Application Performance Analysis on Petascale Systems

Felix Wolf, Brian J. N. Wylie
2010-04-19
• Joint venture between
  – Forschungszentrum Jülich
  – RWTH Aachen University

• Founded in 2007

• Research and education in simulation sciences
  – International Master’s program
  – Ph.D. program
Jülich Supercomputing Centre

Research in
• Computational Science
• Computer Science
• Mathematics

Jülich BG/P 294,912 cores
Jülich Nehalem Cluster 26,304 cores
Outline

• Motivation
• Scalasca overview
• Scalable trace analysis
• Scalable task-local I/O
• Space-efficient time-series call-path profiles
• Conclusion & outlook
Higher degrees of parallelism

• Increasing complexity of applications
  – Higher resolutions
  – Larger simulated time periods
  – Multi-physics
  – Multi-scale

• Increasing parallelism
  – Multi-core
Higher degrees of parallelism (2)

- Also new demands on **scalability of software tools**
  - Familiar tools cease to work in a satisfactory manner for large processor counts

- Optimization of applications more difficult
  - Increasing machine complexity
  - Every doubling of scale reveals a new bottleneck

- Need for scalable performance tools
  - Intelligent
  - Robust
  - Easy to use
• Scalable performance-analysis toolset for parallel codes
• Integrated performance analysis process
  – Performance overview on call-path level via runtime summarization
  – In-depth study of application behavior via event tracing
  – Switching between both options without recompilation or relinking
• Supported programming models
  – MPI-1, MPI-2 one-sided communication
  – OpenMP (basic features)
• Available under the New BSD open-source license
  – http://www.scalasca.org/

Joint project of
Scalasca team
Event tracing

• Typical events
  – Entering and leaving a function
  – Sending and receiving a message
• Problem: width and length of event trace
Scalable trace analysis via parallel replay

• Exploit distributed memory and processing capabilities
  – Keep trace data in main memory
  – Traverse local traces in parallel
  – Exchange data at synchronization points of target application using communication operation of similar type

• Four applications

<table>
<thead>
<tr>
<th>Wait-state analysis</th>
<th>Delay analysis</th>
<th>Synchronization of timestamps</th>
<th>Evaluation of optimization hypotheses</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Parallel replay</td>
</tr>
</tbody>
</table>
Wait-state analysis

- Classification
- Quantification

(a) Late Sender

(b) Late Sender / Wrong Order

(c) Late Receiver
Wait-state analysis (2)
Scalability of parallel wait-state search (SWEEP3D)

The graph shows the time (in seconds) required for different execution methods as a function of the number of processes. The methods include:

- Uninstrumented execution
- Parallel trace analysis
- Parallel trace replay

The graph indicates that the time increases with the number of processes, with the uninstrumented execution method taking the longest time.
Redundant messages in XNS CFD code

Previous peak performance at 132 ts/h

Now scales up to 4096 processes
XNS wait-state analysis of tuned version
Delay analysis

- Delay counterpart of waiting time
- Distinction between direct and indirect waiting times
- Essentially scalable version of Meira Jr. et al.
- Analysis attributes costs of wait states to delay intervals

[Diagram showing the relationship between process, time, delay, direct waiting time, and indirect waiting time]
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

Computation
Short-term costs of indirect delay: Original vs. optimized code
Costs of direct delay in optimized code
Insufficiently synchronized clocks on clusters

- Misrepresentation of logical event order in traces
- Distorted interval lengths
- Simple approach: linear offset interpolation
- Problem unstable drifts
Postmortem correction using logical clocks

- Controlled logical clock algorithm by Rolf Rabenseifner
  - Restores logical event order based on happened-before relation while introducing only marginal local inaccuracies
  - Shortcoming 1: Only point-to-point communication
  - Shortcoming 2: Sequential algorithm
- Extended to cover MPI collective communication and OpenMP shared-memory programming
  - Mapped collectives and OpenMP regions onto point-to-point messages
- Parallelized through parallel replay
  - Challenge: backward replay required to smooth jump discontinuities without introducing new violations
  - Scalability tested on up to 4,000 processors
Evaluation of optimization hypotheses

• Wait states often caused by load or communication imbalance occurring much earlier in the program
  – Hard to estimate impact of potential changes
  – Requires modeling the communication infrastructure to answer “What if…?” questions

• Alternative
  – Parallel real-time replay of modified event traces to verify hypotheses on wait state formation
    • Elapse computation time
    • Re-enact communication
  – Advantage: scalability and accuracy
Simulated removal of redundant messages

- 1 iteration of 1024 processor-run of XNS on Blue Gene/L
- All zero-sized messages removed from trace
  - 90% of all messages > 1.2 billion messages

<table>
<thead>
<tr>
<th>Metric</th>
<th>Original</th>
<th>Hand-Optimized</th>
<th>Simulated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>100.0</td>
<td>50.6</td>
<td>53.1</td>
</tr>
<tr>
<td>MPI</td>
<td>59.9</td>
<td>16.9</td>
<td>19.4</td>
</tr>
<tr>
<td>P2P</td>
<td>54.2</td>
<td>8.6</td>
<td>11.2</td>
</tr>
<tr>
<td>Late Sender</td>
<td>30.6</td>
<td>5.7</td>
<td>8.0</td>
</tr>
<tr>
<td>Wait at Barrier</td>
<td>5.1</td>
<td>7.7</td>
<td>7.7</td>
</tr>
</tbody>
</table>
SIONlib: Scalable parallel I/O for task-local data

- Use case: task-local binary I/O from thousands of tasks
  - Trace files
  - Scratch/checkpoint files
- Often does not scale
  - Contention at metadata server
  - File handling (e.g., directory listing)
- Map many logical files onto a few physical files
  - Application-level file system
  - Optimized I/O via block alignment
Parallel open / create on JUGENE

![Graph showing the time (s) for creating files and SION create files with varying number of tasks. The graph indicates an increasing trend in time as the number of tasks increases.](image)
Time-dependent performance behavior

MPI point-to-point time of 129.tera_tf
Time-series call-path profiling

- Manual instrumentation to distinguish iterations of the main loop
- Complete call-tree recorded for each iteration
  - With multiple metrics collected for every call-path
- Huge growth in the amount of data collected
  - Reduced scalability
Incremental on-line clustering

- Exploits that many iterations are very similar
  - Summarizes similar iterations in a single iteration, their average
- On-line to save memory at run-time
- Process-local to
  - Avoid communication
  - Adjust to local temporal patterns
- The number of clusters can never exceed a predefined maximum
  - Merging of the two closest ones
• Simulates mold-filling in casting processes
• Scalasca used
  – To identify communication bottleneck
  – To compare alternatives using performance algebra utility
• 23% overall runtime improvement
• Further investigations ongoing
Conclusion

• Integrated tool architecture
• Scalability in terms of **machine size**
  – Trace-processing based on parallel replay
    • Versatile: four applications
  – Parallel task-local I/O
  – Demonstrated on up to 295 K cores
• Scalability in terms of **execution time**
  – Runtime compression of time-series profiles
Outlook

- Further scalability improvements
  - Parallelization of internal management operations
  - Scalable output format and GUI
  - In-memory trace analysis
- Emerging architectures and programming models
  - PGAS languages
  - Accelerator architectures
- Interoperability with 3rd-party tools
  - Common measurement library for several performance tools
Thank you!
Sweep3D – late sender
Sweep3D – execution time

[Diagram of Sweep3D execution time metrics, including call trees and time breakdowns.]