Scientific Programming 16 (2008) 167–181
DOI 10.3233/SPR-2008-0255
IOS Press

Performance measurement and analysis of large-scale parallel applications on leadership computing systems

Brian J.N. Wylie [a,*], Markus Geimer [a] and Felix Wolf [a,b]
[a] *Jülich Supercomputing Centre, Forschungszentrum Jülich, Jülich, Germany*
[b] *Department of Computer Science, RWTH Aachen University, Aachen, Germany*

Abstract. Developers of applications with large-scale computing requirements are currently presented with a variety of high-performance systems optimised for message-passing, however, effectively exploiting the available computing resources remains a major challenge. In addition to fundamental application scalability characteristics, application and system peculiarities often only manifest at extreme scales, requiring highly scalable performance measurement and analysis tools that are convenient to incorporate in application development and tuning activities. We present our experiences with a multigrid solver benchmark and state-of-the-art real-world applications for numerical weather prediction and computational fluid dynamics, on three quite different multi-thousand-processor supercomputer systems – Cray XT3/4, *MareNostrum* & Blue Gene/L – using the newly-developed SCALASCA toolset to quantify and isolate a range of significant performance issues.

Keywords: Large-scale parallel applications and systems, performance measurement and analysis

1. Introduction

The modern landscape of high-performance computing (HPC) is potentially highly rewarding for those who wish to achieve previously unreachable goals, yet it is extremely challenging and requires perseverance and willingness to take on the unknown. Manufacturers of HPC computing systems offer powerful machines, in a diverse range of (often custom and idiosyncratic) configurations, yet that raw power must be effectively harnessed by applications to be productive. Standardisation of system components and programming models has made it relatively easy for applications which have already been parallelised with MPI message-passing to run on HPC computers, yet the achieved performance is often disappointing, especially when progressing to larger numbers of processors [17].

Performance analysis and tuning of HPC applications is a quest to reach new highs, taking on the challenges provided by new computer systems and previously unharnessed scales. When a sufficiently pleasing and satisfying plateau is reached, that journey ends naturally (or at least pauses). More often than not, however, an intermediate insurmountable barrier may result in exhaustion and frustration, where it must be carefully decided whether the potential benefits of continuing are worth the further (possibly unsuccessful) effort.

Teams of application developers and analysts working together are best equipped for undertaking these record-breaking tasks, but even as a team they still have limited capacity to learn and become adept with tools and utilities that can assist on the way. Success is more likely with a select set of flexible and multi-functional tools, that are reliable and straightforward to apply, which become indispensable aids to achieve mission goals.

This paper presents the evolution of the established KOJAK toolkit [25], which has been re-designed and re-engineered specifically to address scalability of performance measurement, analysis and investigation as SCALASCA [10]. It embodies relatively familiar techniques and technologies which have been specifically integrated and customised to handle complex HPC applications on the largest system configurations. After introducing the SCALASCA toolset and its usage

[*]Corresponding author: Brian J.N. Wylie, Jülich Supercomputing Centre, Forschungszentrum Jülich, D-52425 Jülich, Germany. Tel.: +49 2461 61 6589; Fax: +49 2461 61 6656; E-mail: b.wylie@fz-juelich.de.

model, three case studies are provided where an early version was successfully applied to important, highly-scalable HPC applications running on the most powerful computer systems, providing valuable insight into application and system scalability hindrances that only manifest at larger scale.

First subject is the SMG2000 benchmark on a combined Cray XT3/4 system which demonstrates the ability of SCALASCA to measure and analyse applications at extreme scales. This is followed by analysis of the WRF-NMM numerical weather prediction code running on the *MareNostrum* system (based on a large cluster of commodity IBM compute blades) that was experiencing intermittent performance anomalies. Finally, SCALASCA was used for the discovery (and remediation) of the primary scalability bottleneck in the XNS computational fluid dynamics solver on the highly-customised IBM Blue Gene/L system.

2. Analysis procedure

SCALASCA analysis of large-scale application executions uses a procedure of preparation followed by measurement and analysis steps [26], as depicted in Fig. 1. After an instrumented application executable has been prepared, running it with specific measurement configurations allows compact summaries or comprehensive event traces to be collected for analysis. The initial measurement configuration typically uses runtime summarisation of all execution events, and provides a complete callpath summary report at execution completion. Analysis of this profile can then be used to configure subsequent measurements such that base profiles are complemented with traces of selected events analysed to provide more comprehensive profiles and reports of instances of performance problems. Local trace data for each process rank is analysed in parallel, using a set of unified definitions and mappings from local-to-global definition identifiers produced at measurement finalisation. As additional insight becomes available, measurements can be reconfigured and repeated, or new application executables prepared with improved instrumentation or implementing improved algorithms as part of an iterative tuning procedure. These new measurement experiments can be compared with the already collected experiments to examine and quantify the impact of the modifications.

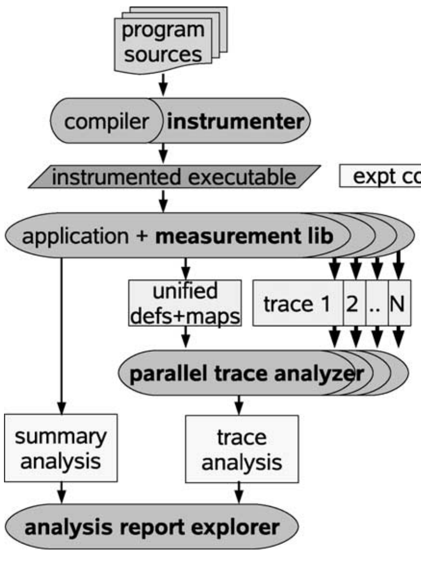


Fig. 1. SCALASCA toolset components for instrumentation of program sources, runtime parallel measurement summarisation and/or event tracing (specified via experiment configuration), parallel trace analysis, and analysis report exploration.

2.1. Application instrumentation

In common with most tools for MPI application measurement, the SCALASCA toolset uses the standard PMPI profiling interface to be able to interpose on MPI library function calls, and thereby examine call arguments and return values. Timers (and hardware counters) can also be accessed to provide event timestamps or durations (and counter values or elapsed counts) which can be recorded for later analysis or directly aggregated into summaries.

While such measurements of the MPI communication and synchronisation behaviour are a valuable part of parallel application execution analysis, especially if they include callpath information for the MPI calls, they offer limited insight into the equally important purely computational part of the execution. SCALASCA therefore can also exploit additional program instrumentation which complements the MPI events.

Program instrumentation is most easily achieved using compilation options to automatically instrument the enter and exits of functions. (Such options and the arguments provided with the instrumentation vary from compiler to compiler and often have little or no documentation, however, they are increasingly prevalent: e.g., `-finstrument-functions` of the GNU Compiler Collection.)

Instrumentation may also be directly added to source files using provided API calls or directives, which are particularly useful for marking application-specific phases of execution or major loops. (Such instrumentation is ignored when preprocessing or compiling unless it has been activated, so that it can conveniently be left as annotations in production builds of the program without instrumentation.)

Generating instrumented application program executables therefore typically involves simply using a SCALASCA instrumenter (that is prepended to compiler/linker commands in the application Makefile) to compile the desired source modules and link the executable. There is no need to change compiler or compilation options, including optimisations.

2.2. *Profile collection and analysis*

Initial measurements are often undertaken without detailed information regarding the frequency and quantity of execution events to be analysed, and which would directly impact the resources required for intermediate storage and processing. Knowledge and experience with an application, from studying its design or prior analyses, may provide some basis for selecting important events and configuring buffer sizes, however, this is often incomplete and unreliable, especially when running on new, larger systems: truisms from small scales can fail at larger scales.

A default measurement experiment using runtime summarisation is collected when the executable containing SCALASCA instrumentation is run, typically under the control of a collection and analysis nexus specified as a command prepended to the MPI application launch command-line, whether part of a batch script or interactive run invocation.

Runtime event summarisation provides a highly-scalable mechanism for executions of arbitrary length and complexity, and is therefore ideally suited for initial measurements and analyses. At execution completion, SCALASCA provides an integrated profile per process and callpath [28] that includes callpath visit count, elapsed time and hardware counter metrics: for MPI applications, these are augmented by counts for various types of message-passing operations (collective or point-to-point, purely synchronising or involving data communication) and corresponding numbers of bytes transfered. Process memory requirements scale modestly with the complexity of the application and are independent of the length of execution: SCALASCA uses local buffers to store definitions and callpaths for each process, which are unified into a set of global definitions at measurement completion with associated local-to-global identifier mappings used when collating measurement data into the final summary report. When it is desired to further reduce storage and processing requirements, unimportant and uninteresting callpaths can be truncated during measurement to eliminate fine detail without compromising the integrity of the complete callpaths.

Although the profile summary includes measurements of aggregate time in MPI functions, and the associated callpaths and process distributions, it does not include metrics which would quantify imbalances in waiting times entering collectives, late sends that resulted in longer than necessary waits for early receivers, etc. Calculation of these metrics requires comparison of message events between processes that would be prohibitively disruptive to acquire during application execution measurement, but that are suited to post-mortem analysis of event traces.

2.3. *Trace collection and analysis*

Tracing provides a potentially complete record of a parallel application's execution, and analysis of time-stamped events by different processes opens an additional dimension with insight into inter-process interaction. To reduce measurement impact, definitions and event records are buffered in each process' memory during measurement, then unified and analysed after measurement is complete. Process trace sizes are proportional to the number of recorded events, which is dependent on the frequency and duration of event recording. When the additional factor of the number of processes is included, aggregate trace sizes rapidly grow and require very careful handling: unnecessary copying and merging large numbers of huge trace files needs to be avoided.

SCALASCA uses separate buffers in each process to store local definitions and event records, allowing straightforward unification of definitions at measurement finalisation (as done when producing summaries). Each definition is given a numeric identifier which provides a concise means for refering to it in event records, and the local definitions and identifiers used by each process during measurement must be unified into a globally-consistent set to be able to relate events that occur on different processes. Mappings from local to global identifiers for each process are therefore also generated and stored in the experiment archive at finalisation, allowing the local identifiers in each event trace to be remapped on-the-fly when they are read. The traces can then be efficiently analysed in parallel, without having been merged or rewritten, using a scalable replay approach that exploits the computational processing power and memory of the system where the traces were collected, with the same number of processes [6]. Trace analysis ideally follows measurement collection using exactly the same processors and configuration (e.g., within a single batch job), and this is managed by the SCALASCA measurement collection and analysis nexus when tracing is requested.

Nevertheless, it is impractical to record complete traces of long complex executions in memory buffers, and unsynchronised intermediate flushes of trace buffers to files introduce perturbations that disrupt execution and compromise analyses. Appropriate event selection and buffer (size) configuration is therefore often necessary for practical trace collection within available process memory. Required trace buffer sizes can be determined by examining profile summary reports for the quantities of message-passing and region enter/exit events, and a report scoring utility is provided with the SCALASCA toolset which performs this calculation. Event handling may also be disabled for frequently-executed routines or callpaths with insignificant elapsed times (that would also be most prone to measurement overheads and introduce perturbation) or callpaths without message-passing events. Users can provide their own filter file listing routines to be excluded from measurements, and the summary report scoring utility is being extended to provide appropriate recommendations based on the frequency of occurrence and corresponding time accumulated by routines on callpaths that do not include communication or synchronisation. Ignoring such instrumented routines during measurement significantly reduces their processing overheads, but is not supported yet on all systems and even where it is the residual instrumentation overhead can still be prohibitive for certain routines [30]. In such cases, the routines (or source modules) to be instrumented (or left uninstrumented) must be specified during the initial application instrumentation stage.

2.4. *Integrated and comparative analyses*

Runtime summary and trace pattern analysis reports in a common callpath profile format can readily be compared and integrated into a holistic presentation of the application execution performance that can be interactively explored with the SCALASCA analysis report examiner GUI. While the summary can cover the entire execution measurement, tracing of selected intervals and events permits more in-depth analyses of key message-passing behaviour.

All measurements include inherent overheads which have an impact on the execution of the subject application. Buffers and data-structures used for measurement are an obvious overhead, which is particularly important on systems with limited amounts of compute node memory. The overhead of handling instrumented events, e.g., acquiring and processing timestamps, results in a small time dilation throughout the measurement which is difficult to isolate. More immediately observable are the major overheads of initialising the measurement infrastructure and experiment archive during `MPI_Init`, and the measurement finalisation during `MPI_Finalize`: on systems without globally synchronised clocks, clock synchronisation is included in both measurement initialisation and finalisation overheads. Although these latter overheads prolong the time for acquiring the measurement, they do not perturb the measurement of the parallel execution since they occur during global synchronisations.

Applications themselves often have substantial initialisation and finalisation phases, generally reading input files and writing output files, which can also vary significantly from run to run. The costs of these are typically amortised by extended periods in solvers or simulation kernels, on which performance analysis and tuning is focused. Selective instrumentation and measurement can reduce or eliminate the overhead of measurement (and analysis) for uninteresting phases, or complete analysis reports can be post-processed with a provided utility which extracts the sections of particular interest.

Measurements are controlled via configuration settings specified in files and environment variables read by each process when measurement is started, allowing a series of customised summary and trace measurements and analyses to be made with a single instrumented executable. Some of the most important configuration settings specify whether summarisation and/or tracing is to be activated, optional hardware counter metrics to be included, an optional filter file listing regions or callpaths to ignore, and buffer sizes. Callpaths are truncated by default at a depth of 32 frames (with no distinction made for recursive calls), however, this is also configurable.

Beyond the user convenience of preparing and using a single instrumented executable, along with straightforward exploitation of profile summaries from earlier measurements to effectively configure trace collection, integration of the two capabilities enables numerous synergies. Most significant is the opportunity to exploit these complementary techniques cooperatively. Execution information that is better suited to runtime summarisation, such as elapsed hardware counter metrics, can be omitted from traces while retained in the profile summary. Similarly, runtime callpath tracking and event selection can eliminate undesired non-message-passing events from traces while retaining the information necessary to perform full analysis of message transfers.

3. Experience

3.1. Supercomputer systems

Early versions of the SCALASCA toolset were installed and tested on a range of computer systems, from a variety of desktops and clusters to several of the most powerful supercomputers. Of particular interest have been IBM Blue Gene/L, IBM BladeCenter clusters and Cray XT3/4 systems with processor counts in the thousands. These systems have been specially designed and highly optimised for large-scale MPI applications, and although they utilise different processor and interconnect technologies, and different operating environments (including compilation systems, batch systems, filesystems, as well as the base operating systems themselves, though these are all based on Linux), it is often straightforward to re-compile and run portable MPI applications on any of them.

Measurements on the leadership computer systems discussed in the remainder of this section were made with the versions of operating systems, MPI libraries and compilers installed at the time, even though such systems are continually upgraded to address functional and performance deficiencies. Timings quoted are representative, however, all measurements and analyses were undertaken on shared systems, where considerable run-to-run variation in I/O and filesystem performance is unavoidable. Furthermore, the substantial time typically required to configure and release a large machine partition is not included.

Basic scalability testing was undertaken with the SMG2000 multigrid solver benchmark on each system, followed by investigation with locally important real-world applications for numerical weather prediction (WRF-NMM) and computational fluid dynamics in a blood pump (XNS-Debakey). After preparing instrumented application executables, measurement and analysis experiments were collected with different numbers of processes, with only some of the largest scale experiments summarised here. In each case, an initial summarisation experiment (or set of experiments) was examined and used to appropriately configure a subsequent trace collection and analysis experiment (using the same instrumented executable and modified measurement configuration).

3.2. SMG2000 on Cray XT3/4

Immediately after the integration of formerly separate Cray XT3 and XT4 systems at Oak Ridge National Laboratory in mid-2007, *Jaguar* [21] was ranked second in the 2007/06 Top500 list [20]. It was configured with 11,508 compute nodes, each with 2.6 GHz dual-core AMD Opteron processors and 4 GB of memory. Each node was connected to a Cray SeaStar router, with the SeaStars interconnected in a 3D-torus. It used UNICOS/lc 1.5.52 combining Linux running on the service nodes with a Catamount microkernel on the compute nodes and the Lustre parallel filesystem. Portland Group compilers generated code for the compute nodes. Notably, the compute nodes did not have globally synchronised clocks, therefore clock offset determination and correction are required before comparing timestamps from different nodes.

ASC benchmark SMG2000 [1] is a parallel semi-coarsening multigrid solver, which uses a complex communication pattern. The MPI version uses a three-dimensional decomposition and performs a lot of non-nearest-neighbour point-to-point communication operations (and only a negligible number of collective communication operations). Configuring with a fixed $64 \times 64 \times 32$ problem size per process and five solver iterations results in a fairly constant application run-time as additional CPUs were employed (i.e., *weak scaling*): with 22,528 processes arranged $32 \times 32 \times 22$ total run-time was 20 seconds, of which 7 seconds was the parallel solver part.

Callpath tracking determined that 3114 distinct callpaths were executed (with a maximum depth of 16 frames), of which 212 (7%) were in the solver phase. 94 of the total callpaths included (instrumented) MPI functions, of which 29 (31%) were in the solver phase. Producing the integrated profile at the end of measurement collection required 19 minutes (for the 22,538-process case): two-thirds of this time was required for unification of identifiers (proportional to the number of callpaths and number of processes)[1], with the remainder for gathering the measurements (from

[1] Subsequent algorithmic optimisations have reduced the time required for unification by more than a factor of 6.

each process for each callpath and metric) and writing the summary report. While summarisation time is proportional to the complexity of the (instrumented) application, it is independent of the length of measurement, and therefore scalable for long-running applications.

Since the initialisation and setup phases are uninteresting, but would skew the percentage calculations, the solver phase was extracted from the analysis report. This functionality is offered by one of several report algebra utilities [10] which read and write callpath profile reports applying a variety of algebraic operations to the analysis data. Figure 2(a) shows the profile summary extract presented by the SCALASCA analysis report examiner GUI, with the metric hierarchy in the left pane partially expanded, the callpath hierarchy in the middle pane expanded for the solver, and the MPI process topology distribution in the right pane. The considerable quantity and volume of point-to-point communication is already evident, and it accounts for 65% of the total solver execution time, as reported in the status bar below the metrics pane.

Within each tree, collapsed nodes present inclusive metrics while expanded nodes present exclusive metrics (i.e., excluding their child node values). Navigation to the most significant severities is aided by boxes colour-coded according to the range scale (at the bottom of the window) and the mode specified for each panel (from the menu above it). Absolute severity values can be shown as percentages of the metric tree root value or percentages of the currently selected metric value, while process severities can be shown in two additional modes corresponding to the percentage of the highest peer process value or in a peer distribution from lowest to highest process value. Depending on the nature of the analysis investigation, these different modes help to keep track of the (relative) significance of individual severity values, from overall context to variation amongst peers. Status fields at the bottom of each panel detail the current selection within that panel and the associated range value.

Investigating further required a trace to be collected, and because the number of events traced for each process increases with the total number of processes, the aggregate trace volume increases faster than linearly. From the callpath and message statistics in the profile summary, the buffering requirements for a complete trace of 22,538 processes was calculated to require 5TB with a maximum buffer size of 328 MB. The complete execution trace took 20 minutes to write (75% of which was identifier unification and writing the global definitions and mappings files), with the following trace analysis and report production requiring an additional 16 minutes.

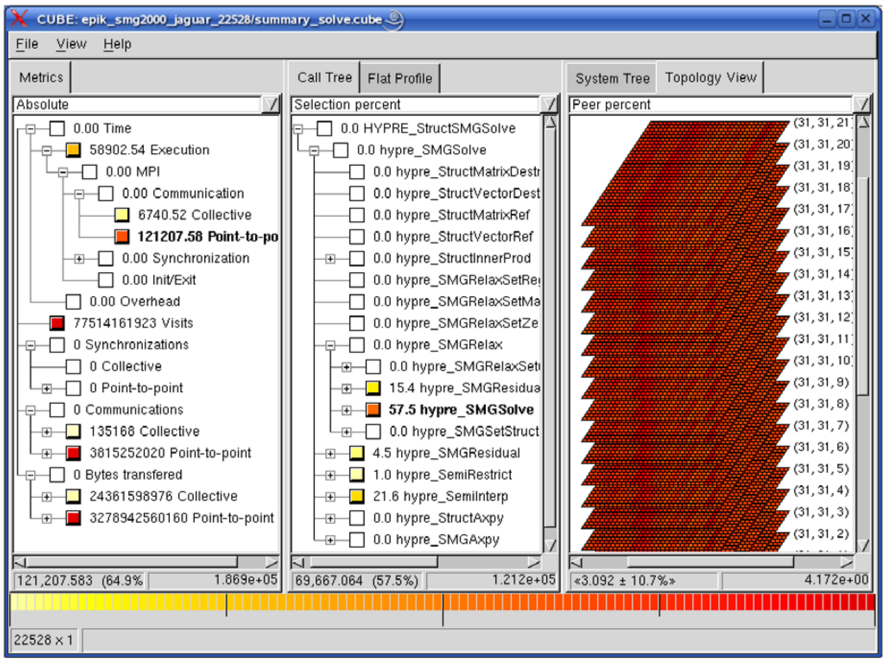
From the trace analysis (Fig. 2(b)), the point-to-point communication time can be refined, with half of the total time determined to be waiting time of early receivers blocked due to senders being late to initiate message transfers, i.e., *Late Sender* time. The distribution of waiting time per process (which is difficult to distinguish in the figure due to the large number of processes) indicates that certain processes in the interior of the grid are most responsible, and this indicates that there could be an opportunity to improve overall performance by rearranging the mapping or work distribution.

With its fixed problem size per process configuration for *weak scaling* behaviour, SMG2000 provides a convenient vehicle for examining the scalability of SCALASCA measurement and analysis on different systems. Figure 3 charts runtime summarisation and trace collection & analysis times on the Cray XT3/4 with different numbers of processes. While total trace size increases slightly faster than linearly, measurement and analysis times scale less than linearly with the number of processes, despite saturating available I/O bandwidth around 30 GB/s when writing and 15 GB/s when reading with 8,192 processes.

3.3. *WRF-NMM on MareNostrum*

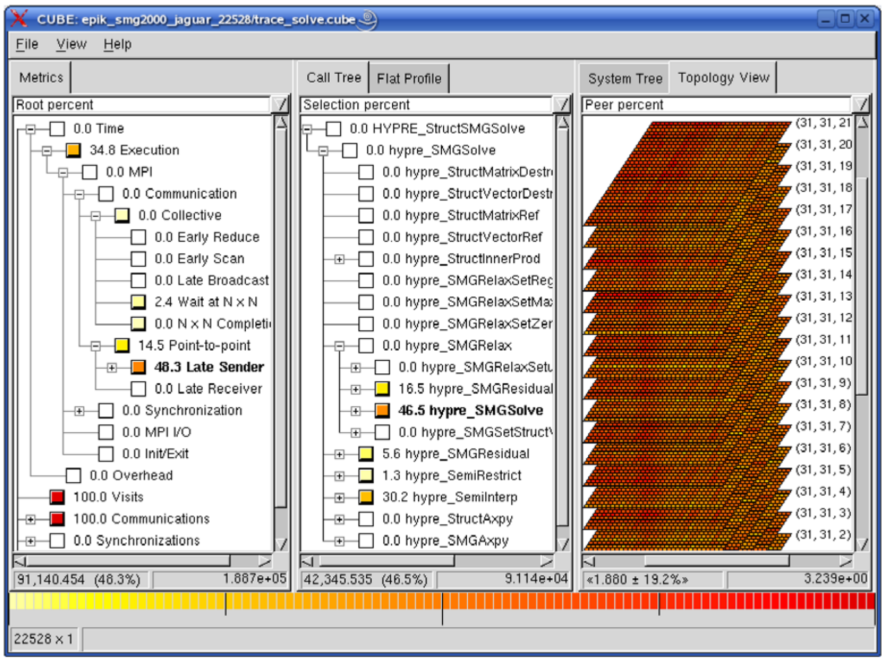
In early 2007, Barcelona Supercomputing Centre *MareNostrum* consisted of 10,240 IBM PowerPC 970MP 2.3 GHz processors (arranged in 44 BladeCenter 1350 xSeries racks of 4-way JS21 blades) [2] and ranked fifth in the 2006/11 Top500 list [20]. Each dual-processor dual-core JS21 Server Blade had 8 GB of shared memory, with Myrinet interconnect between processors, and ran Linux with the GPFS parallel filesystem. Each blade has an independent clock, so measurements required clock offset determination and correction. MPI is based on MPICH optimised for Myrinet, and while both IBM XL and GCC compilers were provided, the former were used in these experiments.

WRF-NMM is a public domain numerical weather prediction code developed by the U.S. National Oceanic and Atmospheric Administration (NOAA) National Centers for Environmental Prediction (NCEP), consisting of the Non-hydrostatic Mesoscale Model (NMM) within the Weather Research and Forecasting (WRF) system [24]. Version 2.1.2 (released January 2006) is a flexible, state-of-the-art atmospheric sim-



```
CUBE: epik_smg2000_jaguar_22528/summary_solve.cube
File  View  Help

Metrics | Absolute
  0.00 Time
    58902.54 Execution
      0.00 MPI
        0.00 Communication
          6740.52 Collective
          121207.58 Point-to-po
        0.00 Synchronization
        0.00 Init/Exit
    0.00 Overhead
  77514161923 Visits
  0 Synchronizations
    0 Collective
    0 Point-to-point
  0 Communications
    135168 Collective
    3815252020 Point-to-point
  0 Bytes transfered
    24361598976 Collective
    3278942560160 Point-to-point
121,207.583 (64.9%)   1.869e+05

Call Tree | Flat Profile | Selection percent
  0.0 HYPRE_StructSMGSolve
    0.0 hypre_SMGSolve
      0.0 hypre_StructMatrixDestr
      0.0 hypre_StructVectorDest
      0.0 hypre_StructMatrixRef
      0.0 hypre_StructVectorRef
      0.0 hypre_StructInnerProd
      0.0 hypre_SMGRelaxSetRe
      0.0 hypre_SMGRelaxSetMa
      0.0 hypre_SMGRelaxSetZe
      0.0 hypre_SMGRelax
        0.0 hypre_SMGRelaxSet
        15.4 hypre_SMGResidua
        57.5 hypre_SMGSolve
        0.0 hypre_SMGSetStruct
      4.5 hypre_SMGResidual
      1.0 hypre_SemiRestrict
      21.6 hypre_SemiInterp
      0.0 hypre_StructAxpy
      0.0 hypre_SMGAxpy
69,667.064 (57.5%)   1.212e+05

System Tree | Topology View | Peer percent
  (31, 31, 21) (31, 31, 20) (31, 31, 19) (31, 31, 18) (31, 31, 17) (31, 31, 16) (31, 31, 15) (31, 31, 14) (31, 31, 13) (31, 31, 12) (31, 31, 11) (31, 31, 10) (31, 31, 9) (31, 31, 8) (31, 31, 7) (31, 31, 6) (31, 31, 5) (31, 31, 4) (31, 31, 3) (31, 31, 2)
«3.092 ± 10.7%»   4.172e+00

22528 x 1
```

(a)



```
CUBE: epik_smg2000_jaguar_22528/trace_solve.cube
File  View  Help

Metrics | Root percent
  0.0 Time
    34.8 Execution
      0.0 MPI
        0.0 Communication
          0.0 Collective
            0.0 Early Reduce
            0.0 Early Scan
            0.0 Late Broadcast
            2.4 Wait at N x N
            0.0 N x N Completi
          14.5 Point-to-point
            48.3 Late Sender
            0.0 Late Receiver
        0.0 Synchronization
        0.0 MPI I/O
        0.0 Init/Exit
    0.0 Overhead
  100.0 Visits
  100.0 Communications
  0.0 Synchronizations
91,140.454 (48.3%)   1.887e+05

Call Tree | Flat Profile | Selection percent
  0.0 HYPRE_StructSMGSolve
    0.0 hypre_SMGSolve
      0.0 hypre_StructMatrixDestr
      0.0 hypre_StructVectorDestr
      0.0 hypre_StructMatrixRef
      0.0 hypre_StructVectorRef
      0.0 hypre_StructInnerProd
      0.0 hypre_SMGRelaxSetReg
      0.0 hypre_SMGRelaxSetMa
      0.0 hypre_SMGRelaxSetZer
      0.0 hypre_SMGRelax
        0.0 hypre_SMGRelaxSetu
        16.5 hypre_SMGResidual
        46.5 hypre_SMGSolve
        0.0 hypre_SMGSetStruct
      5.6 hypre_SMGResidual
      1.3 hypre_SemiRestrict
      30.2 hypre_SemiInterp
      0.0 hypre_StructAxpy
      0.0 hypre_SMGAxpy
42,345.535 (46.5%)   9.114e+04

System Tree | Topology View | Peer percent
  (31, 31, 21) (31, 31, 20) (31, 31, 19) (31, 31, 18) (31, 31, 17) (31, 31, 16) (31, 31, 15) (31, 31, 14) (31, 31, 13) (31, 31, 12) (31, 31, 11) (31, 31, 10) (31, 31, 9) (31, 31, 8) (31, 31, 7) (31, 31, 6) (31, 31, 5) (31, 31, 4) (31, 31, 3) (31, 31, 2)
«1.860 ± 19.2%»   3.239e+00

22528 x 1
```

(b)

Fig. 2. Solver extracts of 22,528-process analysis reports of SMG2000 on Cray XT3/4. (a) Runtime summary showing *Point-to-point communication time*. (b) Trace analysis showing *Late Sender* blocking time.

ulation system designed to be portable and efficient on parallel computing platforms. It consists of 530+ source files running to over 300 thousand lines of code (75% Fortran, 25% C).

Simulations were analysed using the Eur-12km dataset with a default configuration, apart from reducing the duration of the forecast to 3 hours and disabling intermediate checkpoints: it was not determined how

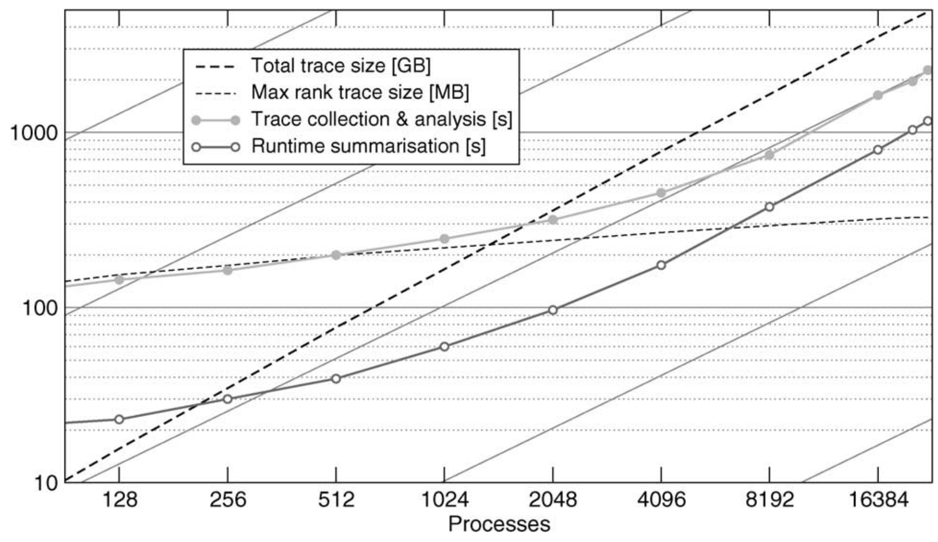


Fig. 3. SCALASCA measurement and analysis scalability for SMG2000 on Cray XT3/4.

to disable dumping the initial state, which resulted in an unnecessarily costly start-up. The simulations used a two-dimensional (squarish) decomposition and mapping to processors, and by running a fixed problem size and the same length of forecast on varying numbers of processors, *strong scaling* was expected. Notably, each simulation timestep has its own characteristics, determined by the configured physics and its implementation, such that some are relatively quick while others take much longer (even when excluding I/O).

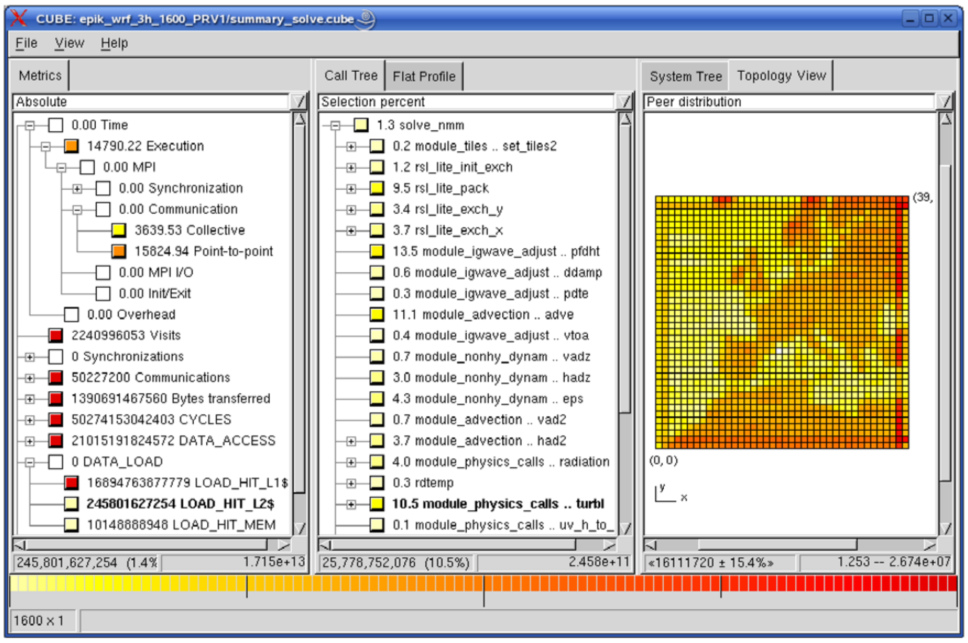
SCALASCA callpath tracking determined that 1970 distinct instrumented callpaths were executed (with a maximum stack depth of 20 frames), of which 272 (14%) were in the simulation timestep phase. 85 of the total callpaths included (instrumented) MPI functions, of which 8 (9%) were in the simulation timestep phase. Producing the integrated callpath profile (including 8 hardware counter metrics) at the end of measurement collection took less than 3 minutes (for the 1,600-process case).

The basic summary profile of WRF-NMM execution time and message-passing statistics is augmented with measured hardware counter metrics and associated derived metrics, similarly structured into hierarchies [27]. Figure 4(a) shows the `solve_nmm` extract of the summary presented by the SCALASCA analysis report browser, with the metric hierarchy partially expanded, the callpath hierarchy with the turbulence physics selected, and the application's 40 $\times$ 40 MPI process topology distribution.

Having selected the metric for the 1.4% of data loads that come from the second level of cache (derived *LOAD_HIT_L2$ = PM_LD_MISS_L1 – PM_DATA_FROM_MEM* from native CPU counts), the values for this metric are shown as percentages for each callpath, where 10.5% occur in `turbl`, and then the distribution of values per process shown using the MPI virtual topology. It is clear that the imbalance arises from the physical nature of the simulation data, since processes with mostly surface water have values notably lower than processes with mostly land, with values also higher close to the simulation grid boundary.

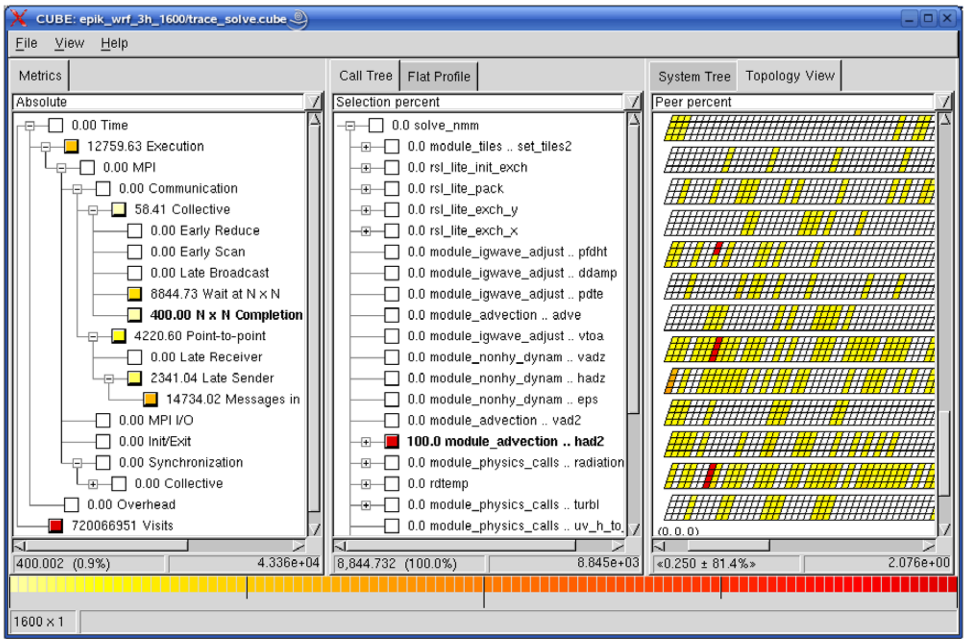
From the callpath and message statistics in the profile summary, the buffering requirements for a complete trace of this execution were estimated to be 118 MB per process (not including tracing hardware counters). Due to the limited I/O capabilities of *MareNostrum*, and to avoid the overhead and perturbation of small, uninteresting functions, a filter file with a short list of functions to be ignored during measurement was prepared, and specified during trace collection to reduce traces below 15 MB.

Flushing the resulting 23 GB of buffered trace data to disk at the end of 10 minutes of execution measurement collection took 3 minutes, and this was subsequently analysed in another 5 minutes. With this new report, it becomes possible to investigate the origins of the various communication and synchronisation times. Most of it is due to the master process broadcasting the initial data to the others, and its relative importance is an artifact of the shortness of the traces. Focusing on the solver itself (Fig. 4(b)), significant amounts of time are identified in point-to-point operations where the receiver was blocked waiting for the sender to initiate the



```
CUBE: epik_wrf_3h_1600_PRV1/summary_solve.cube
File  View  Help

Metrics                          | Call Tree  Flat Profile                          | System Tree  Topology View
Absolute                         | Selection percent                                | Peer distribution
0.00 Time                        | 1.3 solve_nmm                                    |
  14790.22 Execution             |   0.2 module_tiles .. set_tiles2                 |
    0.00 MPI                     |   1.2 rsl_lite_init_exch                         |
      0.00 Synchronization       |   9.5 rsl_lite_pack                              |
      0.00 Communication         |   3.4 rsl_lite_exch_y                            | (39,
        3639.53 Collective       |   3.7 rsl_lite_exch_x                            |
        15824.94 Point-to-point  |   13.5 module_igwave_adjust .. pfdht             |
      0.00 MPI I/O               |   0.6 module_igwave_adjust .. ddamp              |
      0.00 Init/Exit             |   0.3 module_igwave_adjust .. pdte               |
  0.00 Overhead                  |   11.1 module_advection .. adve                  |
2240996053 Visits                |   0.4 module_igwave_adjust .. vtoa               |
0 Synchronizations               |   0.7 module_nonhy_dynam .. vadz                 |
50227200 Communications          |   3.0 module_nonhy_dynam .. hadz                 |
1390691467560 Bytes transferred  |   4.3 module_nonhy_dynam .. eps                  |
50274153042403 CYCLES            |   0.7 module_advection .. vad2                   |
21015191824572 DATA_ACCESS       |   3.7 module_advection .. had2                   |
0 DATA_LOAD                      |   4.0 module_physics_calls .. radiation          | (0, 0)
  16894763877779 LOAD_HIT_L1$    |   0.3 rdtemp                                     |
  245801627254 LOAD_HIT_L2$      |   10.5 module_physics_calls .. turbl             | y
  10148888948 LOAD_HIT_MEM       |   0.1 module_physics_calls .. uv_h_to_           |   x
245,801,627,254  (1.4%   1.715e+13 | 25,778,752,076  (10.5%)   2.458e+11 | «16111720 ± 15.4%»   1.253 -- 2.674e+07
1600 x 1
```

(a)



```
CUBE: epik_wrf_3h_1600/trace_solve.cube
File  View  Help

Metrics                              | Call Tree  Flat Profile                          | System Tree  Topology View
Absolute                             | Selection percent                                | Peer percent
0.00 Time                            | 0.0 solve_nmm                                    |
  12759.63 Execution                 |   0.0 module_tiles .. set_tiles2                 |
    0.00 MPI                         |   0.0 rsl_lite_init_exch                         |
      0.00 Communication             |   0.0 rsl_lite_pack                              |
        58.41 Collective             |   0.0 rsl_lite_exch_y                            |
          0.00 Early Reduce          |   0.0 rsl_lite_exch_x                            |
          0.00 Early Scan            |   0.0 module_igwave_adjust .. pfdht              |
          0.00 Late Broadcast        |   0.0 module_igwave_adjust .. ddamp              |
          8844.73 Wait at N x N      |   0.0 module_igwave_adjust .. pdte               |
          400.00 N x N Completion    |   0.0 module_advection .. adve                   |
        4220.60 Point-to-point       |   0.0 module_igwave_adjust .. vtoa               |
          0.00 Late Receiver         |   0.0 module_nonhy_dynam .. vadz                 |
          2341.04 Late Sender        |   0.0 module_nonhy_dynam .. hadz                 |
            14734.02 Messages in     |   0.0 module_nonhy_dynam .. eps                  |
      0.00 MPI I/O                   |   0.0 module_advection .. vad2                   |
      0.00 Init/Exit                 |   100.0 module_advection .. had2                 |
      0.00 Synchronization           |   0.0 module_physics_calls .. radiation          |
        0.00 Collective              |   0.0 rdtemp                                     |
  0.00 Overhead                      |   0.0 module_physics_calls .. turbl              |
720066951 Visits                     |   0.0 module_physics_calls .. uv_h_to_           | (0, 0, 0)
400.002  (0.9%)   4.336e+04 | 8,844.732  (100.0%)   8.845e+03 | «0.250 ± 81.4%»   2.076e+00
1600 x 1
```

(b)

Fig. 4. Solver extracts of 1600-process WRF-NMM (Eur-12km) analysis reports on *MareNostrum*. (a) Runtime summary showing *LOAD_HIT_L2$* in solver turbulence step with MPI Cartesian virtual topology. (b) Trace analysis showing MPI $N \times N$ *Completion time* in solver advection step with physical machine topology.

message transfer (*Late Sender*) and imbalance when processes enter all-to-all collectives and must wait on others (*Wait at* $N \times N$).

Of particular interest is the highlit $N \times N$ *Completion time*, which indicates an unusual imbalance exiting the `MPI_Allreduce` in the advection module of the solver. Generally this time would be negligible, and initially was thought to be due to limitations of the schemes employed to determine clock offsets and correct clock drifts. From closer investigation of the traces, however, it was determined that in this case it is significant because it shows a single occurrence where a few of the processes were substantially late exiting, over 1.75 seconds.

The distribution can be presented using a representation of the physical topology of *MareNostrum* (Fig. 4(b)), where servers are shown vertically, blades horizontally, and the four processors on a blade obliquely. The affected processes are often in groups of four sharing a JS21 blade, e.g., ranks 120..123 on s02c1b07 and 632..635 on s06c1b08, but not always, e.g., only two of the ranks 888..891 on s09c1b08. While this is a relatively small amount of wasted time in total, it imbalances the other processes who attempt to exchange data immediately afterwards, and is the origin of the bulk of the *Late Sender* instances and subsequent *Wait at* $N \times N$ for the next `MPI_Allreduce`. Occurrences of this uncoordinated exit from collective operations are sporadic and relatively infrequent, but result in serious disruption of the smooth execution of the application.

Other applications using collective operations on *MareNostrum* have also been found to be impacted by this anomalous MPI behaviour, with different processes (and associated processors/blades) in each occurrence. This suggests that neither the applications nor the hardware are responsible, and it appeared that operating system jitter or the MPI(CH) implementation for Myrinet is the culprit: several versions of MPI have been tried, but no fix or workaround has yet been effective. Since the impact of this disruption grows proportionally with the number of processes, it makes the use of larger *MareNostrum* processor configurations unproductive for afflicted applications.

3.4. XNS-DeBakey on Blue Gene/L

At the end of 2006, the JUBL Blue Gene/L system installed in Forschungszentrum Jülich had 8,192 dual-core 700 MHz PowerPC 440 compute nodes (each with 512 MB of memory), 288 I/O nodes, and additional service and login nodes [23,9] and was ranked 13th in the 2006/11 Top500 list [20]. Compute nodes ran a Linux-based microkernel that did not support dynamic linking nor virtual memory. The constrained process memory was a significant challenge for applications and associated tools, though the dedicated network for collectives and fully-connected three-dimensional torus interconnect reliably delivered good MPI performance with IBM MPICH and IBM XL compilers. The system was running the V1R3 software release with GPFS parallel filesystem configured with 4 servers. Globally synchronised clocks avoid the need for clock offset and drift correction when analysing timestamped events.

XNS is a computational fluid dynamics (CFD) code for simulations of unsteady fluid flows, including microstructured liquids, in situations involving significant deformations of the computational domain, developed by the Chair for Computational Analysis of Technical Systems of RWTH Aachen University. Simulations are based on finite-element techniques using stabilised formulations, unstructured three-dimensional meshes and iterative solution strategies [11,29]. The parallel implementation is based on message-passing communication libraries, exploits mesh-partitioning techniques, and is portable across a wide range of computer architectures.

The XNS code, consisting of more than 32,000 lines of Fortran90 in 66 files, uses an EWD substrate library which fully encapsulates the use of BLAS and communication libraries, which is another 12,000 lines of mixed Fortran and C within 39 files. Although the MPI version of XNS was already ported to BG/L, and available datasets allowed simulations to run with up to 4,096 processes, scalability of the code at that point was only acceptable up to 900 processes.

Performance was studied with a test-case consisting of a 3-dimensional space-time simulation of the MicroMed DeBakey axial ventricular assist blood pump. Very high resolution simulation is required to accurately predict shear-stress levels and flow stagnation areas in an unsteady flow in such a complex geometry. The mesh for the pump consisted of 3,714,611 elements (1,261,386 nodes) divided by a graph partitioner into element sets which form contiguous subdomains that are assigned to processes. With each set of elements assigned to a single process, the nodes are then distributed in such a way that most nodes which are interior to a subdomain are assigned to the process which holds elements of the same subdomain. Nodes at a subdomain boundary are assigned to processes sharing that boundary. The formation of element-level components of the system of equations proceeds fully in parallel, with all data related to a given element residing in the same process. The solution of that system of equations takes place within a GMRES iterative solver, and it is here that the bulk of inter-process communication occurs, with the element-based structures (stiffness matrices and local residuals) interacting with node-based structures (global residuals and increments). Required movement of data from node-level to element-level takes the form of a *scatter* and the reverse movement from element-level to node-level takes the form of a *gather*. These operations are done in two stages: one local to the subdomain (and free of communication) and another at the surface of the subdomains (where communication is required). Four Newton–Raphson iterations are typically carried out in the solver per simulation timestep.

SCALASCA callpath tracking determined that 896 distinct callpaths were executed (with a maximum depth of 12 frames), of which 408 (46%) were in the solver timestep loop. 187 of the callpaths included (instrumented) MPI functions, of which 65 (35%) were in the solver timestep phase. For the 4,096-process measurement of the 16 minute XNS execution, producing the integrated profile at the end of measurement required an additional 4 minutes.

Initial SCALASCA runtime summaries with up to 4,096 processes (Fig. 5(a)) showed MPI times starting to dominate when the number of processes approached one thousand, and identified growing quantities of point-to-point synchronisations (in the form of `MPI_Sendrecv` operations) in the scatter and gather routines as particularly worthy of closer investigation.

Such MPI point-to-point sends and receives with no data payload can be useful for pairwise process coordination, or can be indicative of unnecessary message traffic when there is no actual data to transfer. While `MPI_Sendrecv` provides a convenient optimised combination of `MPI_Send` and `MPI_Recv`, employed to exchange boundary elements between partitions, the fact that each process made the same number of calls suggested that an exchange was being done for every possible combination, even when there was no data to transfer. With the current implementation of the SCALASCA measurement system it is not possible to determine the associated (time) cost of such transfers during runtime summarisation, however, it is possible from the trace analysis. (Other profiling tools distinguish point-to-point synchronisation indirectly via metrics for minimum message sizes [22] or from a set of message-size bins [7].)

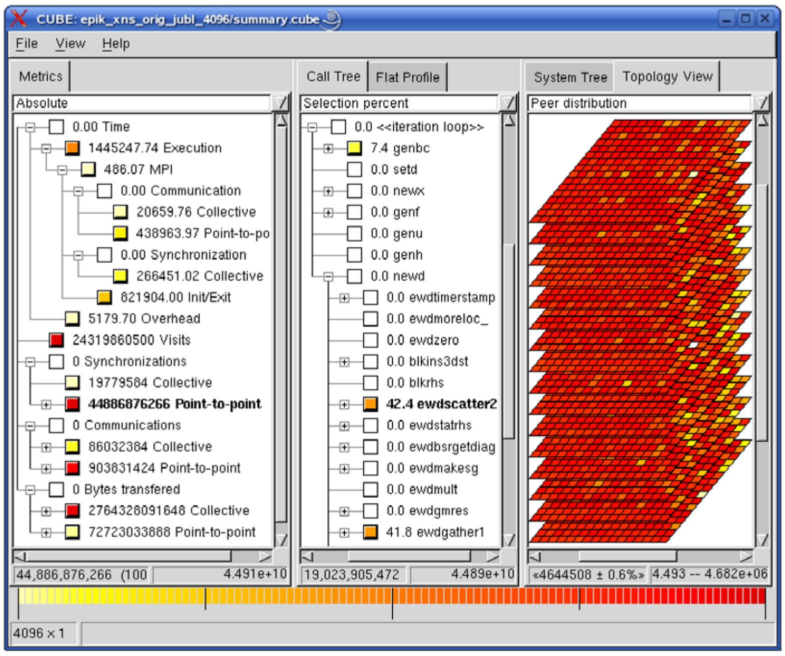
From examination of the summary report, the trace buffer requirements for a complete trace of 4,096 processes were estimated at 277 MB per process (1,100 GB in total), due to both the quantity of those synchronisations and a similarly expensive but uninteresting initialisation phase. This implied that it would not be possible to trace and analyse more than one solver timestep, even after filtering undesirable application functions. Fortunately, profiles of different numbers of timesteps showed no significant difference between the first and subsequent timesteps, so the trace analysis would remain representative. Subsequent analysis of the collected traces (Fig. 5(b)), concentrating on the central timestep solver iteration loop, highlit the crippling cost of those point-to-point synchronisations in both scatter and gather that impedes scalability.

In conjunction with the XNS application developers, the first attempt at remediation was to replace the `MPI_Sendrecv` operations with separate `MPI_Send` and `MPI_Recv` operations, in each case checking the size and only performing the transfer when actually required. Since a static partitioning of the mesh is employed, the number of elements linking each partition is known in advance (by potential senders and receivers), and when there are no links there is no data to transfer. This straightforward (and in this context safe) optimisation, which eliminated point-to-point synchronisation had little benefit on the performance with less than 1,024 processes, since the communication matrix remains relatively dense at this scale. Figure 6(a) shows how performance improved dramatically for larger configurations though, resulting in a more than fourfold overall performance improvement to over 460 simulation timesteps/hour with 4,096 processes (compared to the best performance of the original version which peaked around 1,024 processes).

SCALASCA analysis of the modified version (reported in [29]) confirmed the elimination of this initial performance problem, and has revealed several further inefficiencies which are currently being investigated: e.g., load imbalance indicated by waiting time entering barriers and receivers waiting for late senders, both of which can only be determined by trace analysis. Further scalability is therefore also promising, however, lack of suitably partitioned datasets for larger numbers of processes has prevented pursuing this to date. Use of asynchronous (non-blocking) message transfers within the gather and scatter was found to produce no additional performance improvement, which may be due to the small amount of computation available for overlap with communication within these routines.

4. Related work

There is a substantial history of tools developed for scalability on parallel systems (e.g., [18]), however,



CUBE: epik_xns_orig_jubl_4096/summary.cube

File View Help

Metrics

```
Absolute
0.00 Time
  1445247.74 Execution
    486.07 MPI
      0.00 Communication
        20659.76 Collective
        438963.97 Point-to-po
      0.00 Synchronization
        266451.02 Collective
      821904.00 Init/Exit
    5179.70 Overhead
24319860500 Visits
0 Synchronizations
  19779584 Collective
  44886876266 Point-to-point
0 Communications
  86032384 Collective
  903831424 Point-to-point
0 Bytes transfered
  2764328091648 Collective
  72723033888 Point-to-point
44,886,876,266 (100    4.491e+10
```

Call Tree | Flat Profile

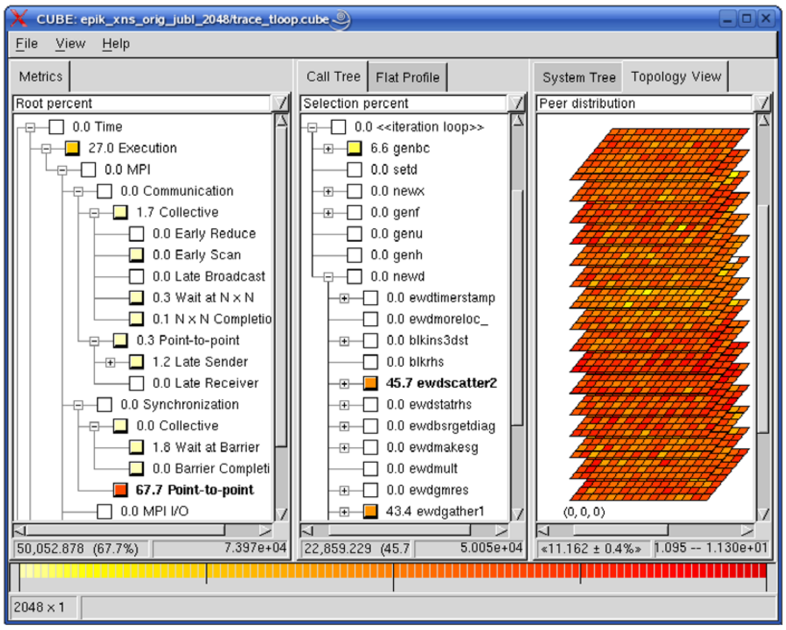
```
Selection percent
0.0 <<iteration loop>>
  7.4 genbc
  0.0 setd
  0.0 newx
  0.0 genf
  0.0 genu
  0.0 genh
  0.0 newd
    0.0 ewdtimerstamp
    0.0 ewdmoreloc_
    0.0 ewdzero
    0.0 blkins3dst
    0.0 blkrhs
    42.4 ewdscatter2
    0.0 ewdstatrhs
    0.0 ewdbsrgetdiag
    0.0 ewdmakesg
    0.0 ewdmult
    0.0 ewdgmres
    41.8 ewdgather1
19,023,905,472    4.489e+10
```

System Tree | Topology View

```
Peer distribution
«4644508 ± 0.6%»  4.493 -- 4.682e+06
```

4096 x 1

(a)



CUBE: epik_xns_orig_jubl_2048/trace_tloop.cube

File View Help

Metrics

```
Root percent
0.0 Time
  27.0 Execution
    0.0 MPI
      0.0 Communication
        1.7 Collective
          0.0 Early Reduce
          0.0 Early Scan
          0.0 Late Broadcast
          0.3 Wait at N x N
          0.1 N x N Completio
        0.3 Point-to-point
          1.2 Late Sender
          0.0 Late Receiver
      0.0 Synchronization
        0.0 Collective
          1.8 Wait at Barrier
          0.0 Barrier Completi
        67.7 Point-to-point
      0.0 MPI I/O
50,052,878 (67.7%)    7.397e+04
```

Call Tree | Flat Profile

```
Selection percent
0.0 <<iteration loop>>
  6.6 genbc
  0.0 setd
  0.0 newx
  0.0 genf
  0.0 genu
  0.0 genh
  0.0 newd
    0.0 ewdtimerstamp
    0.0 ewdmoreloc_
    0.0 blkins3dst
    0.0 blkrhs
    45.7 ewdscatter2
    0.0 ewdstatrhs
    0.0 ewdbsrgetdiag
    0.0 ewdmakesg
    0.0 ewdmult
    0.0 ewdgmres
    43.4 ewdgather1
22,859,229 (45.7    5.005e+04
```

System Tree | Topology View

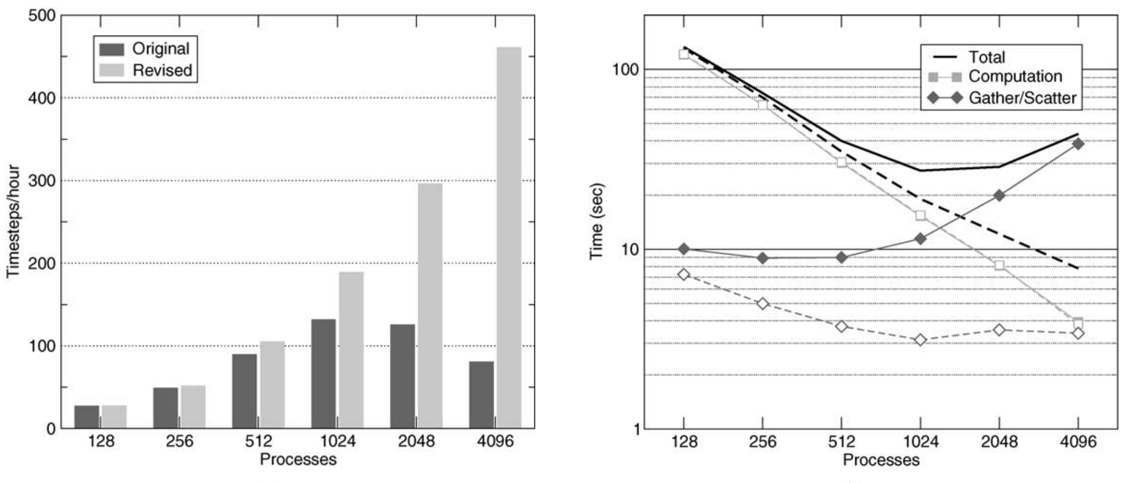
```
Peer distribution
(0, 0, 0)
«11.162 ± 0.4%»  1.095 -- 1.130e+01
```

2048 x 1

(b)

Fig. 5. Analysis reports of XNS (DeBakey dataset) simulation on Blue Gene/L. (a) Runtime summary of 4,096 processes showing *Point-to-point synchronisation operations*. (b) Trace analysis extract of 2,048 processes showing *Point-to-point synchronisation time*.

the requirements and challenges have continued to grow with each generation of highly-parallel computers [15]. Computer systems are typically provided with vendor-specific profile and trace collection and analysis tools, offering broadly similar functionality with distinct user interfaces and generally non-interoperable formats: e.g., IBM HPCT [8] and CrayPat/Apprentice2 [5]. Although custom-developed for their respective large-scale HPC computer systems, significant training is required with each tool to be able to use and apply it



(a) (b)

Fig. 6. Comparison of original and subsequently revised XNS solver performance with DeBakey axial pump on various partition sizes of Blue Gene/L. (a) Originally unacceptable large-scale performance is improved significantly to perform over 460 timesteps/hour. (b) Breakdown of the solver component costs/timestep shows good scalability of the primarily computational components and significant improvement to Gather/Scatter scalability (original: solid lines, revised: dashed lines).

effectively. One recent study [4] surveys a range of performance tools available on Blue Gene/L, and demonstrated how summarisation tools were simple and easy to use compared with tracing tools that required appropriate configuration to avoid I/O problems. By integrating summarisation and tracing capabilities, with improved support for trace collection configuration and analysis, the SCALASCA toolset combines powerful functionality with enhanced ease of use.

Furthermore, for application developers who work on multiple computer systems (from different vendors), including their local (Linux) cluster or laptop/desktop system, proprietary tools that are only available on certain systems are an impediment to their incorporation within their regular development process. Open-source tools are often a preferred alternative which are portable to a range of computer systems and can be customised for new systems when required. Examples include the mpiP and FPMPI-2 profilers of MPI message-passing [7,22] and PAPI for acquiring metrics from system-specific hardware counters [3]. Many other important performance tools exist, though they typically have yet to be ported to the special environments of the largest supercomputers or demonstrate scalability to multi-thousand processes.

The SCALASCA toolset shares functionality (and a certain amount of development) with the widely-available TAU toolset [19], which integrates numerous tools, including profiling and tracing capabilities. TAU can build application executables with different instrumentation and measurement libraries to separately collect profile summaries and traces for automatic or interactive analysis. Furthermore, instrumentation and measurement can be configured to improve its effectiveness. TAU can already be configured to collect SCALASCA traces, and use SCALASCA presentation of analysis reports, and it should be straightforward to integrate the latest SCALASCA functionality into the TAU system. In this combination, TAU profile summaries could thereby improve application instrumentation and configuration of measurement and trace collection, or exploit SCALASCA's integrated profiling and tracing capabilities from a single instrumented executable.

Performance analysis is rarely complete with a relatively high-level analysis of performance properties, regardless of the extent of automation and thoroughness of the search, and generally is complemented with the ability to examine specific instances of performance problems in detail. SCALASCA traces (after conversion, if necessary) can therefore be visualised and analysed with Vampir [16] or Paraver [13], and preliminary work has demonstrated integration which isolates and examines problem instances extracted from execution traces. VampirServer supports parallel trace file visualisation and analysis [12], while Paraver addresses analysis scalability with multi-level processing of event traces, e.g., automatically extracting key phases from long traces and using software counters to summarise event sequences [14].

5. Conclusions and future work

By using SCALASCA toolset capabilities to measure and analyse the performance of quite different complex HPC applications on diverse leading-edge supercomputers, the scalability of applications, toolset and computer systems have all been assessed. Working at these as yet uncommon and unfamiliar scales, numerous immaturities and opportunities for considerable improvement abound, which will be critical when progressing to the next generations where scales will continue to increase.

Complementing runtime summarisation with selective event tracing, the SCALASCA toolset demonstrated that it is fundamentally suited for its task of analysing complex applications at the largest scales. At the same time, scalability limitations became evident in the measurement system, trace analysis, and analysis data model and report examiner. Often these only impact particular systems or configurations, each of which is unique in many ways, and the benefit of customised approaches needs to be evaluated. Management of the voluminous quantities of measurement and analysis data requires ongoing careful consideration of lean and efficient data structures and operations: the open-source SCALASCA 1.0 release [10] incorporates a 25-fold improvement in definition unification time and 15% reduction in memory required for trace analysis, amongst numerous other functionality and performance enhancements.

More flexibility in the configuration of instrumentation, measurement and analyses are currently under investigation to allow more directed and efficient integrated analysis. Integration and automation are also areas where ease-of-use can continue to be improved, and there are many opportunities to further exploit the newly available synergies.

Acknowledgements

This work was partially funded under the German Helmholtz Association Young Investigators Program under Contract No. VNG-118. Collaboration with the developers of the XNS application from RWTH Aachen University was undertaken using resources of the John von Neumann Institute for Computing as part of the first Blue Gene/L Scaling Workshop in Jülich.

Part of this work was carried out using resources of the Barcelona Supercomputing Centre under the HPC-EUROPA project (RII3-CT-2003-506079), with the support of the European Community Research Infrastructure Action under the FP6 "Structuring the European Research Area" program.

This research also used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

The authors are grateful for the assistance and guidance provided by the support and scientific staff of the respective supercomputer centres.

References

[1] Advanced Simulation and Computing Program, The ASC SMG2000 benchmark code, https://asc.llnl.gov/computing_resources/purple/archive/benchmarks/smg/, 2001.

[2] Barcelona Supercomputing Centre (Spain), IBM BladeCenter JS21 cluster *MareNostrum*, //www.bsc.es/.

[3] S. Browne, J.J. Dongarra, N. Garner, G. Ho and P. Mucci, A portable programming interface for performance evaluation on modern processors, *Int. J. High Performance Computing Applications* **14**(3) (2000), 189–204.

[4] I.-H. Chung, R.E. Walkup, H.-F. Wen and H. Yu, MPI performance analysis tools on Blue Gene/L, in: *Proc. SC06*, Tampa, FL, USA, IEEE Computer Society, 2006.

[5] Cray Inc., Cray performance analysis tools, CrayDoc S-2474-31, //docs.cray.com/, 2006.

[6] M. Geimer, F. Wolf, B.J.N. Wylie and B. Mohr, Scalable parallel trace-based performance analysis, in: *Proc. 13th European PVM/MPI User's Group Meeting*, Bonn, Germany, Lecture Notes in Computer Science, Vol. 4192, Springer, 2006, pp. 303–312.

[7] W. Gropp and K. Buschelman, FPMPI-2 fast profiling library for MPI, //www-unix.mcs.anl.gov/fpmpi/, 2006.

[8] IBM Advanced Computing Technology Center, High Performance Computing Toolkit, //www.research.ibm.com/actc/.

[9] The Blue Gene/L Team at IBM and LLNL, An overview of the Blue Gene/L supercomputer, in: *Proc. SC2002*, Baltimore, MD, USA, IEEE Computer Society, 2002.

[10] Jülich Supercomputing Centre, Forschungszentrum Jülich, SCALASCA: Scalable performance analysis of large-scale parallel applications, //www.scalasca.org/.

[11] J.G. Kennedy, M. Behr, V. Kalro and T.E. Tezduyar, Implementation of implicit finite element methods for incompressible flows on the CM-5, *Computer Methods in Applied Mechanics and Engineering* **119** (1994), 95–111.

[12] A. Knüpfer, H. Brunst and W.E. Nagel, High performance event trace visualization, in: *Proc. 13th EuroMicro Conf. on Parallel, Distributed and Network-Based Processing*, PDP, Lugano, Switzerland, IEEE Computer Society, 2005, pp. 258–263.

[13] J. Labarta, S. Girona, V. Pillet, T. Cortes and L. Gregoris, DiP: A parallel program development environment, in: *Proc. 2nd Int'l EuroPar Conf. on Parallel Processing*, Lyon, France, Lec-

ture Notes in Computer Science, Vol. 1124, Springer, 1996, pp. 665–674.

[14] J. Labarta, J. Gimenez, E. Martinez, P. Gonzalez, H. Servat, G. Llort and X. Aguilar, Scalability of visualization and tracing tools, in: *Proc. 11th Parallel Computing Conf. (ParCo 2005)*, Málaga, Spain, John von Neumann Institute for Computing Series, Vol. 33, 2006, pp. 869–876.

[15] B. Mohr, B.J.N. Wylie and F. Wolf, Performance measurement and analysis tools for extremely scalable systems, in: *Proc. 23rd Int'l Supercomputing Conference (ISC'08)*, Dresden, Germany, to appear.

[16] W. Nagel, A. Arnold, M. Weber, H.-C. Hoppe and K. Solchenbach, Vampir: Visualization and analysis of MPI resources, *Supercomputer* **63**(1) (1996), 69–80.

[17] L. Oliker, A. Canning, J. Carter, C. Iancu, M. Lijewski, S. Kamil, J. Shalf, H. Shan, E. Strohmaier, S. Ethier and T. Goodale, Scientific application performance on candidate petascale platforms, in: *Proc. 21st Int'l Parallel and Distributed Processing Symp. (IPDPS)*, Long Beach, CA, USA, IEEE Computer Society, 2007, pp. 1–12.

[18] D. Reed, R.D. Olsen, R.A. Aydt, T.M. Madhyastha, T. Birkett, D.W. Jensen, B.A.A. Nazief and B.K. Totty, Scalable performance environments for parallel systems, in: *Proc. 6th Distributed Memory Computing Conf.*, Portland, OR, USA, IEEE Computer Society, 1991, pp. 562–569.

[19] S.S. Shende and A.D. Malony, The TAU parallel performance system, *Int'l J. High Performance Computing Applications* **20**(2) (2006), 287–331.

[20] TOP500 Supercomputing Systems, //www.top500.org/, 2007.

[21] U.S. National Center for Computational Sciences, Oak Ridge National Lab., Cray XT *Jaguar*, //info.nccs.gov/resources/jaguar.

[22] J. Vetter and C. Chambreau, MPIP – lightweight, scalable MPI profiling, //www.llnl.gov/CASC/mpip/, 2005.

[23] John von Neumann Institute for Computing, Forschungszentrum Jülich, Jülicher Blue Gene/L, //www.fz-juelich.de/zam/ibm-bgl/.

[24] Weather Research Forecast code, //www.wrf-model.org/.

[25] F. Wolf and B. Mohr, Automatic performance analysis of hybrid MPI/OpenMP applications, *J. Systems Architecture* **49**(10/11) (2003), 421–439.

[26] F. Wolf, B.J.N. Wylie, E. Ábrahám, D. Becker, W. Frings, K. Fürlinger, M. Geimer, M.-A. Hermanns, B. Mohr, S. Moore, M. Pfeifer and Z. Szebenyi, Usage of the SCALASCA toolset for scalable performance analysis of large-scale parallel applications, in: *Tools for High Performance Computing*, Springer, 2008.

[27] B.J.N. Wylie, B. Mohr and F. Wolf, Holistic hardware counter performance analysis of parallel programs, in: *Proc. 11th Parallel Computing Conf. (ParCo 2005)*, Málaga, Spain, John von Neumann Institute for Computing Series, Vol. 33, 2006, pp. 187–194.

[28] B.J.N. Wylie, F. Wolf, B. Mohr and M. Geimer, Integrated runtime measurement summarisation and selective event tracing for scalable parallel execution performance diagnosis, in: *Proc. 8th Workshop on State-of-the-Art in Scientific and Parallel Computing (PARA'06)*, Umeå, Sweden, Lecture Notes in Computer Science, Vol. 4699, Springer, 2006, pp. 460–469.

[29] B.J.N. Wylie, M. Geimer, M. Nicolai and M. Probst, Performance analysis and tuning of the XNS CFD solver on Blue Gene/L, in: *Proc. 14th European PVM/MPI User's Group Meeting*, Paris, France, Lecture Notes in Computer Science, Vol. 4757, Springer, 2007, pp. 107–116.

[30] Z. Szebenyi, B.J.N. Wylie and F. Wolf, SCALASCA parallel performance analyses of SPEC MPI2007 applications, in: *Proc. 1st SPEC Int'l Performance Evaluation Workshop (SIPEW)*, Darmstadt, Germany, Lecture Notes in Computer Science, Vol. 5119, Springer, 2008.