Peachy Parallel Assignments (EduHPC 2019)

Mulya Agung*, Muhammad Alfian Amrizal*, Steven Bogaerts†, Ryusuke Egawa*,
Daniel A. Ellsworth‡, Jorge Fernandez-Fabeiro§, Arturo Gonzalez-Escribano§, Sukhamay Kundu¶,
Alina Lazar‖, Allen Malony**, Hiroyuki Takizawa*, David P. Bunde††

*_Tohoku Univ._
Sendai, Japan
Email: {agung@sc.cc.,alfian@ci.cc.,egawa@,takizawa@}tohoku.ac.jp

†_DePauw Univ._
Greencastle, IN, USA
Email: stevenbogaerts@depauw.edu

‡_Colorado College_
Colorado Springs, CO, USA
Email: dellsworth@coloradocollege.edu

§_Univ. Valladolid_
Valladolid, Spain
Email: {jorge,arturo}@infor.uva.es

¶_Louisiana State University_
Baton Rouge, LA, USA
Email: kundu@csc.lsu.edu

‖_Youngstown State Univ._
Youngstown, OH, USA
Email: alazar@ysu.edu

**_Univ. Oregon_
Eugene, OR, USA
Email: malony@cs.uoregon.edu

††_Knox College_
Galesburg, IL, USA
Email: dbunde@knox.edu

***Abstract*—Peachy Parallel assignments are high-quality assignments for teaching parallel and distributed computing. They have been successfully used in class and are selected on the basis of their suitability for adoption and for being cool and inspirational for students. Here we present a fire fighting simulation, thread-to-core mapping on NUMA nodes, introductory cloud computing, interesting variations on prefix-sum, searching for a lost PIN, and Big Data analytics.**

***Index Terms*—Parallel computing education, High-Performance Computing education, Parallel programming, Cloud Computing, Curriculum Development, Thread Mapping, Data Analytics, Prefix-sum, OpenMP, MPI, GPGPU, NUMA, Dask**

I. Introduction

Teaching parallel and distributed computing or high-performance computing requires assignments that help students understand both the potential and the challenges of these areas. They should illustrate the main techniques and be at the right difficulty level for the students being taught. Ideally, the assignments should also be interesting and exciting for the students to encourage their interest in the field. Creating assignments with all these characteristics is not easy. It involves both time and some risk since not every seemingly-great assignment idea becomes a successful assignment. To help educators save time and improve the quality of their assignments, a Peachy Parallel Assignment track was added to the Edu* series of workshops on Parallel and Distributed Computed Education. These assignments are presented at the workshops [18], [3] and also collected on a webpage (https://tcpp.cs.gsu.edu/curriculum/?q=peachy).

This work uses the Extreme Science and Engineering Discovery Environment (XSEDE) Jetstream at Indiana University and the Texas Advanced Computing Center, through allocation TG-CIE180012, which is supported by National Science Foundation grant number ACI-1548562.

Peachy Parallel Assignments all go through a competitive selection process based on the following criteria:

- Tested — All Peachy Parallel Assignments have been successfully used in a class.
- Adoptable — Peachy Parallel Assignments are easy to adopt. This includes not only the provided materials, but also the content being covered. Ideally, the assignments cover widely-taught concepts using common parallel languages and widely-available hardware, have few prerequisites, and (with variations) are appropriate for different levels of students.
- Cool and Inspirational — Peachy Assignments are fun and inspiring for students. They encourage students to spend time with the relevant concepts. Ideal Peachy Assignments are those that students want to demonstrate to their roommate.

This effort is inspired by the SIGCSE conference's Nifty Assignment sessions, which focus on assignments for introductory computing courses. (See http://nifty.stanford.edu for more details.) We are also particularly interested in assignments that are suitable for lower-level courses; these would facilitate the inclusion of parallel and distributed computing topics in those courses and correct a tendency of the first Peachy Assignments to focus more on higher-level courses [11].

In this paper, we present the following Peachy Parallel Assignments:

- A parallel simulation of heat propagation during a fire, with firefighter teams trying to extinguishing the focal points.
- Creating a thread-to-core mapping to optimize memory performance on a simulated NUMA system.
- Assignments and materials for introducing students to cloud resources (targeted to XSEDE, but adaptable to other platforms).
- Devising algorithms based on prefix-sum for a variety of problems.
- Searching for a PIN in parallel to match a hashed value.
- Big Data analytics using the Dask framework.

The Peachy Assignments webpage (https://tcpp.cs.gsu.edu/curriculum/?q=peachy) has the materials for each of these assignments (draft handout, given code, etc). It also lists the Peachy Assignments from previous competitions. Please come and browse for great assignment ideas. Then, consider submitting your own assignments to our next competition.

II. Agent-based Simulation of Fire Extinguishing (Gonzalez-Escribano and Fernandez-Fabeiro)

We present a new assignment used in a Parallel Computing course to solve the same problem in different parallel programming models. It targets concepts of shared-memory programming with OpenMP, distributed-memory programming with MPI, and/or GPU programming with CUDA or OpenCL. This assignment is based on a simplified agent-based simulation where teams of firefighters aim to extinguish a set of fire focal points in a dynamically evolving scenario. The program is designed to be simple, easy to understand by students, and to include specific parallelization and optimization opportunities. Although there is a quite direct parallel solution in the three programming models, the program has plenty of opportunities for further improvements. It extends the ideas of a previously presented assignment, in order to use more interesting data structures, load balancing techniques, and code optimizations. It has been successfully used in parallel programming contests during a real course, using the performance obtained by the students' code as a measure of success.

A. Idea and context

Different programming models use different approaches for the parallelization of application structures. Understanding these differences is key for students to get into more advanced techniques, and to create parallel programs for current heterogeneous platforms. For several years, we have been teaching a course of Parallel Programming that introduces the basics of OpenMP, MPI, and CUDA or OpenCL. A Peachy Assignment was previously presented [3], which was designed to be parallelized by the students using each of these programming models during three one-week programming contests. The students compete to obtain the best performance on each contest. Although this kind of assignment can be used to successfully teach a single programming model, it can also show which concepts and ideas can be reused across different models, and which cannot, exposing the approach differences and the conceptual shift between models. For example, the students learn the differences between controlling race-conditions in shared-memory vs. using distributed data structures with explicit communications vs. dealing with tiling and memory hierarchies in GPU coprocessors.

This new Peachy Assignment maintains a clear focus on simple but effective code parallelizations and optimizations, while introducing more opportunities for the advanced students and more choices to teach a single programming model. This includes dealing with 2D data structures, a wider range of synchronization issues with different solutions in different models, and more interesting load-balancing and code optimization decisions.

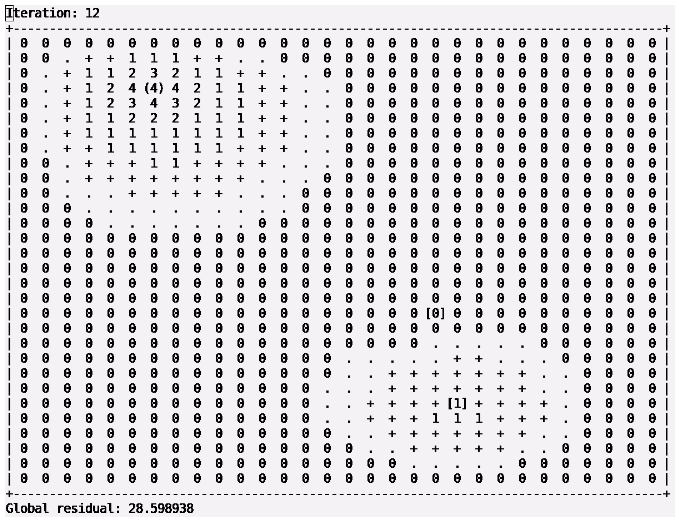
Heat-propagation simulation is a classical example that can be used to teach concepts of parallel programming. For example, in [18] a Peachy Assignment was presented that simulates the heat diffusion of a single campfire point, with the possibility of including isolation surfaces, and with an interesting online graphical representation that helps the students to visualize the effect of typical parallelization bugs.

In this assignment, we merge heat propagation with an agent-based simulation, where agents are fire-extinguishing teams. This combination is the key to allow the design of different load-balance situations. In our assignment, two-dimensional arrays store the heat values in a discretized representation of the simulation arena. Fire focal points arise at established places and simulation times, and their heat is propagated to neighbor cells step by step. Teams of firefighters are represented by agents that make simple decisions to advance, one cell at a time, in the direction of the next nearest focal point to extinguish it. While teams advance, they also reduce the heat of cells within a given radius. There are three different team classes with different movement rules and different heat-reduction radii. The size of the simulation surface grid, the number and start-frequency of focal points, and the number and class of the agents, are the main dials to design a new scenario with different load-balance properties.

As in the previous assignment, the provided material includes a sequential code, a test-bed of input files, and a handout explaining the assignment. The students can use common compilers and PC platforms to develop and test their codes. An automatic judge tool with an on-line public ranking is used to provide a fair arena, and to keep the students engaged during the contests. Other gamification tools are also included [9].

B. Concepts covered

The stages of each simulation step are: (1) Activate new focal points according to the simulation clock; (2) Propagate heat using a stencil operator implementing a Jacobi iterative solver for the Poisson's equation; (3) Reduction to obtain the maximal residual error; (4) Movement decisions for each team; (5) Execute team actions. To balance the speed of the team movements with the heat propagation, ten iterative propagation



```
Iteration: 12
+------------------------------------------------------------------------------------------+
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  .  +  +  1  1  1  +  +  .  .  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  .  +  1  1  2  3  2  1  1  +  +  .  .  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  .  +  1  2  4 (4) 4  2  1  1  +  +  .  .  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  .  +  1  2  3  4  3  2  1  1  +  +  .  .  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  .  +  1  1  2  2  2  1  1  1  +  +  .  .  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  .  +  1  1  1  1  1  1  1  1  +  +  .  .  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  .  +  +  1  1  1  1  1  1  +  +  +  .  .  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  .  +  +  +  1  1  +  +  +  +  .  .  .  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  .  +  +  +  +  +  +  +  +  .  .  .  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  .  .  .  +  +  +  +  +  .  .  .  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  0  .  .  .  .  .  .  .  .  .  .  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  0  0  .  .  .  .  .  .  .  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 [0] 0  0  0  0  0  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  .  .  .  .  .  0  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  .  .  .  .  .  +  +  .  .  .  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  .  .  +  +  +  +  +  +  +  .  .  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  .  .  .  +  +  +  +  +  +  +  .  .  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  .  .  +  +  +  + [1] +  +  +  +  .  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  .  .  +  +  +  1  1  1  +  +  +  .  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  .  .  +  +  +  +  +  +  +  .  .  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  .  .  +  +  +  +  +  .  .  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  .  .  .  .  .  0  0  0  0  0  0  0 |
| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |
+------------------------------------------------------------------------------------------+
Global residual: 28.598938
```

Fig. 1. Graphical representation of the output of fire-extinguishing simulation program at a given step. The numbers and symbols represent the heat at the different points of the 2D surface. Active focal points are represented with brackets. The position of firefighter teams are represented with square brackets.

steps are computed sequentially. The simulation stops after a given number of steps, or when the residual heat reaches a stable point with a given threshold. The output of the program is the number of simulation steps and a list of final heat values at selected positions. In debug mode, the program also writes a text-mode graphical representation that can be used to visualize the simulation at each simulation step; see Figure 1. It represents the heat on the surface, the active focal points, and the team positions.

The basic concepts covered in OpenMP are parallelization of loops, reductions, and avoiding race conditions with atomic operations. In MPI, the students work with array partitions, halos and neighbor communications, and generic reductions. For GPU programming, the main ideas are embarrassingly parallel kernels, minimizing host-device communication operations, thread-block sizes, non-trivial atomic operations, and simple reductions.

More advanced optimizations can be discovered and applied. Apart from those presented in [3], this new assignment introduces opportunities to test different partition policies for 2D arrays, precompute access patterns during the heat-reduction applied by the teams, use different load-balancing techniques in terms of input data features, using loop scheduling clauses in OpenMP, taking decisions about replicated vs. distributed computing in MPI, fusing kernels, new uses for the shared memory or non-trivial reductions on GPUs, etc.

C. Variants

The assignment can easily be adapted and modified by the teacher to include new simulation details, such as wind, obstacles, heat damage, etc. Instead of focusing only in parallelization and code optimization, the decision rules of the agents that simulate the firefighter teams can also be targeted to be optimized by the students. Better graphical and online interfaces can be devised to enrich the learning experience (see for example [18]).

III. Thread-to-core optimization (Amrizal, Agung, Egawa, Takizawa)

Many interesting performance tuning problems exist in the field of HPC. However, these problems mostly require advanced programming skills and deep understanding of the programming models, system architectures, compilers, and so forth. Thus, introducing them to undergraduate level students who generally do not possess such skills is challenging. Therefore, it is important to introduce these tuning problems to the students in a more general way, i.e., a programming-language-free fashion, in order to keep the students interested and motivated to solve such problems. In this assignment, we introduce students to one kind of tuning problem called *thread-to-core optimization problem*.

This assignment is part of a class called "Experiment D", which is a compulsory subject for the fourth-year undergraduate students in the Department of Electrical, Information, and Physics Engineering of Tohoku University. The class is also open to graduate students who are interested in HPC. Three sessions are allocated to introductory experiments using OpenMP and one session to this assignment. Since HPC and parallel and distributed computing (PDC) have not been integrated to the undergraduate level curriculum, the participants do not have any kind of experience with parallel programming prior to this class. Thus, an assignment that puts more emphasize on algorithmic thinking is introduced to increase students' interest in HPC/PDC topics in general.

Given a multi-threaded application and a multi-core non-uniform memory access (NUMA) system (Figure 2), students are challenged to determine on which core of the system each thread should be placed. They need to develop a thread-to-core placement algorithm that minimizes the total execution time of the application. In a NUMA system, processor cores are grouped and a group of processor cores is called a *numa node* (or simply a "node") [10]. A numa node is associated with a local memory sub-system that consists of caches, memory controllers, and DRAM, as shown in Figure 2. Thus, the performance of memory accesses depends on the location of the threads that access the data of the node [7]. Although placing highly-communicating threads to the same node might improve performance due to the improved data access locality, placing too many of them in the same node could also decrease performance due to the increased pressure on the local memory sub-system (cache contention, congestion in memory controllers, etc.) [1]. The total execution time is expected to be minimal when a balance between data access locality and pressure on local memory sub-system is achieved. Therefore, students need to explore and develop some load balancing techniques to solve this problem.

Students are presented with two types of communication matrices extracted from an application. The matrices represent: 1) the total number of communication events and 2) the total size of communication between every pair of threads. Figure 3

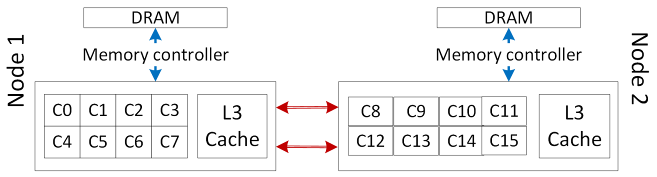
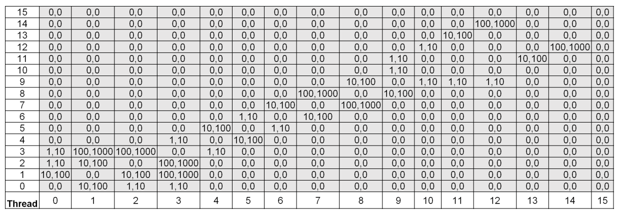


Fig. 2. A modern NUMA system, with two nodes and eight cores per node.



15	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
14	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	100,1000	0,0	0,0	0,0
13	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	10,100	0,0	0,0	0,0	0,0
12	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	1,10	0,0	0,0	0,0	100,1000	0,0
11	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	1,10	0,0	0,0	0,0	10,100	0,0	0,0
10	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	1,10	0,0	0,0	0,0	0,0	0,0	0,0
9	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	10,100	0,0	1,10	1,10	1,10	0,0	0,0	0,0
8	0,0	0,0	0,0	0,0	0,0	0,0	0,0	100,1000	0,0	10,100	0,0	0,0	0,0	0,0	0,0	0,0
7	0,0	0,0	0,0	0,0	0,0	0,0	10,100	0,0	100,1000	0,0	0,0	0,0	0,0	0,0	0,0	0,0
6	0,0	0,0	0,0	0,0	0,0	1,10	0,0	10,100	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
5	0,0	0,0	0,0	0,0	10,100	0,0	1,10	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
4	0,0	0,0	0,0	1,10	0,0	10,100	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
3	1,10	100,1000	100,1000	0,0	1,10	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
2	1,10	10,100	0,0	100,1000	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
1	10,100	0,0	10,100	100,1000	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
0	0,0	10,100	1,10	1,10	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
Thread	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Fig. 3. An example of a communication matrix of a 16-threads application. Each cell (x,y) of the matrix contains two numbers separated by a comma: (the total number of communication events, the total size of communication). In this example, most threads communicate only with their neighbouring threads.

shows an example of one of the communication matrices for an application that consists of four threads. Using this information, students need to create a set of pairs (thread_id:core_id) matching each thread with a core. In addition, students are given a multi-core simulator that simulates a real NUMA system. The simulator takes the set of (thread_id:core_id) pairs as input and gives some outputs such as execution time, L3 cache misses, and DRAM queuing delay. Students can refer to these numbers to see the effectiveness of their thread-to-core placement algorithm. Note that students basically only need these two matrices to solve the thread-to-core optimization problem and hence this assignment is appropriate even for students with little to no parallel programming skill.

This assignment covers some important concepts such as: 1) memory organization, 2) load balancing, 3) simulator based performance tuning, and 4) algorithmic optimizations.

We noted at least two strengths of this assignment: 1) it is challenging, yet doable, and 2) the students enjoy it because they can compete with each other to achieve minimal execution time. As for the weaknesses of this assignment: 1) it is oriented specifically toward thread-to-core optimization problem, which is mainly done at the research-level of HPC (not generally known/used by standard HPC users), and 2) it requires a high-end multi-core server to speed up the simulator. Finally, more assignment variations can be made by not only considering the spatial aspect (data access locality) but also the temporal aspect (considering threads that communicate almost at the same time) to reduce pressure on the memory sub-system.

The material of the assignment, including the assignment handout, the simulator, and some sample results are available at https://www.sc.cc.tohoku.ac.jp/~tacky/index-e.html (user: sc19, passwd: EduHPC-19).

IV. The Cloud for All Levels (Bogaerts)

Undergraduate projects in parallelism often use a multi-core processor on a local machine, but many real-world applications of parallelism use cloud-computing resources. Thus, this adaptable project is designed to give students a first experience in parallel computing in the cloud. This project was applied in an upper-level undergraduate data mining course that uses Python with pandas and scikit-learn. However, the provided materials are also appropriate for lower-level courses, including CS1, and a CS1-level exercise set is included. The project is also highly adaptable in time spent, being suitable as a standalone assignment of a couple hours, or as a component of a much broader project.

While there exist many distinct high-level interfaces for cloud-computing resources, this project focuses on the more uniform command-line interface. Introductory exercises guide students to connect via ssh and scp, manage files, edit code via Emacs, and run code at the command line. Students then work through another tutorial on accessing the cloud computing resource itself – in this case, Jetstream via XSEDE [22], available for educational use through a simple application process. The tutorial explains the idea of a virtual machine, management of a limited computing allocation, setting up public key infrastructure for the transfer of files, configuring the virtual machine, and running code. While some technical details are specific to XSEDE, the concepts are universal. All of the above materials are suitable for all undergraduate levels.

The remainder of the project depends on the desired time allocation and the level of the students. For a short assignment in CS1, students can run the code with different parameters by following the provided exercises, and report results about running time according to Amdahl's Law. For upper-level courses, instructors can give a short assignment by providing the same code as in CS1, and instructing students in conducting a full hyperparameter grid search. In a longer project, students can develop their own code for not just the grid search itself, but also data transformation and algorithm implementation, in a wide range of application domains.

In the data mining course, we use a Kaggle.com dataset for predicting the selling price of homes given several dozen attributes. While the dataset is a modest size, the range of useful experiments and algorithm tuning efforts is quite extensive. As such, students must use parallelism on a cloud-computing resource in order to conduct a detailed analysis.

Having established the procedures for accessing cloud computing resources, students are ready to apply these resources to obtain significant speedup in their data mining projects – starting with serial code either provided by the instructor or developed themselves. Students quickly learn that simply running this serial code on a highly-parallel virtual machine does not give automatic speedup. Rather, their code must be adapted to make use of the parallel resources. Simple adaptations include the use of scikit-learn operations with an `n_jobs` parameter, by which the programmer can specify the number of parallel processes to run simultaneously to complete that operation.

For example, the function `cross_val_score` computes a k-fold cross validation score, while `GridSearchCV` conducts a grid search for hyperparameter tuning of a chosen machine learning algorithm. Both of these functions include an `n_jobs` parameter. Of course this is only the beginning, as students are also ready at this point to create parallel processes via the Python multiprocessing module. Much data processing work is embarrassingly parallel, and can be sped up significantly through this module. Through this process, students also see Amdahl's Law in action. While they have access to up to 44 virtual CPUs in a virtual machine on XSEDE, students can observe how much speedup is actually possible, based on the proportion of their code that is parallel.

To summarize, a couple hours of background is required before students are ready for parallel computing in the cloud. On the one hand, this delays the students' entry into the fun of parallel programming. On the other hand, these barriers, if never surpassed, may prevent students from ever running parallel code on anything beyond a local multicore machine. One major contribution of this Peachy Assignment, then, is the adaptable exercises to get students up to speed quickly in the prerequisite concepts, with no assumption of prior experience. The materials submitted here include the introductory tutorials and exercises, the CS1-level final exercises, and the advice for upper-level students for effectively using cloud computing resources in their project. While some technical details may vary (e.g. a different cloud computing resource), these documents describe primarily universal concepts.

V. Applications of Parallel Prefix Sum (Kundu)

The parallel prefix-sums algorithm and its variations are often used as key substeps in more complex parallel computations. We have successfully introduced the parallel prefix-sum algorithm and its key ideas in a senior level undergraduate Computer Science course. To improve their understanding of these ideas, students were required to solve several problems using simple variations of the basic prefix-sums algorithm. They were also required to solve several non-trivial problems, including their 2-dimensional generalizations. We briefly discuss here four problems of this second category.

Let $x[0..(N-1)]$, in short $x[\,]$ or simply x when no confusion is likely, be an array of numbers. We call the sum $s[j] = \sum_{i=0}^{j} x[i]$ the j^{th} (inclusive) prefix-sum of x. The prefix-sums of $x = [3, 12, -4, 8, 2]$ then equal $s = [3, 15, 11, 19, 21]$. Many computation problems involve prefix-sums in some form [4], [5], [6], [23]. For example, if $x[j]$ is the probability of #(heads) = j in $N-1$ tosses of a coin, then $s[j]$ is the probability of #(heads) $\leq j$. An optimal sequential algorithm to compute the prefix-sums is to let $s[0] = x[0]$ and $s[j] = s[j-1] + x[j]$ for $1 \leq j < N$. This takes $O(N)$ time. A parallel algorithm for computing the prefix-sums takes $O(\log N)$ time using $O(N)$ CPU's or computing agents [4], [5].

We define prefix-mins of x to be the array m, where $m[j]$ = $\min\{x[i] : 0 \leq i \leq j\}$. The prefix-maxs of x is defined similarly, and each of these can be computed in time $O(\log N)$ as well using $O(N)$ agents. In what follows, we use the convention $z[-1] = 0$ for any array $z[0..(N-1)]$.

A. The maximum consecutive sum problem: MCS

For a given x, we want to find a subarray $x[i..j]$ of x with maximum sum of items $s(i,j) = \sum_{k=i}^{j} x[k] = s[j] - s[i-1]$, where s is the prefix-sums of x. Let $MCS(x)$ be the "leftmost" subarray $x[i..j]$ having the maximum $s(i,j)$, with smallest possible j and the largest $i \leq j$ for that j. Also, let $\sigma MCS(x)$ be the sum of items in $MCS(x)$. Let m be the prefix-mins of s and $s_m[j] = s[j] - m[j-1]$, with $s_m[0] = s[0] = x[0]$. Then, $\sigma MCS(x) = \max\{s(i,j) : i \leq j\}$ = $\max\{s_m[j] : 0 \leq j < N\}$. Thus, $\sigma MCS(x)$ can be computed in $O(\log N)$ time. It is not hard to see that $MCS(x)$ can also be computed in $O(\log N)$ time.

Example 1. Figure 4 shows an x and its associated s, m, and s_m. Here, $MCS(x) = x[4..7]$ = [11, −2, −6, 12] and $\sigma MCS(x) = s[7] - m[6] = 10 - (-5) = 15$. Figure 4 also shows the array items $indx[j] = \max\{i : 0 \leq i \leq j$ and $s[i] = m[j]\}$. This is needed to find i in $x[i..j] = MCS(x)$. We compute $indx[j]$'s as we compute $m[j]$'s. Note that j in $x[i..j] = MCS(x)$ equals $\min\{j' : s_m[j'] = \sigma MCS(x)\}$. ■

	$x[0]$	$x[1]$	$x[2]$	$x[3]$	$x[4]$	$x[5]$	$x[6]$	$x[7]$	$x[8]$	$x[9]$
x:	−2	1	3	−7	11	−2	−6	12	−3	−1
s:	−2	−1	2	−5	6	4	−2	10	7	6
m:	−2	−2	−2	−5	−5	−5	−5	−5	−5	−5
s_m:	−2	1	4	−3	11	9	3	15	12	11
$indx$:	0	0	0	3	3	3	3	3	3	3

Fig. 4. Illustration of the arrays s, m, s_m and $indx$ for a given array x.

B. Largest block of consecutive 1's: LBO

Assume that each $x[i] = 0$ or 1, at least one $x[i] = 0$, and at least one $x[i] = 1$, i.e., $0 < s[N-1] < N$, where s is the prefix-sums of x. We can thus test the above property by computing s in $O(\log N)$ time. We want to find a largest length subarray $x[i..j], i \leq j$, of x consisting of only 1's. Let $LBO(x)$ denote the "leftmost" such a subarray $x[i..j]$, with j as small as possible and the largest $i \leq j$ for that j. Also, let $\sigma LBO(x)$ denote length($LBO(x)$) = $j - i + 1$ = the sum of items in $LBO(x)$. Thus, $x = [1, 0, 1, 1, 1, 0, 1, 1, 1]$ gives $LBO(x) = x[2..4]$ and $\sigma LBO(x) = 3$. If we write x' for the array obtained by replacing each 0 in x by $-N$, then it is easy to see that $MCS(x') = x'[i..j]$ if and only if $LBO(x) = x[i..j]$ and hence $\sigma MCS(x') = \sigma LBO(x)$. Using $O(N)$ agents, we can determine x' from x in $O(1)$ time and thus we can compute $LBO(x)$ and $\sigma LBO(x)$ in time $O(\log N)$ using $O(N)$ agents.

C. Two dimensional MCS: MCS2

This is a generalization of the MCS-problem in §V-A to 2-dimensional arrays. For an $M \times N$ matrix x, $0 \leq r_1 \leq r_2 < M$, and $0 \leq c_1 \leq c_2 < N$, let $x(r_1, r_2, c_1, c_2)$ be the submatrix of x consisting of rows r_1 to r_2 and columns c_1 to c_2.

We want to find an $x(r_1, r_2, c_1, c_2)$ such that its sum of items $s(r_1, r_2, c_1, c_2) = \sum_{i=r_1}^{r_2} \sum_{j=c_1}^{c_2} x[i, j]$ is maximum. Let $MCS2(x)$ denote an optimum submatrix of x and as usual we want to find an $x(r_1, r_2, c_1, c_2) = MCS2(x)$ with the smallest r_2, the largest r_1 for that r_2, the smallest c_2 for that r_1 and r_2, etc. Let $\sigma MCS2(x)$ be the sum of items of an $MCS2(x)$. Without loss of generality, we assume $M \leq N$.

Example 2. For the matrix x in Figure 5, each subarray of column 3 has its sum of items negative if it contains -5. This implies that $MCS2(x)$ does not contain the item -5 and that means it equals row 0, i.e., $x(0, 0, 0, 3)$, or row 2, i.e., $x(2, 2, 0, 3)$, or all 3 rows and the first 3 columns of x, i.e., $x(0, 2, 0, 2)$. Thus, $MCS2(x) = x(0, 2, 0, 2)$ and $\sigma MCS2(x) = 13$.

We solve the $MCS2$ problem by reducing it to a set of 1-dimensional MCS problems by considering groups of consecutive rows of x and then taking the best solution of those 1-dimensional MCS problems. For $0 \leq r_1 \leq r_2 < M$, let x_{r_1,r_2} be the 1-dimensional array of column-wise sums of items in rows r_1 to r_2, i.e., $x_{r_1,r_2}[j] = \sum_{i=r_1}^{r_2} x[i, j]$. Clearly, $\sigma MCS(x_{r_1,r_2}) = \sigma MCS2(x)$ if and only if $\sigma MCS(x_{r_1,r_2})$ = $\max\{\sigma MCS(x_{r'_1,r'_2}) : 0 \leq r'_1 \leq r'_2 < M\}$. To compute $MCS2(x)$, we simply choose smallest r_2 and largest $r_1 \leq r_2$ such that $\sigma MCS(x_{r_1,r_2}) = \sigma MCS2(x)$. We can efficiently compute all x_{r_1,r_2} as follows. Let y_c be the column c of x, i.e., $y_c[i] = x[i, c]$ and let s_c be the prefix-sums of y_c. Also, let x' denote the $M{\times}N$ matrix whose columns are $y'_c = s_c$ and let x'_r be the row r of x'. Thus, $x'_r[j] = \sum_{i=0}^{r} x[i, j] = x_{0,r}[j]$ and hence $x_{r_1,r_2} = x'_{r_2} - x'_{r_1-1}$, i.e., $x_{r_1,r_2}[j] = x'_{r_2}[j] - x'_{r_1-1}[j]$, where $x'_{-1}[j] = 0$ by convention. See Figure 5.

Using $O(MN)$ agents, we can compute in parallel all prefix-sums $s_c, 0 \leq c < N$, and hence the matrix x' in $O(\log M)$ time. For each fixed r_1, we can compute all x_{r_1,r_2} in parallel in time $O(1)$ and then compute all the related $MCS(x_{r_1,r_2})$ and $\sigma MCS(x_{r_1,r_2})$ in time $O(\log N)$ based on the results in §V-A. This gives a total time $O(M \log N)$ time as we vary r_1. Finally, the number of (r_1, r_2)-combinations is $M(M{+}1)/2 \leq MN$ and hence the best of all $\sigma MCS(x_{r_1,r_2})$ and its associated $MCS(x_{r_1,r_2})$ can be computed in time $O(\log(MN))$. This gives the total time $O(M \log N)$ for $MCS2(x)$ and $\sigma MCS2(x)$ using $O(MN)$ agents. ■

D. An image processing application of MCS2

Figure 6(i) shows an 8×8 black-and-white image. We want to find a largest size rectangular black area in such an image. The rows 3 to 6 and columns 1 to 3 gives a largest black rectangular area of size 12 in Figure 6(i).

An $N{\times}N$ image can be represented by an $N{\times}N$ matrix x, where $x[i, j]$ = 1 or 0 according as the pixel (unit square) in row i and column j is black or white. One way to solve this problem is by converting it to a 2-dimensional version of LBO problem. Hence, we can call this problem $LBO2$. Let x' be the matrix obtained by replacing each 0 in x by $-N^2$. Assuming that at least one pixel in x equals 1, a solution of $MCS2(x')$ would not involve a 0-pixel of x. Thus, $MCS2(x')$ gives a solution of $LBO2(x)$, i.e., a largest area black rectangle of x. This takes $O(N \log N)$ time using N^2 agents based on the results in §V-C.

$$\begin{bmatrix} 1 & 0 & 2 & 1 \\ 2 & 1 & 2 & -5 \\ 1 & 3 & 1 & 0 \end{bmatrix}$$

(i) A matrix x.

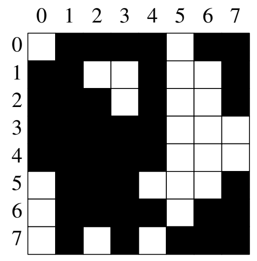
$$\begin{bmatrix} 1 & 0 & 2 & 1 \\ 3 & 1 & 4 & -4 \\ 4 & 4 & 5 & -4 \end{bmatrix}$$

(ii) The matrix x' of prefix-sums of columns of x.

r_1	r_2	$x_{r_1,\,r_2}$	$MCS(x_{r_1,\,r_2})$	$\sigma MCS(x_{r_1,\,r_2})$
0	0	[1, 0, 2, 1]	$x_{0,\,0}[0..3]$	4 = 1+0+2+1
	1	[3, 1, 4, –4]	$x_{0,\,1}[0..2]$	8 = 3+1+4
	2	[4, 4, 5, –4]	$x_{0,\,2}[0..2]$	13 = 4+4+5
1	1	[2, 1, 2, –5]	$x_{1,\,1}[0..2]$	5 = 2+1+2
	2	[3, 4, 3, –5]	$x_{1,\,2}[0..2]$	10 = 3+4+3
2	2	[1, 3, 1, 0]	$x_{2,\,2}[0..2]$	5 = 1+3+1

(iii) $\sigma MCS2(x)$ = max of $\sigma MCS(x_{r_1,\,r_2})$'s = 13 = $\sigma MCS(x_{0,\,2})$ and $MCS2(x) = x(0, 2, 0, 2)$ because $MCS(x_{0,\,2}) = x_{0,\,2}[0..2]$.

Fig. 5. Illustration of computation of $MCS2(x)$ and $\sigma MCS2(x)$.



(i) An 8×8 black-and-white image x; N = 8.

$$\begin{bmatrix} -64 & 1 & 1 & 1 & 1 & -64 & 1 & 1 \\ 1 & 1 & -64 & -64 & 1 & -64 & -64 & 1 \\ 1 & 1 & 1 & -64 & 1 & -64 & -64 & 1 \\ 1 & 1 & 1 & 1 & 1 & -64 & -64 & -64 \\ 1 & 1 & 1 & 1 & 1 & -64 & -64 & -64 \\ -64 & 1 & 1 & 1 & -64 & -64 & -64 & 1 \\ -64 & 1 & 1 & 1 & 1 & -64 & 1 & 1 \\ -64 & 1 & -64 & 1 & -64 & 1 & 1 & 1 \end{bmatrix}$$

(ii) The matrix x'; black pixels correspond to 1 and balck pixels to $-64 = -N^2$.

Fig. 6. The matrix x' of a black-and-white image x; $MCS2(x')$ equals rows 3 to 6 and columns 1 to 3, giving a largest black rectangular area in x.

E. Conclusion

We have presented here four problems, involving the applications of the basic prefix-sums computation, that we successfully used for introducing parallel computation in a senior-level undergraduate Computer Science course. The MCS-problem is the core of these problems, and the students needed several hints to arrive at its solution. The students could easily solve the LBO-problem given the hint to convert it to the MCS-problem by modifying the original input array. The students could solve the $MCS2$-problem, which is a 2-dimensional generalization of the MCS-problem, by applying MCS to groups of rows of the input matrix. Finally, the image processing problem is a 2-dimensional generalization of the LBO problem and the students could easily solve it.

VI. Pin Finder (Ellsworth and Malony)

Pin Finder was initially developed as the first parallel programming lab in a 10-week shared memory parallel programing course at the University of Oregon in 2015[12], [14]. The course used Intel's complier technologies and Structured Parallel Programming[15] as the textbook. Labs in this course were gamified by placing students in the context of developers at the Office of Strategic National Alien Planning (OSNAP). Each lab was 50 minutes and would start with a brief reminder of a parallel pattern from the textbook, an OSNAP business problem, and serial code in C that would slowly solve the problem. Students would spend roughly 45 minutes working on a solution with a grad student to help with questions.

The Pin Finder assignment was also used at Colorado College in a Fall 2017 parallel programming course[8] based on the University of Oregon course. Courses at Colorado College are 18 days long, meeting for 3 hours of class each day. During the parallel patterns portion of the course, the last hour of class was used as a lab. No grad student support was available and the OSNAP gamification was mostly removed for this delivery.

A. Sample Lab Prompt

OSNAP security policies demand workers use an 8-digit pin that may not be written down, cannot be reset, and is changed daily to access secured resources. Executive management has requested a pin recovery tool since the pins are frequently forgotten. To be compliant with organizational security policies, no system or person may keep pins in plain text or reversibly encrypted. A hashed pin has no confidentiality requirements according to OSNAP policies.

1) User carries the hashed pin
2) User enters the hash when the pin is needed
3) Software hashes all pins
4) Software return a matching pin

Serial code has been provided that supports this procedure, but it is too slow. How much faster can you make this using parallelism?

B. Discussion Topics

The lab can be connected to several different individual or class discussion topics based on instructor and student interests.

1) Parallelism: The lab is placed early in the course when the map pattern is being discussed. In my deliveries correctly using a parallel-for is sufficient to produce a passing lab solution. For the right pins, even with a naive parallel-for, good speed-up can be observed on hardware with low core counts. Placing the parallel-for is relatively easy, but a lack of care in variable handling results in errors due to race conditions.

2) Performance: Depending on where the pin is located in the search space, the observed performance improvement of parallel-for over the serial loop varies since the serial code uses early loop termination. This provides an opportunity to talk about the need to be aware of how parallelism is implemented, overheads, and care in design when deploying parallelism. In the Spring 2017 delivery at the University of Oregon a leader board of student submissions was used to encourage students to optimize their code[17].

More performant solutions comparative to the serial solution will involve students considering how the search is conducted and how the threads interact. Students might try a serial loop around the parallel region, a shared variable to flag to skip work loop bodies after a solution has been found, changing how the search space is allocated across threads, or a parallel pattern other than map.

3) Security: Hashing is one-way but collisions must be considered. Hashed passwords are better than plaintext but are not sufficient if the hash is not properly protected. Password crackers, like the simple one being built in this lab, can recover a working password from a hash sniffed over the network or exfiltrated from a database given sufficient resources. Adding blocks to blockchains, via proof of work, frequently involves a similar problem to the pin finder.

C. Common Stuck Points

The lack of guidance around construction of the final solution invites creativity in the problems students encounter.

1) Finding a Parallel Pattern: This assignment has been used when the map pattern is being discussed and most students therefore gravitate toward a parallel-for. Students who are unsure where they might start are encouraged to think about what the while loop in the serial code is doing (iterating over a range) and to modify the code to use a for-loop.

2) Test Iteration Time: Using a large PIN space enables even naive solutions to achieve good observed speedup for some PINs on low core counts, but using the full search space makes testing slow. Students who are waiting a long time for tests should be encouraged to think about whether the whole search space is needed during testing.

3) Wrong/no PIN Returned: The serial code uses variables declared outside of the loop. Of specific interest is a character buffer used in converting pin numbers to corresponding strings for hashing. Naively adding parallelism results in a race condition where the wrong pins will be checked, resulting in no matches, or the returned "matching" pin will be incorrect. Students might be reminded of race conditions from the lecture/textbook or asked to consider how the shared memory model interacts with threads.

VII. Dask processing and analytics for large datasets (Lazar)

This paper describes the assignment titled "Dask Analytics" that is used for the assessment of student learning as part of a graduate data science and data mining course. However, the assignment could be easily adapted for upper division undergraduate courses. For this assignment, students are required to read, process and answer queries using a large dataset that does not fit in the RAM memory of a commodity laptop. Using the Python framework Dask, which extends a small set of Pandas's operations, students can become familiar with parallel and distributed processing. In addition, the assignment teaches students about the basic operations implemented in Dask in a very interesting and applied way, as well as operations and algorithms that are harder to parallelize.

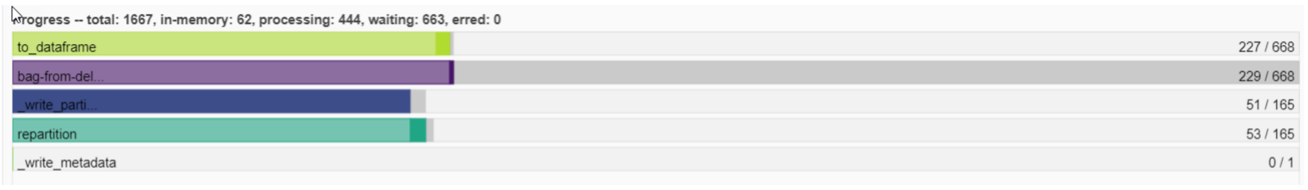


Fig. 7. Part of the Dask dashboard shows the processing progress and how the problem is divided in smaller chunks.

A. *Motivation*

Recently, the Pandas library has become one of the most popular and favourite data science tools for the Python programming language to perform exploratory data analysis. Not only that, but Pandas is usually used for data preprocessing and transformation, operations required before using any algorithms part of the Scikit-Learn machine learning library [19]. Pandas and Scikit-Learn work great for tabular datasets that fit in memory (e.g. the size of the dataset is less than 1 GB), with no concern about the performance. However, for datasets between 1 GB and 100 GB that do not completely fit in the RAM of a commodity computer, other solutions are needed. Usually, students do not have access to parallel or distributed computing. This type of problem provides a good appropriate challenge to stimulate students' learning.

One possible approach to this problem is to divide the large dataset into chunks and load the chunks into multiple Pandas DataFrames. The idea is to load smaller chunks of data into memory, one at the time, process or analyze it, store intermediate results to disk, and aggregate the results in the end. Dividing the data into chunks and coordinating the writing and re-reading of intermediate results to and from disk are tedious and error prone tasks. However, existing good quality, high-level libraries, such as Dask [20] or Ray [16] not only implement the tasks described above, but their APIs are very similar to the Pandas API. These libraries make it possible for students to quickly grasp these concepts and to write short scripts for processing large files.

For this assignment, we are using Dask, a lightweight framework that works well on a single machine by using all the cores to process larger-than-memory datasets. Dask also scales up resiliently and elastically on clusters with hundreds of nodes for solving even larger problems. Dask focuses on parallel analytics, providing Dask-specific modules to be used in place of Numpy Arrays or Pandas Dataframes to facilitate parallel execution. The dask.dataframe module implements a blocked parallel DataFrame object that mimics a large subset of the Pandas DataFrame. To perform any operation on a Dask DataFrame, many Pandas operations on the smaller Pandas DataFrames are executed.

Under the Anconda platform, Dask can be installed with a single conda install command or using the pip command.

B. *Assignment*

The goal of this assignment is to use Dask DataFrames to read, preprocess, aggregate and summarize a large given dataset that does not fit in the RAM of a commodity laptop.

The dataset we used for this assignment comes from the Stack Overflow website [2], one of the most popular "question and answer" sites. Over the years the website has slowly evolved into a large free repository of knowledge. Currently, the site receives around 8,000 questions per day, and includes over 16 million questions, 24 million answers and 66 million comments all available to download in a data dump collection. The total size of the dataset in compressed format is just below 45 GB, with the most important file (Posts.xml) containing mainly the all the questions and answers at around 14 GB.

For the first part of the assignment, students are asked to read the Posts.xml file, upload it in a Dask DataFrame and answer the following questions:

- How many rows and columns are in the dataset?
- How many questions are in the dataset?
- How many answers are in the dataset?
- What is the average length of the title and body for all the questions in the dataset?
- What day of the week has the most questions submitted on average?
- How many closed versus open questions are there?
- Find how many unanswered questions are in the dataset.

For the second part of the assignment, students are required to create a smaller subsample (100 MB) of the dataset and save it in CSV or Apache Parquet format. These files are going to be used in subsequent assignments as input for machine learning tasks such as classification and clustering.

Most of this assignment can be solved using Dask DataFrame operations such as: value counts, row-wise selections, group by aggregations and date time resampling. Debugging and profiling code that runs in parallel can be challenging, but Dask has a diagnostic visual dashboard, partially shown in Figure 7, that provides insights on performance and progress. This dashboard is very useful for the students. Subsets extracted from the Posts file of Stack Overflow have been used before [13] to build classification models for predicting closed questions. Another supervised problem is to predict a question's tags [21].

This challenging assignment introduces students to parallel and distributed computing in an easy, unintimidating way. Students should be familiar with Pandas DataFrames and Scikit-Learn by the time in the semester when they have to solve this assignment. The skills acquired while solving this assignment will be useful for solving other course assignments and other student's projects.

References

[1] M. Agung, M.A. Amrizal, K. Komatsu, R. Egawa, and H. Takizawa. A memory congestion-aware MPI process placement for modern NUMA systems. In *Proc. 2017 IEEE 24th Intern. Conf. High Performance Computing (HiPC)*, pages 152–161. IEEE, 2017.

[2] A. Ahmad. A survey on mining stack overflow: question and answering (Q&A) community. *Data Technologies and Applications*, 52(2):190–247, January 2018.

[3] E. Ayguadé, L. Alvarez, F. Banchelli, M. Burtscher, A. Gonzalez-Escribano, J. Gutierrez, D.A. Joiner, D. Kaeli, F. Previlon, E. Rodriguez-Gutiez, and D.P. Bunde. Peachy Parallel Assignments (EduHPC 2018). In *Proc. IEEE/ACM Workshop on Education for High-Performance Computing (EduHPC)*, Dallas (TX), USA, 2018. IEEE.

[4] G.E. Blelloch. Scans as primitive parallel operations. *IEEE Transactions on Computers*, 38(11):1526–1538, 1989.

[5] G.E. Blelloch. Prefix sums and their applications. Technical Report CMU-CS-90-190, School of Computer Science, Carnegie Mellon University, 1990.

[6] F. Crow. Summed-area tables for texture mapping. In *Proc. of SIGGRAPH*, pages 207–212, 1984.

[7] M. Diener, E.H.M. Cruz, M.A.Z. Alves, P.O.A. Navaux, and I. Koren. Affinity-based thread and data mapping in shared memory systems. *ACM Computing Surveys (CSUR)*, 49(4):64, 2017.

[8] D. Ellsworth. Syllabus for cp341: Parallel programming. http://cs.coloradocollege.edu/~dellsworth/2017b2/.

[9] J. Fresno, A. Ortega-Arranz, H. Ortega-Arranz, A. Gonzalez-Escribano, and D.R. Llanos. *Gamification-Based E-Learning Strategies for Computer Programming Education*, chapter 6. Applying Gamification in a Parallel Programming Course. IGI Global, 2017.

[10] F. Gaud, B. Lepers, J. Funston, M. Dashti, A. Fedorova, V. Quéma, R. Lachaize, and M. Roth. Challenges of memory management on modern numa systems. *CACM*, 58(12):59–66, 2015.

[11] A. Goncharow, A. Boekelheide, M. Mcquaigue, D. Burlinson, E. Saule, K. Subramanian, and J. Payton. Classifying pedagogical material to improve adoption of parallel and distributed computing topics. In *Proc. 9th NSF/TCPP workshop on parallel and distributed computing education (EduPar)*, 2019.

[12] OU Intel Parallel Computing Center (IPCC). http://ipcc.cs.uoregon.edu/curriculum.html.

[13] G.E. Lezina and A.M. Kuznetsov. Predict closed questions on Stack-Overflow.

[14] A.D. Malony. Lab 5: Map. https://classes.cs.uoregon.edu/14S/cis410parallel/lab5.php.

[15] M. McCool, J. Reinders, and A. Robison. *Structured Parallel Programming: Patterns for Efficient Computation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2012.

[16] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M.I. Jordan, et al. Ray: A distributed framework for emerging AI applications. In *Proc. 13th USENIX Symp. Operating Systems Design and Implementation (OSDI)*, pages 561–577, 2018.

[17] B. Norris. Map pattern (aka lab4). https://classes.cs.uoregon.edu/17S/cis431/lab4.php.

[18] O. Ozturk, B. Glick, J. Mache, and D.P. Bunde. Peachy Parallel Assignments (EduPar 2019). In *Proc. 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 342–346, Rio de Janeiro (Brasil), May 2019. IEEE.

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12(Oct):2825–2830, 2011.

[20] M. Rocklin. Dask: Parallel computation with blocked algorithms and task scheduling. In *Proc. 14th Python in Science Conf.*, number 130–136, 2015.

[21] C. Stanley and M.D. Byrne. Predicting tags for Stackoverflow posts. In *Proceedings of ICCM*, 2013.

[22] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. R. Scott, and N. Wilkins-Diehr. XSEDE: Accelerating Scientific Discovery. *Computing in Science & Engineering*, 16(5):62–74, Sept.-Oct. 2014.

[23] R. Vaidyanathan and J. L. Trahan. *Dynamic Reconfiguration: Architectures and Algorithms*. Kluwer Academic/Plenum Publishers, 2004.