PACUE: Processor Allocator Considering User Experience

Tetsuro Horikawa[1], Michio Honda[1], Jin Nakazawa[2], Kazunori Takashio[2], and Hideyuki Tokuda[2,3]

[1] Graduate School of Media and Governance, Keio University
[2] Faculty of Environment and Information Studies, Keio University, 5322, Endo, Fujisawa, Kanagawa 252-8520, Japan
[3] JST-CREST, Japan
{techi,jin,kaz,hxt}@ht.sfc.keio.ac.jp,
micchie@sfc.wide.ad.jp

Abstract. GPU accelerated applications including GPGPU ones are commonly seen in modern PCs. If many applications compete on the same GPU, the performance will decrease significantly. Some applications have a large impact on user experience. Therefore, for such applications, we have to limit GPU utilization by the other applications. It might be straightforward to modify applications to switch compute device dynamically for intelligent resources allocation. Unfortunately, we cannot do so due to software distribution policy or the other reasons. In this paper, we propose PACUE, which allows the end system to allocate compute devices arbitrary to applications. In addition, PACUE guesses optimal compute device for each application according to user preference. We implemented the dynamic compute device redirector of PACUE including OpenCL API hooking and device camouflaging features. We also implemented the frame of the resource manager of PACUE. We demonstrate PACUE achieves dynamic compute device redirecting on one out of two real applications and on all of 20 sample codes.

Keywords: Resource management, OpenCL, binary compatibility, GPU, GPGPU, PC, user experience.

1 Introduction

Graphics Processing Unit (GPU) use has been extended to a wider range of computing purposes on the PC platform. GPU utilization purposes on PCs can be classified into four purposes. The first is 3D graphics computation, such as 3D games and 3D-graphics-based GUI shell (*e.g.*, Windows Aero). The second is 2D graphics acceleration, such as font rendering in modern web browsers. The third is video decoding and encoding acceleration. Video player applications use the video decoding acceleration function of the GPU to reduce CPU load and to increase the video quality. Also, some of GPUs have video encoding acceleration units on the die of the GPU.The last purpose is general-purpose computing, called General-Purpose computing on GPU (GPGPU). On PCs, GPGPU is often used by video encoding applications and physics simulation applications including 3D games.[1]

[1] Some 3D games utilize GPU for general-purpose computing besides 3D graphics rendering.

In today's PCs GPUs are utilized efficiently, because only a few of the applications are accelerated at the same time; these applications do not compete each other on the same GPU. Applications thus choose compute devices statically, such as by user selection in the application configuration menu of the GUI interface.

However, we envisage that more and more applications utilize GPUs. For example, Open Computing Language (OpenCL) [2] allows applications to select the compute device explicitly to execute some parts of the application. Therefore, efficient load balancing between compute devices consisting of CPUs and GPUs is essential for future consumer PCs.

There are three technical challenges to achieve efficient compute device assignment of heterogeneous processors in PCs. First, GPU acceleration is utilized for various purposes, while GPUs are utilized mainly for general-purpose computing in super computers. In addition, some of tasks running in PCs strongly require specific processors. For example, 3D rendering is normally processed by GPUs, and some of 3D graphics transactions cannot be processed by CPUs, whereas some applications can be processed by both CPUs and GPUs. When the GPU load is high, we could run the latter applications explicitly on CPUs.

Second, we must not modify applications. Typically, most of applications installed in major OSes such as Windows and Mac OS cannot be modified by a third person, due to their software distribution policies. Application vendors may not be willing to modify their applications either, because it will not benefit them straightforwardly. For these reasons, existing runtime libraries or libraries to distribute tasks between compute devices [6, 10, 7] proposed for HPC are not deployable on consumer PCs.

Third, performance metric for consumer PCs is complicated, because user preference is one of the most important metrics for assigning compute devices to applications. It is clearly different from general HPC's metrics whose task distributing policy is usually static, such as maximizing task transaction speed or maximizing performance per watt. In PCs, task distributing policies and merits easily change depending on the use. For example, when the user would like to play the 3D game smoothly, the other GPGPU tasks should not be assigned to the GPU. On the other hand, sometimes the user might be willing to transcode videos quickly rather than playing the trifling game smoothly. The compute device selecting method must recognize user preferences to decide the proper compute device to assign. However this is hard, thus user preference recognizing cannot automate. Therefore, the resource management has to infer PC utilization and the users have to be able to tell how they are using PC at that time.

In this paper, we propose PACUE which allocates compute devices to applications efficiently. PACUE has two features, one is dynamic compute device redirecting feature and the other is system-wide optimal device selecting feature. We strongly focus on solving real problems which will occur when we distribute our system over the world via web. Therefore, we prefer choosing politically safer method rather than technically better method. Thus, first advantage of PACUE is the possibility of the deployment. The second advantage of PACUE is designed to maximize PC users' experience. Thus, we bring a new metric for using accelerators, and it will be also beneficial for other computers such as smart phones or game consoles.

Our experimental results show that PACUE can switch compute devices in 1 out of 2 applications, and all of 20 sample codes built with OpenCL. The reminder of this paper is organized as follows: In Sec. 2, we describe the design of PACUE consisting of the dynamic compute device redirecting and the system resource manager. In Sec. 3, we evaluate our prototype implementation. The paper concludes with Sec. 4.

2 Designing PACUE

PACUE is constructed by two components; Dynamic Compute Device Redirector and Resource Manager. We focus on applications built with OpenCL, a widely used framework which supports many types of compute devices such as CPUs and GPUs.

2.1 Dynamic Compute Device Redirection

We design the Dynamic Compute Device Redirection (DCDR) method to meet the "no application modification" requirement. DCDR implements OpenCL API hooking that conceals actual compute devices from applications, and avoids error caused by inconsistent information of devices.

OpenCL API Hooking. OpenCL abstracts compute devices and memory hierarchy to utilize heterogeneous processors within its programming model. To utilize a compute device, applications call OpenCL APIs and specify a compute device. Assigning process are following: Secondly, select possible devices and create an OpenCL context. Thirdly, select one device to use and create a command queue. Lastly, put tasks to the queue created above. In the second and the third steps, the application specifies a concrete device because OpenCL APIs needs device ID as its parameter, which makes system-wide optimal device selection impossible. For optimal device selection, we remove the restriction that the applications need to choose the device by itself because the decision is hard for applications and users. However, decisions by applications or users are rarely optimal (See Sec. 2.2). PACUE hooks a part of OpenCL APIs which concern device selecting, and implements asking function that asks which device to utilize.

There are several methods to hook APIs in Windows 7 where PACUE is implemented. The first possibility is making a thread in the target application by calling a Windows API *CreateRemoteThread()* [12]. With this method we implement an application which make a thread in other applications and map external DLL containing overridden target APIs. However, these applications and DLLs are hard to implement due to complicated procedures. It has a risk being treated as malware by the anti-malware software. The second possibility is Global Hook, the user application hooks specific APIs of all application by calling Windows API *SetWindowsHookEx()* [13]. This method is unsafe, because it has a risk of hooking unknown applications and causing unexpected affect to them. The third possibility is making Wrapper DLL, which is a DLL with the same file name of original DLL and has all APIs of original DLL. Wrapper DLL is almost shell of original DLL, because most APIs are simply calls original DLL APIs except APIs which actually need to do different transaction from original. This method has the most chance of hooking APIs, because wrapper DLL located in the application directory is always loaded prior to the other ones, such as DLLs located in system

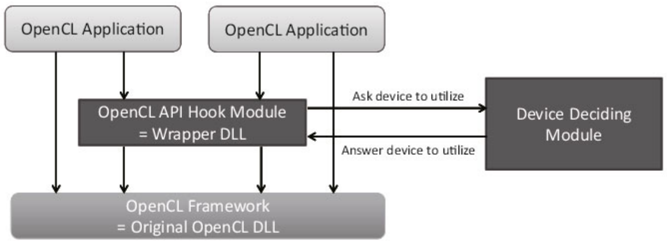

Fig. 1. Dynamic Compute Device Switching by OpenCL API Hooking

directories by default. In addition, when locating wrapper DLL in the directory which target EXE located, only affects applications whose binary is located in the same directory. Therefore, this is really safe way to hook APIs. The last possibility is the use of API hook libraries, such as [14]. These libraries are easy to use, however it has less probability to success to hook APIs than Wrapper DLL. It also has a risk to be treated as malware. From this comparison, we adopt the Wrapper DLL method. Fig. 1 illustrates the architecture to hook OpenCL APIs with this method. Other major PC OSes such as MacOS or Linux do not provide any function like wrapper DLLs, still we can implement a similar system by using API hooking functions offered by other OSes.

Another method to switch devices is making a virtual device. [5] On this method, applications will assign the virtual device and the resource management system choose a real device. This method has a significant advantage that it can switch real devices at any time, however it may conflict with Installable Client Driver(ICD) system of OpenCL. Installer of OpenCL runtime libraries distributed by hardware vendors sometimes overwrite “OpenCL.dll” file, thus installing a virtual device or showing applications only the virtual device is difficult on PCs.

Device Information Camouflaging. When a part of applications’ tasks are assigned to PACUE selected OpenCL device, some applications show errors. This is because device information is different from the application’s intended one, thus some applications recognize it as an unusual event. To avoid these errors, PACUE camouflages OpenCL device details when the desired OpenCL device has been changed dynamically.

However, camouflaging OpenCL device details is risky, because devices have different specifications in the lower level. The first risk is application stability. The memory size of each hierarchy is device dependent, hence the unexpected memory size can result in application crash or error. The second risk is execution speed. If an application implements per-device optimization, mismatch between the intended device and the assigned device can result in unexpected performance degradation. From these reasons, we should camouflages device details only when it is necessary. To minimize the risks, PACUE camouflages devices in following levels.

1. **Device type level camouflage**
   When an application tries to acquire an OpenCL device list, PACUE will overwrite the *cl_device_type* value. As far as possible, PACUE will change this value for CL_DEVICE_TYPE_ALL. Showing all devices instead of the specific type devices is a reasonable choice, because it avoids forcing application using unknown

Table 1. Comparison of Device Camouflaging Methods

Overridden device type/ID	Specified Type when getting device list	Specified ID when creating a Context	Specified ID when creating a Command queue creation	Crash/Error Risk	Compatibility
A. Device type level	CPUs or GPUs	All CPUs or all GPUs	\	Low	Most applications
B. Context level	\	CPUs or GPUs		Low	Low
C. Command queue level	\	ALL	One CPU or one GPU	High	Most applications
D. A + C	ALL	ALL	One CPU or one GPU	Normal	High

device. Occasionally, applications cannot execute their OpenCL code on some device types. In this case, PACUE sets the *cl_device_type* value to the desired type, such as CL_DEVICE_TYPE_CPU or CL_DEVICE_TYPE_GPU.

2. **Context level camouflage**
When creating an OpenCL context, PACUE overrides the *cl_device_id* value and force OpenCL framework to build OpenCL binaries for each compute device. If PACUE recognize that the target application support only specific type of compute devices, PACUE will overwrite the *cl_device_id* value and limit device types for context. In addition, PACUE overrides the *cl_device_id* value when applications requests detailed device information. Therefore, application will see information of the device PACUE selected. This contributes to application's stability, because acquired device information, such as the memory size corresponds to that of the device actually will be used.
3. **Command queue level camouflage**
When the application calls *clCreateCommandQueue()* API, this is the last chance to change the device. Because of the stability issue described above, PACUE tries not to change device this timing, but if necessary, PACUE changes *cl_device_id* in arguments of this API. In this situation, the device is camouflaged completely, thus the application recognizes the camouflaged device as the device application specified. This is a terribly dangerous way to change device, still it improves application compatibility. This is risky in terms of device dependent characteristics, such as the memory size, however, we can switch the processor in more applications with this method. Hence, this method is ace in the hole.

As shown in Table 1, there are several device assignment overriding ways by the combination of these steps. Because they have a trade-off between application compatibility and application stability, we have to make a rule for applying these methods, and some hints are figured out in Sec. 3.

2.2 System Resource Management

We need a system-wide resource manager for heterogeneous processors, because average PC users cannot choose proper compute device for each application, and it is inconvenient that they select compute device every time the application runs. Some advanced PC users can choose proper compute device manually, however it is terribly inconvenient. Besides, many PC users do not know detailed construction of the PC they are using. These users cannot choose the proper compute device which satisfies their preference accurately, even if the application allows the user to select the compute device on its GUI configuration menu. For achieving high user-experience, the resource manager should select a compute device automatically according to user's preferences.

There are many studies in HPC area that build a resource manager to select compute device automatically [7, 8]. They show task distributing algorithm for heterogeneous processors environment that optimized for some specific purposes, such as maximizing performance or maximizing performance-per-watt. However, they cannot be applied to resource management on PC because the requirements are different between PC and HPC. The other approach to differentiate tasks, such as device-driver level approach [9] would be a possibility for our goal. However, we still need a system wide resource manager to consider heterogeneous processors and applications. These are three requirements of the resource manager especially for PCs.

- Considering user preference
  A PC user's preference often changes and they are not simple objects such as maximizing performance. In addition, it is difficult to recognize which application is really important, because we rarely specify priority of the process explicitly. Therefore, we have to build a resource manager, which infers user's preference by collecting PC utilization status and chooses compute devices for each application to achieve user preference accurately.
- Supporting various hardware configurations
  There are plenty of PC hardware components and applications. Because of this reason, combination of hardware components and applications are innumerable. In addition, the specifications of components depend on technology trends. For instance, some new GPU virtualization technologies for PC such as Virtu GPU virtualization [11] seamlessly use discrete GPU when specific APIs called. Thus, we have to build resource manager that supports various hardware configurations.
- Supporting various runtime versions
  Installed runtime libraries for parallel computing may vary in PCs. Application execution speeds are not only depends on hardware, but also depends on runtime libraries like OpenCL frameworks. Thus, a compute device selecting algorithm optimized for specific runtime version, such as designed for HPC, may not show good results on the newer version runtime libraries. We have to build compute device selecting algorithms that do not depend on a specific runtime version.

This resource manager has three features for satisfying the requirements explained above. The first feature is information gathering. PACUE collects information about how PC is utilized, such as whether an AC adapter is connected, temperatures and voltages of components, and processor utilization level such as processor loads and the running applications list. The second feature is the user preference inferring feature. The user describes their requirements by creating several requirement patterns. PACUE infers which pattern is the best for the present situation by using information acquired in the first step. The third feature is compute device selection, which decides the OpenCL device to be assigned to each application. We plan to implement a few compute device selecting algorithms for several user preference patterns. PACUE will assign compute devices to each application based on the algorithm which matches the inferred pattern of user preference. The resource manager works as cycles of these steps:

1. Collect PC utilization information.
2. Guess which profile is the best for the present condition.
3. Wait an inquiry of application and answer which device should be used.

For evaluation purpose, we built a basic resource manager which has communication function to order applications to utilize specific compute device. Because of lack of user preference based compute device selecting algorithms, recent PACUE can only select compute device by manual selection in the resource manager GUI. Still, it can receive an inquiry of compute device selection and answer a compute device to utilize.

3 Evaluation

In this section we confirm PACUE provides compute devices redirection capability for applications without modification on widely used applications. We first state the policy of the evaluation, then show and analyze the results.

3.1 Evaluation Policy

We evaluate PACUE in a PC with Intel Core i7-920 CPU and AMD RADEON HD 4850 GPU. As OpenCL framework, we adopt x86 binary of ATI Stream SDK 2.2 [4]. This framework supports both CPUs and AMD RADEON GPUs as OpenCL devices.

As testing applications, we chose the followings. They are publicly released and widely used for benchmarking, thus suites our purpose.

– DirectCompute & OpenCL Benchmark [1]
– SiSoftware Sandra 2011 [15]
– Sample code of ”OpenCL Introdouction” book [3]

We switch the device to utilize for these applications, and compare the methods for device switching for each of these applications.

3.2 Results

DirectCompute & OpenCL Benchmark. Table 2 shows the results. PACUE can redirect compute device perfectly on DirectCompute & OpenCL Benchmark, but only with method D.

SiSoftware Sandra 2011. Device switching failed. When PACUE tried to switch the device, Sandra 2011 exhibited strange behavior, such as showing the same device twice in the GUI. Because Sandra 2011 is an information & diagnostic utility for PC, it gathers device information by various APIs. Thus, the failure may be caused by the lack of integrity between device information gathered by PACUE hooked OpenCL API and information gathered by other APIs. However, PACUE do not make Sandra crashed.

Table 2. Result of DirectCompute & OpenCL Benchmark

Override Method	A-1	A-2	B-1	B-2	C-1	C-2	D-1	D-2
Specified Device Type	CPU	GPU	\	\	\	\	ALL	ALL
Specified Device ID for Context	\	\	CPUs	GPUs	ALL	ALL	ALL	ALL
Sp. Dev. ID for Command Queue	\	\	\	\	CPU	GPU	CPU	GPU
Application Recognized Devices	CPU*2	GPU*2	CPU*1	GPU*1	CPU*1	CPU*1	CPU*1+GPU*1	CPU*1+GPU*1
Dynamic Device Switching	Impossible	Impossible	Static	Static	Static	Static	Dynamic	Dynamic

Sample Codes of "OpenCL Introduction" Book. These codes are a set of 20 sample applications of OpenCL APIs. The device switching succeeded for all applications in them. However, 1 sample uses device memory information for the optimized array size, thus the result might depend on the device. The complete camouflaging device information might thus be incompatible with the information expected by the sample. This can cause the application crashing or errors, however it seemed to be working correctly while the experiment.

3.3 Analysis

The results show that PACUE can switch the compute devices on real applications. However, it fails for device dependent applications. They use detailed information of the particular device, such as device memory size. Thus, they may crash or behave strangely because of the information camouflaged by PACUE.

Among combinations of the device information overriding, we found the proper order to apply on applications. Shown in Table 1, these methods have a trade-off between application stability and application compatibility. In our evaluation, we found that the complete camouflaging method significantly increase application compatibility for real applications, such as DirectCompute & OpenCL Benchmark. However, it is realized by giving applications the information of the device the application specified, instead of giving the device information actually using. Original application creator is the only one who knows if the application works correctly when using the complete camouflaging method, thus we should avoid using this risky method if possible. In general, we suggest the following method applying order;

1. Override device type ALL and override device id when creating context. (Table 1 B)
2. Override device type ALL and override device id when creating command queue. (Table 1 D)
3. Keep original device type and override device id when creating command queue. (Table 1 C)
4. Override device type CPU or GPU when application requests list of available devices. (Table 1 A)

The first to the third methods similarly realize dynamic device selection. The upper is safer, the lower has more compatibility. Applications that cannot switch devices with the first method should use the second or the third method. The last one has the highest compatibility but it only provides static and restrictive device switching. Thus, this method should be applied when all other methods fail.

4 Conclusions and Future Work

In this paper we presented PACUE. First, PACUE switches the compute devices dynamically for applications on PCs with heterogeneous processors. Second, PACUE chooses compute devices assigned to applications to meet the user's requirement. We conducted experiments of our implementation, and demonstrated that 1 out of 2 real OpenCL applications, and all of 20 sample programs can change the compute device dynamically with the dynamic compute device redirector. In addition, we showed that a few device information camouflaging methods significantly increase application compatibility. From above work, we demonstrated potential availability of the dynamic compute device redirecting without application modified. However, there are 2 technical disadvantages in PACUE. The first disadvantage is that PACUE can switch devices only when creating command queue. This is because there is no support for dynamic device switching in OpenCL, thus the chances for switching devices are limited. We will investigate other methods to expand the chances for switching devices, also we will investigate the frequencies of the device switching timing on other APIs. The second disadvantage is OpenCL kernel optimization. Because of device information camouflaging, there is a possibility of executing kernels designed for other devices. This may decrease the performance significantly, thus we should avoid making situations like that. One answer is caching every type of kernel source codes by API hooking, and switch it according to the device actually using. Another answer is applying just-in-time OpenCL code optimization technique to improve performance. However, both of them can interfere the copyright law or licenses of the applications. Therefore, it may be difficult to apply it for PC applications. Because of this reason, we continue improving camouflage methods and we will avoid showing different devices information as possible as we can.

For our research goals, we have these ongoing works:

Increase Compatibility for Applications. We will address the problem that PACUE cannot switch compute devices in some applications. Also we will experiment application stability tests on applications.

Evaluate in Many Hardware Environment. We will conduct experiments on more hardware configuration such as Virtu, and improve hardware support of PACUE.

Implement the User Preferences Handler in the Resource Manager. We assume that there are several patterns describing user predefined requirements (*e.g.*, playing important game with the AC adaptor, and hasty file compression with unremarkable video encoding). PACUE infers matching pattern from the user's activity and resource utilization.

Implement Compute Device Selecting Algorithm. With user requirement recognition, we select compute devices to follow user preference accurately. We will implement some algorithms and parameter sets for each user requirement pattern. Also, we will explore performance impact while redirecting compute device in real applications and take measure against heavy performance degradation. Showing applications no OpenCL device by overriding OpenCL APIs can be one of the answers. In this case, applications will use internal optimized assembly to execute its transaction and it is often much faster than executing OpenCL code on CPUs. However, it has a disadvantage that compute device cannot change until restarting the application, because the application will never call OpenCL APIs again. Therefore, we will investigate each application's behavior concretely to decide how to let application to use CPUs.

Support for Other Parallel Computing Frameworks. We plan to implement modules for other APIs such as Fusion System Architecture Intermediate Layer Language (FSAIL).

References

1. DirectCompute & OpenCL Benchmark, `http://www.ngohq.com/graphic-cards/16920-directcompute-and-opencl-benchmark.html` (accessed on August 21, 2011)
2. OpenCL 1.1 Specification, `http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf`
3. Fixtars Corporation: OpenCL Introduction - Parallel Programming for Multicore CPUs and GPUs. Impress Japan (January 2010) (in Japanese)
4. AMD. ATI Stream Technology, `http://www.amd.com/US/PRODUCTS/TECHNOLOGIES/STREAM-TECHNOLOGY/Pages/stream-technology.aspx` (accessed on August 21, 2011)
5. Aoki, R., Oikawa, S., Tsuchiyama, R., Nakamura, T.: Hybrid opencl: Connecting different opencl implementations over network. In: Proc. IEEE CIT 2010, pp. 2729–2735 (2010)
6. Brodman, J.C., Fraguela, B.B., Garzarán, M.J., Padua, D.: New abstractions for data parallel programming. In: Proc. USENIX HotPar, p. 16 (2009)
7. Diamos, G.F., Yalamanchili, S.: Harmony: an execution model and runtime for heterogeneous many core systems. In: Proc. ACM HPDC, pp. 197–200 (2008)
8. Gupta, V., Schwan, K., Tolia, N., Talwar, V., Ranganathan, P.: Pegasus: Coordinated Scheduling for Virtualized Accelerator-based Systems. In: Proc. USENIX ATC, pp. 31–44 (2011)
9. Kato, S., Lakshmanan, K., Rajkumar, R., Ishikawa, Y.: TimeGraph: GPU Scheduling for Real-Time Multi-Tasking Environments. In: Proc. USENIX ATC, pp. 17–30 (2011)
10. Liu, W., Lewis, B., Zhou, X., Chen, H., Gao, Y., Yan, S., Luo, S., Saha, B.: A balanced programming model for emerging heterogeneous multicore systems. In: Proc. USENIX HotPar, p. 3 (2010)
11. Lucidlogix. Lucidlogix virtu, `http://www.lucidlogix.com/product-virtu.html` (accessed on August 21, 2011)
12. Microsoft. CreateRemoteThread Function (Windows), `http://msdn.microsoft.com/en-us/library/ms682437.aspx` (accessed on August 21, 2011)
13. Microsoft. SetWindowsHookEx Function (Windows), `http://msdn.microsoft.com/en-us/library/ms644990.aspx` (accessed on August 21, 2011)
14. Microsoft Research. Detours - microsoft research, `http://research.microsoft.com/en-us/projects/detours/` (accessed on August 21, 2011)
15. SiSoftware. Sisoftware zone, `http://www.sisoftware.net/` (accessed on August 21, 2011)