Selecting points of interest in traces using patterns of events

François Trahay, Elisabeth Brunet, Mohamed Mosli Bouksiaa, Jianwei Liao
Context

- Hardware is more and more complex
  - NUMA, hierarchical caches, GPU, ...
- Software is more and more complex
  - Hybrid MPI+OpenMP, MPI+CUDA, ...

- Achieving good performance is hard
- Understanding the performance of an application is difficult

→ Need for performance analysis tools
Performance analysis
Tracing tools

- **Tracing applications**
  - Run the application once
  - Capture interesting events (e.g. MPI functions)
  - Generate an execution trace
    - Visualize & understand the behavior of the application
    - Find problematic parts of the execution

- **Examples**
  - VampirTrace
  - ScalaTrace
  - Intel Trace Analyzer and Collector
  - EZTrace
  - ...
Visualizing large trace files

- Visualizing a large trace is difficult
  - Millions of events
- How to detect the interesting part of the trace?

NPB CG class A 16 MPI Processes – 426 000 events
Visualizing large trace files

- Visualizing a large trace is difficult
  - Millions of events
- How to detect the interesting part of the trace?

NPB CG class A 16 MPI Processes – 426 000 events
Visualizing large trace files
repeating patterns

- A trace is usually structured
  - Loops
  - Functions
- Lots of similar information

NPB CG class A 16 MPI Processes – 426 000 events
Proposal: pointing what users should examine

Detect similarities in a trace
- Application phases that repeat

```plaintext
100 x {
    MPI_SEND (src=0  dest=1  len=16  tag=0)
    MPI_RECV (src=1  dest=0  len=16  tag=0)
}
```

```plaintext
MPI_Barrier
```

```plaintext
10000 x {
    MPI_SEND (src=0  dest=1  len=16  tag=0)
    MPI_RECV (src=1  dest=0  len=16  tag=0)
}
```

```plaintext
MPI_Barrier
```

Select « points of interests » of the trace
- Parts that users should examine first
- Where useful information is
Detecting similarities
Representation of a trace

- A trace can be represented as an event list

- Goal: detect patterns in this list
  - Can be viewed as a factorization
Factorization algorithm
First step: find small patterns

- Find a couple of events \((e_1, e_2)\) that appears several times
  → 2-event patterns
- Browse the event list and search for duplicated sequences
Factorization algorithm
First step: find small patterns

- Find a couple of events \((e_1, e_2)\) that appears several times
  \rightarrow 2\text{-event patterns}

- Browse the event list and search for duplicated sequences
Factorization algorithm
Second step: find loops in patterns

- A loop is a sequence of events that repeats
  - Each iteration has been detected as a pattern
- Browse the patterns lists and search for consecutive sequences

![Diagram of a loop with labeled nodes: P #1, c, d, P #1, P #1, P #1, e]
Factorization algorithm
Second step: find loops in patterns

- A loop is a sequence of events that repeats
  - Each iteration has been detected as a pattern
- Browse the patterns lists and search for consecutive sequences
A loop is a sequence of events that repeats
- Each iteration has been detected as a pattern

Browse the patterns lists and search for consecutive sequences
Factorization algorithm
Second step: find loops in patterns

- A loop is a sequence of events that repeats
  - Each iteration has been detected as a pattern
- Browse the patterns lists and search for consecutive sequences
Is this 2-event pattern a 3-event pattern?
Factorization algorithm
Third step: try to expand patterns

- Is this 2-event pattern a 3-event pattern?
- Case 1: pattern #1 is always followed by event C
Is this 2-event pattern a 3-event pattern?

Case 1: pattern #1 is always followed by event C
   → pattern #1 is a 3-event pattern
Is this 2-event pattern a 3-event pattern?

Case 2: pattern #1 is not always followed by event C, but it sometimes is
  → create pattern #2 that integrates Pattern #1
Factorization algorithm
third step: try to expand patterns

- Is this 2-event pattern a 3-event pattern?
- Case 2: pattern #1 is not always followed by event C, but it sometimes is
  → create pattern #2 that integrates Pattern #1
Factorization algorithm
third step: try to expand patterns

- Is this 2-event pattern a 3-event pattern?
- Case 3: pattern #1 is followed by event C only once
  → do nothing
Factorization algorithm
Limitations

- Only valid for 1 thread/process
  - Based on temporal order
    → the algorithm needs to run for each thread
    → can be done in parallel

- Complexity : $O(n^2)$
  - Worst case complexity (when there is no pattern)
  - In real life : it depends on the size of patterns
Evaluation

- **Implemented in EZTrace**
  - *Post mortem* analysis
  - Parallelized with OpenMP

- **Stark cluster**
  - 4 nodes
  - Quad-core Xeon

- **Results**
  - Detects patterns
  - Detects the applications iterations
  - Cheap compared to data mining techniques

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Pattern detection (ms)</th>
<th># of events</th>
<th># of patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>CG</td>
<td>178</td>
<td>284 000</td>
<td>160</td>
</tr>
<tr>
<td>MG</td>
<td>186</td>
<td>118 000</td>
<td>2 728</td>
</tr>
<tr>
<td>SP</td>
<td>596</td>
<td>557 000</td>
<td>174</td>
</tr>
<tr>
<td>BT</td>
<td>951</td>
<td>400 000</td>
<td>112</td>
</tr>
<tr>
<td>LU</td>
<td>4 564</td>
<td>4 568 000</td>
<td>210</td>
</tr>
</tbody>
</table>

*NPB Class A, Procs=16*
Selecting points of interest
Select representative occurrences

- Instead of examining 1000 occurrences
- Select 1 occurrence per class
Selecting representative occurrences

- Classify occurrences according to their duration
- Search for 'peaks' in the distribution
Filtering traces

- Select one occurrence per peak
  - Filter out 'similar' occurrences
Experimental results
NPB class A, 16 procs

<table>
<thead>
<tr>
<th>Kernel</th>
<th># events</th>
<th># events after filtering</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP</td>
<td>3 090</td>
<td>2 873</td>
</tr>
<tr>
<td>FT</td>
<td>10 256</td>
<td>6 704</td>
</tr>
<tr>
<td>IS</td>
<td>18 552</td>
<td>15 948</td>
</tr>
<tr>
<td>MG</td>
<td>118 688</td>
<td>41 031</td>
</tr>
<tr>
<td>CG</td>
<td>284 754</td>
<td>11 724</td>
</tr>
<tr>
<td>BT</td>
<td>399 944</td>
<td>24 338</td>
</tr>
<tr>
<td>SP</td>
<td>557 318</td>
<td>68 287</td>
</tr>
<tr>
<td>LU</td>
<td>4 568 002</td>
<td>42 881</td>
</tr>
</tbody>
</table>
## Experimental results

### NPB class A, 16 procs

<table>
<thead>
<tr>
<th>Kernel</th>
<th># events</th>
<th>#events after filtering</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP</td>
<td>3 090</td>
<td>2 873</td>
</tr>
<tr>
<td>FT</td>
<td>10 256</td>
<td>6 704</td>
</tr>
<tr>
<td>IS</td>
<td>18 552</td>
<td>15 948</td>
</tr>
<tr>
<td>MG</td>
<td>118 688</td>
<td>41 031</td>
</tr>
<tr>
<td>CG</td>
<td>284 754</td>
<td>11 724</td>
</tr>
<tr>
<td>BT</td>
<td>399 944</td>
<td>24 338</td>
</tr>
<tr>
<td>SP</td>
<td>557 318</td>
<td>68 287</td>
</tr>
<tr>
<td>LU</td>
<td>4 568 002</td>
<td>42 881</td>
</tr>
</tbody>
</table>

**Graph:**

- **Y-axis:** Number of events
- **X-axis:** Kernels (EP, FT, IS, MG, CG, BT, SP, LU)
- **Legend:**
  - Blue: Without filtering
  - Red: With filtering
- **Reduction Ratio (%)**
  - EP: 95.8
  - FT: 87.7
  - IS: 93.9
  - MG: 99
  - CG: 74.6
  - BT: 14
  - SP: 34.6
  - LU: 65.4
Conclusion
Conclusion

- Manually detecting the interesting parts of a trace is difficult

- Proposal: automate the detection of problems
  - Detect repeating patterns of events
  - Compare similar patterns whose duration differ significantly
  - Filter out redundant information

- Future work
  - Analyze patterns
  - Integrate to the stable version of EZTrace
Question?

François Trahay

http://eztrace.gforge.inria.fr/
Backtracking irregularities
Backtracking irregularities
Backtracking irregularities