Exploring Memory Options for Data Transfer on Heterogeneous Platforms*

Chao Liu†
Northeastern University
liu.chao@husky.neu.edu

Janki Bhimani‡
Northeastern University
bhimani@ece.neu.edu

Miriam Leeser
Northeastern University
mel@coe.neu.edu

1 INTRODUCTION

Heterogeneous platforms that use GPUs for acceleration are more and more popular in high performance and parallel computing. However, to accelerate applications with GPUs is not straightforward and great efforts are needed for users to develop applications that can make use of GPUs effectively. In our previous work, we proposed a programming framework that allows users to develop parallel applications based on a high level tasks and conduits abstraction [2]. Here we develop GPU applications based on this framework, implementing computational intensive kernels as GPU tasks in these applications.

We target NVIDIA GPUs. Through the CUDA runtime, there are three different types of memory: pageable memory, pinned memory and unified memory, that can be allocated for data transfer between CPU and GPU. The type of memory that is a good choice for a given application depends on the characteristics of the application [1]. To facilitate developing GPU programs and exploring different types of CPU/GPU memory transfer for performance optimization, we provide a uniform and simple interface to allocate space on system main memory (host memory) and GPU memory (device memory) in GPU task implementations. We defined following basic class:

class ***MemType_t*** *{ pageable, pinned, unified }*;
template<typename T> class ***GpuMem****;*

When we need to allocate memory for a data set that is used on a GPU, we can create a *GpuMem* object with the desired memory type (pageable, pinned or unified). Along with the object, we have different methods to get proper memory pointers and complete host/device data synchronization. Through this uniform interface and helper methods, we can create a GPU task with appropriate memory type arguments to explore the use of different memory schemes for host/device communication easily and effectively.

2 EXPERIMENTS AND ANALYSIS

To demonstrate the use of GPU task based applications and analyze the performance of different memory for CPU/GPU transfer, we developed a benchmark set that currently includes the ten applications shown in Table 1.

Table 1: Benchmarks

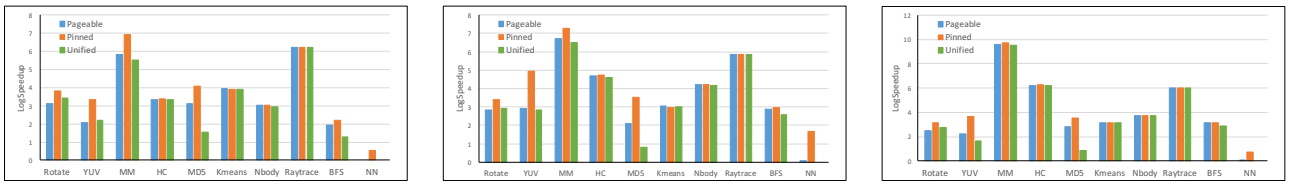
Application	Domain
Image Rotation(Rotate)	Image Processing
Color Conversion(YUV)	Image Processing
Matrix Multiply(MM)	Linear Algebra
2D Heat Conduction(HC)	Linear Algebra
MD5 Calculation(MD5)	Cryptography
K-means Clustering(Kmeans)	Data Mining
N-body simulation(Nbody)	Space Simulation
Ray tracing(Raytrace)	Computer Graphics
Bread First Search(BFS)	Graph Algorithm
Nearest Neighbors(NN)	Data Mining

These applications involve a variety of domains and we implement the parallel version with GPU tasks to make use of GPU for acceleration. For each application, we prepare three different sizes of workload in our experiments, which are referred to as Small, Medium and Large (S, M, L). Our test platform includes three different GPUs: NVIDIA Tesla C2070, NVIDIA Tesla K20m, and NVIDIA Tesla K40m. Tesla C2070 is NVIDIA Fermi architecture GPU and Tesla K20m and K40m belong to NVIDIA Kepler architecture.We run sequential implementations on an Intel Xeon E5-2650 CPU, recording the runtime as the baseline. All the speedup results are computed with regard to this baseline normalized by $\log_2$ for convenience.

We first test the effect of using different types of memory in our benchmarks. With the assistance of single memory interface, we only need to pass different memory type arguments to use different types of memory for the application. Figure 1 shows the speedup results of kernel applications running on a Tesla K20m GPU.

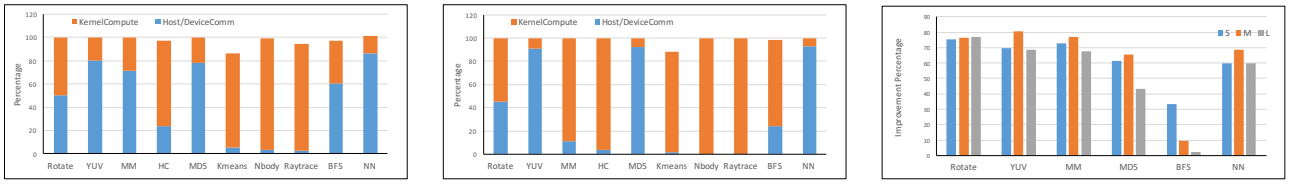
From the results we can see that, for Rotate, YUV, MD5 and NN, pinned memory performs better than pageable or unified memory. For NN, there is no speedup benefit unless using pinned memory. For MM and BFS, using pinned memory is also preferable. However, for large workloads, the performance improvement by using pinned memory is

*No need of demo setup.
†Graduate student of ECE at Northeastern University
‡Graduate student of ECE at Northeastern University



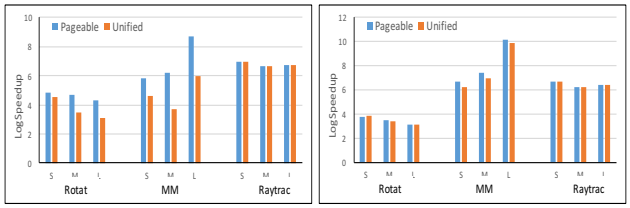
(a) Small workload (b) Medium workload (c) Large workload

Figure 1: Applications speedup of using different types of memory on Tesla K20m



(a) (b) (c)

Figure 2: Computation/Communication time cost percentage using pageable memory in (2a) Small workload, (2b) Large workload and (2c) Host/Device communication improvement from pageable to pinned memory



(a) Tesla C2070 test (b) Tesla K40m test

Figure 3: Unified memory test on different GPUs

not as good as for the small and medium workloads. So considering the scarcity of pinned memory, pageable or unified memory may be a wise choice for applications with a large amount of data to process. For the rest of the applications, performance of the different types of memory are very close under all three workloads.

Figure 2a and 2b show the percentage of computation/communication time in the total time using pageable memory for each application. We can see that host/device communication time of HC, K-Means, Nbody and Raytrace are relatively small compared to other applications. This is also why these four applications change less when adopting different types of memory. Figure 2c shows the host/device communication improvement for some applications when changing pageable memory to pinned memory for some of the benchmark applications.

Next, we test the effect of using unified memory on different types of GPUs. Figure 3 shows the test results of some applications for different sizes of workload. We can find that on Tesla C2070, the performance of Rotate and MM using unified memory is much lower than pageable memory. While on Tesla K40m GPU pageable memory performance improves a lot. For Raytrace, because there is little host/device transfer happened, using unified memory has little affection on application performance. Generally, Tesla K40m which is NVIDIA Kepler architecture GPU has better support for unified memory than Tesla C2070.

From these tests, we conclude that pinned memory can improve host/device communication performance significantly. For applications where memory transfer takes up a substantial amount of total runtime, using pinned memory improves overall performance. But we cannot use pinned memory randomly. Unified memory eases the programming procedure, but the implicit data transfer managed by the underlying system and hardware support are not always efficient. For choosing between different types of memory, our interface saves the user extra programming efforts, thus making the exploration of different design choices easier.

REFERENCES

[1] Janki Bhimani, Miriam Leeser, and Ningfang Mi. 2016. Design space exploration of GPU Accelerated cluster systems for optimal data transfer using PCIe bus. In *High Performance Extreme Computing Conference (HPEC), 2016 IEEE*. IEEE, 1–7.

[2] Chao Liu and Miriam Leeser. 2017. A Framework for Developing Parallel Applications with high level Tasks on Heterogeneous Platforms. In *Proceedings of the 8th International Workshop on Programming Models and Applications for Multicores and Manycores*. ACM, 74–79.

Exploring Memory Options for Data Transfer on Heterogeneous Platforms

Chao Liu; Janki Bhimani; Miriam Leeser
Department of Electrical and Computer Engineering, Northeastern University

Northeastern University
College of Engineering

Abstract

Developing parallel applications for GPU platforms and optimizing GPU related applications for good performance is important. We develop a high level task design framework with a concise interface to choose and allocate desired memory blocks that are used for host/device data transfer flexibly with little modification of the original programs. Memory options studied include pageable, pinned and unified. We developed a test benchmark set containing ten different kernel applications to test and analyze the effect of memory options in GPU applications.

Introduction

In previous work, we proposed a programming framework that allows users to develop parallel applications based on a high level tasks and conduits[2]. Here we develop GPU applications based on this framework, implementing computationally intensive kernels as GPU tasks in each application.

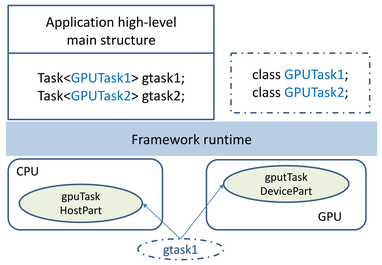


Figure 1. GPU tasks based application

Three basic procedures for each GPU task:

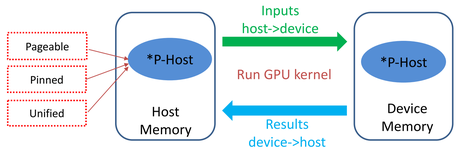


Figure 2. GPU program procedure and memory options

Target NVIDIA GPUs using the CUDA runtime. Three different kinds of memory can be allocated in system for host/device communication: pageable, pinned and unified.

Implementation and Application Development

To ease the use of different kinds of memory in a GPU task, we introduce a concise interface to select and create memory space for host/device data transfer:

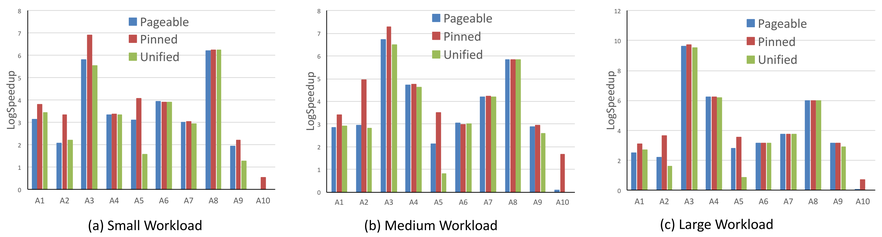
```
enum class MemType_t {
    pageable,
    pinned,
    unified
}

template <typename T> class GpuMem{
    MemType_t   m_memType;
    T           *m_hostPtr;
    T           *m_devPtr;
    ....  }

Task<GPUTask1>   gtask1(...., gTask, MemType_t::pinned, ....);
```

Easily define a GPU task using desired type of memory for host/device data transfer in an application, without change of existed task implementations.

To demonstrate the use of GPU task based applications and analyze the performance of different memory for CPU/GPU transfer, we developed a benchmark set that includes ten applications (see Table 1). For each application, we prepare three different sizes of workload in our experiments, referred to as Small, Medium and Large (S, M, L).

Experiments and Results

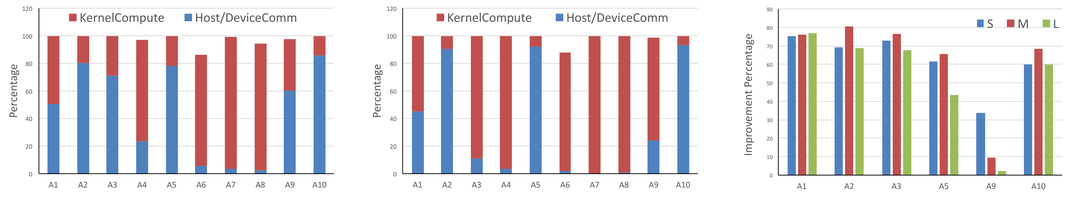
Our test platforms include three different NVIDIA GPUs: Tesla C2070, Tesla K20m, and Tesla K40m. Each test case was conducted for several rounds and average results recorded. We run sequential implementations on an Intel Xeon E5-2650 CPU, recording the runtime as the baseline.



(a) Small Workload (b) Medium Workload (c) Large Workload

Chart 1. Applications speedup of using different types of memory on Tesla K20m.

Pinned memory provides better performance for Rotate(A1), YUV(A2), MM(A3), MD5(A5), BFS(A9) and NN(A10). For MM and BFS, the performance improvement of using pinned memory for large workloads is not as good as for small and medium workloads.



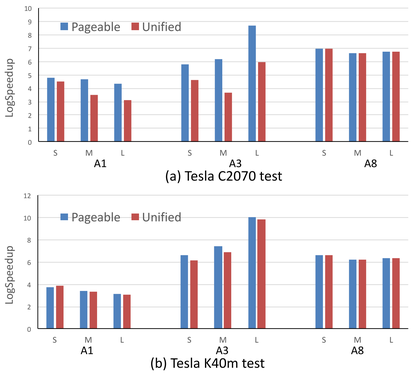
(a) (b) (c)

Chart 2. Computation/Communication time cost percentage using pageable memory for (2a) Small workload, (2b) Large workload and (2c) Host/Device communication improvement from pageable to pinned memory.

Chart 2(a) and (b) show that host/ device communication time of HC(A4), K-Means(A6), Nbody(A7) and Raytrace(A8) are relatively much smaller compared to other applications. Chart 2(c) shows the data transfer improvements of using pinned memory for applications that have substantial data transfer workload.

Table 1. Benchmarks

Application	Domain
Image Rotation(A1:Rotate)	Image Processing
Color Conversion(A2:YUV)	Image Processing
Matrix Multiply(A3:MM)	Linear Algebra
2D heat Conduction(A4:HC)	Linear Algebra
MD5 Calculation(A5:MD5)	Cryptography
K-means Clustering(A6:Kmeans)	Data Mining
N-body simulation(A7:Nbody)	Space Simulation
Ray tracing(A8:Raytrace)	Computer Graphics
Bread First Search(A9:BFS)	Graph Algorithm
Nearest Neighbors(A10:NN)	Data Mining



(a) Tesla C2070 test (b) Tesla K40m test

Chart 3. Unified memory test on different GPUs.

Unified memory tried for three applications: Rotate(A1), MM(A3) and Raytrace(A8). Unified memory causes performance degradation, especially on older GPUs.

Conclusions

From these tests, pinned memory can improve host/device communication performance and is preferable for applications where memory transfer takes up a substantial amount of total runtime. But pinned memory is not always the best. Unified memory eases the programming procedure, but the implicit data transfer managed by the underlying system and hardware support are not always efficient. Our interface saves the user extra programming effort, thus making the exploration of different design choices and types of memory easier.

Contact Information

Chao Liu
Email: liu.chao@husky.neu.edu

Janki Bhimani
Email: bhimani@ece.neu.edu

Miriam Leeser
Email: mel@coe.neu.edu

RCL Lab: http://www.coe.neu.edu/Research/rcl//projects.php

To read further

1. Janki Bhimani, Miriam Leeser, and Ningfang Mi. 2016. Design space exploration of GPU Accelerated cluster systems for optimal data transfer using PCIe bus. In High Performance Extreme Computing Conference (HPEC), 2016 IEEE. IEEE, 1–7.
2. Chao Liu and Miriam Leeser. 2017. A Framework for Developing Parallel Applications with high level Tasks on Heterogeneous Platforms. In Proceedings of the 8th International Workshop on Programming Models and Applications for Multicores and Manycores. ACM, 74–79.