Design and Implementation of a Runtime System for Parallel Numerical Simulations on Large-Scale Clusters

Michael Schliephake, Xavier Aguilar, Erwin Laure
SeRC—Swedish e-Science Research Center and PDC, KTH Royal Institute of Technology, Teknikringen 14, SE-100 44 Stockholm, Sweden

Abstract

The execution of scientific codes will introduce a number of new challenges and intensify some old ones on new high-performance computing infrastructures. Petascale computers are large systems with complex designs using heterogeneous technologies that make the programming and porting of applications difficult, particularly if one wants to use the maximum peak performance of the system. In this paper we present the design and first prototype of a runtime system for parallel numerical simulations on large-scale systems. The proposed runtime system addresses the challenges of performance, scalability, and programmability of large-scale HPC systems. We also present initial results of our prototype implementation using a molecular dynamics application kernel.

Keywords:
hybrid computational methods, parallel computing, advanced computing architectures, runtime systems

1. Introduction

Massively parallel computing is a major driving force in computational science and scientific discovery. The systems are getting bigger and more complex day by day. Petascale computers are composed of hundreds of thousands of cores and have complex designs using heterogeneous technologies. It is thus a complicated task to achieve good application and system performance. In addition, the raising complexity of these machines increases the complexity of applications and operating systems, too. This new kind of heterogeneous systems poses new challenges in the development and porting of applications and requires significant effort to achieve as much as possible of the systems peak capability. Using human experts to optimize and port applications for these systems needs 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. 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.
In this paper we present the design of an adaptive runtime system addressing these challenges and results of a first prototypical implementation based on a molecular dynamics application. The runtime system consists of a resource manager, a library for runtime administration of parallel applications, and a performance monitoring and analysis tool. We base our design 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. For example, our system could detect a load imbalance in an application and try to correct it by assigning computational tasks to less utilized processing elements. It would also be possible to define the number of processing elements dedicated to a certain task dynamically. The runtime system could also redistribute processes between jobs based on its view of the complete cluster installation. This would allow to increase the system throughput and to ensure policies to control the energy consumption, the assignment of the best appropriate amount of resources for application speedup, etc.

In the remainder of this paper we describe the requirements and a general system design in Section 2, present a prototypical implementation and some initial experimental results in Section 3 and 4, and point out areas for future work in Section 5. After discussing related work in Section 6 we close the paper with some concluding remarks in Section 7.

2. Runtime System Design

Requirements

One of the hardest tasks 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, optimizations 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 too critical up to know but with the advent of more heterogeneous architectures this is increasingly becoming a problem. Moreover, different application classes require different optimization strategies making the development of generalized tools difficult. From a software engineering viewpoint a programmer prefers to have reusable software tools available that help to use the systems efficiently, decouple the supporting program parts from the numerical algorithm, and do not introduce too much overhead. It will be especially appreciated if software tools support an adaptive use of best practices, which otherwise would not be applied due to prohibitive implementation effort.

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 like 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 of the application programmers who are experts in their science field and not 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.

There are on-going efforts (e.g. [1]) to improve the situation by determining the complexity and the resulting computational effort in a more detailed way as well as to provide easier and more portable performance formulations. These efforts need to be complemented with tools that are able to use incomplete and imprecise estimates. 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.

An important requirement for any developments in this area is the reuse of existing application codes implemented often 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.

Based on recent hardware developments we can deduce the following requirements for numerical applications:

- Integration of data and task parallelism,
- Use of multi-level parallelism,
- 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,
- Optimized 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.

**System Design**

**General overview.** We have chosen a deductive method for the design and development of our runtime system to meet the requirements stated above. We defined a general concept that is described below and will evaluate it for different numerical simulation application classes. The results from these evaluations will be generalized as much as possible and used to iteratively improve the system design.

Tasks are constructs in such a way that they are typically used to express the parallelism during algorithm design and in program descriptions. Unfortunately this parallelism is not easily expressible in frequently used programming languages. This design information is therefore often lost during the implementation phase and has to be tediously regained again. To overcome this problem the design of our runtime system is based on a hierarchical task model. Tasks may be parallel in themselves and subdivided in an arbitrary depth. This view of a parallel program is commonly used in the decomposition of many problems and will be made explicitly visible in the source code through the use of the runtime system. Our hardware performance model of the computer has a corresponding structure reflecting the hierarchy from cores over nodes up to the complete system. The combined use of both models allows to select the best appropriate system part, i.e. a hardware model subtree, to run a certain part of the application, i.e. a task subtree (see figure 1a).

Our 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 utilization, 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 analyzes recorded monitoring data to provide more sophisticated hints for application control, beyond the capabilities of single run monitoring (see Figure 1b).

**Runtime administration component (Rta-C).** Rta-C provides an API that can be used to define computational tasks and to control their execution. This task model builds a software model of the application. It is matched by a performance model of the computer system used. Scheduling algorithms try to find the best possible mapping of tasks onto the processing elements of the target computer system.
Rta-C sends status information to Mon-C and receives information on the hardware utilization during the program execution from it. The software and hardware models as well as the monitoring information allow to keep the program schedule up to date and to find available processing elements for the execution of other tasks automatically. Furthermore, the application can also query this information and influence the task execution.

The monitoring information will also be used to improve incomplete task specifications. It can be used for instance to compute resource requirements more precisely using e.g. correlation analysis between input sizes and used resources in repetitive tasks.

**Monitoring component (Mon-C).** The monitoring component (Mon-C) is based on Perfminer\(^2\), a performance tool previously developed at PDC. Perfminer collects performance metrics from all the processes active in a given machine in a light-weight and scalable manner and stores this data in a database for further analysis. The data will be used for investigations that provide hints to the programmer as well as to subsequent application runs to optimize performance.

The collected metrics are typical hardware counters such as instructions completed, floating-point operations, cache misses, memory accesses as well as some timers such as amount of time in MPI operations. Furthermore, derived values like MFLOPS, IPC, etc. are also stored in the database. The monitoring component also receives events about the start and termination of tasks from Rta-C. 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.

**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 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.

Pan-C faces these challenges. Through several data mining and machine learning techniques it obtains knowledge from the performance data stored in the database in order to give feedback to Rta-C and help with the scheduling decisions. Pan-C is currently at an early design stage and requires further research and development. Different data mining and machine learning techniques are currently being explored. For example, it will use techniques such as clustering or principal component analysis (PCA) in order to reduce the amount of data needed to be analysed by reducing the information collected from thousands of threads to just a couple of representative ones. It will also utilize correlation techniques, decision-trees and inference engines as well as information about the hardware to predict application behaviour and make decisions.

### 3. Prototypical Implementation

In this section we describe the used task model, the monitoring of the application and the functionality of the runtime administration component to increase the parallel efficiency in a first use case.

**MD application kernel**

To verify the principal usability of our system design we at first a molecular dynamic application kernel which, follows the numerical methods described by Griebel et. al. [3] and allows to work in a well arranged field.

We selected the Strömer-Verlet method that is implemented most commonly for the integration of Newton’s equations of motion. The parallelization of the original code was done by the linked cell method and MPI. The kernel provides in this configuration a typical setting for a simulation with short-range potentials like the Lennard-Jones 6-12 potential.

The principal task structure is shown in figure 2a. In each time step one computes the resulting force that influences a particle. The computation is done for pairwise defined potentials in that way that the force between each particle pair will be summed as contribution to the resulting force. This force causes according to Newton’s law of motion a speed change of the particles. The new position of the particles that move during the time step with a certain speed will be computed after that.

The parallelization according to the linked cell method is done in the following way. In theory every particle influences every other one. But, potentials are decreasing quickly in many cases and have no noticeable influence beyond a certain distance from the originating particle, the so-called cut-off radius (see figure 2d). This helps reducing
the original computational complexity of $O(n^2)$ considerably and provides the basis for our parallelization approach. The simulation domain is divided geometrically in cells at first. The size of the cells may be chosen to be larger than the cut-off radius. Then one has to look only at the neighbouring cells in the computation of the resulting forces for the particles in a certain cell. Multiple cells are typically grouped into a subdomain that will be owned in a MPI program by one process. Most of the computational work will be done inside of such a domain to calculate the interactions between the particles. Particles situated near the border of the domain have not only interactions inside their domain but also receive influence from particles of adjacent subdomains. It happens also that particles leave their subdomain and enter another one due to their movements. The interactions as well as the moves over subdomain borders create the need for communication between the tasks owning the subdomains (see Figure 2c).

Usage of the runtime system

An obvious task model can be derived from the often used domain decomposition and uses subdomains as top-level tasks. A lot of load imbalances develop in the course of the simulation due to changing intensities of the particle interactions as well as varying particle densities in the subdomains.

Options to correct load imbalances are shifting of tasks to other nodes as well as to use parallel subtasks. We refined the original task structure further as shown in Figure 2b. This allows overlapping communication and computation during the calculation of the resulting forces and speeds. The computation for the particles in inner cells will now be done in parallel to the communication with neighbouring subdomains and the computation for the particles in the outer cells considering the influence from the neighbours. These parallel subtasks have been implemented with Pthreads. Furthermore, we have the possibility to parallelize loops iterating over the particles in the cells. This can be realized easily with Pthreads as well as with OpenMP. Conceptually, this parallelization corresponds to a further hierarchical task level. At this point should be remarked that the selection of the parallelization technique like MPI, OpenMP, or Pthreads will not be determined by the runtime system, but selected best by the programmer himself according to his combination of hardware, operating system, compilers, MPI implementations, and others.

Tasks have to be defined as functions of the respective programming language at present. The execution of tasks can be started either directly by the application what corresponds to a simple function call or it can be initiated by the runtime administration on the best possible processing element and time point. The scheduling is based on parameters
that have been specified upon the task creation. These parameters comprise the number and kind of processing elements that are required for the intra-task parallel implementation or subtasks, the required computational effort, the required amount of communication, and a callback function that provides functionality necessary to move the task (see figure 3). The latter will be called if the task has to be moved to another process.

The schedule is computed as good as possible with the information provided by the application. Metis [4] is used at present. It provides a multi-objective optimization for the partitioning of graphs. However, the scheduling module is designed as a plug-in to be able to use different tools in the future according to needs of different application classes.

Another way to keep the workload balanced is not to create or move computational tasks, but to assign a varying number of processing elements to it. The tasks can request a dynamic amount of processing elements in that way that a minimal or maximal number of usable processing elements is specified. The runtime administration evaluates all requests and dedicates a certain number of processing elements to the different competing tasks.

Both mechanisms together, the creation and shifting of tasks as well as the dynamic assignment of processing elements to tasks, allow to implement algorithms that can be adapted dynamically and vary flexibly to load imbalances. The top-level tasks that owns subdomains have been implemented as movable tasks while the inner tasks are able to use a dynamically assigned number of processing elements.

The number of particle interactions in a subdomain gives an estimate of the resource requirements. The number of particles near the subdomain border can be used to describe the communication effort. Measurements of the hardware utilization complement the particle numbers as estimates for the absolute resource usage. This works very reliable because the physical model defines a steady development of the particle system. The measured values are used in a multi-dimensional optimization to find the best possible task mapping onto processing elements under consideration of the task sizes memory and computational effort size -, volume of communication with a height weight to inter-node transfers -, and the costs of the communication to move tasks.

Such optimizations themselves become very quickly computational expensive. It is therefore possible to either define an upper limit for the allowed time to calculate the new mapping or to bundle a certain number of subdomains to workgroups that are balanced independently. In a next version it will be possible to form a hierarchical tree of workgroups and to balance the work between them. In that way is it possible to balance large-scale applications time-efficiently.

Mon-C traces the hardware utilization and correlates it with the active tasks. These parameters are at the moment the floating-point performance, the number of cache misses, the number of function calls to send or receive data as well as the transferred amount of data by these calls. The ratio between the times spent in user and system mode is also measured.

4. Experiments

Measurements. To assess the design and development approach and to identify potential obstacles we performed a number of experiments. These experiments were aimed to identify the potential impact of some of the mentioned approaches on the parallel efficiency.

Our example - the collision of two bodies - is more a qualitative than a realistic collision simulation, but it is well suited for our assessment purpose. To illustrate the setup and the evolution of the particle system we show a few snapshots from different simulation times in Figure 4. The upper small body moves downwards and collides with the resting large body. Bumps between the particles as a result of the movements of the small body plus a randomized small proper motion of all particles cause the destruction of the bodies and the mixture of the particles.

```c++
/**
 * @taskdef compF_inner
 * @inout grid @in sd
 * @pardegree 1..4
 * @size EstimatedSize
 * @migrationfunc shift_compF
 */
void compF_Inner(Cell grid, SubDomain sd) { /*...*/
  // Handover to RT in activating function
  //...
  double estimatedSize = calcSize();
  prts_exec_task(“myComp1”, compF_inner,
                 estimatedSize, myGrid, mySd);
  prts_exec_task(“myComp2”, compF_outer,
                 myGrid, mySd);
  /* Do maybe other things... */
  prts_join_tasks(“myComp1”, “myComp2”);
}
```

Figure 3: Code fragment with an example of a task definition and activation.
Our measurements presented in Table 1 show the influence of different parallelization strategies that are used by our first prototype. The values have been taken from the particle system at the simulation time point \( t = 7s \). The interactions are quite intense in this phase. The particles have been spread already in a larger area, many particles move between cells and subdomains, but there are also areas with a very low workload.

**Parameter space.** Our experiments are based on four implementations, one based purely on MPI, one with overlapping communication and computation tasks according to figure 2b, one where the computation of the resulting forces has been parallelized with OpenMP, and finally, an implementation combining overlapping tasks and the OpenMP parallelized force calculation. Another variation was made through the use of different cell sizes. The first cell size chosen was the cut-off radius that gives the minimal number of particles in a cell. The second cell size was the doubled cut-off radius. In that case a subdomain will consist of fewer but larger cells with more particles. The effect of the changed cell size in the application execution is an increase of the loop lengths during the iterations over the particles as well as that more particles have to be compared during the calculation of the resulting force. The next variation was to run different parallelized algorithms with a default mapping of the processes as the MPI library provides and alternatively with a process mapping that has been optimized for load balancing. Finally the runs were executed with different numbers of processing elements per top-level task (subdomain).

**Results.** The measurements have been done on a cluster with compute nodes consisting of 2 Quadcore AMD Opteron 2374HE (Barcelona) at 2.2 MHz clock frequency. The cache sizes L1/L2/L3 are 64 kB/512 kB/6 MB. Each node has 16 GB of RAM. The operating system is CentOS 5.5, and the MPI library MPICH 1.3.1 using 1 Gb Ethernet.

The load-balancing algorithm could be introduced with low overhead and reduce the execution time in general. However, in some configurations is it not possible to get the required optimal task mapping due to limitations in the prototypical implementation at the moment. The load balancing has been done with a multi-dimensional optimization to balance the computational workload per host as well as to map tasks to nodes in that way that the communication between tasks is maximized for tasks on the same node and minimized for the communication between nodes.

In all configurations the overlapping of communication and computations improves the execution time. However, the effects vary widely. The largest effect can be found in the runs with unbalanced task mappings compared to the balanced ones. The reason is that the communication volume between nodes is larger for unbalanced runs. This leads to a relaxation of the CPU and longer waiting times during the communication operations that can be used for computations. On the other hand, the CPU is much busier and communication operations are shorter in balanced runs with much more node-internal communication at higher bandwidths. It can happen that the overlapping produces an undesired increase of the runtime if this coincides with a configuration where many threads run on only a few cores. In our experiments we could find this situation when only 0.5 cores were provided per subdomain. Threads are not able to provide an increase of the concurrency, and the cache efficiency from the viewpoint of the computational tasks decreases due to the intensive memory usage in communication operations.

The task size in our example was not sufficiently large and did not allow the parallelization of OpenMP to compensate its own overhead. In contrast to that we could not find a significant influence by different loop lengths due to different cell sizes. Position checks before the time-consuming floating-point calculations help to avoid unnecessary calculations of interactions between particles.

Program runs with overlapping tasks and OpenMP active at the same time showed simply the addition of the execution time changes that could be separately measured before. This happens regardless the fact that the combined use of both parallelizations at the same time increases the number of active threads per core noticeably.

Another interesting observation is the result that the execution time became longer again if more than one core has
Table 1: Execution times (wall clock times) of the test application per time-step in milliseconds [ms]. Left values for cell side lengths $r_{cut}$, right values for cell side lengths $2r_{cut}$. Work is the average number of particle interactions per cluster node and $C_l/C_e$ is the ratio of the data amounts exchanged between processes on the same and on different cluster nodes.

<table>
<thead>
<tr>
<th>Cores per task</th>
<th>Class</th>
<th>Without LB</th>
<th>With LB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>Pure MPI</td>
<td>1.81</td>
<td>1.36</td>
</tr>
<tr>
<td></td>
<td>Overlapping</td>
<td>1.69</td>
<td>1.73</td>
</tr>
<tr>
<td></td>
<td>OpenMP</td>
<td>1.90</td>
<td>1.62</td>
</tr>
<tr>
<td></td>
<td>Overlapping + OpenMP</td>
<td>1.79</td>
<td>1.83</td>
</tr>
<tr>
<td></td>
<td>Work: 6.46E+04</td>
<td></td>
<td>Work: 6.68E+04</td>
</tr>
<tr>
<td></td>
<td>$C_l/C_e$: 2.74</td>
<td></td>
<td>$C_l/C_e$: 2.74</td>
</tr>
<tr>
<td>1</td>
<td>Pure MPI</td>
<td>1.40</td>
<td>1.28</td>
</tr>
<tr>
<td></td>
<td>Overlapping</td>
<td>1.19</td>
<td>1.14</td>
</tr>
<tr>
<td></td>
<td>OpenMP</td>
<td>1.46</td>
<td>1.33</td>
</tr>
<tr>
<td></td>
<td>Overlapping + OpenMP</td>
<td>1.26</td>
<td>1.19</td>
</tr>
<tr>
<td></td>
<td>Work: 2.48E+04</td>
<td></td>
<td>Work: 4.40E+04</td>
</tr>
<tr>
<td></td>
<td>$C_l/C_e$: 0.87</td>
<td></td>
<td>$C_l/C_e$: 1.21</td>
</tr>
<tr>
<td>2</td>
<td>Pure MPI</td>
<td>1.32</td>
<td>1.28</td>
</tr>
<tr>
<td></td>
<td>Overlapping</td>
<td>1.28</td>
<td>1.29</td>
</tr>
<tr>
<td></td>
<td>OpenMP</td>
<td>1.38</td>
<td>1.42</td>
</tr>
<tr>
<td></td>
<td>Overlapping + OpenMP</td>
<td>1.33</td>
<td>1.36</td>
</tr>
<tr>
<td></td>
<td>Work: 1.24E+04</td>
<td></td>
<td>Work: 6.06E+02</td>
</tr>
<tr>
<td></td>
<td>$C_l/C_e$: 0.57</td>
<td></td>
<td>$C_l/C_e$: 0.54</td>
</tr>
</tbody>
</table>

been made available per subdomain, i.e. per MPI process. The reason for this is that our prototype cannot pin thread groups optimized according to their memory access and with respect to the ccNUMA architecture of the processors at the moment. Expensive data transfers are the consequence if threads of one group are placed on different sockets.

5. Future Works

The current prototype will be developed to a broader applicable research tool in the next step. We will continue the analysis of molecular dynamic applications also for problems using long-range and many-body potentials as well as shift from application kernels to production codes. The result will be a runtime system with a more generalized API and improved load-balancing capabilities. We will also follow up with the concrete obstacles for high program performance observed in our experiments and complement the task model implemented so far with a hardware performance model as well as tools allowing a more rigorous control of the program execution.

The runtime system will be used in the development of applications from other areas too. This step-wise broadening of the application range will help to abstract the functionality of the runtime and ensures a steady verification of the development due to its application-centric organisation.

The performance analysis component (Pan-C) will be further developed in parallel. New data mining techniques will be implemented in the module in order to process the huge amount of information collected in real time. Among these techniques we will use clustering, classification, correlation, neuronal networks, decision trees or inference engines. The use of clustering, for example, will allow us to deal with executions of thousands of threads, grouping threads in clusters of similar behaviour and studying the most representative thread of each cluster. Correlation analysis as another example will be helpful determining relationships between collected metrics.

Furthermore, using artificial intelligence and machine learning techniques will provide expert analysis processing into the system. For example, using an inference engine with a knowledge base about different machines and hardware will allow the system to suggest the best solution for a problem in that concrete hardware. Moreover, with machine learning and classification methods, the performance analysis component could predict the behaviour of an application on a specific hardware.
6. Related Works

Hierarchical task models that use more than one processor for their processing have been investigated for a long time. Blazewicz [5] introduced them for the scheduling of tasks on a set of identical processors. Turek et al. [6] developed approximate algorithms for the scheduling of parallelizable tasks. Mounie et al. [7] introduced malleable tasks that can use arbitrary numbers of processors and show monotonic behaviour, i.e. that the execution time decreases with increase of the number of processors. Rauber and Rünger [8] use hierarchically structured tasks in their research on automated transformation systems as well as on the mapping and scheduling of tasks.

The work on load-balancing methods is one of the central themes in HPC due to its importance for the efficient use of the system resources. Two tools have to be mentioned especially for the partitioning of graphs. Metis and its parallel variant ParMetis [4, 9] as well as Scotch and its parallel variant PT-Scotch [10, 11] provide a manifold of algorithms. Both can deal with graph and mesh structures and can be used for the ordering of sparse matrices. Nodes as well edges can be weighted and allow multi-dimensional optimizations. Both tools are widely used by application programmers but are suited very well as modules in tool development projects.

Corbalán, J. et al. [12] propose to use OpenMP in order to balance irregular MPI applications on SMP nodes. They implement a system (resource manager and runtime libraries) doing this load balance automatically and dynamically. Yuan-Chieh Chow [13] analyses and compares different load balancing strategies in heterogeneous multiprocessor systems.

There are some performance tools that have started to use data mining techniques in order to help the user to analyse his applications. PerfExplorer [14] uses various data mining techniques such as clustering, linear regression and machine learning techniques such as inference engines in order to compare different executions and experiments as well as help the user with his performance analysis. SimPoint is a processor simulator using clustering to analyse execution phases in applications and reduce the execution time needed for a processor simulation. Gonzalez, J. [15] uses clustering techniques in order to detect parallel application structure. HPCToolkit [16] targets profile-based performance analysis, storing all the information on a database as well as allowing the user to merge and correlate data from multiple experiments.

Examples of comprehensive runtime systems are Charm++ and Zoltan. The parallel programming system Charm++ [17] has been implemented in C++. It provides high-level mechanisms for the implementation of parallel applications that map programmer-defined objects to appropriate processors. Zoltan [18, 19] focuses on services for the data management in parallel, adaptive and unstructured applications. It provides parallel partitioning algorithms, data migration and management tools, and communication routines. The implementation is heavily based callback functions. They realize a software architecture based on the principle known as “inversion of control” that is well-suited for the decoupling of general tools and concrete applications using them.

With our work we contribute to several aspects. Our tool aims at providing useful information to the programmer of numerical simulations as well as to the runtime system to increase the parallel efficiency in real-time. In order to achieve this it will use and combine profiling information from the hardware together with status information describing high-level programming abstractions as well as elicited knowledge from the records of previous executions. To reach the goal of being able to control applications on large-scale clusters we apply hierarchical software and performance models with an arbitrary model depth and will contribute with algorithms showing a high scalability. With respect to software engineering, we investigate solutions to allow a flexible combination of different parallelization technologies.

7. Conclusions

In this paper we presented the initial design and prototypical implementation of an adaptive runtime system for large scale HPC system. The numerical experiments have shown that it is possible to get substantial performance improvements through the introduction of a task model as well as the runtime control of the task execution combined with hardware monitoring in numerical simulations. The starting point of our considerations was that it is very hard to optimize the execution of parallel applications in advance due to the complex and unknown influences of the environment. This has been confirmed in the experiments. They provide evidence for the advantages of dynamic runtime support in simulation programs.
The importance of some of the issues we want to address with the runtime system is confirmed by these measurements. Other issues that have not been implemented yet in the current prototype proved their importance too. The control of thread pinning to cores under consideration of the memory hierarchy, for example, is a very important issue on our agenda. Due to the large effect on the execution times this feature has it is one of the main priorities of our future work.

Furthermore, our experiments have shown that apart from the given computer architecture also the underlying software stack has to be taken into account as it can have significant impact on performance. For instance, the concrete implementations of parallelization technologies like MPI or OpenMP have features that significantly affect performance and thus have to be considered in the optimization steps. One example is how the communication protocols are implemented in different MPI libraries and what their influence on different usage styles with respect to multithreading is. The runtime system has to handle them in such a way that it provides transparently empirical best-practice knowledge to the application.

References

[1] FP7 ICT EU project PEPPHER. URL: http://www.peppher.eu/


