HyVM – Hybrid Virtual Machines

Ada Gavrilovska
Karsten Schwan
Sudha Yalamanchili

and many PhD students

Research Context

Future server and web applications

- Web: media-rich content => increasing computational needs
- Server: financial applications, critical enterprise codes, GPU-supported database codes

Heterogeneous Multicore Platforms (Hybrid Systems)

- off- and on-chip heterogeneous cores
- rich memory hierarchies

New Execution Models

- dynamic parallelism, e.g., in cloud computing
- data-intensive codes, e.g., MapReduce

Future Applications

Media and image processing:

- for dynamic web content
- '**Snapfish**-like' imaging suite

Science and gaming:

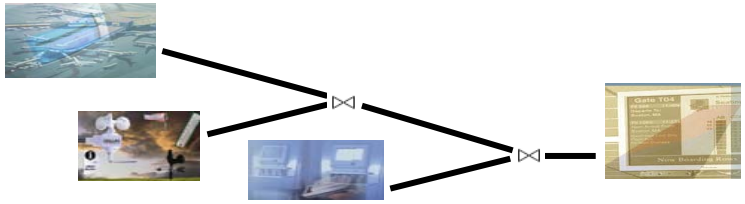
- Fusion modeling
- Ray tracing
- High perf. I/O
- **LAPACK, BLAS**, **VSIPL**, …

Financial and risk analysis:

- Black Scholes
- **Risk Analysis**
- **Derivatives processing**

Data-intensive and web:

- **Critical enterprise codes**
- **Data-intensive codes**
- Sensor data processing



Heterogeneous Platforms

- **Asymmetries**:
  - Performance
    - Different clock speeds, duty cycles
    - Differing issue widths
    - Varying cache sizes
  - NUMA memory, NUCA caches
- **Functional differences:**
  - Multiple accelerators:
    - GPU, Communications, Encryption, …
  - Shared ISA:
    - Missing SSE version
    - Missing floating point
    - Additional instructions for acceleration (Larrabee, …)

HyVM Approach

- Treat all hardware execution units as processors - PCPUs
- Distinguish ‘heterogeneous’ from ‘asymmetric’ (shared ISA) cores
  - aCPUs, sCPUs, CPUs, …
- Create a software platform capable of exporting hybrid CPUs to VMs:
  - **HyVM – Hybrid Virtual Machines:**
    - **supported by heterogeneity- and asymmetry-aware hypervisors**

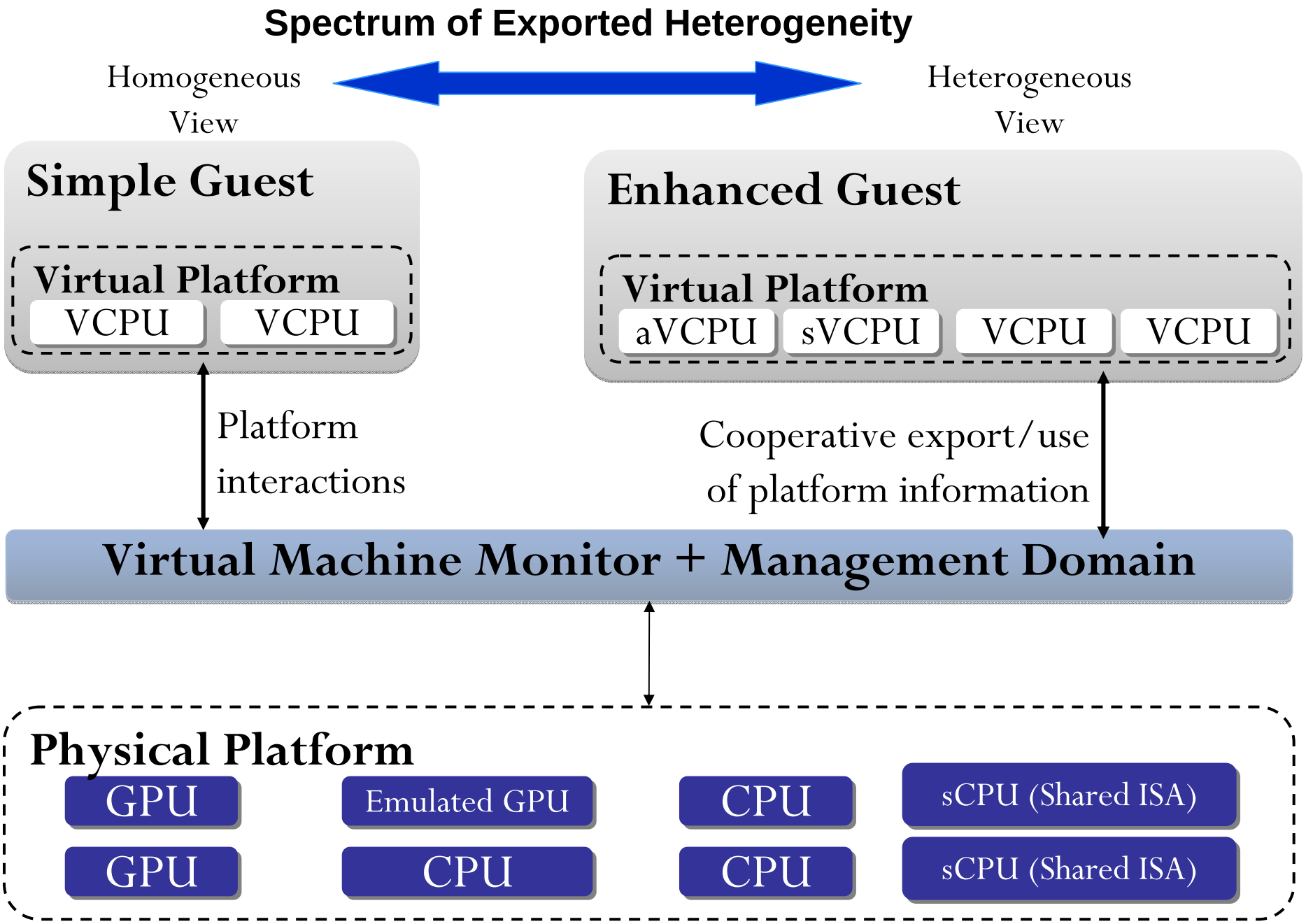
HyVM Project Elements

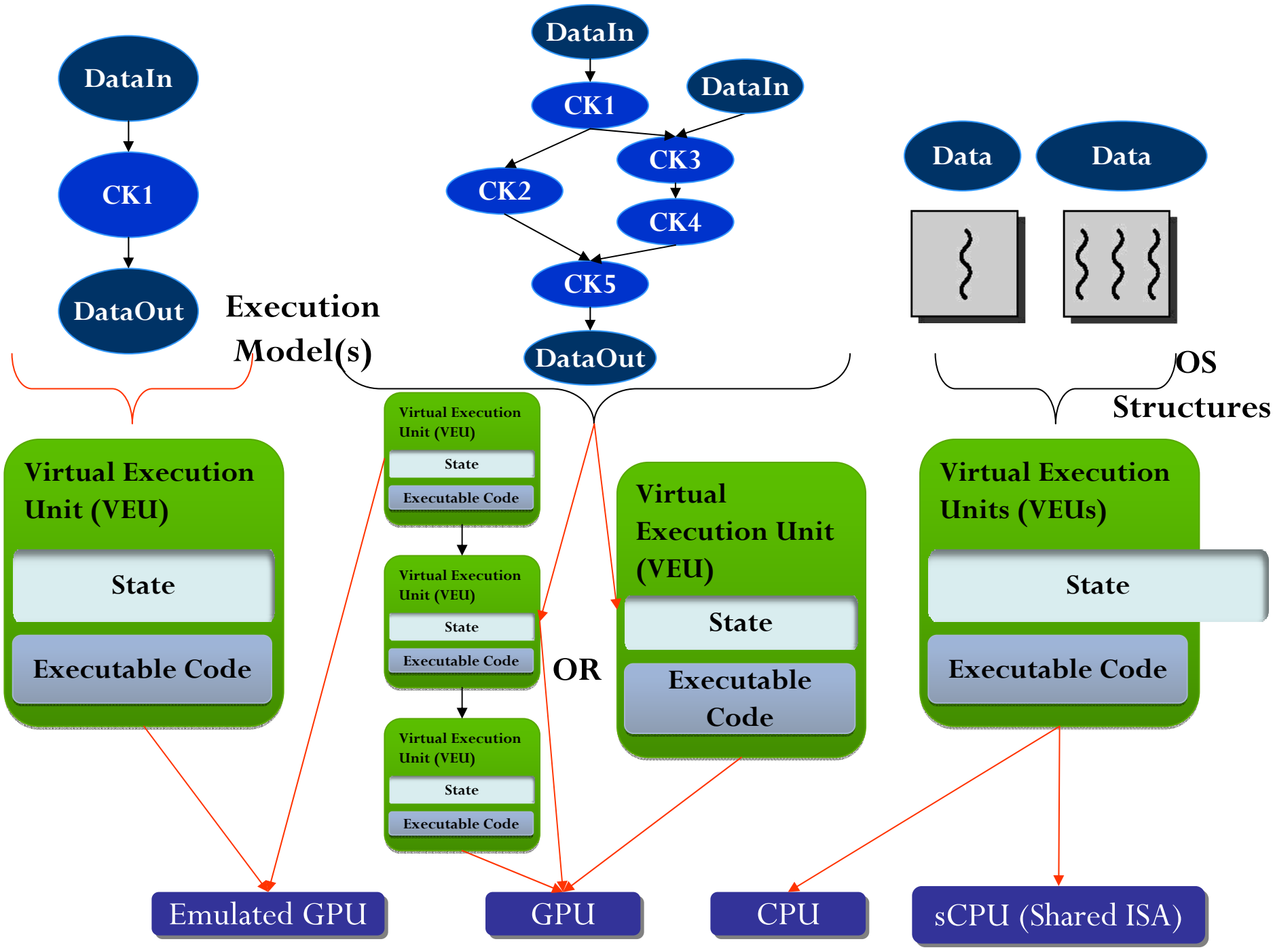
Attaining the uniform HyVM execution model

- **Leveraging virtualization technologies:**
  - Virtual Execution Units (VEUs)
    - Finer grain schedulable entities than VCPUs
  - Specialized execution environments (SEEs) for accelerators
    - **GViM** for efficient GPU virtualization
    - Earlier work on Cell processor creating our own **SEE**
  - Dynamic resource management for sets of VEUs (SLA-awareness, runtime monitoring)
    - **Coordinated scheduling for accelerators** – **VMaCS**
    - **InTune -** system abstractions for coordinated management
    - **Cache-centric 'region' scheduling + correlation scheduling** for shared ISA VEUs
    - Addressing **NUMA** properties
    - **Remote** execution **-** `Keeneland' extension
  - Future hypervisors (using Xen): **fully asymmetry- and heterogeneity-aware (e.g., using Ocelot)**

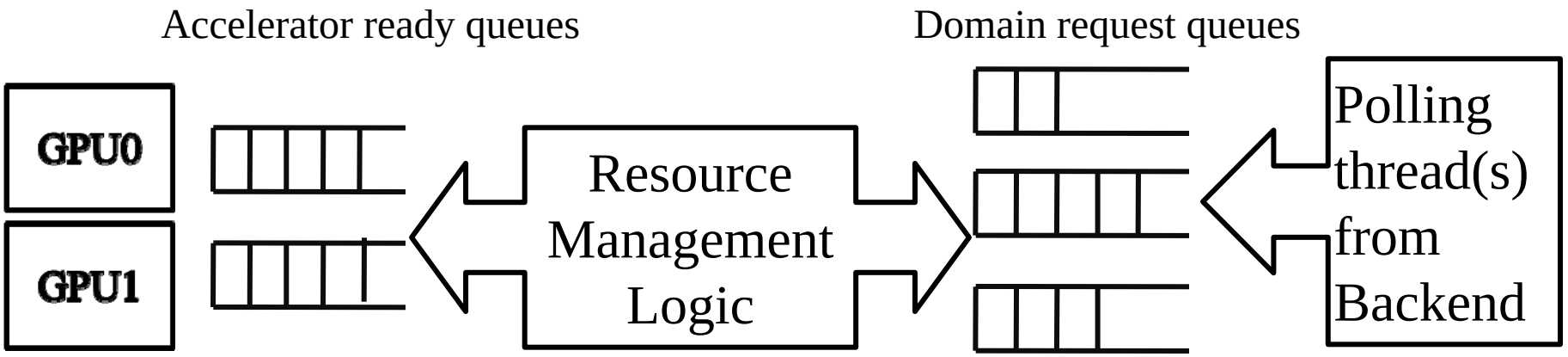
HyVM Project Elements

- **Emulated platform: leveraging evolving industry standards for accelerator APIs and interactions:**
  - Tool chain support for CUDA => LLVM/PTX translation, with future work on OpenCL – **Ocelot**
  - Runtime APIs at CUDA level - **GViM**
- **High performance:**
  - Exploiting compiler methods to optimize execution regimes for sets of VEUs – **Harmony**
  - Scheduling via on-line performance models and dynamic code generation for GPU vs. Host ISAs → Ocelot/LLVM
- **New opportunities:**
  - Analysis tools for debugging and performance tuning via emulation → Ocelot++
  - Exploring new programming models - Datalog to GPU compilation


Spectrum of Exported Heterogeneity
Homogeneous View
Heterogeneous View
Simple Guest
Virtual Platform
VCPU
VCPU
Enhanced Guest
Virtual Platform
aVCPU
sVCPU
VCPU
VCPU
Platform interactions
Cooperative export/use of platform information
Virtual Machine Monitor + Management Domain
Physical Platform
GPU
GPU
Emulated GPU
CPU
CPU
CPU
sCPU (Shared ISA)
sCPU (Shared ISA)


DataIn
CK1
DataOut
Execution Model(s)
DataIn
CK1
DataIn
CK3
CK2
CK4
CK5
DataOut
Data
Data
OS Structures
Virtual Execution Unit (VEU)
State
Executable Code
Virtual Execution Unit (VEU)
State
Executable Code
Virtual Execution Unit (VEU)
State
Executable Code
Virtual Execution Unit (VEU)
State
Executable Code
OR
Virtual Execution Units (VEUs)
State
Executable Code
Emulated GPU
GPU
CPU
sCPU (Shared ISA)

Technical Elements – GViM + VMaCS

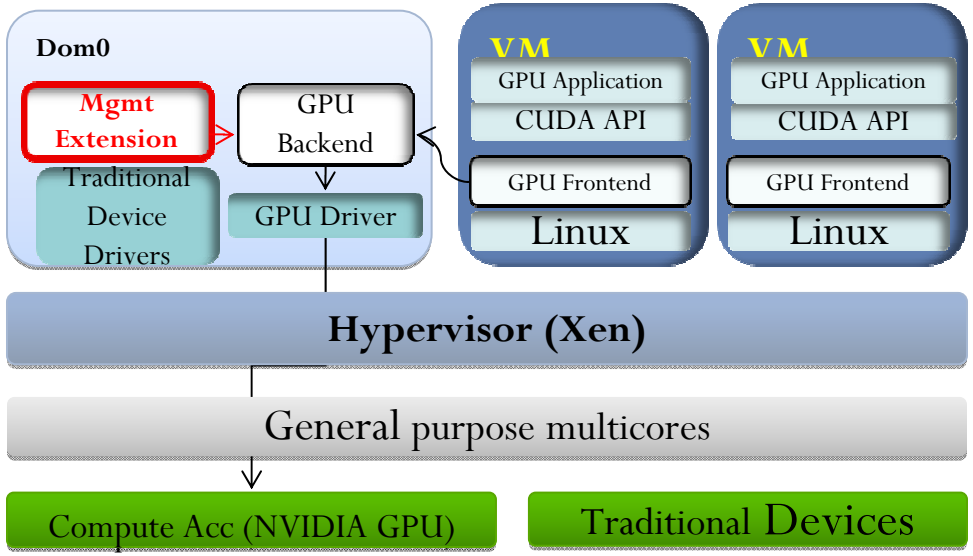


GViM:

- **VEUs on aVCPUs**

VMAcS:

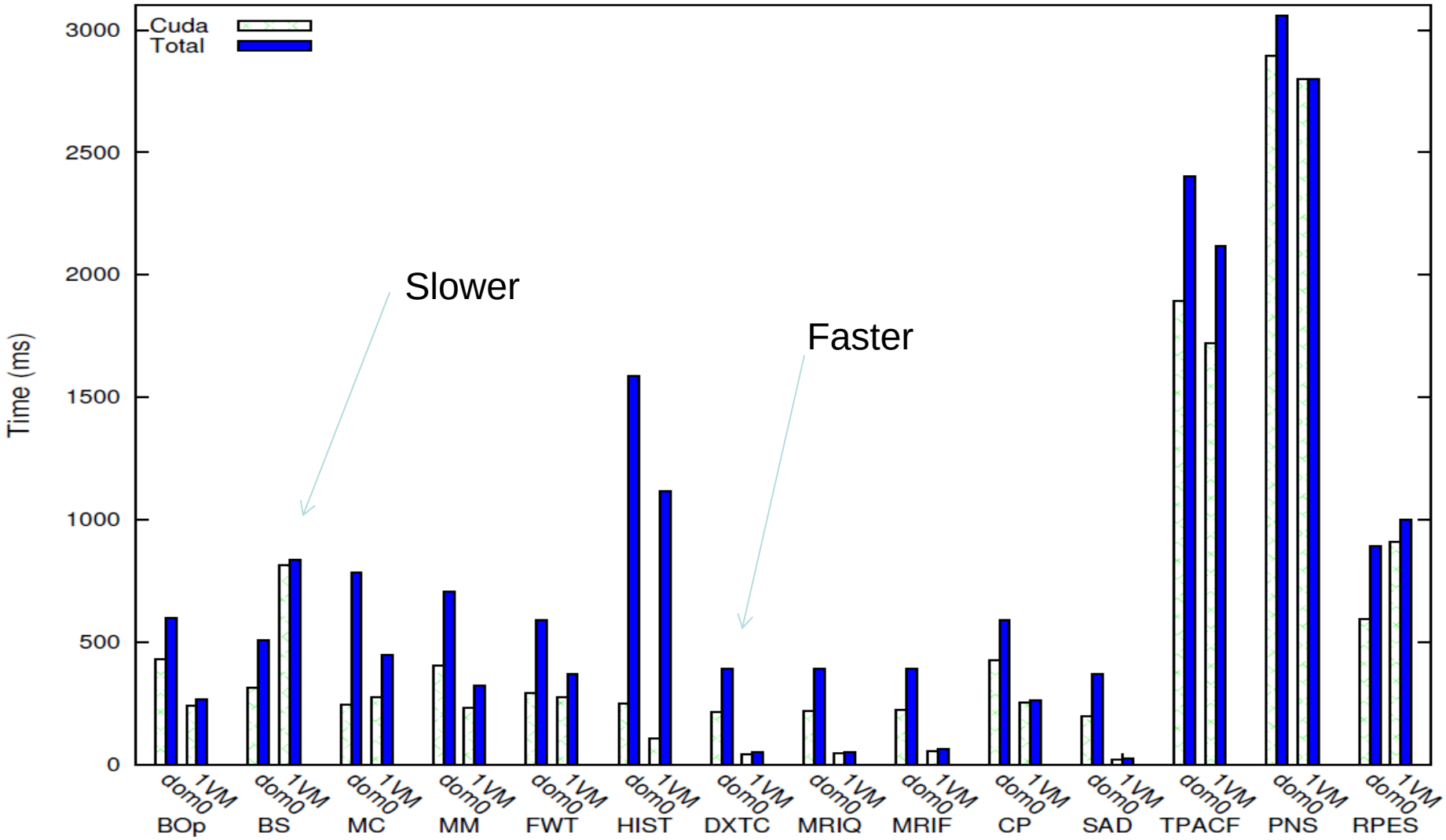
- **Management Extensions for *Coordinated Scheduling***



GViM and VMaCs – Experimental Evaluation

- Xeon quad-core @3GHz, 3GB RAM
- 2 NVIDIA GPUs G92-450
- Xen, with Fedora guest domains with 512M memory, pinned to single cores, some runs with
- Dom0 with `unlimited' resources.
- Data sampled over 50runs, typically reporting means and standard deviations.

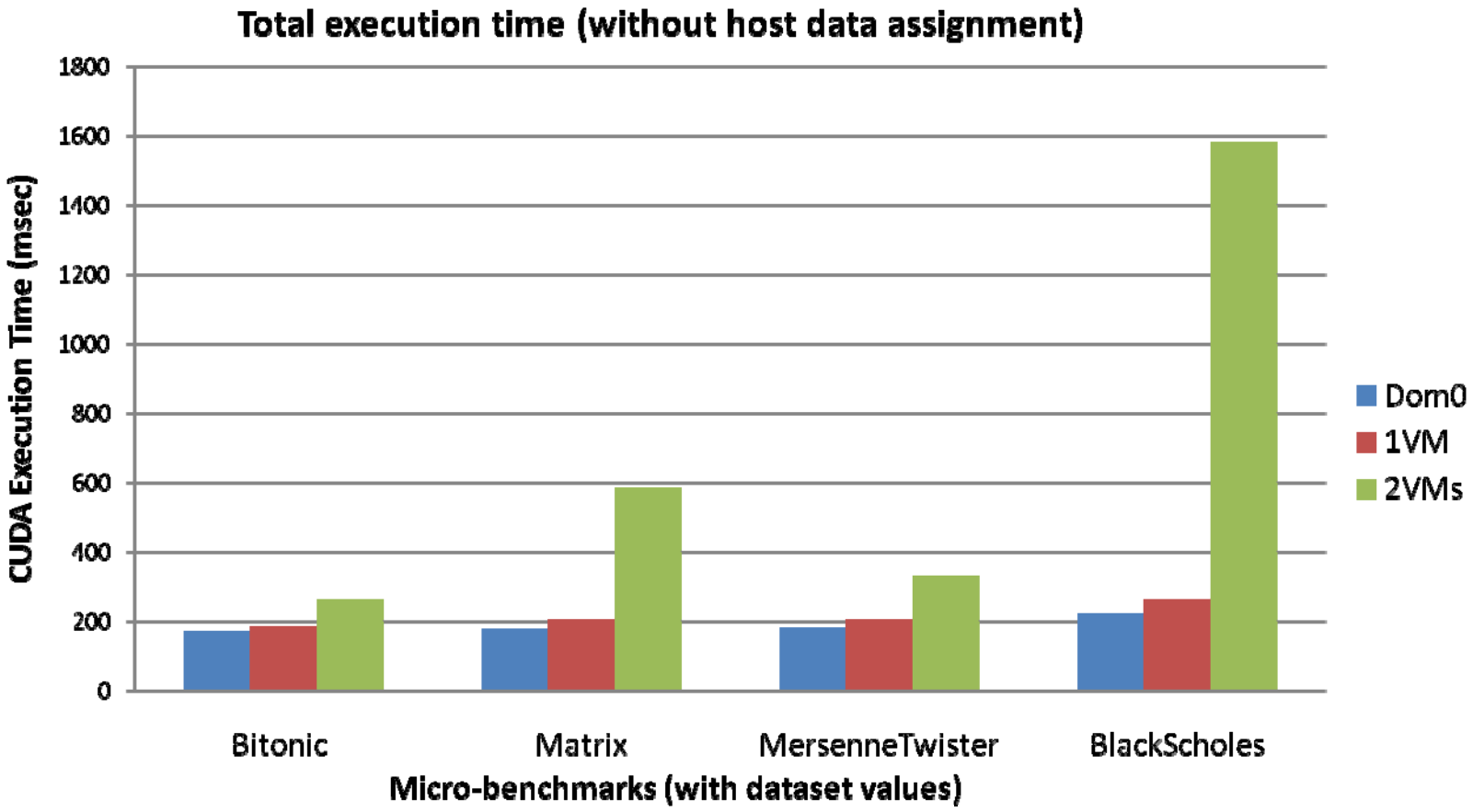
Technical Elements – GViM: Performance Implications of Virtualization



Benchmark Timings when running on Multicore Platform:
all raw CUDA call timings < 60 microseconds

Technical Elements – VMaCS

SLA Compliance with Multiple Guests – Need for Active Resource Management



Without resource management, calls can be variably delayed due to interference from other domain

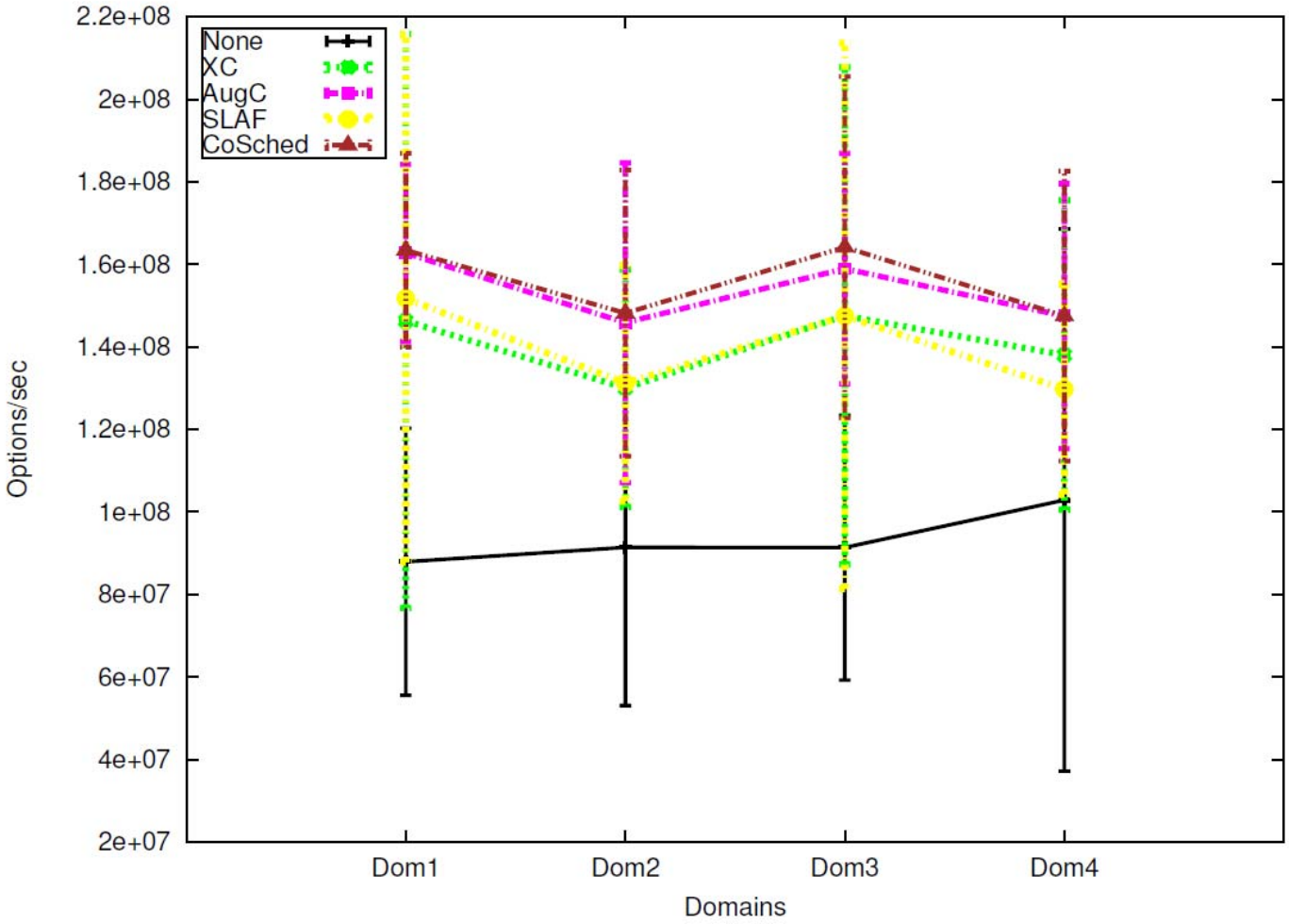
VMaCS Scheduling

- No coordination:
  - Default – GPU driver based – base case
  - Round Robin (RR)
  - AccCredit (AccC) – credits based on static profiling (calibration)
- Coordination:
  - XenCredit (XC) – use Xen CPU credits
  - Augmented Credit based (AugC) – temporarily augment credits for co-scheduling
  - SLA feedback based (SLAF)
- Controlled
  - HypeControlled (HC) or co-scheduled

VMaCS – Experimental Results for Throughput- vs. Latency-Sensitive Applications

- 4 x86 cores, 2 GPUs
- 2, 3, 4 Domains
  - 3 domains: one GPU will be shared
  - 4 domains: CPU sharing between either Dom0 and guest domain or two guest domains
- Throughput-sensitive benchmark combinations:
  - Financial: BlackScholes, MC, Binomial options
  - Imaging: Snapfish with DXTC, Matrix multiplication, histogram, FastWalshTransform
  - Scientific: Coulomb Potential and Petri Net Simulation
- Latency-sensitive benchmark combinations:
  - Financial: BlackScholes
  - Imaging: Snapfish with DXTC, Matrix multiplication, Histogram, FastWalshTransform - measuring latency per image operation
  - Scientific: Parboil

VMaCS Scheduling: Black Scholes : Equal Credits



Scheduling is important and can work well

VMaCS Scheduling

- Other lessons:
  - no one correct scheduling strategy
    - latency-sensitive codes => co-scheduling
    - but: co-scheduling vs. fairness and throughput
  - scheduling not useful for ill-structured GPU codes
    - frequent requests with small compute times
  - AugC and SLAF good when `mixing' domains
  - driver effects: e.g., large variations in execution times when sharing the GPU

Other Project Elements

- Asymmetric Cores: with R. Knauerhase, Intel
  - Region Scheduling – NUCA and NUMA characteristics
  - Correlation Scheduling – Core Asymmetries
  - Power Management : with Vishal Gupta (and with Intel and Microsoft)

- Scalable Hypervisor Structures: Mukil Kesavan, Priyanka Tembey, Dulloor Rao
  - Hypervisor structure should match `islands of cores'
  - e.g., consider Xen `Dom0' bundling of functionality

  $\Rightarrow$ offer solutions for isolation with bundled functionality

- On Petascale Machines: with Jeff Vetter, Scott Klasky (ORNL)
  - Large-scale GPU-based Cluster Machine for HPC codes

- Toward Exascale Systems: with Vanish Talwar, Partha Ranganathan (HP)
  - Scalable Monitoring and Online Behavior Detection

Future Directions

- **Why HyVMs**: future applications:
  - Media-rich web applications
    - online `photoshop' (with HP)
    - dynamic stream customization (with Motorola)
  - Financial and HPC codes
    - Benchmarks (e.g., Black Scholes – IBM, HPC – NVIDIA, Interactive - Intel)
    - Petascale codes (with DOE)
    - Petascale I/O (with DOE)

- **From platforms to clusters to large-scale data centers:**
  - Distributed execution model for petascale machines
  - Other execution models: data-intensive (Hadoop, System S, …)

- **Future platforms:**
  - **Power/performance** tradeoffs and implications
  - **Memory hierarchies** and their effects (on- vs. off-chip)
  - Alternative memory models – e.g., PGAS
  - Massively parallel HyVMs – e.g., 1000s of domains
  - Tool chains: future standards (e.g., OpenCL), compiler research (other faculty)