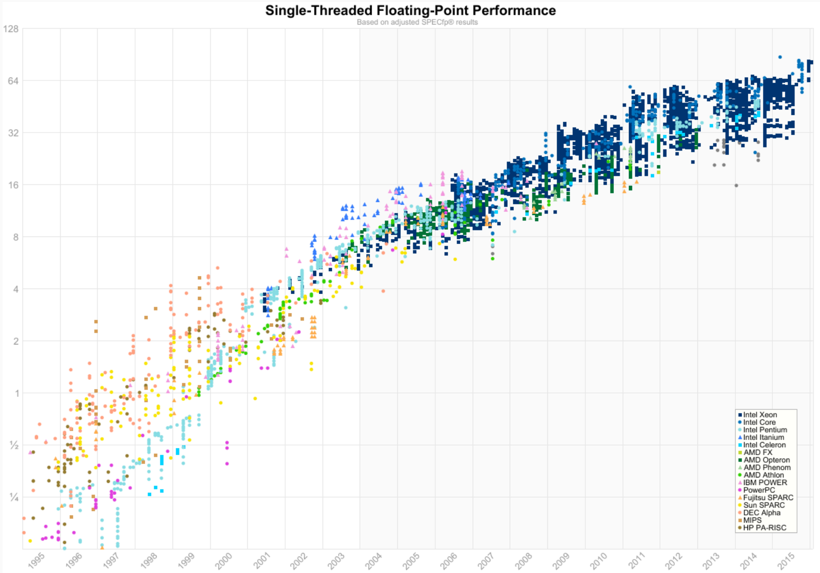
Introduction to parallel computing

Jeremy Iverson

College of Saint Benedict & Saint John's University

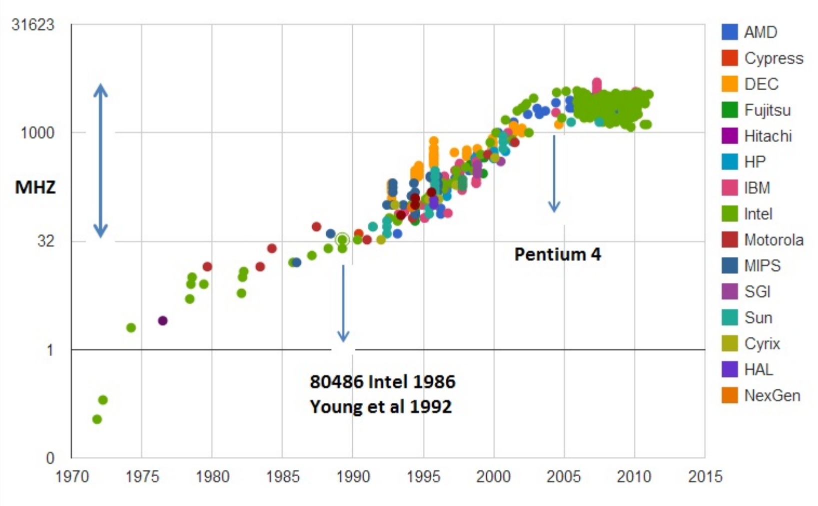
cpu performance



Single-Threaded Floating-point Performance by HenkPoley

- the “almost” current state of things
- what do you notice about this graph?
    - from 1986 – 2002, microprocessor performance increased by roughly 50% every two years, since then it has dropped to about 20%

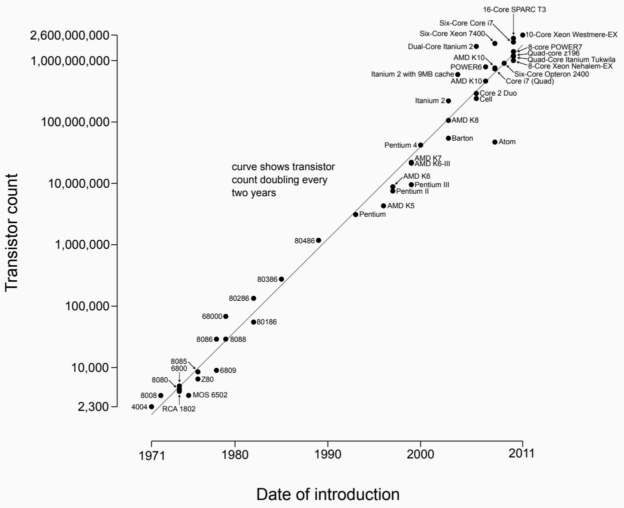
clock speed



Clock CPU Scaling.jpg by Newhorizons msk / CC0 1.0

- do you notice anything odd about this graph, given the previous graph?
  - we are seeing ~20% performance improvement despite stagnating clock speed
- why does clock speed stop increasing?
  - faster processor = increased power consumption.
  - increased power consumption = increased heat.
  - increased heat = unreliable processors.
- we are reaching / have reached the limit that current cooling technologies can remove heat from the chip (a.k.a. the power wall)
- so how does performance improve, when clock speed plateaus?

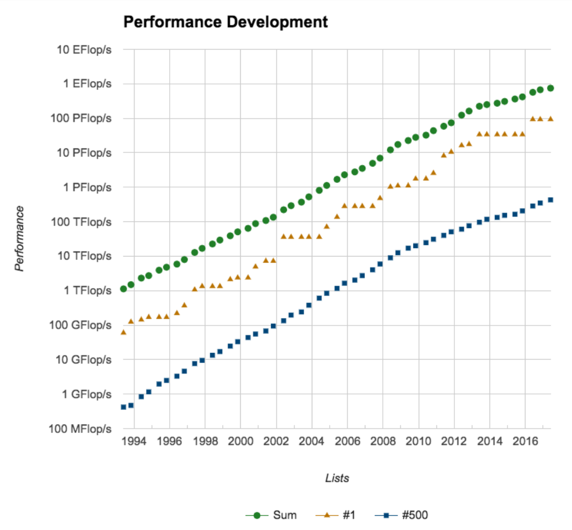
transistor count



Transistor Count and Moore's Law - 2011 by Wgsimon / CC BY 3.0 / cropped from original

- pre 2005 — a golden era when clock speed could be increased while adding more transistors, which resulted in the large performance improvements in the first graph
- post 2005 — performance improvements now have to come from more advanced chip capabilities
- we don't speed up the processor, we make it more sophisticated
  - larger data busses
  - larger caches
  - deeper instruction pipelines (ILP)
- Moore's law (c. 1965) — transistor count roughly doubles every two years
- so, how can we keep continue to improve performance?

supercomputer performance



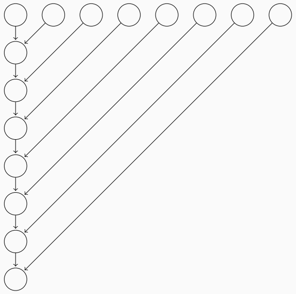
Graph courtesy of The Top 500 List.

- so how does any of this affect us, in this class?
- notice anything about this graph?
  - performance improvement is exponential
  - how can that be?
    - we are no longer constrained by the limitations of a single chip
    - so what are our limitations?

what are the challenges of parallel computing?

- how do we take advantage of available parallel resources, i.e., how to convert serial programs to parallel programs
  - automatic translation has been attempted and is still being studied, but success has been limited

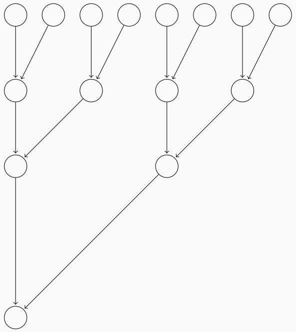
counting



- how many steps, described as a function of the number of nodes?
- where are there dependencies / why can't this be parallelized?

```
for (int i=0; i<n; ++i) {
  a[0] += a[i];
}
```

counting cont.



- how many steps, described as a function of the number of nodes?
- what happens if the number of nodes is not a power of two?
- what would the code look like to accomplish this (what do you need to express, that you do not know how)?