Performance Analysis of Equation Based Simulations

[Work in Progress]

Marcus Walther (marcus.walther@tu-dresden.de)
Outline

1. Motivation
2. Memory Organization
3. Performance Analysis
4. Conclusion & Outlook
Motivation

- Parallelization Project HPCOM (High Performance Computing meets OpenModelica) started in August 2013
- A lot of scheduling and code generation techniques already presented
Motivation

- HPCOM TaskGraph based parallelization:

\[
\begin{align*}
\text{f1: } & R1.v = (-R1.R_{\text{actual}}) \times R2.i \\
\text{f2: } & R1.LossPower = (-R1.v) \times R2.i \\
\text{f3: } & R1.v = (-\text{source.v}) - R1.n.v \\
\text{f4: } & R2.v = R2.R_{\text{actual}} \times R2.i \\
\text{f5: } & R2.LossPower = R2.v \times R2.i \\
\text{f6: } & R2.v = (-C.v) - R1.n.v \\
\text{f7: } & R2.i = C.C \times \text{der}(C.v) \\
\text{f8: } & \text{source.v} = \text{ampl} \times \sin(2\pi \times \text{freq} \times \text{time})
\end{align*}
\]
Motivation

Example CauerLowPassSC of MSL 3.2.:
Motivation

Example CauerLowPassSC of MSL 3.2.

- Equation System
- Simple Equation
Motivation

Example CauerLowPassSC of MSL 3.2.:

Theoretical speedup (if the number of threads tends to infinity) is nearly 6
**Motivation**

- Example CauerLowPassSC of MSL 3.2.: How can we shrink this gap?
To understand why memory organization is important, we have to understand the memory hierarchy.

Example: a simple dual core CPU can have the following memory hierarchy for storing data:

- Smallest organizational unit is a cache line (e.g. stores 8 floating point values)
  - If one value is required by a core, the whole cache line must be loaded into L1-Cache.
  - If a value is not in L1-Cache, a cache miss (delay) occurs.

Diagram:

```
Main Memory (8GB) → L3-Cache (8MB) → L2-Cache (256KB) → L1-Cache (32KB) → Core 1, Core 2
```

For Core 1:
- L1-Cache (32KB)
- L2-Cache (256KB)
- L3-Cache (8MB)

For Core 2:
- L1-Cache (32KB)
- L2-Cache (256KB)
- L3-Cache (8MB)

Center for Information Services & High Performance Computing
Memory Organization

- A lot of variable / cache line accesses required to calculate one time step
  - Variables that are required at the same time or consecutively should be stored in the same or consecutive cache lines
- For serial simulation runs the influence is not large
- But in parallel simulation runs, **false sharing** can occur if the variables are not organized regarding the schedule
Memory Organization – False Sharing

Core 1

L1-Cache (32KB)

L2-Cache (256KB)

L3-Cache (8MB)

Main Memory (8GB)

Core 2

L1-Cache (32KB)

L2-Cache (256KB)

L3-Cache (8MB)

Description: Core 1 will write to A
Core 2 will write to B

Cacheline with 2 values:
Performance Analysis

A lot of tools exist to measure performance

Some Tools are developed at TU Dresden ZIH

- **Score-P**
  - “Scalable Performance Measurement Infrastructure for Parallel Codes”
  - Tracing and Profiling supported
  - Can use PAPI (Performance Application Programming Interface) performance counter to get information about hardware metrics

- **Vampir**
  - Visualization of OTF trace files
Performance Analysis - Vampir
Performance Analysis - Tracing

Serial Simulation of the CauerLowPassSC-Example (Call Stack Visualization):

- Initialization
- Write Output
- Evaluate ODE
- Run simulation
- Solver Step (CVode)
- Event-handling
- Evaluate All

- Calculate Equation system
- Evaluate ODE
- Calculate Simple Equation
- Solver Step (CVode)
- Run simulation

Technische Universität Dresden
Performance Analysis – Tracing

Parallel Simulation of the CauerLowPassSC-Example (Thread Visualization):

<table>
<thead>
<tr>
<th>Thread</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.66210 s</td>
</tr>
<tr>
<td>2</td>
<td>3.66215 s</td>
</tr>
<tr>
<td>3</td>
<td>3.66220 s</td>
</tr>
<tr>
<td>4</td>
<td>3.66225 s</td>
</tr>
<tr>
<td>5</td>
<td>3.66230 s</td>
</tr>
<tr>
<td>6</td>
<td>3.66235 s</td>
</tr>
</tbody>
</table>

Insight into runtime workflow

Overhead of 1000 cycles per block to large for such fine grained analyzation!

Profiling techniques with low overhead required for such analyzations
Performance Analysis – Profiling

- False Sharing can be measured by analyzing L2 Cache Misses
- CauerLowPass Example contains a lot of equation systems → hard to understand
- Wire example contains just single equations
Performance Analysis – Profiling

• Synthetic Wire example with 10 links
Performance Analysis – Profiling

- Synthetic Wire example with 10 links

- OpenModelica stores all variables in one big array, ordered regarding their name

- Optimized memory layout was used to separate cache line accesses

<table>
<thead>
<tr>
<th></th>
<th>L2 cache misses</th>
<th>Total cycles</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>base line</td>
<td>10895</td>
<td>469309</td>
<td></td>
</tr>
<tr>
<td>with optimization</td>
<td>4457</td>
<td>262152</td>
<td>1.79</td>
</tr>
</tbody>
</table>

ZIH Center for Information Services & High Performance Computing
Conclusion & Outlook

- Tracing tools can be used to understand workflow
- Low overhead profiling tools are useful to analyze cache behavior
- False Sharing can have a high influence on performance

- Use traces to talk with experts of other domains to improve runtime
- Include improved memory layout as a fixed part of HPCOM
- Make performance tools a stable part of C++ Simulation Runtime
Thank you for your attention

Questions?

Marcus Walther (marcus.walther@tu-dresden.de)