Parallelizing H.264 and AES Collectively

Heegon Kim[1], Sungju Lee[1], Yongwha Chung[1], and Sung Bum Pan[2]
[1] Dept. of Computer and Information Science,
Korea University, Sejong, Korea
[e-mail: khg86,peacfeel,ychungy@korea.ac.kr]
[2] Dept. of Control, Instrumentation and Robot Engineering,
Chosun University, Gwangju, Korea
sbpan@chosun.ac.kr

*Corresponding author: Yongwha Chung and Sung Bum Pan

Received April 15, 2013; revised August 5, 2013; accepted September 5, 2013; published September 30, 2013

Abstract

Many applications can be parallelized by using multicore platforms. We propose a load-balancing technique for parallelizing a whole application, whose first module (H.264) has data independency and whose second module (AES) has data dependency. Instead of distributing the first module symmetrically over the multi-core platform, we distribute the data-independent workload asymmetrically in order to start the data-dependent workload as early as possible. Based on the experimental results with a compression/encryption application, we confirm that the asymmetric load balancing can provide better performance than the typical symmetric load balancing.

Keywords: Multicore, Amdahl's law, load balancing

This work was supported by a grant from Korea University.

http://dx.doi.org/10.3837/tiis.2013.09.015

1. Introduction

Multicore processors have been used for handheld devices as well as PCs and servers, and parallel processing techniques have been developed for many applications [1] [2] [3] [4]. For example, many approaches have been reported to parallelize multimedia workloads [5] [6]. Many users create their own content using handheld devices such as smartphones as they become powerful. As a result, the growing amount of sensitive information shared by handheld device users raises serious privacy concerns, and motivates the need for appropriate privacy-preserving mechanisms [7].

In this study, we focus on parallelizing a whole application that first compresses input video and then encrypts it for privacy protection. We assme we have multiple video files which should be compressed and then encrypted for a secure transmission. This application has interesting characteristics in that the compression workload has data independency among the video data blocks, and the encryption workload has data dependency among the compressed video data blocks. In general, many parallelization techniques have been developed for each workload individually. However, parallelizing the whole application collectively has rarely been reported. Typically, this application can be parallelized by distributing the first workload evenly across the cores, and then executing the second workload sequentially by a single core. The performance obtained by this typical parallelization is limited according to Amdahl's law [8] [9] [10] [11].

We propose a load balancing technique that can overcome the performance limitations of Amdahl's law. In order to reduce the idle time caused by the data dependency in the second workload, we distribute the data-independent first workload unevenly across the cores and keep the data dependency in the second workload. By doing so, we can provide better performance than Amdahl's law computed with typical load balancing (*i.e.*, distributing the first workload symmetrically).

The rest of the paper is structured as follows. Section 2 explains the characteristics of H.264 multimedia compression [12] and AES encryption [13]. Section 3 describes the proposed asymmetric load balancing technique. The experimental results are given in Section 4, and conclusions are provided in Section 5.

2. Background

2.1 H.264

Currently, one of the best video coding standards in terms of compression performance is H.264 [12]. H.264/MPEG-4 Part 10 or AVC (Advanced Video Coding) is a standard for video compression, and is one of the most commonly used formats for the recording, compression, and distribution of high-definition video. The coding efficiency gains of advanced video codecs such as H.264 come at the price of increased computational requirements. The demands for computing power also increase with the shift towards high-definition resolutions.

Since the size of most multimedia data is substantial, a special-purpose compression chip can be employed to satisfy the real-time requirements in a handheld device. However, current high-performance uniprocessor architectures are not capable of providing the required performance for real-time processing [14]. Therefore, it is necessary to exploit parallelism. The H.264 codec can be parallelized either by task-level or data-level decomposition.

Parallel implementations of H.264 encoding have been reported. Rodriguez et al. [14] implemented an H.264 encoder using Group of Pictures (GOP) and slice-level parallelism on a cluster of workstations with Message Passing Interface (MPI). Although real-time operation can be achieved with such an approach, the latency is very high. As multicore processors have been increasingly used for mobile handheld devices [4], we use PARSEC H.264 [15] which is a pthread-based [16] parallel H.264.

2.2 AES (Advanced Encryption Standard)

To ensure the confidentiality of data for the next few decades, the National Institute of Standards and Technology (NIST) has proposed five modes of block encryption operation in 2001 [13] in order to replace DES (Data Encryption Standard) which was adopted in 1977. For example, the simplest of the encryption modes is the Electronic CodeBook (ECB) mode. A message is divided into blocks, and each block is encrypted separately. The disadvantage of this mode is that identical plaintext blocks are encrypted into identical ciphertext blocks; thus, data patterns are not hidden well. To overcome the security deficiencies of ECB, a technique in which the same plaintext block produces different ciphertext blocks is needed. A simple way to satisfy this requirement is the Cipher Block Chaining (CBC) mode, where the input to the encryption algorithm is the XOR of the current plaintext block and the preceding ciphertext block.

In AES-CBC encryption, there is no parallelism at the block level, as the encryption of each AES block of 128 bits depends on the previous block’s encrypted results. Also, secure multimedia applications generally first require compression (with enough parallelism) in order to reduce the size of the data encrypted. Therefore, a way is needed to efficiently parallelize an application whose first module has data independency and whose second module has data dependency

3. Asymmetric Load Balancing

To understand our target application’s computational characteristics, we explain this application with a 4-core processor execution. **Fig. 1** shows the data dependencies in the target application. 1-1, 1-2, 1-3, and 1-4 denote the raw video data to be compressed by cores 1, 2, 3, and 4, respectively. Then, 2-1, 2-2, 2-3, and 2-4 denote the compressed video data to be encrypted by cores 1, 2, 3, and 4, respectively. The first module consists of four independent operations (1-1, 1-2, 1-3, and 1-4), whereas the second module consists of four dependent operations (2-1, 2-2, 2-3, and 2-4). Each operation of the second module has data dependency with the previous operations, both in the first module and the second module. For example, operation 2-4 should be performed after executing operations 1-4 and 2-3.

In this target application, the performance gain obtained by typical parallelization is limited by Amdahl’s law [9], as shown in **Fig. 2**. To provide better performance, we propose a load balancing technique that can overcome these limitations. In order to reduce the idle time caused by the data dependency in the second module, we distribute the data-independent first module unevenly across the cores, as shown in **Fig. 3**. By distributing the first module asymmetrically and keeping the data dependency in the second module, better performance can be provided than that suggested by Amdahl’s law with typical load balancing (*i.e.*, distributing the first module symmetrically). Note again that, we have multiple cores/platforms having the same computing capability, and typical load balancing approaches distribute the same amount of work to the same capability core/platform [17][18][19].

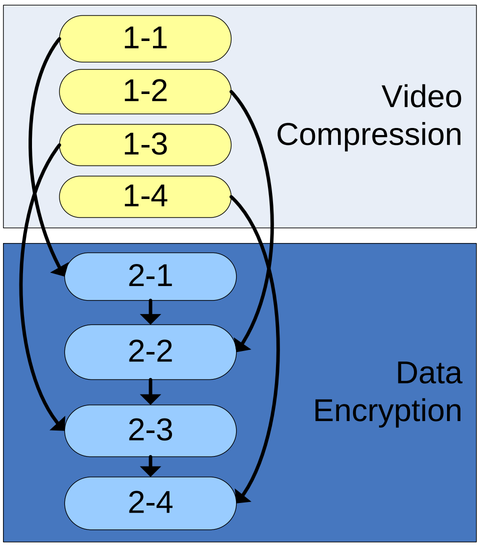


Fig. 1. Data dependencies in the target application

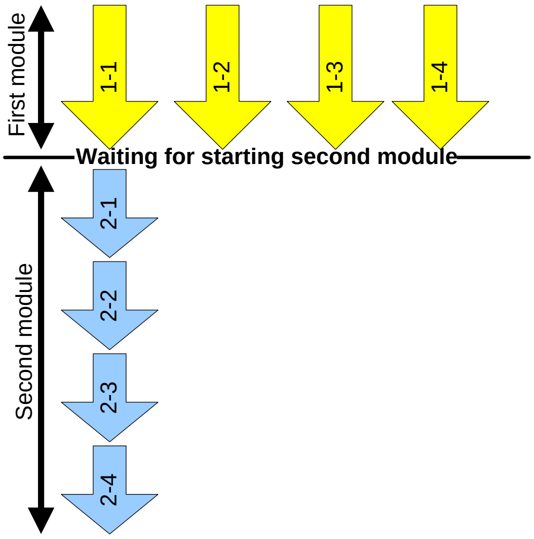


Fig. 2. Symmetric load balancing

That is, in typical load balancing, the operations in the first module are executed symmetrically by 4 cores before barrier synchronization, and then the operations in the second module are executed sequentially by 1 core due to its data dependency (See **Fig. 2**). In contrast, the proposed load balancing approach assigns the four operations in the first module asymmetrically in order to reduce the idle time in the second module caused by the data dependency (See **Fig. 3**). For example, operation 2-1 in the second module can be executed immediately after operation 1-1 in the first module without waiting for operation 1-2. After executing operation 2-1, cores 1 and 2 are synchronized using a *mutex* [16], and then operation 2-2 is performed. In a similar way, cores 1 and 3 are synchronized, and then 2-3 operation is performed.

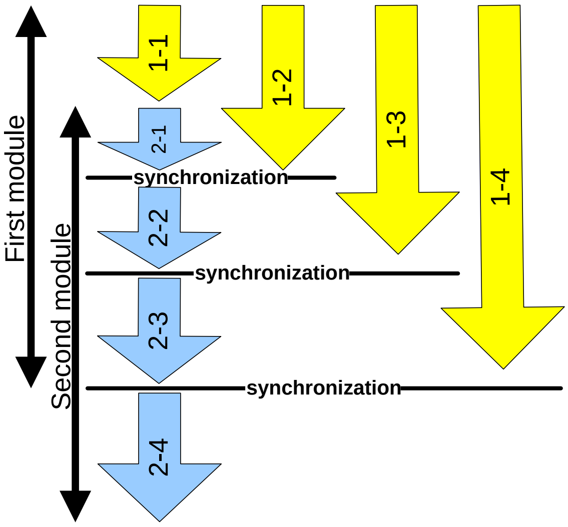
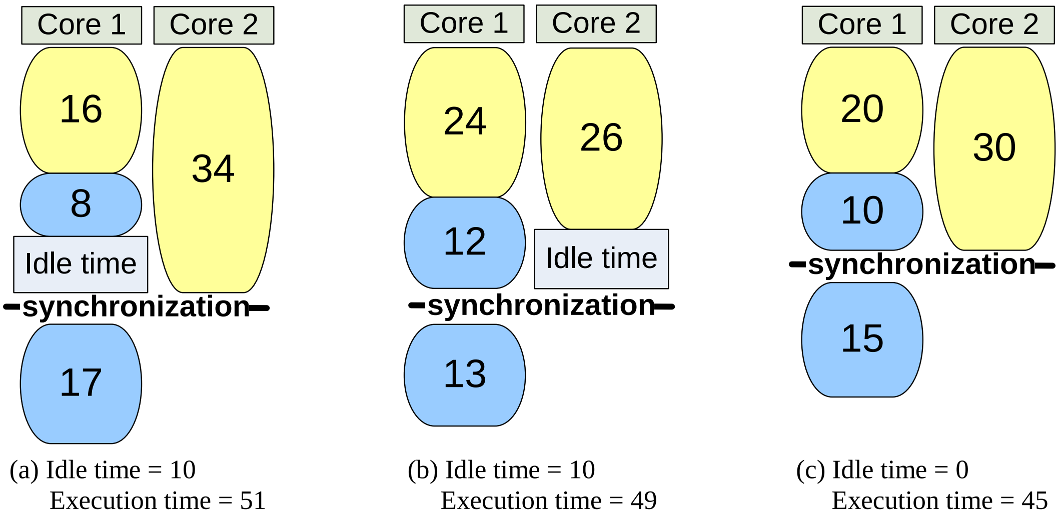


Fig. 3. Asymmetric load balancing

The determination of the amount of workload assigned to each core is explained as follows. We assume the second module is executed sequentially on core 1. As shown in **Fig. 4**, the idle time can be minimized when the sum of the first module and the second module on core 1 is equal to the execution time of the first module on core 2. Let n, P, W_T, and W_i denote the number of cores, the degree of parallelism of the application (*i.e.*, P=0.9 for 90% parallelism), the total workload (*i.e.*, $\text{workload}_{\text{first\_module}} + \text{workload}_{\text{second\_module}}$), and the amount of workload of the first module assigned to the i-th core, respectively. Then, W_i can be computed using (1) and (2).



(a) Idle time = 10
Execution time = 51

(b) Idle time = 10
Execution time = 49

(c) Idle time = 0
Execution time = 45

Fig. 4. Idle times caused by the workload assigned to each core

$$W_1 = \frac{W_T \times P}{\sum_{i=1}^{n} (\frac{1}{P})^{i-1}} \quad (1)$$

$$W_i = W_1 \times (\frac{1}{P})^{i-1} \text{ if } i \geq 2 \quad (2)$$

Since the execution time depends on the assigned workload, we can estimate the execution time of the asymmetric load balancing. The execution time of module 1 (without data dependency) and module 2 (with data dependency) can be computed using (3) and (4), where TP_n and TS_n are the execution time of module 1 and module 2 on the n-th core.

$$TP_n = \frac{W_T \times P}{\sum_{i=1}^{n} (\frac{1}{P})^{i-1}} \times (\frac{1}{P})^{n-1} \quad (3)$$

$$TS_n = \frac{\frac{W_T \times P}{\sum_{i=1}^{n} (\frac{1}{P})^{i-1}} \times (\frac{1}{P})^{n-1}}{\frac{1}{P} - 1} \quad (4)$$

Finally, the total execution time of the asymmetric load balancing can be computed by using (5).

$$T_{Total} = TP_n + TS_n$$
$$= \frac{W_T \times P}{\sum_{i=1}^{n} (\frac{1}{P})^{i-1}} \times (\frac{1}{P})^{n-1} + \frac{\frac{W_T \times P}{\sum_{i=1}^{n} (\frac{1}{P})^{i-1}} \times (\frac{1}{P})^{n-1}}{\frac{1}{P} - 1} \quad (5)$$

4. Experimental Results

4.1 Estimated and Measured Speedup

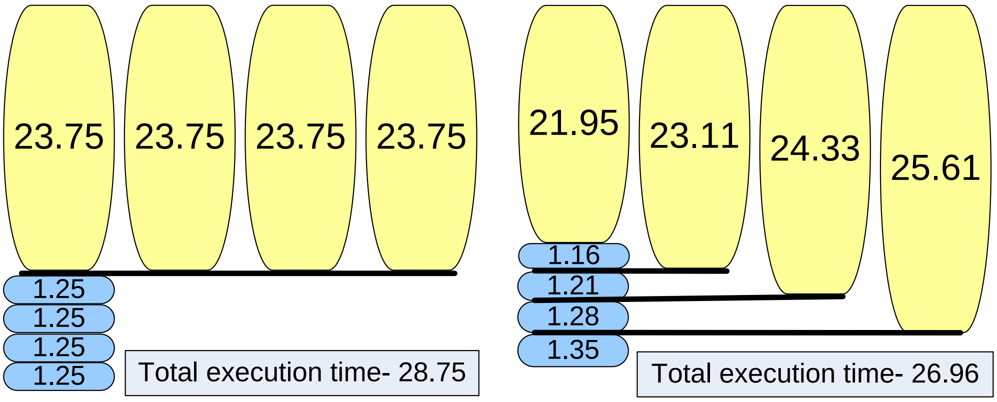
For evaluating the proposed approach, we parallelized the target application (video compression/encryption) with *pthread* on real machines (core2 duo E8400, i5-750, and i5-2500 multicore processors). We also used a data set of 25 YUV frames (4:2:0 format) composed of three frames per video sequence. The size of an experimental video image was 352×288 (CIF format), and the number of frames was 2400.

First, the execution time of multimedia compression and data encryption was measured. H.264 was used for multimedia compression, and AES-CBC was used for data encryption with CIF data (which was about 348 MB). **Table 1** shows that the video encoding time was longer than the encryption time by a factor of about 16. It is because the video encoding techniques generally have higher computational complexity than the encryption techniques, and the data size for encryption is already reduced by the preceding compression.

Table 1. Execution time of compression and encryption with 1 core

	Multimedia Compression (H.264)	**Data Encryption (AES-CBC)**
core2 duo E8400	3869 msec	241 msec
i5-750	3245 msec	254 msec
i5-2500	2761 msec	165 msec

An execution time ratio of about 95:5 was obtained, and the typical symmetric load balancing of H.264 is shown in **Fig. 5**(a). In this application, the asymmetric load balancing of H.264 shown in **Fig. 5**(b) can provide faster execution time. In the symmetric load balancing, the performance is limited by Amdahl's law. If the execution time of the parallel portion (*i.e.*, compression) is 95 seconds and the execution time of the sequential portion (*i.e.*, encryption) is 5 seconds, the ideal speedup of the symmetric load balancing with 4 cores is limited by 3.48. However, the asymmetric load balancing can provide an ideal speedup of 3.71.



(a) Symmetric load balancing (b) Asymmetric load balancing

Fig. 5. An illustration of symmetric/asymmetric load balancing of H.264/AES-CBC workload with 4 cores (Sequential execution time of 100 is assumed.)

Fig. 6, **Fig. 7**, and **Fig. 8** show the comparison of the speedup of the typical and proposed approaches. The three real machines (Core2 Duo E8400, i5-750, and i5-2500) have different numbers of cores, the first having 2 cores, and the latter two each having 4 cores. As shown in **Fig. 6**, **Fig. 7**, and **Fig. 8**, the proposed load balancing approach can provide better performance than the performance obtained by Amdahl's law with typical symmetric load balancing. That is, our load balancing technique can overcome the performance limitations of Amdahl's law, and efficiently apply the multimedia compression/encryption application to the multicore platforms.

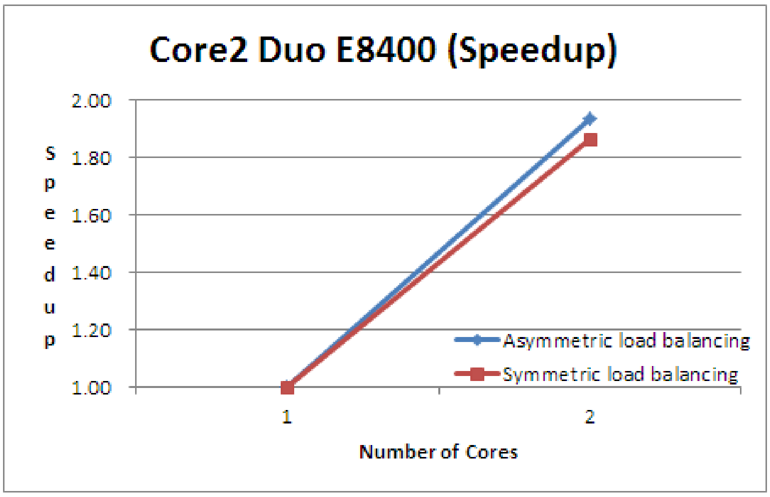


Fig. 6. Speedup on Core2 Duo E8400

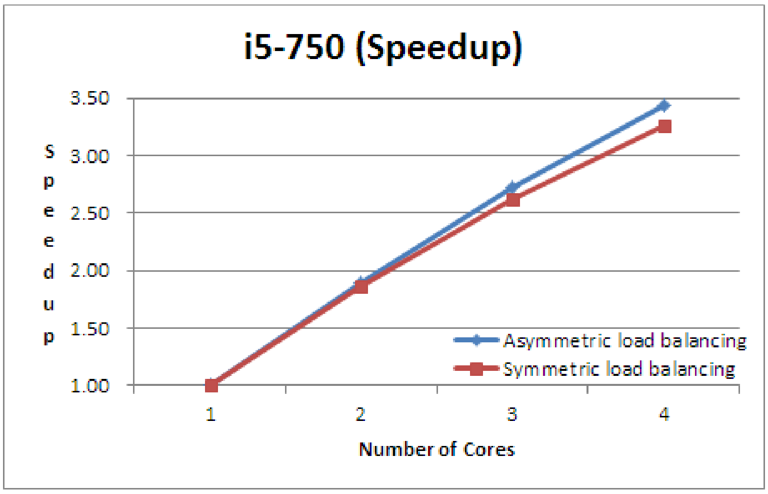


Fig. 7. Speedup on i5-750

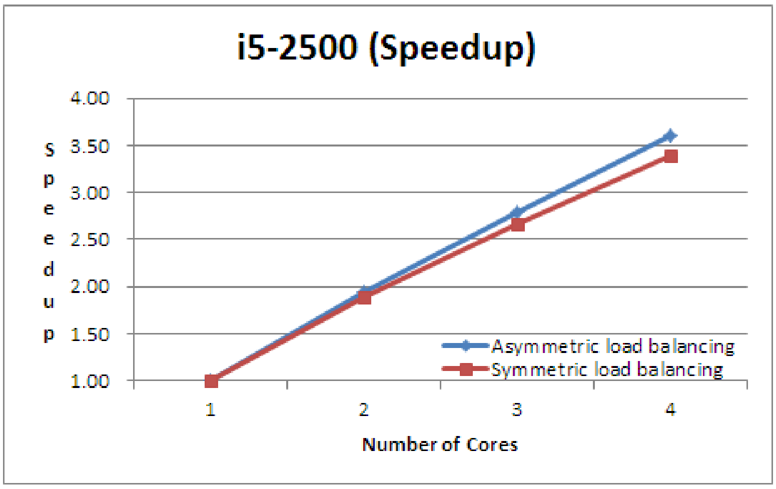


Fig. 8. Speedup on i5-2500

4.2 Analysis

To compare the speedup for the given environments with different application parallelism and numbers of cores, the speedup of symmetric/asymmetric load balancing approaches were estimated in various conditions. First, to verify the accuracy, the estimated results were compared with measured results, as shown in **Table 2**, which shows the estimated and measured speedup on the three real machines. The results show our proposed load balancing technique can overcome the performance limitations of Amdahl's law.

Table 2. Estimated/Measured speedup

	Symmetric load balancing		**Asymmetric load balancing**	
	estimated	measured	estimated	measured
core2 duo E8400 (2 cores)	1.90	1.87	1.95	1.94
i5-750 (4 cores)	3.48	3.27	3.71	3.44
i5-2500 (4 cores)	3.48	3.39	3.71	3.60

Finally, we observed the speedup according to the number of cores and the application parallelism. **Fig. 9** and **Fig. 10** show each speedup for application having 95% and 75% parallelism, respectively. The number of cores ranged from 1 to 8. As shown in **Fig. 9** and **Fig. 10**, the proposed approach has improved speedup compared to the typical approach. With the application having 95% parallelism on the 8-core processor, the typical approach provides the speedup of 5.93, while the proposed approach provides the speedup of 6.73. Based on the estimated results, we confirmed that the proposed approach can provide better speedup than Amdahl's law computed with typical load balancing as the number of cores increased.

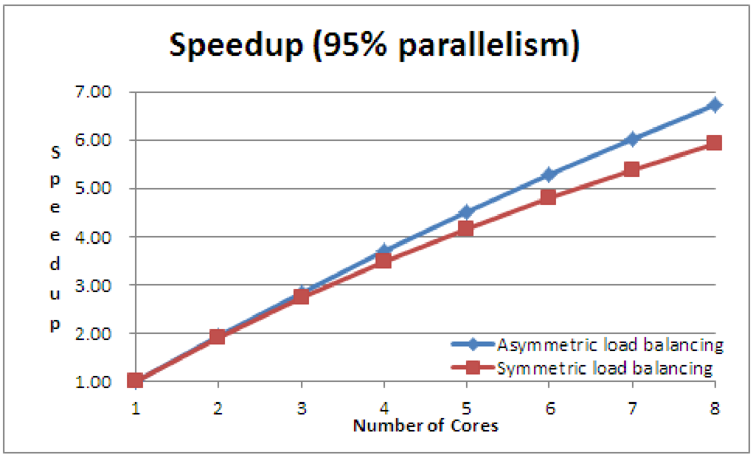


Fig. 9. Speedup of the application having 95% parallelism

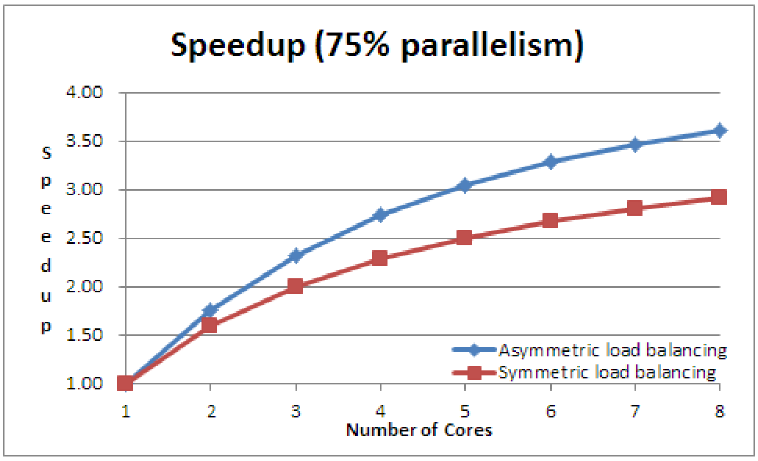


Fig. 10. Speedup of the application having 75% parallelism

5. Conclusion

We have proposed a load balancing technique for parallelizing an application whose first module has data independency and whose second module has data dependency. By distributing the first module asymmetrically and keeping the data dependency in the second module, we can provide better performance than Amdahl's law computed with typical load balancing. With all combinations of the number of cores and the degree of parallelism examined, we have derived the amount of workload assigned to each core.

Experiments with H.264/AES have demonstrated that our load balancing technique can be useful in parallelizing applications whose first module has data independency and whose second module has data dependency (*i.e.*, compressing the video data first, and then encrypting the compressed data). The proposed load balancing technique can be easily extended to the applications with reversed conditions characterized by a first module with data dependency and second module with data independency (*e.g.*, decrypting the encrypted data first, and then decompressing the compressed video data).

References

[1] J. Held, J. Bautista, and S. Koehl, "From a Few Cores to Many: A Tera-Scale Computing Research Overview," *Intel White Paper*, pp.1-12, January 1, 2006. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.114.4764

[2] K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson, W. Plishker, J. Shalf, S. Williams, and A. Yelick, "The Landscape of Parallel Computing Research: A View from Berkeley," *Technical Report*, No. UCB/EECS-2006-183, pp.1-54, December 18, 2006. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.227.1678

[3] S. Borkar, "Thousand Core Chips: A Technology Perspective," in *Proc. of 44th Design Automation Conf.*, pp. 746-749, June 4-8, 2007. Article (CrossRef Link).

[4] M. Levy and T. Conte, "Embedded Multicore Processors and Systems," *IEEE Micro*, vol. 29, no. 3, pp. 7-9, May-June, 2009. Article (CrossRef Link).

[5] K. Sihn, H. Baik, J. Kim, S. Bae, and J. Song, "Novel Approaches to Parallel H.264 Decoder on Symmetric Multicore Systems," in *Proc. of Int. Conf. on Acoustics, Speech, and Signal Processing*, pp. 2017-2020, April 19-24, 2009. Article (CrossRef Link).

[6] I. Ahmad, Y. He, and M. Liou, "Video Compression with Parallel Processing," *Parallel Computing*, vol. 28, no. 7-8, pp. 1039-1078, August, 2002. Article (CrossRef Link).

[7] E. Cristofaro, A. Durnssel, and I. Aad, "Reclaiming Privacy for Smartphone Applications," in *Proc. of IEEE Int. Conf. on Pervasive Computing and Communication*, pp. 84-92, March 21-25, 2011.
Article (CrossRef Link).

[8] G. Amdahl, "Validity of the Single-Processor Approach to Achieving Large-Scale Computing Capabilities," in *Proc. of the American Federation of Information Processing Societies 1967 Spring Joint Computer Conf.*, pp. 483-485, April 18-20, 1967. Article (CrossRef Link).

[9] J. Gustafson, "Reevaluating Amdahl's Law," *Comm. of the ACM*, vol. 31, no. 5, pp. 532-533, May, 1988. Article (CrossRef Link).

[10] J. Hennessy and D. Patterson, *Computer Architecture*, Elsevier, Amsterdam, 2006.
http://textbooks.elsevier.com/web/product_details.aspx?isbn=9780123838728

[11] M. Hill and M. Marty, "Amdahl's Law in the Multicore Era," *IEEE Computer*, vol. 41, no. 7, pp. 33-38, July, 2008. Article (CrossRef Link).

[12] T. Wiegand, H. Sillivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560-576, July, 2003. Article (CrossRef Link).

[13] U.S. National Institute of Standards and Technology, "The Advanced Encryption Standard," *Federal Information Processing Standard Publication 197*, pp. 1-47, November 26, 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[14] A. Rodriguez, A. Gonzalez, and M. Malumbres, "Hierarchical Parallelization of an H.264/AVC Video Encoder," in *Proc. of Int. Symposium on Parallel Computing in Electrical Engineering*, pp. 363-368, September 13-17, 2006. Article (CrossRef Link).

[15] C. Bienia, S. Kumar, J. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *Proc. of Int. Conf. on Parallel Architectures and Compilation Techniques*, pp. 72-81, October 25-29, 2008. Article (CrossRef Link).

[16] D. Butenhof, *Programming with POSIX threads*, Addison-Wesley, Boston, 1997.
http://dl.acm.org/citation.cfm?id=263953

[17] S. Akhter and J. Roberts, *Multi-Core Programming - Increasing Performance through Software Multi-Threading*, Intel Press, Hillsboro, 2006.
http://noggin.intel.com/intelpress/categories/books/multi-core-programming

[18] M. Kim, S. Han, Y. Cui, H. Lee, and C. Jeong, "A Hadoop-based Multimedia Transcoding System for Processing Social Media in the PaaS Platform of SMCCSE," *KSII Transactions on Internet and Information Systems*, vol. 6, no. 11, pp. 2827-2848, November, 2012. Article (CrossRef Link).

[19] J. Diaz, C. Munoz-Caro, and A. Nino, "A Survey of Parallel Programming Models and Tools in the Multi and Many-Core Era," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 8, pp. 1369-1386, August, 2012. Article (CrossRef Link).

Heegon Kim is currently a PhD student in the Dept. of Computer and Information Science at Korea University. He received his MS degree from Korea University in 2013 and his BS degree from Korea University in 2011. His research interests include parallel processing, energy efficiency, image processing.

Sungju Lee received his PhD degree in the Dept. of Computer and Information Science at Korea University in 2012. He received his MS degree from Korea University in 2008 and his BS degree from Korea University in 2006. His research interests include multi-core, energy efficiency, information security.

Yongwha Chung received the BS and MS degrees from Hanyang University, Korea, in 1984 and 1986. He received the PhD degree from the University of Southern California, USA, in 1997. He worked for ETRI from 1986 to 2003 as a Team Leader. Currently, he is a professor in the Dept. of Computer and Information Science, Korea University. His research interests include parallel processing, information security, and performance optimization.

Sung Bum Pan received his PhD degree in Electronics Engineering from Sogang University, Korea, in 1999. He was a team leader at Biometric Technology Research Team of ETRI from 1999 to 2005. He is now an associate professor at Chosun University. His current research interests are multimedia, security, and VLSI architectures for real-time image processing.