Scalasca collaborations

2016-02-23 | Brian J. N. Wylie
Jülich Supercomputing Centre

b.wylie @ fz-juelich.de
Overview

Scalasca toolset

- collaborative development

Use of Scalasca and associated tools on K computer

- ABysS execution analysis & tuning
- NEST execution analysis

VI-HPS Tuning Workshops

- collaborative training
Scalasca

Toolset for scalable performance analysis of large-scale parallel applications

- capable of runtime summary profile and parallel event trace analyses
- for MPI, OpenMP, Pthreads and mixed MPI+threading
  - successfully used with 1.75M threads and 1.25M processes
- supporting most popular HPC computer systems
  - Blue Gene/Q, Cray, Stampede, Tianhe, K computer, etc.
- with CUBE analysis report explorer and utilities & OPARI source instrumenter components
- using libraries for PAPI hardware counters, SIONlib scalable parallel file I/O, etc.
- available under New BSD open-source license
- http://www.scalasca.org (mailto: scalasca@fz-juelich.de)
Parallel execution inefficiency patterns

- **Late Sender**
- **Late Sender, Wrong Order**
- **Wait at NxN**

**Critical path**
Automated execution trace analysis

- automatic search for patterns of inefficient execution behaviour
- classification of behaviour and quantification of significance

- quicker than manual / visual inspection, guaranteed to cover entire event trace
- parallel replay analysis exploits available memory & processors to deliver scalability
Scalasca development

Started 2006 as scalable successor of pioneering KOJAK project (started 1998)
- initial collaboration of JSC with University of Tennessee – Knoxville
- now jointly developed with Technische Universität Darmstadt
- support for K computer and Fujitsu FX10/100 contributed by AICS

Scalasca2 using community-developed Score-P instrumentation & measurement infrastructure
- partnership with GNS, UOregen, RWTH Aachen, TUDarmstadt, TUDresden, TUMunchen
- on-going support and contributions from Fujitsu Ltd
Score-P architecture overview

**Application**

- **Vampir**
- **Scalasca**
- **CUBE**
- **TAU**
- **TAUdb**
- **Periscope**

**Score-P measurement infrastructure**

- **Event traces (OTF2)**
- **Call-path profiles (CUBE4, TAU)**
- **Hardware counter (PAPI, rusage, [PERF])**
- **Online interface**

**Instrumentation wrapper**

- **Process-level parallelism (MPI, SHMEM)**
- **Thread-level parallelism (OpenMP, Pthreads, [OmpSs])**
- **Accelerator-based parallelism (CUDA, OpenCL, [OpenACC])**
- **Source code instrumentation (OPAR1Z, PDT)**
- **User instrumentation**
Scalasca workflow

Measurement library

Instr. target application

Score-P

Instr. executable

Instrumented executable

Instrumenter compiler / linker

Source modules

Optimized measurement configuration

Local event traces

Parallel wait-state search

Walt-state report

Summary report

Report manipulation

Which problem?

Where in the program?

Which process?

CUBE
Initial Scalasca performance analyses (recommended strategy)

Rebuild with automatic instrumentation: MPI library + compiler-instrumented routines

Initial runtime summary measurement scored to determine measurement filter
  - identifies frequently executed short routines which may need to be eliminated
  - to reduce memory requirements (mainly for event trace buffers, but also callpath profiles)
  - and to reduce execution dilation of measured routines and entire application

New summary and trace measurements taken using optimised measurement filter

Event traces automatically analysed to identify and quantify MPI wait-state inefficiencies
  - producing callpath profiles augmented with additional inefficiency metrics, and identifying severest instances

Event traces visualised with Vampir to examine execution time-lines
**ABySS genome sequence assembler**

*de novo*, parallel, paired-end genome sequence assembler

- developed by Canada's Michael Smith Genome Sciences Centre

**Sequence assembly algorithm:**

- load short read sequences, breaking each into \( k \)-mers
- find adjacent (overlapping) \( k \)-mers
- remove \( k \)-mers resulting from read errors
- remove variant sequences (bubbles)
- generate contiguous sequences (contigs)

**ABYSS-P de Bruijn graph assembler implemented in C++ using MPI**

- master process directs remaining worker processes

* all possible (ACTG-) base subsequences of length \( k \) from a read obtained through DNA sequencing
ABYSS-P scalability on K computer

Evaluation with *E. coli* K12 MG1655 read sequence dataset (6.1 GB)

Execution with single MPI rank/node
- 14 GB usable node memory

Linear growth of memory for MPI
- exhaustion for 16,384 ranks
- message buffers required for eager mode receipt

Min. execution time for ~1024 nodes
- rapid deterioration thereafter
Scalasca trace analysis (1024 MPI processes)

Callpath profile augmented with additional metrics quantifying wait states calculated in parallel replay analysis of event trace

Presentation by CUBE analysis report explorer

Tree node severity values: closed = inclusive, open = exclusive; coloured according to scale below pane

Hierarchies of metrics

Tree of execution callpaths (values for selected metric)

Distribution of MPI rank severity values (rank 0 top-left)
Scalasca trace analysis identifying severest instance of Wait at NxN in MPI_Allreduce

Lots of point-to-point Send and Irecv communication, with negligible waiting time

Comparatively large waiting time in MPI collectives (Allreduce, Barrier, etc.)

Severest waiting instances averaging 1.3 seconds and with maximum durations of over 2 seconds!

- generally indicative of load-imbalance in preceding computation, however, distribution pattern seems uniform across processes
Vampir visualisation of same event trace

Graphical (flat) function profile for current time interval

Horizontal time-line of coloured execution states for each MPI process below aggregate state chart provided for navigation when zooming

Distinguished phases of LOADING, ERODE/TRIM, ASSEMBLE with “waves” of communication evident within LOADING stage
Vampir display zoomed to time interval of ERODE stage

Sequence of MPI_Allreduce calls within ERODE stage many of the initial ones with huge waiting times (over 2 seconds duration for severest instance on rank 271) resulting from load-imbalance in preceding computation – alternating distribution of severities
Assessment of initial ABYSS-P performance analyses with Scalasca

Positive insights

- Complex Master-Worker execution with asynchronous message/work queues
- Complex usage of MPI P2P and collective communication that varies by execution stage
- Increasingly costly communication and load-balancing inefficiencies at larger scale
- Extreme run-to-run execution time variations (likely mostly due to file I/O)

Negative insights

- Compiler-instrumentation prohibitively intrusive (even with extensive filtering)
- Difficult to determine effective filter, due to complex (dual-role) code structure
- Difficult to correlate Master directions with Worker activities
- Difficult to distinguish important execution stages
- Difficult to isolate origin of imbalances, e.g., file I/O
Revised performance analysis strategy

MPI-only measurements (without compiler instrumentation) don't suffer intrusion

- however, lack of context makes analyses even more difficult to interpret, and traces are still huge

Therefore, manually annotate ABYSS-P sources with user-instrumentation macros

- to distinguish execution stages and other activities of interest (such as file I/O)

Take execution measurements combining MPI + user instrumentation

- enable trace collection only for selected execution stage(s) to reduce size

Compare alternative ABYSS-P implementations and execution configurations

- quantify how performance of stages varies with scale
DISCOVER / POP BUBBLES stage

Each rank in turn executes POPBUBBLE while all others wait, where in addition to local computation POPBUBBLE creates, writes & flushes a separate file before notifying all of its peers.
ABYSS-P tuning for K computer

Use separate rank-local directories for files

- avoids highly variable file-system contention for files in common directory

Open (create) files in parallel, in advance of writes

- removed from serialisation within POP_BUBBLE stage
- reduces disruption of computation within ASSEMBLE stage

Employ collective communication instead of eager point-to-point communication

- avoids linearly growing MPI memory requirements, and bottleneck for Master rank 0
- exploits MPI_Reduce and MPI_Alltoall optimised for K computer
ABYSS-P profile comparison:

8192 MPI processes on K computer

Original version 941 seconds
- 69% (651s) in coupled POP/DISCOVER_BUBBLES
- 14% (129s) in ASSEMBLE
- 4.2% (45s) in LOADING

Improved version 293 seconds
- 4.4% (13s) in coupled POP/DISCOVER_BUBBLES
- 30% (84s) in ASSEMBLE
- 12% (33s) in LOADING
ABYSS-P scalability on K computer (comparison)

Improved execution time at scale
- ASSEMBLE now dominates
- DISCOVER/POP BUBBLES 50 times faster for 8,192
- LOADING scales well and reduced to a few seconds

Much less memory required for MPI
- only 0.5 GB for 82,944 ranks

Rework of Master-Worker coordination communication needed for remaining ABYSS-P stages
ABySS execution analysis & tuning review

ABYSS-P executions on the K computer suffered critical performance and scaling issues

- slowdown with more than 1024 compute nodes, excessive MPI memory requirements

Analysis using Scalasca/Score-P/Vampir helped identify & quantify execution inefficiencies

- file I/O variability, serial file creation, rank 0 coordination bottleneck, ...
- automatic instrumentation provided a convenient starting point, but needed to be complemented with manual instrumentation

  - Master/Worker execution stages and file I/O

ABYSS-P remediation

- rank-local directories, reorganised file handling, exploiting efficient MPI collectives
- execution (and Scalasca measurement) now possible on full 82,944 compute nodes

  - only initial execution stages completed
NEST neural simulation tool

Neuronal spiking network simulation tool [www.nest-simulator.org]

- C++ implementation using MPI and OpenMP parallelisation
- neurons distributed to MPI processes
- synapses and associated connection information stored on post-synaptic process
- exchange during simulation update based on MPI_Allgather(v) by master threads

Experiments

- empty-gap random connectivity simulation on K computer
  - gather_events performance governed by MPI&OMP computational load imbalances
- HDF5 import module for large-scale data-driven simulation on JUQUEEN BG/Q
  - 1.9 TB connectivity map from high-resolution biological data
  - failure to use collective MPI File I/O significantly diminishes reading performance
NEST 82944x8 measurement on K computer: simulation gather_events

53s to gather spiking events from firing synapses
NEST 82944x8 measurement on K computer: simulation gather_events

Following OpenMP barrier shows imbalance after MPI collective communication
NEST 28672x16 measurement on JUQUEEN BG/Q: HDF5 import_synapses

Roughly 2 minutes to load and redistribute synapse data
NEST 28672x16 measurement on JUQUEEN BG/Q: HDF5 import_synapses

Collective redistribution is expensive, particularly due to preceding computational imbalance
NEST 28672x16 measurement on JUQUEEN BG/Q: HDF5 import_synapses

Collective HDF5 read found to be using inefficient individual MPI file reads due to data structure mismatch
Extreme scaling

7th Jülich Blue Gene Extreme Scaling Workshop (1-3 Feb 2016)

- latest of series starting with JUBL (BG/L), then JUGENE (BG/P), now JUQUEEN (BG/Q)
- full 28 racks dedicated over 50 hours for 8 international code teams

High Q Club [www.fz-juelich.de/ias/jsc/high-q-club]

- application codes with demonstrated scalability to use entire JUQUEEN resource
  - up to 1.75M MPI processes or OpenMP threads running on 458,752 cores
- currently 25 members, several more applications pending

Discussion and information exchange

- aXXLs@ISC-HPC15: Application Extreme-scaling Experience of Leading Supercomputing Centres
- MAXI@ParCo15: Multi-system Application Extreme-scaling Imperative
High-Q Club member characteristics

- MPI
- OpenMP
- pthreads
- Fortran
- C++
- C

**Hardware threads**

- threads per MPI rank

**MPI ranks per node**

2016-02-23 | 6th RIKEN AICS Int'l Symp. (Kobe, Japan)
Scalasca training offered through VI-HPS

Virtual Institute – High Productivity Supercomputing [www.vi-hps.org]

- focus on parallel performance, correctness & debugging tools
- several VI-HPS Tuning Workshops each year
  - 3-5 days for application developers to get introduced to tools and receive guidance and assistance applying tools to their own codes
  - RIKEN AICS hosting VI-HPS-TW20 (24-26 Feb 2016) for users of K computer and related Fujitsu FX10/100 systems
    - Score-P, Scalasca, TAU & BSC tools
  - additional workshops at LRZ in Garching, Germany & CINES in Montpellier, France
VI-HPS tools and their integration

- KCACHEGRIND
- LWM2 / MAP / MPIP / OSIS / MAQAO
- TAU
- SCORE-P
- PERISCOPE
- SCALASCA
- VAMPIR / PARAVER
- PTF / RUBIK / MAQAO

- PAPI
- MUST / ARCHER
- DDT
- STAT
- SYSMON / SPINDLE / SIONLIB / OPENMPI

- Hardware monitoring
- Automatic profile & trace analysis
- Visual trace analysis
- Execution
- Optimization

- Debugging, error & anomaly detection