CATEGORY: **CLUSTERS & GPU MANAGEMENT**
POSTER **CL03**
CONTACT NAME
Michela Becchi: becchim@missouri.edu

GPU TECHNOLOGY CONFERENCE

Node-Level Runtime System to Support Multi-tenancy in Clusters with GPUs

Michela Becchi[1], Kittisak Sajjapongse[1], Ian Graves[1], Adam Procter[1], Vignesh Ravi[2], Srimat Chakradhar[3]
[1]University of Missouri, [2]Ohio State University, [3]NEC Laboratories America

Abstract

In the last few years, thanks to their computational power, their progressively increased programmability and their wide adoption in both the research community and the industry, GPUs have become part of HPC clusters (for example, the Chinese Tianhe-1A and the US Stampede supercomputers). As a result, widely used open-source cluster resource managers (e.g. SLURM and TORQUE) have recently been extended with GPU support capabilities. These systems, however, provide simple scheduling mechanisms that often result in resource underutilization and, thereby, in suboptimal performance.

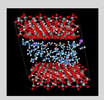
We propose a runtime system that provides abstraction and sharing of GPUs, while allowing isolation of concurrent applications. A central component of our runtime is a memory manager that provides a virtual memory abstraction to the applications. Our runtime is flexible in terms of scheduling policies, and allows dynamic (as opposed to programmer-defined) binding of applications to GPUs. In addition, our framework supports dynamic load balancing, dynamic upgrade and downgrade of GPUs, and is resilient to their failures.

Introduction

GPU applications


image processing


option pricing


molecular dynamics


data mining


weather prediction

GPUs in clusters and clouds


Top-500 supercomputers

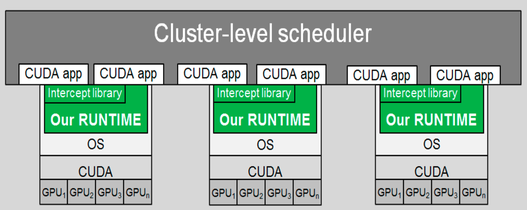
Cloud service providers

Changing paradigm

Accelerator model	Cluster/cloud model
1 application	Multi-tenancy
Explicit procurement of GPUs	Transparency & resource virtualization
GPU: dedicated resource	GPU: shared resource
Intra-application scheduling	Intra- and Inter-application scheduling
Static (or programmer-defined) binding of application to GPUs	*Dynamic* (or runtime) binding of applications to GPUs → better resource utilization and load balancing
Memory management *within* application	Advanced memory management *across* applications required

Deployment

Our node-level runtime component can be deployed in combination with cluster-level resource managers, such as the open-source TORQUE and SLURM, to allow GPU sharing and more effective scheduling and load-balancing schemes.



Objectives of our runtime

- GPU abstraction from end-users
- Applications-to-GPUs scheduling
- Applications-to-GPUs dynamic binding
- GPU sharing
- Virtual memory management
- Dynamic recovery and load balancing in case of GPU failure/upgrade/downgrade

References

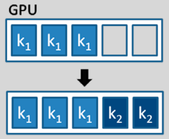
1. V. Ravi, M. Becchi, G. Agrawal, and S. Chakradhar. Supporting GPU Sharing in Cloud Environments with a Transparent Runtime Consolidation Framework. In *Proc. of HPDC '11*, San Jose, CA, USA, June 2011, pp. 217-228.
2. M. Becchi, K. Sajjapongse, I. Graves, A. Procter, V. Ravi, and S. Chakradhar. A Virtual Memory Based Runtime to Support Multi-tenancy in Clusters with GPUs. In *Proc. of HPDC '12*, Delft, The Netherlands, June 2012, pp. 97-108.

GPU sharing models

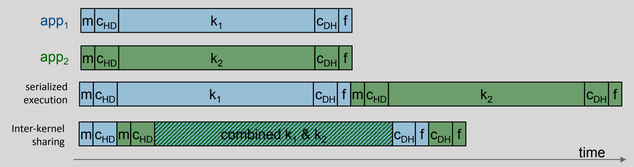
Resource sharing is a standard mechanism used within cluster and cloud environments to meet the requirements of concurrent users while keeping the infrastructure costs low. The increasing computational power of recent GPUs makes them good candidates for sharing. However, GPUs have been traditionally intended to function as accelerators and therefore to run one application at a time. As a consequence, their software stack provides limited support for sharing. We devise the following two sharing mechanisms:

Inter-kernel sharing [HPDC'11]

- **When:** GPU underutilized *within a kernel*
- **Why:** limited parallelism, small datasets
- **How:** kernel consolidation across applications

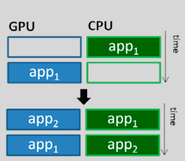


Example

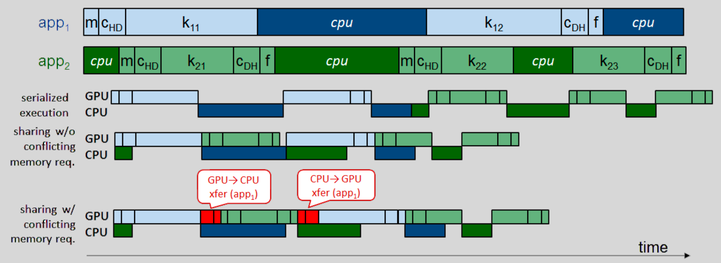


Inter-application sharing [HPDC'12]

- **When:** GPU underutilized *within an application*
- **Why:** long CPU phases
- **How:** application multiplexing on GPU

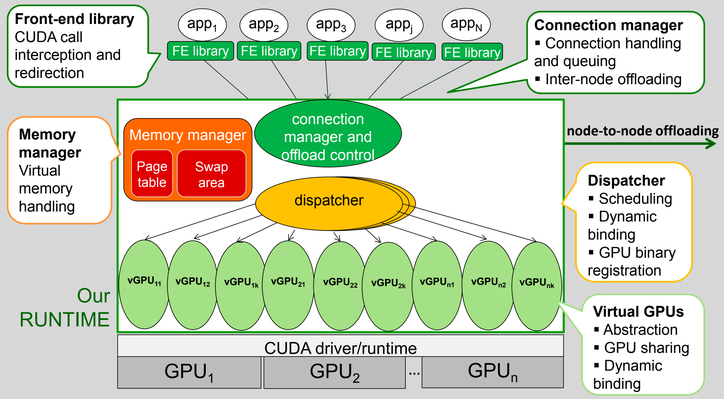


Example



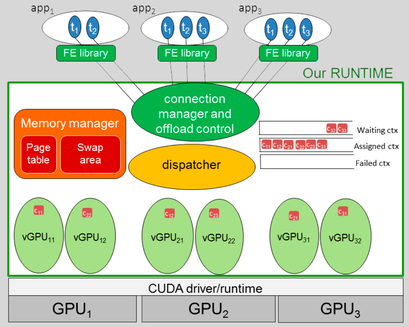
Runtime design

Runtime architecture



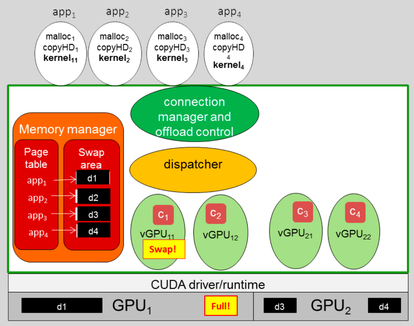
- Multi-threaded dispatcher
- Each virtual GPU is a separate thread

Mapping &Scheduling



- First-come, first-served policy
- Load balancing across GPUs

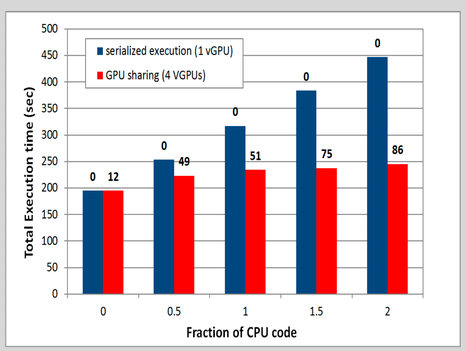
Virtual memory management



- Intra- and inter-application swapping
- Dynamic binding of applications to GPUs

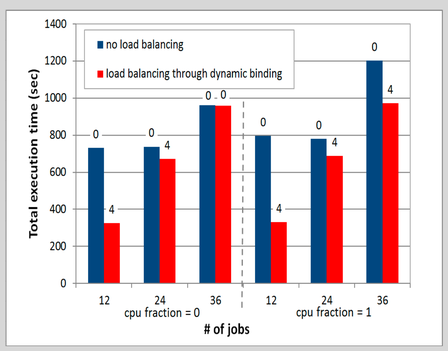
Experimental results

Inter-application swapping



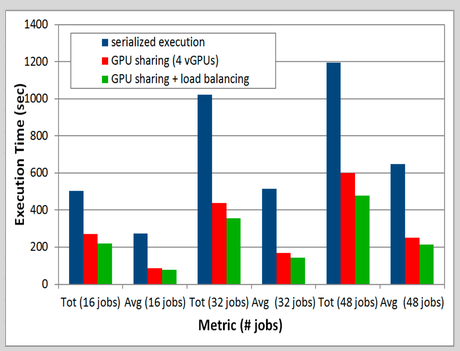
- 1 node: 2 Tesla C2050 & 1 Tesla C1060 GPUs
- 36 matmul jobs w/ 5 kernel calls and varying amount of CPU post-processing work
- Jobs have conflicting memory requirements

Node-level load balancing



- 1 node: 2 Tesla C2050 & 1 Quadro 2000 GPUs
- 12-36 long running jobs w/ no conflicting memory requirements

Cluster-level experiments



- 3-node cluster w/ 2 GPU compute nodes
- 25%/75% distribution of jobs w/ low/high memory requirements