Parallelization of Belief Propagation Method on Embedded Multicore Processors for Stereo Vision

Chi-Hua Lai, Kun-Yuan Hsieh, Shang-Hon Lai, and Jenq Kuen Lee
Department of Computer Science
National Tsing-Hua University, Hsin-Chu, Taiwan
{chlai, kyshieh}@pllab.cs.nthu.edu.tw, {lai, jklee}@cs.nthu.edu.tw

Abstract

Markov random field models provide a robust formulation of low-level vision problems. Among the problems, stereo vision remains the most investigated field. The belief propagation provides accurate result in stereo vision problems, however, the algorithm remains slow for practical use. In this paper we examine and extract the parallelisms in the belief propagation method for stereo vision on multicore processors. The results show that with parallelization exploration on multicore processors, the belief propagation algorithm can have a 13.5 times speedup compared to the single processor implementation. The experimental results also indicate that the parallelized belief propagation algorithm on multicore processors is able to provide a frame rate in 6 frames per second.

1 Introduction

Stereo vision has been extensively investigated in the recent years to provide high-quality and high performance applications. One of the most advanced paradigms is to apply the technique in inferring the 3-D position of an object. By computing the depth and disparity of referenced images through matching the images in the same plane of different view position, the range information of the environment can be extracted to help the robots to adapt to the real world. Over the past years, Markov random field (MRF) models have been used to develop algorithms with good results to solve the problems of stereo vision. However, to apply the MRF models on the stereo matching problem is NP-hard. One of the most significant approximate algorithms is belief propagation (BP) that gathers information from the neighborhoods of each pixel in an image to find the minimum matching cost of the local point and its neighborhoods [1, 2]. Although having highly accurate results with outstanding quality, BP requires a long processing time that makes it less practicable in application domains that require a real-time performance. There are research proposed to optimize BP algorithm to provide a real-time performance on GPU [3]. And also some research target to the parallel of loopy belief propagation on the clusters or multiprocessors [4]. However, the problem remains significant in other architectures.

As the system-on-chip (SoC) technique has been toward the multicore architectures in the recent years, the industries have adopted such novel architecture in products such as, video game machines, personal hand-held devices, home-media center, and devices required high computation power. Such a diverse appliance of multicore processors in variety of application domains provides opportunities to optimize the stereo vision algorithms. In this paper, we examine the performance improvement of BP algorithm on the Cell broadband engine (Cell BE) [5] which is a multicore processor containing a power processing element (PPE) and eight synergistic processing elements (SPE). Cell BE provides a highly parallel architecture with pervasively data-parallel computing mechanism based on the SIMD computing and high performance data transferring management. Such architecture features benefit the stereo vision applications since the algorithms are computation-intensive and conventionally contain high data-parallelism. The results show that with parallelization exploration on multicore processors, the belief propagation algorithm can have a 13.5 times speedup compared to the single processor implementation. The experimental results also indicate that the parallelized belief propagation algorithm on multicore processors is able to provide frame rate in 6 frames per seconds.

This paper is structured as follows. Section 2 provides an overview of the BP algorithm. Section 3 first describes the overview of the Cell BE architecture then proposes the strategies of parallelizing the BP algorithm. The implementation with parallelization extraction of BP on multicore processors is also illustrated in Section 3. Section 4 shows the experimental results of the parallelized BP on Cell BE. Section 5 discusses the future work. Finally, Section 6 concludes this article.

2 Overview of the Belief Propagation Algorithm

Belief propagation (BP) is an efficient method for solving early vision problems. By gathering and incorporating

the information from each pixel's neighbors, the algorithm iteratively updates and optimizes the information to find the best solution. In this paper, we examine and extract the parallelism in the BP algorithm by modifying the algorithm presented by Felzenszwalb and Huttenlocher [6]. The process of the BP for stereo matching is to minimize the energy which is the quality of labeling the disparity to each pixel of the image to be matched [1]. The estimation of the energy includes the process of computing two different costs: the discontinuity cost of labeling the disparity to two adjacent pixels and the data cost of assigning the disparity label to each pixel. If two adjacent pixels are assigned to different labels, the discontinuity cost is increased. Different disparity costs of labeling the pixel produces different data costs. Each pixel in the image is associated to a vector data structure with levels of disparity called the node to store the information required for minimizing the energy of the image including the disparity cost and data cost.

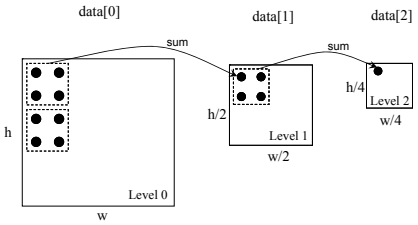


Figure 1. Initialization process of the data pyramid.

Algorithm 1 shows the BP method for stereo matching. The BP method starts from Gaussian filtering two rectified gray-level pictures of different view points, G and G'. Then it measures the data cost by subtracting the images to compute the difference of G and G'. The data cost can be alternatively generated by using the sum of squared difference of G and G' without Gaussian filtering. Then the data cost of each pixel in the image is collected and stored in the data layer, $data^0_{w_0,h_0}$ of width w_0 and height h_0. After the data layer is computed, it then builds the data pyramid from $data^0_{w_0,h_0}$ to form a fine-to-coarse multi-scale matching scheme. The data pyramid is generated by building a hierarchy of data layers from $data^0_{w_0,h_0}$. The data layer i, $data^i_{w_i,h_i}$, of the data pyramid is first formed from the finer data layer $i-1$ by by halving the width and height. In other words, $w_i = \frac{1}{2}w_{i-1}$ and $h_i = \frac{1}{2}h_{i-1}$. Then the value of each nodes of data layer i is calculated by the summing values of adjacent four nodes in finer data layer $i-1$. Figure 1 shows the process of initializing the data pyramid.

Four message layers, up_i, $down_i$, $right_i$, and $left_i$, are associated to each data layer i to indicate the directions where each node sends messages to for estimating the discontinuity cost. Then the method performs BP for T iterations in each level. T is a constant determined by the developers. Greater T produced better quality. It starts at the coarsest layer through gathering and updating the messages

Algorithm 1: BP algorithm for stereo matching

Data: Two rectified gray-level pictures, the left side vision G and the right side vision G'
Data: Distance of passing message in advance, L
Data: Iteration for message updating, T
Result: Disparity graph D

$data^0_{w_0,h_0} \Leftarrow$ pixel by pixel difference between G and G' with width w_0 and height h_0 ;
Initializing data pyramid $data^i_{w_i,h_i}$, $i = 1, \cdots,$L -1
for $i \Leftarrow$ L $- 1$ *to* 0 **do**
  if *not in the top level* **then**
    Get message from the upper level message layers;
  else
    Initialize top level message layer to 0;
  end
  for $t \Leftarrow 0$ *to* T $- 1$ **do**
    for $y \Leftarrow 1$ *to* $h - 1$ **do**
      for $x \Leftarrow (y + t)$ *mod* 2 *to* $w - 1$ **do**
        Update upward-message of node(x, y);
        Update downward-message of node(x, y);
        Update leftward-message of node(x, y);
        Update rightward-message of node(x, y);
        $x = x + 2$;
      end
    end
  end
end
for $y \Leftarrow 1$ *to* $h - 1$ **do**
  for $x \Leftarrow 1$ *to* $w - 1$ **do**
    Accumulate messages delivered by adjacent nodes and compute the disparity, D(x,y);
  end
end

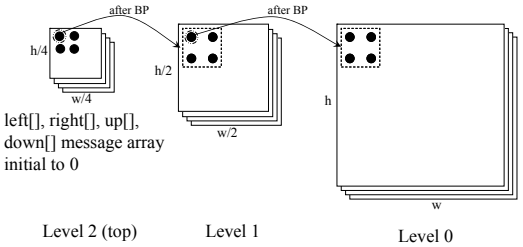


Figure 2. Message updating from coarse to fine.

from each node's neighbors and the data cost of the node. The messages produced at the coarser layer i is then used to initialize the message layers of finer layer $i-1$ to iteratively performs the BP from the coarsest layer to the finest layer. The hierarchical message updating process is illustrated in

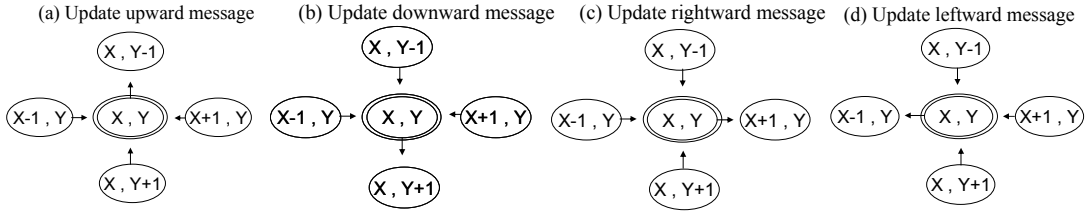


Figure 3. Updating four-direction messages of node (X,Y). The arrows point to the node (X,Y) is the messages send to (X,Y) from neighboring nodes.

Figure 2. Since the image has been built into a data pyramid, it reduces the number of nodes in the coarsest layer. By performing such coarse-to-fine scheme, the finer message layers are initialized to a more possible minimal cost. It is thus helps the algorithm to get convergence rapidly. Such paradigm reduces the processing time of doing the BP on the coarser layer i to update the message [6].

Figure 3 illustrates the four-direction message updating. A node at corresponding position (X, Y) of each data layer i and message layer is associated with four message vectors stored in the message layer i. For example, the message vector of node at (X, Y) for the direction up is denoted as $up_i^t(X, Y)$ to present the message that node (X, Y) passes to the upward node $(X, Y - 1)$ at the ith message layer in the t iteration. Figure 3 (a) depicts the process. The message $up_i^t(X, Y)$ is generated by gathering the messages from the other three directions, $right_i^{t-1}(X - 1, Y)$, $up_i^{t-1}(X, Y + 1)$, $left_i^{t-1}(X + 1, Y)$, provided by adjacent nodes from the last iteration $t - 1$ and the data cost, $data^i_{w_i,h_i}(X, Y)$, of the node (X, Y) at the ith level. After performing the BP in each layer of the image pyramid for T times, the messages are then accumulated from the adjacent nodes of node (X, Y) along with the data cost to compute the disparity graph in position (X, Y) at the finest level.

3 Parallelizing the Belief Propagation on Multicore Processors

3.1 Cell BE Architecture Overview

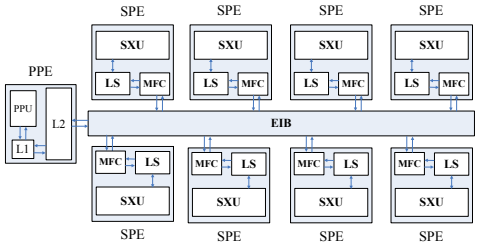


Figure 4. Cell broadband engine architecture.

In this section, we describe the architecture overview of the Cell BE [5] which is a high performance platform for application domains of financial, networking, scientific computing, and high-resolution multimedia. Figure 4 depicts the architecture overview. The Cell BE is composed of one PPE and eight SPEs connected by a high bandwidth on-chip bus called the element interconnect bus (EIB). As a heterogeneous multicore processor, the operating system and applications are running on the PPE while computation-intensive tasks are explicitly dispatched to execute on SPEs by the developers. Such an architectural feature leads to the adoption of master-slave execution model in our implementation of parallelized BP method which partitions the algorithm into two parts: control tasks, and computational tasks. The control tasks are scheduled by the Linux to be executed on PPE while the computational tasks are explicitly invoked and scheduled by the control tasks and the operating system to be run on SPE.

The SPE is a RISC architecture with registers of 128 bits wide which supports SIMD arithmetic instructions. The SPE can only access its local store (LS), which is the main storage of each SPE. To exchange data between the main memory of PPE and LS of other SPEs, the developers use direct memory access (DMA) to explicitly control the data flow. The synchronization, event communication, and data transmission are managed by the memory flow controller (MFC) associated to each SPE. The SPE issues DMA commands to the associated MFC to perform efficient data transmission that is concurrently executed with the computation. Such mechanism is a key factor to gain performance by supporting efficient implementation of multiple buffering through overlapping communication and computation in the parallelized BP. Moreover, the data reusability of the partitioned method is important since the data movement from main memory to LS of each SPE produces extra overhead.

In addition to the efficient data transmission, the Cell BE provides various communication mechanisms to support message passing between processors. Each SPE is associated with one input mailbox with four 32 bits message entries and two output mailbox with one message entry. Writing the input mailbox of an SPE transmits a 32 bits data to the message entry. The output mailbox allows the SPE to send small message in the output mailbox, other cores or devices requiring the message is able to access the output mailbox and empty it after reading the message. The SPE is allowed to trigger an interrupt to the PPE along with a 32 bits message by writing the message to the associated output mailbox entry. The mailbox mechanism is adopted by the proposed parallelized BP to reduce the communication overhead. Section 3.2 details the strategies of paral-

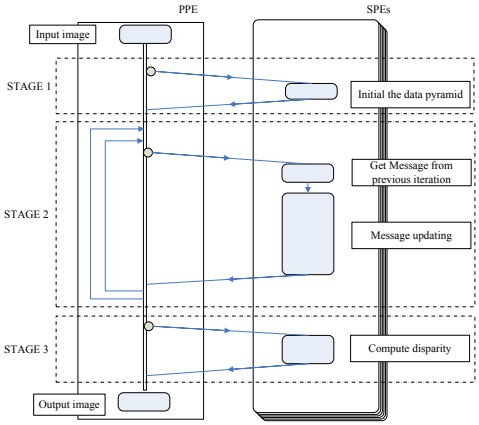


Figure 5. Execution flow of the BP on Cell BE.

lelizing BP method on the Cell BE.

3.2 Parallelization Strategies

In this section, we analyze the parallelism in the BP algorithm proposed in Section 2 and then show the strategies of implementing parallelized BP on the Cell BE. Figure 5 shows the execution flow of BP algorithm that can be divided into three stages: initializing the image pyramid, message updating, and computing the disparity graph. The analysis and implementation strategies are detailed as follows.

3.2.1 Stage 1: initializing the data pyramid

In this stage, the initialization of data pyramid is performed by grouping the adjacent four nodes in $data^0_{w_0,h_0}$ to form the coarser image layer. Considering the generalization of each node in the coarsest data layer, the computation of each node can be performed independently as shown in Figure 6. The initialization from $data^0_{w_0,h_0}$ can thus be partitioned into independent parts. The data parallelism in this stage provides opportunities to optimize the algorithm. To expose the architecture parallelism, assume that there are L data layers in the image pyramid, we first partition the input data $data^0_{w_0,h_0}$ into $\frac{w_0 * h_0}{2^{2*L-1}}$ divisions. Then the divisions are grouped into groups of data to be initialized in the SPE of the Cell BE: Each division in the SPE can be process to form the image pyramid from finer to coarser image layer without synchronizing with other divisions.

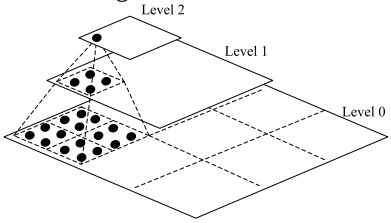


Figure 6. Building the data pyramid from lowest level data layer.

3.2.2 Stage 2: message updating

In this stage, the message updating in each iteration are performed in four directions by referring to the message computed in the last iteration. The process implies that every message updating operation of each iteration can be performed independently. Thus the four-direction updating can be executed independently in each SPE by partitioning the up, down, left, right direction updating to be execute in SPE1, SPE2, SPE3, SPE4, respectively. Although the partitioning provides a way of parallelizing the message passing stage, the data locality reduces the performance gain of task parallelism. The messages fetched from main memory for updating one direction are not reusable for the updating of the next node in the same direction. The SPE has to fetch the messages required for each node.

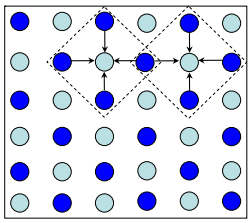


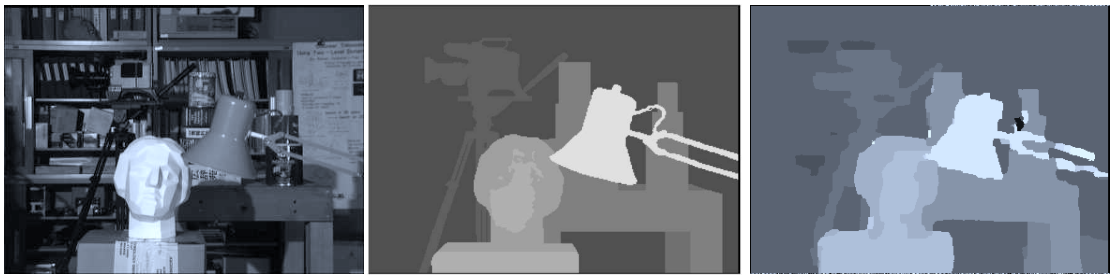
Figure 7. Bipartite graph.

Therefore, we propose another strategy to parallelize this stage. As shown in Algorithm 1, the updating process are not performed for every node in each iteration but in a bipartite behavior. Take iteration t for example, the four directions of light-color nodes are updated as shown in Figure 7. Then in the next iteration of $t+1$, it only updated the message of each dark-color nodes. The bipartite behavior of message updating gives hints of parallelizing this stage. Instead of partitioning one direction of each node to be executed concurrently, we can update the four-direction messages of each node independently since the messages required are computed in the last iteration. Moreover, it provides high data reusability for the reason that the messages required for updating each direction are the same vectors.

Thus, the nodes are partitioned into several groups according to the numbers of SPE available. Each group is scheduled to be processed in an SPE. After dispatching the jobs and data in each SPE, the four-direction messages of each node in a group are calculated independently in each iteration. The computed messages are then transmitted back to the main memory for the message updating of the next iteration. The strategy is able to provide good data and task parallelism with good data reusability.

3.2.3 Stage 3: computing the disparity graph

The last stage of the algorithm is to gather the information computed from the prior two stages to produce the disparity graph. The disparity of each node is computed by gathering the messages of the neighboring nodes toward this node and the information from $data^0_{w_0,h_0}$ to decide the proper disparity. The disparity computation process of each node can be performed in parallel. Therefore, this stage also shows high data parallelism that allows the decision process of each node to be performed in the SPE.



(a) Original image pair. (b) Ground-truth. (b) Result of parallelized BP.

Figure 8. Stereo matching results for the Tsukuba image pair.

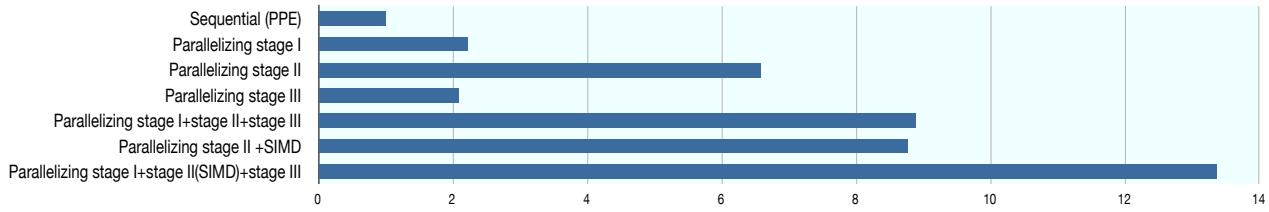


Figure 9. Performance improvement of parallelized BP.

3.2.4 SIMD

Analyzing the data layout and usage of the algorithm, most of the data structures used are vectors and arrays. The SPE of Cell BE provides a set of application interfaces for the developers to expose the architecture benefits of SIMD by exploiting the potential instruction-level data parallelism. Such feature provides another opportunity for optimization by packing the data accessing of the program into SIMD instructions.

4 Experimental Results

We performed the experiments on the PlayStation3 (PS3) platform which provides one 3.2 GHz Cell BE processor with six SPEs and one PPE. The PPE has a 32 KB instruction cache and a 32 KB data cache as L1 cache, and a 512 KB L2 unified cache for instruction and data. The implementation of the parallelized BP was based on the development environment where the PPE runs the Linux kernel 2.6.22. The toolkit provided by IBM includes the libspe for managing SPE, pthread library, compiler, linker, and assembler of PPE and SPE. The data and event communication were through the mailbox mechanism. Since the platform we used only provided six runnable SPEs, the data layer is partitioned to six groups. Our implementation processed five iterations for six disparity levels for the Tsukuba image pair. Figure 8 (a) depicts one of the original image pair. Figure 8 (c) shows the result produced by the parallelized BP. Comparing to the ground-truth of the image pair as shown in Figure 8 (b), the result of the parallelized BP presented a great quality.

The execution time of the parallelized BP method is 0.175 seconds which is about 13.5 times faster than the sequential implementation on PPE of the Cell BE. The result also indicates a 6 times faster than the implementation on 3 GHz Pentium4 computer. Table 1 shows the execution time of different implementation of BP for Tsukuda image pair. Note that the BP on single PPE is about two times slower than on Pentium 4. The main reason is that the BP algorithm requires a lot of memory for processing the images and updating the messages. However, the PS3 platform we used provided limited memory (256MB XDRAM) for the PPE which leads to a dramatically decreasing of the performance.

To demonstrate the performance improvement of the parallelization strategies illustrated in Section 3.2, we measured the performance improvement of each strategy. Figure 9 shows the performance improvement comparing to the sequential implementation on PPE of the Cell BE. As shown in Figure 9, by parallelizing stage I for the data pyramid initialization produced about two times performance improvement in execution time. Parallelizing stage III for producing the disparity graph also produced about two times performance improvement. As the BP method spent around 90% of the execution time in updating messages, parallelizing stage II for the message updating produced great performance improvement by 6.5 times faster. By merely parallelizing the control flow and data communication existed in the algorithm, the method was improved about 9 times

Table 1. Execution time of the different implementation of BP on different machines for Tsukuba image pair.

Method	Sequential	Sequential	Parallelized BP
Platform	3G Pentium 4	PPE on Cell BE	Cell BE
Performance(seconds)	1.195	2.34	0.175

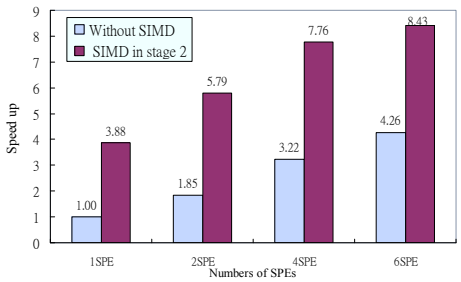


Figure 10. Scalability of the parallel BP algorithm using different numbers of SPEs.

faster comparing to the sequential implementation on PPE.

We also implemented the method with partial SIMD data accessing in the parallelized BP for stage II. The image processing and message updating in the implementation were converted to a set of SIMD operations on arrays and vectors. As shown in Figure 9, the performance was improved by 9 times after packing memory access operations into SIMD instructions. Figure 9 shows the result of parallelized BP after applying all the parallelization strategies which presents an overall 13.5 times performance improvement. The experimental results also shows that the stereo matching can have a nearly real-time performance of about 6 frames per second.

For evaluating the performance gain with difference numbers of SPEs, we consider the implementation of the parallelized BP on one SPE as the baseline. Figure 10 shows the scalability of the parallelized BP algorithm. The bright color bar shows the performance improvement of the implementations of parallelized BP on different numbers of SPEs. The dark color bar shows the performance scaling of parallelized BP with SIMD data accessing running on different numbers of SPEs compared to the implementation of single SPE without SIMD exploration. The performance was linearly improved as the numbers of SPE used were increased which shows a 4.26 times improvement compared to the single SPE implementation. With the four-word SIMD data accessing, the performance of the algorithm was improved by 3.88 times compared to the single SPE implementation. The implementation on six SPEs with SIMD was 8.48 times faster than that on one SPE. However, the speedup of parallelized BP on different numbers of SPEs with SIMD exploitation was not increased linearly for that the execution of I/O processing for image file opening and reading is around 0.09 seconds which is not able to be parallelized in our implementation.

Considering the case when applying the method in a real-time scenario where the image pairs are continuously captured from the devices for stereo matching. The overhead of I/O operations may be eliminated by pipelining the image reading and processing with the BP method. If not considering the overhead of file operations, the parallelized BP can produce 11 frames per second performance in our early evaluation on the Cell BE.

5 Future Work

Packing the array and vector operations to a set of SIMD operations is one of the key factors in tuning the Cell BE applications. It provides facilities to pack four 32-bits data accessing into a single SIMD128-bits data access, which potentially presents four times performance improvement for each data accessing. The early evaluation of our work only includes partial implementation of the SIMD data accessing in the parallelized BP. Analyzing the BP method, the image processing and message updating in the implementation were mainly array and vector structures. As shown in the experimental results, SIMD presented a important role in improving the overall performance. In the future work, packing SIMD instructions for all the memory operations could be a key factor to produce more performance improvement for supporting a real-time performance. Moreover, the adoption of intrinsic function for special operations and carefully choosing compiler optimization phases, such as loop unrolling, also reveal opportunities of further improving the performance.

6 Conclusion

In this paper we examined the parallelization of a belief propagation algorithm on the multicore processors. We have proposed methods to demonstrate the issues in optimizing the algorithm by exploiting the potential parallelism to expose the architecture benefits. The methodology of analyzing and exploiting parallelism presented in this article is applicable to other stereo vision algorithms. The results showed that with careful analysis and parallelizing, the implementation is able to produce a highly accurate result with fast processing time.

Acknowledgment

This research was supported in part by the NSC under grant nos. NSC 97-2218-E-007-009, NSC 97-2218-E-007-008 and NSC 96-2220-E-007-030, and by the MOEA research project under grant nos. 95-EC-17-A-01-S1-034 and 96-EC-17-A-01-S1-034 in Taiwan.

References

[1] Jain Sun, Nan-Ning Zheng, and Heung-Ywung Shum. Stereo matching using belief propagation. *IEEE Tansaction on Pattern Analysis and Machine Intelligence*, 25(7):787–800, 203.

[2] Jonathan S. Yedidia, William T. Freeman, and Yair Wiess. Constructing free-energy approximations and generalized belief propagation algorithms. *IEE Transaction on Information Theory*, 51(7):2282–2312, 2005.

[3] Alan Brunton, Chang Shu, and Gerhard Roth. Belief propagation on the GPU for stereo vision. In *Proceedings of the 3rd Canadian Conference on Computer and Robot Vision*, pages 76–81, June 2006.

[4] Mendiburu A., Santana R., Lozano J. A., and E. Bengoetxea. A parallel framework for loopy belief propagation. In *Proceedings of the 2007 GECCO Conference Companion on Genetic and Evolutionary Computation*, pages 2843–2850, July 2007.

[5] Michael Gschwind. The cell broadband engine: Exploiting multiple levels of parallelism in a chip multiprocessor. *International Journal of Parallel Processing*, 35(3):233–262, 2007.

[6] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient belief propagation for early vision. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 261–268, 2004.