History-Based Adaptive Work Distribution

Evgenij Belikov

2014 Imperial College Computing Student Workshop, London

September 26, 2014
1. Pervasive Parallelism and the Need for Performance Portability

2. Semi-Explicit Parallel Functional Programming in GpH

3. Distributed Graph Reduction in GUM

4. Applications and Experimental Design

5. Results and Evaluation

6. Conclusions and Future Work
Motivation

- Pervasive parallelism in hardware
  \[\Rightarrow\] Parallelism is a key source of performance but hard to exploit
Motivation

- Pervasive parallelism in hardware
  ⇒ Parallelism is a key source of performance but hard to exploit
- Parallel architectures are increasingly heterogeneous and hierarchical
  ⇒ Need for a unified deterministic programming model
Heterogeneous Architectures and Performance Portability

**Motivation**

- Pervasive parallelism in hardware
  - Parallelism is a key source of performance but hard to exploit
- Parallel architectures are increasingly heterogeneous and hierarchical
  - Need for a unified deterministic programming model
- Hardware evolves faster than software
  - Need for automatic optimisations and parallelism management
Heterogeneous Architectures and Performance Portability

Motivation

- Pervasive parallelism in hardware
  ⇒ Parallelism is a key source of performance but hard to exploit
- Parallel architectures are increasingly heterogeneous and hierarchical
  ⇒ Need for a unified deterministic programming model
- Hardware evolves faster than software
  ⇒ Need for automatic optimisations and parallelism management
- Functional languages appear promising in balancing productivity and performance across diverse architectures
  ⇒ Challenge for compiler and run time system: adaptively managing parallelism to achieve performance portability
Glasgow Parallel Haskell

- Based on Haskell: state-of-the-art compiler and RTS (GHC)
- High-level, functional, **semi-explicit, architecture-independent**
- **Unified Deterministic** programming model
- **Advisory** dynamic fine-grained parallelism (sparks)
- Composable abstractions: skeletons, Evaluation Strategies

\[
\text{par} :: \ a \to b \to b \quad \text{seq} :: \ a \to b \to b
\]

\[
\begin{align*}
\text{pfib} \ 0 &= 0 \\
\text{pfib} \ 1 &= 1 \\
\text{pfib} \ n &= x \ 'par' \ y \ 'seq' \ (x + y) \\
\text{where} \ x &= \text{pfib} \ (n-1) \\
\text{y} &= \text{pfib} \ (n-2)
\end{align*}
\]
Graph Reduction on a Unified Machine Model (GUM)

- Supports shared-memory and distributed-memory by using virtual shared memory abstraction and isolated heaps
- Implicit synchronisation and communication
- Adaptive work distribution (work stealing)
- Thread subsumption (for granularity control; lightweight threads)
Random Work Stealing

- Decentralised and scalable; victims chosen uniformly at random
- Amortised cost (idle PEs initiate stealing)
- Close to optimal for 'well-formed' workloads
- Neglects potential architectural and system information
**History-Based Work Stealing**

<table>
<thead>
<tr>
<th>info table field</th>
<th>value = 0</th>
<th>value &gt; 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>history info</td>
<td>failed stealing attempt</td>
<td>number of successes</td>
</tr>
<tr>
<td>time stamp</td>
<td>information is stale</td>
<td>time of last update</td>
</tr>
</tbody>
</table>

**Table:** Overview of the Stored Historical Information

- Idea: keep track of recent stealing successes/failures
  
  $f(PEid) \rightarrow (\text{historyInfo}, \text{timeStamp})$

- Steal from the PEs where stealing was recently successful

- Does parallelism pattern influence distribution of parallelism?

- Invalidation interval (accuracy trade-off: coverage vs staleness)
Experimental Setup

Architecture
- Beowulf-class cluster of 8-cores (64PEs in total), 2/3GHZ CPUs, (256KB+4MB)/6MB cache, 12/16GB RAM, 150-300ms latency

Software
- CentOS 6.5, Gigabit Ethernet, GHC 6.12.3, GCC 4.4.7, PVM 3.4.6

Applications
- parfib: D&C, regular, ca. 5e5 sparks
- coins: D&C, permutation search, ca. 3e6 sparks
- worpitzky: D&C, symbolic, arbitrary length integers, ca. 7e6 sparks
- sumeuler: data-parallel, irregular, 200 sparks
- fixed input size; run times: median of three
- varying number of PEs and history interval (short, medium, long)
Run Times and FISHing Success Ratios

<table>
<thead>
<tr>
<th>PEs</th>
<th>2</th>
<th>4</th>
<th>16</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>parfb</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>coins</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>worpitzky</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>semeuler</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>noqz</td>
<td>qz100</td>
<td>qz1000</td>
<td>qz10000</td>
<td></td>
</tr>
</tbody>
</table>
Findings

- Order of magnitude reduction in run time for all programs
- Benefits of adding PEs lower for higher PE numbers (due to overhead and lack of work)
- Mostly abundant potential parallelism available (D&C often has more sparks than data-parallel\(^2\))
- Using history on 64 PEs: up to 34% reduction in run time
- Improved stealing success ratio for data-parallel sumeuler
- Reduced stealing success for D&C programs for high PE numbers

\(^2\)also depends on the application-level thresholding/chunking
Conclusions

- Using history is beneficial for data parallel applications (with stable parallelism generators)
- Stale information can lead to worse-than-random decisions
- History alone appears insufficient for D&C applications (with more dynamic parallelism generators)
- Used parallelism patterns have influence on distribution of parallelism
Future Work

- Add several larger applications with different patterns
- Investigate ways of auto-tuning the invalidation interval
- Use different policies for parallelism generators than for workers e.g. switching to work-pushing depending on system load
- Co-location of related sparks to improve spatial data locality
- Support for architecture-awareness at RTS level using cost models
### Table: Number of FISH Messages versus Total Sent Messages (on 64 PEs)

<table>
<thead>
<tr>
<th>il</th>
<th>noqz FISHes</th>
<th>Total</th>
<th>qz100 FISHes</th>
<th>Total</th>
<th>qz1000 FISHes</th>
<th>Total</th>
<th>qz10000 FISHes</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>pb</td>
<td>5137</td>
<td>12134</td>
<td>3965</td>
<td>9027</td>
<td>4874</td>
<td>12196</td>
<td>4801</td>
<td>10793</td>
</tr>
<tr>
<td>cs</td>
<td>7710</td>
<td>28774</td>
<td>10541</td>
<td>27856</td>
<td>12694</td>
<td>30804</td>
<td>38715</td>
<td>63320</td>
</tr>
<tr>
<td>wy</td>
<td>73278</td>
<td>153438</td>
<td>71437</td>
<td>125510</td>
<td>74370</td>
<td>125630</td>
<td>82155</td>
<td>139641</td>
</tr>
<tr>
<td>sr</td>
<td>34968</td>
<td>35714</td>
<td>18650</td>
<td>19451</td>
<td>20395</td>
<td>21200</td>
<td>15190</td>
<td>16004</td>
</tr>
</tbody>
</table>