Regular Paper

Torta: Extending Applications' Capability to Select Heterogeneous Computing Resources Adaptively to the PC Usage

Tetsuro Horikawa[1,a) Jin Nakazawa[1,b) Kazunori Takashio[1,c)
Hideyuki Tokuda[1,d)

Received: April 9, 2013, Accepted: August 5, 2013

Abstract: Spread of GPU-accelerated applications on PCs can cause serious degradation of the user experience such as frame dropping on the video playback, due to applications' resource competition on the same GPU due to arbitrary processors selection. In this paper, we propose a processors assignment system for real applications that achieves processors assignment according to condition based rules without modifying applications. To demonstrate the feasibility of our concept, we implemented a prototype of the centralized processors assignment mechanism called Torta. Our experiment using eight practical applications has shown that Torta achieves binary-compatible processors switching with an average performance penalty on only 0.2%. In a particular case where a video playback application is executed with three other GPU-intensive applications, our method enables users to enjoy the video playback with 60 frames per second (FPS) while the FPS decreases to 14 without the mechanism. This paper shows the design and the implementation of Torta on Windows 7 and concludes that our mechanism increases the efficiency of computational resource usage on PCs, thus improves the overall user experiences.

Keywords: processors assignment, resource management, users' preference, users' context, GPU, OpenCL, binary-compatibility

1. Introduction

Recent Graphics Processing Units (GPUs) have been acquiring powerful parallel computation capabilities. PC applications, such as photo editors, have been rewritten to leverage this capability. These so-called GPU-accelerated applications choose their computing resources from CPUs and GPUs arbitrarily, which may cause degradation in performance due to resource competition on a single processor. For instance, in our preliminary experiments, we found that the frame rate of video playback with motion interpolation by the application Splash PRO EX [29] significantly decreases when multiple applications run concurrently on the same GPU. The spread of GPU-accelerated applications will cause such user experience degradation frequently in the future. This necessitates a system-wide mechanism that manages heterogeneous processors, i.e., GPUs and CPUs, on a PC.

To suppress such performance degradation, many studies [24], [25], [34] have proposed fine-grained GPU management schemes. They limit the GPU utilization time of each application and schedule applications according to their priorities. However, they can still cause performance degradation on particularly important applications due to GPU utilization by many unimportant applications. Furthermore, such studies do not achieve tasks distribution between CPUs and GPUs on existing applications. There is another problem with existing studies in that they do not consider the heterogeneity of GPUs use. For instance, GPUs or video player applications on CPUs output different images. In such a case, switching the processor makes non-linear but generally acceptable results in both cases. Therefore, it is difficult to manage heterogeneous processors of PCs by linear scale management models, such as priority models. In summary, there are the following three requirements to prevent the degradation of user experience caused by inappropriate processors assignment on real PCs:

- A tasks distribution mechanism that supports processors installed on PCs such as CPUs and GPUs.
- Binary compatibility for existing applications.
- A mechanism to assign heterogeneous processor by non-linear rules.

In this paper, we implemented a middleware prototype of middleware called **Torta** that achieves processors switching on real applications. Our experimental results show that Torta's high compatibility to real applications incurs a small overhead. Although our implementation has limitations, binary compatibility with existing applications is a large advantage over other studies. In addition, middleware implementation achieves compatibility with existing operating systems (OSes) and drivers. The results demonstrate the short-term feasibility of middleware implementation of devices switching mechanism.

1 Keio University, Fujisawa, Kanagawa 252–0882, Japan
a) techi@ht.sfc.keio.ac.jp
b) jin@ht.sfc.keio.ac.jp
c) kaz@ht.sfc.keio.ac.jp
d) hxt@ht.sfc.keio.ac.jp

This paper makes the following contributions:

- It provides quantitative evidence that a GPU-accelerated application easily and heavily interferes the performance of others.
- Our experimental results show quantitative evidence that redirecting processors of applications significantly prevents performance degradation.
- It outlines the requirements for achieving heterogeneous processors assignment on PCs to suppress problems caused by the PCs' complicated heterogeneity of processors and processors' uses.
- It introduces the Torta system for redirecting real applications' processors with negligible overhead and without modifying applications.
- It provides a mechanism for applying non-linear rules into heterogeneous processors assignment.

On the other hand, this paper does *not* consider the following, which we leave for future work:

- The best principle of heterogeneous resource management mechanism such as fine-grained GPU resource management systems.
- A scheme for expressing actual conditions and policies that are necessary for processors assignment.
- A mechanism for converting actual users' preferences into assignment policies.

The rest of this paper is organized as follows. Section 2 states the requirements for assigning heterogeneous processors adaptive to conditions such as users' preferences. Section 3 describes the design and implementation of Torta, and Section 4 demonstrates our detailed experimental results. Section 5 discusses related work. We conclude with remarks and some directions for future work in Section 6.

2. Torta: Cetralized Heterogeneous Processor Assignment Middleware

To state the necessity of our concept and to organize problems and requirements for heterogeneous processors assignment on PCs, we assume that a PC consists of one or more multi-core CPUs and GPUs that are capable of executing General-Purpose computing on GPU (GPGPU) tasks. GPUs are not limited to discrete GPUs (dGPUs); integrated GPUs (iGPUs) are also our targets.

2.1 Heterogeneity of GPU Uses and Static Processors Selecting

GPU utilization in PCs can be generally classified into one of the following five categories:

3D graphics acceleration 3D games, 3D graphics based GUI shell, modeling applications

2D graphics acceleration font rendering acceleration used in modern web browsers

General purpose computing (GPGPU) simulation and data compression

Video encoding acceleration video transcoders

Video decoding acceleration video players

Most of these GPU-accelerated applications, except for 3D graphics applications, allow processor selection in a configuration menu. They also have programs which allow both CPUs and GPUs to do the almost same computation internally. In particular, applications with history inherit well optimized CPU programs from previous versions. In addition, applications that are built with processors abstraction layers (e.g., OpenCL) select processors by specifying the ID of the processor. Thus, they can also select processors by changing the processor's ID while calling APIs of abstraction layers.

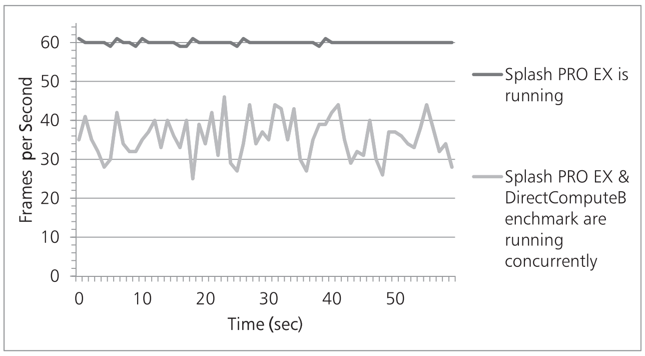


Fig. 1 FPS of video playback decreased when another application started utilizing the same GPU.

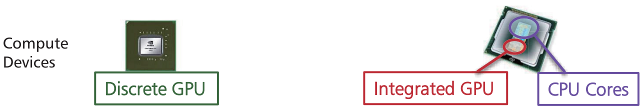


Fig. 2 Common Combination of Processors on PCs.

However, GPU-accelerated applications can compete for the same GPU because they select processors statically. For instance, as shown in **Fig. 1**, the frame per second (FPS) of video playback on Splash PRO EX [29] decreased from 60 to about 35 due to the static assignment of the same GPU by another application called DirectComputeBenchmark. Since performance degradation due to frame dropping results in a serious impact on the causes serious degradation of the user experience, we need a mechanism for applications to dynamically yield the rights of GPU to another.

2.2 Processor Heterogeneity and Run-time Selection

Processors' heterogeneity on PCs is increasing. Many recent CPUs (e.g., 3rd generation Intel Core processors *a.k.a.* Ivy Bridge) have an internal GPU in their dies. As shown in **Fig. 2**, in addition to a CPU-integrated GPU (iGPU), most desktop PCs or desktop-replace laptop computers embed one or more discrete GPUs (dGPUs) to improve performance.

The greatest difference between integrated GPUs (iGPUs) and discrete GPUs (dGPUs) is the location of their video memory (VRAM). The VRAM of a dGPU is usually implemented outside main memory (RAM). GPGPU tasks thus need copy and copy-back data between RAM and VRAM, resulting in a large overhead. This overhead has the potential to impact the performance advantage of GPUs negatively. Therefore, many studies [8], [13], [22], [26] figured out the threshold for selecting a CPU or a GPU, in order to provide a better overall net effect. On the other hand, most iGPUs use a part of main memory as their VRAM. Thus, some iGPUs allow accessing their memory without copying by a zero-copy function. When a PC has an iGPU

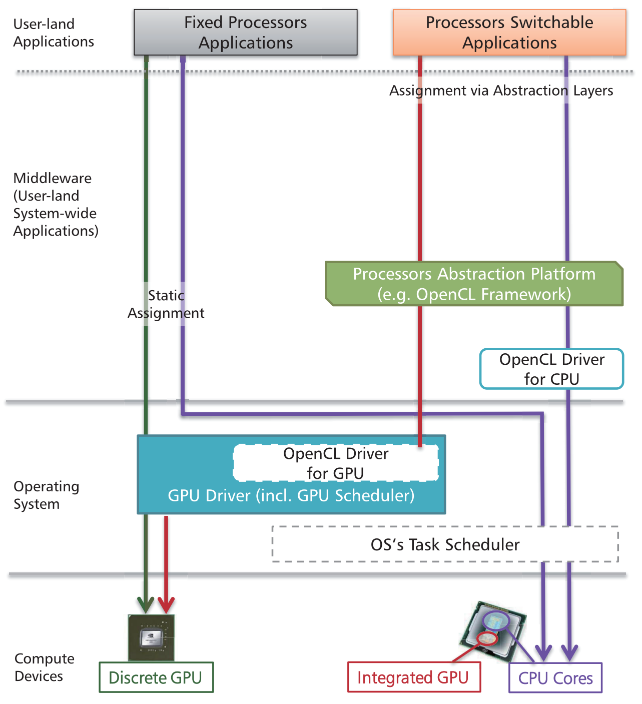

User-land Applications
Fixed Processors Applications
Processors Switchable Applications
Assignment via Abstraction Layers
Middleware (User-land System-wide Applications)
Static Assignment
Processors Abstraction Platform (e.g. OpenCL Framework)
OpenCL Driver for CPU
Operating System
OpenCL Driver for GPU
GPU Driver (incl. GPU Scheduler)
OS's Task Scheduler
Compute Devices
Discrete GPU
Integrated GPU
CPU Cores

Fig. 3 Arbitrary processors assignment by applications.

with a zero-copy function, an iGPU sometimes suits better than a CPU for computing with small data size. Variable clock speed technologies such as Turbo Boost on Intel CPUs also increase processors' heterogeneity. These technologies allow changing the clock speed of each CPU core depending on the load of each core. Since the thermal design of a processor limits the total power consumable by the all cores of a processor, CPU cores and an iGPU can compete for the clock speeds.

Since applications select processors arbitrarily, as in **Fig. 3**, GPUs are not always utilized effectively. The simplest scheme for assigning processors to applications, instead of arbitrary selection, is to make applications select a processor according to the load of the processors. However, this method may let applications select a GPU that results in only a small or no performance improvement. Moreover, it may still cause performance degradation of applications already running on the GPU. In the case of Fig. 1, the frame per second (FPS) of the video playback decreased from 60 to about 35 despite Splash causing only 45% of the GPU load. Due to the difficulty in collecting information about GPU utilization by others, it is difficult for each application to predict the influence of its own GPU utilization. The greatest reason of unpredictable performance influence is the coarse-grained management of GPUs resource. More precisely, current OSes treat GPUs as input/output (IO) devices because GPUs are usually accessible via PCI express buses. Therefore, achieving the performance isolation and the tasks prioritization [24], [25] on GPUs can suppress performance degradation in most cases. However, such methods cannot prevent degradation of the user experience because they still allow unimportant applications to utilize GPUs. Therefore, a centralized manager should assign a processor for each application adaptively to system-wide information of processors utilization.

2.3 Condition Based Assignment

For interactive devices like PCs, users' experience strongly depends on the individual preferences of users. Therefore, we need a condition based assignment model which can adapt to the users' contexts and preferences. For instance, in some cases, a user hopes that the game runs smoothly to achieve a higher score than his/her friends. In another case, he/she does not seriously care about the FPS of the same game because he/she plays it to pass the time. To prevent degradation of the user experience, we need to assign processors adaptively to the current user's context. We believe that the concept of user preference based scheduling will supersede legacy priority based scheduling on user interactive computers.

Current PC OSes scarcely adapt information pertaining to the user's context to their schedulers. Actually, state of the AC power connection and the foreground application are the only parameters that adapt to OS schedulers and processors' drivers. For instance, even when the browser works in the foreground, a user may be just searching for information about a movie that is shown in the adjacent window. On the other hand, a user may just hear the audio of the concert movie while writing a paper. To improve the user experience, resource management systems should use such context information. Moreover, we actually need such information to assign processors on PCs with complicated heterogeneity.

However, in reality, we cannot assign processors only by information about the user context. When we measured the FPS of the same experiment as Fig. 1 with a different PC, the FPS decreased to only 45 FPS from 60 FPS. Thus, frame dropping may not be noticed by the user when the user does not direct much attention to the video. Besides, some users do not mind when so few frames are dropped; this completely depends on the user's preference.

Processors assignment does not only cause a difference in FPS; it also causes small differences in video frames. Because decoding acceleration and post-processing are provided by GPUs hardware and their driver, they are not the same as CPU codes implemented in applications. Some users prefer one of their output in spite of the small difference. Therefore, we cannot decide on the proper processor according to linear metrics such as computing speed. Assigning a faster processor to a task with higher priority does not always satisfy the users' preferences.

2.4 Requirements

There are two key requirements for demonstrating the feasibility of our concept on the processors assignment system for PCs. One is task distribution with binary compatibility to existing applications. The other is condition based assignment for applying users' contexts and preferences.

Firstly, the processors assignment system must retain binary compatibility with existing applications. Previously released applications are the greatest property of PC OSes such as Windows and Mac OS; losing compatibility decreases the user's experience. Tightly limiting assignment targets to applications built with a specific library [8], [13], [22] is one of the straightforward ideas to enable processors switching. However, application vendors are hesitant about re-designing their applications so that they are forced to use a CPU. Another idea to achieve processors switching is modifying the applications binaries. However, modifying applications by a third party can violate the law or applications' licenses. Besides, technically speaking, the cost of making binary patches for each update is too expensive. One idea for achieving binary compatibility is creating a dynamic recompilation system (e.g., Apple Rosetta) or a system like virtual machines. However, these technologies can spoil the benefits of hand-tuned code implemented in applications. Besides, these technologies need a long time for the implementation and the validation. Consequently, we need a mechanism that achieves binary compatible processors switching with short deployment time and small performance penalty.

Secondly, as stated in Section 2.3, resource management systems need to adapt with the users' contexts and preferences. Therefore, we need a new mechanism that can assign devices according to conditions instead of priority models. In our method, we set users' contexts as conditions and preferences as assignment rules for each context.

2.5 Design Space

There are few methods that are supposed to prevent performance degradation of GPU-accelerated applications. One is modifying OSes [25], [34]. OSes can manage GPU resource at a small level of granularity. In addition, OSes have the authority to manage any abstracted computation resource. Therefore, modification to OSes has the potential to distribute tasks to CPUs and GPUs. However, modification of OSes often needs a long time to be merged into released versions. Besides, newly implemented system calls [34] need modifications to applications. Actually, these studies do not distribute tasks between all of the installed GPUs and CPUs. Another option is to modify drivers to manage GPUs' resources [24]. These drivers achieve fine-grained resource management on GPUs without modifying applications. In addition, they can potentially support any APIs designed for utilizing GPUs. However, this approach cannot distribute tasks between GPUs and CPUs either. Although drivers are parts of OSes, the goal of modifying drivers is separate from modifying OSes. Drivers modification methods optimize resource management of a GPU or GPUs provided by the same vendor. On the other hand, OSes modification methods optimize resource management of any GPUs and CPUs installed on the PC. A third option is creating a middleware [8], [13], [22] to switch processors. Although they achieved processors switching, these do not provide binary compatibility with real applications.

To demonstrate the effectiveness of our method, the implemented system must achieve processors switching with real applications. Therefore, we have implemented middleware using an API hooking method. API hooking allows the changing of parameters and/or return values while hooking specific APIs. Hooking APIs allows a system to change the behavior of applications without modifying binaries. However, our method does not limit the design space to middleware; it is also applicable to OSes. Rather, to improve users' experience, resource management systems of OSes should adapt to users' contexts and preferences. Implementing our concept into OSes is supposed to be more appropriate as a long-term solution.

3. Prototype Design and Implementation

In this section, we state our target environments and the design concept of Torta.

3.1 Processors Utilization Model

To evaluate our method, we focus on applications that are built with OpenCL in this paper. However, our method is also applicable to applications that utilize heterogeneous processors through the same model.

Compute Device

Processors, such as CPUs and GPUs, are abstracted as *compute devices* in OpenCL. However, such an abstraction layer is not mandatory when applying our method. It is applicable when the application recognizes each processor as a kind of compute device and has executable code for each compute device.

Devices List

OpenCL APIs provide the device listing function, which lists devices by specifying a device type (e.g., CPU, GPU) on each OpenCL platform provided by each hardware vendor. Our method does not limit the format of devices list; it is applicable to applications that select a compute device from arbitrarily-formatted lists that contains available compute devices.

Context

To utilize a compute device via OpenCL, applications create an *OpenCL context* which is similar to a virtual computer. Before executing tasks on a specific compute device, applications need to compile their *OpenCL C* programs for the context that includes the target compute device. This step is only needed for systems that abstracts compute devices. In other cases, our method does not need objects like OpenCL contexts.

In addition to applications that follow the above model, our method can be applied to applications that have programs for both CPUs and GPUs internally. For instance, a common web browser called "Firefox" can use GPUs to accelerate its rendering. Acceleration can be enabled and disabled by changing settings stored in the "prefs.js" profile. Therefore, we can also switch devices on such applications by changing settings which are stored in their setting files or Windows registries. Although opportunities to switch devices are limited to when applications read their settings, this method also achieves binary compatibility with existing applications. Still, applications can bypass such compute devices switching mechanisms to utilize a compute device arbitrarily. Therefore, our design cannot manage all of the compute device assignment. However, this limitation is not a fundamental problem. More precisely, some tasks such as graphics rendering always need a specific type of compute devices. Also, GPU resource management mechanisms [24], [25] can work cooperatively with Torta to suppress problems caused by applications that do not allow compute devices from switching.

3.2 Centralized Resource Management Mechanism

To eliminate the need for any modification of operating systems, we implemented the **Resource Manager (RM)** of Torta as

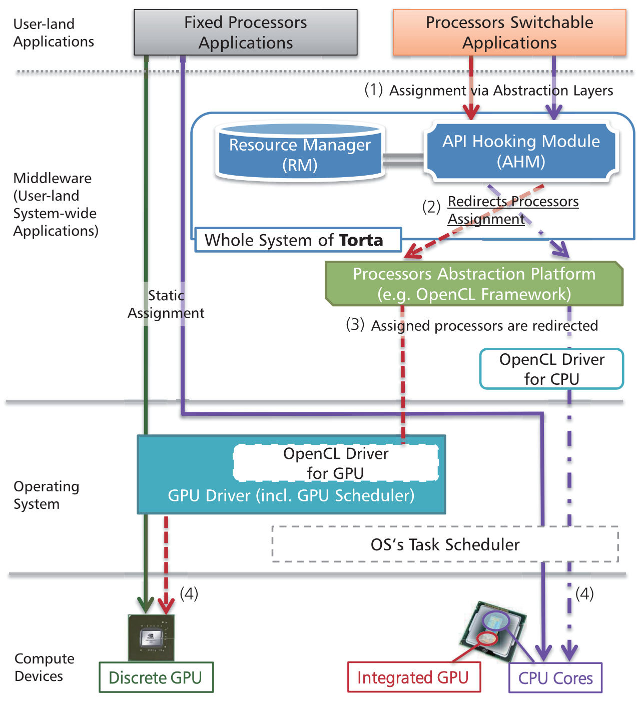

User-land Applications
Fixed Processors Applications
Processors Switchable Applications
(1) Assignment via Abstraction Layers
Resource Manager (RM)
API Hooking Module (AHM)
Middleware (User-land System-wide Applications)
(2) Redirects Processors Assignment
Whole System of Torta
Processors Abstraction Platform (e.g. OpenCL Framework)
Static Assignment
(3) Assigned processors are redirected
OpenCL Driver for CPU
OpenCL Driver for GPU
GPU Driver (incl. GPU Scheduler)
Operating System
OS's Task Scheduler
(4)
(4)
Compute Devices
Discrete GPU
Integrated GPU
CPU Cores

Fig. 4 Assumed environment and centralized resource management model.

a user-land application (middleware).

On the other hand, to achieve devices switching with binary compatibility, we implemented the **API Hooking Module (AHM)** that hooks all OpenCL APIs to change compute device assignment by cooperating with the RM.

Figure 4 shows the working layers of our work. (1) When applications try to utilize a device via an abstraction layer, Torta hooks on to the APIs to change the APIs parameters and/or return values. (2) Then, the AHM asks the RM for the appropriate device of the application. To decide on an appropriate device for applications, the RM always collects various data such as devices' loads and the user's context. The RM can ask about the appropriate device from the user when it cannot automatically decide it. (3) The AHM changes parameter and/or return value according to the answer of the RM. After that, the AHM calls the original APIs of the abstraction layer. (4) As a result, applications use more appropriate devices than originally specified.

To redirect device assignment, the RM and the AHM communicates with each other for sharing devices' information. Firstly, the AHM sends the list of all available compute devices to the RM. Secondly, the RM chooses an *appropriate* device for the application adaptively to the user's context and preference, and answers the selected device to the AHM. Thirdly, the AHM changes the ID of the compute device during each API call to redirect the device assignment. Lastly, the AHM sends the actual assignment information into the RM. Following these steps, both device assignment information and device assignment authority are centralized. Therefore, Torta can achieve central resource management between CPUs and GPUs.

3.3 Management of Users' Contexts and Preferences

As stated in Section 2, resource management of heterogeneous processors on PCs must be based on condition based assignment in order to deal with users contexts and preferences. Because a system cannot infer all the situations that are brought by his/her *context* and *preference*, Torta asks the user to express his/her context and preference via *profiles*. A profile corresponds to a context; users create several profiles to express their contexts. A profile is constructed from two parts, the condition part and the assignment rule part. The condition part expresses the user's context. The assignment rule part expresses the user's preference. Therefore, Torta can assign the appropriate device expressed in the profile which matches the current context.

To express situations, our method accepts various information that can help express the user's situation. Some examples include where the user is, how many key strokes per minute, which part of the display the user looking at, and so on. In particular, the impact on the user experience varies depending on the user's attention to the application. However, we cannot easily imagine all of them because collectible information strongly depends on the hardware configuration. Therefore, we do not focus on creating the format for expressing users' contexts; our current implementation supports running applications and AC connection as inputs to evaluate whole concept. Also, switching profiles adaptively to user's context is inevitably out of our focus because we have not precisely defined how such information can be expressed.

On the other hand, expressing assignment rules is essential for Torta to select devices internally. To avoid writing assignment rules for all applications, we need to reduce the number of applications that express the preferred assignment. Real world applications can be generally classified into the following three types according to the impact on the user experience and computation intensity.

An impact on the user experience with computation intensity: Applications such as 3D games, video players with rich image processing, creative applications for multimedia such as video, image and music usually have a large impact on the user experience when a user seriously works on the task with the application. These applications are computationally intensive and need to be responsive. In addition, users usually need to focus on such applications.

An impact on the user experience without computation intensity: Applications like Microsoft Office file creation, music playback and access of encrypted file systems usually have a large impact on the user experience. Slow response or sound skipping makes users frustrated. However, these applications do not need a large amount of computing resources.

A small impact on the user experience: Although applications like file compression and video or audio file encoding are important, such non real-time tasks do not affect the user experience strongly. Even slow processing of these applications will rarely frustrate the user.

According to this classification, we need special attention to the assignment of compute devices to applications based on the impact on the user experience and computationally intensity. However, necessary computing resources vary depending on the actual purpose of the application. In addition, importance of applications also vary depending on the user's context and preference. Therefore, we define the following two types of rules for expressing assignments that adapt to the user's preference.

Fixed Assignment

This rule fixes the assignment of a specific device to applications based on their importance to the user. Because this rule is *fixed*, Torta always assigns the specified device to the application even when the load of the target device is high. To make the system more effective, we define the following two non-exclusive options:

- **Monopoly**
  This option assigns a specific device to a specific application. Torta never assign the device to other applications managed by Torta. However, we cannot achieve actual *monopoly* because Torta cannot manage all of the running applications. Thus, we need other schedulers to ensure this option remains effective: e.g., OSes' schedulers for CPUs and schedulers of modified drivers [24] for GPUs.
- **Virtual**
  This option is needed for applications that Torta cannot manage assignment of devices. For instance, expressing a game application with a GPU using this option helps to improve frame rate of the game by switching the devices of other applications to the CPU.

Prioritized Assignment

This rule allows expressing the user desire to assign a specific device to an application because the application is moderately important. Applications specified as *Fixed Assignment* always precede others. Torta cannot always assign specified device to *Prioritized* applications. If the specified device is utilized by a *Fixed* application in *Monopoly* mode, then *Prioritized* applications cannot use the device.

To improve the effectiveness of this rule, we defined the following three non-exclusive options.

- **Priority Level**
  Prioritized applications can compete with each other for the same device. To give a specific device to a part of the competing applications, Torta can use *Priority Level* to order the applications' importance. However, we do not focus on creating an assignment algorithm using this value; testing such algorithms needs a number of device switchable applications.
- **Secondary Device**
  Prioritized applications with low priority levels may not be able to use the specified device. Still, these applications are important to the user. Therefore, this *Secondary Device* option allows the assignment of a device when the application cannot use the firstly specified device. *Priority Level* option also affects this option.
- **Forbidden Device**
  Forbidden Device option allows specifying devices that the system must not assign to a specific application. This option is useful when the application gets extremely slow with a specific device, or when a specific device brings only tiny performance improvement with huge power consumption.

The preference of processors assignment can be expressed like the following (using XML in the implementation);

```
Device: CPU
  Fixed: FileCompressor.exe
Device: iGPU
  Fixed Virtual Monopoly: VideoPlayer.exe
Device: dGPU
  Prioritized Level1: Simulator.exe
    Secondary: CPU
```

A method that simply assigns faster device to important applications does not work well. For example, if that the most important application A was assigned to the fastest device α, it is difficult for us to choose a device from device α or β for the second most important application B. If application B works faster by sharing device α with application A than monopolizing device β, we need a threshold to regulate the performance degradation of application A and the performance gain of application B. To decide this threshold, a system needs a linear scale of applications' importance like "application A is 3 times important than the application B." However, it is almost impossible since users can order applications by importance. Calculating the importance of applications is generally more difficult. Furthermore, if an application is designed for real-time processing like video playback, it does not need a device to achieve more than real-time interaction. Besides, we cannot give the generalized score for the video output of GPUs; only that the user prefers one over another. Consequently, existing pure priority models are not applicable.

3.4 Interruption on Devices Selection

To utilize a compute device via OpenCL APIs, applications perform the following steps based on the OpenCL specification:

(1) Acquire the list of devices to recognize the installed devices.
(2) Decide on the devices that can be used.
(3) Create a *context* that includes device(s) chosen in the previous step.
(4) Compile a program with the created context to utilize the devices.
(5) Create a *command queue* to actually send instructions and data by determining the device to use.
(6) Enqueue necessary transactions to the command queue, such as data copying or OpenCL kernels.

To redirect device assignment, Torta changes the value of the device ID when the application tries to assign a device by creating a context. We chose a method called *API Hook* to change the device ID when applications call the OpenCL APIs. In our implementation with Microsoft Windows 7, there are some choices available to achieve API hooking. We chose the method of "Wrapper DLL," which is the shell of the original DLL that implements all of the original entry points. To implement the same APIs of the original, we extracted the original APIs from the original DLL which was created by Khronos and included them in the OpenCL SDKs. Then, as show in **Fig. 5**, we injected some codes before and/or after calling the original APIs from the original DLL. Codes were injected according to the OpenCL Reference Pages [1] to retain compatibility of the APIs. Placing the implemented wrapper DLL with the original file name in the application installed directory always forces to applications to execute injected codes since applications load the DLLs in their installed directory prior to the DLLs installed in the system directory. This

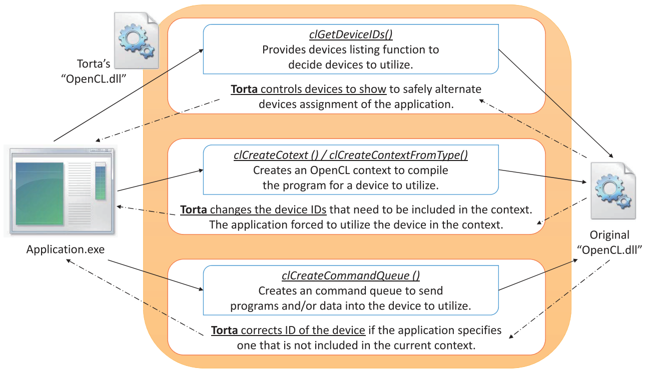

Torta's "OpenCL.dll"
clGetDeviceIDs()
Provides devices listing function to decide devices to utilize.
Torta controls devices to show to safely alternate devices assignment of the application.
clCreateCotext () / clCreateContextFromType()
Creates an OpenCL context to compile the program for a device to utilize.
Torta changes the device IDs that need to be included in the context. The application forced to utilize the device in the context.
Application.exe
Original "OpenCL.dll"
clCreateCommandQueue ()
Creates an command queue to send programs and/or data into the device to utilize.
Torta corrects ID of the device if the application specifies one that is not included in the current context.

Fig. 5 Torta's API hooking of original OpenCL.dll.

method has the advantage of stability brought by isolating each API hook to the applications. The detailed information of all API hooking methods and the comparison between methods are stated in our previous work [18].

To switch devices on each application, the RM needs to know all of the device IDs available on the target application. Because some applications try to list only GPUs, the AHM lists all available devices by using the *clGetDeviceIDs()* API and then sends the list to the RM. Then, the AHM waits until the applications try to assign a device. In our current implementation, the AHM asks the RM to decide on the device in order for the application to create a context by calling *clCreateContext()* or *clCreateContextFromType()*. Then, the RM decides on the device according to the current profile and answers the appropriate device. The AHM assigns a device by changing the device ID; it then sets the parameter *cl_device_id *devices* of *clCreateContext()* according to the RM's order. When applications call *clCreateContextFromType()*, AHM substitutes *clCreateContextFromType()* with *clCreateContext()* to create a context that includes only a device. After these steps, applications keeps working as usual except the assigned device.

In fact, it is possible to create a context with multiple devices that belong to the same *platform*; OpenCL platforms are vendor-provided drivers for OpenCL that include compilers for compute devices. Although OpenCL's ICD (Installable Client Driver) system allows multiple OpenCL platforms concurrently, the specification of OpenCL does not allow the creation of a context with devices that belong to the different platforms. In our previous work [18], we created a context with multiple devices to achieve devices switching after creating a context. However, Ref. [18] achieved device switching using only on a single platform with a few applications. Actually, the supported functions (e.g., OpenCL Extensions) of the devices are subtly different; creating a context with multiple devices limits the number of available functions supported by all devices in the context. Lack of functions does not only decrease the performance of applications, but also decreases applications' compatibility. Therefore, Torta always creates a context with only a single device. In addition, compilation of OpenCL code is slightly faster on a context with a single device than with multiple devices. Therefore, our current implementation creates contexts that include only a single device.

Because of the API hooking stated above, applications are forced to use the compute device which the resource manager of Torta specified instead of the one the application specified. Actually, the current implementation of Torta can *dynamically* switch computing devices at every context creation. Therefore, the frequency of opportunities to switch devices depends on the implementation of each application. More precisely, opportunities to switch devices are restricted to every context creation according to the OpenCL specification. Each abstraction layer of processors has different opportunities to switch devices dynamically.

3.5 Device Information Handling

Even using an abstraction layer like OpenCL, applications still need to collect detailed device information to optimize their performance. Some applications collect the information of all devices before assigning a device. On such applications, changing the device ID can cause incoherence of the device information. This inconsistent information not only causes performance degradation, but it can also cause unexpected behavior in the application. Therefore, Torta improves the applications' compatibility by using the following three methods.

Changing Device Type Information

Some applications show incomprehensible behavior when the information of device type (e.g., CPU, GPU) is inconsistent. For instance, one such application returns an error when a CPU is assigned to it even though the application specified a GPU device. Therefore, the AHM of Torta inquires the *camouflaged* device type from the RM when the application invoked an OpenCL API *clGetDeviceInfo()* with *cl_device_info param_name* as *CL_DEVICE_TYPE* to acquire information of the device type. After the RM answers the device type as CPU or GPU, the AHM changes the information of device type by setting *void *param_value* as *CL_DEVICE_TYPE_CPU* or *CL_DEVICE_TYPE_GPU*.

Remapping Device IDs

To prevent making inconsistent device information, Torta remaps device IDs after assigning a device by creating a context. When one of the application's threads finishes creating a context, the AHM remaps the device ID to the one which is actually assigned when the application calls the information acquiring API called *clGetDeviceInfo()*. Therefore, acquired information from the device matches the information of the one that was actually assigned. The AHM keeps remapping device IDs during the context until the context is released by *clReleaseContext()*.

Correcting Information After Assigning a Device

Each device has different limitation on the maximum working array size in order to execute parallelized tasks. Although applications usually confirm this limitation before utilizing the device, this information also becomes inconsistent by devices switching. If the array size exceeds the limitation of the assigned device, the application will crash or stop with errors. Therefore, Torta changes such information retained by applications after switching devices. In our current implementation, Torta changes a parameter of *clEnqueueNDRangeKernel()*, *size_t local_work_size*. In the OpenCL specification, *size_t global_work_size* must be divisible by *size_t local_work_size*. This value is rounded to an integer by the AHM.

3.6 Control of the Number of Recognized Devices

Some applications utilize multiple devices concurrently to improve performance. However, if an application utilizes all of the installed GPUs concurrently, other GPU-accelerated applications will suffer performance degradation. For example, when such applications are executed on a PC with two GPUs (an iGPU and a dGPU), both GPUs will be used. In some cases, Torta redirects GPU assignment to the other GPU (iGPU & dGPU to dGPU & dGPU). However, this causes meaningless overhead because of the *task distribution* on to the same GPU.

In fact, these applications decide on the concurrency by using the number of devices they recognized. Therefore, Torta limits the number of devices that are shown to such applications. Due to the OpenCL specification that a command queue can deal with only a compute device, such applications need to create and use multiple command queues at once in order to use multiple GPUs concurrently. Therefore, Torta can recognize that the application is implemented as using multiple GPUs concurrently. Reading loads of devices also helps Torta to recognize devices utilization by applications. Then, applications will try to utilize only a GPU because Torta shows only a single GPU. This method prevents performance degradation of other applications and the target application because of the *task distribution*. The limit on the number of devices can be any natural number less than the number of available devices.

Torta changes the parameter of *cl_uint num_entries* in *clGetDeviceIDs()* in order to limit the maximum number of devices on the list. Because *clGetDeviceIDs()* can list only devices in a platform, applications need to invoke this API as many times as the platforms installed on the PC. The AHM asks the RM the maximum number of devices for the application *when the application is launched* since most applications list available devices only when they are launched. Torta distributes remaining slots of devices to platforms in order to control the total number of devices shown.

On the other hand, Torta can increase the number of devices shown by changing *cl_device_type device_type* to *CL_DEVICE_TYPE_ALL* from *CL_DEVICE_TYPE_CPU* or *CL_DEVICE_TYPE_GPU*. This not only improves the applications' compatibility, but is also helps in the increase of the concurrency of the applications' tasks distribution.

4. Evaluation

To demonstrate the deployability of our method, we evaluated the compatibility with various hardware and software, and the performance effects of Torta. Also, we compared Torta with other existing approaches.

4.1 Evaluation Environment

We chose the OpenCL platforms shown in **Table 1** on Windows 7 x64 and x86 for our experiments.

As for hardware configurations, we prepared the PCs shown in **Table 2**.

As target applications for the switching devices, we chose the following applications:

Table 1 OpenCL platforms used in our experiments.

Target Processors	OpenCL Platform
Intel CPUs	Intel SDK for OpenCL* Applications 2012 [20]
Intel iGPUs	Intel HD Graphics Driver 8.15.10.2761 [19]
AMD CPUs (incl. Intel CPUs)	AMD APP SDK ver. 2.7 [4]
AMD GPUs (incl. AMD/Intel CPUs)	Catalyst Software Suite with .NET 4 Support 12.10 [5] (Catalyst Software Suite 12.6 on PC(A) 64 bit)
nVidia GPUs	GeForce 306.97 Driver [32]

Benchmark/Simulation Applications
- DirectComputeBenchmark ver.0.45b [31]
- N-Queen Solver for OpenCL [10] (recompiled with Visual Studio 2010 SP1)
- ratGPU ver.0.5.5 [35]
- LuxMark ver.2.0 32 bit [27]

Utilities
- WinZIP ver.17.0 (10283) 32 bit [38]
- FLACCL (in CUETools) ver.2.1.4 [12]

Image Processing Applications
- Adobe Photoshop CS6 EXTENDED ver.13.0.1 x32 [3]
- GIMP ver.2.8.2 x86 [37]
- PhotoMonkee ver.0.56b [39]

Video Processing Applications
- vReveal 3 ver.3.2.0.13029 [30]
- x264 with OpenCL lookahead patch rev.2230+696_tMod [2]

4.2 Switching Compatibility

To evaluate the compatibility of Torta, we evaluated device switching of the applications stated in Section 4.1. **Table 3** shows the results of these experiments. The denominators and numerators show the number for all PCs and switching respectively.

As a result, Torta achieved devices switching on 87.5% of all applications. Torta also achieved devices switching through any combination of devices with 6 out of 8 applications. Even with 1 out of 8 applications, Torta achieved device switching through any combination of devices on 40% of tested PCs.

PCs used for our experiment include ones with complicated hardware configurations, such as ones with all three OpenCL platforms (A, C, E) and ones with multiple GPUs (A, B, D, E). Thus, the experimental result demonstrates that Torta can achieve device switching on most modern PCs.

Experimented applications can be classified into the following four types:

Completely Switchable

Torta is completely compatible with DirectComputeBenchmark, N-Queen Solver for OpenCL, ratGPU, LuxMark, GIMP and PhotoMonkee. Torta could switch devices except those which are originally incompatible with applications. Intel HD Graphics 4000 on ratGPU and AMD's GPUs and CPUs on PhotoMonkee were originally incompatible. Because GIMP's installer does not allow installing x86 binary on x64 platform, we experimented GIMP on PC(A) with the x86 OS only.

Partially Switchable

Torta could switch devices for most of the available devices on

Table 2 The result of compute device switching compatibility on each application.

	Type	CPU	GPU1	GPU2	GPU3
(A)	Desktop	Intel Core i7-3770K	HD Graphics 4000 (iGPU)	AMD RADEON HD 7770 GHz Edition	nVidia GeForce GTX 660
(B)	Desktop	AMD FX-8150	RADEON HD 6970	AMD RADEON HD 6970	GeForce GTX 550 Ti
(C)	Laptop	Intel Core i7-2760QM	(HD Graphics 3000 iGPU)	GeForce GT 555M	none
(D)	Laptop	AMD A6-3400M	RADEON HD 6520G (iGPU)	AMD RADEON HD 6650M	none
(E)	Ultrabook	Intel Core-i7 3517U	HD Graphics 4000 (iGPU)	nVidia GeForce GT620M	none

Table 3 Compute devices switching compatibility on real applications.

Application Name	Perfectly Compatible PCs	Partially Compatible PCs	Succeed Assignment Combinations	Succeed Combination Ratio
DirectComputeBenchmark	5/5	0/5	58/58	100%
N-Queen Solver for OpenCL	5/5	0/5	19/19	100%
ratGPU	5/5	0/5	17/17	100%
LuxMark	5/5	0/5	19/19	100%
FLACCL	0/5	2/5	4/19	21.1%
GIMP	1/1	0/1	5/5	100%
PhotoMonkee	4/4	0/4	11/11	100%
x264 with OpenCL	2/5	3/5	12/15	80.0%
vReveal 3	N/A	N/A	N/A	N/A
Photoshop CS6	N/A	N/A	N/A	N/A
WinZIP	N/A	N/A	N/A	N/A

x264 with OpenCL lookahead patch. It could not switch devices into CPUs via the AMD platform on PCs that have RADEON GPU(s). When we tried to switch the device into CPUs via the AMD platform, the application showed *CL_INVALID_ARG_SIZE* error on *clSetKernelArg()* due to inconsistent information about the devices.

Not Switchable

Torta could not switch devices except GPUs in the first platform on FLACCL. Because FLACCL is a .NET application, it calls the OpenCL APIs via a .NET class library: OpenCLNet.dll. This specification limits Torta's switching capability. Switching into a device on another platform caused *KeyNotFoundException*. On the other hand, switching into CPUs on the same platform caused *INVALID_WORK_GROUP_SIZE* error on *clEnqueueNDRangeKernel()* due to inconsistent devices' information. However, we could not apply the information correcting methods stated in Section 3.5 due to the restrictions of .NET.

Excluded from Our Experiment

Photoshop CS6, WinZIP and vReveal 3 were excluded from our experiments because they have fatal problems during our experiments with Torta. Photoshop and WinZIP do not load OpenCL.dll. However, our current implementation only targets applications that load OpenCL.dll to call OpenCL APIs. On the other hand, vReveal 3 could not launch the application with Torta's AHM (*OpenCL.dll*). Because of the application's compilation options and the difference between linked MSVCRT versions, it crashed with MSVCRT R6030 [28] error. Because it is a problem with the OS, we excluded vReveal 3.

4.3 Overhead

To evaluate the overhead caused by the cost of device switching, we measured the execution time of the *N-Queen Solver for OpenCL* with Torta and without Torta. **Figure 6** shows the performance difference between these two conditions.

We experimented with PC(A) and chose RADEON HD 7770 GHz Edition as the target compute device. To measure the execution time, we used the "Measure-Command" *cmdlet* of PowerShell. `Measure-Command{.\nqueen\_cl.exe -platform 0 N}`

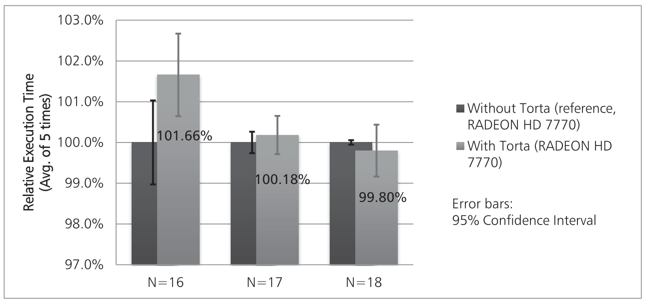


Fig. 6 Overhead of Torta on N-Queen Solver for OpenCL.

To calculate the ratio of the original execution time to Torta's overheads, we changed the *N* (board size) value of the command line option, from 16 to 18. To minimize the impact of other factors, we used a RAMDisk and calculated the average over 5 replicate experiments.

As a result, Torta causes 1.7% of the overhead for handling devices' information, switching device IDs and other API wrapping when *N=16*. However, the overhead kept decreasing according to the board size. It became 0.2% when *N=17*. Moreover, when *N=17* or *N=18*, it seems there is no significant difference according to a 95% confidence interval between the execution time of the two conditions. As expected, hooking APIs results in a small overhead. Thus, Torta cannot shorten the execution time. However, real applications utilize GPUs as GPGPU only for heavy processing like *N=18*. Torta's overheads are negligible in most cases.

4.4 Performance Degradation

Torta cannot always avoid creating inconsistent devices' information in spite of its information correction functions. However, some applications optimize their processing adaptively to the specifications of the device, such as the number of compute

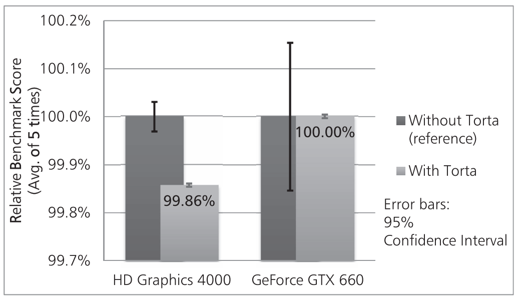


Fig. 7 Result of performance degradation on DirectComputeBenchmark.

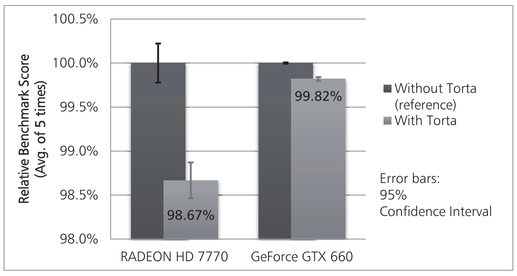


Fig. 8 Result of performance degradation on ratGPU.

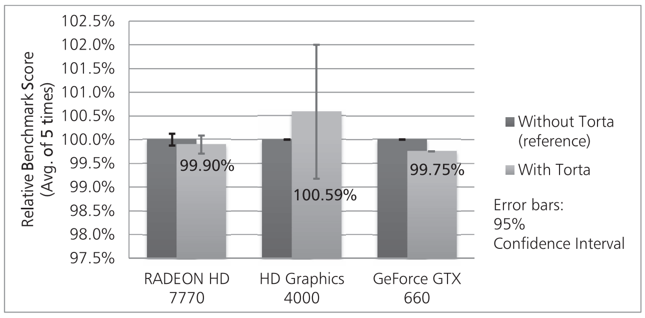


Fig. 9 Result of performance degradation on LuxMark.

units and the hierarchy of device memory. Therefore, we compared the scores of three benchmark applications under the following conditions: switching the applications' devices with Torta and executing applications without Torta. **Figures 7**, **8** and **9** show the results of the relative benchmark scores.

We experimented with PC(A) and used a RAMDisk, just like the previous experiment. As a result, Torta caused 1.3% of performance penalty in the worst case and caused only 0.20% of performance penalty on average. Rather, Torta improved benchmark scores on DirectComputeBenchmark. Torta does not only make possible for the application to utilize RADEON HD 7770 GHz Edition which is not originally recognized by DirectComputeBenchmark, but it also makes it possible to utilize two devices concurrently. Thus, utilizing multiple GPUs significantly increases the score of the benchmark.

4.5 Performance Protection for an Important Application

Since users' preferences vary, we cannot easily evaluate the degradation of the user experience objectively. Therefore, we measured the FPS of video playback by executing other GPU-intensive applications with and without Torta. We assumed the context is that the user is watching a video. The user prefers real-time video playback and he/she does not allow even a few frame droppings. **Figure 10** shows FPS of the video playback of each

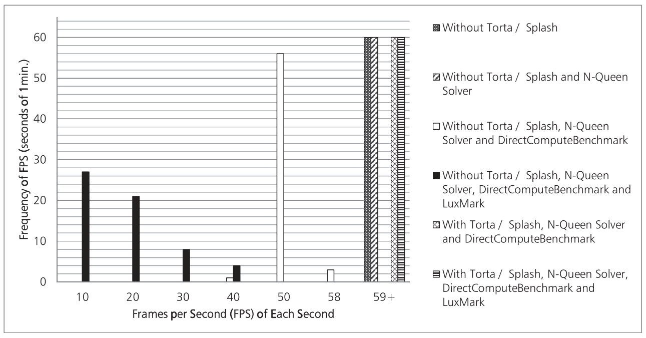


Fig. 10 FPS of video playback during multiple GPU-accelerated application running.

second under each condition. For example, 30 along the X-axis means that the FPS is greater than or equal to 30 and less than 40. The bar on 30 means that there are eight seconds that matches the range of FPS stated above.

We measured the FPS of Splash PRO EX utilizing HD Graphics 4000 on PC(A). Other applications that were executed were the N-Queen Solver for OpenCL, DirectComputeBenchmark and LuxMark. Although DirectComputeBenchmark and LuxMark select HD Graphics 4000 by default, N-Queen Solver for OpenCL does not. To keep the FPS of the video playback, Torta redirected the device of additional applications to another GPU called GeForce GTX 660. As a result, with Torta, the FPS of the video playback stays at 60 FPS (real-time) even when all three of the additional applications were executed. On the other hand, without Torta, the FPS of video playback decreased to about 46 on average when two additional applications were executed. Also, the FPS decreased to about 14 on average when three additional applications were executed. This result demonstrates that Torta can prevent degradation of the user experience at least under specific conditions. Torta achieved realtime video playback as in existing studies [24], [34] with more restricted conditions such as binary compatibility with existing applications.

4.6 Effectiveness of Device Assignment According to Users' Contexts and Preferences

To evaluate the effectiveness of our concept, we compared two device assignment models; an existing priority based model and Torta's model.

We assumed that a video playback application (Task A) and another GPU application (Task B) work concurrently on a PC with two different GPUs. GPU0 is two times faster than GPU1. On the current user's context, the user concentrates on watching the video. Therefore, the user prefers video playback on GPU1 because of the video quality. The GPU load caused by video playback is moderate; 30% on GPU0 and 60% on GPU1.

Table 4 shows the assigned device of video playback (Task A) and another GPU application (Task B). When using a priority model based on the devices' computation resource, giving higher priority to Task A forces device assignment to Task A. Task A is always executed on the faster device, GPU0. The device assignment caused by low priority is better than one with high priority. In this case, prioritizing tasks does not make sense at all. To make devices assignment on priority models as correct as Torta's

Table 4 Comparison of devices assignment models.

Load of GPU0 (which is faster than GPU1) caused when Task B is executed on the GPU0.	**0%**		**25%**		**50%**		**75%**	
	Assigned Device							
	Task A	**Task B**	**Task A**	**Task B**	**Task A**	**Task B**	**Task A**	**Task B**
Without Torta (Target's priority is set as higher than another.)	GPU0	N/A	GPU0	GPU0	GPU0	GPU1	GPU0	GPU1
Without Torta (Target's priority is set as lower than another.)	GPU0	N/A	GPU1	GPU0	GPU1	GPU0	GPU1	GPU0
With Torta (Target has right to monopoly GPU1)	GPU1	N/A	GPU1	GPU0	GPU1	GPU0	GPU1	GPU0

Table 5 Approaches for GPU resource management or tasks distribution on heterogeneous processors.

	OS Modification [25], [34]	**Driver Modification [24]**	**Original Libraries [8], [13], [22]**	**Torta**
Adapts with existing GPGPU applications	No	Yes	No	Yes (OpenCL)
Adapts with other existing GPU-accelerated applications	No	Yes	No	Depends
Distributes tasks between CPUs and GPUs	Depends	No	Yes	Yes
Adapts with new GPUs cooperation technologies	Difficult	Quickly	Easily	Easily
Overheads of resource management	Small	Small	Large	Large
Granularity of tasks distribution	Depends	Small	Small	Large
Supports GPU resource management	Yes	Yes	No	No
Needs modification of applications	Yes	No	Yes	No
Cost of installation	Heavy	Small	Little	Little
Short-term Deployability	Low	Depends on GPU vendors	None	High

assignment, we need to combine users' preferences into the priority models. Interestingly, GPU1 should be assigned to Task A even when the context is that the user does not concentrates on watching the video. If Task A is not important to the user, Task A should work on a device with smaller computing resource. However, assigning GPU1 statically to Task A is a bad idea. In another context, Task C may need to monopolize GPU1 to satisfy the user.

This comparison demonstrates that existing priority models are not suitable for assigning a different device causes a non-linear effect. Reading processors' load does not help to improve the user experience under such non-linear conditions. The comparison also shows that applying users' contexts and preferences gives better assignment performance than priority models under some conditions. Our method helps to improve the user experience by condition based assignment.

4.7 Approaches Comparison

Our experimental results demonstrate that our approach is deployable and effective. Thus, we compared our approach with other existing approaches.

Table 5 shows the difference between the OS modification approach, the driver modification approach, the original library approach and our approach.

When compared to the OS modification approaches, Torta has an advantage in terms of practicality. Our experiments demonstrate its outstanding deployability. However, Torta has a disadvantage of performance overheads and granularity of resource management. It only distributes tasks between compute devices; it does not manage the resource of GPUs. Also, the OpenCL specification limits the granularity of task distribution. When compared to the driver modification approaches, Torta has the advantage of tasks distribution. As stated in Section 2.5, the drivers modification methods only optimize the resource management of the GPUs provided by the specific GPU vendors. Therefore, driver modification approaches cannot distribute tasks between GPUs and CPUs. However, Torta has the disadvantage of overheads, the number of target applications and the granularity of resource management. Thus, driver modification approaches and our approach are complementary rather than competitive. When compared to the original libraries approaches, Torta has one notable advantage: compatibility with practical applications. However, Torta has disadvantage in terms of the granularity of task distribution because the granularity of task distribution and binary compatibility are trade-offs.

Although this comparison shows relative advantages of Torta, this paper does not focus on showing the supremacy of middleware-based solutions. To apply our concept to PC environments in the near future, creating middleware fulfills requirements well.

5. Related Work

Task distributing among heterogeneous processors

Harmony [13] and other studies [8], [22], [26] propose methods for scheduling tasks between heterogeneous processors. Although they achieve improvement for existing metrics such as performance and power consumption, their schedulers are not enough for preventing poor user experience on PCs.

Extending existing task distributing frameworks

Hybrid OpenCL [7] and others [6], [33] extend OpenCL to allow applications to utilize compute devices on remote hosts. Although our current implementation only assumes local compute devices, adapting these works to ours will allow some tasks to be offloaded to the cloud.

Resource management on GPUs

TimeGraph [24] and GERM [9] propose driver based systems for achieving fair resource allocation on GPUs between multiple applications. Since Torta lacks fine-grained GPU resource management, it works more effectively with them than works alone.

Gdev [25] and PTask [34] are OS extensions for GPU resource management. Although they are effective, modification of OSes need a long time. Scheduling methods proposed in Ref. [23] also provide isolation between competitive graphics tasks. However, it does not deal with minor tasks of GPUs such as video decoding and encoding.

GPU virtualization

GViM [16] and other studies [14], [36] virtualize a GPU on a hypervisor to share GPUs between multiple VMs. However, their works deal with only limited GPU utilization purposes; they are not easily applicable to PC platforms.

GPU acceleration

Many acceleration studies are applicable to PC environments. Some studies [17] propose offloading methods for encryption. Meanwhile, Ref. [11] accelerates video encoding using GPUs. Many studies [15], [21] use GPUs for accelerating image analyses and tracking. Increasing the number of purposes of GPUs increases the necessity of tasks distribution between CPUs and GPUs.

6. Conclusion

This paper has presented a centralized processors assignment middleware called Torta that achieves heterogeneous processors assignment on real applications for PCs. It realizes switching compute devices (almost the same as processors) on real applications by enabling to hook to OpenCL APIs without any modification of applications. In addition, Torta uses a condition based model to assign compute devices instead of existing priority models. It allows the appropriate device assignment to adapt to users' contests and users' preferences.

Our experiment using eight practical applications has shown that Torta achieves binary-compatible processors switching with only an average performance penalty of 0.2%. In one particular case where a video playback application is executed with other three GPU-intensive applications, our mechanism enables users to enjoy the video playback with 60 FPS while the FPS decreases to 14 without the mechanism. The results demonstrate the feasibility and effectiveness of our method.

As future work, we plan to design a scheme for expressing users' contexts and preferences, and a user friendly GUI for profiles creation that abstracts raw processors' assignment rules. In addition, we plan to expand our method into other resource managements such as priorities of application processes. Our goal is to adapt users' contexts and preferences to resource management for user interactive devices to improve the overall user experience.

References

[1] OpenCL 1.2 Specification, available from ⟨http://www.khronos.org/registry/cl/specs/opencl-1.2.pdf⟩.

[2] x264 rev2230+696 tMod/avs4x264mod 0.9.0, available from ⟨http://astrataro.wordpress.com/2012/11/11/x264-rev2230696-tmod-avs4x264mod-0-9-0/⟩.

[3] Adobe Systems Incorporated: Adobe Photoshop CS6 Extended, available from ⟨http://www.adobe.com/products/photoshopextended.html⟩.

[4] Advanced Micro Devices, Inc.: Accelerated Parallel Processing (APP) SDK, available from ⟨http://developer.amd.com/tools/heterogeneous-computing/amd-accelerated-parallel-processing-app-sdk/downloads/download-archive/⟩.

[5] Advanced Micro Devices, Inc.: AMD Catalyst Display Driver for Windows Vista/Windows 7, available from ⟨http://support.amd.com/us/gpudownload/windows/Pages/radeonaiw_vista64.aspx⟩.

[6] Aoki, R., Oikawa, S., Tsuchiyama, R. and Nakamura, T.: Improving Hybrid OpenCL Performance by High Speed Network, *Proc. ICNC*, pp.262–263 (2010).

[7] Aoki, R., Oikawa, S., Tsuchiyama, R. and Nakamura, T.: Hybrid OpenCL: Connecting Different OpenCL Implementations over Network, *Proc. IEEE CIT*, pp.2729–2735 (2010).

[8] Augonnet, C., Thibault, S., Namyst, R. and Wacrenier, P.-A.: StarPU: a unified platform for task scheduling on heterogeneous multicore architectures, *Proc. Concurrency and Computation: Practice and Experience*, Vol.23, No.2, pp.187–198 (2011).

[9] Bautin, M., Dwarakinath, A. and Chiueh, T.: Graphic engine resource management, *Proc. MMCN*, Vol.6818 (2008).

[10] N-Queen Solver for OpenCL, available from ⟨http://forum.beyond3d.com/showthread.php?t=56105⟩.

[11] Chen, W.-N. and Hang, H.-M.: H.264/AVC motion estimation implmentation on Compute Unified Device Architecture (CUDA), *Proc. ICME*, pp.697–700 (2008).

[12] FLACCL, available from ⟨http://www.cuetools.net/wiki/FLACCL⟩.

[13] Diamos, G.F. and Yalamanchili, S.: Harmony: An execution model and runtime for heterogeneous many core systems, *Proc. ACM HPDC*, pp.197–200 (2008).

[14] Dowty, M. and Sugerman, J.: GPU virtualization on VMware's hosted I/O architecture, *Proc. ACM SIGOPS Operating Systems Review*, Vol.43, No.3, pp.73–82 (2009).

[15] Fung, J. and Mann, S.: OpenVIDIA: Parallel GPU computer vision, *Proc. ACM Multimedia*, pp.849–852 (2005).

[16] Gupta, V., Gavrilovska, A., Schwan, K., Kharche, H., Tolia, N., Talwar, V. and Ranganathan, P.: GViM: GPU-accelerated virtual machines, *Proc. HPCVirt 2009*, pp.17–24 (2009).

[17] Harrison, O. and Waldron, J.: AES Encryption Implementation and Analysis on Commodity Graphics Processing Units, *Proc. CHES*, Paillier, P. and Verbauwhede, I. (Eds.), Vol.4727, pp.209–226 (2007).

[18] Horikawa, T., Honda, M., Nakazawa, J., Takashio, K. and Tokuda, H.: PACUE: Processor Allocator Considering User Experience, *Proc. Euro-Par 2011: Parallel Processing Workshops*, pp.335–344 (2012).

[19] Intel Corporation: Intel HD Graphics Driver for Windows* 7, available from ⟨http://downloadcenter.intel.com/Detail_Desc.aspx?lang=eng&amp;changeLang=true&DwnldId=21476⟩.

[20] Intel Corporation: Intel SDK for OpenCL* Applications 2012, available from ⟨http://software.intel.com/en-us/vcsource/tools/opencl-sdk⟩.

[21] Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A. and Fitzgibbon, A.: KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera, *Proc. ACM UIST*, pp.559–568 (2011).

[22] Jiménez, V.J., Vilanova, L., Gelado, I., Gil, M., Fursin, G. and Navarro, N.: Predictive Runtime Code Scheduling for Heterogeneous Architectures, *Proc. HiPEAC*, pp.19–33 (2009).

[23] Kato, S., Lakshmanan, K., Ishikawa, Y. and Rajkumar, R.: Resource Sharing in GPU-Accelerated Windowing Systems, *Proc. IEEE RTAS*, pp.191–200 (2011).

[24] Kato, S., Lakshmanan, K., Rajkumar, R. and Ishikawa, Y.: TimeGraph: GPU Scheduling for Real-Time Multi-Tasking Environments, *Proc. USENIX ATC*, pp.17–30 (2011).

[25] Kato, S., McThrow, M., Maltzahn, C. and Brandt, S.: Gdev: First-Class GPU Resource Management in the Operating System, *Proc. USENIX ATC*, pp.37–48 (2012).

[26] Luk, C.-K., Hong, S. and Kim, H.: Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping, *Proc. IEEE ACM MICRO*, pp.45–55 (2009).

[27] LuxMark, available from ⟨http://www.luxrender.net/wiki/LuxMark⟩.

[28] Microsoft Corporation: C Run-Time Error R6030, available from ⟨http://msdn.microsoft.com/en-us/library/9ecfyw6c.aspx⟩.

[29] Mirillis Ltd.: Splash PRO EX, available from ⟨http://mirillis.com/en/products/splashexport.html⟩.

[30] MotionDSP Inc.: vReveal Video Enhancement Software, available from ⟨http://www.vreveal.com/⟩.

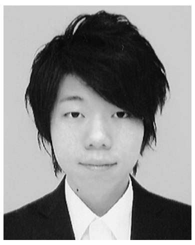
[31] NGOHQ.com: DirectCompute & OpenCL Benchmark, available from ⟨http://www.ngohq.com/graphic-cards/16920-directcompute-and-opencl-benchmark.html⟩.

[32] NVIDIA Corporation: NVIDIA DRIVERS 306.97WHQL, available from ⟨http://www.nvidia.com/object/win8-win7-winvista-64bit-306.97-whql-driver.html⟩.

[33] Ozaydin, R. and Altilar, D.: OpenCL Remote: Extending OpenCL Platform Model to Network Scale, *Proc. HHPCC-ICESS*, pp.830–835 (2012).

[34] Rossbach, C.J., Currey, J., Silberstein, M., Ray, B. and Witchel, E.:

PTask: operating system abstractions to manage GPUs as compute devices, *ACM SOSP*, pp.233–248 (2011).
[35] Santiago Orgaz: ratGPU, available from ⟨http://www.ratgpu.com/⟩.
[36] Shi, L., Chen, H., Sun, J. and Li, K.: vCUDA: GPU-Accelerated High-Performance Computing in Virtual Machines, *IEEE Trans. Comput.*, Vol.61, No.6, pp.804–816 (2012).
[37] The GIMP Team: GIMP, available from ⟨http://gimp-win.sourceforge.net/⟩.
[38] WinZip Computing: WinZip, available from ⟨http://www.winzip.com/win/en/index.htm⟩.
[39] Zimventures, LLC: PhotoMonkee, available from ⟨http://photomonkee.com/⟩.



Tetsuro Horikawa received his BIS (2011) and ME (2013) from Keio University. His research interests include high performance computing, context-aware services, ubiquitous computing systems and audio digital signal processing.

Jin Nakazawa is an Associate Professor of the Faculty of Environment and Information Studies, Keio University, Japan. He obtained his B.S. (1998), M.S. (2000), and Ph.D. (2003) from Keio University. His research interests include dependable systems, dependability assurance, ubiquitous computing systems, sensor networks, and distributed middleware systems. He is a member of IEEE, ACM, IPSJ, and IEICE.

Kazunori Takashio is an Associate Professor in the Faculty of Environment and Information Studies at Keio University, Japan. He received his Ph.D. in computer science from Keio University. He specializes in mobile and ubiquitous computing. His research interests include real-time distributed systems, mobile agents, context-aware services and applications, and privacy protection in ubiquitous computing environment. He is a member of ACM, IEEE, IPSJ and JSSST. He was a Visiting Researcher of Institute for Cognitive Systems (ICS), Technische Universitaet Muenchen (TUM) in 2012–2013.

Hideyuki Tokuda is the Dean of the Graduate School of Media and Governance and a Professor of the Faculty of Environment and Information Studies, Keio University, Japan. He obtained his B.S. (1975), M.S. (1977) from Keio University and Ph.D. (Computer Science) (1983) from University of Waterloo, Canada, respectively. His research interests include ubiquitous computing systems, decentralized autonomous systems, embedded systems, sensor networks, and smart spaces. He is a corresponding member of Science Council of Japan, IPSJ Fellow, JSSST Fellow, and a member of IEEE, ACM, IPSJ, IEICE, and JSSST.