Guided Performance Analysis Combining Profile and Trace Tools

Judit Giménez[1], Jesús Labarta, F. Xavier Pegenaute, Hui-Fang Wen[2], David Klepacki, I-Hsin Chung, Guojing Cong, Felix Voigtländer[3], and Bernd Mohr[4]

[1] {judit.gimenez, jesus.labarta, xavier.pegenaute}@bsc.es BSC-UPC
[2] {hfwen, klepacki, ihchung, gcong}@us.ibm.com IBM T.J. Watson
[3] felix.voigtlaender@gmail.com RWTH Aachen University
[4] B.Mohr@fz-juelich.de Jülich Supercomputing Centre

Abstract. Performance analysis is very important to understand the behavior of applications and to identify bottlenecks. Performance analysis tools should facilitate the exploration of the data collected and help to identify where the analyst has to look. While this functionality can promote the tools usage on small and medium size environments, it becomes mandatory for large-scale and many-core systems where the amount of data is dramatically increased. This paper proposes a new methodology based on the integration of profilers and timeline tools to improve and facilitate the performance analysis process.

1 Introduction

The performance of an application is influenced by multiple and complex factors. Performance measurement and analysis tools allow to understand the applications performance behavior and give hints on how they can be optimized. There are two main approaches on such performance analysis tools:

Profile-based tools work with statistics aggregated over the time dimension, keeping the metrics per function and/or process. Despite the time aggregation, the volume of data can still become huge if a large number of processes is measured or a very large number of metrics is precomputed. But usually the accumulation over time drastically reduces the data volume. The profiled data is presented structured in tables or trees. These two facts (size and type of display) facilitate the correlation between metrics. Many of the profiler tools can link the metric values with the source code showing the location and in some cases the code can be edited from the profiler GUI.

Timeline visualization tools work with performance data in the 2D space defined by processes and time. This approach yields a lot of data, and the justification is that the variance over time is very important. The advocates for traces defend that the aggregation could mask the metrics, preventing the analyst from looking at the metrics details and variance. Keeping the time dimension allows the user to dynamically compute separate values for different time regions while doing the analysis. In addition, some of these tools are capable

of computing new metrics defined on the fly increasing the search tree size. Many timeline tools allow the user to accumulate the metrics in tables to obtain profile-like views or histograms for a given metric.

There are clear strengths in both approaches that make them complementary, but usually these tools are disconnected. A first reason is that profilers and timeline tools work with different types of data, but even if there is a translator, the integration does not go further than being able of working with performance data from the other tool. To make things more difficult, each tool has its own format despite initiatives to promote a common format like OTF [1] for traces and PERI-XML [2] for profiles. The contribution of this work is to demonstrate how profilers and timeline tools can interoperate defining a new methodology that uses at each step the tool that easily answers the analyst question.

The methodology defined herein benefits from both approaches. The analysis is initiated using the profiler to identify regions or metrics of interest for a deep analysis. For these regions/metrics, the timeline tool is used to display the details, e.g., to investigate the context or history of a source of unusual metric values. The integration should provide the feeling of a unified environment capable of collecting all the information required in a single run and the tools should be responsible for keeping the analysis context as transparent as possible. In our work we have interfaced KOJAK and PeekPerf profilers with the Paraver analysis tool. The focus of the methodology is not on the data adquisition but on the analysis phase because we consider it more complex.

2 Performance Analysis Workflows

Post-mortem performance analysis is usually defined in two steps for both profile and timeline tools. The first step collects data from the execution and the second one displays the data to the analyst. If a user wants to use a profiler and a timeline for the analysis, usually this would require to do two different executions of the application, unless there is a translator and the data required by one tool can be extracted from the data collected by the other. Some tools like KOJAK divide the second step in two parts: the automatic analyzer (EXPERT) extracts metrics from the trace that are presented to the user with CUBE.

The proposed approach (Figure 1) extends this structure to explore new paths between the collected data and its visualization. The initial path connects the profile and the tracefile views combining them to carry out the analysis. Other interesting paths would be to generate new profile views from the visualization modules (usually trace visualization, but even from a profile view). This approach refines the metrics based on the results of a previous analysis step.

In the current implementation, the selection is controlled by the user, who decides what to do next. This includes a manual refinement to extract new metrics based on the previous step. But this process can be done automatically by an expert system. The methodology would be defined on a search tree and the intermediate results would decide the branch for the next step. The system would present the relevant data to the analyst who can decide where and how to

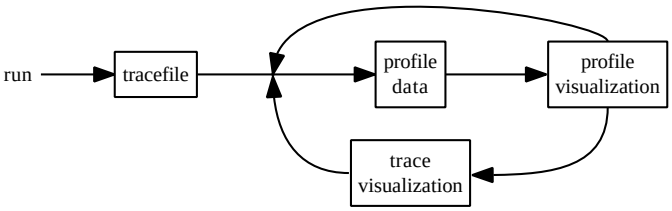

Fig. 1. Proposed workflow.

refine the analysis. An example of semi-automatic approach is the IBM's High Productivity Computing System toolkit (HPCST) [3] which actively searches for known performance patterns instead of recording information. The toolkit integrates performance tools, compiler and expert knowledge.

3 Coupling Profile and Timeline Analyzers

As depicted in the previous section, one of our objectives is to obtain all the information from a single run. It is not only a matter of conserving resources, but also to avoid correlating different runs measurements. When data is collected with instrumentation (as opposite to sampling) the overhead of gathering the information is similar for both timeline and profiling except on the regions where the tracing buffer is flushed to disk. The proposed approach is to collect the timeline information and to extract from it the profiled data.

The module that processes trace data to obtain the profiler input corresponds to the EXPERT module in the KOJAK environment. It is a kind of filter that extracts and aggregates the data relevant to the selected metrics. The user is able to select the metrics to be explored, but it is important to provide a good default set. Examples of performance metrics to be analyzed by the profiling environment are: time, instructions per cycle, load balance, L2 misses, L1/L2 miss ratio and message size. The filter program may use the functionality provided by the trace tool to generate a profile. In order to be able to later connect the profile visualization with the timeline visualization, in this phase some additional references have to be included on the profile data. The references should define which views can be generated and how to compute them (for instance being on a given user function or being on a region with a late sender).

The profile visualizer is used to analyze the generated metrics. The tool has to offer a new functionality for the coupling with the timeline visualizer: it needs to accept as part of its input new hidden metrics used to send a request to an external presenter and to offer to the user the possibility to call the timeline.

Finally, the timeline visualizer provides a mechanism to be driven by the profile analysis, interpreting its requests. The basic mechanism allows the profiler to raise a new window on the timeline tool. In the future it might be interesting to use a mechanism that allows to create a kind of dialog, or to get feedback. There

is a wide range of possibilities: from checking that the view was successfully presented to get details on the data shown to be added on the profiler display.

4 The Tools

This section briefly describes the tools involved in this work. Obviously, the proposed methodology would be applicable to any tool providing the coupling functionality described on the previous section.

4.1 PeekPerf

The PeekPerf GUI is the control center of the IBM High Performance Computing Toolkit (IBM HPCT). The entire tuning process (instrumentation, execution and analysis) can be conducted from here. The instrumentation can be achieved without source code modifications or recompilation. The instrumentation is controlled using symbolic names in the source domain, but all the modifications to the application are transparently performed on the binary. Another feature of the toolkit is the capability to collect various dimensions of performance data, including CPU, message passing, thread activity, IO and memory. This data can be simultaneously collected in a single run, thereby significantly reducing the time needed to profile the application. Once the performance data is generated, PeekPerf presents the information in a user-friendly manner that highlights the relation between the performance metrics and the source code statements. Thus, the user can relate the results better with the source program being examined.

The profile data is available for all of the dimensions of performance data. The trace data that stores the timeline information is available for IO and MPI. PeekPerf provides the viewers to visualize the IO trace and MPI trace data. The profile-based performance data follows some predefined XML syntax so that PeekPerf is able to process the data and present it in a tree representation.

4.2 KOJAK

KOJAK is an automatic performance evaluation system for MPI, OpenMP, SHMEM, and hybrid applications written in C/C++ or Fortran. It generates event traces from running applications and automatically searches them off-line for execution patterns indicating inefficient performance behavior. KOJAK is jointly developed by Forschungszentrum Jülich, Germany, and the University of Tennessee, USA. Since 2008, KOJAK is part of the Scalasca toolset [7], which is a parallel implementation of KOJAK and therefore much more scalable. The work in this paper was still done within the KOJAK toolset, however a reimplementation within the Scalasca context is planned.

The trace is subjected to an off-line analysis performed by EXPERT which attempts to identify specific performance properties. Internally, it represents performance properties as execution patterns that model inefficient behavior. These patterns are used to recognize, classify, and quantify inefficient behavior

in the application. The performance properties addressed by KOJAK include inefficient use of the parallel programming models as well as low CPU and memory performance. The analysis process automatically transforms the traces into a compact call path profile which includes the execution time penalties caused by the different patterns broken down by call path and process or thread.

The call path profile can be viewed using CUBE [8], a generic tool for displaying a multidimensional performance space consisting of the dimensions (i) performance property, (ii) call path, and (iii) system resource. Each dimension is represented as a tree browser which can be collapsed or expanded to achieve the desired level of granularity. The tree browsers are coupled such that the penalty caused by a particular property can be broken down by call path and process or thread (Figure 2). The automatic analysis can be combined with manual timeline analysis using Paraver or Vampir to investigate the context of the previously identified patterns in more detail. For this purpose, KOJAK includes appropriate trace-format conversion utilities. In addition, it is possible to let EXPERT write a second event trace with events describing individual pattern instances. For a detailed description including a complete list of patterns see [4, 5].

4.3 Paraver

Paraver [9, 10] is a very flexible timeline analysis tool. Its analysis power is based on two main pillars. First, its trace format has no semantics; the tool is not aware of concepts such as hardware counter, MPI call or OpenMP loop but it can work with all of them as long as they are defined as events in its trace. Extending the tool to support new programming models or new performance data requires no changes on the visualizer, just to capture such data in a Paraver trace. The second pillar is that the metrics are not hardwired on the tool but programmed. To compute them, the tool offers a large set of time functions, a filter module and a mechanism to combine two timelines. This approach allows to display a huge number of metrics with the available data. To capture the experts knowledge, any view or set of views can be saved as a Paraver configuration file. After that, re-computing the view with new data is as simple as loading the saved file.

Paraver provides two main types of display. The basic view is a timeline with a row per object. Discrete metrics like user functions are drawn with code colors where each color represents a value. A gradient scale (from light green to dark blue) is used for continuous metrics like counters. Statistics are stored on tables used to compute profiles, histograms and to correlate metrics. Both types of views can be displayed as a 2D array of pixels whose scalability is somehow similar to digital photos where image pixels are mapped to display window pixels.

Paramedir [11] is a non-GUI version of Paraver which generates text files for external analysis taking as input a trace and a set of configuration files. Paramedir was used to extract from the trace the metrics to display with PeekPerf.

Paraver has been extended with a signal command functionality to enable external applications to trigger loading new views on a selected time range or zooming on a previous loaded view. This simple yet extremely flexible mechanism allows to easily steer Paraver from an external tool.

5 Examples of Analysis with the Proposed Framework

This section shows how the combination of profile and timeline tools improves the ease-of-use and accuracy of the analysis using the profiler to focus and the timeline tool to look for details/distribution. While the first example uses the trace visualization to complement the facts identified on the profiler, the second example covers the approach of refining the data depending on previous results.

5.1 KOJAK and Paraver

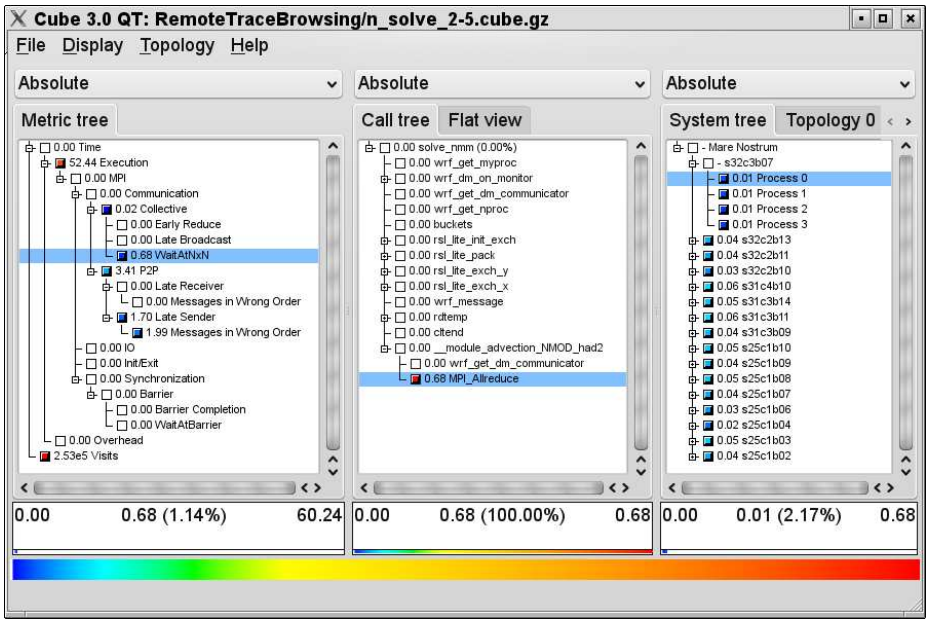

Fig. 2. KOJAK's analysis results for WRF shown in CUBE browser.

The example is based on a measurement of WRF-NMMii [12], 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. It consists of some 530+ source files with over 300 thousand lines of code (75% Fortran, 25% C). Simulations were analyzed using the Eur-12km dataset with a default configuration, apart from varying the duration of the forecast and disabling intermediate checkpoints. The data shown here are from an experiment with 64 processors on the MareNostrum machine at Barcelona Supercomputing Center.

After instrumenting the application with KOJAK tools, executing it on the parallel system, and analyzing the traces with EXPERT, it is possible to

investigate the generated pattern profile with CUBE (see Figure 2). In the "Metric tree" pane on the left side, one can quickly see that KOJAK found two main problems: First, more than half of the point-to-point traffic is *Late Senders* (3.69 seconds compared to 3.41 seconds spent in regular MPI point-to-point functions). More than half of the *late Senders* can be attributed to *Messages in Wrong Order* (1.99 vs. 1.70 seconds). The second detected problem is *WaitAtNxN* with a severity of 0.68 seconds. Selecting the collective operation pattern (as shown in Figure 2), displays in the middle "Call tree" pane where this pattern occurred in the program. In the example, it is a single call of `MPI_Allreduce` from the module function `advection::had2` called from `solvenmm`. By selecting `MPI_Allreduce` in the call tree pane, the distribution of the imbalance over the nodes and processes can be investigated in the "System tree" pane on the right.

It is important to note that waiting time in front of a collective is more of a symptom than a cause of a performance problem. When the problem cannot be resolved by looking at the corresponding portion of the source code, the context of this pattern can be investigated with timeline tools like Paraver. To do this easily, one can use the "Connect to trace browser" item of the "File" menu. This automatically starts a new instance of Paraver, loading the corresponding trace file via remote control. By default, it brings up the "State as is" display of Paraver which shows the change of MPI states over time. The dialog window also allows the user to select other Paraver configuration files if desired.

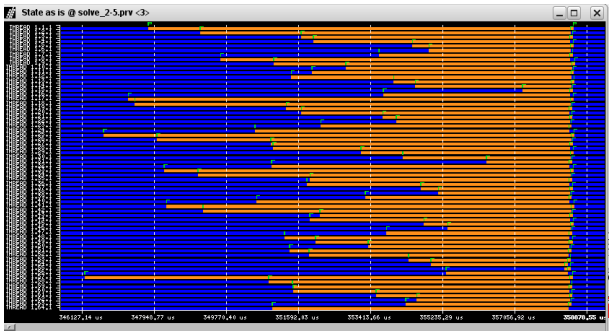

Fig. 3. Zoomed-in Paraver timeline display of WRF trace showing just the instance of the WaitAtNxN pattern with maximum severity.

In the beginning, the complete timelime is shown. By either selecting the desired pattern in the "Metric tree" (e.g. *WaitAtNxN*) or the affected call path (e.g. `MPI_Allreduce`) in the CUBE display and using the context menu item "Max severity in trace browser", CUBE is automatically configured to zoom Paraver's timeline display to the most severe instance of the selected pattern overall in the execution or in the context of the selected call path respectively. The result is shown in Figure 3. In this view dark blue corressponds to the unbalanced computing while light orange represents time in the MPI collective.

One of the tasks on the bottom of the image is the latest to end its computation and around 25% of the time 3 tasks compute while the rest wait for them.

Paraver can be used to investigate the context or the history of the instance of the pattern, for example whether a calculation or communication imbalance causes the imbalanced waiting times indicated by *WaitAtNxN*. The same technique can of course also be used to investigate the detected performance problems of the point-to-point communication of the application.

Finally, as explained earlier, EXPERT is also able to produce a trace of patterns in addition to the pattern profile report. So in this example, we could have done the same analysis steps also with the pattern trace, or even with both traces. In this case, when a zoom to the most severe instance of a pattern is requested, CUBE zooms both timeline displays via remote control of Paraver.

5.2 PeekPerf and Paraver

The environment was used to analyze the scalability of GROMACS [13], a versatile package to perform molecular dynamics, with the testcase "nucleosome." The code was executed starting with 1 task and doubling the number up to 1024. As the scalability decreases over 256 tasks, the analysis compares the run of 256 tasks with the 64 tasks case that achieves better performance. We obtained traces for these configurations and extracted a first set of global metrics to measure the efficiency at the whole execution level such as parallel efficiency and load balance as described in [14]. The values range from 0 to 1 (except IPC) with high values reporting a good performance and low values identifying a problem.

The analysis starts from PeekPerf (Figure 4 captures the metrics for both runs). As PeekPerf displays all global metrics on a line, it is very simple and quick to analyze and compare their values. The parallel efficiency is 55% with 64 tasks decreasing to 31% when there are 256 tasks. Both values indicate a poor performance, but with 256 tasks more than 2/3 of the resources are wasted. The parallel efficiency is computed from two factors: communication efficiency (time all tasks spent on MPI) and load balance (time distribution between tasks).

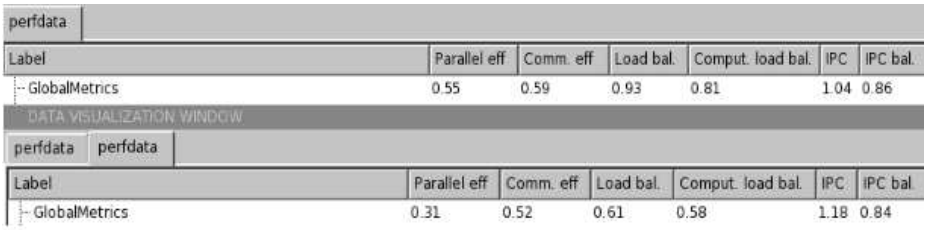

Fig. 4. Global metrics displayed on PeekPerf(top: 64 tasks case; botton: 256).

We can observe that the poor scalability is mainly due to load imbalance (duration) and computation imbalance (instructions) as those are the factors

with a higher decrease. But, the communication efficiency, not significantly penalized when the number of task is increased, is very poor. These observations drive the next analysis step to focus on two targets: (1) analyze time and computation balance of both runs to understand the poor scalability and (2) analyze the 64 tasks run communication performance.

With respect to the first issue, as we are interested in the time and processes distribution, the timeline analysis tool is foreseen as the best alternative. The PeekPerf contextual menu offers the choice "Call Back to the Integrated Tool" that allows to easily raise a set of predefined Paraver windows. Selecting the duration of the computation bursts we detect that GROMACS is composed by two kinds of tasks: a subset performs FFTs that are characterized by a sequence of medium size computations (around 8ms with 64 tasks) while the rest execute particle computations significantly larger (around 20ms for the same case). For simplicity, the details on the load balance are provided only for the FFTs tasks– Figure 5 compares their execution. The x-axis represents time and the y-axis the MPI tasks. Despite the image of the FFTs for the 256 case is compresed and with this window size we cannot isolate the behavior of a given task, it provides enough details on the global behavior (structure, imbalance, duration, etc.).

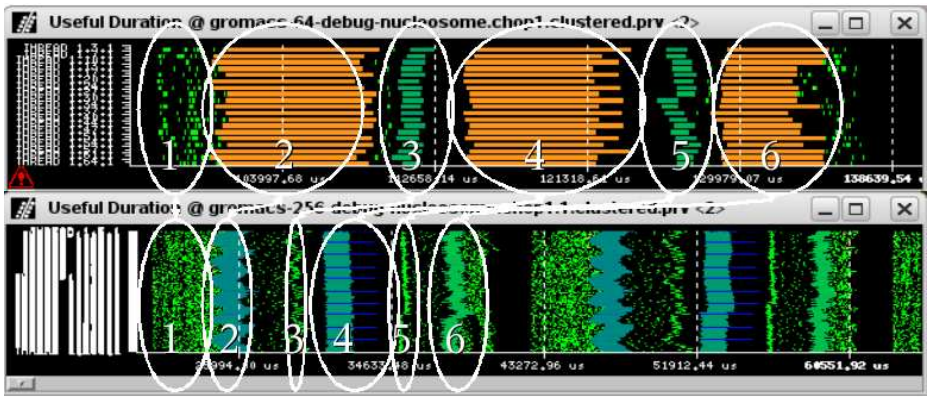

Fig. 5. Analysis of the FFTs duration scalability (top: 64 tasks case; botton: 256). Black corresponds to MPI. 6 ellipsis denote 6 code regions. Note the poor speed-up achieved by the fourth marked region due to imbalance.

On a perfect speed-up, the duration of a region with 256 tasks should be 1/4 of the execution with 64 tasks. Both windows have the same time scale, showing that on the interval where the 64 tasks run executes one iteration with 256 tasks executes a little bit more than two iterations. Observe that the main computation regions (zones 2, 4 and 6) obtain good time reductions. Zone 4 has a problem of imbalance: while with 64 tasks it has a small impact, in the 256 tasks case it becomes the bottleneck as this imbalance does not scale. Zone 1 is dominated by communications and as would be expected, it achieves a poor

speed-up. Within Paraver the callstack can be used to identify where any of those regions are in the source code. Notice that this part of the analysis was easily done opening the Paraver views while would be very complex using a profiler.

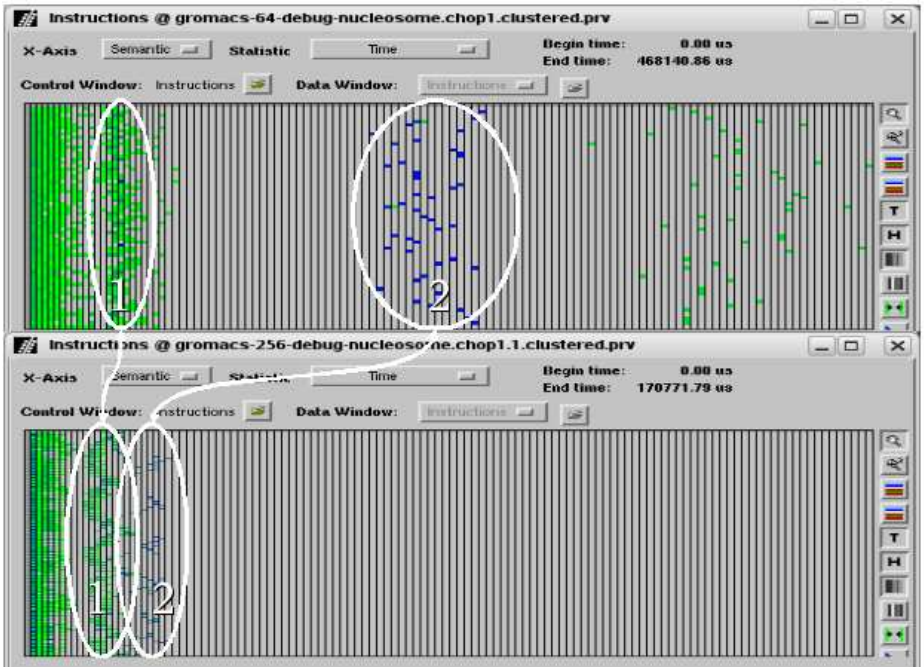

Fig. 6. Analysis of the computation (#instructions) imbalance (top: 64 tasks case; botton: 256). Note that region 1 #instructions is not reduced and imbalance increases.

Restating the hints given by the PeekPerf analysis, the metrics reported a computation balance problem. To analyze this issue, from PeekPerf we opened the instructions histogram. Again, this analysis is done with the timeline tool because we are interested on the distribution. With both histograms at the same scale, we obtain Figure 6. Paraver histograms have processes on the y-axis and the selected metric on the x-axis. In the instructions histogram, colored cells on the right side of the image represent areas with a large number of instructions, colored cells on the left side correspond to regions that execute few instructions.

If a code region is perfectly scalable with respect to #instructions, when tasks are multiplied by 4, the #instructions/task is reduced to 1/4, so both versions execute the same number of instructions (no code replication). This reduction is reflected on the histogram as a proportional shift to the left. With a perfect speed-up the displacement would be 3/4 on the x-axis. While zone 2 (with a high number of instructions) obtains a good reduction (the displacement is close to a perfect scenario), zone 1 not only obtains a poor reduction, but the imbalance is more noticeable when the number of tasks is increased.

Finally, to analyze the poor communication performance with 64 tasks, we extracted new metrics applied at the level of the MPI call lines. These metrics include for example time, number of calls, average duration and message size. Computing these metrics at the level of the call line allows the user to separate, for instance, different broadcasts depending on the calling context. Due to space limitations it is not possible to discuss the details but we would like to remark that the profiling view is the most appropriate tool for identifying the time spent in each MPI call and the largest MPI call. Notice that with the proposed methodology based on refinements, this new profiling would be generated only because a previous step identified it as a relevant peformance data.

6 Related work

There are many performance analysis tools, however most of them only support one analysis mode and do not allow for an easy integration with other tools.

TAU [6] is a framework for performance instrumentation, measurement and analysis. In addition to profile based data, TAU generates event traces and has utilities to export them to Jumpshot, Vampir or Paraver. The integration for the analysis phase is limited to a trace to profile converter. The Parallel Performance Wizard (PPW) [16] is a performance analysis framework for MPI, SHMEM, and UPC. Like TAU, it only supports to export trace data with the external viewers Jumpshot and Vampir without any further integration.

Vampir [15] is a timeline visualizer for Open Trace Format (OTF) traces. There is an internal, unpublished version of VampirServer with remote control via a DBUS interface [18]. An older version was once combined with the KAU OpenMP compiler tools (Guide, GuideView) to the VGV tool [17]. OpenMP constructs were instrumented by the Guide compiler adding statistics into the trace. Selecting a parallel region in the Vampir timeline triggered the corresponding Guideview displays. However, the tool never made it beyond a prototype.

On [19] CAPO (Computer Aided Parallelization tool) was interfaced with Paraver to support the parallelization of existing sequential codes. In the prototype, the user could jointly navigate through program structure and performance data information in order to make efficient optimization decisions. On [20] the prototype was extended with an expert system helping to filter, correlate and interpret the data gathered by the automatic parallelization and analysis tools.

7 Conclusions and Future Plans

We have interfaced KOJAK and PeekPerf with Paraver to facilitate the analysis benefiting from their complementary views. The analysis starts with the profiler that provides a summarized view to identify the most interesting metrics and/or code areas to focus the detailed analysis with the timeline tool. We have shown how the proposed environment can be used with two real-life examples.

The implementation is based on simple and generic interfaces. The signal mechanism implemented in Paraver is used from both KOJAK and PeekPerf.

We believe it is very important to promote the interoperability through generic mechanisms so extending its usage to a new tool requires no modification. As a proof of success, recently the modules to interoperate Paraver with PeekPerf have been extended to implement the same coupling between TAU and Paraver.

Acknowledgments. This paper has been partially supported by the Spanish Ministry of Science and Innovation (contract number TIN2007-60625) and the MareIncognito project under the IBM-BSC collaboration agreement.

References

1. A. Knpfer, R. Brendel, H. Brunst, H. Mix and W. E. Nagel: Introducing the Open Trace Format (OTF), *ICCS 2006*.
2. `http://www.peri-scidac.org/wiki/index.php/PERI_XML`
3. G. Cong, I. Chung, H. Wen, D. Klepacki, H. Murata, et al.: A holistic approach towards automated performance analysis and tuning, accepted in: Euro-Par, 2009.
4. F. Wolf, B. Mohr: Automatic performance analysis of hybrid MPI/OpenMP appl., *Journal of Systems Arch.*, Vol. 49, No. 10–11, 421–439, 2003.
5. F. Wolf et al.: Automatic analysis of inefficiency patterns in parallel appl., *Concurrency and Computation: Practice and Experience*, 19(11):1481–1496, 2007.
6. S. Shende, A. D. Malony, The TAU Parallel Performance System, *Int. Journal of High Performance Computing Appl.*, Vol 20, 2:287–331, 2006.
7. M. Geimer, F. Wolf et al.: The Scalasca performance toolset architecture. Concurrency and Computation: Practice and Experience, 22(6):702-719, April 2010.
8. M. Geimer et al.: Scalable Collation and Pres. of Call-Path Profile Data with CUBE, in: Proc. of the Parallel Computing Conf. (ParCo), NIC series Vol 38 645–652, 2007.
9. V. Pillet et al. : PARAVER: A Tool to Visualize and Analyze Parallel Code, in: 18th World OCCAM and Transputer User Group Technical Meeting. April 1995 `http://www.bsc.es/paraver`
10. J. Labarta, J. Gimenez: Performance Analysis: Till When an Art, in: Parallel Processing for Scientific Computing. M.A. Herroux, et al. (Eds.), SIAM 2006.
11. G. Jost, J. Labarta, J. Gimenez. Paramedir: A Tool for Programmable Performance Analysis in: Int. Conf. on Computational Science (ICCS'04), June 2004.
12. Weather Research Forecast code. `http://www.wrf-model.org/`.
13. GROningen MAchine for Chemical Simulations. `http://www.gromacs.org`.
14. M. Casas-Guix, R. M. Badia, J. Labarta: Automatic analysis of speedup of MPI appl., in: Int. Conf. on Supercomputing (ICS 2008), June 2008.
15. H. Brunst, W. E. Nagel: Scalable performance analysis of parallel systems: Concepts and experiences, in: Proc. of the Parallel Computing Conf., Elsevier, 2003.
16. H. Su et al. Parallel Performance Wizard: A Performance Analysis Tool for Partitioned Global-Address-Space Programming, PDSEC-IPDPS 2008, April 2008.
17. J. Hoeflinger et al., An Integrated Performance Visualizer for MPI/OpenMP Programs, *Workshop on OpenMP Appl. and Tools (WOMPAT 2001)*, July, 2001.
18. T. William et al.: Enhanced Performance Analysis of Multi-core Appl. with an Integrated Tool-chain, in: Mini.Symposium ParMa,Parco Proceedings, 2009.
19. G. Jost, H. Jin et al. :. Interfacing Computer Aided Parallelization and Performance Analysis, in: Int. Conf. on Computational Science (ICCS'03), June 2003.
20. G. Jost, R. Chun, H. Jin, J. Labarta, J. Gimenez: An Expert Asssistant for Computer Aided Parallelization, in: PARA 2004.