Data Partitioning on Heterogeneous Multicore and Multi-GPU systems Using Functional Performance Models of Data-Parallel Applictions

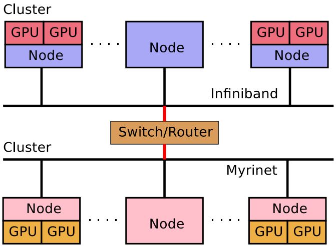
Ziming Zhong Vladimir Rychkov Alexey Lastovetsky

Heterogeneous Computing Laboratory
University College Dublin, Ireland

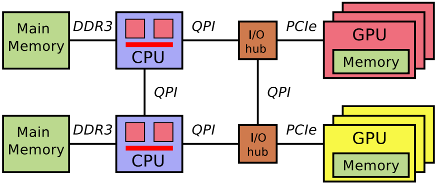
Cluster 2012

Motivation

- Hybrid GPU-accelerated parallel computers
  - Higher power efficiency, performance/price ratio, etc.
  - Successfully applied to bioinformatics, astrophysics, molecular dynamics, computational fluid dynamics, etc.



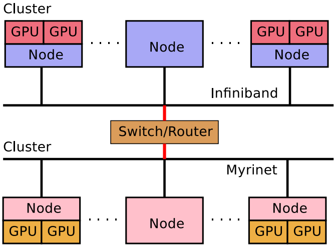
Hybrid Clusters



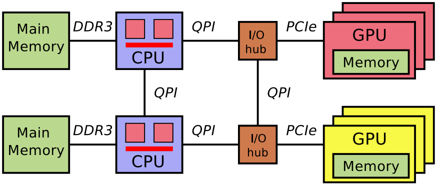
Hybrid Multicore & Multi-GPU System

Motivation

- Hybrid Multicores+GPUs presents challenges
  - Parallel programming is hard
  - Load balancing problem
    - Heterogeneity: processor, memory, etc.
    - Hierarchical levels of parallelism: node, socket, core, etc.
  - and others



Hybrid Clusters



Hybrid Multicore & Multi-GPU System

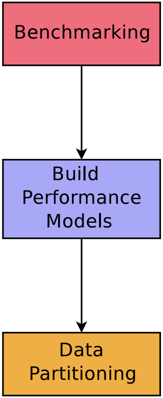
Leveraging Hybrid Multicores/GPUs System

In this work, we target:

- Data parallel application
    - Divisible computational workload
    - Workload proportional to data size
    - Dependent on data locality
- Dedicated hybrid system
- Reuse of optimized software stack

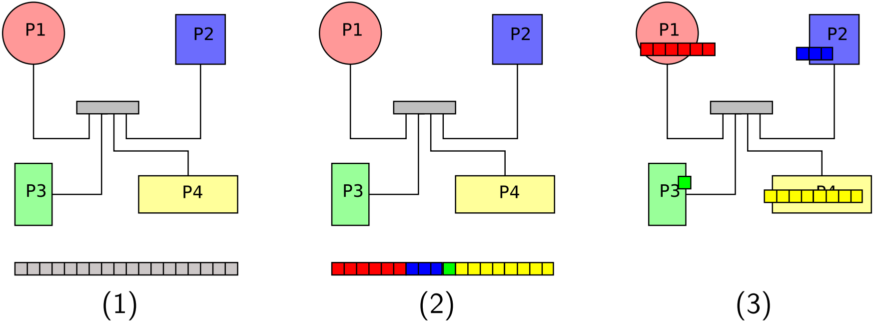
Our approach:

- Heterogeneous distributed-memory system
- Performance modeling of hybrid system
- Model-based data partitioning to balance load



Processing Flow

Data Partitioning on Heterogeneous Platform:



1. Workload is divisible and proportional to data size
2. Workload is partitioned in proportion to processor speed
3. Workload is distributed in proportion to processor speed

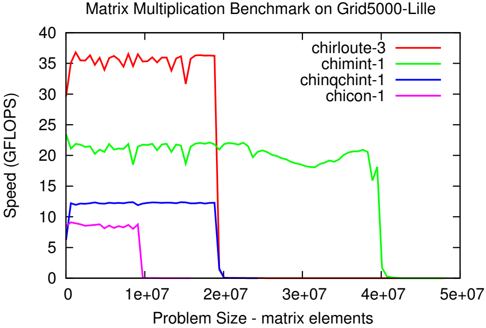
Data partitioning relies on accurate performance models

Traditionally, performance is defined by a single constant number

- Constant Performance Model (CPM)
- Computed from clock speed or by performing a benchmark
- Computational units are partitioned as $d_i = N \times (s_i / \sum_{j=1}^{p} s_j)$
- Simplistic, algorithms may fail to converge to a balanced solution[1]

Functional Performance Model (FPM)[2]:

- Represent speed as a function of problem size
- Realistic
- Application centric
- Hardware specific



[1] D. Clarke et al: Dynamic Load Balancing of Parallel Iterative Routines on Heterogeneous HPC Platforms, 2010

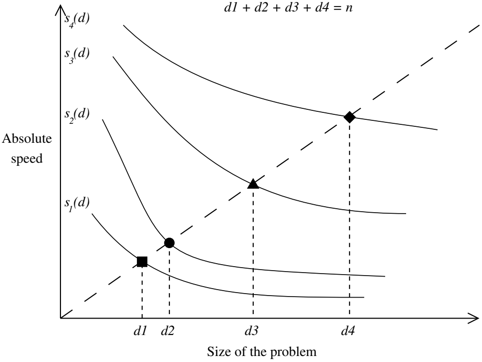
[2] A. Lastovetsky et al: Data partitioning with a functional performance model of heterogeneous processors, 2007.

Partitioning with functional performance models*

Load is balanced when:

$$t_1(d_1) \approx t_2(d_2) \approx \ldots \approx t_p(d_p)$$

$$\begin{cases} t_i(d_i) = d_i / s_i(d_i), \\ d_1 + d_2 + \ldots + d_p = N \end{cases}$$



- All processors complete work within the same time
- Solution lies on a line passing through the origin when $d_i / s_i(d_i) = \textit{constant}$
- However, only designed for heterogeneous uniprocessor cluster

* A. Lastovetsky et al: Data partitioning with a functional performance model of heterogeneous processors, 2007.

Outline

1 Model-based Data Partitioning

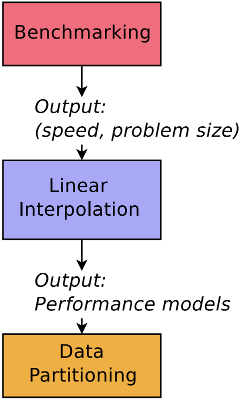
2 Data Partitioning on Hybrid System

3 Experiment Results

4 Conclusions and Future Work

Performance Modeling of Hybrid System

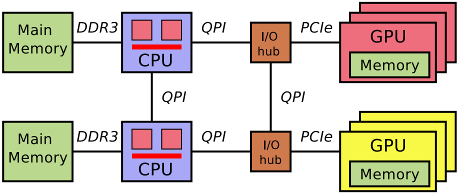
- Multicore/GPUs are modeled independently
  - Separate memory, programming models
  - Represented by speed functions (FPM)
  - Benchmarking with computational kernels
- Performance model of multicore:
  - Approximate the speed of multiple cores
  - e.g. all cores in a processor except the ones dedicated to GPUs
- Performance model of GPU:
  - Approximate combined speed of a GPU and it's dedicated core



Processing Flow

Performance Measurement of Hybrid System

- Generic measurement techniques
  - Process binding - avoid process migration
  - Synchronization - ensure resources are shared between cores
  - Repeating measurement- ensure statistically reliable results
- However, how to measure the processor performance accurately on a hybrid system?

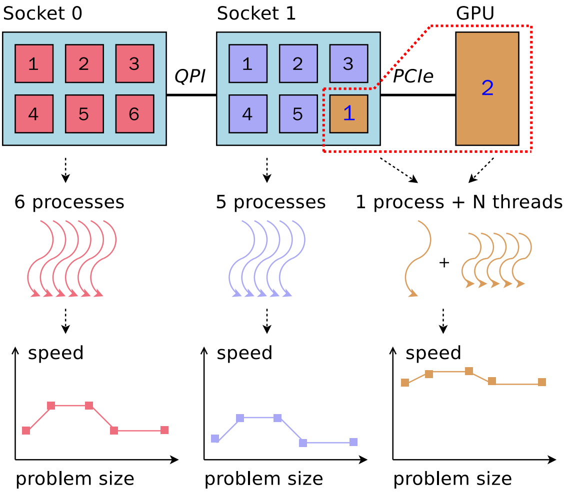


Hybrid Multicore & Multi-GPU System

Performance Measurement of Hybrid System

Performance measurement of multiple cores:

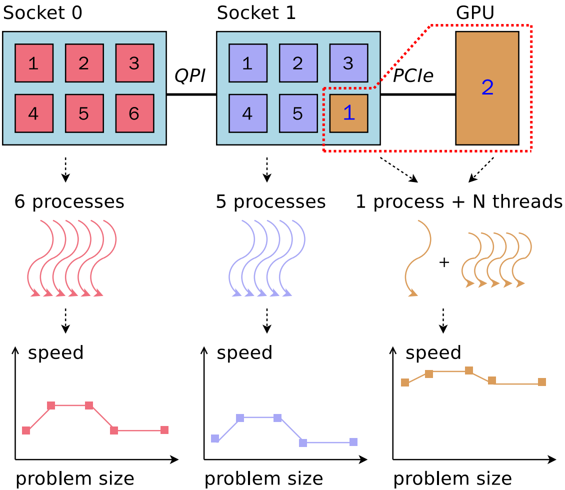
- Programming model: one process (thread) per core to achieve high performance
- Cores interfere with each other due to resource contention
- Performance are evaluated in group
- All cores in the group executing the same amount of workload in parallel



Performance Measurement of Hybrid System

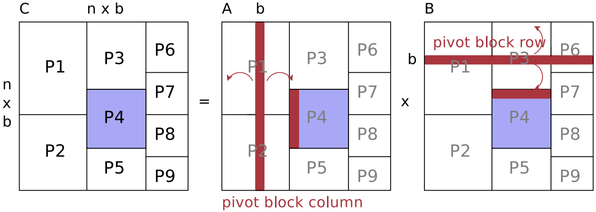
Performance measurement of GPU:

- One core dedicated to the GPU, other cores being idle
- Kernel computation time and data transfer time are both included
- Additional issue: Host NUMA affects PCIe transfer throughput in Dual-IOH system

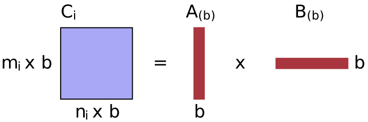


Application: Matrix Multiplication on Heterogeneous Platform*

- Matrices partitioned unevenly to achieve load balancing
- Processors arranged so that communication is minimized



- Computational kernel: panel-panel update
  - Reuse vendor-optimized GEMM routine
  - Computation is proportional to the area of submatrix C_i
  - The same memory access pattern as the whole application



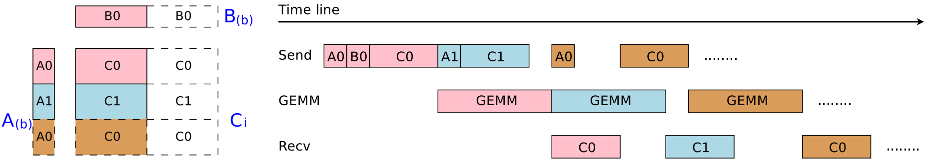
* Beaumont,O. et al: Matrix Multiplication on Heterogeneous Platforms. IEEE Trans. Parallel Distrib. Syst. 2001.

Development of Computational Kernel

- Multicore CPU:
  - Use GEMM routine from ACML library
  - Multiple processes running sequential routine
  - Alternative: Single process running threaded routine
- GPU accelerator:
  - Use GEMM routine from CUBLAS library
  - Develop out-of-core kernel to overcome memory limitation
  - Overlap data transfers and kernel execution to hide latency

Out-of-core Kernel, Overlap of Data Transfers and Kernel Execution:

- allocated 5 buffers in device memory: A0, A1, B0, C0, C1



Experimental Setup

Hybrid Multicore and Multi-GPU Node

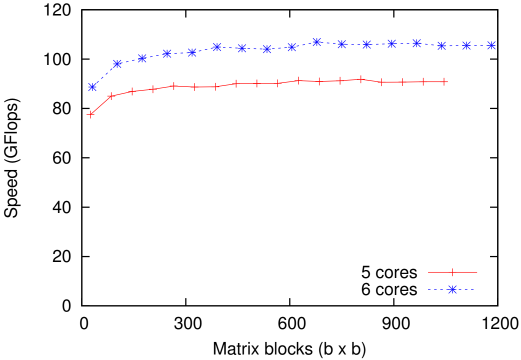
	CPU (AMD)	GPUs (NVIDIA)	
Architecture	Opteron 8439SE	GF GTX680	Tesla C870
Core Clock	2.8 GHz	1006 MHz	600 MHz
Number of Cores	4×6 cores	1536 cores	128 cores
Memory Size	4×16 GB	2048 MB	1536 MB
Memory Bandwidth		192.3 GB/s	76.8 GB/s

Building Functional Performance Models (FPMs)

- $s_c(x)$: approximate the speed of multiple cores when executing CPU kernel on c cores simultaneously, with problem size x/c one each core
- $g(x)$: approximate the combined speed of a GPU and it's dedicated core

Functional Performance Models of multicore CPU:

- Platform: consist of four 6-core sockets
- Modeling performance of cores in each socket
- $s_5(x)$, 5 cores running CPU kernel, 1 core being idle
- $s_6(x)$, all 6 cores running CPU kernel

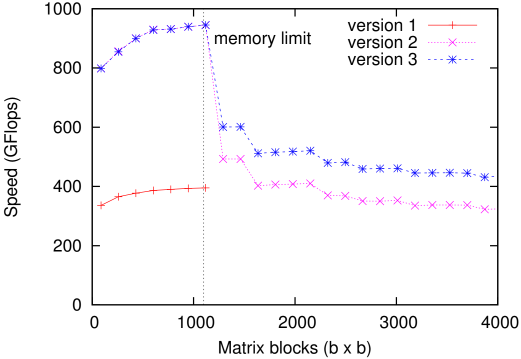


Building Functional Performance Models (FPMs)

- $s_c(x)$: approximate the speed of multiple cores when executing CPU kernel on c cores simultaneously, with problem size x/c one each core
- $g(x)$: approximate the combined speed of a GPU and it's dedicated core

Functional Performance Model of GPU:

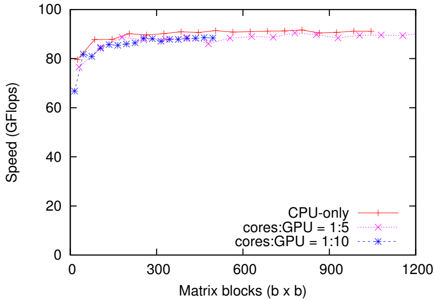
- version 1: naive implementation
- version 2: accumulate intermediate result, out-of-core overcome memory limitation
- version 3: overlap data transfers and kernel execution time



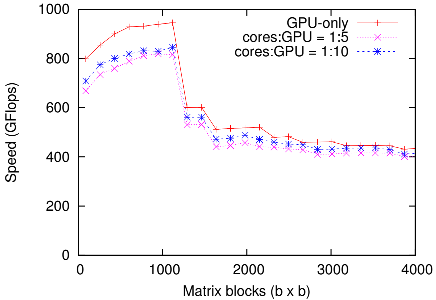
Impact of Resource Contention to Performance Modeling

- CPU and GPU kernel benchmarking simultaneously in a socket
- FPM of multiple cores $s_5(x)$ are barely affected
- FPM of GPU $g(x)$ gets 85% accuracy (speed drops by 7 - 15%)

$s_5(x)$, speed of multiple cores



$g(x)$, speed of a GPU



Note: the above two figures have different scales, 1:10

Outline

Execution time of the application under different configurations

Matrix size (blks)	CPUs (sec)	GTX680 (sec)	Hybrid-FPM (sec)
40×40	99.5	74.2	26.6
50×50	195.4	162.7	77.8
60×60	300.1	316.8	114.4
70×70	491.6	554.8	226.1

Column 1: block size is 640×640
Column 2: 24 CPU cores, homogeneous data partitioning
Column 3: 1 CPU core + 1 GPU
Column 4: 24 CPU cores + 2 GPUs, FPM-based data partitioning

Execution time of the application under different configurations

Matrix size (blks)	CPUs (sec)	GTX680 (sec)	Hybrid-FPM (sec)
40×40	99.5	74.2	26.6
50×50	195.4	162.7	77.8
60×60	300.1	316.8	114.4
70×70	491.6	554.8	226.1

Column 1: block size is 640×640
Column 2: 24 CPU cores, homogeneous data partitioning
Column 3: 1 CPU core + 1 GPU
Column 4: 24 CPU cores + 2 GPUs, FPM-based data partitioning

Execution time of the application under different configurations

Matrix size (blks)	CPUs (sec)	GTX680 (sec)	Hybrid-FPM (sec)
40×40	99.5	74.2	26.6
50×50	195.4	162.7	77.8
60×60	300.1	316.8	114.4
70×70	491.6	554.8	226.1

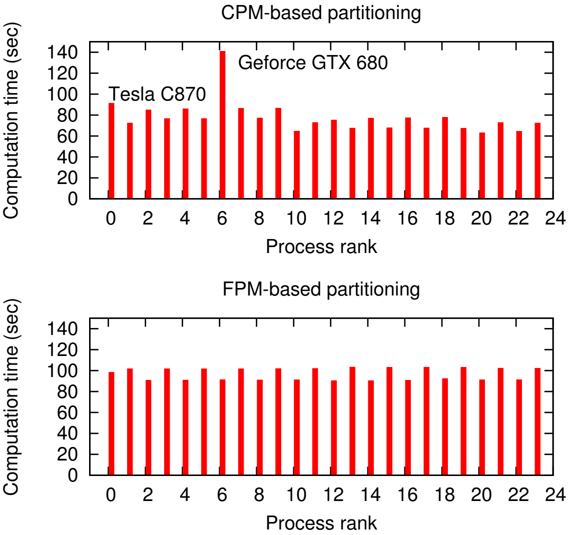
Column 1: block size is 640×640
Column 2: 24 CPU cores, homogeneous data partitioning
Column 3: 1 CPU core + 1 GPU
Column 4: 24 CPU cores + 2 GPUs, FPM-based data partitioning

Execution time of the application under different configurations

Matrix size (blks)	CPUs (sec)	GTX680 (sec)	Hybrid-FPM (sec)
40×40	99.5	74.2	26.6
50×50	195.4	162.7	77.8
60×60	300.1	316.8	114.4
70×70	491.6	554.8	226.1

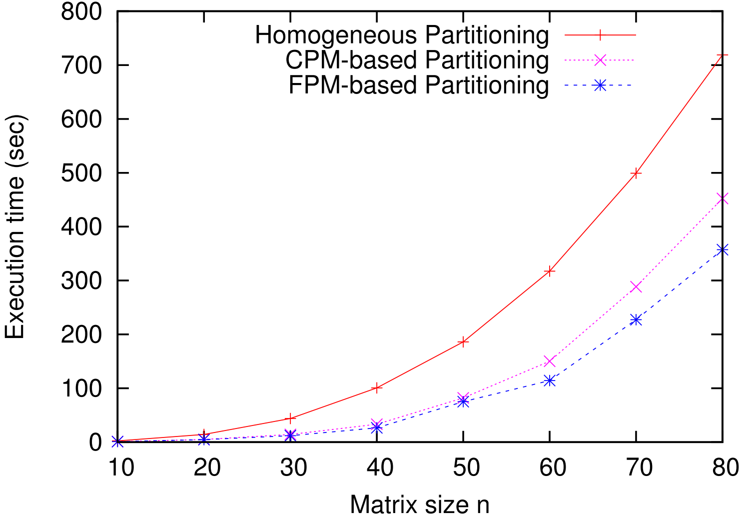
Column 1: block size is 640×640
Column 2: 24 CPU cores, homogeneous data partitioning
Column 3: 1 CPU core + 1 GPU
Column 4: 24 CPU cores + 2 GPUs, FPM-based data partitioning

Computation time of each process



Matrix size 60×60, Computation time reduced by 40%

Execution time of the application under different partitioning algorithms



Execution time reduced by 23% and 45% respectively

Conclusions and Future Work

- Conclusions
  - Defined and built functional performance models (FPMs) of hybrid multicore and multi-GPU system, considering it as a distributed memory system
  - Adapted FPM-based data partitioning to hybrid system, achieved load balancing and delivered good performance
- Future Work
  - Apply the approach to hybrid cluster
  - Partitioning with respect to interconnect speed

Thank You!

Science Foundation Ireland

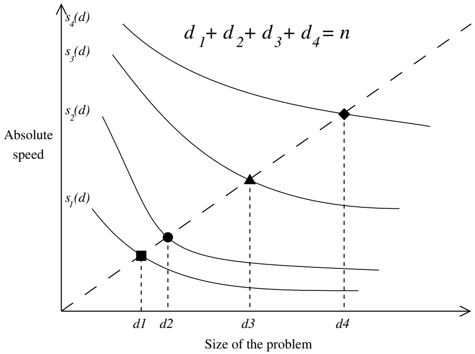
University College Dublin

Heterogeneous Computing Laboratory

China Scholarship Council

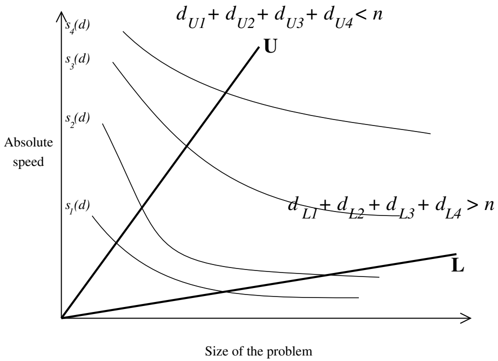
Partitioning with functional performance models

- Want all devices to compute assigned workload d_i within same time.
- Points $\left(d_i, s_i(d_i)\right)$ lie on a line passing through the origin when $\frac{d_i}{s_i(d_i)} = constant$.
- Total problem size determines the slope.
- Algorithm iteratively bisects solution space to find values d_i.



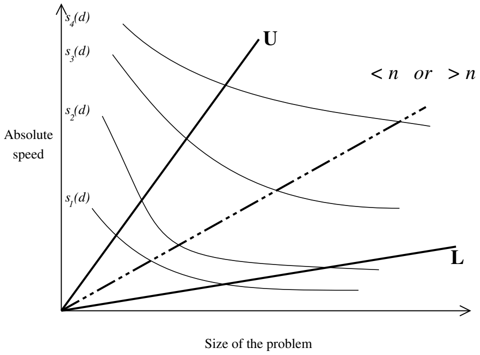
Partitioning with functional performance models

- Want all devices to compute assigned workload d_i within same time.
- Points $\left(d_i, s_i(d_i)\right)$ lie on a line passing through the origin when $\frac{d_i}{s_i(d_i)} = constant$.
- Total problem size determines the slope.
- Algorithm iteratively bisects solution space to find values d_i.



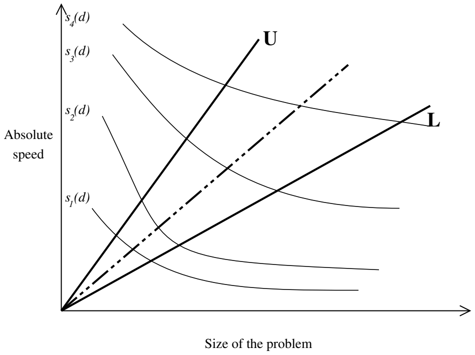
Partitioning with functional performance models

- Want all devices to compute assigned workload d_i within same time.
- Points $\left(d_i, s_i(d_i)\right)$ lie on a line passing through the origin when $\frac{d_i}{s_i(d_i)} = constant$.
- Total problem size determines the slope.
- Algorithm iteratively bisects solution space to find values d_i.



Partitioning with functional performance models

- Want all devices to compute assigned workload d_i within same time.
- Points $(d_i, s_i(d_i))$ lie on a line passing through the origin when $\frac{d_i}{s_i(d_i)} = constant$.
- Total problem size determines the slope.
- Algorithm iteratively bisects solution space to find values d_i.



Partitioning with functional performance models

- Want all devices to compute assigned workload d_i within same time.
- Points $(d_i, s_i(d_i))$ lie on a line passing through the origin when $\frac{d_i}{s_i(d_i)} = constant$.
- Total problem size determines the slope.
- Algorithm iteratively bisects solution space to find values d_i.