Dense Stereo Reconstruction in a Field Programmable Gate Array

by

Siraj Sabihuddin

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto

Abstract

Dense Stereo Reconstruction in a Field Programmable Gate Array

Siraj Sabihuddin

Master of Applied Science

Graduate Department of Electrical and Computer Engineering

University of Toronto

2008

Estimation of depth within an imaged scene can be formulated as a stereo correspondence problem. Software solutions tend to be too slow for high frame rate (i.e. ≥ 30 fps) performance. Hardware solutions can result in marked improvements. This thesis explores one such hardware implementation that generates dense binocular disparity estimates at frame rates of over 200 fps using a dynamic programming formulation (DPML) developed by Cox *et al.* [4]. A highly parameterizable field programmable gate array implementation of this architecture demonstrates equivalent accuracy while executing at significantly higher frame rates to those of current approaches. Existing hardware implementations for dense disparity estimation often use sum of squared difference, sum of absolute difference or other similar algorithms that typically perform poorly in comparison to DPML. The presented system runs at 248 fps for a resolution of 320×240 pixels and disparity range of 128 pixels, a performance of 2.477 billion DPS.

Dedication

This thesis is dedicated to all the people who kept me company during my studies. Without them, I might very well have found myself trapped alone in a rather dingy little room. I think there is a somewhat profound connection between a person's motivations in life and their environment ... thank you all for making my stay a pleasant one.

Acknowledgements

First and foremost, I would like to extend my gratitude towards W. James MacLean. James you have been a good supervisor. I enjoyed our conversations and really appreciated your advise and input. And thank you for being patient through all my fumbling attempts at research.

I would also like to thank all the *old* folks from the Vision Lab. I particularly enjoyed the company of Christina Cabani and Joshua Worby early during my Masters. Thanks also to Divyang Masrani who occasionally pops in to restart a crashed simulation. Oh and then there is my Romanian, Iranian, Chinese, Polish, French, Russian and Indian friends in the Systems Lab. I don't know how you all ended up in Canada, but I'm glad you did — we had some truly interesting arguments.

I am grateful to my friends in Ryerson and Queens who helped me with my work and, after some gentle persuasion, also fed me. A big thank you to Jamin Islam, Valeri Kirishchian, Peter Chun and Michael Belshaw — don't think you have gotten rid of me!

Finally, I'm grateful to the dedicated people at ECE Help who patiently diverted repeated requests for the root password. It was probably for the best, who knows what kind of mess I would have gotten into. And of-course thanks, in advance, to my committee members for slogging through another rather long thesis.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Humans use vision to effortlessly detect, identify and track objects in a three-dimensional (3D) world. A key component of this ability is the perception of depth. While this perception has many facets, one of particular importance is stereo imaging, whereby a 3D scene is projected onto two individual eyes from slightly different viewpoints. These differing viewpoints are used by the brain to reconstruct the original 3D scene for tracking and navigation purposes.

The process of 3D reconstruction can be duplicated in artificial systems using digital cameras in place of eyes and a computer system as the brain. Since extraction of such 3D structure from two (or more) two dimensional (2D) images is essentially a matter of triangulation, it becomes necessary to identify points, in each viewpoint, that represent the same 3D position in the scene. This process of finding point corresondences between multiple 2D images, of a particular 3D environment, is referred to as *stereo matching* or *stereo correspondence*.

Algorithms for determining these correspondences are far from trivial and have been studied extensively by the computer vision community with the aim of mimicing and exceeding biological vision — it is conceivable that such algorithms could find applications in environments not normally encountered in human activities where resolutions and

motion capture speeds in excess of human visual acuity may be required.

This thesis performs such high speed depth estimation using a hardware architecture for a dynamic programming maximum likelihood (DPML) matching algorithm originally developed by Cox *et al.* [4]. This highly parallelized DPML hardware (DPMLHW) implementation demonstrates vastly superior frame rates ($\geq$ 200 fps) to existing stereo matching systems. These frame rates are generated at high resolutions (640×480 pixels) and with high degrees of accuracy in the resulting 3D reconstruction — an accuracy comparable to the best of existing systems.

Correspondence algorithms are typically classified into dense and sparse methods. DPML is a dense method which attempts to find correspondences, and subsequently the disparity between correspondences, for every pixel location in a particular 2D reference image. This disparity, in-turn, allows the reconstruction of depth at all pixel locations. Sparse methods look at higher level features (*e.g.* corners) and generate correspondences between these features for a depth estimate at the particular feature's pixel location — typically there are far fewer high level features in an image than there are pixels.

The search for matches may be cast into a probabilistic framework, whereby regions in a stereo pair demonstrating high correlation are considered more likely candidates for a correspondence. DPML makes use of a maximum likelihood (ML) formulation for this purpose.

Frameworks and models of increasing complexity may improve performance, in such situations, at the cost of additional computational and memory resources. This complexity, combined with the high volume of data coming from stereo camera system, limits the viability of implementations of stereo matching algorithms for *real* time use in general purpose processor systems. Dynamic programming (DP) optimization techniques, as utilized by DPML, can help to reduce computational load and make such implementations more feasible. This is done by exploiting the bottom up substructure of a solution to reduce repeated calculations, and thus, the time required to arrive at an optimal solution

(see Cormen *et al.* [3] for details).

Further improvements in speed and frame rates can be achieved by using custom processing hardware (HW). These custom implementations are often termed as Application Specific Integrated Circuits (ASICs). The highly parallelized nature of stereo matching tasks allows ASICs to process multiple pieces of data simultaneously rather than seqentially. However, design and development cycles for ASICs tend to be lengthy and costly, so it is common to design prototypes using Field Programmable Gate Arrays (FPGAs). These FPGAs contain arrays of reconfigurable logic blocks that allow for rapid hardware prototyping at relatively low costs.

This document begins in Chapter 2 with an introduction to stereo reconstruction, optimization techniques, FPGA systems and a review of recent literature. Following this, Chapter 3 proposes a set of novel hardware implementations of Cox's DPML [4] algorithm. These are the only hardware implementations of DPML based stereo correspondence presently known to exist. Finally, Chapter 4 demonstrates the results generated from an actual hardware implementation and compares them to other state-of-the-art stereo implementations. Chapter 5 concludes with a proposal of possible directions for future work.

Chapter 2

Background

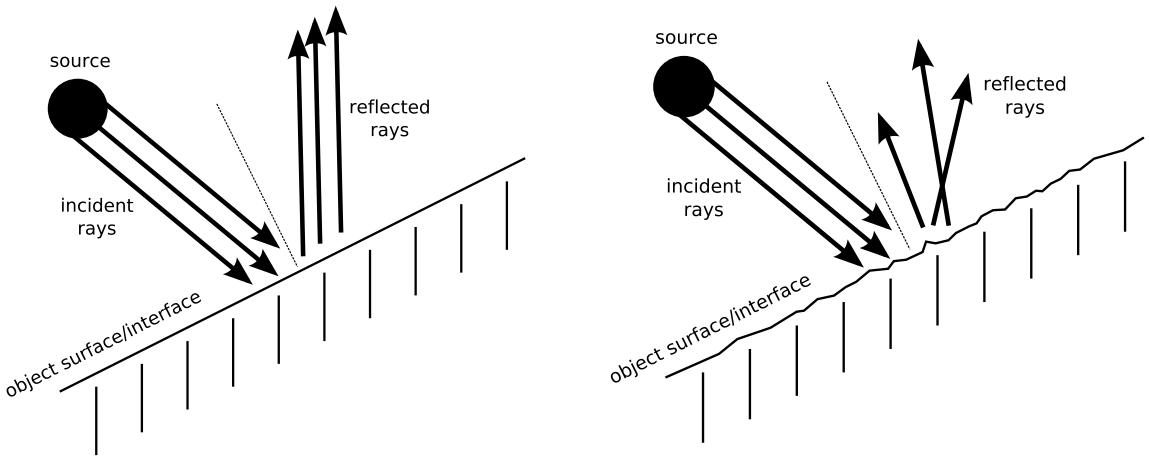
The process of 3D reconstruction from multiple 2D images is based on a vast body of pre-existing literature. This chapter discusses some of the theories that have formed the foundations of present day stereo reconstruction and, likewise, the foundations of hardware based reconstruction. This chapter begins, first, in Section 2.1 with a discussion of typical camera systems. Section 2.2 continues to put these camera systems into the context of stereo and 3D reconstruction. In order to perform matching tasks associated with reconstruction, some form of optimization process takes place — the set of most likely matches resulting from these optimizations are used as a basis for triangulation. These correspondence and optimization techniques are discussed in Section 2.3. Some of these techniques are inherently more suited to hardware implementations. Section 2.4 introduces the paradigm of hardware-based algorithms along with the advantages and problems associated with such systems. To provide context for results in Chapter 4, Section 2.5 reviews recent literature pertaining to hardware implementations of stereo correspondence algorithms.

2.1 Scene and Camera Modeling

2.1.1 Reflectance

Ultimately, the goal of a computer vision system is the interpretation of an imaged scene. In order to perform any kind of interpretation, some understanding of the visual environment is necessary, at least at a macroscopic level. Such an environment typically consists of light sources and sinks. Sources emit rays of light that travel in straight lines until they encounter some obstacle. The power or intensity of the light emitted is denoted by the term irradiance. As the number of rays passing through a given surface patch increase, this intensity or power also increases.

Due to inherent spectral properties of an encountered object, certain wavelengths of light are absorbed. These absorbed rays may then be re-emitted at the same or lower energies. In the macroscopic world this absorbtion and emission of light is responsible for the visualization of objects (*i.e.* certain frequencies of light reflected or re-emitted by an object give it its characteristic colour or shape). Reflectance modeling provides a method of predicting how light is absorbed and emitted by objects. Typically this reflectance is roughly classified as specular or diffuse (see Figure 2.1). Specular reflectance describes light re-emitted to generate a more mirror-like reflection with spectral distribution properties of the incoming (incident) rays being similar to those of the outgoing (reflected) light. Diffuse reflectance refers to a scattering of reflected rays such that the spectral distribution does not match that of the incident rays. This has the effect of making the apparent brightness of a viewed object the same regardless of viewing angle. Modeling the reflectance properties of objects provides a way to predict intensity patterns of the resulting images taken by a camera system. These properties become particularly important when discussing stereo matching — Section 2.3 provides a more detailed analysis of this topic.



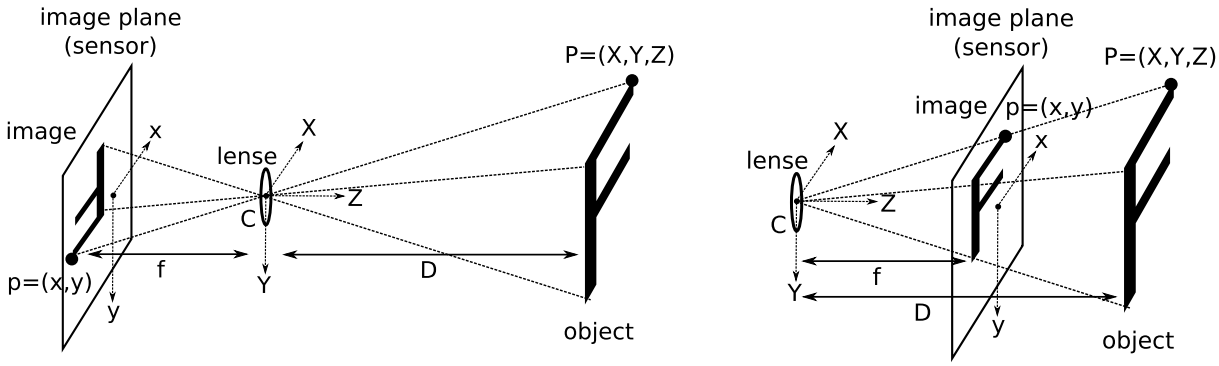
(a) Specular Reflectance (b) Diffuse Reflectance

Figure 2.1: A rough categorization of reflectance. Modeling reflectance properties of an object is useful for predicting the way light enters a camera.

2.1.2 Single View Geometry: The Camera Model

Camera systems take snapshots of the light reflected by an object. It is common to model such cameras using the pinhole camera model. Figure 2.2 presents a visual of this single view model. Rays of light pass through a lense assembly and intersect with an imaging patch or plane (Figure 2.2a). This patch takes the form of photographic film or digital imaging sensors and records the spectral distribution of light at that particular location in space. For mathematical convenience it is common to move the image plane in front of the lense assembly as shown in Figure 2.2b. Furthermore, the lense assembly is also typically replaced and modeled as a pinhole.

The value f represents the focal length of the camera lense while D represents the depth or distance of the object. The focal length and depth measurements are typically in world coordinates and relative to the camera centre, C. The optic axis of the lense assembly passes through this camera centre at an angle perpendicular to the image plane. Imaged data exists only in 2D (*i.e.* in image coordinates) and is relative to the intersection of the optic axis with the aforementioned plane. The point of intersection is called the principal point. The process of transforming the 3D visual information received by the



(a) A pinhole camera model

(b) A modified pinhole camera model

Figure 2.2: A pinhole camera model. Figure A shows the standard model while Figure B shows a modified model with the image plane moved in front of the lense assembly.

camera into a 2D image is referred to as perspective projection. A point, $P = (X, Y, Z)$, on an object may be related to its perspective projection $p = (x, y)$ by Equation 2.1. Equation 2.1 can then be written as a system of equations in homogenecous coordinates (with $\bar{p} = [p \mid 1]^T$ and $\bar{P} = [P \mid 1]^T$) as shown in Equation 2.2. Note that $\lambda = 1/Z$ and that $O_{3\times1}$ is a zero matrix of size 3×1.

$$-\frac{f}{Z} = \frac{x}{X} = \frac{y}{Y} \tag{2.1}$$

$$\bar{p} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \lambda \begin{bmatrix} -f & 0 & 0 \\ 0 & -f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \lambda K \left[I_{3\times3} \mid O_{3\times1} \right] \bar{P} \tag{2.2}$$

The pinhole camera model is useful as a starting point for depth estimation algorithms (see Section 2.2). However, while a useful guide, it deviates from real camera systems. It also assumes that the alignment of the lense assembly is such that the optic axis is

perfectly perpendicular to the imaging plane — in reality, alignment of the assembly is never exact and can thus result in skew distortions. Furthermore, the lense itself may cause radial distortions to the incoming light. Light may fall unevenly on the image sensor and result in scaling differences between x and y coordinates. Such uneven distribution could, for example, be due to flaws or asymmetries in the lense assembly. Tangential distortions are an example of such asymmetries.

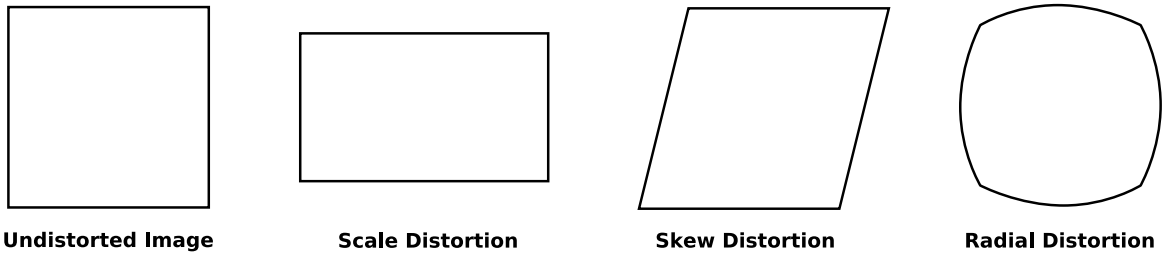


Figure 2.3: Types of distortion resulting from the lense assembly. These distortions can be accounted for by modifying the basic pinhole camera model.

To correct for these problems Equation 2.2 can be be modified with parameters that represent these distortions. These parameters are known as the intrinsic parameters of the camera. Equation 2.3 models these parameters. K forms what is known as the camera calibration matrix with skew parameter, s, scaling factors, k_x and k_y and camera principal point, (x_c, y_c). The function $L(\bar{r})$ describes the radial distortion introduced by the lense assembly as a function of the radius, $\bar{r}$, relative to the principal point. Note that tangential distortion is typically not a major contributor to distortion in the resulting image and as such is omitted from the model.

$$\bar{p} = \lambda\ L(\bar{r})\ K\ \ [I_{3\times3} \mid O_{3\times1}]\ \ \bar{P} = \lambda\ L(\bar{r}) \begin{bmatrix} k_x f & s & x_c \\ 0 & k_y f & y_c \\ 0 & 0 & 1 \end{bmatrix} [I_{3\times3} \mid O_{3\times1}]\, \bar{P} \qquad (2.3)$$

The perspective projection is defined in terms of a camera centric coordinate system. It is possible to move from these camera centric coordinates, $P = (X, Y, Z)$, to world coordinates, $\tilde{P} = (\tilde{X}, \tilde{Y}, \tilde{Z})$. A relationship can be defined by performing euclidean transformations (rotation (R) and translation (T)) to align the two coordinate systems. The parameters associated with this transformation form the extrinsic parameters of the camera system and are defined by the matrix τ (see Equation 2.4). Equation 2.5 combines models for the intrinsic and extrinsic parameters into a projection matrix P_r.

$$\bar{P} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} \tilde{X} \\ \tilde{Y} \\ \tilde{Z} \\ 1 \end{bmatrix} = \tau \tilde{P} \tag{2.4}$$

$$\bar{p} = \lambda \; L(\bar{r}) \; K \left[I_{3\times 3} \mid O_{3\times 1} \right] \; \tau \; \tilde{P} = P_r \; \tilde{P} \tag{2.5}$$

Other issues cannot be modeled explicitly with Equation 2.5. A perfect pinhole aperture ensures an infinite depth of focus and perfectly sharp images on the image plane — practical systems cannot ensure such focus resulting in images with blurred edges. Additionally, recording mediums tend to be more sensitive to certain frequencies of light than others. This means that the visual representation of the scene may differ from the actual spectral information entering the camera. This same recording medium also quantizes light information into discrete values at discrete sample locations, a process which has associated quantization errors and aliasing problems.

Another particularly important form of error occurs when the recording medium itself is susceptible to noise. This noise may be a result of the electronic properties of a CCD or CMOS sensor for a particular pixel location. Sensors are often prone to producing

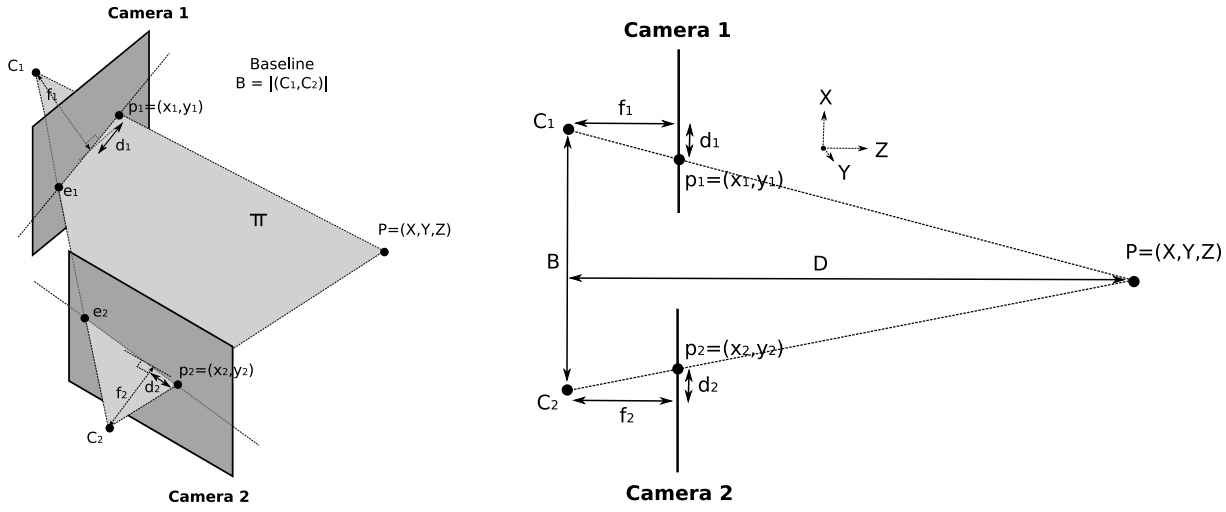
spurious data under certain lighting conditions. To correct for these forms of noise it is common to apply image filtration techniques. Gaussian and median filtration are two common forms of filtering used to smooth image data and improve signal to noise ratio. These techniques have been applied in Chapter 4 to improve disparity estimation on noisy inputs.

A more thorough review of theory associated with single view geometry and related camera modeling can be found in work published by Hartley and Zisserman [10].

2.2 Two View Stereo Geometry

An image from a single camera (in a fixed position) cannot be used to perform the inverse of the perspective projection, the transformation from 2D to 3D, without additional information. This information may take the form of contextual knowledge (*e.g.* some knowledge about how objects scale over distance). This additional knowledge, assuming that object identification is possible first, provides constraints by which a particular position in 2D image data may be related to depth. A somewhat simpler method relies on two images of a particular scene from different viewpoints. With the second image it is possible to reduce depth estimation to a problem of triangulation. This triangulation relies on the fact that points in each of the two images can be identified as representing the same 3D position. Figure 2.4a presents a typical binocular (*i.e.* two camera) stereo vision system.

Each camera in the figure is represented by the pinhole camera model. The points e_1 and e_2 are the epipoles of the camera system and are given by the intersection of the baseline with the image planes of each camera. The baseline, itself, is defined as the line segment that joines the principal points of each camera (*i.e.* as line (C_1, C_2)). B is the length of this line segment. A quick review of scene geometry demonstrates that a point, $P = (X, Y, Z)$, projects on each camera at $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ such that the



(a) Generalized Stereo (b) Rectified Stereo Setup

Figure 2.4: A binocular stereo camera system.

pair lie on epipolar lines defined by (e_1, p_1) and (e_2, p_2) respectively. Furthermore, the plane represented by these two lines is called the epipolar plane, π, and contains the scene point and its corresponding projections in camera 1 and 2. Given a point in one image, the epipolar lines define a region, in the other, within which the matching point is expected to be found. Since the camera model is known, finding the corresponding points in the stereo image pair for a given scene point, P, remains the greatest challenge of triangulation, and hence, depth estimation. The search for finding matching pixel correspondences can be reduced to a 1D search once the epipolar geometry is known. This geometry, given by epipolar lines l_1 and l_2, can be determined by computing the fundamental matrix F. For a given point in l_1 the corresponding point can be found in l_2. Equations 2.6 and 2.7 define the epipolar lines. Note that $\bar{p}_1 = [p_1 \mid 1]^T$ and $\bar{p}_2 = [p_2 \mid 1]$ represent p_1 and p_2 in homogeneous coordinates. Likewise $\bar{l}_1$ and $\bar{l}_2$ are the epipolar lines represented in homogeneous coordinates.

$$\bar{l_2} = \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = F\ \bar{p_1} \tag{2.6}$$

$$\bar{l_1} = \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix}^T \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = F^T\ \bar{p_2} \tag{2.7}$$

It is often useful to transform images obtained from the two camera systems to reflect this epipolar geometry more clearly. A process called rectification determines the transformation between the two camera images such that their respective epipolar lines are aligned parallel and colinear along a single axis (see Figure 2.4b). The rectification process uses F to warp the camera images such that pixels along corresponding scanlines are representative of matching epipolar lines. That is, rectification ensures that matching pixels in the two images lie on the same scanline. In a nut shell, the fundamental matrix encapsulates the intrinsic and extrinsic camera parameters such that Equation 2.8 holds true. A calibration process, requiring several known P, p_1, p_2 point relationships, is required to determine this matrix. Relating Equations 2.5 and 2.8 it is clear that knowledge of the fundamental matrix allows the computation of projection matrices, P_{r1} and P_{r2}, for each of the cameras in the stereo setup. It is important to note that the relationship in Equation 2.8 holds true even if the two views are not rectified.

$$\bar{p_2}^T F \bar{p_1} = 0 \tag{2.8}$$

The values x_1 and x_2 allow disparity computation. The disparity of a particular pixel pair is the displacement of the two pixels with respect to each other and relative to the

principal point of the images (*i.e.* $d = |d_2 - d_1| = |x_2 - x_1|$ pixels). Make note that this is possible because the principal points of the two images are aligned to the same image coordinates after rectification. Depth of a 3D world point corresponding to the matched pixel pair can be computed from the disparity and camera parameters using Equation 2.9. $f = f_1 = f_2$ represents the focal length of the two rectified cameras while B represents the length of the baseline, α a pixel/mm scale factor, d the disparity, and D the depth.

$$D = \frac{fB\alpha}{d} \tag{2.9}$$

Clearly, disparity is inversely proportional to depth up to some scale factor. This scale factor is determined by camera parameters. It is convenient to use disparity as a unit of measure for 3D structure in a scene since it is independant of the particular camera system being used for stereo imaging. This document will, therefore, use the terms disparity and depth interchangeably to refer to 3D reconstruction results.

A more thorough review of theory associated with two view geometry and 3D reconstruction may be found in work published by Hartley and Zisserman [10].

2.3 Stereo Matching

A stereo matching process is used to identify corresponding points in each of two 2D images from a stereo camera system. Point correspondences, as mentioned earlier, provide a means to perform triangulation. Some form of cost function, likelihood function or similarity metric is required to identify pairs of points across images that are good candidates for matches.

Algorithms for determining correspondences have been studied extensively by the computer vision community. This thesis focuses on image based algorithms which are classified into dense and sparse methods. Sparse methods use some form of feature

matching (*e.g.* Corners, SIFT, *etc.*) to identify matching points in each view of the stereo image. Disparity estimates for these matching points are used to generate sparse depth maps. Some algorithms may then perform a region growing or filling operation to propagate disparity estimates to neighbouring regions around the sparse points. These methods can thus be used to generate dense maps also. Dense methods work at the pixel level rather than the feature level but operate in much the same way as sparse methods (as far as feature/pixel matching is concerned). Correlation window based matching is quite common among matching algorithms. These methods compute a correlation score between windows centred at a particular pixel or feature in each of the two images of the stereo pair. For a given feature in one image the window slides over nearby coordinates in the other image to produce similarity scores between the representative features. The best scoring feature pairs are thus labeled as matches and a depth or disparity estimate computed. It is important to remember that the rectification process provides an important step to reducing the search space over which cost estimates are generated and makes the matching problem more computationally tractable. As mentioned earlier, this is because it ensures that matching features in the two images lie on the same scanline. A more thorough review of both sparse and dense methods is provided in a brief literature review in Section 2.5. Furthermore, Figure 2.5 provides a visual flow diagram of how a sparse or dense estimation approach may work.

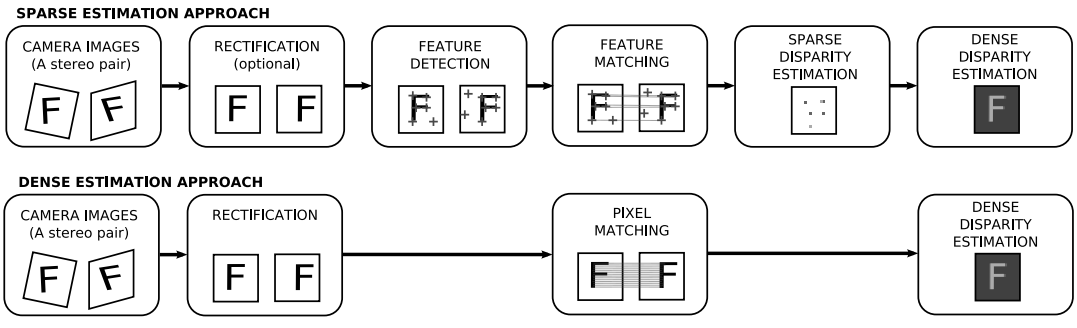


Figure 2.5: A flow chart of sparse and dense disparity estimation. Features may be pixel level or higher level features.

As an example, Equations 2.10, 2.11 and 2.12 describe three potential cost functions that can be used for feature or pixel comparisons. Equations 2.10 and 2.11 describe Sum of Squared Difference (SSD) and Sum of Absolute Difference (SAD) functions with windows W_1 and W_2 containing camera 1 and 2 features/pixels p_1 and p_2. $I(p)$ indicates the intensity or magnitude of feature/pixel p. Equation 2.12 describes a Normalized Cross-Correlation (NCC) based function that takes into account image statistics within a window (*i.e.* the mean (μ_{W_1}, μ_{W_2}) and the standard deviation (σ_{W_1}, σ_{W_2})).

$$C_{SSD} = \sum_{\forall p_{1,2} \in W_{1,2}} (I(p_1) - I(p_2))^2 \tag{2.10}$$

$$C_{SAD} = \sum_{\forall p_{1,2} \in W_{1,2}} |I(p_1) - I(p_2)| \tag{2.11}$$

$$C_{NCC} = \sum_{\forall p_{1,2} \in W_{1,2}} \frac{(I(p_1) - \mu_{W_1})(I(p_2) - \mu_{W_2})}{\sigma_{W_1}\sigma_{W_2}} \tag{2.12}$$

2.3.1 Stereo Issues

Some important issues arise when performing stereo matching. The first of these issues occurs when pixels or features are visible from one camera but not the other. The absence of an expected feature indicates the presence of an occlusion in the scene. The absence of points b and e in Figure 2.6 demonstrate occlusions in the right and left cameras respectively. During stereo matching this occlusion could result in an attempt to assign a depth estimate to a feature that has no corresponding point for triangulation. As such, algorithms penalize the similarity metric to take the presence of occlusions into account - this penalization could be done by adding a fixed occlusion cost to the metric. With large baseline stereo cameras there may be situations where a significant percentage of the scene visible in one camera will not be visible in the other.

A second issue occurs as a result of thin objects. A typical approach to stereo matching

assumes that certain ordering is preserved between neighbouring scene points and their respective images. Consider a particular scene point which generates corresponding image points, p_1 and p_2, as a result of perspective projection. Suppose, now, that a narrow object were positioned close to the stereo camera system at some central location. This *thin occluder* imaged in camera 1 might generate an image point, $\check{p}_1$, to the left of p_1 while generating an image, $\check{p}_2$ to the right of p_2 in camera 2. In effect neighbouring points $\check{p}_1$ and p_1 in image 1 occur in the reverse order to neighbouring points $\check{p}_2$ and p_2 in image 2 thus violating the ordering assumption. The ordering of these points in the presence of a thick occluder, however, is not reversed.

Cameras in a stereo setup may not be equidistant from a particular object. Alternatively differences in magnification between the two cameras may exist. This means that the resulting images, in the two views, can show scale variations with respect to each other. These scale changes can adversely effect feature detection and subsequent stereo matching. Scale changes are a significant issue and may result in a loss of detail or other distortions. Another issue occurs in regions of low texture. Most feature detection algorithms make use of image gradients. Notable exceptions use image statistics via histograms, correlation windows *etc.* In regions where there is very little texture gradient estimates tend to be extremely inaccurate resulting in poor quality depth estimates. Likewise, pixel intensity comparisons yield similar cost values for locations along an untextured region, which, in turn, yield potentially incorrect depth assignments. Highly textured regions or points of discontinuity (resulting from object boundaries) yield good estimates. Uncorrelated noise can also yield wildly varying gradients in a local neighbourhood. A common assumption treats this noise as normally distributed and models it into the cost function.

A further issue relates to changes in illumination across cameras. The scene lighting and camera positioning might be such that the resulting image pairs have widely varying illumination effects. These effects could, for example, introduce shadowing, which, in

turn, could confound the matching process and generate spurious disparity estimates. Images may also have differences in exposure causing further problems — exposure variations may be decreased through image normalization. It is worth noting that objects may also have varying opacities. These set of semi-transparent objects layered one on top of another can cause apparently un-correlated changes in image intensities in each distinct stereo viewpoint. The result is further correspondence problems.

A final issue returns back to reflectance modeling. Specular characteristics of objects have profound impact on disparity estimation. Light reflected off of such objects transmits information about the illuminating light source. For partially specular surfaces, some information about the material properties of the object are also transmitted. The intensity of the imaged points associated with the particular specularity are dependant upon one or more light sources at unknown positions in 3D space. In situations involving two or more cameras, each camera will view objects from slightly different viewpoints or angles — the position of the specularity present on the object will change relative to the position of the actual underlying object between these different views. Specularity may thus contradict two key assumptions that most matching processes make, *i.e.* the assumption that surfaces are lambertian (or diffusely reflective) and that ordering of points along an object remain the same between the two views.

2.3.2 Dynamic Programming Based Correspondence

In addition to the epipolar line constraints disussed earlier, further constraints and assumptions can be imposed upon the stereo correspondence search process. The first set of constraints is to assume that all objects in a scene are thick, opaque and lambertian (*i.e.* diffusely reflective) — the last two assumptions, in a sense, avoid the necessity for dealing with the more complicated aspects of reflectance modeling. An additional set of more formalized constraints are as follows:

- Ordering Constraint: The order of neighbouring correspondences on the particular epipolar lines is always perserved.
- Uniqueness Constraint: Each pixel or feature in one image only maps to one pixel or feature in the other image. A violation of this rule can be used to identify occlusions or outliers.
- Disparity Range Constraint: Corresponding pixels or features in one image can be found within some limited distance in the other. This range can be predicted from camera geometry and the location of the closest scene point.

Figure 2.6 provides a graphical description that demonstrates that these constraints hold true given opacity and thick scene objects. Points from an object will map to points in the camera images in a fixed order. Likewise, a point in one image will correspond uniquely to another point in the other image — these correspondences will occur within a fixed maximum pixel distance of each other.

Combined, these constraints provide limitations on cost computations. Once cost values are computed, the pixel distance between feature pairs corresponding to the minimum cost becomes a measure for the disparity estimate. In order to compute the parameters (*i.e.* the features or pixel pair) that minimize this cost some form of optimization must take place. This thesis focuses on a dense disparity estimation technique based on a Maximum Likelihood Dynamic Programming (DPML) optimization approach originally developed by Cox *et al.* in [4].

Dynamic programming solutions attempt to minimize computational load during optimization. They utilize three key principles: overlapping subproblems, optimal substructure and memoization. If a problem can be broken down into a set of subproblems which can then be reused several times, it is said to have overlapping subproblems. If each of these subproblems can then be shown to be the optimal solution within the scope

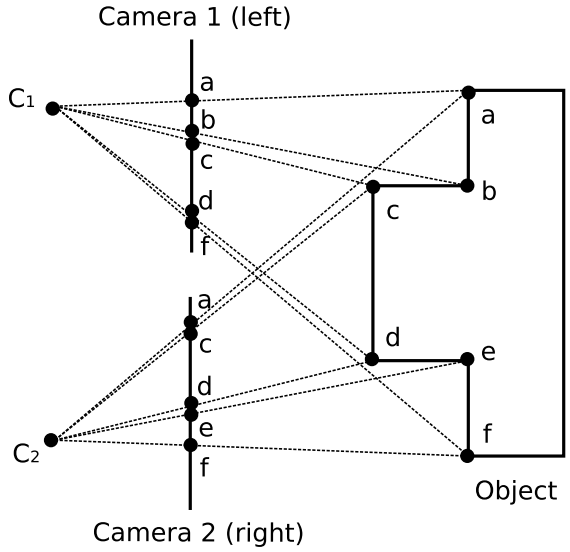


Figure 2.6: Ordering and uniqueness constraints. Points on the object map to the left and right cameras uniquely and in order. Notice that point e is occluded in camera 1 while point b is occluded in camera 2.

of its inputs, then the larger problem is said to have optimal substructure and provides a globally optimal solution. Furthermore, reuse is facilitated by mapping results to a memory indexed by the input space of the subproblems, much like a Look-Up-Table (LUT). This mapping process is called memoization.

For cases of dense disparity estimation these three principles can be utilized by relating the stereo matching process to a problem of determining the Longest Common Subsequence (LCS). The corresponding scanlines in the left and right images can be seen as a two distinct sequences, I_L and I_R. The set of matches for these two scanlines is the longest common subsequence. DPML is a solution to the LCS problem. The optimal substructure of DPML solutions means that the disparity estimation results generated from input scanlines are globally optimal. Furthermore, the simplicity of cost computations, combined with the ability to reuse previous costs, lends this solution to easy and high performance implementation. The DPML solution consists of a two pass algorithm: the forward and backward pass. The forward pass occurs first (see Algorithm 1). Right

and left image pixel streams, from corresponding scanlines, are fed in and compared sequentially to each other up to some maximum disparity range, D_{max}.

$$NOC(I_L(x), I_R(x+d), \sigma) = \frac{(I_L(x) - I_R(x+d))^2\sigma^2}{4} \tag{2.13}$$

$$OC(P_d, \sigma^2, \phi) = \log \frac{P_d\phi}{(1-P_d)}\sqrt{\frac{2\pi}{\sigma^2}} \tag{2.14}$$

Pixel comparisons produce cost estimates for a search region constrained by D_{max}. Costs are generated by comparing every pixel in the left scanline to every pixel in the right scanline. Equations 2.13 and 2.14 describe the cost functions used for pixel comparisons. $I_L(x)$ and $I_R(x)$ are pixel values at position x in a scanline, while d is a disparity within range: 0 to $D_{max}-1$. σ^2 represents the variance associated with camera sensor noise. P_d is the probability of each camera imaging a point in the scene, while ϕ is the associated field of view. Generally the values P_d, ϕ, and σ^2 may be fixed since they model the physical properties of the imaging system and scene. Please refer to [4] for further details.

To develop a more intuitive understanding consider the two corresponding scanlines from the toy example in Figure 2.6. Every pixel in the left scanline is compared to every pixel in the right scanline using Algorithm 2 - this is a simplified version of Algorithm 1. The lengths of the longest common subsequence resulting from comparisons between $I_L(l)$ and $I_R(r)$, in Algorithm 2, are stored into a cost matrix (CM) as shown in Figure 2.7. A backward pass through the constructed cost matrix is used to determine the LCS and hence the set of point correspondences. These correspondences can be used to construct the disparity map, *disp.*, relative to the left scanline.

Returning to Algorithm 1, after cost values for all pixel locations are stored in an N by N cost matrix, CM, the backward pass initiates (see Algorithm 3). During cost computation a match matrix (MM) stores indices indicating the presence of occlusions

Algorithm 1 Forward pass algorithm for a DPML stereo correspondence formulation developed by Cox *et al.* [4]. This algorithm is a modified formulation of the LCS dynamic programming solution. Note that OC is a constant computed from camera and sensor parameters (Equation 2.14). NOC is a non-occlusion cost computed from an SSD-like function (Equation 2.13) which compares left and right input pixels. CM and MM represent cost and match matrices respectively. Note that the algorithm is written in MATLAB-like notation.

```
1:  % Initialize match and cost matrices
2:  for c = 1 to N do
3:     MM(c, 1 : c) = 2;
4:     MM(c, (c + 1) : N) = 1;
5:     CM(c : N, c) = (((c − 1) : (N − 1)) ∗ OC);
6:     CM(c, c : N) = (((c − 1) : (N − 1)) ∗ OC);
7:  end for
8:  % For each pixel in a row compute NOC
9:  for l = 2 to N do
10:    for r = l to (l − D_max) do
11:       min1 = CM(r − 1, l − 1) + NOC;
12:       min2 = CM(r − 1, l) + OC;
13:       min3 = CM(r, l − 1) + OC;
14:       CM(r, l) = min(min1, min2, min3) = cmin;
15:       if min1==cmin then
16:          MM(r, l) = 0; % No occlusion
17:       else if min2==cmin then
18:          MM(r, l) = 2; % Right occlusion
19:       else if min3==cmin then
20:          MM(r, l) = 1; % Left occlusion
21:       end if
22:    end for
23: end for
```

Algorithm 2 A simplified forward pass algorithm used to generate costs for the toy example shown in Figure 2.6. An initialization phase occurs before this algorithm begins. A backtracking phase occurs after this algorithm finishes execution. Note that the algorithm is written in MATLAB-like notation.

```
1:  % For each pixel in a row compute the length of the longest common subsequence
2:  for l = 1 to N do
3:     for r = l to N do
4:        if I_L(l) == I_R(r) then
5:           CM(r, l) = 1 + C(r − 1, l − 1); % No occlusion
6:        else
7:           CM(r, l) = max(CM(r − 1, l), CM(r, l − 1)); % Occlusion
8:        end if
9:     end for
10: end for
```

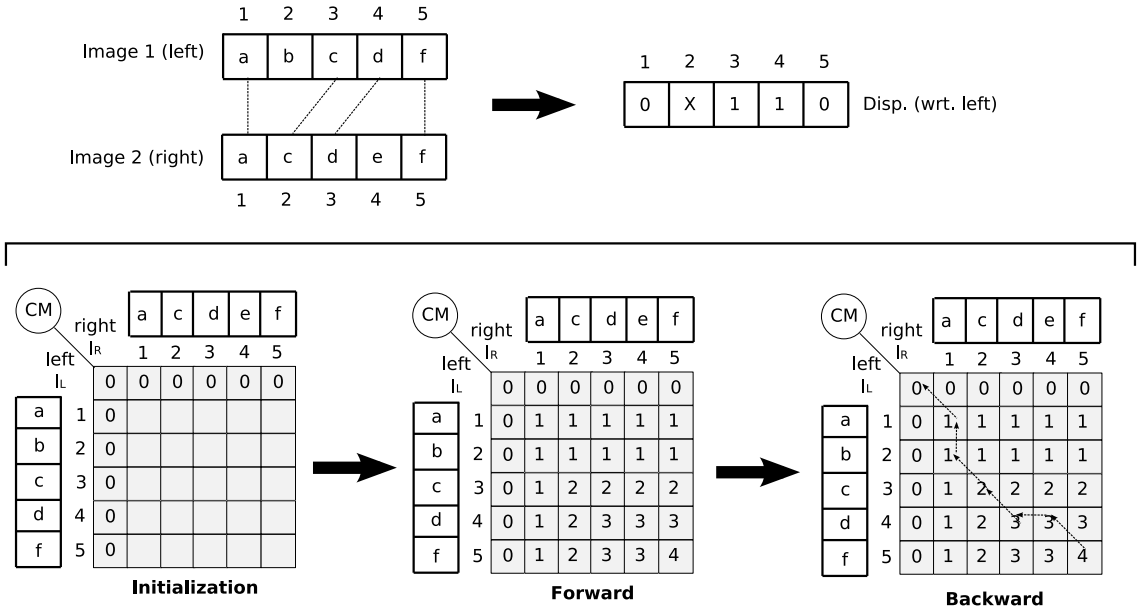


Figure 2.7: Constructing the cost matrix (CM) for an LCS solution to scanlines imaged by the toy example in Figure 2.6. Cost values are computed according to Algorithm 2 for this toy example. A globally optimal solution for the set of point correspondences between left and right scanlines is determined by backtracking through this cost matrix. These set of point correspondences can then be used to construct the disparity map (*disp.*). Note that X marks an occlusion.

occuring between the two subsequences defined by corresponding scanlines. The backward pass backtracks through this matrix computing the shortest path that minimizes the cost values. The left and right pixel locations corresponding to points along this shortest path are used to compute the disparity or to indicate occlusion. Note that N is the number of pixels in an image scanline.

Algorithm 3 Backward pass algorithm for a DPML stereo correspondence formulation developed by Cox et al.[4]. This formulation is a modification of an LCS solution. The algorithm is written in MATLAB-like notation.

```
1: p = N; q = N;
2: while p ≠ 1 and q ≠ 1 do
3:    if MM(p,q)==0 then
4:       DISP(q) = abs(p − q);
5:       OCC(q) = 0;
6:       p − −;
7:       q − −;
8:    else if MM(p,q)==1 then
9:       OCC(q) = 1;
10:      p − −;
11:   else if MM(p,q)==2 then
12:      OCC(q) = 1;
13:      q − −;
14:   end if
15: end while
```

2.4 Custom Hardware

Software solutions to 3D reconstruction problems tend to be slow. General purpose processors, while excellent tools, tend to have large overheads and limited facilities for the finer grain and customized parallelization required by image processing applications. As Chapter 4 will show typical computer systems have trouble producing depth estimates at high frame rates. Custom hardware solutions for the same software algorithms provide an alternative approach that promises substantial speedups of up to 200 fps. This improvement is achieved in two ways: by parallelizing computations to process multiple

pieces of data simultaneously and by pipelining to maximize throughput of computations over multiple pixels, scanlines or frames.

Common approaches towards the design of these custom hardware solutions take the form of Application Specific Integrated Circuits (ASICs) or Field Programmable Gate Arrays (FPGAs). ASICs utilize Very Large Scale Integration (VLSI) to fit thousands of transistor based circuits into a single integrated package (see Figure 2.8). It is convenient to think of these individual transistor based circuits as encoding some sort of underlying digital logic via switching behaviour. A key problem with ASIC hardware implementation is the long development cycle required to get from an initial design to the finished product. Likewise, specialized expertise and tools required for the design and implementation of these ASICs come with a very large price tag. Changes required at the design phase can thus also become very time consuming and costly. FPGA systems were first introduced to provide fast and cheap prototyping facilities for eventual ASIC implementations. These FPGAs are constructed as a set of interconnected reconfigurable logic blocks as shown in Figure 2.9. Modern day FPGA systems have evolved to also utilize embedded ASIC hardware in the form of Digital Signal Processing (DSP) blocks — embedded processors have also become quite popular DSP blocks. This hybridization has come hand in hand with the use of FPGAs directly in commercial products.

The major advantage of FPGA systems lies in their high reconfigurablity. This reconfigurability allows testing of a wide range of different designs quickly and cheaply. Furthermore, field deployed FPGA systems can be updated with new hardware without significant down time. This added flexibility comes at a price. To provide reconfigurability logic cells/blocks consist of Look-up-tables (LUTs), Multiplexers (MUX), Flip flops (FF) and gates. Each of these add logic delays that are not typically present in ASICs. Additionally, routing between these logic cells is provided by an interconnect bus and routing logic. This means that routing delays tend to be larger than equivalent ASIC implementations also. ASICs can thus be clocked at significantly higher speeds than their

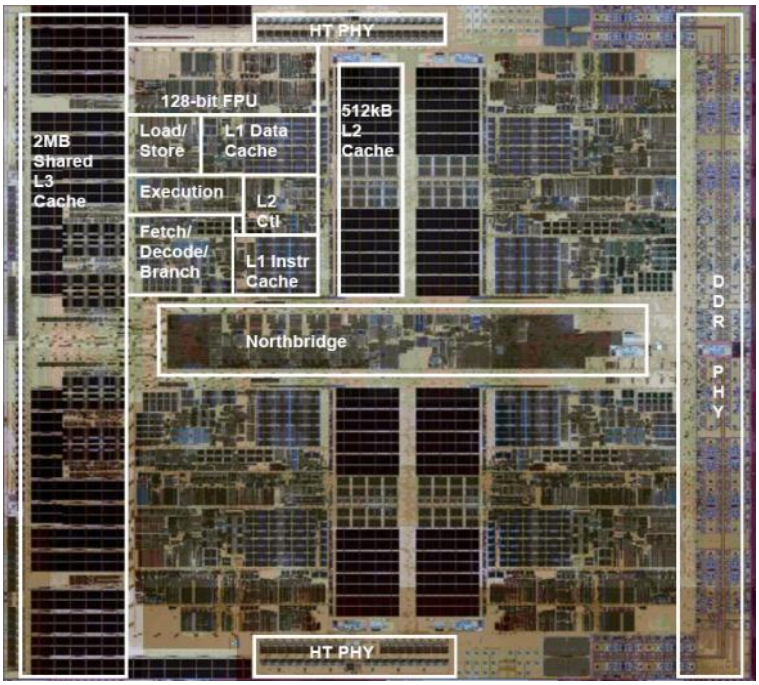
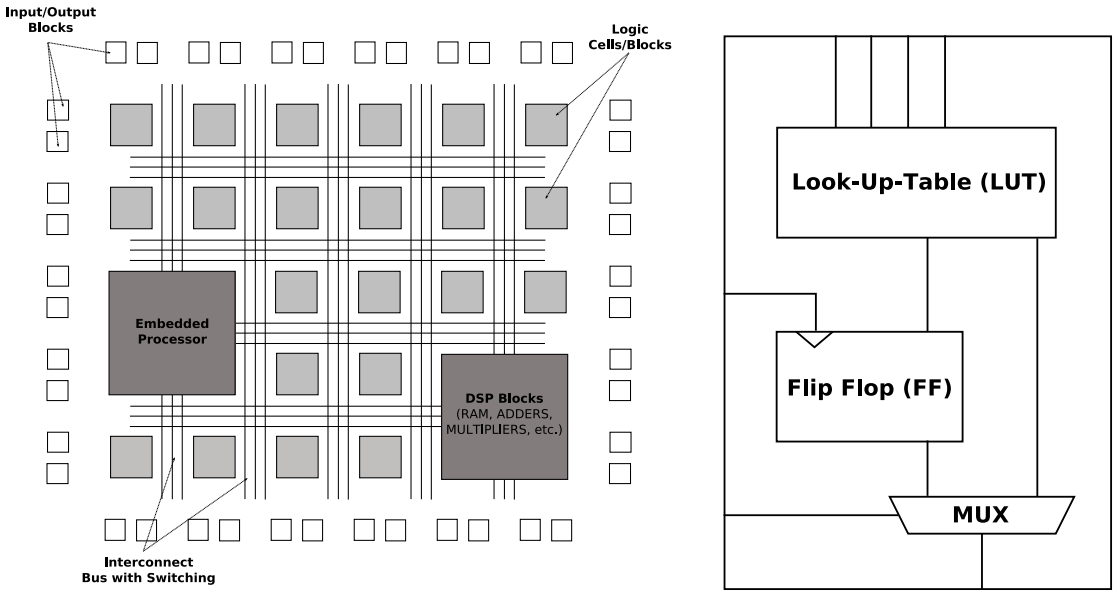


Figure 2.8: A die for an ASIC implementation of the AMD Barcelona Microprocessor. Image courtesy of the AMD Corporation website (http://www.amd.com).



(a) FPGA Architecture (b) Logic Cell Architecture

Figure 2.9: A highly simplified example of (a) an FPGA architecture and (b) its internal logic cell architecture.

FPGA counterparts. Furthermore, the additional logic means that FPGA implementations have higher power consumption and larger surface areas than their ASIC counter parts. Recent FPGA designs try to balance reconfigurability with high speed logic by implementing DSP blocks.

This thesis develops a hardware solution, DPMLHW, for the DPML stereo correspondence algorithm discussed in Algorithms 1 and 3. While this solution utilizes an FPGA system for implementation, it is clearly possible to migrate to ASICs for improved performance. A discussion of this topic will follow in Chapter 4.

2.5 Literature Review

2.5.1 General Stereo Correspondence

The problem of depth estimation has been extensively studied by the computer vision community. Detailed reviews of existing algorithms for stereo correspondence have thus also been published by a number of researchers. Gong *et al.* [7] provide an overview of general stereo correspondence approaches. These approaches can take the form of sparse (feature based), dense, volumetric or level-set algorithms. Furthermore, while this chapter has implied two view stereo, some methods for stereo correspondence generate depth estimates using multiple views. Seitz *et al.* [25] provide a review of these multi-view reconstruction methods. Earlier it was noted that rectification can simplify depth estimation. Image registration techniques can be applied to determine the transformation matrices normally estimated by rectification during calibration. These techniques also utilize sparse, feature based estimation of correspondences and are discussed by Zitova *et al.* in [30].

DPML is a global dense disparity estimation algorithm — Scharstein *et al.* [22] and Brown *et al.* [2] provide extensive discussions and comparisons of similar dynamic programming approaches to other dense stereo correspondence algorithms. These compar-

isons demonstrate that dynamic programming provides accurate estimates that compete well even with the best of existing methods. It is seen, however, that graph cuts and belief propogation algorithms provide significantly better performance. These solutions, sadly, are not suitable for high speed disparity estimation due to their computational complexity — this complexity results in computation times in the seconds range for a single image frame (see [27]). Closely related to the studies by Scharstein and Brown, Lu *et al.* [16] provide a further survey on cost aggregation and make note that dynamic programming approaches, being global methods, are not adequate for real time performance and should be substituted by local cost optimization. This thesis demonstrates that hardware implementations of dynamic programming based stereo correspondence can achieve real time performance. The next section reviews current hardware approaches to dense disparity estimation.

2.5.2 Hardware Based Stereo Correspondence

In section 2.4 it was noted that hardware solutions to the stereo correspondence problem show the potential for improved frame rate performance. This thesis demonstrates that significant speedups can be achieved resulting in disparity estimates generated at up to 200 fps. It is prudent to examine existing hardware implementations for 3D reconstruction so as to provide context.

By far the most common approaches towards hardware depth estimation are based on correlation or area based methods. Of these, the most common methods make use of SAD aggregated cost functions applied to pixel intensities. As shown by Scharstein *et al.* in [22] SAD correlation approaches do not provide very accurate disparity estimates. Typically these SAD implementations make use of line buffering (via First In First Out (FIFO) buffers) to align pixels in left and right images of a stereo pair for parallel windowing computations. Works by Miyajima *et al.*[20], Hariyama *et al.* [9], Perri *et al.* [21], Mitéran *et al.* [19] and Han *et al.* [8] all utilize this buffering. Miyajima *et al.* [20] introduces a

method which provides occlusion detection by computing costs using first one and then the other viewpoint as a reference. This method implements what is known as the left-right consistency constraint. The costs associated with windows are fed into a comparator tree to extract the match with minimum cost. Both Perri *et al.* [21] and Mitéran *et al.* [19] observe that neighbouring windows of SAD computations use many of the same values. These values are stored and re-used as required. Hariyama *et al.* [9] defines two levels of parallelism, one at the pixel level for absolute difference computations within a window and another at the window level for comparisons across windows. Progressively windows are divided into smaller regions to perform coarse to fine refinement of disparities within localized regions. Both Hariyama [9] and Simhadri *et al.* [26] utilize adder trees to perform cost and comparison computations. The presented solutions appear to scale proportionally with the disparity range with some solutions such as Hariyama *et al.* [9] generating very large non-linear increases in hardware resources. It is worth noting that adder trees are primarily useful in situations such as windowing — they do not typically apply to dynamic programming type solutions. However, coupling the DPML solution, presented earlier in this chapter, with area based correlation and introducing adder trees can have the added result of improving performance for noisy images (see Chapter 4). Most of these implementations produce results at approximately 30 fps. Hariyama *et al.* [9] reports far better results, but resource utilization is so high that implementations need to occur across multiple FPGAs. Han *et al.* [8] also reports better results of 60 fps. Some proposed methods (*i.e.* Horst *et al.* [28], Jia *et al.* [14]) fail to make details of their design clear.

Additional stereo correspondence methods use phase based correlation. Díaz *et al.* [6], Darabiha *et al.* [5] and Masrani *et al.* [18] provide examples of these approaches implemented in hardware. These methods use gabor filters to model retinal cell responses. Like SAD approaches line buffers are used to facilitate parallel computations. Phase based methods have the advantage of producing significantly better results than SAD

algorithms obtaining results that compete well with dynamic programming solutions. The problem with these methods lies in the square root computations required — these computations are difficult to implement in hardware and come with an associated high resource cost, especially when implemented in parallel. Both Darabiha *et al.* [5] and Masrani *et al.* [18] provide an implementation that uses multi-scale Locally Weighted Phase Correlation (LWPC). Gaussian pyramids provide a method for coarse to fine refinement that cancels out erroneous matches. Further left-right consistency checks are used for refining accuracy of disparity estimation results. Like SAD based implementations these phase based approaches achieve approximate 30 fps performance. Díaz *et al.* [6] demonstrate an algorithm that achieves over 200 fps performance but does so by reducing the search range to only four neighbouring pixels.

A final approach that produces very high frame rates (over 200 fps), on par with the approach presented in this thesis, makes use of the census algorithm (see Woodfill *et al.* [29]). The census algorithm computes costs within a pixel neighbourhood using the census tranform — the transform essentially consists of bit-wise OR operations that transform the image space into binary vectors compared according to their hamming distance. The high frame rates and very simple computations associated with this approach come with poor accuracy.

To date no hardware implementations of dynamic programming type algorithms have been explored by the vision community. Furthermore most existing solutions produce relatively low frame rates of approximately 30 fps at resolutions that are typically lower than 640×480 pixels. They achieve higher frame rates by sacrificing on accuracy, resolution or disparity search range. This thesis explores a dynamic programming solution that achieves high frame rate performance with no compromise on accuracy and limited compromise on disparity search range or resolution. The solution provides good scaling characteristics relative to SAD and phase based correlation approaches.

2.6 Summary

A dynamic programming method, DPML, is presented as a solution for problems of dense 3D reconstruction. Dense algorithms require some sort of camera and scene modeling to constrain the search space and make computations tractable. Reflectance modeling and the pinhole camera model introduce some basic principals for understanding how to setup these constraints. These models can be extended to two view stereo imaging and the idea of perspective projection used to introduce epipolar geometry. This epipolar geometry, in-turn, generates constraints that reduces the problem of finding pixel correspondences for disparity estimation to a 1D search problem. A rectification process can provide techniques for warping images, in each view, to align epipolar lines and simplify this search. Pixel comparisons for this search, however, must still utilize effective cost or similarity metrics. SSD, SAD and NCC are examples of common cost functions utilized for determining the likelihood that a pair of pixels correspond to the same 3D world point. Dynamic programming, furthermore, provides a method for finding a minimum cost among sets of these pixel-pair costs. It is interesting to note that dynamic programming is particularly suited to hardware implementations. Implementations of DPML algorithms can utilize available hardware logic for pipelining and SIMD type parallelization. Typically this hardware is prototyped using an FPGA. However, to produce truly fast stereo correspondence circuitry, ASIC designs can be more effective. Finally, it is noted that existing hardware approaches for disparity estimation primarily utilize SAD based correlation, an approach that has relatively poor accuracy. Barring some notable exceptions, these existing hardware methods produces frame rates in the range of 30 fps. In the next chapter, a method for hardware disparity estimation that utilizes DPML and produces frame rates in excess of 200 fps is presented. It is noted that no such dynamic programming solution has been attempted in hardware before.

Chapter 3

System Design

Chapter 2 discussed the general formulation of a dense stereo correspondence problem from the perspective of geometry, search, dynamic programming and maximum likelihood estimation. Moreover, it was noted that hardware implementations can take advantage of the inherent SIMD nature of image processing thus allowing for high speed computation. The challenge of producing high frame rate depth estimates, from the DPML perspective, then, lies in the ability to reformulate the general algorithm to take into account inherent constraints imposed by hardware. These constraints necessitate the reduction of algorithm complexity with respect to computational load and memory so as to optimally utilize existing hardware resources.

In this chapter, Section 3.1 first begins with a brief presentation of a direct serial hardware implementation, DPMLHW(S), of Cox's DPML [4] algorithm shown earlier in Chapter 2. Some of the short falls of this approach are discussed and an improved approach for stereo computations proposed in Section 3.2. The section describes two modified algorithms and hardware architectures, DPMLHW(P) and DPMLHW(PP), for very high frame rate correspondence.

3.1 Serial Architecture

Algorithms 1 and 3 in Chapter 2 describe the basic DPML algorithm used for stereo correspondence. DPML is a two pass algorithm, consisting of a forward and backward phase, which operates on a pair of rectified (left and right) input image pairs. The forward phase computes local costs. These costs provide an estimate of the degree of similarity between all possible matching pixel pairs extracted from a scanline in the input images. This local cost is stored in a cost matrix of size $N \times N$, where N is the width of the scanline. During the backward phase, a minimum cost path is traced through the cost matrix, CM, (via another matrix called the match matrix, MM) such that the set of most likely pixel correspondences are selected for every pixel in the scanline. The pixel distance between these correspondences is the set of dense disparity estimates ($DISP$). Accompanying these depth estimates is a value, OCC, that indicates regions in the depth map that are occluded and have no correspondences. This procedure is repeated over all scanlines, M, in the stereo pair for a dense depth reconstruction for all pixel locations in the original stereo image.

Given that pixel intensity (I_R and I_L) is generally a $P = 8$ bit value, the associated cost (computed via Equations 2.13 and 2.14), can then be represented with $C = 16$ bits. Likewise, only $I = 2$ bits are required to represent match indices stored in the match matrix, since these vary between values: 0, 1 and 2. Since most FPGA systems have significant memory limitations, it becomes important to minimize such utilization. In particular, it is noted that for an N pixel scanline the cost matrix requires $C \times N^2$ bits of memory and the match matrix requires $I \times N^2$ bits of memory. For full size input images of width: $N = 640$ this would mean a total memory utilization of 7.37 megabits (Mb) — an amount that comes close to exceeding memory resources on some FPGA devices.

It is observed that only the match matrix values need be stored for backtracking. Furthermore, the cost computation at any particular pixel location only requires costs from the previous two rows of the cost matrix. It is, therefore, possible to reduce mem-

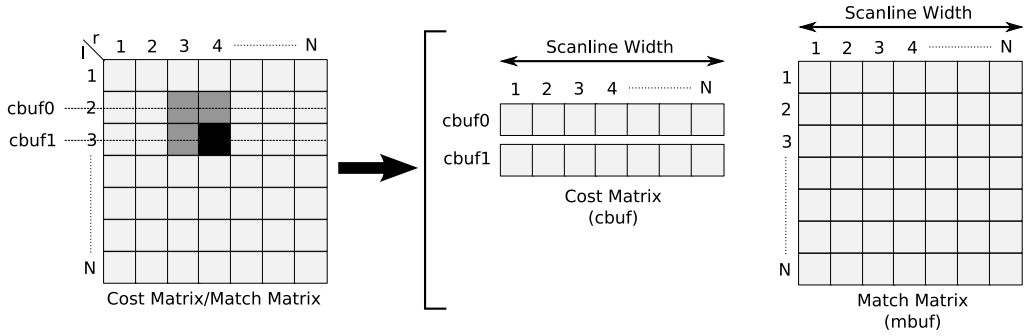


Figure 3.1: An observation of the structure of the cost matrix and associated writes in the match matrix demonstrate that it is possible to reduce memory utilization.

ory utilization. Figure 3.1 demonstrates that cost matrix memory utilization can be decreased to: $C \times 2 \times N$ bits, thus reducing total utilization to 839.68 kilobits (Kb). This reduced version of the cost matrix, *CBUF*, requires an updating process which preserves the structure of the matrix over subsequent iterations. In order to do this, *CBUF* from Figure 3.1 is updated as shown in Figure 3.2. Recall that l and r variables represent the pixel positions under consideration in the left and right scanlines respectively.

Further observing the initialization of the cost matrix in Figure 3.3 it is seen that the maximum disparity range, D_{max}, can also be taken into consideration. In such a scenario only cost values up to this range need be computed — all other cost values can be estimated by the occlusion cost at the boundary (*inita* and *initb*) that delimits regions outside the D_{max} search zone.

3.1.1 System Overview

Figure 3.4 introduces a naïve approach, DPMLHW(S), towards a hardware implementation of Cox's [4] DPML algorithm. The hardware incorporates the algorithmic changes presented earlier in this section. Since forward and backward passes must be processed separately and sequentially, the data path of the DPMLHW(S) implementation is divided into two. The lightly shaded blocks represent the forward pass data path, while

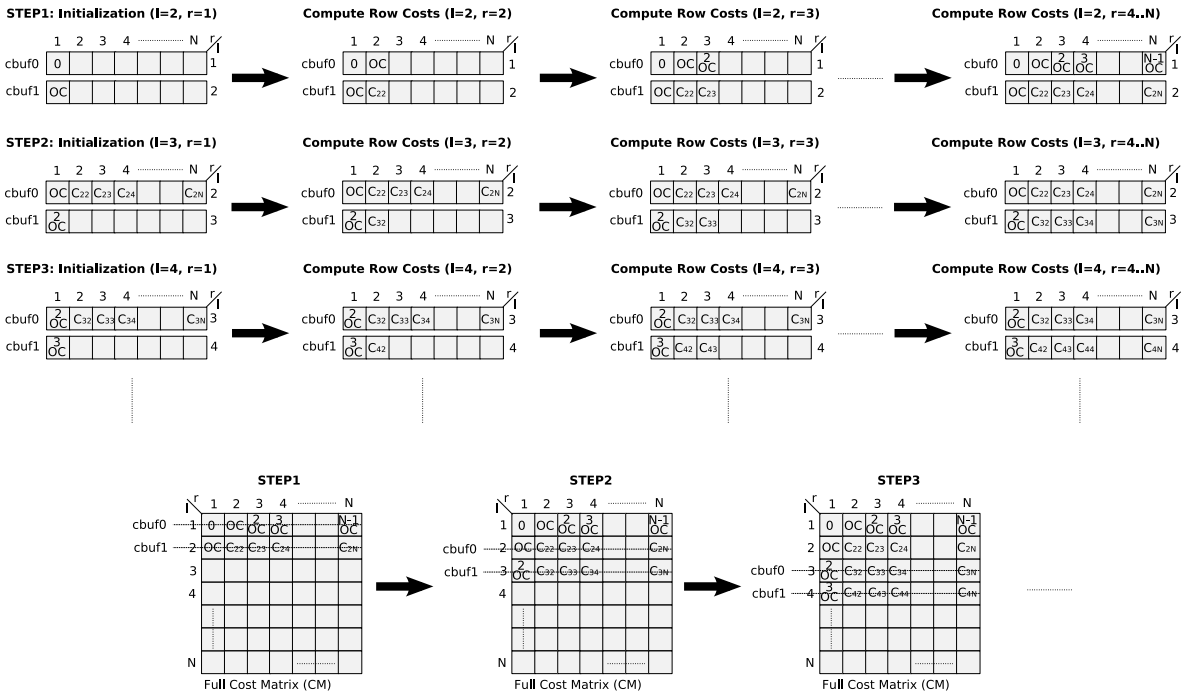


Figure 3.2: Retaining cost matrix structure in *CBUF*. An overview of step by step updates to *CBUF*. At each step an additional row of the cost matrix is computed.

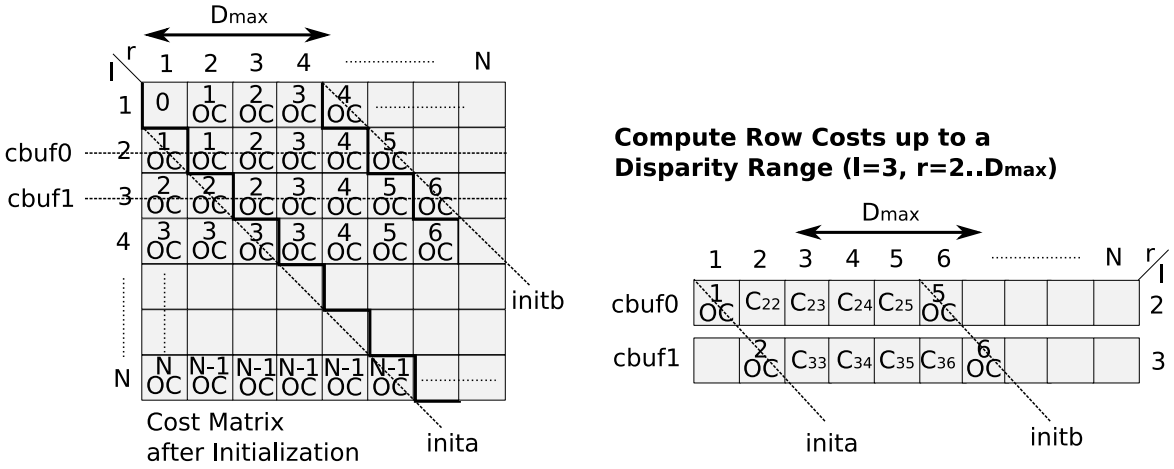


Figure 3.3: Retaining cost matrix structure in *CBUF* given a disparity range D_{max}. The left image shows the initialized cost matrix *CM*. The right image is a snapshot of *CBUF* during processing of $I_{R=2..N}$ and $I_{L=3}$.

the darker blocks refer to the backward pass data path. Indices from Algorithms 1 and 3 have been shifted to start at zero in this implementation.

Stereo image data is first stored into off-chip memory by a camera system and subsequently rectified (by *RTBUF*). During the forward pass a scanline from the rectified image data is retrieved and stored in an image pixel buffer *IBUF*. This buffer contains a set of counters that iterate through the particular scanline. Addresses generated by these counters index into *IBUF*, retrieve 8 bit pixel intensities for the the current left and right scanline and feed them into the *NOC* module. The *NOC* block then computes the non-occlusion cost as presented by Equation 2.13 and Algorithm 1. The three resulting 16 bit cost values (*min1*, *min2*, and *min3*) are fed into *MIN*, a module that consists of a set of comparators. The *MIN* module determines the minimum cost among the three inputs along with the associated index. The index is stored in the match matrix buffer, *MBUF*, and is used to determine the optimal cost path based on whether the current pixel location is occluded. The computed minimum cost, from *MIN*, along with the current pixel location, as represented by counters in *IBUF*, is used to generate address and data for a write into the *CBUF* partial cost matrix.

Once all match matrix index values have been computed for a particular scanline, the backward pass blocks in the DPMLHW(S) implementation are activated. In this phase, the *IBUF* counters are disabled and counters from *DISP* used to backtrack from the last written address of the match matrix (*MBUF*). The backtracking phase proceeds by outputting the match index every clock cycle to the *DISP* unit. This unit computes the disparity from the current internal counter values and writes it to the *DBUF* module. When the *DISP* counters reach zero values the computations for all pixels in a scanline are complete and the next scanline is retrieved for processing by *IBUF*.

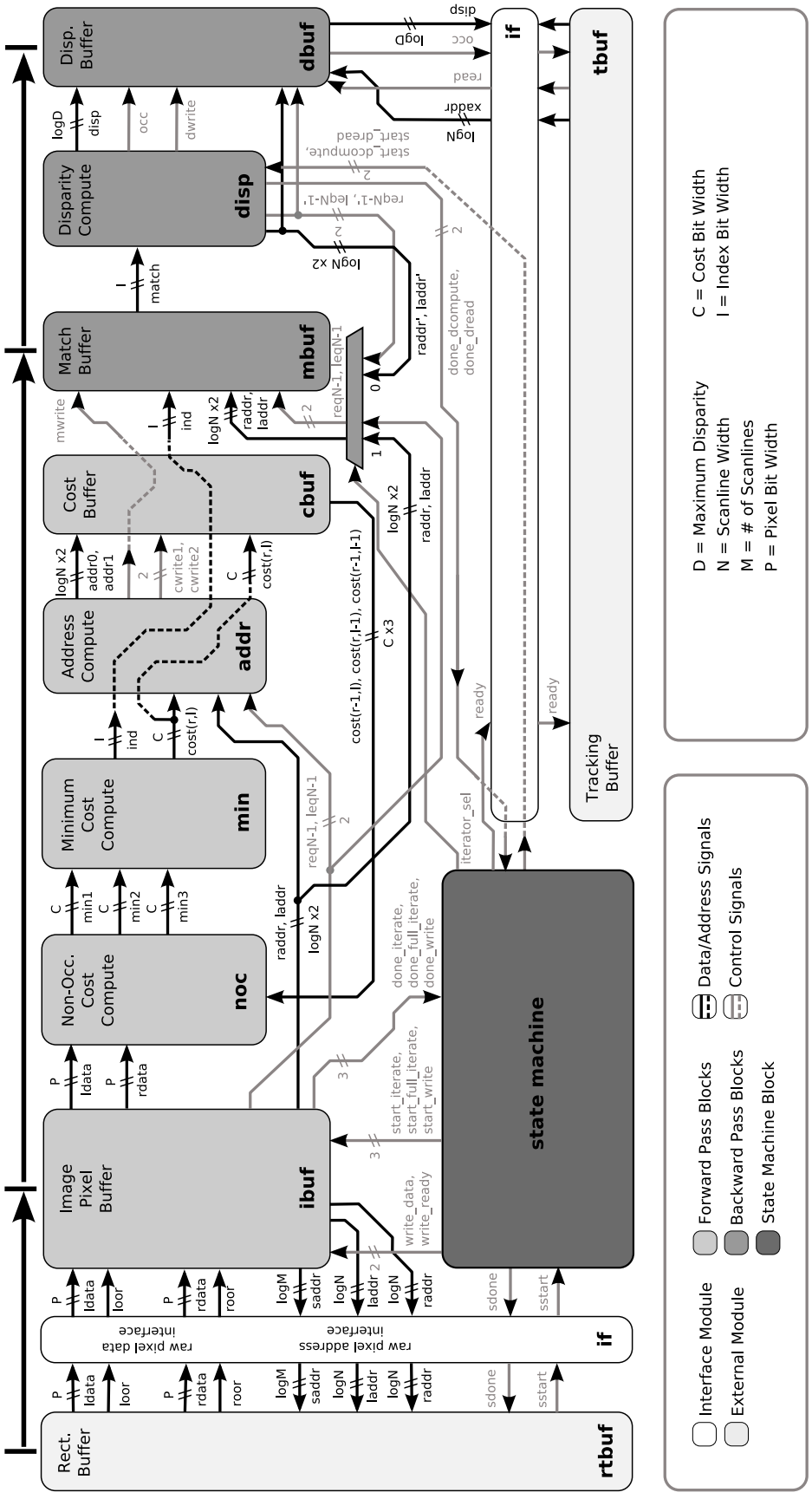


Figure 3.4: High level architecture for a naïve serial hardware implementation of the DPML algorithm originally developed by Cox *et al.*[4]. The black vertical bars, above key blocks, mark registers, with arrows indicating the series of register transfers that occur.

3.1.2 Discussion

The DPMLHW(S) hardware makes significant improvements over software implementations of the same algorithm. A specific comparison with other implementations can be found in Chapter 4. Further optimizations, however, may be made to the hardware — these optimizations are discussed below.

First, the serial implementation stores cost data associated with an entire scanline (a spatial complexity of $O(N)$). It also stores match data for the entire space of solutions (a spatial complexity of $O(N^2)$). This is somewhat redundant especially given that search only takes place over a limited disparity range, D_{max}. It would be more efficient to reduce the width of *CBUF* and *MBUF* such that only some fraction of the N memory elements are required for disparity computations. Furthermore, FPGA systems containing specialized DSP blocks for RAM can reduce circuit delays and logic cell utilization. An appropriate implementation of *CBUF* and *MBUF* can be made to utilize these DSP units for fast memory access.

Second, the current implementation runs through a long initialization phase whereby *MBUF* must be initialized completely. However, most of the initialization values are overwritten by computed match indices during the forward pass. The only initializations necessary occur at the boundaries of the matrix delimited by *inita* and *initb* in Figure 3.3. These boundary initializations prevent the backward pass from backtracking beyond the defined regions of the matrix. A saving of approximately N^2 clock cycles can be achieved with improvements to the matrix initialization process.

Third, DPMLHW(S) does not take advantage of the flexibility of custom hardware implementations. The cost computations are identical operations that can be re-framed in the SIMD context. Loops iterating through $r = l$ to $r = D_{max}$ can be unrolled and computations for the minimum costs ($min1$, $min2$, $min3$ and $cmin$) done in parallel given that that initial three cost values: $CM(r-1, l)$, $CM(r, l-1)$ and $CM(r-1, l-1)$, from Algorithms 1 and 3, are available. The current implementation runs with a time

complexity of $O(N^2)$ — this complexity could, thus, be reduced to $O(N)$ with appropriate parallelization. This speed up would come at the cost of logic resources, increasing the utilization to some order proportional to D_{max}.

Fourth, a long combinational path in DPMLHW(S) means low maximum clock frequency (F_{max}). A low clock frequency, in-turn, means that the speed at which pixels are processed will be lower and hence frame rates at which depth values are generated will also be lower. The longest path runs from the memory module *IBUF* to the memory module *CBUF*. Inserting pipelining register at key points (*i.e.* at the output of *NOC* and *MIN*) along this path breaks up this long path resulting in faster clock frequencies.

3.2 Parallel Architecture

By incorporating some of the improvements discussed in the previous section higher frame rates than the DPMLHW(S) implementation can be achieved. As seen earlier, examining the structure of the cost matrix (see Figure 3.5) demonstrates that a cost computation is dependant upon three neighbouring costs. Computing cost values, for a particular pixel pair, in a naïve manner (*i.e.* row by row in the cost matrix) would require the sequential computation of costs for all previous contiguous elements in that particular row. This serial computation can be avoided by computing cost values along anti-diagonals instead [17].

Computations along a particular anti-diagonal are only dependant upon data from adjacent elements in the two previous anti-diagonals. Thus the sequential delay normally associated with the cost computations along a row in the cost matrix does not occur for anti-diagonals. This suggests that as long as the previous two anti-diagonals are available it is possible to compute all cost values along the current anti-diagonal simultaneously (*i.e.* in parallel).

To facilitate parallel computation along the anti-diagonals, pixels from the left and

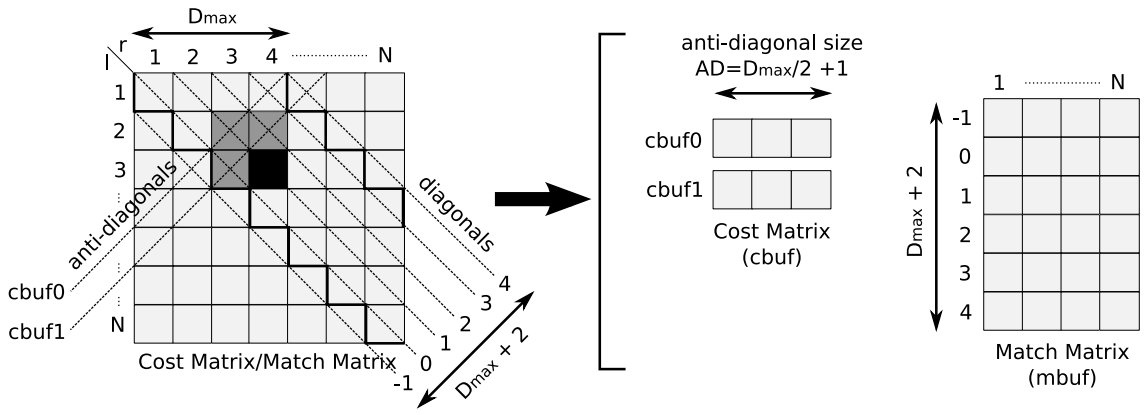


Figure 3.5: Careful observation of the structure of the cost matrix demonstrates that it is possible to parallelize cost computation. Three previous costs (dark grey blocks) located in the two previous anti-diagonals are used to compute the current cost (black block).

right scanline of an image pair are fed serially into two AD element buffers, $LBUF$ and $RBUF$. Pushing data into these buffers causes pixel values to mesh in a manner consistent with the particular anti-diagonal in the cost matrix. As shown by Figure 3.6 the width of $LBUF$ and $RBUF$ is related to the disparity by: $AD = D_{max}/2 + 1$, where the $+1$ refers to the one additional element at the *inita* or *initb* boundaries introduced in Figure 3.3. Odd and even anti-diagonals have boundaries on either the left (*inita*) or the right (*initb*) sides of the $LBUF$ and $RBUF$ buffers respectively.

The dimensions of the partial cost matrix ($CM \rightarrow CBUF$) can be reduced to $2 \times AD$ elements and those of the partial match matrix ($MM \rightarrow MBUF$) reduced to $(D_{max} + 2) \times N$ elements — the dimensions of both matrices are thus proportional to D_{max}.

Since cost computations occur in parallel, their corresponding match indices are also generated in parallel. This requires that the match matrix be updated such that all indices for a given anti-diagonal are written at the same time. Figure 3.7 demonstrates the writing process. An entire anti-diagonal is written to the matrix in a single iteration. For example, value E_5 indicates writes associated with even anti-diagonal number five, while value O_6 indicates writes associated with the sixth odd anti-diagonal. For even anti-diagonals, index values are written to row $d = 2i$ and column $p = r - i$, where

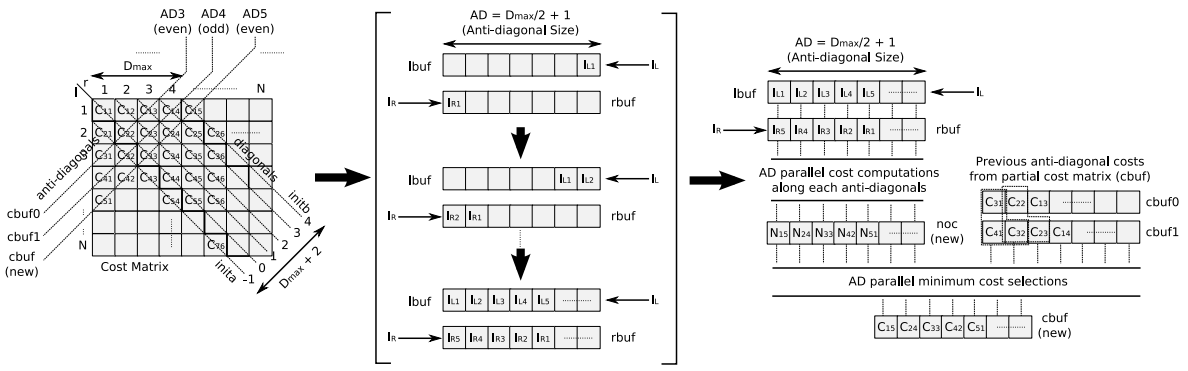


Figure 3.6: Parallelization requires buffers that recieve a serial pixel input stream. The *LBUF* and *RBUF* align these pixels such that they line up as required for cost computations. This figure provides an example for the computation of cost values for anti-diagonal 5. The *CBUF(new)* data generated is pushed into *CBUF1* and data from *CBUF1* pushed into *CBUF0*.

i is the the position of the index along the current anti-diagonal. Similarly, for odd anti-diagonals, index values are written to row $d = 2i - 1$ and column $p = r - i$ of the match matrix (*MBUF*). The original match matrix, *MM*, from Algorithm 1 and 3, required that all locations be initialized. *MBUF* only requires initialization of values at the $p = 1$ and $p = 2$ boundary. Other boundary regions at $d = 1$ and $p = N$ are initialized during regular anti-diagonal cost computations or are never visited during the backtracking phase.

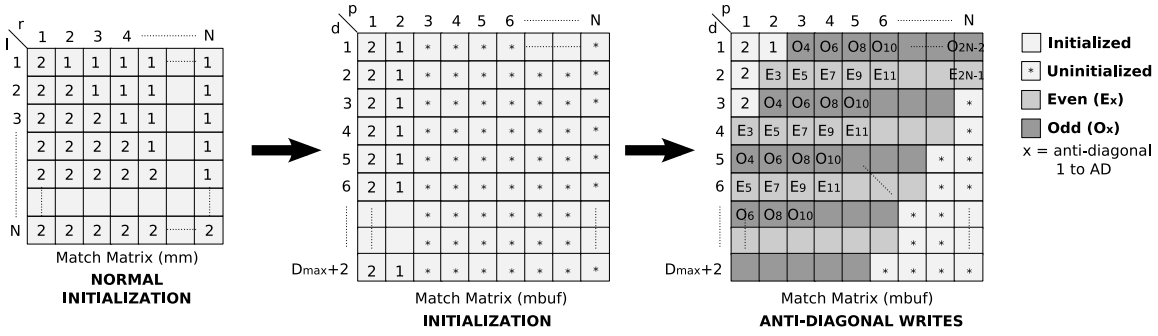


Figure 3.7: Initialization structure and parallelization of match matrix writes.

Based on these observations a modified DPML algorithm can be developed. This algorithm is divided into three phases: an initialization phase, a forward pass phase and

a backward pass phase.

Algorithm 4 presents the initialization phase of the modified DPML algorithm. This phase fills *LBUF* and *RBUF* such that they are aligned to compute the third anti-diagonal of the cost matrix. Furthermore it ensures that *CBUF* is setup with cost values for the first two anti-diagonals and that *MBUF* is initialized to avoid backtracking outside the bounds of the match matrix. *VBUF* ensures that boundary regions at the limits of the disparity range $d \in [0, D_{max} - 1]$ of *LBUF* and *RBUF* are marked as invalid and appropriately updated with the boundary costs.

Algorithms 5, 6 and 7 present the forward phase of the modified DPML algorithm. Continuing after initialization, left and right pixels are alternately pushed into the *LBUF* or *RBUF* based on whether the current anti-diagonal is odd or even – in odd anti-diagonals left pixels are pushed into *LBUF* and vice versa on even anti-diagonals. Cost values are computed differently at distinct positions along an anti-diagonal. Values that hit the boundaries *inita* or *initb* from Figure 3.6 are initialized to some multiple of the occlusion cost *OC*. *Valid* and *invalid* indices further determine how cost values are computed at other locations. It is important to remember that while cost computations vary along an anti-diagonal, all positions are still computed in parallel. Match values at the boundaries are generated to ensure that backtracking does not move beyond the dimensions of the *MBUF* matrix. The forward pass completes, with $back = 1$, once all $AD = 2N - 1$ anti-diagonals and their associated costs and match indices have been computed.

Algorithm 8 presents the backward phase of the modified DPML algorithm. Like the original implementation by Cox *et al.* [4], the backward pass algorithm remains much the same. However, the variation in the match matrix structure presented in Figures 3.5 and 3.7 necessitates an alternate scheme for decrementing and incrementing the d and p counters.

Algorithm 4 Initialization phase for a parallelized DPML stereo correspondence formulation. AD is the number of anti-diagonals, D_{max} is the maximum disparity range and N is the number of pixels in a scanline. OC is a constant cost computed based on camera and sensor parameters. Note that this algorithm operates on a single scanline. Algorithm is written in MATLAB-like syntax.

1: *% Create left, right, valid buffers and initialize counters*
2: $AD = D_{max}/2 + 1$;
3: $even = 1;\ back = 0$;
4: **new** $LBUF[AD],\ RBUF[AD],\ VBUF[AD]$;
5: **new** $CBUF[AD][2],\ MBUF[D_{max} + 2][N]$;
6:
7: *% Push first pixel into buffers*
8: $l = 1; r = 1$;
9: **qPushLeft**$(LBUF \leftarrow I_L(l));\ l++$;
10: **qPushRight**$(I_R(r) \rightarrow RBUF);\ r++$;
11: **qPushRight**$(0 \rightarrow VBUF)$;
12:
13: *% First initialization of MBUF and CBUF*
14: $MBUF[1...(D_{max} + 2)][1] = [2, 2, ..., 2]$
15: **qPushLeft**$(CBUF[1...AD] \leftarrow [0, 0, ..., 0])$;
16:
17: *% Fill up LBUF*
18: **while** $l \leq AD$ **do**
19: **qPushLeft**$(LBUF \leftarrow I_L(l));\ l++$;
20: **qPushRight**$(0 \rightarrow VBUF)$;
21: **end while**
22:
23: *% Push final intialization pixels into buffers*
24: **qPushLeft**$(LBUF \leftarrow I_L(l));\ l++$;
25: **qPushRight**$(I_R(r) \rightarrow RBUF);\ r++$;
26: **qPushRight**$(1 \rightarrow VBUF)$;
27:
28: *% Initialize match and cost buffers*
29: $MBUF[1...(D_{max} + 2)][N] = [1, 1, ..., 1]$;
30: **qPushLeft**$(CBUF[1...AD] \leftarrow [OC, OC, ..., OC])$;

Algorithm 5 Forward pass algorithm for the parallelized DPML stereo correspondence formulation. Algorithm is written in MATLAB-like syntax.

while *back* == 0 **do**
  if *even* == 1 **then**
    compute_even *% see Algorithm 6*
  else
    compute_odd *% see Algorithm 7*
  end if
end while

Algorithm 6 Forward pass algorithm for computing even anti-diagonal costs for the parallelized DPML stereo correspondence formulation. AD is the number of anti-diagonals, D_{max} is the maximum disparity range and N is the number of pixels in a scanline. OC is a constant cost computed based on camera and sensor parameters. Note that this algorithm operates on a single scanline. Algorithm is written in MATLAB-like syntax.

if *even* == 1 **then**
  % Anti-diagonal boundary/invalid region cost computations
  $invalid = \textbf{find}(VBUF[1...AD] == 0)$ *% Generate set of indices for invalid regions*
  $C[AD] = CBUF[AD][1] + OC;$
  $MBUF[2 \times AD][r - AD] = 2;$
  $C[invalid] = CBUF[invalid][2] + OC;$
  $MBUF[2 \times invalid][r - invalid] = 2;$

  % Anti-diagonal valid region cost computations
  $valid = \textbf{find}(VBUF[1...AD] == 1) - 1$ *% Generate set of indices for valid regions*
  $min1[valid] = CBUF[valid][0] + NOC = costA;$
  $min2[valid] = CBUF[valid][1] + OCC = costC;$
  $min3[valid] = CBUF[valid + 1][1] + OCC = costD;$
  $C[valid] = \textbf{min}(min1[valid],\ min2[valid],\ min3[valid]);$
  $MBUF[2 \times valid][r - valid] = \textbf{min\_index}(min1[valid],\ min2[valid],\ min3[valid]);$

  % Shift computed cost values int CBUF
  $\textbf{qPushLeft}(CBUF[1..AD] \leftarrow C[1...AD]);$

  if $r > N$ **then**
    $d = 2;\ p = N;\ back = 1;$
  else
    $\textbf{qPushRight}(I_R(r) \rightarrow RBUF);\ r++;$
    if $l \leq N$ **then**
      $\textbf{qPushRight}(1 \rightarrow VBUF);$
    end if
    $even = 0;$
  end if
end if

Algorithm 7 Forward pass algorithm for computing odd anti-diagonal costs for the parallelized DPML stereo correspondence formulation. AD is the number of anti-diagonals, D_{max} is the maximum disparity range and N is the number of pixels in a scanline. OC is a constant cost computed based on camera and sensor parameters. Note that this algorithm operates on a single scanline. Algorithm is written in MATLAB-like syntax.

1: **if** $even = 0$ **then**
2: *% Anti-diagonal boundary/invalid region cost computations*
3: $invalid = \textbf{find}(VBUF[1...AD] == 0)$ *% Generate set of indices for invalid regions*
4: $C[1] = CBUF[1][1] + OC$;
5: $MBUF[2 \times 1 - 1][r - 1] = 1$;
6: $C[invalid] = CBUF[invalid - 1][2] + OC$;
7: $MBUF[2 \times invalid - 1][r - invalid] = 2$;
8:
9: *% Anti-diagonal valid region cost computations*
10: $valid = \textbf{find}(VBUF[1...AD] == 1)$ *% Generate set of indices for valid regions*
11: $min1[valid] = CBUF[valid][0] + NOC = costA$;
12: $min2[valid] = CBUF[valid - 1][1] + OCC = costB$;
13: $min3[valid] = CBUF[valid][1] + OCC = costC$;
14: $C[valid] = \textbf{min}(min1[valid],\ min2[valid],\ min3[valid])$;
15: $MBUF[2{\times}valid{-}1][r{-}valid] = \textbf{min\_index}(min1[valid],\ min2[valid],\ min3[valid])$;
16:
17: *% Shift computed cost values int CBUF*
18: $\textbf{qPushLeft}(CBUF[1..AD] \leftarrow C[1...AD])$;
19:
20: **if** $l > N$ **then**
21: $\textbf{qPushLeft}(LBUF \leftarrow 0)$;
22: $\textbf{qPushLeft}(VBUF \leftarrow 0)$;
23: **else**
24: $\textbf{qPushLeft}(LBUF \leftarrow I_L(l));\ l++$;
25: **end if**
26: $even = 1$;
27: **end if**

Algorithm 8 Backward pass algorithm for the parallelized DPML stereo correspondence formulation.

```
1: p = N; d = 2;
2: while p > 0 and d > 0 do
3:    if MBUF[d][p] == 2 then
4:       if d == 0 then
5:          DISP[p] = 0; OCC[p] = 1; d − −;
6:       else
7:          DISP[p] = 0; OCC[p] = 1; d + +; p − −;
8:       end if
9:    else if MBUF[d][p] == 1 then
10:      DISP[p] = 0; OCC[p] = 1; d + +; p − −;
11:   else if MBUF[d][p] == 0 then
12:      DISP[p] = d − 2; OCC[q] = 0; p − −;
13:   end if
14: end while
```

3.2.1 System Overview

Figure 3.8 introduces a highly parallelized hardware implementation of Cox's [4] DPML algorithm. The hardware incorporates the algorithmic changes presented earlier in Algorithms 4–8. Two implementations of this hardware are considered: A partially pipelined implementation DPMLHW(PP) and a fully pipelined implementation DPMLHW(P). The distinction between the two lies in the presence of two additional components: *PBUF* and *BMUX* which are used for finer grain pipelining. For clarity only DPMLHW(P) will be discussed in detail. Like the serial implementation (DPMLHW(S)), DPMLHW(P) consists of two separate data paths. Note that the lightly shaded blocks represent the forward pass modules and the darker blocks represent backward pass modules. Furthermore note that for the sake of convenience the hardware shifts indices such that they start at zero rather than one when compared to the parallelized algorithm presented earlier in this section.

The modules labeled with the suffix *BUF* form the pipeline registers along which data is transferred on consecutive cycles. The first of these registers, *RTBUF* provides an interface to an external rectification module. Pixel data arriving from this module

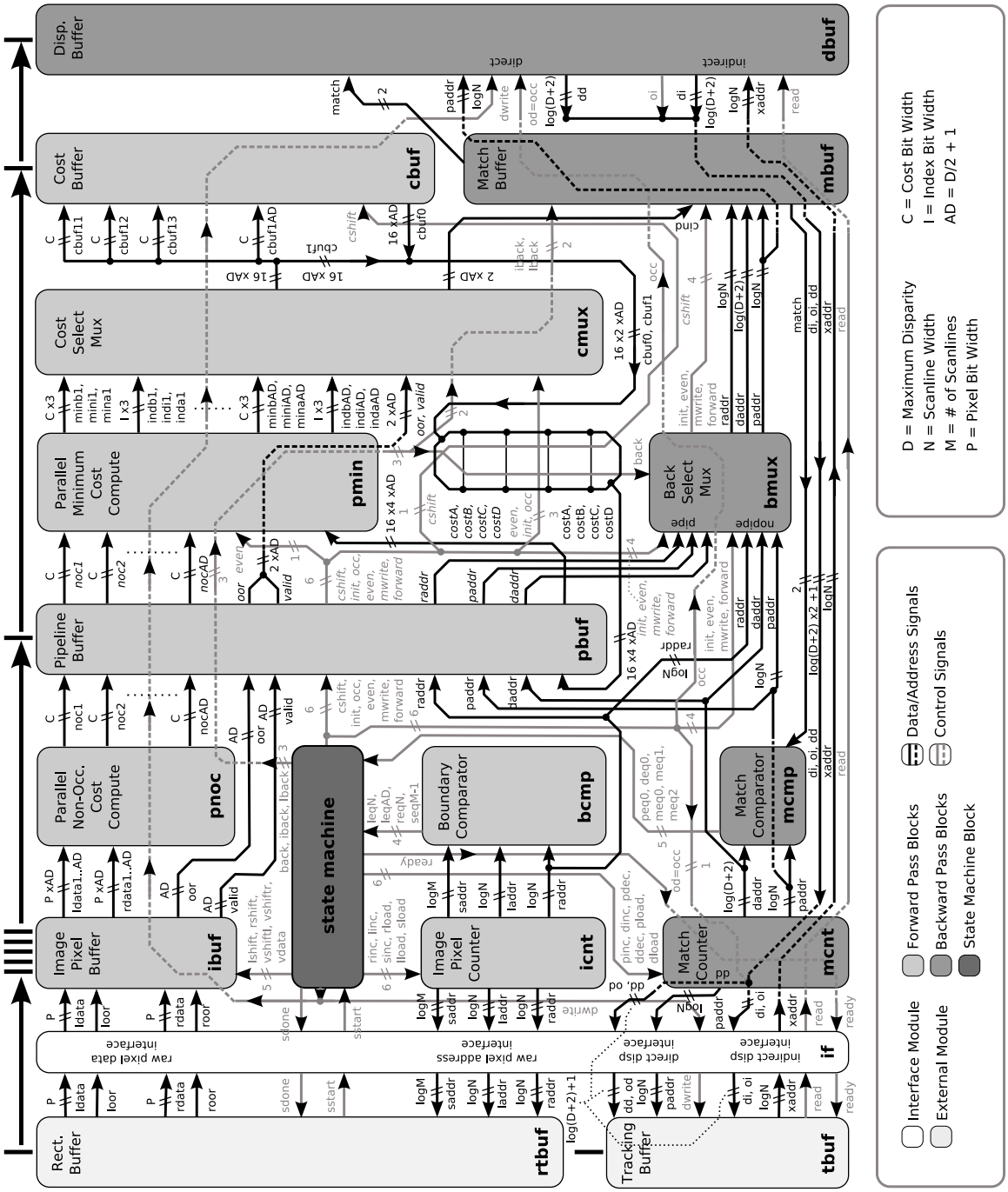


Figure 3.8: High level architecture for a highly parallelized hardware implementation of the DPML algorithm originally developed by Cox *et al.*[4]. The vertical black bars, above key modules, represent registers with arrows pointing in the direction of register transfer operations.

is buffered into *IBUF* which aligns the input for parallel cost computations along an anti-diagonal. The aligned data enters the *PNOC* unit which, in turn, computes a set of parallel non-occlusion costs associated with the particular left and right *IBUF* pixel outputs. Parallel *PNOC* costs pass through a pipeline buffer, *PBUF*, and onto the *PMIN* module which computes the minimum cost and associated minimum match index for each position in the parallelized anti-diagonal. Depending on the position along the anti-diagonal, either a boundary cost or an actual cost must be chosen. The *CMUX* performs this selection based on the current anti-diagonal position under consideration. From *CMUX*, cost values and match index values are sent to *CBUF* and *MBUF*, respectively, for storage. A combined *CBUF* output is also sent back to the *PBUF* pipeline register for subsequent cost computations. Note that indexing counters *ICNT* are used to access raw pixel data from a rectification module, *RTBUF*, and to write to the partial match matrix, *MBUF*. *BCMP* performs comparisons on *ICNT* values to determine when a scanline has been completely or partially processed.

Once all anti-diagonals for a particular scanline have been processed the backward pass initiates. The *MCNT* counter unit generates addresses for *MBUF* by using *MCMP* to compare the current match value to preset constants: 0, 1 and 2. Results of the comparison yield an indication of whether to decrement or increment internal counters. Disparity and occlusion results also become available for the current match matrix (*MBUF*) address. These disparity values are stored into *DBUF* for later retrieval by an external tracking module *TBUF*. Note that the backtracking multiplexer, *BMUX*, multiplexes values to address *MBUF* either from the forward pass pipeline or from the backward pass counters.

3.2.2 Interfaces: Rectification and Tracking

The rectification (*RTBUF*) and tracking (*TBUF*) modules provide interfaces to important external pre- and post-processing logic circuits. Rectification attempts to transform

a pair of images such that search during stereo correspondence can take place along a single scanline. However, this transformation can result in rectified images that contain regions with no valid pixel data. The interface for *RTBUF* thus has an out of range signal, *oor*, accompanying the data signals. After some initial latency, the rectification module buffers enough pixels in a scanline and sends a *start* signal to initiate stereo correspondence. The DPMLHW(P) hardware uses internal counters to address the particular pixel locations in the left and right scanline stored in *RTBUF*.

Stereo correspondence generates depth maps that can then be used for tracking of some 3D target. The *DBUF* module in the DPMLHW(P) implementation provides an interface similar to *RTBUF* for tracking tasks. The tracking buffer, *TBUF*, may request data explicitly from the stereo correspondence hardware after a scanline has been processed (*i.e.* by requesting the previous scanline while the current scanline is being processed in the forward pass). Alternatively, the *TBUF* module can snoop during the backtracking phase to retrieve data as it is being generated by the match matrix (*MBUF*).

Figure 3.9 provides a high level block diagram of how the current DPMLHW(P) stereo correspondence hardware may be integrated into a larger hardware system. Please refer to work by Kirischian [15], Islam [12] and Belshaw [1] for further details on hardware image acquisition, rectification and tracking. Note that pre-filtering circuits can be added to reduce the effect of noise in images acquired by a camera system.

3.2.3 Parallelizing Buffers

A set of five modified First-In-First-Out (*FIFO*) buffers provide a method for parallelizing sequentially inputted pixel intensities entering the image pixel buffer, *IBUF*. Figure 3.10a and 3.10b presents a schematic diagram of the *IBUF* and its internal *FIFO* buffers, respectively. The *LBUF* and *RBUF* components are directly analogous to the buffers discussed earlier in the parallelized algorithm presented in this section. Associated with

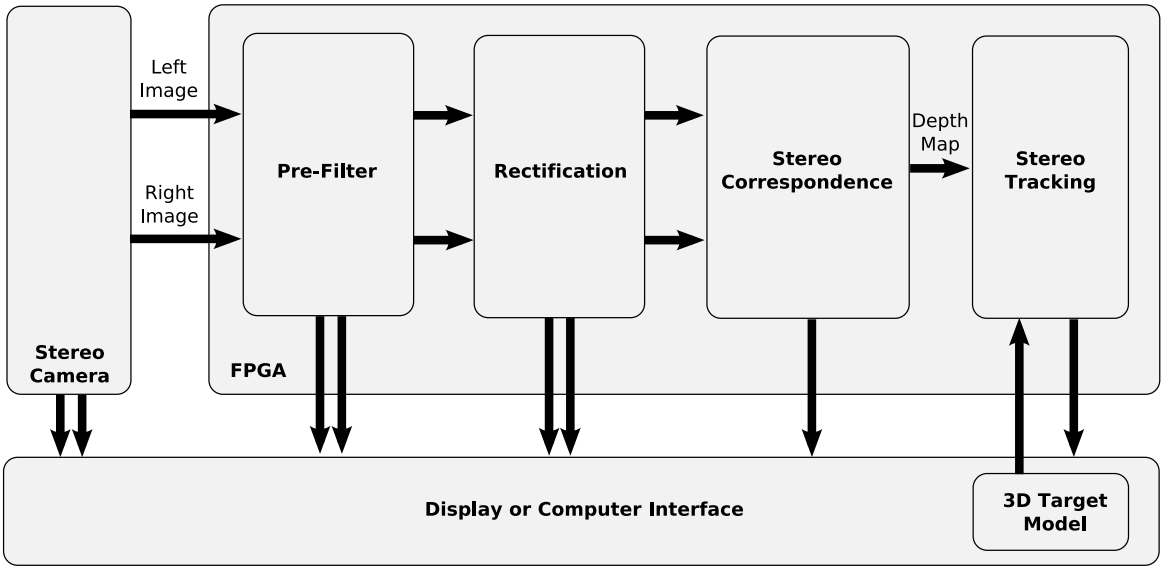
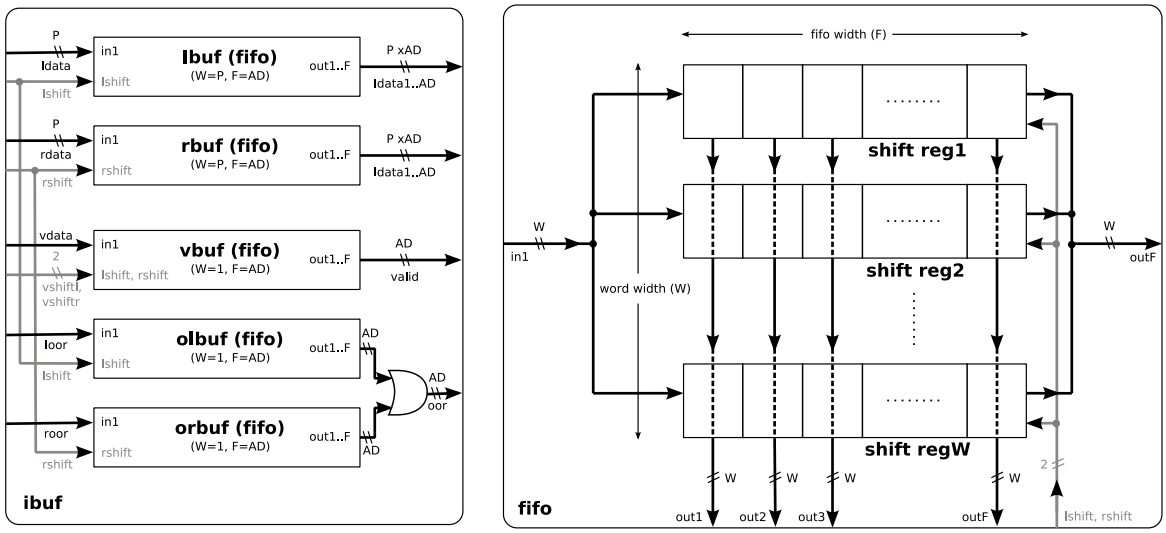


Figure 3.9: Integrating stereo correspondence into a larger system — a high level block diagram.

these buffers are three others: *VBUF*, *OLBUF* and *ORBUF*. These additional buffers are used to keep track of boundary regions or other invalid regions in *LBUF* and *RTBUF*. The buffers output data to a multiplexer, *CMUX*, for each position in the anti-diagonal and select which of three boundary, invalid or actual cost/index values to select for storage into *CBUF*. These values are analogous to signals: *minb*, *mini*, *mina*, *indb*, *indi* and *inda* entering *CMUX*)

3.2.4 Memory Addressing

The *ICNT* and *MCNT* counters perform critical addressing functions. Figures 3.11a 3.11b present the schematic diagrams for these counter blocks, respectively. The outputs of the *ICNT* unit are equivalent to variables r and l from the parallelized DPML algorithm discussed earlier in Algorithms 4–8. These variables determine how data is pushed into the internal right and left buffers of *IBUF*. Furthermore, *ICNT* address signals also provide addressing for the rectification buffer (which feeds into *IBUF*) and addressing for writes to the partial match matrix *MBUF*. Likewise, signals generated by *MCNT* corre-



(a) Image Pixel Buffer (*IBUF*) (b) First In First Out Buffer (*FIFO*)

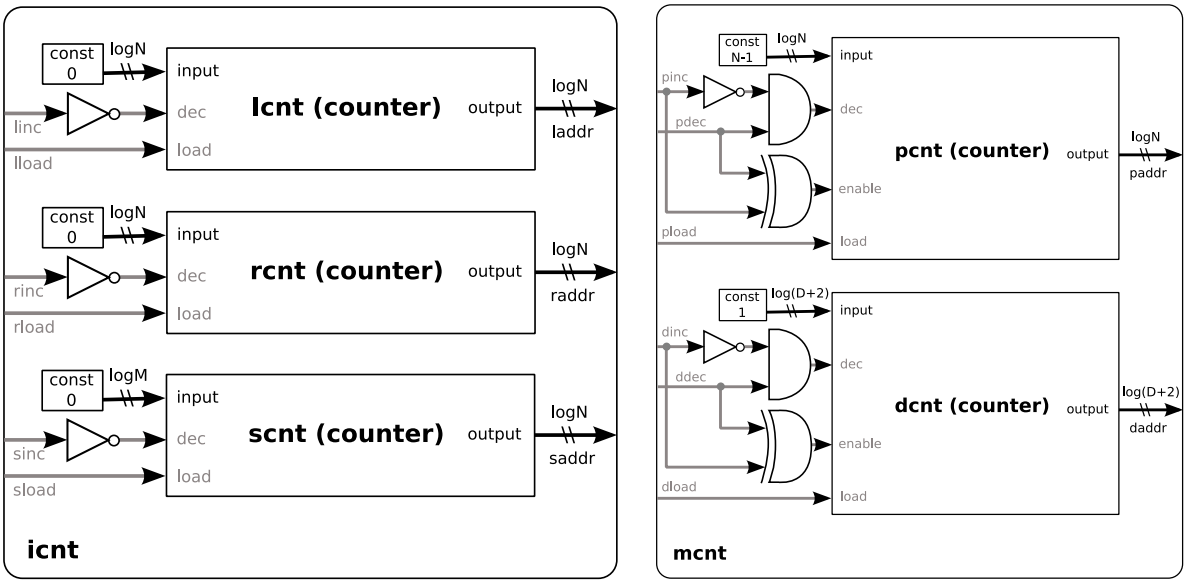
Figure 3.10: The schematic diagram for the image pixel buffer. (a) *IBUF* consists of a set of FIFO buffers that mesh left and right input pixels to allow parallel computation of cost values. (b) Modified *FIFO* buffers — these buffers allow for serial input and parallel output and thus form an essential part of the *IBUF*.

spond to variables: d and p in the algorithm. These counters are used for backtracking through the match matrix. The counters are either decremented or incremented based on the current match value output by *MBUF*.

To facilitate state transitions between initialization, forward and backward passes, *ICNT* signals are fed into a set of boundary comparators (*BCMP*) shown in Figure 3.12a. The *MCMP* comparators, in Figure 3.12b, provide control signals that determine which decrement and increment operations should occur within the *MCNT* counter block. These comparators also help assess when the d and p counters reach zero values (*i.e.* the boundary regions of the match matrix) during backtracking.

3.2.5 Cost Computation

A core chunk of hardware logic is utilized by parallel cost computations. These computations make use of Equation 2.13 and a minimization function to generate appropriate cost values. The occlusion cost, *OC* is hard coded into the hardware via parameter-



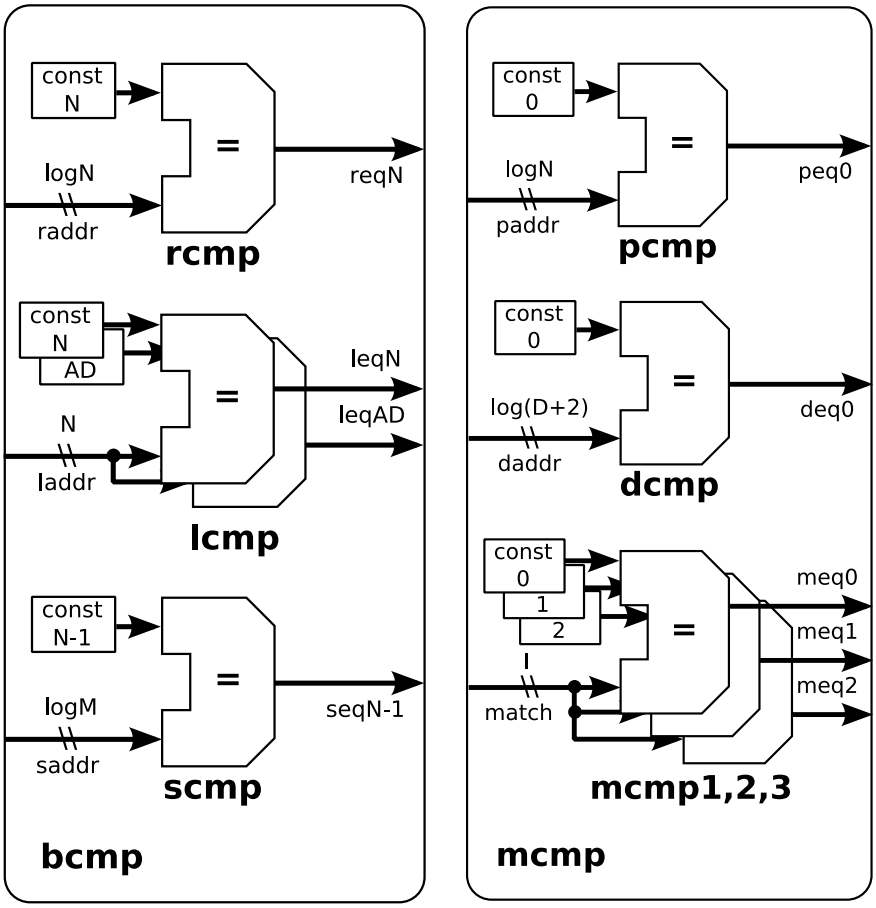
(a) Image Pixel Counter (*ICNT*) (b) Match Counter (*MCNT*)

Figure 3.11: The schematic diagram for image pixel and match memory addressing counters. (a) *ICNT* is used to retrieve raw data and index the match matrix during the forward pass. (b) *MCNT* is used to backtrack through the match matrix and to compute and store disparity data.

izations. Figure 3.13a demonstrates the design of the parallelized *NOC* computation module. Data from this module is fed through a pipeline register, *PBUF*, and inputted into the *PMIN* minimization circuitry. Figure 3.14a presents this circuit. Multiplexers in the *PMIN* module select the appropriate cost values (*costA*, *costB*, *costC* and *costD*) and add them to the occlusion cost or non-occlusion cost. The minimum of the results of the three additions is used as the actual cost (*mina*). Boundary and invalid regions (*minb* and *mini*) along the anti-diagonals have a special cost associated with them also.

Selecting between the three cost values (*mina*, *minb* and *mini*) is done by the *CMUX* module. Figure 3.14b and 3.13b show schematics of the *CMUX* module and its internal *MUX* block respectively.

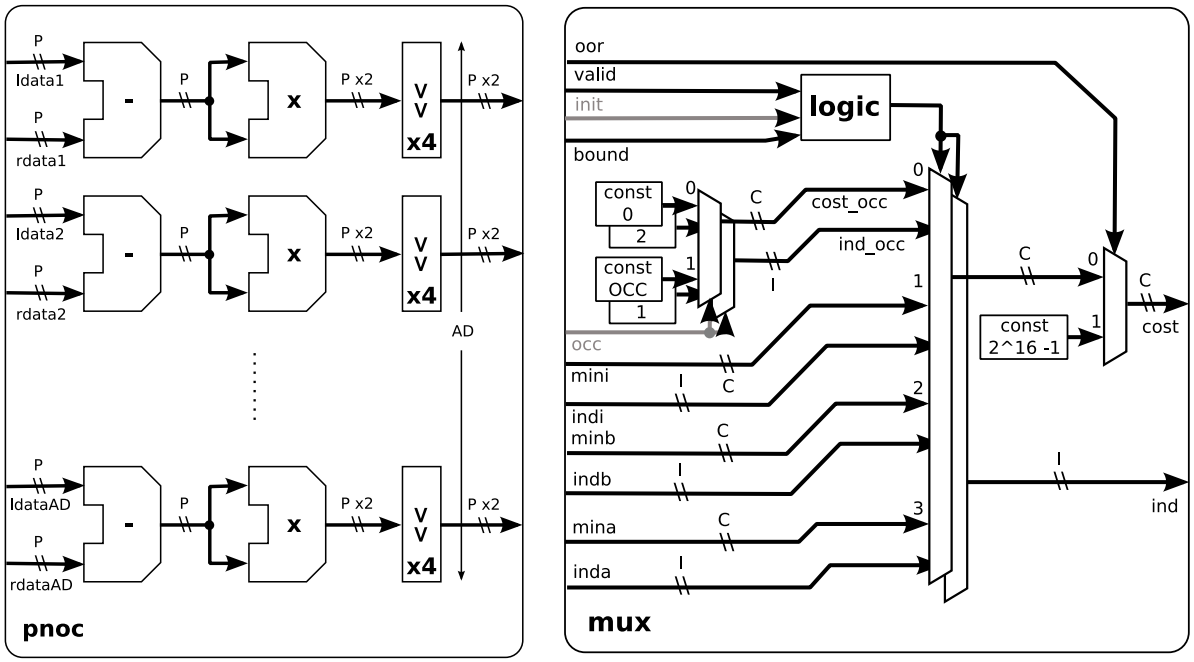
The multiplexed cost data from the *CMUX* module is inputted into the *CBUF* buffer. A schematic diagram of the buffer is shown in Figure 3.15. *CBUF* stores two previous



(a) Boundary Comparator (*BCMP*)

(b) Match Comparator (*MCMP*)

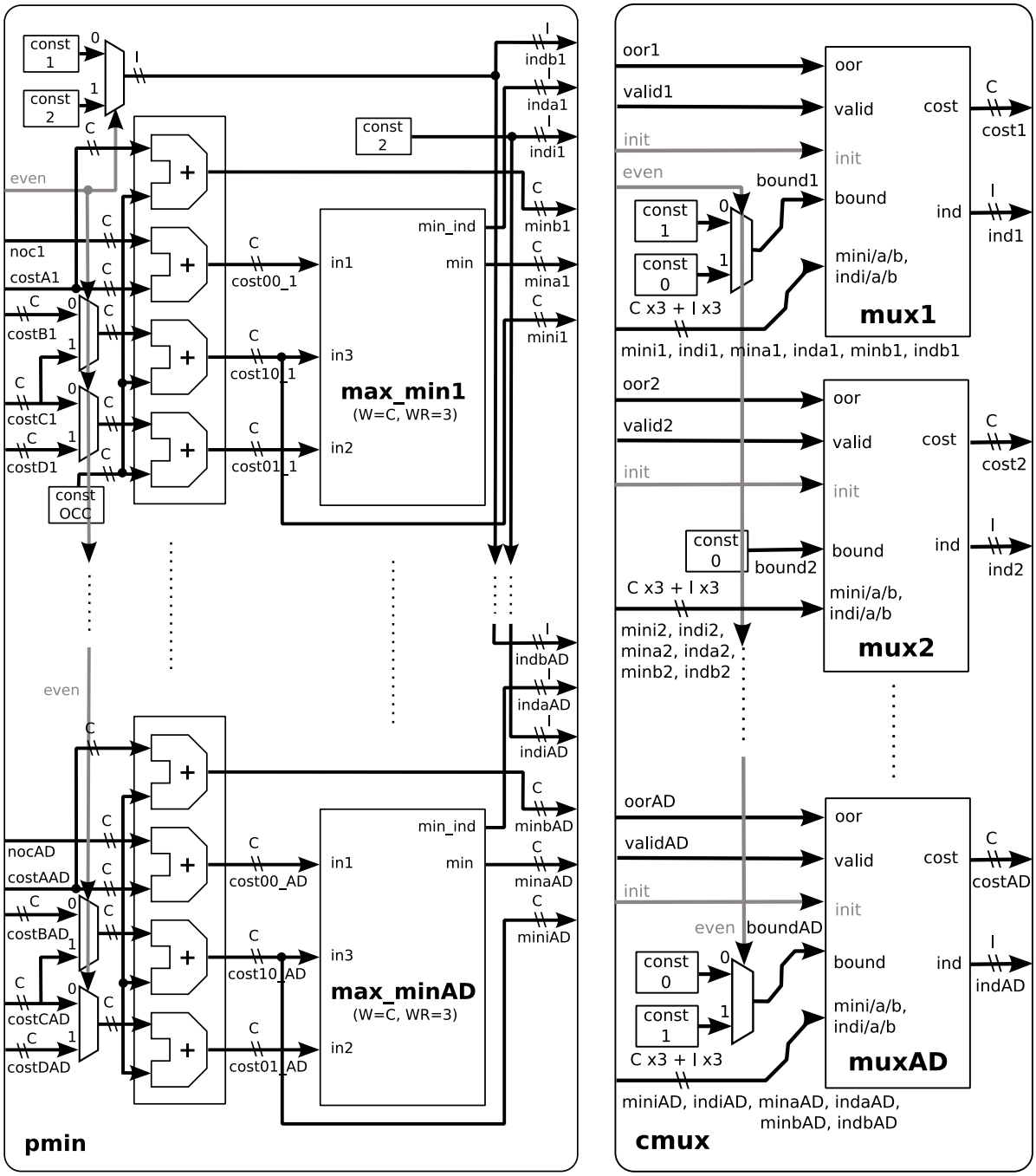
Figure 3.12: Schematic diagrams for comparators. (a) *BCMP* is used to determine end conditions, using *ICNT*. The comparator output indicates the end of the initialization phase as well as the completion of the forward and backward passes. (b) *MCMP* is used to generate counter control signals for match matrix counter: *MCNT*.



(a) Parallel Non-Occlusion Cost Unit (*PNOC*)

(b) Internal *CMUX* multiplexer (*MUX*)

Figure 3.13: Schematic diagram for the (a) Parallel non-occlusion cost unit, *PNOC* and (b) *CMUX* internal multiplexer logic unit, *MUX*.



(a) Parallel Minimum Unit (*PMIN*) (b) Cost Multiplexer (*CMUX*)

Figure 3.14: Schematic diagram for the (a) Parallel minimum cost computation unit, *PMIN* and (b) Cost multiplexer, *CMUX*.

anti-diagonals in a set of *FIFO* buffers (see Figure 3.10b). Input data from the *PNOC* module (for the next set of cost computations) is sent into the pipeline buffer, *PBUF*, at the same time that cost data from the current anti-diagonal becomes available for storage into *CBUF*. On the following clock cycle this current cost data is stored into *CBUF*. However, in order to make this data available for the next computation entering the pipeline, it is necessary to route current computations directly into the *PBUF* register from *CMUX*. A further discussion of pipelining follows later in this section.

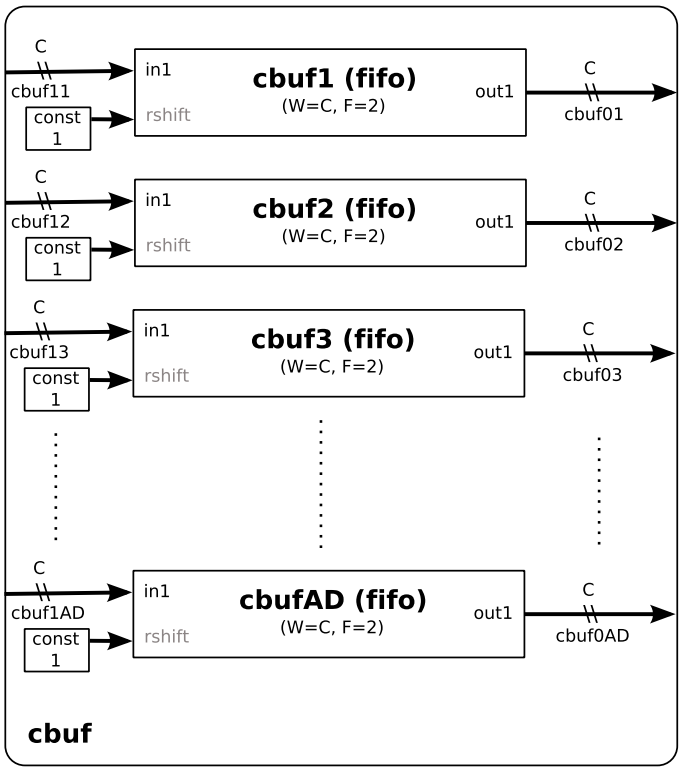


Figure 3.15: A schematic diagram for the Cost Buffer (CBUF).

3.2.6 Match and Depth Computations

During the forward pass, index data is fed from *CMUX* into the match matrix, *MBUF*. In order to prevent delays in data storage the match matrix is constructed with multiple banks, whereby data from different positions along the anti-diagonal can be stored simul-

taneously into all banks. The schematic diagram for the *MBUF* and its internal blocks is shown in Figures 3.16 and 3.17. Write signals for the Multibank RAM are generated via a decoder activated by the *even* and *init* signals. It is worth noting that subtractors 1 to *AD* are used to create appropriate addresses for each bank and its associated anti-diagonal element. During the backward pass this windowing is disabled by setting the *forward* signal to 0. It is important to observe that each bank is a synchronous RAM block that requires one clock cycle for reads. This means that during backtracking there is a one clock cycle delay before results from the placement of the next p counter address are available on the *match* signal line. This is a significant problem since the next p (and d) value is dependant on the current *match* output. This results in a one clock delay for every match value retrieved from the match matrix during the backward pass. In order to reduce this delay, to a small initial value of one clock cycle at the beginning of the backward pass, a pre-fetching process is implemented via four multiplexers: *A*, *B*, *C*, *D*.

Table 3.1 demonstrates how this pre-fetching occurs for the first few clock cycles of the backward pass. At the end of the forward pass *paddr* and *daddr* have values: N and 1 respectively. The pre-fetch takes a clock cycle, in state *INIT*, to extract the first match index value at address $p = N = 10$ (multiplexer A) and $p = N - 1 = 9$ (multiplexer B) from all banks ($d = 0$ to $d = D_{max} + 1$) of *MBUF*. A second clock cycle, in state *LOAD*, is used to output the first match value directly from memory via multiplexer A while at the same time storing this value in register *REGM*. Finally for all remaining iterations of the backward pass the match output is taken from multiplexers *B*, *C* and *D*. Recall the following operations from the parallelized DPML algorithm: if *match*$= 0$, $p--$; if *match*$= 1$, $d++$; $p--$; and if *match*$= 2$, $d--$.

A disparity buffer, *DBUF*, is used to store data generated from *MBUF*. This buffer stores the disparity value: $d-1$ as a representation of the distance between two matching pixels at location p of the backward pass scanline. Figure 3.18 presents the architecture of *DBUF*.

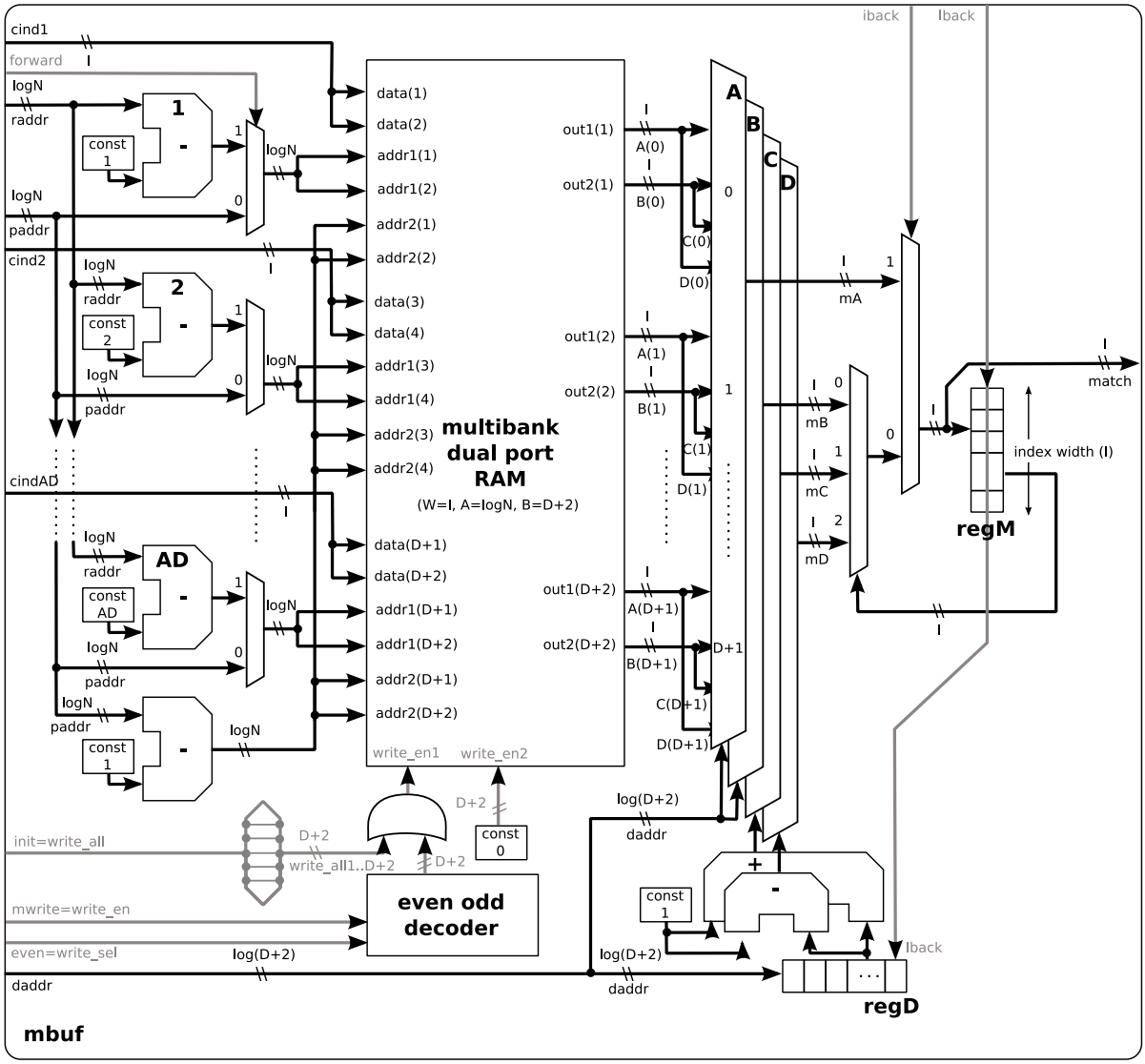


Figure 3.16: A schematic diagram for the match matrix (*MBUF*).

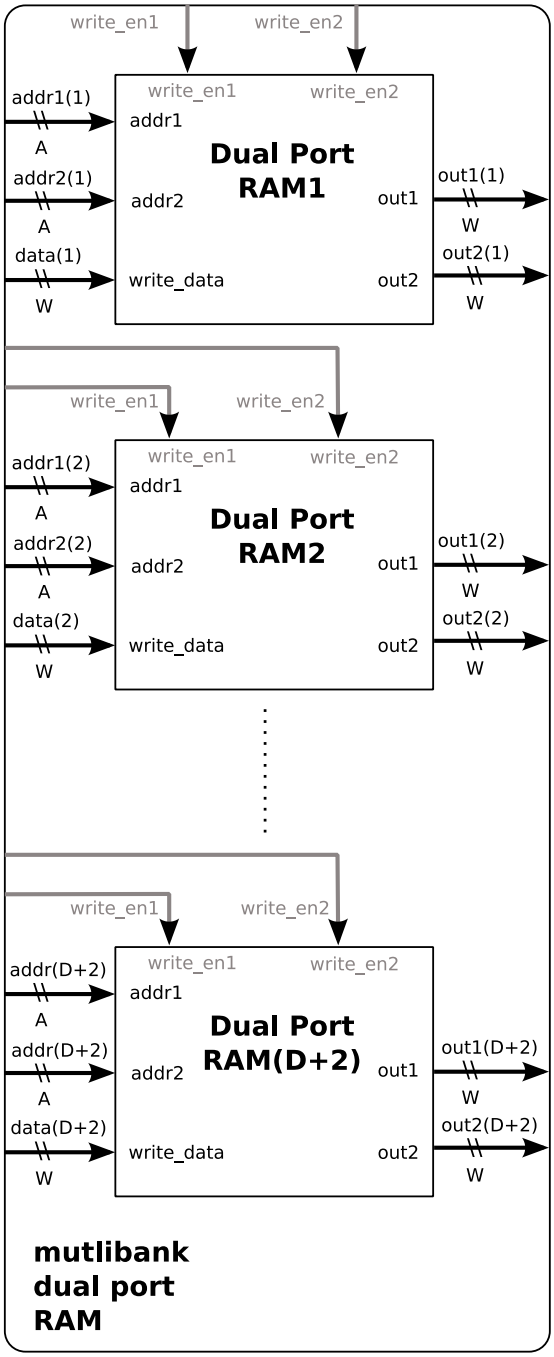


Figure 3.17: A schematic diagram for multibank dual port RAM used by *MBUF*.

Signals	**Clock 0**	**Clock 1**	**Clock 2**	**Clock 3**	**Clock 4**
paddr (p)	10	10	9	8	8
daddr (d)	1	1	1	2	1
regD	X	X	1	1	2
regM	X	X	mA(10,1)=0	mB(9,1)=1	mC(8,2)=2
mA(p,d)	X	mA(10,1)=0	mA(10,1)=0	mA(9,1)=1	mA(8,2)=2
mB(p-1,d)	X	mB(9,1)=1	mB(9,1)=1	mB(8,1)=0	mB(7,2)=0
mC(p-1,d+1)	X	mC(9,2)=2	mC(9,2)=2	mC(8,2)=2	mC(7,3)=1
mD(p,d-1)	X	mD(10,0)=1	mD(10,0)=1	mD(9,0)=2	mD(8,1)=1
match	X	mA(10,1)=0	mB(9,1)=1	mC(8,2)=2	mD(8,1)=1
STATE	INIT	LOAD	BACKTRACK	BACKTRACK	BACKTRACK

Table 3.1: A timing diagram for the *MBUF* during the first few clock cycles of the backward pass. This diagram assumes that the width of the scanline is $N = 10$ pixels. Note that a value of X indicates that the signal is not defined at the particular clock cycle. Furthermore note that an example set of match values have been chosen as outputs of *MBUF*.

3.2.7 Pipelining

The purpose of the pipeline is two fold. First, it provides a mechanism for reducing combinational delay, which in-turn allows for increased circuit clock frequency. Second, it allows micro/atomic computations (computations within an asynchronous block) to occur much like they would in an assembly line thus increasing throughput at the cost of a small initial delay required to fill the pipeline.

To better understand Figure 3.8, it is instructive to isolate the pipeline and observe the data flow path as a series of register transfers along *BUF* elements (see Figure 3.19).

At each clock cycle a new pixel is pushed into the *IBUF*. *AD* costs are computed with their associated match values stored in *MBUF*. Cost computations from a previous clock cycle are re-used by looping back to *PBUF*. The *PBUF* register acts to reduce combinational delay (and hence, increase maximum clock frequency) by breaking down cost

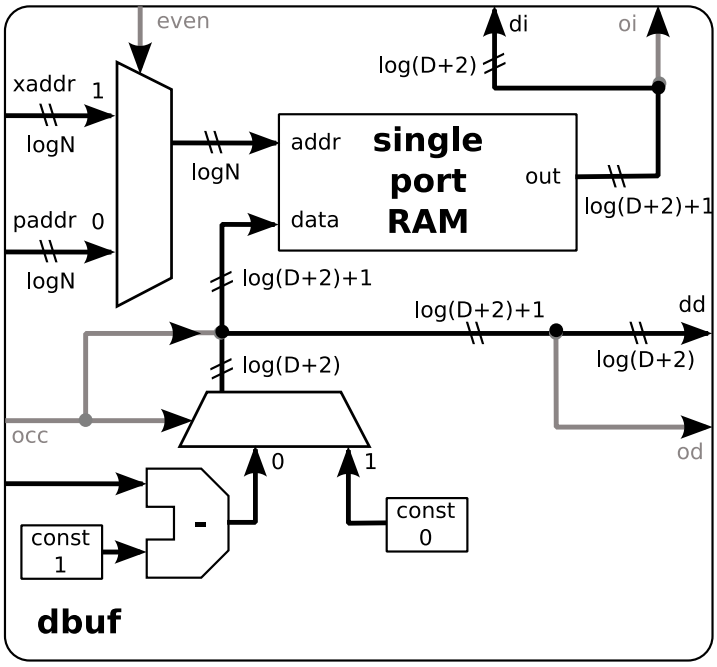


Figure 3.18: A schematic diagram for the disparity memory, *DBUF*.

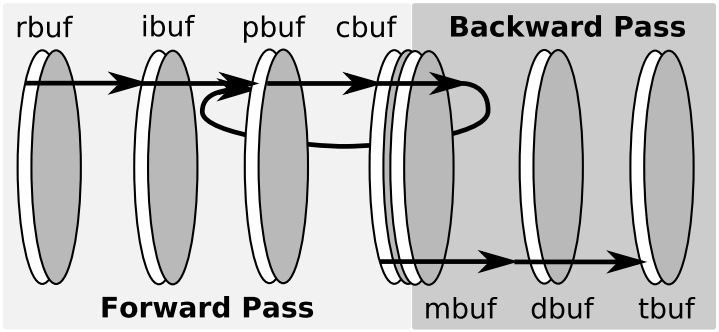


Figure 3.19: The hardware pipeline. The forward pass takes place first, repeating until cost computations for all pixels within a disparity range are completed. The backward pass then proceeds to locate the minimum energy path to generate the final disparity and depth results.

computations into a non-occlusion cost (*PNOC*) and minimum cost selection (*PMIN*) phase which together constitute the longest path. *PNOC* and *PMIN* together compute *AD* cost values simultaneously using an assortment of adders, multipliers and comparators. Since data in the forward pass and backward pass go through two distinct pipelines, control signals must be routed via *BMUX* to ensure synchronization (see Figure 3.20).

3.2.8 State Machine

The state machine ensures that data is fed correctly through the pipeline (see Figure 3.21). The *SKIP* state fills the *IBUF* pipeline registers. During this filling process a full anti-diagonal of pixel data may not be available. The buffer, *VBUF*, *ORBUF* and *OLBUF* containing a list of valid pixels ensures that these boundary regions don't affect cost computations.

The *AD_EVEN* and *AD_ODD* states process data for even and odd anti-diagonals respectively and ensure that the staggered alignment of previous anti-diagonals is appropriately considered for boundary cost computations. The *STALL* state at the end of the forward pass ensures that the last pixel computation has time to reach the end of the forward pipeline. This is necessary due to the one clock cycle delay incurred when initially transferring data into the *PBUF*.

During the backward pass, *MBUF* synchronous read operations have a one clock cycle latency from the time that an index address from the p counter (signal *paddr*) is placed on the address lines. The *INIT* state ensures that this latency is taken into account for the first read operation. Since the next read address for the *MBUF* is determined by the currently read data, it is necessary to pre-fetch the data for all potential next address candidates. Without this pre-fetch, a one clock cycle penalty would be incurred for every read operation executed on the *MBUF*. The *LOAD* and *BACKTRACK* states implement this pre-fetch and compute disparity values for all pixel locations in the current scanline.

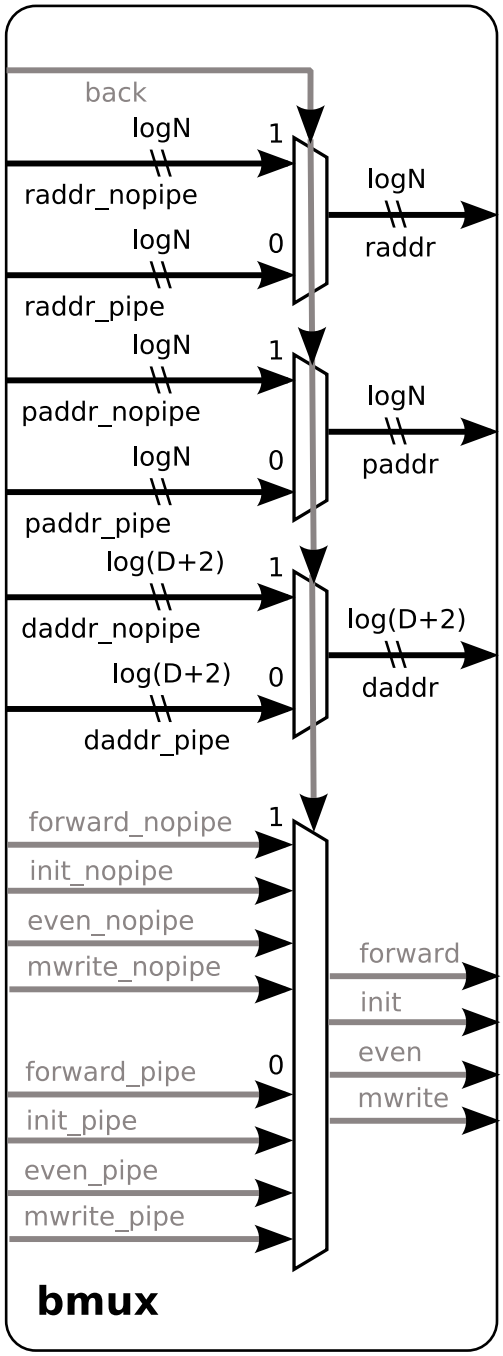


Figure 3.20: A schematic diagram for the Backtracking Multiplexer (BMUX).

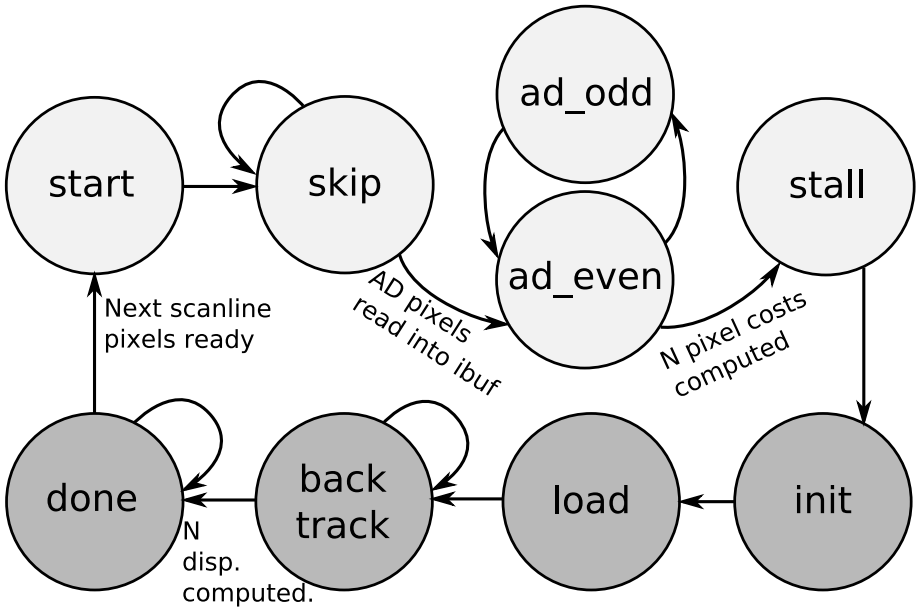


Figure 3.21: The state machine for the pipelined hardware architecture shown in Figure 3.8. Dark states indicate the backward pass while light ones indicate the forward pass.

3.2.9 Discussion

The DPMLHW(P) hardware makes significant improvements over both software and hardware implementations of stereo correspondence algorithms. A specific comparison with other implementations can be found in Chapter 4. Further improvements can still be achieved — these improvements are discussed below and have been denoted as the DPMLHW(P)* and DPMLHW(P)** implementations.

First, as the number of parallelized units, AD, increases so does fanout. Relatively few control and data signals end up driving a large number of computational units. The large fanout requires signal boosting, while the large number of parallelized units result in congestion on the particular FPGA device. The effect of parallelization, thus, is a potentially increased routing delay and hence lower clock frequencies. Reconfigurable logic tends to take up large surface areas. In comparison ASIC implementations generally take up less space. Some FPGA systems (such as the Xilinx Virtex 5) implement ASIC versions of multipliers, adders and other components in the form of DSP blocks.

Utilization of these DSP blocks can result in less congestion and likewise reduced routing delays. Furthermore, ASIC logic delays tend to be lower. An implementation of the DPMLHW(P) that utilizes appropriate DSP logic can run significantly faster.

It was noted earlier that the key difference between the partially pipelined implementation, DPMLHW(PP), and the fully pipelined implementation, DPMLHW(P) lay in the presence of an additional buffer, *PBUF*, that performed finer grain pipelining. This pipelining reduced the combinational delay by effectively reducing the longest path between *IBUF* and *CBUF*. A reduced combinational delay, in-turn, meant a higher maximum operating clock frequency F_{max_clk}. Further and even finer grain pipelining of the data path is possible by inserting registers within the *PMIN* and *CMUX* modules. The methods used to perform this pipelining differ. Since a particular next cost computation is dependant upon the results of a previous cost computation, the insertion of additional registers along the datapath will require halting the pipeline in order to make data available for a loop back of cost data (*costA*, *costB*, *costC*, and *costD*) into *PBUF*.

A method known as architectural retiming [11] can be used to implement this additional finer grain pipelining by inserting negative registers that predict cost values for the next cost computation before they have been completely computed in the current pipeline cycle. These registers act as pipelining registers that send data backwards along the pipeline. For example, the outputs of the adders in *PMIN* (Figure 3.14a) could be used as a prediction of the next cost. Additional test circuitry that compares this predicted cost to the actual cost can be added to restore the pipeline to its original state in the event that the prediction turns out to be incorrect. An incorrect prediction would thus result in a significant penalty to the overall speed of the circuit. The challenge with additional pipelining would then be to find a good predictor for the next cost values entering *PBUF*. Chapter 4 provides quantitative estimates for the effectiveness of a sample predictor that may be used for architectural retiming. Note that the abbreviation DPMLHW(P)* has been assigned to denote this hypothetical and improved

implementation that takes additional pipelining and delay reduction into consideration.

Currently both backward and forward passes operate in a sequential manner. This, however, need not be the case. The two phases themselves can be pipelined such that the forward pass of the next scanline begins while backtracking is in progress. The difficulty of pipelining in this manner lies in the fact that the forward pass writes data to increasing indices of the match matrix while the backward pass iterates through the same matrix in the opposite direction. As some point, the data overwritten by computations of the next scanline, in the forward pass, will be required by the backward pass phase of the current scanline. One method of storing this data is to use a double buffering scheme. The scheme would make two copies of the match matrix such that at any given time the forward pass would write to a different memory than that being used for backtracking. A less resource intensive approach is also possible. Instead of duplicating the memory associated with the match matrix, the forward pass can be made to write into the match matrix in the reverse order that it would normally write. If this reversal lags behind the backward pass such that it is always one p address value behind the backward pass computations then no unused data will be overwritten. The next scanline backward pass thus also proceeds in the opposite direction of the normal backward pass. Over the course of M scanlines, the forward and backward passes then flip flop between writing and reading the match matrix using reversed addressing and the normal addressing of the current DPMLHW(P) implementation. This backward and forward pass pipelining increases scanline throughput resulting in a net increase in the frame-rate of the stereo correspondence system. Chapter 4 discusses the timing performance of this interleaved DPMLHW(P)** implementation further.

3.3 Summary

Two key hardware implementations for Cox's [4] DPML algorithm are presented. DPMLHW(S) presented a naïve sequential implementation, while DPMLHW(P) and DPMLHW(PP) presented pipelined and parallelized implementations. The hardware implementations attempt to optimize the speed at which stereo correspondence results are generated. The DPMLHW(S) approach noted that the entire cost matrix did not need to be stored for backtracking. However, computations for a particular position in the cost matrix required the sequential computation of all other costs in that particular row of the matrix. The DPMLHW(PP) approach went further and noted that anti-diagonals in the cost matrix were dependant only on two previous anti-diagonals. This suggested that AD parallel cost computations could be performed in a single clock cycle resulting a significant improvement over the DPMLHW(S) implementation. Finally, the DPMLHW(P) approach attempted to reduce combinational delay and hence increase the maximum clock frequency at which the stereo correspondence circuit operated. A pipelining register, *PBUF*, was introduced between the *PNOC* and *PMIN* module — this register effectively increased clocking frequency by breaking up the longest path from *IBUF* to *CBUF* registers. A backtracking multiplexer, *BMUX*, ensured that this additional pipelining would not interfere with the addressing of the partial match matrix, *MBUF*, during backtracking.

Finally, it was noted that further speed increases are possible if the DPMLHW(P) architecture is modified to accomodate additional pipelines between the *PBUF* and *CBUF* registers. These pipelines will, however, require a method to predict cost values, for subsequent cycles, before the data in the current cycle has reached the end of the pipeline. A method known as architectural retiming [11] is proposed for these future speed optimizations. Additional speed improvements are possible by utilizing FPGA DSP blocks or by pipelining the backward and forward passes associated with the stereo depth estimation.

Chapter 4

Results and Discussion

This chapter provides a performance summary of the DPMLHW stereo matching system discussed in this document. Section 4.1 begins with a discussion of the accuracy of 3D reconstruction with respect to other stereo matching algorithms. A comparsion is made not only to Cox's DPML algorithm but also to SSD, Correlation and additional algorithms compiled by Scharstein *et al.* [22][23]. Following this, Section 4.2 demonstrates that the particular DPMLHW implementation presented in this thesis operates at vastly superior frame rates to existing algorithms. Finally, Section 4.3 looks at the hardware resource utilization and associated integration and timing concerns.

It is worth noting that the results presented were generated from standard stereo data sets and quality metrics compiled and used by Scharstein *et al.* [22][23] as well as data sets obtained from the actual 3D reconstruction system implemented in hardware (DPMLHW). To function in a standalone manner, the hardware system can integrate image aquisition circuitry and rectification logic developed by Kirischian *et al.* [15] and Islam [12] respectively. A final point to note is that existing hardware was developed and tested on a Xilinx Virtex 2 pro FPGA development platform.

4.1 Accuracy

As noted earlier (in Chapter 2), most high speed stereo algorithms achieve high frame rates at the expense of accuracy and resolution. In contrast, the DPMLHW implementation presented in this document demonstrates a comparable degree of accuracy to existing matching algorithms. It is interesting to note that the DPMLHW implementation differs from many of existing hardware implementations. Existing implementations tend to use simplistic SSD, SAD or other similar Correlation-based formulations that tend to perform poorly during 3D reconstruction. Table 4.1 shows the ranking of disparity estimates generated via hardware with respect to a list of existing algorithms compiled by Scharstein *et al.* in [22][23].

The specific ranking for the hardware reconstruction were obtained from automatic evaluation tools available from Scharstein *et al.* [24]. A complete list of rankings, for comparison purposes, may also be found in [24]. Furthermore, results are compared to ground truth data using root mean squared error and percent bad matching pixels metrics. These metrics have been used extensively by the vision community to evaluate stereo correspondence results and are likewise used by Scharstein *et al.* in [22] [23]. For the sake of completeness they have been defined in Equations 4.1 and 4.2.

$$RMS = \sqrt{\frac{1}{K}\sum_{(x,y)}^{K}\left(D_{est}(x,y) - D_{true}(x,y)\right)^2} \tag{4.1}$$

$$BAD = \frac{1}{K}\sum_{(x,y)}^{K}\left|D_{est}(x,y) - D_{true}(x,y)\right| > \delta_D \tag{4.2}$$

Figure 4.1 provides a visualization of the specific data sets, ground truth disparities and occlusions used for evaluating the accuracy of correspondence results. Figure 4.2 demonstrates some reconstruction results for Sum of Squared Difference (SSD) and Correlation (CORR) algorithms, while Figure 4.3 demonstrates additional results for the

DPML and DPMLHW implementations that are the focus of this thesis.

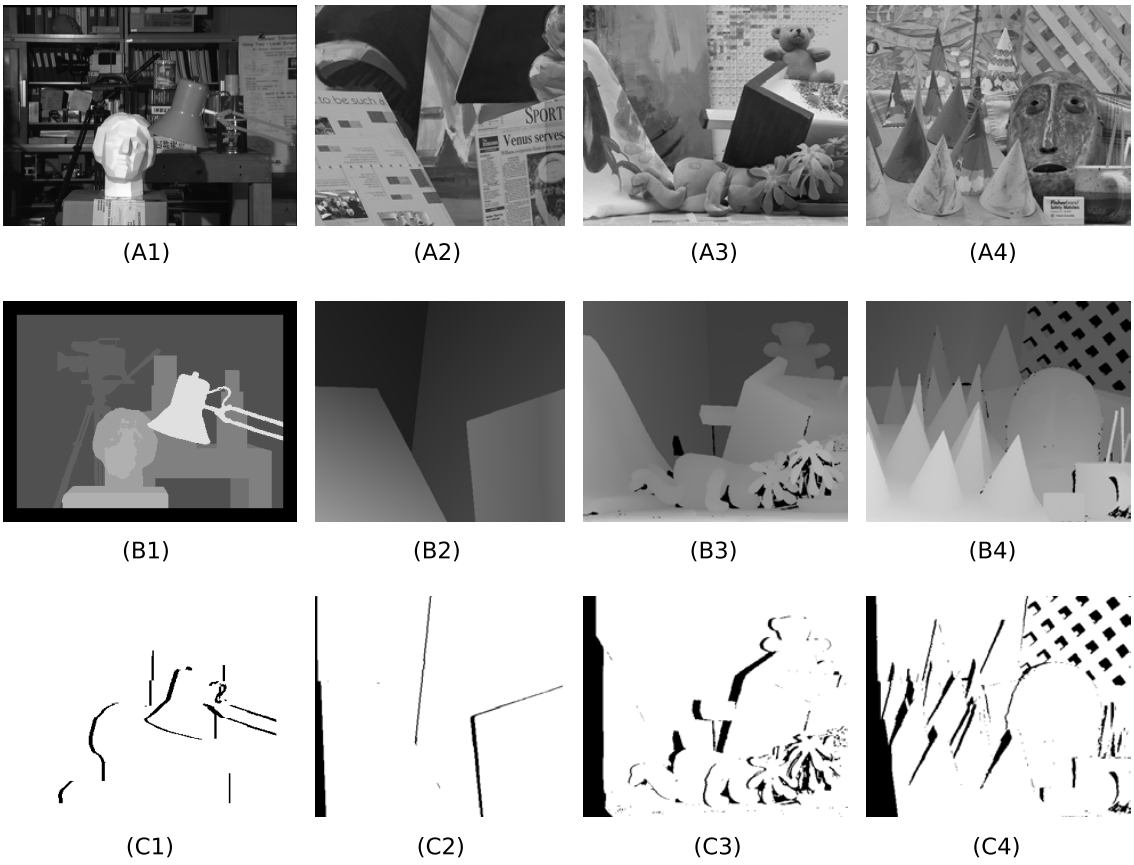


Figure 4.1: Tsukuba, Venus, Teddy and Cones data sets and their ground truths. Sub-images A1-A4 show the left image acquired by the stereo camera system. Sub-images B1-B4 and C1-C4 show ground truth disparity and occlusion maps respectively. These standard data sets are used for evaluating accuracy of stereo correspondence results [22][23][24].

4.1.1 Improving Accuracy

Upon examination of visual results it is noted that in contrast to the standard data sets in which disparity estimation is shown to be quite successful, real world stereo images with large amounts of uncorrelated noise tend to confuse the algorithm. This is demonstrated in Figure 4.4 which presents images acquired from a high speed stereo camera designed by Kirischian *et al.* [15] — the images contain uncorrelated or structured noise that makes pixel comparison across the two views difficult. In order to minimize errors in 3D

Algorithm	**Avg. Rank**	**RMS Error**			
		Tsukuba	Venus	Teddy	Cones
DPMLHW	36.8	0.7400	1.1308	1.0658	1.1248
DPML	36.8	0.7400	1.1307	1.0658	1.1248
CORR	37.0	1.3607	1.0720	2.3884	2.1656
SSD	38.0	2.4551	3.7694	6.9529	5.3069

Algorithm	**Avg. Rank**	**% Bad Pixel Match**			
		Tsukuba	Venus	Teddy	Cones
DPMLHW	36.8	2.8127	4.7524	3.4415	3.8386
DPML	36.8	2.8127	4.7520	3.4414	3.8384
CORR	37.0	6.0886	3.6928	8.9299	6.5164
SSD	38.0	12.3657	13.7543	27.2931	17.2437

Table 4.1: Accuracy rankings, root mean squared error and percent bad matching pixels for four standard data sets from Scharstein *et al.* [22] [23]: Tsukuba, Venus, Teddy and Cones. Note that accuracy rankings are determined by evaluation tools from [24]. Lower values indicate better performance. Also note that four algorithms are compared: (1) DPMLHW, Dynamic Programming Maximum Likelihood in Hardware. (2) DPML, Dynamic Programming Maximum Likelihood in software. (3) CORR, Correlation with 11 by 11 window size. (4) SSD, Sum of Squared Difference with 11 by 11 window.

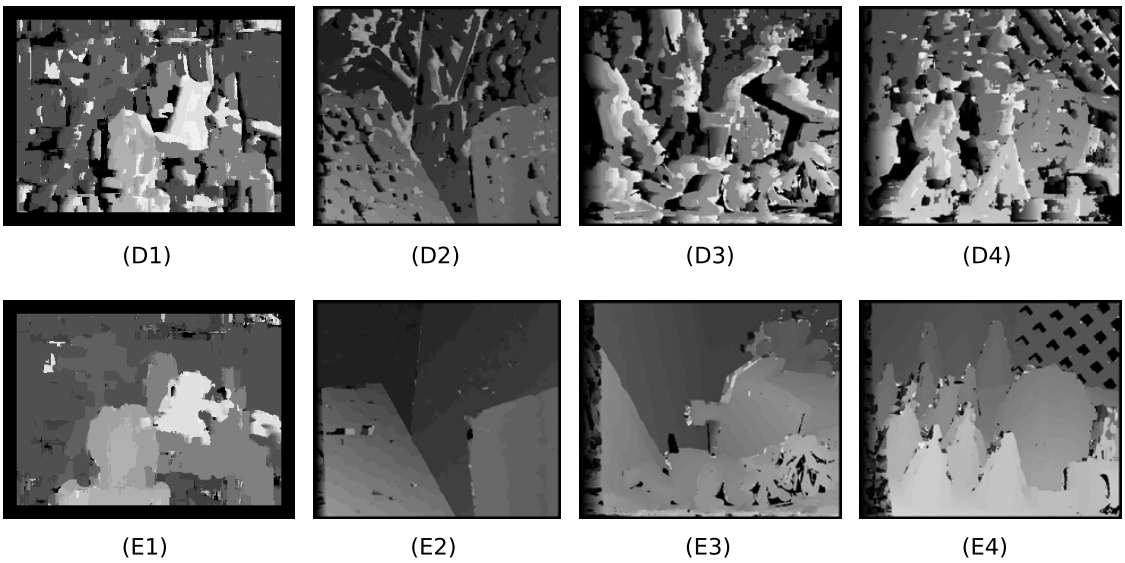


Figure 4.2: Stereo correspondence results for Sum of Squared Difference (SSD) and Correlation (CORR) algoritms. Sub-images D1-D4 refer to the SSD algorithm while sub-images E1-E4 refer to the CORR algorithm.

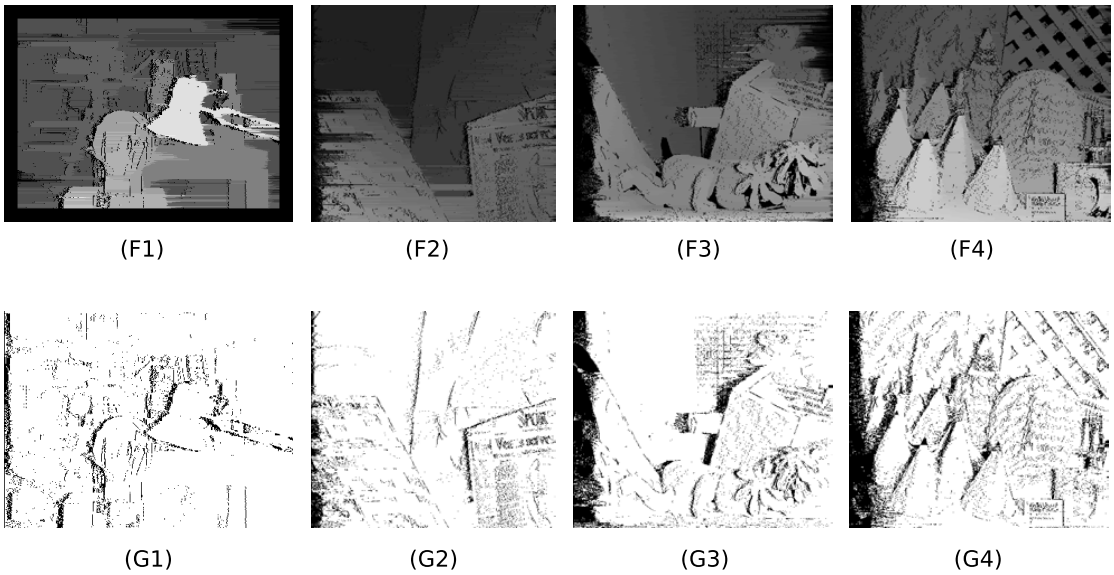


Figure 4.3: Stereo correspondence results for Dynamic Programming Maximum Likelihood (DPML) and DPML Hardware implementations. Note that both the DPML and DPMLHW algorithms are virtually visually indistinguishable. Sub-images F1-F4 show the disparity estimates for the Tsukuba, Venus, Teddy and Cones data sets respectively, while sub-images G1-G4 show the occlusion estimates.

reconstruction, pre-filtering of image data can improve performance. This improvement is shown in Figure 4.5 and 4.6 where input images are smoothed using Gaussian and median filters. It is worth noting that the system shown produces poor disparities only as a result of high levels of noise — as noted by Figure 4.3 when this noise is relatively low, disparity estimates are more accurate.

As mentioned in Chapter 2 differences in illumination between two camera views can cause inaccuracies during depth estimation. Since the cost function in 2.13 is dependant on pixel intensities these differences directly effect the cost value. Figure 4.4 suffers from these illumination changes. Performing some form of image normalization can alleviate errors resulting from variance in illumination as long as the image statistics across both cameras are expected to be similar — this would not necessarily be true for large baseline stereo cameras which may contain widely divergent image intensity distributions. Performing this normalizing over localized regions may prove to be more

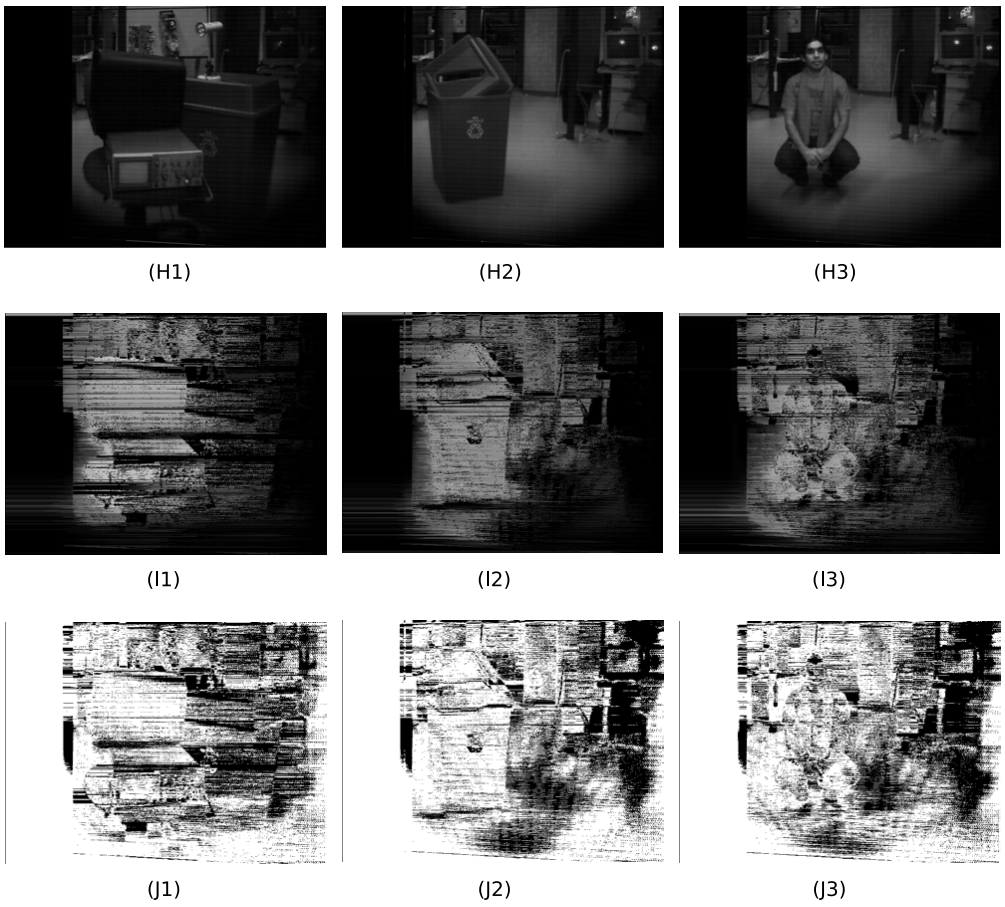


Figure 4.4: Stereo correspondence results from the Dynamic Programming Maximum Likelihood Hardware (DPMLHW) implementation presented in this document. Input images were acquired by a stereo camera system developed by Kirischian [15]. Sub-images H1-H3 show the input data, while sub-images I1-I3 and J1-J3 show the disparity and occlusion maps, respectively.

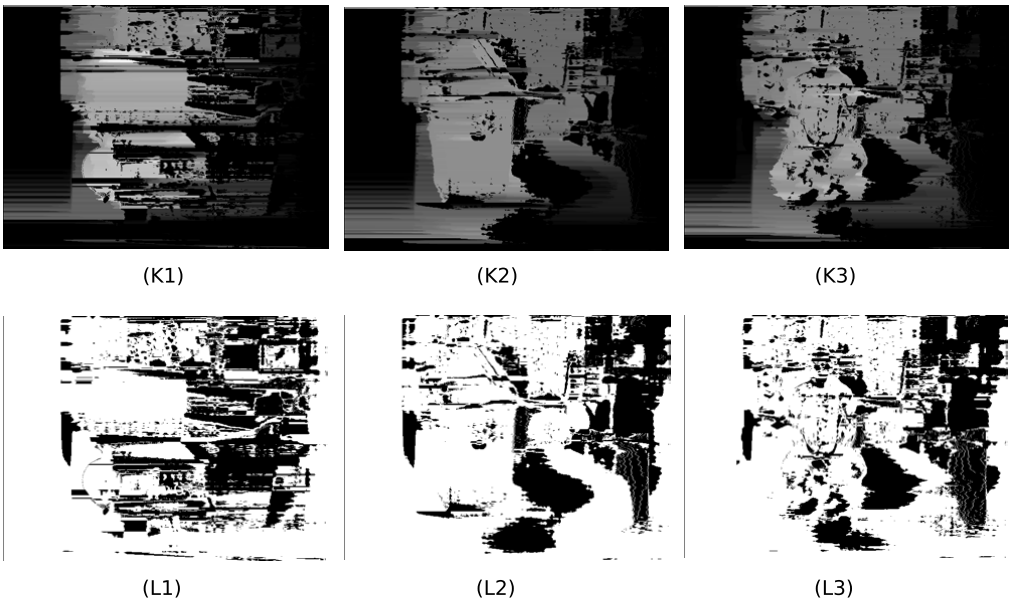


Figure 4.5: Stereo correspondence results from the Dynamic Programming Maximum Likelihood Hardware (DPMLHW) implementation presented in this document. Input images were acquired by a stereo camera system developed by Kirischian [15] and pre-filtered using a 5×5 gaussian smoothing window. Sub-images K1-K3 and L1-L3 show the disparity and occlusion maps respectively.

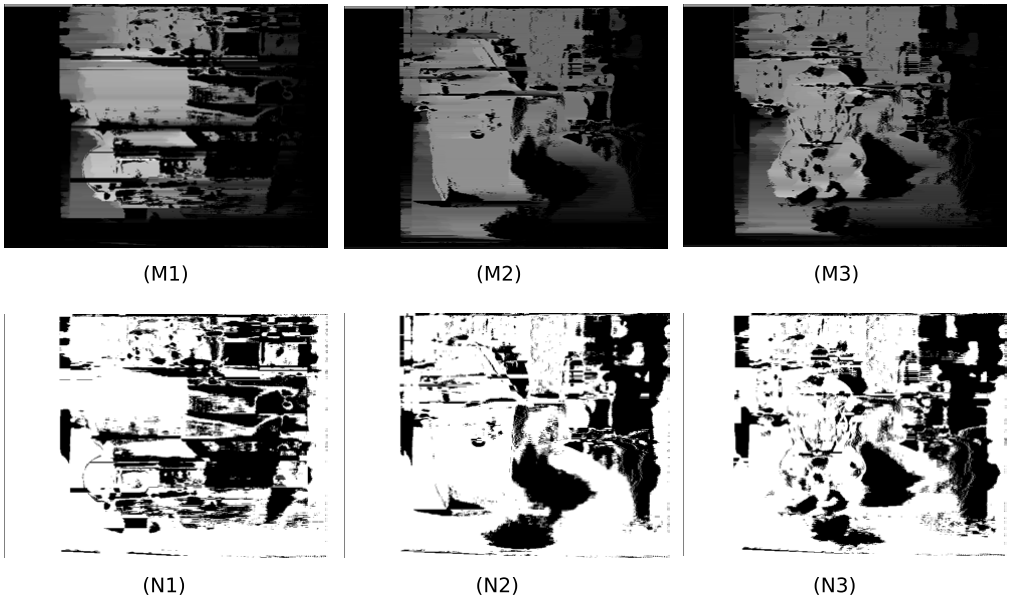


Figure 4.6: Stereo correspondence results from the Dynamic Programming Maximum Likelihood Hardware (DPMLHW) implementation presented in this document. Input images were acquired by a stereo camera system developed by Kirischian [15] and pre-filtered using a 5×5 median smoothing window. Sub-images M1-M3 and N1-N3 show the disparity and occlusion maps respectively.

successful.

Modifications to the DPML algorithms, presented in Chapter 3, also hold the potential for improved accuracy in noisy conditions. These modifications involve a hybrid approach to dynamic programming which utilizes costs aggregated over a correlation window. Hardware that utilizes adder tree like structures can then be used to provide fast window summation for these windowed computations.

4.2 Speed and Timing

Performance may, additionally, be characterized by the frame rate at which stereo correspondence results are generated by hardware. A summary of this runtime performance is shown in Table 4.2 for maximum disparities: $D_{max} = 16$ and $D_{max} = 128$ and data resolutions: $M \times N = 640 \times 480$ and $M \times N = 320 \times 240$ pixels. At frame rates of $FPS = 63$ fps and image resolution of $M \times N = 640 \times 480$ pixels, the fully parallelized DPMLHW implementation introduced in this paper vastly out-performs existing correspondence algorithms. At slightly lower input image resolutions a significantly higher frame rate of $FPS = 248$ fps is achieved, a speedup factor of over 200 when compared to Cox's DPML [4] software implementation and over 10 when compared to existing hardware implementations.

Note that Table 4.2 shows timing results from four different implementations. DPMLHW(P), DPMLHW(PP) and DPMLHW(S) refer to the fully parallelized, partially parallelized and serial FPGA hardware implementations discussed in Chapter 3. The Xilinx Virtex 2 pro FPGA development platform was used for these implementations. DPML refers to a C language software implementation of the dynamic programming algorithm (see Cox *et al.* [4]). The results for this DPML software were generated on a computer system with 4 GB of physical memory and a quad core 2.66 Ghz Intel Xeon processor running the Linux 2.6.18 kernel. Note that DPMLHW(P)* indicates results

that may potentially be possible given that signal routing and logic delays, occuring in DPMLHW(P), can be optimized - a Xilinx Virtex 5 implementation has demonstrated that this is possible. Likewise, DPMLHW(P)** indicates the theoretical results of interleaving backward and forward passes with respect to each other. A further discussion regarding both the DPMLHW(P)* and DPMLHW(P)** implementations will follow towards the end of this section. It should be noted that the variability in F_{max_clk} for the DPMLHW(P) implementation is a result of routing delays generated due to increasing parallelization. This F_{max_clk} was estimated using post-synthesis simulations.

Results show that the fully pipelined and parallelized architecture DPMLHW(P) discussed in this thesis produces the highest frame rates at high resolutions and disparity ranges. Note that in the case of DPMLHW, the listed frame rates are the worst case frame rates given a maximum operating clock frequency, F_{max_clk}. On average, depending on the incoming image data, performance may be significantly better than this worst case.

Equations 4.3 and 4.4 are the result of a timing analysis of the serial and pipelined hardware implementations presented in Chapter 3. The equations listed provide worst case estimates. In reality, the length of time taken by the backtracking phase is dependant upon input data. The backward pass may take anywhere between N and $2N$ clock cycles resulting in frame rates faster than this worst case.

$$FPS_S = \frac{F_{max\_clk}}{M(5N + D_{max}N)} \tag{4.3}$$

$$FPS_P = \left[\left(4N + \frac{D_{max}}{2} - 1\right)\frac{M}{F_{max\_clk}}\right]^{-1} \tag{4.4}$$

A further comparison of the DPMLHW implementation relative to 10 other existing FPGA hardware approaches is shown in Table 4.3. For the purposes of this comparison a measure for the computational speed, in Depth Pixels per Second (DPS), will be used. This metric can be used directly or after normalization (DPSN) with the clock frequency

Algorithm	D_{max}	F_{max_clk}	**Resolution**	**FPS**
DPMLHW(P)*	128	125 Mhz	640 × 480	99.28
DPMLHW(P)**	128	80 Mhz	640 × 480	123.85
DPMLHW(P)	128	80 Mhz	640 × 480	63.54
	16	100 Mhz	640 × 480	81.16
	128	80 Mhz	320 × 240	248.20
	16	100 Mhz	320 × 240	323.75
DPMLHW(PP)	128	67 Mhz	640 × 480	53.22
	16	67 Mhz	640 × 480	54.37
	128	67 Mhz	320 × 240	207.86
	16	67 Mhz	320 × 240	216.91
DPMLHW(S)	128	47 Mhz	640 × 480	1.15
	16	47 Mhz	640 × 480	7.28
	128	47 Mhz	320 × 240	4.60
	16	47 Mhz	320 × 240	29.14
DPML	128	2.6 Ghz	640 × 480	0.18
	16	2.6 Ghz	640 × 480	0.24
	128	2.6 Ghz	320 × 240	1.24
	16	2.6 Ghz	320 × 240	2.01

Table 4.2: A timing comparison of various DPML hardware and software implementations used to compute stereo correspondence. DPMLHW(P)* indicates results that are possible if signal routing delay can be optimized — implementations on the Xilinx Virtex 5 FPGA have shown this improved performance. Similarly DPMLHW(P)** indicates speed improvements from pipelining of forward and backward passes with respect to each other.

of the system to yield a reasonable measure of performance — equations for computing the DPS and DPSN are shown in equations 4.5 and 4.6. Furthermore note that the table presents techniques for hardware disparity estimation that fall roughly into three categories: SAD, Phase and Census transform approaches. As mentioned earlier, most present day hardware disparity estimation techniques utilize SAD-like correlation.

$$DPS = N \times M \times D_{max} \times FPS \tag{4.5}$$

$$DPSN = \frac{N \times M \times D_{max} \times FPS}{F_{max\_clk}} \tag{4.6}$$

There is a strong dependancy between frame rate and maximum clock frequency. Maximum frequency (F_{max_clk}) is affected by routing and combinational delay. Routing delay mainly derives from the time required for signals to propagate between components. Combinational delay mainly derives from complexity of internal logic within a component. Improperly connected components result in long routing delays which, ultimately, affect circuit speed. When logic utilization is high, routing becomes difficult. It is, therefore, prudent to minimize utilization. These delays can be minimized by introducing synchronous buffers (pipeline registers) between blocks of combinational logic (Chapter 3). Table 4.4 demonstrates the longest path delays for each of the (P), (PP) and (S) flavours of the DPMLHW hardware. Moving from serial to partial and complete parallelization significantly reduces internal logic delays.

In the case of the DPMLHW(P) and DPMLHW(PP) implementations it is important to note that high levels of parallelization can increase routing delays. These delays are a result of duplicated logic elements (such as adders, comparators and multipliers) that take up, on a unit level, large areas of the FPGA. The footprint of these elements forces signal routing over longer and longer distances. Furthermore, another side effect of the

Algorithm	D_{max}	**Resolution**	F_{max_clk}	**FPS**	**DPS**	**DPSN**
DPMLHW(P)	128	640×480	80 Mhz	63.54	2.477×10^9	30.96
SAD_MIYA	200	640×480	40 Mhz	18.90	1.161×10^9	29.03
SAD_PERR	256	512×512	286 Mhz	25.60	1.717×10^9	6.01
SAD_JACO	178	178×146	158 Mhz	—	1.400×10^9	8.86
SAD_JIA	64	640×480	60 Mhz	30.00	0.590×10^9	9.83
SAD_SIMH	64	512×512	100 Mhz	0.34	0.006×10^9	0.06
CENSUS_WOOD	52	512×480	—	200.00	2.556×10^9	—
PHASE_MITE	20	256×256	200 Mhz	25.00	0.032×10^9	0.16
PHASE_DIAZ	4	640×480	65 Mhz	211.00	0.259×10^9	3.98
PHASE_MASR	128	640×480	—	30.00	1.180×10^9	—
PHASE_DARA	20	256×360	—	33.00	0.061×10^9	—

Table 4.3: A comparison of frame rates and depth pixels per second of various existing hardware implementations used to compute stereo correspondence. The prefix in front of the algorithm name represents the underlying technique, SAD for Sum of Squared Difference, CENSUS for the Census Transform, PHASE for Phase-based correlation and DPML for Dynamic Programming. The suffix indicates the authors associated with the implementation: MIYA - Miyajima *et al.*[20]; PERR - Perri *et al.* [21]; JACO: Jacobi *et al.* [13]; JIA - Jia *et al.* [14]; WOOD - Woodfill *et al.* [29]; MITE - Mitéran *et al.* [19]; MASR - Masrani *et al.* [18]; DARA - Darabiha *et al.* [5].

Algorithm	**Path**	**Routing**	**Logic**	**Total**	F_{max_clk}
DPMLHW(P)	PBUF to CBUF	5.739 ns	6.726 ns	12.465 ns	80.22 Mhz
DPMLHW(PP)	IBUF to CBUF	4.551 ns	10.158 ns	14.710 ns	67.98 Mhz
DPMLHW(S)	IBUF to CBUF	8.048 ns	13.492 ns	21.540 ns	46.42 Mhz

Table 4.4: Longest path delays, in nanoseconds, for fully/partially parallelized and serial hardware implementations of Cox's DPML [4] algorithm. These paths are for hardware synthesized at 640×480 disparity resolution and $D_{max} = 128$.

increased parallelization is the presence of a few key signals that drive a large number of parallelized inputs. These signals are said to have large fanouts — in order to drive gates to which they are conected some form of signal boosting buffer must be inserted at key points along the signal's path. This buffering introduces further routing delays. Inserting additional pipelining registers can help alleviate this problem. The variability in the maximum clock frequency (F_{max_clk}) seen in Table 4.2 is a direct result of this routing delay. When a small number of parallelized adder and multiplier units are generated, *i.e.* when D_{max} is small, routing delays are shorter (F_{max_clk} is high) and the signal has to travel a shorter distance to reach all of the elements in the parallelized blocks. As the level of parallelization increases the delays increase, resulting in a lower F_{max_clk}.

4.2.1 Improving Timing Performance

As seen in Table 4.4, a decrease in the logic and routing delay can increase clock frequency. In order to achieve 100 fps circuit operation, as shown by the DPMLHW(P)* entry in Table 4.2, this combined logic and routing delay would have to be reduced by 4.465 ns. While inserting pipelines into the the DPMLHW(P) longest path from *PBUF* through *PMIN* and *CMUX* would reduce delays, it cannot be done directly. This is because a computation in the current cycle relies on previous cost values generated by an atomic block consisting of *PMIN* and *CMUX*. As discussed in Chapter 3, to introduce further pipelining a method known as architectural retiming [11] can be used whereby negative registers are inserted into the path. These registers attempt to predict the cost computations for the next clock cycle before they have actually been made available by the current cycle. The predictions can then be compared to an actual result after it has been computed. If they are wrong, some form of state restoration is required.

A single additional pipelining register, for example, can be inserted within *PMIN* at the output of parallel adder blocks. (*i.e.* at signals *cost00*, *cost01* and *cost10* in Figure 3.14a). A naïve approach to computing cost data entering *PBUF* would then involve

feeding one of *cost00*, *cost01* or *cost10* back into the buffer as a predictor for the actual cost. This predictor is a static predictor, *i.e.* it does not adapt to choose the signal most likely to represent the actual cost. One cycle later, once the actual minimum of the three costs is computed, a comparison to determine whether the predictor matches the minimum is required. In the event that they do not match a penalty of two clock cycles is incurred as a result of the mis-prediction (*i.e.* one clock cycle wasted as a result of the actual prediction and a second clock cycle required for restoring the pipeline register states). Using this simple prediction scheme on the Tsukuba image set, experiments demonstrate that selecting *cost00* as a predictor would yield a match (or hit) with the actual cost about 45% of the time. Likewise, using *cost01* this hit rate would stand at 51% and using *cost10* the hit rate would be 43%. Clearly, such low rates of successful prediction do not provide any real benefit given a two clock cycle penalty. Improvements in these rates may be achievable by utilizing statistical properties of the cost matrix for dynamic cost prediction. For example, a scheme that uses past cost computation patterns for future selection of *cost00*, *cost01* or *cost10* could produce far better hit rates.

Much more significant improvements can be made in both logic and routing delays via ASIC implementations of the same hardware at the cost of development and prototyping time. Some FPGA systems, such as the Xilinx Virtex 5, contain DSP blocks for adder units that, if utilized, have demonstrated resulting clocking frequencies in excess of 125 Mhz for the DPMLHW(P) implementation, thus achieving the performance of the hypothetical DPMLHW(P)* implementation.

$$FPS_{P**} = \left[\frac{2N + \left(2N + \frac{D_{max}}{2} - 1\right) M}{F_{max\_clk}} \right]^{-1} \tag{4.7}$$

To conclude, by interleaving backward and forward passes a significant speed up is also possible. As discussed in Chapter 3 this interleaving behaves much like pipelining.

As data for the current scanline is being processed by the forward pass, the backward pass data continues and generates disparity estimates for the previous scanline. The difficulty in this form of interleaving lies in the fact that forward and backward passes utilize the same match matrix, *MBUF*. The problem can be solved by implementing a double buffering scheme whereby the forward pass and backward pass write and read from different match matrices — each match matrix contains data associated with either the current scanline or the previous scanline. Equation 4.7 provides a formulation for the worst case timing performance of such an interleaved architecture. The DPMLHW(P)** entry listed in Table 4.2 demonstrates that a significant improvement can be made to the DPMLHW(P) implementation with this interleaving. Disparity estimates can be generated at a frame rate of 123.85 fps for a maximum disparity range $D_{max} = 128$ pixels and a resolution of 640×480 pixels.

4.3 Resource Utilization

As discussed in Chapter 2, hardware resources for an FPGA can be broken down into logic cells and DSP blocks. Despite the intensive nature of the dynamic programming problem, the utilization of these resources remains fairly low. Note that, internally, logic cells are divided further into a series of multiplexers (MUXs), look-up tables (LUTs) and flip flops (FFs) that can be interconnected to form larger logic such as random access memory (RAM), multipliers (MULTs), adders (ADD), comparators (CMP) or more. With respect to both ASIC and FPGA implementations breaking down hardware into these logical elements provides a rough estimate of resource utilization. Table 4.5 presents a summary of higher level logical blocks utilized by the DPMLHW hardware for a pixel resolution of 640×480 and a disparity range of $D_{max} = 128$ pixels.

The large chunk of adders, comparators, multiplexers and multipliers presented in Ta-

Algorithm	MUX	RAM	MULT	ADD	CNT	REG	ENC	CMP	XOR
DPMLHW(P)	4113	333	66	394	5	183	130	788	396
DPMLHW(PP)	4108	269	65	394	5	81	130	788	369
DPMLHW(S)	26	8395	5	21	9	99	2	34	20

Table 4.5: A quantative look at the number of higher level logical blocks required for an implementation of DPMLHW for a Xilinx XC2VP100 device. Resources are divided into multiplexers (MUX), random access memories (RAM), multipliers (MULT), adders (ADD), counters (CNT), registers (REG), encoders (ENC), comparators (CMP) and xor gates (XOR).

ble 4.5 are synthesized for the parallelized circuitry in *PMIN*, *PNOC* and *CMUX*. Table 4.6 shows logic cell and DSP block utilization on a Xilinx XC2VP100 Virtext 2 pro device for the DPMLHW(S), DPMLHW(PP) and DPMLHW(P) hardware implementations at a resolution of 640×480 and disparity range of 128 pixels. For DPMLHW(P), the fastest of the three hardware implementations, at a resolution of 640×480 and $D_{max} = 16$, logic cell utilization stands at: 4% of the FPGA resources. At $D = 128$, this utilization increases to 32% of the FPGA resources. Similarly, at $D = 16$ the utilization of Block RAM and Multiplier DSP blocks stands at: 8% and 2%, of total FPGA resources, respectively. At $D = 128$ this utilization increases to: 58% and 14% respectively.

4.3.1 Improving Resource Utilization

As discussed earlier, routing delay was a significant component contributing to slow processing speeds. It is, therefore, worth noting that the large number of parallelized logic units present in *PMIN* and *CMUX* result in congestion at the pipeline and hence longer routes to sub-units at the periphery of the component. Some FPGA's contain additional DSP blocks for adders and comparators that can alleviate the congestion while also reducing logic delay — these blocks are, typically, optimized ASIC implementations embedded within the FPGA. An example of one such FPGA is the Xilinx Virtex 5. By utilizing the DSP blocks present in such an FPGA, circuit performance could be improved to a more desirable speed of 99 fps (as shown by DPMLHW(P)* in Table 4.2). Likewise

Algorithm	Resource	Type	Used	Available	% Util.
DPMLHW(P)	Slices	—	14448	44096	32%
	LUT/MUX	Logic Cell	20314	88192	23%
	FF	Logic Cell	8750	88192	9%
	RAM	Logic Cell	72	88192	0%
	RAM	Block RAM	260	444	58%
	MULT	Block Mult.	65	444	14%
DPMLHW(PP)	Slices	—	11325	44096	25%
	LUT/MUX	Logic Cell	21409	88192	24%
	FF	Logic Cell	3378	88192	3%
	RAM	Logic Cell	0	88192	0%
	RAM	Block RAM	269	444	60%
	MULT	Block Mult.	65	444	14%
DPMLHW(S)	Slices	—	39646	44096	89%
	LUT/MUX	Logic Cell	11271	88192	12%
	FF	Logic Cell	585	88192	0%
	RAM	Logic Cell	8384	88192	9%
	RAM	Block RAM	2	444	0%
	MULT	Block Mult.	1	444	0%

Table 4.6: Resource utilization resulting from an implementation of DPMLHW for a Xilinx XC2VP100 device. Resources are divided into multiplexers (MUXs), look-up tables (LUTs), flip flops (FFs), memories (RAMs) and multipliers (MULTs).

logic cell utilization can also be decreased further in this manner.

Also discussed earlier was a double buffering scheme for improved timing performance. Such a scheme requires the duplication of the match matrix, *MBUF*, resulting in an increased RAM utilization. By implementing an alternate approach to the same forward and backward pass interleaving whereby forward and backward passes write and read data from the match matrix in reverse directions, it is possible to maintain a single RAM module for *MBUF*. Further details may be found in Chapter 3, Section 3.2.9.

4.4 Summary

The accuracy, timing characteristics and resource consumption in the hardware (DPMLHW) implementation were compared to existing software and hardware algorithms. The comparisons demonstrated that the fully parallelized hardware implementation, DPMLHW(P) demonstrated equivalent and comparable accuracy to the best of existing stereo correspondence algorithms when presented with standard stereo datasets. However, testing on real world data demonstrated that uncorrelated noise in the input data has significant impact on the quality of 3D reconstruction. It was noted that pre-filtering could improve results in such situations. Many existing algorithms and hardware implementations also pre-filter incoming data to improve the quality of correspondence results.

Furthermore, hardware routing and logic complexity was shown to have significant impact on frame rates. Delays introduced by signal routing, particularly, reduce the maximum clock frequency in unpredictable ways, and, in-turn, slow down frame processing. DPMLHW(P)* introduced methods for optimizing these delay paths to produce significantly higher clock frequencies. Likewise, the DPMLHW(P)** implementation demonstrated that interleaving of backward and forward passes produces vastly superior frame rates (*i.e.* 123.85 fps for $D_{max} = 128$ pixels and resolution 640×480 pixels) in comparison to other existing algorithms while still maintaining a reasonably low resource footprint.

Chapter 5

Conclusion and Future Work

This thesis presents a novel hardware implementation, DPMLHW, of a DPML based stereo correspondence algorithm [4] — the first known hardware implementation of a dynamic programming disparity estimation solution. The DPMLHW implementation makes use of parallelization by exploiting anti-diagonal structure of cost matrices. This structure allows the computation of all costs along an anti-diagonal in parallel (within a clock cycle) using stored cost values from two previous anti-diagonals. Furthermore, throughput is increased by noting that the long combinational paths can be pipelined.

The architecture demonstrates that very high frame rate disparity and depth estimates can be achieved while also maintaining high degrees of accuracy. While other competing algorithms are known to produce higher accuracy (*e.g.* graph cuts, belief propagation, *etc.*), they are extremely computationally complex (slow) and thus less suited for hardware implementation. Likewise, faster SAD and phase correlation algorithms, while hardware friendly, have typically shown architectures that produce frame rates of approximately 30 fps after implementation. The DPMLHW(P) approach has demonstrated frame rates of 63 fps on the Xilinx Virtex 2 pro FPGA and 100 fps on the Xilinx Virtex 5 FPGA. These frame rates are generated at high resolutions (640×480 pixels) and high disparity ranges of 128 pixels. At the expense of resolution and dispar-

ity range, depth estimates can be produced at even faster speeds of 248 fps thus vastly outperforming existing software and hardware methods.

As in other techniques, accuracy drops off quite rapidly in the presence of noisy input data. Improvements in this accuracy can be achieved by Gaussian or median pre-filtering of the aforementioned input. A lower noise camera system can also result in marked improvements by reducing noise during image acquisition. A future modification, of the DPMLHW algorithm, whereby costs are aggregated over a correlation window shows potential for further improved performance in the presence of noise. This windowing could employ hardware adder trees for fast cost computation. Uneven illumination or exposure across cameras in a stereo setup can cause significant loss in accuracy. Localized image normalization, combined with windowing, may improve accuracy. Other techniques that also model specular reflectance properties and transparency could also improve results.

Further speed increases are possible if the DPMLHW architecture is modified to accomodate additional pipelines between the *PBUF* and *CBUF* registers. These pipelines, however, require a method to predict cost values, for subsequent cycles, before the data in the current cycle has reached the end of the pipeline. A method known as architectural retiming [11] is proposed for these future speed optimizations. Additional speed improvements are possible by utilizing FPGA DSP blocks (as shown by the 100 fps performance on the Virtex 5 FPGA). Interleaving backward and forward passes associated with the DPMLHW solution also promises frame rates of 123 fps for 640×480 pixel images at 128 pixel disparity range — a significant increase in performance. This interleaving can be implemented by introducing a double buffering scheme for reads and writes to the match matrix. Furthermore it is noted that ASIC implementations will provide far greater speed increase at the cost of development time and financial resources.

Resource consumption for the DPMLHW implementation remains reasonably low with logic utilization standing at 32% of the Virtex 2 pro FPGA. Approximately 4113 multiplexers, 333 RAMs, 66 Multipliers, 394 adders, 5 counters, 183 registers, 130 en-

coders and 788 comparators formed the bulk of these resources. Problems resulting from congestion, during parallelization, are a key source of signal delays and hence performance limitations — future ASIC implementation can allow for reduced component surface area to alleviate congestion. FPGA DSP blocks (*e.g.* in the Virtex 5 FPGA) utilized to this end have shown marked performance improvements.

Bibliography

[1] Michael Belshaw. Personal Communication. Design document, Queens University, December 2007.

[2] Myron Z. Brown, Darius Burschka, and Gregory D. Hager. Advances in Computational Stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(8):993–1008, August 2003.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press/McGraw-Hill, 2001.

[4] Ingemar J. Cox, Sunita L. Hingorani, Satish B. Rao, and Bruce M. Maggs. A Maximum Likelihood Stereo Algorithm. *Computer Vision and Image Understanding (CVIU)*, 63(3):542–567, May 1996.

[5] Ahmad Darabiha, W. James MacLean, and Jonathan Rose. Reconfigurable Hardware Implementation of a Phase-Correlation Stereo Algorithm. *Machine Vision and Applications*, 17(2):116–132, March 2006.

[6] Javier Diaz, Eduardo Ros, Silvio P. Sabatini, Fabio Solari, and Sonia Mota. A Phase-Based Stereo Vision System-On-A-Chip. *Biosystems*, 87(2–3):314–321, July 2006.

[7] Minglun Gong, Ruigang Yang, Liang Wang, and Mingwei Gong. A Performance Study on Different Cost Aggregation Approaches Used in Real-Time Stereo Match-

ing. *International Journal of Computer Vision (IJCV)*, 75(2):283–296, February 2007.

[8] Dongil Han and Dae-Hwan Hwang. A Novel Stereo Matching Method for Wide Disparity Range Detection. In *International Conference on Image Analysis and Recognition (ICIAR)*, pages 643–650, 2005.

[9] Masonori Hariyama, Yasuhiro Kobayashi, Haruka Sasaki, and Michitaka Kameyama. FPGA Implementation of a Stereo Matching Processor Based on Window-Parallel-and-Pixel-Parallel Architecture. In *Midwest Symposium on Circuits and Systems (MWSCAS)*, volume 2, pages 1219–1222, 2005.

[10] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[11] S. Hassoun and C. Ebeling. Architectural Retiming: An Overview. In *International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU)*, November 1995.

[12] Jamin Islam. Stereo Image Rectification Module. Design document, Ryerson University, 350 Victoria Street, Toronto, Ontario, Canada M5B2K3, September 2007.

[13] Ricardo P. Jacobi, Renato B. Cardoso, and Geovany A. Borges. VoC: A Reconfigurable Matrix for Stereo Vision Processing. In *International Parallel and Distributed Processing Symposium (IPDPS)*, April 2006.

[14] Yunde Jia, Xiaoxun Zhang, Mingxiang Li, and Luping An. A Miniature Stereo Vision Machine (MSVM-III) for Dense Disparity Mapping. In *Internationl Conference on Pattern Recognition (ICPR)*, 2004.

[15] Valeri Kirishchian. Personal Communication. Design document, Ryerson University, 350 Victoria Street, Toronto, Ontario, Canada M5B2K3, December 2007.

[16] Ye Lu, Jason Z. Zhang, Q. M. Jonathan Wu, and Ze-Nian Li. A Survey of Motion-Parallax-Based 3-D Reconstruction Algorithms. *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews*, 34(4), November 2004.

[17] W. James Maclean, Siraj Sabihuddin, and Jamin Islam. Leveraging Cost Matrix Structure for Dynamic Programming in Hardware. *Submitted for review: Computer Vision and Image Understanding (CVIU)*, June 2008.

[18] Divyang K. Masrani and W. James MacLean. A Real-Time Large Disparity Range Stereo-System using FPGAs. In *International Conference on Computer Vision Systems (ICVS)*, 2006.

[19] Johel Miteran, Jean-Philippe Zimmer, Michel Paindavoine, and Julien Dubois. Real-Time 3D Face Acquisition Using Reconfigurable Hybrid Architecture. *EURASIP Journal on Image and Video Processing*, December 2007.

[20] Yosuke Miyajima and Tsutomu Maruyama. A Real-Time Stereo Vision System with FPGA. In *International Conference on Field Programmable Logic and Applications (FPL)*, pages 448–457, 2003.

[21] Stefania Perri, Daniela Colonna, Paolo Zicari, and Pasquale Corsonello. SAD-Based Stereo Matching Circuit for FPGAs. In *International Conference on Electronics, Circuits and Systems (ICECS)*, pages 846–849, December 2006.

[22] Daniel Scharstein and Richard Szeliski. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *International Journal of Computer Vision (IJCV)*, 47(1–3):7–42, April 2002.

[23] Daniel Scharstein and Richard Szeliski. High Accuracy Stereo Depth Maps using Structured Light. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 195–202, June 2003.

[24] Daniel Scharstein and Richard Szeliski. Middlebury Stereo Vision Page, 2007.

[25] Steven M. Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 519–526, 2006.

[26] Vikram Simhadri, Premanand Chandramani, and Yusuf Ozturk. RASCor: Realtime Associative Stereo Correspondence. In *International Conference on Image Processing (ICIP)*, September 2007.

[27] M.F. Tappen and W.T. Freeman. Comparison of Graph Cuts with Belief Propagation for Stereo, using Identical MRF Parameters. In *IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 900–906, October 2003.

[28] Jan van der Horst, Rien van Leeuwen, Harry Broers, Richard Kleihorst, and Pieter Jonker. A Real-Time Stereo SmartCam, using FPGA, SIMD and VLIW. In *Workshop on Applications of Computer Vision*, May 2006.

[29] John Iselin Woodfill, Gaile Gordon, and Ron Buck. Tyzx DeepSea High Speed Stereo Vision System. In *Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, volume 3, pages 41–46, 2004.

[30] Barbara Zitova and Jan Flusser. Image Registration Methods: A Survey. *Image and Vision Computing*, 21:977–1000, June 2003.