On Running Data-Intensive Algorithms with Intelligent SSD and Host CPU: A Collaborative Approach

Yong-Yeon Jo
Dept. of Computer & Software
Hanyang University, Korea
jyy0430@hanyang.ac.kr

SungWoo Cho
Dept. of Computer & Software
Hanyang University, Korea
nera@hanyang.ac.kr

Sang-Wook Kim*
Dept. of Computer & Software
Hanyang University, Korea
wook@hanyang.ac.kr

Duck-Ho Bae
Dept. of Electronics and
Computer Engineering
Hanyang University, Korea
dhbae@agape.hanyang.ac.kr

Hyunok Oh
Dept. of Information Systems
Hanyang University, Korea
hoh@hanyang.ac.kr

ABSTRACT

A solid state device (SSD), which has the characteristics such as high IO bandwidth and low access latency, is drawing attention as a next-generation storage device. Even though SSD provides a high internal bandwidth, the performance bottleneck exists on the host interface of relatively low bandwidth in spite of the increased internal bandwidth of SSD. To overcome the performance bottleneck, the notion of intelligent SSD (iSSD) has been proposed. In iSSD, there are still problems in processing the algorithms of high complexity. In this paper, we address an effective collaboration of iSSD and host CPU in order to maximize the performance of data-intensive algorithms. Extensive experimental results show that our approach performs faster up to 2.43 times than a previous approach.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]

General Terms

Performance

Keywords

SSD, collaborative approach, heterogeneous, data-intensive algorithms, scheduling

1. INTRODUCTION

Due to an explosive growth of Internet applications such as Internet business, online shopping, emails, and social networking services, the amount of data has been created enormously. In order to store and process this big data efficiently, a solid state device (SSD), which has nice characteristics such as high IO bandwidth and low access latency, is drawing attention as a next-generation storage device [1][2][3].

Figure 1 represents the internal structure of SSD. Each channel of SSD is composed of multiple NAND flash memory cells for storing data, a flash memory controller (FMC) to control the cells, and SRAM and DRAM to store the data to be read/written from/to the cells. All the channels in SSD are controlled by firmware called *flash translation layer (FTL)*. Data are transferred between SSD and Host DRAM through a host interface, which is controlled by the host interface controller [4].

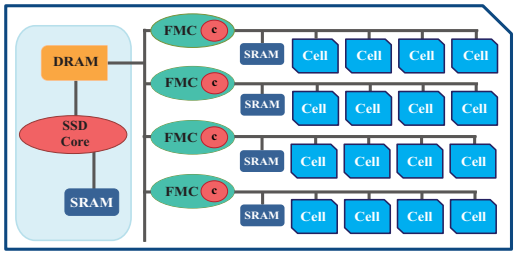


Figure 1: Internal architecture of SSD.

Even though SSD provides a high internal bandwidth thanks to both of multi-way and multi-channel parallelism, its final bandwidth is currently limited by the bandwidth of host interface [3]. In the case of traditional in-host processing (IHP) where the data stored in SSD is processed by the host CPU, the performance bottleneck exists on the host interface having relatively low bandwidth in spite of the internal bandwidth of SSD being increased[6].

To overcome such performance bottleneck, in-storage processing (ISP), an approach of processing data with the processors equipped within storage devices, has been proposed [7]. Also, the notion of intelligent SSD (iSSD) has been proposed [5]. iSSD replaces FMC with general-purpose cores (called FMC cores) and employs more SSD cores in existing SSD. As a result, we can directly process data by using the FMC cores and SSD cores *within* iSSD [6].

ISP with iSSD has the following advantages over IHP: (1) the time of transferring the data to cores from flash memory cells is reduced since the data is processed *within* iSSD by FMC cores or SSD cores instead of the host CPU cores; (2) the degree of paral-

*Corresponding author

SAC'15 April 13-17, 2015, Salamanca, Spain.

http://dx.doi.org/10.1145/2695664.2695943

lelism increases because the number of FMC cores could be much larger than that of host CPU cores; (3) the host CPU, which is quite expensive, is in charge of much less processing workloads because it receives a much smaller result processed by iSSD; (4) the host interface is not a performance bottleneck any longer for data processing [6].

Existing researches on iSSD have focused on processing data by *using only FMC cores and SSD cores* inside iSSD *without using the host CPU* [6]. However, the performance of those cores in iSSD is relatively lower than that of the host CPU. This causes ISP on iSSD to have problems in processing the algorithms of high complexity or troubles in parallel processing. In this paper, we are going to get over the problems by making iSSD and host CPU *collaborate*, thereby maximizing the performance of data-intensive algorithms.

The collaborative environment is *heterogeneous* because it is composed of FMC/SSD cores in iSSD and cores in host CPU that provide different performance and also have their own memory independently. In this environment, *scheduling* needs to decide *which core* is going to run for *which function* at *which time* aiming at processing algorithms in an optimal way. Our collaborative environment is assumed to have a large number of cores, which requires a large amount of time for scheduling [8]. In this environment, static scheduling is more effective than dynamic scheduling because it makes the same schedule used multiple times once it has been generated.

Best imaginary level (BIL) scheduling [9], one of static scheduling algorithms for heterogeneous situations, is used for our collaborative environment. BIL scheduling produces a schedule by using the acyclic-precedence expanded graph (APEG), which models the functions in a given algorithm, as an input [10]. APEG is produced by using synchronous data flow (SDF), one of graph modeling techniques [11]. In this graph, nodes and edges represent functions and calling relationships between functions in an algorithm, respectively.

In this paper, we identify the issues related to graph modeling of an algorithm. First, the *function granularity* of a node needs to be determined [10]. It indicates the size of a function that corresponds to a node in a graph. Second, the *data granularity* needs be considered [10] in graph modeling. It indicates the amount of data to be *processed at a time* in a node. The function and data granularity of a node in an SDF graph affects the number of nodes in its corresponding APEG, thereby considerably influencing not only the scheduling time but also the execution time.

In addition to the issues of granularity in graph modeling, we have another issue to handle input data of different sizes. The number of function calls changes depending on the size of input data. A schedule made for a given input data (under predefined function/data granularity) cannot be used later for other input data of a different size due to the property of static scheduling.

In this paper, we address all these issues as follows. For determining the function and data granularity, we first analyze how the change of granularity affects the scheduling and execution times of an algorithm. Then, we determine the optimized function and data granularity to minimize the execution time based on the analysis result. In the case of the input data whose size is different from that used in scheduling, we use the same schedule for processing the data by only changing its data granularity, which could lead to a little loss in execution time but a much large gain in scheduling time.

The contributions of this paper are summarized as follows.

- We analyze the problems of ISP when using only iSSD and propose an approach of processing data with the collaboration of iSSD and the host CPU.
- We identify and solve the issues in graph modeling for the input data. Then, we also address how to deal with the future input data of a different size.
- Through extensive experiments, we show that our collaborative processing approach outperforms other existing ones such as IHP and ISP. The experimental result shows that the proposed collaborative processing performs faster up to 2.43 and 1.7 than IHP and ISP, respectively.

The rest of this paper is organized as follows. Section 2 explains background related to scheduling. Section 3 proposes our collaborative processing approach and addresses issues and our solutions in applying collaborative processing to data mining tasks. Section 4 shows the effectiveness of our collaborative processing through experiments in comparison with other existing approaches of IHP and ISP. Section 5 summarizes and concludes the paper.

2. BACKGROUND

2.1 Scheduling

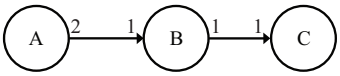
Scheduling is a process of determining a schedule, which specifies *which processor* executes *which function* at *which time*. A scheduling algorithm needs the following inputs: (1) a graph of an algorithm and (2) a time table including the execution time of each function and the inter-processor communication (IPC) time [12][13].

Scheduling can be classified into (1) homogeneous and heterogeneous ones according to the types of the target processors [14] and (2) static and dynamic ones according to the time point of scheduling [9]. *Homogeneous scheduling* is for environment where multiple processors have identical performance and are located closely to each other hardly occurring IPC time; *Heterogeneous scheduling* is for environment where processors are of different types and are located relatively far from each other, involving a considerable amount of IPC time. *Dynamic scheduling* runs whenever each node in a graph needs to be assigned for its execution; *Static scheduling* analyzes the graph *offline* and produces a schedule. Once the schedule is decided for an algorithm, it can be used a number of times for the execution of the algorithm.

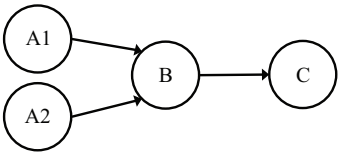
In this paper, we use the *best imaginary level (BIL)* scheduling, which is a well-known heterogeneous static scheduling [9]. BIL scheduling selects a node whose execution time is longest in a graph and also selects a processor that can process the node most efficiently. BIL scheduling assigns a priority value to each node; as a node has longer execution time, it has a higher priority value. In other words, BIL scheduling reduces the overall time by processing the node that needs more time at first.

2.2 Graphs for Scheduling

Functions and the calls among them in an algorithm are represented as nodes and edges, respectively. In this paper, we use a widely used graph modeling technique, *synchronous dataflow (SDF)* [11]. In SDF, both ends of an edge have numbers that represent the number of samples or tokens (sample rate) that its connected two nodes exchange. Figure 2 shows an example of an SDF graph. In Figure 2(a), Node A generates a token during its execution. Node B needs two tokens (obtained from Node A) for its execution. Figure 2(b) shows an *acyclic precedence expanded graph (APEG)*, which contains all the nodes generated according to sample rate in SDF graph. In BIL scheduling, APEG is used as an input [15].



(a) Synchronous data flow graph



(b) Acyclic precedence expanded graph

Figure 2: Graph representation.

3. SCHEDULING FOR COLLABORATIVE PROCESSING

In this section, we discuss how to schedule a data-intensive task in collaborative environment for more efficient processing.

3.1 Graph Modeling

For graph modeling for an algorithm, we employ a widely used SDF [11]. An SDF graph is then converted to APEG for a given data in order to be used as an input for scheduling. The time complexity of BIL scheduling is O($n^2 plogp$) where n is the number of nodes in APEG and p is the number of cores to be used [9]. If APEG has a small number of nodes, the scheduling time is small. In this case, however, the execution time could be large because cores in collaborative environment are unlikely to be fully utilized due to too large granules of nodes in APEG. Therefore, deciding the number of nodes in APEG seriously affects the time both for scheduling and execution. There are two factors that affect scheduling and execution times: the function granularity and data granularity [10].

First, the *function granularity* indicates the size of a function corresponding to a node in SDF. By using the notion of granularity, a function in an original algorithm may correspond to a single node while it may correspond to multiple nodes after it is divided into multiple smaller functions. To find reasonable function granularity, we first generate multiple SDFs having different function granularity for a given algorithm, and then select one of them that has the function granularity resulting in small execution time.

Second, the *data granularity* indicates the amount of data to be processed at a time in a node. When deciding data granularity, we consider the size of memory space owned by cores. An FMC core normally has the memory space capable of storing 1 or 2 pages of 16KB or 32KB. If the data granularity exceeds the size of memory space for an FMC core, the FMC core will be excluded from scheduling for processing the node. This may incur a situation of not using a large number of FMC cores. To solve this problem, we allow the memory space of SSD cores to be also shared with FMC cores. Because the memory of SSD cores is physically located more closely to FMC cores than main memory in host, a smaller IPC time is required. The memory size of SSD cores is known to be 0.1% of the size of flash memory in SSD[6]. Furthermore, iSSD is assumed to have more memory than SSD [6]. Therefore, when necessary, it is possible for FMC cores to use the memory space of SSD cores in loading/storing their data. This strategy enables us to determine the data granularity more flexibly.

3.2 Schedule Recycling

In static scheduling, the function and data granularity is determined according to the size of input data. The function and data granularity needs to change if an input data of a different size arrives. Since different granularity leads to a different schedule, a new schedule is normally required for the new input data. However, because scheduling consumes a great amount of time, it is undesirable to generate a new schedule for every input data.

In order to solve this problem, we propose a strategy of *recycling a pre-generated schedule* by slightly modifying the data granularity of nodes in APEG. We first generate a schedule for the first input data (hereafter, we refer to this as a *base schedule*). When a new input data of a different size arrives, we adjust the data granularity of a base schedule to process the new data. Also, the execution time of nodes and IPC time are re-computed according to the adjusted granularity. For example, if the size of input data is twice of the original one, then the data granularity in APEG of the base schedule is doubled. The execution time and the IPC time in the time table become doubled as well.

Of course, the modified version from the base schedule is unlikely to be optimal for the new input data. However, we note that new scheduling requires a lot of time, which is much more than or (at least) comparable to that of execution time, in processing a large volume of data. It is undesirable to require a large amount of time for re-scheduling to get a small gain amount of execution time. In this paper, we therefore decided to recycle the base schedule only with some modifications, instead of re-computing the optimal schedule for the new input data.

4. PERFORMANCE EVALUATION

In this section, we evaluate the performance of our collaborative approach in comparison with previous ones via extensive experiments.

4.1 Algorithms

In this section, we explain data-intensive algorithms such as k-means [16], PageRank [17], and SimRank [18] used for our evaluation and present how to build their SDF graphs.

k-means is one of the most-widely used clustering algorithms in data mining applications. It performs as follows. First, it reads every object from a storage device (Read) and computes the distances between the object and centroids of current k clusters (CalcDist). Based on the distances, it assigns the object into the cluster nearest to the object (SelClust). After assigning all the objects, it recomputes new centroids of the k clusters newly made (NewClust). It repeats this process by the pre-defined number of times. When the process is completed, it stores the result into storage device (Write).

Figure 3 is an SDF graph of k-means that follows above-mentioned steps, where p indicates data granularity of each node, k does the number of central clusters, m does the number of iterations for k-means, and $1D$ does delayed buffer which holds the initial centroids of k clusters for $CalcDist$ in the first iteration.

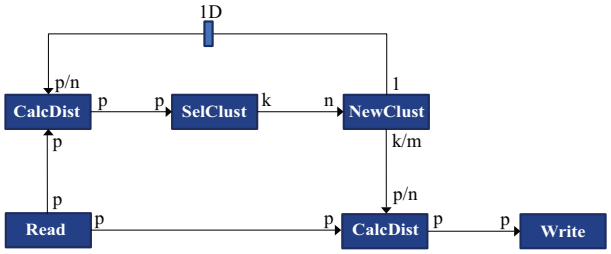


Figure 3: SDF graph for k-means.

PageRank is a well-known ranking algorithm computed by Equation 1, where r_i is a ranking vector, W is an adjacency matrix, and a is a damping factor.

$$r_i = (1-a)W * r_{i-1} + ar_0 \quad (1)$$

PageRank performs as follows. An adjacency matrix is read from a storage device (Read). Then, an element value is obtained by multiplying a row of matrix W by Vector r_{i-1} (Multiply). A vector is made by combining all the elements computed by $Multiply$ (MakeVec). Vector r_i is generated by adding this vector and ar_0 in $MakeVec$ (AddDamp). This process is repeated by the predefined number of times. When the process is completed, vector r_i is stored to a storage device (Write). Figure 4 shows an SDF graph of PageRank having m iterations, where n represents the number of elements in a ranking vector, $1D$ does delayed buffer that holds the initial vector r_0 for $Multiply$.

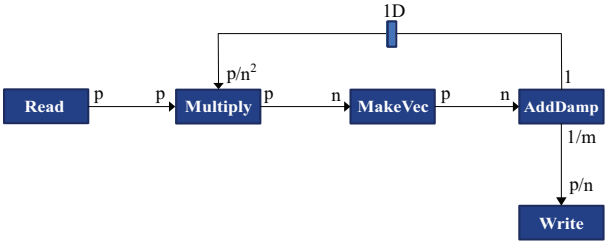


Figure 4: SDF graph for PageRank.

SimRank is an algorithm to compute link-based similarities by employing matrix multiplications as in Equation 2 where S represents a similarity matrix, W does an adjacency matrix, and c does a damping factor.

$$S_k = cW^t S_{k-1} W + (1-c)S_0 \quad (2)$$

SimRank performs as follows. An adjacency matrix is read from a storage device (Read). Matrix W^t is multiplied with S_{k-1}, and then, matrix W is multiplied with a resulting matrix of multiplying W^t and S_{k-1} (MakeMat). S_k is obtained by multiplying the result of $MakeMat$ and c and adding $(1-c)S_0$ to it (AddDamp). This process is repeated by the pre-defined number of times. Matrix S_k is stored into a storage device (Write).

Figure 5 shows an SDF graph of SimRank with m iterations, where n is the number of elements in matrix S_{k-1} and $1D$ is delayed buffer that holds the initial matrix S_0.

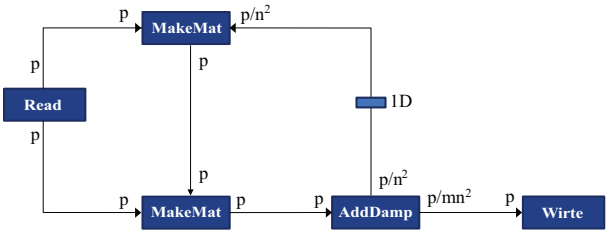


Figure 5: SDF graph for SimRank.

4.2 Experimental Setup

In our experiments, SDF graphs stated in Section 4.1 are used. The function execution time and IPC time between cores are measured by using Vtune[19].

Since iSSD is a currently non-existing storage device, we modified a cost-model[6] for different processing approaches. The hardware-related parameters of the cost-model are set as shown in Table 1. We compare three processing approaches: (1) IHP, (2) ISP, (3) our collaborative approach (CP). We employ BIL scheduling only for ISP and CP because IHP having only CPU cores does not require heterogeneous scheduling.

Data used by each algorithm are all synthetic ones of 1GB. The data are assumed to be evenly distributed over all the cells in iSSD.

Table 1: Hardware-related parameter settings

Parameters	value
# of host CPU cores (2.5GHz)	4
# of SSD cores	2
# of FMC cores	32
SSD core rates (MHz)	800
FMC core rates (MHz)	400

4.3 Performance with Varying Granularity

In this section, we investigate the performance change with different function and data granularity.

4.3.1 *Function Granularity*

We first determine the function granularity to have good execution time. For deciding the function granularity, however, data granularity needs to be fixed. Since a *page* is the basic unit of IO in SSD [6], the data granularity is initially set as a page.

Two types of SDF graphs having different function granularity are generated for each algorithm. One is the graphs with a fine-grained level mentioned in Section 4.1, and the other is the graphs with a coarse-grained level generated in the following.

In k-means, the graph with a coarse-grained level is made by combining $CalcDist$ and $SelClust$ shown in Figure 3 together into one node. The reasoning behind this decision is as follows.

Read and Write, functions related to data IO, are executed only on FMC cores. If $CaclDist$, which requires a lot of computations, is combined with Read or Write nodes, it should be processed in FMC cores. In this case, using FMC cores of relatively low performance could be inefficient when executing functions of high complexity is performed. Also, the number of executions of $CaclNewClust$ is much smaller than those of $SelClust$ or $CalcDist$. Thus, if $NewClust$ and other nodes are combined, these combined nodes should be assigned to the same cores along with $CalcDist$. This could cause the performance to decrease because the number of available cores decreases.

For PageRank, the graph with a coarse-grained level is created by combining $MakeVec$ and $AddDamp$ in Figure 4 into one node. In APEG, the numbers of $MakeVec$ and $AddDamp$ generated are the same. Therefore, combining these two nodes does not adversely affect the utilization of cores.

For SimRank, the graph with a coarse-grained level has $MakeMat$ and $AddDamp$ in Figure 5 merged in one node. If two $MakeMat$s in Figure 5 are combined together, the complexity of the nodes becomes too high. This would deteriorate the utilization of iSSD since FMC cores provide too a low performance to deal with functions of high complexity.

Table 2 shows the performance results with different function granularity. For k-means and PageRank, the execution time with the graphs of a coarse-grained level decreases by 1.54% and 3%, respectively. This is due to the absence of IPC time. The two nodes are merged into one in the graph with a coarse-grained level and

then assigned to the same core. Thus, IPC time is eliminated in this case, leading to slightly shorter execution time. For SimRank, the execution time with the graph of a fine-grained level decreases by 3.70%. Since a large number of $MakeMat$ and $AddDamp$ in APEG are those functions of high complexity, the effect of parallelism in processing the nodes becomes high.

In the following experiments, the graph with a coarse-grained level is used for k-means and PageRank, and the graph with a fine-grained level is used for SimRank.

Table 2: Execution times of different function granularity (sec)

Algorithm \ Graph	Fine-grained level	Coarse-grained level
k-means	9.104	8.966
PageRank	5.503	5.343
SimRank	83.302	91.524

4.3.2 Data Granularity

In this series of experiments, we change data granularity to find the schedules having reasonable execution and scheduling times. The function granularity identified in Section 4.3.1 is used.

We perform experiments as follows. We set data granularity as one page and generate a base schedule for initial input data. Whenever a new input data is given, its schedule is obtained by adjusting the base schedule with different data granularity. For instance, if the schedule for input data of 4MB needs to be applied for new input data of 1GB, we increase data granularity of each node in the schedule by 256 times and also enlarge the execution time and IPC time by 256 times as well.

Figure 6 shows the scheduling and execution times obtained when applying the base schedule created for the input data smaller than 1GB to the new input data of 1GB. The x-axis represents the size of input data to be scheduled (in log scale). The y-axes in Figure 6(a) and Figure 6(b) indicate the scheduling time (in log scale) and execution time (in linear scale), respectively. For example, the first point in each graph indicates the time when applying the base schedule created for input data of 4MB to the new input data of 1GB; the last point does the time when applying the base schedule created for input data of 1GB to the new input data of 1GB, which could be regarded as an *optimal one*.

As the input data gets larger, the number of nodes in APEG becomes larger, thereby increasing the scheduling time. On the contrary, as the input data used for the base schedule gets larger, the execution time for the new input data decreases because the gap between the two input data becomes smaller.

For k-means, the execution time for the new input data of 1GB by using the base schedule made by the input data of 64MB is shown close to that using the base schedule made by the input data of 1GB; The time for making a base schedule with the input data of 64MB is much smaller than that for the input data of 1GB. Based on this observation, we decided to use the base schedule made by the input data of 64MB for further experiments. For PageRank and SimRank, we decided to use the base schedule construced by the input data of 128MB in the same reason.

4.4 Performance Comparisons

In this set of experiments, we show the superiority of our collaborative approach compared with existing IHP and ISP. The hardware-related parameters are set as shown in Table 1.

Table 3 shows the execution times of all the approaches. In all the results, CP shows the best performance. In the case of k-means, it outperforms ISP and IHP by 1.34 times and 1.92 times, respectively. In the case of PageRank, CP provides better performance than ISP and IHP by 1.40 times and 1.60 times, respectively. In the case of SimRank, it performs better than ISP and IHP by 1.70 times and 2.43 times, respectively.

IHP processes all the functions only on the host CPU. It is observed that it shows poor performance for processing the functions that have low complexity but require a large number of cores. On the contrary, ISP performs poorly for the functions that are difficult to be parallelized and also have high complexity. CP shows the best performance among all the approaches since it processes the functions easy to be parallelized on cores in iSSD and the functions of high complexity on cores in the host CPU.

Table 3: Performance comparisons (sec)

Algorithm \ Approaches	CP	ISP	IHP
k-means	8.948	12.003	17.238
PageRank	5.399	7.543	8.645
SimRank	83.969	142.572	204.151

5. CONCLUSIONS

iSSD is a new type of a storage device that employs FMC cores and SSD cores in existing SSD to realize ISP. Existing researches on iSSD mainly focused on how to utilize these two types of cores inside iSSD effectively *without considering the collaboration with the host CPU*. In this paper, we have addressed an effective collaboration of iSSD and host CPU in order to maximize the performance of data-intensive algorithms.

The contributions of our paper are summarized as follows.

First, the problems of existing IHP and ISP have been analyzed. IHP is not suitable to process the functions that can be processed in parallel; ISP has a limitation to process the functions of high complexity. Therefore, we have proposed a collaborative approach called CP that fully exploits the computing resources in CPU and iSSD.

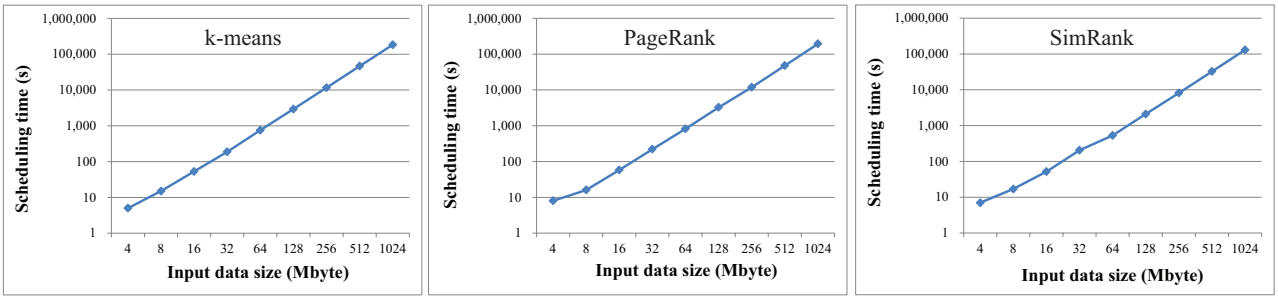
Second, the issues of function and data granularity have been introduced and their solutions have also been addressed. Furthermore, a method of recycling the base schedule to execute new input data has been proposed.

Third, through extensive experiments, we have shown that CP outperforms IHP and ISP significantly. The result shows that CP outperforms IHP and ISP up to 2.43 and 1.7 times, respectively.

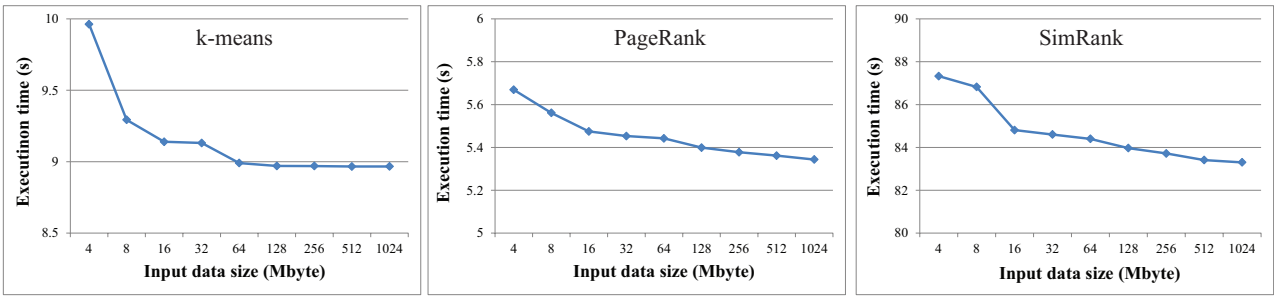
As future work, we have a plan to improve the current CP by employing additional computing resources including graphic processor units (GPU). GPU has a huge number of cheap cores and thus is regarded more suitable for parallel processing. We are going to incorporate GPU into our CP framework to achieve higher performance in data-intensive applications such as big data analysis.

6. ACKNOWLEDGMENTS

This work was supported by (1) Semiconductor Industry Collaborative Project between Hanyang University and Samsung Electronics Co. Ltd., (2) the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (NIPA-2014-H0301-14-1022) supervised by the NIPA (National IT Industry Promotion Agency), and (3) Business (Grants No. C0191469) for Cooperative R&D



(a) Scheduling time



(b) Execution time

Figure 6: Results with varying data granularity.

between Industry, Academy, and Research Institute funded Korea Small and Medium Business Administration in 2014.

7. REFERENCES

[1] S. Lee, B. Moon, and C. Park, "Advances in Flash Memory SSD Technology for Enterprise Database Applications," In *Proc. of ACM Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 863-870, 2009.

[2] S. Kim et al., "Fast, Energy Efficient Scan inside Flash Memory SSDs," In *Proc. Int'l Workshop on Accelerating Data Management Systems using Modern Processor and Storage Architectures*, ADMS, 2011.

[3] Y. Hu et al., "Exploring and Exploiting the Multi-level Parallelism Inside SSDs for Improved Performance and Endurance," *IEEE Trans. on Computers*, Vol. 62, No. 6, pp. 1141-1155, 2013.

[4] S. Lee et al., "A Case for Flash Memory SSD in Enterprise Database Applications," In *Proc. ACM Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 1075-1086, 2008.

[5] J. Do et al., "Query Processing on Smart SSDs: Opportunities and Challenges," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp.1221-1230, 2013.

[6] D. Bae et al., "Intelligent SSD: A Turbo for Big Data Mining," In *Proc. of ACM Int'l Conf. on Information and Knowledge Management*, ACM CIKM, pp. 1553-1556, 2013.

[7] E. Riedel, G. Gibson, and C. Faloutsos, "Active Storage for Large-Scale Data Mining and Multimedis," In *Proc. of Conf. on Very Large Databases*, VLDB, pp. 62-73, 1998.

[8] H. Topcuoglu, S. Hariri and M. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 3, No. 3, pp. 260-274. 2002.

[9] H. Oh and S. Ha, "A Static Scheduling Heuristic for Heterogeneous Processors," In *Proc. Int'l Conf. Euro-Par Parallel Processing*, pp. 573-577, 1996.

[10] S. Ha and H. Oh, "Decidable Signal Processing Dataflow Graphs: Synchronous and Cyclo-Static Dataflow Graphs," *Handbook of Signal Processing Systems*, pp.851-874, 2010.

[11] E. Lee and D. Messerschmitt, "Synchronous data flow," *Proc. of the IEEE*, Vol. 75, No. 9, pp. 1235-1245, 1987.

[12] S. Ranaweera, and D. Agrawal, "A Task Duplication Based Scheduling Algorithm for Heterogeneous Systems," In *Proc. IEEE Int'l Parallel and Distributed Processing Symp.*, IPDPS, pp. 445-450, 2000.

[13] Y. Kwok and I. Ahmad, "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors," *ACM Computing Surveys*, Vol. 31, No. 4, pp. 406-471, 1999.

[14] G. Sih and E. Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 4, No. 2, pp. 175-187, 1993.

[15] J. Pino et al., "Software Synthesis for DSP Using Ptolemy," *Journal of VLSI Signal Processing*, Vol. 9, No. 1, pp. 7-21, 1995.

[16] J. MacQueen et al., "Some Methods for Classification and Analysis of Multivariate Observations," In *Proc. of Berkeley Symp. on Mathematical Statistics and Probability*, Vol. 1, No. 14, pp. 281-297, 1967.

[17] L. Page et al., The PageRank Citation Ranking: Bringing Order to the Web, Technical Report, Stanford University, 1999.

[18] G. He et al., "Parallel SimRank Computation on Large Graphs with Iterative Aggregation," In *Proc. ACM Int'l Conf. on Knowledge Discovery and Data Mining*, ACM SIGKDD, pp. 543-552, 2010.

[19] Intel, Intel VTune Amplifier, https://software.intel.com/en-us/node/529213, 2014.