Meow meow meow
Devlin Junker and Phil Lenox
CS 472 – Lab 3

Memory Optimization Slides summary:

The slide deck goes into some of the poor practices in the use of memory that often times compile into poor code, that is slow and inefficient. The guide opens with discussing at a higher level why cache misses occur, with the 3 C's of compulsory, conflict, capacity misses. The practices of pre-fetching, non-blocking loads of data before it is needed for processing, and pre-loading (psuedo-prefetching) to help lessen the compulsory misses. Strategies to minimize compiler buffering of structures, it is recommended placing the items in the structure in order of their size. The bulk of the information is about minimizing aliasing, where two pointers point to the same location in memory. Since the compiler cannot know ahead of time if two pointers are pointing to the same memory location, it must assume the worst. If a* = b* + 1, followed by c* = b* +2. These instructions cannot be done in parallel because a might equal b. in that case, b would be modified by the first instruction, and therefore must be loaded a second time. A way around this is to use local temporary variables where you as a programmer know that the variable are going to be different. These aliasing conflicts can occur due to class members, global variables, etc. The problem with using these, is the compiler does not know implicitly if a member variable has been modified by a pointer reference, and as such must access the memory each time. The use of the restrict keyword can help with this, but specifying that a function argument that is declared a pointer, points to a unique location in memory from other arguments, and it can be assumed that the value stored at that location will not change unless changed by referencing that variable.

Memory Optimization Paper summary:

The paper discusses current memory and caching technologies, and the differences between different architectures and implementations. It also covers virtual memory and techniques for optimizing your programs use of page tables and other virtual memory constructs. After the summaries of the technology that is available now, the author moves on to discussing techniques for writing or modifying a program to optimize it's use of the cache using prefetching, cache avoidance, and well-planned cache access techniques. After basic methods for improving performance, the paper continues on to multiprocessor and NUMA (Non-Uniform Memory Architecture) optimizations.  Finally, the paper discusses memory performance and benchmarking tools that can be used by programmers to measure the program and simulate different conditions. Additionally, the author briefly touches upon upcoming technology that will continue to help improve memory access and optimizations.