Master's Thesis (Academic Year 2012)

Torta:
Extending applications' capability to select heterogeneous computing resources from CPUs and GPUs

adaptive to the PC usage

Keio University
Graduate School of Media and Governance

Tetsuro Horikawa

Tokuda Laboratory

January 2013

Abstract Of Master's Thesis Academic Year 2012

Torta: Extending applications' capability to select heterogeneous computing resources from CPUs and GPUs adaptive to the PC usage

Summary

In these days, many PC applications utilize a GPU as an accelerator to improve their performance because recent GPUs are powerful and flexible. However, heavy GPU utilization by an application easily interferes other applications' performance because operating systems treat GPUs as IO devices. In fact, our preliminary experiment shows that the frames per second(FPS) of a video playback decreased into 35 from 60 when another application utilized the same GPU concurrently. To address such problems, we need to switch processors assignment adaptive to both the PC usage and the user's preference. To prevent degradation of the user experience, legacy load model of resource management is not appropriate. Although some users expect video playback without frame dropping, other users may allow frame dropping when they do not pay attention a lot to the video. Besides, we also need to switch processors on real applications without modifying them because existing applications are the property of PC platforms. In this thesis, we proposed Torta that enables processors switching on existing real applications. Torta manages processors assignment centrally adaptive to profiles that contain the user's preference. According to the centralized decision, Torta switches processors assignment using API hooking to achieve binary compatibility. Our experimental results show the deployability of our system; Torta achieved processors switching on 87.5% of all the tested applications only with 0.20% of performance penalty in the average. The results also show that Torta prevented degradation of the user experience; it kept the frame rate of the video playback as real-time(60FPS) even when additional applications that are three benchmark applications and a scientific application tried to utilize a GPU concurrently.

Keywords

heterogeneous processors, resource management, GPU, GPGPU, OpenCL, user experience, binary compatibility

Graduate School of Media and Governance
Keio University

Tetsuro Horikawa

修士論文　2012 年度（平成 24 年度）

Torta:
PC の利用状況に応じて
CPU・GPU 間で計算リソースを選択可能にする
アプリケーション拡張機構

論文要旨

近年，GPU の性能向上や利用手法の増加により，多くの PC 向けアプリケーションがパフォーマンス向上のために GPU を用いる傾向がある．しかし，OS は GPU を IO デバイスとして管理しているため，あるアプリケーションによって GPU に高い負荷がかかると，他のアプリケーションを巻き込み，著しいパフォーマンスの低下を招くことがある．動画再生による事前実験では，1 つのアプリケーションによる GPU 負荷によって，フレームレートが 60FPS(frames per second) から 35FPS にまで低下する状況が見られた．このような問題を解決するためには，プロセッサの割り当てを PC の利用状況およびユーザーの好み双方に基づいて行う，新たな割り当てモデルが必要となる．例えば，ユーザーによって動画のコマ落ちを許容出来るか否かが異なるなど，適切なプロセッサ割り当ては一律に定義出来ない．そのため，プロセッサの使用率に基づく既存のリソース管理手法ではユーザーエクスペリエンスの低下は防止出来ない．一方，PC では既存アプリケーションに対する互換性が強く要求されるため，アプリケーションの改変を必要とする手法は現実的でない．以上を踏まえ，本論文では，上記の要件を満たし，PC 上の実アプリケーションにおいて使用プロセッサの切り替えを実現するシステム，Torta を提案した．Torta では，割り当てに関する情報を記述したプロファイルに基づき中央集権的にプロセッサの割り当てを行うことで，利用状況やユーザーの好みを反映したプロセッサ割り当てを実現している．また，API フックを用いることで，アプリケーションのバイナリ変更が不要なプロセッサ切り替えを実現している．評価実験において，Torta は対象とした実アプリケーションの 87.5%でプロセッサの切り替えを実現し，同時に切り替えに伴うパフォーマンスの低下を平均で 0.20%に抑え，実環境における高い適用可能性を証明した．また，評価実験において Torta は動画再生時に他に 3 つのベンチマークアプリケーションと 1 つの数学的探索アプリケーションが同時に GPU の利用を試みる状況においても，動画のコマ落ちを防止し，60FPS を維持出来た．これにより，Torta がユーザーエクスペリエンス低下防止に一定の効果を持つことが証明された．

キーワード

heterogeneous processors, resource management, GPU, GPGPU, OpenCL, user experience, binary compatibility

慶應義塾大学大学院 政策・メディア研究科

堀川 哲郎

Contents

List of Figures

List of Tables

List of Source Codes

Chapter 1

Introduction

1.1 Background

Recent Graphics Processing Units(GPUs) are getting remarkable due to their powerful parallel computation capability. Thus, many applications such as video encoders and photo editors were rewritten to utilize GPUs' computation capability optimally. Since GPUs are mainly used only for general computation in High Performance Computing(HPC) area, GPUs are used for a wider range of computing purposes on user interactive computers, such as PCs.

GPU utilization purpose in PCs are various however they can be usually classified into five purposes. The first purpose is 3D graphics computation, such as 3D games and 3D graphics based GUI shell (*e.g.* Windows Aero [61]). 3D graphics APIs such as DirectX [22] and OpenGL [52] are designed for these purposes. The second purpose is 2D graphics acceleration, such as font rendering acceleration used in modern web browsers. For example, DirectWire [21] in DirectX provides font rendering acceleration. The third purpose is general-purpose computing, called General-Purpose computing on GPU (GPGPU). On PCs, GPGPU is often used for video or picture editing applications and physics simulation computation in 3D game applications. These applications invoke GPGPU APIs such as CUDA [19]. The fourth purpose is video decoding acceleration. Video player applications invoke video decoding acceleration functions that are physically implemented in GPUs. Such video acceleration functions can play videos as superior image quality with lower CPU load than CPU playback. DirectX Video Acceleration(DXVA) [23] is one of sets of APIs that provides video decoding acceleration, which supports major video encoding formats like H.264. Additionally, some of GPUs have image quality improvement functions, such as noise reduction and up-scaling. The last purpose is video encoding acceleration. Some of GPUs have video encoding acceleration units on the die of GPU itself. As such case, we can transcode videos quickly by using the acceleration function via special APIs(*e.g.* Intel Quick Sync Video [40]).

Not only purposes of GPUs are increased, but PCs processors' heterogeneity is also increased.

Many recent CPUs such as 3rd generation Intel Core processors [1] have an internal GPU in their dies. However, most of the desktop PCs or the desktop-replace laptops also have a discrete GPU to increase its graphics performance. Though many recent PCs have two or more GPUs, these GPUs are not always utilized effectively. Though GPU cooperation technologies that are designed for internal and discrete GPUs [10] [92] [85] exist, they only work on limited variations of hardware configurations.

Because of complicated hardware combinations and various GPU uses, it is difficult to manage the resources of heterogeneous processors of PCs. Current operating systems do not manage the computing resources between CPUs and GPUs at all. Despite the lack of the resource management, GPUs on PCs are utilized as moderately effective. Only a few of applications are accelerated by the GPU concurrently today; these applications rarely compete each other on the same GPU.

There are two types of GPU accelerated applications for PCs. The first type is implemented as forcing to use the GPU. Therefore, these applications are not fully functional on PCs that do not include any compatible GPUs. On the other hand, the other type is implemented as choosing their computing resources from CPUs or GPUs. They choose the computing resource according to some reasons, such as stored settings. In usual, users can change the setting in the configuration menu on their user interface.

In our preliminary experiment, we found that applications' performance can significantly decrease when many applications work concurrently on the same GPU. For instance, frame per second(FPS) of the video playback decreased from 60 to about 35 when the GPU is heavily loaded(see fig.2.1). Although this is a rare case, users probably feel unhappy when such performance degradation occurs. The decrement of FPS does not only make the user a little unhappy, but it is also unacceptable to the user in some cases. In the future, the spread of GPU accelerated applications will cause such resource competition more frequently. Therefore, the demand of a resource management system designed for heterogeneous processors is certainly increasing.

1.2 Motivation

As stated in sec.1.1, applications' performance can decrease drastically if many applications work concurrently on the same GPU. The most influential reason is the method of GPUs accessing. Today's discrete GPUs are connected to CPUs via PCI Express bus. Adapting to this hardware structure, operating systems treat GPUs as IO devices. Accessing on GPUs needs to be serialized. Thus, it is not possible for other applications or the operating system to interrupt GPUs while another application transferring data between main memory(RAM) and video memory(VRAM). This means that large and/or frequent memory copying can easily cause resource competition between applications. As many studies [94] [83] [82] indicate, this

limitation makes difficult to utilize a GPU effectively as shared computing resource.

To loosen this limitation, some studies [82] propose a fine-grained task scheduling method for GPUs. On PCs, GPUs do not shoulder only general computing tasks, but shoulder also graphics rendering tasks. These rendering tasks need to be done in the real-time, thus prioritizing graphics rendering can prevent degrading of user experience caused by delay of the display refreshment.

However, methods for preventing decreased graphics rendering performance are not limited to fine-grained resource management on GPUs. Redirecting some tasks to CPUs can also prevent the resource competition on GPUs. For instance, if there are some applications that can select their computing resource from CPUs or GPUs, we can change the settings of these applications to redirect some tasks to CPUs. Reducing tasks planed to be assigned from the GPU will certainly increase the performance of other applications that utilize the GPU. Even if computing resource of GPUs is well managed, redirecting some tasks to the CPU can improve applications performance more. Therefore, distributing tasks between CPUs and GPUs is another crucial technology for future PC environments.

In fact, majority of GPU accelerated applications have also the CPU code internally for the same computation. These applications have to run on PCs that are not constructed with compatible GPUs, due to commercial reasons. In addition, applications with history inherit well optimized CPU codes from previous versions. Though applications that utilize GPUs for pure graphics(*e.g.* 3D games) always need GPUs, these are usually considered as *important* applications for the user. Moreover, these applications seldom run at the same time, due to their purpose. Thus, it is not a serious problem that we cannot switch computing resource on these applications.

On some applications, GPUs are utilized by generic frameworks designed for heterogeneous processors. There are some currently used abstraction layers for heterogeneous processors. For instance, Open Computing Language (OpenCL) [51] can abstract CPUs, GPUs and other accelerators as "compute devices". With OpenCL, applications can utilize heterogeneous processors through the same steps. C++ AMP [16] and OpenACC [58] are similar to OpenCL. Usually, these applications execute a part of their tasks on the processor specified by the processor ID. Therefore, changing the processor IDs will probably redirect the computation resource from a processor to another. It will be easier and more configurable to switch processors on these applications than applications that are not built with such abstraction layers.

As stated above, to make computing resources switchable on real applications is one of the most serious problems need to be solved. However, even computing resources are became switchable, it is quite difficult to *choose* an appropriate processor for each application. Prioritizing the tasks on the same processor causes linear difference between priority values. However, the performance is not easily predictable when tasks are distributed between heterogeneous processors. In addition, the performance cannot be considered as linear when switching processors

on PC environments. For instance, video decoding on GPUs and decoding on CPUs output a same frame with small differences. Decoding acceleration and post-processing are provided by GPUs hardware and their driver. Thus, they are not same as CPU code implemented in video playback applications. Though the difference between these images is small, some users notice the difference and prefer the one of their output. Because the performance metrics of PCs are complicated like this case, it is considerably difficult for a system to choose a processor to each application. To select a processor algorithmically, we need a certain reason that why the processor is more appropriate than another.

On user interactive computers, such as PCs, we believe that the greatest factor to decide a processor is the *user's preference*. For example, some users prefer watching a video with higher quality than executing other tasks quickly. On the other hand, some users even allow frame dropping to execute other tasks quickly. In addition, a user dynamically change his/her preference adaptive to the context. In some cases, the user focuses on achieving a higher score on a game than his/her friends. On another case, the user plays the same game only for killing time. To prevent degradation of the user experience, we need to assign computing resource adaptive 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.

To achieve computation resource assignment adaptive to the context and the user preference, we need a new scheduling mechanism that achieves converting the user preference into resource assignment algorithms. To make the user preference and their context as scheduler's input, a computer recognizable description format is needed. Also, a new user interface is needed, which allows a user to input his/her preference easily. This user interface also helps users to understand their preference by themselves. It is definitely difficult to make an algorithm that can maximize the user experience. However, we cannot develop new algorithms without any environment to apply them. Thus, at this time, we focus on creating a processor assigning system that is flexible with applying such new algorithms.

In this thesis, we designed and implemented a method for extending applications' capability to select heterogeneous computing resources, and a method for selecting processors adaptive to the user's preference.

1.3 Outline of Thesis

In this thesis, we state the introduction first, then state problems around processors switching and resource management for heterogeneous processors. Next, we state the design of our system, and then state the detailed implementation of our system. After that, we state the result of our experiments, then close with the conclusion.

Chapter 2

Problems Statement

As stated in sec.1.2, switching applications' computing resources is one of crucial technologies for future PC environment. In this chapter, we state detailed problems that need to be solved to achieve processors switching on PC environments.

2.1 Difficulty caused by complicated heterogeneity in PC environments

In these days, PCs are widely spread as everyone has his/her own PC. Therefore, there are huge variations of hardware configuration of PCs. Besides, variations of PC uses are also increasing due to the spread of PCs. In this section, we state the difficulty caused by such complicated heterogeneity.

2.1.1 Limitations of GPU accessibility

As stated in sec.1.2, current GPUs are accessed as IO devices. In addition, GPUs have their own memory addressing space. On most of GPUs, especially discrete GPUs, their memory is implemented in physically different from the main memory. This means that we need to copy and copy-back data between main memory(RAM) and video memory(VRAM) to execute tasks on GPUs. These copying processes are often heavy overhead; they can spoil the performance advantage of GPUs in some cases. Therefore, many studies [73] [69] [86] [80] figured out the threshold for selecting a processor that can compute faster. Data copies do not only cause overhead, but also cause serious competition between applications. Due to serialized accessing through PCI express bus, data copies block other applications that are trying to access GPUs. Therefore, even display refreshment can be interfered by copying large data. For instance, [94] shows that even the mouse polling rate can significantly decrease while the GPU is heavily loaded.

To confirm the influence of GPUs' heavy load, we measured the frame rate of the video

playback during another application made heavy load to the same GPU. We chose the video playback application as Splash PRO EX [57], which supports a motion interpolation technology. Splash can interpolate a lower frame rate of the original video to a higher rate, such as 24 frames per second(FPS) to 60FPS. Motion interpolation significantly improves the smoothness of video playback. However, it needs huge computing resource. Therefore, Splash uses a GPU to execute the interpolation processing. For our preliminary experiment, we chose an ultrabook PC, ASUS ZENBOOK Prime UX32VD [15]. It contains an CPU "Intel Core i7 3517U" [36] with an internal GPU "Intel HD Graphics 4000" [38], and a discrete GPU, "nVidia GeForce GT 620M" [28]. While Splash chooses a GPU which is connected to the display the window located, Splash selected HD Graphics 4000 on our experiment. HD Graphics 4000 is the primary graphics adapter of that PC. On this ultrabook, it needs about 45% of iGPU(HD Graphics 4000) computing resource to play a video in full screen mode with interpolated frame rate. Because the GPU load is below 50%, it seems possible to run another application that utilize the same GPU without dropping frames. However, when we started executing a benchmark program called DirectCompute Benchmark [20], the FPS of the video playback decreased drastically. Fig.2.1 shows the difference of FPS between two conditions; only the Splash was executed and both Splash and DirectComputeBenchmark were executed. As a result, the FPS decreased to about 35FPS from 60FPS.

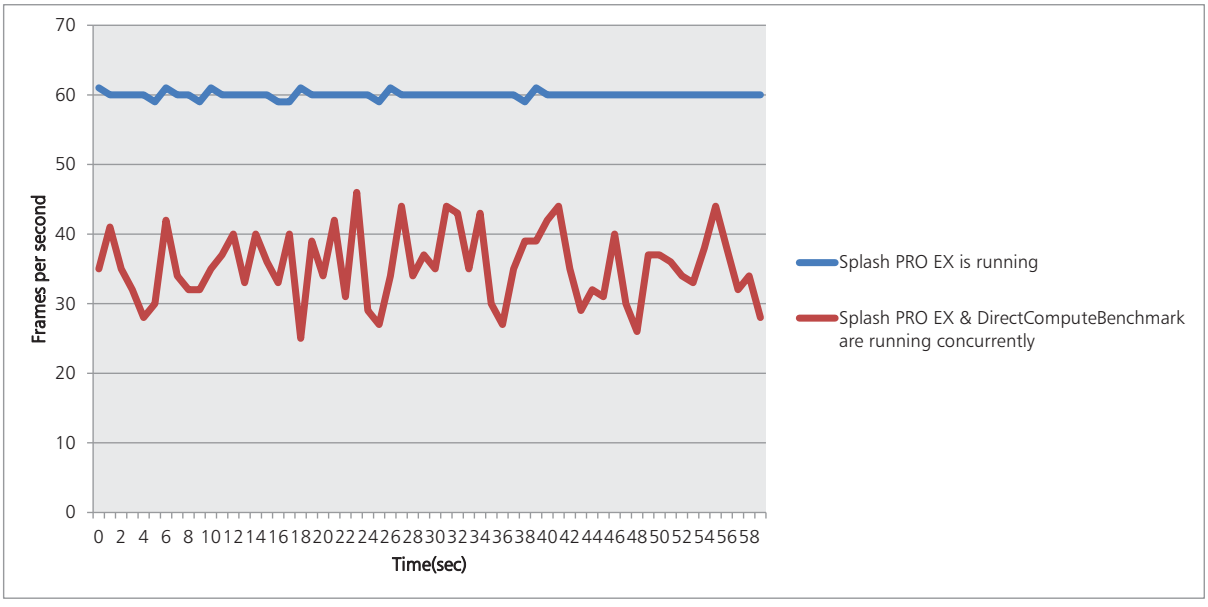

Figure 2.1: Result of Preliminary Experiment of Video Playback FPS Decrement during Another Application Utilize the Same GPU (1)

Through this result, we confirmed that an application with heavy GPU computation easily interferes another application. In cases like this, the user probably notices the frame dropping, and the user will feel uncomfortable with this situation. We defined these situations as

decrement of the user experience caused by the competition on the GPU resources.

As stated in sec.1.2, there are two methods applicable for avoiding such situations; to manage GPUs' resource in smaller granularity, or to distribute tasks between GPUs and CPUs.

For instance, a driver based system, TimeGraph [82] shows an event-driven model to schedule GPU commands in a responsive manner. It is very effective for protecting the performance of real-time applications while multi-tasking on a GPU. However, TimeGraph deals only with tasks on the GPUs; it cannot distribute tasks between GPUs and CPUs.

An OS extension system, PTask [94] is a set of APIs implemented as system calls that focuses on reducing unnecessary data copying between main memory and GPU's memory. Proposed system significantly improves the performance of applications like gestural interface. Also, it guaranteed the fairness of the division of the GPU resource while multiple applications tried to utilize the same GPU concurrently. However, it does not distribute tasks between GPUs and CPUs.

Another OS extension system, Gdev [83] is a total ecosystem for managing resources of GPUs. It provides a virtual memory function and a shared memory function for multiple GPU contexts. In addition, it can also virtualize GPUs to isolate GPU resource between users. However, it focuses on managing the resource of a single GPU; it does not distribute tasks between processors.

Indeed, implementing resource management system into OS is a straightforward idea, and it is easy to imagine that these systems achieve high performance. However, modification to OSes considerably needs a long time to be deployed into the released version. In addition, OS extensions may not easily cooperate with new GPU technologies [85] [92] [10] [31] [7] because these new technologies are implemented by tricky ways. Therefore, implementing a resource management system into OSes is not realistic as a short term solution, especially for PC environment.

On the other hand, implementing resource management system into drivers [82] seems to be more deployable. However, in usual, drivers are dependently provided by GPU vendors. Therefore, it is not possible to distribute tasks between GPUs manufactured by different vendors. Besides, drivers are usually proprietary, thus cooperativeness of GPUs vendors conditions the speed of deployment. Contrary, driver based systems work more effectively when used with a task distributing systems that are designed for heterogeneous processors. A driver based system and a task distributing system are complementary.

2.1.2 Integrated GPUs and Variable Clock Speed Technologies

Integrated GPUs in CPU packages are getting more and more powerful and flexible in these days. Such integrated GPUs(iGPUs) [38] [4] have the capability of GPGPU; thus we can use them as general computing resources. These iGPUs are special by two reasons if compared to

common discrete GPUs(dGPU).

One of the differences is the location of video RAM(VRAM). While discrete GPUs have own VRAM to store data, most of the iGPUs use a part of main memory as their VRAM. Thus, it is theoretically possible to use stored data without copying data between main memory and VRAM. Processor vendors are now starting to support this *zero-copy* feature [71] between these RAMs. As stated in [84], due to limitation of the number of processing units on iGPUs, iGPUs do not always overcome dGPUs. If zero-copy architecture spreads, we will need new algorithms for distributing tasks between processors. Many studies such as SPRAT [97] propose processors selecting algorithms that stand on the latency of data copies. When a PC has an iGPU with zero-copy function, an iGPU sometimes suits better than a CPU even if the data size is small. In addition, methods that reduce the frequency of memory copy between these RAMs [94] may have less effectiveness with iGPUs. However, there are many restrictions(*e.g.* maximum CPU=iGPU shared memory size) to use zero-copy function. Thus, for optimizing performance, new algorithms for distributing tasks between CPUs, iGPUs and dGPUs will be needed. However, we focus on switching processors on real applications in this thesis; we do not focus on creating such algorithms for tasks distribution in small blocks.

The other difference is the thermal design. Because iGPUs are implemented in the CPU die, they share the thermal design with CPU cores. To optimize performance adaptive to the limitation of the thermal design, CPUs varies their clock speed [6]. In addition, today's CPUs can set different clock speed to each core depending on the load of cores [42] [9]. However, current OS schdulers do not adapt to these technologies; scheduling and clock speed management work independently. Therefore, adapting clock speed management into scheduling system [95] should improve performance in some cases. However, these systems need to be implemented in OSes' scheduler themselves; thus we do not focus on this problem.

The thermal design does not affect only the clock speed of CPU cores but affects also clock speed of the iGPU. Thus, iGPUs cannot operate at its maximum frequency while CPU cores operate at high frequency. For instance, in our preliminary experiment(see sec.1.1), this limitation is one reason of the drastic FPS decrement. As Core i7-3517U [36] specification, the iGPU can operate at upto 1150MHz. However, due to the thermal design, the clock speed of iGPU were capped as 350MHz even when multiple applications used the iGPU concurrently. Hence, these variable clock speed technologies are one of considerable factors to select a processor for each application. However, the control of processors' clock speed is complicated. For instance, BIOSes often manage the limitation of the CPU's power consumption independently with OSes. Thus, we need to change the management model of processors power consumption. We plan to tackle this problem as future work, by extending our processors switching system.

However, according to our experience, assigning video playback tasks into iGPUs seems to be an appropriate idea. Video decoding acceleration functions are still powerful on weak GPUs such as iGPUs. In addition, because video decoding accelerators are implemented as hardware

logic, iGPUs can usually decode videos without increasing its operating frequency. In other words, assigning video decoding tasks into iGPUs hardly affect the clock speed of CPU cores. Therefore, we chose the iGPU as a video decoding accelerator on our experiments.

2.1.3 Variety of GPUs' Uses

Some studies [94] [83] [81] set their first target as PC platform; however, they do not consider seriously about the variety of GPUs uses. In reality, as stated in 1.1, GPUs are utilized by several kinds of uses on PC platform. Therefore, task scheduling algorithms designed for consumer GPUs should adapt to all of these GPU utilization methods.

To shoulder many types of tasks, consumer GPUs are not constructed only with graphics and general computing components, but they also include some special components. For instance, video decoding and encoding acceleration functions are implemented as hardware logic. Practically, applications often use both functions provided by hardware logic and functions provided by general computing components at once. For example, some video playback applications decode video data using hardware decoding accelerator in a GPU, then they apply some post-processing(*e.g.* noise reduction) to the video data using a GPU as general computing resource. In cases like this, an application can achieve full performance only when both graphics processing capability and video decoding capability is sufficiently usable by the application. Because GPUs have several different functions that are provided by different components, we cannot estimate GPUs' remaining processing capability by a linear variable such as legacy *load* model. In addition, one or more GPUs shoulder rendering tasks to show user interface. Decrement of the screen refresh rate can easily degrade user experience; users feel the system is *freezing*. Consequently, a processors assignment system need a new method to select a processor, alternative to "load" based algorithms.

2.1.4 Hardware Combinations and Multi GPUs Environments

On PC platforms, combinations of a CPU and GPUs are quite various. PCs have a socket of CPU and one or a few GPUs. The number of GPUs can even reach four or five on enthusiast class PCs. From tablet PCs through mobile laptop PCs to large desktop PCs, processor vendors provide different products into each segment. Multiplying the number of CPUs and GPUs makes variations huge. In addition, combinations of GPUs are not limited to combinations of the same GPUs; they are sometimes different GPUs. On multi GPU PCs, at least one of GPUs must shoulder the graphics rendering. These features are considerably different from HPC area; these features make processors selecting more difficult.

To utilize multiple GPUs effectively, there are two types of systems. The first type is GPUs switching system. Virtu [85] is a middleware that provides GPUs switching between an iGPU and a dGPU. It hides a GPU from applications to force applications to recognize only

another GPU. It selects a GPU by totally static method; the user can statically configure the setting in its GUI. Although Virtu is useful in some cases, there is strict limitation on the combinations of installed GPUs. Besides, it can spoil applications' capability to utilize multiple GPUs concurrently. Optimus [92] is a driver based solution designed for PCs constructed with both an Intel's iGPU and a nVidia's dGPU. It works much effectively than Virtu in some cases. For instance, it can select GPUs adaptive to the state of AC connection. Moreover, it does not spoil applications' capability to utilize multiple GPUs concurrently. However, it only works with very limited hardware configurations. Besides, it does not adapt to other factors, such as working applications.

The other type is GPUs cooperation system. nVidia SLI [31] is a solution for two to four nVidia's graphics boards. CrossFire X [7] is a solution for two to four AMD's graphics boards. These are designed for cooperating multiple GPUs with the same specification. They do not provide GPUs switching because they focus on cooperating GPUs to improve graphics performance. However, there are variations of these technologies [10] [34] that work with different GPUs. For instance, AMD Dual Graphics [10] is a solution for PCs with an AMD's iGPU of AMD APU and a AMD's dGPU. It does not provide only GPUs cooperation, but provides also GPUs switching between iGPU mode and cooperative(dual GPUs) mode. However, these cooperation technologies are designed for graphics uses; they do not affect other types of processing like GPGPU in most cases.

Though many technologies are provided for switching GPUs on multiple GPUs environments, they do not adapt to the PC usage. Rather, they cannot switch processors between GPUs and CPUs. Therefore, such technologies cannot be said enough for preventing degradation of the user experience.

2.2 Real World Limitations

In this section, we state limitations that need to be considered to achieve processors switching on real environment.

2.2.1 Task Distribution between CPUs and Accelerators

There are many studies [73] [69] [80] that focus on distributing tasks between CPUs and GPUs. They are mainly designed for High Performance Computing(HPC) environment. However, many PC applications can only utilize CPUs. Because they do not have GPU computation code internally, we cannot switch computing resource to GPUs. Actually, some applications are difficult to be implemented as GPU accelerated; non-parallel processing cannot run quickly on GPUs. Besides, there is no guarantee that all of applications with both CPU and GPU code allow processors switching by an external system. Therefore, systems assuming that the majority of applications can run with GPUs are not suitable for PCs. Because a deployable

system can only switch a part of applications that have both CPU and GPU codes internally, we cannot achieve drastic performance improvement. Currently, the ratio of PC applications with both CPU and GPU code is still low. Therefore, a system that loosen significant performance decrement is needed on PCs, rather than a system that focuses on improving performance.

2.2.2 Binary Compatibility

Previously released applications are the most important property of PC OSes such as Windows and Mac OS. Therefore, systems should keep the compatibility with exiting applications. Like other studies, it is a straightforward idea that tightly limiting target applications to enable processors switching. For instance, some studies [73] [80] limits their targets as applications built with their original library. Though these methods show the effectiveness of their concepts, they are not deployable on PC platforms; we cannot switch processors on real world applications with such strict limitation. It is not expectable that application vendors are cooperative. Redesigning their applications with a new library do not benefit vendors straightforwardly. As stated in sec.2.2.1, the ratio of potentially processors switchable applications is low. Such systems cannot switch processors on most applications installed in PCs. Therefore, methods that need redesigning applications to distribute tasks are not deployable.

Another method to achieve switching processors on real applications is modifying the applications binaries. However, it is not acceptable because modification by a third person will violate the law or their software distribution policies. Besides, as a technical side, the cost of making binary patches for each update is too expensive. Therefore, we need a new method that can change the behaviors of applications without modifying applications' binary.

One of ideas to achieve binary compatibility is creating a dynamic recompilation system(*e.g.* Apple Rosetta [14]) or a system like virtual machines. However, these technologies can spoil the benefit of hand-tuned code implemented in applications. Furthermore, some parts of applications need specific processors. For instance, we cannot switch the computation resource for graphics rendering to CPUs. Besides, these technologies need long time to implement and validate. Therefore, this method is not suitable for enabling processors switching on PCs.

Another method to enabling processors switching is modifications to OSes [94] [83]. OSes can manage GPU resource in small granularity. However, modification of OSes often needs a long time to be merged into the released version. Also, some studies [94] only treats applications which built with their original system calls. Although the modification keeps the basic compatibility with legacy applications, the effectiveness of the system is reduced. Furthermore, many of these studies [94] [83] do not distribute tasks between GPUs and CPUs.

The other possibility is that modifying drivers to manage GPUs' resources [82]. These drivers achieve fine-grained resource management on GPUs without modifying applications. In addition, they potentially support any APIs designed for utilizing GPUs. However, this

approach cannot distribute tasks between GPUs and CPUs.

As stated above, the binary compatibility is one of most important requirements to achieve processors switching on real applications. Hence, we implemented a middleware that achieves processors switching by *API hooking*, called **Torta**. API hooking is a method that enables changing parameters and/or return values during hooking specific APIs. By hooking APIs, we can change the behavior of applications without modifying binary. The details of API hooking of Torta are stated in sec.3.5, 3.6 and 3.7.

2.3 States of PC usage and Users' Preference

Like our preliminary experiment of Splash(see. sec.1.1), irrelevant processors assignment causes serious performance issues. The appropriate processor varies depending on PC usage. Therefore, to satisfy users adequately, it is important for resource management systems to recognize the state of PC usage and adapt this information to management algorithms.

However, current PC OSes do not adapt information of the user context to their schedulers. For instance, it is probably possible to acquire contexts by sensors, such as where the person is and/or what the person try to do. Actually, today's laptop PCs are already rich with sensors, and spread of tablet PCs and appearance of Windows 8 [63] accelerate the richness of PCs with sensors. To improve the user experience of interactive computers, resource management systems should use these context information. Moreover, we actually need such information to assign computation resource on PCs with complicated heterogeneity stated in sec.2.1.

However, in reality, we cannot assign computation resource only by information of the user context. Fig.2.2 shows the result of FPS of video playback by Splash, on a different PC.

In this case, the FPS decreased only 45FPS from 60FPS. Thus, frame dropping may not be noticed by the user when the user does not direct his/her attention much to the video. In addition, some of the users do not care about such a little frame dropping; this completely depends on the user's preference. Playing games are another good example for explaining the user's preference. Some users seriously play a game to win. On the other hand, some users play the same game for killing time. Therefore, to improve the user experience, resource management systems need to use both information, the user's context and the user's preference.

State of AC power connection is the only factor that adapts to scheduler of current OSes. Variable clock speed technologies implemented to today's processors(see sec.2.1.2) also adapts with AC connection. However, PC OSes does not have any function to schedule tasks adaptive to the user's context and the user's preference. There is no way to express our preference to OSes at this time. Thus, following three methods are needed to achieve context/preference based scheduling;

- A meta language that can express PC usage state, users' contexts and users' preferences
- A user interface for expressing the information easily

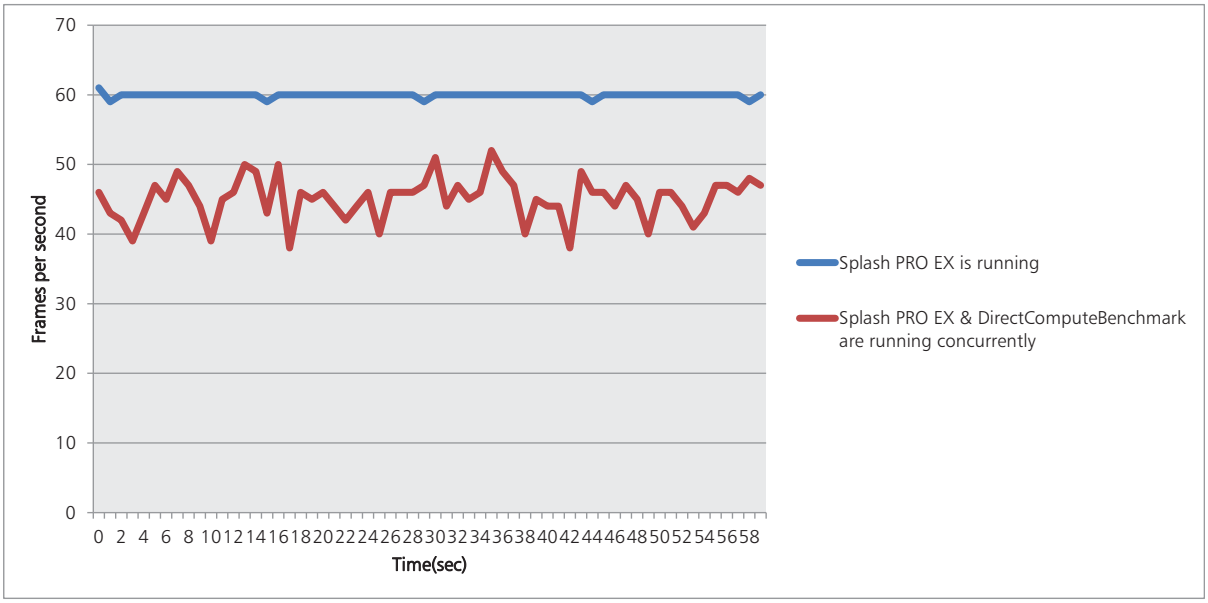

Figure 2.2: Result of Preliminary Experiment of Video Playback FPS Decrement during Another Application Utilize the Same GPU (2)

- A method that can convert the information into computer recognizable scheduling algorithms

2.4 Related Works

Task distributing between heterogeneous processors

Harmony [73] and other studies [69] [86] [80] propose methods for scheduling tasks between heterogeneous processors. They focus on improving existing metrics such as performance and power consumption. Therefore, their schedulers are not appropriate for preventing the user experience on PCs. However, their scheduling algorithms are considerable for assigning processors on PCs with applications that can distribute tasks with small granularity between heterogeneous processors.

Extending existing task distributing frameworks

Hybrid OpenCL [68] and others [67] [93] extend OpenCL to allow applications to utilize compute devices in remote hosts. Although our current implementation only assumes local compute devices, adapting these works to ours will make possible to offload some tasks into compute devices in the cloud to prevent degrading the user experience.

Resource management on GPUs

TimeGraph [82] and GERM [70] propose driver based system for achieving fair resource allocation on GPUs between multiple applications. Because Torta does not provide a GPU resource management, it works more effectively with these drivers.

Gdev [83] and PTask [94] are OS extensions for GPU resource management. PTask proposes dataflow programming model to reduce excessive data copy. Though they are actually effective, modification to OSes need a long time.

Scheduling methods proposed in [81] 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 [76], Pegasus [75] and vCUDA [96] virtualize on a hypervisor to share GPUs between multiple VMs. However, their works deal only with GPGPU tasks; It is not appropriate for applying them to PC platforms because they do not support other GPU tasks like video decoding.

GPU offloading

There are many offloading studies applicable to PC environments. Some studies [87] [77] propose offloading methods for encryption. [72] accelerates video encoding using GPUs. KinectFusion [79] enables real-time detailed 3D reconstructions using GPUs. Many studies [74] [88] use GPUs for accelerating video and image analysis. Increasing number of the GPU purposes increases the necessity of tasks distribution between CPUs and GPUs.

Chapter 3

Design of Torta

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

3.1 Assumed PC Environment

In this thesis, we assume that a PC is constructed with a multi-core CPU and one or more GPUs that are capable to execute GPGPU tasks. GPUs are not limited to dGPUs; iGPUs are also our targets. Because we assume GPUs are capable of GPGPU tasks, we can distribute some tasks between CPUs and GPUs.

As application programs, we assume 3 types of applications that can run concurrently. The first type of applications has both CPU code and GPU code internally. Also, Torta can switch the processors on these applications. The second type of applications also has both CPU code and GPU code; However, Torta cannot switch processors on these applications. The third type of applications is not capable to switch processors, due to lack of internal code for heterogeneous processors.

On our current implementation, we set our target applications as ones that are built with OpenCL. However, our concept does not only applicable to OpenCL applications, but also applicable to applications that utilize heterogeneous processors through following model.

Compute Device

Processors such as CPUs and GPUs are abstracted as *compute devices* in OpenCL. Because of this abstraction, applications can utilize different types of processors through the same steps. However, an abstraction layer is not necessary to apply our concept. Our concept is applicable when the application recognizes each processor as a kind of computing device and has code for each computing device.

Devices List

One of OpenCL APIs provides 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 concept does not limit the format of devices list; it is applicable with applications that select a compute device from a kind of lists that contain available computing devices.

Context

To utilize a compute device via OpenCL, applications create an *OpenCL context* which is similar to a virtual machine. Before executing tasks in 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 computing devices. In other cases, our concept does not need an object like OpenCL context.

3.2 Centralized Resource Management Model

As stated in sec.1.1, resource competition on the same compute device causes performance degradation of applications. The simplest way to avoid this problem is that applications select a compute device by themselves according to loads of each compute device. For instance, if a video playback application notices that another application is utilizing almost 100% of the GPU resource, then the video playback application can avoid frame dropping by selecting the CPU instead of the GPU.

However, some applications utilize a large part of the GPU resource even if the speed up brought by the GPU is small. Applying the method above, an application will avoid choosing the GPU due to its high load, even if the application is highly optimized with GPUs. Though this method probably prevents degradation of the user preference, it does not bring *total optimization*.

Like Splash case stated in Sec.1.1, this *load* information is not always useful for selecting a processor. Although Splash utilized about 45% of GPU computation resource, DirectComputeBenchmark easily interfered the FPS of video playback. It is difficult for applications to predict whether assigning a device critically affects other applications or not. It depends on both running applications and hardware configuration.

Consequently, processors assignment should be done by a centralized resource manager. It is straightforward idea that implementing such resource manager into OSes. However, as stated in sec.2.1.1, modifying OSes needs a long time. Thus, at this time, we implement the **Resource Manager(RM)** of Torta as user-land application. However, our concept does not restrict the implementation of the resource manager as user-space program. Our concept is also applicable to the resource management system implemented in OSes.

To achieve binary compatibility, we implemented **API Hooking Module(AHM)** that hooks APIs to change compute device assignment. (See sec.3.5, 3.6 and 3.7 for detailed design of AHM.)

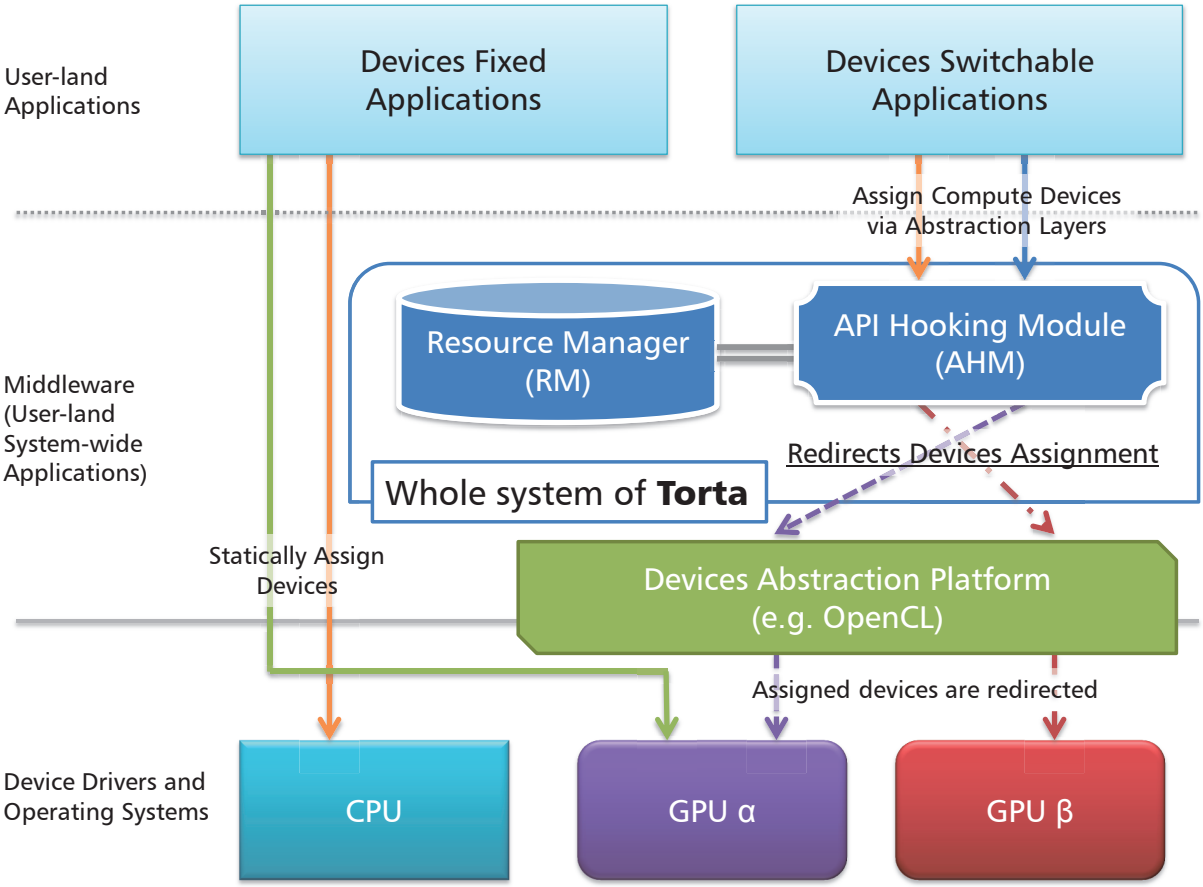

Figure 3.1: Assumed Environment and Centralized Resource Management Model

Fig.3.1 shows assumed environments of our work. Torta is constructed from two parts stated above, RM and AHM. To redirect device assignment, RM and AHM work together by communicating each other. Firstly, AHM sends the list of all available compute devices to RM. Secondly, RM chooses an *appropriate* device for the application adaptive to the user's context and preference, and answers selected device to the application. Thirdly, AHM changes the ID of compute device during API call to change the device assignment. Lastly AHM sends the actual assignment information into RM; thus RM can manage device assignment centrally.

Through these steps, both device assignment information and device assignment authority are centralized. Therefore, Torta can centrally manage the resources between CPUs and GPUs.

3.3 Managing PC Usage States and User Preferences

As stated in sec.2.3, meaningful compute devices assignment can only be achieved when the resource management system adapts to the PC usage and the user's preference.

Generalizing PC usage is inappropriate to prevent degradation of the user experience. Appropriate device assignment will vary depending on the user, even if the same application is launched. A user may be a professional graphics designer, on the other hand, another user may be just a beginner about editing pictures. The importance of the application can vary even on the same user. Sometimes, a user concentrates his/her attention to do his/her task using the application. However, the same user sometimes launches the same application just for fun.

Because of the reasons above, a system cannot list all the situations that are brought by his/her *context* and *preference*. Therefore, Torta asks the user to express his/her context and preference to *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 is prepared for expressing the user's context. The assignment rule part is prepared for expressing his/her preference. Thus, we can choose a profile by comparing the present context and the context expressed in the condition part. Then, the appropriate device will be selected for each application, by following the assignment rules expressed in the second part.

To express a situation, Torta accepts the conditions like executed applications and AC connection. Moreover, we assume that various information can help to express the user's situation, such as where the user is, how many key types per minutes, which part of display the user looking at, and so on.

However, we cannot easily imagine all of the information that can help to express contexts. Besides, collectible information strongly depends on the hardware configuration. Therefore, we do not focus on creating format for expressing users' contexts; we chose executed applications and AC connection to implement our whole concept.

On the other hand, the method for expressing assignment rule is essential for selecting devices. We defined the rules for expressing assignment as following;

Fixed Assignment

This rule allows assigning a specific device to applications that are specially important for the user. Because this rule is *fixed*, Torta always assigns specified device to the application even when load of the target device is high. To make the system more effective, this rule has two options. The options are not exclusive; thus users can specify both options concurrently.

- **Monopoly**
  This option allows assigning a specific device only to a specific application. If this option is

enabled, Torta never assign the device to other applications managed by Torta. However, we cannot achieve actual *monopoly* because Torta cannot manage devices assignment of all applications running on the PC. Especially, it is difficult to assign most of the device's resource to the application when the user specifies this option with the CPU. However, we can make this option meaningful by increasing the priority value of the OS's scheduler.

- **Virtual**
  This option is prepared for expressing applications that Torta cannot manage devices assignment. Users can express the devices assignment fixedly done by the application. For instance, when the user registered the device assignment of a game application as the GPU with both Monopoly and Virtual options, Torta will help to improve the FPS of the game by switching the devices of other applications to the CPU.

Prioritized Assignment

This rule allows expressing that the user is willing to assign a specific device to an application because the application is moderately important for the user. Actually, applications specified as *Fixed Assignment* always precede; Torta assigns the specified device to the application when there is still enough resource available on the target device. Needless to say, if specified device is utilized by a *Fixed* application with *Monopoly* mode, then *Prioritized* applications cannot use the device.

To improve the effectiveness of this rule, there are three options that can be specified with. As same as options of *Fix Assignment*, these options are not exclusive.

- **Priority Level**
  In some cases, *Prioritized* applications compete with other *Prioritized* applications. To give a specific device to a part of competing applications, we can set this *Priority Level* to order applications' importance. Applications with low *Priority Level* are often forced to use a device which is not specified in the profile. At this time, we set four grades of *Priority Level*, 0 to 3. Higher level is more important than lower ones. However, the optimal number of the priority grades may need to be changed depending on the granularity of devices switching timing.
- **Secondary Device**
  As stated above, these prioritized applications may not be able to use the specified device. Still these applications are important for the user; this *Secondary Device* option allows specifying a device which should be assigned when the application cannot use the firstly specified device. *Priority Level* option also affects this option; applications may not be able to use even the *Secondary Device*.
- **Forbidden Device**
  Forbidden Device option allows specifying devices that the system must not assigned into

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.

As you can imagine, a method that simply assigns faster device to important applications will not work well. Assuming that the most important application A was assigned the fastest device α, for instance, we cannot easily decide the device for the second most important application B. To decide it algorithmically, we need information of execution speed on both of these situations; (i)application A and B work together on device α, (ii)application A works on device α and application B work on device β.

If application B works faster on (ii) than (i), it is clear that assigning devices as (ii) is appropriate. However, if application B works faster on (i), it is difficult for the system to select a device for application B. As stated in sec.2.1.1, resources of GPUs are not well managed. Therefore, (i) will cause performance degradation to application A, perhaps great degradation. A system needs a threshold between performance degradation of application A and performance gain of application B to decide the processor for application B.. To decide this threshold, a system need linear scale of applications' importance like "application A is 3 times important the application B". However, it is almost impossible for users to define such values. Though human beings can order applications by importance in most cases, they cannot calculate the importance of them.

Furthermore, if an application is designed for real-time processing like video playback, it does not need a device that can achieve more than real-time. Although applications designed for real-time processing should be important, they do not always need the fastest device. Some users interested in the difference of the video quality relying on the decoding device, rather than assigning the fastest device to the application. We cannot give the score for the video output of each application; just the user prefers one to another. All things considered, pure priority model is not applicable.

3.4 Switching Profiles

As stated in sec.3.3, profiles contain context conditions such as running applications and AC connection. Therefore, we can choose a most appropriate profile by comparing actual context and the condition written in profiles.

However, we do not focus on automatic switching of profiles; it is not possible to create a method for automatic switching without detailed design of the context description format.

3.5 Interruption on devices selection

In OpenCL, processors such as CPUs and GPUs are abstracted as *compute devices*. Therefore, Torta change the device ID to switch the device when an application try to assign a device to utilize. To change the value of device ID, Torta uses a method called *API Hook*(see sec.4.2 for detailed implementation).

To switch devices on each application, RM of Torta need to know all the device IDs available on the target application. Because some applications try to list only GPUs, AHM lists all available devices by itself and sends the list to RM.

After sending the device list to RM, Torta does not assign a device to the application until the application try to decide a device to utilize. The timing that the application tries to decide the device depends on both the application's implementation and the using computing framework(*e.g.* OpenCL). When the application tries to decide a device, AHM of Torta asks RM to decide a device for the application. Then, RM decides a device according to the current profile and answers the appropriate device. AHM switches the device ID from one that the application selected to one that RM specified. After these steps, applications keeps working as usual except the difference of assigned compute device.

3.6 Device Information Handling

Even if using an abstraction layer like OpenCL, applications need to collect detailed device information to optimize their performance. *Compute Devices* are not completely abstracted, there are still some difference between devices. However, some applications collect the detailed information of devices before selecting a device to utilize. In other words, these applications implemented as selecting a device according to the devices' detailed specification. On this kind of applications, changing device ID can cause incoherence of the device information. The device information of actually using one can be different from the device information that the application acquired. This incoherent information does not only cause performance degradation, but it can also cause unexpected behavior to the application. Therefore, Torta improves applications' compatibility by following three methods;

Changing Device Type Information

Some of the applications show incomprehensible behavior when the information of device type(*e.g.* CPU, GPU) is incoherent. For instance, one of such applications returns an error when that the CPU is assigned to it despite that the application specified a GPU device. Therefore, AHM of Torta inquire the *corrected* device type from RM. Then, AHM changes the information of device type as RM answered to improve applications' compatibility.

Remapping Device IDs

To prevent making incoherent device information as well as possible, Torta remaps device IDs after assigning a device. When one of the application's threads finishes assigning, Torta starts remapping device IDs to actually assigned device ID. Therefore, gathered information of the device becomes coherent to the information of actually assigned one. This also helps to improve applications' compatibility.

Correcting Information After Assigning a Device

Each device has the different limitation of maximum working array size to execute parallelized tasks. Applications usually confirm this limitation before utilizing the device. However, this information also becomes incoherence when Torta switched the device. If the array size exceeds the limitation of assigned device, the application cannot work correctly; the application will crash or stop with errors. Because information incoherence is unavoidable in some cases, Torta also changes such information retained by applications after switching devices.

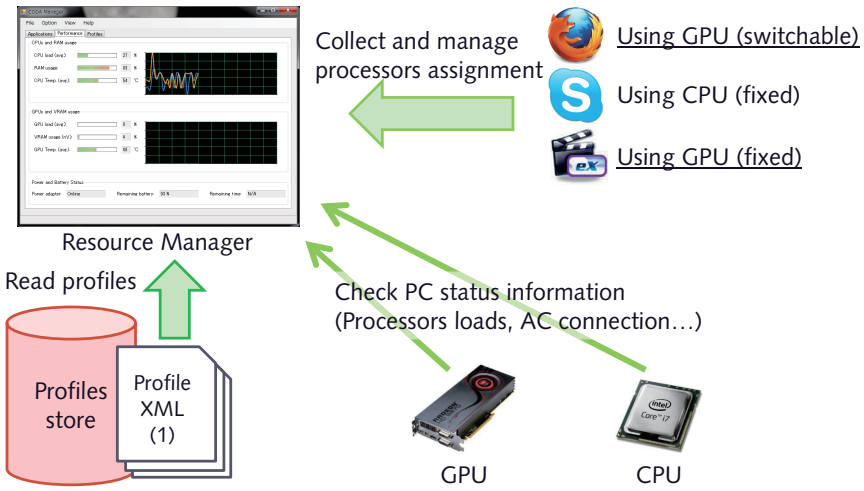

Figure 3.2: Summarized Conceptual Torta's Behavior (1)

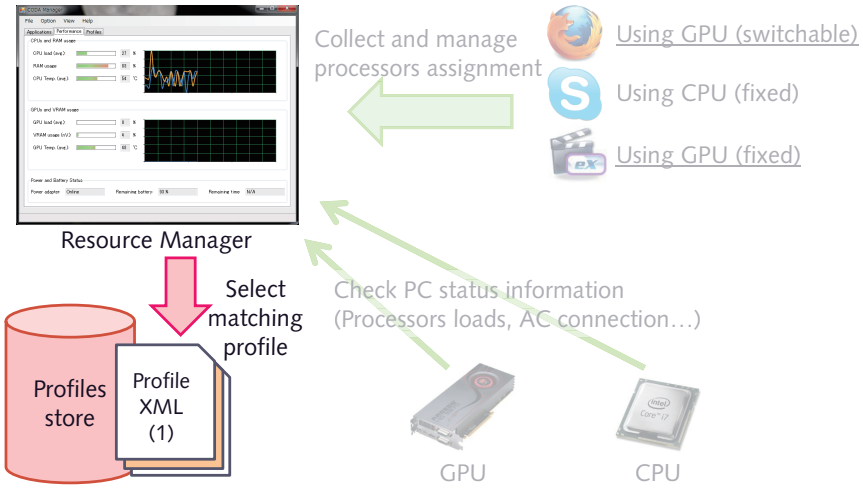

Figure 3.3: Summarized Conceptual Torta's Behavior (2)

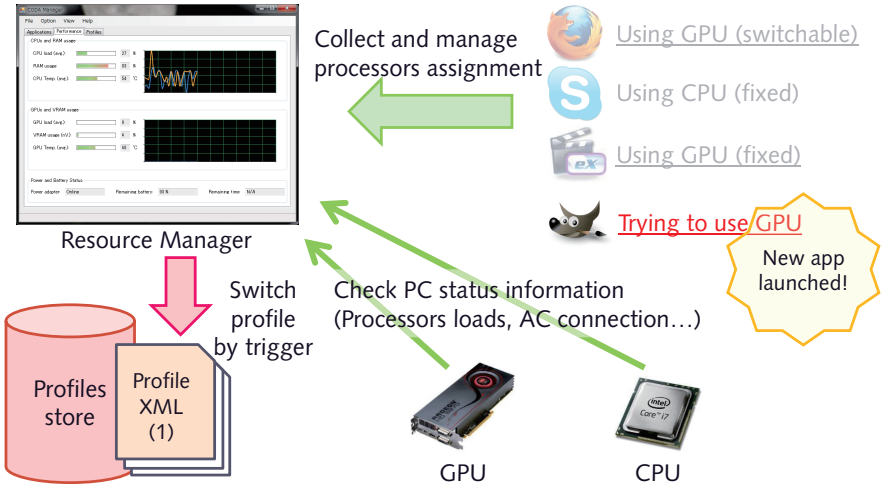

Figure 3.4: Summarized Conceptual Torta's Behavior (3)

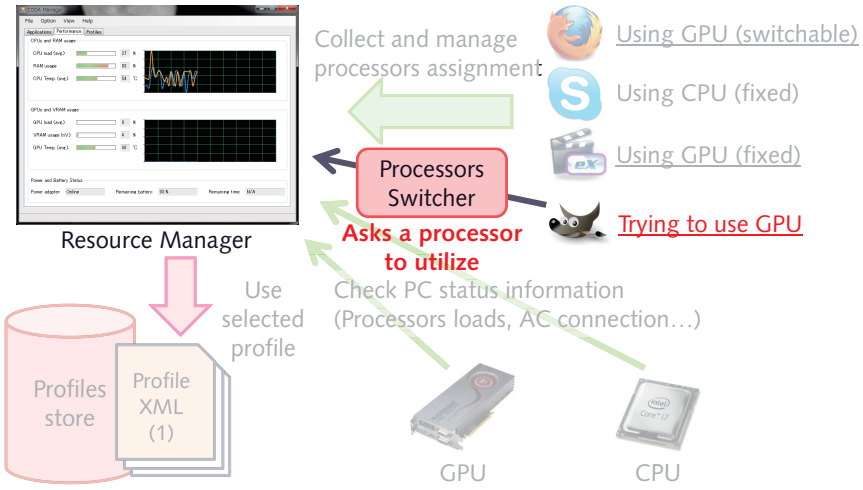

Figure 3.5: Summarized Conceptual Torta's Behavior (4)

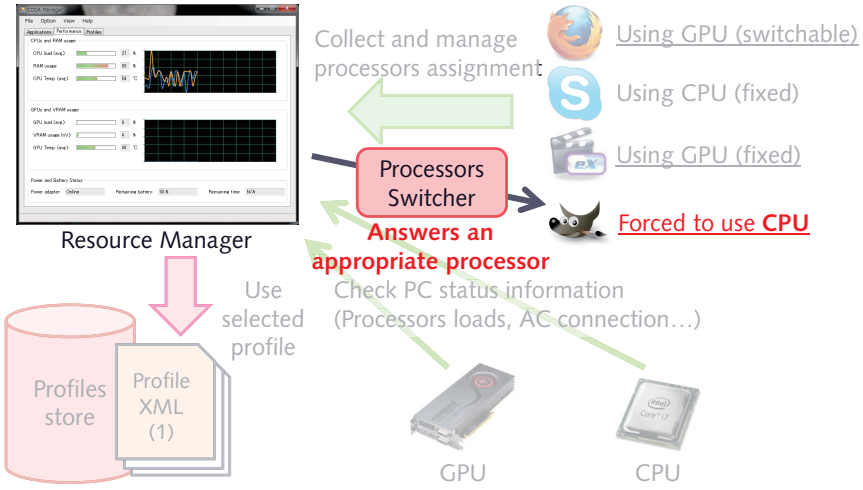

Figure 3.6: Summarized Conceptual Torta's Behavior (5)

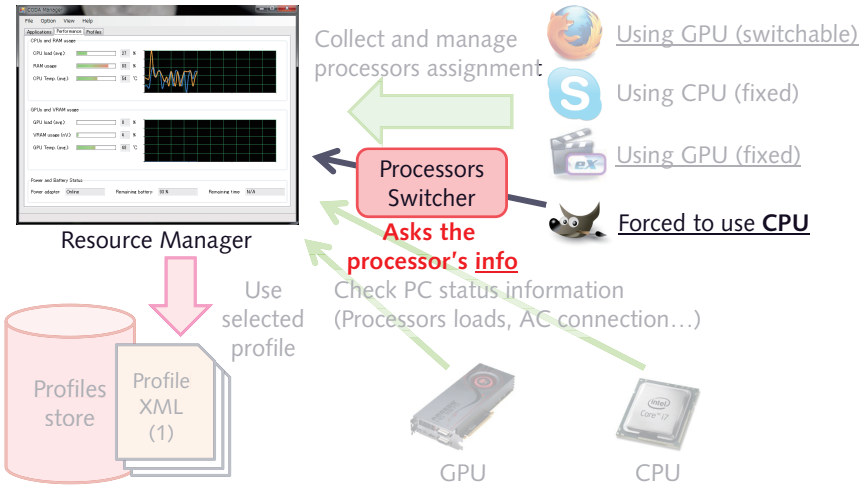

Figure 3.7: Summarized Conceptual Torta's Behavior (6)

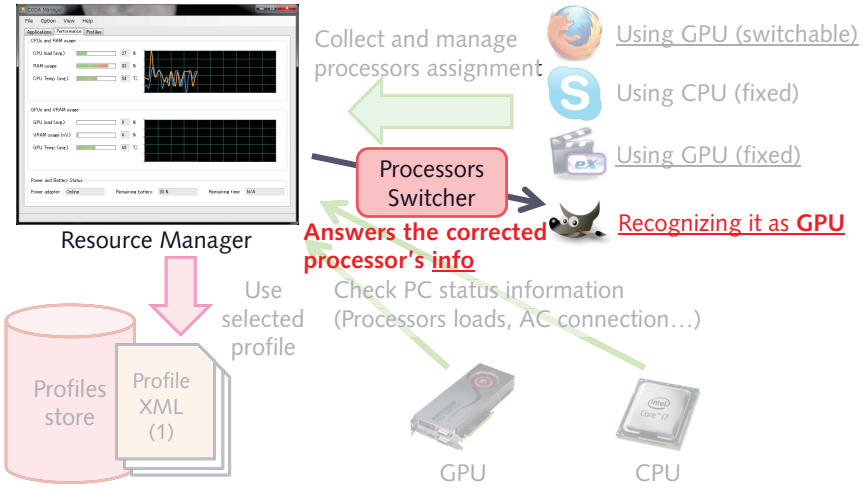

Figure 3.8: Summarized Conceptual Torta's Behavior (7)

Fig.3.2 to 3.8 show that the summarized conceptual behavior of whole our system. RM of

Torta assigns a device according to the current profile and AHM applies actual device switching to applications.

3.7 Limiting the number of recognized devices

Some applications utilize multiple devices concurrently to improve their performance. In reality, applications always need utilizing the CPU to manage tasks assigned into GPUs. However, if an application utilize all of the installed GPUs concurrently, other GPU accelerated applications suffer performance degradation caused by resource competition.

For instance, an application is implemented as utilizing all of the available GPUs. When this application is executed on a PC with two GPUs(an iGPU and a dGPU), the application will try to utilize both of them. If Torta redirects one of the GPU assignment to another GPU, such as iGPU & dGPU to dGPU & dGPU, performance of application will decrease due to meaningless overhead brought by *task distribution* on the same GPU.

These applications decide the number of devices to utilize by the number of devices they recognized. Therefore, Torta limits the number of devices to show to such applications. Assuming the same application stated above, the application will try to utilize only one GPU if Torta shows only the dGPU. This method prevents two types of performance degradation; one is performance of other applications and the other is performance of the target application. It prevents performance degradation by avoiding meaningless *task distribution*.

The limiting number of devices is not limited to one; this method also allows any natural number less than the number of all available devices.

Chapter 4

Implementation

In this chapter, we state detailed implementation of Resource Manager and API Hooking Module.

4.1 Resource Manager

As stated in sec.3.2, Resource Manager of Torta manages device assignment centrally. To achieve centralized management, we devised implementation of RM as following.

4.1.1 Applications Handling

RM manages applications by their file name(*e.g.* notepad.exe). To switch devices stably, Torta prepares following three options that can be specified with applications' file name.

Overriding the Device Type for Device Listing

Option Name: TypeOverride
Values:

- ALL: shows all types of devices (*default*)
- CPU: shows only CPU-type devices
- GPU: shows only GPU-type devices

As stated in sec.3.5, some applications list only specific type devices. Because AHM has own list that includes all devices, Torta can assign a device that is not included in the application's device list. However, to improve applications' stability, Torta can show all types of devices by setting this value as *ALL*, even when applications try to list only specific type devices such as GPUs. Showing all devices avoids assigning a device unknown to the application. On the

other hand, showing specific type devices improves the compatibility on applications that work correctly only when showing specific type devices.

AHM changes a parameter of the device listing API according to this value.(see sec.4.2.4 for detailed implementation.)

Device Type Camouflaging

Option Name: InfoOverride
Values:

- CPU: shows the device as a CPU
- GPU: shows the device as a GPU
- *Not specifying*: shows the device as real type (*default*)

As stated in sec.3.6, some application behave strangely when they find incoherent information of device types between the device actually assigned and one they specified. Therefore, Torta changes the device type information of devices to improve applications' compatibility. For instance, when setting this value as *GPU*, applications recognize a device as a GPU even when the device is a CPU.

As same as above, AHM changes a parameter of the information acquiring API according to this value.(see sec.4.2.6 for detailed implementation.)

Limiting the Number of Listing Devices

Option Name: DeviceNumberReduction
Values:

- true: limits the number of devices to one (same as value of *1*)
- *Natural numbers*: limits the number of devices to specified value
- *Not specifying*: does not limit the number of devices (*default*)

As stated in sec.3.7, some applications try to utilize multiple devices concurrently when they recognize multiple devices. To avoid performance degradation of other applications caused by this implementation, Torta limits the number of devices shown to such applications. For instance, setting this value as *2* makes the maximum number of recognized devices to two. Of course, the number of recognized devices cannot exceed the number of all devices.

4.1.2 Inquiry Handling

Torta manages applications by their file names to specify compatibility options stated above. On the other hand, to assign different devices to multiple processes of the same application or

multiple threads of the same process, Torta manages running applications by process IDs and thread IDs.

As you can imagine, our mechanism dose not assume POSIX *fork()*. We assume situations like that multiple audio encoder processes run concurrently to finish encoding tasks early on a multi-core CPU. Windows applications usually make threads for executing tasks in parallel. On applications implemented as multi threaded, we can assign different devices for each thread. Therefore, Torta manages running applications by both process IDs and thread IDs.

To answer various inquiries from every thread, RM communicate with all threads of the target processes. Because of per-thread management model, Torta cannot only assign different devices for each thread, but it can also collect various information like the execution time of OpenCL code on each thread.

4.2 API Hooking Module

As stated in sec.3.2, API Hooking Module of Torta actually handles device assignment. To achieve binary compatible switching on real applications, we devised implementation of AHM as following.

4.2.1 Target OpenCL Platforms

There are three OpenCL platforms which are widely used on Windows platforms.

- AMD Accelerated Parallel Processing
- NVIDIA CUDA
- Intel(R) OpenCL

We targeted all of these platforms to make our system deployable. OpenCL includes ICD(Installable Client Driver) system that fundamentally allows installing multiple OpenCL platforms concurrently. Therefore, we assume our target PCs as PCs that include one to three of these platforms. Torta does not achieve devices switching only on a single platform, but it also achieve devices switching between multiple platforms.

4.2.2 API Hooking Methods

On Windows 7 [62], our implementation and evaluation environment, there are several choices of API hooking methods as following:

- Creating a remote thread by calling *CreateRemoteThread()* [89] Windows API
- Applying global hook by calling *SetWindowsHookEx()* [90] Windows API
- Using one of API Hooking libraries [91] [45] [24]
- Creating a wrapper DLL and placing it to application installed directory

The first method is creating a thread in the target application by calling a Windows API *CreateRemoteThread()*. If choosing this method, RM has to observe each application start and create threads into target applications. Therefore, it has a considerable risk that causes crash of target applications or making the PC unstable when RM has crashed. The second method is called *Global Hook*, which enables system wide API hooking by user-land applications. Due to its easiness and convenience, this is a major method for API hooking. Registering global hot keys is a major use of this method. If choosing this method, it has the same risk as above because the responsibility of RM is heavy. The third method is achieved by API hooking libraries. They are easy to use and provide stable API hooking. However, this method also has the same problem as the first and the second ones, because we have to implement the API hooking function into RM. The last method is creating a *Wrapper DLL* which is a DLL contains all API entry points of the original DLL. A wrapper DLL is almost shell of the original DLL because most APIs implemented in a wrapper DLL finally call the original API to do their tasks. Wrapper DLLs only implement processing that are different from the original. A wrapper DLL is loaded from the application when placing it to the application directory that contains the application binary(*.exe). It has almost 100% chance to successfully hooking the APIs because applications always load the DLLs in its own directory first by Windows default. Because a wrapper DLL affects only the application stored in the same directory, each application is hooked in isolated. Therefore, it improves the stability of both RM and each application. Due to its stability, this method is used for extending 3D sound APIs for real applications. [18]

According to this comparison, we chose the *Wrapper DLL* method. Stability is essential for creating a deployable system.

Because these methods are provided by Windows, we cannot apply the same method to other OSes such as MacOS. However, our design does not limit the methods of API hooking; our design can also adapt to other API hooking methods provided by other OSes.

4.2.3 Overridden OpenCL APIs

AHM of Torta hooks all the API entry points of *OpenCL.dll* ver.1.2.0.0 to log all API calling. This logging function is needed for analyzing the applications' behavior during assigning a device by Torta. The log data helped recognizing how the information incoherence occurs.

On the other hand, for actual device switching, there are five especially important APIs hooked by AHM.

clGetDeviceIDs

Source Code 4.1: Format of clGetDeviceIDs()

```
1 cl_int clGetDeviceIDs ( cl_platform_id  platform ,
2   cl_device_type  device_type ,
```

```
3   cl_uint  num_entries ,
4   cl_device_id  *devices ,
5   cl_uint  *num_devices )
```

This API is used for listing available devices in the PC. Actual processing during hooking this API is stated in sec.4.2.4.

clGetDeviceInfo

Source Code 4.2: Format of clGetDeviceInfo()

```
1 cl_int clGetDeviceInfo ( cl_device_id  device ,
2   cl_device_info  param_name ,
3   size_t  param_value_size ,
4   void  *param_value ,
5   size_t  *param_value_size_ret )
```

This API is used for gathering information of a specific device. Actual processing during hooking this API is stated in sec.4.2.6.

clCreateContext, clCreateContextFromType

Source Code 4.3: Format of clCreateContext()

```
1 cl_context clCreateContext( const cl_context_properties *properties,
2   cl_uint num_devices,
3   const cl_device_id *devices,
4   (void CL_CALLBACK  *pfn_notify) (
5 const char *errinfo,
6                         const void *private_info, size_t cb,
7                         void *user_data
8 ),
9   void *user_data,
10  cl_int *errcode_ret)
```

Source Code 4.4: Format of clCreateContextFromType()

```
1 cl_context clCreateContextFromType ( const cl_context_properties
  *properties,
2   cl_device_type  device_type,
3   void  (CL_CALLBACK *pfn_notify) (const char *errinfo,
```

```
4    const void  *private_info,
5    size_t  cb,
6    void  *user_data),
7    void  *user_data,
8    cl_int  *errcode_ret)
```

These APIs are used for creating OpenCL context. Applications choose devices to include the context when calling these APIs. Actual processing during hooking this API is stated in sec.4.2.5.

clEnqueueNDRangeKernel

Source Code 4.5: Format of clEnqueueNDRangeKernel()

```
1 cl_int clEnqueueNDRangeKernel ( cl_command_queue command_queue,
2   cl_kernel kernel,
3   cl_uint work_dim,
4   const size_t *global_work_offset,
5   const size_t *global_work_size,
6   const size_t *local_work_size,
7   cl_uint num_events_in_wait_list,
8   const cl_event *event_wait_list,
9   cl_event *event)
```

This API is used for enqueueing a command to execute an OpenCL kernel on a device. To avoid errors caused by unusual *local_work_size* value, AHM corrects this value as a number by which the *global_work_size* is divisible. Such unusual values are caused by incoherence of device's information. Therefore, Torta corrects such unusual values to improve compatibility. On this processing, AHM can correct values by itself; this processing does not need an inquiry to RM because corrected values are conducted by simple calculation.

4.2.4 Device Listing

Torta changes the type of listing devices and limits the maximum number of listing devices by hooking *clGetDeviceIDs()*(Source Code 4.1).

Changing the Device Type

When an application calls *clGetDeviceIDs()* API, AHM asks RM the device type to list which is appropriate for the application. The answer is returned as *CPU*, *GPU* or *ALL*, thus, AHM converts the answer as an OpenCL constant, such as *CL_DEVICE_TYPE_CPU*, *CL_DEVICE_TYPE_GPU*

or *CL_DEVICE_TYPE_ALL*. Then, AHM calls original *clGetDeviceIDs()* API with setting *device_type* as one of these three types.

For instance, setting *device_type* as *CL_DEVICE_TYPE_ALL* allows showing all available devices to the application. However, the number of listed devices can vary depending on the specified device type. Thus, applications that specified the GPU may not prepare enough length of the array for listing all devices. In fact, most of the applications confirm the number of devices before listing the devices, by calling the same API with setting *devices* parameter as *NULL*. Therefore, Torta sets the same *device_type* value to both API calling to allow applications to prepare enough length of the array.

Limiting the Number of Devices

In *clGetDeviceIDs()*, the parameter of *num_entries* limits the maximum number of listing devices. Because we need to call this API for every platform, Torta need to set *num_entries* properly for each platform. AHM asks RM the maximum number of the devices for the application *when the application is launched*. Usually, applications check available devices when they are launched, then select one from the list when they start their parallelized processing. Because acquiring devices list again and again is meaningless, it is rational that acquiring devices list only when the application launched. Therefore, applications decide the number of devices to concurrently utilize when they are launched. Hence, Torta also asks the maximum number allowed to be used only when the application launched.

The maximum number that RM answered is stored in a variable of AHM. As stated in above, applications usually call *clGetDeviceIDs()* twice for the same platform. At the first time, *clGetDeviceIDs()* is called with *devices* as *NULL* to recognize the number of devices available in the platform. When the API is called as this method, AHM sets **num_devices* value as the minimum of following values;

- (The maximum number of devices to show) - (The number of devices already shown)
- The number of available devices in the platform

At the second time, *clGetDeviceIDs()* is called with *devices* as non-*NULL* value to list the device IDs. When the API is called as this method, AHM sets *num_entries* value as the minimum of following values;

- **num_devices* which was set when the API is called with the first method
- *num_entries* value that the application specified

However, if the minimum of these values is 0, setting *num_entries* as 0(equals *NULL*) makes the API calling as same meaning as the first method. Therefore, in such cases, AHM sets the *num_entries* to 1 first, then call the *clGetDeviceIDs()*. After calling the API, Torta corrects **num_devices* to 0. Because **num_devices* is zero, applications recognize that there is no available device in the platform.

4.2.5 Device Assignment

With OpenCL APIs, applications utilize a device by following steps:

1. Acquire the list of devices
2. Decide a device to utilize, or decide a platform that includes devices to utilize
3. Create a context that includes device(s) chosen in the previous step
4. Compile a program with the created context
5. Create a command queue(see Source Code 4.6) by determining a device to utilize
6. Enqueue necessary transactions to the command queue, such as data copying or OpenCL kernels

Source Code 4.6: Format of clCreateCommandQueue()

```
1 cl_command_queue clCreateCommandQueue( cl_context context,
2   cl_device_id device,
3   cl_command_queue_properties properties,
4   cl_int *errcode_ret)
```

Because of steps stated above, we need to determine the target device substantially when creating a context. OpenCL specification does not allow creating a context that includes devices in different platforms. For instance, we cannot create a context with both an Intel's CPU(Intel's platform) and a nVidia's GPU(nVidia's platform). Therefore, in most cases, we need to fix the device when creating a context.

According to this limitation of OpenCL, Torta always determine a device to assign when creating a context. To determine a device to assign, AHM asks RM the device when the applications calls *clCreateContext()*(Source Code 4.3) or *clCreateContextFromType()*(Source Code 4.4). Because RM returns a device ID, AHM creates a context that includes only the answered device. In fact, it is possible to create a context with multiple devices in the same platform. Our previous implementation [78] creates a context with multiple devices to achieve devices switching after creating a context. However, supported functions(*e.g.* OpenCL Extensions) of devices are subtly different; creating a context with multiple devices limits available functions of the context to functions supported by all devices in the context. Lack of functions does not only decrease the performance of applications, but it also decreases applications' compatibility. Therefore, our current implementation always creates a context with only a device. In addition, compilation of OpenCL code is slightly faster on a context only with a device than a context with multiple devices.

This limitation is caused by non-perfect abstraction of devices. However, OpenCL focuses on optimizing applications' performance by showing detailed hardware information to applications. Therefore, this limitation is inevitable.

Different from *clCreateContext()*, *clCreateContextFromType()* only allows specifying a platform instead of specifying device(s). Therefore, Torta substitutes *clCreateContextFromType()* with *clCreateContext()* to create a context that includes only a device.

After creating a context, the application creates a command queue via *clCreateCommandQueue()*(Source Code 4.6) to start utilizing a device. However, the device ID specified by RM can differ from the device ID specified by the application when calling *clCreateCommandQueue*. Because it is not possible to create a command queue of a device that is not included in the context, calling *clCreateCommandQueue()* with such device IDs causes an error. Therefore, when an application specifies a device ID that is not included in the context, AHM overrides the *device* parameter to the device ID that is included in the context. To override this parameter, AHM does not need to ask the device to RM because AHM can confirm the correct device ID included in the context by itself.

4.2.6 Device Information Handling

As stated in sec.3.6, to improve applications' compatibility, Torta changes the information of device type and remaps the device ID when the API for acquiring device information is called.

Overriding Device Type Information

When *clGetDeviceInfo()* is called for acquiring information of the device's type, AHM asks the *corrected* device type to RM. To apply this transaction only for the device type acquiring, AHM checks whether the *param_name* is *CL_DEVICE_TYPE*. Because RM answers the device type information as CPU or GPU, AHM sets the value of **param_value* as *CL_DEVICE_TYPE_CPU* or *CL_DEVICE_TYPE_GPU*. By changing the result of this information inquiry, Torta can force applications to recognize a device as a different type of device.

Remapping Device IDs

Once an application creates a context, device assignment of the thread becomes fixed. Therefore, after creating a context, Torta remaps the device ID specified in the application's *clGetDeviceInfo* calling to the device ID which is actually included in the context. Remapping transactions do not need communication between AHM and RM; AHM can confirm the device included in the context by itself. By remapping device IDs, the applications gain more chance to know the correct information of the assigned device. Thus, this method certainly improves applications' compatibility. AHM keeps remapping device IDs during the context is valid; AHM stops remapping device IDs when the *clReleaseContext()*(Source Code 4.7) is called.

Source Code 4.7: Format of clReleaseContext()

```
1 cl_int clReleaseContext ( cl_context context)
```

Chapter 5

Evaluation

As stated in sec.3, we focused on enabling compute devices switching on real applications adapting to the rules described in the selected profile. Therefore, we evaluated deployability, compatibility with various hardware and software, and performance effects of Torta.

5.1 Evaluation Environment

As our experiment environment, we chose these OSes and OpenCL platforms.

Operating Systems

- Windows 7 SP1 x64 (Ultimate, Professional and Home Premium)
- Windows 7 SP1 x86 Professional

OpenCL Platforms

- For Intel CPUs: Intel SDK for OpenCL* Applications 2012 [41]
- For Intel iGPUs(incl. Intel CPUs): Intel HD Graphics Driver 8.15.10.2761 [39]
- For AMD CPUs(incl. Intel CPUs): AMD APP SDK ver. 2.7 [2]
- For AMD GPUs(incl. AMD/Intel CPUs): Catalyst Software Suite with .NET 4 Support 12.10 [5] (Catalyst Software Suite 12.6 [54] on PC(A) 64bit)
- For nVidia GPUs: GeForce 306.97 Driver [50]

As hardware configurations, we prepared following PCs for our experiments.

(A) Desktop PC with Windows 7 x64 / x86

- Intel Core i7-3770K [37] CPU
- Intel HD Graphics 4000 [38] integrated GPU
- AMD RADEON HD 7700 GHz Edition [13] GPU

- nVidia GeForce GTX 660 [30] GPU

(B) Desktop PC with Windows 7 x64

- AMD FX-8150 [8] CPU
- AMD RADEON HD 6970 [12] GPU x2
- nVidia GeForce GTX 550 Ti [29] GPU

(C) Laptop PC(MouseComputer LuvBook LB-K801X) with Windows 7 x64

- Intel Core i7-2760QM [35] CPU
- Intel HD Graphics 3000 [38] integrated GPU
- nVidia GeForce GT 555M [27] GPU with Optimus [92] Technology

(D) Laptop PC(ASUS K53TA) with Windows 7 x64

- AMD A6-3400M [49] APU
- AMD RADEON HD 6520G [49] integrated GPU
- AMD RADEON HD 6650M [11] GPU works as RADEON HD 6720G2 by Dual Graphics

(E) Ultrabook PC(ASUS UX32VD) with Windows 7 x64

- Intel Core-i7 3517U [36] CPU
- Intel HD Graphics 4000 [38] integrated GPU
- nVidia GeForce GT620M [28] with Optimus Technolgy

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

- DirectComputeBenchmark ver.0.45b [20]
  DirectComputeBenchmark is a benchmark application that can benchmark performance of both CPUs and GPUs. It does not only support DirectCompute, but also supports OpenCL.
- N-Queen Solver for OpenCL [47]
  N-Queen Solver for OpenCL is an application that can solve n-queens problem quickly by utilizing GPUs via OpenCL. It can use both GPUs and CPUs. We compiled this application from the original source code with Visual Studio 2010 [44] SP1 for our experiments.
- ratGPU ver.0.5.5 [56]
  ratGPU is an OpenCL ray tracing renderer. Though it fundamentally works as a renderer for 3D modeling applications, the package also includes a standalone renderer(StandAloneRenderer.exe) for benchmarking. We used the 32bit version of *StandAloneRenderer.exe* for our experiments.

- LuxMark ver.2.0 32bit [43]
  LuxMark is an OpenCL benchmark application, based on LuxRender which is a physically based and unbiased rendering engine. To accelerate its heavy rendering, it uses OpenCL.
- WinZIP ver.17.0(10283) 32bit [65]
  WinZIP is one of most famous file archiver and compressor application. It can compress files rapidly using GPUs via OpenCL.
- FLACCL (in CUETools) ver.2.1.4 [26]
  FLACCL is an application that can encode audio files into lossless audio format "FLAC" [25] quickly by utilizing GPUs via OpenCL. It supports both GPUs and CPUs for its processing.
- Adobe Photoshop CS6 EXTENDED ver.13.0.1 x32 [3]
  Adobe Photoshop is the world most famous picture editing application. It supports OpenCL to accelerate its processing.
- GIMP ver.2.8.2 x86 [32]
  GIMP is also a well known picture editing application. It also supports OpenCL to accelerate its processing.
- PhotoMonkee ver.0.56b [53]
  PhotoMonkee is a new picture editing application that strongly focuses on accelerating its processing utilizing OpenCL.
- vReveal 3 ver.3.2.0.13029 [60]
  vReveal 3 is an application that can improve video quality by many kinds of processing such as sharpening and noise reduction. To accelerate its heavy processing, vReveal 3 utilize GPUs via OpenCL.
- x264 with OpenCL lookahead patch rev.2230+696_tMod [66]
  x264 [59] is one of most famous H.264/MPEG-4 AVC [33] encoder. We used an experimental version of x264 that was built with OpenCL lookahead patch. Processing of lookahead is only a small part of encoding; it does not increase encoding speed much. However, we chose this application because x264 is a major application.

5.2 Quantitative Evaluation

In this section, we state policy and methods of quantitative evaluation. Firstly we state the evaluation of switching compatibility, Secondly we state the evaluation of switching overheads. Lastly, we state the evaluation of performance degradation caused by Torta.

5.2.1 Switching Compatibility

To evaluate compatibility of devices switching, we experimented Torta with applications written in sec.5.1 on PCs written in the same section. Fig.5.1 shows the result of our switching

experiments.

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 lookahead patch	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

Table 5.1: The result of compute device switching compatibility on each application

As shown in the table.5.1, Torta achieved devices switching on 87.5% of all applciations. Moreover, Torta achieved devices switching through any combination of devices with 6 of 8 applications. Even with 1 of 8 applications, Torta achieved device switching through any combination of devices on 40% of tested PCs.

To make our evaluation meaningful, we experimented with PCs that have highly complicated combination of devices; PCs that include all of the three(AMD's, Intel's and nVidia's) OpenCL platforms(A)(C)(E), PCs that include dGPUs and an iGPU with OpenCL capability(A)(D)(E), PCs that support cooperating mode between multiple GPUs(B)(D), PCs that include three GPUs(A)(B). They include most major types of combination of CPUs and GPUs on real world PCs. Moreover, combinations of devices on some of our PCs are rare; we seldom see such complicated hardware configurations. Therefore, the experimental result demonstrates that Torta can certainly switch devices on most of the real world PCs.

Below is the detail of experiments of each application such as switching options and experiment procedures.

DirectComputeBenchmark

Torta's options are following;

- TypeOverride: ALL
- InfoOverride: GPU
- DeviceNumberReduction: Disabled

Experiment procedure for DirectComputeBenchmark is as following;

1. Launch the application
2. Select “OpenCL” for API
3. Press “Benchmark”
4. Select the compute device(s)

Because of the application’s implementation, it tried to utilize multiple devices concurrently. Therefore, we assigned the same or different devices concurrently for our experiments. As a result, Torta achieved device switching on all tested machine with any combination of switching devices.

N-Queen Solver for OpenCL

Torta’s options are following;

- TypeOverride: ALL
- InfoOverride: Disabled
- DeviceNumberReduction: Disabled

Experiment procedure for N-Queen Solver for OpenCL is as following;

1. Launch the application by “nqueen_cl.exe 16”
2. Select the compute device

Actually, the application allows selecting a device by its own command-line options, by specifying a platform and a device type. However, to evaluate Torta, we just add the number of “N” size as command line option. As a result, Torta achieved device switching on all tested machine with any combination of switching devices.

As a side note, because of application’s compiling options, we could not launch the distributed binary with Torta’s AHM(*OpenCL.dll*). It crashed with MSVCRT R6030 [17] error. The application is implemented with MSVCRT of Visual C++ 2008 as *dynamic link*. Thus, it cannot work with Torta’s AHM which is built with MSVCRT of Visual C++ 2010 as static link. To avoid this OS side issue, we built the application with Visual C++ 2010 by own.

ratGPU

Torta’s options are following;

- TypeOverride: Disabled
- InfoOverride: Disabled

- DeviceNumberReduction: 1

Experiment procedure for ratGPU is as following;

1. Launch the application
2. Select the compute device (by Torta)
3. Check the device in the applications' window and press "Benchmark" button.

This application does not recognize HD Graphics 4000(Intel's iGPU) at all. Therefore, we exclude the condition that switching into HD Graphics 4000 from our experiment. As a result, Torta achieved device switching on all tested machines with any combination of switching devices.

WinZIP

We could not hook the OpenCL APIs on this application because the application does not load "OpenCL.dll". Our current implementation only support applications that load "OpenCL.dll" to call OpenCL APIs. Therefore, this result does not mean that Torta had problems in its device switching mechanism.

FLACCL

Torta's options are following;

- TypeOverride: Disabled
- InfoOverride: Disabled
- DeviceNumberReduction: Disabled

Experiment procedure for ratGPU is as following;

1. Launch the application by "CUETools.FLACCL.cmd.exe -o encoded.flac input.wav"
2. Select the compute device

This application is implemented as a .NET [48] application. Therefore, it calls OpenCL APIs via a .NET wrapper for OpenCL, "OpenCLNet.dll". Because of this difference, the result is also considerably different from others. Torta could switch devices only between GPUs in the first listed platform. When switching into a device in another platform, the application showed *KeyNotFoundException*. Because the "OpenCLNet.dll" manages device IDs by paring with platforms, it recognizes the incoherent information between devices. On the other hand, when switching into CPUs in the same platform, the application showed *INVALID_WORK_GROUP_SIZE* error on *clEnqueueNDRangeKernel()*. This problem is also caused by the incoherent information of devices. However, .NET strictly restricts changing the value of variables in the application from

lower level systems such as API hooking. Therefore, we could not make Torta compatible with this application.

Photoshop CS6 EXTENDED

We could not hook the OpenCL APIs on this application because the application does not load "OpenCL.dll". Our current implementation only support applications that load "OpenCL.dll" to call OpenCL APIs. Therefore, this result does not mean that Torta had problems in its device switching mechanism.

GIMP

Torta's options are following;

- TypeOverride: Disabled
- InfoOverride: Disabled
- DeviceNumberReduction: Disabled

Experiment procedure for GIMP is as following;

1. Launch the application
2. Select the compute device
3. Create a new image as 640x480
4. Apply a filter called "Plasma"
5. Apply "pixelize" in GEGL operation as 8x8

Because the installer of GIMP does not allow installing 32bit version of the application into 64bit OS environment, we experimented it with 32bit OS environment of PC(A). As a result, Torta achieved device switching with any combination of switching devices. Because GIMP is one of most famous applications, the result demonstrates that Torta actually enables device switching on real applications.

PhotoMonkee

Torta's options are following;

- TypeOverride: Disabled
- InfoOverride: Disabled
- DeviceNumberReduction: 1

Experiment procedure for PhotoMonkee is as following;

1. Launch the application
2. Select the compute device
3. Draw a simple picture

This application is not compatible with AMD's new OpenCL platform included in GPU drivers. Therefore, it always crashed on PCs with RADEON GPUs. However, Torta made the application runnable on PC(B)(A) by switching the device into a device in another platform. Torta could even improve the hardware compatibility by avoiding assignment into an incompatible device.

Overall, on PhotoMonkee, Torta achieved device switching on any combinations of devices on PC(C)(E), achieved switching on any combinations of devices except AMD platforms' devices on PC(B)(A).

vReveal 3

Because of application's compiling options, we could not launch the application with Torta's AHM(*OpenCL.dll*). It crashed with MSVCRT R6030 [17] error. The application is implemented with MSVCRT of Visual C++ 2008 as *dynamic link*. Thus, it cannot work with Torta's AHM which is built with MSVCRT of Visual C++ 2010 as static link. Because this is OS side issue, we exclude this application from our experimental targets.

x264 with OpenCL lookahead patch

Torta's options are following;

- TypeOverride: ALL
- InfoOverride: Disabled
- DeviceNumberReduction: 1

Experiment procedure for x264 with OpenCL lookahead patch is as following;

1. Launch the application by "x264_32_tMod+OreAQ-8bit-all.exe —opencl -o encoded.mp4 input.mp4"
2. Select the compute device

This application is not compatible with Intel's iGPU, HD Graphics 4000. Therefore, we experimented with switching devices into other devices. However, as a result, Torta could not switch into CPUs via AMD platform, on PCs that include RADEON(s). When we tried to switch the device into CPUs via AMD platform, the application showed *CL_INVALID_ARG_SIZE* error on *clSetKernelArg()*. This is also caused by incoherent information of devices. Nevertheless, Torta achieved device switching to any other devices on all tested PCs.

5.2.2 Overheads of Switching

To evaluate the overheads caused by device switching, we evaluated the execution time of *N-Queen Solver for OpenCL* with two conditions, with Torta and without Torta. Fig.5.1 shows the result of the performance difference between two conditions.

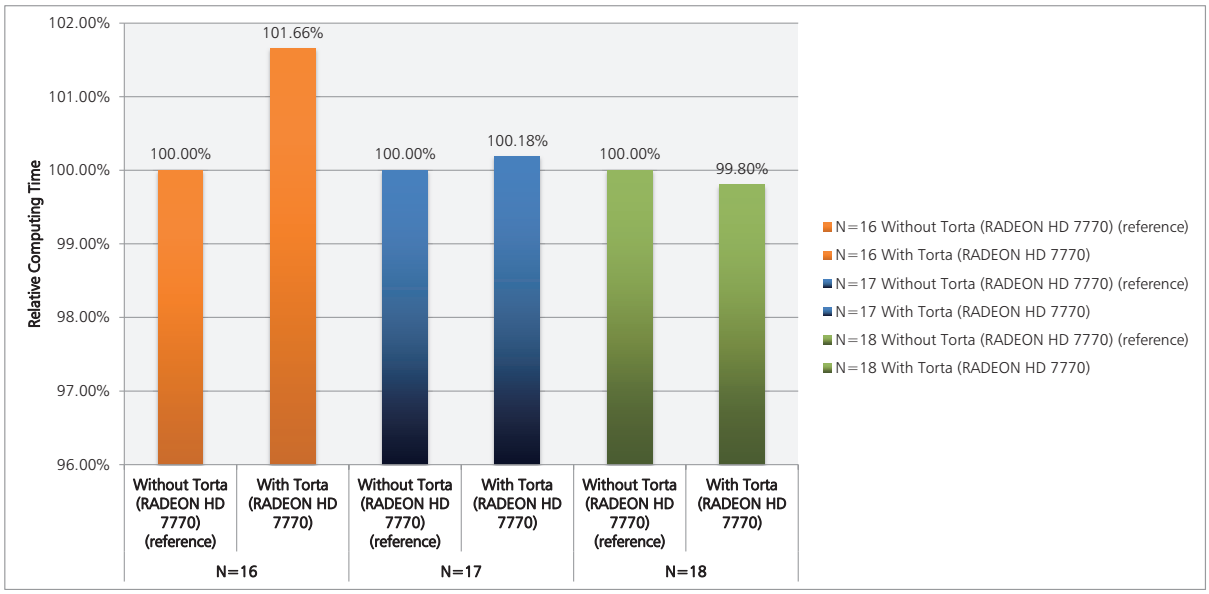

Figure 5.1: Result of Overhead Experiment using N-Queen Solver for OpenCL

We chose the test environment as PC(A), and chose RADEON HD 7700 GHz Edition as the target compute device. To measure the execution time, we used "Measure-Command" *cmdlet* of PowerShell [64]. For executing the program, the commands for each condition are following;

- With Torta: Measure-Command{.
  nqueen_cl.exe -platform 0 N} (N=16∼18)
- Without Torta: Measure-Command{.
  nqueen_cl.exe -platform 0 N} (N=16∼18)

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 experimented 5 times for each condition and calculated the average. In addition, we placed the files of *N-Queen Solver for OpenCL* and files of Torta into the RAMDisk created by Dataram RAMDisk [55].

Torta causes about 10ms to 30ms of overheads for handling devices' information, switching device IDs and other API wrapping. This overhead is not small when the actual processing time is short. However, it is definitely negligible when the processing time is long enough. In fact, 1.7% of overhead time when *N=16* decreased into 0.2% when *N=17*. Real applications utilize GPUs as GPGPU only for heavy processing. Overheads of data copying between main memory

and device memory spoils the performance advantage of GPUs on short time computation. Therefore, the ratio of Torta's overheads is usually small enough on real applications.

5.2.3 Performance Degradation

As stated in sec.3.6, Torta sometimes causes incoherence of devices' information. If applications gather devices' information before assigning a device, Torta cannot correct incoherent device's information on most cases. However, some applications optimize their processing adaptive to specification of the device, such as the number of compute units and hierarchy of device memory. Therefore, incoherence of device information may cause serious performance degradation.

Fig.5.2, 5.3, 5.4, 5.5, 5.6, 5.7 and 5.8 show the differences of performance between conditions; switching applications' devices with Torta and executing applications without Torta.

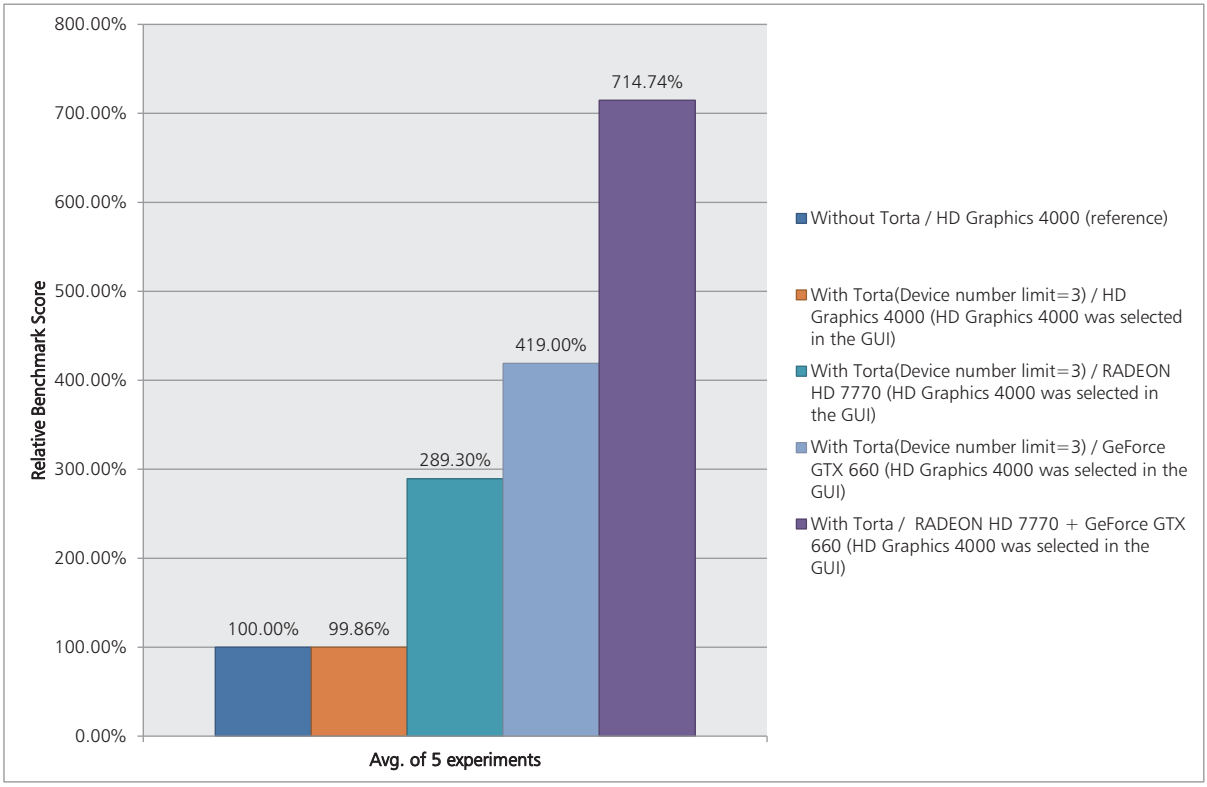

Figure 5.2: Result of performance change on DirectComputeBenchmark (1)

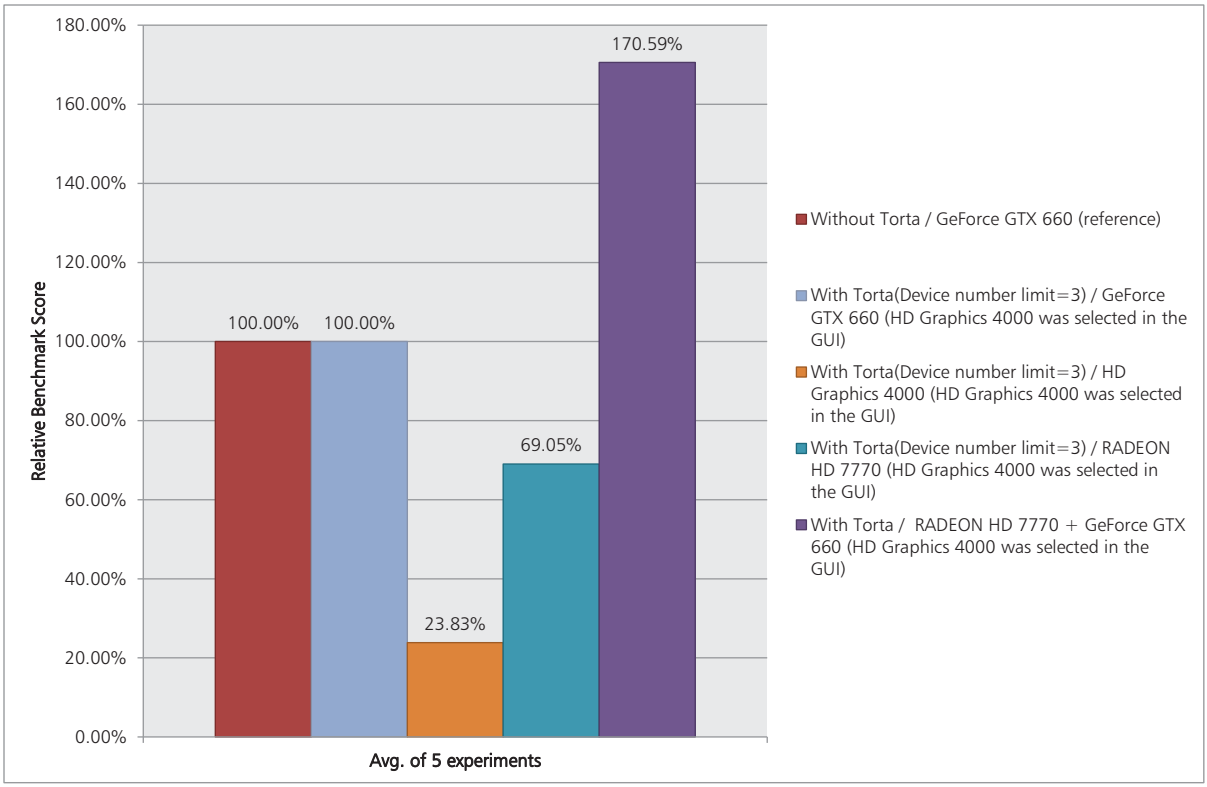

Figure 5.3: Result of performance change on DirectComputeBenchmark (2)

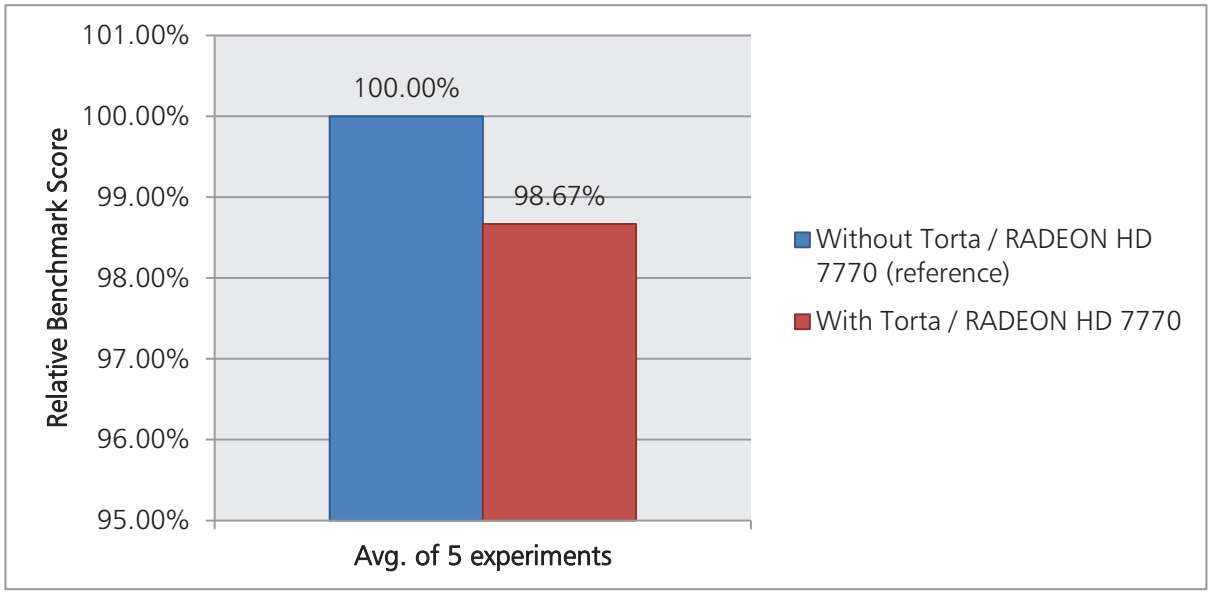

Figure 5.4: Result of performance change on ratGPU (1)

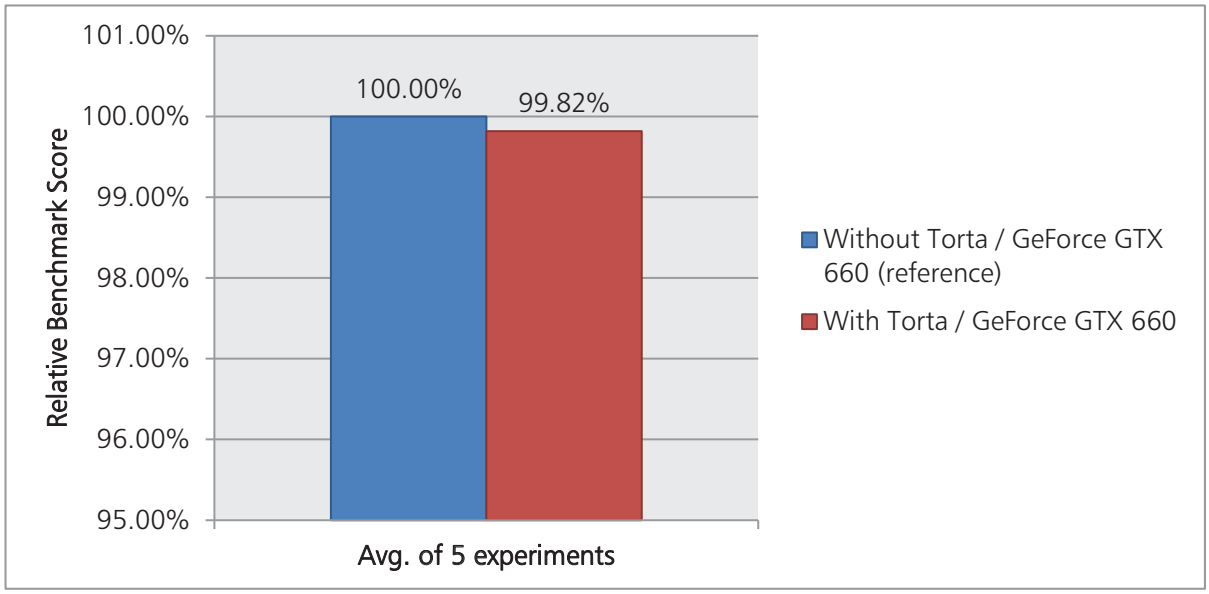

Figure 5.5: Result of performance change on ratGPU (2)

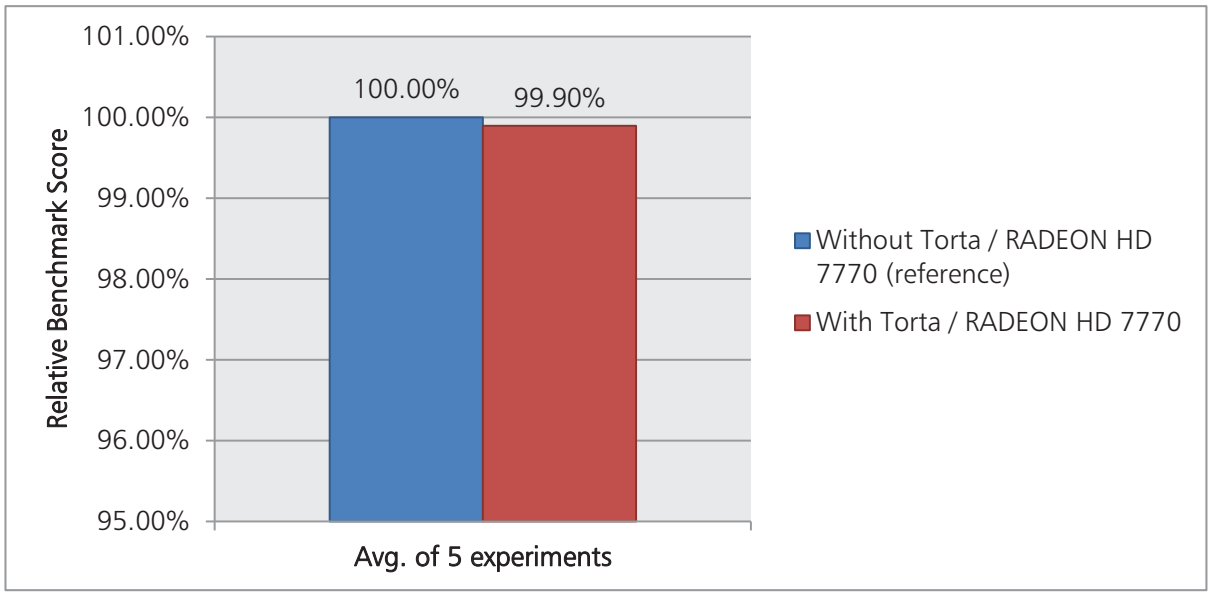

Figure 5.6: Result of performance change on LuxMark (1)

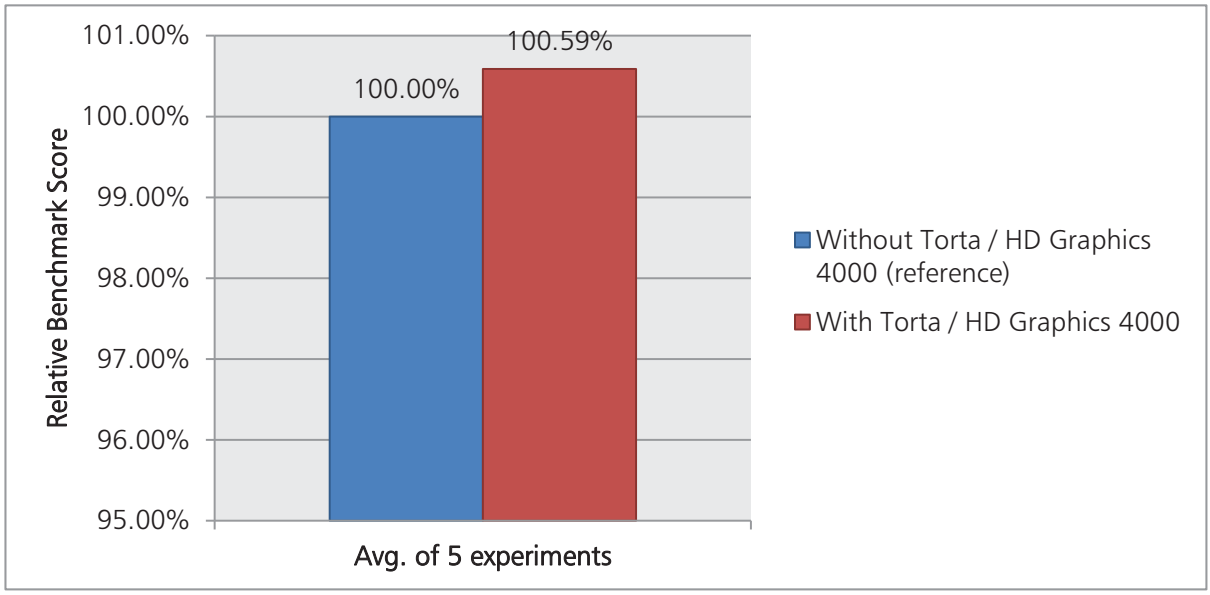

Figure 5.7: Result of performance change on LuxMark (2)

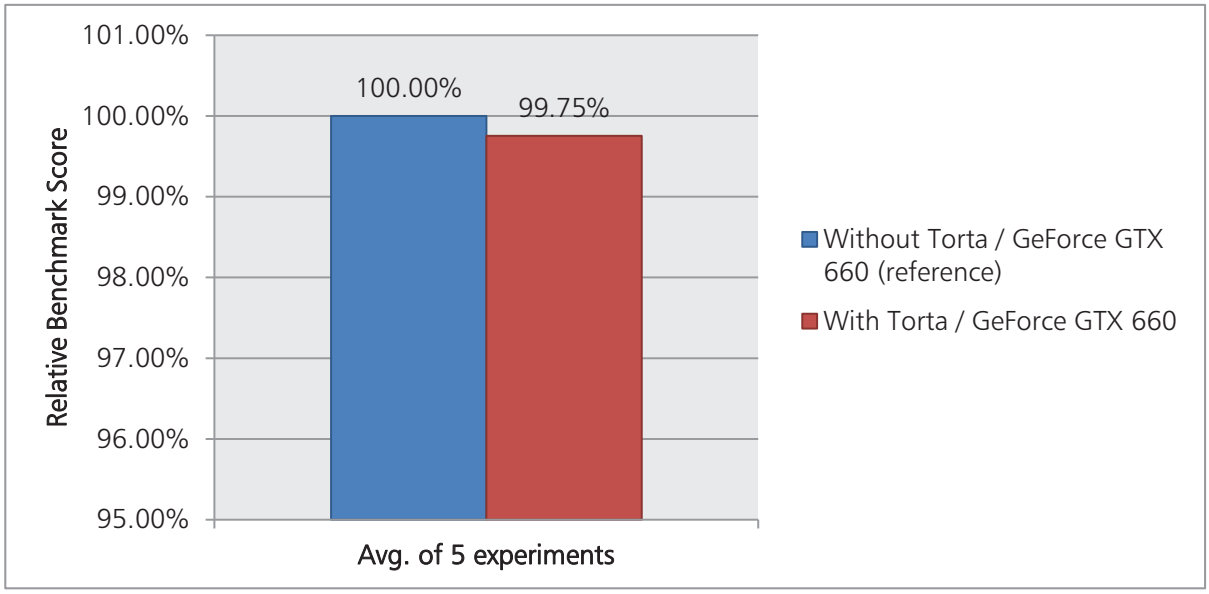

Figure 5.8: Result of performance change on LuxMark (3)

We chose the PC(A) for our experiments, and like experiments in sec.5.2.2, we placed files of target applications and Torta into the RAMDisk.

As a result, Torta caused 1.3% of performance penalty in the worst case and caused 0.20% of performance penalty in the average. Contrary to our expectations, performance degradations caused by Torta are negligible in most cases. Moreover, Torta improved benchmark scores on DirectComputeBenchmark. Torta does not only make possible for the application to use RADEON HD 7770 GHz Edition which is not recognized by DirectComputeBenchmark by

itself, it also makes possible to utilize two devices concurrently. The benchmark score with both RADEON HD 7770 GHz Edition and GeForce GTX 660 reaches more than 7 times of the score of HD Graphics 4000.

We excluded command line applications (*e.g.*x264, FLACCL and N-Queen Solver for OpenCL) from this experiment because they usually have the capability to select a device by their command line options. In addition, these applications are executed for each a transaction. FLACCL, for instance, the application is executed for each encoding of audio files. Therefore, creating a launcher application is a better solution than API hooking for such applications. The performance penalty of a launcher should be smaller than API hooking.

5.3 Qualitative Evaluation

In this section, we state policy and methods of qualitative evaluation. Firstly, we evaluated the Torta's capability of avoiding decrement of user experience. Secondly, we compared our approach with existing approaches.

5.3.1 Avoiding Decrement of User Experience

As stated in sec.2.3, a user's feeling is strongly depending on the user. Hence, we cannot easily evaluate degradation of the user experience objectively.

To evaluate the degradation of user experience as simulation, we measured the FPS of video playback by executing other GPU accelerated applications in both condition; with Torta and without Torta. While the FPS keeps its real-time, the video playback never degrades the user experience even when other applications utilize GPUs concurrently.

Fig.5.9 shows FPS of the video playback in each condition.

As experiment environment, we chose the PC(A) with Windows x64 and Splash PRO EX as the video playback application. Applications additionally executed were N-Queen Solver for OpenCL, DirectComputeBenchmark and LuxMark. As shown in fig.5.9, with Torta, FPS of the video playback keeps its real-time even when three of additional applications were executed. On the other hand, without Torta, the FPS of video playback decreased to about 46 even if two additional applications were executed. The FPS decreased more according to the number of additional applications; When three additional applications were executed, the FPS decreased to about 14.

According to the result, we can sate following two points as the effect of Torta.

- Torta is not worse than nothing
  The FPS of video playback with Torta was always the same or more than the FPS of experiments without Torta. If assuming that higher FPS never causes lower user experience than lower FPS, it can be said that Torta never causes lower user experience than envi-

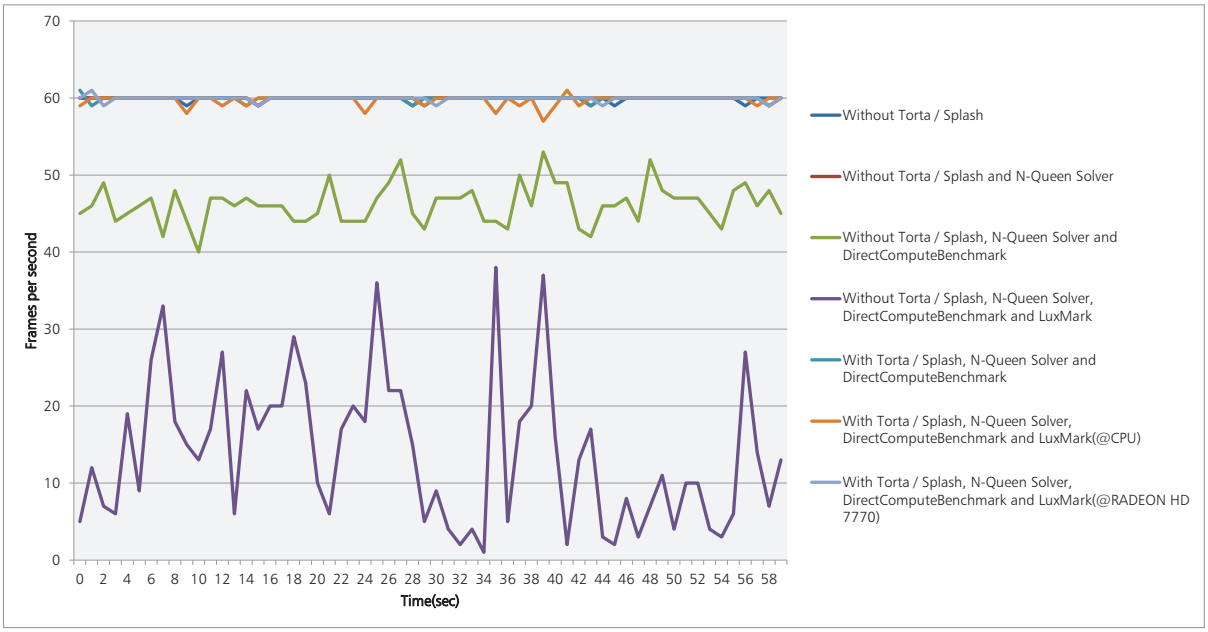

Figure 5.9: Result of Video Playback FPS During Multiple GPU Accelerated Application Working

ronments without Torta. Even if the assumption is not always true, Torta never causes lower user experience than environments without Torta while Torta keeps real-time FPS by avoiding FPS decrement.

- Torta brings better user experience in some specific cases
  People sometimes notice frame dropping when the FPS of video playback decreases. In other words, there IS a threshold of noticing frame dropping on each user for each video clip. Therefore, it can be said that Torta achieves better user experience when Torta avoids decrement of FPS and increase it to above the threshold.

5.3.2 Approaches Comparison

Our experiments show that Torta moderately works as expected. The results also demonstrate that our approach is deployable and moderately effective. Thus, we compared our approach with other existing approaches lastly.

Table.5.2 shows the difference between OS modification approach, driver modification approach, original library approach and our approach. According to this comparison, Torta is said an actually deployable solution for environments that strongly need compatibility.

When compared to OS modification approach, Torta has a disadvantage of performance overheads. Granularity of resource management is also a disadvantage of Torta. Torta only manages resources of compute devices by distributing tasks between processors; it does not manage resource of GPUs. Specification of OpenCL also limits the granularity of task distribu-

	OS Modification	Driver Modification	Original Library	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

Table 5.2: Comparison between OS modification approach, driver modification approach, original library approach and our approach

tion. On some applications, Torta can switch devices only once, right after the application has launched. However, its short term deployability is a definite advantage of Torta. Our experimental results show Torta's outstanding compatibility with existing real applications. Torta is enough practical from this point of view.

When compared to driver modification approach, Torta has a disadvantage of overheads, the number of target applications and granularity of resource management. The driver modification approach can theoretically deal with all tasks run on GPUs. Moreover, it can manage GPUs in small granularity. However, it does not distribute tasks between compute devices; driver modification approach and our approach are complementary rather than competitive.

When compared to original library approach, Torta has a disadvantage of granularity of task distribution. Because Torta extends OpenCL, which is one of existing platforms, granularity of task distribution is limited. However, granularity of task distribution and applications' compatibility are trade-off. Compatibility with existing real applications is a great advantage of Torta.

Chapter 6

Conclusion

In this thesis, we proposed **Torta** that achieves centralized processors assignment on real applications for PCs.

Because Torta switches compute devices(almost same as processors) by hooking OpenCL APIs, it achieved devices switching on 87.5% of real applications on our experiments. Although its high compatibility, it only causes 0.20% of performance penalty in the average, 1.3% in the worst case. In addition, its centralized devices assignment model achieved prevention of user-experience decrement by preventing the performance degradation on important applications for users. On our experiments, Torta protected the frame rate of video playback as 60FPS(real-time) even when four of additional GPU accelerated applications tried to utilize GPUs concurrently.

Our current implementation only supports OpenCL applications. However, our concept is also applicable to other processors abstraction frameworks. In addition, it is even applicable to applications that do not use such frameworks. For instance, on Firefox [46], one of most famous web browsers, we can enable/disable GPU acceleration mode by modifying "prefs.js" file with only 2 lines. Although Firefox does not allow changing acceleration mode during the application is running, we can change the acceleration mode before its execution. By creating a launcher application that communicates with the centralized resource manager and modifies the setting file, we can apply our concept into other applications like Firefox.

As our future work, we plan to complete our system by following works; to reduce overheads of our system; to make a list of methods for acquiring PC usage contexts; to design the description format of PC usage contexts and user preference; and to design a user friendly interface for expressing a user's preference.

Acknowledgment

本研究に取り組むにあたり，主査の慶應義塾大学環境情報学部教授徳田英幸博士には大学 2 年次の研究会所属以降，大変お世話になりました．本論文の執筆においても，技術的側面から論文の構成方法に至るまで多大なるご助言を頂きました．この場を借りて，深く感謝の意を表します．また，副査の環境情報学部専任講師中澤仁博士には，本論文執筆において研究の方向性設定から研究の進め方に至るまで，多くの具体的ご指導を賜りました．大変深く感謝しております．また，研究室所属のきっかけを下さり，学部在籍時より様々なご助言を下さった環境情報学部准教授高汐一紀博士に，ここで深く感謝を申し上げたいと存じます．

大学院にて研究を遂行するにあたり，研究室 OB の本多倫夫博士には対外発表に向けた論文のご指導など，様々な面でお世話になり，深く感謝しております．また，研究室 OB の米澤拓郎博士並びに環境情報学部講師榊原寛氏には本研究テーマの決定から取りかかりに至るまで多大なご助言を頂き，大変感謝しております．加えて，研究室において先輩として様々なご助言を下さいました伊藤友隆氏，生天目直哉氏に感謝申し上げます．また，日々の研究活動を共に過ごした，同輩の星北斗氏，蛭田慎也氏，瀧本拓哉氏，並びに後輩の加藤碧氏，伊藤瑛氏，八木彩香氏，安形憲一氏，鈴木幸大氏，坂村美奈氏，櫻井理央氏には，普段より様々なご協力を頂きました．他の研究室メンバーを含め，日々励まし合いながら研究活動に取り組むことが出来る環境に恵まれたことに，心より感謝しております．

本論文執筆のテンプレートをご作成下さった山本伶氏並びに田中佑樹氏には，この場を借りて感謝の意を表します．拙いながらも論文としての体裁を整えることが出来たのは，両氏のお力添えあってのことです．

最後に，互いに励まし合いながら修士論文に取り組んだ同輩の星北斗氏，蛭田慎也氏，山本伶氏，秋山博紀氏，白崎琢也氏，遠藤忍氏，大下和希氏，そして日々気の利いた言葉を掛けて下さった米持愛美氏の皆様に深い感謝の意を表して，本論文の謝辞と致します．

Reference

[1] 3rd Generation Intel Core Processor Desktop Product Brief. http://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/3rd-gen-core-desktops-brief.pdf.

[2] Accelerated Parallel Processing (APP) SDK. http://developer.amd.com/tools/heterogeneous-computing/amd-accelerated-parallel-processing-app-sdk/downloads/download-archive/.

[3] Adobe Photoshop CS6 Extended. http://www.adobe.com/products/photoshopextended.html.

[4] AMD A-Series Processor Model Number and Feature Comparisons. http://www.amd.com/us/products/desktop/processors/a-series/Pages/a-series-model-number-comparison.aspx.

[5] AMD Catalyst Display Driver for Windows Vista/Windows 7. http://support.amd.com/us/gpudownload/windows/Pages/radeonaiw_vista64.aspx, http://support.amd.com/us/gpudownload/windows/Pages/radeonaiw_vista32.aspx.

[6] AMD Cool'n'Quiet Technology. http://www.amd.com/us/products/technologies/cool-n-quiet/Pages/cool-n-quiet.aspx.

[7] AMD CrossFire. http://sites.amd.com/us/game/technology/Pages/crossfirex.aspx.

[8] AMD FX Model Number Comparison. http://www.amd.com/us/products/desktop/processors/amdfx/Pages/amdfx-model-number-comparison.aspx.

[9] Amd phenom ii key architectural features. http://www.amd.com/us/products/desktop/processors/phenom-ii/Pages/phenom-ii-key-architectural-features.aspx.

[10] AMD Radeon Dual Graphics. http://www.amd.com/us/products/technologies/dual-graphics/Pages/dual-graphics.aspx.

[11] AMD Radeon HD 6700M/HD 6600M Series Graphics. http://www.amd.com/us/products/notebook/graphics/amd-radeon-6000m/amd-radeon-6700m-6600m/pages/amd-radeon-6700m-6600m.aspx.

[12] AMD Radeon HD 6970 Graphics. http://www.amd.com/us/products/desktop/graphics/amd-radeon-hd-6000/hd-6970/Pages/amd-radeon-hd-6970-overview.aspx.

[13] AMD Radeon HD 7770 GHz Edition. http://www.amd.com/us/products/desktop/graphics/7000/7770/pages/radeon-7770.aspx.

[14] Apple - Rosetta. http://www.apple.com/asia/rosetta/.

[15] ASUS ZENBOOK UX32VD-R4SSHS. http://www.asus.co.jp/Notebooks/Superior_Mobility/ASUS_ZENBOOK_UX32VD/.

[16] C++ AMP : Language and Programming Model. http://download.microsoft.com/download/4/0/E/40EA02D8-23A7-4BD2-AD3A-0BFFFB640F28/CppAMPLanguageAndProgrammingModel.pdf.

[17] C Run-Time Error R6030. http://msdn.microsoft.com/en-us/library/9ecfyw6c.aspx.

[18] Creative ALchemy. http://www.creative.com/soundblaster/alchemy/.

[19] CUDA. http://www.nvidia.com/object/cuda_home_new.html.

[20] DirectCompute & OpenCL Benchmark. http://www.ngohq.com/graphic-cards/16920-directcompute-and-opencl-benchmark.html.

[21] DirectWrite (Windows). http://msdn.microsoft.com/en-us/library/dd368038.aspx.

[22] DirectX. http://www.microsoft.com/japan/directx/default.mspx.

[23] DirectX Video Acceleration (Windows Drivers). http://msdn.microsoft.com/en-us/library/ff553861.aspx.

[24] EasyHook - The reinvention of Windows API Hooking. http://easyhook.codeplex.com/.

[25] FLAC - Free Lossless Audio Codec. http://flac.sourceforge.net/.

[26] FLACCL. http://www.cuetools.net/wiki/FLACCL.

[27] GeForce GT 555M. http://www.geforce.com/hardware/notebook-gpus/geforce-gt-555m.

[28] GeForce GT 620M. http://www.geforce.com/hardware/notebook-gpus/geforce-gt-620m.

[29] GeForce GTX 550 Ti. http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-550ti.

[30] GeForce GTX 660. http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-660.

[31] GeForce SLI. http://www.geforce.com/hardware/technology/sli.

[32] GIMP. http://gimp-win.sourceforge.net/.

[33] H.264 : Advanced video coding for generic audiovisual services. http://www.itu.int/rec/T-REC-H.264.

[34] Hybrid SLI technology. http://www.nvidia.com/object/hybrid_sli.html.

[35] Intel Core i7-2760QM Processor (6M Cache, up to 3.50 GHz). http://ark.intel.com/products/53474/Intel-Core-i7-2760QM-Processor-6M-Cache-up-to-3_50-GHz.

[36] Intel Core i7-3517U Processor (4M Cache, up to 3.00 GHz). http://ark.intel.com/products/65714/intel-core-i7-3517u-processor-4m-cache-up-to-3_00-ghz.

[37] Intel Core i7-3770K Processor (8M Cache, up to 3.90 GHz). http://ark.intel.com/products/65523/.

[38] Intel HD Graphics Delivers Incredible Visuals. http://www.intel.com/content/www/us/en/architecture-and-technology/hd-graphics/hd-graphics-developer.html.

[39] Intel HD Graphics Driver for Windows* 7. http://downloadcenter.intel.com/Detail_Desc.aspx?lang=eng&changeLang=true&DwnldId=21476.

[40] Intel Quick Sync Video. http://www.intel.com/content/www/us/en/architecture-and-technology/quick-sync-video/quick-sync-video-general.html.

[41] Intel SDK for OpenCL* Applications 2012. http://software.intel.com/en-us/vcsource/tools/opencl-sdk.

[42] Intel Turbo Boost Technology - On-Demand Processor Performance. http://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html.

[43] LuxMark. http://www.luxrender.net/wiki/LuxMark.

[44] Microsoft Visual Studio 2010 Ultimate. http://www.microsoft.com/japan/visualstudio/products/2010-editions/ultimate.

[45] MinHook - The Minimalistic x86/x64 API Hooking Library. http://www.codeproject.com/Articles/44326/MinHook-The-Minimalistic-x86-x64-API-Hooking-Libra.

[46] Mozilla Firefox Web Browser. http://www.mozilla.org/en-US/firefox/new/.

[47] N-Queen Solver for OpenCL. http://forum.beyond3d.com/showthread.php?t=56105.

[48] .NET Framework. http://msdn.microsoft.com/en-us/vstudio/aa496123.

[49] Notebooks With AMD Accelerated Processors. http://www.amd.com/us/products/notebook/apu/mainstream/Pages/mainstream.aspx.

[50] NVIDIA DRIVERS 306.97WHQL. http://www.nvidia.com/object/win8-win7-winvista-64bit-306.97-whql-driver.html.

[51] OpenCL 1.2 Specification. http://www.khronos.org/registry/cl/specs/opencl-1.2.pdf.

[52] Opengl. http://www.opengl.org/.

[53] PhotoMonkee. http://photomonkee.com/.

[54] Previous AMD Catalyst Display Drivers. http://support.amd.com/us/gpudownload/windows/previous/12/Pages/radeon.aspx?os=Windows%20Vista%20-%2064-Bit%20Edition&rev=12.6.

[55] RAMDisk - Software - Server Memory Products & Services - Dataram. http://memory.dataram.com/products-and-services/software/ramdisk.

[56] ratGPU. http://www.ratgpu.com/.

[57] Splash PRO EX. http://mirillis.com/en/products/splashexport.html.

[58] The OpenACC Application Programming Interface. http://www.openacc.org/sites/default/files/OpenACC.1.0_0.pdf.

[59] VideoLAN - x264, the best H.264/AVC encoder. http://www.videolan.org/developers/x264.html.

[60] vReveal Video Enhancement Software. http://www.vreveal.com/.

[61] What is Windows Aero? http://windows.microsoft.com/en-US/windows-vista/What-is-Windows-Aero.

[62] Windows 7. https://www.microsoft.com/japan/windows/windows-7/default.aspx.

[63] Windows 8 and Windows RT - Microsoft Windows. http://windows.microsoft.com/en-US/windows-8/meet.

[64] Windows PowerShell. http://technet.microsoft.com/en-us/library/bb978526.aspx.

[65] WinZip. http://www.winzip.com/win/en/index.htm.

[66] x264 rev2230+696 tMod / avs4x264mod 0.9.0. http://astrataro.wordpress.com/2012/11/11/x264-rev2230696-tmod-avs4x264mod-0-9-0/.

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

[68] Ryo Aoki, Shuichi Oikawa, Ryoji Tsuchiyama, and Takashi Nakamura. Hybrid OpenCL: Connecting Different OpenCL Implementations over Network. In *Proc. IEEE CIT 2010*, pp. 2729–2735, 2010.

[69] C 辿 dricAugonnet, Samuel Thibault, Raymond Namyst, Pierre-Andr 辿 Wacrenier. StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation: Practice and Experience*, Vol. 23, No. 2, pp. 187–198, 2011.

[70] Mikhail Bautin, Ashok Dwarakinath, and Tzi-cker Chiueh. Graphic engine resource management. 2008.

[71] P. Boudier and G. Sellers. Memory system on fusion APUs: The benefits of zero copy. In: AMD Fusion developer summit, AMD. http://developer.amd.com/afds/assets/presentations/1004_final.pdf.

[72] Wei-Nien Chen and Hsueh-Ming Hang. H.264/AVC motion estimation implmentation on Compute Unified Device Architecture (CUDA). In *Multimedia and Expo, 2008 IEEE International Conference on*, pp. 697 –700, 23 2008-april 26 2008.

[73] Gregory F. Diamos and Sudhakar Yalamanchili. Harmony: an execution model and runtime for heterogeneous many core systems. In *Proc. ACM HPDC*, pp. 197–200, 2008.

[74] James Fung and Steve Mann. OpenVIDIA: parallel GPU computer vision. In *Proceedings of the 13th annual ACM international conference on Multimedia*, MULTIMEDIA '05, pp. 849–852, New York, NY, USA, 2005. ACM.

[75] V. Gupta, K. Schwan, N. Tolia, V. Talwar, and . Ranganathan. Pegasus: Coordinated Scheduling for Virtualized Accelerator-based Systems. In *Proc. USENIX ATC*, pp. 31–44, 2011.

[76] Vishakha Gupta, Ada Gavrilovska, Karsten Schwan, Harshvardhan Kharche, Niraj Tolia, Vanish Talwar, and Parthasarathy Ranganathan. GViM: GPU-accelerated virtual machines. In *Proceedings of the 3rd ACM Workshop on System-level Virtualization for High Performance Computing*, HPCVirt '09, pp. 17–24, New York, NY, USA, 2009. ACM.

[77] Owen Harrison and John Waldron. AES Encryption Implementation and Analysis on Commodity Graphics Processing Units. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007*, Vol. 4727 of *Lecture Notes in Computer Science*, pp. 209–226. Springer Berlin Heidelberg, 2007.

[78] Tetsuro Horikawa, Michio Honda, Jin Nakazawa, Kazunori Takashio, and Hideyuki Tokuda. PACUE: Processor Allocator Considering User Experience. In *Euro-Par 2011: Parallel Processing Workshops*, Vol. 7156 of *Lecture Notes in Computer Science*, pp. 335–344. Springer Berlin / Heidelberg, 2012.

[79] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, pp. 559–568, New York, NY, USA, 2011. ACM.

[80] Víctor J. Jiménez, Lluís Vilanova, Isaac Gelado, Marisa Gil, Grigori Fursin, and Nacho Navarro. Predictive Runtime Code Scheduling for Heterogeneous Architectures. In *HiPEAC '09: Proceedings of the 4th International Conference on High Performance Embedded Architectures and Compilers*, pp. 19–33, Berlin, Heidelberg, 2009. Springer-Verlag.

[81] S. Kato, K. Lakshmanan, Y. Ishikawa, and R. Rajkumar. Resource Sharing in GPU-Accelerated Windowing Systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE*, pp. 191 –200, april 2011.

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

[83] S. Kato, M. McThrow, C. Maltzahn, and S. Brandt. Gdev: First-Class GPU Resource Management in the Operating System. In *Proc. of the USENIX Annual Technical Conference*, 2012.

[84] Kenneth Lee, Heshan Lin, and Wu-chun Feng. Performance characterization of data-intensive kernels on AMD Fusion architectures. *Computer Science - Research and Development*, pp. 1–10, 2012.

[85] Lucidlogix. Virtu GPU Virtualization Software. `http://www.lucidlogix.com/product-virtu-gpu.html`.

[86] Chi-Keung Luk, Sunpyo Hong, and Hyesoon Kim. Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pp. 45 –55, dec. 2009.

[87] S.A. Manavski. CUDA Compatible GPU as an Efficient Hardware Accelerator for AES Cryptography. In *Signal Processing and Communications, 2007. ICSPC 2007. IEEE International Conference on*, pp. 65 –68, nov. 2007.

[88] P. Michel, J. Chestnut, S. Kagami, K. Nishiwaki, J. Kuffner, and T. Kanade. GPU-accelerated real-time 3D tracking for humanoid locomotion and stair climbing. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pp. 463 –469, 29 2007-nov. 2 2007.

[89] Microsoft. CreateRemoteThread Function (Windows). `http://msdn.microsoft.com/en-us/library/ms682437.aspx`.

[90] Microsoft. SetWindowsHookEx Function (Windows). `http://msdn.microsoft.com/en-us/library/ms644990.aspx`.

[91] Microsoft Research. Detours - Microsoft Research. `http://research.microsoft.com/en-us/projects/detours/`.

[92] nVidia. Optimus Technology. `http://www.nvidia.com/object/optimus_technology.html`.

[93] R. Ozaydin and D.T. Altilar. OpenCL Remote: Extending OpenCL Platform Model to Network Scale. In *High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*, pp. 830 –835, june 2012.

[94] Christopher J. Rossbach, Jon Currey, Mark Silberstein, Baishakhi Ray, and Emmett Witchel. PTask: operating system abstractions to manage GPUs as compute devices. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pp. 233–248. ACM, 2011.

[95] Jan H. Schoenherr, Jan Richling, Gero Muehl, and Matthias Werner. A Scheduling Approach for Efficient Utilization of Hardware-Driven Frequency Scaling. *Architecture of Computing Systems (ARCS), 2010 23rd International Conference on*, pp. 1 –10, feb. 2010.

[96] Lin Shi, Hao Chen, Jianhua Sun, and Kenli Li. vCUDA: GPU-Accelerated High-Performance Computing in Virtual Machines. *Computers, IEEE Transactions on*, Vol. 61, No. 6, pp. 804 –816, june 2012.

[97] Hiroyuki Takizawa, Katsuto Sato, and Hiroaki Kobayashi. SPRAT: Runtime processor selection for energy-aware computing. In *Proceedings of the 2008 IEEE International Conference on Cluster Computing, 29 September - 1 October 2008, Tsukuba, Japan*, pp. 386–393, 2008.