Locating Parallelization Potential in Object-Oriented Data Structures

Korbinian Molitorisz
Location and Current Research

- Karlsruhe Institute of Technology (KIT)
  - Faculty of Computer Science
  - Chair Prof. Dr. Walter F. Tichy
  - Research Group AParT

- Research Group AParT
  - Parallelizing existing „legacy“ software
  - Support engineering parallel software
  - Active collaboration with Siemens Corporate Technology
  - 4 PhDs, several Bachelor-/Master-students
What makes parallelization so hard?

- Parallelization is **time-consuming and error-prone**
- Case Studies
  - Implementing a video processing pipeline: Several weeks [OS+10]
  - Implementing a desktop search engine: The third parallelization approach achieved acceptable performance [MT10]
  - „Meanwhile, multicore processors have become *mainstream, but not the knowledge how to program them*“ [VM11]

Pattern-based parallelization process für legacy software is urgently needed

[OS+10] – Frank Otto, Christoph Schafer et al. *A language-based tuning mechanism for task and pipeline parallelism*, Euro-Par 2010
Pattern-based Parallelization Process

- Identifies source patterns and runtime-relevant tuning parameters
- Transforms them to parallel patterns like Pipeline or Master/Worker

Motivation – Let’s try something different!

- Today, detecting parallel potential always looks for hotspots (high runtime share, high number of executions)

- Observations
  - Object-oriented paradigm heavily used
  - Data structures (DS) comprise containers plus algorithms

- Can we derive parallel potential from monitoring accesses to object-oriented data structures?
Empirical Study

- **Research Questions**
  - Q₁ – **Frequency**: What object-oriented data structures are used in reality? (Not those that can be found in text books!)
  - Q₂ – **Access patterns**: Can we find recurring regularities in the access profiles of these object-oriented data structures?
  - Q₃ – **Parallelizability**: Do they carry parallel potential?

- **Benchmark**
  - 37 open source projects from 11 application domains
  - 2 – 718 DS instances, in total 1,960
  - 300 – 460,000 LOC, in total >936,000 LOC
Q₁: Data structure frequency?

- Assumption: Arrays are heavily used static DS
- Research focus: dynamic DS
- Result: List and Dictionary together make up >81%
Q₂: Access Patterns?
Q₂: Access Patterns? (Linpack benchmark)

- Subsequent write operations (red dots)
  - From front → end (offset one at the front)
  - From front → end (offset one at the end)
Q$_2$: Access Patterns? (Mesh Routing)

- Insert elements at the end of DS (blue lines)
- Overlapping read operations from front $\rightarrow$ end (green lines)
- When read reaches the last element: clear()
### Q₂: Access Patterns!

<table>
<thead>
<tr>
<th>Program</th>
<th>...with detectable patterns</th>
<th>...with already identified patterns</th>
<th>...without patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>astrogrep</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>borys-MeshRouting</td>
<td>4</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>clipper</td>
<td>3</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>compgeo</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>contentfinder</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>csparser</td>
<td>2</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>,dsa‘</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>dotqcf</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>fire</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ManicDigger2011</td>
<td>1</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>MidiSheetMusic</td>
<td>4</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>Net_With_UI</td>
<td>3</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>netinfotrace</td>
<td>4</td>
<td>13</td>
<td>4</td>
</tr>
<tr>
<td>rrrsroguelike</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>TerraBIB</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TreeLayoutHelper</td>
<td>0</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td><strong>∑</strong></td>
<td><strong>31</strong></td>
<td><strong>81</strong></td>
<td><strong>30</strong></td>
</tr>
<tr>
<td>Parallel Potential</td>
<td>Description / Threshold values</td>
<td>Exploit Potential Advice</td>
<td></td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Long-Insert (49x in 21 programs)</td>
<td>&gt;100 insert patterns, &gt;100 insert operations, &gt;30% of all accesses</td>
<td>Parallelize the insert operation</td>
<td></td>
</tr>
<tr>
<td>Frequent-Long-Read (10x in 8 programs)</td>
<td>&gt;10 consecutive reads, &gt;50% read operations, read &gt;50% of the data structure</td>
<td>Check access origin. It might be a search operation for a specific element: Transform this operation to a parallel search operation</td>
<td></td>
</tr>
</tbody>
</table>
| Frequent-Search (3x in 2 programs) | >1,000 search operations for a specific element, >2% of all access events are read operations | - Employ a parallel data structure for searches  
- Parallelize the search operation (e.g. data parallel search) |
| Implement-Queue (3x in 3 programs) | DS is used as queue, >100 read/write operations to one or both ends (FIFO, FILO), >60% of all accesses | Employ a parallel queue as data container                                                   |
| Sort-After-Insert (1x in 1 program) | DS is sorted after a Long-Insert                                                             | Parallelize both insert and search phases                                                  |
Evaluation

- Assembled new benchmark, >15,000 LOC
- Test system: 8-core AMD FX 8120h, 3.1 Ghz, 8 GB RAM
- Rather moderate speedups, but for free
Conclusion

- Software engineers need a tool support for parallelization.
- „Traditional“ hotspot analyses are well-established.
- Exploratory empirical study in >936 KLOC revealed that List and Dictionary make up >81% of all dynamic DS.
- Runtime profile of dynamic data structures in commodity software contains recurring access patterns.
- They can be identified automatically and serve for parallelization with rather moderate speedups.
- After IPDPS available on our website http://www.apart-project.de
Thank you for your attention!
Any questions?

molitorisz@kit.edu