CODE ISOLATION FOR ACCURATE PERFORMANCE SCORING USING RASPBERRY PIS *

Bryan Dixon
Computer Science Department
California State University Chico
Chico, CA 95929-0410
530-898-6442
bcdixon@csuchico.edu

ABSTRACT

A problem encountered when teaching code optimization is providing a way to consistently and quickly grade students' code on an appropriate hardware platform. The primary issue is generating a consistent measurement of performance from machine to machine as well as on machines with different loads running on them. The project discussed in this paper tries to solve this problem utilizing a distributed system of inexpensive hardware.

INTRODUCTION

Performance-based projects help students understand the relationship between hardware and software, and teach them valuable practical skills by allowing them to exploit certain hardware features to make code run faster. For example, students are given an un-optimized image-filtering program and asked to optimize it. The program uses embedded assembly to get the current clock cycles on the system, using the difference in clock cycles from the start of filtering to the end to generate the overall number of clock cycles it takes to run the program, while also generating a clock cycles per pixel result at the end. The code typically takes about 5000-6000 clock cycles per pixel initially, and students are tasked with using code optimization techniques learned in class to get the code running time below 300 clock cycles per pixel. Code that is significantly lower than the target clock cycles earns the students extra credit. Some students have taken this as a challenge and gotten the code down into the teens for their performance.

Grading this project presents two main challenges. The first is that code performance will vary depending upon the machine it is run on, as it relies heavily on students writing code optimizations that leverage specific hardware features such as pipelines, out of order operations, and multiple ALU arithmetic units. This can be solved by hosting a server specifically for grading all of these projects rather than relying on individual computers. However, if multiple students attempt to use this server for grading at the same time, their code competes for resources and produces wildly inconsistent performance numbers. Even when taking the median of numerous runs across a multitude of filters, these differences can be so great that students' scores will vary run to run, even if they don't make any changes to the code.

The system discussed in the remainder of this paper enables economical and fair testing and grading of code optimization assignments by providing an environment that isolates each code run, enabling students to get a consistent result each time.

PROJECT HARDWARE

The clock cycle estimates given earlier are for the default and goal runtimes on the platform chosen for the project. If a modern Core i7 were used, there would be an initial runtime closer to 3000 clock cycles per pixel. Modern processors feature large caches, extremely deep pipelines, and the ability to compensate for branches, all of which allow poorly-optimized code to run faster. This is an issue in teaching code optimization projects. A system is needed that is capable of providing some modern features, but in a limited fashion, to force students to really optimize their code to get good performance out of the project. The Raspberry Pi 2 computer was chosen as the system students would use to optimize their projects' runtimes for this reason[1].

The Raspberry Pi 2 has a small 512KB cache and has a quad core Cortex-A7 ARMv7 processor. This means that even a small picture can easily overflow the cache, which forces students to write cache-optimized code. In addition to optimizing for hardware, the Raspberry Pi 2 has multiple cores, so students can leverage multi-threading as another mechanism to increase the performance of their code.

A final benefit of the Raspberry Pi 2 is the cost, as it costs about $40 per computer including an SD Card with the OS. Provided with a $1600 budget, this project could be built out of 24 Raspberry Pi 2s, including additional devices to provide power, for less than the provided budget. The rest of the system was pieced together with outdated equipment that was previously going to be disposed of by campus IT. An old server was used to host a website for students to interact with, as well as a DHCP server to connect the Raspberry Pi 2s via an old 48-port network switch.

PROJECT

To enable resource isolation, a batch system was designed to queue and run each student's code, which could be uploaded by the student through a web interface. This system required the following technologies: an asynchronous task manager consisting of an atomic mechanism to keep track of the number of available Raspberry Pi 2s, a database to store the state of the Raspberry Pi 2s and where they live on the network, a

front end script to provide responsiveness and updates to the user, and a web framework to tie it all together.

The system was built upon the Django Python based web framework [2].This web framework makes it easy to get an interactive web server backend up and running, and provides an object relational model-based interaction with the database backend. PostgreSQL was used as the database backend on the Django web server [3]. It was chosen because it integrates nicely and provides all features needed for this project.

Redis was utilized to store an atomic count of available servers [4]. To be able to run students' code while at the same time allowing the web server to continue to respond to web requests, an asynchronous task server was needed. To this end the Celery distributed task queue was utilized [5]. Celery allows a task to run asynchronously from the Django web server, but also provides a mechanism for Django to inquire on the status of a task so that information can be given back to the student. Celery was also used to enable real-time feedback on the progress of a given task. This is important, as an un-optimized version of the performance lab tends to take over six minutes to execute all of the tests, and if the student doesn't get feedback they might feel that the website has frozen or failed to run their project and try to run it again.

Facebook's React Javascript library for building user interfaces was leveraged to allow the client to get status updates [6]. React was chosen as it simplified the implementation complexity over using libraries like JQuery or Prototype [7,8]. This allowed for quick implementation of the front-end requirement that polls for updates and communicates to the students their progress throughout the tests. It also allowed for more detailed feedback to be given easily.

The last crucial bit of implementation was how to tie all these technologies together on a single system. Docker was used to provide isolation and simplify deployment [9]. Docker allows the virtual containers to be set up and easily moved to the production machine once the test system is working. Docker also provides some code isolation to keep the database, web, and other resources on different virtual systems but provide easy links between them. Five different Docker containers were leveraged in the production and test versions of this project. This allowed the project to have the webserver, web backend with task server, database, and two Redis servers on each individual container.

IMPLEMENTATION ISSUES

The most difficult part to get working was clock cycle counters on the ARM processors used by the Raspberry Pi 2s. The ARM's performance clock cycle counters are not enabled for user code to access by default, and are only accessible via privileged code. To enable the performance clock cycle counters for use by user code, an assembly instruction has to be executed in privileged code. A kernel module was written for this purpose; however, once written it was discovered that this instruction only enables the clock cycle counters on the single processor core that it is running on, out of the four cores on the ARM processor. The kernel module was then rewritten to use four kernel threads, each with core affinity to affix the assembly instruction to one of the four cores on the quad core ARM processor of the Raspberry Pi 2.

There were also issues with getting the Raspberry Pi 2 to recognize this kernel module, but after following a multitude of instructions on compiling the Raspberry Pi 2 kernel from scratch, and compiling kernel modules for the Raspberry Pi 2, the issue was resolved. This was likely because simple methods for compiling the kernel leave out configuring the kernel features, and allowing loadable kernel modules was disabled by default. Once the kernel was configured to allow loadable kernel modules, the module could be loaded successfully.

This was the biggest issue with the implementation, and for anyone wanting to leverage clock cycle counters as a method of measuring performance, this issue has to be overcome to run code performance optimization assignments on a Raspberry Pi 2 or other ARM-based test system.

STUDENT OUTCOMES

A crucial component of student projects is that they allow the student to apply classroom teachings to a practical problem. The project that this platform was built for is a modified version of one originally written by Dirk Grunwald at the University of Colorado at Boulder, to replace the performance assignment from the Carnegie Mellon CSAPP course of materials [10,11,12]. The project forces students to take a working un-optimized image filter and apply code optimizations taught in lecture, while teaching the Carnegie Mellon material and expanding on it.

Optimizing the image filtering code teaches students how the order of their loops impacts the performance of code by making better use of the cache. Students also learn the benefit of not calling functions that return the same value inside of loops or as values in the loop declaration. Students also apply techniques to improve performance in the way a modern processor will pipeline instructions and use multiple ALU units on a processor to allow for code parallelism. This includes using temporary accumulators, multiple accumulators, etc. In addition to this, students can apply techniques learned in an earlier project, which had them manipulate binary in C, and allowed them to accomplish division without using the division operator, therefore saving clock cycles. Finally, students can leverage the fact that the Raspberry Pi 2 has multiple cores to further increase code parallelism by making their code multi-threaded either explicitly or through implicit threading libraries such as OpenMP that are supported on the Raspberry Pi 2 platform [13].

Additionally, in semesters where this project was used, students were assessed through through exam questions that required them to apply the knowledge learned from this project to answer the question. Students typically did exceptionally well on these questions, with the majority of students getting these questions completely right. Students were also asked for feedback on the project and this submission tool in the semesters when it has been utilized. Students have really enjoyed this project, some even responding that it was the most applicable project they had worked on while in school, and it made them re-think how they had been writing code up to this point.

FUTURE DEVELOPMENT

The first three projects from the CMU CSAPP set of materials involve a real-time scoreboard with anonymized student handles that allow students to see how they are doing in comparison to the rest of the class. Motivated students can also see it as a competition or race to complete the phases of the project first. It would be relatively easy to add a scoreboard feature onto this project. This will allow students to compete for the lowest number of cycles per pixel. This is something many students already do, but now their progress would be visible to the whole course.

Based on the feedback and observed student utilization of the system, future development is needed to make the project more resilient to student submissions. It has been observed that individual students may manage to submit numerous times, resulting in no servers for any other students to use. Additionally, there are no mechanisms currently on the platform to catch or stop infinite loops or other flaws in submitted code. It would also be beneficial to have a web portal to see the state of all the Raspberry Pi 2 systems to enable administrators to see more information about which ones are tasked with a job, which ones are online and available, etc.

Another potential avenue of expansion on this project is to further expand both the number of Raspberry Pi 2s available as well as expanding the task server functionality to allow the system to act as a batch job tool for another course project. A faculty member has a project that requires the use of Open MPI and currently leverages instances of Open MPI running on lab computers for students to run their projects [14]. This method suffers from the same issues as the project described in this paper, where code isolation and task queues are important in the learning process for students as they will then get a more accurate performance score.

CONCLUSIONS

Code isolation through a web-based code runner tool can be achieved on a small budget utilizing Raspberry Pi 2s, enabling students to learn practical code optimization skills with code benchmarked on a consistent standard.

REFERENCES

[1] Element14: The new raspberry pi 2 model b 1gb, http://www.element14.com/community/community/raspberry-pi , retrieved November 3, 2015.

[2] Django: The web framework for perfectionists with deadlines, https://www.djangoproject.com , retrieved November 3, 2015.

[3] PostgreSQL: The world's most advanced open source database, http://www.postgresql.org/ ,?retrieved November 3, 2015.

[4] Redis, http://redis.io/ , retrieved November 3, 2015.

[5] Celery: Distributed task queue, http://www.celeryproject.org , retrieved November 3, 2015.

[6] React: A javascript library for building user interfaces, http://facebook.github.io/react/ , retrieved November 3, 2015.

[7] JQuery: Write Less Do More, https://jquery.com/ , retrieved November 3, 2015.

[8] Prototype Javascript Framework, http://prototypejs.org/ , retrieved November 3, 2015.

[9] Docker - build, ship, and run any app, anywhere, http://www.docker.com , retrieved November 3, 2015.

[10] Dirk Grunwald, http://systems.cs.colorado.edu/people/faculty/dirk-grunwald/ , retrieved November 3, 2015.

[11] CS:APP2e, Bryant and O'Hallaron, http://csapp.cs.cmu.edu/2e/home.html , retrieved November 3, 2015.

[12] Bryant, R., O'Hallaron, D., *Computer systems: a programmer's perspective, 2nd Edition, volume 2*, Lebanon, Indiana: Addison-Wesley, 2010.

[13] OpenMP, http://openmp.org/wp/ , retrieved November 3, 2015.

[14] Open MPI: Open Source High Performance Computing, http://www.open-mpi.org/ , retrieved November 3, 2015.