Scaling Performance Tool MPI Communicator Management

Markus Geimer[1], Marc-André Hermanns[2], Christian Siebert[2], Felix Wolf[1,2,3], and Brian J. N. Wylie[1]

[1] Jülich Supercomputing Centre, Forschungszentrum Jülich, Germany
{m.geimer,b.wylie}@fz-juelich.de

[2] German Research School for Simulation Sciences, Aachen, Germany
{m.a.hermanns,c.siebert,f.wolf}@grs-sim.de

[3] RWTH Aachen University, Aachen, Germany

Abstract. The Scalasca toolset has successfully demonstrated measurement and analysis scalability on the largest computer systems, however, applications have growing complexity and increasing demands on performance tools. One such application is the *PFLOTRAN* code for simulating multiphase subsurface flow and reactive transport. While *PFLOTRAN* itself and Scalasca runtime summarization both scale well, MPI communicator management becomes critical for trace collection with tens of thousands of processes. Re-design and re-engineering of key components of the Scalasca measurement system are presented which encompass the representation of communicators, communicator definition tracking and unification, and translation of ranks recorded in event traces.

Keywords: MPI communicators, performance measurement tools, scalability

1 Introduction

Scalasca is an open-source toolset for analyzing the execution behavior of applications based on the MPI and/or OpenMP parallel programming interfaces supporting a wide range of current HPC platforms [7,9]. It combines compact runtime summaries, that are particularly suited for obtaining an overview of execution performance, with in-depth analysis 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 and consisting of unprecedented numbers of processes [17].

Experience with a growing number of HPC applications on leadership IBM Blue Gene and Cray XT systems has shown that they often scale surprisingly well to effectively exploit hundreds of thousands of processor cores [11]. Many codes explicitly use MPI for communication and synchronization, whereas others make extensive use of libraries that encapsulate MPI usage. An example of the latter, the *PFLOTRAN* three-dimensional reservoir simulator [2] has featured prominently in the US Department of Energy SciDAC program, where it

Published in "Proc. of EuroMPI 2011", LNCS 6960, pp. 178–187, Springer, 2011.
The original publication is available at www.springerlink.com

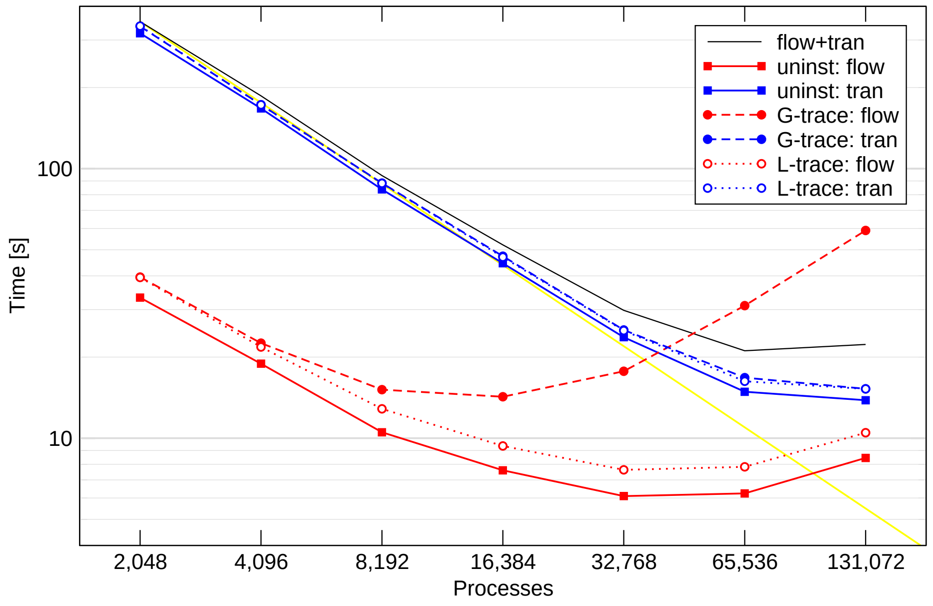


Fig. 1. Average simulation timestep durations reported by *PFLOTRAN* for Scalasca trace experiments on IBM BG/P, with breakdown into 'flow' and 'tran'(sport) phases, compared to reference uninstrumented executions. Distortion in the trace with event communication ranks translated to global ranks (G-trace) is avoided in the trace that records the local ranks (L-trace).

has been used to simulate geologic CO_2 sequestration and migration of radionucleide contaminants in groundwater [8]. Recent measurement and analysis of *PFLOTRAN* execution performance with a 'petascale' dataset on IBM BG/P and Cray XT5 systems with Scalasca [16] identified significant performance opportunities in the application, but also several serious scalability issues with the Scalasca measurement approach that needed to be resolved to produce viable performance analyses. Figure 1 shows the strong scaling of *PFLOTRAN* simulation timesteps on the Jugene BG/P at Jülich Supercomputing Centre along with corresponding time for Scalasca summary and trace experiments, including breakdown of 'flow' and 'tran'(sport) execution phases.

With the provided '2B' test case, *PFLOTRAN* (via the HDF5 and PETSc libraries) was found to create 18 copies of the `MPI_COMM_WORLD` global communicator and 4 copies of `MPI_COMM_SELF` on each process. For Scalasca runtime summarization experiments, MPI communicators are ignored, however, for parallel trace analysis it is necessary to record communicator definitions and their usage in MPI communication and synchronization event records (to allow communicators to be reconstructed and used in replaying trace events). The prior implementation of communicator management proved to be inadequate, requiring storage space and processing time that grew linearly or worse with the number of processes, such that collection and analysis of large-scale *PFLOTRAN* traces was not possible.[4] Furthermore, dilation of application execution time during trace collection was found to be severe for the 'flow' phase (as shown in Fig. 1) due to the cost of translating local to global ranks.

To address these issues, we re-designed and re-engineered communicator management and representation for the Scalasca measurement system as described in the remainder of this paper. We start our discussion in Sect. 2 with a review of related work, followed by a description and analysis of the original communicator handling in Sect. 3. Section 4 then discusses the improved data layout and algorithms in detail. Next, in Sect. 5, we show an experimental evaluation of our approach with respect to various key metrics, before concluding the paper in Sect. 6.

2 Related Work

The data that a measurement tool needs to collect and store depends on the analyses that are intended. Even for tools serving similar purposes, communicator management and rank translation can be done very differently, as demonstrated by a brief survey of current open-source software releases. mpiP-3.3 [10] doesn't use communicator recording or rank translation since it doesn't distinguish these in its profile analysis. Periscope-1.3.2 [13] similarly doesn't need to store communicators or translate ranks for its on-line communication analysis. FPMPI-2.1g [3] profiles do provide a matrix of point-to-point communication sources and destinations, however, only in terms of local ranks without distinguishing communicators. For its communication matrix TAU-2.20.2 [15] translates point-to-point source and destination ranks to global ranks during measurement, and it can also distinguish by communicator. Translated ranks also appear in TAU traces of point-to-point communications, but not for the roots of collective communications, which is also the approach adopted by Extrae-2.1.1 [5]. While communicators are distinguished, the communicator composition is neither recorded nor part of their analysis. VampirTrace-5.11 [12], like the Scalasca predecessor from which it derives, translates ranks of both point-to-point and collective communication events. These tools convert local to global ranks using the standard provided `MPI_Group_translate_ranks` routine as communication events are handled during measurement, with shortcuts to avoid unnecessary rank translation for communicators that are identical or congruent to `MPI_COMM_WORLD`. In comparison, the MPE logger provided with MPICH2-1.4 [4] writes traces entirely with local (untranslated) ranks, which are translated when traces are read using communicator rank mappings recorded separately for each communicator and rank.

[4] Notably the amount of trace event data collected, which is often an impediment, was not a limitation for Scalasca in this case.

3 Original Scalasca Scheme

In the original scheme used by Scalasca, each MPI group and communicator was represented by a bitstring where bit i indicates whether the global rank i is part of the group or communicator (=1) or not (=0). Additional fields in the record distinguished between the two types (i.e., group or communicator) and assigned a process-local numerical identifier used by communication events to refer to this definition. As such, multiple distinct communicators required the storage of the full bitstring, even if they comprise the same group of processes. Each PMPI wrapper function creating a new group or communicator determined this bitstring by calling `MPI_Group_translate_ranks` to map the group or the group of the newly created communicator, respectively, onto the group of `MPI_COMM_WORLD` and then setting the corresponding bits. Since in this scheme communicators are defined in terms of global ranks, all events generated for MPI communication operations need to use global rank information as well to allow for a proper determination of sender, receiver or collective root processes. This required another call to `MPI_Group_translate_ranks` in the PMPI wrapper of each communication function to convert the local rank in the communicator provided as arguments into the corresponding global rank, unless the communicator is `MPI_COMM_WORLD` (i.e., the ranks are already global).

To establish a global view, these per-process communicator definitions were "unified" at the end of measurement. That is, communicator definitions from different processes were merged to create a unique set of global communicator definitions, requiring some complicated logic to correctly distinguish between multiple copies of a communicator. Moreover, a per-process mapping from local to global communicator identifiers was created, which could be applied to the corresponding identifiers stored in the communication events while reading the trace data.

Although this solution works reasonably well for small scale measurements, its drawbacks became evident at scale. The $\mathcal{O}(p)$ storage requirements for each local definition mean that a significant amount of memory is already required at measurement time. In particular, the bitstring representation is extremely bad for `MPI_COMM_SELF` and duplicates since only a single bit is set. Moreover, the amount of data to be processed during unification is $\mathcal{O}(p^2)$. While algorithmic improvements in the unification process using a hierarchical scheme [6] successfully parallelized the work, the reduction of the overall workload needed further attention. Since the bitstring for `MPI_COMM_SELF` is different on every process, no merging is possible during unification, leading to $\mathcal{O}(p^2)$ storage requirements for their global definitions. And finally, the bitstring records are also created for every duplicate of a communicator, leading to a lot of redundancy for application codes such as *PFLOTRAN*, quickly resulting in gigabytes of communicator definition records, such that trace analysis was not possible for more than 48k processes. Along with the quadratic growth in size, unification times of the original implementation were also unacceptable as seen in Fig. 2.

Times reported by *PFLOTRAN* for summary and trace collections employing runtime filters on Jugene IBM BG/P compared with reference times from

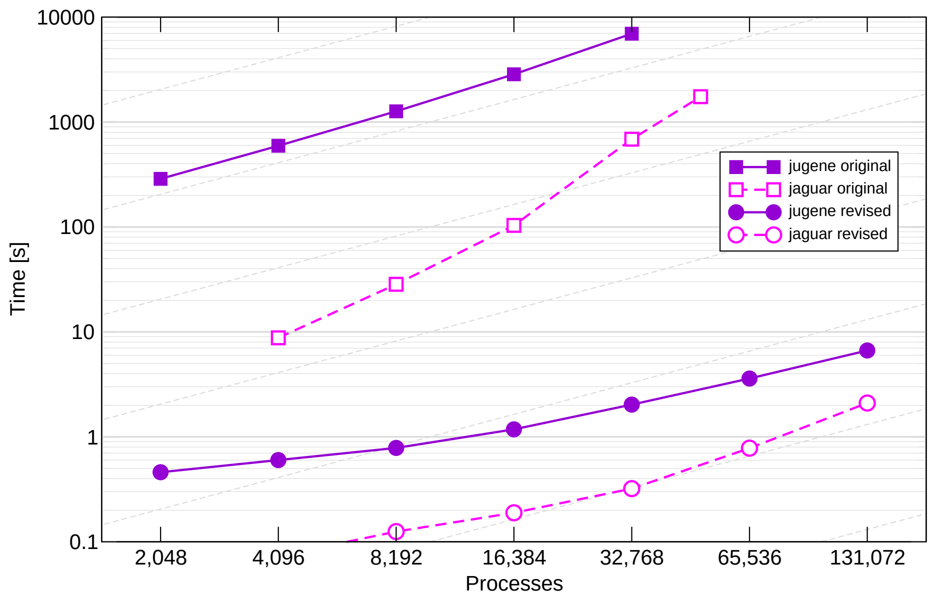


Fig. 2. Time to unify *PFLOTRAN* identifier definitions (and write them to disk with associated mappings for each process) on Jugene IBM BG/P and Jaguar Cray XT5, comparing original and revised Scalasca implementations.

the uninstrumented executions in Fig. 1 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 only significant for 'flow' at 128k processes (not shown), it is much more pronounced in measurements on Jugene IBM BG/P for even 16k processes and for 128k processes grows to a factor of seven! This difference can be attributed to translations of communication partner/root process ranks (from the local rank in the MPI communicator to the global rank in `MPI_COMM_WORLD`) in every communication operation event recorded, and the relative speeds of computation and communication on both systems.

The cost of the MPI standard routine `MPI_Group_translate_ranks` provided for this conversion increases with the size of the communicator, however, it also depends on the rank(s) being translated with the worst-case cost that for the largest rank. On Jugene BG/P with 128k processes, translation of the largest rank in `MPI_COMM_WORLD` takes 3.0 ms on average.[5] While this is small compared to the time for collective operations like `MPI_Comm_dup` (84 ms for `MPI_COMM_WORLD` at this scale), it is much larger than typical point-to-point communication operations. Even more insidious, the variable cost according to the partner rank results in severe distortion of the measurement.

[5] On Jaguar Cray XT5 with 128k processes the average translation time is 1.0 ms.

4 Communicator Management during Trace Collection

To address the scalability limitations described in the previous section, communicator management in Scalasca was completely re-designed. In the following, we present the solutions we have implemented with respect to scalable communicator tracking, unification and representation.

4.1 Distributed Communicator Tracking

Instead of determining and storing the whole group information on communicator creation on each process at measurement time, we developed a distributed communicator tracking scheme requiring very little memory and allowing the efficient reconstruction of the global communicator information at the end of measurement.

In the distributed tracking scheme, each process stores a record with a globally unique pair of integers as a key, the process-local identifier of this communicator used by the event records referring to it, the process' local rank within the communicator, and its size. The 2-tuple used as key is the foundation for the efficient reconstruction of the global communicator structure. It needs to be globally unique to detect which distributed partial definition records build a global communicator record. The local identifier alone cannot provide this, as it merely represents the information that this record belongs to the ith locally defined communicator. However, different processes can define different communicators, giving this identifier a purely local meaning.

To build these unique keys, each process keeps a state variable during measurement to count the number of communicators where this process was rank 0. For improved readability, we will henceforth write that a process p *defines* a communicator record when it is the process with rank 0 in the corresponding communicator. As the value of this counter is strictly increasing on each rank, and the global rank of the defining process is unique, the combination of those two values forms a unique key for each communicator. Also, both values can be determined during measurement at very low cost.

In principle, each process participating in a newly created communicator can determine the global rank of the defining process by mapping local rank zero onto the group of `MPI_COMM_WORLD` using `MPI_Group_translate_ranks`. However, the local state variable of this process is unknown to all but the defining process and needs to be distributed. For simplicity, we avoid the call to `MPI_Group_translate_ranks` and use a broadcast on the new communicator with the defining process as the root, sending its global rank as well as the aforementioned count. The defining process increments its counter after the broadcast, as its counter value has now been used for the new entry. Since communicator creation is a collective operation – and we are not aware of any MPI implementation not synchronizing all of the participating processes – the additional overhead for this communication operation is negligible.

4.2 Unification of Definition Identifiers

As mentioned before, each process assigns a local numerical identifier to each communicator it is part of. This identifier is used in event records referencing communicators (such as sending or receiving a message), later being translated into a global identifier using a per-process mapping table during analysis.

In the final communicator definition record stored with the trace, the distributed entries created during measurement have to be combined. During this stage the unique 2-tuple key needs to be transformed into the global identifier of the communicator. Here, we need special handling for `MPI_COMM_SELF`-like (i.e., single-process) communicators, which get added to the global list of communicators after applying the unification algorithm presented below. In the remainder of this section, we therefore only refer to multi-process communicators.

For those, we assign strictly increasing values to the communicator records, starting from 0 with the first communicator defined by rank 0, which in any case will be that of `MPI_COMM_WORLD`. All communicators defined by rank 0 will get assigned to the next available identifiers, until the same process is performed with all other communicators and ranks. To facilitate the unique numbering, we use a single exclusive prefix reduction where each process provides the number of communicators it defined. The resulting value on any process k then denotes the number of communicators defined by processes with a rank lower than k. This information is then distributed to every process using `MPI_Allgather`. With this knowledge, local counter values initially used in the tuple can be shifted by the offset of the corresponding defining process, making them globally unique. The resulting record therefore already enables the mapping of local to global communicator identifiers.

The next step assembles the list of global ranks for each process participating in a communicator. First, the total number of multi-process communicators c is broadcast to every process. This value is a by-product of the earlier prefix sum, requiring only one addition on the process with the highest rank number. Finally, we perform c gather operations, where each process provides either its local rank, if it was part of the specific communicator, or -1 to denote that it was not. The root can then assemble the list of processes by extracting them from the gathered values.

In total, our new distributed communicator tracking scheme has a local memory requirement of $\mathcal{O}(1)$ per communicator per process during measurement, and can be unified and consolidated with $\mathcal{O}(c \cdot \log p)$ communications.

4.3 Representation of Communicators

To eliminate the inherent redundancy of the original communicator storage scheme for duplicates, we adopted the approach taken by the MPI standard of separating groups and communicators. In the revised scheme each group is therefore stored only once, potentially being referenced by multiple communicator definition records. These now only consist of two integers, a global communicator identifier and the global identifier of the associated group definition.

Moreover, we no longer represent groups as bitstrings, but rather as an ordered list of integers where the entry at position i stores the rank in the group of MPI_COMM_WORLD of rank i in the local group. The global rank of a process can then always be reconstructed by a simple table lookup at the corresponding entry in the communicator's group. Also, the memory representation of this rank list is much more compact than the bitstring for sparsely populated communicators.

Special flags are also included in the group record for groups corresponding to the standard MPI communicators, MPI_COMM_SELF and MPI_COMM_WORLD. This provides an obvious space saving for the ubiquitous world-group record, but more importantly the generic self-group record avoids proliferation of distinct records for each rank. Compact representations for other MPI groups have been investigated by others (e.g., [14]) and may be considered in future work.

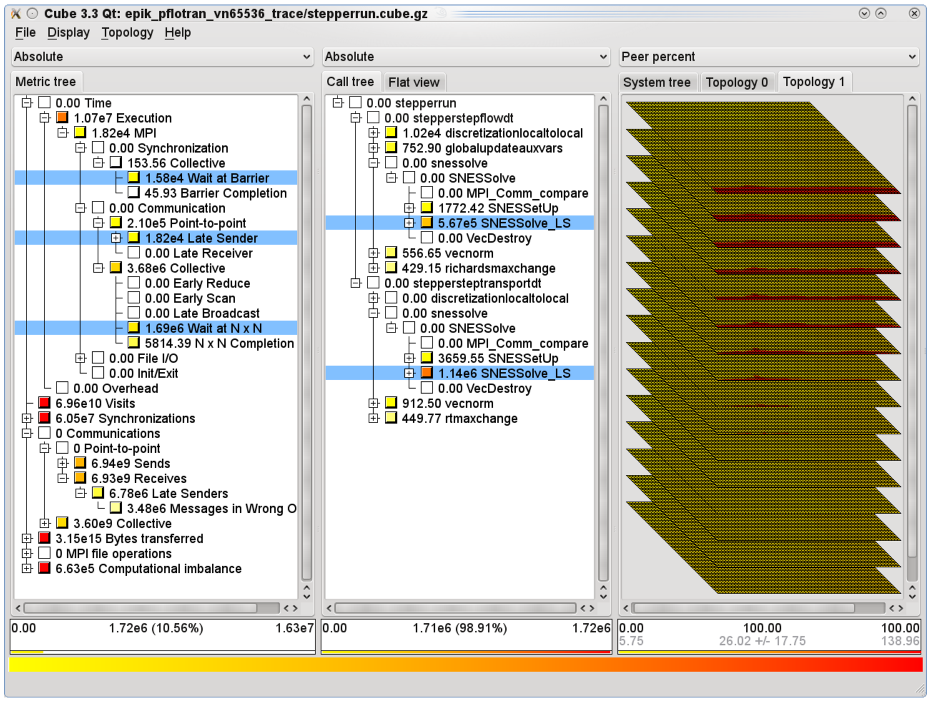
4.4 Rank Translation

Since *PFLOTRAN* only uses duplicates of the MPI standard communicators, for which rank determination is trivial, it would be straightforward to incorporate special handling for this case. Unfortunately, applications using general MPI communicators would not benefit. However, the new storage scheme of groups and communicators allowed us to use local ranks in communication events (as the global rank can always be reconstructed, if necessary). Although it required changing the trace file format, this unnecessary translation overhead during measurement has therefore been eliminated. Trace reading also needed to be adapted, however, parallel event replay required untranslated ranks in communicators in any case, so analysis performance is not degraded.

5 Evaluation

The effectiveness of avoiding rank translation for every communication event during Scalasca trace measurement is evident in Fig. 1, which compares the *PFLOTRAN* 'flow' and 'tran' phase execution times on BG/P when ranks are globalized (G-trace) versus when they remain local ranks (L-trace). With the new communicator management traces are now collected with minimal dilation as formerly only possible for runtime summarization experiments.

Although communicator definitions are now much more compact, total trace sizes, and the associated storage for buffering event records during measurement, remain essentially unchanged with the new scheme. For the 4.0 TB event trace from 128k processes, unification now takes only 6.7 seconds to produce 10.4 MB of global definitions and 242.9 MB of mappings. Focusing on communicator definition records only, the records for 48k processes executing *PFLOTRAN* originally exceeded 1.4 GB, and were consequently too large for the Scalasca trace analyzer to handle, whereas the new records for 128k processes present no such problem. Figure 3 shows the trace analysis for an execution with 64k processes revealing the distribution of MPI communication and synchronization waiting times that complement the application's inherent computational imbalance [16].



```
0.00 Time
  1.07e7 Execution
    1.82e4 MPI
      0.00 Synchronization
        153.56 Collective
          1.58e4 Wait at Barrier
          45.93 Barrier Completion
      0.00 Communication
        2.10e5 Point-to-point
          1.82e4 Late Sender
          0.00 Late Receiver
        3.68e6 Collective
          0.00 Early Reduce
          0.00 Early Scan
          0.00 Late Broadcast
          1.69e6 Wait at N x N
          5814.39 N x N Completion
      0.00 File I/O
      0.00 Init/Exit
  0.00 Overhead
6.96e10 Visits
6.05e7 Synchronizations
0 Communications
  0 Point-to-point
    6.94e9 Sends
    6.93e9 Receives
      6.78e6 Late Senders
        3.48e6 Messages in Wrong O
  3.60e9 Collective
3.15e15 Bytes transferred
0 MPI file operations
6.63e5 Computational imbalance
```

```
0.00 stepperrun
  0.00 stepperstepflowdt
    1.02e4 discretizationlocaltolocal
    752.90 globalupdateauxvars
    0.00 snessolve
      0.00 SNESSolve
        0.00 MPI_Comm_compare
        1772.42 SNESSetUp
        5.67e5 SNESSolve_LS
        0.00 VecDestroy
    556.65 vecnorm
    429.15 richardsmaxchange
  0.00 stepperstepransportdt
    0.00 discretizationlocaltolocal
    0.00 snessolve
      0.00 SNESSolve
        0.00 MPI_Comm_compare
        3659.55 SNESSetUp
        1.14e6 SNESSolve_LS
        0.00 VecDestroy
    912.50 vecnorm
    449.77 rtmaxchange
```

```
0.00        1.72e6 (10.56%)        1.63e7
0.00        1.71e6 (98.91%)        1.72e6
0.00            100.00             100.00
5.75        26.02 +/- 17.75        138.96
```

Fig. 3. Scalasca analysis report explorer showing timestep loop extract of *PFLOTRAN* trace experiment with 64k processes on BG/P. MPI communication and synchronization waiting time metrics selected in the left pane correspond to over 10% of the total time. The central pane shows PETSc `SNESSolve_LS` line search solver calls employed in the flow and transport phases are responsible for 99% of this, whereas the distribution of waiting times for the 64k processes in the right pane reveals that it complements the application's inherent computational load imbalance.

6 Conclusion

For trace collection and analysis of the *PFLOTRAN* application at large scale, Scalasca management of MPI communicators needed to be comprehensively reengineered. Eliminating the translation to global ranks of communicator ranks of partner and root processes in communication operations to avoid associated measurement dilation also motivated more efficient tracking and storage of communicator specifications required for message replay during analysis. With the revised implementation, formerly impossible trace analysis with 128k and more processes has now been achieved. Small extensions are under investigation for the rare applications using MPI inter-communicators. The new communicator management scheme has also been contributed to the open-source Score-P measurement system [1] being developed for the next generation of the Scalasca, Periscope, TAU and Vampir performance tools.

Acknowledgments. Financial support from the Helmholtz Association of German Research Centers through Grant VH-NG-118 is gratefully acknowledged.

References

1. an Mey, D., Biersdorff, S., Bischof, C., Diethelm, K., Eschweiler, D., Gerndt, M., Knüpfer, A., Lorenz, D., Malony, A.D., Nagel, W.E., Oleynik, Y., Rössel, C., Saviankou, P., Schmidl, D., Shende, S.S., Wagner, M., Wesarg, B., Wolf, F.: Score-P–A unified performance measurement system for petascale applications. In: Proc. Competence in High Performance Computing, HPC Status Konferenz der Gauß-Allianz e.V. (CiHPC, Schwetzingen, Germany). Springer (June 2010), (to appear)
2. ANL/LANL/ORNL/PNNL/UIUC: PFLOTRAN, http://ees.lanl.gov/pflotran/
3. Argonne National Laboratory, USA: FPMPI-2.1g (August 2010), http://www.mcs.anl.gov/research/projects/fpmpi/
4. Argonne National Laboratory, USA: MPICH2-1.4 MPE (June 2011), http://www.mcs.anl.gov/research/projects/mpich2/
5. Barcelona Supercomputing Centre, Spain: Extrae-2.1.1 (March 2011), http://www.bsc.es/ssl/apps/performanceTools/
6. Geimer, M., Saviankou, P., Strube, A., Szebenyi, Z., Wolf, F., Wylie, B.J.N.: Further improving the scalability of the Scalasca toolset. In: Proc. PARA2010 (Reykjavík, Iceland). Lecture Notes in Computer Science, Springer (June 2010)
7. Geimer, M., Wolf, F., Wylie, B.J.N., Ábrahám, E., Becker, D., Mohr, B.: The Scalasca performance toolset architecture. Concurrency and Computation: Practice and Experience 22(6), 702–719 (April 2010)
8. Hammond, G.E., Lichtner, P.C.: 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. Jülich Supercomputing Centre, Germany: Scalasca toolset for scalable performance analysis of large-scale parallel applications, http://www.scalasca.org/
10. Lawrence Livermore National Laboratory, USA: mpiP-3.3 (June 2011), http://mpip.sourceforge.net/
11. Mohr, B., Frings, W. (eds.): 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/
12. Technische Universität Dresden, Germany: VampirTrace-5.11 (June 2011), http://www.tu-dresden.de/zih/vampirtrace/
13. Technische Universität München, Germany: Periscope-1.3.2 (February 2011), http://www.lrr.in.tum.de/periscope/
14. Träff, J.L.: Compact and efficient implementation of the MPI group operations. In: Proc. 17th European MPI Users' Group meeting (Stuttgart, Germany). Lecture Notes in Computer Science, vol. 6305, pp. 170–178. Springer (2010)
15. University of Oregon, Eugene, USA: TAU-2.20.2 (May 2011), http://tau.uoregon.edu/tau/
16. Wylie, B.J.N., Geimer, M.: Large-scale performance analysis of PFLOTRAN with Scalasca. In: Proc. 53rd CUG meeting (Fairbanks, AK, USA). Cray User Group, Inc. (May 2011)
17. Wylie, B.J.N., Geimer, M., Mohr, B., Böhme, D., Szebenyi, Z., Wolf, F.: Large-scale performance analysis of Sweep3D with the Scalasca toolset. Parallel Processing Letters 20(4), 397–414 (December 2010)