Brandon Poblette

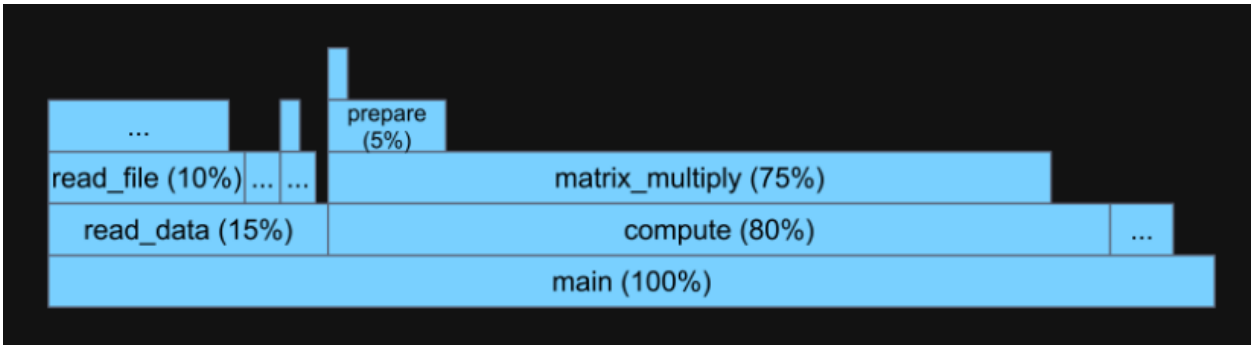
CPSC326 Section 2

5/11/2023

For my MyPl extension, I decided to implement a profiler. The job of a performance profiler is to figure out where your code is spending its time. This information can help developers resolve performance issues and optimize their applications. By figuring out where the program is spending most of its time, we can figure out what could probably be optimized. An example output would look something like the following snippet.

```
70% main(); compute(); matrix_multiply()
10% main(); read_data(); read_file()
 5% main(); compute(); matrix_multiply(); prepare()
...
```

In this example, we can gather that the code spent 70% of the stack traces were in the body of the matrix_multiply() function. In this example, they also showed the path of functions that called matrix_multiply. Main called compute, which then called matrix_multiply. This can be useful for letting the user know which parts of the code could be optimized. However, this output has it's drawbacks. For example, we can't see how much time compute() is using. To fix this we could change the output into a flamegraph as seen below.



In this output, we now get to see a much more accurate visualization of how much time is spent on certain functions. However, implementing a flamegraph has a very high degree of difficulty so don't expect anything like this. My plan for this project was to implement specifically a sampling-based profiler. The basic idea behind sampling-based profilers is to periodically interrupt the program and record the current stack trace. To implement this, I largely worked in upgrading the VM part of the MyPL pipeline. I added instrumentation so that there would be a clock running. I also had to create a new map object to hold the function names and how many traces they appeared in while the program was running. Every 1 microsecond, I would interrupt the program to get a trace of which function the program was in. Then depending on which function it was in; I would increase the trace count in the map. Because each function had its own frame, simply by checking frame->info.function_name I was able to figure out which function the program was in at any time.

In this project, I was able to output something like the first figure. If I had more time, I think I would have been able to figure out the complexity of a flame graph but unfortunately that was a titan too tall to tackle on this adventure. I also added an option to see what percentage of instructions belonged to each function because I thought this would be interesting information to add. More info is good info. Then I also added a trace flag that prints out the actual instructions being executed. For example, if there is an if else statement, it will only print out the instructions that are actually being executed. So either the if or the else instructions depending on the conditions being passed.

For testing, I mostly tested this out by running different programs. Unit tests are really difficult because I couldn't figure out what the outputs should be. For testing, I had several different programs that would require different functions to do different amounts of the work. For example, in a fibonacci program, or any program that uses recursion, it would make sense that most of the traces would be in the Fibonacci function. If most of it ended up in the main function, then there must have been an issue on how the traces were being made. Then there was a program I wrote in which there were functions that weren't called in main. That program would pass my "test" if that function didn't produce any stack traces.

Running the program is very simple.

./mypl --per_prof file-name.mypl uses the clock based profiler

./mypl –sprof file-name.mypl shows which percentage of instructions are made for each function

./mypl –trace file-name.mypl prints out the trace of the instructions being run

For any other questions, I updated the –help flag so that it will print out the possible flags.

Also here is the link to my youtube video!

https://www.youtube.com/watch?v=9vsJmXBDUeI