CMPSCI 630 Systems **Fall 2019**

Lecture 7

Lecturer: Emery Berger *Scribe: Chenhao Huang, Shaurya Patel*

7.1 WASM CISC or RISC?

RISC stands for Reduced Instruction Set Computer. As opposed to CISC that stands for Complex Instruction Set Computer, RISC only contains the basic instruction whereas CISC contains many specialized instructions.

7.2 Special purpose chips vs. General purpose chips

Many special purpose chips were created over the years, WASM had a chip which was optimized for WASM and was stack oriented. This is called tagged architecture. LISP machines were created to run LISP code very fast. They had support for instructions like CAR, CDR and even a Garbage collector. But, the target userbase of these chips were quite small whereas there was a huge market of general purpose chips. Intel which creates general purpose chips and has a huge market and invests in something called FABS a 1 billion dollar venture because of something called economies of scale. FABS are where chips are created. Semiconductors are able to hold charges and the smaller they are the better. Intel invested in FABS which is a very expensive process but they are extremely good at making very small transistors with a very high yield, say 95 percent. On the other hand every time you introduce a new architecture you need an entirely new FAB to support it, Intel makes a new FAB for every architecture but you cannot get them to create custom chips for you. You need to compete on two things, micro architecture design, there were a lot of them and you have to deal with hardware process for special purpose market.

Intel is no longer the biggest manufacture of chips. The market got bigger but then other companies gave up. People thought Intel just won. AMD is another company that exists with a bit of the market share. AMD has their own FABS as well but as it turns out FABS are complicated to make and thus AMD's FABS have a lower yield, which leads to a lower profit. During the switch to the multicore era AMD pulled ahead of Intel for a while as well.

Then there is Moore's Law which hasn't stopped but Dennard scaling did. Dennard scaling basically said that we can increase the clock speed without increasing the voltage, but that no longer holds true. From 70s it was like 1MHz, 1981 almost 5MHz, 2006 2GHz for processor, started the curve doubling of cycle frequency every 18 months. In 25 years we obsered a 400X improvement in cycle speed. But that was unsustianable as Dennard scaling stopped being true so heat disappiated was hard to handle. In 2006 the voltage consumed was like 50 watts but then in 2015 it was upto a 100 watts. Chip real estate increased over the years as transistors got smaller so we started adding more cache to it. But there was still more real estate, so they decided to add more CPU's and that's how the multicore era was born. But as it turned out making things run in parallel was really hard. Intel also had this idea that they would switch the CPU's in a hot/cold state where if 1 CPU got too hot they would switch the workload to the cool CPU.

7.3 Amdahl's law and Gustafson's law.

AMD 70% yield is not that bad, they had an ingenuous way to deal with chips that came out with bad cores, they just disabled the bad cores and sold 5 and 7 core chips. What is the real problem with harnessing the power of multicore? You have to write code that is parallel, because auto parallelization of code is really hard. One way code was made parallelizable was to break it into threads. This was also not an easy solution as it was hard to get it right and often didn't increase performance. There are global variables and private variables that a program has suppose x is one. You can read and write x with different threads. In an ideal sequential scenario you would expect atomicity, but it is not guaranteed in a multi-threaded environment as any thread could execute first and that depends on the scheduler. Atomes means indivisible. In computing it stands for indivisible i.e transactions are all or nothing. You either commit a transaction and then it becomes visible or you have to roll it back and retry. Java has a syntactic structure called synchronized which can be used to emulate this. It helps in acquiring a lock before executing a certain piece of code, so that only one thread can access it at a time. The java lock is actually recursive and allows the same thread to acquire the lock again, where as C doesn't and you would get stuck in a deadlock. Hardware has a special thing that allows for atomic binary transactions to change the lock values which makes this implementation possible. A spin lock is a lock where the process just spins idly on the CPU till the lock is acquired. It wastes CPU cycles. There is also a problem called the 'Thundering Herd Problem' where when a thread releases a lock all the other threads rush to acquire the lock, this problem is unfair as the last thread to have waited also may acquire the lock. There are also queuing locks, they are build on these lock primitives. You can eventually put synchronized everywhere and slow down the performance, adding locks is neither necessary for correctness nor sufficient for performance. Fine grained locks at every statement don't work because of scheduling so we need to make them coarse.

Suppose at the start of the below program we set y = x.

$$lock(x = x + 1)$$

$$lock(x = x - 1)$$

$$lock(y = y + 1)$$

$$lock(y = y - 1)$$

so $x = y$ should always hold, but if there is no coarse lock then a context switch might result in another process changing the value of y.

Dead lock problem or "Deadly embrace" as Djikstra had called it happens when there is cyclic between threads in lock acquistion, everytime there is a cycle there is a deadlock.

Another problem that can arise due to a multicore environment is the False Sharing problem. It occurs when two processors are sharing a cache and even though they may be referring to two seperate variables x and y which might be at either ends of the cache line, everytime a value is updated the other processor has to go to memory to get the value back. This leads to an 8-10x slowdown in performance. This is another reason why it is hard to take advantage of multicore.

Amdahl's law is a mathematical equation. Consider a process which is sequential and then forms 3 threads. These threads perform some work and then are connected back and essentially form a DAG. Suppose the time taken by one processor to complete the task is T1, this is also called the total work of computation. The goal is to achieve perfect scalability or $T_n \approx T1/N$. The longest path from top to bottom in the DAG graph T_∞ is called the critical path(height of dag), $T_P \approx \frac{T_1}{P} + T_\infty$ p is parallel work and T_∞ is the serial path that cannot be executed in parallel. Suppose 90% of program is perfectly parallelable, $T_P \approx \frac{90\%}{P} + 10\%$ even if p goes to 0 that is we have infinite processors, then it still takes 10% of the time to run the process, from 100% it speeds up to 10%, but even with infinite processors it is only a 10x speed up. In reality we only have a 20% to 50% speedup. That's why we don't have more cores on processors.

Along with multiple cores and multiple CPUs, we also have massive data centers now. They work due to the difference in workloads, running Virtual machines is embarrasingly parallel and it scales up by the number of customers. They have no serial part and no communication. The more customers' there are the number of parallel processes increase. Amdahl's law is concerned about one workload, with a fixed size and it estimates the effect of parallelism on it. Netflix has people independently watching movies which is Embarrassingly parallel. Facebook and Google have different ML workloads which are also embarrassingly parallel. The Size of the data and number of users have scaled up.

Gustafson's law essentially states that the problem size will increase faster than the number of processors. $\triangle S(problemsetS) >> \triangle P(numberofprocessers)$ if this holds true, there is no reason stop getting more processors. This is a counterpoint to Amdahl's law and it is basically the reason why data centers work.