D3.2.2 – Adaptive runtime support design document (Update)

WP3: Development Environment

Project Acronym	CRESTA
Project Title	Collaborative Research Into Exascale Systemware, Tools and Applications
Project Number	287703
Instrument	Collaborative project
Thematic Priority	ICT-2011.9.13 Exascale computing, software and simulation

Due date:	M10
Submission date:	30/09/2013
Project start date:	01/10/2011
Project duration:	36 months
Deliverable lead organisation	Royal Institute of Technology (KTH)
Version:	1.0
Status	Final
Author(s):	Xavier Aguilar (KTH), Michael Schliephake (KTH)
Reviewer(s)	Harvey Richardson (CRAY UK), Michael Gienger (HLRS)

Dissemination level	
<PU/PP/RE/CO>	*PU - Public*

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	05/09/2013	Text draft	Xavier Aguilar (KTH) Michael Schliephake (KTH)
1.0	24/09/2013	Final version for submission after review comments addressed	Xavier Aguilar (KTH) Michael Schliephake (KTH)

Table of Contents

1 Executive Summary

In Subtask 3.2.2 “Hybrid and adaptive runtime systems” an experimental runtime system will be developed that will explore the power of adaptive runtime support for exascale applications”.

In the deliverable D3.1 “State of the art and gap analysis - Development environment”, CRESTA performed an analysis of existing approaches in the field as well as technical boundary conditions and requirements.

The deliverable D3.2.1 provided a design of a runtime system that aims to develop further approaches to adapt simulation applications dynamically in the best way to computer systems and to extend such approaches to upcoming exascale architectures. This deliverable therefore proposed an adaptive runtime-support design where simulation applications based on a task-orientated programming model with hierarchical tasks are combined with runtime supporting performance analysis and runtime administration enabling an increased efficiency of large-scale numerical simulations.

To this updated deliverable D3.2.2 have been added conclusions that could be drawn from the ongoing implementation of the runtime administration and monitoring components. It points out that the overhead of the runtime system in a typical molecular dynamics simulation has to be expected at about 5% allowing noticeable performance improvements of the overall runtime. A new performance monitoring API has been developed with the aim to allow the use of IPM with low overhead in the runtime system.

2 Introduction

This deliverable describes the design of an adaptive runtime-system with the purpose of improving the match between the software and the hardware used for its execution in the field of scientific simulations.

We sketch briefly some technological trends and problems of simulation applications on future exascale computer systems in Section 3. Section 4 specifies requirements on an adaptive runtime-system, whilst the principal software design based on the requirements and experience from first implementation activities has been described in Section 5. Finally, Section 6 describes the test plan to be used to measure the achievements from the tool development.

2.1 Glossary of Acronyms

Acronym	Definition
API	Application Programming Interface
D	Deliverable
EC	European Commission
HPC	High Performance Computing
Mon-C	Monitoring Component
Pan-C	Performance analysis Component
Rta-C	Runtime administration Component
WP	Work Package

3 Rationale for Software Design

Massive parallel computing is a major driving force in computational science and scientific discovery and the systems are getting larger and more complex day-by-day. Future exascale systems will be composed of hundreds of thousands of cores and will have complex designs that are likely to use heterogeneous technologies. It will, therefore, be a challenging task to achieve good application and system performance. In addition, the increasing complexity of these machines will also increase the complexity of the applications and operating systems.

These new kinds of heterogeneous systems pose new challenges in the development and porting of applications, and require significant effort to achieve the systems peak capability.[1] Human experts who optimise and port applications for these systems need to be complemented with intelligent software tools providing support in a transparent and automated way. These tools should also help to detect and solve various kinds of performance problems, not only overall speed-up, but also system-throughput, power consumption, etc.

In order to achieve good performance, typically highly system specific features have to be exploited, which often means that best practices in programming and software development have to be relaxed and the resulting code is difficult to port to different systems.

We therefore require new tools that ease the task of building portable applications for a broad range of HPC infrastructures in a modular way.[2] They should support the reuse of building blocks hiding the different technologies as well as implementing algorithms in the best way for the selected kind of technology. In that way, the resulting software would become more robust, reusable and maintainable.

The design of an adaptive runtime system addressing these challenges is presented in this deliverable. The runtime system consists of a resource manager, a library for runtime administration of parallel applications, and a performance monitoring and analysis tool. The design is based on a task model that will help programmers to exploit the parallelism of their applications. The main idea is to have a system capable of reacting automatically to the application's behaviour, that is, supporting a high parallel efficiency and improving the performance of the application based on the combined use of hints provided by the programmer as well as the transparent supervision of the program execution.

4 Conceptual Approach

4.1 Requirements

One of the hardest requirements in the development of simulation applications is their adaptation to different computer systems due to the varying technical parameters that have a huge influence to the numerical performance: cache- and memory hierarchies, the number of cores per CPU, the number of sockets per node, and the characteristics of the interconnect network.

Today, optimisations are typically implemented directly in the code causing limitations in performance portability and higher maintenance costs. Due to the similar microprocessor architectures primarily used today this problem was not, up until now, too critical, but with the advent of more heterogeneous architectures this is increasingly becoming more important. Moreover, different application classes require different optimisation strategies making the development of generalised tools difficult. From a software engineering viewpoint development tools must support the implementation of reusable software components that help to use the systems efficiently and decouple the supporting program parts from the numerical algorithm. This requirement always has to be seen together with the need to introduce only limited runtime overhead.

An important requirement for a tool development is the reuse of existing application codes often implemented in Fortran or C. The introduction of new software tools should allow its incremental adoption, keeping the need for reimplementation or adaptation of existing code to a minimum. A further requirement connected to the previous one is the wish that software tools support an adaptive use of best practices, which otherwise would not be applied due to prohibitive implementation effort.

4.2 Hardware and Software Models in the Runtime-System

Based on recent hardware developments we can summarise the following requirements of numerical applications:

- Integration of data and task parallelism,
- Use of multi-level parallelism in the algorithm design,
- Development of algorithms with a high degree of parallel executable tasks, which have a moderate size, can be created very quickly, and avoid global communication operations,
- Usage of multi-threading, asynchronous communication and one-sided communication,
- Consideration of the increasing depth of the memory hierarchy,

- Optimised scheduling and mapping taking into account chip-architectures, memory hierarchies, internal communication abilities, etc. to provide a higher degree of parallelism and decrease memory and communication bandwidth usage.

These requirements are tackled in many research efforts and projects. We analysed these in the CRESTA deliverable D3.1 'State of the art and gap analysis - Development environment' and only highlight a small number here. We looked at the runtime-systems StarPU, StarSs, and ForestGOMP. The tool PerfMiner is one of the starting points for the development of performance analysis tools. Furthermore, promising approaches to derive valuable knowledge of the performance behaviour in codes have been analysed. More information can be found in the deliverable mentioned above as well as in the original literature [3-14].

The runtime-system developed in CRESTA supports a task-oriented programming model featuring hierarchical multiprocessor tasks. Such tasks are computational units that can be also parallel in themselves and can be subdivided hierarchically again into subtasks. The example in figure 1 shows which tasks could be defined in a typical algorithm of a molecular dynamic simulation. The hierarchical nature of the computational tasks and their inner parallelism that should be supported by the runtime system is clearly visible. Such a task model matches how programmers typically express parallelism during algorithm design and in program descriptions. The runtime system overcomes with its task model the problem that this parallelism is not easily

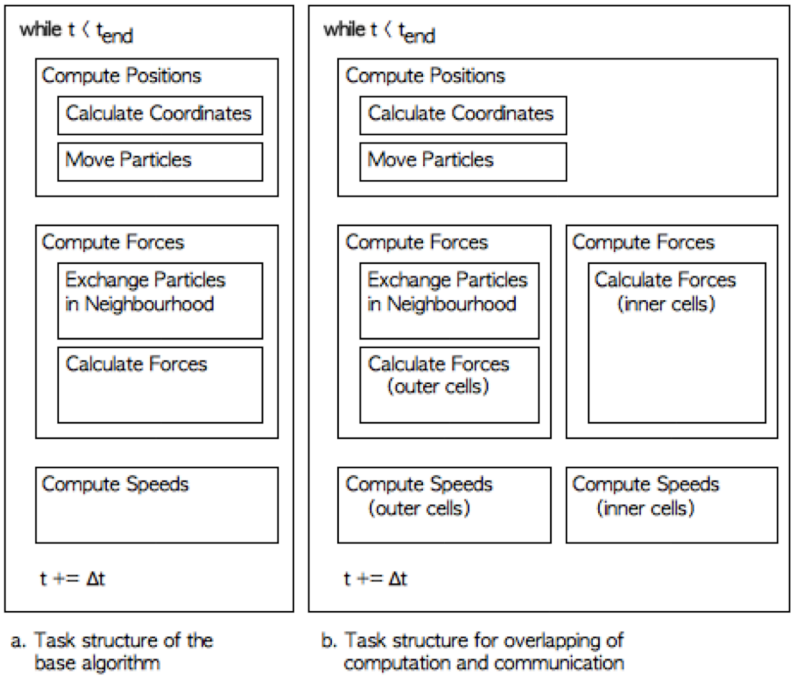

Figure 1: Examples for an application structure using a coarse-grained hierarchical task-decomposition. [2]

expressible in frequently used programming languages. This design information is therefore often lost during the implementation phase and has to be tediously recovered again. The use of hierarchical multiprocessor tasks makes the parallelism explicitly visible in the source code.

The runtime system uses a hardware performance model of the computer that has a structure reflecting the hierarchy from cores over nodes up to the complete system. The combined use of both models allows graph partitioning and mapping algorithms the selection of the most appropriate system part, i.e. a hardware model subtree, to run a certain part of the application, i.e. a task subtree (see figure 2).

The development of simulation applications often happens under conditions where it is not possible to specify the computational effort and other resource requirements completely and precisely. The algorithmic complexity of basic building blocks such as BLAS routines or other fundamental algorithms has been analysed very deeply and consequently highly optimized implementations exist on almost all platforms. But the theoretical analysis of more complex numerical algorithms is a very hard task beyond the possibilities of most application programmers who are experts in their science field and not in complexity theory. Furthermore, applications in production often use a significant superstructure on top of well-known basic libraries to guarantee the numerical stability of the algorithms for the whole range of input data.

The described environment often provides only vague information about performance and needs to be considered in the development of the runtime-system. The same as for the algorithmic side can be said about the resource provisioning of computer systems. The complex nature of the hardware as well as of the operating systems makes it very hard to develop complete and precise performance models. Scheduling algorithms of the runtime system have to be designed therefore in such a way that they are able to use incomplete and imprecise estimates.

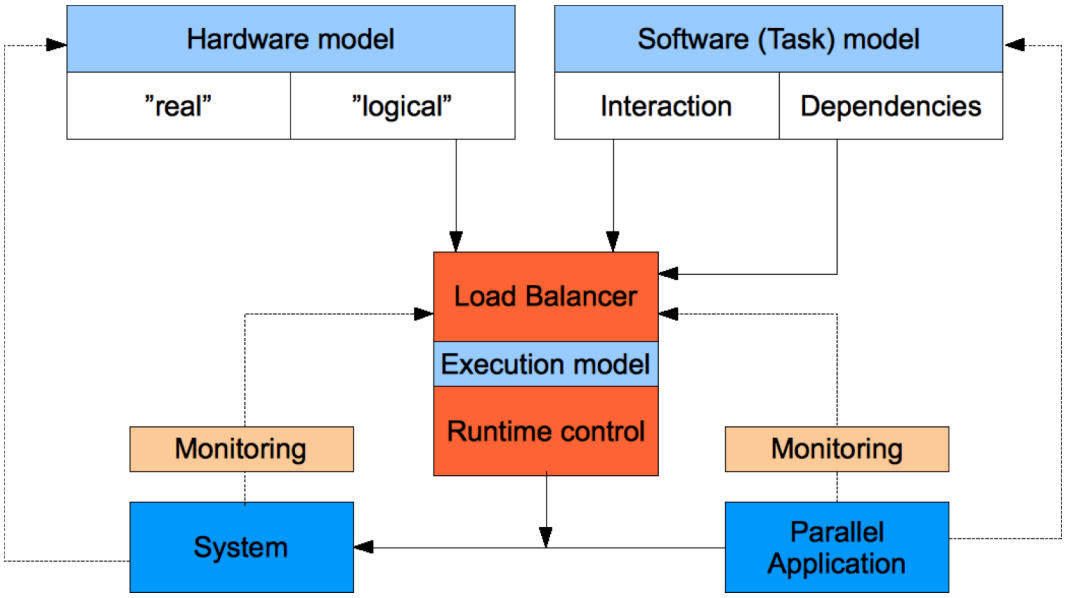

Figure 2: Usage of the software task model and the hardware model to optimize the program execution.

5 Software Architecture

The runtime system consists of three main components: a runtime administration component (Rta-C) schedules tasks and monitors their execution status; a monitoring component (Mon-C) provides information on the hardware utilisation, which is for scheduling decisions as well as to complement potentially incomplete or imprecise resource requirement specifications; and finally a performance analysis component (Pan-C) that analyses recorded monitoring data to provide more sophisticated hints for application control, beyond the capabilities of single run monitoring (see figure 3).

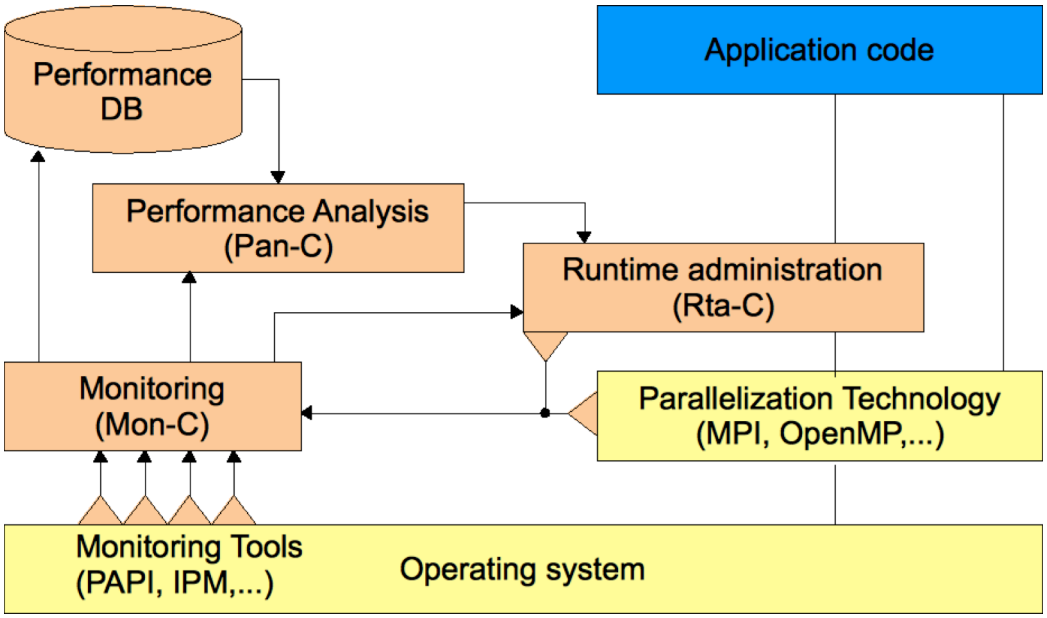

Figure 3: Components of the runtime-system

5.1 Runtime Administration Component (Rta-C)

Rta-C provides an API that can be used to define computational tasks as described in Section 4 and to control their execution. Furthermore, the component accesses a performance model of the computer system used for the execution of the application. The software model will be mapped onto the hardware performance model as well as possible. Scheduling algorithms will be used to calculate it as well as possible.

The definition of computational tasks will be done in the first step by means of an API provided from a library. Function calls mark, for example, the beginning and the end of such computational tasks. Developer-provided information can be given to the runtime system with arguments. Other functions will allow the transfer of the control flow between the application and the runtime system back

and forth. The integration with compilers or pre-processors in order to reduce the programming effort will be defined in a second step.

Rta-C moves the data of computational tasks to the processing elements by means of MPI functionality according to the execution plan provided by the schedule calculation.

Rta-C sends during the execution of computational tasks status information to Mon-C as well as receives information on the hardware utilisation during the program execution from it. The software and hardware models as well as the monitoring information allow it to recalculate the program schedule on the fly and to find available processing elements for the execution of subsequent tasks automatically. Furthermore, the application can also query Rta-C to get scheduling as well as monitoring information and influence the task execution.

The monitoring information will also be used as input to the scheduling algorithm for another purpose than to point to the deviations from the execution plan only. It complements incomplete task specifications and can be used, for example, to compute resource requirements more precisely using, for example, correlation analysis between input sizes and used resources in repetitive tasks.

An application kernel implementing a parallel molecular dynamics simulation for short-range potentials has been used in order to estimate the performance improvements that can be reached by load balancing and the allowed overhead for the additional program overhead induced by the runtime system. The results met expectations two-fold and provide guidance for the further development and practical application. The example application showed that noticeable performance improvements could already be achieved with load balancing based on a simple performance model. We used as metric of the computational work the number of particles per process and as metric of the communication the data amount of the MPI communication sent resp. received in point-to-point resp. collective operations. A process mapping based on these measurements reduced the execution time per time step by 15% without changes in the algorithm other than to introduce the marks for the computational tasks and the hooks to re-map them. Benchmarks of the communication operations in the application showed reduced costs up to 50% due to the automatic topology-optimised re-mapping of processes. On-going research on the load-balancing algorithm that includes also optimisations in the application algorithms itself reaches in the test-bed nowadays up to 30%.

Secondly the analysis of the application kernel underlines that a performance improvement depends on many complex interacting factors as known generally. System parameters, different technologies used for the parallelisation and specific use cases influence each other in such a way that it can happen that performance gains provided by a certain approach will be compensated by

another factor. We found for example that the same load balancing, which provides 15% performance gain using MPI_Recv and MPI_Send operations, allowed only a very small improvement in an implementation that uses overlapping of computation and communication.

The benchmarking results of the application kernel show that it will be possible to achieve noticeable accelerations of parallel algorithms compared to the expected introduction of overhead by the runtime system. The overhead introduced by the runtime system is expected to be at about 5% of the execution time as it could be measured during the benchmarks.

These observations guide the further practical implementation of the runtime system design. A reasonable assumption is that application developers and users will have implemented the best-known algorithm as well as that they will setup application runs in the best possible way for a computer system. Starting from that, the runtime system will observe the execution of the computational tasks in the program and apply its load-balancing algorithms in order to improve their performance. However, the runtime system must be deactivated automatically in the case that it should be impossible to achieve a performance improvement above a certain threshold in order to cover the computational costs of the load balancing algorithms.

5.2 Monitoring Component (Mon-C)

The Monitoring Component (Mon-C) collects performance metrics from the processes of a given application in a lightweight and scalable manner. It will be developed in a modular manner and able to use different information sources and profiling tools to extract the performance data. The Integrated Performance Monitoring tool IPM will be used in the current implementation.

The performance data is stored in a database for further analysis. The data will be used for investigations that provide hints to the programmer as well as to optimise performance in subsequent application runs. We are evaluating if an embedded database (Berkeley DB) can be used for this purpose. In that way, the performance data will be also packed within the application, freeing the user from the burdens of setting an external database. We are comparing the overhead generated by this embedded database in comparison to a typical relational database. We are also evaluating if our data can be adapted to fit in such a simple database.

The collected metrics are typical hardware counters collected through the PAPI interface such as instructions completed, floating-point operations or cache misses. MPI information is also gathered through the PMPI interface. The monitoring component also receives events about the start and termination of tasks from the Runtime Component. This high-level information about the

execution status of the application can be correlated with the profiling information and thus support the scheduling of tasks.

IPM is now fully working as a monitoring component (Mon-C) for the runtime system. We have also implemented an API on top of IPM that allows the runtime component (Rta-C) to communicate in real time with the Mon-C component. Thus, the Rta-C can ask about the performance of an application as it runs.

This new API offers two kinds of different data that can be accessed, regions and activities. Regions are measurement intervals defined within the application by the Rta-C. These intervals have associated different performance metrics such as time in MPI, time in computation, number of times executed or hardware counters selected by the Rta-C. On the other hand, activities are metrics associated to certain events such as MPI calls, POSIX calls or OpenMP regions among others. For instance, the Rta-C could access the activity for MPI_Recv obtaining how many times it has been called, total time inside the call, maximum and minimum time for that call and amount of bytes received.

The tests performed so far showed that the overhead introduced by the monitoring component and its API for real time monitoring never exceeded 1 % of the total application execution time. In the upcoming months we will perform further testing of this API and its scalability limits.

5.3 Performance Analysis Component (Pan-C)

Nowadays an overwhelming quantity of performance data can be collected and the handling and analysis of these huge amounts of data is a major challenge, in fact, it could even become impossible in the near future due to the increased resource and application sizes. We are therefore in high need of tools capable of analysing all of the generated performance data, reducing the quantity of performance information in a meaningful way, as well as behaving like a human expert who gives solutions to an unskilled user.

The Performance Analysis Component faces these challenges through several data mining and machine learning techniques. On one hand, it will reduce the amount of redundant information stored using techniques such as clustering or principal components analysis (PCA). On the other hand, it will build models from the data using techniques including Bayesian Networks to help the Runtime Component to make scheduling decisions. Further research is needed during the project to extend the early design stage of this component at the moment.

6 Test Plan

The Runtime System will be tested with several different scientific applications and kernels from different fields such as computational fluid dynamics, molecular dynamics, or weather prediction.

The starting point will be an application kernel for molecular dynamics simulations developed at KTH. This application implements basic numerical algorithms that are widely used in molecular dynamics and has been parallelized with MPI. Further tests will be performed with the CRESTA benchmark suite that is developed in WP2 as well as with the co-design applications of WP6. It is planned to use IFS and NEK5000.

The testing process is defined as follows: first, computational tasks are defined within the application; afterwards, the application is run in conjunction with the Runtime System; finally, runs with and without the Runtime System are compared. This process guarantees two main aspects. On one hand we check that the runtime does not change the application results, on the other hand, we have real measures on how much improvement can be gained with the runtime-system in the application.

This process will be repeated using several scientific kernels from different fields as mentioned earlier. In that way, we could spot real application needs that will be useful for further development of the Runtime System.

7 References

[1] State of the art and gap analysis - Development environment, Project CRESTA Deliverable 3.1 (2012).

[2] Michael Schliephake, Xavier Aguilar, Erwin Laure: Design and Implementation of a Runtime System for Parallel Numerical Simulations on Large-Scale Clusters. Procedia Computer Science, Volume 4, Proceedings of the International Conference on Computational Science, ICCS 2011, 2011, Pages 2105-2114.

[3] C. Augonnet, S. Thibaul, R. Namyst, “StarPU: a Runtime System for Scheduling Tasks over Accelerator-Based Multicore Machines”, (2010)

[4] StarSS homepage, http://www.bsc.es/computer-sciences/programming-models (accessed January 2012)

[5] F. Broquedis, N. Furmento B. Goglin, P.A. Wacrenier and Raymond Namys, “ForestGOMP: An Efficient OpenMP Environment for NUMA Architecture”, International Journal of Parallel Programming (2010)

[6] L.V. Kale, E. Bohm, C.L. Mendes, T. Wilmarth and G. Zheng, “Programming petascale applications with Charm++ and AMPI”, Petascale Computing: Algorithms and Applications (2008)

[7] A.Tabbal, M. Anderson, M. Brodowicz, H. Kaise and T. L. Sterling, “Preliminary design examination of the ParalleX system from a software and hardware perspective”, SIGMETRICS Performance Evaluation Review (2011)

[8] P. Mucci, D. Ahlin, J. Danielsson, P. Ekman and L. Malinowski, “PerfMiner: Cluster-Wide Collection, Storage and Presentation of Application Level Hardware Performance Data”, Euro-Par 2005 Parallel Processing (2005)

[9] K. Fuerlinger, N.J. Wright, D.Skinner, "Effective Performance Measurement at Petascale Using IPM", Parallel and Distributed Systems (ICPADS) (2010)

[10] J. Gonzalez, J. Gimenez and J. Labarta, "Automatic detection of parallel applications computation phases", Parallel & Distributed Processing 2009 IPDPS 2009 (2009)

[11] H. Servat, G. Llort, J. Giménez and J. Labarta, “Detailed Performance Analysis Using Coarse Grain Sampling”, EURO-PAR 2009 - PARALLEL PROCESSING WORKSHOPS (2010)

[12] P. C. Roth, D. C. Arnold, and B. P. Miller. “MRNet: A Software-Based Multicast/Reduction Network for Scalable Tools”, SC 2003, (2003)

[13] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, “Automatically characterizing large scale program behavior”, Proceedings of the 10th

international conference on Architectural support for programming languages and operating systems (ASPLOS-X) (2002)

[14] X. Liu, J. Zhan, K. Zhan, W. Shi, L. Yuan, D. Meng, L. Wang, “Automatic performance debugging of SPMD-style parallel programs”, Journal of Parallel and Distributed Computing (2011)