Chapter 4
Recent Advances in Periscope for Performance Analysis and Tuning

Yury Oleynik, Robert Mijaković, Isaías A. Comprés Ureña, Michael Firbach, and Michael Gerndt

Abstract State of the art High Performance Computing (HPC) systems pose considerable programming challenges to application developers when tuning their applications. Periscope toolkit is one of a number of performance engineering instruments supporting application programmers in meeting those challenges. Due to the variety of architectures, programming models, runtime environments, and compilers on those systems, programmers need to apply multiple tools to understand and improve program performance. In this paper, we present the latest developments in Periscope aiming at (1) improving its interoperability and integration with other tools, (2) integrating automatic tuning support with performance analysis and (3) further extending performance analysis capabilities. The add-on for Periscope, called PAThWay, allows for the integration of multiple tools into performance tuning workflows. Further, Periscope is currently being extended with the ability to automatically tune parallel applications with respect to execution performance and energy consumption. And finally, new analysis capabilities were added to Periscope for the automatic evaluation of the temporal performance behavior of long-running applications.

4.1 Introduction

Programmers have always been required to write correct algorithms and, depending on the problem, this may already be challenging and time consuming. When working in projects where performance is part of the requirements, additional challenges rise from heterogeneity, non-uniform memory access and parallelism of the modern High Performance Computing (HPC) systems. Programmers need to find ways to map the software to the available memory and computing elements. Initially, the application is mapped to the hardware based on previous experience and guesswork.

Y. Oleynik (✉) • R. Mijaković • I.A. Comprés Ureña • M. Firbach • M. Gerndt
Institute of Informatics, Technical University of Munich (TUM),
Boltzmannstr. 3, 85748 Garching, Germany
e-mail: oleynik@in.tum.de

© Springer International Publishing Switzerland 2014
A. Knüpfer et al. (eds.), *Tools for High Performance Computing 2013*,
DOI 10.1007/978-3-319-08144-1_4

The tuning process of a parallel application involves several tools that specialize on different aspects that affect performance. These tools may be hardware or programming model specific. Even under a specific hardware and programming model, it is common that a tool will target only a subset of aspects that affect performance. For these reasons, programmers rely on a larger collection of tools to optimize their parallel applications.

The overall performance of a parallel application depends on its single node performance, its scalability and how well its load is balanced. On a single node, its performance will depend on the compilers available, the compiler flags and the programming models. In some systems, especially if heterogeneous, several compilers and programming models are used; this multiplies the tuning effort required (although it is not easy to quantify this effort, since it is a human factor). Typically, the scalability of the application depends on how much information has to be shared across nodes and the performance of the communication library and network. Finding a good partitioning scheme to balance the load and minimize communication is also essential.

Furthermore, the majority of scientific codes perform simulations iteratively, where a main loop of an application, sometimes referred to as the progress loop, executes the same computational kernel many times. Though the code being executed is often identical, application performance across the iterations vary significantly. As a consequence, the impact and location of performance bottlenecks, or degradations in this case, are time dependent. Taken into account that some simulations are run for several days and even month, performance analysis tools should be able to efficiently handle temporal dimension of performance measurements.

Recent developments in Periscope are focused on assisting application developers in overcoming the challenges described above. This is achieved

by enabling Periscope to answer the following typical user questions:

1. How to integrate the use of various tools in a single, well-defined workflow?
2. What is the optimal combination of tunable parameters for an application?
3. What are the most relevant runtime performance degradations?

In Sect. 4.2 we present PAThWay, which is part of the Periscope toolkit. It automates performance engineering workflows and is designed to methodically provide answers to the first question. Section 4.3 is dedicated to the Periscope Tuning Framework (PTF), which implements multiple specialized tuning plugins in order to automatically search for optima and thus answer the second question. Periscope's performance dynamics analysis, described in Sect. 4.4, answers the third question by searching for temporal degradations of performance in long running applications. We present an overview of the related work in Sect. 4.5 and, finally, we draw conclusions and perspectives for future work in Sect. 4.6.

4.2 Cross-Experiment Analysis and Tuning Using Workflows

A considerable challenge in optimizing HPC applications is the management of the overall optimization process. Which tools are used, and when? How is the collected performance data analyzed and archived? What were the performance characteristics of the application 3 weeks ago? With PAThWay, we are reaching out for the next level of integration, abstracting details of the HPC system and integrating the tools usage with human tasks.

PAThWay is a high-level tool that uses formal workflow definitions to provide structure to the overall optimization process. Major goals of this development are:

- Workflow automation
- System and Tools abstraction
- Experiment history

4.2.1 Workflow Automation

Whether formally defined or not, the process of performance tuning usually follows a specific workflow, which itself can be broken down into several activities. For example, typical activities involve making a snapshot of the source code for future reference, code instrumentation, job submission on the HPC system, etc. A simplified example of a performance tuning workflow is illustrated in Fig. 4.1.

The goal of the PAThWay development is to automate these activities to a large extent. This helps the application developer in two ways: First, by automating these repetitive and tedious tasks, time is saved and the number of unnecessary errors

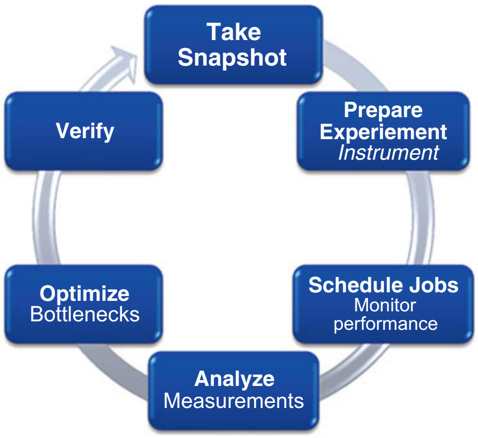
Take Snapshot
Prepare Experiment *Instrument*
Schedule Jobs Monitor performance
Analyze Measurements
Optimize Bottlenecks
Verify

Fig. 4.1 Performance tuning workflow

is reduced. Second, the use of such automated workflows provides structure to the optimization process, encouraging methodical performance engineering.

Workflows can be defined using the standard Business Process Model and Notation (BPMN) format, for which there are graphical editors. Workflows have successfully been defined for common tasks, such as scalability analysis, cross-platform analysis and single-core profiling. Ideally, all activities that are performed during optimization, have a representation within the workflow. Once defined, the workflows can be *executed*, which triggers its individual activities. Workflow execution itself is performed by the jBPM workflow engine,[1] whereas the individual activities can invoke arbitrary tools or even human tasks.

By integrating all optimization steps in a single workflow, we emphasize structured and methodical performance engineering, which we believe is currently under-utilized in real-world HPC application development.

4.2.2 System and Tools Abstraction

The variety of tools that need to be integrated in an optimization workflow is a major complication for HPC application developers. The development of the joint measurement infrastructure Score-P [11] was motivated by this issue. Invoking tools from a workflow defined in PAThWay allows non-experts to use these tools and optimize their application, which more likely represents their actual area of expertise, e.g. physics simulation. In addition, workflows can use the tools to build higher-level constructs like scalability analysis, without requiring the user to have in-depth knowledge of the tools used on a low-level.

There are also several differences between HPC systems, which make cross-platform analysis difficult. We use the Parallel Tools Platform from Eclipse to provide the means to communicate with a variety of platforms. Jobs can be scheduled and monitored transparently on SLURM, LoadLeveler and MPICH2-enabled systems at the time of this writing.

Some tasks, such as analyzing tools output and optimizing the application, can only be done by humans. We call these tasks *human tasks* and alert the user when such a task needs to be performed according to the current state of workflow execution.

4.2.3 Experiment History

PAThWay records so-called *experiments* that result from workflow execution. An experiment is the notion of a job scheduled on an HPC system, its status and output,

[1] http://www.jboss.org/jbpm/

combined with a source snapshot and tools configuration. The experiment browser allows for later inspection of the experiments' outcome. There are plans for creating a variety of tools that can automatically draw conclusions from the experiment history in the future, such as analyzing cross-experiment performance dynamics.

4.3 Periscope Tuning Framework

Parallel software usually has several parameters that can be adjusted for better performance while keeping results correct. Examples of such parameters are compiler flags used during compilation and configuration parameters of communication libraries. Parallel software can also include tuning parameters, such as: algorithm selection, block sizes, partition factors, etc. There are also hardware settings that can be manipulated, such as performance governors and CPU frequencies (usually only available to administrators, but exposed in some systems through user level services).

The right combination of software and hardware parameters, that are best or at least good enough, is not always clear. There is typically a time consuming process of guesswork and collection of empirical data, that is done manually by the application developers. Given the large number of degrees of freedom, and therefore the large amount of time to empirically search for optimal or good combinations of these software and hardware parameter combinations, automated tools for this task are currently a necessity.

In this section we present an extension to Periscope, called the Periscope Tuning Framework (PTF). This framework allows for the implementation of plugins that automate the analysis and tuning process of parallel software. The plugins are built as shared objects that are loaded at runtime. With this approach, companies or individuals can develop custom plugins independently of the PTF core developers.

PTF follows the main Periscope principles, i.e., the use of formalized expert knowledge in form of performance properties and strategies, automatic execution, online search based on program phases, and distributed processing.

Periscope is extended by a number of tuning plugins that fall into two categories: online and semi-online plugins. An online tuning plugin is one that performs transformations to the application and/or the execution environment without requiring a restart of the application. In contrast, a semi-online tuning plugin will require one or more restarts of the application to achieve these effects.

Figure 4.2 illustrates the control flow of PTF. The tuning process is started with a preprocessing step. In this step, the application source files are instrumented and static analyses are performed on them. Periscope applies source level instrumentation for C/C++ and Fortran. The instrumenter generates a SIR file (Standard Intermediate Representation) that includes static information such as the instrumented code regions and their nesting information. The instrumentation and the static analysis process have been extended to support HMPP, OpenCL, and common parallel patterns (such as master-worker and pipelines).

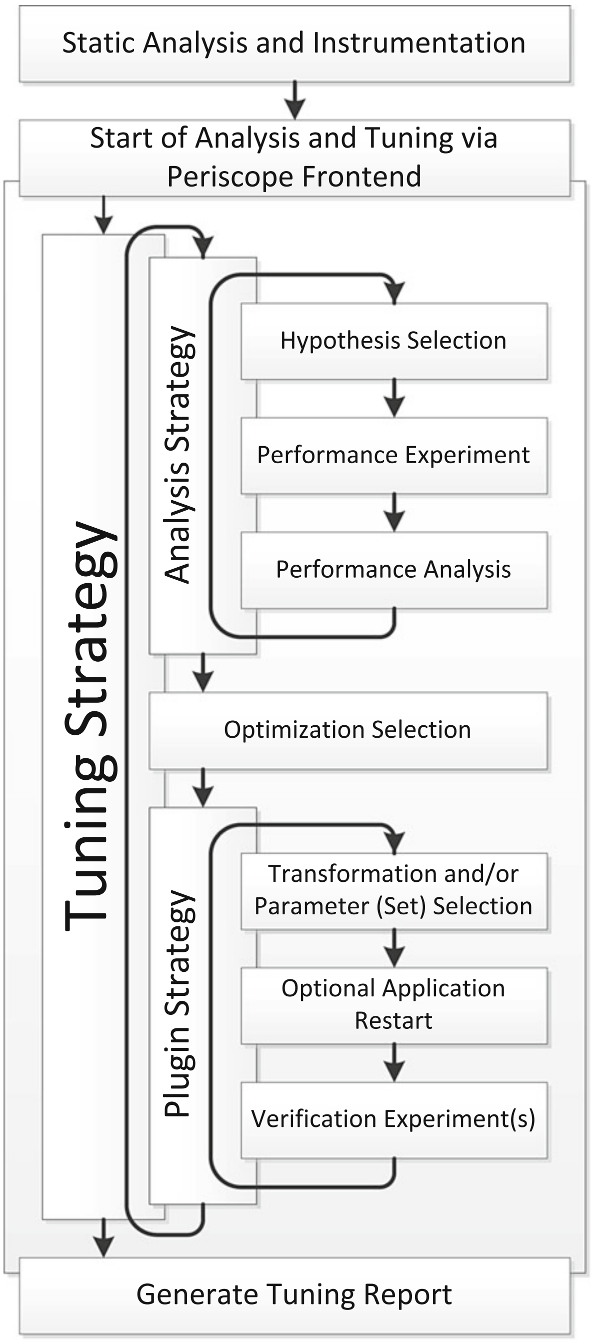
Static Analysis and Instrumentation
Start of Analysis and Tuning via Periscope Frontend
Tuning Strategy
Analysis Strategy
Hypothesis Selection
Performance Experiment
Performance Analysis
Optimization Selection
Plugin Strategy
Transformation and/or Parameter (Set) Selection
Optional Application Restart
Verification Experiment(s)
Generate Tuning Report

Fig. 4.2 Tuning control flow

The tuning is started via the Periscope frontend, either interactively or in a batch job. As done in Periscope, the application is started by the frontend along with the required agent hierarchy (given the size of the application and the hardware platform).

Periscope uses analysis strategies (e.g., MPI, OpenMP or single core analysis strategies) to guide the search for performance properties. This overall control strategy has now become part of a higher level tuning strategy. The tuning strategy controls the sequence of analyses and tuning steps.

Typically, an analysis can be used to determine application properties that will allow the plugin to restrict its search space. The search space can also be restricted by other plugin specific means, such as expert knowledge or machine learning approaches.

Once the search is done and an optimum or good enough solution is found, the tuning process is finished. A tuning report is then generated and presented to the user through standard output or files. This report documents the found performance properties as well as the tuning actions recommended by PTF. These tuning actions can be integrated into the application such that subsequent production runs will be more efficient.

Several plugins are being developed concurrently with the PTF. These include:

Compiler Flags Selection: Improves single node performance by evaluating possible valid compiler flags for a parallel application. The flags can be specified per file in the build and depend on the compilers available in the target system. For applications where their correctness depends on specific flags, these can be enforced.

High-Level Parallel Patterns for GPGPU Systems: Optimizes common parallel patterns on GPU based accelerators. Currently focuses on the pipeline pattern with OpenCL or CUDA. The total pipeline length and with of specific stages are optimized.

Hybrid Manycore HMPP Codelets: Evaluates the best selection of HMPP codelets for a specific application and hardware combination. In the HMPP application, several codelets that perform the same operation are prepared. The specific codelets and parameters are then evaluated and the best chosen.

Energy Consumption via DVFS: Performs a multi-objective optimization of performance and energy. For measurement, it relies on the use of hardware counters, timers and energy measurement hardware. The result selected will depend on the target desired performance and energy budget.

Master-Worker Pattern with MPI: Optimizes MPI applications that use the master-worker pattern. It evaluates the number of workers, the number of masters and the size of work packets.

MPI Runtime Parameters: Optimizes MPI communication performance for an application. It first analyses the application to find the location and MPI operations that take the most time. It then proceed to tune runtime parameters that are specific to these operations. The parameters include limits for the different communication protocols used by the library, as well as the selection of internal algorithms. Depends heavily on the available MPI implementations.

4.4 Runtime Performance Dynamics

Runtime Performance Dynamics Analysis is a new type of analysis supported by Periscope. It is based on dynamic phase profiling [13] and scale-space filtering [19] with consequent summarization of the resulting multi-scale representation. The algorithm can be split in the following steps: *measurements collection*, *preprocessing*, *qualitative summarization* and *property detection*. We explain the techniques used and the analysis algorithm in the following paragraphs.

4.4.1 Measurement Collection

Dynamic profiles collected using Score-P are used as an input for the algorithm. A dynamic profile is a time-series of measurements of some metric m sampled for each iteration of the application's progress loop, measured in a source-code region r on a process p. Such representations capture the evolution of the performance over the iterations of the simulation and attributes a time-series with a specific source-code location and process. An example dynamic profile time-series is plotted with blue color on Fig. 4.3.

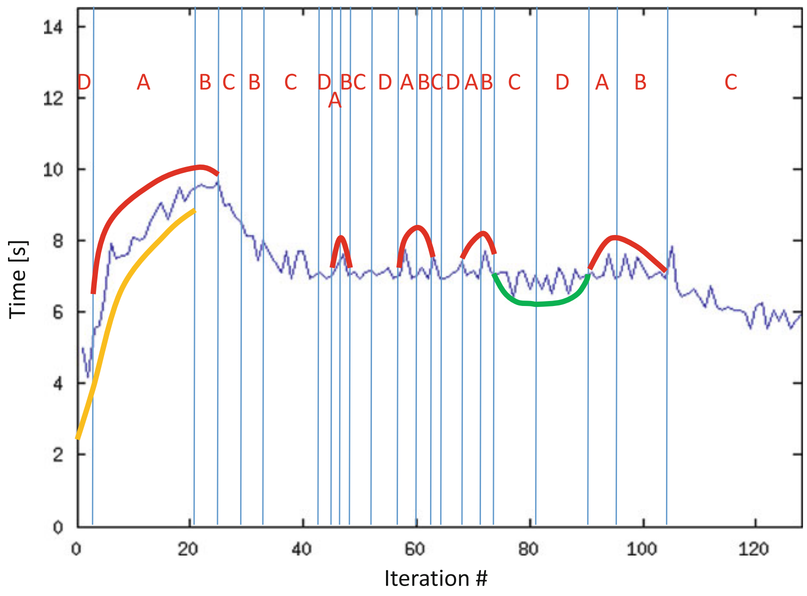

Fig. 4.3 Dynamic profile time-series overlaid with the extracted maximum-stability intervals and properties search results. The *red*, *green* and *yellow curves* highlight the results found for the queries "all peaks", "the most distinctive valley" and "the tallest increase", respectively

4.4.2 Preprocessing

Though the dynamic profiles capture dynamic behavior of the application performance, this information is implicit and requires manual interpretation by the user. A representation in terms of the intervals enclosed by zero-crossings of the first and second order derivatives offers an alternative qualitative semantic which is natural for the human comprehension. We apply alphabetical notation to denote the seven possible intervals defined by the combination of the signs of the first and second derivative: “concavely increase” – A, “concavely decrease” – B, “convexly decrease” – C, “convexly increase” – D, “linear increase” – E, “linear decrease” – F, “constant” – G.

However, the process of local extrema detection requires proper selection of the scale or the neighborhood over which the derivative is taken. A systematic way of handling the scale issue can be obtained by scale-space filtering. Scale-space filtering is a process of incremental low-pass filtering with the Gaussian filter of variable standard deviation, also known as the scale parameter. The resulting sequence of smoothed versions of the original time-series at different scales form a scale-space image. An important property of the scale-space image is that extrema persist over multiple scales and form a tree. Therefore, by detecting local extrema and connecting them from coarse to fine scales we can build a multi-scale hierarchy of intervals enclosed by two nearest extrema, called an interval tree.

4.4.3 Qualitative Summarization

In the next step we classify each interval in the tree to one of the seven possible shape types, defined above, and build a tree of qualitatively described segments. Each node of the tree is represented by a character denoting the shape of the interval and additional quantitative information, such as location, duration and magnitude of the corresponding interval.

It was shown empirically [19] that the number of scales over which an interval persists corresponds to the perceptual salience. We rely on this parameter, called stability, in order to quantify the relevance of the corresponding segments in the tree. Furthermore, we use the stability value to perform additional summarization by descending the tree in a breadth-first fashion and selecting the level with the maximum stability.

4.4.4 Automatic Search for Performance Dynamics Properties

The maximum stability level represents a dynamic profile time-series as a sequence of segments, each characterized by a qualitative shape descriptor, a stability value

Table 4.1 Example performance dynamics properties, corresponding queries and the color used to highlight resulting segment on the Fig. 4.3

Property	Query	Color
Local peaks	Select all "AB" sequences	Red
The most distinctive valley	Select "CD" sequence with max(stability)	Green
The tallest increase	Select "DA" sequence with max(magnitude)	Yellow

and supplementary quantitative information. Such representation allows efficient search for patterns specified with the following parameters:

- Qualitative shape specification (i.e. combination of shape descriptors)
- Perceptual salience (i.e. stability value)
- Magnitude and/or duration of the segment
- A combination of the parameters above

We demonstrate our technique on an example dynamic profile time-series plotted with blue color on the Fig. 4.3. The time-series represents samples of the metric "Execution time" measured for the body of the main loop for the first 128 iterations. The qualitative descriptors, represented with red characters, of the extracted maximum stability segments and their borders are overlaid on top of the time-series.

The proposed algorithm automatically answers the third question stated in the introduction section. In this case, the phrase "relevant degradation" can be defined in terms of the parameters described above. This allows very flexible specification of the performance dynamics properties, which can then be automatically searched in time-series. Table 4.1 shows three example properties and the corresponding queries. The resulting segments are highlighted with red, green and blue color on the Fig. 4.3.

4.5 Related Work

The complexity of today's parallel architectures has a significant impact on application performance. In order to avoid wasting energy and money due to low utilization of processors, developers have been investing significant time into tuning their codes. However, tuning implies searching for the best combination of code transformations and parameter settings of the execution environment, which can be fairly complicated. Thus, much research has been dedicated to the areas of performance analysis and auto-tuning.

The explored techniques, similar in approach to ours, can be grouped into the following categories:

- Tools that utilize techniques (such as optimization-space exploration [18], non-parametric inferential statistics [8], optimization orchestration [15] and machine learning [6, 12]) to automatically analyze alternative compiler optimizations and search for their optimal combinations; and

- Auto-tuners (such as Active Harmony [4, 17] and MaSiF [5]) that search a space of application-level parameters that are known to impact performance. The search can be directed with the use of modeling or machine learning techniques [14]; and
- Frameworks that combine ideas from all the other groups (such as Autopilot [16] and the Insieme Compiler and Runtime infrastructure [10]).

Performance analysis and tuning are currently supported via separate tools. Periscope Tuning Framework aims at bridging this gap and integrating support for both steps in a single tuning framework.

Dynamic performance behavior is typically a subject for analysis using tracing techniques. However, due to scalability issues, profiling becomes a more attractive alternative. In particular, dynamic phase profile is used for performance dynamics evaluation and was first introduced in TAU [13]. Though the technique captures temporal performance evolution, the information about the relevant changes is implicit and requires manual interpretation by the user. In our work we build on top of this approach and apply scale-space filtering [19] with consequent qualitative summarization of dynamic profiles in order to automatically extract relevant performance changes. A different combination of signal processing algorithms is used in [3], where wavelet, morphological analysis and autocorrelation is used to extract periodical application phases from traces. Also in [7] clustering analysis and Multiple Sequence Alignment algorithm were used to automatically detect the computational structure of an SPMD program.

Using workflow engines to guide the overall performance tuning process and to integrate all individual tools with human tasks seems to be a rather unique approach within the HPC community. Although workflows are today more commonly used in scientific work in general [2], their use is much more common in business applications, which is why standards are usually centered around those. Popular standards include BPEL (Business Process Execution Language) [9], XPDL (XML Process Definition Language) and BPMN (Business Process Modeling and Notation) [1], of which the latter is also used by PAThWay.

4.6 Conclusion and Future Work

While working on performance analysis and optimization of HPC applications, programmers are faced with a wide spectrum of issues. These range from questions on how to organize performance experiments, choice of optimal configurations for the application and the runtime, to tracking and pinpointing dynamically appearing and disappearing performance bottlenecks. Recent developments in Periscope toolkit are aimed at assisting the programmer in handling these challenges by a number of technologies.

A new performance engineering automation tool, called PAThWay, is designed to assist the programmer in defining and automating the typical workflows involved in this process. The tool automates tasks like job submission, monitoring and retrieving results of the jobs on remote HPC systems; book-keeping of the carried out experiments and their configurations, maintenance of the documentation. In the future, we plan to extend the functionality of PAThWay for supporting analysis of cross-experiment performance dynamics.

In order to support programmers in configuring the application and the execution environment, Periscope is extended with a tuning interface and a number of plugins. The latter will automatically analyze application performance and based on the detected inefficiencies perform a search for the optimal parameters settings, where the parameters range from compiler flags to MPI runtime settings. In the future work, we plan to implement meta-plugins that will orchestrate other plugins in tuning multiple orthogonal parameter sets.

Finally, we present a new analysis algorithm which is aimed at supporting programmers in analyzing performance of long-running applications. A particular challenge of such analyses is that the location and severity of performance bottlenecks is time dependent. Using dynamic profiling and multi-scale analysis techniques we are able to search for complex patterns in the temporal performance behavior. For the future work we plan to extend the algorithm to search for similarities in temporal behavior observed on different processes of the parallel applications.

Acknowledgements The authors thank the European Union for supporting AutoTune project under the Seventh Framework Programme, grant no. 288038 and German Federal Ministry of Research and Education (BMBF) for supporting LMAC project under the Grant No. 01IH11006F.

References

1. Allweyer, T.: BPMN 2.0: Introduction to the Standard for Business Process Modeling. BoD–Books on Demand, Norderstedt (2010)
2. Barker, A., Van Hemert, J.: Scientific workflow: a survey and research directions. In: Parallel Processing and Applied Mathematics, pp. 746–753. Springer, Berlin/New York (2008)
3. Casas, M., Badia, R.M., Labarta, J.: Automatic phase detection and structure extraction of MPI applications. Int. J. High Perform. Comput. Appl. **24**(3), 335–360 (Aug 2010). http://dx.doi.org/10.1177/1094342009360039
4. Chung, I.H., Hollingsworth, J.: Using information from prior runs to improve automated tuning systems. In: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, SC '04, Pittsburgh, pp. 30–. IEEE Computer Society, Washington, DC (2004). http://dx.doi.org/10.1109/SC.2004.65
5. Collins, A., Fensch, C., Leather, H.: MaSiF: machine learning guided auto-tuning of parallel skeletons. In: Yew, P.C., Cho, S., DeRose, L., Lilja, D. (eds.) PACT, Minneapolis, pp. 437–438. ACM (2012). http://dblp.uni-trier.de/db/conf/IEEEpact/pact2012.html#CollinsFL12
6. Fursin, G., Kashnikov, Y., Wahid, A., Chamski, M.Z., Temam, O., Namolaru, M., Yom-tov, E., Mendelson, B., Zaks, A., Courtois, E., Bodin, F., Barnard, P., Ashton, E., Bonilla, E., Thomson, J., Williams, C.: Milepost GCC: machine learning enabled self-tuning compiler (2009)

7. Gonzalez, J., Gimenez, J., Labarta, J.: Automatic evaluation of the computation structure of parallel applications. In: 2009 International Conference on Parallel and Distributed Computing, Applications and Technologies, Higashi Hiroshima, pp. 138–145. IEEE (2009)
8. Haneda, M., Knijnenburg, P., Wijshoff, H.: Automatic selection of compiler options using non-parametric inferential statistics. In: International Conference on Parallel Architectures and Compilation Techniques, Saint Louis, pp. 123–132 (2005)
9. Jordan, D., Evdemon, J., Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., et al.: Web Services Business Process Execution Language Version 2.0. OASIS Standard 11 (2007)
10. Jordan, H., Thoman, P., Durillo, J., Pellegrini, S., Gschwandtner, P., Fahringer, T., Moritsch, H.: A multi-objective auto-tuning framework for parallel codes. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '12, Salt Lake City. IEEE Computer Society Press, Los Alamitos, pp. 10:1–10:12 (2012). http://dl.acm.org/citation.cfm?id=2388996.2389010
11. Knüpfer, A., Rössel, C., Mey, D., Biersdorff, S., Diethelm, K., Eschweiler, D., Geimer, M., Gerndt, M., Lorenz, D., Malony, A., et al.: Score-P: a joint performance measurement run-time infrastructure for periscope, scalasca, TAU, and vampir. In: Tools for High Performance Computing 2011, pp. 79–91. Springer, Berlin/Heidelberg (2012)
12. Leather, H., Bonilla, E.: Automatic feature generation for machine learning based optimizing compilation. In: Code Generation and Optimization (CGO), Seattle, pp. 81–91 (2009)
13. Malony, A.D., Shende, S.S., Morris, A.: Phase-based parallel performance profiling. In: G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, E.Z. (eds.) Proceedings of the International Conference ParCo 2005, Malaga. NIC Series, vol. 33, pp. 203–210. John von Neumann Institute for Computing, Julich, (2006)
14. Nelson, Y., Bansal, B., Hall, M., Nakano, A., Lerman, K.: Model-guided performance tuning of parameter values: a case study with molecular dynamics visualization. In: International Parallel and Distributed Processing Symposium, Miami, pp. 1–8 (2008)
15. Pan, Z., Eigenmann, R.: Fast and effective orchestration of compiler optimizations for automatic performance tuning. In: Proceedings of the International Symposium on Code Generation and Optimization (CGO), New York, pp. 319–332 (2006)
16. Ribler, R., Vetter, J., Simitci, H., Reed, D.: Autopilot: adaptive control of distributed applications. In: Proceedings of the 7th IEEE Symposium on High-Performance Distributed Computing, Chicago, pp. 172–179 (1998)
17. Tiwari, A., Chen, C., Chame, J., Hall, M., Hollingsworth, J.: A scalable auto-tuning framework for compiler optimization. In: International Parallel and Distributed Processing Symposium, Rome, pp. 1–12 (2009)
18. Triantafyllis, S., Vachharajani, M., Vachharajani, N., August, D.: Compiler optimization-space exploration. In: Proceedings of the international symposium on Code generation and optimization, San Francisco, pp. 204–215. IEEE Computer Society (2003)
19. Witkin, A.: Scale-space filtering: a new approach to multi-scale description. In: IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP'84, San Diego, vol. 9, pp. 150–153. IEEE (1984)