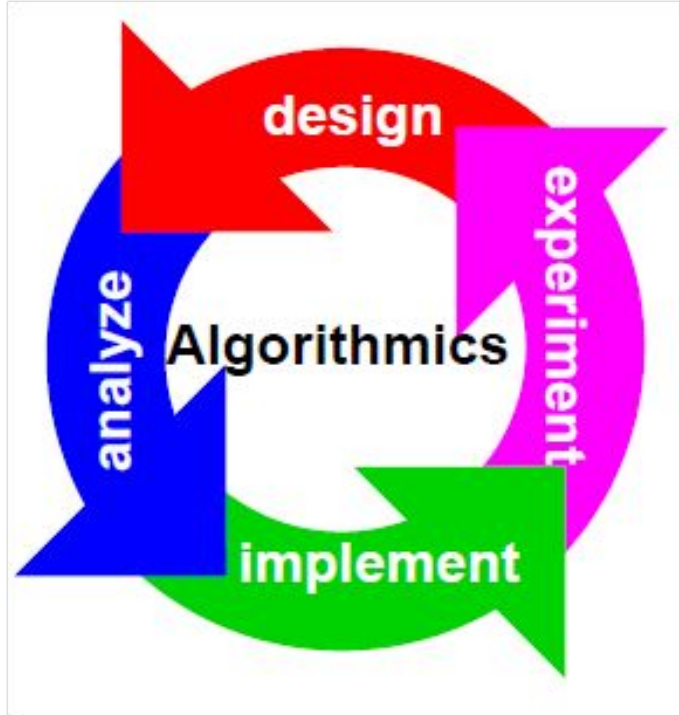


Motivation

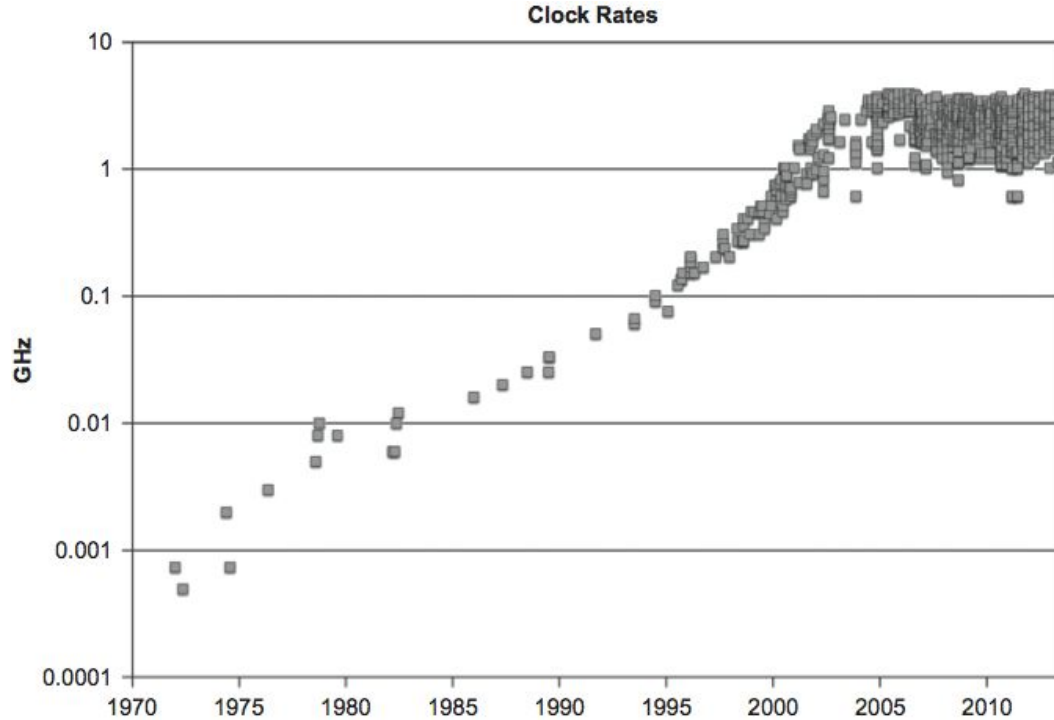
- Algorithm theory focuses on **worst-case behavior** (Big-O notation)
- Leads to discrepancies between theory and practice
Merge-Sort $O(n \cdot \log n)$ vs. Quicksort $O(n^2)$
Linked-List vs. Dynamic Array
- Algorithm Engineering strives for the unification of **theory and practice**

Motivation

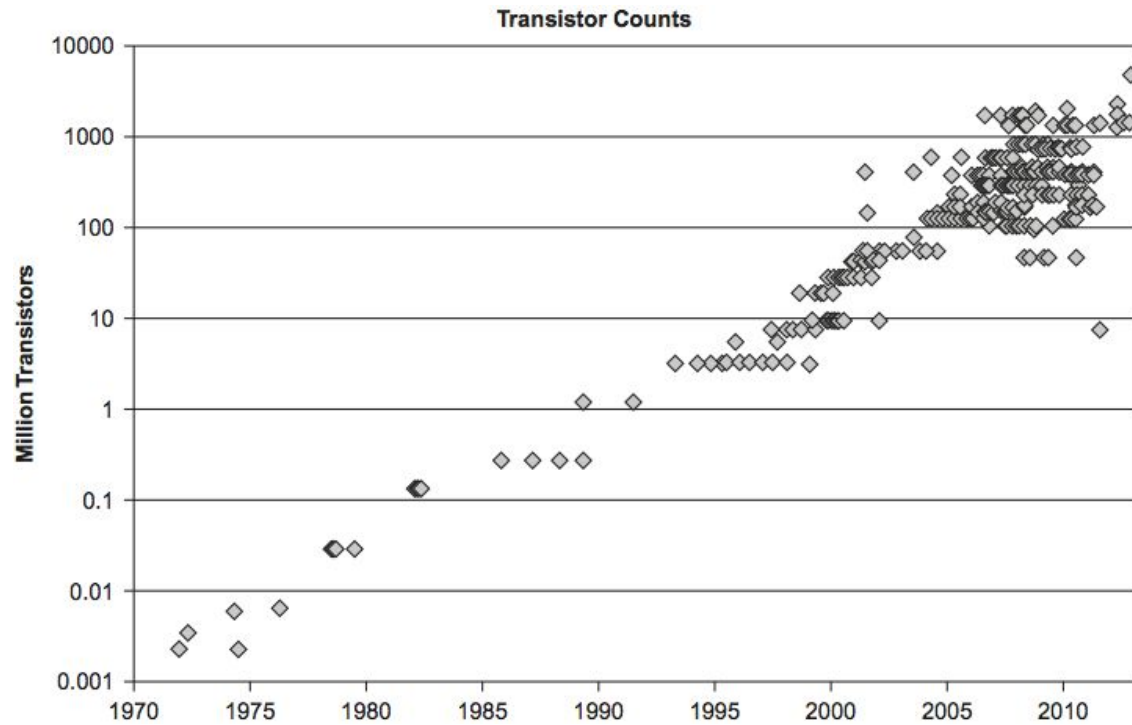


- Design
simple and implementable
- Analysis
heuristics and randomization
- Implementation
respect hardware features (caches, vector units, ...)
- Experimentation
Profiling and tuning on realistic input

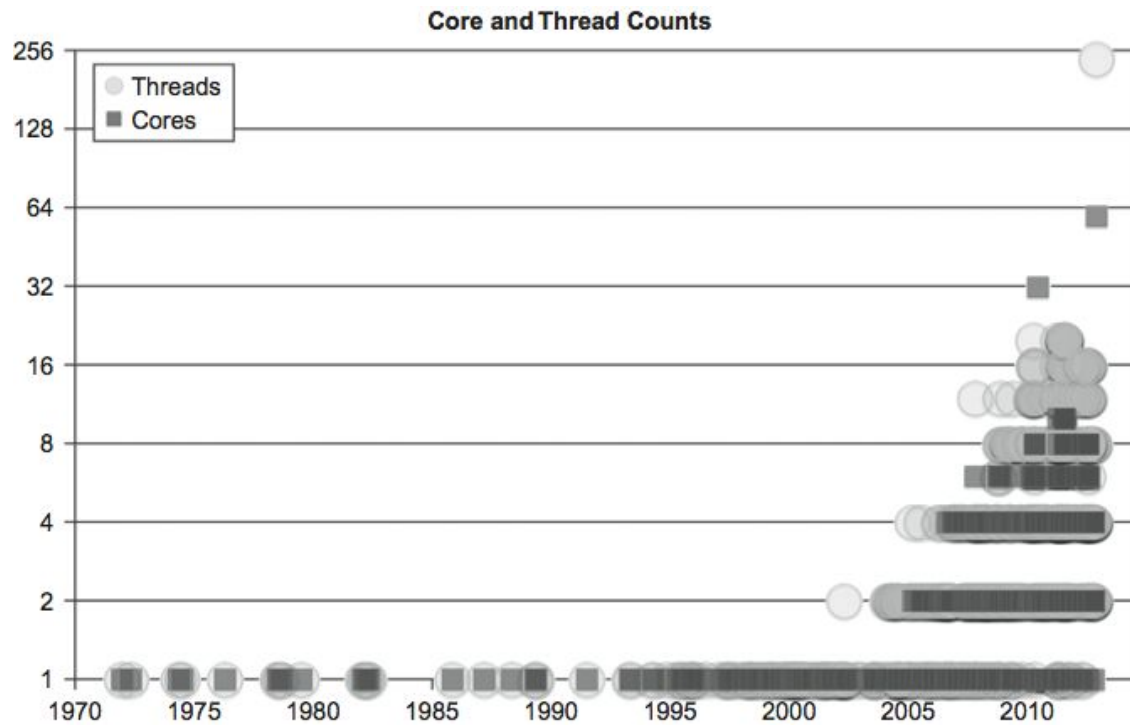
So what does a *realistic* environment look like?



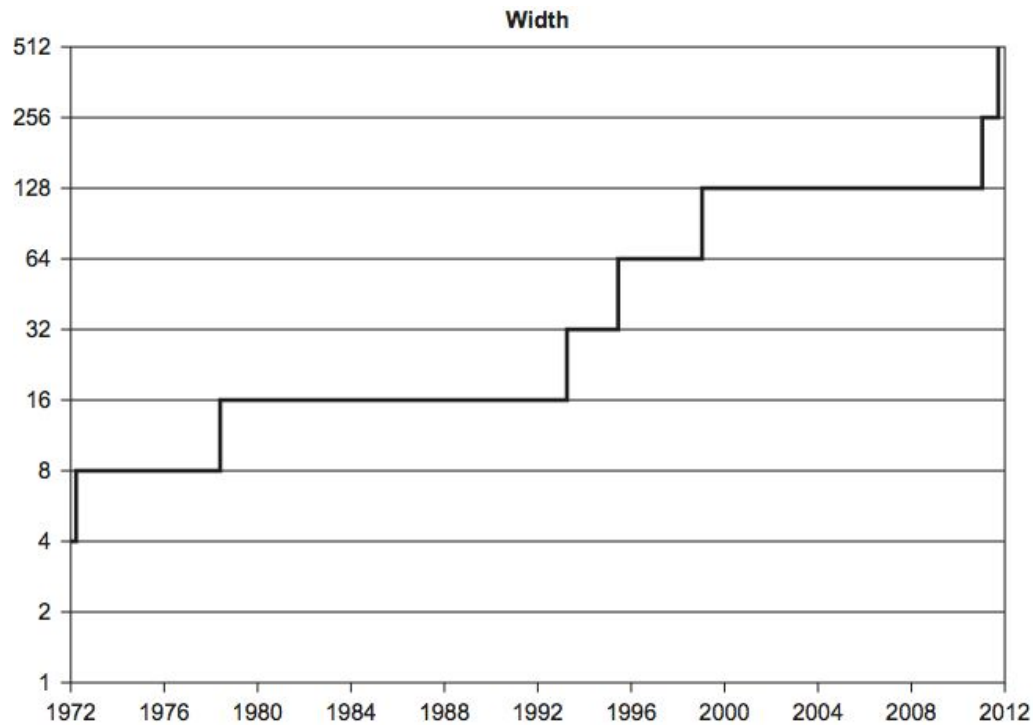
Moore's Law



More Cores



Wider Vector Units



Consequences for Developers

- free lunch is over / disruptive
 - software used to speed up with clock
 - to make free lunch work again, software now needs to scale
- Efficient Cache Utilization
restructure software to more predictable access patterns
- Vectorization
use vector units for SIMD work

This Course: more Parallelism

- 2 CPUs
- 8 Cores, 16 Threads
- 256 GB RAM (NUMA)
- 4 Xeon Phi Cards
- 60 cores, 240 Threads
- 8GB RAM
- $(4.04 + 0.2)$ TFlop/s



Course outlook

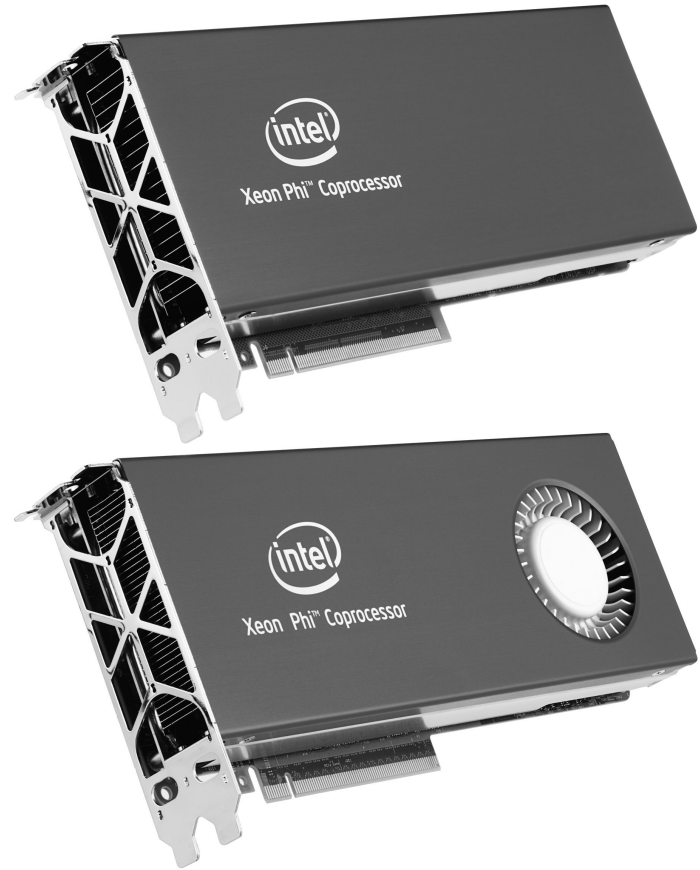
- Programming of highly parallel machines
 - OpenMP
 - MPI
 - threading libraries (TBB)
 - (OpenCL)
- Profiling and Debugging
- Performance Optimizations
- Applications

History

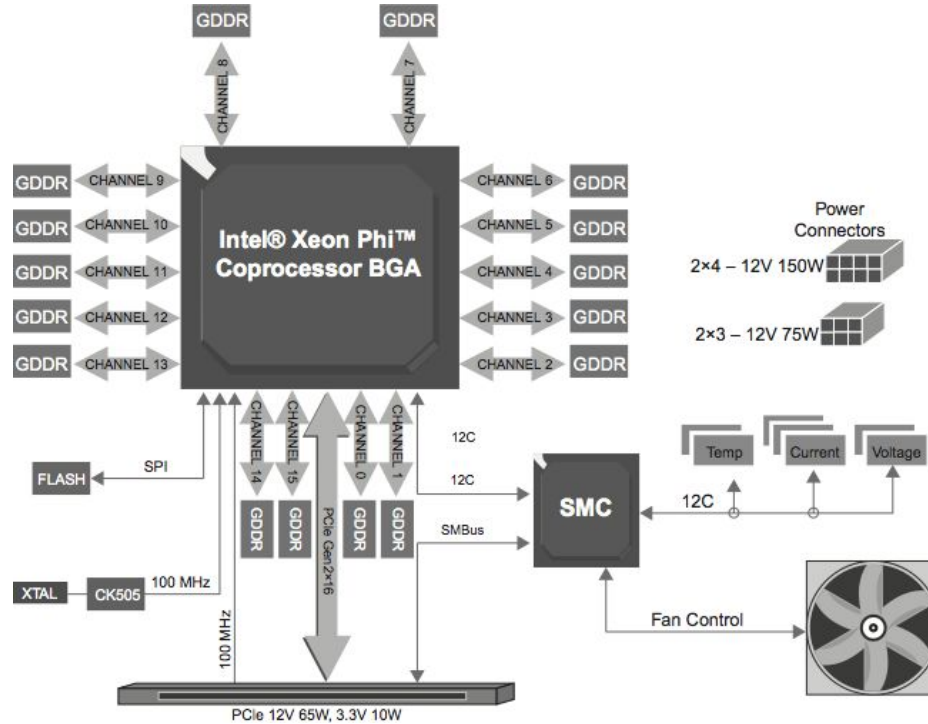
- 2008 Intel announced GPGPU
- Larrabee - was never produced
- 2010 MIC architecture announced
Many Integrated Core
- 2012 First Cards shipped under the name
Xeon Phi (Knights Corner)
- 2015 Next MIC Generation
Xeon Phi (Knights Landing) 3 TFlop

Xeon Phi Architecture

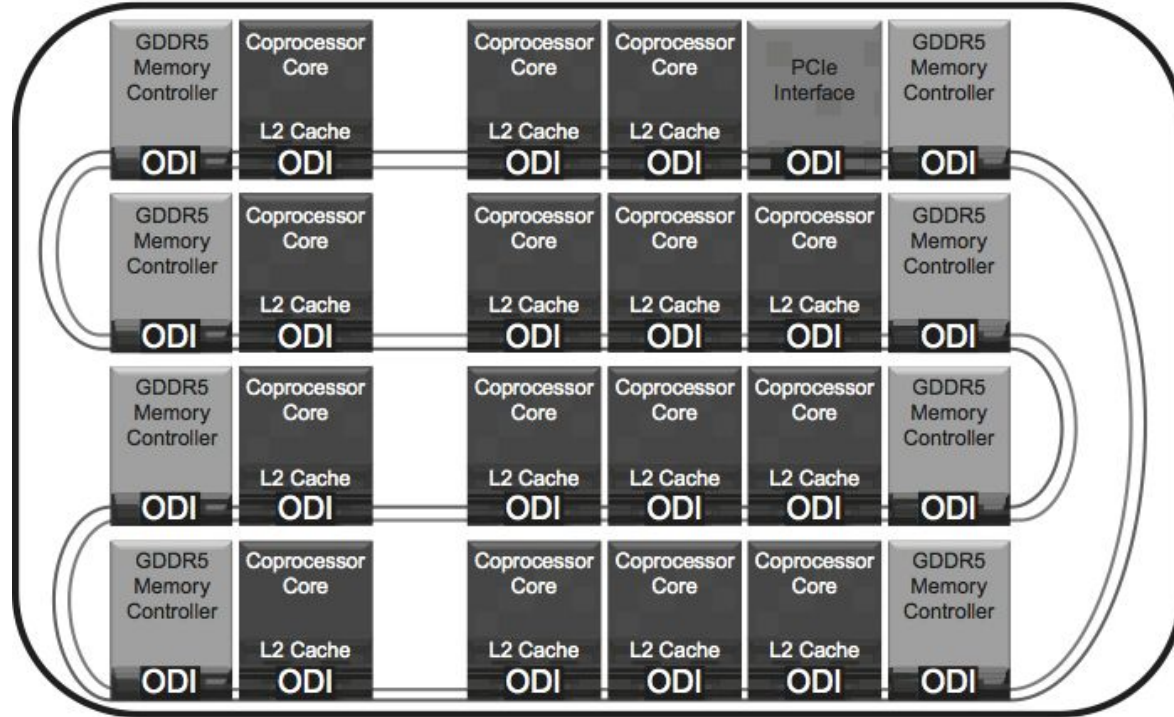
- accelerator card connected via the PCIe bus
- „motherboard“ for a computer with 60 cores and 8 GB RAM
- flash memory with boot routine that boots a small linux system from the host System
- emulates a network interface over PCIe



Xeon Phi Architecture

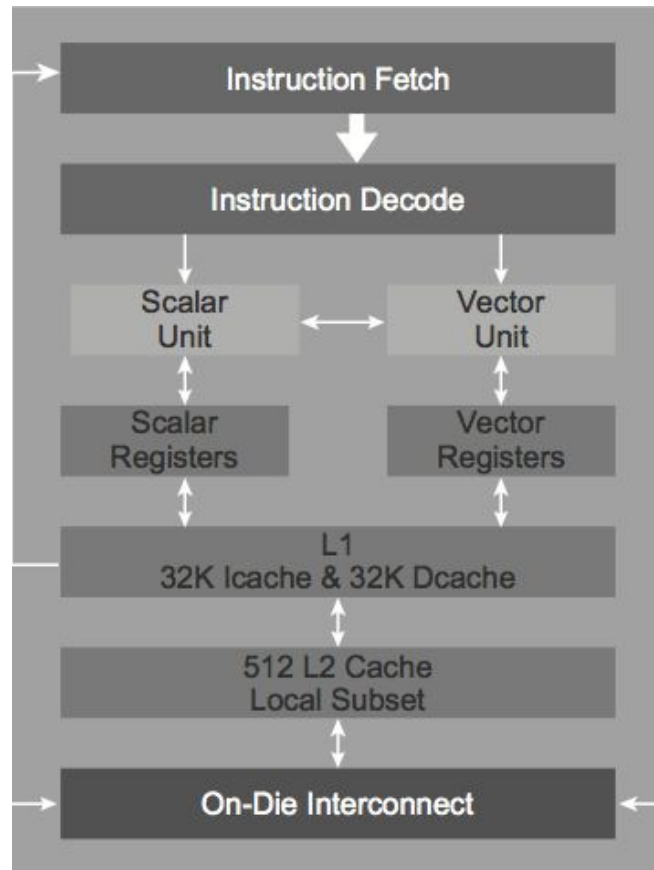


Chip Layout



Core Structure

- 64-bit architecture
- in-order execution
- favors multithreading over complex core
- instruction decode has latency of 2 cycles! →
Single threaded (compute-bound) code will only reach 50% peak! 512-bit VPU



Hardware Threads

- Every core has replicated state for 4 hardware Threads
- Similar to Hyperthreading(HT) on regular x86 Cores
- Difference: while HT might improve performance by about 5-10% (sometimes performance even drops), Hardware-Threads on the Xeon Phi are **always** beneficial! (use at least 2!)
- scheduled in **round-robin** fashion

Cache organization

- 31kb L1-data and 31kb L1-Instruction Cache
- 512kb L2 (last-level) cache
- **fully coherent** among the cores
- total cache of Xeon Phi Die: 30 MB
 - software may use everything between 512kb and 30MB as cache memory
- L1 latency: 1 cycle, L2 latency: 11 cycles
- Translation-Lookaside-Buffer (TLB) supports **huge pages** of 2MB

Prefetching

- a request to the Cache/Memory-Access Subsystem
- look ahead and bring the data that is expected to be used into L1/L2 cache for the near future
- fetching memory is slow → **hide latency!**
- hardware prefetching & software prefetching
- software prefetching either manually or automatically by the compiler
- can be scheduled simultaneously to VPU instruction

Vector Processing Unit

- Each core has a VPU of 512-bit width
- similar to SSE/AVX but not backwards compatible
- can process 16 single-/8 double-precision floats in a single instruction
- at least as important to scaling as scaling to Threads (SIMD workload)
- Special support for transcendental functions, **fused-multiply-add** (FMA)
 $c += a * b$

Performance Metrics

- Floating Point Operations / second
Throughput: 1 **retired** operation / cycle
1.053 GHz: 1053000000 instr./second
8-unit wide vector instructions
2 Flops per unit (FMA)
60 cores
→ $1.053 * 10^9 * 8 * 2 * 60 = 1011 \text{ GFlop/s}$
- Memory Bandwidth
8 Memory Controllers
GDDR5 -> dual Channel, 4 Bytes per Transfer
5.0 GT/s ($5 * 10^9$ transfers per second)
-> 320 GB/s (**realistic ~60% of this peak**)

Reality of Parallelization

- in an ideal world **n-way** multiprocessor provides an n-fold of computational power
- in practice this almost never happens
- often parallelization comes with **communication and coordination** costs

Example

- Five friends want to paint a five-room house
- all rooms are the same size and everyone paints at about the same rate
→ low coordination/communication: 5x speedup
- What if the rooms are of different size?
Same example, but: one room is twice as big as the others
→ dominates the overall time
- adding more painters incurs coordination/communication cost

Amdahl's Law

- the limit to which we can speed up any complex job is determined by the size of its sequential part
- Speedup $S_n = \frac{\text{time taken on a single processor}}{\text{time taken on } n \text{ processors}}$
- Amdahl's law characterizes the maximum speedup given n processors and a parallel fraction p

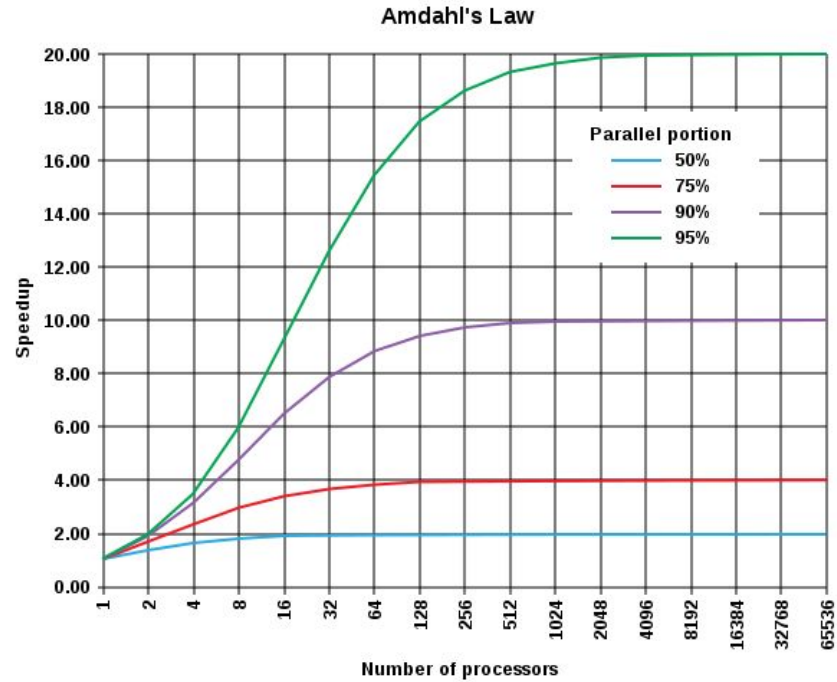
Amdahl's Law

$$S_{n,p} = 1 / (1 - p + p / n)$$

Same example: 10 painters, 10 rooms, one room is twice the size of the others:

$$S_{10,10/11} = 1 / (1/11 + 1/11) = 5.5$$

Amdahls Law



Conclusion

- there will always be a **serial part** of our application
- with the right **tools and engineering principles** we can minimize this serial part
- the eventual **partition** between serial and parallel regions as well as the **hardware setup** at hand will guide the **programming models** we choose