Science and Supercomputing in Europe, HPC-Europa Transnational Access report 2007, pp. 453–461, CINECA Consorzio Interuniversitario, Bologna, Italy.
ISBN 978-88-86037-21-1

Scalable performance analysis of large-scale parallel applications on MareNostrum

Brian J. N. Wylie
Jülich Supercomputing Centre, Forschungszentrum Jülich, Jülich, Germany

Abstract

The scalasca *toolset was readily ported to Mare Nostrum and its measurement and automated analysis scalability verified to 2,048 processes with the* SMG2000 *benchmark and* WRF *numerical weather prediction application. It was subsequently compared with the local Paraver/MPItrace tools, albeit with difficulty due to their quite different modes of operation, and found to be complementary:* scalasca *is able to rapidly generate automated analysis summaries from the largest configurations, which can be used to direct much more extensive Paraver trace analyses to particularly significant performance issues. A previously unexpected problem with exiting MPI communication collectives was identified and codified in a new* scalasca *"N x N Completion" property, allowing its significance to be investigated and instances to be examined with Paraver.*

1 Introduction

The visit to Barcelona Supercomputer Centre funded by HPC Europa allowed the collaboration between the groups developing trace-based parallel program performance analysis tools in Barcelona and Jülich to be extended. Collaboration was initiated in spring 2006 with a port of the KOJAK toolkit measurement system to the IBM JS21-blade cluster 'MareNostrum' and development of a converter enabling KOJAK traces to be visualised and analysed by Paraver [1, 4]. Specifically addressing the scalability requirements of long-running applications on large-scale systems of thousands of processes/processors, the scalasca toolset [2] builds on and extends the KOJAK toolkit [3].

scalasca development to date has focussed primarily on MPI-1 applications consisting of thousands of processes. A new streamlined measurement system can collect traces from each process rank for distributed analysis based on parallel replay of message-passing events, and has demonstrated good scalability to tens of thousands of processes on IBM BlueGene/L and Cray XT systems (as well as a variety of smaller-scale platforms). At such scales, unique challenges of functionality, reliability and scalability are encountered which are generally not repeatable with smaller configurations. Fundamental operating system, message-passing system, memory system and filesystem limitations encountered porting scalasca match similar challenges experienced by application developers scaling their codes to these largest scales.

Although the merits of event tracing have been demonstrated for performance analysis and tuning parallel applications at large scale, the approach is

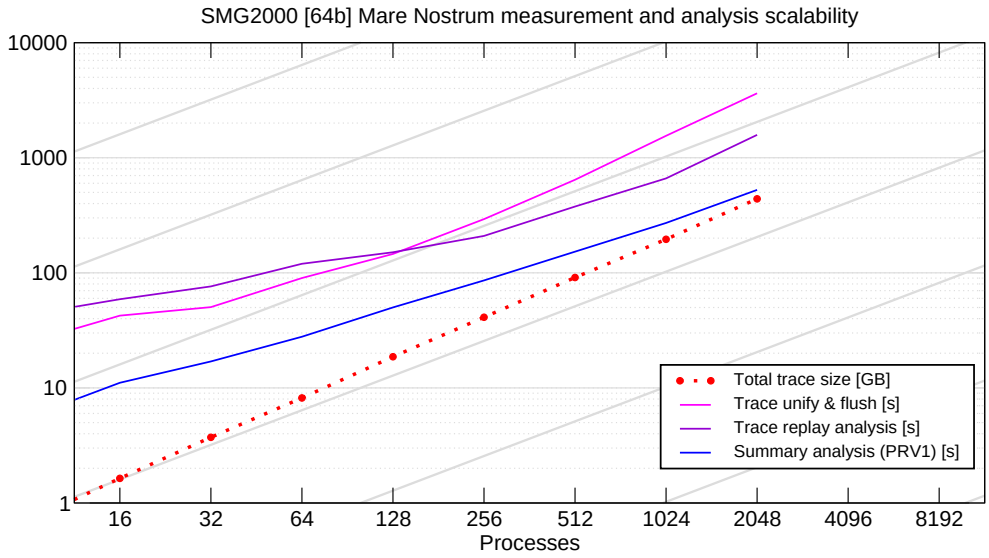

Figure 1: scalasca measurement/analysis scaling for MareNostrum SMG2000.

prone to measurement perturbation and excessive trace volume when instrumentation is inappropriately configured or measurement continues for prolonged periods. These issues can be avoided by carefully-directed instrumentation and bounded measurement intervals, however, it is often difficult to determine these in advance. In this respect, runtime summarisation of key execution performance characteristics in an initial measurement execution can be a basis for directing instrumentation (both statically compiled and configured at runtime) in subsequent measurements which can comprise both overview summaries and selective event traces. Runtime summarisation is also often a more effective approach for hardware counter measurements that can easily result in voluminous traces.

2 Preparation and Validation

Configuration and building the scalasca toolset on MareNostrum required a little customisation to the local system environment, particularly to handle code-generation inconsistency: by default, 64-bit code is generated by the MPI compilers versus 32-bit for the non-MPI compilers. After basic installation testing, measurement and analysis scalability was validated with the communication-intensive SMG2000 benchmark [5] up to 2,048 processes (which was the maximum accessible during the visit): the benchmark was configured to run 5 solver iterations of a 64×64×32 problem size per process for *weak scaling* behaviour. Figure 1 shows total trace size (corresponding to the number of execution events) growing slightly more than linearly up to over 400GB, with trace writing following suit, while both summary and trace replay analyses both scale less than linearly with increasing numbers of processors. (Traces

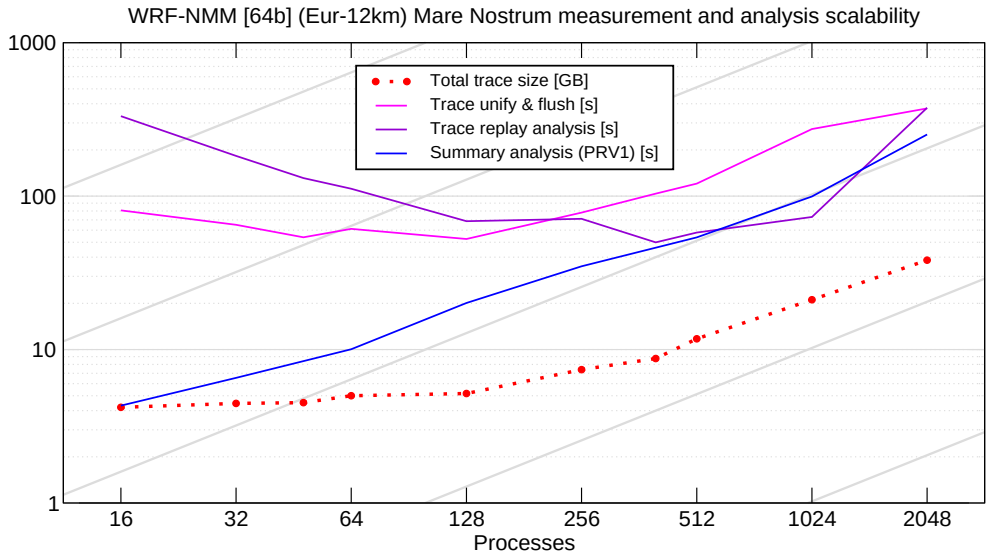

Figure 2: scalasca measurement/analysis scaling for MareNostrum WRF-NMM.

were collected without hardware counter measurements, whereas the runtime summaries contain an additional set of 8 hardware counter metrics.) 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.

This was followed by application of scalasca to measure and analyse one of the local 'production' applications, the WRF-NMM numerical weather prediction code [6]. The application and its analysis are discussed in the following section, however, an example analysis report is shown in Figure 3. Despite its significantly greater complexity, not limited to building an executable and its runtime configuration, scalasca measurement and analysis was similarly validated with WRF-NMM using a fixed-size Eur-12km dataset for *strong scaling* up to 2,048 processes: Figure 2 shows the total trace size growing less than linearly to 38GB, with trace writing following the same curve, runtime summary analysis time grows also less than linearly, while trace replay analysis time reduces to approximately one minute (for all but the largest process configuration).

Although scalasca trace collection and analysis were limited by MareNostrum I/O bandwidth of 150MB/s writing and reading back large trace files, particularly cumbersome trace merging and rewriting were avoided. This is not the case with MPItrace/Paraver (Figure 4), where trace processing was found to take many hours (rather than minutes), even when a distributed, hierarchical merge was employed: it also uses a separate follow-on batch job, whereas scalasca trace analysis efficiently re-uses the processes of the measurement batch job.

For an initial overview analysis, a prototype of scalasca runtime summarisation of event measurements was demonstrated to avoid event buffering and

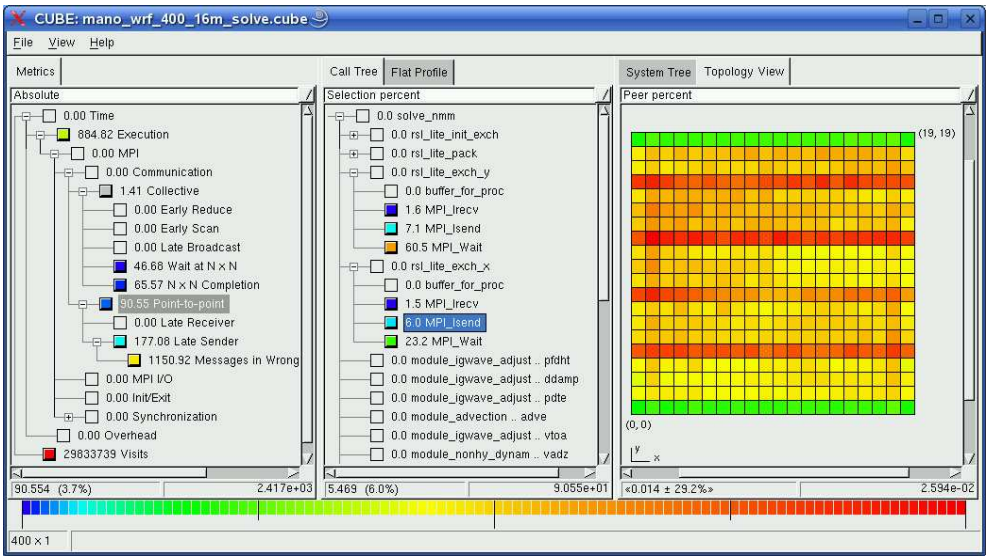

Figure 3: scalasca analysis report presentation of 400-process WRF-NMM execution experiment. The point-to-point communication time metric chosen from the metric hierarchy (*left pane*) is shown for call-paths of the solver (*central pane*) and its associated severity distribution per process (*right pane*).

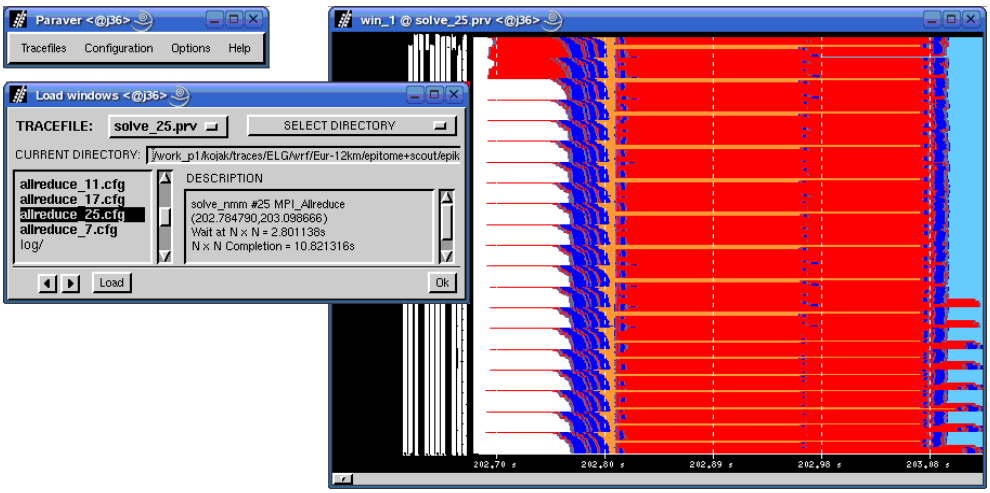

Figure 4: Paraver presentation of timestep 25 of 400-process WRF-NMM trace with `MPI_Allreduce` depicted orange: imbalanced entry results in 2.8s of *Wait at N x N*, dwarfed by problem with staggered exit (10.8s *N x N Completion*).

tracefile handling, rapidly delivering a summary profile of the application execution. Although lacking detailed analysis that is only possible from traces, the summary report is produced in the same format as the trace analysis reports, so that the same presentation and postprocessing tools can be used.

MareNostrum PowerPC-970MP processor hardware counters, in groups of up to 8 counters, could also be recorded with measured events and included in the summary reports. The associated metrics can then be combined (from one or more measurement experiments) into hierarchies of derived metrics, e.g., for types of data accesses, instructions and cycles. By processing the names of the compute nodes recorded in measurement experiments, they could be converted into server, chassis, blade and processor indices and analyses then presented graphically according to system hardware topology. Notably, applications are typically allocated non-contiguous blades, though this is not known to degrade performance.

While scalasca and Paraver/MPItrace offer largely comparable functionality, direct comparison is complicated by their different default configurations. For example, Paraver/MPItrace typically dynamically interposes on MPI library calls, and makes little use of application callstack information to relate performance to application source code. Although it also provides support for application function instrumentation, and measurement configuration to control the user functions and amount of associated callstack recorded with MPI events, to produce detailed traces comparable to those with scalasca, the resulting traces rapidly become huge and awkward to handle. Despite these limitations, it was possible to directly compare scalasca and Paraver/MPItrace traces and analyses from short small-scale executions, and verify that they matched (to within run-to-run measurement variation). Previously-developed utilities were also employed to convert a scalasca trace into Paraver format for more direct comparison of the analyses of the same measurement experiment.

3 Analysis of WRF-NMM on MareNostrum

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 Nonhydrostatic Mesoscale Model (NMM) within the Weather Research and Forecasting (WRF) system [6]. Version 2.1.2 (released January 2006) is a flexible, state-of-the-art atmospheric simulation system designed to be portable and efficient on parallel computing platforms. It consists of some 530+ source files amounting to over 300 thousand lines of code (75% Fortran, 25% C).

Simulations on MareNostrum were analysed which used the Eur-12km dataset with a default configuration, apart from varying the duration of the forecast and disabling intermediate checkpoints: it was not determined how 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 investigated. 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 many times longer (even when excluding I/O).

scalasca runtime summarisation (with trace collection disabled) determined that 1,972 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 profile (including 8 hardware counter metrics) at the end of measurement collection took 250 seconds (for the 2,048-process case).

As well as providing a basic profile of the WRF-NMM execution time and message-passing characteristics, base hardware counter metrics and associated derived metrics are structured into hierarchies [7]. The top of Figure 5 shows the profile summary presented by the scalasca analysis report browser, with the metric hierarchy partially expanded, the callpath hierarchy expanded to the solver, and the MPI process topology distribution. The colour scale presents metric severity values according to the mode chosen for each panel.

Having selected the metric for data loads that miss both levels of cache and must come from memory (*LOAD_HIT_MEM=PM_DATA_FROM_MEM*), the values for this metric are shown as percentages for a section of the callpath tree down to the solver, `solve_nmm`, where 98% occur, and then the distribution of solver values per process shown using the application's MPI virtual topology. Although the variation is small, it may be indicative of imbalance with a significant performance penalty.

From the callpath and message statistics in the summary, the buffering requirements for a complete trace of this execution were estimated to be 133MB 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 short blacklist of functions to be ignored during measurement was prepared, and specified during trace collection.

Flushing the 6.4GB 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 3 minutes. With this new report (at the bottom in Figure 5), 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, significant amounts of time are identified in point-to-point operations where the receiver was blocked waiting for the sender to initiate the message transfer (*Late Sender*) and there is also a significant amount of imbalance when processes enter NxN collectives (*Wait at N x N*) and must wait on others.

Of particular interest is the highlit *N x N Completion* time, a newly incorporated metric which indicates an unusual imbalance exiting `MPI_Allreduce` in the advection module of the solver. Generally this time would be negligible, and was initially thought to be due to limitations of the schemes employed to determine clock offsets and correct clock drifts. From closer examination of the traces with Paraver, however, it was determined that in this case it is significant because it shows a single occurrence where a few of the processes were over 1.25 seconds late exiting. Furthermore, these processes are in groups of three or four of those sharing a JS21 blade. While this is a rela-

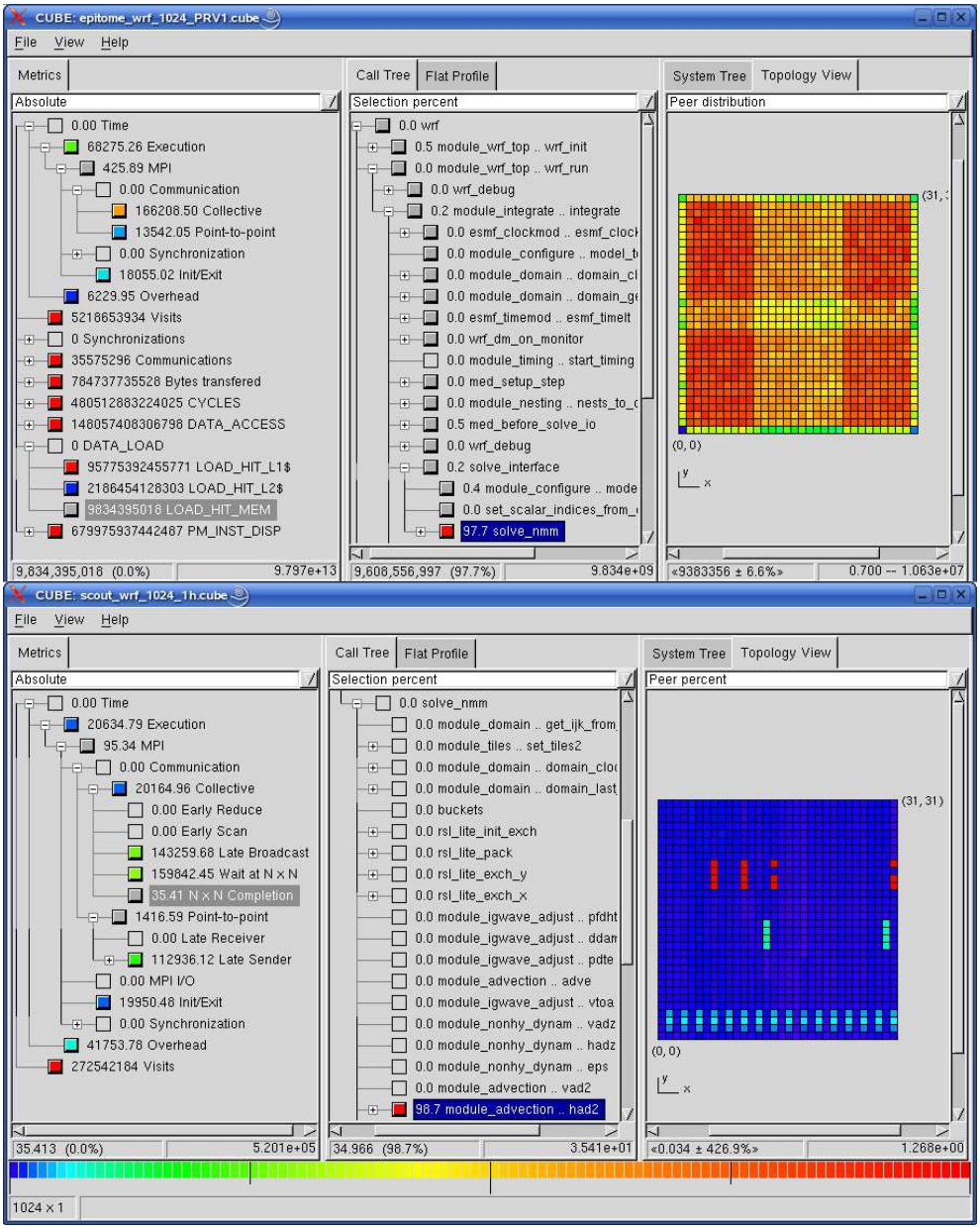

Figure 5: scalasca analysis reports of 1,024-process WRF-NMM (Eur-12km) simulation on MareNostrum. Runtime summary showing *LOAD_HIT_MEM* hardware counter metric in solver (*top*) and trace analysis showing *N* x *N* *Completion* time in solver advection step (*bottom*).

tively 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 x N* for the next `MPI_Allreduce`. Occurrences of this uncoordinated exit from collective operations are sporadic and relatively infrequent, affecting slightly more than 1% of collectives (irrespective of the number of processes involved), but result in serious disruption of the smooth execution of the WRF-NMM application.

It appears that the MPI(CH) implementation for Myrinet is responsible, and no fix or workaround has yet been found. 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. With the incorporation of the *N x N Completion* metric, scalasca is now able to quantify its impact on applications.

4 Integration Prototype

Paraver includes an experimental capability for reading a command file when signalled, which allows a specified interval of the trace to be presented on demand. An alternate mechanism is for intervals to be specified within Paraver configuration files, such that Paraver can provide a menu of intervals each with descriptive comments. With these means, trace intervals identified from automated analysis as having particularly significant performance issues can be presented with Paraver and subsequently investigated in greater detail. Figure 4 shows the Paraver presentation of one WRF-NMM simulation timestep, showing several of the 400 processes delayed more than 0.25s exiting from `MPI_Allreduce`: this problem and others identified by scalasca automatic trace analysis are described in specially-prepared Paraver window configuration files.

Several difficulties must be addressed in realising this process. Firstly, the scalasca traces are typically much larger than can be conveniently converted into Paraver format, and also too large for Paraver to manage directly. Appropriate trace extracts, containing the intervals of interest, can therefore be prepared and converted to reduce size problems. During conversion it is important that the timebase is retained, so that events and intervals in the original trace can be located in the converted trace (extract).

The scalasca trace analyser was modified to log problem time-stamped instances with associated severities in a textual report for each process rank which could subsequently be merged into a complete problem instance severity report. For the most severe instances, Paraver configuration files for presenting the associated intervals and performance problem descriptions were prepared, and could then be examined with Paraver. If desired, thresholds were provided so that only the most significant individual events could be selected.

While this approach was adequate for some performance problems, particularly those for global collective communications, it was unsatisfactory for situations where multiple distinct events impact upon each other. Most point-to-point wait events are individually insignificant, however, they are found to frequently cascade into large groups of related events. This was investigated by grouping together individual problem instances that overlapped in

time, and summarising the resulting groups in Paraver configuration files. Although related events are now shown together for analysis with Paraver, the groups were unfortunately found to rapidly grow to many thousands of events, within which it becomes difficult to identify the most significant individuals. Furthermore, thresholding which eliminates less significant events also reduces the fidelity of grouping.

Prototyping this potential integration of scalasca and Paraver has therefore been very helpful in identifying a range of issues that need further investigation. There is also engineering work required to complete the necessary infrastructure functionality, however, it appears to be worth considering in future product development.

Acknowledgements This work was carried out 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" programme. The author is grateful for the assistance and guidance provided by his host, Prof. Jesus Labarta, the Paraver group, and the support and scientific staff of Barcelona Supercomputer Centre, that made the visit both extremely productive and enjoyable.

References

[1] B. Mohr: Transnational Access – Success Story at BSC. HPC-Europa newsletter 4 (2005) //www.hpc-europa.org/Newsletter4.pdf

[2] M. Geimer, F. Wolf, B. J. N. Wylie, B. Mohr: Scalable Parallel Trace-based Performance Analysis. Proc. 13th European PVM/MPI User's Group Meeting (Bonn, Germany), Lecture Notes in Computer Science 4192, Springer (2006) 303–312

[3] F. Wolf, B. Mohr: Automatic Performance Analysis of Hybrid MPI/OpenMP Applications. J. Systems Architecture, 49(10–11). Elsevier (2003) 421–439

[4] J. Labarta, S. Girona, V. Pillet, T. Cortes, L. Gregoris: DiP: A Parallel Program Development Environment. Proc. 2nd Int'l EuroPar Conf. on Parallel Processing (Lyon, France), Lecture Notes in Computer Science 1124, Springer (1996) 665-674

[5] Advanced Simulation and Computing Program: The ASC SMG2000 benchmark code. (2001) //www.llnl.gov/asc/purple/benchmarks/limited/smg/

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

[7] B. J. N. Wylie, B. Mohr, F. Wolf: Holistic Hardware Counter Performance Analysis of Parallel Programs. Proc. 12th Parallel Computing (ParCo 2005, Málaga, Spain), John von Neumann Institute for Computing Series Vol. 33 (2006) 187–194