Teaching High-Performance Computing on a High-Performance Cluster

Martin Bernreuther, Markus Brenk, Hans-Joachim Bungartz, Ralf-Peter Mundani, and Ioan Lucian Muntean

IPVS, Universität Stuttgart, Universitätsstraße 38, D-70569 Stuttgart, Germany
bungartz@ipvs.uni-stuttgart.de

Abstract. The university education in parallel and high-performance computing often suffers from a significant gap between the effects and potential performance taught in the lectures on the one hand and those practically experienced in exercises or lab courses on the other hand. With a small number of processors, the results obtained are often hardly convincing; however, supercomputers are rarely accessible to students doing their first steps in parallel programming. In this contribution, we present our experiences of how a state-of-the-art mid-size Linux cluster, bought and operated on a department level primarily for education and algorithm development purposes, can be used for teaching a large variety of HPC aspects. Special focus is put on the effects of such an approach on the intensity and sustainability of learning.

1 Introduction

The education in high-performance computing (HPC) at an academic level has to contend with several difficulties, the first being a disciplinary one. Within a math environment, the corresponding courses (if existing at all) are typically not that different from standard numerical courses; in computer science study programs, the focus is often restricted to the architecture of supercomputers and to their programming; from the point of view of a field of application, finally, HPC is frequently limited to speeding up some specific code as much as possible. Really interdisciplinary courses for mixed target groups are still rather rare. In addition to the standard fears of contact, curricular issues or different educational backgrounds may hinder the implementation of appropriate HPC courses.

A second and perhaps even more serious problem is the frequent lack of accessibility of suitable computers. Typical courses in parallel computing, e.g., focus on algorithmic and complexity issues, and they do this from a more or less theoretical point of view. If there is an accompanying practical (i.e. programming) part, students have to program small toy problems of toy size in MPI or OpenMP. One reason is that, in most cases, only a network of work stations just combined ad hoc via standard Ethernet connections, a department server (perhaps some smaller shared memory machine), or possibly a small experimental Linux cluster are accessible to students. The big machines in the computing

V.S. Sunderam et al. (Eds.): ICCS 2005, LNCS 3515, pp. 1–9, 2005.

centres, however, where tasks of a more realistic size could be tackled and where real and significant experiences could be gathered, are out of reach. They are reserved for research projects, not for teaching; they are needed for production runs and must not be wasted for the first trial and error steps of a class of twenty students running twenty bug-ridden programs for the same tasks.

Hence, university teachers often are confronted with an, at least at first glance, astonishingly low enthusiasm on the students' side – which actually is, of course, far from being astonishing: sorting ten thousand numbers in 0.5 seconds instead of 1.8 seconds can hardly be expected to be considered as a scientific breakthrough. And a fictitious upscaling of both problem sizes and peak performance may be a rational argument helpful in a lecture, but it won't be sufficient to infect even motivated students by the professor's enthusiasm.

What are the consequences? On the one hand, a parallel computer to be used in teaching should be sufficiently large to also allow for large-scale runs; the number of processors should be big enough to allow for reasonable studies on parallel performance, i.e. speed-up, parallel efficiency, scaled problem analysis, and so on. Briefly, all effects that can be observed on the really big machines should be visible here as well. On the other hand, of course, for being really accessible even to beginners, it must be installed and operated locally, i.e. in the respective department and not in some computing centre, where always service aspects are dominating. Consequently, we need an affordable solution which does not require a lot of staff for operation.

Against this background, two institutes of the Universität Stuttgart – the Institute of Applied Analysis and Numerical Simulation located in the math and physics faculty and the Institute of Parallel and Distributed Systems located in the computer science, electrical engineering, and information technology faculty – decided to invest in a mid-size cluster slightly below Top500 performance in the list of June 2004. The, probably, somewhat unusual aspect is that this cluster is primarily used for teaching and education, ranging from beginners' courses up to PhD studies. Since its installation in March 2004, our cluster *Mozart* [9] has been used in a large variety of courses and projects [8], many of which have been remodelled and tailored to *Mozart*. It is the intention of this contribution to discuss the possibilities and chances of such a kind of HPC education.

The remainder of this paper is organized as follows: In Sect. 2, we briefly present the cluster *Mozart* and its main features. Then, in Sects. 3 and 4, the types of courses involved and some representative exercises are described. In Sect. 5, we discuss the expected and the actually observed outcomes for our participating students. Finally, some concluding remarks close the discussion.

2 The Cluster Mozart

Mozart [9] is a standard mid-size homogeneous Linux cluster which was completely delivered and installed as a whole by the German company MEGWARE [2]. *Mozart* consists of 64 dual-Xeon computing nodes and of one additional

dual-Xeon node playing the part of the master and front-end node. In brief, the node configuration is as follows: a supermicro X5DPA-GG or X5DPL-8GM (master) ATX chipset; two Intel Xeon 3.06 GHz FSB 533 1 MB cache CPUs; 4 GB DDR-RAM; finally, an IBM 180GXP/60 GB or Ultrastar 146Z10/73 GB (master) hard disk drive. The cluster's theoretical peak performance resulting from this hardware configuration is close to 785 GFlops. As interconnect technology, *Mozart* has an InfiniBand 4x network (8 Gbit/s, 72-port Mellanox Gazelle Switch). Additionally, for administrative tasks, there is a Gigabit Ethernet interconnect (1 Gbit/s, three manageable HP ProCurve 4824 24-port switches and one HP ProCurve 2708 switch) in use. The system's overall cost was about 390.000 Euro.

Currently, *Mozart* is run with the Redhat 9.0 Linux operating system. For administration and monitoring purposes, Clustware v2.0.13 and a couple of other tools are used. For parallelisation, the MPI implementations MPICH-1.2.5.2 and MVAPICH-0.9.2 with InfiniBand support [10] are available. As compilers, there are the GNU gcc 3.x compiler and the current Intel compilers for C/C++ and FORTRAN with OpenMP support. To evaluate *Mozart's* potential, the Linpack benchmark runs were done in April 2004, showing a maximum sustained performance of about 597 GFlops or 76% of the theoretical peak performance. Figure 1 illustrates the Linpack performance for various numbers of processors.

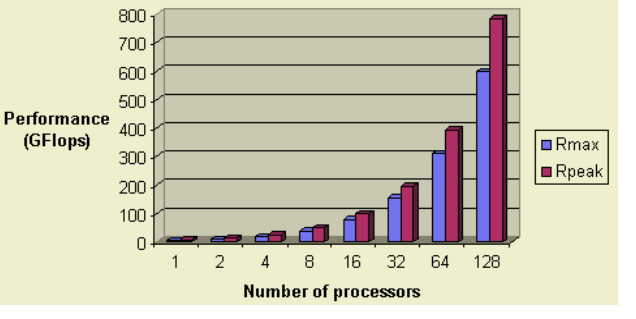

Fig. 1. Peak performance (R_{peak}) and maximum sustained performance ($R_{\max}$) of the HPL benchmark on *Mozart* for various numbers of processors

At the moment, besides studies of parallel computing itself and besides the teaching activities mentioned in the following sections, most of *Mozart's* remaining CPU time is used for finite element algorithm development and for simulation projects on flow simulations, fluid-structure interactions, molecular dynamics simulations, and traffic simulations.

3 Types of Courses Involved

At present, *Mozart* is used for a large variety of teaching activities and types of courses at graduate level, mainly within the diploma programs *Mathematics*,

Computer Science, and *Software Engineering*, and within the master program *Information Technology*.

First, there was a seminar *Cluster Computing* tailored to *Mozart*. Starting from a list of topics such as *Processor and Board Alternatives*, *Parallel I/O*, *State-of-the-art Network Technologies*, *MPI*, *Programming Models and Environments*, *Tuning Distributed Applications*, *Performance Evaluation and Benchmarking*, *Dynamic Load Balancing*, or *From Cluster to Grid Computing*, each student had to choose a topic and to prepare both a lecture and a paper on it. Wherever it was useful, students got accounts to work on *Mozart* and to use their results for their presentations. In the course evaluation, all participants showed enthusiastic about such an opportunity in a (typically more theoretically oriented) format such as a seminar, and quite a lot of them used their seminar participation as an entry point to some more practical project work on the cluster.

Second, we use *Mozart* as a platform for the practical exercises in the *Parallel Programming* course, where more or less standard problems for first parallel experiences are to be solved. Since programming plays only a minor part here, and since the programs to be written are, typically, rather short and simple, a four- or eight-node machine, e.g., would be sufficient, too. Nevertheless, the bigger cluster's function as a motivator and appetizer should not be neglected.

These first steps are then continued and extended in the lab course *Parallel and Distributed Programming*, where students shall get a deeper insight into MPI and OpenMP programming as two crucial fundamentals of HPC [1, 5, 6, 7]. We use OpenMP for both fine-grain parallelism on the two processors of each of *Mozart's* nodes and programming available shared-memory machines. During the first two thirds of the semester, students program parallel algorithms for classical problems from computer science such as sorting, coding and decoding, or solving linear systems of equations. In the last third, each group of 2-3 students selects one project which shall lead to an (at least to some extent) bigger piece of parallel software and which has some more realistic and, possibly, HPC background. We will discuss a few of these project tasks in the next section. It shall also be mentioned that we offer another lab course *Scientific Computing* that emphasizes the numerical aspects of HPC, where *Mozart* has also been integrated.

Furthermore, *Mozart* is a main pillar in our student project *Computational Steering*. Student projects are a specific format in the graduate part of the *Software Engineering* diploma program. Here, for one year, a group of 8-10 students work together on some larger software engineering project, under conditions which have been designed to be as realistic as possible. For example, one assistant plays the customer part, specifying the final product at the beginning and controlling progress throughout the year, but without being involved in the actual work; another assistant works as the adviser. Both project and configuration management have to be implemented, and students have to organize all aspects of teamwork and time management by themselves. In particular, the *Computational Steering* student project aims at getting a virtual wind tunnel software system, where numerical flow simulations and the visualization of the resulting flow fields are combined in an interactive VR environment in order to

control, foster, and accelerate the design process of a car, for example. For all numerical computations, *Mozart* is the target platform. This student project will also be discussed a bit more in the next section. Additionally, of course, a lot of individual projects such as diploma theses are running in the context of *Mozart*.

Finally, our group has several cooperations with Eastern European universities. For example, three students from the NTU Donezk, Ukraine, and fourteen students from different universities of the Balkan States have spent a 2-6-month internship at our department during the last three years. The main objective is to provide some insight into modern parallel computing and simulation scenarios. Recently, these internships have involved *Mozart*, too. For example, in the summer of 2004, one PhD student of computer science from Cluj-Napoca, Romania, studied aspects of Quality of Service over InfiniBand networks, and at the moment, one Ukrainian student is working on program development tools for *Mozart*. For both of them, this is the first contact with a parallel computer of reasonable size.

It is important to note that, formerly, we also integrated the use of parallel computers into our HPC-related teaching, of course. These were a small 8-processor cluster, our 8-processor (shared memory) department server, and various machines available at our university's computing centre (who also operates one of Germany's four federal supercomputing centres). However, the use and the outcomes were very much restricted: On our own machines, access was easy, but the learning goals were hard to achieve with eight processors; in the computing centre, the size was no problem, but the accessibility was limited (batch mode with long turnaround times, the need of writing proposals before getting accounts, or even no access for classes).

4 Typical Tasks and Exercises

In this section, we present some representative tasks from the lab course *Parallel and Distributed Programming* and the student project *Computational Steering*.

4.1 Projects in the Lab Course *Parallel Programming*

With their respective project, students finish the lab course *Parallel Programming*. All projects last for roughly six weeks, and they shall deal with some scalable parallel application, either programmed in OpenMP or MPI, where at least a few typical challenges have to be encountered.

One such project is to program a parallel simulator of the game "Scotland Yard", where a group of players tries to track down the mysterious Mr. X who moves around in London using buses, taxis, or the subway. Various algorithmic variants have to be implemented and to be analysed statistically in order to improve the chances of either Mr. X or of his pursuers. With sufficient computing resources, the resolution of the field can be increased, making the game more realistic and the job more complex from the computational point of view.

Another project, which is a bit closer to simulation and, hence, standard HPC applications, deals with the implementation of a parallel traffic simulator based on a cellular automaton algorithmic approach as suggested in [3]. To be precise, a parallel simulator for the simple scenario of car traffic on a one-lane highway has to be written. For parallelisation purposes, the highway is subdivided into an appropriate number of overlapping sections, each section corresponding to one parallel process. The basic underlying algorithm is rather simple. Each cell may contain at most one car at a time. The different states of a cell describe either the state of being empty or the speed of the respective car. This speed changes depending on the distance to the next car driving ahead, typically with some probabilistic component being included. There are quite a lot of interesting issues to be considered: How large should the regions of overlap be chosen? When is the optimum starting point of communication during a simulation step? How should an efficient data transfer be organized? How can a both simple and efficient dynamic load balancing be designed, which turns out to be necessary if the single nodes are not dedicated for the traffic simulation, but have to serve other parallel applications, too? For analysing communication, tools such as profilers are to be used. Finally, to prepare a further upscaling of the application (for the simulation of larger traffic networks, for example) and, hence, the extension to grid computing, the necessary changes in organizing the data transfer due to a now strongly varying communication bandwidth between the nodes have to be considered and implemented.

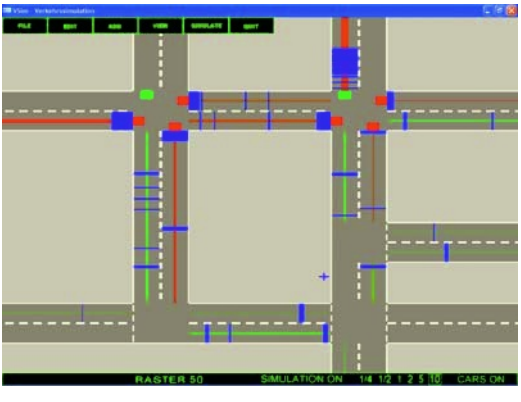

Fig. 2. Visualization of a basic traffic simulator

4.2 Tasks in the Student Project *Computational Steering*

The main idea behind the development of a virtual wind tunnel is to change the classical workflow of the design process. Preprocessing, simulation, and postprocessing phases merge to a simulation steering process illustrated in Fig. 3. There are several consequences, especially for the simulation part. We need a reactive software immediately handling user requests such as model changes or results. Since speed is a crucial prerequisite, efficient algorithms are necessary, as well

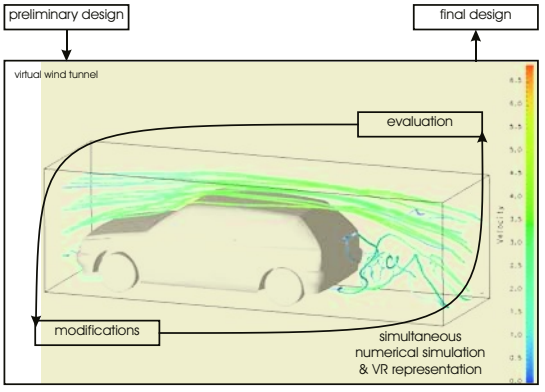

Fig. 3. Simulation steering workflow

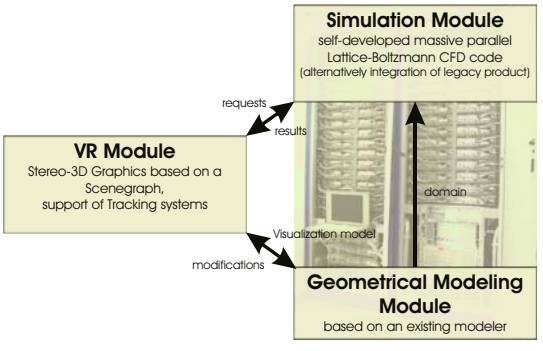

Fig. 4. Software modules

as a powerful hardware platform like *Mozart*. The machine is needed exclusively during the whole run and is busy doing online calculations.

From the technical point of view, the project consists of the three modules shown in Fig. 4, which form a distributed system: the *Geometric Modelling Module*, the *Simulation Module*, and the *Virtual Reality Module*. A common technique for representing solid objects is the boundary representation based on free form surfaces. In this project, we base the modelling on OpenCASCADE [4], a freely available open source development platform for 3D modelling applications. Surface meshing generates the visualization model, which is transfered, as well as the control points, to the VR module. The simulation module also derives its domain from this solid model. Next, as a CFD approach suitable for the steering context and for a massively parallel target system, the Lattice-Boltzmann method was chosen. A voxelization process generates the domain from the geometric model. There's also an interface to the commercial flow solver StarCD as an alternative. Finally, the user shall interact with the software system in a VR environment, in our case a power wall. Modifications are done with the help

of the control points through a flystick, and the resulting changes are submitted to the geometric modelling module. At the moment, the VR part runs on a SGI Onyx; in the future, this job shall be done by a cluster, too. The direct access to this hardware is an indispensable prerequisite for the realization of this project, which is still under development.

5 Expected and Experienced Outcomes

Among all positive effects that we expected and, actually, can observe, the following three are probably the most important. First, the direct integration of a moderate supercomputer into undergraduate and graduate teaching allows students to get a much more integral way of looking at scientific computing and HPC, letting them experience the effects of underlying models, numerical algorithms, and implementation issues on parallel performance in a very natural way. Second, several properties of parallel algorithms such as communication behaviour or scalability become obvious only with a larger number of processors. Thus, the possibility of running experiments with up to 128 processors improves students' practical experiences with parallel and supercomputer programming, as well as their intuitive ability of evaluating parallel algorithms. Finally, the variety of starting points for improving parallel performance such as algorithmic variants, compiler support, alternative parallelisation strategies, general tuning mechanisms, and so on can be explored in a by far more intense way.

6 Concluding Remarks

In this paper, we have discussed various ways how a modern mid-size Linux cluster can be used in university education in parallel computing or HPC. Although most of the exercises introduced could also be done on smaller parallel machines, on simple Ethernet-based networks of workstations, or on the supercomputer(s) available in the local computing centre, our experiences show that the educational outcomes and the learning effects are clearly improved. Furthermore, an increased motivation of students to further dive into HPC can be observed.

References

1. A. Grama et al. *Introduction to Parallel Computing.* Addison-Wesley, 2003.
2. MEGWARE Computers, Chemnitz, Germany. www.megware.com.
3. K. Nagel and M. Schreckenberg. *A cellular automaton model for freeway traffic.* J. Phys. I France **2** (1992), pp. 2221-2229.
4. *OpenCASCADE* www.opencascade.org/.
5. P. S. Pacheco. *Parallel Programming with MPI.* Morgan Kaufmann, 1997.
6. M. Quinn. *Parallel Programming in C with MPI and OpenMP.* Internat. ed., McGraw-Hill, New York, 2003.

7. M. Snir et al. *MPI: The Complete Reference (vol. 1 and 2)*. MIT Press, 1998.
8. Universität Stuttgart, IPVS. *Courses offered by the simulation department*. www.ipvs.uni-stuttgart.de/abteilungen/sgs/lehre/lehrveranstaltungen/start/en.
9. Universität Stuttgart, IPVS. *The Linux Cluster Mozart*. www.ipvs.uni-stuttgart.de/abteilungen/sgs/abteilung/ausstattung/mozart/.
10. *MPI over InfiniBand Project*. nowlab.cis.ohio-state.edu/projects/mpi-iba/.