Fast Disparity Map Estimation Using Multi-thread Parallel Processing

Yunseok Song and Yo-Sung Ho

Gwangju Institute of Science and Technology (GIST)
261 Cheomdan-gwagiro, Buk-gu, Gwangju 500-712, Korea
Telephone: +82-62-715-2258 Fax: +82-62-715-3164
E-mail: {ysong, hoyo}@gist.ac.kr

***Abstract*—Parallel processing is a concurrent operation aimed at execution time reduction. Programmers can take advantage of equipped multi-core systems by dividing tasks and assigning them to multiple threads. In this paper, we explain basic concepts of parallel processing: parallel architectures, parallel programming models and data race. Subsequently, we implement two disparity map estimation algorithms—sum of absolute differences (SAD) and adaptive support-weights (ASW)—using OpenMP, an API for shared memory parallel programming. Experimental results demonstrate that multi-thread parallel processing reduce execution time significantly compared to the default serial processing.**

I. Introduction

Parallel processing is an operation that runs several tasks simultaneously. The main goal is to accelerate computation. In industry, parallel processing can be used in fields that process huge amount of data, e.g., data mining, web search engines, medical imaging, virtual reality and multimedia technologies [1].

Parallelization can be achieved by exploiting computer-equipped multiple cores (processors). As technologies evolved, hardware manufacturers have released cheaper and enhanced multi-core computers, expanding the market size. High-end computers even contain thousands of CPUs.

In theory, ideally, if m processors are used, execution time becomes the $1/m$-th of single processor execution time [2]. However, this would not always be the case since CPUs may interfere in practical situations.

Disparity map estimation is a pixel-matching operation for stereo pairs—left and right images. Estimation methods can be classified to global and local methods [3]. Global methods are generally more computationally expensive than local methods. In addition, many difficult-to-determine parameters are required. On the other hand, local methods are based on similarity measures using a window, which are our focus.

In this paper, we apply parallel processing to disparity map estimation and examine performance. We use OpenMP, an API for parallel programming, to parallelize computationally expensive portions. Section II reveals several basic concepts of parallel processing. In Section III, we describe two similarity measures used in disparity map estimation. Afterward, we present experimental results in Section IV and discuss them in Section V. Finally, we conclude the paper in Section VI.

II. Parallel Processing

Understanding parallel architectures, parallel programming models and data race is crucial to comprehending parallel processing.

A. Parallel Architectures

Three parallel architectures exist: shared, distributed and hybrid (shared/distributed) [4]. Fig. 1 shows the diagrams.

Initially, shared memory systems allow all processors to access the same memory space. A memory location change could affect all other processors. Uniform memory access (UMA) and non-uniform memory access (NUMA) are the two main classes of shared memory. UMA, also known as symmetric multi-processor (SMP), allows equal memory access time for all CPUs—inputs and outputs are shared resources. UMA is easy to administer and efficient in resource use but expensive. NUMA is the exact opposite of UMA in every way—resources are distributed not shared.

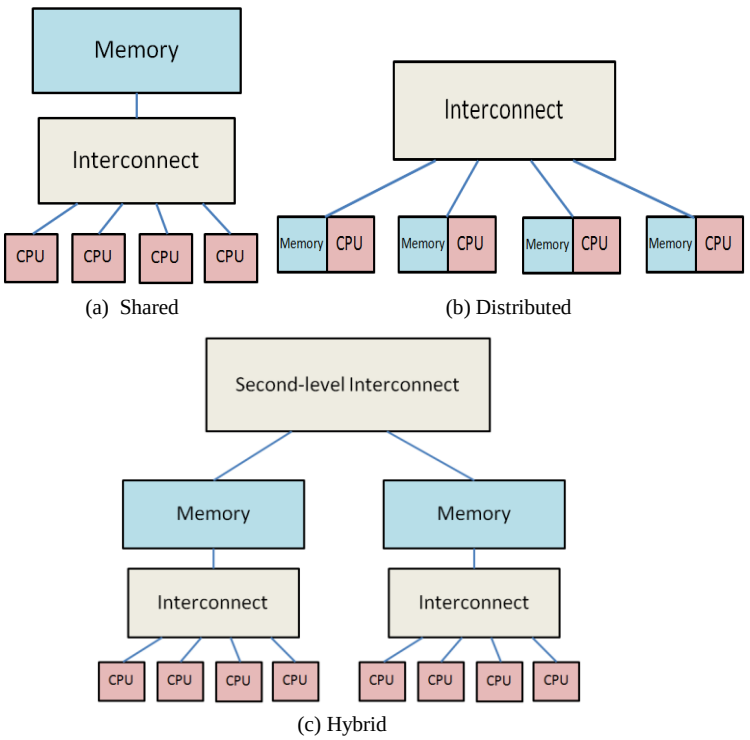


Figure 1. Diagrams of parallel architectures.

Moreover, in distributed memory systems, each CPU has its own local memory. Since memories are not connected, a local change does not affect other CPUs. If data access to another CPU is required, the programmer needs to synchronize communication. Distributed memory allows fast memory access without interference but designing inter-processor communication is very difficult.

Lastly, the hybrid architecture employs both advantages and disadvantages of shared and distributed memory architectures. The shared and distributed components are cache coherent processors and multi-processor networking, respectively. Concerning high-end computing, hybrid architecture is getting more and more popular. Fig. 1 contains diagrams of parallel architectures.

B. Parallel Programming Models

Five parallel programming models exist: shared memory, threads, message passing, data parallel and hybrid. Generally, models are not dependent to underlying hardware, i.e., any model can be implemented on any hardware. Although it is hard to say which model is the finest, there are better implemented models over others. The two most well-known and widely available implementations are OpenMP and message passing interface (MPI). They are de facto standards for threads and message passing models, respectively. Both support C, C++ and Fortran.

MPI is expedient for distributed memory systems, i.e., clusters. The model consists of a cluster of systems. All threads have access to its own memory only. Thus, data are shared and transferred by exchanging buffers. Benefits include flexibility since any size of any cluster can be used. In addition, MPI call plug-in procedure allows a straightforward approach. However, programmers would be burdened with a lot of work due to the complicated underlying system.

In regards to OpenMP, all threads can access global and shared memories. Programmers can control the number of threads. Optimal performance occurs when the number of threads represents the number of processors. A master thread exists to assign tasks to threads, i.e., fork-join. Fork-join time increases when there are more threads than processors. Moreover, data can be labeled with private or shared. In particular, private data are visible to one thread while all threads can spot shared data. In practical programs, local variables which are about to be parallelized should be private. Additionally, global variables must be assigned as shared data. Unlike MPI, OpenMP requires a compiler. Most IDEs today accommodate OpenMP. Numerous benefits exist to using OpenMP, e.g., preservation of serial code, simplicity, flexibility and portability. Nevertheless, explicit synchronization remains as an issue that to be addressed.

C. Data Race

Data race is a flaw caused by shared variable updating, creating unintended results. If a thread changes a shared variable, and then another thread accesses and attempts to employ it, the variable would not represent the expected value anymore. Thus, with regards to parallel processing, all

This occurs when a thread updates shared variables. When other threads access and use them, the variables will not hold expected values due to the previous update. Thus, changing variables must be taken carefully in parallel loops. Figure 2 shows an example of data race.

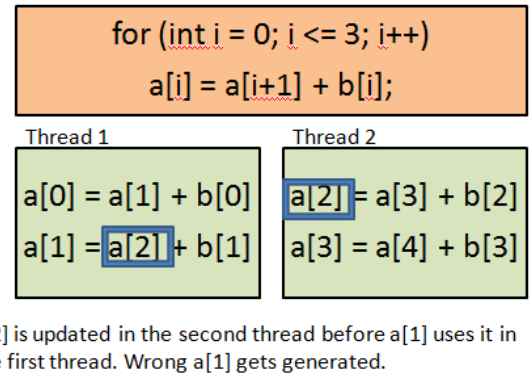


Figure 2. Example of data race.

III. Disparity Map Estimation

Disparity map estimation is a process of finding corresponding pixels for stereo pair images. Such an operation is executed on a pixel-by-pixel basis, i.e., procedure is independent. Hence, parallel processing is suitable. We briefly explain two disparity map estimation methods—sum of absolute differences (SAD) and adaptive support-weights (ASW).

SAD is a widely used matching algorithm. The idea is based on accumulating absolute differences of left image and right image pixels within a given window [5]. Equation (1) describes SAD. The more similar the pixels are, the less the SAD value becomes. The d that generates the least SAD is determined to be the disparity.

$$\mathrm{SAD}(x,y,d) = \sum_{i=-\frac{1}{2}(w_x-1)}^{\frac{1}{2}(w_x-1)} \sum_{j=-\frac{1}{2}(w_y-1)}^{\frac{1}{2}(w_y-1)} \{ \sum_{c\in\{R,G,B\}} \left| I_{L,c}(x+i,y+j) - I_{R,c}(x+i+d,y+j) \right| \}. \quad (1)$$

Yoon and Kweon have proposed the ASW approach. Such a method adjusts pixel-weights by considering both color similarity and geometric proximity of pixels within a given window. Equations (2), (3) and (4) describe the flow of algorithm.

Support-weight is defined to be dependent on color similarity and geometric proximity between center and neighboring pixels [6]. $w(I_p,I_q)$ denotes support-weight, I_p and I_q being center and neighboring pixels of image I, respectively.

Considering the human visual system, CIELab space is selected due to the three-dimensional representation of color perception [7]. c_{pq} represents the Euclidean distance between two CIELab space colors while g_{pq} is the Euclidean distance of p and q. γ_p is a constant that controls color similarity and γ_p is the radius of window. $E(L_p, R_{p+d})$ and $e(L_q, R_{q+d})$ denote dissimilarity and pixel-based raw matching cost according to d, respectively. Similar to SAD, the d that generates the least dissimilarity is determined to be the disparity.

Our main focus is parallelizing pixel-similarity measure process. Since this is independent from other pixel measures, multi-thread parallel processing fits well for shortening computation time.

$$w(I_p, I_q) = \exp\left(-\frac{c_{pq}}{\gamma_c} - \frac{g_{pq}}{\gamma_p}\right). \quad (2)$$

$$E(L_p, R_{p+d}) = \frac{\sum_{L_q \in N_{L_p}, R_{q+d} \in N_{R_{p+d}}} w(L_p, L_q) w(R_{p+d}, R_{q+d}) e(L_q, R_{q+d})}{\sum_{L_q \in N_{L_p}, R_{q+d} \in N_{R_{p+d}}} w(L_p, L_q) w(R_{p+d}, R_{q+d})}. \quad (3)$$

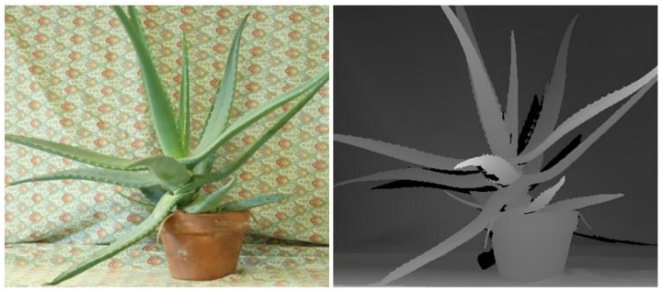
$$e(L_q, R_{q+d}) = \min\{\sum_{c \in \{R,G,B\}} |I_c(L_q) - I_c(R_{q+d})|, T\}. \quad (4)$$

IV. Experimental Results

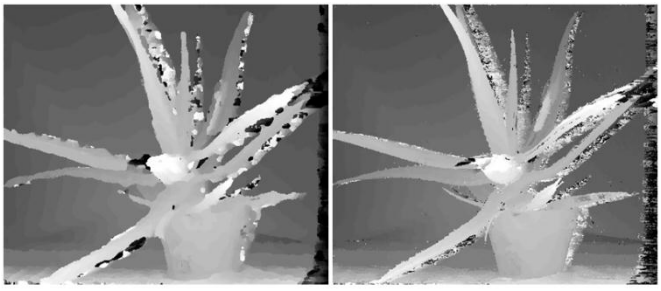
We conducted experiments on four stereo image pairs using an Intel(R) Core(TM)2 Quad CPU @ 2.40 GHz processor. Three stereo image pairs—"Aloe"(427 × 370), "Cones"(450 × 375) and "Teddy"(450 × 375)—provided by Middlebury College and "Newspaper"(512 × 384) from Gwangju Institute of Science and Technology were used as test data.

For simplicity, we assumed the image pairs were pre-processed with rectification. We set maximum disparity to 50 for both methods. 9×9 and 35×35 windows were applied for SAD and ASW, respectively. The parameter setting for ASW includes $T = 40$ and $\gamma_c = 5$. Figures 3-6 display the generated disparity maps. Table I-IV exhibit execution time by number of threads used.

As the number of threads exceeded the number of processors, performance did not enhance anymore. This is due to the increased time caused by thread distribution of master thread.



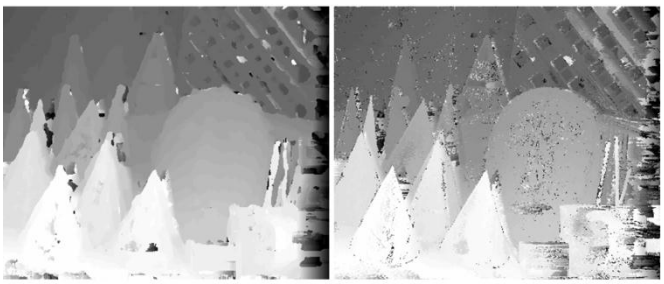
(a) Right image (b) Ground truth



(c) SAD (d) ASW

Figure 3. Disparity maps for "Aloe".

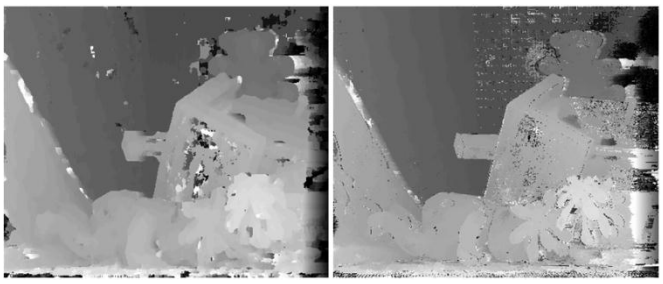
(a) Right image (b) Ground truth



(c) SAD (d) ASW

Figure 4. Disparity maps for "Cones".

(a) Right image (b) Ground truth



(c) SAD (d) ASW

Figure 5. Disparity maps for "Teddy".

For both SAD and ASW, 4-thread execution time was about 3.9× faster than that of serial processing. Such results are due to the quad-core processor use.

V. Conclusions

In this paper, we presented basic concepts of parallelization and applied multi-thread parallel processing to disparity map estimation using OpenMP. Since SAD and ASW are both based on independent window operations and require massive computation, parallel processing is suitable for fast execution. We experimented on four stereo image pairs with varying number of threads. Our quad-core processor allowed maximum performance when four threads were used. By using the 4-thread parallel processing, we observed that execution became faster by a factor of about

3.9. Hence, parallel processing is greatly effective for computationally expensive tasks.

(a) Right image

(b) SAD (c) ASW

Figure 4. Disparity maps for "Newspaper".

TABLE I
EXECUTION TIME: "ALOE"

		using SAD (sec)	using ASW (sec)
Serial		189.70	16982.64
Parallel: Number of threads	1	190.33	16980.55
	2	102.21	8910.89
	3	70.91	6541.59
	4	48.22	4310.04
	5	58.13	5211.22
	6	49.15	4811.83
Speed Factor: 4-thread / Serial		3.93	3.94

TABLE II
EXECUTION TIME: "CONES"

		using SAD (sec)	using ASW (sec)
Serial		186.55	17030.59
Parallel: Number of threads	1	188.27	17112.81
	2	94.06	8966.04
	3	67.44	6809.77
	4	47.58	4310.39
	5	59.21	5119.95
	6	48.30	4515.27
Speed Factor: 4-thread / Serial		3.92	3.95

TABLE III
EXECUTION TIME: "TEDDY"

		using SAD (sec)	using ASW (sec)
Serial		188.71	17136.10
Parallel: Number of threads	1	188.82	17139.73
	2	94.85	8922.58
	3	63.24	6211.48
	4	47.89	4370.77
	5	56.98	5188.91
	6	48.95	4529.03
Speed Factor: 4-thread / Serial		3.94	3.92

TABLE IV
EXECUTION TIME: "NEWSPAPER"

		using SAD (sec)	using ASW (sec)
Serial		217.90	19921.87
Parallel: Number of threads	1	218.28	19930.66
	2	119.51	10217.29
	3	76.19	6920.48
	4	55.82	5106.54
	5	69.37	6481.09
	6	64.19	5582.33
Speed Factor: 4-thread / Serial		3.90	3.90

ACKNOWLEDGMENTS

This research was supported by the IT R&D program of MKE/KEE/KEIT. [KI001932, Development of Next Generation DTV Core Technology].

REFERENCES

[1] B. Barney, "Introduction to Parallel Computing," May 25, 2010. [Online]. Available: https://computing.llnl.gov/tutorials/parallel_comp. [Accessed: Aug. 18, 2010]

[2] R. van der Pas, "Basic Concepts of Parallelization," July 13, 2010. [Online]. Available: http://openmp.org/wp/. [Accessed: Aug. 12, 2010]

[3] K.-J. Yoon and I.S. Kweon, "Adaptive Support-Weight Approach for Correspondence Search," *IEEE Transactions on Pattern Analysis Machine Intelligence*, vol. 28, no. 4, pp. 650-656, April 2006.

[4] "Parallel Computing and Data Parallelism - CodeProject," Feb. 2, 2010. [Online]. Available: http://www. codeproject.com/KB/ /cpp/Loop_Parallelizing.aspx. [Accessed: Aug. 15, 2010]

[5] K. Muhlmann, D. Maier, J. Hesser and R. Manner, "Calculating Dense Disparity Maps from Color Stereo Images, an Efficient Implementation," *International Journal of Computer Vision*, vol. 47, no. 1-3, pp. 79-88, April 2002.

[6] C. Tomasi and R. Manduchi, "Bilateral Filtering for Gray and Color Images," *Proceedings of International Conference on Computer Vision*, pp. 839-846, Jan. 1998.

[7] J.-I. Jung and Y.-S. Ho, "Color-compensated Stereo Matching Algorithm for Color Inconsistent Stereo Pair," *International Technical Conference on Circuits/Systems, Computers and Communications*, pp. 717-720, July 2009.

[8] D. Scharstein, "vision.middlebury.edu," July 27, 2007. [Online]. Available: https://vision.middlebury.edu/stereo/data. [Accessed: July 14, 2010]