From Mathematical Model to Parallel Execution to Performance Improvement: Introducing Students to a Workflow for Scientific Computing

Franziska Kasielke[1(✉)] and Ronny Tschüter[2]

[1] Faculty of Computer Science, Technische Universität Dresden, 01062 Dresden, Germany
{franziska.kasielke,ronny.tschueter}@tu-dresden.de
[2] Center for Information Services and High Performance Computing, Technische Universität Dresden, 01062 Dresden, Germany

Abstract. Current courses in parallel and distributed computing (PDC) often focus on programming models and techniques. However, PDC is embedded in a scientific workflow that incorporates more than programming skills. The workflow spans from mathematical modeling to programming, data interpretation, and performance analysis. Especially the last task is covered insufficiently in educational courses. Often scientists from different fields of knowledge, each with individual expertise, collaborate to perform these tasks. In this work, the general design and the implementation of an exercise within the course "Supercomputers and their programming" at Technische Universität Dresden, Faculty of Computer Science is presented. In the exercise, the students pass through a complete workflow for scientific computing. The students gain or improve their knowledge about: (i) mathematical modeling of systems, (ii) transferring the mathematical model to a (parallel) program, (iii) visualization and interpretation of the experiment results, and (iv) performance analysis and improvements. The exercise exactly aims at bridging the gap between the individual tasks of a scientific workflow and equip students with wide knowledge.

Keywords: Workflow for scientific computing · Teaching
Parallel programming · Performance analysis · Heat transfer

1 Introduction

Besides theory and experiment, simulation is the third pillar of science [8]. The increasing numerical complexity of simulation models results in a high computational effort. Furthermore, the memory demands of scientific simulations often exceed the amount of memory accessible by a single process. These factors render sequential execution infeasible. *Parallel and distributed computing* (PDC)

G. Mencagli et al. (Eds.): Euro-Par 2018 Workshops, LNCS 11339, pp. 211–221, 2019.
https://doi.org/10.1007/978-3-030-10549-5_17

enables fine-granular and large-scale simulations on highly parallel computer systems. As a consequence, PDC acts as a central service for computational science and has to be an integral part of educating future scientists. However, the task of training students in PDC is more than just teaching programming skills. Developers of parallel scientific applications need a profound knowledge about:

- Mathematical modeling to express the problem by, e.g., algebraic operators, differential operators, and/or functions,
- Computer science (CS) and computer engineering (CE) to transfer the mathematical model into statements of a programming language, and
- Visualization and interpretation of experiment results to gain knowledge out of raw data.

Additionally, performance analysis and improvements of scientific applications are important aspects. Despite these aspects being an integral part of the daily work of scientists, they are often missed in education. Scientific applications need to be tuned in order to leverage the full potential of computer systems, as well as to scale parallel applications to a larger amount of processes.

In summary, in order to successfully implement scientific applications a *workflow for scientific computing* has to cover all aspects: mathematical modeling, programming, working with HPC systems, data visualization and interpretation, as well as performance analysis and improvement [4].

In this work, a lecture and, especially, an associated exercise at Technische Universität Dresden is presented. Both lecture and exercise introduce students to the workflow for scientific computing. In addition, current feedback revealed that students experience a lack of practical programming exercises in their courses. Therefore, the exercise also aims to improve their programming skills. The specific contributions of this work comprise:

- The general design and goals of the course "Supercomputers and their programming" and, especially, one of the associated exercises at Technische Universität Dresden,
- The implementation of the exercise in order to address current limitations/drawbacks in the education of students,
- Bridging the gap between domain scientists and computer experts by
  - Introducing students to a complete workflow of implementing scientific applications,
  - Emphasize on both mathematical modeling as well as programming skills, and
- Reporting on experiences gained during the course and exercises.

The remainder of this work is organized as follows: In Sect. 2, design aspects of the course and one of the associated exercises are described in detail. The implementation of these aspects within the exercise is presented in Sect. 3. The feedback from current students and tutors is summarized in Sect. 4. Additionally, an outlook on future enhancements is given.

2 Design of the Exercise: Workflow for Scientific Computing

The course "Supercomputers and their programming" at Technische Universität Dresden, Faculty of Computer Science, is characterized by a heterogeneous audience. This course addresses undergraduate and graduate students of computer science, information systems engineering, mathematics, computational science and engineering, as well as natural and engineering sciences. The focus is on strategies and methods for parallel processing including common programming models, architecture and networking concepts, and required algorithmic components of parallel and distributed computing. Furthermore, the course is influenced by experiences of the interdisciplinary application area at the Center for Information Services and High Performance Computing (ZIH). At Technische Universität Dresden, the academic year consists of a summer and a winter semesters. Each semester includes a teaching period of 15 weeks. Both, the lecture and the associated exercise, take place once a week with a duration of 90 min each.

Since the students attending the course come from different fields of science, their existing knowledge varies widely. Either the students have comprehensive expertise in (parallel) programming and only basic to none expertise in numerical modeling of scientific applications, or vice versa. An exercise comprising two sessions was created to bridge this gap. The idea of this exercise is to convey expertise in both areas: numerical modeling of scientific applications as well as parallel programming including performance analysis and improvement. Based on the example of a heat transfer simulation, the students practically pass through a complete, albeit simplified, workflow for scientific computing. Considering the entire workflow for scientific computing represents the unique characteristic of this exercise.

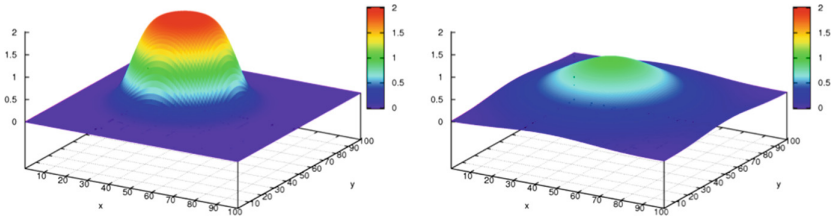
In the following subsections, the design of the exercise is described in more detail: the mathematical model, the parallel implementation and execution, as well as visualization and performance analysis aspects. The implementation of these aspects within the exercise is explained in Sect. 3.

2.1 Mathematical Model

For convenience and without loss of generality, in the exercise the heat transfer simulation in a two-dimensional space is considered. The propagation of thermal energy in a given two-dimensional space is described by the following parabolic partial differential equation:

$$\frac{\partial}{\partial t}u(x,y,t) = a \cdot \left(\frac{\partial^2}{\partial x^2}u(x,y,t) + \frac{\partial^2}{\partial y^2}u(x,y,t) \right), \tag{1}$$

where a denotes the thermal diffusivity. A visualization of the heat distribution in a two-dimensional space with a source of heat at the center of the region is shown in Fig. 1.



(a) Initial state of the simulation (b) Intermediate state of the simulation

Fig. 1. Heat distribution in a two-dimensional space, heat source at the center

The finite difference method is used to obtain the numerical solution of Eq. 1. The continuous partial differential equation is approximated with a discrete equation. The heat distribution u is determined on a grid $\Omega = \{(x_i, y_j, t_k)\}$, with $x_i := i \cdot \Delta x$ $(i = 1, \ldots, n_x)$, $y_j := j \cdot \Delta y$ $(j = 1, \ldots, n_y)$, and $t_k := k \cdot \Delta t$ $(k = 1, \ldots, n_t)$, where Δx, Δy, and Δt denote the increments in x-, y-, and t-direction. The heat distribution in a given cell (x_i, y_j) of the grid at a given time step t_k is denoted as $u(x_i, y_j, t_k) := u|^k_{i,j}$. Approximating the time derivative by the forward differencing scheme and the space derivatives by the 2nd order central differencing scheme yields:

$$\frac{u|^{t+1}_{i,j} - u|^t_{i,j}}{\Delta t} = a \cdot \left(\frac{u|^t_{i+1,j} - 2u|^t_{i,j} + u|^t_{i-1,j}}{\Delta x^2} + \frac{u|^t_{i,j+1} - 2u|^t_{i,j} + u|^t_{i,j-1}}{\Delta y^2} \right). \quad (2)$$

The solution of Eq. 2 requires the specification of boundary conditions. Well-known representatives are Dirichlet and Neumann boundary conditions. The Dirichlet boundary condition specifies the value by a function, whereas the Neumann boundary condition specifies the value by the normal derivative of the function. Periodic boundary conditions represent a special case. For the sake of simplicity, periodic boundary conditions are assumed, given by:

$$\begin{aligned} u|^t_{0,j} &= u|^t_{n_x,j}, & u|^t_{n_x+1,j} &= u|^t_{1,j} \quad (\forall j = 1, \ldots, n_y), \\ u|^t_{i,0} &= u|^t_{i,n_y}, & u|^t_{i,n_y+1} &= u|^t_{i,1} \quad (\forall i = 1, \ldots, n_x). \end{aligned} \quad (3)$$

2.2 Parallel Implementation and Execution on HPC Resources

The implementation of the heat distribution (Eq. 2) with periodic boundary conditions uses two two-dimensional grids of the size $n_x \times n_y$. One is the present grid, the other one is the temporary grid. For calculation of the heat distribution at the boundaries according to Eq. 3, these grids are expanded at the boundaries resulting in a grid size of $(n_x + 2) \times (n_y + 2)$.

The heat distribution is computed for all inner cells in the present grid for one time step, the results are saved in the temporary grid. After completing the calculations of one time step, the boundary cells are updated with the new values (according to Eq. 3). The present and the temporary grid are swapped

in order to prepare computations of the next time step. A visualization of this computing scheme is shown in Fig. 2.

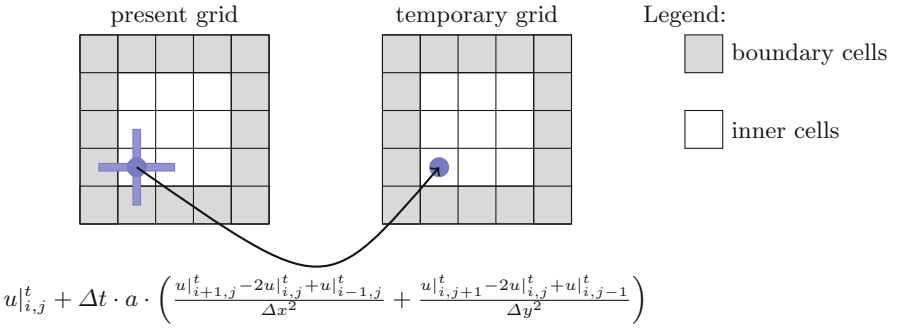


Fig. 2. Computing the heat distribution at one time step using the present (left-hand side) and the temporary (right-hand side) grid

The heat distribution is parallelized using the Message Passing Interface (MPI) [6]. MPI is widely used and proved its performance on a wide range of hardware platforms. It is assumed that the available processes $P_1, \ldots, P_k$ can be arranged in a two-dimensional cartesian grid. The computational grid is evenly partitioned over the process grid. The partitioning of the computational grid over four processes is shown in Fig. 3.

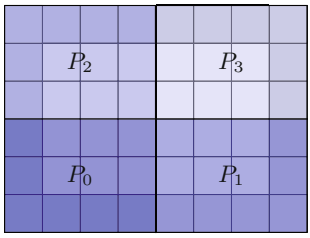
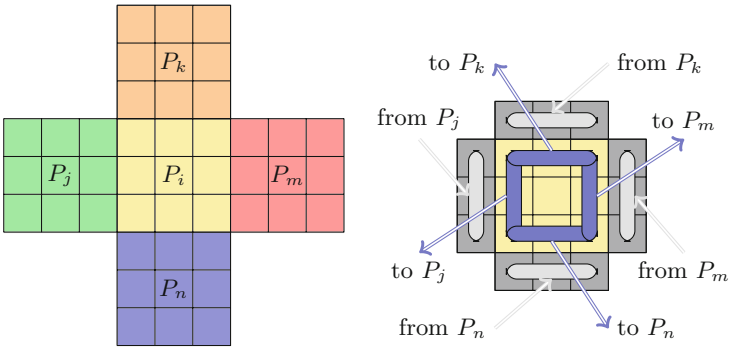


Fig. 3. Partitioning of the computational grid over four processes

Each process works on its own parts of the present and temporary grid. Communication is necessary for computing the heat distribution at the boundaries of the partial present grid. Therefore, the grids of the processes are extended by halo cells at the boundaries. The required data transfers of a process P_i ($i \in 1, \ldots, k$) including the neighborhood relations are shown in Fig. 4.

After finishing the simulation, the overall energy of the system is computed by gathering and adding the final values of all inner grid cells. A loss of energy



(a) Neighborhood relationships of process P_i

(b) Data transfers of process P_i

Fig. 4. Neighborhood relations and corresponding data transfers of process P_i

in the system between the start and the end of the simulation detects failures in the implementation.

The parallel implementation of the heat distribution is executed on *Taurus*. This Bull HPC system at Technische Universität Dresden, Germany, consists of 2,085 nodes with a total theoretical peak performance of 2,087 TFLOP/s.

2.3 Visualization and Interpretation of the Simulation Results

The numerical solution of the heat distribution is written to a file periodically. A simple visualization tool is offered in the exercise. The correct distribution of the heat energy in the computational domain over time can be determined intuitively. The tool is described in more detail in Subsect. 3.3.

2.4 Performance Analysis and Improvements

Performance analysis is an essential step in the workflow for scientific computing. Due to the increasing numerical complexity of the underlying simulation models scientific applications show a high demand on compute resources. Performance analysis and corresponding improvements of the applications can help to reduce execution times or increase applications' scalability. This enables time critical use case scenarios like weather forecasts. Additionally, if the simulation requires less time it often directly translates into reduced cost in terms of energy. In the exercise, students use established tools (e.g., Score-P [5], Cube [2], Vampir [3]) for the performance analysis of the parallel heat distribution application.

3 Implementation of the Exercise: Workflow for Scientific Computing

In this section, the implementation of an exercise within the course is highlighted. Within this exercise students learn the basic concepts of PDC, e.g.,

domain decomposition, communication, and synchronization between processing elements. Another important aspect of this exercise is teaching students to build upon existing libraries and tools instead of starting from scratch. Additionally, students are introduced to the concepts of working with HPC systems.

3.1 Mathematical Model

The exercise starts with a brief introduction to the numerical solution of the heat distribution. In a slide set, the mathematical model (as shown in Sect. 2.1) is presented to the students. Typically, students with a science background are more familiar with these aspects than CS/CE students.

3.2 Parallel Implementation and Execution on HPC Resources

In the exercise, the students implement a heat distribution simulation in the C programming language and use MPI in order to parallelize the application. Due to time constraints, the students would not be able to implement the application from scratch. Therefore, they receive a source code skeleton from the tutors. This skeleton already contains the basic program structure (see Listing 1.1). However, essential parts of the source code (e.g., domain decomposition, data transfer between processes) are left blank and marked to be implemented by the students. The programming exercises start with fairly simple tasks, such as, initializing the MPI environment (`MPI_Init`), determining the number of all MPI processes (`MPI_Comm_size`) or the global MPI rank (`MPI_Comm_rank`). More challenging tasks include parallel I/O (`MPI_File_open`, `MPI_File_set_view`, `MPI_File_read`, `MPI_File_write`, `MPI_File_close`) to read/write data from/to files. In order to distribute data over participating processes, the students create cartesian topologies and associated MPI communicators (`MPI_Dims_create`, `MPI_Cart_create`). The implementation of the halo update after each iteration requires the determination of the neighbor ranks in the cartesian communicator (`MPI_Cart_shift`) and subsequent data exchanges with the appropriate neighbor ranks (`MPI_Isend`, `MPI_Recv`, `MPI_Wait`). In addition, the update of the vertical halo cells makes use of derived datatypes (`MPI_Type_vector`, `MPI_Type_commit`, `MPI_Type_free`). Finally, a collective operation (`MPI_Reduce`) computes the overall energy of the system.

For some students, this exercise is the first opportunity to work on HPC systems. Using HPC systems differs fundamentally from the experience students gained by working on their local machines.

First, HPC systems typically provide a wide range of software components (e.g., libraries, compilers, tools). Often multiple versions of software components are available. Therefore, the students are introduced to the idea and usage of environment modules. In the exercise, students use the *LMOD* module system to select the compiler and MPI runtime.

Second, in contrast to a local system, multiple users share the compute resources of a HPC machine. Therefore, a job scheduling system allocates

Listing 1.1. Pseudo code illustrating the algorithm of the heat distribution simulation

```
/* initialize grid from file */
loadGridFromFile();

/* initialize temporary grid */
initializeTempGrid();

/* heat distribution calculation phase */
for ( count = 0; count < max_steps; count++ ) {
    /* save intermediate result to file */
    if ( count % 20 ) {
        saveGridToFile();
    }
    heatCalculation();
}

/* save result to file */
saveGridToFile();
```

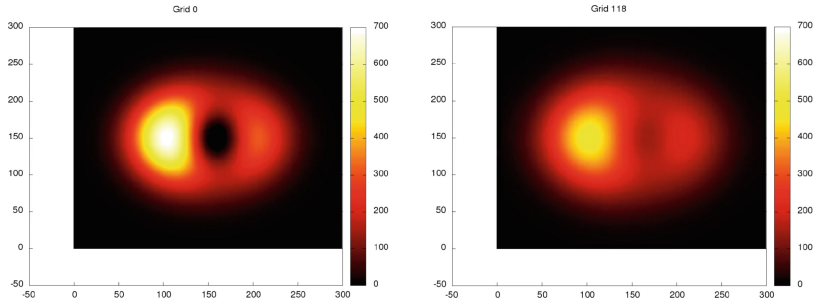
resources for each application run (job) initiated by the user. Taurus is operated with the *Slurm* job scheduling system. The students write their own job script to request appropriate compute resources (e.g., select compute nodes from partitions equipped with Haswell CPUs). Afterwards, the students learn how to submit, cancel, and monitor their jobs.

3.3 Visualization and Interpretation of the Simulation Results

As shown in Listing 1.1, every 20 iterations the heat simulation writes its intermediate results to a file. At the end of the application run, also the final heat distribution is written to this file. Consequently, the result file contains a series of snapshots. Each snapshot represents an individual state of the heat distribution at a specific simulation time step. In this exercise, the students use a prepared bash script to generate a movie showing the heat flow over time. The bash script opens the result file. Within the main loop of the bash script an individual snapshot is read and converted to a PNG image using *Gnuplot*. Two of these PNG images are illustrated in Fig. 5. Finally, the bash script calls *ffmpeg* to create an MP4 video based on the series of PNG images.

3.4 Performance Analysis and Improvements

Tasks with respect to performance analysis and improvements complement the overview of the workflow for scientific computing. The goal is to monitor the application and observe its runtime behavior. Therefore, the students recompile the application with the Score-P [5] measurement infrastructure. For the presented example, Score-P automatically enables compiler instrumentation for user



(a) Initial state (b) State after 118 iterations

Fig. 5. Visualization of the result data generated by the heat distribution simulation code

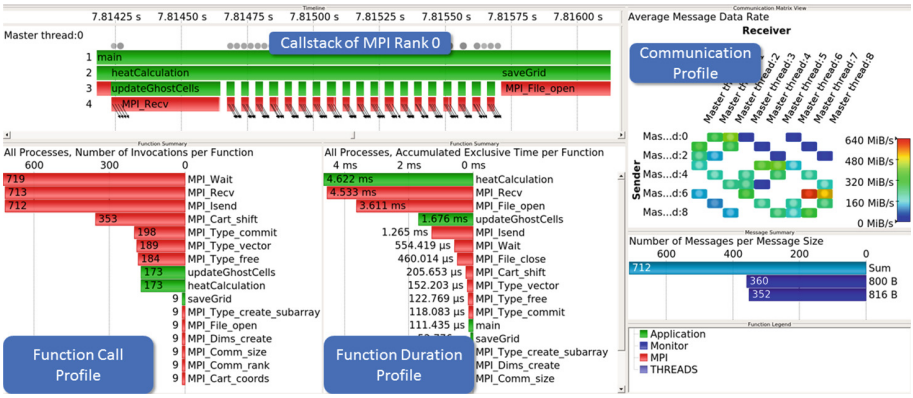


Fig. 6. Trace visualization of the time interval of 20 iterations in the heat simulation, at the end of this interval the application starts writing intermediate results to a file

functions within the source code and intercepts calls to the MPI library. As a consequence, calls to MPI and user functions trigger the measurement system at application runtime. Whenever triggered, the measurement system collects performance data (e.g., timestamp, function name, hardware performance counter) and stores the information as a *profile* (aggregated data) or *trace* (log of individual events). Guided by the tutors, the students use established tools, e.g., Cube [2] and Vampir [3], to visualize and analyze this performance data. The visualization of a trace in the Vampir analysis tool is shown in Fig. 6. The students learn how to interpret profiles and traces, correlate performance patterns with source code, and gain knowledge about the application behavior. Based on this knowledge, students and tutors discuss about ideas to improve the performance of the application. For example, the performance analysis of the initial application reveals that most of the runtime is spent in MPI. On the one hand, this performance issue stems from inefficient usage of MPI routines (e.g., the

result file is opened and closed for each snapshot). On the other hand, the ratio between communication and computation can be improved. After the discussion, the students can modify the source code in order to resolve performance problems or investigate the effects of different file systems on the I/O performance of the application. Comparing the performance data collected during the run of the initial and modified application directly reveals the effectiveness of the changes.

4 Conclusion and Future Work

In this work, the idea behind an exercise in the course "Supercomputers and their programming" is presented. In this exercise, the students from computer science, information systems engineering, mathematics, computational science and engineering, as well as natural and engineering sciences are practically introduced to a complete workflow for scientific computing. While working on a simplified example, the students familiarize with the general idea of the workflow for scientific computing. This workflow can be applied to other problems of computational science as well. The students learn to work with mathematical models, transfer these models into statements of programming languages, use HPC systems, visualize and interpret result data generated by simulation runs, as well as analyze and improve the performance of scientific applications.

In the course "Supercomputers and their programming" the students are introduced to theoretical background in the field of parallel and distributed computing. The exercise described in this work supplements the gained knowledge from the course with practical experiences. With completion of the exercise the students are well prepared for their future scientific work. The exercise presents a holistic training for students and is often the first practical experience with a complete workflow for scientific computing. This workflow represents a general methodology and can be applied to other scientific problems as well. The feedback from the students is very positive. In contrast to common curriculums, the course covers not only theoretical knowledge. Moreover, in this exercise the students have to implement, execute, and analyze a parallel program on a HPC system. The students highly appreciate the chance to gain skills or improve their expertise in parallel programming. Furthermore, the tutors noticed that the exercise encourages the cooperation between students from different courses of study. The students benefit from each others expertise and complement their knowledge. Furthermore, the tutors also benefit from the close cooperation with the students. First contacts are established for acquiring student workers or thesis topics.

While the feedback of the exercise is very positive, there are options to enhance the practical session. For example, the parallel implementation can be extended to shared-memory parallelism by an hybrid MPI+OpenMP version. Multi-core architectures with shared memory are omnipresent. Although, OpenMP [7] is theoretically introduced in the lecture, it is currently not part of the exercises. A practical implementation of the shared memory paradigm would complement the course. The visualization approach presented in the exercise can

be improved as well. Students would benefit from replacing the self-implemented visualization approach by established frameworks (e.g., VisIt [1]) and their file formats.

References

1. Childs, H., et al.: VisIt: An end-user tool for visualizing and analyzing very large data. In: High Performance Visualization-Enabling Extreme-Scale Scientific Insight, pp. 357–372 (2012)
2. Geimer, M., Saviankou, P., Strube, A., Szebenyi, Z., Wolf, F., Wylie, B.J.N.: Further improving the scalability of the scalasca toolset. In: Jónasson, K. (ed.) PARA 2010. LNCS, vol. 7134, pp. 463–473. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28145-7_45
3. Knüpfer, A., et al.: The Vampir performance analysis tool-set. In: Resch, M., Keller, R., Himmler, V., Krammer, B., Schulz, A. (eds.) Tools for High Performance Computing, pp. 139–155. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68564-7_9
4. Koumoutsakos, P., Chatzi, E., Krzhizhanovskaya, V.V., Lees, M., Dongarra, J., Sloot, P.M.A.: The art of computational science, bridging gaps - forming alloys. Preface for ICCS 2017. Procedia Comput. Sci. **108**, 1–6 (2017)
5. Mey, D., et al.: Score-P: a unified performance measurement system for petascale applications. In: Bischof, C., Hegering, H.G., Nagel, W.E., Wittum, G. (eds.) Competence in High Performance Computing. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-24025-6_8
6. MPI Forum: Message Passing Interface (MPI), May 2018. http://mpi-forum.org/
7. OpenMP: The OpenMP API specification for parallel programming, May 2018. http://openmp.org/
8. Riedel, M., Streit, A., Wolf, F., Lippert, T., Kranzlmüller, D.: Classification of different approaches for e-science applications in next generation computing infrastructures. In: 2008 IEEE Fourth International Conference on eScience, pp. 198–205 (2008). https://doi.org/10.1109/eScience.2008.56