Architecture-Aware Cost Modelling for Parallel Performance Portability

Evgenij Belikov, Hans-Wolfgang Loidl, Greg Michaelson, Phil Trinder

ATPS’12, Berlin

February 27, 2012
1 Motivation and Goal

2 Architecture-Aware Cost Modelling

3 Experimental Design

4 Evaluation

5 Conclusions

6 Discussion
Perceived State of Affairs

- Trend towards heterogeneous and hierarchical architectures
Perceived State of Affairs

- Trend towards heterogeneous and hierarchical architectures
- Increasing complexity of software systems
Perceived State of Affairs

- Trend towards heterogeneous and hierarchical architectures
- Increasing complexity of software systems
- Parallel architectures are mainstream, parallel programming models lag behind
Motivation and Goal

Perceived State of Affairs

- Trend towards heterogeneous and hierarchical architectures
- Increasing complexity of software systems
- Parallel architectures are mainstream, parallel programming models lag behind
- Hardware evolves faster than software making manual code changes infeasible
Perceived State of Affairs

- Trend towards heterogeneous and hierarchical architectures
- Increasing complexity of software systems
- Parallel architectures are mainstream, parallel programming models lag behind
- Hardware evolves faster than software making manual code changes infeasible
- → Need for performance portability
Motivation and Goal

Problem Statement

On a large scale
To manage the complexity of future computing systems self-optimisation is crucial (in accord with the vision of Autonomic Computing)

The goal of the presented work
- Explore and quantify the impact of heterogeneity of modern parallel architectures on parallel programs
- Introduce a viable approach to parallel performance portability
Target Platforms and Configurations

Available architectures (we focus on clusters of multi-cores, MIMD):

<table>
<thead>
<tr>
<th>alias</th>
<th>arch (bit)</th>
<th>cores</th>
<th>speed (GHz)</th>
<th>L2 cache (MB)</th>
<th>RAM (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>linux_lab</td>
<td>32</td>
<td>2</td>
<td>3.40</td>
<td>2x2</td>
<td>3</td>
</tr>
<tr>
<td>linux01</td>
<td>32</td>
<td>2</td>
<td>2.40</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>lxpara</td>
<td>32</td>
<td>8</td>
<td>2.33</td>
<td>2x6</td>
<td>8</td>
</tr>
<tr>
<td>bwlf</td>
<td>64</td>
<td>8</td>
<td>2.00</td>
<td>2x4</td>
<td>6</td>
</tr>
</tbody>
</table>

Environment: CentOS 5.5 Linux, Gigabit Ethernet (linux01 100MBit)
Configurations: 1..16 processing elements (PEs)

- all-local vs. partly remote (emulated using netem)
- e.g. 2x4 2x2 4x1 (lxpara, bwlf, linux_lab)
General Idea and Parameter Choice

- More powerful PEs should receive more work (static load balancing)
- Architectural characteristics are used to estimate relative computational power of PEs
- Static, analytic models lead to negligible overhead

<table>
<thead>
<tr>
<th>Parameter/Variable Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
<td>number of PEs</td>
</tr>
<tr>
<td>$N$</td>
<td>data size</td>
</tr>
<tr>
<td>$CPU_i$</td>
<td>speed of processor $i$ in MHz</td>
</tr>
<tr>
<td>$L_2_i$</td>
<td>L2 cache size in KB</td>
</tr>
<tr>
<td>$RAM_i$</td>
<td>RAM size in MB</td>
</tr>
<tr>
<td>$L_i$</td>
<td>latency in ms</td>
</tr>
<tr>
<td>$C_i$</td>
<td>absolute chunk size in elements</td>
</tr>
<tr>
<td>$S_i$</td>
<td>relative individual computational power</td>
</tr>
<tr>
<td>$S_{all}$</td>
<td>relative cumulative computational power</td>
</tr>
<tr>
<td>$a, b, c, d$</td>
<td>scaling factors (default = 1)</td>
</tr>
</tbody>
</table>
Cost Models

Generic model:

\[ C_i = \frac{S_i \cdot N}{S_{all}}, \quad \text{where } S_{all} = \sum_{j=1}^{P} S_j \]

<table>
<thead>
<tr>
<th>Cost Model</th>
<th>( S_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline (no CM)</td>
<td>1</td>
</tr>
<tr>
<td>CM0</td>
<td>( CPU_i )</td>
</tr>
<tr>
<td>CM1</td>
<td>( aCPU_i \cdot bL2_i )</td>
</tr>
<tr>
<td>CM2</td>
<td>( aCPU_i \cdot \frac{bL2_i}{cRAM_i} )</td>
</tr>
<tr>
<td>CM3</td>
<td>( aCPU_i \cdot \frac{bL2_i}{cRAM_i} \cdot dL_i )</td>
</tr>
</tbody>
</table>
Representative Application: Sparse Matrix Multiplication

- Scheduling: Static
- Language: C + MPI
- Cost Model Integration: as a C function, informs partitioning
- Work Distribution: Master-Worker (Master also computes)
- Matrix is sparse if number of nonzero elements is < 30%
- Storage format: (row, col, value) array + management information
- Cache miss rate: 0.03% (cachegrind)
Application Structure: Master Process

- parse command line arguments
- generate matrices A and B
- gather parallel architecture descriptions
- broadcast matrix A to all PEs
- scatter parts of matrix B among the PEs
- perform local computation (matrix-vector multiplication)
- gather partial results from the PEs
- combine partial results
- (optional) display final result (matrix C)
Application Structure: Worker Process

- send architecture description to master
- receive matrix A from master
- receive part of matrix B from master
- perform local computation (matrix-vector multiplication)
- send results (part of matrix C) back to master
Focus of Enquiry

We measure the runtimes and compare relative speedups for different setups (baseline with no cost model vs. cost model-enhanced versions) on the most heterogeneous architecture regarding

- Impact of Heterogeneity
- Scalability
- Impact of Latency
Heterogeneity

![Graph showing the relationship between speedup and number of processors for different configurations.

- 2x8: lxpara
- 8x2: linux lab
- 1x8 2x2 4x1: lxpara, linux lab
- 2x4 2x2 4x1: lxpara, linux lab
- 2x4 2x2 4x1: lxpara, beowulf, linux01, linux lab

8192x8192 matrix, sparsity: 1%, nonzeros: 37,420,929]
Scalability (1)

- no cost model (baseline)
- cost model 0 (CPU speed)
- cost model 1 (CPU speed, L2 Cache)
- cost model 2 (CPU speed, L2 Cache, RAM)

8192x8192 matrix, sparsity: 1%, nonzeros: 37,420,929
2x4 2x2 4x1 unshuffled all-local (lxpara + beowulf + linux_lab)
Scalability (2)

- all cost models perform better than the baseline case
Scalability (2)

- all cost models perform better than the baseline case
- negligible overhead (only difference in granularity, since partitioning required in any case)
Scalability (2)

- all cost models perform better than the baseline case
- negligible overhead (only difference in granularity, since partitioning required in any case)
- cost models are effective in achieving performance portability almost reaching the performance of the homogeneous case
Scalability (2)

- all cost models perform better than the baseline case
- negligible overhead (only difference in granularity, since partitioning required in any case)
- cost models are effective in achieving performance portability almost reaching the performance of the homogeneous case
- the results indicate scalability beyond 16 PEs
Impact of Latency (1)

8192x8192 matrix, sparsity: 1%, nonzeros: 37,420,929, 2x4 2x2 4x1 unshuffled (2 remote PEs)
Impact of Latency (2)

- network characteristics are critical on partly remote setups
Impact of Latency (2)

- network characteristics are critical on partly remote setups
- do not send work to far remote PEs (avoid slowdown)
Impact of Latency (2)

- network characteristics are critical on partly remote setups
- do not send work to far remote PEs (avoid slowdown)
- send more work to compensate for communication overhead (ideally obtain additional speedup)
Impact of Latency (2)

- network characteristics are critical on partly remote setups
- do not send work to far remote PEs (avoid slowdown)
- send more work to compensate for communication overhead (ideally obtain additional speedup)
- → off-loading involves a trade-off
Findings

- performance portability eliminates the need of manual code changes to adapt to new architectures
Findings

- performance portability eliminates the need of manual code changes to adapt to new architectures
- using architecture-aware cost models is a viable approach to performance portability (on low-latency clusters of multi-cores, CPU speed and cache-to-RAM ratio are good predictors)
Findings

- performance portability eliminates the need of manual code changes to adapt to new architectures
- using architecture-aware cost models is a viable approach to performance portability (on low-latency clusters of multi-cores, CPU speed and cache-to-RAM ratio are good predictors)
- overhead associated with the static, analytic cost model is negligible
Related Work

- HWSkel (Armih et al.)
- HeteroMPI (Lastovetsky and Reddy)
- SKaMPI (Reussner et al.)
- hwloc (Broquedis et al.)
- Multi-core + GPU, static autotuning (O’Boyle et al.)
Limitations

- Only one application
- No application-specific parameters
- No dynamic system parameters
- Later cost models do not outperform the simple one
- Static, analytic approach may be inadequate for other application-architecture combinations
Future Work

- Add further applications or extend algorithmic skeletons
- Investigate the use of latency and bandwidth within the cost model
- Use application-specific and dynamic parameters in a run-time environment
- Include locally available GPUs to increase heterogeneity
- Devise a model of parallel computation that would account for heterogeneity and hierarchy
Thank you for your attention!

Any questions?