Disparity Map Computation on a Cell Processor

Jiaju Liu[1], Yanyan Xu[1], Reinhard Klette[2], Hui Chen[1], and Tobi Vaudrey[2]

[1] Information and Engineering School, Shandong University, Jinan, China
[2] The *.enpeda..* Project, The University of Auckland, Auckland, New Zealand

Abstract. This report describes an efficient implementation of dynamic programming and belief propagation algorithms on a cell processor that may be used to speed up stereo image analysis. Dynamic programming is a method for efficiently solving optimization problems by caching subproblem solutions rather than recomputing them again. It processes image data by scanline optimization; thus it is easily implemented on a cell processor; our results show that this algorithm runs very efficiently on the cell processor. Belief propagation differs from dynamic programming by having potentially the whole image area as an area of influence for every pixel; this global optimization scheme produces improved results, but requires more run time than the dynamic programming method on a normal PC. Besides the tests on synthetic data, we use real-world image sequences captured by a test vehicle (HAKA1); they are typically degraded by various types of noise, changes in lighting, differing exposures, and so on. We use two methods to process the original images: Sobel edge detection and residual image analysis. Our results show that a cell processor also reduces running time for these processes. Sobel and residual images can improve the stereo matching result compared to the use of original real-world images, however, due to the used block structure and cell architecture limitations, the accuracy is also degraded slightly.

1 Introduction

Stereo vision has been extensively investigated in recent years for high-quality and high performance applications. One of the most advanced paradigms is to apply the technique in inferring the 3D position of multiple objects in stereo sequences recorded outdoors. The computation of a depth (or disparity) map of referenced images is achieved by matching positions in both images, recorded in the same (rectified) plane but at different viewing positions. The inferred range information about the environment may help robots or intelligent vehicles to adapt to the real world. Stereo matching is a computationally demanding task. Previous work on stereo matching computation is mainly limited to software-based techniques on general-purpose architectures.

In this report, we examine performance improvements of two stereo matching algorithms on a specialized computer, in comparison to standard PC technology. For this study, we have selected a *belief propagation* (BP) algorithm as published in [4], and a *dynamic programming* (DP) algorithm as published in [11]. The specialized computer is a cell broadband engine (Cell BE) which is a multi-core

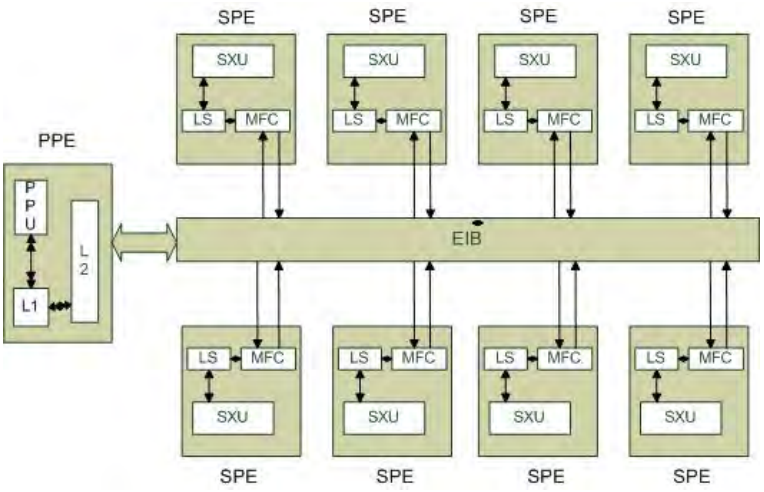

Fig. 1. Architecture of Cell BE.

processor containing a power processing element (PPE) and eight synergistic processing elements (SPE). For the traditional way of implementation, we use a normal PC (Dual Core 2.13 GHz CPU, 3 GB RAM) to implement DP and BP; the time taken for one stereo pair was typically around 6 s and 32 s, respectively. This means that DP or BP stereo vision could not be used for real time systems, such as driver assistance systems (DAS).

The Cell BE provides a highly parallel architecture with a pervasively parallel computing mechanism based on a SIMD computing architecture and high performance data transfer management. Both features are of benefit for stereo vision applications since these algorithms are computation-intensive and conventionally contain inherent data-parallelism at a high degree. Our results show that with utilizing parallelization on multi-core processors, the DP and BP algorithms are significantly sped-up compared to the traditional implementation.

This report is structured as follows. Section 2 provides an overview of the Cell BE architecture. Section 3 outlines the test data. Section 4 describes the strategies of parallelizing the DP and BP algorithms. Section 5 shows the experimental results of parallelized DP and BP on Cell BE. Finally, Section 6 concludes this article.

2 Cell BE

Compared to implementations on the used normal PC (see above), the DP or BP implementation on the specialized cell processor has to take the processor's architecture into account. In case of BP we face a particular challenge. Figure 1 shows the overall structure of the cell processor.

Here we present the implementation of the Cell, used in the Playstation 3 (PS3). It is basically a chip with one *power pc* PPC hyper-threaded core called *powerpc processor element* PPE and eight specialized cores called *synergistic processor element* SPEs. The challenge to be solved by the cell was to put all those cores together on a single chip. This was made possible by the use of a bus with outstanding performance. One can have an idea of the organization of a cell chip in Figure 1. It shows that 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). The PPE consists of the *power processing unit* PPU, 512 KB 8-way write-back cache (L1) and 32 KB 2-way reload-on-error instruction cache (L2). SPE includes local store memory (LS), memory flow controller (MFC) and Synergistic eXecution Unit (SXU). Besides this, Figure 1 also shows how the Cell is communicating with the rest of the world, through its memory controllers and input/output controllers.

2.1 The PPE

The PowerPC Processor Element, usually denoted as PPE, is a dual-threaded PowerPC processor version 2·02. This 64-bit RISC processor also has the Vector/SIMD Multimedia Extension. Any program written for a common PowerPC 970 processor should run on the Cell Broadband Engine. Figure 2 shows a very simplified view of the PPE. The PPE consists of the Power Processing Unit (PPU) and a unified (instruction and data) 512 KB 8-way set associative write-back cache. The PPU includes a 32 KB 2-way set associative reload-on-error instruction cache and a 32 KB 4-way set associative write-through data cache. The PPE is often used to control thread. The PPE role is crucial in the cell architecture since it is on the one hand running the OS, and on the other hand controlling all other resources, including the SPEs [9].

2.2 The SPE

Figure 3 shows the overall structure of the SPE. The real power of the cell processor does not lie in the PPE, but the other cores (SPEs). Each cell chip has 8 Synergistic Processor Elements, usually referred to as the SPEs. They are 128-bit RISC processor which are specialized for data-rich, computation-intensive SIMD applications. They consist of two main units. Figure 3 shows a very simplified view of the SPE. The SPE can only access its local store memory (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 a mechanism is a key factor to gain performance by supporting efficient implementation of multiple buffering through overlapping communication and computation in the parallelized DP and BP. Moreover, the data reusability of the

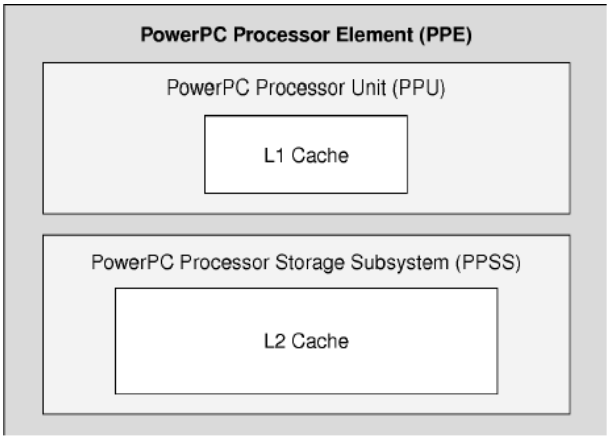

Fig. 2. PPE block diagram

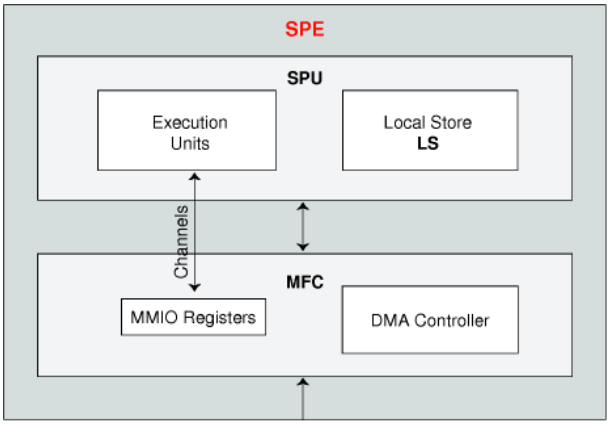

Fig. 3. SPE block diagram.

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 mailboxes with one message entry. Writing the input mailbox of an SPE transmits 32 bits of data to the message entry. The output mailbox allows the SPE to send small messages to 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

Fig. 4. Top row: *Tsukuba* stereo pair. Bottom row: Real image sequence recorded with cameras in HAKA1.

PPE along with a 32 bit message by writing the message to the associated output mailbox entry. The mailbox mechanism is adopted by the proposed parallelized DP and BP to reduce the communication overhead [9].

3 Test Data

We use test images from the Middlebury stereo website (image size 384×288) [12]. This demonstrates the application on synthetic data. We also use a real-world image sequence captured by HAKA1 (test vehicle) in New Zealand (image size 640×480). This contains real world issues, such as lighting difficulties, blurring, specular reflections, internal reflections, and so forth. See Figure 4 for example images for comparison.

4 Parallellization Strategies on Cell

In this section, we analyze the parallelism in the DP and BP algorithm and show the strategies of implementing parallelized DP and BP in the Cell BE. Figures 5 and 8 show the results of the experiments.

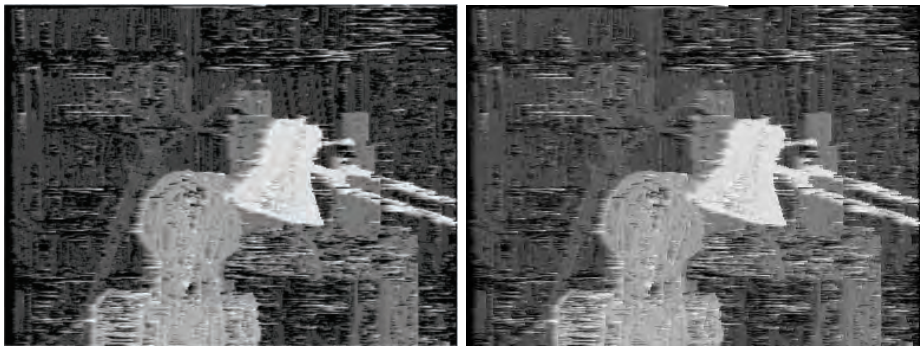

Fig. 5. Left: disparity image by DP on cell. Right: disparity image by DPs on cell.

4.1 Implementation of Dynamic Programming

Program Model. Dynamic programming processes images by scanline, so it is suited to be implemented on a cell processor. We implemented as follows:

1. PPE starts the SPE threads.
2. PPE loads images from file to main memory then sends address to SPE by mailbox.
3. SPE loads unprocessed rows of images to local memory by DMA,
   - computes disparity on local data, and
   - stores results back to main memory.
4. PPE writes disparity map to file.

When implementing we used a double buffer to reduce time. It works a lot like the doctor's office, we know that when we get there we are going to spend a lot of time in the waiting room. Therefore, we always bring something else to do while we are waiting. The same principle applies to programming. Rather than waste good processor cycles waiting around for data to transmit, your code can instead have a second buffer waiting to process. Therefore, while you wait for one set of data to transmit, you can be processing another. The results show that the use of a double buffer shows a clear speed-up in computation.

Dynamic programming tends to propagate errors along the scanline, resulting in horizontal streaks in the calculated depth map. To reduce inter-scanline inconsistencies, disparities in adjacent scanlines may be used to define a discontinuity term in the energy function.

We consider a simple spatial propagation (DPs) method [10]. Let f^y and f^{y-1} denote calculated disparity functions on current and previous scanlines. Given a weighting factor λ_1, the adjusted disparity function h^y of the current scanline becomes

$$h^y = (1 - \lambda_1) f^y + \lambda_1 f^{y-1} \tag{1}$$

The right hand image in Figure 5 shows the DPs result.

4.2 Implementation Belief Propagation

Belief propagation (BP) is an iterative algorithm that is an efficient method for solving early vision problems. In the stereo vision structure estimation problem, BP is used to perform approximate inference on a NP-hard energy minimization problem to find an accurate disparity map between a pair of images. In this report, we examine and extract the parallelism in the BP algorithm by modifying the algorithm presented by Felzenszwalb and Huttenlocher [4]. 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. The total energy represents the sum of the data costs $D_p(d)$ for each pixel p and the discontinuity costs $V(d_p, d_q)$ for each pair (p, q) of neighboring pixels. In the stereo vision problem, the data cost $D_p(d)$ represents the cost of assigning disparity d to pixel p, using the brightness consistency assumption. The discontinuity cost $V(d_p, d_q)$ represents the cost of assigning disparities d_p and d_q to neighboring pixels [13]. 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. The energy

$$E(f) = \sum_{p \in P} D_p(f_p) + \sum_{(p,q) \in A} V(f_p - f_q) \tag{2}$$

of a labeling f of the whole image needs to be minimized. Although BP has highly accurate results with outstanding quality, it requires a longer processing time than dynamic programming, which makes it less practical in application domains requiring real-time performance. That is why we use a cell processor to process it.

Strategy 1. Each SPE only has 256 KB of LS and DMA can only carry 16 KB data one time. We could not transfer a whole image into an SPE to process. We cut images into blocks as follows, every blocks is 32 by 16 pixels. There are three methods we can use to communicate between PPE and SPE: MailBox, DMA, and notice. The MailBox is only 32-bit (4 byte), so we can use it to transfer the memory address of image blocks and transfer some tags, which can have special

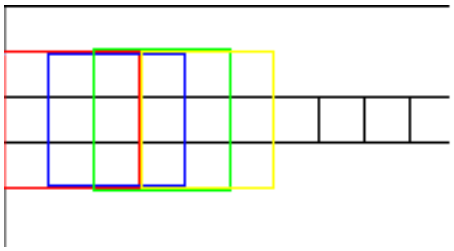

Fig. 6. Overlapping windows.

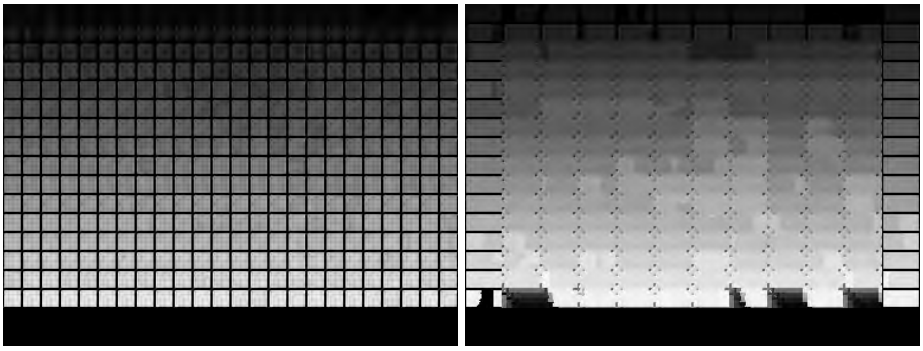

Fig. 7. Results on Tsukuba. Left: blocks without overlap. Right: blocks with overlap.

meanings (e.g., error and completion messages). The DMA can transfer 16 KB one time, so we can use DMA to carry larger data, such as image data, and result data between the PPE and SPE. We use overlapping windows as shown in Figure 6. Tiling an image into pairwise disjoint blocks is shown in Figure 7 (left); Figure 7 (right) shows the resulting tiling when using overlapping windows.

Strategy 2. From Figure 7 it is obvious that cutting the image into blocks shows a poor result, so we should try a new parallelization strategy on BP. Instead of cutting the image into blocks, we divide the execution flow of BP algorithm into four stages: compute energy, build the image pyramid, message updating, and computing the depth map [2]. Figure 8 shows the procedure of this strategy. From Figure 8 we can see that the PPE only sends data to the SPEs and stores process result from the SPEs. One of the most important stages is message updating. In this stage we rebuild the node's construction. Then we send node's information to the SPEs. After sending the data to each SPE, the four-direction messages at each node are calculated independently in

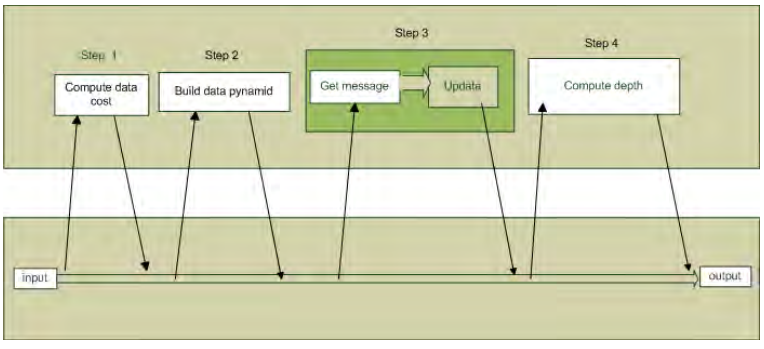

Fig. 8. Sketch of the BP procedure on the Cell BE.

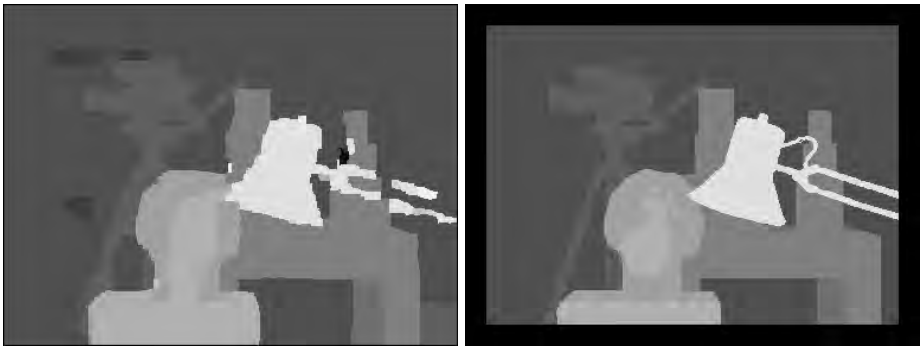

Fig. 9. Left: The result when following Strategy 2. Right: Ground truth of the shown *Tsukuba* stereo pair.

each iteration. The computed messages are then transmitted back to the main memory for the message updating of the next iteration [2]. Figure 9 (left) shows the result of this strategy. From this figure we can see that this strategy can get much better results than Strategy 1.

We use Strategy 2 to test real image sequences taken by HAKA1. From Figure 11 (top right) we can see that the results on original images are not ideal. That is because in the real-world, the illumination may be different between left camera and right camera, and images are typically degraded by various types of noise. We use two methods to pre-process the original image pairs: Sobel edge detection or a residual operator.

Figure 10 (left) shows an example of a Sobel edge map; such images are used as input for the BP algorithm as suggested in [7]. A residual image [15] is the difference between an original image and a smoothed version of itself. For this report we use a 3×3 mean filter. Let f denote the original image, and $S(f)$ denotes the smoothed version of image f; the residual image r equals $r = R(f) = f - S(f)$; see Figure 10 (right) for an example.

5 Experimental Results

We performed the experiments on the PlayStation 3 (PS3) platform, which provides one 3.2 Cell processor with six SPEs (we use four of them) and one PPE. The implementation of the parallelized BP and DP was based on the development environment where the PPE runs the fedora 10. The toolkit provided by IBM is CellSDK 3·1 [5].

Figure 11 compares results of BP on original images, Sobel edge images, and residual images, for the real world images. From this we can see that the use of residual and Sobel edge images provides better results than those for original images.

Fig. 10. Left: Sobel edge image. Right: mean residual image.

For dynamic programming, the normal PC took 6·3 s for one stereo pair (however, without aiming at time optimization here). When using four SPEs the time is 0·09 s, using six SPEs the time is 0·06 s.

Fig. 11. Top: an original image (left), and a BP result when using original image data (right). Bottom: a BP result when using Sobel edge images (left) and when using 3× 3 mean residual images (right).

The results for belief propagation are as follows. The standard images Tsukuba are 384×288 with a maximum disparity of 16 pixels. Five levels (of the pyramid) with five iterations per level are processed on the complete images, the σ-constant for smoothing the input images equals 0·7, the truncation-of-discontinuity-costs parameter equals 1·7, the truncation-of-data-cost parameter equals 15, and the weight λ of the data cost equals 0·07. The running time using these settings is 32·4 s for the normal PC. The time of BP on the Cell processor is 4·8 s (Strategy 1), and 2·1 s (Strategy 2). The disparity map is much better for Strategy 2 compared with Strategy 1, as Figures 7 and 9 show. However, the PS3 platform we used provided limited memory (256 MB XDRAM) for the PPE and only 256 KB local storage for SPE, which leads to a dramatic decrease in performance. The result shows the high suitability of this algorithm on the cell processor to reduce time.

6 Conclusions

In our implementation we utilized the SIMD architecture for parallelizing DP and BP stereo algorithms, and we examined those parallelizations of DP and BP on the multicore processors. We proposed ways to parallelize DP and BP in accordance with given architecture and memory limitations, and proposed a way to evaluate obtained results. The methodology of analyzing and exploiting parallelism, as presented in this article, is applicable to other stereo vision algorithms. Obtained results show that, after a careful analysis and parallelization, the implementations were sped-up considerably.

In future work, packing SIMD instructions for all the memory operations could be a key factor to produce further performance improvements, for supporting real-time performance. Moreover, the adoption of intrinsic functions for special operations, and carefully choosing compiler optimization phases, such as loop unrolling, also reveal opportunities for further performance improvements. Figures 9 and 11 show that there is still some noise, so we should also find improved stereo matching methods to improve calculated disparity maps. Having faster implementations at hand will allow for more efficient testing while designing modified stereo matchers.

References

1. S. Baker, S. Scharstein, J. P. Lewis, S. Roth, M. J. Black, and R. Szelisky. A database and evaluation methodology for optical flow. In Proc. *IEEE Int. Conf. Computer Vision*, CD, 2007.
2. L. Chi-Hua and H. Kun-Yuan. Parallellization of belief propagation method on embedded multicore processors for stereo vision. In Proc. *IEEE ESTImedia*, pages 39–44, 2008.
3. *.enpeda..* image sequence analysis test site. www.mi.auckland.ac.nz/EISATS/.
4. P. F. Felzenszwalb and D. P. Huttenlocher. Efficient belief propagation for early vision. In Proc. *CVPR*, volume 1, pages 261–268, 2004.

5. IBM. CBE programmer's guide v3.1. www.ibm.com/developerworks/power/cell/,2008
6. J. Fung and S. Mann. Using graphics devices in reverse: GPU-based image processing and computer vision. In Proc. *IEEE Int. Conf. Multimedia Expo*, pages 9–12, 2008.
7. S. Guan and R. Klette. Belief-propagation on edge images for stereo analysis of image sequences. In Proc. *Robot Vision*, pages 291–302, Springer, 2007.
8. S. Guan, R. Klette, and Y. W. Woo. Belief propagation for stereo analysis of night-vision sequences. In Proc. *PSIVT*, LNCS 5414, pages 932–943, 2009.
9. B. Jonathan. Programming high-performance applications on the Cell BE processor www.ibm.com.
10. Z. Liu and R. Klette, Dynamic programming stereo on real-world sequences. In Proc. *ICONIP*, pages 527–534, Springer, 2009.
11. Y. Ohta and T. Kanade. Stereo by two-level dynamic programming. In Proc. *IJCAI*, pages 1120–1126, 1985.
12. D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. J. Computer Vision*, **47**:7–42, 2002.
13. G. Scott and K. Chandra. GPU implementation of belief propagation using CUDA for cloud tracking and reconstruction. In Proc. *PRRS*, pages 1–4, 2008.
14. P. Seebach. The little broadband engine that could: An introduction to using SPEs for cell broadband engine development. www.ibm.com/developerworks/library/pa-tacklecell1/, June 2007.
15. T. Vaudrey and R. Klette. Residual images remove illumination artifacts for correspondence algorithms! In Proc. *Pattern Recognition - DAGM*, to appear, 2009.

Some Useful Links and Notes

Links

When your simulator crashes (CellSDK IDE) on startup then see:
http://www.ibm.com/developerworks/forums/thread.jspa?messageID=14237064&#14237064

If you could not download or lose some rpm packages, download from here:
http://www.bsc.es/projects/deepcomputing/linuxoncell/cellsimulator/sdk3.1/CellSDK-Open-Fedora/x86/

If you have used libspe(before cellsdk2.1), this link tells you the difference between libspe2 and libspe:
http://www.ibm.com/developerworks/library/pa-libspe2/

Notes

- If you use FC10.Xterm not being installed by default. So you should install it by yourself if you want to use simulator.
- When you use simulator you may get an "out of stack space (infinite loop)" message. The solution is before you start the simulator type the command: `ulimit -s unlimited`
- CellSDK depends on some specific libs or packages such as ppu-binutils-2.18.50-21.xxx.rpm and spu-binutils-2.18.50-21.xxx.rpm. However, when you install linux it often installs new version by default, so you need to remove the new one download and install the version I write above. You can download them from link above (http://www.ibm.com/developerworks/library/pa-libspe2/).
- If you have already install SDK successfully, don't update linux!
- If you use Cellide you will crash, because there is bug in the ide. Download fix cellide-3.1.0-7 and do as website says(the link is above).
- If you use vector program memory must be aligned to 16 or 128. In C we use memalign in C++ we overload operator new, see: http://bytes.com/groups/cpp/591742-overload-new-delete-memalign-malloc