I. INTRODUCTION

The efforts on the utilization of GPUs for general purpose computing, also known as GPGPU, have expanded from the user-space programs to the kernel-space applications, in handling massive parallel tasks. For example [1] adopt GPUs as network processors (NPs) to complete the packet processing in routers. However, due to lack of kernel-space APIs, it is impossible to use CUDA or OpenCL to manipulate GPUs in kernel space. An alternative approach is to introduce additional user components [4] so that GPU computing can be completed through existing APIs in user space.

The Gdev [2] shows a promising solution that introduces a set of low-level functions for GPU computing in system kernel, called “Gdev API”. Through Gdev API, the GPU can be invoked in kernel space directly, and high-level APIs (e.g. CUDA), can be built on top of it. This solution focuses on the resource management and laid the foundation for kernel-space GPU application, but does not provide a development environment that uses high-level APIs and has very good portability (CUDA is only implemented on NVIDIA’s GPUs).

To meet these challenges, we present a programming model called “GReplay”, to develop kernel-space GPU applications. This model is based on the idea that the GPU processing in kernel can be completed by “replay” the operations submitted by user-space applications. Thus kernel applications for GPU processing can be migrated from user-space applications, which are developed with high-level APIs, such as OpenGL and OpenCL. As we are focusing GPGPU tasks rather than graphic tasks currently, our prototype only implements functions on GPU computing.

II. DESIGN

Our GReplay prototype consists of two distinct parts: a user component enhanced from open-source project Mesa3D[3], and a kernel component manipulate GPUs in the Linux kernel. A GReplay application, developed as Linux kernel modules, would use the object code compiled by the user component and call APIs provided by the kernel component.

A. Enhancement on the User-space Component

The object code executed on the GPU is generated by the user-space component, and submitted to the kernel-space device driver through the ioctl system call. We modifies the OpenCL library in Mesa3D so that it can generate C source code that contain all essential data, including object code, information on memory requirements and OpenCL kernel arguments. These data are outputted when OpenCL programs are executed while being linked to our modified OpenCL library.

B. Kernel-space Component

To develop the GPU application in system kernel, we introduce an abstraction layer in our programming model to hide the differences between drivers of different GPUs. Based on the abstraction layer, kernel applications can “replay” the activities recorded from user-space applications. However, the simple “replay” cannot achieve our goals, and we implement the equivalent API of “clSetKernelArg” in OpenCL, so that developers can execute the compiled OpenCL program with new arguments.

III. EVALUATION AND WORK-IN-PROGRESS

The evaluations show our applications in our GReplay model, executed in kernel space, are about 30% to 300% faster than their equivalent OpenCL programs executed in user space, because of our low overheads. For example, GReplay is able to copy 4342.29 megabytes of data per second, while the same program executed by Catalyst (official GPU driver) can only complete 1051.54 megabytes. However, the compiler used by GReplay is less efficient than the compiler used in Catalyst. In experiments on matrix additions, GReplay can process 3822.66 million of elements per second while Catalyst can process 3818.27 million in the same time.

There are several additional features that can be implemented in our prototype. First, we planned to support multiple instruction set architectures (ISAs) instead of single one. Second, our user component is not compatible with the OpenCL standard because of some missing features, and we are working on this problem. Third, the user component need to be improved as a cross compiler, to support multiple ISAs at the same time.

REFERENCES