“A picture is worth 1000 words...”
“But what about 1000’s of pictures?”
Automatic trace analysis

- **Idea**
  - Automatic search for patterns of inefficient behavior
  - Classification of behavior & quantification of significance

- **Advantages**
  - Guaranteed to cover the entire event trace
  - Quicker than manual/visual trace analysis
  - Helps to identify hot-spots for in-depth manual analysis
    - Complements the functionality of other tools
Higher degrees of parallelism

Number of Cores share for TOP 500 November 2012

<table>
<thead>
<tr>
<th>NCore</th>
<th>Count</th>
<th>Share</th>
<th>$\Sigma R_{max}$</th>
<th>Share</th>
<th>$\Sigma N_{Core}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1025-2048</td>
<td>1</td>
<td>0.2%</td>
<td>122 TF</td>
<td>0.1%</td>
<td>1,280</td>
</tr>
<tr>
<td>2049-4096</td>
<td>2</td>
<td>0.4%</td>
<td>155 TF</td>
<td>0.1%</td>
<td>7,104</td>
</tr>
<tr>
<td>4097-8192</td>
<td>81</td>
<td>16.2%</td>
<td>8,579 TF</td>
<td>5.3%</td>
<td>551,624</td>
</tr>
<tr>
<td>8193-16384</td>
<td>206</td>
<td>41.2%</td>
<td>24,543 TF</td>
<td>15.1%</td>
<td>2,617,986</td>
</tr>
<tr>
<td>&gt; 16384</td>
<td>210</td>
<td>42.0%</td>
<td>128,574 TF</td>
<td>79.4%</td>
<td>11,707,806</td>
</tr>
<tr>
<td>Total</td>
<td>500</td>
<td>100%</td>
<td>161,973 TF</td>
<td>100%</td>
<td>14,885,800</td>
</tr>
</tbody>
</table>

Average system size: 29,772 cores  
Median system size: 15,360 cores
Higher degrees of parallelism (II)

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

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

- Need for scalable performance tools
  - Efficient to meet performance expectations
  - Effective to use so that programmer productivity is maximized
The Scalasca project

- Project started in 2006
  - Follow-up to pioneering KOJAK project (started 1998)
    - Automatic pattern-based trace analysis
  - Initial funding by Helmholtz Initiative & Networking Fund
  - Many follow-up projects

- Objective:
  - Development of a scalable performance analysis toolset
  - Specifically targeting large-scale parallel applications
    - such as those running on IBM Blue Gene or Cray XT with 10,000s or 100,000s of processes

- Now joint project of
  - Jülich Supercomputing Centre
  - German Research School for Simulation Sciences
The Scalasca toolset

Scalasca 1.4.3

- Custom instrumentation & measurement system
- Scalasca trace analysis components based on custom trace format EPILOG
- Analysis report explorer & algebra utilities CUBE v3
- New BSD license

Scalasca 2.0β

- Community instrumentation & measurement system Score-P
- Scalasca trace analysis components based on community trace format OTF2
- Analysis report explorer & algebra utilities CUBE v4
- New BSD license

http://www.scalasca.org
Scalasca features

- Open source
- Portable
  - Blue Gene/Q, Blue Gene/P, IBM SP & blade clusters, SGI Altix, Solaris & Linux clusters
  - Scalasca 1.4.3 only: Cray XT, NEC SX, K Computer, Fujitsu FX10
- Supports parallel programming paradigms & languages
  - MPI, OpenMP & hybrid MPI+OpenMP
  - Fortran, C, C++
- Scalable trace analysis
  - Automatic wait-state search
  - Parallel replay exploits memory & processors to deliver scalability
Scalasca workflow

- Measurement library
  - HWC
- Instr. target application
- Local event traces
- Parallel wait-state search
- Wait-state report
- Summary report
- Instrumenter compiler / linker
- Instrumented executable
- Source modules

Optimized measurement configuration

Which problem?

Where in the program?

Which process?
Example: MPI patterns

(a) Late Sender

(b) Late Receiver

(c) Late Sender / Wrong Order

(d) Wait at N x N

M. Geimer | SEA Conference’13, Boulder, CO
Example: Late Sender

Sender:
Triggered by send event
Determine enter event
Send both events to receiver

Receiver:
Triggered by receive event
Determine enter event
Receive remote events
Detect \textit{Late Sender} situation
Calculate & store waiting time
10 min sweep3D runtime
11 sec analysis
4 min trace data write/read (576 files)
7.6 TB buffered trace data
510 billion events

Scalasca trace analysis bt-mz@1,048,704 BG/Q

Execution imbalance “z_solve”
Scalasca trace analysis bt-mz@1,048,704 BG/Q

Wait at implicit barrier “z_solve”
Research: 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., indirect waiting time)
- Goal: Enhance performance analysis to find the root cause of wait states

Approach

- Distinguish between direct and indirect waiting time
- Identify call path/process combinations delaying other processes and causing first order waiting time
- Identify original delay

![Diagram showing direct and indirect wait states]
CESM Sea Ice Module – Direct Wait Time
CESM Sea Ice Module – Indirect Wait Time
CESM Sea Ice Module – Delay Costs
Acknowledgements

Scalasca team (JSC)

Markus Geimer
Jie Jiang
Michael Knobloch
Daniel Lorenz
Bernd Mohr
Peter Philippen
Christian Rössel

Pavel Saviankou
Marc Schlütter
Alexandre Strube
Anke Visser
Brian Wylie
Ilja Zhukov

(GrS)

David Böhme
Alexandru Calotoiu
Marc-André Hermanns
Monika Lücke

Guoyong Mao
Aamer Shah
Sergei Shudler
Felix Wolf

Sponsors
Thank you!

http://www.scalasca.org