Improved Disparity Map Computation on Stereoscopic Streaming Video with Multi-core Parallel Implementation

Cheong Ghil Kim[1] and Yong Soo Choi[2]
[1] Dept. Of Computer Science, Namseoul University
Cheonan, Choongnam, Korea
[e-mail: cgkim@nsu.ac.kr]
[2] Division of Liberal Arts and Teaching(Multimedia), Sungkyul University
Anyang-si, Kyeonggi-do, Korea
[e-mail: ciechoi72@gmail.com]
*Corresponding author: Yong Soo Choi

Received May 10, 2014; revised November 18, 2014; accepted January 13, 2015; published February 28, 2015

Abstract

Stereo vision has become an important technical issue in the field of 3D imaging, machine vision, robotics, image analysis, and so on. The depth map extraction from stereo video is a key technology of stereoscopic 3D video requiring stereo correspondence algorithms. This is the matching process of the similarity measure for each disparity value, followed by an aggregation and optimization step. Since it requires a lot of computational power, there are significant speed-performance advantages when exploiting parallel processing available on processors. In this situation, multi-core CPU may allow many parallel programming technologies to be realized in users computing devices. This paper proposes parallel implementations for calculating disparity map using a shared memory programming and exploiting the streaming SIMD extension technology. By doing so, we can take advantage both of the hardware and software features of multi-core processor. For the performance evaluation, we implemented a parallel SAD algorithm with OpenMP and SSE2. Their processing speeds are compared with non parallel version on stereoscopic streaming video. The experimental results show that both technologies have a significant effect on the performance and achieve great improvements on processing speed.

Keywords: Disparity map, stereoscopic 3D, parallel processing, shared memory, sub-word parallelism

This work was supported by the ICT R&D program of MSIP/IITP. [I5501-14-1007, 3D Smart Media/Augmented Reality Technology, Korea-China-Japan-Russia Cooperation International Standardization] and by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT & Future Planning(No. 2013R1A1A1013410)

http://dx.doi.org/10.3837/tiis.2015.02.014 **ISSN : 1976-7277**

1. Introduction

Recent developments on 3D technologies have made 3DTV and mobile 3DTV the most spotlighted technology in the area of audio-video entertainment and multimedia. This development enables auto-stereoscopy or glasses-free 3D viewing on smartphone without the use of special headgear or glasses on the part of the viewer [1]. The principle of 3D video is based on stereo vision with two cameras to obtain two different views on the same scene, and the relative depth information is obtained by comparing two images, known as disparities; therefore, depth maps which contains information relating to the distance of the surfaces of scene objects from a viewpoint may provide an essential description of the world seen through cameras. This technology has been studied extensively due to its usefulness in many applications like 3D scene reconstruction, robot navigation, civil engineering, manufacturing, and so on.

In order to construct a stereo vision system, two cameras located at two different positions is usually used, which is known as binocular stereopsis based on epipolar constraint. Here, for a pixel in the left image the corresponding point in the right image lies on the same horizontal line, the epipolar line. As a result, the geometry associated with solving this problem is simplified by assuming that the two cameras are coplanar with aligned image coordinate systems.

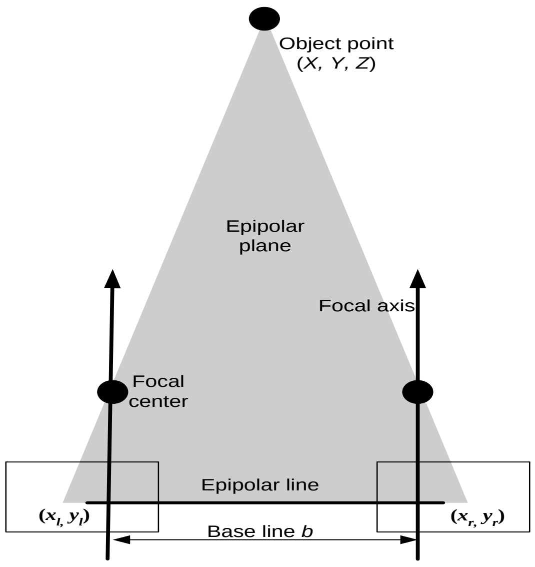


Fig. 1. Stereo camera geometry

Fig. 1 shows the basic structure of the stereo image formation and the stereo camera geometry [2]. The center of the lens is called the camera focal center and the axis extending from the focal center is referred to as the focal axis. The line connecting the focal centers is called the baseline, *b*. The plane passing through an object point and the focal centers is the epipolar plane. The intersection of two image planes with an epipolar plane makes the epipolar line. Let *(X, Y, Z)* denote the real world coordinates of a point. The point is projected onto two corresponding points, (x_l, y_l) and (x_r, y_r), in the left and right image, respectively. The disparity is defined as the difference vector between two points in the stereo images, $v = (x_l - x_r; y_l - y_r)$. According to the stereo camera geometry shown in Fig. 1, the disparity is defined with the difference between two focal centers denoted as C_1 and C_2 in **Fig. 2** and calculated as below:

$$D = C_2 - C_1 \tag{1}$$

The depth d therefore is calculated by triangulation as below:

$$d = b\frac{f}{D} \tag{2}$$

where b is the distance of the two optical centers and f is the focal length. A disparity of zero indicates that the depth of the appropriate point equals infinity. In order to assure the stereo epipolar geometry, the rectification of both images is necessary; therefore, both cameras require calibration first to acquire camera parameters for the rectification.

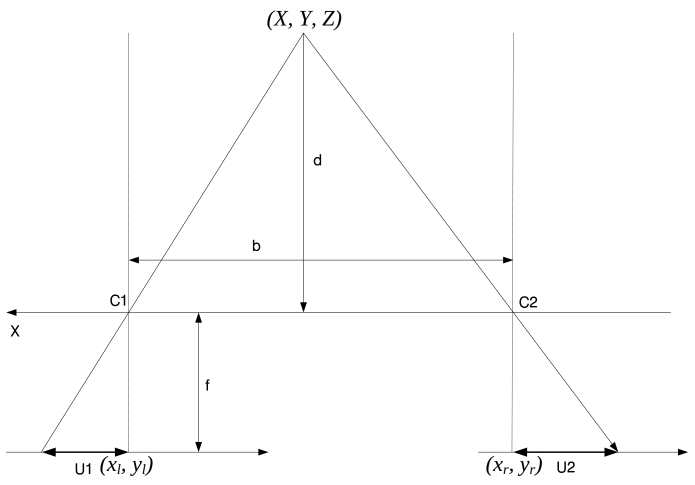


Fig. 2. Disparity calculation

In general, the most difficult area in stereo vision is matching points or features between the left and right image. This is called as stereo matching or stereo correspondence problem [3], which has been an intensive research area for decades [4, 5, 6]. In order to solve this problem, many algorithms have been introduced and most of previous studies to solve and improve the performance of stereo matching can be grouped into three categories according to matching primitives: area-based [7], feature-based [8], and phased-based approaches [9]. The most common ones are absolute intensity differences (AD), the squared intensity differences (SD) and the normalized cross correlation (NCC) [10]. Evaluation of various matching costs can be found in [11,12]. In recent years, this challenge has been extended to Multi-view Video Coding (MVC) which comprises rich 3D information and requires more computational powers [13-15].

Stereo matching inherently requires considerably high computational expenses, especially when extracting dense disparity map which can produce accurate segmentation for more reliable applications. This problem becomes more serious with large high resolution images, and it is still far away from achieving real time requirements in systems with either general purpose CPU or DSP. As a result, there have been several researches on implementing a specialized processing hardware to achieve the required computational complexity using FPGA [16-18].

In the meantime, recent advances of CPU performance on speed and architecture can be expected to bring an alternative way of improving the calculation of disparity map. In the market, multi-core processor architecture has emerged as a dominant trend in desk-top PCs while preserving the CPU clock speed between 3 GHz and 4 GHz [19-21]. This performance enhancement allows a single chip to increase its processing capability without requiring a complex system. Especially, the increase in processor core brings new opportunities in parallel computing. Therefore, a performance improvement on computing depth map could be achieved by taking advantage of parallelism in software rather than implementing dedicated hardware.

This paper presents a fast dense stereo-correspondence algorithm using two parallel programming techniques on multi-core CPU. The Sum of Absolute Differences (SAD) algorithm is used to reconstruct disparity map requiring heavy computations but simple and identical calculation with heavy memory accesses. The proposed method mainly aims at improving the execution time using OpenMP (Open Multi-processing) [22], an open specification API (Application Programming Interface) and SSE (Streaming SIMD Execution) [23]. They can provide an easy low-burdensome method for threading applications based on shared memory architecture and a SIMD (Single Instruction Multiple Data) parallel execution with sub-word parallel instruction set, respectively. And the performance evaluation was made by comparing its processing time with the serial implementation.

The organization of this paper is as follows. Section 2 introduces the background of SAD algorithm and parallel technologies of OpenMP and SSE. Section 3 discusses the design of parallel SAD using OpenMP and SSE. Section 4 covers the results of simulation about the implementations of parallel SADs with comparison with its serial implementation. In section 5, we conclude our results.

2. Background

2.1 SAD algorithm

The SAD is the most commonly used algorithm for measuring a similarity between image blocks. Here, a search iteration is performed for each candidate block by taking absolute difference between each pixel in the original block and the corresponding pixel in a block being used for comparison. These differences are summed to create a simple metric of block similarity. It computes intensity differences for each center pixel (i, j) in a window W(x, y) as follows:

$$SAD(x,y,d) = \sum_{(i,j)\in w(x,y)}^{N} \left| I_L(i,j) - I_R(i-d,j) \right|, \tag{1}$$

where I_L and I_R are pixel intensity functions of a left and right image, respectively. $W(x, y)$ is square window that surrounds the position (x, y) of the pixel. A disparity SAD (x, y, d) calculation is repeated within the x-coordinate frame in the image row, defined by zero and maximum possible disparity d_{max} of the searched scene. A minimum difference value over the frame indicates the best matching pixel. And a position of the minimum defines the disparity of an actual pixel. On each search iteration, a SAD corresponding of a candidate block is computed using all its pixels simultaneously. The value obtained is compared with the reference SAD which is the minimum SAD computed before this iteration. If a current SAD is less than the reference SAD, it is stored as the reference SAD for the remaining search iterations. This process can be summarized as three steps: 1) computation of differences between corresponding elements; 2) determine the absolute value of each differences; 3) add all absolute values.

Usually, the matching costs are aggregated over support regions. Those support regions, often referred to as support or aggregating windows, could be square or rectangular, fix-sized or adaptive ones. In this work, we have implemented with different window sizes of 4, 8, and 16 for performance evaluations.

2.2 OpenMP

OpenMP is an open specification API supporting multi-platform shared-memory parallel programming. In recent years, OpenMP has extended its use in accelerators, embedded systems, multi-core and real-time systems although it solely focused on shared memory systems in the early days. As for OpenMP programming model, parallelization have been implemented by high-level abstractions which allow parallelized C/C++ programs for parallel processors with a shared memory by adding specific OpenMP directives into C-program codes. These directives support the distribution of autonomous subtasks (threads) over the available processor cores [24]. As a result, all threads can access global and shared memories and programmers can control the number of threads. Optimal performance occurs when the number of threads represents the number of processors.

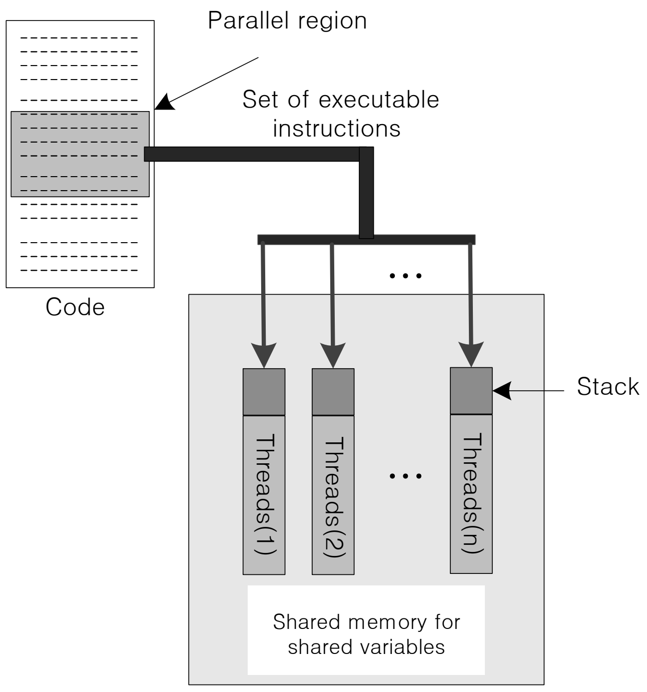

Parallel region
Set of executable instructions
Code
Stack
Threads(1)
Threads(2)
Threads(n)
Shared memory for shared variables

Fig. 3. Operational block diagram of OpenMP

Fig. 3 depicts the operational block diagram of OpenMP in which how a process can be divided into several threads. Here, 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. OpenMP requires a compiler and most IDEs today accommodate it. 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 [25].

2.3 Stream SIMD Extension

Another parallelization technique, called SSE technology, is vectorization by converting a scalar code to a code using SSE instructions, in which a single instruction will process several elements at the same time.

The increasing popularity of high quality digital contents processing for many areas of multimedia applications such as digital image processing, video processing, 3D graphics, and so on, leads the ISA (Instruction Set Architecture) of general purpose processors to have quipped with dedicated instructions based on sub-word parallelisms [25]. They demand high performance computing power for data intensive processing and have a great potential for SIMD parallel processing. Current microprocessors for desktop and mobile computing devices have the capability of hardware supporting for vector operations with SIMD type instructions.

This vectorization started form MMX (MultiMedia eXtension), the first IA-32 SIMD instructions on Intel CPU. It performed arithmetic or logical operations on the packed data in parallel. Later this was extended to SSE using eight new 128-bit registers and SSE2 technology introduced new SIMD double-precision floating-point instructions and new SIMD integer instructions into the IA-32 Intel architecture. More specifically, SSE2 has the memory streaming instruction extensions, which allows programmers to prefetch data into a specified level of the cache hierarchy. Most multimedia applications present the streaming data access pattern. This means that data are accessed sequentially and seldom reused. Therefore, prefetching this type of data into the L2 cache is an effective way to improve the memory system performance [26].

SSE2 contains integers, 2, 4, 8, or 16 variables with a width of 8, 4, 2 or 1 byte. Therefore it is possible to perform the same operation on several variables in parallel. **Fig. 4** shows various packed data types in XMM registers. The introduction of sub-word parallelism to an existing ISA requires new instructions with packed types arithmetic and logic operations and data/sub-word packing and rearrangement operations. SSE instructions are divided into the following four functional groups: packed and scalar single-precision floating-point instructions, 64-bit SIMD integer instructions, state management instructions, and cacheability control, prefetch, and memory ordering instructions

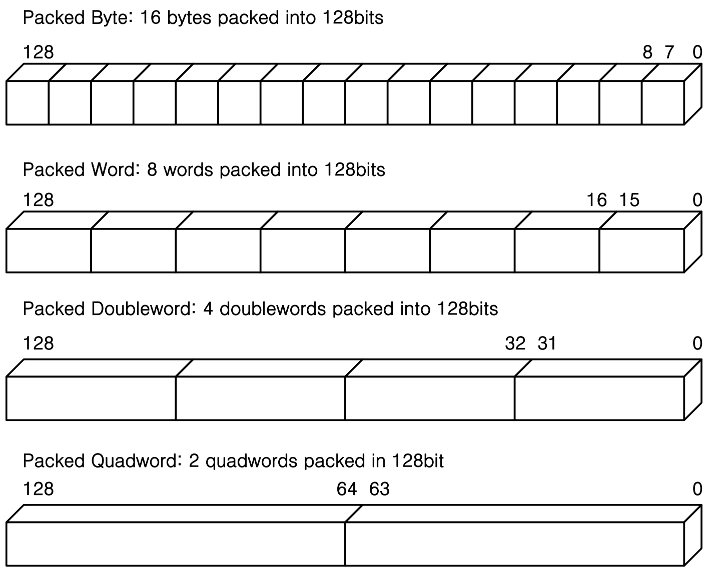

Packed Byte: 16 bytes packed into 128bits
128
8 7 0
Packed Word: 8 words packed into 128bits
128
16 15 0
Packed Doubleword: 4 doublewords packed into 128bits
128
32 31 0
Packed Quadword: 2 quadwords packed in 128bit
128
64 63 0

Fig. 4. Packed data types

3. Parallel implementation

This section introduces efficient parallel implementations of SAD algorithm, demanding high performance computing power, in multi-core CPUs. SAD is a representative computation intensive algorithm accompanied by heavy memory access with relatively low computational complexity. This operational model is well suited for exploiting the streaming SIMD extension technology and OpenMP based on thread. By doing so, we can take advantage of all the hardware features of multi-core processor concurrently for data- and task-level parallelism.

3.1 OpenMP

Disparity map calculation is a pixel-by-pixel basis to find corresponding pixels for a left and right image pair. Generally, this computing procedure can be characterized as having no dependency between processes. It means that these kinds of application are inherently suitable for making use of parallel processing. The basic processing concept of SAD is accumulating absolute differences of a left and right image pixels within a given window. Using the Equation (1), the least SAD is determined to be the disparity.

By using OpenMP, SAD program is initially executed by one process; and then it activates light-weight processes (threads) at the entry of a parallel region. After that each thread executes a task comprised of a group of instructions. OpenMP program is an alternation of sequential regions and parallel regions. A sequential region is always executed by the MASTERthread, the one whose rank equals 0. A parallel region can be executed by many threads at once.Threads can share the work contained in the parallel region.

Fig. 5 shows the general concept of three possible parallel implementation methods: 1) executing a loop by dividing up iterations between the threads; 2) executing many code sections but only one per thread; 3) executing many occurrences of the same procedure by different threads. During the execution of a task, a thread available can be read and/or updated in memory. When it is defined either in the stack (local memory space) of a thread, it becomes a private variable; when in a shared-memory space accessible by all the threads, it is a shared variable [23]. This work takes advantage of loop-level parallelism for fast SAD and the *pragma omp parallel for num_threads()* is used to fork threads and specify the number of threads. Four threads are used for this four core system.

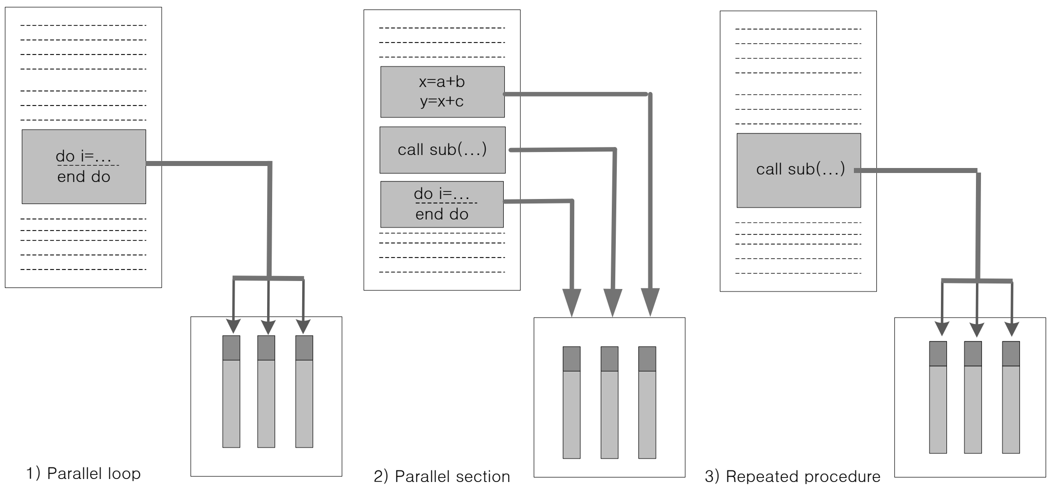


Fig. 5. Different parallel implementation methods

3.2 Stream SIMD Extension

Current microprocessors for desktop and mobile computing devices have the capability of hardware supporting for vector operations with SIMD type instructions as a representative enhancements to traditional superscalar processors. The advantage is to exploit sub-word parallelism in SIMD style by making small data elements packed together into one register and operating on all data elements in a register in parallel.

The source code block depicted in **Fig. 6** shows a SAD function based on 4 x 4 block with current and reference data passed as parameters. Data is stored in XMM registers and differences are calculated with a SIMD instruction of PSUBUSB and negative values are clipped at zero. This subtraction is executed once more with swapping operands. PSUBUSB performs a SIMD subtract of packed unsigned integers of a source operand from the packed unsigned integers of the destination operand with unsigned saturation, and stores the packed unsigned integer results in the destination operand. The final absolute difference can be obtained by adding the two results of subtraction by using a SIMD add instruction of PADDB.

```
unsigned int mmx_GetSAD(unsigned char *pRef, unsigned char *pOrg, int strip_ref,
int strip_org)
{
 unsigned int sad=0;

 __asm{

  mov  eax, [pRef];   mov  ebx, [pOrg];
  mov  esi, 16;

_LOOP:

  // Calculate 8 pixels of Absolute differenceS with mmx register and PSUBUSB
  // Clipping the minus results

  movq mm0, [eax];  movq mm1, [eax + 8];
  movq mm2, [ebx];  Movq mm3, [ebx + 8];

  movq mm4, mm0;   movq mm5, mm;

  psubusb mm0, mm2; psubusb mm2, mm4;
  psubusb mm1, mm3; Psubusb mm3, mm5;

  paddb mm0, mm2;  paddb mm1, mm3;

  // SUM of absolute differences

  movq  mm2, mm0; Movq  mm3, mm1;
  pxor  mm4, mm4;

  punpckhbw mm0, mm4; Punpcklbw mm2, mm4;
  punpckhbw mm1, mm4; Punpcklbw mm3, mm4;

  paddw  mm0, mm2; Paddw  mm1, mm3; Paddw  mm0, mm1;
  movq  mm1, mm0;

  punpckhwd mm0, mm4; Punpcklwd mm1, mm4;
  paddd  mm0, mm1

  movd  ecx, mm0; Psrlq  mm0, 32; movd  edx, mm0

  add   ecx, edx;   add   sad, ecx;
  add   eax, 16 ;   add   ebx, 16;

  dec   esi
  jnz   _LOOP

  emms
 }

 return sad;
}
```

Fig. 6. SSE impelentation of SAD on 4 x 4 block

4. Experimental Classification Results and Analysis

For the simulation, we estimate the computation speed of SAD algorithm on the sample stereo video stream, SampleVideo.avi, with different block sizes of which window sizes are 4 × 4, 8 × 8, and 16 × 16. They are denoted as 4, 8, and 16, respectively. The details of SampleVideo are summarized in **Table 1**. Three versions of the SAD algorithm have been implemented with C++ in this paper. They are serial, OpenMP, and SSE execution version and denoted as normal, OpenMP, and SSE, respectively. The normal version is an ordinary sequential execution version without writing any parallel code. OpenMP and SSE signify the implementation exploiting task- and data-level parallelism, respectively. In order to evaluate the speedup of parallel SAD implementations over its serial one, the processing time was measured. Here, speedup is defined as the ratio of performance with the enhancement to performance without the enhancement. **Table 2** describes the experimental hardware environment using Laptop with Intel Core I7 CPU. In this work, we just estimate the processing time of SAD algorithm and to obtain the execution time in terms of CPU clock cycles, we use RDTSC (Read Time Stamp Counter) instruction to measure the execution time of the three different code blocks.

Execution times are obtained by executing each implementation 10 times inside a loop, and then the summed averaged execution time was selected with milliseconds measure unit. **Fig. 7** shows the simulation UI for selecting simulation parameters and monitoring the running process.

Table 1. Simulation video

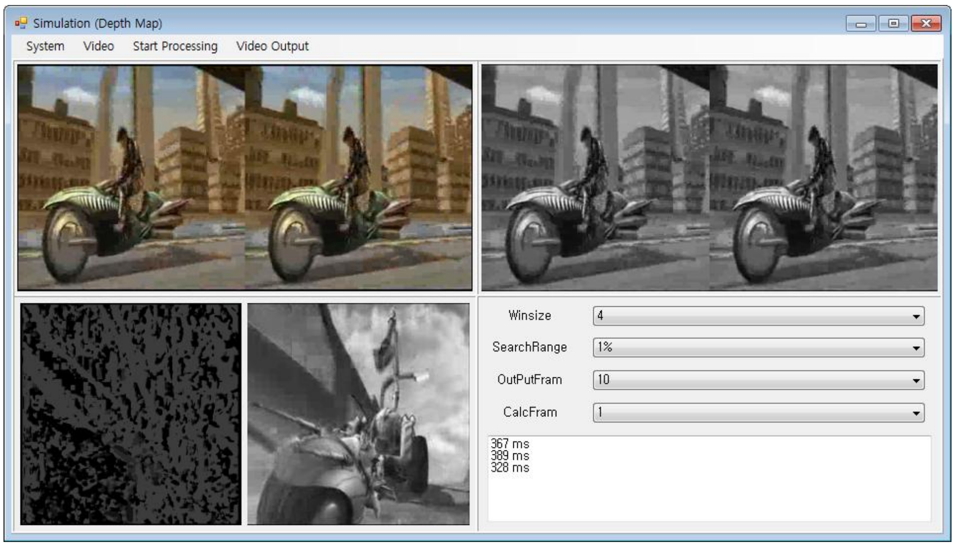
Items	Descriptions and Values
File Name	SampleVideo.avi
File Size	761 KB
Format	AVI
Duration	7733 ms
Bitrate	692 Kbps
Width	800
Height	600
Frame Rate	15.00 fps
Video Codec	MPEG-4 (DivX 4) (Simple@L1)
Audio Codec	MPEG Audio (MP3) (Version 1) (Layer 3) (Joint Stereo / MS Stereo)

Table 2. Simulation hardware

Type	Laptop
CPU	Intel® Core™ i7 4700MQ Processor
Chipset	Intel® HM87
RAM	KINGMAX DDR3 8GB x 2 (ea)
GPU	Nvidia GeForce GTX 765M / 2GB GDDR5
HDD	HGST 1TB 7200RPM

Table 3 shows the simulation results and **Fig. 8** depicts the average performance improvement over the processing time of normal, the serial implementation version. The performance improvements are normalized to the normal execution time.

SSE shows significant performance improvement as the growing of window sizes shown in **Fig. 9**. This result signifies that the computation of SAD algorithm is a typical data intensive computing which use a data parallel approach to processing large volumes of data with heavy memory accesses; but relatively low computational complexity. As for OpenMP, it is necessary to select the number of thread by users. We used 4 threads. The increase of number of threads over 4 resulted in the degradation of the overall performance. This was because the increased threads caused the overhead of thread distribution of master thread.



Winsize 4
SearchRange 1%
OutPutFram 10
CalcFram 1

```
367 ms
389 ms
328 ms
```

Fig. 7. UI for simulation

Table 3. Simulation results

		Elapsed Time Average			Time Reduced		
WinSize	Search Range	Normal	OpenMP	SSE	Normal	OpenMP	SSE
4	1	387.6637931	234.2758621	112.8189655	100%	60%	29%
	5	685.6293103	540.0258621	468.4396552	100%	79%	68%
	10	1345.568966	936.5258621	264.7844828	100%	70%	20%
8	1	533.4827586	470.8534483	340.9137931	100%	88%	64%
	5	2630.5	1437	405.7155172	100%	55%	15%
	10	5190.310345	1328.112069	984.112069	100%	26%	19%
16	1	2063.293103	1243.025862	438.5086207	100%	60%	21%
	5	10240.09483	2095.448276	1828.112069	100%	20%	18%
	10	19854.65517	4464.241379	3583.568966	100%	22%	18%

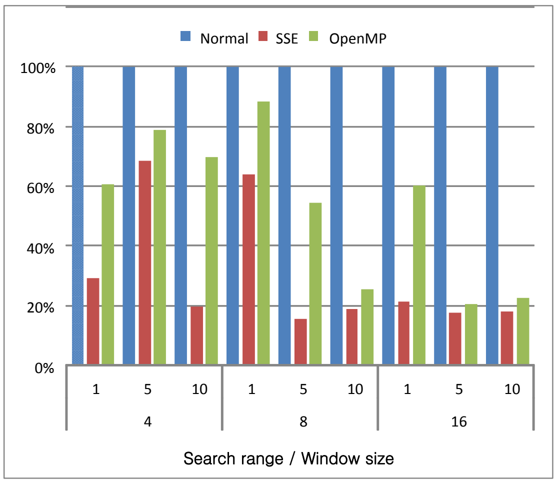


Fig. 8. Performance Improvement on three versions

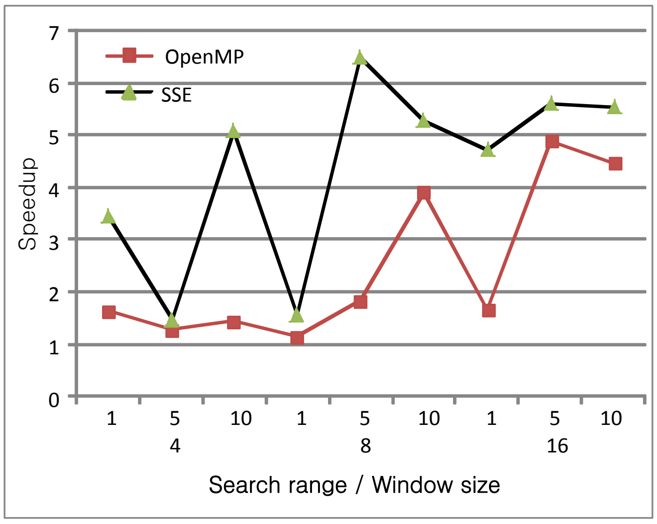


Fig. 9. The ratio of execution time on OpenMP and SSE

5. Conclusion

This paper introduced parallel SAD implementations for the fast depth map computation on multi-core CPU. For this purpose, we used a parallel algorithm for calculating disparity map using OpenMP, a shared memory programming which can provide the advantage to simplify managing and synchronization of program threads. Also, this paper introduced the effectiveness of exploiting the streaming SIMD extension instructions. As a result, we were able to take advantage of all the hardware and software features of current multi-core processors. The performance evaluation was carried out on stereoscopic streaming video rather than a stereoscopic image pair and the processing times of each implementation were measured. The experimental results show that both technologies of OpenMP and SSE have a significant effect on the performance and achieve great improvements on processing speed. Especially, the SSE implementation could achieve more performance improvements than the OpenMP implementation.

References

[1] C. G. Kim, "A Characteristic Analysis Study of Android based Stereoscopic 3D Technology," *The Journal of Korea Society of Communication and Space Technology*, Vol. 8, No. 2, pp. 68-73, June 2013. Article (CrossRef Link)

[2] C. G. Kim, V. P. Sirni, and S. D. Kim, "High Performance Coprocessor Architecture for Real-Time Dense Disparity Map," *The KIPS Transactions: Part A*, Vol. 14-A, No. 5, pp. 301-308, Oct. 2007. Article (CrossRef Link)

[3] S. T. Barnard and W. B. Thompson, "Disparity analysis of images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 2, No. 4, pp. 333–340, 1980. Article (CrossRef Link)

[4] M. Z. Brown, D. Burschka, and G. D. Hager, "Advances in computational stereo," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 25, Issue 8, pp. 993-1008, Aug. 2003. Article (CrossRef Link)

[5] H. Sunyoto, W. van der Mark, and D. M. Gavrila, "A comparative study of fast dense stereo vision algorithms," *in Proc. of IEEE Intelligent Vehicles Symposium 2004*, pp. 319-324, 14-17 June 2004. DOI: Article (CrossRef Link)

[6] A. Kuhl, "Comparison of Stereo Matching Algorithms for Mobile Robots," Technische Universität Ilmenau, 2004. Article (CrossRef Link)

[7] T. Kanade, A. Yoshida, K. Oda, H. Kano, and M. Tanaka, "A stereo machine for video-rate dense depth mapping and its new applications," *in Proc. of IEEE CVPR '96*, pp. 196-202, 1966. 10.1109/CVPR.1996.517074. Article (CrossRef Link)

[8] J. Y. Goulermas and P. Liatsis, "Feature-based stereo matching via coevolution of epipolar subproblems," in *Proc. of Seventh International Conference on Image Processing And Its Applications*, Vol. 1, pp. 23-27, 13-15 July 1999. Article (CrossRef Link)

[9] D. Fleet, A. Jepson, and M. Jenkin, "Phase-based disparity measurement," *CVGIP: Image Understanding*, Vol. 53, pp. 198-210, 1991. Article (CrossRef Link)

[10] N. Lazaros, G. C. Sirakoulis, and A. Gasteratos, "Review of stereo vision algorithms: From software to hardware," *International Journal of Optomechatronics*, vol. 2, pp. 435–462, 2008. Article (CrossRef Link)

[11] D. Scharstein and R. Szeliski, "A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms," *International Journal of Computer Vision*, Vol. 47, Issue 1-3, pp. 7-42, April-June 2002. Article (CrossRef Link)

[12] H. Hirschmuller and D. Scharstein, "Evaluation of cost functions for stereo matching," in *Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1–8, June 2007. Article (CrossRef Link)

[13] Y. Pang, W. D. Hu, L. F. Sun, and S. Q. Yang, "Adaptive data-driven parallelization of multi-view video coding on multi-core processor," *Science in China Series F: Information Sciences 2009*, Vol. 52 No. 2, pp. 195-205, 2009. Article (CrossRef Link)

[14] Y. Yang, G. Jiang, M. Yu, and D. Zhu, "Parallel process of hyper-space-based multiview video compression," in *Proc. of* 2006 IEEE International Conference on Image Processing, pp. 521 – 524, 8-11 Oct. 2006. Article (CrossRef Link)

[15] B. Zatt, M. Shafique, S. Bampi, and J. Henkel, "Multi-Level Pipelined Parallel Hardware Architecture for High Throughput Motion and Disparity Estimation in Multiview Video Coding," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 14-18 March 2011. Article (CrossRef Link)

[16] C. Banz, S. Hesselbarth, H. Flatt, H. Blume, and P. Pirsch, "Real-time stereo vision system using semi-global matching disparity estimation: Architecture and FPGA-Implementation," in *Proc. of 2010 International Conference on Embedded Computer Systems (SAMOS)*, 19-22 July 2010. Article (CrossRef Link)

[17] J. Diaz, E. Ros, R. Carrillo, and A. Prieto, "Real-time system for high-image resolution disparity estimation," *IEEE Transactions on Image Processing*, Vol. 16, No. 1, pp 280-285, Jan. 2007. Article (CrossRef Link)

[18] Ahmad Darabiha, W. James MacLean, Jonathan Rose, Reconfigurable hardware implementation of a phase-correlation stereo algorithm, Machine Vision and Applications, Springer-Verlag (2006) 17(2): 116–132. Article (CrossRef Link)

[19] J. Chhugani, M. Macy, A. Baransi, A. D. Nguyen, M. Hagog, S. Kumar, V. W. Lee, P. Dubey, and Y. K. Chen, "Efficient implementation of sorting on multi-core SIMD CPU architecture," *Journal: Proceedings of the VLDB Endowment*, Vol. 1, Issue2, pp. 1313–1324, 2008. Article (CrossRef Link)

[20] A. Kayi, Y. Yao, T. El-Ghazawi, and G. Newby, "Experimental evaluation of emerging multi-core architectures," *In Proceeding of IPDPS 2007*, pp. 1–6, 2007. Article (CrossRef Link)

[21] C. G. Kim and Y. S. Choi, "A High Performance Parallel DCT with OpenCL on Heterogeneous Computing Environment," *Multimedia Tools and Applications*, Vol. 64, Issue 2, pp 475-489, May 2013. Article (CrossRef Link)

[22] H. Blume, J. von Livonius, L. Rotenberg, H. Bothe, J. Brakensiek, and T. G. Noll, "OpenMP-based parallelization on an MPCore multiprocessor platform – A performance and power analysis,"

Journal of Systems Architecture, Vol. 54, pp. 1019–1029, 2008. Article (CrossRef Link)

[23] A. Peleg and U. Weiser, “MMX technology extension to the Intel architecture,” *IEEE Micro*, Vol. 16, Issue 4, pp. 42–50, 1996. Article (CrossRef Link)

[24] Y. Song and Y. S. Ho, “Fast Disparity Map Estimation Using Multi-thread Parallel Processing,” in *Proc. of International Conference on Embedded Systems and Intelligent Technology (ICESIT)*, pp. 1 – 4, 2011. Article (CrossRef Link)

[25] C. G. Kim, D. H. Lee, and J. G. Kim, "Optimizing Image Processing on Multi-core CPUs with Intel Parallel Programming Technologies," *Multimedia Tools and Applications*, Vol. 68, Issue 2, pp 237-251, Jan. 2014. Article (CrossRef Link)

[26] C. G. Kim, "Parallel SAD for Fast Dense Disparity Map Using a Shared Memory Programming," *Information Technology Convergence*, Vol. 2, pp. 1055-1060, Jul. 2013. Article (CrossRef Link)

Cheong Ghil Kim received the B.S. in Computer Science from University of Redlands, CA, U.S.A. in 1987. He received the M.S. and Ph.D. degree in Computer Science from Yonsei University, Korea, in 2003 and 2006, respectively. Currently, he is a professor at the Department of Computer Science, Namseoul University, Korea. His research areas include Multimedia Embedded Systems, Mobile AR, and 3D Contents.

YongSoo Choi was born in Kangwon Do, Korea, in 1972. He received the B.S., M.S. and Ph.D. degrees in the Department of Instrumentation and Control Engineering from the Kangwon National University, Korea, in 1998, 2000 and 2006, respectively. From 2006 to 2007, he was a research professor with the Center for Technology Fusion in Construction, YonSei University, Korea. From 2007 to 2013, he was a research professor with the Brain Korea 21 of Ubiquitous Information Security, Korea University, Korea. He is currently a Assistant Professor with the Division of Liberal Arts & Teaching, Sungkyul University, Korea. From 2013 to present, he was a Delegate of Korea for ISO/IEC JTC1/SC29. His research interests include multimedia signal processing, digital watermarking, steganography and multimedia hashing. Dr. Choi is a member of the IEIE Computer Society. He is currently a Editor in Chief of Journal of the Institute of Electronics Engineers of Korea, the Section of Computer and Information, Korea. He also serve as Reviewer in Journal of Multimedia Tools and Applications, LNCS Transactions on Data Hiding and Multimedia Security and Transactions on Internet and Information Systems.