Optimization of data parallel applications for heterogeneous and hierarchical HPC platforms based on multicores and multi-GPUs

Alexey Lastovetsky

Heterogeneous Computing Laboratory
University College Dublin, Ireland

Parallel Processing and Applied Mathematics
Warsaw, Poland – September 8-11, 2013
Acknowledgment

- David Clarke, UCD
- Aleksandar Ilic, TU Lisbon
- Vladimir Rychkov, UCD
- Leonel Sousa, TU Lisbon
- Ziming Zhong, UCD
Introduction

- Modern HPC platform = complex system of highly heterogeneous devices and links
- How to execute data parallel applications efficiently?

Hybrid Multicore & Multi-GPU Node

Interconnected Hybrid Clusters
Modern HPC platform = complex system of highly heterogeneous devices and links

How to execute data parallel applications efficiently?

Traditional heterogeneous clusters: balance the load of relatively independent processors and optimize communications

Load balancing for data parallel applications = data partitioning

Hybrid Multicore & Multi-GPU Node

Interconnected Hybrid Clusters
Modern HPC platform = complex system of highly heterogeneous devices and links

How to execute data parallel applications efficiently?

Traditional heterogeneous clusters: balance the load of relatively independent processors and optimize communications

Load balancing for data parallel applications = data partitioning

How to apply data partitioning to multicore/multi-GPU?

Compute devices are more tightly coupled (and less independent), as resources are shared between devices
Our target:

- Data parallel application
  - Divisible computational workload
  - Workload proportional to data size
- Dedicated hybrid system
- Reuse of optimized software stack
Our target:

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

Our approach:

- Partitioning devices into independent groups
  - Each group = abstract processor
    - May be uni- or multi-processor depending on software kernel
- Accurate performance modeling of the abstract processors
- Model-based data partitioning between the heterogeneous abstract processors
Outline

1 Introduction
2 Background
3 Programming Models for Hybrid Systems
4 Performance Modeling on Hybrid Node
5 Applications: Linear Algebra
6 Matrix multiplication on hybrid node
7 Data partitioning on heterogeneous cluster of hybrid nodes
8 Conclusion
Background

Data Partitioning on Heterogeneous Platform

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 $d_i = N \times \left( \frac{s_i}{\sum_{j=1}^{p} s_j} \right)$
- Simplistic, algorithms may fail to converge to a balanced solution [1]

Functional Performance Model (FPM):

- Represent speed as a function of problem size [2]
- Realistic
- Application centric
- Hardware specific

Partitioning with functional performance models*

Load is balanced when:

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

$$t_i(d_i) = \frac{d_i}{s_i(d_i)},$$

$$d_1 + d_2 + \ldots + d_p = N$$

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

FPM-based data partitioning algorithm

- Total problem size determines the slope
- Algorithm iteratively bisects solution space to find values $d_i$

Size of the problem

$\begin{align*}
\sum_{i=1}^{4} d_i &= n \\
&= d_1 + d_2 + d_3 + d_4
\end{align*}$
FPM-based data partitioning algorithm

- Total problem size determines the slope
- Algorithm iteratively bisects solution space to find values $d_i$

\[
\begin{align*}
    d_{U1} + d_{U2} + d_{U3} + d_{U4} &< n \\
    d_{L1} + d_{L2} + d_{L3} + d_{L4} &> n
\end{align*}
\]
FPM-based data partitioning algorithm

- Total problem size determines the slope
- Algorithm iteratively bisects solution space to find values $d_i$
FPM-based data partitioning algorithm

- Total problem size determines the slope
- Algorithm iteratively bisects solution space to find values $d_i$
FPM-based data partitioning algorithm

- Total problem size determines the slope
- Algorithm iteratively bisects solution space to find values $d_i$
Dynamic FPM-based data partitioning

Functional Performance Models may be built:
- exhaustively in advance
- dynamically at run time
Dynamic FPM-based data partitioning

Functional Performance Models may be built:
- exhaustively in advance
- dynamically at run time
Dynamic FPM-based data partitioning

Functional Performance Models may be built:

- exhaustively in advance
- dynamically at run time

Initial: point \((n/p, s_i^0)\) with speed

\[ s_i^0 = \frac{n/p}{t_i(n/p)} \]

first function approximation \(s'_i(x) \equiv s_i^0\)
Dynamic FPM-based data partitioning

Functional Performance Models may be built:

- exhaustively in advance
- dynamically at run time

Initial: point \((n/p, s^0_i)\) with speed 
\[
s^0_i = \frac{n/p}{t_i(n/p)}
\]

first function approximation 
\[
s'_i(x) \equiv s^0_i
\]
Functional Performance Models may be built:

- exhaustively in advance
- dynamically at run time

Initial: point \((n/p, s^0_i)\) with speed

\[ s^0_i = \frac{n/p}{t_i(n/p)} \]

first function approximation \(s'_i(x) \equiv s^0_i\)

Iterations: point \((d^k_i, s^k_i)\) with speed

\[ s^k_i = \frac{d^k_i}{t_i(d^k_i)} \]

approximation \(s'_i(x)\) updated by adding the point
Functional Performance Models may be built:

- exhaustively in advance
- dynamically at run time

Initial: point \((n/p, s_i^0)\) with speed 
\[ s_i^0 = \frac{n/p}{t_i(n/p)} \]

first function approximation \(s'_i(x) \equiv s_i^0\)

Iterations: point \((d_i^k, s_i^k)\) with speed 
\[ s_i^k = \frac{d_i^k}{t_i(d_i^k)} \]

approximation \(s'_i(x)\) updated by adding the point

![Predicted Performance](image)
Dynamic FPM-based data partitioning

Functional Performance Models may be built:
- exhaustively in advance
- dynamically at run time

Initial: point \( \left( \frac{n}{p}, s^0_i \right) \) with speed \( s^0_i = \frac{n}{p} t_i \left( \frac{n}{p} \right) \)

First function approximation \( s_i'(x) \equiv s^0_i \)

Iterations: point \( \left( d^k_i, s^k_i \right) \) with speed \( s^k_i = \frac{d^k_i}{t_i \left( d^k_i \right)} \)

Approximation \( s_i'(x) \) updated by adding the point

\[ s_i(d) \]
\[ s_i'(d) \]
\[ (d'_i, s'_i) \]
\[ (d'_i, s'_i) \]

Absolute speed

Size of the problem
Dynamic FPM-based data partitioning

Functional Performance Models may be built:
- exhaustively in advance
- dynamically at run time

Initial: point \((n/p, s^0_i)\) with speed 
\[
s^0_i = \frac{n/p}{t_i(n/p)}
\]
first function approximation \(s'_i(x) \equiv s^0_i\)

Iterations: point \((d^k_i, s^k_i)\) with speed 
\[
s^k_i = \frac{d^k_i}{t_i(d^k_i)}
\]
approximation \(s'_i(x)\) updated by adding the point
Dynamic FPM-based data partitioning

Functional Performance Models may be built:
- exhaustively in advance
- dynamically at run time

Initial: point \((n/p, s^0_i)\) with speed 
\[ s^0_i = \frac{n/p}{t_i(n/p)} \]
first function approximation 
\[ s'_i(x) \equiv s^0_i \]

Iterations: point \((d^k_i, s^k_i)\) with speed 
\[ s^k_i = \frac{d^k_i}{t_i(d^k_i)} \]
approximation 
\[ s'_i(x) \] updated by adding the point

Models Updated

Absolute speed

Size of the problem
Dynamic FPM-based data partitioning

Functional Performance Models may be built:

- exhaustively in advance
- dynamically at run time

Initial: point \((n/p, s_i^0)\) with speed 
\[ s_i^0 = \frac{n/p}{t_i(n/p)} \]

first function approximation \(s'_i(x) \equiv s_i^0\)

Iterations: point \((d_i^k, s_i^k)\) with speed 
\[ s_i^k = \frac{d_i^k}{t_i(d_i^k)} \]

approximation \(s'_i(x)\) updated by adding the point
Dynamic FPM-based data partitioning

Functional Performance Models may be built:

- exhaustively in advance
- dynamically at run time

**Initial:** point \((n/p, s_i^0)\) with speed 
\[ s_i^0 = \frac{n/p}{t_i(n/p)} \]
first function approximation \(s'_i(x) \equiv s_i^0\)

**Iterations:** point \((d_i^k, s_i^k)\) with speed 
\[ s_i^k = \frac{d_i^k}{t_i(d_i^k)} \]
approximation \(s'_i(x)\) updated by adding the point

---

**Diagram:**
- Actual Performance
- Absolute speed
- Size of the problem

*Note: Diagram shows a graph with two curves representing actual performance and approximation, with points indicating iterations.*
Dynamic FPM-based data partitioning

Functional Performance Models may be built:

- exhaustively in advance
- dynamically at run time

Initial: point \((n/p, s_i^0)\) with speed \(s_i^0 = \frac{n/p}{t_i(n/p)}\)

First function approximation \(s'_i(x) \equiv s_i^0\)

Iterations: point \((d_i^k, s_i^k)\) with speed \(s_i^k = \frac{d_i^k}{t_i(d_i^k)}\)

Approximation \(s'_i(x)\) updated by adding the point
Dynamic FPM-based data partitioning

Functional Performance Models may be built:

- exhaustively in advance
- dynamically at run time

Initial: point \((n/p, s_i^0)\) with speed \(s_i^0 = \frac{n/p}{t_i(n/p)}\)

first function approximation \(s_i'(x) \equiv s_i^0\)

Iterations: point \((d_i^k, s_i^k)\) with speed \(s_i^k = \frac{d_i^k}{t_i(d_i^k)}\)

approximation \(s_i'(x)\) updated by adding the point

\[
\frac{n\,t_i(n/p)}{t_i(n/p)} = \frac{n}{p}
\]
Outline

1. Introduction
2. Background
3. Programming Models for Hybrid Systems
4. Performance Modeling on Hybrid Node
5. Applications: Linear Algebra
6. Matrix multiplication on hybrid node
7. Data partitioning on heterogeneous cluster of hybrid nodes
8. Conclusion
Programming Models for Hybrid Systems

- Data-parallel MPI program with calls to MT and GPGPU kernels
  - Hierarchical or flat execution on the cluster of hybrid nodes
- Partitioning compute devices of the node into independent groups
  - Identical cores
    - Running optimized MT kernel
    - Running multiple single-threaded kernels (one per core)
  - Core + GPU
    - Running native GPGPU kernel
    - Running out-of-core version of native GPGPU kernel
  - Identical core+GPU pairs
    - Running multiple native GPGPU kernels
  - Core + multi-GPU
    - Running multi-GPU kernel
Assumptions about program configuration

- No idle compute devices
  - May not be the optimal configuration (out of scope of this study)
  - May affect the independence of groups
- Even load of identical abstract processors
  - No evidence that uneven load will improve performance
- One-to-one mapping of processes/threads to compute devices
  - No evidence that many-to-one will improve performance
- Same one-to-one mapping for all runs of the program
  - The mapping is not delegated to the operating environment
3 groups of devices: 6 cores, 5 cores and 1 core + GPU
- Cores in one group interfere with each other due to resource contention
- All cores in the group execute the same amount of workload in parallel
- Kernel computation time and data transfer time are both included
- Host core for GPU is chosen to maximize data throughput between GPU and NUMA memory
Outline

1. Introduction
2. Background
3. Programming Models for Hybrid Systems
4. Performance Modeling on Hybrid Node
5. Applications: Linear Algebra
6. Matrix multiplication on hybrid node
7. Data partitioning on heterogeneous cluster of hybrid nodes
8. Conclusion
Functional Performance Models of multicore

- $s(x)$ speed of a core executing a single-threaded kernel exclusively
  
  \[ s(x) = \frac{x}{t} \]
Functional Performance Models of multicore

- $s(x)$ speed of a core executing a single-threaded kernel exclusively
  \[ s(x) = \frac{x}{t} \]
- $s_c(x)$ speed of a core that executes a single-threaded kernel and shares the system resources with identical cores, each core receives $x$ units
  \[ s_c(x) = \frac{x}{\max_1^c(t_i)} \]
Functional Performance Models of multicore

- $s(x)$ speed of a core executing a single-threaded kernel exclusively
  $s(x) = x/t$

- $s_c(x)$ speed of a core that executes a single-threaded kernel and shares the system resources with identical cores, each core receives $x$ units
  $s_c(x) = x/\max_1^c(t_i)$

- $S_c(x)$ speed of $c$ cores that execute a multi-threaded kernel and share system resources, $x$ units distributed between cores
  $S_c(x) = x/t$
Functional Performance Models of multicore: Example

- $S_5(x)$: 5-threaded kernel on a socket, 1 core idle
- $S_6(x)$: 6-threaded kernel on a socket
Performance Modeling on Hybrid Node

Functional Performance Models of GPU

- \( g(x) \): combined speed of a GPU and its dedicated core, exclusive PCIe
  \[ g(x) = \frac{x}{t} \]
Functional Performance Models of GPU

- $g(x)$: combined speed of a GPU and its dedicated core, exclusive PCIe
  
  $g(x) = x/t$

- $g_d(x)$: combined speed of a GPU and its dedicated core, that share PCIe with identical pairs of processors, each pair receives $x$ computation units
  
  $g_d(x) = x/\max_1^d(t_i)$
Functional Performance Models of GPU

- $g(x)$: combined speed of a GPU and its dedicated core, exclusive PCIe
  $g(x) = \frac{x}{t}$

- $g_d(x)$ combined speed of a GPU and its dedicated core, that share PCIe with identical pairs of processors, each pair receives $x$ computation units
  $g_d(x) = \frac{x}{\text{max}_1(t_i)}$

- $G_d(x)$ combined speed of $d$ GPUs and a dedicated CPU core that execute a multi-GPU kernel and share PCIe, $x$ computation units are distributed between GPUs
  $G_d(x) = \frac{x}{t}$
Functional Performance Models of GPU: Example

- $g(x)$ (version 1): naive kernel
- $g(x)$ (version 2): accumulate intermediate result + out-of-core
- $g(x)$ (version 3): version 2 + overlap data transfers and kernel executions
CPU and GPU kernels benchmarked simultaneously on a socket
- FPM of multiple cores $S_5(x)$ is 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
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

Benchmarking

Output: (speed, problem size)

Linear Interpolation

Output: Performance models

Data Partitioning

Alexey Lastovetsky (UCD HCL) Optimization of data parallel applications for multi-CPU/GPU PPAM 2013 20 / 41
Outline

1. Introduction
2. Background
3. Programming Models for Hybrid Systems
4. Performance Modeling on Hybrid Node
5. Applications: Linear Algebra
6. Matrix multiplication on hybrid node
7. Data partitioning on heterogeneous cluster of hybrid nodes
8. Conclusion
Applications: Linear Algebra

Linear Algebra applications:

- Matrix multiplication
- LU decomposition
- Jacobi iterative method
- ...

How to optimally partition matrices?

- Partition matrices between nodes
- Sub-partition between devices within a node
- To achieve load balancing, partition with respect to device and node speed
- Minimise total volume of communication
Matrix Partitioning

Simple Partitioning

\[ \begin{array}{c}
\mathbf{P}_1 \\
\mathbf{P}_2 \\
\vdots \\
\mathbf{P}_i \\
\vdots \\
\mathbf{P}_n 
\end{array} \]

2D Partitioning

\[ \begin{array}{c}
\mathbf{P}_1 \\
\vdots \\
\mathbf{P}_i \\
\vdots \\
\mathbf{P}_n 
\end{array} \]

\[ \begin{array}{c}
\mathbf{m}_i \\
\mathbf{n}_i 
\end{array} \]
Matrix Multiplication on Heterogeneous Platform*

- Input: constant processor speeds
- Matrices partitioned so that
  - Area of the rectangle proportional to the speed
  - Volume of communication minimized


Matrix Multiplication on Heterogeneous Platform*

- Input: constant processor speeds
- Matrices partitioned so that
  - Area of the rectangle proportional to the speed
  - Volume of communication minimized

More accurate solution is based on speed functions as input**

---


Matrix Multiplication on Heterogeneous Platform

- Computational kernel: panel-panel update
Matrix Multiplication on Heterogeneous Platform

- Computational kernel: panel-panel update

- Processor speed - function of area
  
  Built by running the kernel for square matrices
Matrix Multiplication on Heterogeneous Platform

- **Computational kernel**: panel-panel update

- **Processor speed - function of area**
  *Built by running the kernel for square matrices*

- **FPM-based partitioning algorithm** finds the optimal areas
  *The areas are used as input to the matrix partitioning algorithm*
Outline

1. Introduction
2. Background
3. Programming Models for Hybrid Systems
4. Performance Modeling on Hybrid Node
5. Applications: Linear Algebra
6. Matrix multiplication on hybrid node
7. Data partitioning on heterogeneous cluster of hybrid nodes
8. Conclusion
Matrix multiplication on hybrid node

Experimental platform

<table>
<thead>
<tr>
<th></th>
<th>CPU (AMD)</th>
<th>GPUs (NVIDIA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Opteron 8439SE</td>
<td>GF GTX680</td>
</tr>
<tr>
<td>Core Clock</td>
<td>2.8 GHz</td>
<td>1006 MHz</td>
</tr>
<tr>
<td>Number of Cores</td>
<td>4 × 6 cores</td>
<td>1536 cores</td>
</tr>
<tr>
<td>Memory Size</td>
<td>4 × 16 GB</td>
<td>2048 MB</td>
</tr>
<tr>
<td>Memory Bandwidth</td>
<td>192.3 GB/s</td>
<td>76.8 GB/s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Computational Kernels for Hybrid Node

- **Multicore CPU:**
  - GEMM routine from ACML library
  - Multi-threaded processes (one per socket)

- **GPU accelerator:**
  - 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

Diagram:

- A0, B0, C0, A1, C1
- Time line:
  - Send: A0 B0 C0 A1 C1
  - GEMM: GEMM GEMM
  -Recv: C0 C1 C0

Alexey Lastovetsky (UCD HCL)  Optimization of data parallel applications for multi-CPU/GPU  PPAM 2013
Experiments on hybrid multicore multi-GPU node

Execution time of the application under different configurations

<table>
<thead>
<tr>
<th>Matrix size (blks)</th>
<th>CPUs (sec)</th>
<th>GTX680 (sec)</th>
<th>Hybrid-FPM (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>40 × 40</td>
<td>99.5</td>
<td>74.2</td>
<td>26.6</td>
</tr>
<tr>
<td>50 × 50</td>
<td>195.4</td>
<td>162.7</td>
<td>77.8</td>
</tr>
<tr>
<td>60 × 60</td>
<td>300.1</td>
<td>316.8</td>
<td>114.4</td>
</tr>
<tr>
<td>70 × 70</td>
<td>491.6</td>
<td>554.8</td>
<td>226.1</td>
</tr>
</tbody>
</table>

Column 1: block size is 640 × 640
Column 2: 4 × 6 CPU cores, homogeneous data partitioning
Column 3: CPU core + GPU
Column 4: 2 × 6 CPU cores + 2 × 5 CPU cores + 2 × (CPU core + GPU), FPM-based data partitioning
Experiments on hybrid multicore multi-GPU node

Execution time of the application under different configurations

<table>
<thead>
<tr>
<th>Matrix size (blks)</th>
<th>CPUs (sec)</th>
<th>GTX680 (sec)</th>
<th>Hybrid-FPM (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>40 × 40</td>
<td>99.5</td>
<td>74.2</td>
<td>26.6</td>
</tr>
<tr>
<td>50 × 50</td>
<td>195.4</td>
<td>162.7</td>
<td>77.8</td>
</tr>
<tr>
<td>60 × 60</td>
<td>300.1</td>
<td>316.8</td>
<td>114.4</td>
</tr>
<tr>
<td>70 × 70</td>
<td>491.6</td>
<td>554.8</td>
<td>226.1</td>
</tr>
</tbody>
</table>

Column 1: block size is 640 × 640
Column 2: 4 × 6 CPU cores, homogeneous data partitioning
Column 3: CPU core + GPU
Column 4: 2 × 6 CPU cores + 2 × 5 CPU cores + 2 × ( CPU core + GPU ), FPM-based data partitioning
Experiments on hybrid multicore multi-GPU node

Execution time of the application under different configurations

<table>
<thead>
<tr>
<th>Matrix size (blks)</th>
<th>CPUs (sec)</th>
<th>GTX680 (sec)</th>
<th>Hybrid-FPM (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>40 × 40</td>
<td>99.5</td>
<td>74.2</td>
<td>26.6</td>
</tr>
<tr>
<td>50 × 50</td>
<td>195.4</td>
<td>162.7</td>
<td>77.8</td>
</tr>
<tr>
<td>60 × 60</td>
<td>300.1</td>
<td>316.8</td>
<td>114.4</td>
</tr>
<tr>
<td>70 × 70</td>
<td>491.6</td>
<td>554.8</td>
<td>226.1</td>
</tr>
</tbody>
</table>

Column 1: block size is 640 × 640
Column 2: 4 × 6 CPU cores, homogeneous data partitioning
Column 3: CPU core + GPU
Column 4: 2 × 6 CPU cores + 2 × 5 CPU cores + 2 × (CPU core + GPU), FPM-based data partitioning
Experiments on hybrid multicore multi-GPU node

Execution time of the application under different configurations

<table>
<thead>
<tr>
<th>Matrix size (blks)</th>
<th>CPUs (sec)</th>
<th>GTX680 (sec)</th>
<th>Hybrid-FPM (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>40 × 40</td>
<td>99.5</td>
<td>74.2</td>
<td>26.6</td>
</tr>
<tr>
<td>50 × 50</td>
<td>195.4</td>
<td>162.7</td>
<td>77.8</td>
</tr>
<tr>
<td>60 × 60</td>
<td>300.1</td>
<td>316.8</td>
<td>114.4</td>
</tr>
<tr>
<td>70 × 70</td>
<td>491.6</td>
<td>554.8</td>
<td>226.1</td>
</tr>
</tbody>
</table>

Column 1: block size is 640 × 640
Column 2: 4 × 6 CPU cores, homogeneous data partitioning
Column 3: CPU core + GPU
Column 4: 2 × 6 CPU cores + 2 × 5 CPU cores + 2 × (CPU core + GPU), FPM-based data partitioning
Matrix multiplication on hybrid node

Computation time of each process

![Graph showing computation time for different processes and GPU models.

CPM-based partitioning

FPM-based partitioning

Matrix size $60 \times 60$, Computation time reduced by 40%

Alexey Lastovetsky (UCD HCL)
Optimization of data parallel applications for multi-CPU/GPU
PPAM 2013 30 / 41
Execution time reduced by 23% and 45% respectively
Outline

1. Introduction
2. Background
3. Programming Models for Hybrid Systems
4. Performance Modeling on Hybrid Node
5. Applications: Linear Algebra
6. Matrix multiplication on hybrid node
7. Data partitioning on heterogeneous cluster of hybrid nodes
8. Conclusion
Data partitioning on heterogeneous cluster of hybrid nodes

- **Target platform** - dedicated heterogeneous cluster of hybrid nodes
- **Hierarchical partitioning algorithm**
  - Dynamic algorithm - no a priori information about performance required.
  - **Inputs:**
    - Problem size
    - Number of nodes
    - Number of devices per node
    - Device type (e.g., cpu, gpu, ...).
  - Link computational kernel to be benchmarked for each device.
- Initially distribution is partitioned evenly between nodes and between devices within a node
- Algorithm converges towards optimum solution
Hierarchical Partitioning Algorithm

- $q$ nodes, $Q_1, \ldots, Q_q$.
- Node $Q_i$ has $p_i$ devices, $P_{i_1}, \ldots, P_{i_{p_i}}$.
- Hierarchy in platform $\rightarrow$ hierarchy in partitioning
  - Nested parallelism
  - *inter-node partitioning algorithm* (INPA)
  - *inter-device partitioning algorithm* (IDPA)
- IDPA is nested inside INPA
Hierarchical Partitioning Algorithm

- $W$ computational units to partition between nodes
- Inter-node partitioning algorithm (INPA) creates node-FPM’s and computes $w_1, \ldots, w_q$
  so that $w_1 + \ldots + w_q = W$. 
Hierarchical Partitioning Algorithm

- Communication minimising algorithm has input: $w_1, \ldots, w_q$ and output: $(m_1, n_1), \ldots, (m_q, n_q)$ such that $m_i \times n_i = w_i$ and matrix is completely tiled.
Hierarchical Partitioning Algorithm

- inter-device partitioning algorithm (IDPA) creates device-FPM’s and computes \( d_{i1}, \ldots, d_{ip} \), such that \( d_{i1} + \ldots + d_{ip} = bn_i \)
inter-device partitioning algorithm (IDPA) creates device-FPM’s and computes $d_{i1}, \ldots, d_{ip}$, such that $d_{i1} + \ldots + d_{ip} = bn_i$
Data partitioning on heterogeneous cluster of hybrid nodes

\[ w_i = m_i \times n_i \]

\[ \sum_{j=1}^{p} d_{ij} = b \times n_i \]

models of nodes

models of devices (from 1 node)
Data partitioning on heterogeneous cluster of hybrid nodes

\[ w_i = m_i \times n_i \]

\[ \sum_{j=1}^{p} d_{ij} = b \times n_i \]

models of nodes

models of devices
(from 1 node)
Data partitioning on heterogeneous cluster of hybrid nodes

\[ w_i = m_i \times n_i \]

\[ \sum_{j=1}^{P} d_{ij} = b \times n_i \]
Data partitioning on heterogeneous cluster of hybrid nodes

\[ w_i = m_i \times n_i \]

\[ \sum_{j=1}^{p} d_{ij} = b \times n_i \]
Data partitioning on heterogeneous cluster of hybrid nodes

\[ w_i = m_i \times n_i \]

\[ \sum_{j=1}^{p} d_{ij} = b \times n_i \]
Data partitioning on heterogeneous cluster of hybrid nodes

\[ w_i = m_i \times n_i \]

\[ \sum_{j=1}^{p} d_{ij} = b \times n_i \]
Data partitioning on heterogeneous cluster of hybrid nodes

\[ w_i = m_i \times n_i \]

\[ \sum_{j=1}^{p} d_{ij} = b \times n_i \]
Data partitioning on heterogeneous cluster of hybrid nodes

\[ w_i = m_i \times n_i \]

\[ \sum_{j=1}^{p} d_{ij} = b \times n_i \]
Data partitioning on heterogeneous cluster of hybrid nodes

\[ w_i = m_i \times n_i \]

\[ \sum_{j=1}^{p} d_{ij} = b \times n_i \]
Data partitioning on heterogeneous cluster of hybrid nodes

\[ w_i = m_i \times n_i \]

\[ \sum_{j=1}^{p} d_{ij} = b \times n_i \]
Data partitioning on heterogeneous cluster of hybrid nodes

- $w_i = m_i \times n_i$
- $\sum_{j=1}^{p} d_{ij} = b \times n_i$

models of nodes

$S(w_i)$

models of devices

(from 1 node)

$s(d_{ij})$

d_{ij} (for fixed $m_{ij}$)
Data partitioning on heterogeneous cluster of hybrid nodes

\[ w_i = m_i \times n_i \]

\[ \sum_{j=1}^{p} d_{ij} = b \times n_i \]
Data partitioning on heterogeneous cluster of hybrid nodes

\[ w_i = m_i \times n_i \]

\[ \sum_{j=1}^{p} d_{ij} = b \times n_i \]
Data partitioning on heterogeneous cluster of hybrid nodes

\[ w_i = m_i \times n_i \]

\[ \sum_{j=1}^{p} d_{ij} = b \times n_i \]
Data partitioning on heterogeneous cluster of hybrid nodes

\[ w_i = m_i \times n_i \]

\[ \sum_{j=1}^{p} d_{ij} = b \times n_i \]
Data partitioning on heterogeneous cluster of hybrid nodes

\[ w_i = m_i \times n_i \]

\[ \sum_{j=1}^{P} d_{ij} = b \times n_i \]
Data partitioning on heterogeneous cluster of hybrid nodes

$$w_i = m_i \times n_i$$

$$\sum_{j=1}^{p} d_{ij} = b \times n_i$$
Data partitioning on heterogeneous cluster of hybrid nodes

\[ w_i = m_i \times n_i \]

\[ \sum_{j=1}^{p} d_{ij} = b \times n_i \]
Data partitioning on heterogeneous cluster of hybrid nodes

\[ w_i = m_i \times n_i \]

\[ \sum_{j=1}^{p} d_{ij} = b \times n_i \]
Data partitioning on heterogeneous cluster of hybrid nodes

\[ w_i = m_i \times n_i \]

\[ \sum_{j=1}^{p} d_{ij} = b \times n_i \]

models of nodes

models of devices (from 1 node)
### Experimental Setup

90 Nodes from Grid5000 Grenoble site

<table>
<thead>
<tr>
<th>Cores:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>Nodes</th>
<th>Cores</th>
<th>GPUs</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adonis</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>12</td>
<td>48</td>
<td>12</td>
<td>2.27/2.4GHz Xeon, 24GB</td>
</tr>
<tr>
<td>Edel</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>50</td>
<td>250</td>
<td>0</td>
<td>2.27GHz Xeon, 24GB</td>
</tr>
<tr>
<td>Genepi</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>28</td>
<td>28</td>
<td>134</td>
<td>0</td>
<td>2.5GHz Xeon, 8GB</td>
</tr>
<tr>
<td>Total</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>90</td>
<td>432</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

- All nodes connected with InfiniBand communication network.
- Open MPI for inter node communication.
- OpenMP for inter-device parallelism.
Experimental Results

Performance models for nodes from Grid5000 Grenoble

- adonis 7CPU + 1GPU
- adonis 1CPU + 1GPU
- adonis 0CPU + 1GPU
- genepi 8CPU
- genepi 4CPU
- genepi 1CPU
- edel 8CPU
- edel 4CPU
- edel 1CPU

Problem Size $w_i$ ($128 \times 128$ blocks updated)

Speed (GFLOPS)
Data partitioning on heterogeneous cluster of hybrid nodes

Experimental Results

- Functional performance model (FPM): the proposed algorithm
- Multiple constant performance models (CPM): Redistribute based on previous benchmark.
- Single-CPM: One benchmark is preformed.
- Homogeneous distribution: Partitioned evenly between nodes, then evenly between devices within each node.
Conclusion

- 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 node, achieved load balancing and delivered good performance
- Adapted dynamic FPM-based data partitioning to hybrid cluster, achieved self-adaptiveness
Thank You!

University College Dublin
Heterogeneous Computing Laboratory
Science Foundation Ireland
China Scholarship Council

Instituto de Engenharia de Sistemas e Computadores
INSTITUTO SUPERIOR TÉCNICO
Universidade de Lisboa
Complex HPC EU COST Action IC0805