Sean Penney and Paul Atkinson
May 13, 2014
CS 472 Lab 5

Memory Optimization

The memory operation paper talks about practices to improve memory optimization, both in general and in more specific cercumstances.

The paper starts with why memory optimization is actually necessary. It is necessary because memory technology is progressing at a very slow pace when compared to how much CPUs are getting better. Thus, memory optimization is needed so our applications aren't bottlenecked by the memory rather then the processor. Moreover, there are a lot of easy practices you can develop that subtely help the run time of applications.

There are three main ways of optimizing code: rearranging, reducing and reusing. Rearranging is when you have variables that are accessed at the same time, you should have the variables declared next to each other. Reducing is reducing the amount of code to run, which obviously makes it more efficent. However, sometimes something that looks like less code in C is actually more lines of machine code, so it doesn't actually save time. In these curcumstances, you want to compile the code and then analyze it to see if there are some ways of shorting the amount of code that needs to be run. The final R is reusing, which means reusing as much code as possible so the program doesn't have to load in more then it has to. This usually means making functions if you have to do the same thing more then once.

There are also three main ways of screwing up the cache: compulsory misses, capacity misses and conflict misses. Compulsory misses are the first miss that happens when the data is read for the first time. While these are inevitible, it is good practice to avoid these as possible. Capacity misses don't happen quite as much anymore, but it is when the cache is too small to hold all the active data you want to hold. Sometimes you can hit capacity if you are accessing too much data inbetween successive uses. Conflict misses happen when you have two things that are mapped to the same line in cache.

Software prefetching and preloading can also help speed up memory operations. However, if you fetch too early, the data can take up unnecessary space or may be evicted from the memory before it is ever actually used. If the memory is fetched too late, then everything will be bottlenecked by it loading. Preloading is sort of the same as prefetching, however it isn't sure what is going to be needed so it guesses. Sometimes it guesses wrong and it ends up wasting time loading in the correct thing, but overall it usually saves more time then it loses.