents difficulty when implementing in VLSI. The Cholesky decomposition 3-dimensional DDG was derived and the systolic arrays formed from projections in five different directions were considered. Each contained 10 PEs and displayed similar operation time. Of these five projections, the array resulting from the $[1, 1, 1]$ mapping, equivalent to Brent and Luk's hexagonally connected array [80], displayed the lowest gate cost but a slightly higher communication cost. Unlike the the other arrays though, each of the PEs in the $[1, 1, 1]$ array were only required to work in single modes, and this factor in conjunction with the low cost led to this array being selected.

The LU and LDL^T non-square root methods were then examined. The similarity of the recursive structure of the DDGs for these methods with that of the Cholesky was used to narrow projection the projection vectors considered down to the $[1,0,0]$ and $[1,1,1]$ vectors which had previously produced the two optimal Cholesky decomposition arrays. A new array for $[1,0,0]$ LU decomposition was proposed which allowed on-the-fly retrieval of data without significantly increasing the PE complexity in the design proposed by Kung [54] in which this problem had not been discussed. Two more new arrays were produced from a DDG which re-represented the LDL^T decomposition algorithm, to reduce the hardware cost of a design proposed by Brent & Luk [80], which utilised a different communication strategy. In the non-square-root array comparison the simplicity and low cost of the LU $[1,0,0]$ array in terms of the orthogonal communication structure and the internal PE layout led to its selection.

An in-depth comparison of the selected Cholesky $[1,1,1]$ and LU $[1,0,0]$ systolic arrays was then made to consider the cost of system integration and the effect of finite word-length rounding on the estimated filter parameters. In a cost/benefit analysis the hardware and communication costs incurred were weighed against a benefit function, treated as the inverse of mean frequency and bandwidth estimation error, for a range of word-lengths. In the fixed point error analysis the LU array was shown to suffer from catastrophic overflow errors at lower word-length while for the Cholesky array errors only really became significant at the very low word-lengths. When these results were substituted in the cost/benefit analysis the Cholesky decomposition array at a 12 bit word-length was found to be optimal for estimation of both mean frequency and bandwidth. Further analysis revealed that, providing the products were accumulated at double precision in the matrix element calculation, a 10 bit word-length was sufficient for the quantisation of the Doppler signal.

8.1.2 The Problem Size Independent Systolic Array Approach

The advantage of the problem size dependent systolic array architectures was that they required simple control, communication and processor layout. Once the array was designed, PE arithmetic unit design could then be based upon the allowable lag for real-time operation to be achieved. It was found however, notably in the case of the filter parameter calculation, that the degree of concurrency achieved allowed PEs to be designed to quite low specification, and use of word-parallel arithmetic clearly could not be justified. An alternative approach to systolic design which allowed the degree of partitioning to be based upon a given processor specification to utilise the arithmetic units maximum potential, was then discussed. This involved the design of problem size independent systolic arrays which were shown to be more adaptable to a wider range of problems of varying model order. Another reason behind the adoption of problem size independent partitioning schemes was that when considering large model order problems, sometimes used for estimation of other spectral features, then problem size dependent arrays would become excessively large and not feasible to implement. Nevertheless the knowledge gained in the development of problem size dependent systolic arrays proved to be extremely useful when considering the design of problem size independent systolic arrays.

A partitioning technique was applied to the multiplication DDG for the PMA design in order to tackle the covariance matrix element calculation in the high model order Modified Covariance spectral estimator. The partitioning involved projecting a number of rows of the DDG into a single PE as opposed to one row per PE as previously considered. It was shown that there were optimal partition sizes when mapping a certain model order problem into a fixed number of PEs. With increasing optimal partition size the size of the partitioned systolic array for a given model order problem reduced, but at the same time the number of registers in the accumulation feedback increased. The general trend though

was toward lower cost and this was especially pronounced in the word-parallel approach. Mapping into small arrays however put more burden onto each PE as the total number of clock cycles for complete operation was expanded leading to a tradeoff between array size and operation time.

With increased model order the processing burden in the filter parameter stage became much greater and the development of an efficient partitioning scheme was crucial. The design of a programmable model order systolic array for Cholesky decomposition stemmed from the research of Navarro et al. [28] into the derivation of an LU decomposition spiral systolic array using TBP and DBT design methods. A LU spiral systolic array was designed using DDG partitioning and reindexing methods to address several shortcomings apparent in the design. The resulting array incorporated systolic delay feedback links to reschedule intermediate results, removing the need for storage/fetch cycles from a memory source to improve system throughput and reduce operation time. Redundant processor states could be used in setting up data for subsequent subproblems and chaining gave an indication of data scheduling for the complete computation.

A problem size independent systolic array for Cholesky decomposition had not previously been proposed and the design of such an array was considered for the first time in this thesis. The design methodology used, was based upon the DDG partitioning and reindexing methods used in the LU design. It was discovered that the communication network of the standard spiral systolic array architecture used for the LU decomposition was unsuitable for performing the Cholesky decomposition. This led to proposal of a modified DDG subproblem reindexing approach which resulted in a novel spiral architecture. When the Cholesky and LU decomposition spiral systolic arrays with equal numbers of PEs were applied to the same model order problems, the Cholesky array cost was slightly higher but this extra cost was deemed worth incurring as the number of clock cycles was approximately halved in the Cholesky computation.

In conclusion, systolic arrays have been shown to offer a viable, cost-effective solution to the problem of implementing the Modified Covariance spectral estimator in real-time, and to improve accessibility to accurate blood flow diagnostic equipment. Two distinct classes of array, problem size dependent and problem size independent, have been designed and analysed. Problem size dependent systolic arrays were suitable for the small model order algorithms of the Modified Covariance spectral estimator, displaying basic control requirement and fully localised communication. On the other hand, problem size independent systolic arrays required an involved control strategy, some global interconnections and more complex internal PE structures. But, they did however offer flexible, programmable model order solutions, which were capable of computing high model order problems while making efficient use of arithmetic unit potential.

Data dependence graph design methods have shown a great deal of versatility in the design of hardware for real-time spectral estimation. A design platform has been established on which the systolic array implementation of other more demanding spectral estimators such as ARMA methods could be based.

8.2 Further Research Possibilities

The next step in the design process is to test an implementation of the fixed model order $p = 4$ Modified Covariance spectral estimator on hardware. Field programmable gate arrays (FPGAs), such as those produced by Xilinx [129], offer an ideal prototyping environment and use of their reprogrammability can be made to easily facilitate design optimisation. Tools such as FRADL (FPGA Regular Array Description Language) [128][132] can be used to simplify word-parallel arithmetic unit construction on FPGAs by describing the regular layout within a few lines of simple high level language code. FRADL was used to produce the bit-serial and word-parallel divider arrays presented in chapter 7 (figures 7.25 and 7.26) and could easily be extended to the other arithmetic units described.

Xilinx FPGAs have been shown to be suitable for the implementation of bit-serial devices by utilisation of the CLB's flip-flops for the required bit-level pipelining. The use of bit-serial arithmetic in systolic array processors presents a greater challenge to the designer than word-parallel as the timing of individual data in each word needs to be matched to the systolic clock rate. Extra control signals are required for indication of sign bits, bit-level clocking and arithmetic unit reset.

The DDG methodology was also applied to the design of bit-serial pipelined arithmetic units. The bit-level DDG projection described by McCanny et al. for multiplication [118] was extended to produce two's complement multipliers and the ideas were used in bit-serial divider and square root unit design. Problems were however encountered when the matching of data flows between systolic PEs in the Cholesky decomposition array was considered. In two's complement multiplication, addition and subtraction the least significant result bit is computed first, due to the direction of the carry dependence arcs in their DDGs. The input data is usually LSB first, but, depending on the projection vector the input operands can both be bit-serial or one of the inputs could be word-parallel. Conversely, division and square root restoring and nonrestoring algorithms [88] inherently output the MSB first, which causes a mismatch between the systolic PEs in the Cholesky decomposition array. Thus further study on bit-serial implementation is required.

Another problem with division and square-root algorithms is that there is an intrinsic dependence between iteration steps which causes their hardware units to be considerably slower than those for multiplication. The division and square-root PEs cause bottlenecks which determine the maximum clock frequency for the systolic array. The design of high speed word-parallel and bit-serial division/square-root hardware still remains a serious undertaking, and is of great importance in the improvement of decomposition systolic array throughput.

Implementation of the fixed model order $p = 4$ spectral estimator on problem size dependent systolic arrays using word-parallel is initially recommended as this requires the basic control scheme and allows easier testability [133]. Efficiency can then be improved by either going on to the bit-serial approach or by considering the problem size independent schemes. When considering the implementation of higher model systolic arrays the likelihood of the covariance system of equations becoming more ill-conditioned increases, decreasing the accuracy of the filter parameter results which are more susceptible to any error in the covariance matrix elements. Longer word-length is therefore necessary in order to maintain sufficient accuracy and use of a cost/benefit analysis similar to that described for the model order $p = 4$ estimator should be investigated to obtain optimal word-length.

When implementing the high model order decomposition arrays attention should be paid to the global communications which are expensive to implement in VLSI. The spiral buses carry a certain number of systolic delay elements and with some further study they could possibly be replaced by a series of local communications.

Another possible future study is the design of systolic arrays for the other types of spectral estimator as reviewed in chapter 2. Spectral estimation techniques are the subject of continuous research and thus there is need to produce hardware implementations as quickly as possible before these techniques are further developed. The DDG methods demonstrated in this thesis can be used to reduce design cycle periods and speed hardware implementation of new techniques.

Appendix A

Publications

S. J. Bellis, W. P. Marnane, D. Wilde, and P. J. Fish, "Systolic arrays for Modified Covariance spectral estimation used with ultrasonic Doppler blood flow detectors," in *Signal Processing VII - Theories and Applications, Proceedings of EUSIPCO-94*, vol. 3, (Edinburgh, Scotland), pp. 1361–1364, Sept. 1994.

S. J. Bellis, W. P. Marnane, and P. J. Fish, "Systolic architectures for the Modified Covariance spectral estimator used with Doppler blood flow detectors," in *Proceedings of the Irish DSP and Control Colloquium (IDSPCC'94)*, (Dublin, Ireland), pp. 71–78, July 1994.

S. J. Bellis, P. J. Fish, and W. P. Marnane, "Optimal systolic arrays for real-time implementation of the Modified Covariance spectral estimator," *Parallel Algorithms and Applications*, in press.

S. J. Bellis, W. P. Marnane, and P. J. Fish, "Mapping arbitrary dimension LU and Cholesky decomposition algorithms onto small spiral systolic arrays," *Parallel Algorithms and Applications*, in review.

S. J. Bellis, W. P. Marnane, and P. J. Fish, "An alternative systolic array for non-square-root Cholesky decomposition," *IEE Proceedings: Computers and Digital Techniques*, in review.

References

[1] D. W. Baker, "Pulsed ultrasonic Doppler blood-flow sensing," *IEEE Transactions on Sonics and Ultrasonics*, vol. SU-17, pp. 170–185, July 1970.

[2] D. H. Evans, W. McDicken, R. Skidmore, and J. Woodcock, *Doppler ultrasound: Physics, instrumentation and clinical applications.* London: John Wiley and Sons, 1989.

[3] M. Kassam, R. S. C. Cobbold, P. Zuech, and K. W. Johnston, "Quantification of carotid arterial disease by Doppler ultrasound," in *Proceedings IEEE Ultrasonics Symposium*, 1982.

[4] A. V. Oppenheim and P. W. Schafer, *Digital signal processing.* Englewood Cliffs, NJ: Prentice Hall, 1975.

[5] F. J. Harris, "On the use of windows for harmonic analysis with the discrete Fourier transform," *Proceedings of the IEEE*, vol. 66, pp. 51–83, Jan. 1978.

[6] A. Papoulis, *Signal analysis.* New York: McGraw Hill, 1984.

[7] F. S. Schlindwein and D. H. Evans, "A real-time autoregressive spectrum analyzer for Doppler ultrasound signals," *Ultrasound in Medicine and Biology*, vol. 15, no. 3, pp. 263–272, 1989.

[8] P. J. Fish, "Nonstationarity broadening in pulsed Doppler spectrum measurements," *Ultrasound in Medicine and Biology*, vol. 17, no. 2, pp. 147–155, 1991.

[9] S. M. Kay, *Modern Spectral Estimation.* Prentice Hall, 1988.

[10] M. G. Ruano and P. J. Fish, "Cost/benefit criterion for selection of pulsed Doppler ultrasound spectral mean frequency and bandwidth estimation," *IEEE Transactions on Biomedical Engineering*, vol. 40, no. 12, pp. 1338–1341, 1993.

[11] M. G. Ruano and P. J. Fish, "Cost / benefit selection of spectral estimators for use with ultrasonic Doppler blood flow instruments," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'92)*, vol. 5, (San Francisco, California), pp. 513–516, Mar. 1992.

[12] K. Kaluzynski, "Analysis of application possibilities of autoregressive modelling to Doppler blood flow signal spectral analysis.," *Medical & Biological Engineering & Computing*, vol. 25, pp. 373–376, 1987.

[13] S. Marple, *Digital spectral analysis with applications.* Prentice Hall, 1987.

[14] M. G. Ruano, D. F. G. Nocetti, P. J. Fish, and P. J. Fleming, "A spectral estimator using parallel processing for use in a Doppler blood flow instrument," in *Proceedings of the European Workshop on Parallel Computing (EWPC'92)*, pp. 397–400, 1992.

[15] M. G. Ruano, D. F. G. Nocetti, P. J. Fish, and P. J. Fleming, "Alternative parallel implementations of an AR-modified covariance spectral estimator for diagnostic ultrasound blood-flow studies," *Parallel Computing*, vol. 19, no. 4, pp. 463–476, 1993.

[16] S.-Y. Kung, "VLSI array processors," *IEEE ASSP Magazine*, vol. 2, pp. 4–22, July 1985.

[17] H. T. Kung and C. E. Leiserson, "Systolic arrays (for VLSI)," in *Sparse Matrix Proceedings*, pp. 256–282, Society for Industrial and Applied Mathematics, 1979.

[18] S.-Y. Kung and Y. H. Hu, "A highly concurrent algorithm and pipelined architecture for solving Toeplitz systems," *IEEE Transactions on Acoustics Speech and Signal Processing*, vol. ASSP-31, pp. 66–75, Feb. 1983.

[19] V. Boriakoff, "FFT computation with systolic arrays, a new architecture," *IEEE Transactions on Circuits and Systems*, vol. 41, pp. 278–284, Apr. 1994.

[20] C. R. Ward, A. J. Robson, P. J. Hargrave, and J. G. McWhirter, "Application of a systolic array to adaptive beamforming," in *IEE Proceedings-F*, vol. 131, pp. 638–645, Oct. 1984.

[21] J. C. White, J. V. McCanny, A. P. H. McCabe, J. G. McWhirter, and R. A. Evans, "A high speed CMOS/SOS implementation of a bit level systolic correlator," in *Proceedings ICCASP'86 (Tokyo, Japan)*, pp. 1161–1164, 1986.

[22] S.-Y. Kung, S.-C. Lo, and P. S. Lewis, "Optimal systolic design for the transitive closure and the shortest path problems," *IEEE Transactions on Computers*, vol. C-36, pp. 603–614, May 1987.

[23] S.-Y. Kung, *VLSI Array Processors.* Prentice Hall, 1988.

[24] H. T. Kung, "Why systolic architectures?," *IEEE Computer*, vol. 15, pp. 37–46, Jan. 1982.

[25] S. M. Kay and S. L. Marple, "Spectrum analysis - a modern perspective," *Proceedings of the IEEE*, vol. 69, pp. 1381–1419, Nov. 1981.

[26] D. F. G. Nocetti, M. G. Ruano, P. J. Fish, and P. J. Fleming, "Parallel implementation of a parametric spectral estimator for a real-time Doppler flow detector," in *Transputer Research and Applications 6, Proceedings of the Sixth Conference of the North American Transputer Users Group* (S. Atkins and A. S. Wagner, eds.), vol. NATUG-6, (Vancouver, Canada), pp. 131–141, May 1993.

[27] G. H. Golub and C. F. van Loan, *Matrix Computations.* Kogan Page Ltd, London: North Oxford Academic Publishers Ltd, 1986.

[28] J. J. Navarro, J. M. Llaberia, and M. Valero, "Partitioning: An essential step in mapping algorithms into systolic array processors," *IEEE Computer*, vol. 20, pp. 77–89, July 1987.

[29] S. J. Bellis, W. P. Marnane, D. Wilde, and P. J. Fish, "Systolic arrays for Modified Covariance spectral estimation used with ultrasonic Doppler blood flow detectors," in *Signal Processing VII - Theories and Applications, Proceedings of EUSIPCO-94*, vol. 3, (Edinburgh, Scotland), pp. 1361–1364, Sept. 1994.

[30] S. J. Bellis, P. J. Fish, and W. P. Marnane, "Optimal systolic arrays for real-time implementation of the Modified Covariance spectral estimator," *Parallel Algorithms and Applications*, in press.

[31] D. N. White, "Johann Christian Doppler and his effect - a brief history," *Ultrasound in Medicine and Biology*, vol. 8, pp. 583–591, 1982.

[32] E. J. Jonkman, "Doppler research in the nineteenth century," *Ultrasound in Medicine and Biology*, vol. 6, pp. 1–5, 1980.

[33] D. A. Christensen, *Ultrasonic Bioinstrumentation.* John Wiley & Sons Inc., 1988.

[34] K. J. W. Taylor, P. N. Burns, and P. N. T. Wells, *Clinical Applications of Doppler Ultrasound.* Raven Press, 1988.

[35] P. Atkinson and J. P. Woodcock, *Doppler Ultrasound and its use in Clinical Measurement.* Academic Press, 1982.

[36] "WFUMB symposium on safety and standardization in medical ultrasound - issues and recommendations regarding thermal mechanisms for biological effects of ultrasound." Ultrasound in Medicine and Biology, vol. 18, no. 9, 1992.

[37] S. A. Jones, "Fundamental sources of error and spectral broadening in Doppler ultrasound signals," *Critical Reviews in Biomedical Engineering*, vol. 21, no. 5, pp. 399–483, 1993.

[38] D. L. Franklin, W. A. Schlegel, and R. F. Rushmer, "Blood flow measured by Doppler frequency shift of backscattered ultrasound," *Science*, vol. 132, pp. 564–565, 1961.

[39] P. J. Fish, *Physics and Instrumentation of Diagnostic Medical Ultrasound.* John Wiley & Sons, 1990.

[40] C. E. Shannon, "Communications in the presence of noise," *Proceedings of the IRE*, vol. 37, pp. 10–21, 1949.

[41] J. M. Brennan, N. L. Rogers, and F. W. Ingle, "Real-time spectral analysis of Doppler bloodflow signals," in *IEEE Ultrasonics Symposium*, pp. 673–674, 1982.

[42] C. Kasai, K. Namekawa, A. Koyano, and R. Omoto, "Real-time two-dimensional blood flow imaging using an autocorrelation technique," *IEEE Transactions on Sonics and Ultrasonics*, vol. SU-32, pp. 458–463, May 1985.

[43] R. I. Kitney and D. P. Giddens, "Extraction and characterisation of underlying velocity waveforms in poststenotic flow," in *Proceedings of the IEE*, vol. 129, pp. 651–662, 1982.

[44] M. Kassam, K. W. Johnston, and R. S. C. Cobbold, "Quantitativē estimation of spectral broadening for the diagnosis of carotid arterial disease: Method and in vitro results," *Ultrasound in Medicine and Biology*, vol. 11, pp. 425–433, 1985.

[45] M. G. Ruano, *Investigation of real-time spectral analysis techniques for use with pulsed ultrasonic Doppler Blood-Flow detectors.* PhD thesis, School of Electronic Engineering and Computer Systems, University of Wales, Bangor, 1992.

[46] L. Y. L. Mo and R. S. C. Cobbold, ""Speckle" in continuous wave Doppler ultrasound spectra: A simulation study," *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*, vol. UFFC-33, pp. 747–752, Nov. 1986.

[47] W. R. Brody and J. D. Meindl, "Theoretical analysis of the CW Doppler ultrasonic flowmeter," *IEEE Transactions on Biomedical Engineering*, vol. BME-21, pp. 183–192, May 1974.

[48] L. Y. L. Mo and R. S. C. Cobbold, "A stochastic model of the backscattered Doppler ultrasound from blood," *IEEE Transactions on Biomedical Engineering*, vol. BME-33, pp. 20–27, Jan. 1986.

[49] K. K. Shung, R. A. Sigelmann, and J. M. Reid, "Scattering of ultrasound by blood," *IEEE Transactions on Biomedical Engineering*, vol. BME-23, pp. 460–467, Nov. 1976.

[50] B. A. J. Angelsen, "A theoretical study of the scattering of ultrasound from blood," *IEEE Transactions on Biomedical Engineering*, vol. BME-27, no. 2, pp. 61–67, 1980.

[51] P. A. J. Bascom, R. S. C. Cobbold, and B. H. M. Roelofs, "Influence of spectral broadening on continuous wave Doppler ultrasound spectra: A geometric approach," *Ultrasound in Medicine & Biology*, vol. 12, no. 5, pp. 387–395, 1986.

[52] P. A. J. Bascom and R. S. C. Cobbold, "Effects of transducer beam geometry and flow velocity profile on the Doppler power spectrum: A theoretical study," *Ultrasound in Medicine & Biology*, vol. 16, no. 3, pp. 279–295, 1990.

[53] J. A. Cadzow, *Foundations of Digital Signal Processing and Data Analysis.* Macmillan Publishing Company, 1987.

[54] S. Y. Kung, H. J. Whitehouse, and T. Kailath, *VLSI and Modern Signal Processing.* Prentice-Hall Inc., 1985.

[55] H. Baher, *Analogue and Digital Signal Processing.* John Wiley & Sons Ltd., 1990.

[56] N. Mohanty, *Signal Processing: Signals, Filtering and Detection.* Van Nostrand Reinhold - New York, 1987.

[57] K. S. Shanmagan and A. M. Breipohl, *Random signals: Detection, Estimation and Data Analysis.* John Wiley and Sons, Inc., 1988.

[58] J. Proakis and D. Manolakis, *Introduction to Digital Signal Processing.* Macmillan Publishing, 1988.

[59] S. Haykin, *An Introduction to Analog and Digital Communications.* John Wiley and Sons, Inc., 1989.

[60] R. K. Otnes and L. Enochson, *Digital Time Series Analysis.* New York: Wiley, 1972.

[61] J. Capon, "High-resolution frequency-wavenumber spectrum analysis," *Proceedings of the IEEE*, vol. 57, pp. 1408–1418, Aug. 1969.

[62] R. Challis and R. Kitney, "Biomedical signal processing. Part III: the power spectrum and coherence function," *Medical & Biological Engineering & Computing*, vol. 29, pp. 225–241, 1991.

[63] R. B. Blackman and J. W. Tukey, *The Measurement of Power Spectra from the Point of View of Communications Engineering.* New York: Dover, 1959.

[64] G. M. Jenkins and D. G. Watts, *Spectral Analysis and its Applications.* Holden-Day, 1966.

[65] R. J. Higgins, *Digital Signal Processing in VLSI.* Prentive Hall Inc., 1990.

[66] S. E. Rittgers, W. W. Putne, and R. W. Barnes, "Real-time spectrum analysis and display of directional Doppler ultrasound blood velocity signals," *IEEE Transactions on Biomedical Engineering*, vol. 27, pp. 723–728, 1980.

[67] N. Levinson, "The Wiener RMS error criterion in filter design and prediction," *Journal of Mathematics and Physics*, vol. 25, pp. 261–78, 1947.

[68] J. P. Burg, "Maximum entropy spectral analysis," in *Modern Spectrum Analysis*, New York: IEEE Press, 1978.

[69] A. Van den Bos, "Alternative interpretation of maximum entropy spectral analysis," *IEEE Transactions on Information Theory*, vol. IT-17, pp. 493–494, July 1971.

[70] H. Akaike, "Fitting autoregressive models for prediction," *Annals of the Institute of Statistical Mathematics*, vol. 21, pp. 243–247, 1969.

[71] H. Akaike, "Power spectrum estimation through autoregression model fitting," *Annals of the Institute of Statistical Mathematics*, vol. 21, pp. 407–419, 1969.

[72] H. Akaike, "A new look at the statistical model identification," *IEEE Transactions on Automatic Control*, vol. AC-19, pp. 716–723, Dec. 1974.

[73] E. Parzen, "Some recent advances in time series modelling," *IEEE Transactions on Automatic Control*, vol. AC-19, pp. 723–730, Dec. 1974.

[74] M. J. Flynn, "Very high-speed computing systems," *Proceedings of the IEEE*, vol. 54, pp. 1901–1909, Dec. 1966.

[75] Inmos Ltd., *Transputer Overview - The Transputer Databook*, 2nd ed., 1989.

[76] D. May, "The transputer implementation of OCCAM," in *Parallel Processing in Control: The Transputer and Other Architectures* (P. Fleming, ed.), pp. 85–98, Peter Peregrinus Ltd., 1988.

[77] D. K. Wilde and O. Sié, "Regular array synthesis using ALPHA," in *Proc. International Conference on Application Specific Array Processors ASAP'94*, pp. 200–211, 1994.

[78] D. I. Moldovan, "On the design of algorithms for VLSI systolic arrays," in *Proceedings of the IEEE*, vol. 71, pp. 113–120, 1983.

[79] S. V. Rajopadhye, "Systolic arrays for LU decomposition," in *Proceedings of IEEE International Symposium on Circuits and Systems*, pp. 2513–2516, 1988.

[80] R. P. Brent and F. T. Luk, "Computing the Cholesky factorisation using a systolic architecture," in *Proceedings of 6th Australian Computer Science Conference, Sydney*, vol. 5, pp. 295–302, 1983.

[81] W. M. Gentleman and H. T. Kung, "Matrix triangularization by systolic arrays," in *SPIE: Real-Time Signal Processing IV*, vol. 298, pp. 19–26, 1981.

[82] P. Quinton and Y. Robert, *Systolic Algorithms and Architectures.* Prentice Hall, 1991.

[83] C. R. Ward, P. J. Hargrave, and J. G. McWhirter, "A novel algorithm and architecture for adaptive digital beamforming," *IEEE Transactions on Antennas and Propagation*, vol. AP-34, pp. 338–346, Mar. 1986.

[84] I. K. Proudler, J. G. McWhirter, and T. J. Shepherd, "Computationally efficient QR decomposition approach to least squares adaptive filtering," in *IEE Proceedings-F*, vol. 138, pp. 341–353, Aug. 1991.

[85] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Transactions on Electronic Computers*, vol. EC-8, pp. 330–334, Sept. 1959.

[86] J. S. Walther, "A unified algorithm for elementary functions," in *Proceedings of Spring Joint Computer Conference*, vol. 33, pp. 379–385, AFIPS, 1971.

[87] H. M. Ahmed, J.-M. Delosme, and M. Morf, "Highly concurrent computing structures for matrix arithmetic and signal processing," *IEEE Computer*, vol. 15, pp. 65–82, Jan. 1982.

[88] K. Hwang, *Computer Arithmetic: Principles Architecture and Design.* John Wiley & Sons, 1979.

[89] G. L. Haviland and A. A. Tuszynski, "A CORDIC arithmetic processor chip," *IEEE Transactions on Computers*, vol. C-29, pp. 68–79, Feb. 1980.

[90] M. Morf, C. H. Muravchik, P. H. Ang, and J.-M. Delosme, "Fast Cholesky algorithms and adaptive feedback filters," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'82)*, vol. 3, (Paris, France), pp. 1727–1731, May 1982.

[91] J.-A. Lee and T. Lang, "Matrix triangularization by fixed point redundant CORDIC with constant scale factor," in *SPIE: Advanced Signal-Processing Algorithms, Architectures, and Implementations*, vol. 1348, pp. 430–447, 1990.

[92] J. M. Delosme, "VLSI implementations of rotations in pseudo-Euclidean spaces," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 2, pp. 927–930, 1983.

[93] E. C. Ifeachor and B. W. Jervis, *Digital Signal Processing - A Practical Approach.* Addison-Wesley Publishing Co., 1993.

[94] J. H. Wilkinson, *Rounding Errors in Algebraic Processes.* Her Majesty's Stationary Office, 1963.

[95] T. A. E. Problem, *J. H. Wilkinson.* Clarendon Press, Oxford, 1967.

[96] R. S. Martin, G. Peters, and J. H. Wilkinson, "Symmetric decomposition of a positive definite matrix," in *Linear Algebra*, Springer-Verlag Berlin, 1971.

[97] C. Dixon, *Numerical Analysis.* Blackie Chambers, 1974.

[98] D. I. Moldovan and J. A. B. Fortes, "Partitioning and mapping algorithms into fixed size systolic arrays," *IEEE Transactions on Computers*, vol. C-35, pp. 1–12, Jan. 1986.

[99] J. J. Navarro, J. M. Llaberia, and M. Valero, "Computing size-independent matrix problems on systolic array processors," in *IEEE 13th International Symposium on Computer Architecture*, pp. 271–279, 1986.

[100] J. J. Navaro, J. M. Llaberia, and M. Valero, "Solving matrix problems with no size restriction on a systolic array processor," in *IEEE International Conference on Parallel Processing*, pp. 676–683, 1986.

[101] A. Bateman and W. Yates, *Digital Signal Processing Design.* Pitman Publishing, 1988.

[102] G. M. Megson, *An Introduction to Systolic Algorithm Design.* Clarendon Press - Oxford, 1992.

[103] L. W. Chang and M. Y. Chen, "A new systolic array for discrete Fourier transform," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 36, pp. 1665–1666, Oct. 1988.

[104] J. A. Beraldin, T. Aboulnasr, and W. Steenaart, "Efficient one-dimensional systolic array realization of the discrete Fourier transform," *IEEE Transactions on Circuits and Systems*, vol. 36, pp. 95–100, Jan. 1989.

[105] D. C. Kar and V. V. B. Rao, "A new systolic realization for the discrete Fourier transform," *IEEE Transactions on Signal Processing*, vol. 41, pp. 2008–2010, May 1993.

[106] N. R. Murthy and M. N. S. Swamy, "On the real-time computation of DFT and DCT through systolic architectures," *IEEE Transactions on Signal Processing*, vol. 42, pp. 988–991, Apr. 1994.

[107] H.-G. Yeh and H.-Y. Yeh, "Implementation of the discrete fourier transform on 2-dimensional systolic processors," *IEE Proceedings, Part G*, vol. 134, pp. 181–185, Aug. 1987.

[108] K. J. Jones, "High-throughput, reduced hardware systolic solution to prime factor discrete Fourier transform algorithm," *IEE Proceedings, Part E*, vol. 137, pp. 191–196, May 1990.

[109] M. H. Lee, "High speed multidimensional systolic arrays for discrete Fourier transform," *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, vol. 39, pp. 876–879, Dec. 1992.

[110] C.-Y. Chen and C.-L. Wang, "A new efficient systolic architecture for the 2-D discrete fourier transform," in *Proceedings of IEEE International Symposium on Circuits and Systems*, pp. 689–692, 1992.

[111] D. Wood, R. A. Evans, and K. W. Wood, "An 8 bit serial convolver chip based on a bit level systolic array," in *IEEE Proceedings of the Custom Integrated Circuits Conference*, pp. 256–261, May 1983.

[112] M. A. Bayoumi and C. H. Yang, "Reconfigurable testable bit-serial multiplier for DSP applications," in *IEE proceedings-E*, vol. 136, pp. 517–523, Nov. 1989.

[113] H. L. Groginsky and G. A. Works, "A pipeline fast Fourier transform," *IEEE Transactions on Computers*, vol. C-19, pp. 1015–1019, Nov. 1970.

[114] J. E. Whelchel, J. P. O'Malley, W. J. Rinard, and J. F. McArthur, "The systolic phase rotation FFT - a new algorithm and parallel processor architecture," in *Proceedings of International Conference on Acoustics, Speech and Signal Processing (ICASSP 90)*, vol. 4, pp. 1021–1024, 1990.

[115] J. Choi and V. Boriakoff, "A new linear systolic array for FFT computation," *IEEE Transactions on Circuits and Systems-II Analog and Digital Signal Processing*, vol. 39, pp. 236–239, Apr. 1992.

[116] S. I. Sayegh, "A pipeline processor for mixed size FFT's," *IEEE Transactions on Signal Processing*, vol. 40, pp. 1892–1900, Aug. 1992.

[117] T. Willey, R. Chapman, H. Yoho, T. S. Durrani, and D. Preis, "Systolic implementations for deconvolution, DFT and FFT," *IEE Proceedings, Part F*, vol. 132, pp. 466–472, Oct. 1985.

[118] J. V. McCanny, J. G. McWhirter, and S.-Y. Kung, "The use of data dependence graphs in the design of bit-level systolic arrays," *IEEE Transactions on Acoustics Speech and Signal Processing*, vol. 38, pp. 787–793, May 1990.

[119] J. V. McCanny, J. G. McWhirter, and K. Wood, "Optimised bit level systolic array for convolution," in *IEE Proceedings-F*, vol. 131, pp. 632–637, 1984.

[120] D. P. Agrawal, "High-speed arithmetic arrays," *IEEE Transactions on Computers*, vol. C-28, pp. 215–224, Mar. 1979.

[121] M. Ercegovac and T. Lang, *Division and Square Root, Digit-Recurrence Algorithms and Implementations.* Kluwer Academic Publishers, 1994.

[122] C. A. Mead and L. A. Conway, *Introduction to VLSI Systems*, ch. 8, pp. 263–332. Reading Massachusetts: Addison-Wesley Publishing Company, 1980.

[123] V. K. Jain, G. E. Perez, and J. M. Wills, "Novel reciprocal and square-root cell: Architecture and application to signal processing," in *Proceedings of International Conference on Acoustics Speech and Signal Processing (ICASSP'91)*, vol. 2, (Toronto, Ontario, Canada), pp. 1201–1204, 1991.

[124] H. H. Guild, "Some cellular logic arrays for nonrestoring binary division," *The Radio and Elec. Engineer*, vol. 39, pp. 345–348, 1970.

[125] Cadence Design Systems, Inc., *Digital Logic Simulation Verilog XL Reference Manual*, Mar. 1991.

[126] J. W. Bruce, P. J. Giblin, and P. J. Rippon, *Microcomputers and Mathematics.* Cambridge University Pree, 1990.

[127] The Math Works, Inc., *Matlab Reference Guide*, Aug. 1992.

[128] W. P. Marnane, C. J. Jordan, and F. J. O'Reilly, "Compiling regular arrays onto FPGAs," in *Lecture Notes in Computer Science 975, Field-Programmable Logic and Applications* (W. Moore and W. Luk, eds.), pp. 178–187, Springer-Verlag, 1995.

[129] Xilinx, 2100 Logic Drive, San Jose, California 95124, *Xilinx - The Programmable Logic Data Book*, 1994.

[130] A. Fort, C. Manfredi, and S. Rocchi, "Adaptive SVD-based AR model order determination for time-frequency analysis of Doppler ultrasound signals," *Ultrasound in Medicine and Biology*, vol. 21, no. 6, pp. 793–805, 1995.

[131] Xilinx, Inc, 2100 Logic Drive, San Jose, California 95124, *Xilinx - User Guide and Tutorials*, 1991.

[132] W. P. Marnane, C. J. Jordan, and F. J. O'Reilly, "Synthesising a FIR filter onto a FPGA," in *Proceedings of the Irish DSP and Control Colloquium*, pp. 5–12, June 1995.

[133] W. P. Marnane, *Structured Test of VLSI Arrays.* PhD thesis, University of Oxford, 1989.