Lecture Outline for Week 1

Prof. Naga Kandasamy
ECE Department, Drexel University

- Limits of single-core machines, focusing on thermal issues, wire delays, DRAM access latency, and diminishing returns of instruction level parallelism (or implicit parallelism).

  – **Thermal issues:** Main factors contributing to CPU power consumption are dynamic power consumption, short-circuit power consumption, and power loss due to transistor leakage currents:

$$P_{\text{cpu}} = P_{\text{dyn}} + P_{\text{sc}} + P_{\text{leak}}$$

The dynamic power consumption originates from the activity of logic gates inside a CPU. When the logic gates toggle, energy is flowing as the capacitors inside them are charged and discharged. The dynamic power consumed by a CPU is approximately proportional to the CPU frequency, and to the square of the CPU voltage:

$$P_{\text{dyn}} = \alpha C V^2 f,$$

where C is capacitance, f is frequency, and V is voltage. The transistor switching activity is captured by α. Power consumption can be reduced in several ways: Voltage reduction (dual-voltage CPUs, dynamic voltage scaling, under-volting, etc.); frequency reduction (under-clocking, dynamic frequency scaling, etc.); capacitance reduction; clock gating; techniques to reduce the switching activity; recycling some of that energy stored in the capacitors (adiabatic circuit, energy recovery logic, etc.)

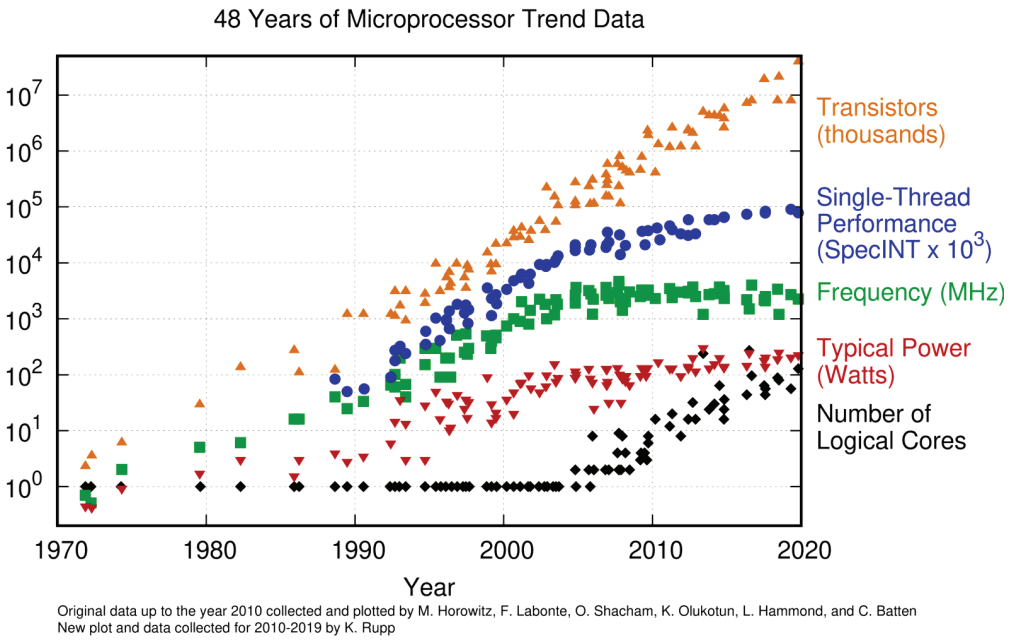


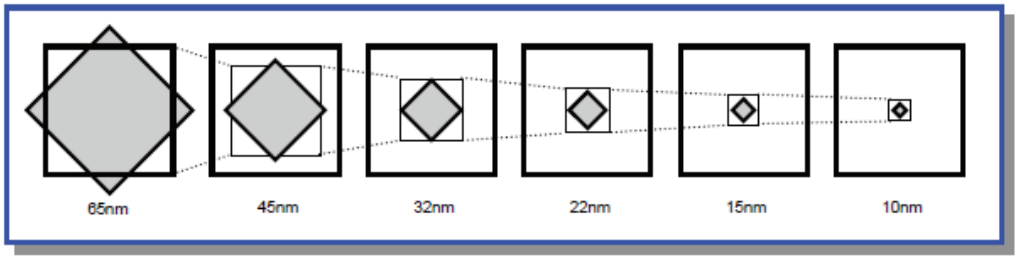
Fig. 1: Microprocessor trend data.

Figure 1 shows 48 years worth of trend data for processors obtained from `github.com/karlrupp/microprocessor-trend-data`. Transistor counts in integrated circuits are still growing, but corresponding performance improvements are more gradual than the speed-ups resulting from significant frequency increases. At small transistor sizes, current leakage is a major contributing factor to heat generated in the chip, creating a threat of thermal runaway. So, to balance thermal issues with leakage current, dynamic power consumption must be kept low by reducing the operating frequency.

The inability to increase clock frequencies significantly has caused most CPU manufacturers to focus on multi-core processors as an alternative way to improve performance. Parallelism helps reduce power consumption while improving performance. Example: NVidia 640 GT has 384 cores, each clocked at about 800 MHz, whereas the GTX Titan Z has 5760 cores, each clocked at 700 MHz. The 1080 GTX that we will use for programming assignments has 2560 cores.

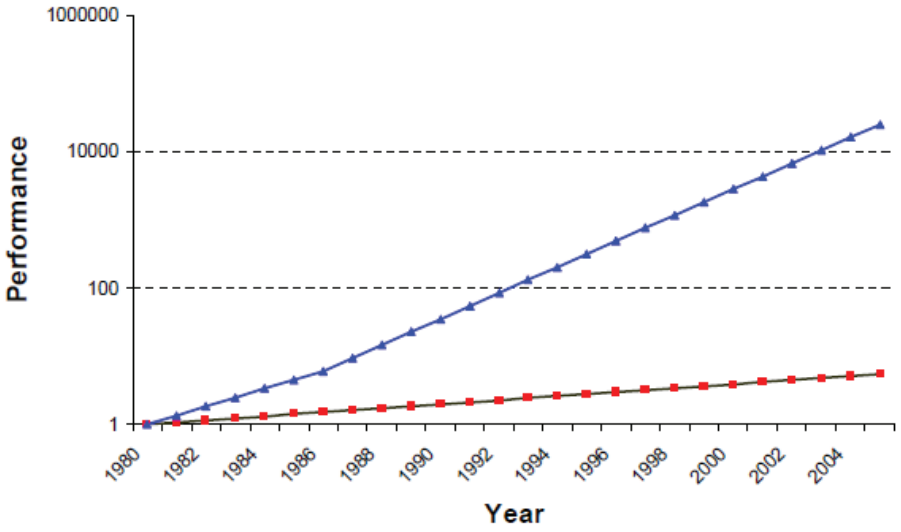
– **Wire delays:** Today's chips are like yesterday's circuit board. The critical path is affected due to circuit delays, called the propagation delay, as well as delays due to long wires on chips.

  The critical question is how far can a wire reach in a clock cycle. Each technology scaling step reduces everything except die size (assume die complexity grows); wire reach falls a little faster than the original block size shown in the figure below.



  We need repeaters or buffers to extend the reach of a wire, slowing down the system.

– **DRAM access latency:** DRAM (main memory) has large capacity but the access time is slow compared to a CPU cycle. It takes multiple CPU cycles (≈ 100 cycles) to access an item in main memory. The gap is growing as the following figure shows.



A cache hierarchy comprising of more expensive SRAM memory closer to the processor (L1 and L2 caches) is used to speed things up. However, SRAM is getting harder to scale.

  - **Limit on extracting ILP:** Compilers can extract "implicit" parallelism from serial code efficiently. Pipelined hardware aims to provide an ideal cycles per instruction or CPI of 1. Multiscalar processors (n-way issue pipelines) aim to get CPI ≤ 1, or instructions per cycle (IPC) > 1. We have squeaked out the last implicit parallelism using 2-way to 6-way issue, out-of-order issue, branch prediction, speculative execution, etc. We have hit a limit at around 0.5 CPI in terms of extracting implicit parallelism from serial code.

- Types of parallelism: implicit vs. explicit.
  - Implicit parallelism: Superscalar processors issue varying numbers of instructions per clock cycle. Instructions can be statically scheduled by the compiler and issued in-order by the hardware, or the hardware can dynamically schedule instructions in parallel as data becomes available in an out-of-order fashion (scoreboarding and Tomasulo's method were covered in ECEC 412/621).
  - Explicit parallelism: Exposed to the programmer or to the compiler.
- Flynn's classification of parallel machines.
  - **SISD:** Single instruction, single data. Serial code.
  - **SIMD:** Single instruction, multiple data. Vector code.
  - **MIMD:** Multiple instruction, multiple data.
- Shared memory vs. distributed memory machines.
  - **Shared memory:** Communication between threads/processes is via loads/stores.
  - **Distributed memory:** Communication between threads/processes is via message passing.
- Key issues when programming shared memory machines:
  - Cache coherence.
  - False sharing.

  Note: Review lecture notes on cache design, available on BBLearn.
- Viability of large-scale parallelization:
  - Amdahl's law.
  - Gustafson's law.
- Writing parallel programs:
  - Reduction (show the program design for few cores and for large numbers of cores such as in a GPU).
  - Histogram generation (use Pacheco).
  - Gauss Seidel (use lecture notes).