Trace-based and Automatic Performance Measurement and Analysis

2012-06-27  Christian Rössel  Jülich Supercomputing Centre  
c.roessel@fz-juelich.de

EU-US HPC Summer School  
Dublin
Measurement Methods: Tracing

- Recording **information about** significant points (**events**) during execution of the program
  - Enter/leave a code region (function, loop, …)
  - Send/receive a message ...

- Save information in **event record**
  - Timestamp, location ID, event type
  - plus event specific information

- Abstract execution model on level of defined events

- **Event trace** = *Chronologically ordered sequence of event records*
## Event tracing

### Process A
```c
void foo() {
    trc_enter("foo");
    ...
    trc_send(B);
    send(B, tag, buf);
    ...
    trc_exit("foo");
}
```

### Process B
```c
void bar() {
    trc_enter("bar");
    ...
    recv(A, tag, buf);
    trc_recv(A);
    ...
    trc_exit("bar");
}
```

### Local trace A
- **Local trace A**
- **global trace**
- **merge**
- **unify**

### Global trace

<table>
<thead>
<tr>
<th>Action</th>
<th>Process</th>
<th>Event</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTER</td>
<td>A</td>
<td>58</td>
<td>1</td>
</tr>
<tr>
<td>SEND</td>
<td>A</td>
<td>62</td>
<td>B</td>
</tr>
<tr>
<td>EXIT</td>
<td>A</td>
<td>64</td>
<td>1</td>
</tr>
<tr>
<td>ENTER</td>
<td>B</td>
<td>60</td>
<td>2</td>
</tr>
<tr>
<td>SEND</td>
<td>B</td>
<td>62</td>
<td>A</td>
</tr>
<tr>
<td>EXIT</td>
<td>B</td>
<td>64</td>
<td>1</td>
</tr>
<tr>
<td>RECVR</td>
<td>B</td>
<td>68</td>
<td>A</td>
</tr>
<tr>
<td>EXIT</td>
<td>B</td>
<td>69</td>
<td>2</td>
</tr>
</tbody>
</table>

### Local trace B
- **Local trace B**
- **global trace**
- **merge**
- **unify**

<table>
<thead>
<tr>
<th>Action</th>
<th>Process</th>
<th>Event</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTER</td>
<td>A</td>
<td>60</td>
<td>1</td>
</tr>
<tr>
<td>RECVR</td>
<td>A</td>
<td>68</td>
<td>1</td>
</tr>
<tr>
<td>EXIT</td>
<td>A</td>
<td>69</td>
<td>1</td>
</tr>
<tr>
<td>RECVR</td>
<td>B</td>
<td>68</td>
<td>A</td>
</tr>
<tr>
<td>EXIT</td>
<td>B</td>
<td>69</td>
<td>2</td>
</tr>
</tbody>
</table>
Event Tracing: “Timeline” Visualization

<table>
<thead>
<tr>
<th></th>
<th>foo</th>
<th>bar</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>58</td>
<td>A</td>
<td>ENTER</td>
<td>1</td>
</tr>
<tr>
<td>60</td>
<td>B</td>
<td>ENTER</td>
<td>2</td>
</tr>
<tr>
<td>62</td>
<td>A</td>
<td>SEND</td>
<td>B</td>
</tr>
<tr>
<td>64</td>
<td>A</td>
<td>EXIT</td>
<td>1</td>
</tr>
<tr>
<td>68</td>
<td>B</td>
<td>RECV</td>
<td>A</td>
</tr>
<tr>
<td>69</td>
<td>B</td>
<td>EXIT</td>
<td>2</td>
</tr>
</tbody>
</table>

...
Why Tracing?

- Event traces preserve the **temporal** and **spatial** relationships among individual events (⇒ context!)

- Can be used to reconstruct the **dynamic behavior**
  - Detect time-dependent behavior
  - Detect "bogus" balanced code
  - Get *insight into causes of communication and synchronization bottlenecks*
  - Find opportunities to overlap computation and communication
Issues with Tracing

- **Amount of trace data very quickly becomes overwhelming**
  - High processor counts
  - Long running programs

  - Very selective instrumentation is critical to manage data
  - Isolate important phases of program (e.g. computational kernel)
    - Find these with profiling measurements
  - Trace minimum number of (selected) iterations

- **Limited memory available for trace buffers results in flushing**
  - Huge runtime overhead / distortion
  - Avoid (unsynchronized) flushing at all costs!

- **Complexity of manually analyzing large traces is an issue**
  - Scalasca provides automated analysis of traces to determine communication and synchronization problems
Trace Visualizers

- **Jumpshot (ANL)**
  - Free, basic MPI visualizer (routines, messages)
  - SLOG-2 format
  - MPE tracing + converters from TAU, (EPILOG)
- **Paraver (BSC)**
  - Free, extremely flexible and programmable visualizer
  - PRV format
  - Extrae tracing + converters from TAU, EPILOG, (OTF)
- **Vampir (TUD)**
  - Commercial portable trace visualizer
  - VTF3 (obsolete), OTF, EPILOG format
  - VampirTrace, Scalasca, TAU tracing + many converters
- **Intel Trace Collector and Analyzer**
  - Commercial, intel-only trace collection and visualizer
Vampir Event Trace Visualizer

- **Visualization and Analysis of MPI Programs**
- **Commercial product**

- Originally developed by Forschungszentrum Jülich
- Now all development by Technical University Dresden
Vampir: Time Line Diagram

- Functions organized into **groups**
- Coloring by group
- Message lines can be colored by tag or size
- Information about states, messages, collective and I/O operations available through clicking on the representation
Vampir: Process and Counter Timelines

- Process timeline show call stack nesting
- Counter timelines for hardware or software counters
Vampir: Execution Statistics

- Aggregated profiling information: execution time, number of calls, inclusive/exclusive

- Available for all / any group (activity) or all routines (symbols)

- Available for any part of the trace ⇒ selectable through time line diagram
Vampir: Process Summary

- Execution statistics over all processes for comparison
- Clustering mode available for large process counts
Vampir: Communication Statistics

- Byte and message count, min/max/avg message length and min/max/avg bandwidth for each process pair
- Message length statistics

Available for any part of the trace
VampirServer:
PEPC, 16384 PEs, Global Timeline
VampirServer:
PEPC, 16384 PEs, Global Timeline (zoomed)
VampirServer:
PEPC, 16384 PEs, Message Statistics
VampirServer: PEPC, 16384 PEs, Cluster Analysis
“A picture is worth 1000 words…”

- MPI ring program
- “Real world” example
“What about 100’s of pictures?”
(with 100’s of menu options)
Example Automatic Analysis: Late Sender
Example Patterns

(a) Late Sender
(b) Late Receiver
(c) Late Sender / Wrong Order
(d) Wait at N x N

ENTER   EXIT   SEND   RECV   COLLEXIT
Basic Idea **Automatic Performance Analysis**

- **“Traditional” Tool**
  - Huge amount of Measurement data
  - For non-standard / tricky cases (10%)
  - For expert users

- **Automatic Tool**
  - Simple: 1 screen + 2 commands + 3 panes
  - Relevant problems and data

- Pattern Analyzer

⇒ More productivity for performance analysis process!

- For standard cases (90% ?!)
- For “normal” users
- Starting point for experts
The Scalasca Project

- **Scalable Analysis of Large Scale Applications**
- Started in January 2006

- Objective 1: do not rely on tracing only
  ⇒ Supports scalable **call-path profiling**
- Objective 2: develop a scalable trace analysis
  ⇒ Basic idea: **parallelization of trace analysis**

- Supports MPI 2.2 (P2P, collectives, RMA, IO), basic OpenMP (no nesting), or hybrid
- [http://www.scalasca.org/](http://www.scalasca.org/)
Scalasca Overview

• **Goal**
  - Measure and analyze performance of parallel programs

• **Context**
  - “Real-world” HPC applications
  - Batch processing
  - Large distributed memory machines

• **Current focus**
  - Automation
  - Portability
  - Extreme scalability w.r.t. number of cores
CUBE Result Browser

• Representation of results (3D severity matrix) along three hierarchical axes
  ▪ Metric
  ▪ Call tree path
  ▪ System location

• Three coupled tree browsers

• Each node displays severity
  ▪ As colour: for easy identification of bottlenecks
  ▪ As value: for precise comparison
CUBE Result Browser (II)

- Each node displays severity
  - as colour
  - as value

- Dependent on state
  - **Collapsed**
    - **Inclusive** time
    - Entire time spent in the function
  - **Expanded**
    - **Exclusive** time
    - Time spent in the function without taking calls to children into account
CUBE Result Browser (III)

Value boxes colored according to scale
Metric Dimension

What kind of performance problem?

Right-click metric context menu for info or description
Call Tree Dimension

Where is it in the source code? In what context?

Right-click function context menu to go to source location or to manipulate tree
System Tree Dimension

How is it distributed across the system
Alternative: Topology View

View ➞ Topology menu adjusts topology view
Summary analysis sweep3D@294,912
Trace analysis sweep3D@294,912

- 10 min sweep3D runtime
- 11 sec replay
- 4 min trace data write/read (576 files)
- 7.6 TB buffered trace data
- 510 billion events
Scalasca Unified Command: `scalasca`

- Run without action argument for basic usage info
  
  ```
  % scalasca
  ```

1. prepare application objects and executable for measurement:
   `scalasca -instrument <compile-or-link-command>`  # skin

2. run application under control of measurement system:
   `scalasca -analyze <application-launch-command>`  # scan

3. interactively explore measurement analysis report:
   `scalasca -examine <experiment-archive|report>`  # square

[-h] show quick reference guide (only)

- **For full details see Scalasca Quick Reference Guide**
Scalasca Instrumenter: skin

- **Usage:** `scalasca -instrument [opts] <compile_cmd>`
  - `scalasca -instrument mpicc -fast -c bar.c`
  - `skin mpif90 -Openmp -O3 -o foobar foo.c bar.o -lm`

- Processes source modules during compile and augments link with measurement library
  - Configures automatic function instrumentation capability of native compiler (if available)
    - All functions in source module(s) are instrumented
  - `[-pomp]` option enables processing of POMP directives
    - Optional manual source annotation of functions + regions
    - Replaces automatic function instrumentation
  - `[-user]` option activates Scalasca user-annotation API
  - `[-pdt]` option uses TAU PDT instrumenter
Scalasca Collector + Analyzer:  

**Usage:** `scalasca -analyze [opts] <launch_cmd>`

- **scan** `mpirun -np 2 ./myprog infile`

**Prepares and runs measurement collection, with follow-on trace analysis (if appropriate):**

- `[-n]` preview without executing launches
- `[-s]` enables runtime summarization [default]
- `[-t]` enables trace collection & automatic pattern analysis

**names default measurement experiment archive**

`epik_${(TARGET)}_${(MODE)}${(NP)}x${(NT)}_[sum|trace]`

- `[-f filter]` specifies file listing functions not to be measured
- `[-m metric1:metric2:...]` includes HW ctr metrics
Scalasca Analysis Report Explorer: square

- **Usage:** `scalasca -examine [opts] <epik_archive>
  - `scalasca -examine epik_sweep3d_co32_trace`
  - `square epik_sweep3d_co32_trace`
- Prepares and presents final analysis report
  - Checks EPIK archive directory for CUBE files
  - Remaps primitive initial analysis report(s) into refined formal report(s) with enriched metrics + metric hierarchies
    - `epitome.cube ⇒ summary.cube`
    - `scout.cube ⇒ trace.cube`
  - Presents refined report in CUBE3 browser
    - Trace analysis shown in preference to summary analysis
    - Additional reports can be loaded via File/Open menu
  - `[-s]` skip display and output textual score report
CUBE Algebra Tools

- CUBE files can be compared/combined with some useful command line tools
- Note that these work directly on CUBE files and not on archive directories

General usage:
- `cube3_TOOL [-o <output file>] <input file>`
CUBE Algebra Tools (II)

- **cube3_mean**
  - Can eliminate “measurement noise” by averaging the results of several experiments

- **cube3_cut [-p prune] [-r root]**
  - Creates a new CUBE file without pruned subtrees and/or containing only the specified call tree node as new root(s)

- **cube3_diff**
  - Calculates the difference of two experiments
  - Useful to measure improvement/degradation due to a modification
Diff Example

Negative value
[Sunken color box]  ⇒ Degradation

Positive value
[ Raised color box]  ⇒ Improvement
Root Cause Analysis

- **Root-cause analysis**
  - Wait states typically caused by load or communication imbalances earlier in the program
  - Waiting time can also propagate (e.g., second order waiting time)
  - Goal: Enhance performance analysis to find the root cause of wait states

- **Approach (work in progress)**
  - Distinguish between first and higher order waiting time
  - Identify call path/process combinations delaying other processes and causing first order waiting time
  - Identify original delay

---

```
sendrecv
```

- `foo bar`  
  - `Send`  
  - `Delay`  

- `foo bar`  
  - `Recv`  
  - `Send`  

- `foo bar`  
  - `Recv`  
  - `Delay`
Scalasca Limitations

- **MPI 2.2 apart from dynamic process creation**
  - C++ interface deprecated with MPI 2.2

- **OpenMP 2.5 apart from nested thread teams**
  - partial support for dynamically-sized and/or conditional thread teams (*)
    - automatic trace analysis currently not supported
    - no support for OpenMP used in macros or included files

- **No OpenMP 3.0 tasks yet (work in progress)**

- **Hybrid OpenMP+MPI**
  - partial support for non-uniform thread teams (*)
  - no support for MPI_THREAD_MULTIPLE

(*) Summary & trace measurements are possible, and traces may be analyzed with Vampir or other trace visualizers
Scalasca Outlook

• Replace EPIK measurement component with Score-P
  ▪ Score-P writes profiles and traces in CUBE4 and OTF2 format
  ▪ One instrumentation and measurement-configuration interface
  ▪ Will be used by Vampir, TAU, Periscope, …
  ▪ Supports OpenMP 3.0 tasking
  ▪ Node-Level, external counters
  ▪ CUDA, HMPP, OmpSs, Pthreads (*work in progress*)
  ▪ [http://www.score-p.org](http://www.score-p.org)
The Scalasca Team

JSC

- Michael Knobloch
- Bernd Mohr
- Peter Philippen
- Markus Geimer
- Daniel Lorenz
- Christian Rössel
- Marc Schlütter
- Ilija Zhukov
- Pavel Saviankou
- Alexandre Strube
- Brian Wylie

GRS

- David Böhme
- Marc-André Hermanns
- Christian Siebert
- Felix Wolf

• Sponsors
Thank you!

WRF-NMM weather prediction code on MareNostrum @ 1600 CPUs

http://www.scalasca.org
scalasca@fz-juelich.de
Acknowledgments

- Bernd Mohr, Brian Wylie, Markus Geimer (JSC)
- Andreas Knüpfer, Jens Doleschal (TU Dresden)
Further Documentation

- http://www.vi-hps.org/training/material/
  - Performance Tools LiveDVD image
  - Links to tool websites and documentation
  - Tutorial slides