Large-scale performance analysis of *PFLOTRAN* with Scalasca

Brian J. N. Wylie & **Markus Geimer**, *Jülich Supercomputing Centre, Forschungszentrum Jülich, Germany*

ABSTRACT: *The PFLOTRAN code for multiphase subsurface flow and reactive transport has featured prominently in US Department of Energy SciDAC and INCITE programmes, where is has been used to simulate migration of radionucleide contaminants in groundwater. As part of its ongoing development, execution performance with up to 128k processor cores on Cray XT and IBM BG/P systems has been investigated, and a variety of aspects have been identified to inhibit PFLOTRAN performance at larger scales using the open-source Scalasca toolset. Scalability of Scalasca measurements and analyses themselves, previously demonstrated with a range of applications and benchmarks, required re-engineering in key areas to handle the complexities of PFLOTRAN executions employing MPI within PETSc, LAPACK, BLAS and HDF5 libraries at large scale.*

KEYWORDS: MPI, performance measurement and analysis, scalability

1 Introduction

Scalasca is an open-source toolset for analysing the execution behaviour of applications based on the MPI and/or OpenMP parallel programming interfaces supporting a wide range of current HPC platforms [1, 7]. It combines compact runtime summaries, that are particularly suited to obtaining an overview of execution performance, with in-depth analyses of concurrency inefficiencies via event tracing and parallel replay. With its highly scalable design, Scalasca has facilitated performance analysis and tuning of a range of applications on Cray XT and XE systems [22] and consisting of unprecedented numbers of processes, namely 294,912 on IBM Blue Gene/P and 196,608 on Cray XT5 [23].

In the context of a 2010 workshop on "Program development for extreme-scale computing" [12], developers of debugging and performance tools were challenged to analyse two MPI-based applications executing with at least 10,000 processes on the two dominant HPC architectures, IBM Blue Gene/P (*jugene.fz-juelich.de*) and Cray XT5 (*jaguar.nccs.ornl.gov*). The Virtual Institute – High-Productivity Supercomputing and Performance Evaluation and Analysis Consortium generously provided accounts and sizable resource allocations on the leadership systems at Jülich Supercomputing Centre and Oak Ridge National Laboratory to research groups and vendors (including Cray and IBM) for this challenge, and support staff at each facility assisted as necessary to get system software and the subject applications installed and running.

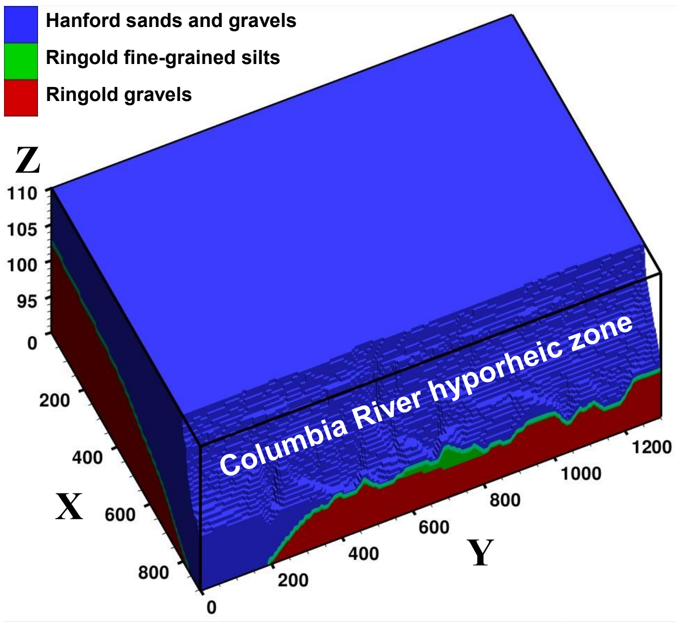
After reviewing the challenge applications, performance measurement and analysis of *PFLOTRAN* with Scalasca is presented. While the version of Scalasca available at the time proved able to master the set challenge, important lessons were learnt in the process that drove research and development of improved measurement and analysis for complex applications at large scales.

2 Challenge applications

One of the subject applications was the *PEPC* three-dimensional parallel tree code used for various plasma and astrophysics simulations available as part of DEISA and PRACE benchmark suites [11]. It could be built and run without difficulty on both systems, however, with the available datasets its scalability was limited to a maximum of 16,384 processes. This was therefore not particularly challenging, and many tools successfully demonstrated debugging and analysis of its execution performance. Scalasca was able to compare analyses of *PEPC* executions on Jugene IBM BG/P and Jaguar Cray XT5 and the relative efficiency of collective communication for the tree walk used updating fields [22].

The other subject application, and focus of this paper, was the *PFLOTRAN* three-dimensional reservoir simulator [4] that has featured prominently in US Department of Energy SciDAC [18] and INCITE [10] programmes, where it has been used to simulate migration of radionucleide contaminants in groundwater and geologic CO_2 sequestration [13, 9, 14, 15, 8]. The challenge test case [19] consisted of the version of the code from January 2010, with already improved performance and scalability as a result of PERI [17] "Tiger Team" active liaison, and 10 simulation time-steps of a "2B" radionucleide transport problem dataset for the DOE Hanford 300 area next to the Columbia River in southeastern Washington state as shown in Figure 1.

This coupled flow and transport simulation consists of alternating groundwater flow (*PFLOW*) and reactive, multi-component contaminant transport (*PTRAN*) in each time-step, where fluid fluxes and phase saturation states are used to compute solute transport. The problem consists of $850 \times 1000 \times 160$ cells and includes 15 chemical components, resulting in 136E6 and 2.04E9 total degrees of freedom in the flow and transport solves, respectively. Simulations use a structured grid containing inactive cells along the river boundary and the associated domain decomposition results in idling about 5% of processes. Some 80,000 lines of Fortran9X are organised in 97 source files, and the code uses the PETSc, LAPACK, BLAS and HDF5 I/O libraries which encapsulate MPI usage. An outer, inexact New-


Hanford sands and gravels
Ringold fine-grained silts
Ringold gravels
Z
X
Y
Columbia River hyporheic zone

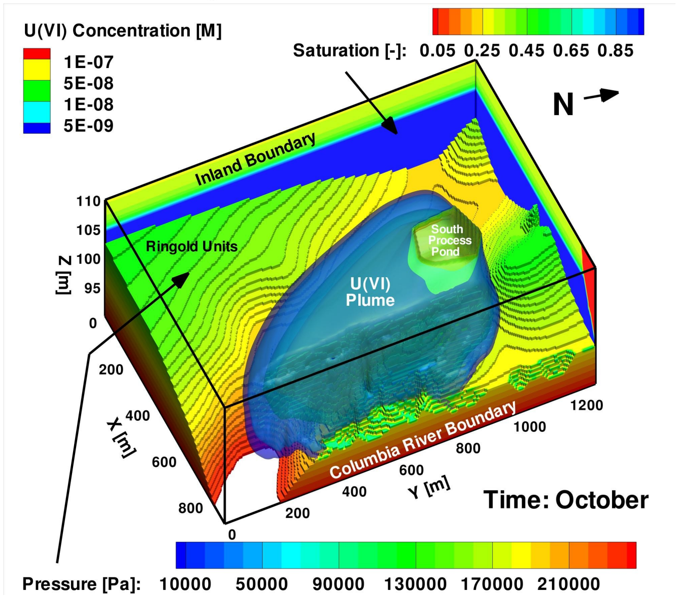


Figure 1: Geology of DOE Hanford 300 Area (WA, USA) in *PFLOTRAN* "2B" test case, showing 6–7% of grid cells inactive within channel of Columbia River, and isosurfaces of acqueous radionucleide concentration in October timeframe (used with authors' permission [8]).

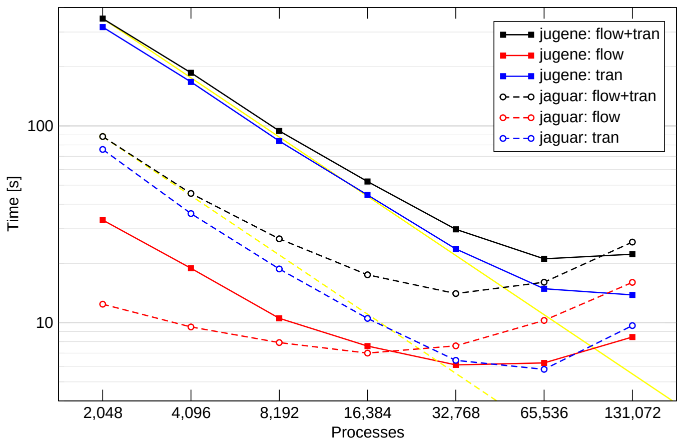


Figure 2: Average timestep durations of *PFLOTRAN* "2B" test case on Jaguar Cray XT5 and Jugene IBM BG/P at a range of scales, with separation of 'flow' and 'tran'(sport) phases. Ideal scaling shown by yellow lines.

ton method with an inner, exact stabilised bi-conjugate gradient solver (BiCGstab) from PETSc is employed. For the *PFLOTRAN* test case a developer version (petsc-dev) of PETSc [5] was required, which itself proved to be rather challenging to build and install on the test systems and led to the majority of support requests from the tools teams. (Converting *PFLOTRAN* input data files from DOS to Unix format so that they were read correctly was a relatively minor inconvenience by comparison.)

On both HPC platforms the recommended compilers and MPI libraries were employed, namely PGI 10.3 compilers and Cray XT-MPT on Jaguar and IBM XL 9.0/11.1 compilers and BG-MPI on Jugene, to build both PETSc and *PFLOTRAN* (while linking other system-installed libraries). Jaguar has compute nodes comprising dual hex-core 2.6 GHz Opteron processors whereas Jugene has quad-core 850 MHz PowerPC processors, with Jaguar also having 1333MB memory per core compared to 512MB on Jugene. Where available compute node memory allowed it, executions with the test data set were done in "virtual node" mode with an MPI process running on each processor core in a dedicated system partition.

PFLOTRAN execution times for *strong scaling* on Jaguar and Jugene, as reported by the application itself for 10 simulation timesteps, are shown in Figure 2. (Initialization where the input dataset is read and distributed, and final writing of simulation results, were part of an on-going I/O optimisation activity, and excluded here as they are subject to considerable variation on the shared parallel filesystems.) Note that rather than using full 12-core nodes on Jaguar, the process counts used there were also powers of two, as this was found to result in better performance due to preferred three-dimensional decompositions.

For this test case, solver scalability ('flow+tran') is seen to be very good up to 32k processes on Jugene IBM BG/P, and although it is more than four times faster with 4k processes on Jaguar Cray XT5 (and indeed fastest overall with 32k processes on Jaguar) inferior scalability is evident. Larger executions with up to 128k processes on both systems have diminished performance. Considering the two solver phases separately, reactive transport ('tran') scales much better than groundwater 'flow' on both platforms, and while 'flow' is faster at smaller scales there is a crossover with more than 16k processes on Jaguar Cray XT5.

As the *PFLOTRAN* developers note [14], although the "2B" test case is considered 'petascale' it is still a very small problem to be running on so many processors. They also report substantial performance benefits achieved via both algorithmic improvements and adjustment of a number of MPI communication-related environment variables. Neither of these aspects were pertinent to the Dagstuhl challenge of analysing the execution performance and scalability of the specified configuration, diagnosis of its inefficiencies and proposing potential remedies.

3 Performance analysis with Scalasca

PFLOTRAN presented a variety of challenges for its performance analysis with Scalasca, ranging from instrumentation, measurement collection and analysis, to analysis report exploration. Some of these challenges were mastered with the Scalasca 1.3.1 release that was current for the challenge in May 2010, others were improved in subsequent releases, and some are work in progress.

3.1 Scalasca instrumentation

Configuring and building PETSc is a complex and lengthy undertaking on IBM BG/P and Cray XT5, partially due to their architecture with separate compute-nodes running Linux microkernels. Initial trials using the default (uninstrumented) version of PETSc and only using Scalasca to instrument *PFLOTRAN* presented no undue difficulty, however, since most of the MPI activity is delegated to PETSc it is desirable in this case to instrument PETSc sources to be able to relate measurements to operations within the library.

Repeating the process to build a fully-instrumented version was a straightforward matter of configuring PETSc using the Scalasca instrumenter as a preposition to the compilers, e.g.,

```
--with-cc="scalasca -instrument cc"
--with-fc="scalasca -instrument ftn"
```

and with `SKIN_MODE=none` set to disable instrumentation during configuration. `SKIN_MODE` was then unset before building PETSc and *PFLOTRAN* itself so that all of their sources were automatically instrumented by the compilers and the Scalasca measurement libraries included when linking (thereby interposing on calls to MPI library routines). No compiler optimizations are (explicitly) disabled by instrumentation with PGI or IBM compilers, and Scalasca instrumented builds don't take notably longer than regular optimized builds (and the resulting executable is only somewhat larger).

3.2 Scalasca measurement & analysis

The Scalasca measurement collection and analysis nexus facilitates both runtime summarization (*summary*) and automatic trace analysis (*trace*) experiments from a single instrumented executable and stores each experiment in a unique measurement archive directory. Environment variables can be set adjust the runtime configuration, e.g., to increase measurement buffer sizes. A runtime filter specifying instrumented routines to ignore during measurement was generated from an initial

Table 1: Comparison of *PFLOTRAN* measurements using PMPI interposition and instrumentation of user source routines by compilers (and after applying runtime filtering of USR routines on non-MPI callpaths).

System	Jugene	Jaguar
Architecture	IBM BG/P	Cray XT5
Compilers	IBM XL	PGI
USR routines	1148 (292)	1137 (254)
Unique callpaths	12728 (1733)	10972 (1377)
Max. frame depth	63 (22)	63 (21)
MPI routines	29	28
MPI callpaths	633	452
Trace buffer content [MB]	3832.4 (26.1)	3819.7 (18.1)
USR routines	3807.5 (0.0)	3802.8 (0.0)
COM routines	11.1	7.6
MPI routines	14.9	10.5

summary experiment on each system: this approach both reduces the size of callpath profiles and the amount of measurement overhead (particularly for frequently executed routines) and thereby significantly improves the quality of subsequent measurements. Table 1 summarizes the measured routines and associated execution callpaths for default, unfiltered measurements and when employing a runtime filter (in parenthesis).

With *PFLOTRAN* and PETSc routines instrumented by the compilers, plus almost 30 routines used from the MPI library, over 1100 instrumented routines were executed. Measurement filters generated listing all routines not on a callpath to MPI (i.e., purely local calculation) removed almost 900 of the instrumented (USR) user source routines for each compiler and reduce the estimated per-process trace buffer content from almost 4 GB to much more reasonable sizes around 20 MB. This still left over one thousand unique callpaths up to 22 frames deep, of which approximately one third were callpaths to MPI routines. Of the set of 399 'flow' callpaths, only 40 were missing from the 'tran' set, 14 had similarly named callpaths (e.g., `richardsjacobian` vs `rtjacobian` and `_MPIAIJ` vs `_MPIBAIJ`) and an additional 20 only were found in 'tran,' showing the considerable structural similarity between the two operations (also apparent in Figure 7).

Times reported by *PFLOTRAN* for summary and trace collections employing runtime filters on Jugene and Jaguar compared with reference times from the uninstrumented executions in Figure 3 show that measurement dilation is generally acceptably small, apart from trace collection with larger configurations of processes. Whereas on Jaguar Cray XT5 the dilation is 100% only for 'flow' at 128k processes, it is much more pronounced in measurements on Jugene IBM BG/P for even 16k processes and grows for 128k processes to 7x! This difference can be attributed to translations of process ranks (from the local rank in the MPI communicator to the global rank in `MPI_COMM_WORLD`)

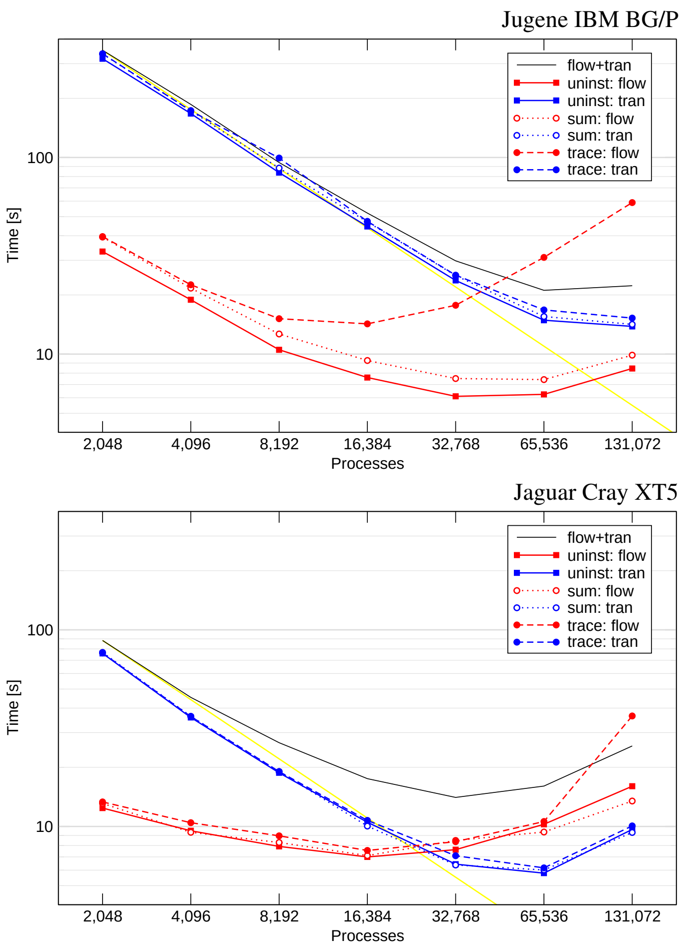


Figure 3: Average simulation timestep durations reported by *PFLOTRAN* for Scalasca 1.3.2 summary and trace experiments, with breakdown into 'flow' and 'tran'(sport) phases, compared to reference uninstrumented executions.

in every communication operation event recorded, and the relative speeds of computation and communication on both systems. Although it requires changing the trace file format, work has been initiated to eliminate this unnecessary translation overhead during measurement.

These filtered Scalasca summary measurements introduce minimal overhead (even with hardware counter metrics included) and require only a short amount of additional time to collate the analysis report at completion, making them convenient for acquiring an overview of execution performance. On the other hand, tracing experiments need to be approached with care. When submitting tracing experiments, it is important to increase the job timelimit of the normal (uninstrumented) application execution with the extra time for trace collection and analysis (plus a reasonable cushion to allow for I/O variability). Figure 4 shows the time for complete *PFLOTRAN* executions, including initialization and finalization, and the associated additional trace collection and analysis times.

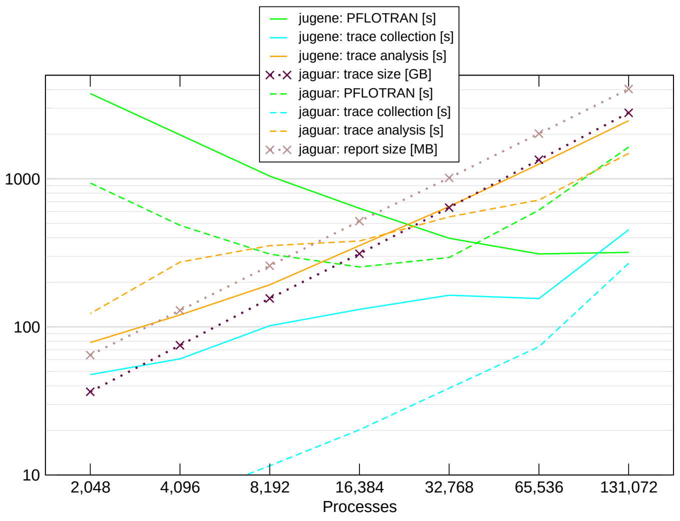


Figure 4: Additional time to collect and analyse Scalasca traces (which increase linearly in size with the number of processes) with complete time of uninstrumented *PFLOTRAN* executions on Jugene IBM BG/P and Jaguar Cray XT5.

Also shown in Figure 4 are the amounts of event trace data, which grow linearly with the number of MPI processes and need to be stored temporarily on an efficient parallel filesystem with sufficient capacity. Traces on Jugene were 47% larger than on Jaguar, due to different IBM and PGI compiler instrumentation approaches, reaching over 4.0 TB compared to 2.7 TB.[1]

After trace collection, Scalasca automatically initiates trace analysis in parallel using the same computational resources to produce analysis reports, which also grow linearly with the number of processes.[2] Performance naturally depends on the filesystem, both I/O bandwidth and the cost of metadata operations, as well as the number of corrections that are required to restore event timestamp consistency on systems which don't provide a globally synchronized clock. Figure 4 shows the best trace collection and analysis times achieved with *PFLOTRAN* on Jugene and Jaguar: since timing was done on service systems with other applications running at the same time, I/O performance can be highly variable. Trace collection times typically scale better than linearly since they can exploit additional I/O resources, with over 92 GB/s achieved on Jaguar (with its Lustre filesystem) and over 40 GB/s on Jugene (with GPFS). Trace analysis time is almost linear on BG/P, however, much less than linear on XT5 where an additional replay of the trace is required to correct timestamps.

Callpath profiles for complete executions of complex applications such as *PFLOTRAN* are large and unwieldy, and with separate profiles contained for each process they grow with scale. Initialization and finalization phases, which contain highly vari-

[1]These traces didn't include hardware counter metrics, which would further increase trace size significantly.

[2]Intermediate reports are post-processed offline to derive additional metrics and also compressed, such that the final reports are almost the same size.

able application file I/O, can be eliminated or report extracts can be generated only for the subtree corresponding to the simulation timesteps, i.e., `stepperrun`. Furthermore, pruning can be employed to remove uninteresting callpaths such as `MPI_Comm_rank` and `MPI_Comm_size`, which constitute over 200 of the 1500 callpaths within `stepperrun`. During post-processing of analysis reports, the three-dimensional grid topology used by *PFLOTRAN* was also incorporated.

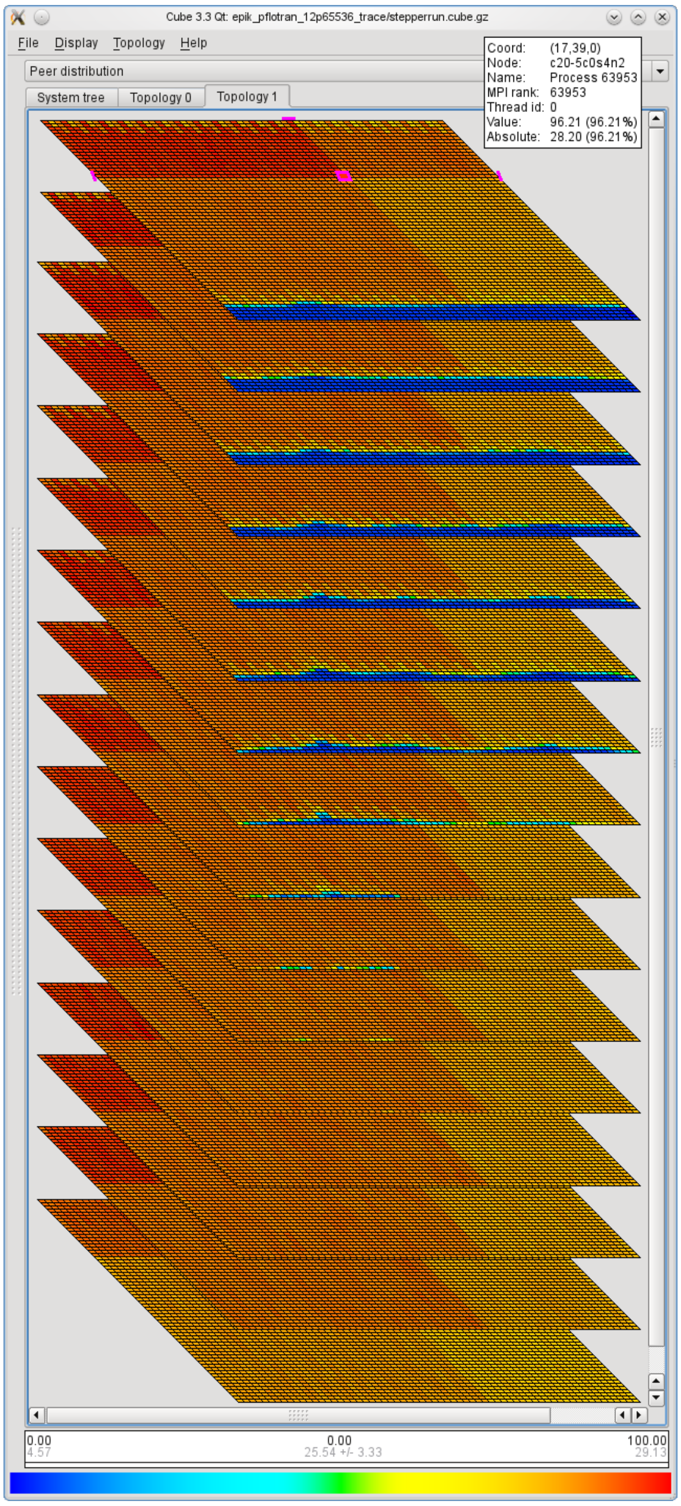
3.3 Scalasca analysis report examination

The Scalasca analysis report explorer presents calculated metrics in a hierarchical tree display equivalent to that representing the tree of the application's executed callpaths, shown in the left and middle panels in Figure 5 for a *PFLOTRAN* experiment with 8,192 processes on Jaguar Cray XT5. Showing all of the MPI processes in a similar system tree is possible, however, it is generally preferable to show them graphically using the application's logical topology, in this case a 32×32×8 three-dimensional grid, as in the right panel.

The upper view shows metrics that can be calculated during measurement with runtime summarization (plus derived metrics such as *Computational imbalance* determined during report post-processing). Values shown in the metric tree are aggregated from the values for each application callpath, which are themselves aggregated over all processes. *PAPI_FP_OPS* is the selected metric, which is a PAPI [20] predefined metric based on hardware counters available on the processor for the number of floating-point operations executed.[3] Since this metric relates to the computational work done by *PFLOTRAN*, it is expected that most of this is to be found in the callpaths of the simulation `stepperrun`, with indeed considerably more in the demanding 'transport' phase than the simpler 'flow' phase. (Following the colour coding in the boxes next to each node, corresponding to the node's metric value on the scale below each panel, the tree of callpaths can be expanded as desired to reveal a next level of callpaths, with the value shown for a node changing from the *inclusive* to *exclusive* value when the node is expanded.) A pronounced imbalance is immediately evident in the distribution of *PAPI_FP_OPS* for each process: the metric value of the process with the largest value defines the scale, and its peers have their metric values shown as a percentage of this maximum.

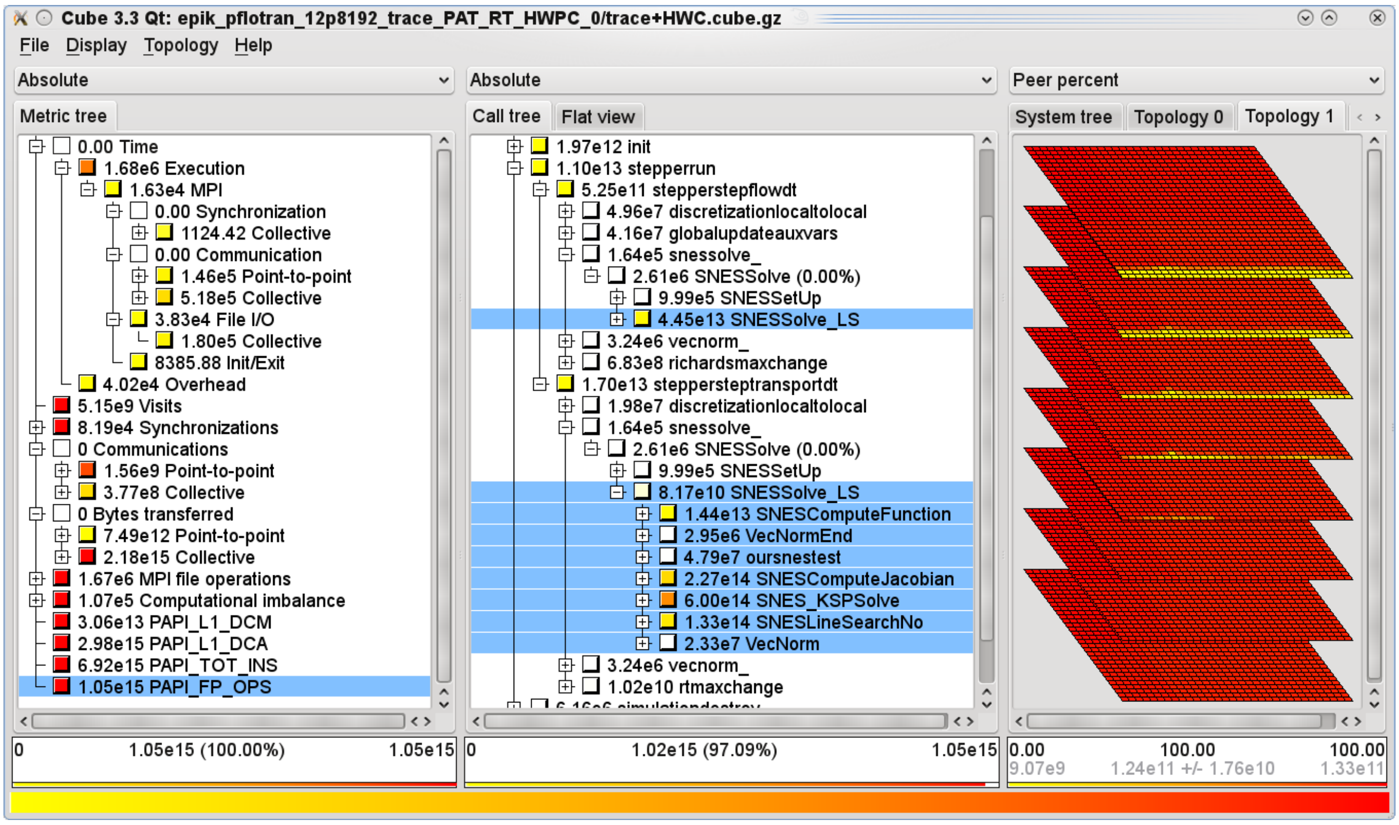
In the lower view, metrics relating to MPI inefficiencies which are calculated by Scalasca automatic trace analysis are selected. They are seen to amount to over 7% of the total execution time for the SNES solves in the 'flow' and 'transport' phases at this scale, where the distribution of metric values complements that of the computation. This imbalance directly relates to the inactive grid cells within the river channel as expected from Figure 1.

[3] *PAPI_FP_OPS* is a single native counter on Cray XT5 Opteron processors, however, on IBM BG/P PowerPC processors it is derived from 13 native counters. Due to system hardware and software limitations, PAPI counters on BG/P can only be used meaningfully in SMP mode (i.e., with a single MPI process per quad-core processor).



```
Coord:      (17,39,0)
Node:       c20-5c0s4n2
Name:       Process 63953
MPI rank:   63953
Thread id:  0
Value:      96.21 (96.21%)
Absolute:   28.20 (96.21%)
```

Figure 6: Distribution of individual process computation times in *PFLOTRAN* `stepperrun` ('flow'+'transport') with 65,536 processes on Jaguar Cray XT5 presented by Scalasca using the application's $64 \times 64 \times 16$ process topology. Processes assigned gridpoints in the river channel are underloaded (blue) compared to the others, whereas processes with coordinates $(x < 18, y < 40)$ in each z-plane are overloaded (red) with extra gridpoints due to the grid decomposition employed.


Cube 3.3 Qt: epik_pflotran_12p8192_trace_PAT_RT_HWPC_0/trace+HWC.cube.gz

File Display Topology Help

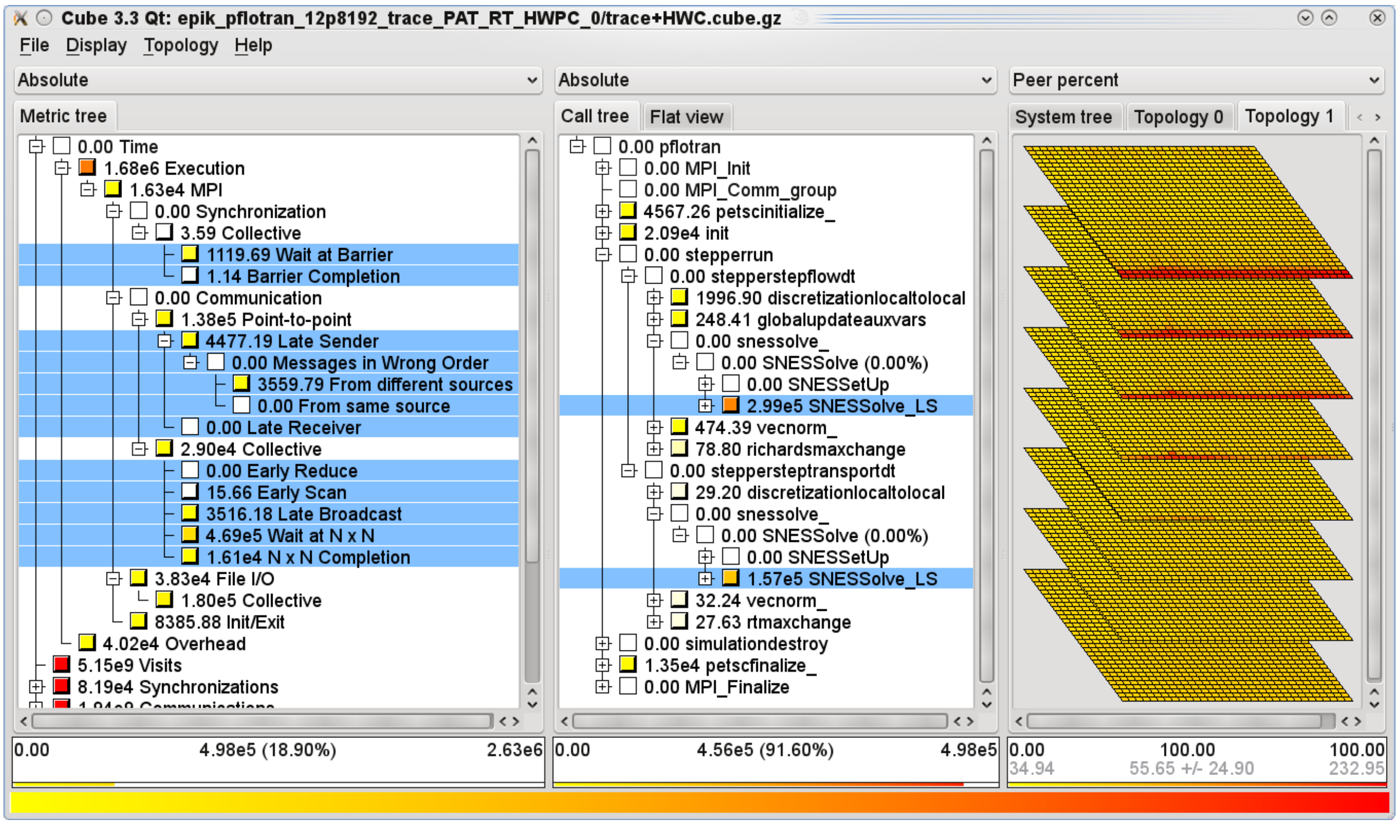
Absolute | Absolute | Peer percent

Metric tree
```
0.00 Time
  1.68e6 Execution
    1.63e4 MPI
      0.00 Synchronization
        1124.42 Collective
      0.00 Communication
        1.46e5 Point-to-point
        5.18e5 Collective
      3.83e4 File I/O
        1.80e5 Collective
      8385.88 Init/Exit
  4.02e4 Overhead
5.15e9 Visits
8.19e4 Synchronizations
0 Communications
  1.56e9 Point-to-point
  3.77e8 Collective
0 Bytes transferred
  7.49e12 Point-to-point
  2.18e15 Collective
1.67e6 MPI file operations
1.07e5 Computational imbalance
3.06e13 PAPI_L1_DCM
2.98e15 PAPI_L1_DCA
6.92e15 PAPI_TOT_INS
1.05e15 PAPI_FP_OPS
```

Call tree | Flat view
```
1.97e12 init
1.10e13 stepperrun
  5.25e11 stepperstepflowdt
    4.96e7 discretizationlocaltolocal
    4.16e7 globalupdateauxvars
    1.64e5 snessolve_
      2.61e6 SNESSolve (0.00%)
        9.99e5 SNESSetUp
        4.45e13 SNESSolve_LS
    3.24e6 vecnorm_
    6.83e8 richardsmaxchange
  1.70e13 steppersteptransportdt
    1.98e7 discretizationlocaltolocal
    1.64e5 snessolve_
      2.61e6 SNESSolve (0.00%)
        9.99e5 SNESSetUp
        8.17e10 SNESSolve_LS
          1.44e13 SNESComputeFunction
          2.95e6 VecNormEnd
          4.79e7 oursnestest
          2.27e14 SNESComputeJacobian
          6.00e14 SNES_KSPSolve
          1.33e14 SNESLineSearchNo
          2.33e7 VecNorm
    3.24e6 vecnorm_
    1.02e10 rtmaxchange
```

System tree | Topology 0 | Topology 1

```
0        1.05e15 (100.00%)     1.05e15
0        1.02e15 (97.09%)      1.05e15
0.00     100.00                100.00
9.07e9   1.24e11 +/- 1.76e10   1.33e11
```


Cube 3.3 Qt: epik_pflotran_12p8192_trace_PAT_RT_HWPC_0/trace+HWC.cube.gz

File Display Topology Help

Absolute | Absolute | Peer percent

Metric tree
```
0.00 Time
  1.68e6 Execution
    1.63e4 MPI
      0.00 Synchronization
        3.59 Collective
          1119.69 Wait at Barrier
          1.14 Barrier Completion
      0.00 Communication
        1.38e5 Point-to-point
          4477.19 Late Sender
            0.00 Messages in Wrong Order
              3559.79 From different sources
              0.00 From same source
          0.00 Late Receiver
        2.90e4 Collective
          0.00 Early Reduce
          15.66 Early Scan
          3516.18 Late Broadcast
          4.69e5 Wait at N x N
          1.61e4 N x N Completion
      3.83e4 File I/O
        1.80e5 Collective
      8385.88 Init/Exit
  4.02e4 Overhead
5.15e9 Visits
8.19e4 Synchronizations
```

Call tree | Flat view
```
0.00 pflotran
  0.00 MPI_Init
  0.00 MPI_Comm_group
  4567.26 petscinitialize_
  2.09e4 init
  0.00 stepperrun
    0.00 stepperstepflowdt
      1996.90 discretizationlocaltolocal
      248.41 globalupdateauxvars
      0.00 snessolve_
        0.00 SNESSolve (0.00%)
          0.00 SNESSetUp
          2.99e5 SNESSolve_LS
      474.39 vecnorm_
      78.80 richardsmaxchange
    0.00 steppersteptransportdt
      29.20 discretizationlocaltolocal
      0.00 snessolve_
        0.00 SNESSolve (0.00%)
          0.00 SNESSetUp
          1.57e5 SNESSolve_LS
      32.24 vecnorm_
      27.63 rtmaxchange
  0.00 simulationdestroy
  1.35e4 petscfinalize_
  0.00 MPI_Finalize
```

System tree | Topology 0 | Topology 1

```
0.00     4.98e5 (18.90%)       2.63e6
0.00     4.56e5 (91.60%)       4.98e5
0.00     100.00                100.00
34.94    55.65 +/- 24.90       232.95
```

Figure 5: Scalasca analysis report explorer presentations of *PFLOTRAN* experiment with 8,192 processes on Jaguar Cray XT5, combining runtime summarization and automatic event trace analysis, showing that imbalance in computation (as defined by the PAPI_FP_OPS hardware counter metric in upper view) complements MPI waiting times (the selected metrics calculated from the trace in lower view).

Computational load imbalance becomes more significant at larger scales, as seen in Figure 6 showing the distribution of `stepperrun` execution times for 65,536 processes arranged in a $64 \times 64 \times 16$ grid. The processes assigned inactive grid points within the river channel clearly have much less computation, however, there are relatively few of them. Also evident is overloading of processes with grid coordinates $(x < 18, y < 40)$ resulting from the excess grid-points in the x and y dimensions: while the 160 grid-points in z are assigned equally to the 16 planes, there are (850/64=)18 extra grid-points in x and (1000/64=)40 extra grid-points in y.

While it would be relatively simple to reverse the assignment of grid-points to processes in the x-dimension, such that extra grid-points were more likely to be assigned to processes with inactive grid-points, the expected benefit would be small since some z-planes have no inactive grid-points. On the other hand, a perfect distribution would assign 2075.2 grid-points to each process, whereas the current distribution assigns 2240 grid-points to 20% of the processes, for an 8% computation overload that manifests as waiting times in subsequent collective communication, e.g., `MPI_Allreduce`. Using a balanced regular distribution would therefore be a significant benefit (particularly at larger scales), without taking into account the (dataset-specific) inactive grid-points.

3.4 Complementary analyses and visualizations

Scalasca analysis reports and trace data use open file formats and libraries are provided for reading and writing which facilitates interoperability with third-party tools. While analysis reports can grow to several gigabytes in size, they can still be transferred to desktop or other local systems for more convenient interactive examination, however, with traces growing to terabytes this is usually not a viable option.

Analysis reports can be imported into TAU/ParaProf [2], which offers a callgraph display seen in Figure 7 as well as an extensive range of profile visualizations as shown in Figure 8. Since ParaProf was unable to load entire analysis reports on a system with 21 GB of node memory, even at modest scale it was necessary to extract only the main simulation section (`stepperrun`) into a separate report (using one of the report algebra utilities provided with Scalasca).

Merging and converting event trace data files is often impractical or prohibitively costly, but smaller measurements can be converted for visualization and analysis. Alternatively, current versions of the Vampir tools [3] are able to directly read and visualize distributed event traces generated by Scalasca without conversion. Using VampirServer running on 512 cores to interactively examine a Scalasca event trace of the execution of 8,192 *PFLOTRAN* processes, Figure 9 shows the imbalance in *MPI_Allreduce* on some processes and the detailed communication behaviour within iterations in the 'flow' and 'transport' phases of each simulation timestep.

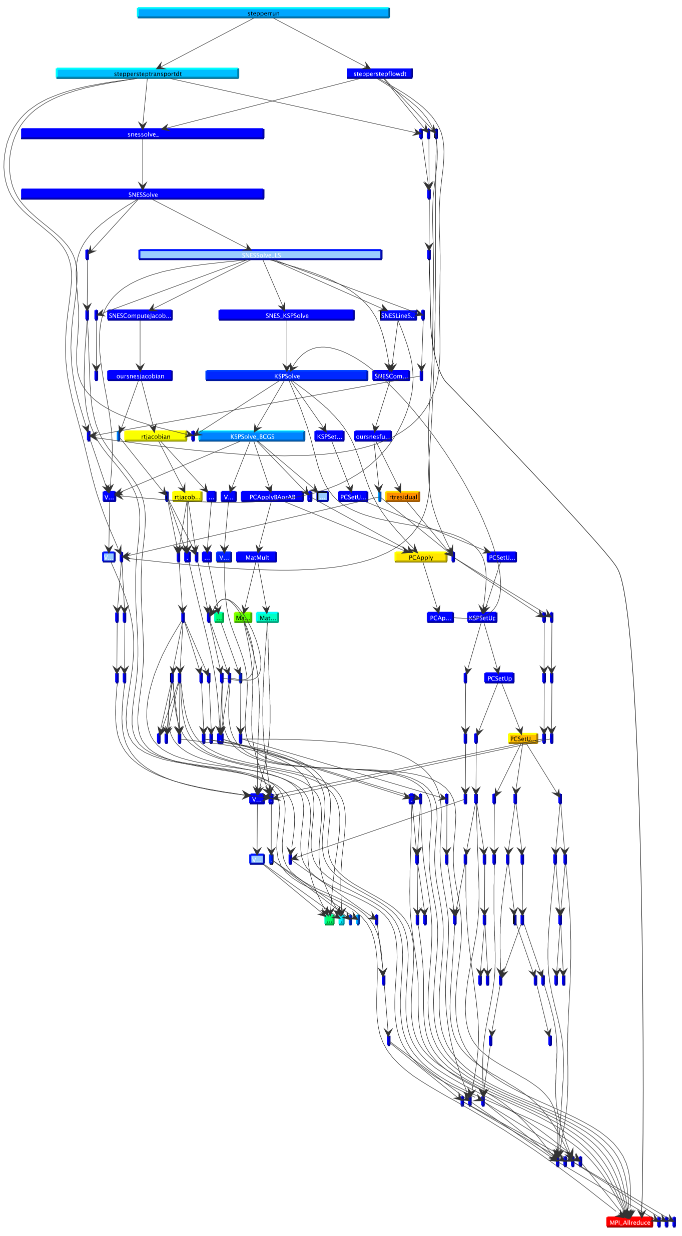
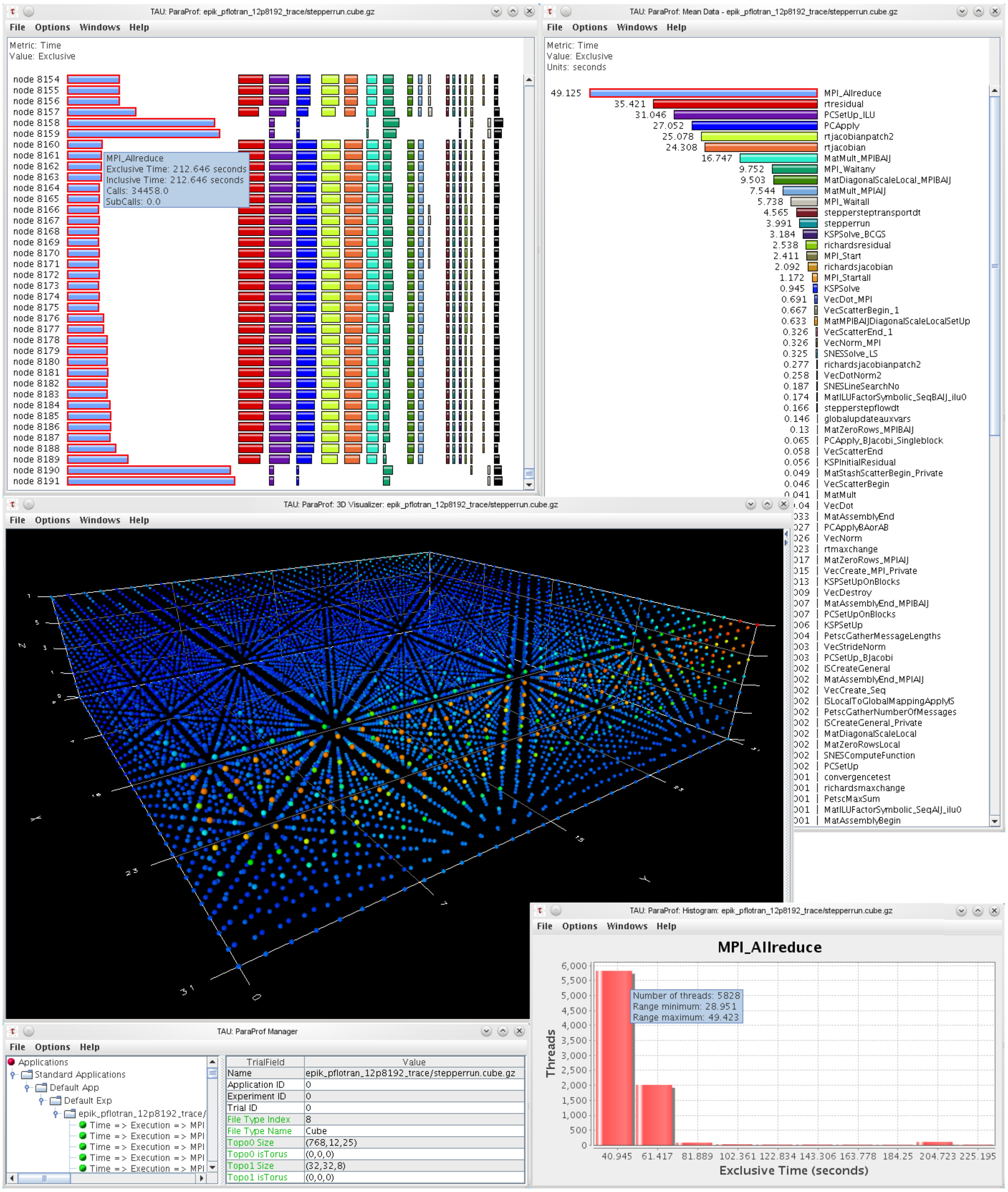


Figure 7: TAU/ParaProf presentation of *PFLOTRAN* "2B" callgraph with 8,192 processes on Jaguar Cray XT5, showing execution time on paths from `stepperrun` to `MPI_Allreduce` and other (non-pruned) MPI operations.

4 Scalasca scalability improvement

As part of measurement, Scalasca creates definition records on each process for each routine, callpath, MPI communicator, etc., and at measurement completion it unifies these into a consistent global set of definitions plus associated mappings for each process [7]. The number of definitions depends on how many instances of each object are encountered during measurement, and the total size depends on the lengths of names and sizes of communicators. While the routines executed typically remain unchanged with scale, larger numbers of processes often result



MPI_Allreduce
Exclusive Time: 212.646 seconds
Inclusive Time: 212.646 seconds
Calls: 34458.0
SubCalls: 0.0

TAU: ParaProf: Mean Data - epik_pflotran_12p8192_trace/stepperrun.cube.gz
File Options Windows Help

Metric: Time
Value: Exclusive
Units: seconds

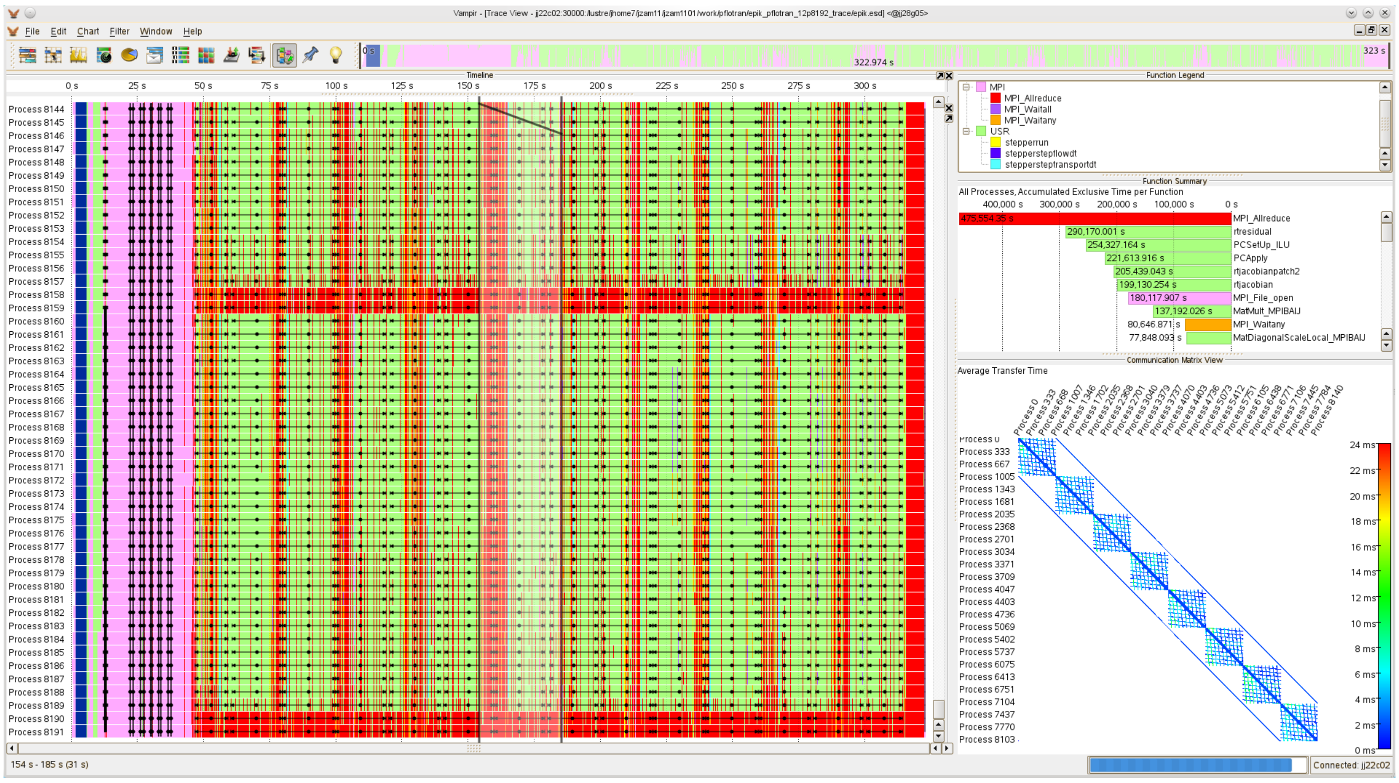
```
49.125  MPI_Allreduce
35.421  rtresidual
31.046  PCSetUp_ILU
27.052  PCApply
25.078  rtjacobianpatch2
24.308  rtjacobian
16.747  MatMult_MPIBAIJ
 9.752  MPI_Waitany
 9.503  MatDiagonalScaleLocal_MPIBAIJ
 7.544  MatMult_MPIAIJ
 5.738  MPI_Waitall
 4.565  steppersteptransportdt
 3.991  stepperrun
 3.184  KSPSolve_BCGS
 2.538  richardsresidual
 2.411  MPI_Start
 2.092  richardsjacobian
 1.172  MPI_Startall
 0.945  KSPSolve
 0.691  VecDot_MPI
 0.667  VecScatterBegin_1
 0.633  MatMPIBAIJDiagonalScaleLocalSetUp
 0.326  VecScatterEnd_1
 0.326  VecNorm_MPI
 0.325  SNESSolve_LS
 0.277  richardsjacobianpatch2
 0.258  VecDotNorm2
 0.187  SNESLineSearchNo
 0.174  MatILUFactorSymbolic_SeqBAIJ_ilu0
 0.166  stepperstepflowdt
 0.146  globalupdateauxvars
  0.13  MatZeroRows_MPIBAIJ
 0.065  PCApply_BJacobi_Singleblock
 0.058  VecScatterEnd
 0.056  KSPInitialResidual
 0.049  MatStashScatterBegin_Private
 0.046  VecScatterBegin
 0.041  MatMult
   .04  VecDot
   033  MatAssemblyEnd
   027  PCApplyBAorAB
   026  VecNorm
   023  rtmaxchange
   017  MatZeroRows_MPIAIJ
   015  VecCreate_MPI_Private
   013  KSPSetUpOnBlocks
   009  VecDestroy
   007  MatAssemblyEnd_MPIBAIJ
   007  PCSetUpOnBlocks
   006  KSPSetUp
   004  PetscGatherMessageLengths
   003  VecStrideNorm
   003  PCSetUp_BJacobi
   002  ISCreateGeneral
   002  MatAssemblyEnd_MPIAIJ
   002  VecCreate_Seq
   002  ISLocalToGlobalMappingApplyIS
   002  PetscGatherNumberOfMessages
   002  ISCreateGeneral_Private
   002  MatDiagonalScaleLocal
   002  MatZeroRowsLocal
   002  SNESComputeFunction
   002  PCSetUp
   001  convergencetest
   001  richardsmaxchange
   001  PetscMaxSum
   001  MatILUFactorSymbolic_SeqAIJ_ilu0
   001  MatAssemblyBegin
```

TAU: ParaProf: 3D Visualizer: epik_pflotran_12p8192_trace/stepperrun.cube.gz
File Options Windows Help
Z
X
Y
23
31
0
15
7

Number of threads: 5828
Range minimum: 28.951
Range maximum: 49.423

TrialField	Value
Name	epik_pflotran_12p8192_trace/stepperrun.cube.gz
Application ID	0
Experiment ID	0
Trial ID	0
File Type Index	8
File Type Name	Cube
Topo0 Size	(768,12,25)
Topo0 isTorus	(0,0,0)
Topo1 Size	(32,32,8)
Topo1 isTorus	(0,0,0)

Figure 8: TAU/ParaProf presentations of Scalasca trace analysis report from *PFLOTRAN* execution with 8,192 processes on Jaguar Cray XT5. From the diverse graphical profiles of routine execution times, `MPI_Allreduce` is clearly dominant and has significant imbalance.


Vampir - [Trace View - jj22c02:30000:/lustre/jhome7/jzam11/jzam1101/work/pflotran/epik_pflotran_12p8192_trace/epik.esd] <@jj28g05>
File Edit Chart Filter Window Help
Timeline
Function Legend
MPI
MPI_Allreduce
MPI_Waitall
MPI_Waitany
USR
stepperrun
stepperstepflowdt
stepperstepransportdt
Function Summary
All Processes, Accumulated Exclusive Time per Function
475,554.35 s MPI_Allreduce
290,170.001 s rtresidual
254,327.164 s PCSetUp_ILU
221,613.916 s PCApply
205,439.043 s rtjacobianpatch2
199,130.254 s rtjacobian
180,117.907 s MPI_File_open
137,192.026 s MatMult_MPIBAIJ
80,646.871 s MPI_Waitany
77,848.093 s MatDiagonalScaleLocal_MPIBAIJ
Communication Matrix View
Average Transfer Time
154 s - 185 s (31 s)
Connected: jj22c02

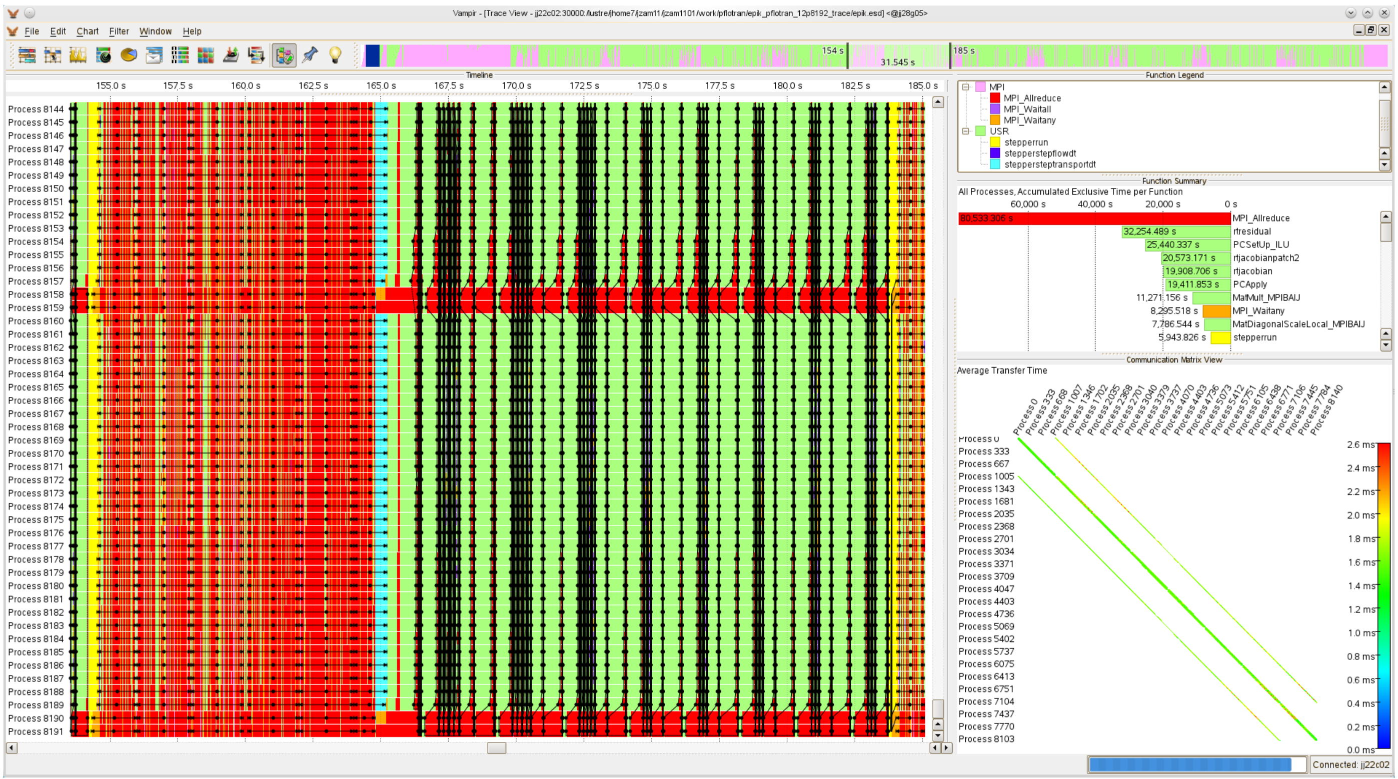

Vampir - [Trace View - jj22c02:30000:/lustre/jhome7/jzam11/jzam1101/work/pflotran/epik_pflotran_12p8192_trace/epik.esd] <@jj28g05>
File Edit Chart Filter Window Help
Timeline
Function Legend
MPI
MPI_Allreduce
MPI_Waitall
MPI_Waitany
USR
stepperrun
stepperstepflowdt
stepperstepransportdt
Function Summary
All Processes, Accumulated Exclusive Time per Function
80,533.306 s MPI_Allreduce
32,254.489 s rtresidual
25,440.337 s PCSetUp_ILU
20,573.171 s rtjacobianpatch2
19,908.706 s rtjacobian
19,411.853 s PCApply
11,271.156 s MatMult_MPIBAIJ
8,295.518 s MPI_Waitany
7,786.544 s MatDiagonalScaleLocal_MPIBAIJ
5,943.826 s stepperrun
Communication Matrix View
Average Transfer Time
Connected: jj22c02

Figure 9: Vampir visualizations of a Scalasca event trace of *PFLOTRAN* execution with 8,192 processes on Jaguar Cray XT5. From the global timeline view (top) the initialization phase and ten simulation timesteps can be identified, along with the serious imbalance in `MPI_Allreduce` on groups of processes. An interactive zoom into the fourth timestep (bottom) distinguishes its particularly slow 'flow' phase (with solver 4 iterations between yellow and cyan events) followed by the 'transport' phase (with solver 8 iterations between cyan and yellow events) and their distinct patterns of MPI point-to-point and collective communication. Further zooming and scrolling allows examination of finer detail and individual events in the trace.

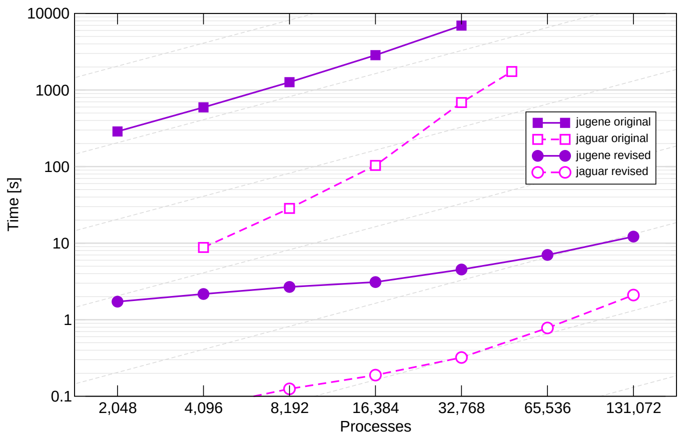


Figure 10: Time to unify *PFLOTRAN* identifier definitions (and write them to disk with associated mappings for each process) on Jugene IBM BG/P and Jaguar Cray XT5, comparing original and revised Scalasca implementations.

in larger communicator definitions (since they must denote the contained ranks).

The resulting trees of *PFLOTRAN* callpaths are substantial, both in their measurement and subsequent analysis requirements. Scalasca analysis reports for larger numbers of processes are very large and unwieldy, requiring 64-bit versions of processing tools and GUI, lots of RAM and patience. The latter limitations derive from storing only exclusive metric values in analysis reports, such that the entire data must be loaded to calculate the total values of each metric. On-going work based on storing and selectively reading metrics as inclusive values is expected to make interactive analysis practical in future [6].

Although MPI communicator definitions are not needed for Scalasca summary analyses, they are required to be able to replay and analyse event traces, and the extensive use of MPI communicators by PETSc and HDF5 proved to be much more demanding particularly at scale. 18 duplicates of `MPI_COMM_WORLD` and 4 copies of `MPI_COMM_SELF` communicators were created during "2B" test case executions. On Jaguar with 128k processes, this resulted in 13% of the total execution time spent in collective calls to `MPI_Comm_dup` due to interspersed file I/O!

For MPI communicators that are duplicates of `MPI_COMM_WORLD` (which contains all ranks) definition records grow linearly with the number of processes. For `MPI_COMM_SELF` (which only contains a single rank), each definition is a small fixed size, however, when defined for every process the total number also grows linearly. The original Scalasca implementation of MPI communicator definitions and their unification therefore quickly resulted in gigabytes of communicator definition records, such that trace analysis was not possible for more than 48k processes. Along with the exponential growth in size, unification times of the original implementation were also unacceptable as seen in Figure 10.

Scalasca definition and unification of MPI communicators (and associated MPI groups) were re-designed for the 1.3.2 release to address these two deficiences [6]. Crucially for *PFLOTRAN* and codes using `MPI_COMM_SELF`, special encoding was used to distinguish this communicator and duplicates of it, and handling during communicator unification updated to recognise this (and avoid retaining separate definitions for each process rank). Unification of all types of definition records was also implemented in a hierarchical binary tree fashion for improved scalability over the original serialised approach. With versions of Scalasca using the revised implementation, Figure 10 shows that *PFLOTRAN* unification times were reduced more than 1500-fold for measurements with 32k processes, and now takes only a few seconds even for 128k processes.

5 Conclusions

The outcome of a series of Extreme Scaling Workshops hosted by Jülich Supercomputing Centre has been the conclusion that many application codes from a variety of disciplines are able to deliver breakthrough science by effectively exploiting hundreds of thousands of cores [16]. On the way to this rather remarkable result, however, the codes naturally had to be comprehensively debugged and execution performance analyzed at each increase in scale, to be able to identify, localize and remedy a series of scalability bottlenecks.

Performance analysis of complex applications executing at large-scale requires care. Full automatic instrumentation of an application like *PFLOTRAN* and its associated libraries (PETSc) is convenient, but produces more detail and measurement overhead than desirable. Selective instrumentation and/or measurement filtering must be used to reduce measurement overhead and sizes of callpath profiles and event traces to managable levels. Even basic execution callpath profiles for many thousands of processes rapidly become awkwardly large, and powerful analyses and interactive visualizations are required for an effective initial overview leading to in-depth refinement of performance issues. Application developers and analysts therefore often require training and coaching in the use of available tools with their HPC applications, such as offered by regular international hands-on tuning workshops of the Virtual Institute – High-Productivity Supercomputing [21].

While the Scalasca toolset has demonstrated its effectiveness with numerous applications on current HPC platforms including Cray XT/XE and IBM BlueGene, significant challenges are presented by the combination of extreme scale and growing application complexity. The highly-scalable *PFLOTRAN* code, where MPI usage is encapsulated by PETSc and HDF5 libraries, is one such manifestation (but far from exceptional). Its usage of dozens of (duplicate) MPI communicators from deep within object-oriented libraries required a fundamental redesign and new implementation of communicator management by Scalasca, leading to much improved measurement scalability.

Acknowledgements

This work would not have been possible without the generous assistance of the *PFLOTRAN* developers, in particular Glenn Hammond (Pacific Northwest National Laboratory), Peter Lichner (Los Alamos National Laboratory) and Richard Mills (Oak Ridge National Laboratory). It was supported by allocations of advanced computing resources on the *Jugene* IBM Blue Gene/P of Jülich Supercomputing Centre at Forschungszentrum Jülich and the *Jaguar* Cray XT5 of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the US Department of Energy under contract DE-AC05-00OR22725.

References

[1] Scalasca toolset for scalable performance analysis of large-scale parallel applications. `http://www.scalasca.org/`.

[2] TAU. `http://www.cs.uoregon.edu/research/tau/`.

[3] Vampir. `http://www.vampir.eu/`.

[4] ANL, LANL, ORNL, PNNL & UIUC. PFLOTRAN. `http://software.lanl.gov/pflotran/`.

[5] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushlik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang. PETSc: Portable, extensible toolkit for scientific computation. `http://www.mcs.anl.gov/petsc/`.

[6] M. Geimer, P. Saviankou, A. Strube, Z. Szebenyi, F. Wolf, and B. J. N. Wylie. Further improving the scalability of the Scalasca toolset. In *Proc. PARA2010 (Reykjavík, Iceland)*, Lecture Notes in Computer Science. Springer. (to appear).

[7] M. Geimer, F. Wolf, B. J. N. Wylie, E. Ábrahám, D. Becker, and B. Mohr. The Scalasca performance toolset architecture. *Concurrency and Computation: Practice and Experience*, 22(6):702–719, Apr. 2010.

[8] G. E. Hammond and P. C. Lichtner. Cleaning up the Cold War: Simulating uranium migration at the Hanford 300 Area. In *Proc. Scientific Discovery through Advanced Computing (SciDAC, Chattanooga, TN, USA)*, Journal of Physics: Conference Series. IOP Publishing, July 2010.

[9] G. E. Hammond, P. C. Lichtner, R. T. Mills, and C. Lu. Toward petascale computing in geosciences: Application to the Hanford 300 Area. In *Proc. Scientific Discovery through Advanced Computing (SciDAC, Seattle, WA, USA)*, volume 125 of *Journal of Physics: Conference Series*, page 012051. IOP Publishing, July 2008.

[10] Innovative and Novel Computational Impact on Theory and Experiment (INCITE). `http://science.energy.gov/ascr/facilities/incite/`.

[11] Jülich Supercomputing Centre. PEPC: A multi-purpose parallel tree-code. `http://www2.fz-juelich.de/jsc/pepc/`.

[12] J. Labarta, B. P. Miller, B. Mohr, and M. Schulz, editors. *Program Development for Extreme-scale Computing*, number 10181 in Dagstuhl Seminar Proceedings. Schloss Dagstuhl, Leibniz-Zentrum für Informatik, Germany, May 2010. `http://www.dagstuhl.de/10181`.

[13] R. T. Mills, C. Lu, P. C. Lichtner, and G. E. Hammond. Simulating subsurface flow and transport on ultrascale computers using PFLOTRAN. In *Proc. Scientific Discovery through Advanced Computing (SciDAC, Boston, MA, USA)*, volume 78 of *Journal of Physics: Conference Series*. IOP Publishing, June 2007.

[14] R. T. Mills, V. Sripathi, G. Mahinthakumar, G. E. Hammond, P. C. Lichtner, and B. F. Smith. Experiences and challenges scaling PFLOTRAN, a PETSc-based code for subsurface reactive flow simulations, towards the petascale on Cray XT systems. In *Proc. 51st CUG meeting (Atlanta, GA, USA)*. Cray User Group, Inc., May 2009.

[15] R. T. Mills, V. Sripathi, G. Mahinthakumar, G. E. Hammond, P. C. Lichtner, and B. F. Smith. Engineering PFLOTRAN for scalable performance on Cray XT and IBM BlueGene architectures. In *Proc. Scientific Discovery through Advanced Computing (SciDAC, Chattanooga, TN, USA)*, Journal of Physics: Conference Series. IOP Publishing, July 2010.

[16] B. Mohr and W. Frings, editors. *Jülich Blue Gene/P Extreme Scaling Workshop*. FZJ-JSC-IB reports 2010-02, 2010-03 & 2011-02, Jülich Supercomputing Centre, 2009, 2010 & 2011. `http://www2.fz-juelich.de/jsc/bg-ws11/`.

[17] Performance Engineering Research Institute (PERI). `http://www.peri-scidac.org/`.

[18] Scientific Discovery through Advanced Computing (SciDAC). `http://www.scidac.gov/`.

[19] Secure Water Resources Systems Research Group. PFLOTRAN Tiger Team instructions. `http://secure-water.org/wiki/`.

[20] UTK. Performance Application Programming Interface (PAPI). `http://icl.cs.utk.edu/papi/`.

[21] Virtual Institute – High-Productivity Supercomputing (VI-HPS). `http://www.vi-hps.org/`.

[22] B. J. N. Wylie, D. Böhme, W. Frings, M. Geimer, B. Mohr, Z. Szebenyi, D. Becker, M.-A. Hermanns, and F. Wolf. Scalable performance analysis of large-scale parallel applications on Cray XT systems with Scalasca. In *Proc. 52nd CUG meeting (Edinburgh, Scotland)*. Cray User Group, Inc., May 2010.

[23] B. J. N. Wylie, M. Geimer, B. Mohr, D. Böhme, Z. Szebenyi, and F. Wolf. Large-scale performance analysis of Sweep3D with the Scalasca toolset. *Parallel Processing Letters*, 20(4):397–414, December 2010.

About the authors

Brian J. N. Wylie and Markus Geimer are research scientists in the team developing and supporting the Scalasca toolset at Jülich Supercomputing Centre of Forschungszentrum Jülich in Germany. They both provide hands-on training and coaching with parallel performance analysis tools under auspices of the Virtual Institute – High-Productivity Supercomputing. E-mail: `b.wylie@fz-juelich.de`, `m.geimer@fz-juelich.de`.

Scalasca is developed in a collaboration led by Bernd Mohr at Forschungszentrum Jülich and Felix Wolf at the German Research School for Simulation Sciences. Software and documentation can be found at `www.scalasca.org` and the development team can be reached at `scalasca@fz-juelich.de`.