Carl Patterson

Ping Chung

Computer Architectures

Fall 2018

Parallel Processors Advantages, Disadvantages and Challenges

Abstract

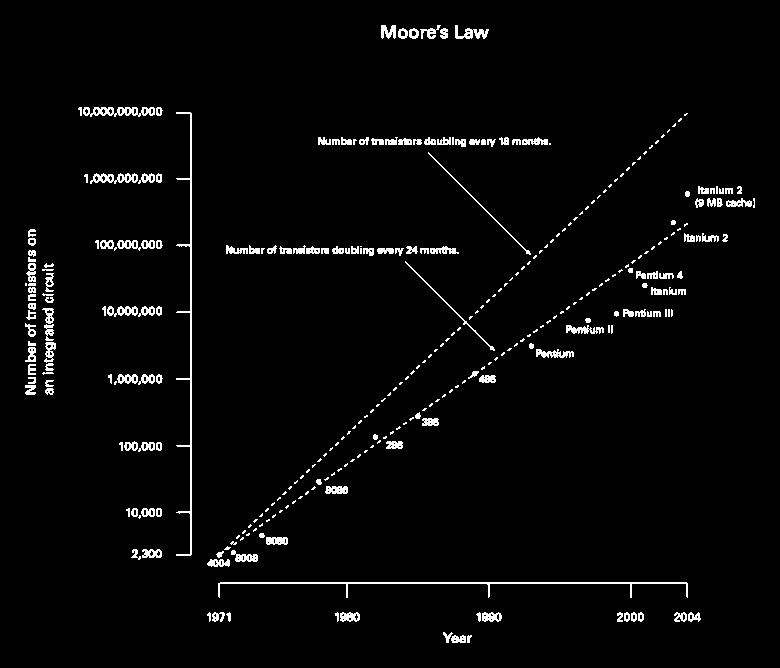
This paper is an overview of parallel processors. Parallel processors are a fairly recent phenomenon in the computing world and as such there are many improvements to be made in order for us to realize their maximum potential. In this paper we will explore their history, their advantages as well as the challenges that come with them and what computer scientists today are doing to face some of these challenges.

Computers have come a very long way since their inception. From being very heavy bulky machines occupying an entire room to tiny devices that can fit in the palm of your hand. Not only did the size of computers get smaller, but their processing power improved along with it. For some perspective, according to consumer reports, the computer used by NASA to guide the Apollo 11 spaceship boasted “0.043 MHz” of processing power. Compared to the “under powered” netbooks of 2009 (the 40 year anniversary of the Apollo 11 moon landing) that had 1.6MHz of processing power which is “several thousand times faster.”

The increase in processing power is in part due to advancements in computer technology as a whole. The process of designing and building circuitry has gotten much better and has led to improved reduced latency. Memory technology performance has drastically improved for both volatile and non-volatile, reducing the time it takes to perform read/write operations. Even I/O devices have gotten much faster, reducing the interaction time with computers. Probably the most important performance improvements have come from the improved performance from our modern day processors.

A processor or central processing unit (CPU) is a small chip that receives and executes instructions from input and then passes the results on as output (Techterms.com). The processors in our computers are a major reason why we are able to do so many amazing things with our computers. From simple mathematical operations to more complex operations like calculating the trajectory of a satellite launch into space. Originally processors would handle all types of operations by itself, including calculations for applications, graphics and audio. Nowadays a processor will pass on more specific types of jobs like audio and graphic rendering to an audio to other more specialized chips like an audio card or a graphics processing unit (GPU). This leaves the CPU with more time to focus on other tasks, saving both time and energy.

The very first general purpose processor was the Intel 4004, released in 1971 (W. Warner). According to Warner it was a 4-bit chip that had a clock speed of 108KHz. The chip came about, according to Intel’s own history museum, from a request for processors to be used in “printing calculators.” Intel obliged and developed the chips but later realized the 4004 could be used for more general purpose applications and so they “renegotiated the contract” (Warner) to be able to sell the chips commercially. According to Intel the 4004 had “2300 transistors” which was a lot back in 1971. An Intel Core processor in 2010 had “560 million transistors”, about 243478.26 times more transistors.

The increase in the number of transistors in CPUs is in line with predictions made by Gordon Moore, a founder of Intel, when he stated that the number of transistors in processors will double every two years [CITE], this became known as Moore’s Law. Processor manufacturers followed this for a certain time but after a while this proved to be inefficient on single processor chips and so they began to make processor chips with multiple “cores” (Brian Schauer). According to Schauer manufacturers focused on increasing performance on single core processors by employing a number of different techniques including, superscalar processing, reorder buffers, dynamic scheduling, register renaming, data value prediction and trace caches.

Superscalar processing as defined by the University of Indiana’s knowledge base, is when “the CPU manages multiple instruction pipelines to execute several instructions concurrently during a clock cycle. This is achieved through “feeding different pipelines through a number of execution units within the processor” (Indiana University). The instructions in the pipelines are “pre-fetched, split into sub-components and executed out of order” and then at the end the results are collected and evaluated(Schauer). One caveat of superscalar processing is that if an incorrect decision is made by the CPU then it leads to a stall, because the processor will have to backtrack to the previous instructions and then continue on from there (UI). Dynamic scheduling, as the name implies, is a technique used by operating systems (OS) designers to schedule the order of program execution, in order to increase efficiency and throughput, to make better use of CPU cycles. “The hardware determines the order in which the processes are executed” (Allan Tong). Some different methods used to implement dynamic scheduling is scoreboarding and Tomasulo’s algorithm.

Despite the use of these techniques and methods to try and improve the performance improvements on single core processors was still slowing and it also provided a different kind of problem. The processors consumed a lot of power (Shuka, Mishara, Zilic) had poor heat dissipation (Schauer) which led to performance and longevity issues of processors. To gain further increase in performance without further pushing the limits of the processors the idea of parallel processing by placing two cores on the same chip came about. Parallel processing, according to Quentin F. Stout, is “the use of two or more processors (cores, computers) in combination to solve a problem. Parallel processing can be done by using multiple methods, however the two common ways to do it is have multiple cores in one chip or by creating a cluster. A cluster, or grid computer, is a group of computers connected together over a high speed, local network that share resources such as storage (UI). Grid computers require a different type of scheduling since the processes have to be split among not just one processor chip but two or more, an example of this is Linux’s message passing interface (MPI).

The dual core chips were predicted to consume less power and heat while increasing performance (Samuel K. Moore). Phillip E. Ross wrote that Intel reported underclocking a CPU saves “half of the power while sacrificing 13 percent of the performance.” This means dividing the work between two cores running at 80 percent clock speed would increase performance by 73 percent while saving power and dissipating heat in a better fashion (Phillip).

In 2005 AMD released its first dual core processor, the “Anthlon 64 X2 3800+” (Computerhope.com). According to cnet.com the Anthlon is a 64bit processor that has 2Ghz clock speed with 1MB of L2 cache. Ever since AMD released the Anthlon the multi-core processor race began and we’ve seen manufacturers try to cram more and more cores unto a single chip.

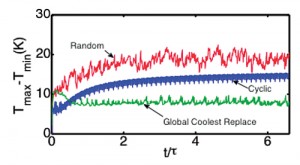
Once the multicore revolution got going, it had a natural momentum. “As soon as we got to two cores, it became obvious we needed to start thinking about going to four,” says AMD corporate fellow Chuck Moore. “And as soon as we got to four, we started thinking about going to six or eight. (Moore)

Today 8 core processors are common especially in mobile phones and tablets where minimal power consumption while providing excellent performance is a very big challenge for the designers. There have even been companies that sell 64 and 100 core processors with plans to create 200 core processors. Increasing the number of cores in a processor comes with its own set of challenges. Adding multiple cores means that designers how have to account for “power and temperature management,” according to Schauer, for more than one cores. Shauer also said that memory and cache coordination becomes a big problem because these are now shared between two processors. Lastly, probably the biggest challenge, is that programmers now have to figure out how to write programs that take advantage of parallel processing to fully capitalize on the increased performance.

The trick will be to invent ways for programmers to write applications that exploit the increasing number of processors found on each chip without stretching the time needed to develop software or lowering its quality (David Patterson).

The goal of thermal management on a processor is to prevent the chip from becoming too hot during, especially heavy, usage to prevent loss of performance and reduced lifespan of the chip. The preferred temperature management method (TMM) will ideally be able to reduce the operating temperature of the processor while it provides optimal performance, predictable enough to deliver constant results, adaptable enough to handle fluctuations (Bartolini, Tilli, Benini). In addition to this it should be scalable enough to handle different numbers of cores, robust enough to filter out noise from sensors and have enough modularity to be able to work with other thermal management methods to keep the temperature of the chip as low as possible (Bartolini, Tilli, Benini).

There are multiple methods for thermal management in multicore processors. A few of these methods are air cooling (done with fans), heatsinks, water cooling, immersion cooling and power multiplexing. One issue with the the above methods is that they are either expensive, power consuming, require a lot of space and they are physical methods, save for power multiplexing (PMUX), and they do not meet all the requirements for being the ideal TMM as listed above. PMUX is done at the software level and therefore it can be used in both mobile and slim devices that cannot fulfill the space requirements for the methods listed above and it can also be used in parallel with those methods to manage CPU temperature. This is because PMUX is a “proactive method” which can be “utilized as a supplementary approach” to other methods of cooling a CPU (Gupta, Kumar).

“Power multiplexing technique involves redistribution or migration of the workload of the cores in the chip at regular time intervals to control the thermal profile of the chip” (Gupta, Kumar). There are three main ways PMUX can be done. The workload can either be distributed randomly, globally or cyclically. Random sounds exactly like it is, the workload is randomly passed around from one core to another at the predetermined time interval. Global PMUX always switches the workload from the hottest cores to the coolest cores. Cyclical “preserves checkerboard configuration during multiplexing” (Gupta, Kumar). The three methods of PMUX had different effects on CPU temperature with global being the most efficient at keeping temperatures lower. The main issue I could find with this approach is that switching processes between cores takes time, that could slow down the throughput of the CPU because it is constantly switching processes between cores.

Memory management with multicore processors is another big challenge for designers. Each core will have its own Level 1 (L1) cache right next to that is used to temporarily store data needed for very fast access (Schauer). In addition to L1 there will also be L2 cache. Schauer writes that “Level 2 (L2) cache is just off-chip, slower than L1 cache, but still much faster than main memory; L2 cache is larger than L1 cache and used for the same purpose.” In addition to these two the processor will also be placing and taking data from main memory, which will be larger than both L1 and L2 cache but much much slower.

The issue with having multiple cores comes up with the data stored on each core’s L1 cache. Without proper communication between cores then you have an issue where one core updates a piece of data in it’s L1 cache but hasn’t committed the changes back to main or even L2 cache yet. If another core tries to do some work on the same data, it will only have access to the outdated piece of data. If both cores then write data back to main memory with their changes then it causes issues for the program or the computer, depending on whether or not it is a user or system process. This issue can be handled by having a directory that stores information on what pieces of memory are being shared or hosted on a core’s L1 cache, allowing processors to better coordinate when splitting the workload, especially for the same program (Schauer).

As stated earlier multicore processors not only provided the designers with challenges, but it also provided a very new, massive problem for programmers. Prior to mulitcore programmers only had to write applications that would run on a single core, the code was a lot more straightforward as they didn’t have to worry about managing threads or how scalably their programs were. Now with multicore this issue came up, how do we write programs that fully take advantage of the new performance improvements we gained from having more than one core?

David Patterson wrote that programming for multicore is akin to 10 reporters working on the same story in order to achieve a 10-fold speedup, if the work is divided equally among all 10.

Complications would arise, however, if one part of the story couldn't be written until the rest of it was done. The 10 reporters would also need to ensure that each bit of text was consistent with what came before and that the next section flowed logically from it, without repeating any material. Also, they would have to time their efforts so that they finished simultaneously. After all, you can't publish a story while you're still waiting for a piece in the middle to be completed. These same issues—load balancing, sequential dependencies, and synchronization—challenge parallel programmers.(Patterson)

Programmers have to figure out how to write their programs so that its workload is properly split up among all the cores and that they stay in sync in order to piece the data together at the end of computation (András Vajda). If programs are not written to capitalize on the available cores it will only run on a single core.This issue is known as starvation which Schauer describes as a situation where one core has a lot of work to be done while the others sit idly. This also means that one core will be running a lot hotter than the rest, a problem that was discussed above. Some programmers, Patterson wrote, suggested writing programs to “automatically parallilize” sequential programs. While there has been some progress in this regard, we still have a long way to go. Programs not only have to be written in a way that allows them to coordinate the workload across multiple cores, caches and memory locations, but the issue of priority also exists. How does one decide what program gets it’s jobs run on the CPU first? Scheduling tasks is a lot simpler when you only have one core to worry about, but with more than one you now have to deal with somehow prioritizing jobs across multiple cores.

Questions such as what happens when a system program needs it’s job to be done but a parallellized user level program currently has a job running on multiple cores? Do you stop the user level program? If yes then what happens if the other core finishes it’s slice of the work but now needs the work from the other core to be completed? What if there are multiple higher priority jobs that needed to be ran? Because of these issues operating system designers had to come up with more complex scheduling protocols, to handle prioritizing tasks across multiple cores, and memory locations. In addition to this the issue of scalability also comes up (Vajda). The programs written for multiple cores would need to be able to properly scale depending on the number of cores available on the computer’s CPU; be it 2, 8 or 200. We can already see how difficult this would be for programmers to achieve. Nowadays most, if not all, programming languages have libraries dedicated to allowing programmers to write parallel programs. I have used Python’s multithread and multicore libraries in a small web scraper before and saw increased performance from the high level functions in these libraries.

While it will prove difficult to take full advantage of multicore CPUs there have been different industries that have found it to be very useful and have been able to utilize them to their advantage. Examples of this include, banks for handling multiple transactions, airlines for handling flight reservations, search engines, car crash simulations, weather predictions, computer graphics (both still and video), cloud computing and the scientific community (Patterson). From personal experience, given the chance to do research where my team optimized software that runs on a cluster used to process scientific data, this is especially important for scientists in fields that need to process terabytes of data at a time that could take hours or even days to complete. My team was able to successfully increase the throughput of the cluster by optimizing the software using work stealing, leading to a significant reduction in the processing time.

In summary, parallel or multicore processors came about because of the need for increased performance without the drawbacks of forcing faster clock speeds like increased temperature and power consumption. Multicore CPUs came with their own set of challenges however, that shifted the burden of increased performance over to software developers who were now forced to figure out how to write programs that are able to take advantage of having multiple cores. Despite making some progress, there is still a long way for us to go if we truly want to utilize the full potential of parallel processors.