Performance analysis on Blue Gene/Q
with
scalasca
+ other tools and debugging

Michael Knobloch
Jülich Supercomputing Centre
scalasca@fz-juelich.de

Based on slides by Brian Wylie and Markus Geimer

July 2012
Performance analysis, tools & techniques

- Profile analysis
  - Summary of aggregated metrics
    - per function/callpath and/or per process/thread
  - Most tools (can) generate and/or present such profiles
    - but they do so in very different ways, often from event traces!
    - e.g., gprof, mpiP, ompP, Scalasca, TAU, Vampir, ...
- Time-line analysis
  - Visual representation of the space/time sequence of events
  - Requires an execution trace
  - e.g., Vampir, Paraver, JumpShot, Intel TAC, Sun Studio, ...
- Pattern analysis
  - Search for event sequences characteristic of inefficiencies
  - Can be done manually, e.g., via visual time-line analysis
  - or automatically, e.g., KOJAK, Scalasca, Periscope, ...
Automatic trace analysis

• Idea
  - Automatic search for patterns of inefficient behaviour
  - Classification of behaviour & quantification of significance

- Guaranteed to cover the entire event trace
- Quicker than manual/visual trace analysis
- Parallel replay analysis exploits memory & processors to deliver scalability
The Scalasca project

● Overview
  ■ Helmholtz Initiative & Networking Fund project started in 2006
  ■ Headed by Bernd Mohr (JSC) & Felix Wolf (GRS)
  ■ Follow-up to pioneering KOJAK project (started 1998)
    ▶ Automatic pattern-based trace analysis

● Objective
  ■ Development of a **scalable** performance analysis toolset
  ■ Specifically targeting **large-scale** parallel applications
    ▶ such as those running on Blue Gene/Q or Cray XT/XE/XK with 10,000s to 100,000s of processes

● Latest release February 2012: Scalasca v1.4.1
  ■ Download from [www.scalasca.org](http://www.scalasca.org)
  ■ Available on POINT/VI-HPS Parallel Productivity Tools DVD
Scalasca features

- Open source, New BSD license
- Portable
  - Cray XT, IBM Blue Gene, IBM SP & blade clusters, NEC SX, SGI Altix, SiCortex, Solaris & Linux clusters, ...
- Supports parallel programming paradigms & languages
  - MPI, OpenMP & hybrid OpenMP+MPI
  - Fortran, C, C++
- Integrated instrumentation, measurement & analysis toolset
  - Automatic and/or manual customizable instrumentation
  - Runtime summarization (aka profiling)
  - Automatic event trace analysis
  - Analysis report exploration & manipulation
**Scalasca support & limitations**

- MPI 2.2 apart from dynamic process creation
  - C++ interface deprecated with MPI 2.2
- OpenMP 2.5 apart from nested thread teams
  - partial support for dynamically-sized/conditional thread teams*
  - no support for OpenMP used in macros or included files
- Hybrid OpenMP+MPI
  - partial support for non-uniform thread teams*
  - no support for MPI_THREAD_MULTIPLE
  - no trace analysis support for MPI_THREAD_SERIALIZED (only MPI_THREAD_FUNNELED)

* Summary & trace measurements are possible, and traces may be analyzed with Vampir or other trace visualizers
  - automatic trace analysis currently not supported
• Application code compiled & linked into executable using MPICC/CXX/FC*
• Launched with MPIEXEC*
• Application processes interact via MPI library

*Juqueen setup covered later
Application instrumentation

- Automatic/manual code instrumenter
- Program sources processed to add instrumentation and measurement library into application executable
- Exploits MPI standard profiling interface (PMPI) to acquire MPI events
- Measurement library manages threads & events produced by instrumentation
- Measurements summarized by thread & call-path during execution
- Analysis report unified & collated at finalization
- Presentation of summary analysis
- During measurement, time-stamped events are buffered for each thread.
- Flushed to files along with unified definitions & maps at finalization.
- Follow-up analysis replays events and produces extended analysis report.
- Presentation of analysis report.
Generic parallel tools architecture

- Automatic/manual code instrumenter
- Measurement library for runtime summary & event tracing
- Parallel (and/or serial) event trace analysis when desired
- Analysis report examiner for interactive exploration of measured execution performance properties
Scalasca toolset components

- Scalasca instrumenter = SKIN
- Scalasca measurement collector & analyzer = SCAN
- Scalasca analysis report examiner = SQUARE
One command for everything

```bash
% scalasca
```

Scalasca 1.4
Toolset for scalable performance analysis of large-scale apps
usage: scalasca [-v][-n] {action}

1. prepare application objects and executable for measurement:
   ```bash
   scalasca -instrument <compile-or-link-command>  # skin
   ```

2. run application under control of measurement system:
   ```bash
   scalasca -analyze <application-launch-command>  # scan
   ```

3. post-process & explore measurement analysis report:
   ```bash
   scalasca -examine <experiment-archive|report>  # square
   ```

[-h] show quick reference guide (only)
• Measurement & analysis runtime system
  ■ Manages runtime configuration and parallel execution
  ■ Configuration specified via EPIK.CONF file or environment
    ▶ epik_conf reports current measurement configuration
  ■ Creates experiment archive (directory): **epik_<title>**
  ■ Optional runtime summarization report
  ■ Optional event trace generation (for later analysis)
  ■ Optional filtering of (compiler instrumentation) events
  ■ Optional incorporation of HWC measurements with events
    ▶ via PAPI library, using PAPI preset or native counter names

• Experiment archive directory
  ■ Contains (single) measurement & associated files (e.g., logs)
  ■ Contains (subsequent) analysis reports
• Automatic instrumentation of OpenMP & POMP directives via source pre-processor
  ■ Parallel regions, worksharing, synchronization
  ■ OpenMP 2.5 with OpenMP 3.0 coming
    ▶ No special handling of guards, dynamic or nested thread teams
    ▶ OpenMP 3.0 ORDERED sequentialization support
    ▶ Support for OpenMP 3.0 tasks currently in development
  ■ Configurable to disable instrumentation of locks, etc.
  ■ Typically invoked internally by instrumentation tools
• Used by Scalasca/Kojak, ompP, Periscope, Score-P, TAU, VampirTrace, etc.
  ■ Provided with Scalasca, but also available separately
    ▶ OPARI 1.1 (October 2001)
    ▶ OPARI2 1.0 (January 2012)
- Parallel program analysis report exploration tools
  - Libraries for XML report reading & writing
  - Algebra utilities for report processing
  - GUI for interactive analysis exploration
    - requires Qt4 library
    - can be installed independently of Scalasca instrumenter and measurement collector/analyzer, e.g., on laptop or desktop
- Used by Scalasca/KOJAK, Marmot, ompP, PerfSuite, Score-P, etc.
  - Analysis reports can also be viewed/stored/analyzed with TAU Paraprof & PerfExplorer
  - Provided with Scalasca, but also available separately
    - CUBE 3.4.1 (January 2012)
    - CUBE 4.0 (December 2011)
Analysis presentation and exploration

- Representation of values (severity matrix) on three hierarchical axes
  - Performance property (metric)
  - Call-tree path (program location)
  - System location (process/thread)

- Three coupled tree browsers

- CUBE displays severities
  - As value: for precise comparison
  - As colour: for easy identification of hotspots
  - Inclusive value when closed & exclusive value when expanded
  - Customizable via display mode
Scalasca analysis report explorer (summary)

What kind of performance problem?

Where is it in the source code?
In what context?

How is it distributed across the processes?
Additional metrics determined from trace.
Scalasca 1.4 functionality

- Automatic function instrumentation (and filtering)
  - CCE, GCC, IBM, Intel, PathScale & PGI compilers
  - optional PDToolkit selective instrumentation (when available) and manual instrumentation macros/pragmas/directives

- MPI measurement & analyses
  - scalable runtime summarization & event tracing
  - only requires application executable re-linking
  - P2P, collective, RMA & File I/O operation analyses

- OpenMP measurement & analysis
  - requires (automatic) application source instrumentation
  - thread management, synchronization & idleness analyses

- Hybrid OpenMP/MPI measurement & analysis
  - combined requirements/capabilities
  - parallel trace analysis requires uniform thread teams
Scalasca 1.4 added functionality

- Improved configure/installation
- Improved parallel & distributed source instrumentation
  - OpenMP/POMP source instrumentation with OPARI2
- Improved MPI communicator management
- Additional summary metrics
  - MPI-2 File bytes transferred (read/written)
  - OpenMP-3 ORDERED sequentialization time
- Improved OpenMP & OpenMP+MPI tracefile management via SIONlib parallel I/O library
- Trace analysis reports of severest pattern instances
  - linkage to external trace visualizers Vampir & Paraver
- New boxplot and topology presentations of distributions
- Improved documentation of analysis reports
Scalasca case study: BT-MZ

- NPB benchmark code from NASA NAS
  - block triangular solver for unsteady, compressible Navier-Stokes equations discretized in three spatial dimensions
  - performs ADI for several hundred time-steps on a regular 3D grid and verifies solution error within acceptable limit
- Hybrid MPI+OpenMP parallel version (NPB3.3-MZ-MPI)
  - ~7,000 lines of code (20 source modules), mostly Fortran77
  - intra-zone computation with OpenMP, inter-zone with MPI
    - only master threads perform MPI (outside parallel regions)
  - very portable, and highly scalable
  - configurable for a range of benchmark classes and sizes
  - dynamic thread load balancing disabled to avoid oversubscription
- Run on juqueen BG/Q with up to 524,288 threads (8 racks)
  - Summary and trace analysis using Scalasca
  - Automatic instrumentation by compiler & OPARI source processor
NPB class E problem:
64 x 64 zones
4224 x 3456 x 92 grid

Best pure MPI execution time over 2000s

3.8x speed-up from 256 to 1024 compute nodes

- Best performance with 64 OpenMP threads per MPI process
- 55% performance bonus from exploiting all 4 threads per core
Good scaling starts to tail off with 8192 nodes (8192x64 threads)

Measurement overheads minimal until linear collation dominates

3.6x speed-up from 1024 to 4096 nodes

NPB class F problem:
128 x 128 zones
12032 x 8960 x 250 grid
Idle threads time

Half of CPU time attributed to idle threads (unused cores) outside of parallel regions
Idle threads time

... particularly during MPI communication performed only by master threads
MPI point-to-point communication time

- Explicit MPI time is less than 1%
- Only master threads perform communication but require widely varying time for receives
MPI point-to-point time

... with correspondence to MPI rank evident from folded BG/Q torus network topology
MPI point-to-point receive communications

... though primarily explained by the number of messages sent/received by each process
MPI point-to-point bytes received

... and variations in the amount of message data sent/received by each process.
Comparable computation times for three solve directions, with higher numbered threads slightly faster (load imbalance)
For \( z_{\text{solve}} \) imbalance is rather larger and shows more variation by process particularly in OpenMP parallel do loop.
... resulting in faster threads needing to wait in barrier at end of parallel region
OpenMP implicit barrier synchronization time (z)

... but with 8192 processes to examine scrolling displays showing only several hundred at a time is inconvenient.
... however, a boxplot scalably presents value range and variation statistics.
OpenMP implicit barrier synchronization time (x) for rapid comparison and quantification of metric variations due to imbalances.
Scalasca scalability issues/optimizations

- Runtime summary analysis of BT-MZ successful at largest scale:
  - 8192 MPI processes each with 64 OpenMP threads = ½ million
  - Only 3% measurement dilation versus uninstrumented execution
    - Latest XL compiler and OPARI instrumentation more efficient
      - Compilers can selectively instrument routines to avoid filtering

- Integrated analysis of MPI & OpenMP parallelization overheads:
  - Performance of both need to be understood in hybrid codes
  - MPI message statistics can explain variations in comm. times

- Time for measurement finalization grew linearly with num. processes:
  - Only 39 seconds for process and thread identifier unification
  - But 745 seconds to collate and write data for analysis report

- Analysis reports contain data for many more processes and threads than can be visualized (even on large-screen monitors):
  - Fold & slice high dimensionality process topology
  - Compact boxplot presents range and variation of values
• Scalasca available as UNITE package
  ■ Accessible via Modules
  ■ module load UNITE scalasca
  ■ Scalasca 1.4.2rc2 with improvements for BG/Q
  ■ Configured with PAPI & SIONlib
  ■ Comes with Cube 3.4
  ■ Works with LoadLeveler

• Separate Cube installation also available
  ■ Accessible via Modules
  ■ module load UNITE cube
• **Instrumentation**
  - automatic instrumentation with "skin mpixl{cc,cxx,f}[_r]"
  - compatibilities of different compilers/libraries unknown
    ▶ if in doubt, rebuild everything

• **Measurement collection & analysis**
  - runjob & qsub support likely to be incomplete
    ▶ quote ignorable options and try different variations of syntax
    ▶ can't use "scan qsub" with qsub script mode
      - use "scan runjob" within script instead
    ▶ in worst case, should be able to configure everything manually
  - node-level hardware counters replicated for every thread
  - scout.hyb generally coredumps after completing trace analysis
• Tracing experiments collect trace event data in trace files, which are automatically analysed with a parallel analyzer
  ▪ parallel trace analysis requires the same configuration of MPI processes and OpenMP threads as used during collection
  ▪ generally done automatically using the allocated partition
• By default, Scalasca uses separate trace files for each MPI process rank stored in the unique experiment archive
  ▪ for pure MPI, data written directly into archive files
    ▶ the number of separate trace files may become overwhelming
  ▪ for hybrid MPI+OpenMP, data written initially to files for each thread, merged into separate MPI rank files during experiment finalization, and then split again during trace analysis
    ▶ the number of intermediate files may be overwhelming
    ▶ merging and parallel read can be painfully slow
Scalasca can be configured to use the SIONlib I/O library:
- optimizes parallel file reading and writing
  - avoids explicit merging and splitting of trace data files
- can greatly reduce file creation cost for large numbers of files
- ELG_SION_FILES specifies the number of files to be created
  - default of 0 reverts to previous behaviour with non-SION files
- for pure MPI, try one SION file per (I/O) node
- for hybrid MPI+OpenMP, set ELG_SION_FILES equal to number of MPI processes
  - trace data for each OpenMP thread included in single SION file
  - not usable currently with more than 61 threads per SION file due to exhaustion of available file descriptors
• Everything should generally work as on other platforms (particularly BG/P), but runjob & Cobalt qsub are unusual

• scalasca -instrument
  ▪ **skin** mpixlf77 -O3 -c bt.f
  ▪ **skin** mpixlf77 -O3 -o bt.1024 *.o

• scalasca -analyze
  ▪ **scan -s** mpirun -np 1024 -mode SMP -exe ./bt.1024
    ▷ epik_bt_smp1024_sum
  ▪ **scan -s** runjob --np 1024 --ranks-per-node 16 : ./bt.1024
    ▷ epik_bt_16p1024_sum
  ▪ **scan -s** qsub -n 16 --mode c16 ./bt.1024
    ▷ epik_bt_16p1024_sum (after submitted job actually starts)

• scalasca -examine
  ▪ **square** epik_bt_16p1024_sum
Everything should generally work as on other platforms (particularly BG/P), but runjob & Cobalt qsub are unusual

scalasca -instrument
- **skin** `mpixlf77_r -qsmp=omp -O3 -c bt-mz.f`
- **skin** `mpixlf77_r -qsmp=omp -O3 -o bt-mz.256 *.o`

scalasca -analyze
- **scan** `-s mpirun -np 256 -mode SMP -exe ./bt-mz.256`
  - **env** `OMP_NUM_THREADS=4`
  - epik_bt-mz_smp256x4_sum

- **scan** `-s runjob --np 256 --ranks-per-node 16`
  - **--envs** `OMP_NUM_THREADS=4` : ./bt-mz.256
  - epik_bt-mz_16p256x4_sum

- **scan** `-s qsub -n 16 --mode c16`
  - **-env** `OMP_NUM_THREADS=4` : ./bt-mz.256
  - epik_bt-mz_16p256x4_sum (after submitted job actually starts)
Scalasca experiment archive directories uniquely store measurement collection and analysis artefacts
- experiment title prefixed with `epik_`

Default EPIK experiment title composed from
- executable basename (without suffix): `bt-mz`
- ranks-per-node: `16p`
- number of MPI ranks: `256`
- number of OMP threads: `x4`
- type of experiment: `sum` or `trace`
- (+ HWC metric-list specification)

Can alternatively be specified with `-e` command-line option or `EPK_TITLE` environment variable
- Scalasca experiments can include hardware counters
  - specify lists of PAPI presets or native counters
  - via `-m` option or `EPK_METRICS` environment variable
    - `EPK_METRICS=PAPI_FP_OPS:PEVT_IU_IS1_STALL_CYC`
  - alternatively create a file defining groups of counters, specify this file with `EPK_METRICS_SPEC` and use the group name
- Available hardware counters (and PAPI presets) and supported combinations are platform-specific
- Shared counters are read and stored for each thread
- Although counters are stored in Scalasca traces, they are (currently) ignored by the parallel trace analyzers
  - storage for counters is not included in `max_tbc` estimates
  - summary+trace experiments produce combined analysis reports including measured hardware counter metrics
Plans for future improvements on BG/Q

- Support for 5D torus network
  - Similar to new MPI cartesian topologies

- Support for transactional memory and speculative execution
  - Time spend in this constructs
  - # of transactions and replays

- Node-level metrics
  - Integration of shared counters into Scalasca
  - e.g. counters for L2 cache or network unit
Other tools available on Juqueen - TAU

- TAU – Tuning and Analysis Utilities (University of Oregon)
  - Portable profiling and tracing toolkit
  - Automatic, manual or dynamic instrumentation
  - Includes ParaProf visualization tool

www.tau.uoregon.edu
• HPCToolkit (Rice University)

compile & link (full optimization)

app. source → optimized binary

profile execution [hpcrun/hpclink]

binary analysis [hpcstruct]

program structure → call path profile

presentation [hpcviewer]

database → analysis & attribution to source [hpcprof/mpi]

www.hpctoolkit.org
Other tools on Juqueen

- Other tools will be installed upon availability
  - VampirTrace / Vampir (TU Dresden / ZIH)
  - ExtraE / Paraver (BSC)
  - IBM High Performance Toolkit HPCT

- More tools under investigation
  - Vprof
  - MPITrace
  - Probably installed soon – check modules and Juqueen documentation
Debugging on Blue Gene/Q

Doesn't work
(yet)

But: TotalView for BG/Q announced and will be installed on Juqueen (once available)
What is TotalView?

- Supports many platforms
- Handles concurrency
  - Multi-threaded Debugging
  - Multi-process Debugging
- Integrated Memory Debugging
- Reverse Debugging available
- Supports Multiple Usage Models
  - GUI and CLI
  - Long Distance Remote Debugging
  - Unattended Batch Debugging
TotalView on Blue Gene

- Blue Gene support since Blue Gene/L
- Installed on the 212k core BG/L @ LLNL
  - Used to debug jobs with up to 32k cores
  - Heap memory debugging support added

- TotalView for BG/P installed on many systems in Germany, France, UK, and the US
- Support for shared libraries, threads, and OpenMP
- Used in several workshops
  - JSC's Blue Gene/P Porting, Tuning, and Scaling Workshops
Development started June 2011
Basic debugging operations in October
Used in Synthetic Workload Testing in December
Fully functional in March 2012

Installed on Blue Gene/Q systems
- Lawrence Livermore National Lab
- Argonne National Lab
- Some IBM internal systems
• BG/Q TotalView is as functional as BG/P TotalView
  ■ MPI, OpenMP, pthreads, hybrid MPI+threads
  ■ C, C++, Fortran, assembler; IBM and GNU compilers
  ■ Basics: source code, variables, breakpoints, watchpoints, stacks, single stepping, read/write memory/registers, conditional breakpoints, etc.
  ■ Memory debugging, message queues, binary core files, etc.
• PLUS, features unique to BG/Q TotalView
  ■ QPX (floating point) instruction set and register model
  ■ Fast compiled conditional breakpoints and watchpoints
  ■ Asynchronous thread control
• Working on debugging interfaces for TM/SE regions
Questions?

If not now:

scalasca@fz-juelich.de