Event Trace Analysis with the Complete Call Graph Data Structure

Knoxville, 2009-05-29

Zellescher Weg 12
Willers-Bau A114
Tel. +49 351 - 463 - 38323

Andreas Knüpf er (andreas.knuepf er@tu-dresden.de)
Overview

- Introduction to Event Tracing [link]
- Trace Compression [link]
- CCG Data Structure [link]
- Evaluation [link]
- Conclusions [link]
Overview

- Introduction to Event Tracing  
- Trace Compression  
- CCG Data Structure  
- Evaluation  
- Conclusions
Introduction to Event Tracing

- Instrumentation and monitoring
- Profiling: collect statistic summary
- Event tracing: record individual events
Introduction to Event Tracing

- Event tracing for parallel performance analysis and optimization
- More expressive than statistics (profiling, sampling)
- Record individual events during execution
- More overhead, larger data volumes

Established tracing tools from TU Dresden:
- Vampir for trace visualization
- VampirTrace for instrumentation and run-time recording
- Open Trace Format (OTF)
Introduction to Event Tracing

Vampir Displays
Introduction to Event Tracing
Overview

- Introduction to Event Tracing
- Trace Compression
- CCG Data Structure
- Evaluation
- Conclusions
Starting Point and Objectives

- Event traces tend to become very large

- Single events consume 10 to 100 bytes

- But huge event counts due to
  - Long running applications (hours)
  - Highly parallel applications (hundreds/thousands of CPUs)
  - Very detailed event recording (microsecond range)

Is the amount of data equal to the amount of information?

No, usually not. There are redundancies.
Starting Point – Redundancy

Redundancy in program execution:

- Temporal repetitions due to iteration (and recursion)
- Spatial repetitions due to SPMD parallelism

Repeated event sequences in traces
Potential redundancy in event traces
Objectives – Data Compression

- Exploit repetitions in event traces during trace analysis (offline)
  - For data compression
  - For more efficient analysis

Tasks:
- Capture identical and similar repetitions – define “similar”
- Fast and efficient detection of repetitions
- Continuous compression, without large temporary data
- Convenient memory data structure
- Analysis methods – without explicit decompression
Overview

- Introduction to Event Tracing
- Trace Compression
- CCG Data Structure
- Evaluation
- Conclusions
**Solution – Tree Data Structure**

Tree data structure similar to call tree

**Complete Call Graph**
Performance Counter Values

- Timestamps are monotonous increasing
- Trivial transformation to durations (and back)
- Reveals repetitive behavior

The same applies for typical performance counter values
- Consider them as “alternative timers”
Solution – Graph Nodes

- Map event properties to node properties
- Use specific node types
- Separate hard and soft properties
  - Hard properties: no deviation, lossless compression
  - Soft properties: limited deviation, lossy compression
  - Control deviation bounds for soft properties

<table>
<thead>
<tr>
<th>Hard properties</th>
<th>Soft properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node type</td>
<td>Time Durations</td>
</tr>
<tr>
<td>Process identifiers</td>
<td>Counter Samples</td>
</tr>
<tr>
<td>Function identifiers</td>
<td>(Message sizes)</td>
</tr>
<tr>
<td>Child references</td>
<td></td>
</tr>
</tbody>
</table>
Combined construction & compression of sub-trees

- In bottom-up order (order of creation)
- Fast sub-tree comparison with constant effort \((b = \text{const.})\)
  - Hard properties identical, soft properties compatible
  - Child references identical (due to previous compression)
- Fast look-up of candidate nodes
Solution – Evaluation Algorithms

- Collect statistics
- Cache + re-use intermediate results per sub-tree
- For global statistics as well as for timeline diagrams
  - Faster evaluation, individual statistics per pixel column
Overview

- Introduction to Event Tracing  
- Trace Compression  
- CCG Data Structure  
- Evaluation  
- Conclusions
Evaluation – Compression Ratios

How to measure compression?

- Comparing memory consumption: $M \geq m$ (without/with compr.)
- Counting graph nodes: $N \geq n$ (without/with compr.)

$$R_{memory} = \frac{M}{m} \geq 1 \quad R_{nodes} = \frac{N}{n} \geq 1$$

Larger is better

- Memory compression ratio: reduction of storage space
- Node compression ratio: reduction of evaluation effort (assuming constant effort per node)
Evaluation – Parameters

Prototype implementation
- C++ code of approx. 2000 lines
- Proof-of-concept integration in the VampirServer tool

Influencing parameters:
- Absolute and relative time deviation bounds $abs$ and $rel$
- Branching factor $b$
- Trace size (event count or $N$), ...

Result terms:
- Compression ratios $R_{\text{memory}}$ and $R_{\text{nodes}}$
- Run-time for compression or evaluation $t$
Evaluation – Compression Model

- Average Compression Model
  - Consider $d$ independent soft properties
  - Nodes cover $d$-dimensional volume with diameter $2p$
  - Cover $d$-dimensional area with diameter $L$ with $n$ nodes
Evaluation – Small Scale Compression

- Small scale compression: $0 \leq \text{abs} \leq 10000$, $0 \leq \text{rel} \leq 100\%$
- Compression ratios up to 19 (memory) and 55 (nodes)
- Better compression for larger deviation bounds (not strictly)
- Slow with quadratic influence of $n$!
Large scale compression: \(10^4 \leq \text{abs} \leq 10^6, \ 10\% \leq \text{rel} \leq 1000\%\)

Compression ratios up to 220 (memory) and 649 (nodes)

Maximum compression ratios of 458 (memory) and 1112 (nodes)

Faster, linear influence of node count to run-time!
Caching for initial/successive queries (empty/pre-filled caches)

- Initial: strong influence of compressed node count $n$
- Successive: faster, influence of branching factor $b$
Evaluation – Further Interesting Results

- Non-monotone compression
- Influence of branching factor $b$ to $R_{memory}$ and $R_{nodes}$
- Influence of branching factor $b$ to construction and evaluation
- Re-compression is not recommended
Overview

- Introduction to Event Tracing
- Trace Compression
- CCG Data Structure
- Validation and Evaluation
- Conclusions
Conclusions

Special-purpose data compression scheme
Tunable for different compression scenarios
Increased scalability:
  – Compression factors up to several hundred
  – Reduction of evaluation effort in same order

Outlook

Incorporation into existing performance analysis tools
Consider new event types, metrics, evaluation algorithms, …
Extend trace visualization, show regular/irregular structures
Outlook – User Interface

Extend trace visualization, show regular/irregular structures
Thank you!

Questions?
Further Topics (1)

- Tracing tools, trace file formats, memory data structures
- Typical trace evaluation procedures
- Related work

- Guaranteeing deviation bounds for soft properties
- Graph node encoding and allocation
- Fast search for replacement nodes, influence of search order
- Advanced CCG construction & compression
  - Re-compression, merging
  - Adaptive deviation bounds
Further Topics (2)

- CCG evaluation algorithms
  - Iterator and search
  - Statistical summary, timeline rendering, cache reduction
  - MPI send-receive matching
- Persistent storage and restore
- Distributed construction & compression
- Best case, worst case, and non-monotone compression
- Influence of parameters to compression & evaluation
  - Essential results in nine corollaries
- Comparison to state-of-the-art tools