The multicore era has started. Are 40 years of sequential source code obsolete?

3rd FITTEST Industrial Day, 7th IEEE International Conference on Research Challenges in Information Science (RCIS 2013, Sorbonne, Paris)
Korbinian Molitorisz
Motivation

- Multicores are ubiquitous
- Parallel software skills are not
- Parallel software is neither
- The free lunch is over – clock frequency stagnation

Do we all have to become parallel experts now?

- Refactoring support for existing software needed.

Automation?

2) Hans Vandierendonck: *Averting the Next Software Crisis* (2011)
Motivation

Gold standard: Parallel loop?

AviStream Process(AviStream aviIn)
{
    AviStream aviOut = new AviStream();
    foreach(Image i in aviIn.Images)
    {
        Image edge = edgeFilter.Apply(i);
        Image thres = thresholdFilter.Apply(i);
        Image fade = fadingFilter.Apply(i);
        Image res = addFilter.Apply(edge, thres, fade);
        aviOut.Images.Add(res);
    }
    return aviOut;
}

Parallel loop

- Threads need to **wait very long** (acquisition/release of common used lock)
- Correct sequence **not guaranteed** without additional logic
- Can architecture patterns be used and derived automatically?
Architecture pattern: Pipeline

- Divide tasks into **different stages** that can be executed consecutively.
- Dependencies between stages may exist (i.e. output of stage $s_i$ is the input for the following stage $s_{i+1}$).
  - Partly sequential execution, but dependencies within procedures and across iterations are preserved.
- Data passes stages in a sequence.
  - Might be cached between stages.

Timothy G. Mattson, Beverly A. Sanders, Berna K. Massingill – *Patterns for Parallel Programming*, 2004
Architecture pattern: Pipeline

Illustration for data elements $e_1$ to $e_6$

Stage 1

Stage 2

Stage 3

Stage 4

Pipeline at distinct time $t_i$
Observations

- Recurring patterns exist: Architecture / design / code access patterns...
  - Pattern-based approach

- Code areas consume different amounts of runtime

- Separation of concerns used in object-orientation

- Existing software builds on object-orientation
  - Split up control flow

- Modern object-oriented environments heavily use references
  - Combine static and dynamic analyses

- Parameters exist that have influence on the runtime behaviour
  - Derive tuning parameters from sequential runtime behaviour

- Race detectors exist but not handy for real-world applications
  - Unit tests as small fractions of a whole program
Pattern-based refactoring concept: AutoPar

AutoPar\textsubscript{PD} (parallel detection)
- Detection of Parallel Regions

AutoPar\textsubscript{TA} (tunable architectures)
- Language Specification for Optimizable Parallel Architectures

AutoPar\textsubscript{RE} (runtime extension)
- Compiler extension and Runtime Library

How can I parallelize a certain location?
Is it still correct?
How about the performance?

Where shall I parallelize my code?
Pattern-based refactoring concept: AutoPar

- Analysis pattern: *Single Static Multiple Dynamic*
- Detection modules operate on extended AST
- Explicit architecture language with tuning information
- Runtime library with stencils for patterns
- Automatic unit test generation
- Interface for auto tuners
5 separate steps to parallelize and test correctness in an automated process on the base of architecture patterns.
Pattern-based refactoring concept: AutoPar

- **Code annotation**
  - Architecture description language with defined operators (for architecture description) and operands (for the architecture compartments)

- **Runtime library**
  - Input: Architecture description and architecture compartments
  - Output: Instances of the runtime library, tuning file and unit tests

```java
01 AviStream Process(AviStream aviIn)
02 {
03   AviStream aviOut = new AviStream();
04   #region TADL: (A+ || B || C) => D => E
05   foreach(Image i in aviIn.Images)
06   {
07     #region A: Image e = edgeFilter.Apply(i); #endregion
08     #region B: Image t = thresholdFilter.Apply(i); #endregion
09     #region C: Image f = fadingFilter.Apply(i); #endregion
10     #region D: Image r = addFilter.Apply(e, t, f); #endregion
11     #region E: aviOut.Images.Add(r); #endregion
12   }
13   #endregion
14   return aviOut;
15 }
```

```java
01 AviStream Process(AviStream aviIn)
02 {
03   Item p1 = new Item (edgeFilter.Apply());
04   Item p2 = new Item (thresholdFilter.Apply());
05   Item p3 = new Item (fadingFilter.Apply());
06   Item p4 = new Item (addFilter.Apply());
07   Item p5 = new Item (aviOut.Images.Add());
08   MasterWorker mw = new MasterWorker (p1, p2, p3);
09   mw.Item(p1).replicable = true;
10   Pipeline p = new Pipeline (mw, p4, p5);
11   p.Input = aviIn.Images;
12   p.Run();
13   return p.Output;
14 }
```
Back to the example...

- Results:
  - Speedup on an 8-core machine: 3.12
  - Manual implementation of this architecture pattern: 6.2

```
Results:
- Speedup on an 8-core machine: 3.12
- Manual implementation of this architecture pattern: 6.2
```
Back to the example…

- **Results:**
  - Speedup on an 8-core machine: 5.3
  - Manual implementation of this architecture pattern: 6.2

![Diagram of the architecture pattern](image)
Pattern-based refactoring concept: AutoPar

Parameter configuration x Speedup

Sequential
Evaluation

- 6 real-world projects
- 27,000 LOC
- Average search space reduction: 95%
- Average precision: 66%

<table>
<thead>
<tr>
<th>Project</th>
<th>#HotSpots</th>
<th>#Architectures identified</th>
<th>Reduction</th>
<th>Correct &amp; faster</th>
<th>Correct &amp; ¬faster</th>
<th>¬Correct</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>MergeSort</td>
<td>19</td>
<td>1</td>
<td>95%</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>100 %</td>
</tr>
<tr>
<td>RayTracer</td>
<td>63</td>
<td>6</td>
<td>90%</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>17 %</td>
</tr>
<tr>
<td>DesktopSearch</td>
<td>47</td>
<td>2</td>
<td>96%</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>50 %</td>
</tr>
<tr>
<td>CompGeo</td>
<td>169</td>
<td>1</td>
<td>98%</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>100 %</td>
</tr>
<tr>
<td>VideoProcessing</td>
<td>16</td>
<td>1</td>
<td>94%</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>100 %</td>
</tr>
<tr>
<td>PowerCollections</td>
<td>3,641</td>
<td>15</td>
<td>97%</td>
<td>4</td>
<td>9</td>
<td>2</td>
<td>27 %</td>
</tr>
<tr>
<td><strong>∑ = 3.955</strong></td>
<td><strong>∑ = 26</strong></td>
<td></td>
<td><strong>Ø = 95%</strong></td>
<td><strong>∑ = 9</strong></td>
<td><strong>∑ = 15</strong></td>
<td><strong>∑ = 2</strong></td>
<td><strong>Ø = 66%</strong></td>
</tr>
</tbody>
</table>
Thank you for your attention.
Any questions?

molitorisz@kit.edu