CSC367: Parallel Computing

QiLin Xue

Spring 2022

Contents

1 Introduction

Why are all computers since 2005 parallel computers?

- Moore's Law: number of transistors in a given space doubles every 18 months.
- Similarly, the frequency of clocks are also increasing exponentially.
- NB: Above was only mainly true for pre-2000 computing.
- **Power Wall:** The dynamic power is proportional to V^2fc (since $P = \frac{V^2}{Z}$ and impedance of capacitor is $\frac{1}{\omega C}$). Therefore, increasing both the frequency and voltage causes a cubic effect, which is *bad*.
- An alternative is to linearly increase the capacitance but save power by *lowering* the clock speed. To save power, some companies use less-powerful transistors but use more of them: leads up to parallelism.
- Since 2000, chip density is still doubling every 2 years, but the *clock speed is not*. Instead, we see a new trend where the number of processor cores may double. This allows power to be under control, i.e. it is no longer growing.
- One main problem is that all code were sequential: How can we adapt them to multi-core architectures?

What are some of the world's fastest computers? (Top 500 list)

- To measure high performance computing (HPC), we use the units:
  - Flop: floating point operation, usually double precision unless noted
  - Flop/s : floating point operations per second
  - Bytes: size of data (double precision floating number is 8 bytes)

  Laptops are usually Gflop/s $= 10^9$ flop/sec, while the clusters we'll be working with in this course are Pflop/s $= 10^{15}$ flop/s.
- See the Top 500 List.

What does a parallel computer look like?

- Regular Computer: We have a Proc/Cache/L2 Cache connected to a L3 Cache which is connected to Memory
- For parallel computers, there are potential interconnects between Proc/Cache/L2 Cache and L3 Cache and between L3 Cache and memory.

Does performance engineering really matter?

- Example: Dense matrix multiplication. Consider we are trying to multiply two square matrices.
- Let's code this up in Python, Java, C and implement it on the Intel Haswell Computer System. Consider $n = 4096$,
  - Python: 21042 seconds = 6 hours
  - Java: 2387.32 seconds
  - C: 1156 seconds = 19 minutes
- Back of the envelope calculation of the target speed. There are $2n^3 = 2^{37}$ floating point operations. Therefore, we have $2^{37}/6.25 = 0.007$ GFLOPS. For Java, we have 0.058 GFLOPS and C gives 0.119 GFLOPS, which is 0.01% of the peak.
- The reason is that Python is interpreted: dynamic interpretation at cost of performance. Java is compiled to byte code and C is compiled to machine code.
- When we implement matrix multiplication using three nested for loops, the loop order affects running time by a factor of 15!
  - In C, matrices are stored in memory in row-major order.
  - Caches: Each processor reads and writes from main memory in contiguous blocks, called cache lines. Previously accessed cache lines are stored in a smaller memory called the cache. If the data required by the processor resides in cache we get a cache hit, fast! Data accesses not in cache can be slow.
  - The reason one order might be better is because we can minimize the last-level cache miss rate.
- With simple optimizations, we can improve this to 0.3%.

- With parallel loops, we can get to 5.4% of peak.
- There are a lot of other optimizations we can do, and we can improve that to 42% (which uses a different implementation, i.e. Intel MKL). That's a 50,000 times improvement!