J Supercomput (2013) 65:886–902
DOI 10.1007/s11227-013-0870-6

An efficient scheduling scheme using estimated execution time for heterogeneous computing systems

Hong Jun Choi · Dong Oh Son · Seung Gu Kang · Jong Myon Kim · Hsien-Hsin Lee · Cheol Hong Kim

Published online: 19 January 2013

Abstract Computing systems should be designed to exploit parallelism in order to improve performance. In general, a GPU (Graphics Processing Unit) can provide more parallelism than a CPU (Central Processing Unit), resulting in the wide usage of heterogeneous computing systems that utilize both the CPU and the GPU together. In the heterogeneous computing systems, the efficiency of the scheduling scheme, which selects the device to execute the application between the CPU and the GPU, is one of the most critical factors in determining the performance. This paper proposes a dynamic scheduling scheme for the selection of the device between the CPU and the GPU to execute the application based on the estimated-execution-time information. The proposed scheduling scheme enables the selection between the CPU and the GPU to minimize the completion time, resulting in a better system performance, even though it requires the training period to collect the execution history. According to our simulations, the proposed estimated-execution-time scheduling can improve

H.J. Choi · D.O. Son · S.G. Kang · C.H. Kim (✉)
School of Electronics and Computer Engineering, Chonnam National University, Gwangju, Korea
e-mail: chkim22@jnu.ac.kr

H.J. Choi
e-mail: chj6083@gmail.com

D.O. Son
e-mail: sdo1127@gmail.com

S.G. Kang
e-mail: freexz84@gmail.com

J.M. Kim
School of Electrical Engineering, University of Ulsan, Ulsan, Korea
e-mail: jongmyon.kim@gmail.com

H.-H. Lee
School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA
e-mail: leehs@gatech.edu

the utilization of the CPU and the GPU compared to existing scheduling schemes, resulting in reduced execution time and enhanced energy efficiency of heterogeneous computing systems.

Keywords Computer system · Scheduling · CPU · GPU · CUDA · Heterogeneous system

1 Introduction

Improving the performance of computing systems by increasing the throughput of the CPU (Central Processing Unit) is restricted by the limits such as transistor scaling and temperature constraints [1]. For this reason, solutions to reduce the workload of the CPU while improving the performance of computing systems have been explored. One solution to improve the performance of computing systems, when the CPU performance is saturated, is utilizing the GPU (Graphics Processing Unit) which is a highly specialized processor designed for graphics processing. In recent computing systems, the GPU reduces the workload of the CPU by processing complicated graphics-related computations instead of the CPU [2]. Moreover, recent GPUs can process general-purpose applications as well as graphics-related applications with the help of integrated development environments such as CUDA, OpenCL, Cg, HLSL, and OpenGL [3–5]. Especially CUDA (Compute Unified Device Architecture) developed by NVIDIA has been widely used, since it introduced a new data-parallel programming API (Application Programming Interface) based on the C language known to the users [6]. By using CUDA, programmers can easily utilize the GPU resources for reducing the workload of the CPU while improving the performance of computing systems [7, 8]. Therefore, heterogeneous computing systems using both the CPU and the GPU together can be a solution for improving the system performance while reducing the workload of the CPU [9].

The efficiency of the scheduling scheme, which selects the device between the CPU and the GPU to execute the application, is one of the most critical factors that determine the performance of heterogeneous computing systems. For this reason, we should consider the efficient scheduling methods for heterogeneous computing systems. If the programmers determine the device between the CPU and the GPU to execute an application when they implement the code, the status of the computing system cannot be considered dynamically, as the decision is made statically at compile time, resulting in the non-optimal system performance. On the other hand, a dynamic scheduling scheme, which selects the device to execute the application between the CPU and the GPU at runtime, can consider the status of the computing system dynamically, resulting in the improved performance. Therefore, many researchers have focused on the dynamic scheduling schemes for heterogeneous computing systems. The dynamic scheduling methods typically use the information such as device utilization, input data size and performance prediction to select the device between the CPU and the GPU dynamically, and they can be implemented through the OS (Operating System) [10–12].

In this work, we propose a new dynamic scheduling scheme to improve the performance of the heterogeneous computing system by using both execution history

for the incoming application and the remaining execution time for the currently executed application. Contrary to the existing scheduling schemes, the proposed dynamic scheduling scheme considers the remaining execution time of each device for currently executed applications. Moreover, the proposed scheme keeps track of the execution history for the optimal determination of the device to execute an incoming application. This paper analyzes the impact of various scheduling methods including the proposed scheduling scheme on heterogeneous computing systems in terms of performance and energy efficiency.

The rest of this paper is organized as follows. Section 2 describes the CPU/GPU performance comparison and related scheduling schemes. The proposed dynamic scheduling scheme is explained in Sect. 3, and Sect. 4 presents the experimental methods and the detailed evaluation results. Finally, Sect. 5 concludes this paper.

2 Background

2.1 CPU/GPU performance comparison

Figure 1 shows the evaluation results when four different applications are executed on the CPU and the GPU. The horizontal axis denotes the execution time in seconds. The vertical axis represents the executed applications (MM, IMAGE, MP3, and TPACF). MM (Matrix Multiplication) and TPACF (One of the parboil benchmark applications [13]) denote the applications requiring highly intensive computation. IMAGE (Image processing) represents the application requiring highly intensive computation and frequent I/O operations whereas MM and TPACF do not require frequent I/O operations. MP3 (MP3 decoding program) represents the application requiring identical execution time independent on the type of executed device. We used four different applications to compare the performance of the CPU and the GPU with various types of applications. More detailed description of simulated applications and experimental methods is provided in Sect. 4.1.

As shown in the graph, when MM and TPACF are assigned to the GPU, the execution time is reduced compared to the CPU because the GPU is more suitable for processing parallel operations due to its many-core architecture. MM and TPACF contain a large amount of computations, which can be processed in parallel. Therefore, the GPU completes the MM and TPACF much faster than the CPU. For IMAGE application, the GPU cannot provide big performance gain due to frequent I/O operations even though the image processing application is suitable for parallel operations. Contrary to the other applications, the GPU and the CPU show the same execution time when the MP3 decoding program is executed as the MP3 decoding program requires equal playback time.

Note that the performance gain from the GPU is dependent on the application type. The GPU can obtain large performance gain when the application requires a large amount of parallel operations, whereas it can obtain little or no performance gain when the application does not require parallel operations. The performance gain of the GPU is also reduced when frequent I/O operations are required. For this reason, the performance of the heterogeneous computing system can be more improved if the dynamic scheduling scheme can consider the application type.

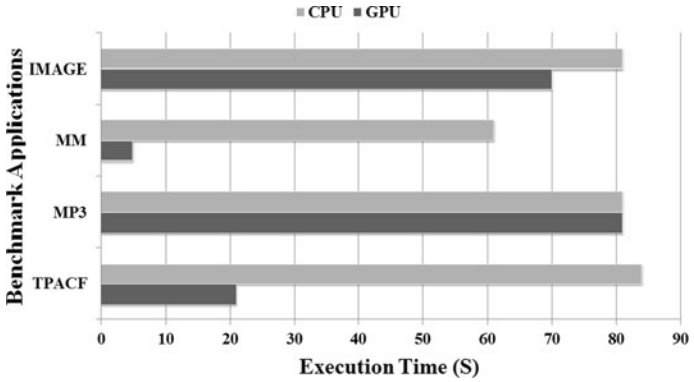

Fig. 1 Execution time of CPU and GPU

Fig. 2 Alternate-Assignment scheduling scheme

```
void Alternate_Assignment(int i)
{
    if(i%2 == 0)
        getCPU(i);
    else
        getGPU(i);
}
```

2.2 Related scheduling schemes

Several scheduling schemes for heterogeneous computing systems have been proposed. The scheduling schemes can be divided into two steps: application and device selection [10]. The application selection is the process for choosing the application to be executed. The device selection is the process for the determination of the device between the CPU and the GPU to execute a selected application. In this work, we apply the first-come, first-served (FCFS) scheme for the application selection to all the scheduling schemes, as this work focuses on the method for the device selection. The existing scheduling schemes for the device selection can be listed as follows: Alternate-Assignment, First-Free, and Performance-History scheduling methods.

The CPU and the GPU are used in round-robin in the Alternate-Assignment scheduling. Figure 2 shows the pseudo code for the Alternate-Assignment scheduling scheme. In the pseudo code, 'getCPU()' and 'getGPU()' imply that the application is assigned to the CPU and the GPU, respectively. As described in the pseudo code, the Alternate-Assignment scheduling does not consider the application type or the status of the device in the selection of the device to execute the application.

The First-Free scheduling scheme is the dynamic scheduling which assigns the application to the idle device [10]. When both the CPU and the GPU are not busy, the application is assigned to the GPU, since the GPU provides better throughput than the CPU. If both the CPU and the GPU are busy, the assignment of the application is delayed until one of the devices becomes available. Figure 3 shows the pseudo code for the First-Free scheduling scheme. In the pseudo code, 'isCpuFree()' and 'isGpuFree()' check the idle status of the CPU and the GPU, respectively. Contrary to

Fig. 3 First-Free scheduling scheme

```
void First_Free(int i)
{
    if (isCpuFree() && isGpuFree())
    {
        if(isGpuFree())
            getGPU(i);
        else
            getCPU(i);
    }
    else if(!isCpuFree() && isGpuFree()){
        getGPU(i);
    }
    else if(isCpuFree() && !isGpuFree()){
        getCPU(i);
    }
    else{
        wait();
    }
}
```

the Alternate-Assignment scheduling, the First-Free scheduling considers the status of the device by checking whether or not the device is occupied by previous applications. By considering the status of devices, the First-Free scheduling can provide better performance in most of the cases than the Alternate-Assignment scheduling. However, the First-Free scheduling cannot guarantee better results consistently over the Alternate-Assignment scheduling as it just considers whether or not the device is occupied by previous applications.

Figure 4 presents the pseudo code of the Performance-History scheduling scheme proposed in [10]. The Performance-History scheduling reads execution-time history information from the file system before selecting the device for the execution of the application. Then, the ratio value is calculated based on (1):

$$ratio = Execution\ Time\ on\ the\ CPU / Execution\ Time\ on\ the\ GPU. \quad (1)$$

By using the calculated ratio value, the scheduler selects the device between the CPU and the GPU to execute the application. If the ratio value is bigger than the predefined upper bound then the application is assigned to the GPU. When the ratio value is between the predefined upper bound and the predefined lower bound, the application is assigned to the device according to the First-Free scheduling scheme. Otherwise, the application is assigned to the CPU as the utilization of the CPU can provide comparable performance to the GPU if the ratio value is less than the predefined lower bound. The Performance-History scheduling uses the execution time information to select the appropriate device between the CPU and the GPU. This results in better performance than with Alternate-Assignment and First-Free scheduling schemes. However, the Performance-History scheduling does not consider the remaining time of currently executed application for each device. It can cause over-utilization or under-utilization of the GPU and the CPU. For example, in case that all the incoming applications are assigned to the GPU depending on the calculated ratio value, the GPU becomes over-utilized while the CPU becomes under-utilized, resulting in performance degradation. Gregg et al. pointed out such problems in [11],

Fig. 4 Performance-History scheduling scheme

```
void Performance_History(int i)
{
    float CPUTime=0, GPUTime=0;
    float ratio=0;

    readExecutionTime(CPlTTime, GPUTime);

    ratio=calculateRatio(CPUTime, GPUTime);

    if(ratio >= upper_bound)
    {
        if(isGpuFree())
            getGPU(i);
    }
    else if( ratio < upper_bound && ratio >= lower_bound)
    {
        First_Free(i);
    }
    else
    {
        if(isCpuFree( ))
            getCPU(i);
    }
}
```

but they did not propose a specific method to overcome these problems. In this work, we propose a new dynamic scheduling scheme for heterogeneous computing systems to select the device to execute the application by considering both execution time history and remaining time.

3 Proposed scheduling scheme

As mentioned above, in heterogeneous computing systems, the selection between the CPU and the GPU for an incoming application is a very important factor in determining the system performance [14]. The objective of the proposed scheduling, called EET (Estimated-Execution-Time) scheduling, is to select the device which can complete the incoming application more quickly by considering both the execution history for incoming applications and the remaining time for currently executed applications. To enable the efficient selection between the CPU and the GPU for an incoming application, the proposed EET scheduling requires history table containing execution time history and remaining time table containing estimated remaining time. The execution time history is the information of previously executed applications, and the estimated remaining time is the information for currently executed applications.

The proposed scheduling requires two history tables: one for the CPU and the other for the GPU. The history table is composed of six entries: *TaskName, Size, Count, Sum, Average,* and *Lifetime*. *TaskName* denotes the application name, and *Size* represents the size of input data. *Count* and *Sum* represent the number of executions and total execution time for corresponding application, respectively. *Average* represents the average execution time for corresponding application calculated by

using *Count* and *Sum*. *Lifetime* is the entry for removing the corresponding entry from the history table when the entry has not been referenced for a predefined time interval to reduce the storage overhead. When selecting between the CPU and the GPU for incoming applications, history tables for the CPU and for the GPU are accessed in parallel to obtain the average execution time if there is a corresponding entry for the incoming application. The history table is indexed by using application name (*TaskName*) and the size of input data (*Size*). The proposed scheduling cannot consider the type of input data yet while it considers the size of input data.

The remaining time table has two entries: one entry for the CPU and the other entry for the GPU. When an application is assigned to the CPU or the GPU, the corresponding entry of the remaining time table is updated by using the information from the history table. After that, the remaining time value in each entry is decreased by one every second to keep track of the remaining execution time information of the CPU and the GPU.

The pseudo code for the proposed scheduling is described in Fig. 5. The proposed scheduling uses the information from the history table similar to the Performance-History scheduling. However, contrary to the Performance-History scheduling, the remaining execution time of the device for the currently executed application is also used in the selection of the device to execute the incoming application. The 'remainingCPUTime' and 'remainingGPUTime' in Fig. 5 denote the remaining execution time information for currently executed applications of the CPU and the GPU, respectively. The remaining execution time can be obtained from the remaining time table. 'CPUTime' and 'GPUTime' are used to store the execution time history of the application to be assigned. They can be read from the file system by using the 'readExecutionTime()' function which reads the value from the history table. Then, the 'estimatedCPUTime' is the sum of 'CPUTime' and 'remainingCPUTime'. The 'estimatedGPUTime' can be calculated by adding 'GPUTime' and 'remainingGPUTime'. Therefore, 'estimatedCPUTime' and 'estimatedGPUTime' denote the estimated execution time when the incoming application is assigned to the CPU and the GPU, respectively.

By using the Estimated-Execution-Time information, the scheduler selects the device that is suitable for the execution of the application. In other words, to select the device between the CPU and the GPU, the 'estimatedCPUTime' is compared with the 'estimatedGPUTime'. The comparison of the 'estimatedCPUTime' with the 'estimatedGPUTime' is classified into three cases. The first case is when the 'estimatedGPUTime' is smaller than the predefined portion of the 'estimatedCPUTime', as shown in Fig. 6(a). The predefined portion is determined by the predefined threshold value (tv), as described in the pseudo code. It implies that the GPU can execute the application much faster than the CPU. Therefore, the application is assigned to the GPU in this case. Figure 6(b) describes the second case when the 'estimatedGPUTime' and the 'estimatedCPUTime' have little difference. In this case, the device to execute the application is selected according to the First-Free scheduling scheme. Therefore, in this case, the application is assigned to the device by considering the idle status of the device. The third case is when the 'estimatedCPUTime' is smaller than the predefined portion of the 'estimatedGPUTime', as shown in Fig. 6(c). In this case, the application is assigned to the CPU in order to reduce the completion time.

```
voidEstimated-Execution-Time(int i)
{
    float CPUTime=0, GPUTime=0;
    float estimatedGPUTime, estimatedCPUTime;

    readExecutionTime(CPUTime, GPUTime);

    estimatedGPUTime = remaingGPUTime + GPUTime;
    estimatedCPUTime = remaingCPUTime + CPUTime;

    if(estimatedGPUTime <= (estimatedCPUTime*(tv)))
    {
        getGPU(i);
        remaingGPUTime = estimatedGPUTime;
    }
    else if(estimatedGPUTime > (estimatedCPUTime*(tv)) &&
          stimatedGPUTime <= estimatedCPUTime ||
          estimatedCPUTime >(estimatedGPUTime*(tv)) &&
          estimatedCPUTime <= estimatedGPUTime)
    {
        if(isGpuFree())
        {
            getGPU(i);
            remaingGPUTime = estimatedGPUTime;
        }
        else if(isCpuFree())
        {
            getCPU(i);
            remaingCPUTime = estimatedCPUTime;
        }
    }
    else if(estimatedCPUTime <= (estimatedGPUTime*(tv)))
    {
        getCPU(i);
        remaingCPUTime = estimatedCPUTime;
    }
}
```

Fig. 5 Proposed Estimated-Execution-Time scheduling scheme

In the proposed scheduling scheme, the scheduler selects the device which can complete the application more quickly by considering the current status of the device and the execution history information. The scheduler selects the device by using the remaining execution time of the CPU and the GPU for currently executed applications and the execution history information of the application to be assigned, resulting in better system performance. After the selection, the remaining execution time of the selected device is updated by using the estimated execution time.

The threshold value (tv) is selected from our previous experiments (shown in Fig. 7) for maximizing the efficiency of the proposed scheduling scheme. In order to determine the efficient threshold value, we use the average for all possible execution sequences of applications, as explained in Sect. 4. In Fig. 7, the vertical axis represents the average execution time of the proposed scheduling according to the threshold value. As shown in the graph, the threshold value of 0.5 provides the best performance even though 0.6 and 0.7 also give comparable performance. For this reason, we set the threshold value to 0.5 for our proposed scheduling scheme.

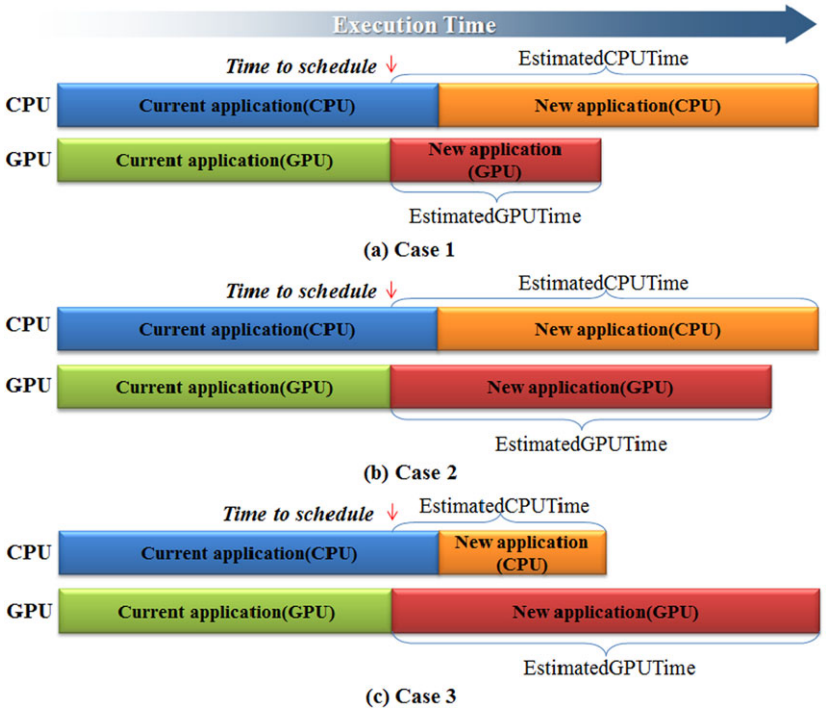

Fig. 6 Case analysis for the proposed Estimated-Execution-Time scheduling scheme

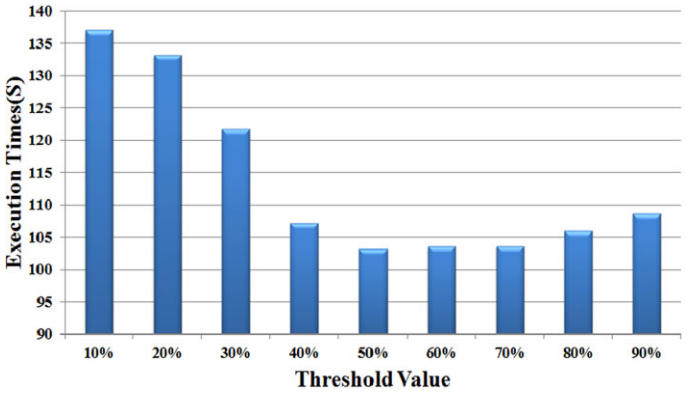

Fig. 7 Performance of the proposed scheduling according to the threshold value (tv)

In the proposed scheduling, the gap between the estimated CPU time and the estimated GPU time can be reduced by the fair distribution of the workload between the CPU and the GPU, enabling more efficient use of computational resources than the existing scheduling schemes. For an example, if the GPU executes a heavy application that requires long execution time, the application optimized for the GPU is

Table 1 Benchmark applications

Application	Abbr.	Description	Input data range
Matrix multiplication	MM	Matrix multiplication GPU version is coded by CUDA	up to 1024 / up to 1744
Two-point angular correlation function	TPACF	TPACF measures the probability of finding an astronomical body at a given angular distance from another astronomical body [13] GPU version is coded by CUDA	up to 170 MB
MPEG audio layer-3	MP3	MP3 decoding program GPU version is coded by CUDA	up to 240 KB
Image processing	IMAGE	Image processing with BMP files GPU version is coded by CUDA	up to 256×512 / up to 512×512

assigned to the CPU than waiting for the GPU. This leads to a better performance than the existing scheduling schemes.

One drawback of the proposed scheduling is that it requires a training period to collect the execution history, since the proposed scheduling cannot be applied without collecting the history information. In this work, the proposed scheme adopts the First-Free scheduling scheme during the training period. Therefore, the proposed scheduling may only be useful for a non-wide set of applications due to the need of the training period.

4 Experiments

4.1 Experimental methods

All experiments were performed under Fedora v.10. Our simulation environment was composed of an Intel 2.66 GHz Core2Quad Q9400 CPU with 2 GB RAM including 3 KB cache per one core and an NVIDIA Geforce 8500GT GPU providing 43.2 GFlops throughput per one shader core with 16 shader cores. In the evaluations, the performance was measured by the actual execution time in seconds and the energy efficiency was measured by the inspector2 tool.

Table 1 presents the detailed description of four benchmark programs (MM, TPACF, MP3, IMAGE). For the evaluation of performance and energy efficiency, we have 24 simulation cases for four applications that vary the sequence of executed applications. We tested all the execution sequences for four applications, as the sequence of the executed applications has strong impact on the efficiency of the scheduling scheme. Then, we used the average value for the comparison.

In the evaluations, ‘CO’ represents the CPU-Only scheduling scheme which assigns all the applications to the CPU and ‘GO’ denotes the GPU-Only scheduling scheme which assigns all the applications to the GPU. ‘AA’, ‘FF’ and ‘PH’ indicate the Alternate-Assignment scheduling scheme, First-Free scheduling scheme and Performance-History scheduling scheme, respectively. ‘EET’ represents the proposed Estimated-Execution-Time scheduling scheme.

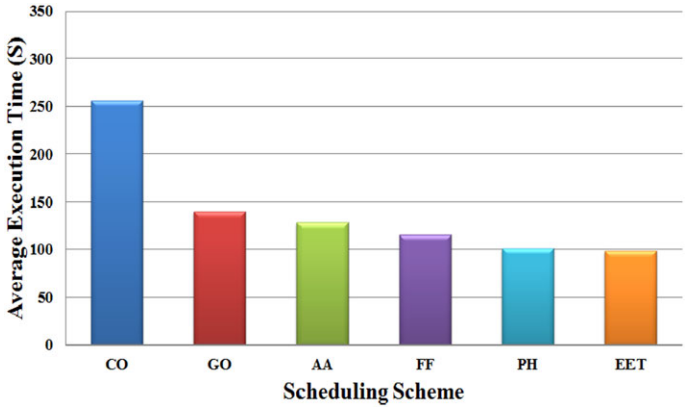

Fig. 8 Average execution time according to the scheduling scheme (Input size of MM: up to 1024, Input size of IMAGE: up to 256 × 512)

4.2 Performance

Figures 8 through 10 show the average execution time for all the execution sequences of four benchmark applications according to the scheduling scheme varying the input size range of MM and that of IMAGE. Figure 8 represents the result where the input size of MM is less than 1,024 and that of IMAGE is less than 256 × 512. Figure 9 depicts the result where the input size of MM is less than 1,024 and that of IMAGE is less than 512 × 512. Figure 10 represents the result where the input size of MM is less than 1,744 and that of IMAGE is less than 512 × 512. As shown in the graphs, the GO scheduling performs much faster than the CO scheduling because the GPU provides higher throughput than the CPU, especially for the parallel applications. Even though the GPU has a powerful throughput, GO shows worse performance compared to the other scheduling schemes that uses both the CPU and the GPU together such as AA, FF, PH, and EET.

AA scheduling scheme selects the device to execute the application depending on the sequence of applications, where the applications are assigned alternately to the CPU and to the GPU. Therefore, each device executes two applications in the experiments without considering the status of the devices. This results in the worst performance among the compared scheduling schemes that use both the CPU and the GPU together. Contrary to the AA scheduling scheme, the FF scheduling scheme considers the status of the device to assign the application to the idle device than the busy device. The FF scheduling scheme improves the performance by 53 % compared to the CO scheduling scheme. However, the FF scheme shows worse performance than the PH scheme and the EET scheme, as the FF scheme only checks whether the device is occupied or not. The AA and the FF scheduling schemes do not consider the characteristics of the applications to be executed in selecting the device. This results in worse performance than with the PH and the EET schemes. For example, the proposed EET scheduling scheme considers that the computation-intensive application assigned to the GPU provides shorter execution time than that to the CPU. This leads to better performance compared to the FF scheme. The PH and the EET

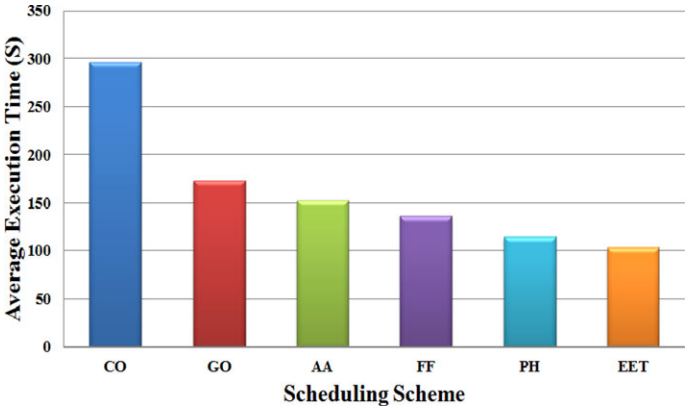

Fig. 9 Average execution time according to the scheduling scheme (Input size of MM: up to 1024, Input size of IMAGE: up to 512 × 512)

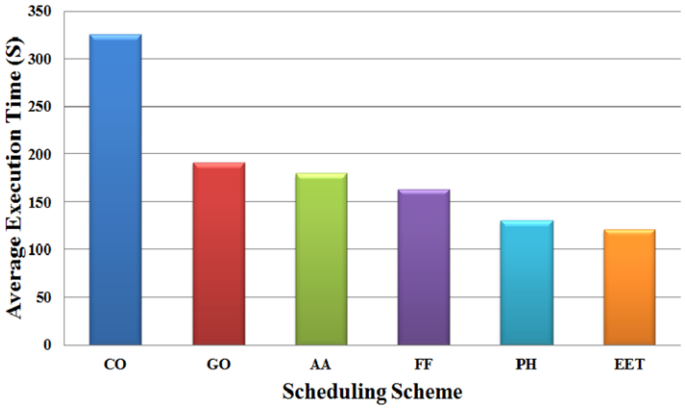

Fig. 10 Average execution time according to the scheduling scheme (Input size of MM: up to 1744, Input size of IMAGE: up to 512 × 512)

scheduling schemes show better performance than the other schemes by considering the execution time history information. The PH scheduling scheme improves the performance by 60 % and the EET scheduling scheme improves the performance by 63 % compared to the CO scheduling scheme. As shown in the graphs, the proposed EET scheduling scheme shows the best performance. The performance gain of the proposed EET scheme compared to the PH scheme comes from the fact that the EET scheduling can improve the utilization of the CPU and the GPU by considering the execution history and remaining time information together, as shown in Table 2.

Figures 11 and 12 present the ratio of the execution time of the CPU and the GPU for PH and EET scheduling schemes, respectively. The vertical axis of the graph indicates the execution time ratio. The horizontal axis of the graph denotes the executed sequence of the simulated applications. In the graphs, P, M, I, and T represent MP3, MM, IMAGE, and TPACF, respectively. For example, T–M–I–P means that the ap-

Table 2 Utilization of CPU/GPU for PH and EE scheduling methods

	Device	PH scheduling	EET scheduling
Figure 8 (MM: up to 1024 IMAGE: up to 256 × 512)	CPU	82.06 %	82.25 %
	GPU	69.75 %	84.81 %
Figure 9 (MM: up to 1024 IMAGE: up to 512 × 512)	CPU	90.85 %	83.08 %
	GPU	80.68 %	100 %
Figure 10 (MM: up to 1744 IMAGE: up to 512 × 512)	CPU	79.99 %	70.86 %
	GPU	83.23 %	100 %

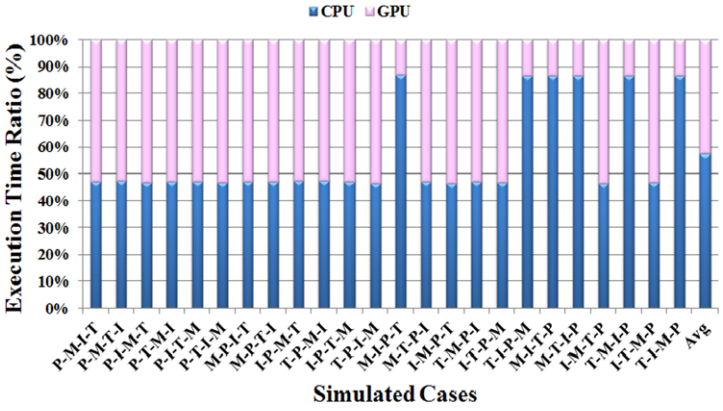

Fig. 11 Execution time ratio of CPU/GPU for the PH scheduling scheme

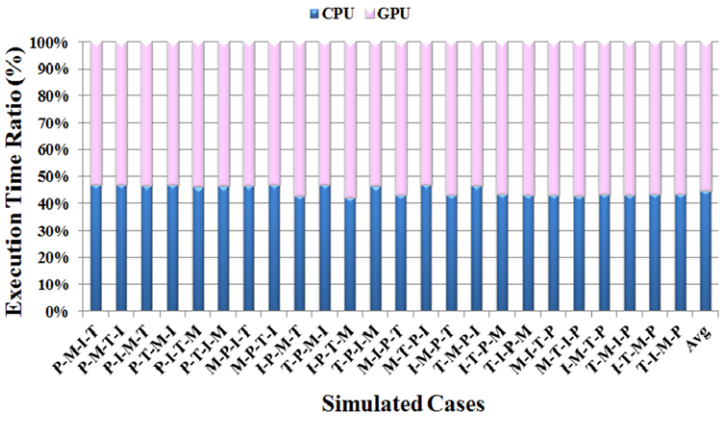

Fig. 12 Execution time ratio of CPU/GPU for the EET scheduling scheme

plications are executed starting from TPACF, then MM, then IMAGE, and then MP3. The lower portion of the bar denotes the execution time ratio of the CPU and the upper portion of the bar represents the execution time ratio of the GPU.

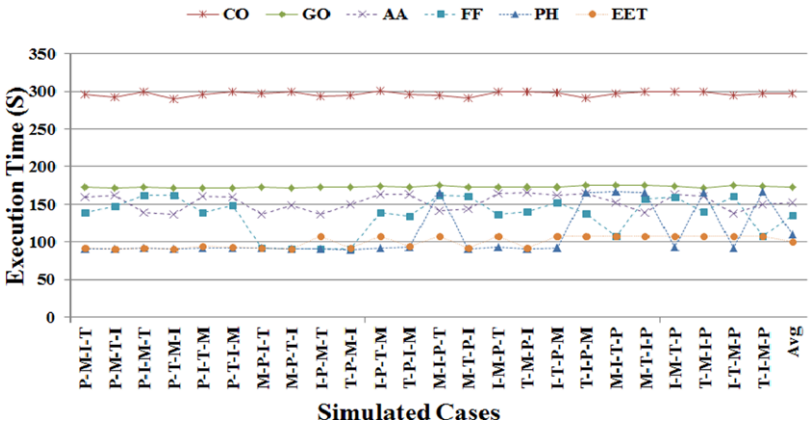

Fig. 13 Execution time for all the simulated cases

As shown in Figs. 11 and 12, the boundary between the execution time ratio of the CPU and that of the GPU is nearer to 50 % in the proposed EET scheduling compared to the PH scheduling. In other words, the execution time of the CPU and that of the GPU are almost the same in the proposed scheme. In the PH scheduling, most of the boundaries are near to 50 % but in some cases, the boundaries are near to 85 %, as shown in Fig. 11. It implies that the execution time of the CPU is much longer than that of the GPU due to the unbalanced workload distribution between the CPU and the GPU. Therefore, the proposed EET scheduling shows better performance than the PH scheduling by providing more balanced workload distribution between the CPU and the GPU.

4.3 Performance consistency

In this section, we analyze the performance consistency for all the executed sequences of applications according to the scheduling scheme. The execution time for all the simulated cases is shown in Fig. 13. The CO scheduling and the GO scheduling show more consistent performance compared to the other four scheduling schemes, but they show worse performance compared to the other schemes. The CO and GO scheduling schemes provide consistent performance as they use only one device, resulting in no difference according to the executed sequence. As shown in the graph, the execution time of AA, FF, PH, and EET scheduling schemes are affected by the execution order.

The AA and the FF scheduling schemes show worse performance consistency than the other schemes, as they do not consider the characteristics of the applications to be assigned. The PH scheduling also shows worse performance consistency than the proposed EET scheduling, as it selects the device based on the execution time history, not considering the remaining execution time for currently executed applications. This results in an unbalanced workload distribution between the CPU and the GPU. The AA, FF, and PH scheduling schemes are affected by the execution order of the applications and show unstable results. Compared to these schemes, the proposed EET scheduling provides the best average execution time and the best performance consistency.

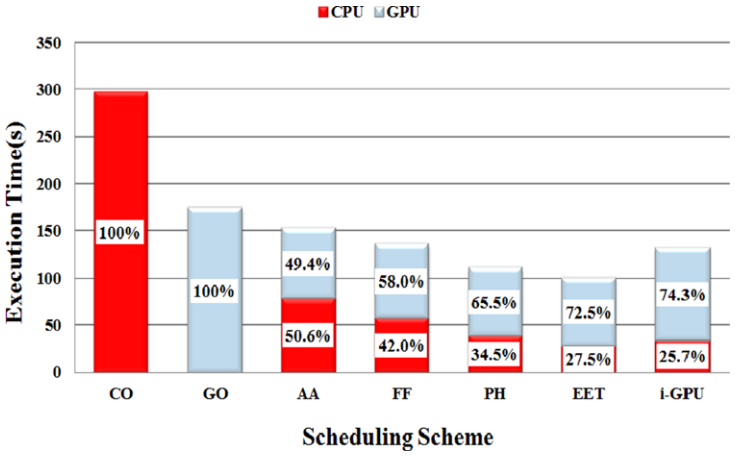

Fig. 14 Execution time and distribution of workload

4.4 Workload distribution

In order to compare the workload distribution on the CPU and the GPU according to the scheduling scheme, we use the following equations:

$$\begin{aligned} &\textit{CPU Workload Distribution} \\ &\quad = \textit{CPU Execution Time} \,/\, \textit{CPU Execution Time of the CO}; \end{aligned} \tag{2}$$

$$\textit{GPU Workload Distribution} = 1 - \textit{CPU Workload Distribution}. \tag{3}$$

As shown in Fig. 14, the horizontal and the vertical axis indicate the scheduling scheme and the average execution time, respectively. Each bar is divided into two parts: the lower portion implies the workload distribution on the CPU and the upper portion denotes the workload distribution on the GPU. The execution time is generally reduced as the workload of the GPU increases as shown in Fig. 14.

We also test the case where the workload of the GPU is three times the workload of the CPU. It is denoted as intensive-GPU (i-GPU) in the graph. Compared to the proposed EET scheduling, the i-GPU has more workload on the GPU. Nevertheless, this scheme shows longer execution time than the proposed scheduling scheme. From these results, we know that assigning lots of workloads unconditionally to the GPU cannot guarantee high performance. As shown in the graph, the proposed EET scheduling provides the best performance for the heterogeneous computing system by the fair distribution of the workload between the CPU and the GPU.

4.5 Energy consumption

Figures 15 and 16 show the average power consumption and the energy consumption according to the scheduling scheme, respectively. As shown in Fig. 15, the scheduling schemes that use both the CPU and the GPU together require more power than the CO scheduling and the GO scheduling, as they utilize more resources in the execution of

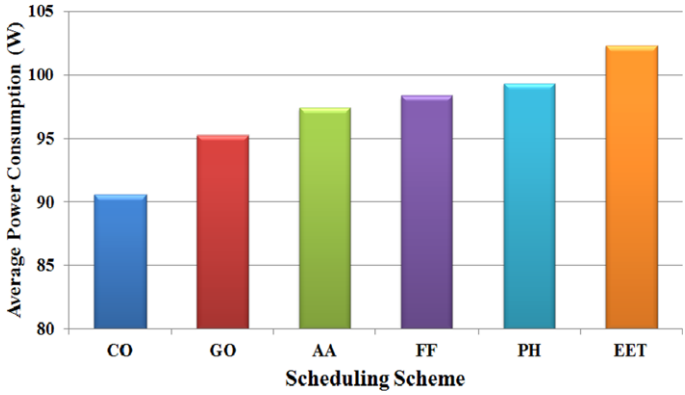

Fig. 15 Average power consumption

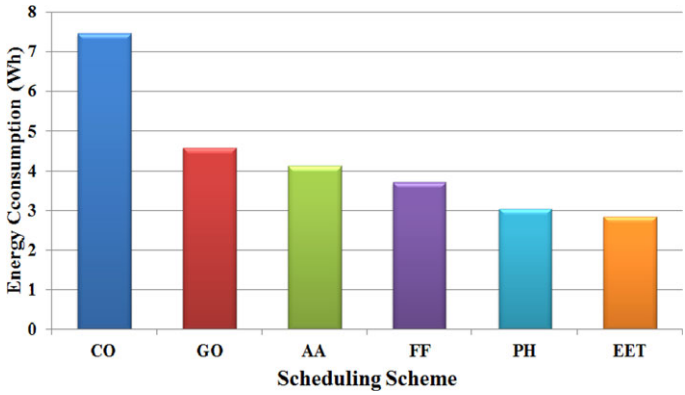

Fig. 16 Energy consumption

the applications. However, in terms of energy consumption, the scheduling schemes providing better performance consume less energy by reducing the execution time as the energy consumption is proportional to the execution time and power consumption. Therefore, although the proposed EET scheduling consumes more power than the other scheduling schemes, it consumes less energy than the other schemes by reducing the execution time. Compared to the CO scheduling, the EET scheduling reduces the energy consumption by 62 %. Thus, the proposed EET scheduling can be a solution for reducing the energy consumption while providing high performance.

5 Conclusions

In this paper, we analyzed the impact of the scheduling scheme on the heterogeneous computing systems that use both the CPU and the GPU together. The proposed estimated-execution-time scheduling considers the remaining execution time of the CPU and the GPU for currently executed applications. It also considers the expected

execution time for the incoming application based on the execution time history. According to our simulations, the proposed scheduling scheme improves the system performance by maximizing the resource utilization of the CPU and the GPU. Moreover, the proposed scheduling scheme provides better performance consistency than existing scheduling schemes for the executed order of applications. The estimated-execution-time scheduling also shows the best energy efficiency by reducing the execution time. Therefore, we expect that the proposed scheduling can be a solution for improving the performance and the energy efficiency of heterogeneous computing systems. One drawback of the proposed scheduling scheme is that it requires a training period to collect the execution history. In a future work, we will investigate how the training period can be reduced.

Acknowledgements This work was supported by the National Research Foundation of Korea Grant funded by the Korean Government (NRF-2011-013-D00105, 2012R1A1B4003492) and the ITRC (Information Technology Research Center) support program supervised by the NIPA (NIPA-2012-H0301-12-3005).

References

1. Agarwal V, Hrishikesh MS, Keckler SW, Burger D (2000) Clock rate versus IPC: the end of the road for conventional microArchitectures. In: Proceedings of 27th international symposium on computer architecture, pp 248–259
2. Eberly DH (2001) 3D game engine design. Morgan Kaufmann, San Francisco
3. Buck I, Foley T, Horn D, Sugerman J, Fatahalian K, Houston M, Hanrahan P (2004) Brook for GPUs: stream computing on graphics hardware. In: Proceedings of 31th annual conference on computer graphics (SIGGRAPH), pp 777–786
4. Owens JD, Luebke D, Govindaraju N, Harris M, Kruger J, Lefohn AE, Purcell TJ (2005) A survey of general-purpose computation on graphics hardware. In: Euro-graphics 2005, state of the art reports, pp 21–51
5. GPGPU. Available at http://www.gpgpu.org
6. NVIDIA CUDA Programming. Available at http://www.nvidia.com/object/cuda_home_new.html
7. Che S, Meng J, Sheaer J, Skadron K (2008) A performance study of general purpose applications on graphics processors using CUDA. J Parallel Distrib Comput 68(10):1370–1380
8. Ryoo S, Rodrigues CI, Baghsorkhi SS, Stone SS, Kirk DB, Hwu WW (2008) Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. In: Proceedings of the symposium on principles and practice of parallel programming, pp 73–82
9. Akenine-Moller T, Haines E (2002) Real-time rendering, 2nd edn. AK Peters, Natick
10. Gregg C, Brantley JS, Hazelwood K (2010) Contention-aware scheduling of parallel code for heterogeneous systems. In: Proceedings of the 2nd USENIX workshop on hot topics in parallelism, 6 pages
11. Gregg C, Boyer M, Hazelwood K, Skadron K (2011) Dynamic heterogeneous scheduling decisions using historical runtime data. In: Proceedings of the 2nd workshop on applications for multi- and many-core processors, 12 pages
12. Jimenez V, Vilanova L, Gelado I, Gil M, Fursin G, Navarro N (2009) Predictive runtime code scheduling for heterogeneous architectures. In: Proceedings of the 4th international conference on high performance embedded architectures and compilers, pp 19–33
13. Parboil benchmark suite. Available at http://www.crhc.uiuc.edu/impact/parboil.php
14. YuHai Y, Shengsheng Y, XueLian B (2007) A new dynamic scheduling algorithm for real-time heterogeneous multiprocessor systems. In: Proceedings of the workshop on intelligent information technology application, pp 112–115