Auto-Grading for Parallel Programs

Maha Aziz
Department of Computer Science, Rice University
6100 Main St
Houston, TX
mta2@rice.edu

Heng Chi
Department of Computer Science, Rice University
6100 Main St
Houston, TX
hc23@rice.edu

Anant Tibrewal
Department of Computer Science, Rice University
6100 Main St
Houston, TX
avt3@rice.edu

Max Grossman
Department of Computer Science, Rice University
6100 Main St
Houston, TX
jmg3@rice.edu

Vivek Sarkar
Department of Computer Science, Rice University
6100 Main St
Houston, TX
vsarkar@rice.edu

ABSTRACT

Fundamentals of Parallel Programming (COMP 322) is a required course for all Computer Science majors at Rice University. It introduces students to several basic concepts of parallel programming and parallel algorithms and follows a "pedagogic approach that exposes students to intellectual challenges in parallel software without enmeshing them in jargon and lower-level details of today's parallel systems". The material from COMP 322 has also been used in related courses at other universities including Harvey Mudd College and Brigham Young University.

Currently, programming assignments in this class are manually graded by teaching assistants (TAs) for correctness, performance, style, and documentation. Students receive limited feedback as they progress through the assignment because TAs grade their homework only after the submission of a final version. Auto-graders are a common solution to this problem: they allow students to receive feedback as they work on their assignments rather than only after they have completed them. This results in higher quality submissions by allowing students to learn from mistakes as they make them, rather than days or weeks later. It also prevents the development of bad habits or mislearned knowledge by addressing mistakes early.

Web-CAT is an advanced, customizable, and comprehensive automated grading system developed at Virginia Tech. It supports many models for program grading, assessment, and feedback generation. This paper describes our work on extending Web-CAT to address the requirements of Rice University's introductory parallel programming course, thereby creating infrastructure that can be used for similar courses at other universities and in online courses.

Keywords

Parallel programming, Auto-grading, Habanero Java, Web-CAT

1. BACKGROUND

1.1 Motivation

All undergraduate Computer Science students at Rice University are required to take the Fundamentals of Parallel Programming course (COMP 322) to introduce them to basic parallel programming concepts. Currently, programming assignments in this class are manually graded by teaching assistants (TAs) for correctness, performance, style, and documentation. This approach to grading becomes challenging for the TAs, in the face of the current rapid increase in the number of CS majors. At the same time, students receive limited feedback as they progress through the assignment because TAs grade their homework only after the submission of a final version.

Auto-graders are a common solution to this problem: they allow students to receive feedback as they work on their assignments rather than only after they have completed them. This results in higher quality submissions by allowing students to learn from mistakes as they make them, rather than days or weeks later. It also prevents the development of bad habits or mislearned knowledge by addressing mistakes early. Further, auto-grading can enable TAs to focus on providing important high-level subjective feedback on the submissions, while lower-level objective feedback on the correctness and performance of the submissions can be provided more promptly by the auto-grading system.

1.2 Overview of Habanero Java

The programming component of COMP 322 is carried out using homework and labs. In these, students gain practical knowledge of implementing parallel programs and algorithms using Habanero-Java (HJ). HJ is a parallel programming model developed at Rice University. HJ is based around four core concepts [12]:

```
// launches the code to be run by Habanero runtime
launchHabaneroApp(() -> {
  // waits for all nested asynchronous tasks before exiting
  finish(() -> {
    // chunks data to give to cores
    for (int chunk = 0; chunk < nchunks; chunk++) {
      final int mychunk = chunk;
      // creates an asynchronous task to process mychunk
      async(() -> {
        ...
      });
    }
  });
});
```

Figure 1: HJlib sample program

1. Lightweight dynamic task creation and termination using async, finish, future, forall, forasync, ateach constructs
2. Collective and point-to-point synchronization using phasers
3. Mutual exclusion and isolation
4. Locality control using the "place" abstraction.

The Habanero-Java library [2] (HJlib) is a library implementation of the HJ model in Java 8. HJlib is a parallel programming library that combines several parallel programming concepts. HJlib is utilized in the Fundamentals of Parallel Programming course at Rice University (COMP 322) to teach students about introductory parallel programming concepts [11]. Figure 1 shows an HJlib sample program. This program is designed to split a large input data into `nchunks` asynchronous tasks.

1.3 Current Grading Process

The current grading process in COMP 322 is defined by a detailed rubric distributed to all graders. Performance and correctness tests are run using scripts which submit batch jobs to a compute cluster. These scripts output files which are then automatically committed to a shared Subversion (SVN) repository. Graders check out the results from SVN and manually inspect each output file. They then assign the submission a grade based on the rubric, the output of the tests, and a student-provided report.

This process is inefficient, tedious, error-prone, and opaque to students. It relies on human graders to manually and consistently assign scores across many student submissions. Each grader may have different styles and opinions in their grading and scoring, causing assignments to not be graded uniformly. They may also make mistakes in their grading due to human errors, which could even include misreading the rubric. In addition, manual grading takes a significant amount of time, which 1) increases the latency between a homework being submitted and being returned with feedback, thus resulting in students repeating the same mistakes in labs and subsequent assignments, and 2) reduces the time the teaching staff can spend on mentoring students. In this work, we address these shortcomings by extending the Web-CAT auto-grading tool to support performance and correctness testing of parallel programs. Our approach promotes a multi-phased grading process where each phase focuses on individual facets of the assignment. This work enables:

- A more transparent grading process for the students
- A faster and simpler grading process with a tighter feedback loop
- Automated performance evaluation of student-submitted parallel programs
- Helpful feedback to students about their code
- Enablement of future offerings of COMP 322 modules as massive open online courses (MOOCs) on parallel programming, thanks to a scalable grading process

1.4 Course Requirements

In response to the problems currently present in the COMP 322 grading process, we have implemented several features in the auto-grader that address these issues. These features fulfill the goals outlined in Section 1.3 by automating the majority of the grading process, increasing feedback given to students, and allowing the course to potentially support larger class sizes in the future.

Correctness grading of HJlib programs. As a first pass, the auto-grader checks if the student submissions function correctly on a limited set of tests. These are quick correctness tests and are run with a single runtime thread so that they don't take up all the cores in the Web-CAT host. Relative to manual grading, performing correctness testing in Web-CAT increases the speed of the grading and reduces the number of errors in the grading. In addition, it ensures that the grading is always done in a consistent and defined process which increases grading transparency. This feature paves the way for a potential creation of a MOOC because the grading of assignments is significantly faster and more automated. With manual grading, a MOOC would be infeasible as graders would not be able to keep up with the number of submissions.

Performance grading of multi-threaded HJlib programs. This feature allows the auto-grader to evaluate the efficiency of the student submissions by measuring the performance and scalability of student submissions across multiple test inputs on a remote compute cluster. The benefits of this feature similar to those for the correctness grading feature. Manual grading of performance testing is difficult, so adding an automated step for grading assignments on performance tests that are executed on the same platform makes the entire grading process more consistent and less error-prone.

Disaster recovery mechanism. To prevent an accidental loss of student submissions or results, a backup feature has been added to our auto-grading process. Student submissions are backed up in a quick and reliable manner using

a standard version control tool. Students' original code is saved to the version control tool before grading, just in case Web-CAT crashes or another unexpected issue occurs, and the code and results are then saved after grading.

Static analysis tools. The use of static code analysis tools has also been incorporated in our auto-grader to provide students with additional feedback about their code. The kinds of feedback provided range from style suggestions to potential bugs in the code. Students can use this feedback to improve their code, knowledge, and possibly increase their grades.

Profiling. Performance profiling provides useful feedback to students, helping students optimize their parallel algorithm and implementation. In this work, we inform students where the hotspots of their program are to facilitate code optimization. This information would be difficult for students to obtain otherwise and can be a valuable resource for them, quickly pointing out the parts of their programs that have the most potential for optimization.

Leaderboard. The leaderboard offers a real-time, anonymized view of student submission performance across the entire class. Students no longer have to guess at their performance relative to others in the class, offering a more organic answer to the question "how fast should my code be?". The students can view the top scores from each performance test in a user-friendly webpage and see how well their code is doing in relation to their peers.

2. EXISTING AUTO-GRADING APPROACHES

Several auto-grading methods and programs exist already. In this section we summarize past work on auto-grading, including Mooshak [3], Marmoset [7], JavaBrat [8], Codewebs [5], and built-from-scratch auto-grading systems, as well as their limitations. In the next section, we describe the auto-grading system that we've developed (based on Web-CAT) to address many of these limitations.

2.1 Mooshak

Mooshak is an auto-grading program originally created to host programming contests online. It has been used in programming courses because it provides instant feedback for assignments. It has a simple configuration and requires Linux, Apache, and Tcl. Mooshak's UI is outdated and unattractive, thereby increasing friction to its adoption by students.

2.2 Marmoset

Marmoset was developed at the University of Maryland for use in programming courses. Students submit their projects along with unit tests which Marmoset uses to grade the code. Students can then view the results of their submission on a webpage. Students are given a limited number of "tokens" with which they can submit their programs, forcing them to look over their code carefully before submitting. Past experiences with Marmoset in our research group indicated that it was complex to configure, install, and maintain, in part due to lack of support.

2.3 JavaBrat

JavaBrat is an auto-grading system that allows students to practice Java and Scala. It builds on the existing Labrat Java grader and WebLabrat web interfaces. JavaBrat provides a more comprehensive and intuitive user interface than WebLabrat and allows users to view the entire problem set. It also allows users to easily add their own programming problems to the problem set through the web interface. Additionally, JavaBrat integrates with some online learning platforms, such as Moodle. However, JavaBrat has an outdated user interface and requires many complex configurations for homework, thereby making it too complicated to serve as a foundation for the autograding tool that we aim to build.

2.4 Codewebs

Codewebs is designed for massive open online courses (MOOCs) and emphasizes efficient grading and searching of student programs. It applies a method for indexing code using "code phrases", which reasons about the behavior of code within their specific contexts. Student submissions are treated as strings which are then parsed into abstract syntax trees, allowing Codewebs to analyze the code without having to actually run it. In a process known as probabilistic semantic equivalence, it links the semantic meaning of code with syntax to compare differing code so that it can determine what the code is doing. In doing this, Codewebs is also able to discern bugs in the code and provide feedback to the students. Codewebs has demonstrated its effectiveness by being successfully implemented in Stanford University's Machine Learning MOOC, which has over 120,000 students. However, the requirements for its grading method include the mechanism for parsing source code into an abstract syntax tree and the manipulation of the nodes that will represent statements, identifiers or constants. However, this level of investment is too complicated and expensive for us to undertake in developing an autograding tool for our parallel programming class.

2.5 Building a new auto-grading system

One option for our needs was, of course, to build a new auto-grading system from scratch. We decided not to pursue this option because of the added time and maintenance costs. Using an existing system allows us to bypass creating the core architecture of the system and much of the user interface. While the requirements of COMP 322 are somewhat unique and not satisfied by any existing solution out of the box, extending an existing auto-grader with sufficient flexibility allows us to construct a system that fulfills our requirements without building something completely new from scratch.

3. OVERVIEW OF Web-CAT

Web-CAT is an auto-grading program designed to serve as a shared infrastructure for computer science instructors and students to assign, submit, and grade programming assignments. It has several advantages:

- **Extensibility** Web-CAT's design and architecture emphasizes extensibility, pluggability and flexibility. It has an ANT-based automated build for Java projects and supports arbitrary package structures. It uses different plugins to test and score programs in different programming languages. Web-CAT generates concrete, directed, and timely feedback to help the student learn and improve.
- **Portability** Web-CAT's core code is written in Java and is stored as a WAR file. It runs on any standard J2EE servlet container. The Web-CAT user interface is accessed via the browser and contains all the necessary tools and information for students and instructors. This structure allows Web-CAT to be portable and easy to access.
- **Usability** Web-CAT is able to perform a wide variety of tasks. Students can submit assignments and view feedback through the web interface. Additionally, instructors can administer classes, assignments, and grading plugins. The intuitive user interface makes it easy for students, instructors, and administrators to use Web-CAT and to configure system behavior.

Web-CAT Plugins. While the core Web-CAT code is not trivially modifiable, the plugins that carry out the actual testing, grading, and scoring of student submissions are designed to be customizable. This allows instructors to tailor Web-CAT to the specific needs of their courses. Instructors can either create new plugins from scratch or modify existing plugins provided by Web-CAT. The Java plugin that is included with Web-CAT already includes several features, including "ANT-based compilation, JUnit processing of student-written tests, support for instructor-written reference tests, PMD and Checkstyle analysis, and Clover-based tracking of code coverage during student testing." [1] Plugin behavior is defined using a combination of Perl scripts and configuration files. Many features of Web-CAT plugins can be modified; for example, instructors can specify style errors they want Checkstyle to search for, change the hints that students receive for failed tests and errors in their code, and adjust the scoring scheme for assignments. In this work, we extend the existing Web-CAT Java plugin to fulfill the specific requirements of the introductory parallel programming course at Rice University, COMP 322.

4. HJlib PLUGIN FOR Web-CAT

Figure 2 shows our system diagram. The box labeled Web-CAT is the central Web-CAT installation that orchestrates the grading process. It is hosted on a Linux machine that also stores the assignment information, submissions, and results data. The box labeled Version Control represents an off-site repository that serves as a disaster recovery and content distribution mechanism. The box labeled Remote Cluster represents a large, remote computing center that accepts batch jobs. The Leaderboard is a separate Linux web server that tracks the results of all student submissions and displays the anonymized results to students in the class. This allows students to compare their own results with their peers and understand where they stand in relation to their classmates.

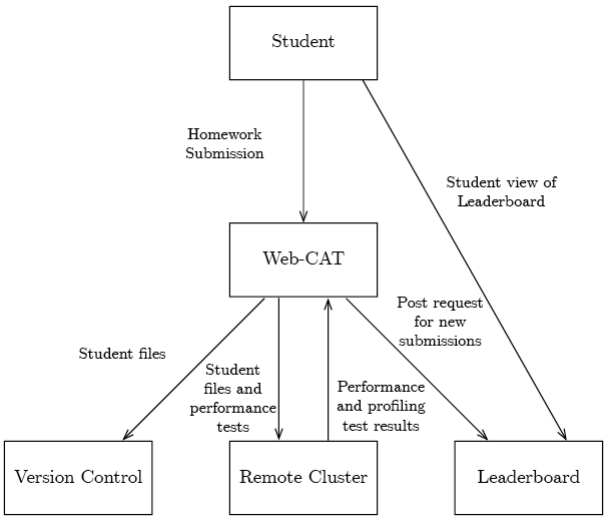

Figure 2: Architecture of our Web-CAT extension.

The following is the typical auto-grading process:

1. Students submit a zip file through the Web-CAT UI. Our plugin commits their submission to the SVN repository, which acts as a backup for student submissions in the case of hardware failure on the Web-CAT host. This also prevents TAs or students from having to manually upload each submission to SVN.
2. Web-CAT transfers all student-provided source code and instructor-provided tests to a remote cluster and uses Maven to compile the code there. The reason we choose Maven over ant is because the existing COMP 322 infrastructure makes use of Maven and so we aim to support the existing assignment structure with minimal modifications to the assignments themselves. However, our tool can support the use of Ant as well, in case that is preferred by other institutions that may want to use this infrastructure.
3. After compiling the student code, our plugin submits a batch job to the remote cluster that runs all performance tests on the student submission and outputs the results to a text file. Additional runs of each performance test are executed using the Lightweight Java Profiler [4] to identify hotspots in the program. The results of the profiler are saved in text files as well. Once the batch job completes, our plugin transfers all output files back from the remote cluster to the Web-CAT host and adds them to SVN.
4. Our plugin then runs a series of correctness and static analysis tests on the student submission from within the Web-CAT host machine. First, lightweight correctness tests provided by the instructor are run with a single core. Next, our plugin runs the FindBugs [6] program on the compiled student code. The Checkstyle [13] tool is also run on the code, using configuration and scoring files to determine what style errors to look for and how many points to deduct for each specific error.

5. After the grading process is completed, Web-CAT sends an HTTP POST to the Leaderboard, informing the Leaderboard that there has been a new submission. The Leaderboard fetches the submission results from SVN and saves them in its database. The Leaderboard is accessible via a web browser, showing the scores for each performance test. All data on the webpage is kept anonymous so students can see how their submission compares to their peers without knowing who received the scores.

6. The results of these performance, correctness, and static analysis tests are aggregated together and formatted as HTML for display to the user.

The following sections will go into further detail on the implementation of each of the components in Figure 2.

4.1 Correctness Grading of HJlib Programs

Web-CAT comes with a plugin for evaluating submissions in the Java programming language. This plugin uses JUnit testing methods to evaluate whether the submission functions correctly and delivers the correct outputs for the inputs. However, the plugin cannot immediately grade COMP 322 assignments out of the box. Therefore, the plugin was modified to be better suited for the requirements of COMP 322, which uses HJlib to teach parallel programming. The Java plugin was modified to be compatible with HJlib by adding the required classpath and Java agents for the library. To ensure that these tests don't take over all the cores in the Web-CAT machine, these correctness tests are run with a single runtime thread. The HJlib cooperative runtime enables running single-threaded versions of programs irrespective of the synchronization constraints used. Being able to run single-threaded versions allows using standard off-the-shelf machines (including local machines) for this phase of grading.

4.2 SVN Backup

To prepare for the (rare) possibility of data corruption in Web-CAT, it is necessary for us to backup all students' submissions before the grading process begins. For this, we have added an SVN plugin to Web-CAT. Our SVN plugin uploads students' submissions to the SVN repository before Web-CAT begins the grading process. This ensures persistent backup of student submissions in case of data loss on the Web-CAT host. Backups are also taken of test outputs. Additionally, because the Leaderboard uses SVN to get the student results, SVN acts as a communication channel or document store between Web-CAT and the Leaderboard, as well as between Web-CAT and the course graders.

4.3 Static Analysis

Web-CAT includes the Checkstyle tool, which is useful for improving code readability and structure. However, it does not help students with debugging and ensuring correctness. When writing large programs it is often time consuming and difficult to identify and locate small bugs in the code. One can easily fail to spot these mistakes due to the immense amount of code and the complexity of the program. This can cause students to lose many points in their grade because of minor errors. To assist students and reduce the impact of minor bugs, our plugin was also extended to support the FindBugs program. FindBugs, developed at the University of Maryland, is a static analysis tool that detects possible bugs in Java programs. Similar to Checkstyle, FindBugs outputs a bug location along with a brief description of the bug. To incorporate it into Web-CAT, it is installed onto the server in which Web-CAT runs and a command to run it is added to the Perl script that is responsible for running the testing of student code. The results are then collected and displayed to the user in the results page. By adding FindBugs to Web-CAT, students automatically receive information to help debug their programs in a convenient location.

4.4 Performance Grading

In COMP 322, an important aspect of parallel program grading is performance. Programs must use parallelism effectively in order to use resources properly and be efficient. In order to have a clear view of students' performance of their code, we have added performance grading process to the plugin. There students' code is graded by measuring the speedup of the program as the number of cores the program is given access to is increased. Multiple tests are run on the program, and the program is given a different number of cores in each test. The time it takes to run each test and the number of cores given are recorded. Looking at these results, one can determine if the program is using the cores efficiently and effectively using parallel programming concepts.

The steps to achieve remote performance grading of student submissions are as follows:

1. Build folder structure on the remote cluster
2. Send student code to the remote cluster
3. Send all performance tests given by the instructor to the remote cluster
4. Copy performance tests to the designated location according to the created folder structure
5. Use Maven to compile Java code
6. Send packaged jar file back to original server
7. Send Java execution script to the remote cluster; the Java execution script detects and executes all Java test classes
8. A batch job is submitted to the cluster which calls the Java execution script.
9. Performance and profiling results generated on the cluster are transferred back to the Web-CAT host
10. The results of the performance and profiling tests are committed to SVN

One of the most common issues that auto-graders have to deal with is long-running student submissions hogging grading resources. To ensure the fairness of grading resource allocation, there are several timeouts in Web-CAT and our plugin:

Tables

EXECUTION_TIME Performance

Show 10 entries

test4					
TEST NAME	EXECUTION TIME (milliseconds)	SUBMISSION #	CORES	Customer #	SUBMISSION TIME
test4	6344	46	1	143	2015-09-07 20:42:40
test4	3470	46	2	143	2015-09-07 20:42:40
test4	900	46	4	143	2015-09-07 20:42:40
test4	307	46	8	143	2015-09-07 20:42:40
test4	9992	4	1	004	2015-09-07 20:42:53
test4	7120	4	2	004	2015-09-07 20:42:53
test4	740	4	4	004	2015-09-07 20:42:53
test4	233	4	8	004	2015-09-07 20:42:53

Figure 3: Sample Leaderboard table of performance test execution time with varying cores

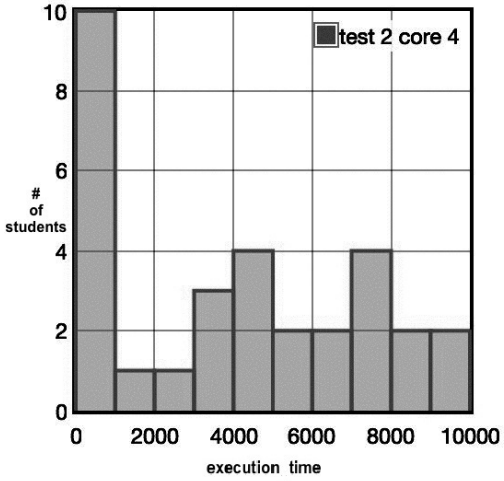

Figure 4: Sample Leaderboard histogram of performance test execution time with one core

- Web-CAT total grading timeout: This is a configurable Web-CAT timeout for all actions taken as part of grading a single student submission. When this timeout occurs, an internal error is shown in the Web-CAT UI.

- Batch job timeout: This is a time limit specified by the batch job configuration, limiting how long the performance tests may take on the remote cluster. When this timeout occurs, a timeout error is shown in the Web-CAT UI.

4.5 Leaderboard

In COMP 322, students are often concerned about how well they are doing compared to their peers. For example, a student may wish to know what the average speedup is to see whether they need to optimize their code further. To address these concerns, we have created the Leaderboard so students can view the results that the entire class is receiving for an assignment.

After the grading process finishes, our Web-CAT plugin notifies the Leaderboard that a new submission has completed by transmitting the student ID and submission ID to it. The Leaderboard updates an internal database with the results and displays them in an HTML table on the Leaderboard webpage. All scores data goes to the Postgres [10] database and the database is connected to the webpage using Flask [9] and Psycopg2 [14]. We have created a table on the webpage containing the unique ID, test name, execution time, number of cores used, and submission date and time so students can see how their scores compare with others anonymously. Figure 3 shows the table of ranked submissions in the Leaderboard webpage. Each student receives a unique ID so that they can find their scores on the Leaderboard. The Leaderboard displays the test name, the execution time of the submission, the submission number, the core count, and the time of the submission. We also have included graphs and additional statistics (mean, median, etc.) for each test so students can utilize this information when analyzing their performance. Figure 4 displays the bar graphs and their data. It displays the execution time distribution among all students. The y-axis shows how many students are in a certain range on the x-axis. For example, for test two running on four cores, there are ten students who have run times from 0 to 1000 milliseconds.

4.6 Profiling

In COMP 322, program efficiency and optimization is an important factor in grading students' assignments. While performance tests tell students how well their code is performing, they don't explain why. For this, the Lightweight Java Profiler [4] was incorporated into our plugin to allow students to see which methods in their programs took the most time. As the program is running, this profiler repeat-

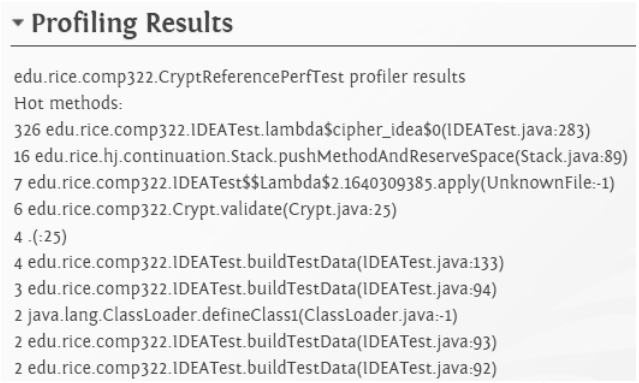

Figure 5: Sample profiler output in the Web-CAT UI

edly samples stack traces of all threads at fixed intervals. It compiles a list of the "hot methods" where the program spends the most time. In our evaluation, the Lightweight Java Profiler demonstrated minimal overhead. The profiler is run on all performance tests on the same dedicated compute node. The results are saved and parsed for the relevant data and then displayed to the user in the results page in Web-CAT. Students can then view the "hot methods" and take steps to optimize their code.

Figure 5 shows an example of the student profiling view. The first line specifies the test that was run. Each line from the third line onward lists the number of times a sample was taken of the program stack that included a given method. The profile in Figure 5 informs the student that most of the time was spent in the lambda in line 283 of the IDEATest.java file.

5. FUTURE WORK

While the system implemented in this work meets the requirements of COMP 322, there are many possible extensions to it. First, we would like to include more features in the Leadboard (e.g. statistics on how many students are passing a given test) to give feedback on the relative difficulty of different tests. We might improve the security of the Leaderboard by implementing student ID authentication so only students in the class can view the performance test results, even though they are anonymized. We also want to add a scalability property to the Leaderboard to show the scalability of each student submission on each test, as a function of CPU cores used. Second, we also plan to write documentation on Web-CAT for students and instructors to use. Furthermore, we also are interested in establishing a peer review system in COMP 322, possibly through Web-CAT. Finally, we are very interested in adding new debugging tools for data race detection, determinism checking, and deadlock detection.

6. CONCLUSIONS

Through Web-CAT and our custom HJlib plugin, we have created a system that efficiently and transparently automates the grading of student assignments in the Fundamentals of Parallel Programming course at Rice University. We have extended Web-CAT to be compatible with the Habanero Java library, allowing it to grade parallel programs for correctness. Additionally, our plugin grades the submissions' performance by sending them to a remote compute cluster and testing them there. This process eliminates manual grading completely, creating a tight, fast feedback loop for students. Students get detailed information on their grades and results which increases the level of transparency in the grading. This system also helps the course's teaching staff, reducing their grading burden and thereby increasing the time that they can spend on teaching.

Our plugin implements a number of features that had not been present previously in the course to help students improve their programs. Students automatically receive Checkstyle and FindBugs static code analysis reports to assist them with their code styling and help them spot potential bugs in their code. When the submissions are sent to the remote cluster for performance grading, they are also run using the Lightweight Java Profiler that informs students of where to optimize and parallelize their code. Finally, the Leaderboard offers students valuable insight into how the rest of the class submissions are performing, further improving transparency. With the necessary modifications, we believe that this auto-grading system can be an asset to any parallel programming course, including related courses at Harvey Mudd College and Brigham Young University.

7. ACKNOWLEDGMENTS

This work was supported by the Habanero Extreme Scale Research group at Rice University and the Department of Computer Science. It was also supported in part by the Cyberinfrastructure for Computational Research funded by NSF under Grant CNS-0821727 and Rice University.

8. REFERENCES

[1] S. H. Edwards. Web-cat wiki. `http://wiki.web-cat.org/WCWiki/JavaTddPlugin`.

[2] S. Imam and V. Sarkar. Habanero-java library: a java 8 framework for multicore programming. In *Proceedings of the 2014 International Conference on Principles and Practices of Programming on the Java Platform Virtual Machines, Languages, and Tools*, pages 75–86. ACM DL, 2014.

[3] J. P. Leal. Mooshak. `https://mooshak.dcc.fc.up.pt/`.

[4] J. Manson. Lightweight java profiiler. `https://code.google.com/p/lightweight-java-profiler/`.

[5] A. Nguyen, C. Piech, J. Huang, and L. Guibas. Codewebs: Scalable homework search for massive open online programming courses. In *Proceedings of the 23rd international conference on World Wide Web*, pages 491–502. ACM DL, 2014.

[6] U. of Maryland. Findbugs - find bugs in java programs. `http://findbugs.sourceforge.net/`.

[7] U. of Maryland. The marmoset project. `http://marmoset.cs.umd.edu/index.shtml`.

[8] A. Patil. Automatic grading of programming assignments. Master's thesis, San Jose State University, 5 2010.

[9] Pocoo. Flask (a python microframework). `http://flask.pocoo.org/`.

[10] PostgreSQL Global Development Group. PostgreSQL. `http://www.postgresql.org`, 2008.

[11] V. Sarkar. Comp 322: Fundamentals of parallel

programming wiki. https://wiki.rice.edu/confluence/display/PARPROG/COMP322/.

[12] V. Sarkar. Habanero-java wiki. https://wiki.rice.edu/confluence/display/HABANERO/Habanero-Java.

[13] C. D. Team. Checkstyle. http://checkstyle.sourceforge.net/.

[14] D. Varrazzo. Psycopg. http://initd.org/psycopg/.