Performance Analysis with Scalasca

Carmen Navarrete

Content inputs from VI-HPS
More tools at: www.vi-hps.org
Scalasca: Automatic Trace Analysis

- **Background idea**
  - Automatic search for patterns of inefficient behaviour
  - Classification of behaviour and quantification of significance
  - Guaranteed to cover the entire event trace
  - Quicker than manual/visual trace analysis
  - Parallel replay analysis exploits available memory and processors to deliver scalability
The Scalasca project: Overview

- **Project started in 2006**
  - Initial funding by Helmholtz Initiative & Networking Fund
  - Many follow-up projects

- **Follow-up to pioneering KOJAK project (started 1998)**
  - Automatic pattern-based trace analysis

- **Now joint development of**
  - Jülich Supercomputing Centre
  - German Research School for Simulation Sciences
Scalasca Trace Tools: Objective

- Development of a scalable performance analysis toolset for most popular parallel programming paradigms (like MPI, OpenMP)

- Reliable analysis of large-scale parallel applications
  - such as those running on large systems like IBM BlueGene, SuperMUC or Cray systems with thousands or more processes/threads

- Latest release:
  - Scalasca v2.3.1 coordinated with Score-P v3.0
Scalasca Trace Tools features

- Open source, New BSD license
- Portable to many architectures
  - IBM Blue Gene, Cray XT/XE/XK/XC, SGI Altix, Fujitsu FX10/100 & K computer, Linux clusters, Intel Xeon Phi, ARM...
- Uses Score-P instrumenter and measurement libraries
  - Scalasca core package focuses on trace-based analyses
  - Supports common data formats
    - Reads event traces in OTF2 format
    - Writes analysis reports in CUBE4 format
- Current limitations:
  - Unable to handle traces
    - With MPI thread level exceeding MPI_THREAD_FUNNELED
    - Containing CUDA or SHMEM events, or OpenMP nested parallelism
  - PAPI/rusage metrics for trace events are ignored
Scalasca workflow

- Measurement library
- Instr. target application
- Instrumented executable
- Source modules

Optimized measurement configuration

- Score-P
- Scalasca trace analysis
- Local event traces
- Parallel wait-state search
- Wait-state report

- Instrumenter compiler / linker
- Instrumented executable

Summary report

Report manipulation

Intr.
application

- Measurement
library
- HWC

- Instr.
target
application
- Instrumenter
compiler / linker
- Source
modules

Optimized measurement configuration

- Score-P
- Scalasca trace analysis
- Local event traces
- Parallel wait-state search
- Wait-state report

Which problem?

Which process?

Where in the program?
Frequently Encountered Performance Issues
Example: Wait at NxN

- Time spent waiting in front of synchronizing collective operation until the last process reaches the operation

- Applies to:
  - MPI_Allgather, MPI_Allgatherv, MPI_Alltoall, MPI_Allreduce
  - MPI_Reduce_scatter, MPI_Reduce_scatter_block,
Example: Late Broadcast

- Waiting times if the destination processes of a collective 1-to-N operation enter the operation earlier than the source process (root)

- Applies to:
  - MPI_Bcast, MPI_Scatter, MPI_Scatterv
Waiting time caused by a blocking receive operation posted earlier than the corresponding send

Applies to blocking as well as non-blocking communication
Hands-on Session:
IvyMUC Cluster
Scalasca: Workflow commands

- One command for (almost) everything...

```bash
% scalasca
Scalasca 2.3.1
Toolset for scalable performance analysis of large-scale applications
usage: scalasca [-v][-n][c] {action}
  1. prepare application objects and executable for measurement:
     scalasca -instrument <compile-or-link-command> # skin (using scorep)
  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
```

- The `scalasca -instrument` or `skin` command is deprecated and only provided for backwards compatibility with Scalasca 1.x.

- Recommended: use Score-P instrumenter directly
Scalasca: *skin* (compatibility command)

- Scalasca application instrumenter

```
% skin
Scalasca 2.3.1: application instrumenter using scorep
    -comp={all|none|...}: routines to be instrumented by compiler
        (... custom instrumentation specification for compiler)
    -pdt: process source files with PDT instrumenter
    -pomp: process source files for POMP directives
    -user: enable EPIK user instrumentation API macros in source code
    -v: enable verbose commentary when instrumenting
    --*: options to pass to Score-P instrumenter
```

- Provides compatibility with Scalasca 1.x
- **Deprecated**: use Score-P instrumenter directly
Scalasca: \textit{scan}

\begin{itemize}
\item Scalasca measurement collection & analysis nexus
\end{itemize}

\begin{verbatim}
\% scan
Scalasca 2.3.1: measurement collection & analysis nexus
usage: scan {options} [launchcmd [launchargs]] target [targetargs]
 where {options} may include:
 -h    Help: show this brief usage message and exit.
 -v    Verbose: increase verbosity.
 -n    Preview: show command(s) to be launched but don't execute.
 -q    Quiescent: execution with neither summarization nor tracing.
 -s    Summary: enable runtime summarization. [Default]
 -t    Tracing: enable trace collection and analysis.
 -a    Analyze: skip measurement to (re-)analyze an existing trace.
 -e    exptdir : Experiment archive to generate and/or analyze.
              (overrides default experiment archive title)
 -f    filtfile : File specifying measurement filter.
 -l    lockfile : File that blocks start of measurement.
\end{verbatim}
Scalasca: *square*

- Scalasca analysis report explorer (cube interface)

```
% square
Scalasca 2.3.1: analysis report explorer
                | cube file>
-F           : Force remapping of already existing reports
-f filtfile  : Use specified filter file when doing scoring
-s           : Skip display and output textual score report
-v           : Enable verbose mode
```
Setup environment

- Load module
  ```bash
  % module load png qt cube scorep
  % module use -a /lrz/sys/share/modules/extfiles/
  % module load scalasca/2.3
  ```

- Change to directory containing NPB3.3-MZ-MPI sources

- Existing instrumented executable in bin.scorep/directory can be reused
Exercise:
Summary measurement collection...

- Change to directory with executable and edit job script

```
% cd bin.scorep
% cp ../jobscript/CoolMUC-3/scalasca.sbatch .
% vi scalasca.sbatch

[...]

export OMP_NUM_THREADS=4
# Scalasca2 configuration
export SCOREP_FILTERING_FILE=../config/scorep.filt
export SCOREP_TOTAL_MEMORY=140M

scalasca -analyze -s mpiexec -n 4 ./bt-mz_C.4
-----alternatively-----
scan mpiexec -n 4 ./bt-mz_C.4
```

- Submit the job

```
% sbatch scalasca.sbatch
```
Exercise: Summary measurement

- Run the application using the Scalasca measurement collection & analysis nexus prefixed to launch command

```
% OMP_NUM_THREADS=4
% scan -f ../config/scorep.filt mpiexec -n 4 ./bt-mz_C.4
```

S=C=A=N: Scalasca 2.2 runtime summarization
S=C=A=N: ./scorep_bt-mz_C_4x4_sum experiment archive
S=C=A=N: Wed Mar 11 16:31:26 2015: Collect start
/lrz/sys/intel/impi/5.0.1.035/bin64/mpiexec -n 4 ./bt-mz_C.4

NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark

Number of zones: 16 x 16
Iterations: 200 dt: 0.000100
Number of active processes: 4
Use the default load factors with threads
Total number of threads: 16 ( 4.0 threads/process)
Calculated speedup = 16.00

[... More application output ...]

S=C=A=N: Wed Mar 11 16:33:44 2015: Collect done (status=0) 50s
S=C=A=N: ./scorep_bt-mz_C_4x4_sum complete.

- Creates experiment directory ./scorep_bt-mz_C_4x4_sum
The post-processing derives additional metrics and generates a structured metric hierarchy.
Split base metrics into more specific metrics
Event trace collection
Event trace examination & analysis
● Change to directory with executable and edit job script

```bash
% cd bin.scorep
% cp ../jobscript/CoolMUC-3/scalasca.sbatch .
% vi scalasca.sbatch

[...]

export OMP_NUM_THREADS=4
export SCOREP_FILTERING_FILE=../config/scorep.filt
export SCOREP_TOTAL_MEMORY=140M

scalasca -analyze -t mpiexec -n 4 ./bt-mz_C.4
```

● Submit the job

```bash
% sbatch scalasca.sbatch
```
Trace analysis

- **Automatic trace analysis of existing instrumented binary**

  Analyzing experiment archive ./scorep_bt-mz_C_4x4_trace/traces.otf2

  Opening experiment archive ... done (0.001s).
  Reading definition data ... done (0.003s).
  Reading event trace data ... done (4.652s).
  Preprocessing ... done (2.918s).
  Timestamp correction ... done (1.408s).
  Analyzing trace data ...
    Wait-state detection (fwd) (1/5) ... done (4.205s).
    Wait-state detection (bwd) (2/5) ... done (0.295s).
    Synchpoint exchange (3/5) ... done (1.028s).
    Critical-path & delay analysis (4/5) ... done (4.304s).
    Propagating wait-state exchange (5/5) ... done (0.274s).
   done (10.361s).
  Writing analysis report ... done (0.439s).

  Total processing time : 21.213s
  S=C=A=N: Wed Mar 11 17:59:01 2015: Analyze done (status=0) 21s
  Warning: 366.145MB of analyzed trace data retained in
  ./scorep_bt-mz_C_4x4_trace/traces!
  S=C=A=N: ./scorep_bt-mz_C_4x4_trace_complete.
Trace analysis report exploration

- Produces trace analysis report in experiment directory containing trace-based wait-state metrics

% square scorep_bt-mz_C_4x4_trace
INFO: Post-processing runtime summarization result...
INFO: Post-processing trace analysis report...
INFO: Displaying ./scorep_bt-mz_C_4x4_trace/trace.cubex...

[GUI showing trace analysis report]
Post-processed trace analysis report

Additional trace-based metrics in metric hierarchy
Access online metric description via context menu
Online metric description

**Late Sender Time**

**Description:**
Refers to the time lost waiting caused by a blocking receive operation (e.g., MPI_Recv or MPI_Wait) that is posted earlier than the corresponding send operation.

If the receiving process is waiting for multiple messages to arrive (e.g., in an call to MPI_WaitAll), the maximum waiting time is accounted, i.e., the waiting time due to the latest sender.

**Unit:**
Seconds

**Diagnosis:**
Try to replace MPI_Recv with a non-blocking receive MPI_Irecv that can be posted earlier, proceed concurrently with computation, and complete with a wait operation after the message is expected to have been sent. Try to post sends earlier, such that they are available when receivers need them. Note that outstanding messages (i.e., sent before the receiver is ready) will occupy internal message buffers, and that large numbers of posted receive buffers will also introduce message management overhead, therefore moderation is advisable.

**Parent:**
MPI_Point-to-point Communication Time

**Children:**
Critical-path analysis

Critical-path profile shows wall-clock time impact
Critical-path analysis

Critical-path imbalance highlights inefficient parallelism
Pattern instance statistics

Click to get statistics details

Access pattern instance statistics via context menu
Scalable performance analysis of large-scale parallel applications

- toolset for scalable performance measurement & analysis of MPI, OpenMP & hybrid parallel applications
- supporting most popular HPC computer systems
- available under New BSD open-source license
- sources, documentation & publications:
  - http://www.scalasca.org
  - mailto: scalasca@fz-juelich.de
Acknowledgements

- Course material courtesy
  - Ilya Zukhov, JSC/FZ-Juelich
  - Anupam Karmakar, ex LRZ APP Team
  - VI-HPS: www.vi-hps.org
  - Score-P Team
  - Scalasca and Cube Team