SAUCE: A Web-Based Automated Assessment Tool for Teaching Parallel Programming

Moritz Schlarb(✉), Christian Hundt, and Bertil Schmidt

Institute of Computer Science, Johannes Gutenberg University,
D-55128 Mainz, Germany
{schlarbm,hundt,bertil.schmidt}@uni-mainz.de

Abstract. Many curricula for undergraduate studies in computer science provide a lecture on the fundamentals of parallel programming like multi-threaded computation on shared memory architectures using POSIX threads or OpenMP. The complex structure of parallel programs can be challenging, especially for inexperienced students. Thus, there is a latent need for software supporting the learning process. Subsequent lectures may cover more advanced parallelization techniques such as the Message Passing Interface (MPI) and the Compute Unified Device Architecture (CUDA) languages. Unfortunately, the majority of students cannot easily access MPI clusters or modern hardware accelerators in order to effectively develop parallel programming skills. To overcome this, we present an interactive tool to aid both educators and students in the learning process. This paper describes the "System for AUtomated Code Evaluation" (SAUCE), a web-based open source (available under the AGPL-3.0 license at https://github.com/moschlar/SAUCE) application for programming assignment evaluation and elaborates on its features specifically designed for the teaching of parallel programming. This tool enables educators to provide the required programming environments with a low barrier to entry since it is usable with just a web browser. SAUCE allows for immediate feedback and thus can be used interactively in class room settings.

1 Introduction

The teaching of parallel programming techniques has increasingly gained importance during the last decade due to the ubiquity of multi-core architectures both on portable devices and workstations. Moreover, it is a well-known fact that despite the exponential growth of modern CPUs' compute capabilities their single-threaded performance has barely increased during the recent past. The development of parallel algorithms can be exceedingly difficult for inexperienced students since scaling up the number of cores involves complex restructuring of the program's control flow. As a result, an extensive education of parallelization techniques is becoming increasingly important for every student in computer science.

Besides the theoretical education of parallel algorithms, their practical implementation can be challenging for the students. Race conditions and erroneous

S. Hunold et al. (Eds.): Euro-Par 2015 Workshops, LNCS 9523, pp. 54–65, 2015.
DOI: 10.1007/978-3-319-27308-2_5

synchronization may lead to incorrect results, implicit serialization of concurrent tasks and deadlocks may degrade performance or render the program defective. Hence, practical programming exercises are indispensable to develop the relevant domain knowledge and skills. This process is ideally supervised, such that the student receives immediate feedback after writing the source code. However, due to limited human resources a supervising assistant can often not be provided. A common workflow consists of the following steps:

1. Provide remote logins for a compute cluster or workstation.
2. Submit the student's program to a queuing system and wait for execution.
3. Manually evaluate the program's functionality by verifying its output.

As a result, small programming exercises embedded in lectures are often difficult to realize due to the lack of time. In this paper, we present a unified framework for the automated assessment and evaluation of source code in the field of parallel programming which can be used from any device with just a web browser. The presented "System for AUtomated Code Evaluation" (SAUCE) is free software (AGPL-3.0) and can be downloaded at [12]. A demo instance of SAUCE including the discussed examples using OpenMP, MPI and CUDA can be accessed at our website [13].

The rest of this paper is organized as follows: Sect. 2 discusses related work and compares the presented software solutions regarding the use in teaching environments. Technical aspects of SAUCE including extensibility of the software, teaching-related features and security matters are discussed in Sect. 3. The use case for the computation of a Poisson problem using Jacobi iteration on an MPI cluster is presented in Sect. 4. Further examples include multi-threaded programming with OpenMP and massively parallel programming using CUDA. Section 5 concludes the paper.

2 Related Work

For an educated view on previous work, the functionality offered by SAUCE must be split between offering a modern web-based interface for writing, compiling and running software, which could be considered an Integrated Development Environment (IDE), and the automated assessment of a written piece of software, which is more alike to the principles of Continuous Integration (CI) extended for educational purposes.

While there are some recent projects which provide an IDE in a web browser, like compilr [6] (closed source, paid access only), ideone [9] (closed source, usable anonymously) or Cloud9 [5] (open source), there are far less systems that can be used for programming assessment and to enhance the classical educational feedback loop of practical exercises. There are older approaches to this task for academic environments, like CourseMarker [8] or PC^2 [2], which are both used through Java-based GUI client programs that need to be installed on the student's computer. Current projects that combines a web-based editing interface and an automated assessment of the written sequential programs

embedded within a classical university course structure include WebLab (TU Delft) [14], Jack (University of Duisburg-Essen) [7] (both closed source) and Praktomat (KIT) [4] (open source). However, none of the above are explicitly targeted at parallel programming.

3 Technical Aspects

3.1 Python

SAUCE is written in the Python programming language. Python's syntactical and semantical features, along with the vast amount of third-party packages for all purposes, make it a good choice for the development of this state-of-the-art web application. Its widespread use and popularity increase the chance of the project to be further developed and extended in the future.

Python has a clear and readable syntax, which makes the source code comfortably readable, even for people not familiar with the language or with programming at all. A major strength of Python is its extensibility. Moreover, modules can be written in C or C++ with the use of the Python language bindings. This allows developers to combine the power of the interpreted Python language with more efficient and hardware-oriented languages like C or C++ and CUDA.

3.2 SAUCE Web Application

SAUCE is a web application written using the TurboGears 2 rapid web development framework [10], which follows the Model-View-Controller (MVC) pattern (see Fig. 1), which is a common design principle for web applications to achieve separation of concerns. TurboGears 2 provides basic building blocks for controllers and facilitates the coordination between the various components.

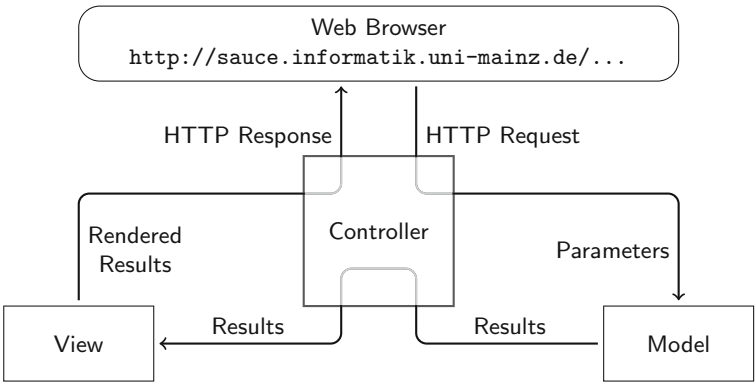

Fig. 1. The control flow within the Model-View-Controller pattern for one request.

Model. The model component defines the structure of the data for the application and the so-called business logic. The model part of SAUCE is realized with the SQLAlchemy object relational mapper (ORM) [3], which allows to use a relational database management system (e.g., SQLite, MySQL, PostgreSQL) like an object database. Hence, the ORM is an abstraction layer between the application and the database that facilitates the usage of object oriented programming paradigms and eliminates many problems regarding security and complexity of classical relational databases.

Controller. The controller is the central part of the application, which is responsible for user interaction. A user might also be another program that accesses the MVC application as a web service. The user requests an action by accessing a specific URL, which triggers a method that handles the input data (i.e., form field values or URL parameters), performs operations on the model data and returns some data to a specific view.

View. Following the paradigm of separation of concerns, the view is only responsible for the presentation of data retrieved from the controller and must not perform any modification on the data. It can also return information in a machine-readable format like XML or JSON. All the templates for SAUCE make use of the Twitter Bootstrap CSS framework, which is a state-of-the-art frontend framework containing styles for basic and advanced HTML5 elements. Bootstrap has been chosen because it features a clean and technical layout with focus on informational and form elements, which both suit the use case of this application.

3.3 Learning Tools Interoperability

SAUCE allows for the usage of its testing functionality from within other teaching platforms like Moodle or Coursera through the Learning Tools Interoperability (LTI) specification [1]. Using this interface makes it possible to provide a seamless experience, since students do not need to log in separately or join a course manually — they simply use the already existing authentication on the calling teaching platform. The testing results will be submitted back to the calling platform for central grading and feedback. As a result, SAUCE can be used as a service in a "headless" mode.

3.4 Security Considerations

Apart from classical security implications of web applications, additional aspects have to be considered since an application like this essentially allows a potential attacker to submit arbitrary code that will be executed within the server operating system.

There are two dangerous classes of users to be considered: Inexperienced programmers that submit faulty programs without intent and programmers who try to intentionally provoke and stress the system or to find ways of gaining unauthorized access to the system. The most common types of attacks in the given context are:

- Intentional and unintentional denial of service attacks like filling the memory or the hard disk with data or endless calculation loops, rendering the system unusable for other users or causing the application to subsequently fail.
- Information leakage by getting credentials to access the system or the database or opening network connections to transfer data in or out of the system.

Explicit security requirements regarding the execution of submitted programs can be summarized as follows:

- Deny read or write access to arbitrary files and directories (white-listing only a temporary directory used for the test run).
- Deny access to arbitrary system resources (e.g., hardware devices).
- Restrict CPU and memory usage and the process runtime.
- Prohibit network access (or at least access to the outside network).

SAUCE uses different techniques to address the aforementioned security requirements.

Traditional Unix Permissions that are based on users, groups and file or directory permissions, are enforced by the operating system with regards to the traditional Unix paradigm that "everything is a file" and thus provide a basic amount of access control on a file system level. Given that the SAUCE application is running under an unprivileged user account, its access to important or sensitive parts of the file system is prohibited.

Sandboxing, which means to separate possibly endangered parts of the application into a container, where their impact is minimized and malicious operations are revertible. It is advisable to use virtualization techniques for this kind of separation, like chroot, LXC or dedicated virtual machines. This also implies using an HTTP reverse proxy which forwards HTTP requests and their responses to and from arbitrary back end "worker" machines where the application itself runs. The distributed execution architecture as outlined in Sect. 3.5 provides implicit sandboxing, regardless of whether individual parts run on physical or virtual machines.

Resource Limits for process groups can be defined on various physical and logical resources and are enforced directly by the operating system kernel. Restrictions can be placed on memory usage, CPU execution time, open file handles, amongst others. By limiting the number of processes, it is possible to prevent a "fork bomb", where a process tries to infinitely spawn new child processes.

Firewall solutions are mandatory for any kind of web application or virtual machine setup. For example, with the Linux firewall tool "iptables", rules can be based on network packet attributes (like source or destination addresses, ports, etc.), but it is also possible to filter based on the Unix user of the application that created the packet. This can be used to allow normal operation of the web server and the application but restrict all network access for the user account that is used for executing the submitted programs.

3.5 Distributed Execution

Especially for programming parallel architectures, the possibility to dispatch steps that require specialized hardware and software, like the compilation and execution of submitted programs to worker nodes depending on their configuration, is important. A dedicated web server, where the SAUCE web application is running will most likely not have direct access to an accelerator card or an MPI cluster. Moreover, it would be inconvenient to set up an instance of the web application on several machines that feature a required piece of hardware. Therefore, we develop a lightweight queuing system for running submission tests on worker nodes instead of the host system (see Fig. 2). A test job is a contiguous unit of work consisting of compilation and repeated execution, once for each defined test case.

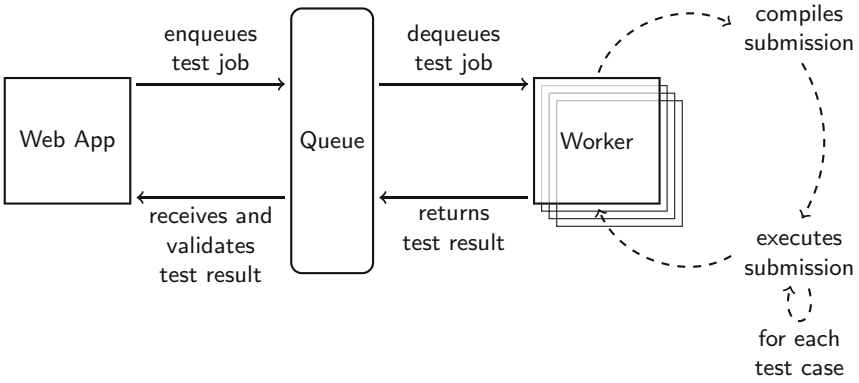

Fig. 2. Schematic overview of the distributed architecture that uses a task queue to dispatch compilation and execution of test jobs to one of many workers.

4 Use Cases

We provide a demo instance[1] of SAUCE featuring three interactive programming exercises for the parallelization techniques MPI, OpenMP and CUDA[2]. The submitted code is compiled and executed on the following platform:

CPU: Intel Xeon X5650 Hex-Core @ 2.67GHz with 96 GB attached RAM
GPU: Nvidia Tesla K40c with 12 GB attached video RAM
Software: GCC 4.8.2, OpenMPI 1.6.5, OpenMP 4.0, NVCC 6.5.12

Note, this instance runs as a freely accessible cloud service and thus the provided resources are limited to one CPU and one GPU. Nevertheless, SAUCE can handle programs on multiple nodes especially during the execution of MPI programs.

[1] Visit [13] and log in with username `teacher1` and password `teachpass`.
[2] These assignment examples are also available in the SAUCE repository at [12].

4.1 Solving the Poisson Equation Using MPI

A popular example for the teaching of communication primitives on distributed memory architectures is the iterative computation of the steady-state solution of the Poisson equation $\Delta\phi = f$ over the rectangular domain Ω with Dirichlet boundary condition $\phi(p) = g(p)$ for all $p \in \partial\Omega$. The discretized update rule for unit step size $h = 1$ and a vanishing exterior heat potential $f = 0$ is given by the ordinary average over the four-neighbourhood of a pixel $(i, j) \in \Omega \backslash \partial\Omega$ [11]:

$$\phi[i,j] \leftarrow \frac{\phi[i-1,j] + \phi[i,j-1] + \phi[i,j+1] + \phi[i+1,j]}{4}.$$

```
double phi[N*M], tmp[N*M]; // input image and temp storage
double error = INFINITY, epsilon = 1E-6;

// update call for entry (i, j) as C++11 lambda
auto update = [&] (const int& i, const int& j) {
    tmp[i*M+j] = 0.25f*(phi[(i-1)*M+j] + phi[i*M+j-1]
                      + phi[(i+1)*M+j] + phi[i*M+j+1]);};

while (error > epsilon) {
    for (int i = 1; i < N-1; i++)
        for (int j = 1; j < M-1; j++)
            update(i, j);
    // determine maximum residue ||phi-tmp||_oo and copy tmp to phi
    error = uniform_norm(phi, tmp, N, M);
    copy_image(tmp, phi, N, M);
    // Note: consider synchronization of processes in your solution
}
```

Fig. 3. Repetitive convolution of an image ϕ of size $N \times M$ by averaging the four-neighborhood of a pixel. Note that the border pixels are not altered by the averaging.

The sequential implementation is similar to a repetitive convolution of an image ϕ with a cross-shaped stencil. Figure 3 depicts the source code for the single-threaded computation of the steady-state solution. A parallelization of the sequential algorithm can be achieved by independently updating each of the $(N-2)\cdot(M-2)$ interior points. Note, the implicit barrier at the end of the body of the while-loop. Using MPI, a suitable partitioning of the image ϕ has to be distributed to the individual processes. For the sake of simplicity, we choose a block distribution such that p tiles of size $\frac{(N-2)\cdot M}{p}$ are computed independently on p processes. The communication of the adjacent rows between the p tiles shall be accomplished asynchronously in each iteration of the while-loop (see Fig. 4). Afterwards, a global Allreduce collective determines the maximum error of all tiles and thus enforces synchronization. The educational goal of this task is the teaching of asynchronous communication primitives and the realization of synchronization with global barriers.

The students' task is to write the corresponding source code that handles the communication between the tiles. Figure 5 depicts a code skeleton that has to be completed by the students. This task was embedded as a pair programming exercise during a lecture on High Performance Computing (HPC) at the

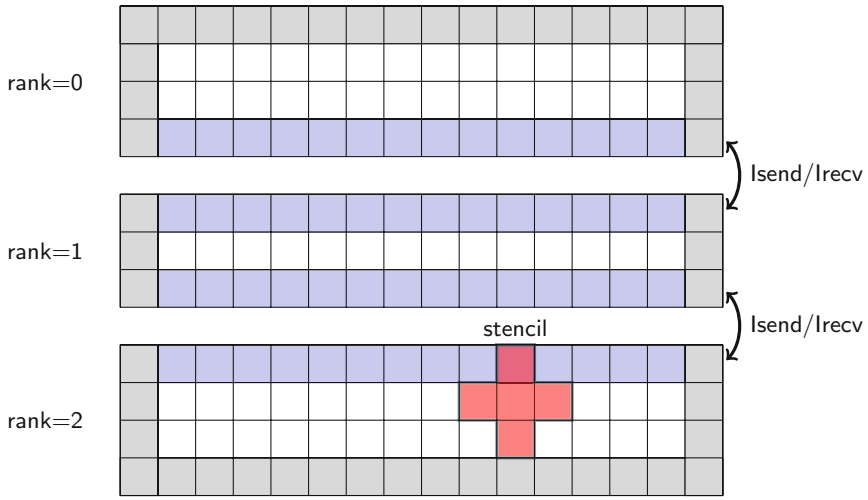

Fig. 4. An example of a block distribution of the image ϕ ($N = 11$ and $M = 16$) into $p = 3$ tiles of height $\frac{N-2}{3} = 3$. Border pixels (gray cells) are not altered during the update step. The adjacent rows (blue cells) have to be communicated between the processes during each iteration. The interior pixels (white cells) can be updated independently while waiting for the asynchronous communication (Color figure online).

Johannes Gutenberg University in the winter term 2014/15. Approximately half of the students could solve it within 15 min. The output verification is presented in textual and visual form: First, a test method determines if the submitted program computes an image ϕ_{par} that agrees with the sequential result ϕ_{seq} within a predefined error threshold (see Fig. 6). Second, SAUCE provides a visual comparison between the expected image and the computed result to the student (see Fig. 7). The additional visual information can be helpful to spot errors in the asynchronous communication calls. Furthermore, it is possible to expose additional information about the program which is not taken into account during the correctness test. As an example, the student can evaluate the program's speedup and efficiency in comparison to sequential code using runtime information.

Further MPI exercises including matrix multiplication using submatrix scattering, traditional matrix vector multiplication and the distributed numeric integration of functions using the trapezoidal rule have been successfully embedded in the tutorials and lectures of the aforementioned HPC course.

4.2 Odd-Even Sort Using OpenMP

SAUCE can also be used to teach parallel programming on shared memory architectures. In this paper, we present a multi-threaded example based on OpenMP. The parallelization of Odd-Even Sort, a modified variant of Bubble Sort that interleaves swaps of odd and even indices, can be used to illustrate the usage of thread pools in OpenMP (see Fig. 8). Furthermore, exercises using POSIX

```
// state sends and receives (upper and lower being the borders of the tile)
/** TO BE COMPLETED BY STUDENTS **
if (rank+1 < size) {
    double * lst_row = phi + (upper-1)*M, * nxt_row = lst_row + M;
    MPI_Isend(lst_row, M, MPI_DOUBLE, rank+1, 0, MPI_COMM_WORLD, req+2);
    MPI_Irecv(nxt_row, M, MPI_DOUBLE, rank+1, 0, MPI_COMM_WORLD, req+3);
}

if (rank > 0) {
    double * prv_row = phi + (lower-1)*M, * fst_row = prv_row + M;
    MPI_Isend(fst_row, M, MPI_DOUBLE, rank-1, 0, MPI_COMM_WORLD, req+0);
    MPI_Irecv(prv_row, M, MPI_DOUBLE, rank-1, 0, MPI_COMM_WORLD, req+1);
}
**/

// update the interior of the tile
for (int i = lower+(rank>0); i < upper-(rank+1<size); i++)
    for (int j = 1; j < M-1; j++)
        update(i, j);

// wait until asynchronous communication done (MPI_Wait)
/** TO BE COMPLETED BY STUDENTS **
if (rank+1 < size)

    MPI_Wait(req+3, sts+3);
if (rank > 0)
    MPI_Wait(req+1, sts+1);
**/

// update halo pixels at the tile borders
if (rank > 0)
    for (int j = 1; j < M-1; j++)
        update(lower, j);
if (rank+1 < size)
    for (int j = 1; j < M-1; j++)
        update(upper-1, j);

// now update error of approximation and copy tmp to phi
error = uniform_error(phi, tmp, N, M, lower, upper, 0, M);
copy_image(phi, tmp, N, M, lower, upper, 0, M);

// make sure all threads have the same error (why is this important?)
MPI_Allreduce(&error, &error, 1, MPI_DOUBLE, MPI_MAX, MPI_COMM_WORLD);
```

Fig. 5. The provided code skeleton of the while-loop body. The students are asked to fill in the blue-colored comments. Furthermore, they have to explain the importance of the Allreduce collective in order to ensure a deadlock-free MPI program (Color figure online).

Threads have also been successfully included in the lecture, e.g., the proper use of atomics, compare and swap loops, condition variables.

4.3 Array Reversal Using CUDA

Massively parallel accelerators can also be used with SAUCE since it effectively supports any programming language that can be compiled and executed on Unix-like operating systems. The demo instance provides a CUDA array reversal algorithm that uses shared memory to ensure coalesced accesses to global memory. The major goal of this task is to train the students' ability to utilize shared memory and the proper usage of the `__syncthreads()` primitive for

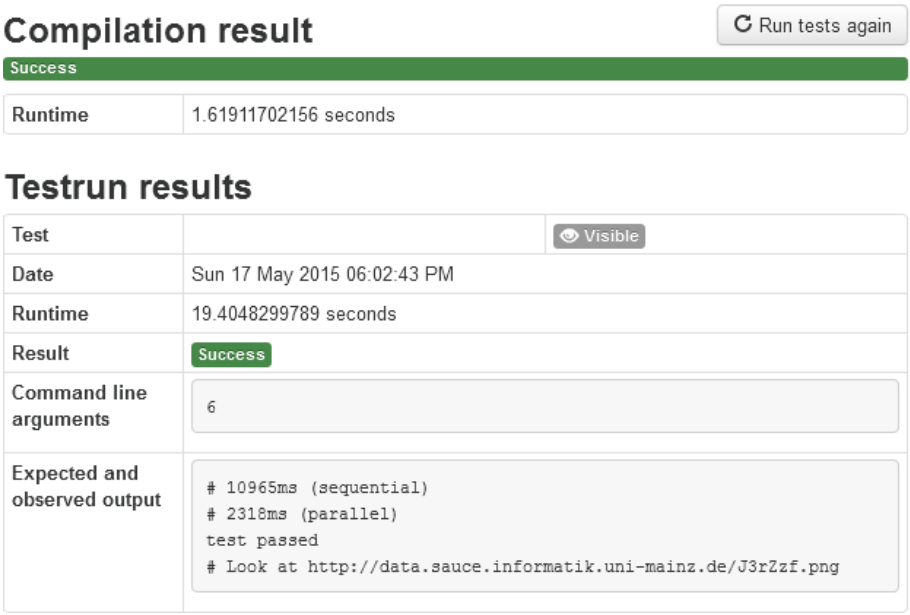

Fig. 6. Textual feedback for the correctness of the computed image ϕ provided by SAUCE. The test is passed if the maximum residue between the sequential and the parallel solution $\|\phi_{\text{seq}} - \phi_{\text{par}}\|_\infty$ is smaller than a predefined threshold. Furthermore, the teacher or student may write comments to the output via a print statement (here lines with leading #) that are not evaluated during the test e.g., the runtime of the program.

the synchronization of CUDA threads within a thread block. Further CUDA examples including matrix multiplication and image convolution have been successfully embedded in the aforementioned HPC course. The possibility to print out execution times allows for a detailed discussion whether the use of shared memory is beneficial in each specific case. Alternatively, a direct comparison of execution times between device and host code is conceivable.

4.4 Grading Features

SAUCE offers several features for the manual correction and grading of submitted source code. First, the teacher can annotate parts of the source code and leave suggestions or an improved version of the program for the students. Second, SAUCE features grading of individual submissions and team submissions of arbitrary size. Third, in rare cases where students submit source code that does not compile correctly, the teacher may clone the submission to his own user account and manipulate it freely. Fourth, it is possible to search the submission database for cases of plagiarism. To achieve this, a similarity score based on the Jaccard index is computed for all pairs of submissions and presented to the

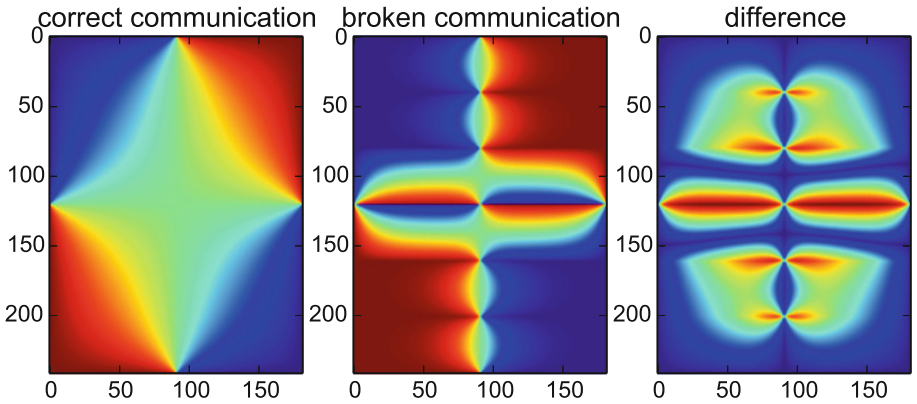

Fig. 7. Visual feedback for the resulting image ϕ provided by SAUCE. The left panel depicts the steady-state solution computed with correct asynchronous communication. The solution in the middle lacks any communication and thus five sharp horizontal edges can be observed using $p = 6$ processes, each of them computing an independent solution. The modulus of the difference between both solutions is plotted in the right panel. The color map (jet) ranges from small values (blue) to high values (red) (Color figure online).

```
void parallel_sort(std::vector<unsigned int>& X) {

    unsigned int i, phase, N = X.size();
    auto swap = [] (unsigned int& x, unsigned int& y)
                  { auto tmp = x; x = y; y = tmp; };
    /** TO BE COMPLETED BY STUDENTS
    #pragma omp parallel private(phase)
    **/
    {
        for (phase = 0; phase < N; phase++)
            if (phase % 2 == 0) {                  //even phase
                /** TO BE COMPLETED BY STUDENTS
                #pragma omp for
                **/
                for (i = 1; i < N; i += 2)
                    if (X[i-1] > X[i])
                        swap(X[i], X[i-1]);
            } else {                               // odd phase
                /** TO BE COMPLETED BY STUDENTS
                #pragma omp for
                **/
                for (i = 1; i < N-1; i += 2)
                    if (X[i] > X[i+1])
                        swap(X[i], X[i+1]);
        }
    }
}
```

Fig. 8. The provided code skeleton for the parallelization of Odd-Even Sort using OpenMP pragma statements. The students are asked to fill in the blue-colored comments. A further task could be the discussion of implicit barriers introduced by the `#pragma omp for` clauses (Color figure online).

teacher in multiple varieties (e.g., a list of the top-k similarity scores or dendrograms). Finally, SAUCE can also be used in programming contest scenarios since it provides dedicated features like hidden test cases as well as a fine-grained definition of time and resource limits.

5 Conclusion

In this paper, we have presented a unified framework for the automated assessment and evaluation of source code in the field of parallel programming which can be used from any device with just a web browser. The presented "System for AUtomated Code Evaluation" (SAUCE) is free and open source software (AGPL-3.0) and can be downloaded at [12]. Moreover, we have discussed its use in interactive programming exercises in the context of parallel programming. Three examples for the major parallelization paradigms MPI, OpenMP and CUDA have been described and provided as SAUCE exercises. A demo instance of SAUCE including the discussed examples can be accessed at our website [13].

References

1. Learning Tools Interoperability. http://www.imsglobal.org/lti/
2. Ashoo, S.E., Boudreau, T., Lane, D.A.: Programming Contest Control System. http://www.ecs.csus.edu/pc2/
3. Bayer, M.: The Python SQL Toolkit and Object Relational Mapper. http://www.sqlalchemy.org/
4. Breitner, J., Hecker, M.: Quality control for programming assignments. https://github.com/KITPraktomatTeam/Praktomat/
5. Cloud9: Online Code Editor. https://c9.io
6. Compilr: Online Editor and Sandbox. https://compilr.com
7. Goedicke, M., Striewe, M., Balz, M.: Computer aided assessments and programming exercises with JACK. Technical report (2008)
8. Higgins, C., Hegazy, T., Symeonidis, P., Tsintsifas, A.: The CourseMarker CBA system: improvements over Ceilidh. Educ. Inf. Technol. **8**(3), 287–304 (2003). http://dx.doi.org/10.1023/A:1026364126982
9. Ideone: Online Compiler and Debugging Tool. http://ideone.com
10. Molina, A.: TurboGears: Rapid Web Development Framework. http://turbogears.org
11. Quinn, M.J.: Parallel Programming in C with MPI and OpenMP. McGraw-Hill Education Group, New York (2003)
12. Schlarb, M.: System for AUtomated Code Evaluation on Github. https://github.com/moschlar/SAUCE
13. Schlarb, M., Hundt, C., Schmidt, B.: System for AUtomated Code Evaluation Cloud Service. http://sauce.informatik.uni-mainz.de
14. WebLab: Learning Management System. https://weblab.tudelft.nl/