Recent Scalasca Research

Felix Wolf
02-08-2011
• Joint venture of
  – Forschungszentrum Jülich
  – RWTH Aachen University

• Four research laboratories
  – Computational biophysics
  – Computational engineering
  – Computational materials science
  – Parallel Programming

• Education
  – M.Sc. in simulation Sciences
  – Ph.D. program

• About 50 scientific staff members
Rheinisch-Westfälische Technische Hochschule Aachen

- Strong focus on engineering
- ~ 200 M€ third-party funding per year
- Around 31,000 students in over 100 academic programs
- > 5,000 international students from 120 different countries
- Cooperates with Jülich within the Jülich Aachen Research Alliance (JARA)
• Scalable performance-analysis toolset for parallel codes
  – Focus on communication & synchronization
• Integrated performance analysis process
  – Performance overview on call-path level via runtime summarization
  – In-depth study of application behavior via event tracing
• Supported programming models
  – MPI-1, MPI-2 one-sided communication
  – OpenMP (basic features)
• Available for all major HPC platforms
Scalasca

Scalasca is a software tool that supports the performance optimization of parallel programs by measuring and analyzing their runtime behavior. The analysis identifies potential performance bottlenecks – in particular those concerning communication and synchronization – and offers guidance in exploring their causes.

more...

News

7th VE-HPS Tuning Workshop
HLRS, Stuttgart/Germany, March 29-30, 2011 Three day hands-on workshop covering the...
moro...

Scalasca at SC’10
November 13-19, 2010: Join us at SC’10 in New Orleans, LA, USA. Scalasca team... more...
Outline

• Recent scalability improvements
• Mapping wait states onto their root cause
• Two approaches to low-overhead MPI profiling
  – Low-overhead direct instrumentation using prior static analysis
  – Reconciling sampling and direct instrumentation
Hierarchical unification

- Unification maps local identifiers of regions, call paths etc. onto global ones
  - Generation of a unified set of global definitions
  - Generation of local-to-global identifier mappings for each process
  - Writing the global definitions and the identifier mappings to disk.

Communicator management

- Scalasca records communicators in event traces
  - Needed for trace replay
- Previous method created multiple types of overhead
  - Memory due to replication of data across processes
  - Measurement dilation due to runtime rank translation
  - Unification of local communicator IDs
- New method creates global comm. ID at runtime
  - Stores information only once per communicator
  - Avoids runtime rank translation by storing translation tables at the end
  - Essentially eliminates unification

Incremental loading of report

- Currently, the entire report is loaded in to the GUI in one piece
  - Severe limitation of interactive experience
- In the future, data sets exceeding a certain size will be loaded incrementally
- At 288k processes, initial loading time reduced from 200s to 4s in prototype

Propagation of waiting time

- Direct waiting time
- Indirect waiting time
- Delay
Identifying delays in traces and assessing their cost

- Essentially scalable version of Meira Jr. et al.
- Classification of waiting times into
  - Direct vs. indirect
  - Propagating vs. terminal
- Attributes costs of wait states to delay intervals
  - Requires forward and backward replay

David Böhme et al.: Identifying the root causes of wait states in large-scale parallel applications. In Proc. of the 39th International Conference on Parallel Processing (ICPP), San Diego, CA, IEEE Computer Society, September 2010. Best Paper Award
Origin of delay costs in Zeus-MP/2

- Computation
- Waiting time
- Delay costs
Delay analysis of code Illumination

- Particle physics code (laser-plasma interaction)
- Delay analysis identified inefficient communication behavior as cause of wait states
Direct instrumentation

- Every entry or exit event of an instrumented function causes the measurement system to be called.
- Even instrumentation of all user functions plus MPI wrappers results in reasonable overhead (< 15%) in many cases.
- In others, overhead can be excessive.
  - Especially C++ codes with their many tiny methods.
- Filtering (black/white listing) can help reduce overhead.
  - Static: filtered functions are not instrumented in the first place.
  - Dynamic: invocation of the measurement system suppressed.
- Tradeoff: lower overhead but loss of information.
Configurable binary instrumenter (Cobi)

- Generating the filter usually requires extra run
- Idea: determine filter based on prior static analysis
  - Rules to specify analysis objectives (i.e., limit loss of information)
Cobi - filters

- Are specified in a separate XML file
- Start with all or no function
- Include/exclude functions with filter rules
- Rules can be combined by logical operators

Possible rules

- Are on call path
- Lines of source code
- Cyclomatic complexity
- Number of instructions
- Number and nesting level of loops
- Number of function calls
- Depth in call tree
- Name matching
- Prefix
- Suffix

Instrumented fraction of functions with filters

[C++ code and bar chart showing the fraction of functions instrumented across different projects and categories]
Runtime overhead in percent
Reconciling sampling and direct instrumentation

- Sampling allows better control of overhead
- But may miss critical events
  - Hard to capture accurate communication metrics
- New hybrid approach
  - Applies low-overhead sampling to user code
  - Intercepts MPI calls via direct instrumentation
  - Relies on efficient stack unwinding
  - Integrates measurements in statistically sound manner


Joint work with

Lawrence Livermore National Laboratory
How it works

Ignore samples inside MPI calls and restart timer

Account for extended / shortened effective interval length
Overhead of compiler instrumentation

- PMPI + compiler
- PMPI + compiler (with filter)
- PMPI (without callpath)
Overhead of hybrid method
SIONlib update

- Support for OpenMP and hybrid programs (MPI/OpenMP)
- Multi-file support in serial tools and API
- New installation process with configure tool
- Fortran interface enhanced

http://www.fz-juelich.de/jsc/sionlib/
Summary

• Ensuring scalability is continuous labor-intensive effort
  – Next step: validation of new enhancements in concert
• Delay analysis offers new insight into the actual cost of load and communication imbalance
• No single cure for measurement dilation
  – However, combination of different methods often successful
Thank you!