Collaborative Processing of Data-Intensive Algorithms with CPU, intelligent SSD, and GPU

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

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

ABSTRACT

The graphic processing unit (GPU) is a computing resource to process graphics-related applications. The intelligent SSD (iSSD) is a solid state device (SSD) that is provided with data processing power. These days, CPU, GPU, and SSD are equipped together in most processing environment. If SSD is replaced with iSSD later on, we have a new processing environment where three computing resources collaborate one another to process a huge volume of data (so called big data) quite effectively. In this paper, we address how to exploit all these computing resources for efficient processing of data-intensive algorithms.Through extensive experiment, we verify the effectiveness and potential of the proposed collaborative processing environment by processing data concurrently with multiple computing resources. The results reveal that processing in the our environment outperforms that in the traditional one by up to 3.5 times.

CCS Concepts

•**Computing methodologies** → *Simulation evaluation;*

Keywords

SSD, GPU, Collaborative processing, Heterogeneous, Scheduling

1. INTRODUCTION

Owing to the advances of social network & media services and Internet & mobile businesses, the amount of data has been increasing explosively. The era of big data has begun. There have been a number of research efforts to deal with the big data efficiently by exploiting other computing resources in addition to CPU in host. Typical examples are GPU and iSSD [1][2].

The graphic processing unit (GPU) is a computing resource to process graphics-related applications [3]. There are some methods proposed recently to exploit GPU for processing general applications rather than graphics-related ones. GPU consists of a larger number of cores organized in multiple sets, each of which processes the same group of instructions in parallel with cores in it [4]. The intelligent SSD (iSSD) is a solid state device (SSD) that is provided with data processing power [5]. iSSD is equipped with a core in each channel, thereby making it possible to process data inside iSSD without transferring data to host [6].

Computing resources have their own characteristics. Some existing methods process applications by using a single computing resource [1][2] while some other methods exploit collaborations of two different resources (e.g., CPU & GPU or CPU & iSSD) for more efficient processing [6][7].

These days, CPU, GPU, and SSD are equipped together in most processing environment. If SSD is replaced with iSSD later on, we have a new processing environment where three computing resources collaborate one another to process a huge volume of data (so called big data) quite effectively. In this paper, we address how to exploit all these computing resources for efficient processing of data-intensive algorithms.

The collaborative processing environment is a heterogeneous one because it has different types of computing resources that provide different performance and also have their own memory independently, thereby requiring inter-processor communication (IPC) time [8]. In this environment, scheduling is required to decide which core is going to run for which function at which time, aiming at processing algorithms in an optimal way [6]. We need heterogeneous scheduling [8][9] for our collaborative processing environment because it has computing resources of different types.

A schedule can be obtained from scheduling that requires a target algorithm normally modeled as a graph [10][11]. This modeling needs to consider types of computing resources, memory sizes in computing resources, and IPC times between computing resources. In this paper, we are going to focus on how to model a given algorithm as a graph for effective collaborations of computing re-

*Corresponding author

SAC 2016,April 04-08, 2016, Pisa, Italy
©2016 ACM. ISBN 978-1-4503-3739-7/16/04…$15.00
DOI: http://dx.doi.org/10.1145/2851613.2851741

sources.

Our graph modeling for an algorithm in our processing environment is performed as follows: First, we classify functions into two categories, parallelizable functions and merging functions. GPU and iSSD have architecture appropriate for parallel processing of data, and thus are beneficial when assigned to parallelizable functions. On the other hand, CPU is beneficial when assigned to such functions that require merging or have high time complexity. Second, we determine the amount of data to be processed at a time by a function. For example, GPU processes a large amount of data simultaneously by using a large number of cores in parallel. It is more effective to set the amount of data to be matched with the number of cores in GPU. This paper proposes an effective way of data-intensive algorithms by using the three kinds of computing resources based on the strategies above.

The main contributions of the paper are summarized as follows.

- First, assuming a collaborative processing environment feasible in the (very) near future, we propose a new paradigm of data-intensive algorithms based on the collaborations of heterogeneous computing resources.
- Second, we identify important issues occurring in data processing in a collaborative processing environment, and propose solutions to them.
- Third, we verify the effectiveness and potential of the proposed collaborative processing environment through extensive experiments. The results reveal that processing in the our environment outperforms that in the traditional one by up to 3.5 times.

The organization of the paper is as follows. Section 2 introduces characteristics of three computing resources and previous work related to data processing based on different computing resources. Section 3 presents the proposed collaborative processing environment, and proposes our strategies for effective collaborative processing. Section 4 shows the effectiveness of our collaborative processing via a variety of experiments. Finally, Section 5 summarizes the contributions of the paper.

2. RELATED WORK

Contemporary computing systems equip GPU and SSD together with CPU these days. This section briefly introduces these computing resources and their related work for efficient data processing.

2.1 Graphic processing unit (GPU)

Figure 1 shows the architecture of GPU. GPU consists of a set of streaming multiprocessors (SM), each of which is again composed of a set of streaming processors (SP). There are memories with different sizes and latency on GPU such as device memory, shared memory, texture memory, constant memory, and register [12].

GPU follows the single instruction multiple threads (SIMT) model on which SPs in an SM execute the same instruction at one time [4]. GPU is suitable for processing algorithms that execute simple operations (addition, multiplication) iteratively [12].

2.2 Intelligent SSD (iSSD)

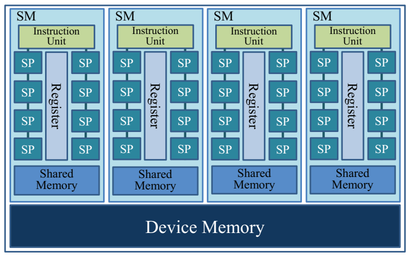

Figure 1: Architecture of GPU.

Figure 2 shows the internal structure of iSSD [1]. iSSD is SSD that has a flash memory controller (FMC) equipped with general-purpose cores (called FMC cores) in each channel and employs more powerful SSD cores in existing SSD. It also has SRAM and DRAM to store the data to be read/written from/to the cells. Data are transferred between DRAM in iSSD and the main memory in host through a host interface, which is controlled by the host interface controller [1].

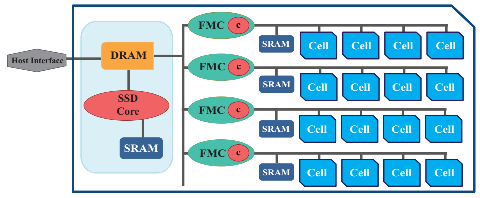

Figure 2: Architecture of iSSD.

The in-storage processing (ISP) with iSSD has the following advantages over in-host processing (IHP) [1]: (1) the degree of parallelism increases because the number of FMC cores could be much larger than that of CPU cores; (2) 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 CPU cores; (3) CPU, which is quite expensive, is in charge of much less processing workloads because it receives a much smaller result processed by iSSD.

3. DATA PROCESSING IN COLLABORATIVE ENVIRONMENT

In this section, we discuss how to process data-intensive algorithms in collaborative processing environment having CPU, GPU, and iSSD to achieve high efficiency.

3.1 Scheduling

In order to get high performance in our collaborative processing environment, scheduling is required to decide which core is going to run for which function at which time aiming at processing data-intensive algorithms. Our collaborative processing environment is a heterogeneous one where processors of different types have their own memory and are located relatively far from each other, involving a considerable amount of inter-processor communication (IPC) time. In this paper, we use the best imaginary level (BIL) scheduling [8], which is a well-known heterogeneous static scheduling algorithm to obtain a schedule appropriate for heterogeneous environment.

BIL scheduling selects a node whose execution time is longest in a APEG graph and also selects a processor that can process the node most efficiently. BIL scheduling assigns a priority value to each node; if a node has longer execution time, it has a higher priority value. In other words, it reduces the overall time by processing the node that needs more time at first [8]. BIL scheduling needs the following inputs: (1) a graph of an algorithm and (2) a time table including the execution time of each function and the IPC time [8]. Nodes and edges in a graph represent functions and calling relationships between functions in an algorithm, respectively.

3.2 Graph Modeling

In this paper, we use a widely used graph modeling technique, synchronous dataflow (SDF) [13]. It simply represents a graph with relationships between functions in an algorithm. In an SDF graph, functions and the calls among them are represented as nodes and edges, respectively. Both ends of an edge have numbers that represent the number of IO tokens (called sample rate) that the two nodes connected to the edge exchange. Figure 3 shows an example of an SDF graph. There are three nodes. Node A only generates a IO token during its execution. Node B needs two IO tokens (obtained from Node A) for its execution and generates a IO token during its execution. Finally, Node C only needs a IO token for its execution.

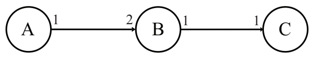

Figure 3: Synchronous dataflow (SDF) graph.

BIL scheduling produces a schedule by using the acyclic precedence expanded graph (APEG), which contains all the nodes generated according to the sample rate in the SDF graph. Figure 4 shows an example of an APEG generated from the graph in Figure 3. There are four nodes because Node A and Node B are executed twice and once, respectively, for the execution of Node C.

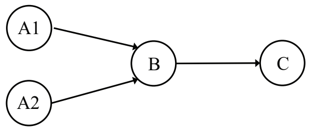

Figure 4: Acyclic precedence expanded graph.

3.3 Graph Modelling Strategy

We have the following challenges in generating an SDF graph that is appropriate for the proposed collaborative processing environment.

First, we need to decide the function granularity in a graph, which is a granularity of a node corresponding to a function. In our collaborative processing environment, GPU and iSSD have a large number of cores in spite of low processing power, thus providing a high degree of parallelism. On the other hand, CPU has high processing power, thus beneficial to merging multiple data, rather than parallel processing. We thus classify nodes into two categories as follows: (1) Nodes for merging data from different other nodes (called merging nodes) and (2) Nodes that are able to process data in parallel (called parallelizable nodes). We can build a variety of SDF graphs with the nodes such classified.

Second, we need to decide the data granularity in a graph, which is the number of IO tokens. GPU in our collaborative processing environment is quite effective in processing a large amount of data simultaneously since it employs SIMT architecture, and thus enables to process efficiently a node having a high data granularity.

It is not allowed to assign a specific node directly to GPU because it is the job of the scheduler. In order to induce parallelizable nodes assigned to GPU by scheduling, we make such nodes have a large number of IO tokens (i.e., high data granularity). If a node needs to be assigned to GPU, its number of IO tokens should be determined by Equation 1. With this number of IO tokens, all the cores in GPU could be fully utilized. Here, the number of IO tokens, number of cores in GPU, memory size in GPU, and average size of IO tokens are denoted as n_{IO_tokens}, n_{cores}, S_{mem}, and S_{IO_tokens}, respectively. In other words, the summation of numbers of all IO tokens should be smaller than or equal to the memory size in GPU, and also the number of IO tokens should be multiples, m, of the number of cores in GPU.

$$\begin{aligned} n_{IO\_tokens} &= m \times n_{cores} \\ n_{IO\_tokens} &\leqslant S_{mem}/S_{IO\_tokens} \end{aligned} \tag{1}$$

In order to determine the best data and function granularities for finding the most efficient schedule for a given algorithm, we perform the following process. We first model each function by using multiple nodes as fine-grained as possible, and classify each node into one of the two categories above. We then select a node to be assigned to GPU, and then determine its data granularity according to Equation 1. Now, we have an SDF graph in the most fine-grained level. We also build a coarse-grained SDF graph in the similar way, after merging those nodes belonging to the same function. Then, we create different schedules for SDF graphs thus obtained. We run the algorithm multiple times with those schedules, and select the best one as our final schedule of the algorithm for the future use.

Because BIL scheduling is a 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. In order to solve this problem, we use the strategy proposed by [6] which recycles a pre-generated schedule by slightly modifying the data granularity.

4. PERFORMANCE EVALUATION

In this section, we evaluate the performance of our processing environment in comparison with different ones via extensive experiments.

4.1 Experimental Settings

In this section, we perform three data-intensive algorithms such as k-means [14], PageRank [15], and SimRank [16]. SDF graphs for these algorithms are generated by [6], and they are adopted for our evaluation. Now, we present 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 5 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.

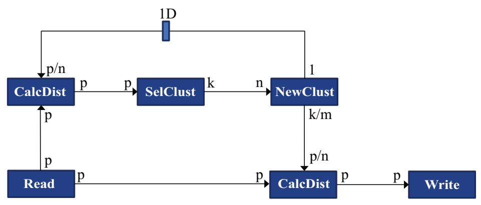

Figure 5: SDF graph for k-means.

PageRank is a well-known ranking algorithm computed by Equation 2, where r_i is a ranking vector, W is an adjacency matrix, and a is a damping factor.

$$r_i = (1-a)W * r_{i-1} + ar_0 \tag{2}$$

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 pre-defined number of times. When the process is completed, vector r_i is stored to a storage device (Write). Figure 6 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 $Multiply$.

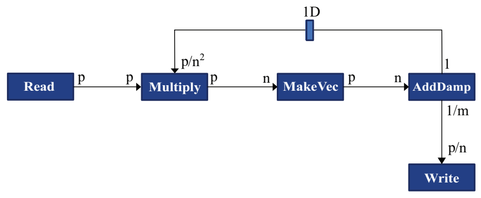

Figure 6: SDF graph for PageRank.

SimRank is an algorithm to compute link-based similarities by employing matrix multiplications as in Equation 3 where S represents a similarity matrix, W does an adjacency matrix, and c does a damping factor.

$$S_k = cW^t S_{k-1} W + (1-c)S_0 \tag{3}$$

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 7 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.

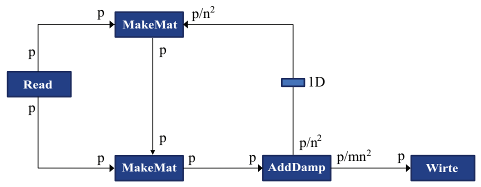

Figure 7: SDF graph for SimRank.

We classify the nodes in SDF graphs of the three algorithms into merging nodes and parallelizable nodes as in Table 1. The nodes for Read and Write are not classified because they are the functions related to data IO and thus need to be definitely executed in FMC cores even though they can be processed in parallel.

Table 1: Node categories

Node / Algorithm	Merging	Parallelizable
k-means	NewClust	CalcDist, SelClust
PageRank	MakeVec, AddDamp	Multiply
SimRank	AddDamp	MakeMat

For our performance evaluation, we use a simulator based on a cost model [6]. We employ Vtune [17] to measure the time for executing a node and the time for IPC, both of which are needed as inputs in scheduling. The parameters related to hardware for simulation are given in Table 2. We synthetically generate 100GB data in a random way for experiments. The dataset for k-means consist of a list of objects, each of which has eight dimensions. A value of each dimension ranges from 1 to 32000. The datasets for PageRank and SimRank consist of a list of edge having source and destination nodes. The number of nodes is about 800,000. A value of each dimension in a object and the index of a node are randomly selected. We assume that the data are uniformly stored over all the cells in iSSD, thereby making all the cells have similar IO workloads.

We have three processing environments for evaluation: (1) In-host processing (IHP), (2) In-storage processing (ISP), and (3) collaborative processing (CP). Also, we have three different configurations of CP: (1) CP using CPU and iSSD denoted as CP(iSSD), (2) CP using CPU and GPU denoted as CP(GPU), and (3) CP using CPU, GPU, and iSSD denoted as CP(GPU+iSSD). In this section, we compare the performances of these five processing environments. We note that BIL scheduling is applicable only to ISP and CP (i.e., no IHP) because it is a heterogeneous scheduling algorithm.

4.2 Determining Granularities

Table 2: Hardware related parameters

	Parameter	Value
Host	# of host CPU cores	4
	Host CPU rate (GHz)	3
	Interface bandwidth (Gbps)	3
iSSD	# of SSD cores	4
	SSD core rate (Mhz)	800
	# of FMC cores	32
	FMC core rate (MHz)	400
	Cell read time (us)	50
	Cell Write time (us)	1,200
GPU	# of GPU cores	3,000
	GPU core rate (MHz)	900

In this section, we perform experiments to find the function and data granularities appropriate for the best schedule.

First, we determine the SDF graphs in a most fine-grained level. For this purpose, we use the graphs introduced in Section 4.1. Then, for the parallelizable nodes, we determine the data granularity according to Equation 1.

Next, we build an SDF graph in a coarse-grained level. In k-means, we merge *CalcDist* and *SelClust* into *SetClust*, a coarse-grained node. In PageRank, we build an SDF graph by combining *MakeVec* and *AddDamp* (i.e., those requiring merging), to *ResultVec*, a coarse-grained node. In SimRank, to build an SDF graph in a coarse-grained level, we merge *MakeMat* (parallelizable node) and *AddDamp* (merging node) into *ResultMat*, a coarse-grained node even though they belong to different categories.

Table 3 shows execution times of each algorithm with the schedules obtained from SDF graphs in different levels. Here, we set m in Equation 1 as one for the data granularity.

In k-means, we try to reduce the IPC time by combining two parallelizable nodes into a single node. However, the SDF graph in a fine-grained level shows better performance than that in a coarse-grained level. We decide the data granularity of *CalcDist* as large as possible in such a way that it is processed in GPU. *SetClust*, which contains *CalcDist* and *SelClust* in a single node, has a large data granularity in order that it is processed in GPU as well. We note, however, *SelClust* (parallelizable node) can be processed in iSSD or CPU without being transferred to GPU. This strategy of employing coarser *SetClust* adversely affects the performance. In PageRank, the performance gets better by combining *MakeVec* and *AddDamp* into *ResultVec*, which reduces the IPC time in its execution. In SimRank, it is more effective to use separate *MakeMat* and *AddDamp* because they belong to different categories. In the following experiments, we thus use the SDF graphs in a fine-grained level for k-means and SimRank, but the SDF graph in a coarse-grained level for PageRank.

Table 3: Execution times with different function granularities (seconds)

Algorithm \ **Graph**	Fine-grained	Coarse-grained
k-means	**402.4**	536.9
PageRank	2791.5	**1902.9**
SimRank	**3791.5**	4239.9

Table 4 shows execution times of the three algorithms according to the schedules obtained with different data granularities. The result shows that the execution time is reduced in the cases when m is set as the multiples of cores in GPU. In other cases, the execution time becomes longer because the cores in GPU are not fully utilized. In the following experiments, we use the data granularity with m set as 1 in Equation 1.

Table 4: Execution times with different data granularities (seconds)

Algorithm \ **m**	0.5	1	1.5	2
k-means	454.7	**402.4**	421.3	402.4
PageRank	2975.8	**1902.9**	2290.6	1944.8
SimRank	7044.3	**3791.5**	5687.5	3802.9

4.3 Performance Comparisons

In this section, we perform experiments to compare the performances of algorithms in five different processing environments. Table 5 shows the execution times and rankings (in parentheses) for each processing environment.

The result indicates that the execution time in a collaborative processing environment is much smaller than that in a non-collaborative processing environment. Also, the performance gap increases as the time complexity and parallelizability of nodes get higher. The reason for this is that the parallelizable nodes can be processed in iSSD or GPU in parallel, thereby reducing the execution time. Thus, CP(GPU) and CP(iSSD) that use CPU and iSSD, respectively, and ISP perform better than IHP. CP(GPU+iSSD) that use GPU and iSSD together show the best performance.

4.4 Resource Utilization

In this section, we verify computing resources in CP(GPU+iSSD) are well collaborated with one another when performing algorithms.

Figure 8 shows the utilization of resources for each function. The x-axis represents nodes in algorithms and the y-axis does the utilization of resources. We exclude *Read* and *Write*, i.e., IO nodes because they are all processed in FMC cores.

The result shows that collaborations among resources are effectively done in processing all the algorithms. In k-means, *CalcDist* (merging node) and *SelClust* (parallelizable node) are primarily processed in CPU. On the other hand, CalcDist (parallelizable node), is assigned mainly in GPU by the BIL scheduler. Thus, iSSD seems to play a role of supplementing other computing resources. In PageRank, *ResultVec* (merging node) is processed in CPU, *Multiply* (parallelizable node) is mostly processed in GPU and iSSD. In SimRank, however, *AddDamp* is processed in all computing resources except for GPU even though it requires merging. We conjecture it exploits all the resources due to the high time complexity of SimRank.

5. CONCLUSIONS

Recently, ISP has attracted a lot of interests in industry as well as in academia. Also, together with SSD, GPU becomes more popular as one of important secondary computing resources and is equipped in general computing environment. This makes it possible in the (very) near future to exploit CPU, SSD, and GPU to process a huge amount of data.

Assuming this advanced environment, this paper has addressed how to maximize the processing performance of data-intensive algo-

Table 5: Execution time for each environment (seconds)

Algorithm \ Environment	CP(GPU+iSSD)	CP(GPU)	CP(iSSD)	ISP	IHP
k-means	**402.46 (1)**	419.92 (2)	650.77 (3)	977.16 (4)	1264.71 (5)
PageRank	**1902.93 (1)**	2328.28 (2)	2482.63 (3)	3807.54 (4)	3796.69 (5)
SimRank	**3791.59 (1)**	5980.02 (3)	5884.29 (2)	10488.23 (4)	13612.28 (5)

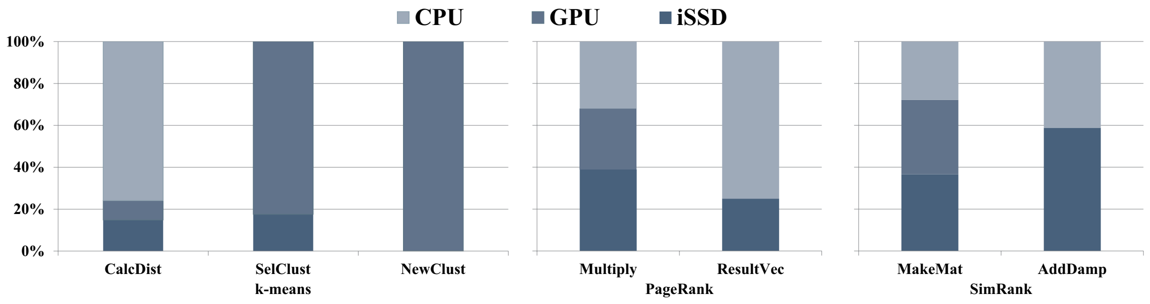

Figure 8: Utilization of resources.

rithms. The contributions of this paper can be summarized as follows.

First, we propose a collaborative processing environment that utilizes both of iSSD and GPU in addition to traditional CPU. CPU is suitable for processing functions of high time complexity while iSSD and GPU are effective for processing functions that can be parallelized and are of low time complexity.

Second, we present how to use such a collaborative processing environment for running data-intensive algorithms effectively. We address the issue of determining data and function granularities that affect the performance in collaborative processing environments. We propose how to set the two granularities to get better performance.

Third, we show the superiority of data processing in collaborative processing environments by a series of experiments. The results show that data processing in a collaborative processing environment outperforms that in a non-collaborative processing environment by up to 3.5 times. Also, the experiments on utilization of computing resources indicate that the collaborations are done quite effectively.

6. ACKNOWLEDGMENTS

This research was supported by (1) a Semiconductor Industry Collaborative Project between Hanyang University and Samsung Electronics Co. Ltd., (2) the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (NRF-2014R1 A2A1A10054151).

7. REFERENCES

[1] 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.

[2] N. Gov et al., "GPUTeraSort: High Performance Graphics Coprocessor Sorting for Large Database Management," *In Proc. ACM Int'l. Conf. on Management of Data,* ACM SIGMOD, pp. 325-336, 2006.

[3] J. Fung and S. Mann, "Using Graphics Devices in Reverse: GPU-based Image Processing and Computer Vision," In *Proc. IEEE Int'l Conf. Multimedia and Expo*, pp. 9-12, 2008.

[4] S. Ryoo et al., "Optimization Principles and Application Performance Evaluation of a Multithreaded GPU using CUDA," In *Proc. ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, PPoPP, pp. 73-82, 2008

[5] 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.

[6] Y. Jo et al., "On Running Data-Intensive Algorithms with Intelligent SSD and Host CPU: A Collaborative Approach," In *Proc. Int'l Conf. on ACM/SIGAPP Symposium On Applied Computing*, ACM SAC, pp. 2060-2065, 2015.

[7] S. Pabst, A. Koch, and W. Straber, "Fast and Scalable CPU/GPU Collision Detection for Rigid and Deformable Surfaces," *Computer Graphics Forum*, Vol. 29, No. 5, pp. 1605-1612, 2010.

[8] 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.

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

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

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

[12] N. Bell and M. Garland, Efficient Sparse Matrix-Vector Multiplication on CUDA, NVIDIA Technical Report, NVIDIA Corporation, 2008.

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

[14] J. MacQueen et al., "Some Methods for Classification and Analysis of Multivariate Observations," In *Proc. of Berkeley Symp. on Mathematical Statistics and Probability*, pp. 281-297, 1967.

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

[16] G. Jeh and J. Widom, "SimRank: a measure of structural-context similarity," In *Proc. of ACM Int'l. Conf. on Knowledge discovery and data mining,* ACM SIGKDD, pp. 538-543, 2002.

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