Pegasus: Coordinated Scheduling for Virtualized Accelerator-based Systems

Vishakha Gupta, Karsten Schwan @ Georgia Tech
Niraj Tolia @ Maginatics
Vanish Talwar, Parthasarathy Ranganathan @ HP Labs
Increasing Popularity of Accelerators

2007
• IBM Cell-based Playstation

2008
• IBM Cell-based RoadRunner
• CUDA programmable GPUs for developers

2009
• Increasing popularity of NVIDIA GPUs powered desktops and laptops

2010
• Amazon EC2 adopts GPUs
• Tianhe-1A and Nebulae supercomputers in Top500

2011
• Tegras in cellphones
• Keeneland
Example x86-GPU System
Example x86-GPU System

Proprietary NVIDIA Driver and CUDA runtime
- Memory management
- Communication with device
- Scheduling logic
- Binary translation
Example x86-GPU System

C-like CUDA-based applications (host portion)

Proprietary NVIDIA Driver and CUDA runtime
- Memory management
- Communication with device
- Scheduling logic
- Binary translation

PCIe
Example x86-GPU System

CUDA Kernels

C-like CUDA-based applications (host portion)

Proprietary NVIDIA Driver and CUDA runtime
- Memory management
- Communication with device
- Scheduling logic
- Binary translation

PCIe

C

C-like CUDA-based applications (host portion)

Proprietary NVIDIA Driver and CUDA runtime
- Memory management
- Communication with device
- Scheduling logic
- Binary translation
Example x86-GPU System

CUDA Kernels

C-like CUDA-based applications (host portion)

Proprietary NVIDIA Driver and CUDA runtime
- Memory management
- Communication with device
- Scheduling logic
- Binary translation

Design flaw: Bulk of logic in drivers which were meant to be for simple operations like read, write and handle interrupts

Shortcoming: Inaccessibility and one scheduling fits all
Sharing Accelerators

2010
- Amazon EC2 adopts GPUs
- Other cloud offerings by AMD, NVIDIA

2011
- Tegras in cellphones
- HPC GPU Cluster (Keeneland)
Sharing Accelerators

2010
• Amazon EC2 adopts GPUs
• Other cloud offerings by AMD, NVIDIA

2011
• Tegras in cellphones
• HPC GPU Cluster (Keeneland)

• Most applications fail to occupy GPUs completely
  - With the exception of extensively tuned (e.g. supercomputing) applications
Sharing Accelerators

• Most applications fail to occupy GPUs completely
  – With the exception of extensively tuned (e.g. supercomputing) applications

• Expected utilization of GPUs across applications in some domains “may” follow patterns to allow sharing

2010
• Amazon EC2 adopts GPUs
• Other cloud offerings by AMD, NVIDIA

2011
• Tegras in cellphones
• HPC GPU Cluster (Keeneland)
Sharing Accelerators

• Most applications fail to occupy GPUs completely
  - With the exception of extensively tuned (e.g. supercomputing) applications

• Expected utilization of GPUs across applications in some domains “may” follow patterns to allow sharing

Need for accelerator sharing: resource sharing is now supported in NVIDIA’s Fermi architecture
Concern: Can driver scheduling do a good job?
NVIDIA GPU Sharing – Driver Default

- Xeon Quadcore with 2 8800GTX NVIDIA GPUs, driver 169.09, CUDA SDK 1.1
- Coulomb Potential [CP] benchmark from parboil benchmark suite
- Result of sharing two GPUs among four instances of the application
NVIDIA GPU Sharing – Driver Default

- Xeon Quadcore with 2 8800GTX NVIDIA GPUs, driver 169.09, CUDA SDK 1.1
- Coulomb Potential [CP] benchmark from parboil benchmark suite
- Result of sharing two GPUs among four instances of the application

**Driver can:** efficiently implement computation and data interactions between host and accelerator

**Limitations:** Call ordering suffers when sharing – any scheme used is static and cannot adapt to different system expectations
Re-thinking Accelerator-based Systems
Re-thinking Accelerator-based Systems

- Accelerators as first class citizens
  - Why treat such powerful processing resources as devices?
  - How can such heterogeneous resources be managed especially with evolving programming models, evolving hardware and proprietary software?
Re-thinking Accelerator-based Systems

- **Accelerators as first class citizens**
  - Why treat such powerful processing resources as devices?
  - How can such heterogeneous resources be managed especially with evolving programming models, evolving hardware and proprietary software?

- **Sharing of accelerators**
  - Are there efficient methods to utilize a heterogeneous pool of resources?
  - Can applications share accelerators without a big hit in efficiency?
Re-thinking Accelerator-based Systems

• Accelerators as first class citizens
  – Why treat such powerful processing resources as devices?
  – How can such heterogeneous resources be managed especially with evolving programming models, evolving hardware and proprietary software?

• Sharing of accelerators
  – Are there efficient methods to utilize a heterogeneous pool of resources?
  – Can applications share accelerators without a big hit in efficiency?

• Coordination across different processor types
  – How do you deal with multiple scheduling domains?
  – Does coordination obtain any performance gains?
Pegasus addresses the urgent need for systems support to smartly manage accelerators.
Pegasus addresses the urgent need for systems support to smartly manage accelerators.
(Demonstrated through x86--NVIDIA GPU-based systems)
Pegasus addresses the urgent need for systems support to smartly manage accelerators. (Demonstrated through x86--NVIDIA GPU-based systems)

It leverages new opportunities presented by increased adoption of virtualization technology in commercial, cloud computing, and even high performance infrastructures.
Pegasus addresses the urgent need for systems support to smartly manage accelerators.
(Demonstrated through x86--NVIDIA GPU-based systems)

It leverages new opportunities presented by increased adoption of virtualization technology in commercial, cloud computing, and even high performance infrastructures.
(Virtualization provided by Xen hypervisor and Dom0 management domain)
ACCELERATORS AS FIRST CLASS CITIZENS
Manageability
Extending Xen for Closed NVIDIA GPUs

Management Domain (Dom0)

Traditional Device Drivers

Hypervisor (Xen)

General purpose multicores

Tradical Devices

VM

Linux
Manageability
Extending Xen for Closed NVIDIA GPUs

Hypervisor (Xen)

Management Domain (Dom0)

VM

Linux

General purpose multicores

Compute Accelerators (NVIDIA GPUs)

Traditional Devices

Traditional Device Drivers
Manageability
Extending Xen for Closed NVIDIA GPUs

Management Domain (Dom0)
- Traditional Device Drivers
- Runtime + GPU Driver

Hypervisor (Xen)

VM
- Linux

General purpose multicores

Compute Accelerators (NVIDIA GPUs)

Traditional Devices
Manageability
Extending Xen for Closed NVIDIA GPUs

Hypervisor (Xen)

General purpose multicores

Compute Accelerators (NVIDIA GPUs)  Traditional Devices

NVIDIA’s CUDA – Compute Unified Device Architecture for managing GPUs
Manageability
Extending Xen for Closed NVIDIA GPUs

NVIDIA’s CUDA – Compute Unified Device Architecture for managing GPUs
Manageability
Extending Xen for Closed NVIDIA GPUs

Management Domain (Dom0)
- Traditional Device Drivers
- Runtime + GPU Driver

Hypervisor (Xen)

VM
- GPU Application
  - CUDA API
- GPU Frontend
  - Linux

VM
- GPU Application
  - CUDA API
- GPU Frontend
  - Linux

General purpose multicores

Compute Accelerators (NVIDIA GPUs)

Traditional Devices

NVIDIA’s CUDA – Compute Unified Device Architecture for managing GPUs
Manageability
Extending Xen for Closed NVIDIA GPUs

NVIDIA’s CUDA – Compute Unified Device Architecture for managing GPUs
Accelerator Virtual CPU (aVCPU) Abstraction

VM

Pegasus Frontend
- Interposer library
- Frontend driver

CUDA calls + Responses
- Application data
- Shared pages for data

Xen shared ring for CUDA calls (per VM)
- call buffer
Accelerator Virtual CPU (aVCPU) Abstraction

VM

Pegasus Frontend
- Interposer library
- Frontend driver

Pegasus Backend

Dom0

Xen shared ring for CUDA calls (per VM)
- call buffer

CUDA calls + Responses

Application data

Shared pages for data

CUDA calls + Responses

Application data

Polling thread
Accelerator Virtual CPU (aVCPU) Abstraction

**VM**

- **Pegasus Frontend**
  - Interposer library
  - Frontend driver

- **CUDA calls + Responses**
  - Application data
  - Shared pages for data

- **Xen shared ring for CUDA calls (per VM)**
  - Call buffer

**Dom0**

- **Pegasus Backend**

- **CUDA calls + Responses**
  - Application data

- **Polling thread**

- **CUDA Runtime + Driver**

**Interposer library**
Polling thread is the VM’s representative for call execution.

It can be queued or scheduled to pick calls and issue them for any amount of time ⇒ the accelerator portion of the VM can be scheduled.

Hence, we define an “accelerator” virtual CPU or aVCPU.
First Class Citizens

- The aVCPU has execution context on both, CPU (polling thread, runtime, driver context) and GPU (CUDA kernel)
- It has data used by these calls
First Class Citizens

- The aVCPU has execution context on both, CPU (polling thread, runtime, driver context) and GPU (CUDA kernel)
- It has data used by these calls

**VCPU**: first class schedulable entity on a physical CPU

**aVCPU**: first class schedulable entity on GPU (with a CPU component due to execution model)

Manageable pool of heterogeneous resources
SHARING OF ACCELERATORS
Scheduling aVCPUs

Too fine
Per call granularity

Too coarse
Per application granularity

Time slot based methods
Scheduling aVCPUs

Too fine
Per call granularity

RR: Fair share

Too coarse
Per application granularity

Time slot based methods
Scheduling aVCPUs

aVCPUs are given equal time slices and scheduled in a circular fashion.

Too fine
Per call granularity

RR: Fair share

Too coarse
Per application granularity

Time slot based methods
Scheduling aVCPUs

Too fine
Per call granularity

RR: Fair share

XC: Proportional fair share

Time slot based methods

Too coarse
Per application granularity
Scheduling aVCPUs

Adopt Xen credit scheduling for aVCPU scheduling. E.g. VMs 1, 2 and 3 have 256, 512, 1024 credits, they get 1, 2, 4 time ticks respectively, every scheduling cycle.
Scheduling aVCPUs

Too fine
Per call granularity

Too coarse
Per application granularity

RR: Fair share

XC: Proportional fair share

AccC: Prop. fair share

Time slot based methods
Scheduling aVCPUs

Instead of using the assigned VCPU credits for scheduling aVCPUs, define new accelerator credits. These could be some fraction of CPU credits.

**RR:** Fair share

**XC:** Proportional fair share

**AccC:** Proportional fair share

**Too coarse**
Per application granularity

**Too fine**
Per call granularity

**Time slot based methods**
Scheduling aVCPUs

Too coarse
Per application granularity

SLAF: Feedback-based prop. fair share

AccC: Prop. fair share

XC: Proportional fair share

RR: Fair share

Too fine
Per call granularity

Time slot based methods
Scheduling aVCPUs

Periodic scanning can lead to adjustment in the timer ticks assigned to aVCPUs if they do not get or exceed their assigned/expected time quota.

**Too fine**
- Per call granularity

**Too coarse**
- Per application granularity

**RR**: Fair share

**XC**: Proportional fair share

**AccC**: Prop. fair share

**SLAF**: Feedback-based prop. fair share

**Time slot based methods**
Performance Improves but Still High Variation

- BlackScholes <2mi,128>
- Xen 3.2.1 with 2.6.18 linux kernel in all domains
- NVIDIA driver 169.09 + SDK 1.1
- Dom1, Dom4 = 256, Dom2 = 512, Dom3 = 1024 credits
Performance Improves but Still High Variation

- BlackScholes <2mi,128>
- Xen 3.2.1 with 2.6.18 linux kernel in all domains
- NVIDIA driver 169.09 + SDK 1.1
- Dom1, Dom4 = 256, Dom2 = 512, Dom3 = 1024 credits

Still high variation: due to the hidden driver and runtime coordination: Can we do better?
COORDINATION ACROSS SCHEDULING DOMAINS
Coordinating CPU-GPU Scheduling

- Hypervisor co-schedule [CoSched]
  - Hypervisor scheduling determines which domain should run on a GPU depending on the CPU schedule
  - Latency reduction by occasional unfairness
  - Possible waste of resources e.g. if domain picked for GPU has no work to do
Coordinating CPU-GPU Scheduling

- **Hypervisor co-schedule [CoSched]**
  - Hypervisor scheduling determines which domain should run on a GPU depending on the CPU schedule
  - Latency reduction by occasional unfairness
  - Possible waste of resources e.g. if domain picked for GPU has no work to do

- **Augmented credit [AugC]**
  - Scan the hypervisor CPU schedule to temporarily boost credits of domains selected for CPUs
  - Pick domain(s) for GPU(s) based on GPU credits + remaining CPU credits from hypervisor (augmenting)
  - Throughput improvement by temporary credit boost
Coordination Further Improves Performance

- BlackScholes <2mi,128>
- Xen 3.2.1 with 2.6.18 linux kernel in all domains
- NVIDIA driver 169.09 + SDK 1.1
- Dom1, Dom4 = 256, Dom2 = 512, Dom3 = 1024 credits
Coordination: Aligning the CPU and GPU portions of an application to run almost simultaneously, reduces variation and improves performance.

- BlackScholes <2mi,128>
- Xen 3.2.1 with 2.6.18 linux kernel in all domains
- NVIDIA driver 169.09 + SDK 1.1
- Dom1, Dom4 = 256, Dom2 = 512, Dom3 = 1024 credits
Pegasus Scheduling Policies

- **No coordination:**
  - Default – GPU driver based – base case (None)
  - Round Robin (RR)
  - AccCredit (AccC) – credits based on static profiling

- **Coordination based:**
  - XenCredit (XC) – use Xen CPU credits
  - SLA feedback based (SLAF)
  - Augmented Credit based (AugC) – temporarily augment credits for co-scheduling

- **Controlled**
  - HypeControlled or coscheduled (CoSched)
Pegasus Scheduling Policies

• No coordination:
  − Default – GPU driver based – base case (None)
  − Round Robin (RR)
  − AccCredit (AccC) – credits based on static profiling

• Coordination based:
  − XenCredit (XC) – use Xen CPU credits
  − SLA feedback based (SLAF)
  − Augmented Credit based (AugC) – temporarily augment credits for co-scheduling

• Controlled
  − HypeControlled or coscheduled (CoSched)
Pegasus Scheduling Policies

- No coordination:
  - Default – GPU driver based – base case (None)
  - Round Robin (RR)
  - AccCredit (AccC) – credits based on static profiling

- Coordination based:
  - XenCredit (XC) – use Xen CPU credits
  - SLA feedback based (SLAF)
  - Augmented Credit based (AugC) – temporarily augment credits for co-scheduling

- Controlled
  - HypeControlled or coscheduled (CoSched)
Logical View of the Pegasus Resource Management Framework
Logical View of the Pegasus Resource Management Framework
Logical View of the Pegasus Resource Management Framework
Logical View of the Pegasus Resource Management Framework

Guest VM
Accelerator Application

Acc. Frontend
OS

Management Domain

Accelerator Selection Module

Accelerator Ready Queues

Domains to Schedule

DomA Scheduler

Hypervisor

Domains (Credits)

CPU Ready Queues

Domains to Schedule

Picked

VCPU

Physical Platform

Acc1 (Compute)
Acc2 (Compute)

C1
C2
C3
C4
Logical View of the Pegasus Resource Management Framework
Logical View of the Pegasus Resource Management Framework
Logical View of the Pegasus Resource Management Framework
Logical View of the Pegasus Resource Management Framework
Logical View of the Pegasus Resource Management Framework
Testbed Details

• Xeon 4 core @3GHz, 3GB RAM, 2 NVIDIA GPUs G92-450
• Xen 3.2.1 – stable, Fedora 8 Dom0 and DomU running Linux kernel 2.6.18, NVIDIA driver 169.09, SDK 1.1
• Guest domains given 512M memory and 1 core mostly
  • Pinned to different physical cores
  • Launched almost simultaneously: worst case measurement due to maximum load
• Data currently sampled over 50 runs for statistical significance despite driver/runtime variation
• Scheduling plots report h-spread with min-max over 85% readings or total work done over all runs in an experiment
## Benchmarks

<table>
<thead>
<tr>
<th>Category</th>
<th>Source</th>
<th>Benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Financial</td>
<td>SDK</td>
<td>Binomial (BOp), BlackScholes (BS), MonteCarlo (MC)</td>
</tr>
<tr>
<td>Media processing</td>
<td>SDK/parboil</td>
<td>ProcessImage(PI)=matrix multiply+DXTC, MRIQ, FastWalshTransform(FWT)</td>
</tr>
<tr>
<td>Scientific</td>
<td>Parboil</td>
<td>CP, TPACF, RPES</td>
</tr>
</tbody>
</table>
Benchmarks

<table>
<thead>
<tr>
<th>Category</th>
<th>Source</th>
<th>Benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Financial</td>
<td>SDK</td>
<td>Binomial (BOp), BlackScholes (BS), MonteCarlo (MC)</td>
</tr>
<tr>
<td>Media processing</td>
<td>SDK/parboil</td>
<td>ProcessImage(PI)=matrix multiply+DXTC, MRIQ, FastWalshTransform(FWT)</td>
</tr>
<tr>
<td>Scientific</td>
<td>Parboil</td>
<td>CP, TPACF, RPES</td>
</tr>
</tbody>
</table>

- **Diverse benchmarks**: from different application domains show -
  (a) different throughput and latency constraints, (b) varying data and CUDA kernel sizes and (c) different number of CUDA calls
## Benchmarks

<table>
<thead>
<tr>
<th>Category</th>
<th>Source</th>
<th>Benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Financial</td>
<td>SDK</td>
<td>Binomial (BOp), BlackScholes (BS), MonteCarlo (MC)</td>
</tr>
<tr>
<td>Media processing</td>
<td>SDK/parboil</td>
<td>ProcessImage(PI)=matrix multiply+DXTC, MRIQ, FastWalshTransform(FWT)</td>
</tr>
<tr>
<td>Scientific</td>
<td>Parboil</td>
<td>CP, TPACF, RPES</td>
</tr>
</tbody>
</table>

- **Diverse benchmarks**: from different application domains show - (a) different throughput and latency constraints, (b) varying data and CUDA kernel sizes and (c) different number of CUDA calls
- **BlackScholes worst in the set**: Throughput + latency sensitive due to large number of CUDA calls (depending on iteration)
Benchmarks

<table>
<thead>
<tr>
<th>Category</th>
<th>Source</th>
<th>Benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Financial</td>
<td>SDK</td>
<td>Binomial (BOp), BlackScholes (BS), MonteCarlo (MC)</td>
</tr>
<tr>
<td>Media processing</td>
<td>SDK/parboil</td>
<td>ProcessImage(PI)=matrix multiply+DXTC, MRIQ, FastWalshTransform(FWT)</td>
</tr>
<tr>
<td>Scientific</td>
<td>Parboil</td>
<td>CP, TPACF, RPES</td>
</tr>
</tbody>
</table>

- **Diverse benchmarks**: from different application domains show -
  (a) different throughput and latency constraints, (b) varying data and CUDA kernel sizes and (c) different number of CUDA calls
- **BlackScholes worst in the set**: Throughput + latency sensitive due to large number of CUDA calls (depending on iteration)
- **Latency sensitive FastWalshTransform**: multiple computation kernel launches and large data transfer
Ability to Achieve Low Virtualization Overhead

Cuda Time: Time within application to execute CUDA calls
Total Time: Total execution time of benchmark from command line

Increased # of CUDA Calls
Speed improvement for most benchmarks
Appropriate Scheduling is Important

Scheduler - RR

- Cuda Time
- Total Time

Overhead (1VM, 2Dom0, 2VM/1Dom0)
Appropriate Scheduling is Important
Without resource management, calls can be variably delayed due to interference from other application(s)/domain(s), even in the absence of virtualization.
Pegasus Scheduling
Black Scholes – Latency and throughput sensitive

Equal credits for all domains

Work done = \( \sum \text{all runs} \frac{\text{options}}{\text{time}} \)
Pegasus Scheduling
FWT – Latency sensitive

Per Call Latency (microsec)

Dom1, Dom4 – 256, Dom2 - 1024, Dom3 – 2048 credits
Insights

- Pegasus approach efficiently virtualizes GPUs
Insights

• Pegasus approach efficiently virtualizes GPUs
• Coordinated scheduling is effective
  – Even basic accelerator request scheduling can improve sharing performance
  – While co-scheduling is really useful [CoSched], other methods can come close [AugC], keep up utilization and give desirable properties
Insights

• Pegasus approach efficiently virtualizes GPUs
• Coordinated scheduling is effective
  – Even basic accelerator request scheduling can improve sharing performance
  – While co-scheduling is really useful [CoSched], other methods can come close [AugC], keep up utilization and give desirable properties
• Scheduling lowers degree of variability caused by un-coordinated use of the NVIDIA driver.
Insights

• Pegasus approach efficiently virtualizes GPUs
• Coordinated scheduling is effective
  – Even basic accelerator request scheduling can improve sharing performance
  – While co-scheduling is really useful [CoSched], other methods can come close [AugC], keep up utilization and give desirable properties
• Scheduling lowers degree of variability caused by un-coordinated use of the NVIDIA driver.

No single `best' scheduling policy
Clear need for diverse policies geared to match different system goals and to account for different application characteristics
Conclusion

- We successfully virtualize GPUs to convert them into first class citizens
Conclusion

• We successfully virtualize GPUs to convert them into first class citizens

• Pegasus approach abstracts accelerator interfaces through CUDA-level virtualization
  – Devise scheduling methods that coordinate accelerator use with that of general purpose host cores
  – Performance evaluated on x86-GPU Xen-based prototype
Conclusion

• We successfully virtualize GPUs to convert them into first class citizens

• Pegasus approach abstracts accelerator interfaces through CUDA-level virtualization
  – Devise scheduling methods that coordinate accelerator use with that of general purpose host cores
  – Performance evaluated on x86-GPU Xen-based prototype

• Evaluation with a variety of benchmarks shows
  – Need for coordination when sharing accelerator resources, especially for applications with high CPU-GPU coupling
  – Need for diverse policies when coordinating resource management decisions made for general purpose vs. accelerator core
Future Work: Generalizing Pegasus

- **Applicability**: concepts applicable to open as well as close accelerators due lack of integration with runtimes
  - Past experience with IBM Cell accelerator [*Cellule*]
  - Open architecture allows finer grained control of resources
Future Work: Generalizing Pegasus

- **Applicability**: concepts applicable to open as well as close accelerators due lack of integration with runtimes
  - Past experience with IBM Cell accelerator [Cellule]
  - Open architecture allows finer grained control of resources

- **Toolchains**: sophistication through integration
  - Instrumentation support from Ocelot [GTOcelot]
  - Improve admission control, load balancing and scheduling
Future Work: Generalizing Pegasus

- **Applicability**: concepts applicable to open as well as close accelerators due lack of integration with runtimes
  - Past experience with IBM Cell accelerator [Cellule]
  - Open architecture allows finer grained control of resources

- **Toolchains**: sophistication through integration
  - Instrumentation support from Ocelot [GTOcelot]
  - Improve admission control, load balancing and scheduling

- **Heterogeneous platforms**: Scheduling different personalities for a virtual machine [Poster session]
  - More generic problem where even processing resources on the same chip can be asymmetric
Future Work: Generalizing Pegasus

- **Applicability**: concepts applicable to open as well as close accelerators due lack of integration with runtimes
  - Past experience with IBM Cell accelerator [Cellule]
  - Open architecture allows finer grained control of resources
- **Toolchains**: sophistication through integration
  - Instrumentation support from Ocelot [GTOcelot]
  - Improve admission control, load balancing and scheduling
- **Heterogeneous platforms**: Scheduling different personalities for a virtual machine [Poster session]
  - More generic problem where even processing resources on the same chip can be asymmetric
- **Scale**: Extensions to cluster-based systems with Shadowfax [VTDC`11]
Related Work

- Heterogeneous and larger-scale systems – [Helios], [MultiKernel]
- Scheduling extension – [Cypress], [Xen Credit Scheduling], [QoS Adaptive Communication], [Intel Shared ISA Heterogeneity], [Cellular Disco]
- GPU Virtualization: [OpenGL], [VMWare DirectX], [VMGL], [vCUDA], [gVirtuS]
- Other related work
  - Accelerator Frontend or multi-core programming models: [CUDA], [Georgia Tech Harmony], [Georgia Tech Cellule], [OpenCL]
  - Some examples: [Intel Tolapai], [AMD Fusion], [LANL Roadrunner]
  - Application domains: [NSF Keeneland], [Amazon Cloud]
  - Interaction with higher levels: [PerformancePointsOSR]
  - Cluster level: [rCUDA], [Shadowfax]
Thank you!