Algorithmic Time, Energy, and Power

Introduction

1. Danny Hillis at MIT in 1980s was developing a supercomputer called the "connection machine"
   - Final chapter titled "New Computer Architectures and Their Relationship to Physics, or Why Computer Science is no Good"
   - Parallel algorithms researchers were, at the time, abstracting away too many details about the physical constraints of algorithms
     - Speed of light bounds communication
2. What would it mean to consider physical costs when designing an algorithm?

Speed Trends

1. An Intel Ivy Bridge CPU can, in the best case, execute ~100 billion operations per second
2. Trend: Performance doubles every two years
3. How fast will a processor be in 10 years?
   - 2 ^ 5 * 100 gigaops = 3200 gigaops

Speed Limits

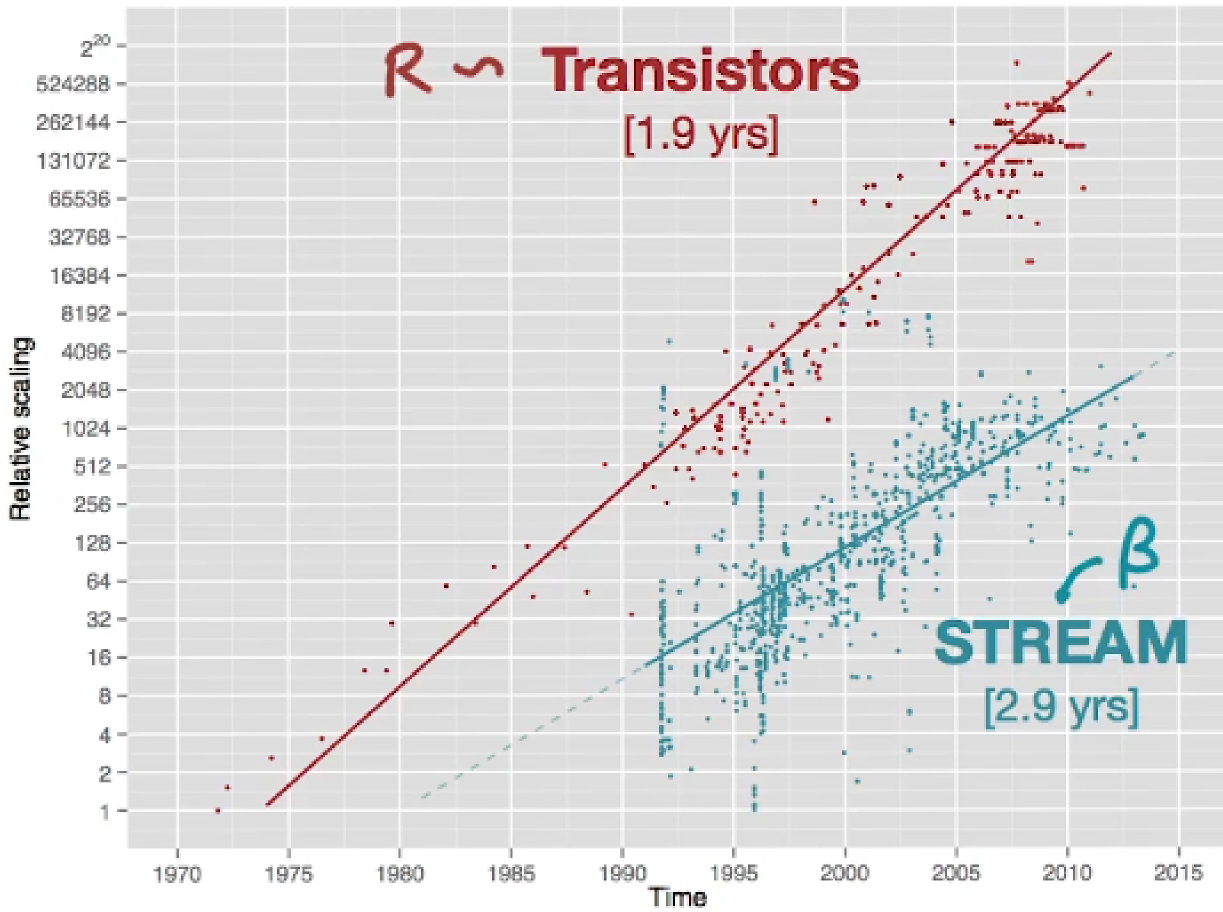
1. Consider a 2D mesh of physical processors (LxL)
   - Every interior point is connected to its 8 nearest neighbors
   - Single operation: Starts at center, travels to a unit at a corner, turns around and returns to the center
   - Want to do this operation 3 trillion times per second
2. How large can L be?
   - Distance for one operation: L * sqrt(2)
   - Total distance traveled: 3e12 * L * sqrt(2)
   - c = 3e8 m/s
   - 3e12 op/s = 1op/1RT * 1RT/(L * sqrt(2)) * 3e8 = 70 microns
     - RT = round trip

Space Limits

1. Consider a chip with area = 4900 microns^2
   - Need to be able to store 1 TB of data
2. What is the physical area of a single bit?
   - 4900 / 8 * 1e12 = 6.125e-10
   - This is on the order of a single Angstrom (size of an atom)
     - At some point, we have to consider locality

Balance in Time

1. Processor can perform R operations per second, which is related to transistor density
   - Doubles roughly every 1.9 years
2. Can move data between fast and slow memory at a rate of beta (units of words per time)
   - Referred to as "stream"
   - Doubles roughly every 2.9 years
3. What is the doubling time of B = R/beta?
   - R(t) = 2 ^ (t/1.9)
   - Beta(t) = 2 ^ (t/2.9)
   - B(t) = 2 ^ (t/1.9) - 2 ^ (t/2.9) - 2 ^ (t/5.5)



Transistor Density and Stream over Time

Balance Principles

1. DAG model of computation characterizes computation by two components
   - Work W = W(n) = total operations
   - Span D = D(n) = total path length (operations)
   - Transactions Q = Q(n; Z,L) <= W
     - Z: Size of fast memory
     - P: Number of processors
     - L: Number of words transferred between slow and fast memory at a time
2. Cost of doing a memory operation is still 1/R, same as a computational instruction
   - If we can parallelize memory accesses, our bandwidth is L/beta, but we must pay a 1/R cost for each of these instructions
3. Tp >= max(D/R, W/PR, QL/beta)
   - Assume W/P » D (Compute time needs to dominate communication time)
   - W/Q >= RPL/beta

Double-Double Toil and Trouble

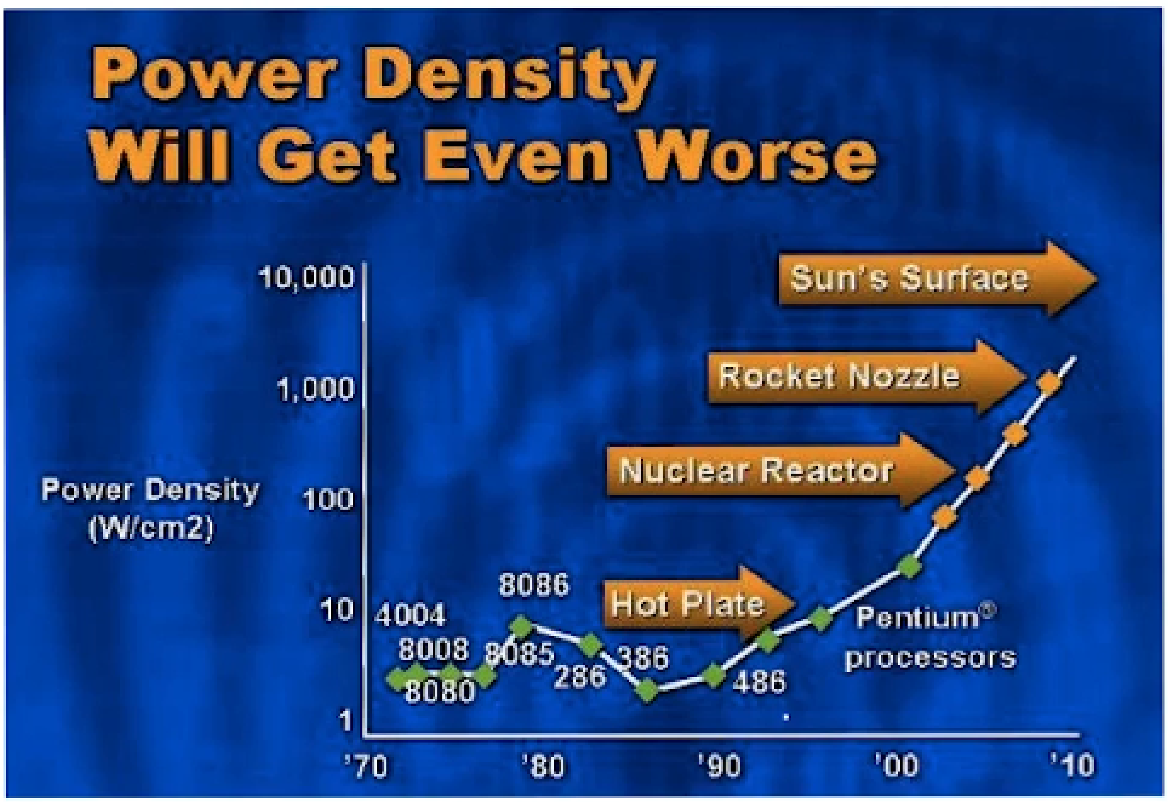
1. Suppose a machine is perfectly balanced for sorting large arrays
2. Boss suggests selling a system with double the cores
3. How can you maintain balance if the number of cores doubles? For sorting, W/Q ~ Llog(Z/L)

- 1/2 bandwidth, 2x peak
- Square Z and square L (true)
- Double fast memory size
- Double bandwidth (true)

4. Llog(Z/L) = 2RPL/beta
    - log(Z/L) = 2RP/beta

Power Limits

1. Power = Energy / Time
2. Increasing clock rate causes power per unit area to skyrocket
3. Power = Po + delta(P)
    - Po = Constant power
    - delta(P) = Dynamic power



Processor Power over Time

The Dynamic Power Equation

1. Energy per gate = C * V^2
    - C = Capacitance
    - V = Supply voltage
2. Clock rate (frequency) = f
    - Max switching frequency
3. Activity factor = a
    - Number of switches per cycle

4. Dynamic power = a * f * CV^2
   - f is proportional to V
     - Important for maintaining reliability of circuit

Power Motivates Parallelism

1. CPU 1 has f = 4 GHz and dynamic power = 64 watts
2. CPU 2 has f = 1 GHz
3. What is the dynamic power of CPU 2?
   - Dynamic power = 64 / 4 ^ (1/3) = 1 Watts because V is proprtional to f
4. What is the relative time to run a program on CPU 2 vs CPU 1?
   - 4 * T1

Power Knobs

1. Which of the factors of the dynamic power equation can be controlled in software?
   - C = Capacitance (false)
     - Geomtric and electrical property of material
   - V = Supply voltage (true)
     - Dynamic voltage and frequency scaling (DVFS)
     - cpufreq in Linux
   - f = Clock frequency (true)
     - Dynamic voltage and frequency scaling (DVFS)
     - cpufreq in Linux
   - a = Activity factor (true)
     - Could turn off chunks of hardware if you knew you didn't need them

Powerless to Choose

1. Consider two systems, A and B
   - Each are characterized by their execution time and energy to do the same computation, E and T
   - Suppose Ea < Eb and Ta > Tb
2. Which system has lower average power?
   - System A because power is energy divided by time

Exploiting DVFS

1. Consider two systems, A and B
   - Eb = 2 * Ea
   - Tb = Ta / 3
2. Suppose you use DVFS to rescale B so that its power matches A. Will B still be faster than A?
   - Yes; B is three times faster but only for twice the energy

Algorithmic Energy

1. Time: Can reduce or hide by overlap (parallelism)
2. Energy: Must pay energy cost for every operation
3. Recall the metrics of the work-span model. Which metric best quantifies energy?
   - Work, W(n) (true)
   - Span, D(n)
   - Average available parallelism W/D
   - Time = max(D, W/P) <= Tp <= D + (W - D)/P
   - Speedup Sp = T1 / Tp
4. Work counts the number of operations

Algorithmic Dynamic Power

1. Recall the metrics of the work-span model. Which metric best expresses dynamic power? Ignore constant power and assume constant energy per operation.
   - Work, W(n)
   - Span, D(n)
   - Average available parallelism W/D
   - Time = max(D, W/P) <= Tp <= D + (W - D)/P
   - Speedup Sp = T1 / Tp (true)
2. Power is energy per time

Parallelism and DVFS

1. Let sigma = frequency slowdown
   - P' = sigma^3 * P
2. If Tp <= D + (W-D)/P, what is the best value of sigma to use?
   - 2 * ((W-D) / (PD)) ^ (1/3)
   - Tp <= sigma * (D + (W-D)/(P * sigma^3)
     - Take the derivative and set it equal to 0, solve for sigma

Conclusion

1. Simple models of computers have been extremely productive
   - CS community produces useful applications without having to think too hard about the physical limitations of machines
2. How do we make the most of the machines we have?
3. Is there a role for physical reality in the design of algorithms and software?
4. How do we do this and be productive developers?