Architecture-Aware Cost Modelling for Parallel Performance Portability

Evgenij Belikov

Diploma Thesis Defence

August 31, 2011
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 (Many-cores, Systems-on-Chip, GPGPUs, FPGAs)
Perceived State of Affairs

- Trend towards heterogeneous and hierarchical architectures (Many-cores, Systems-on-Chip, GPGPUs, FPGAs)
- Hardware evolves faster than software
Perceived State of Affairs

- Trend towards heterogeneous and hierarchical architectures (Many-cores, Systems-on-Chip, GPGPUs, FPGAs)
- Hardware evolves faster than software
- Increasing complexity of software systems
Perceived State of Affairs

- Trend towards heterogeneous and hierarchical architectures (Many-cores, Systems-on-Chip, GPGPUs, FPGAs)
- Hardware evolves faster than software
- Increasing complexity of software systems
- Conceptual complexity of parallel programming
Motivation and Goal

Problem Statement

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

In this thesis
- Explore and quantify the impact of heterogeneity of modern parallel architectures on parallel programs
- Introduce a viable approach to parallel performance portability
General Idea and Parameter Choice

- More powerful PEs should receive more work (static load balancing)
- Architectural features (contextual information) are used to estimate relative computational power of PEs
- Static, analytic models lead to negligible overhead
- Accuracy of each model needs careful assessment

<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>( L2_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_{CPU}, b_{L2}, c_{RAM}, d_{L} )</td>
<td>corresponding scaling factors</td>
</tr>
</tbody>
</table>
### Cost Models

#### Generic model:

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

<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>
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 variations, Gigabit Ethernet (linux01 100MBit)

Configurations (1..16 PEs): e.g. 2x4 2x2 4x1 (lxpara, bwlf, linux-lab)

- all-local vs. partly remote (emulated using netem)
- unshuffled vs. shuffled
- without linux01 vs. with linux01
Application: Sparse Matrix Multiplication

- Design Methodology: PCAM (Foster)
- Environment: UNIX/Linux
- Language: C+MPI
- Model: Shared-Nothing Message Passing
- 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

- Heterogeneity
- Scalability
- Performance Predictability
- Impact of Latency
Heterogeneity (1)

8192x8192 matrix, sparsity: 1%, nonzeros: 37,420,929
Heterogeneity (2)

- heterogeneity has negative impact on performance
Heterogeneity (2)

- heterogeneity has negative impact on performance
- we choose the most heterogeneous case (2x4 2x2 4x1)
Heterogeneity (2)

- heterogeneity has negative impact on performance
- we choose the most heterogeneous case (2x4 2x2 4x1)
- starting point of the investigation
Scalability (1)

![Graph showing scalability with different cost models. The x-axis represents the number of processors, ranging from 0 to 16, and the y-axis represents speedup, ranging from 0 to 10. The graph compares different cost models: no cost model (baseline), cost model 0 (CPU speed), cost model 1 (CPU speed, L2 Cache), and cost model 2 (CPU speed, L2 Cache, RAM).]

The graph illustrates the performance of different cost models with varying numbers of processors. The baseline model shows the least improvement, while the models that include L2 Cache and RAM perform better. The graph is labeled with the following details:

- Matrix size: 8192x8192
- Sparsity: 1%
- Nonzeros: 37,420,929
- 2x4 2x2 4x1 unshuffled all-local (lpara beowulf linux_lab)
Scalability (2)

![Graph showing scalability with different cost models.](image)

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

**Graph Details:**
- **X-axis:** Number of Processors
- **Y-axis:** Speedup
- **Data:** 8192x8192 matrix, sparsity: 1%, nonzeros: 37,420,929
- **Task:** 2x4 2x2 4x1 shuffled all-local (lxpara + beowulf + linux_lab)
Scalability (3)

- all cost models perform better than the baseline case
Scalability (3)

- all cost models perform better than the baseline case
- low overhead (only difference in granularity)
Scalability (3)

- all cost models perform better than the baseline case
- low overhead (only difference in granularity)
- cost models are effective in achieving performance portability (speedup increases from 6.7 to 9.0)
Scalability (3)

- all cost models perform better than the baseline case
- low overhead (only difference in granularity)
- cost models are effective in achieving performance portability (speedup increases from 6.7 to 9.0)
- we expect the increase in performance to grow with increasing number of PEs
Performance Predictability (1)

Number of Processors

- 1x8 2x4  lpxpara; unshuffled
- 1x8 2x4  lpxpara; shuffled
- 1x8 2x2 4x1  lpxpara, linux lab; unshuffled
- 1x8 2x2 4x1  lpxpara, linux lab; shuffled
- 2x4 2x2 4x1  lpxpara, beowulf, linux01, linux lab; unshuffled
- 2x4 2x2 4x1  lpxpara, linux01, linux lab; shuffled

8192x8192 matrix, sparsity: 1%, nonzeros: 37.420.929
Performance Predictability (2)

- unshuffled placement strategy improves utilisation on multi-cores
Performance Predictability (2)

- unshuffled placement strategy improves utilisation on multi-cores
- shuffled placement strategy improves predictability
unshuffled placement strategy improves utilisation on multi-cores
shuffled placement strategy improves predictability
cost models reduce the variation between both on heterogeneous setups (i.e. increase performance for shuffled and increase predictability for the unshuffled)
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 exceedingly remote PEs
Impact of Latency (2)

- network characteristics are critical on partly remote setups
- do not send work to exceedingly remote PEs
- send even more work if this can compensate for communication overhead
Impact of Latency (2)

- network characteristics are critical on partly remote setups
- do not send work to exceedingly remote PEs
- send even more work if this can compensate for communication overhead
- off-loading work seems to involve a trade off
Conclusions

Findings

- performance portability is important for exploiting the power of rapidly evolving heterogeneous parallel architectures
Findings

- performance portability is important for exploiting the power of rapidly evolving heterogeneous parallel 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 is important for exploiting the power of rapidly evolving heterogeneous parallel 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)
- execution of parallel programs is adapted without the need to change the code
Findings

- performance portability is important for exploiting the power of rapidly evolving heterogeneous parallel 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)
- execution of parallel programs is adapted without the need to change the code
- there is no additional overhead associated with the static, analytic cost model
Findings

- Performance portability is important for exploiting the power of rapidly evolving heterogeneous parallel 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).
- Execution of parallel programs is adapted without the need to change the code.
- There is no additional overhead associated with the static, analytic cost model.
- Shuffled placement strategy leads to more predictable performance at low additional cost.
Related Work

- HeteroMPI (Lastovesky and Reddy)
- HWSkel (Armih et al.)
- hwloc (Broquedis et al.)
Limitations

- Only one application used
- No application-specific parameters used
- No dynamic system parameters used
- Relatively weak discussion of the later cost models
- Static, analytic approach may be inadequate for other application-architecture combinations
Future Work

- Add further applications or extend algorithmic skeletons
- Provide a portable format for architecture description as part of a discovery mechanism
- Investigate the use of latency and bandwidth within the cost model
- Use application-specific and dynamic parameters in a run-time environment
- Explore different modelling approaches
- Include locally available GPUs to increase heterogeneity
- Devise a model of parallel computation that would account for heterogeneity and hierarchy
Objection 1
‘Comprehensive state of the research review’ as contribution is clearly overstated.

‘Comprehensive’ does not mean exhaustive or complete, but appropriately broad yet detailed.
Objection 2

'Memory Wall' (discrepancy between the processor speed and memory speed) is not explicitly discussed.

Although it could be more explicit, it is discussed in Ch.2 as the 'von-Neumann Bottleneck'; we also consider remote PEs - an even more severe problem - the 'Network Wall'.
Objection 3

It would be nice to see a comparison to dynamic models.

Indeed, however, it was not within the scope of the thesis (may be pursued in the future).
Objection 4

It would be nice to have more applications.

True, however one application should suffice for an MSc level work. We decided to have a more detailed analysis of a single representative application on different architectures instead (investigating further applications or enhancing algorithmic skeletons is part of future work).
Objection 5

Is sparse matrix multiplication representative?

Yes, many HPC applications use it as a core (PDE solvers, graph algorithms, etc.) and it exploits *irregular* parallelism. It is also relevant since it is second ’hottest’ *motif* according to the authoritative ’Berkeley View on Parallel Computing Landscape’.
Objection 6

Would other algorithms yield comparable results?

Regarding other applications that use naive load distribution scheme on heterogeneous systems improvements are most likely. Considering other adaptation algorithms merits further investigation.
Objection 7

Is manual tuning still worth the effort?

Generally, no. Unless peak performance is critical, the software is tied to a single architecture, or there are no deadlines and the budget is unlimited. With growing complexity of modern software system automated performance optimisation is crucial. Compilers show that automated register allocation beats manual optimisation.
Objection 8
How much further improvement is possible on each of the architectures?

Theoretically, linear speedup is the upper bound. Practically, we emphasise the need for a unified model of parallel computation that takes into account heterogeneity and hierarchy of emerging parallel architectures to facilitate a more systematic approach. For sparse matrix multiplication the suggested comparison to manually optimised versions is to be conducted.
Objection 9

Analytic models are not accurate.

Analytic models may be accurate enough. We show how significant benefits are achievable with relatively low effort for one application by using cost models to calculate *relative* computational power of PEs to determine static load balancing. Bulk Synchronous Parallel Model enhanced with compositional cost models provides accurate estimates for shared memory architectures. Additionally, sophisticated modelling approaches are often incurring significant overheads (profiling, training runs, etc.). We envision run-time adaptation and probabilistic models to provide best results in dynamic environments. Analytic models may still suffice for rather static configurations.
Objection 10

Discussion of later cost models and the explanation of lower performance achieved by these models is weak.

Due to carrying out the study part-time this came short in the end. However, we discuss potential weaknesses of our approach in Ch.3 by discussing the design space and decision time. Clearly, some modelling approaches are missed, which were hard to identify due to missing links to cost modelling and performance portability. This underlines the danger of a heuristic approach, more revisions would be needed to possibly establish a total ordering among the cost models and to finalise the CM3-enhanced implementation.
Objection 11

Why not use benchmarks and sparse matrix collections?

It deemed infeasible within the scope of the project to extend a whole benchmark suite to use cost models. We use randomly generated matrices and have tested matrices generated with different seeds to avoid false conclusions. We also target sparse matrix multiplication in general, not a specific application (the impact of representation format is briefly discussed in Ch.4). Moreover testing a large set of matrices would substantially increase the time to conduct the experiments.
Thank you!

Any questions?