Abstract:

In this paper, I discuss the objectives, work performed, and results of my investigation into the memory stack traces of the SPEC2017 test suites. I worked closely with the Intel Pin Tool and SPEC to automate processes required to generate memory stack traces for each of the SPEC2017 test suites. I also wrote a python script capable of processing these memory stack traces into memory reuse distances, and then compiling these memory reuse distances into different statistics and graphs.

Work Description:

To begin this project, I started by obtaining memory traces of any program to get the formatting for processing them. A common theme I found for this project was depending heavily on the tech department, as I would often need various permissions to install programs or modify settings to allow them to run on Pitt's Linux Cluster. After getting access to the Pin Tool, I was able to generate memory traces on some relatively simple programs, which allowed me to begin work on my memory trace processing program.

At first, I was focused only on getting the individual memory reuse distances, meaning the number of unique memory page accesses between each access of the same memory page. This value is important, as it is very relevant to data caching, as larger memory reuse distances for a given page makes it more likely for it to be evicted from memory and later reloaded, which is an extremely slow operation. To generate these memory access values, I needed to use an efficient approach to iterate through the entire list of memory accesses, while also knowing how many unique pages have been accessed since the last time any given page was accessed. A simple brute force approach would be able to achieve this in O(n^2) by iterating backwards through the list at each memory access to find the last page access, counting all the unique pages

on the way of the backtrack, but this could be problematically slow, especially given that the memory trace files can easily have millions of different memory accesses. To resolve this, I kept a dictionary for each page in memory, which then held a unique page list that would store the unique pages seen since the last access of the initial page. Once a page is first encountered, it is given its own base dictionary entry, which is initially empty. Then, that page is added to all of the existing unique page lists for every other page, if it was not already there. Then, when that same page is encountered another time, we simply count the length of its unique page list, and then reset it, as this length value represents all of the unique memory page accesses that occurred between the current page access and the previous time it was accessed. This method allows for a runtime of O(nm), with space requirements of O(m^2), where m is the total number of memory pages present in the memory access trace file. My program then saved these memory reuse distances in order to be used later for generating different statistics and graphs.

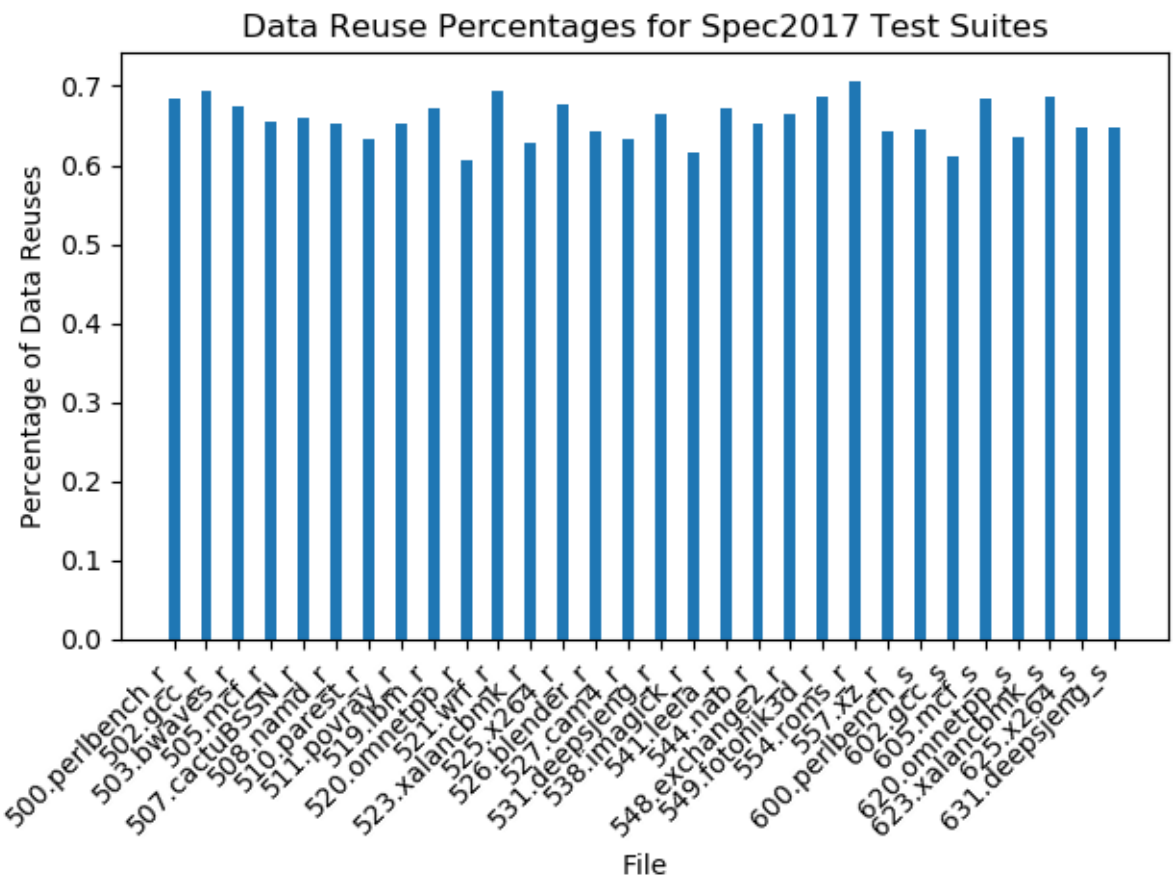
After I had my memory reuse distances generated, I next needed to get the memory traces specifically for the SPEC2017 test suites. This part of the project brought another batch of issues on which I had to turn to tech for help with. For starters, I had found that the SPEC2017 ISO that was offered on the Linux cluster was incomplete, which after alerting tech to, they were able to restore a complete copy for use. Next, I ran into issues with getting the SPEC tools required to run any of the test suites to build correctly. I attempted to build these tools manually, and after digging through a variety of debug files, came across some error messages regarding failing file operations that were being tested while the tools were being built. After mentioning these errors to the tech department, they suggested that it could be an issue with the afs directory in which I was attempting to build the tools. I moved outside of this directory, which then resolved that specific issue, and allowed the tools to build correctly. From here, I had to do some

experimentation with the configuration settings being used to build the test suites themselves. I was experiencing several build errors, which I eventually traced down to a specific flag that was being set in the configuration settings by default. After removing this flag, I was finally able to build and run the test suites.

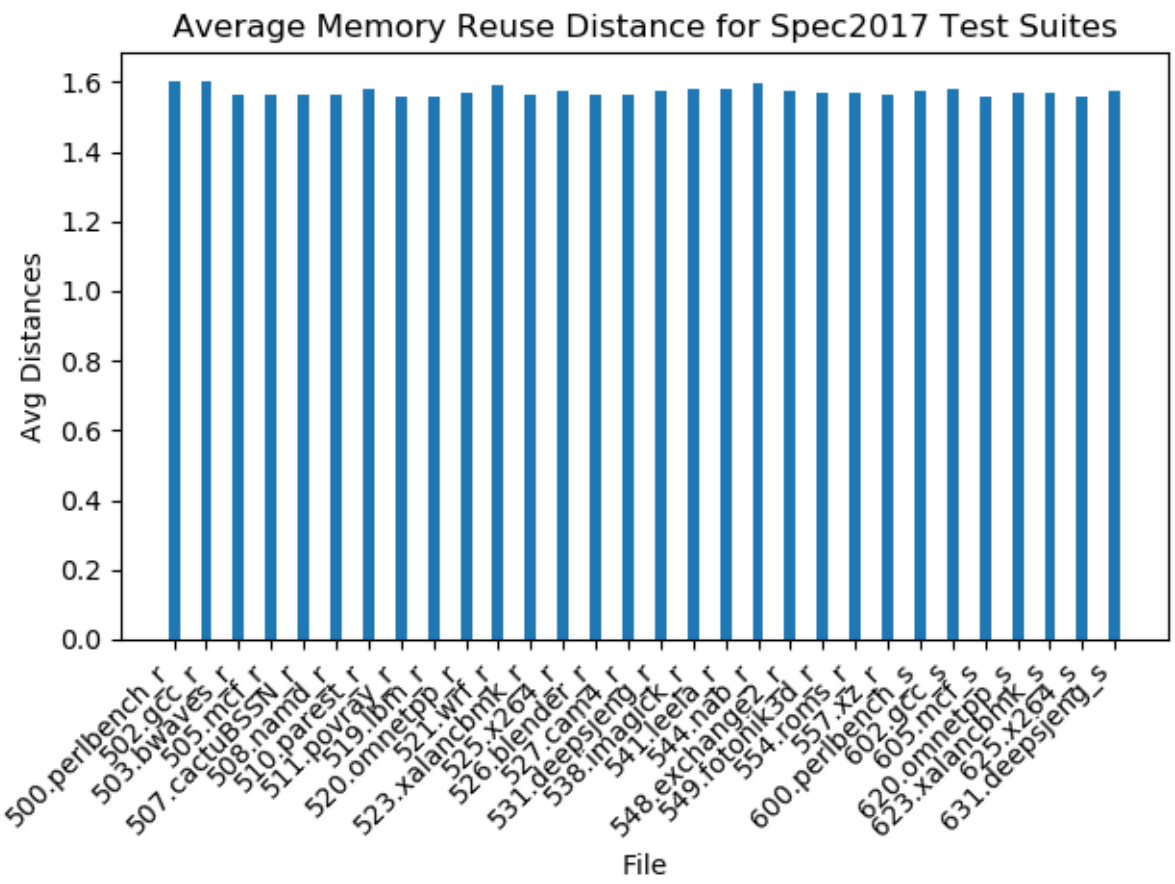
From here, it was relatively simple to use the Pin tool to generate memory traces for each of the test suites. As mentioned before, I couldn't use the afs directory that offers permanent storage, so I wrote a script that automates the process of building, running, and transferring the memory trace files called runpin.sh. This was very helpful since each suite took about 10 minutes to build and run, and having to repeat that process for all 40+ test suites manually would be very tedious.

Now that I had all of the memory traces, it was just a matter of using my previously created memory reuse distance script to get all of the memory reuse distances for each of the files. Once this was complete, I computed the percentage of data reuses and average reuse distances for all test suites, as well as an individual breakdown of the reuse distances for each file, all of which would be displayed as part of a graph generated using matplotlib.

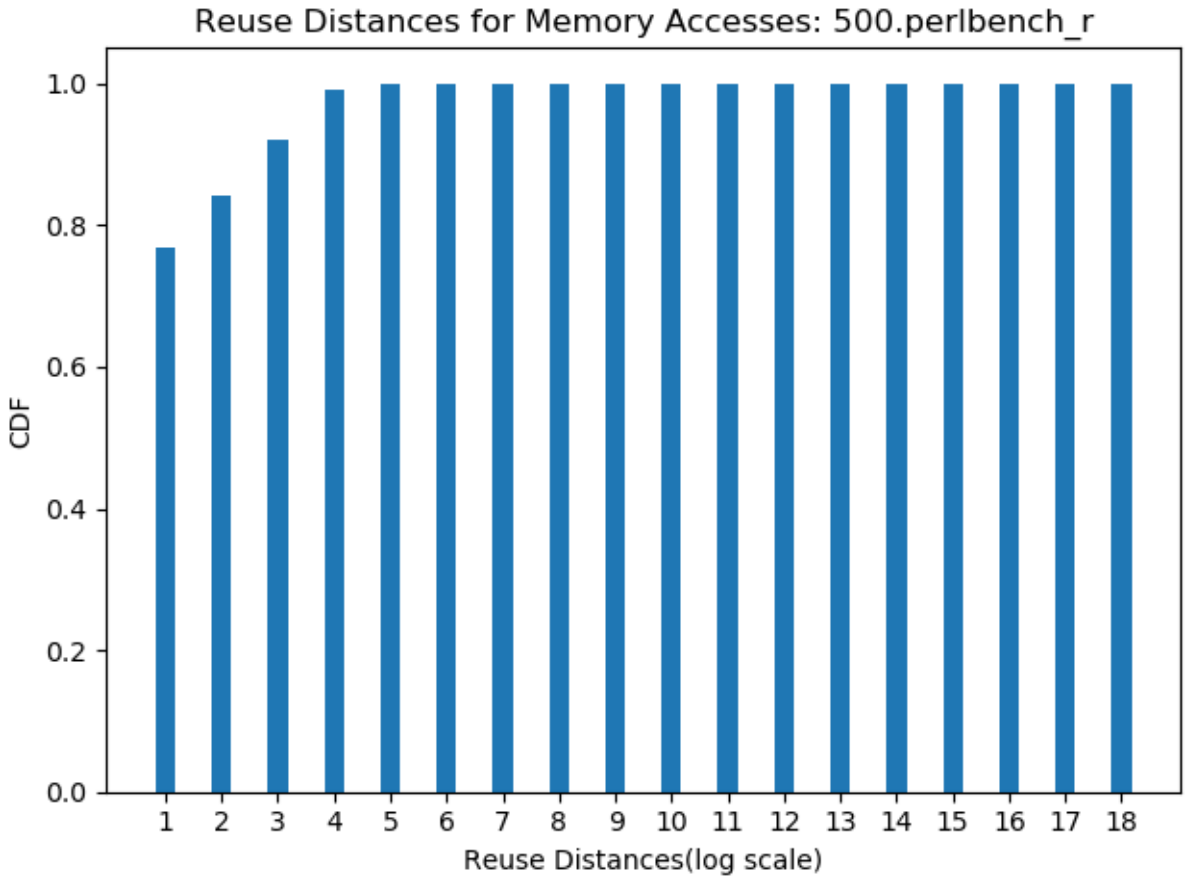
Results:



The above graph shows the percentage of memory pages that are reused during the running of each of the different SPEC test suites. As you can see, the memory reuse percentages are reasonably high for all of the test suites. This is good from an optimization perspective, as it means there is a greater potential to optimize these memory accesses and reduce the number of time-expensive memory page loads.



The graph above shows the average memory reuse distances for each of the test suites. As you can see, the average reuse distance is very similar between each of the test suites, and is low relative to the number of unique pages that are accessed during the running of the test suite. The homogeneity of the graph could likely be attributed to an issue with the data collection process that caused the memory trace files to be incomplete, which is discussed in more detail in the next section. Regarding the small reuse distance, this can be attributed to a majority of memory accesses being from the exact same page that was previously accessed, with only the occasional larger reuse distance driving up the average.



The above graph shows the reuse distances for one of the SPEC test suites: 500.perlbench_r. As you can see, the vast majority of memory reuse distances for this test suite are smaller than 2^5. This is in line with the small average reuse distance that we noted in the previous graph. I have not included the graphs for all of the test suites in this report, but they are all relatively similar to this graph, and are all available in the graphs directory.

Known Issues:

There are two main issues I have noticed with the memory traces generated by the Pin tool. First, I have noticed that the larger trace files are being cut-off early. I've noticed this when attempting generate the memory traces specifically for the SPEC test suites. It is not entirely clear how much of the memory trace files are being cut off, though it would not be unreasonable

to assume that these cut off amounts are substantial, which could be why the results for each test suite are so similar. This was due to the Pin pinatrace tool attaching by default to the incorrect SPEC process when collecting the data for the project. It seems like is attaching to the process that initializes the test suite, and not the test suite itself. I did find a potential solution at: https://software.intel.com/sites/landingpage/pintool/docs/97438/Pin/html/index.html, which mentions setting specific flags in the SPEC configuration file to correctly attach to the test suite process. However, after attempting this, the test suite ran for too long of a time for me to obtain the full memory traces prior to the submission deadline for this project. This long runtime is due to the default Pin tool for acquiring memory traces not being optimized for programs with larger runtimes. I am still looking into a solution to this problem to avoid these long waiting times moving forward, and it will likely involve optimizing the pinatrace tool, or creating a new custom Pin tool that would be better suited to generate the memory accesses in a more efficient way.

Second, I have noticed that in larger memory trace files, there occasionally may be an invalid memory address. These can vary in how they are ill-formed, but typically the second address in the access will be longer or shorter than the other addresses. Sometimes, they have more blatant problems, such as having missing line breaks or duplicated characters where there would be no reason for them to be duplicated. As it currently stands, I have handled these issues by simply removing the lines that have invalid addresses or formatting, since they are so small in number relative to the size of the entire memory trace file, I imagine it will not drastically influence the statistics generated for the file.

Conclusions:

This was one of my first introductions to a self-guided project that worked directly with a lot of software that was not well documented, relative to many of the projects I did in more guided classroom settings. This experience definitely pushed me outside of my comfort zone, as I would often need to consult directly with others who had more experience with systems software when I ran into issues, as there weren't any regularly documented instances of people running into the exact same issues I was having. This project also forced me to get better at reading through large log files, as I would often need to look for a single error line in a file with hundreds if not thousands of lines. If I had to do the project over again, I would also definitely reconsider the order in which I was doing tasks. I often attempted to focus primarily on the coding segments of this project, since that was what I was more comfortable and familiar with, though this often left me in times where I was blocked from doing any work, as I would be waiting to hear a response back from tech regarding an issue I was running into. Overall, I found this experience to be very eye-opening about the overall flow of larger, long-term projects, which will certainly be relevant to me in many future projects.