The Evolution of System Performance:

A Computer Technology Paper

Samuel Scalf

University of Maryland Global Campus

Author Note

This paper was prepared for CMIS 310 (6380), taught by Professor Yul Williams.

The Evolution of System Performance:

A Computer Technology Paper

Since humans were able to create the first tools, they have been striving to become more efficient. Early on there was a push to more efficiently grow and harvest crops. This desire for improvement has carried on into the Information Age. The earliest computers had next to no abilities when compared to even a scientific calculator, let alone a modern personal computer. The efforts towards increasing the performance of systems, that brought us our modern computers, are displayed by innovations such as Reduced Instruction Set Computing, pipelining, cache memory, and virtual memory.

Reduced Instruction Set Computing (RISC) is not a new concept.  The term was coined by David Patterson sometime in the early 1980s (Reilly, 2003).  RISC's main concept is that of simple instructions. Some of its other main concepts are one operation per clock cycle, one word per instruction, and fixed-length opcodes. By limiting the instructions in such a way, the performance of the system is increased significantly.

Running simple instructions is great in that they can be completed quite rapidly. This is not as efficient as it could be. Imagine stacking sandbags for a moment. It is not efficient for one worker to get a sandbag, take it over to the stacking area, and walk back with all the other workers waiting for the first to return before sending the next. It would be far more efficient if the second worker grabbed a sandbag right after the first, placing the sandbag on the stack right after the first, and so on for all the other workers. The same is true for a program, as well. When it comes to computers, this process is called pipelining. In the words of Shen and Lipasti (2005): “Pipelining involves partitioning the system into multiple stages with added buffering between the stages. . . . A new task can start into the pipeline as soon as the previous task has traversed the first stage” (Motivations, para. 2).

To give an example of how much faster this form of pipelining is, let the fetch phase require one clock cycle, the decode phase require two clock cycles, and the execute phase require three clock cycles. If a program is composed of 100 RISC instructions, without pipelining, it would take 600 clock cycles to complete the entire program. By implementing the pipelining in the previous example, known as scalar pipelining, the program would complete in only 105 clock cycles. That’s a performance increase of approximately 570 percent! If you throw in a bit of parallelism with a second process, for instance, there would be an even greater improvement. The program would now complete in just 55 clock cycles – an improvement of approximately 190% from the scalar pipeline model or approximately 1090% from the original model.

To even further increase system performance, scientists and engineers attempted to reduce the amount of time required to access memory. This was a big deal since RISC programs need a multitude of instructions, especially when computing more complex programs. The entire program would need to reside in memory, so if an instruction was to be fetched from memory each clock cycle, the clock cycle could not be faster than the time it took to receive the instruction from memory.

With the advances made in transistors by this time, scientists and engineers were able to create a small amount of memory on the same physical chip as the CPU. This effectively eliminated most of the latency caused by accessing separate memory. Kelley (2019) notes the relation:

Ultimately, the highest possible performance potential of a computer is data sitting at the CPU. When the CPU has to go down-stream to get needed data, latency delay causes performance to fall off from its maximum. The further the required information is from the processor the longer the wait due to limits of the speed of travel of the electrical current. (para. 8)

This new memory, called cache memory, was not as large as main memory. In order to overcome this issue, only small chunks of instructions, called blocks, were copied over to cache memory at a time. While the first instruction may take some time to load, the next few instructions would be already in cache memory. If the cache memory ever filled, some of it would be discarded and replaced by different instructions from main memory. There are a few different algorithms used to decide which memory to replace, such as first-in-first-out. Regardless the method used, it is clear that there is at least some performance increase. Modern processors tend to have two or three different sized layers of cache memory, further increasing system performance by decreasing latency.

Just as cache memory is limited, so is main memory. If a program was large enough that it couldn’t fit in main memory, the program would not be able to be loaded and would fail. This would require further advancement in technology in the opposite direction of cache memory. It was decided that exceptionally large programs could be stored in even larger memory, such as hard disk drives (HDD). The program, stored on an HDD, would send chunks of instructions, now called pages, to main memory. This process is very similar, but not identical, to how instructions are loaded into cache memory from main memory. While this did not increase system performance, per se, it did enable systems to be able to run much larger programs than ever before.

These four innovations – RISC, pipelining, cache memory, and virtual memory – have increased system performance, for sure. Given human nature, it is a guarantee that the future holds even more improvements. Perhaps one day a replacement for transistors will be found that exponentially increases the speed of switching. It can be exciting to wonder and speculate on the improvements that are soon to come.

References

Kelley, R. (2019, February 1). *Compute Performance – Distance of Data as a Measure of Latency.* Retrieved from Formulus Black: https://www.formulusblack.com/blog/compute-performance-distance-of-data-as-a-measure-of-latency/

Reilly, E. D. (2003). *Milestones in Computer Science and Information Technology.* Westport, CT: Greenwood Press.

Shen, J. P., & Lipasti, M. H. (2005). *Modern Processor Design: Fundamentals of Superscalar Processors.* Long Grove, IL: Waveland Press, Inc.