Mitglied der Helmholtz-Gemeinschaft

JÜLICH FORSCHUNGSZENTRUM

Analysis tools for the results of Scalasca

Guest Student Programme 2010

October 19, 2010 | Markus Mayr (Vienna University of Technology)

What is Scalasca? (1/2)

Scalasca is a ...

Performance analysis tool set

for parallel applications

Who is involved?

- Started in January 2006 as follow-up project to KOJAK
- Jointly developed by JSC & GRS
- Developed in collaboration with ICL/UT

What is Scalasca? (2/2)

Performance analysis tool set?

- How much time was spent? And where?
- How much communication happened where?
- How much time was spent waiting for other processes?

Features

- Scalable
- Automatic pattern-based performance analysis
- Open source & Portable
- Various languages & parallel programming paradigms

Motivation

My project is about ...

- Providing a testing framework for Scalasca analysis data
- Checking data for sanity
- Comparing data to reference data

- Before my project: high manual effort
- After it: Flexible tool set to automatize many steps

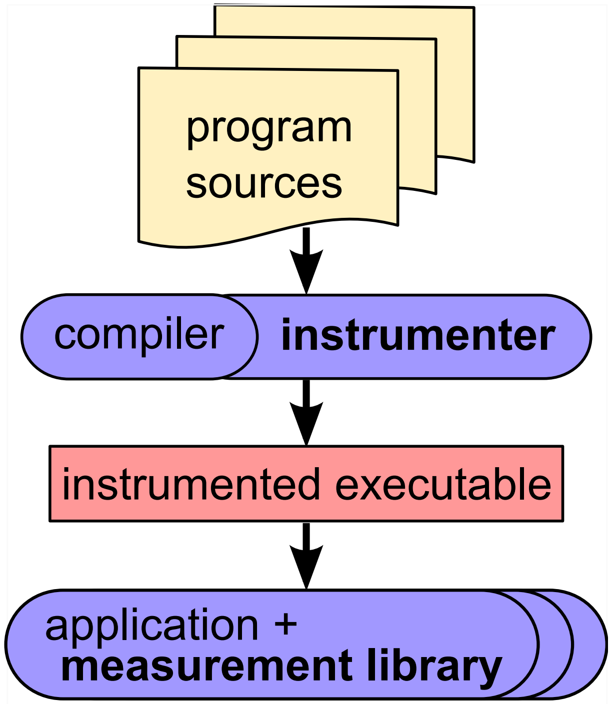
Application instrumentation

Automatic / Manual code instrumenter

- Processes program sources
- Adds instrumentation and measurement library into application executable

Measurement library

- Exploits MPI standard profiling interface (PMPI)
- Provides measurement infrastructure & instrumentation API



Runtime summarization and Tracing

Runtime summarization

- Summarizes by thread & call-path
- Lower memory requirements, less overhead

Trace-based analysis

- Time-stamped events buffered during measurement for each thread
- Follow-up analysis replays events and produces extended analysis report
- Only relatively few events can be recorded (memory!)

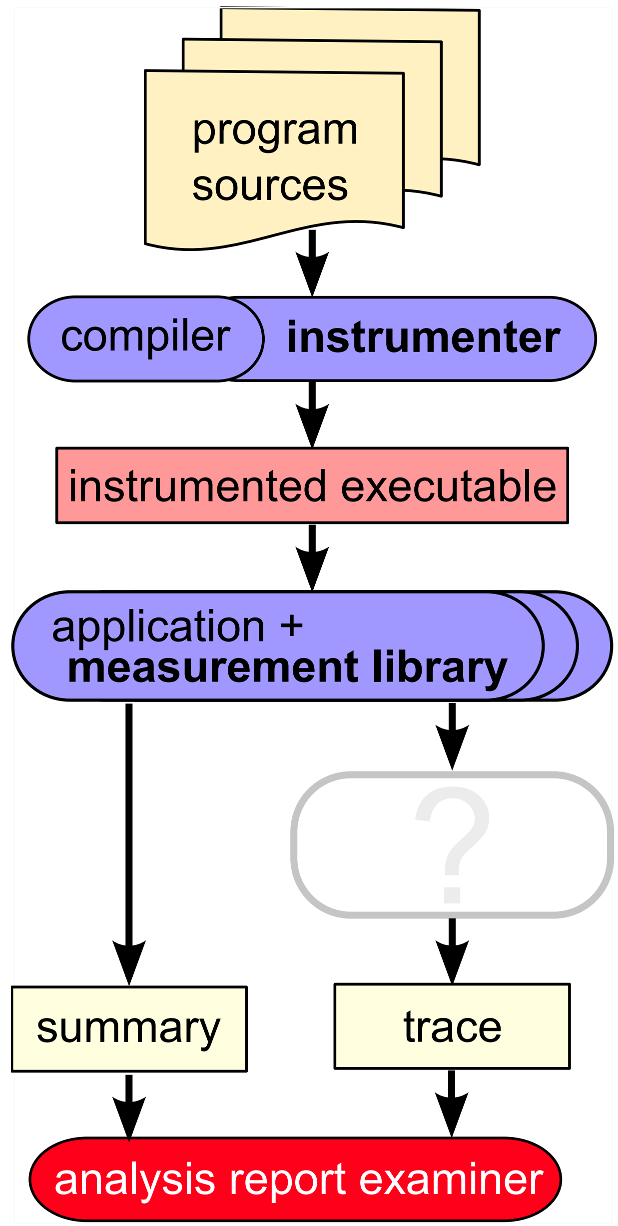
Overall structure of Scalasca

Components

- Automatic / Manual code instrumenter
- Unified measurement library
- Common analysis report examiner

Remarks

- Details about generation of trace-based analysis left out
- My work: Same stage as analysis report examiner, post-processing



Structuring Measurement Data

Basic terms

- Metric: What is measured, e.g. Time, Visits, Bytes send
- Call tree: Which function called which one
- Machine: Where on the machine has the data been measured, i.e. thread number

$\Longrightarrow$ Measurement data is a map from

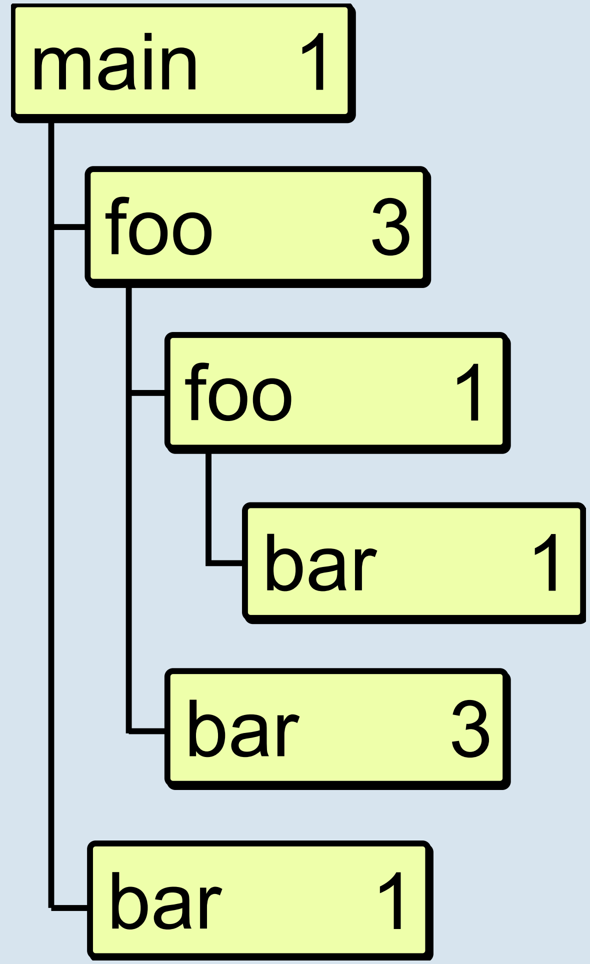
$$\{\text{Metrics}\} \times \{\text{Call tree nodes}\} \times \{\text{Thread numbers}\} \to \mathbb{R}.$$

Call Tree: An Example

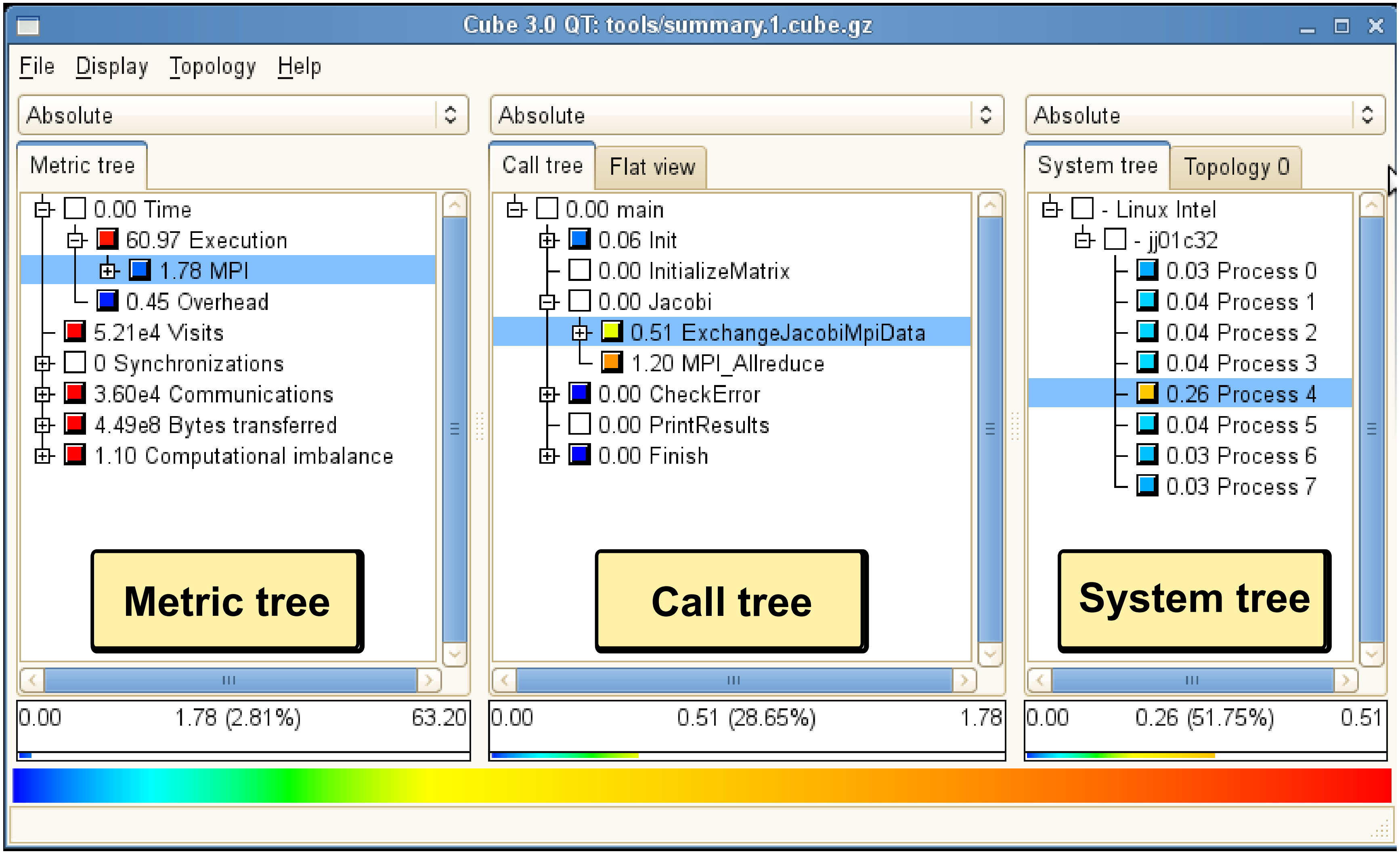
Source Code

```
void bar() {}
void foo(int n) {
  if (n >= 1)
    foo(--n);
  bar();
}
main() {
  foo(1);
  foo(0);
  bar();
  foo(0);
}
```

Call tree



What is Cube?


Cube 3.0 QT: tools/summary.1.cube.gz
File Display Topology Help
Absolute
Absolute
Absolute
Metric tree
Call tree
Flat view
System tree
Topology 0
0.00 Time
60.97 Execution
1.78 MPI
0.45 Overhead
5.21e4 Visits
0 Synchronizations
3.60e4 Communications
4.49e8 Bytes transferred
1.10 Computational imbalance
0.00 main
0.06 Init
0.00 InitializeMatrix
0.00 Jacobi
0.51 ExchangeJacobiMpiData
1.20 MPI_Allreduce
0.00 CheckError
0.00 PrintResults
0.00 Finish
- Linux Intel
- jj01c32
0.03 Process 0
0.04 Process 1
0.04 Process 2
0.04 Process 3
0.26 Process 4
0.04 Process 5
0.03 Process 6
0.03 Process 7
Metric tree
Call tree
System tree
0.00
1.78 (2.81%)
63.20
0.00
0.51 (28.65%)
1.78
0.00
0.26 (51.75%)
0.51

Organizing Data

Organizing measurement space with trees

- Organize metrics, call tree, machines in Trees
- Metric, example: Communication parent of *Point-to-Point* and *Collective*
- Call tree
- Machine: Machine - Nodes - Processors - Threads

Exclusive and Inclusive Values ...

- Exclusive value, e.g. how much time was spent in function X?
- Inclusive value, e.g. how much time was spent in function X and all functions X called on this call path?

Other Tools for Report Manipulation

Cube3 ”algebra tools”

Algebraic tools

- Compute difference: `cube3_diff`
- Merge files, create mappings
- Cut and re-root trees

Statistic tools

- Print out metrics, function types
- Statistical information

Short Introduction to Cube File Format

Basic characteristics

- (Compressed) XML format.
- Contains all meta data and all measurement data.
- For report viewing and manipulation tools.

Scalasca's two flavors of Cube files

- Trace file, more metrics
- Summary file

System Tree and Metrics

System Tree

- Complex entity of different data structures: Machine, Node, Process, Thread
- Not too important for my topic

Metric Tree

- Metric contains basic information like name, data type.
- Tree structure represented in XML file.
- Amount of metrics depends on certain factors, e.g. trace or summary file, MPI and/or OpenMP, etc.

Regions and Cnodes

Region

- Represents a function, subroutine, loop, block, etc.
- Basic information like position in source
- No structural information

Call tree node $\rightarrow$ Cnode

- Encapsulates information about calling relationships between Regions.
- Generally: Call graph
- For Cube: Tree structure and cut-off at certain level

Need for a Testing Framework

Current situation

- Scalasca supports high number of architectures, compilers
- Testing across all these done manually $\rightarrow$ high effort

A brighter future?

- Automatic sanity checks for cube files
- Automatic comparison between cube files
- Meaningful reports about suspicious things for Scalasca developers.

Basic Requirements I

For Scalasca Developers

- Get meaningful reports without manual effort.
- Compare slightly different call trees, for example caused by inlining or other optimizations.
- Identify corresponding Regions that got slightly different information at instrumentation.

Basic Requirements II

For Test Writers

- Fuzzy and exact matching
- Selection of call tree subsets to apply tests on
- Definition of new metrics
- Consistent ways to report errors
- Flexibility to cover future testing scenarios.

Starting Point cube3_info

Purpose

- Command-line tool, prints out metrics for cube files
- Compares multiple cube files side-by-side
- Gives access to custom metrics
- Means to show most relevant parts of tree
- Tool to test part of testing library during development

Usage example

```
cube3_info -r metric,threshold -m metric <file1> <file2>
```

cube3_info: An example

```
cube3_info -r time,0.05 -m time summary.cube.gz
```

```
|      Time | Diff-Call tree
|   63.2041 |  * main
|    0.0672 |  |  * ***** Aggr.  4 children (+3 nodes)
|    0.5103 |  |  * Init
|    0.0008 |  |  |  * ***** Aggr.  3 children (+0 nodes)
|    0.5144 |  |  |  * MPI_Init
|    0.4532 |  |  |  |  * TRACING
|   62.6150 |  |  * Jacobi
|   21.9683 |  |  |  * ExchangeJacobiMpiData
|    0.0648 |  |  |  |  * ***** Aggr.  2 c (+0 nodes)
|    0.4442 |  |  |  |  * MPI_Waitall
|    1.2047 |  |  |  * MPI_Allreduce
```

Canonicalizing Regions

Need for Canonicalization

- Region names, file names, line numbers may differ
- $\Longrightarrow$ Problems for merging tool

cube3_canonicalize solves this by ...

- Removing additional information
- Transforming region name

Future improvements

- Store deleted information externally

Trees not Matching Exactly

Reasons

- Compiler optimizations
- Differently operating instrumenting tools

Handling of a concerned node

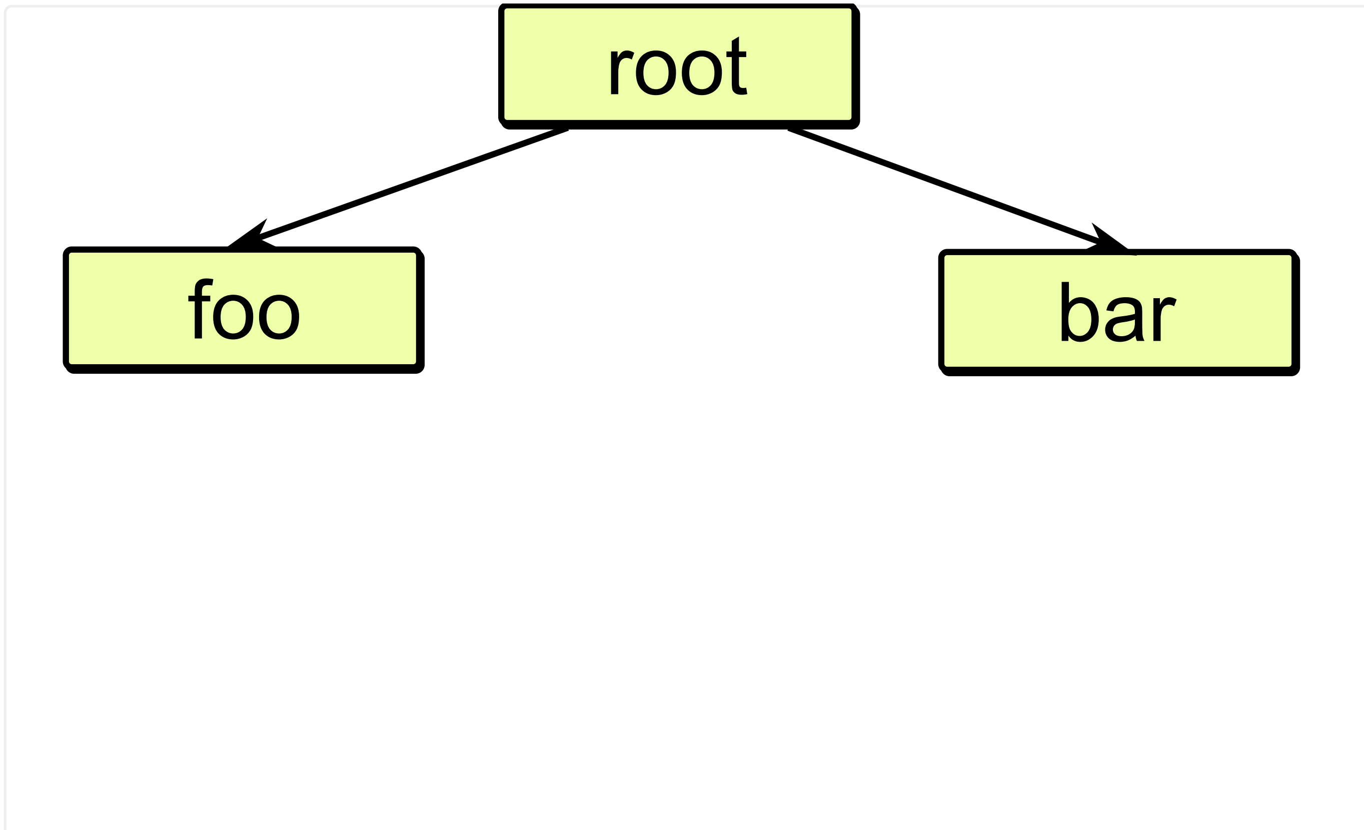
1. Merge node's children with parent's children
2. Add (exclusive) metrics to parent's metrics
3. Remove node

Implemented as part of cube3_cut.

Cut by example

In this example, we remove “remove”

1. Merge children
2. Add metrics
3. Remove node



$\Longrightarrow$ This is what would have happened if “remove” had not been instrumented (or inlined).

Testing Tool Chain

Modular tool chain

1. Sanity checks
2. Canonicalize
3. Region-based checks
4. Filter tree, make them match
5. Merge & Call tree node based checks

Problems with this Design

- Overhead: Loading/Saving file
- Most users will want all steps anyway

Special Metrics: CnodeMetric

Basic concept

- Is a map

$$\{\text{Call tree nodes}\} \to \mathbb{R}$$

- Information about metrics, system resources, etc. encapsulated
- Easily accessible for testing library
- Supports Caching (done by testing library)

Examples

- Aggregated metrics
- The visitors metric

Selection of Subsets

Required because ...

- Hide irrelevant nodes
- Fuzzy matching
- Restrictions on type, file, etc.

Usage example

```
MdAggrCube cube << some_ostream;
CnodeSubForest* all = cube.get_forest();
CnodeSubForest* time_relevant = new CnodeSubForest(all);
AbridgeTraversal("time", .1).run(time_relevant);
DiffPrintTraversal(
  vector<string>(1, "time"), cout).run(time_rel);
```

Specification of Tests I

Simplified Example

```
class CnodesSimilarTime :  public CnodeConstraint {
 public:
  CnodesSimilarTime(CnodeSubForest* forest)
    :  CnodeConstraint(forest, vector<string>(1,"time"))
  virtual string get_name() { return "Time similar"; }
  virtual string get_description() { return "desc"; }
  virtual void cnode_handler(CCnode* n, CnodeMetric* m)
  {
    if (fabs( m->compute(n, (unsigned int) 0)
          - m->compute(n, (unsigned int) 1)) > 1e-5)
      return fail(n,m,"difference too big.");
    ok();
   }
};
```

Specification of Tests II

Basic concepts

- Base class: `AbstractConstraint`
- Tests structured in tree
- Common output routines
- Extended through sub-classing

`CnodeConstraint` is ...

- a sub-class of `AbstractConstraint`
- for test writers' convenience

Outlook, Future Developments

Reveal inadequatenesses by ...

- Testing in practice
- Formulating new tests

Some ideas:

- More integrated testing tool, simplification of testing
- Better ways to define tests, e.g.
  - *annotations* within cube files
  - exposing parts of cube to scripting language
- Better ways to determine severity of failures

JÜLICH FORSCHUNGSZENTRUM

Thanks for your attention!

Questions?