Assignment 4

Omkar Thakur

November 26, 2015

1 Memory Optimization

The CPU speed has increased tremendously over the last few decades. But, there have been no improvements in memory. If you compare the memory performance to the CPU speed, it has actually decreased. In order to improve memory performance, we need to understand how the memory hierarchy is constituted. With the use of the memory hierarchy, we can come to know how to use cache more efficiently. There are two types of caches. There is code cache and then there is data cache. The code cache is used for the instructions, while the data cache is used for data. The cache lines divide the cache which are of the size of 32/64 bytes each. In virtual memory, there are hits and misses of cache. More the hits, more is the performance of performance. Miss is when the data is not present in the cache and the CPU has to retrieve from the disk. The hit is when the data is present in the cache. The cache has most frequently used data. Therefore hit takes less time. The cache miss takes more time due to retrieving of data. A cache miss can happen due to 3 reasons. The compulsory miss is the first one. This is unavoidable when the data is read the first time. The second miss occurs when there is not cache memory to hold all the active cache data. This could be when there is a lot of data accessed between successive uses. When data is mapped to the same cache, the cache miss can also occur. This is the third case.

We can use some approaches to solve the issue of cache misses. The first one is rearranging the layout of the code and data to increase spatial locality. Spatial locality means that if you need the contents of one memory address then you will also need the contents of some of the other memory addresses around it. So due to keeping this in mind the cache hits can increase. Another way is temporal locality. Temporal locality means that if you need the contents of one memory address then you will need them again at some point of time. The third approach will be reducing the size of the number of cache lines read. This will require to use smaller and smarter formats and compress data.

The data cache can also be optimized in several different way. This can be done with prefetching, constructing cache-conscious structure layout, field reordering. Pre-fetching is used to place a cache line in memory before it is used. Therefore reducing the latency involved if the chip has to go off memory

to get data. Few different issues that we need to address before implementing it. The first thing is timing. If the fetching is done too early then the data might be evicted out of the cache before the program even used it and if it is fetched too late, the data will not be there when the data needs it. Another way of optimizing data will be to use cache conscious layout such as field reordering and hot/cold splitting. The padding can also be used for the data structures. For example, if we use padding in our structures to force the data on different cache lines, we can improve the performance and reduce false sharing. We also have a option of using tree-data structures for data optimization. This can be done by rearranging nodes in a way that increases spatial locality. We can reduce the size by eliminating pointers. Implicit pointers can do this work.

If the cache is linearized then it is the best possible way to improve performance because it provides the best possible spatial locality. Not only has it made easily prefetchable. This can be achieved by linearizing data at runtime. This can be achieved by fetching data and storing it linearly in custom cache. Another major issue will be of the anti-aliasing. This decreases the performance by hindering the reordering or elimination of loads and store and negatively effects instruction scheduling. There are a few ways in which we can combat this by using languages that have less aliasing and lower abstraction penalty. And we also need better compilers and better programmers who can effectively help the compiler and not just rely on it completely. The use of global variables, pointers and references should be minimized by the programmers. Mostly local variables should be used. They should also be declared close to the point where we will use them.

2 What Every Programmer Should Know About Memory

Software developers were targeted in the paper "What Every Programmer Should Know about Memory" by Ulrich Drepper. Difference between the structures of RAM, virtual memory, CPU cache, writing code advice, NUMA, future outlook and tools of trade were mostly discussed in the paper. In the RAM section, there was a comparison between SRAM and DRAM. SRAM stands for static random access memory and DRAM stands for dynamic random access memory. In the cache section of the CPU, utilizing SRAM as temporary storage within the CPU was discussed. NUMA section covers the commonality in the present days. The implementation details of subsystem and associated costs are discussed in the virtual memory section. NUMA stands for Non-uniform memory access. Tips regarding the method in handling memory in bypassing and accessing the cache, multi-thread optimization, prefetching, and NUMA programming are widely discussed in the sixth section of the paper. In the seventh section of the paper, some tools are discussed that are widely available to the programmer with improving and understanding the characteristics of the program. The tools provide instructions within the linux machine with cache and memory handling.

The issue of how Moore's Law is taking into the picture and causing issues with the software side is explained in the last topic. One of the ways to utilize all the cores is to use the parallel programming. Programming parallel is not an easy task even today.

The first section familiarizes the users with the concept of memory hierarchy The paper mostly discusses the difference between SRAM and DRAM. SRAM is comparatively faster than DRAM. But due to this, the DRAM costs less than the SRAM. The SRAM is expensive than DRAM due to the speed. The RAM requires constant power for immediate reading. The concept of how memory is cost effective since it is being produced and being sold to the electronic consumer market. In case of businesses and data centers the case is different. It is very expensive for them since those markets are mainly based on high performance and they are different than consumer products. This gives rise to the issue known as bottleneck. The bottleneck for the high performance memory must be balanced with concept called DMA or also known as direct memory access. The DMA store and receive data in RAM without the CPU intervention. Normally the CPU must be utilized or communicated with in order to use RAM.

In summary, SRAM has basic structure which includes six transistors. SRAM is faster and expensive in cost. This RAM will also require constant power for immediate reading. The cell state will always be stable which results without the need of any refresh cycles. DRAM is very different in structure. The DRAM contains one capacitor and one transistor. But there is a drawback in this design. The drawback is the leakage caused by the capacitor. There is dissipation from the capacitor which causes it to leakage. DRAM will have a refresh cycle and the SRAM does not contain any refresh cycle. The every read done in DRAM must be followed by a recharge in the capacitor. The DRAM is not instantaneous as compared to the SRAM. This is the main drawback of the DRAM.

In conclusion, SRAM is much faster and has an independent power, while DRAM is simpler design and is more compact. The memory management is discussed in depth for the programmer.

3 References

http://www.akkadia.org/drepper/cpumemory.pdf
http://www.stackoverflow.com
http://www.wikipedia.com
http://www.quora.com
http://www.realtimecollisiondetection.net/pubs