Scaling Computers:

Why Streaming Is Interesting

Mark Horowitz
Bill Dally, Christos Kozyrakis, Kunle Olukotun
Computer Systems Laboratory
Stanford University
horowitz@stanford.edu

Processor Model

- User model has been very stable for 30 years
  - Sequentially executes instructions
  - IO operations interact with outside world

- Model has hidden the scaling of technology
  - Efficiently transformed transistors to performance
  - 8008 – 3,500 transistors, and ran at 200kHz
  - P4 – 42M transistors, runs at 3GHz
  - Performance changed from 0.06MIPS to >1000MIPS

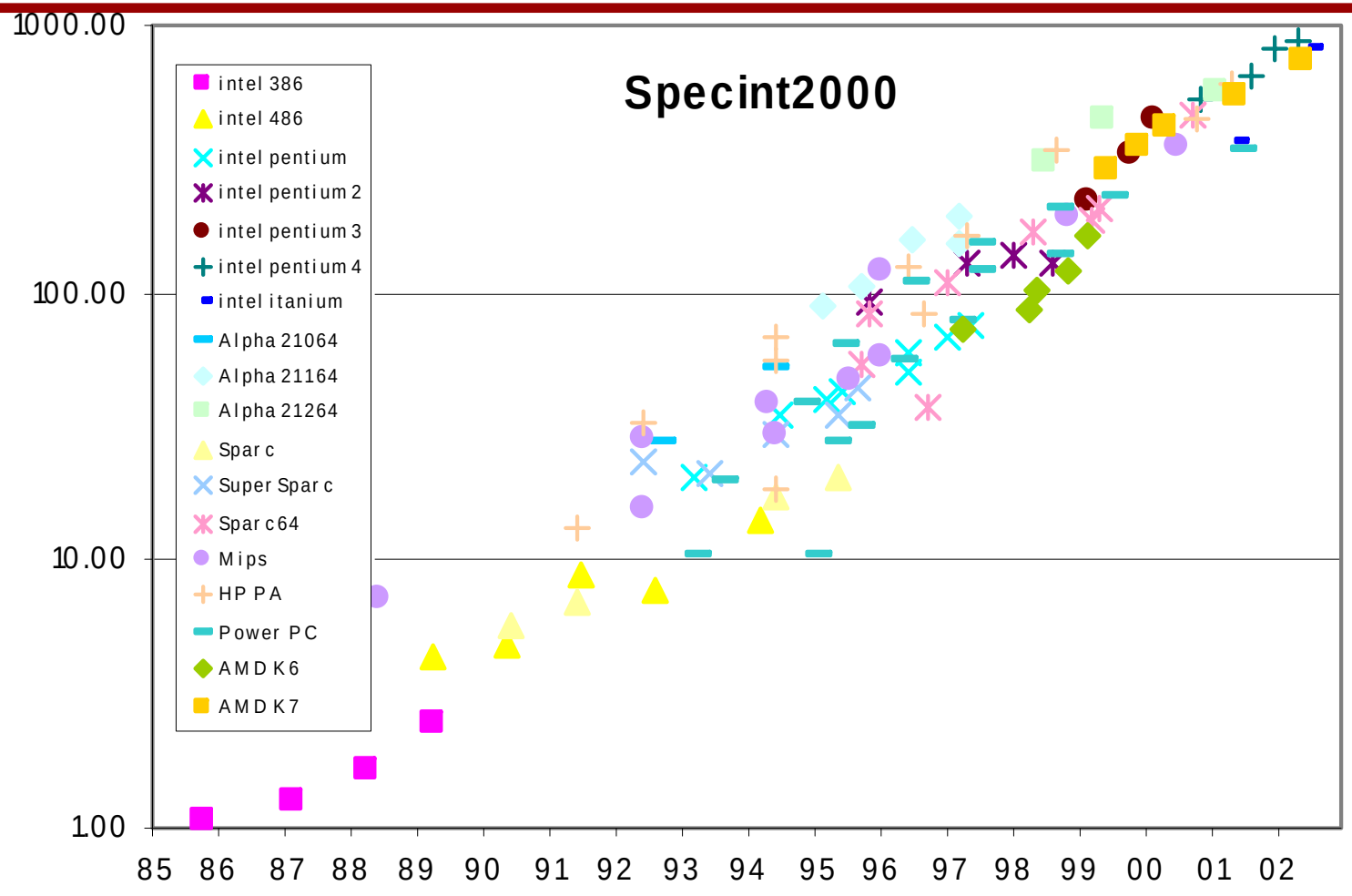
- C is a perfect fit to this programming model
  - They grew up together …

Why Talk About Anything Else?

- People hate change
  - Programmers especially
- Most applications are written for this computation model
  - It is also is what the cheapest machines execute
- It is hard enough to make things work in this model
  - Parallel programs are much more complex
  - Need to focus on increasing productivity of programmers
- Hardware is always getting faster
  - Wait 18 months and the machine will be 2x performance

- DARPA funding is not a good answer
  - IMHO

Supporting Data



The World Is About To Change

- Processor performance will not continue to scale
  - We will fall off the current performance curve soon
- Many factors will cause this to occur
  - VLSI wire issues (global structure are hard to build)
  - Insufficient recoverable ILP
  - Power

- This performance growth was partially an illusion

Technology Scaling

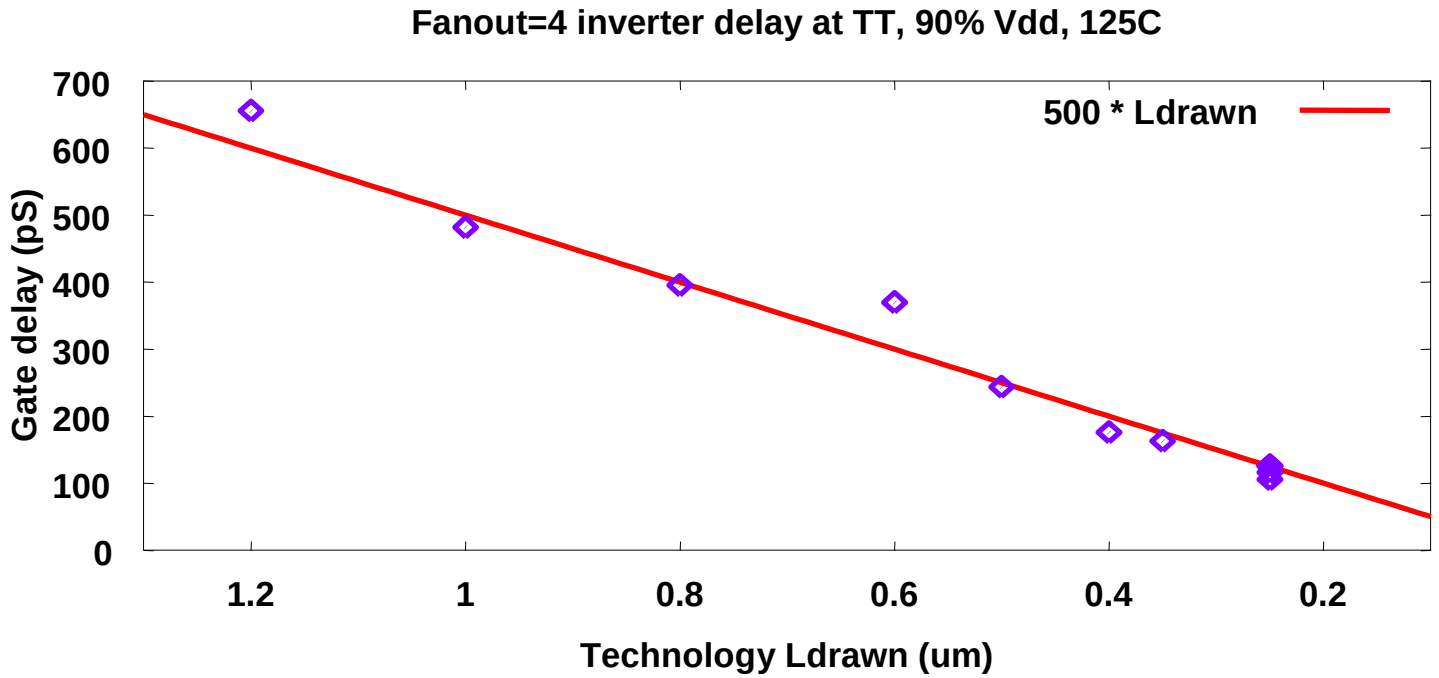
Scaling CMOS has two direct effects:

- Devices get smaller
  - Both transistors and wires
  - Get more per square mm
  - Generally means they get cheaper
  - Enables more complex devices
- Transistors get faster
  - So do wires when viewed the ‘right’ way

FO4 Inverter Delay Under Scaling

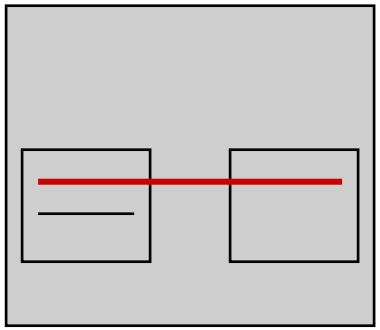
Gate delay varies linearly with process technology (so far)

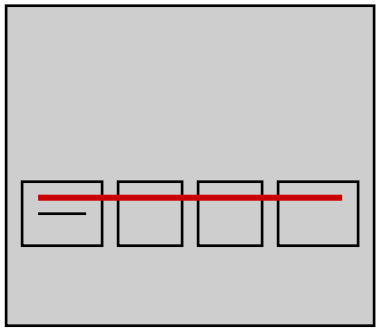
- Useful rule of thumb: D_{gate} = 500pS*L_{drawn} at TTLH



- Issues with being able to continue this scaling
  - Some current technologies are faster (le < feature size)

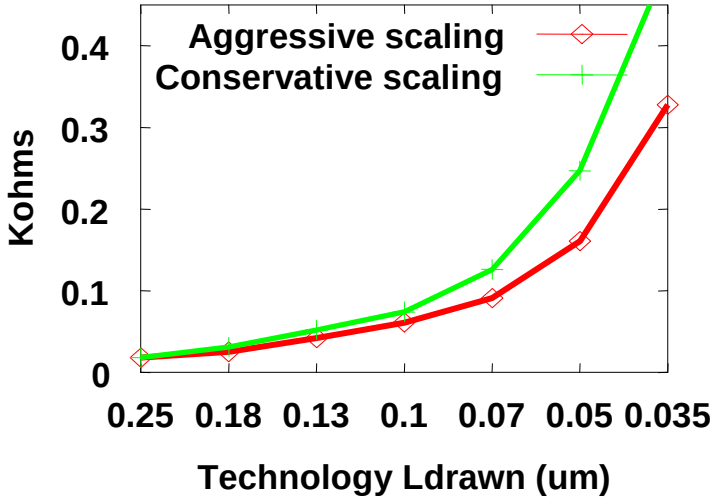
Fixed Length Wire Scaling



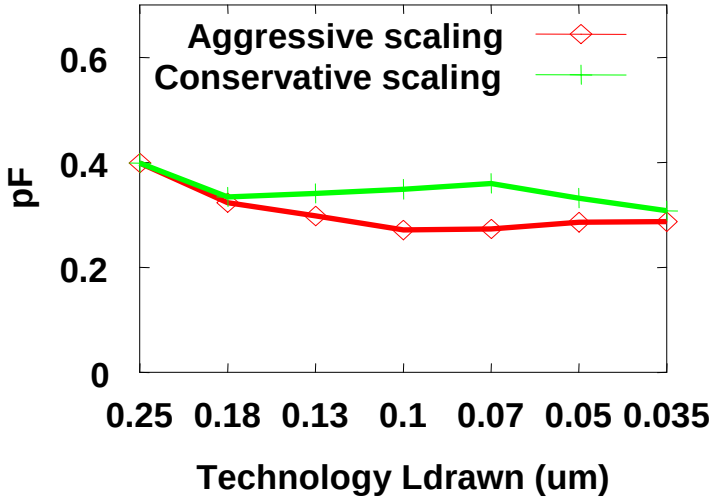


- R gets quite a bit worse with scaling; C basically constant

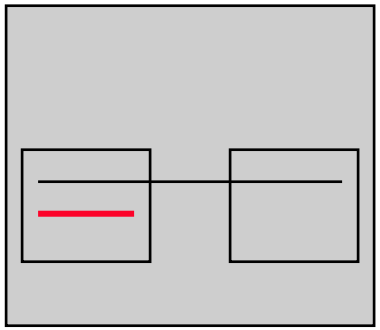
Semi-global wire resistance, 1mm long

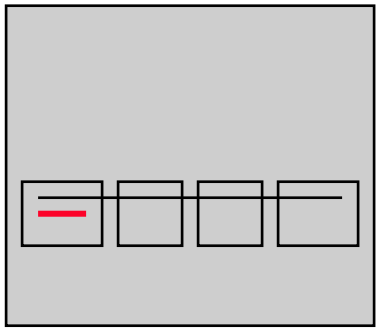


Semi-global wire capacitance, 1mm long



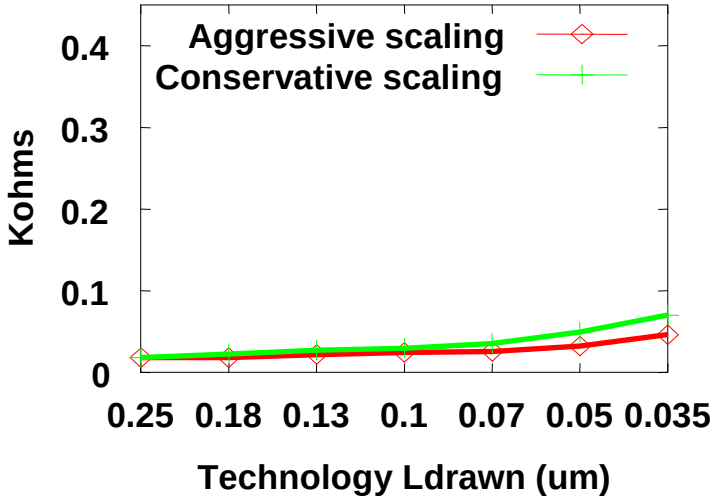
Module Level Wire Scaling



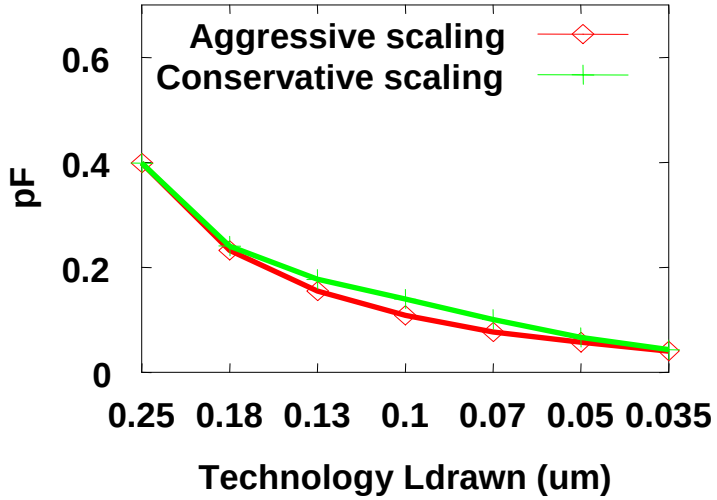


- R is basically constant, and C falls linearly with scaling

Semi-global wire resistance, scaled length

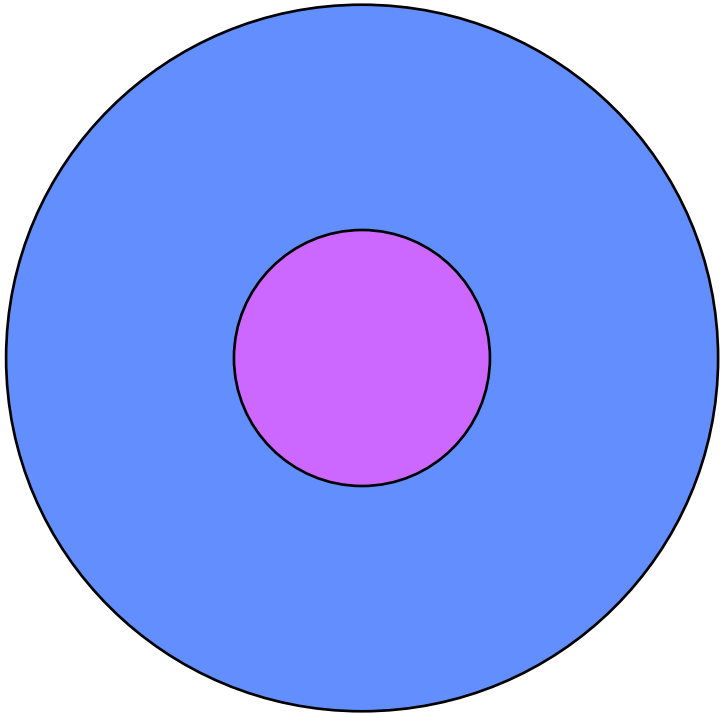


Semi-global wire capacitance, scaled length



The World is Growing

- The problem associated with wires is really due to complexity
- Diagram shows the logical span you reach in a cycle
  - It also show the logical span of a chip



Old view: a chip looks small to a wire

Logical chip size

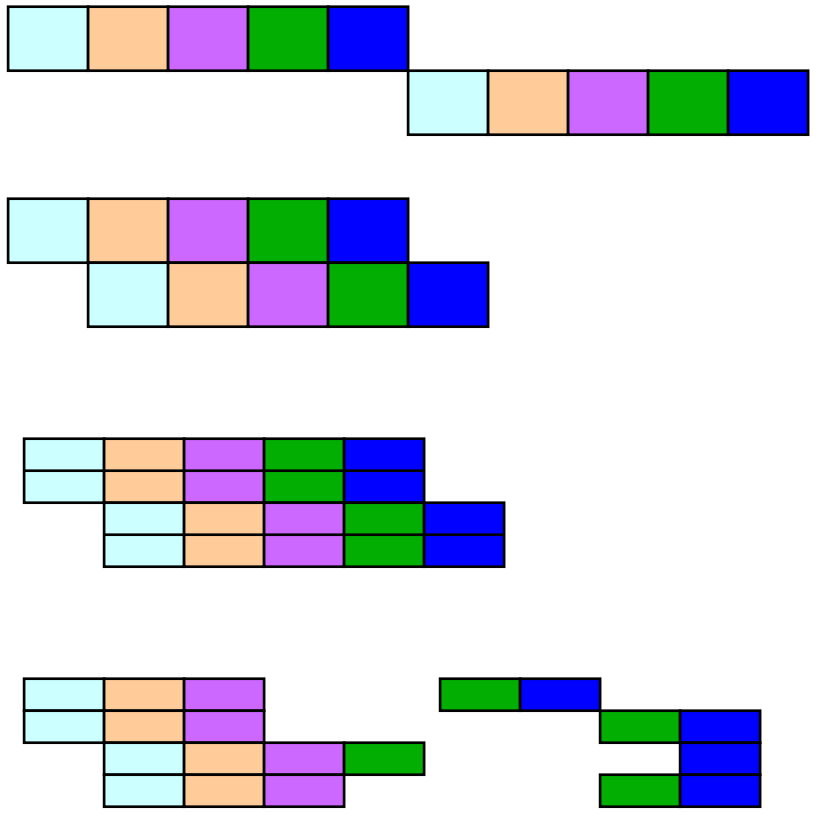
Distance I can go in 1 cycle

New view: a chip looks really big to a wire

Communication is expensive, even on-chip

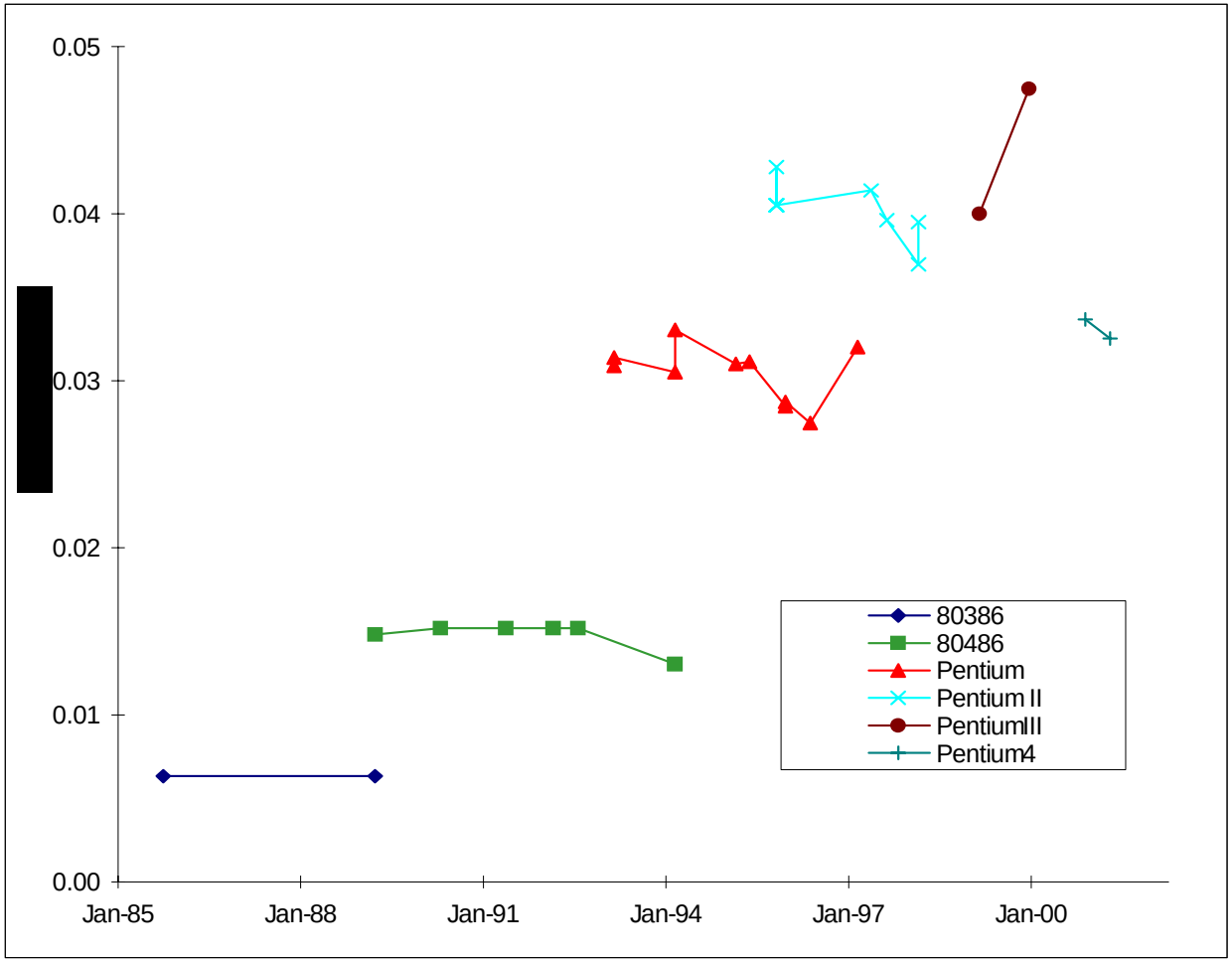
Architecture

- Convert transistors to performance
- Use transistors to
  - Exploit parallelism
  - Or create it (speculate)
- Processor generations
  - Simple machine
    - Reuse hardware
  - Pipelined
    - Separate hardware for each stage
  - Super-scalar
    - Multiple port mems, function units
  - Out-of-order
    - Mega-ports, complex scheduling
  - Speculation
- Each design has more logic to accomplish same task (but faster)

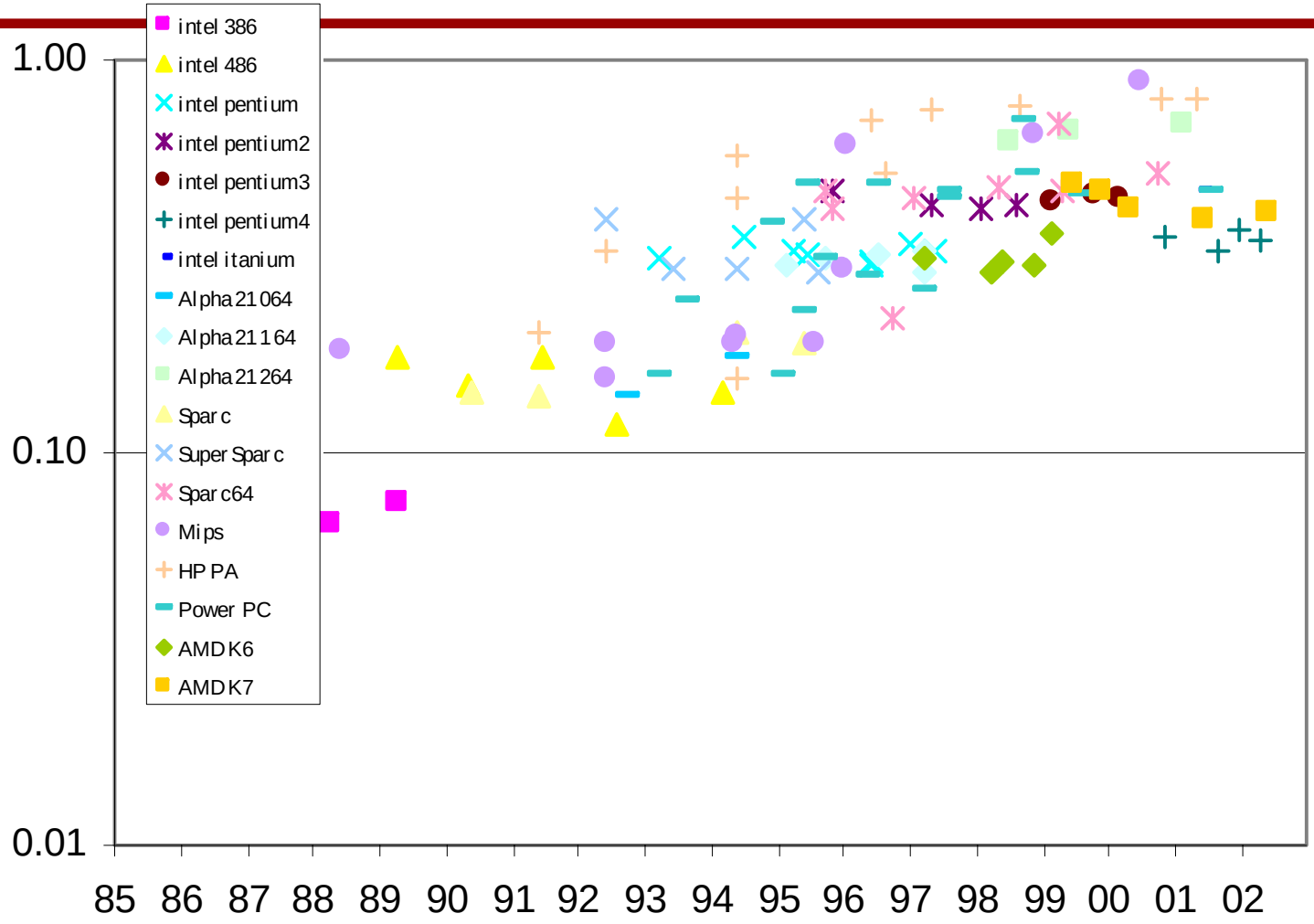


Architecture Scaling

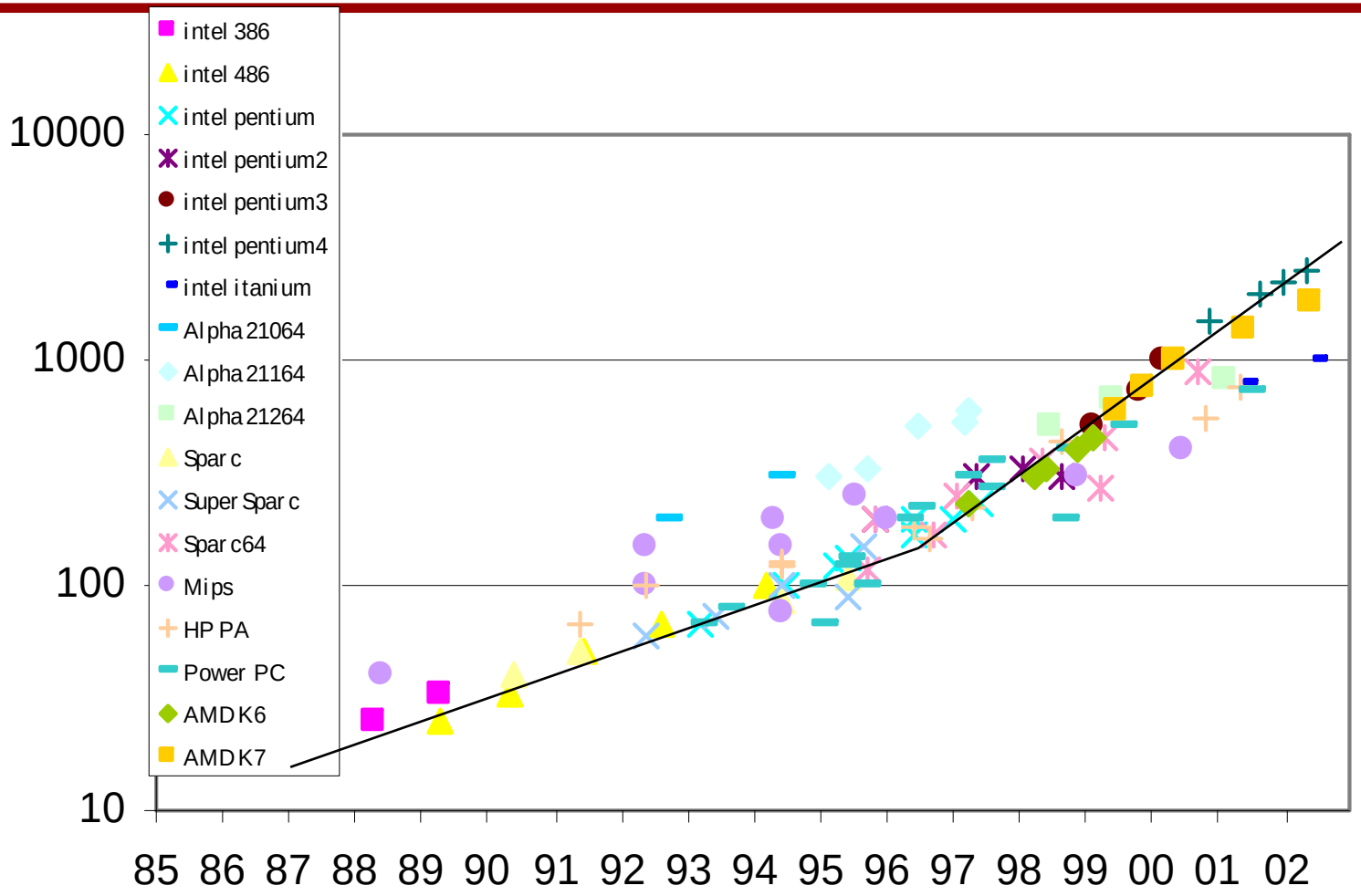
- Plot of IPC
  - Compiler + IPC
  - 1.5x / generation
  - Until PIII, now falling
- There is a lot of hardware to make this happen
  - Many transistors
  - Lots of power
  - Lots of designers



SpecInt/MHz

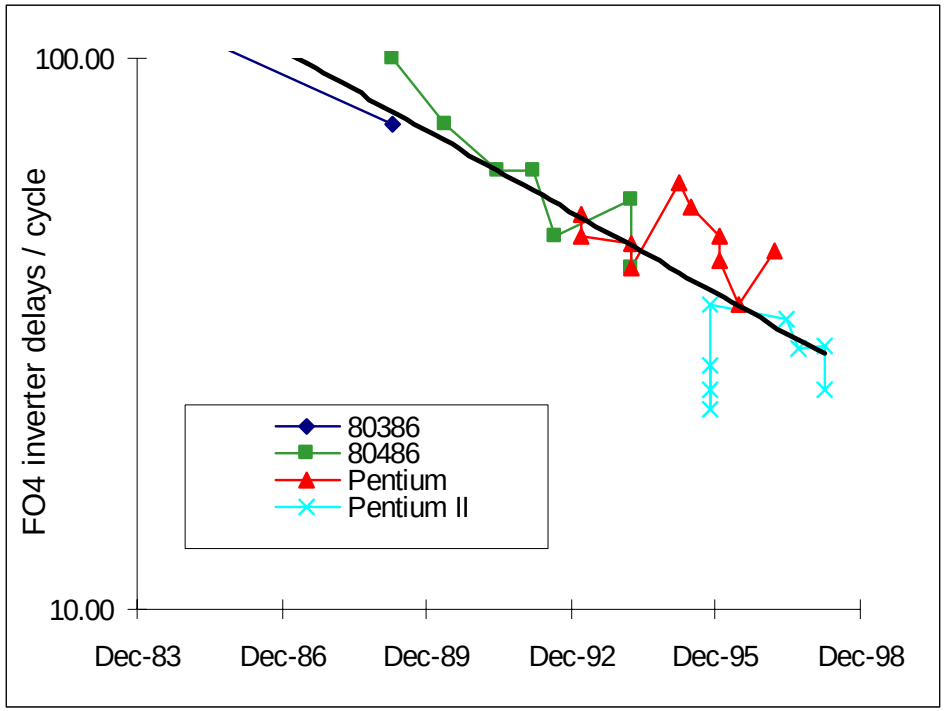

intel 386
intel 486
intel pentium
intel pentium2
intel pentium3
intel pentium4
intel itanium
Alpha 21064
Alpha 21164
Alpha 21264
Sparc
SuperSparc
Sparc64
Mips
HP PA
Power PC
AMD K6
AMD K7
1.00
0.10
0.01
85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 00 01 02

Clock Frequency Scaling



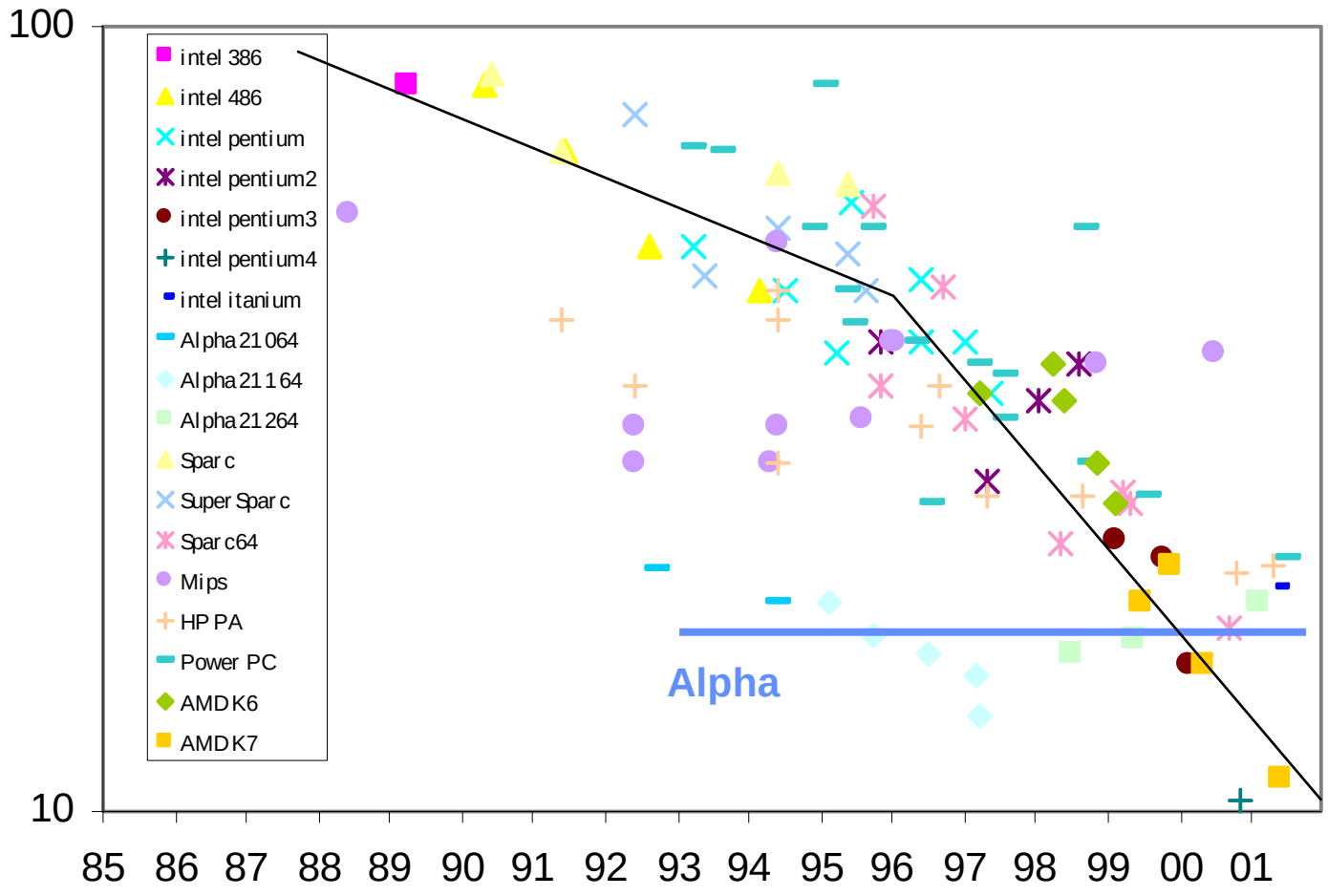
Gates Per Clock

- Clock speed has been scaling faster than base technology
- Number of FO4 delays in a cycle has been falling



- Number of gates decrease 1.4x each generation
- Caused by:
  - Faster circuit families (dynamic logic)
  - Better optimization
  - Better micro-architecture
  - Better adder/mem arch
- All this generally requires more transistors

Clock Cycle in ‘FO4’



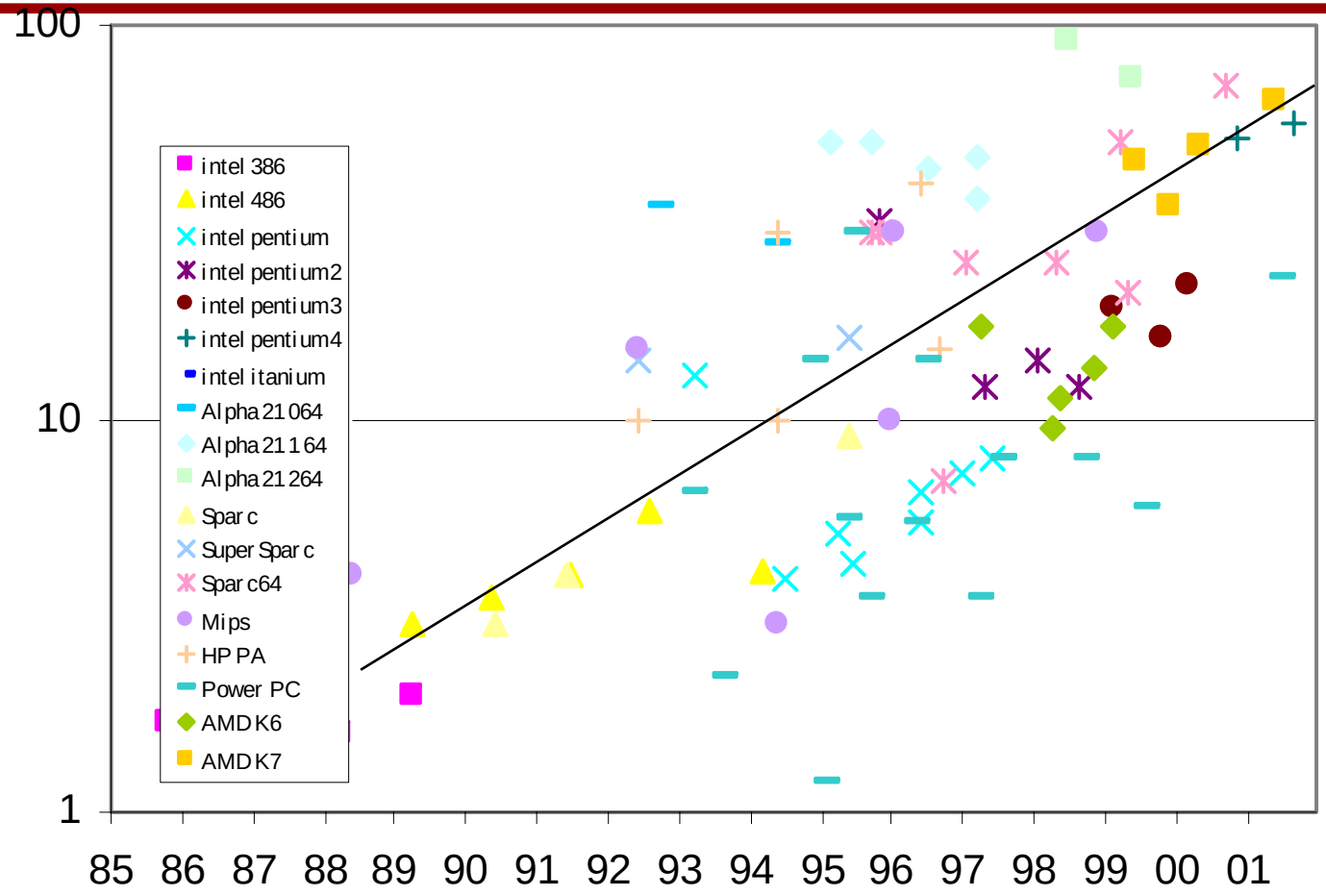
Note that the points at around 10 FO4 are not correct. The FO4 for these technologies is about ½ my simple formula

Gates Per Clock

- Current SOA machines are at 16 FO4 gates per cycle
  - Historical low values (Cray) were at this level
- Overhead for short tick machines grows rapidly
  - Power
    - Increases clock power per logic function
  - Latency
    - Flops are already 10-20% of cycle today
  - Logic reach grows smaller
    - What fits in a cycle (how many bits/gates) decreases
- Difficult to generate a clock at less than 8 FO4 gates

- Continued scaling of gates/clock will be hard
  - Performance gain from 16 FO4 to 8 FO4 is only 20% anyhow

Power


100
10
1
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
00
01
intel 386
intel 486
intel pentium
intel pentium2
intel pentium3
intel pentium4
intel itanium
Alpha 21064
Alpha 21164
Alpha 21264
Sparc
SuperSparc
Sparc64
Mips
HP PA
Power PC
AMD K6
AMD K7

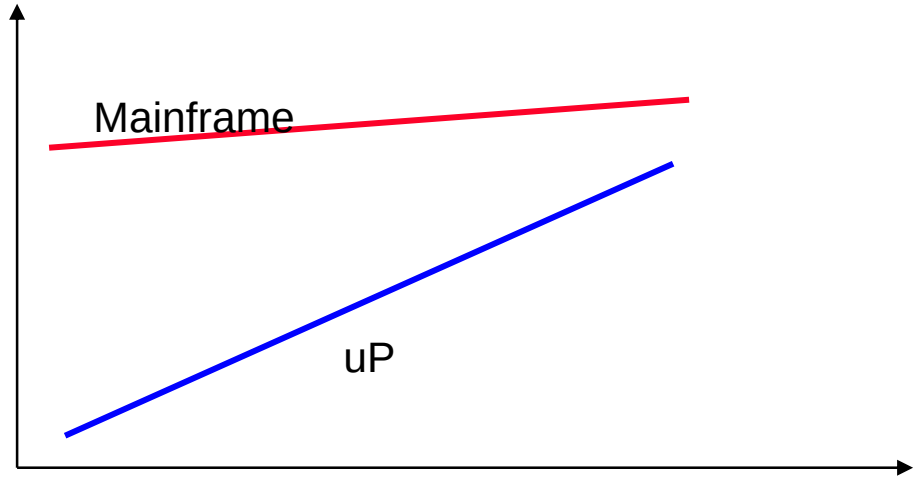
Complexity and Power – The Dark Side

Two other factor will limit conventional scalar processors:

- Power
  - Current SOA designs are power limited
  - If you put all the techniques you know about
    - It will dissipate too much power
    - Need to get the most performance for XX Watts
  - Performance improvements must be energy efficient
- Complexity
  - Very few companies can afford the design costs
    - Fewer players every year

The Illusion:

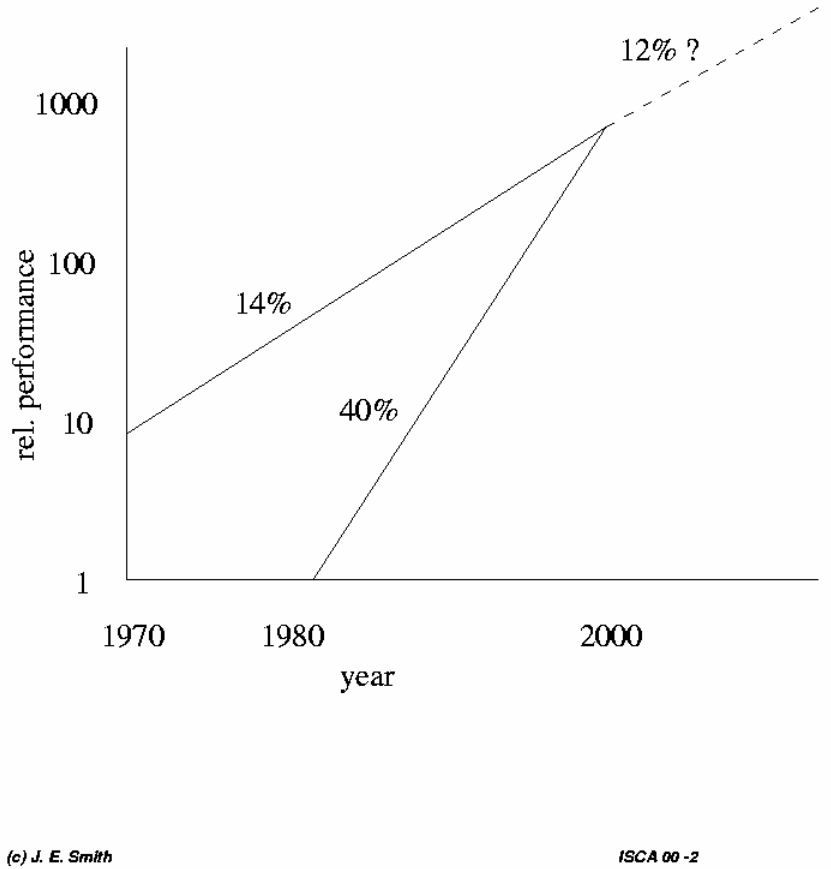
- I remember processor performance plots used to have two lines



- Microprocessors and mainframes
  - Mainframes had maxed out and improved at technology rate
    - 2x every 6 years
  - Unless something changes micros will be there soon

Jim Smith's Graph

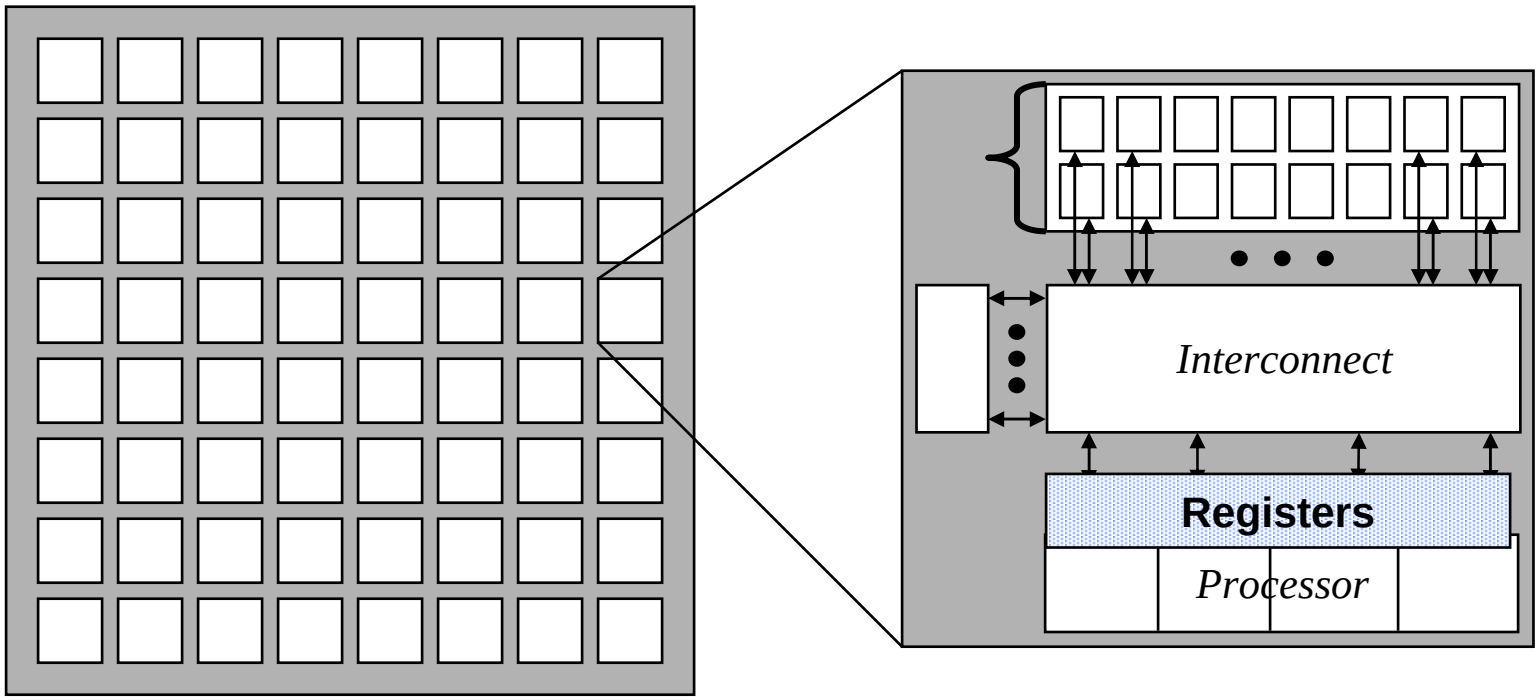
- Graph Jim Smith presented at ISCA 2000
  - Fastest uniprocessor
  - Then growth has been modest
  - Micros were starting from a slow point, so have a large growth rate



Coming Opportunity

- Conventional processor scaling is going to slow down
  - Design costs are enormous
  - Improving IPC is getting harder
  - Improving cycle time is getting harder
- For performance need to exploit parallelism
  - EV8, Pentium 4 – SMT
  - Power 4 is an explicit multiprocessor
    - Power 5 is explicit multiprocessor with SMT
- How do we do this well?
  - Create other programming models
  - Make the models match VLSI constraints
  - Don’t worry about universality

What VLSI Wants You to Build



- Modular machine
  - Lots of potential compute units, w/ memory

Making Communication Explicit

- In VLSI, communication is what matters

  It is the wires, stupid

- Another way of saying this is:
  - In VLSI building computation elements is easy
  - Keeping them feed is hard
  - Hence, most of a modern processor stages data

- What a computation model that
  - Makes communication explicit
  - Provides feedback to the programmer about communication

The “Ideal” VLSI Machine

- Lots of simple compute units
  - Units feed by cheap (in energy, area) sources – local regs
  - Relatively cheap instruction issue logic
- Memory (FIFOs) to decouple data fetch/execute
  - Communication takes time (it is the LAW)
  - Need to enable the machine to tolerate latency
- Interconnection network with high-bandwidth
  - And as small latency as possible
- Connections to large backing store
  - Main memory and disk

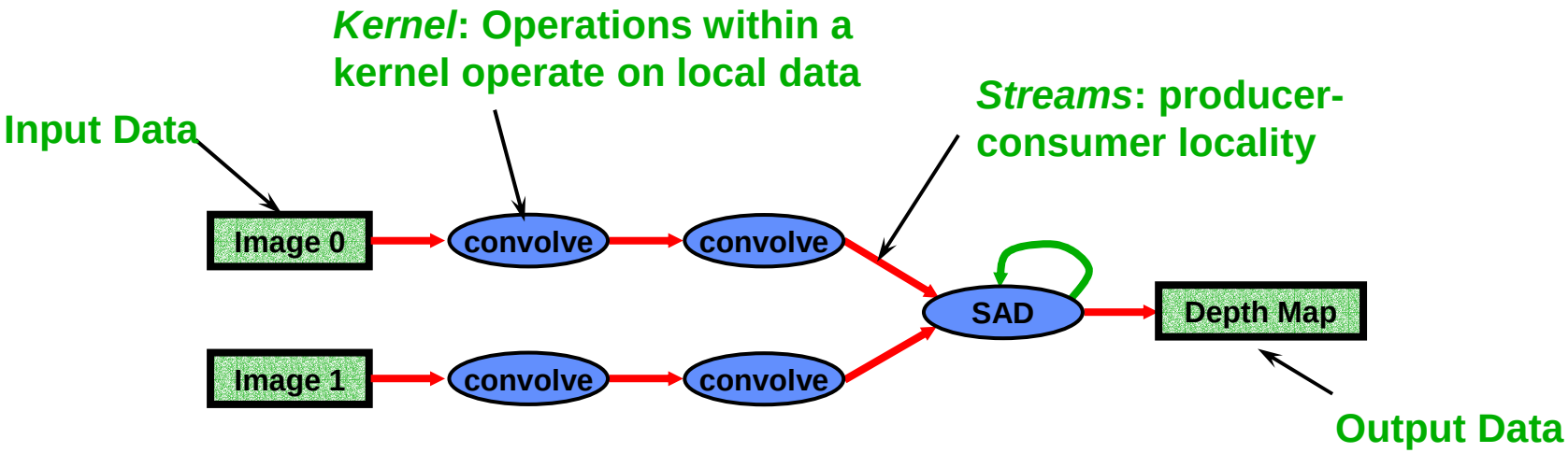
- Streams are a programming model that matches this machine

What Stream Programming Means To Me

- Programming model where
  - Applications have large amounts of data parallelism
    - So you don't have to invent it
  - Communication is made explicit
    - So the programmer can tell what will be expensive
  - Easy to estimate performance of applications
- Similar to
  - Vector programming
  - Synchronous dataflow graphs
  - Mathlab/Simulink
  - Graphics machines, etc

It does not bother me that this is not general. It is common enough we should have an efficient solution for these applications

Stream Programming Abstraction



- Stream model makes communication / locality explicit
  - Temporaries within kernels, producer-consumer variables between kernels never are written back to memory
  - Most communication is represented by arcs, or memory gather/scatter
- Locality helps support large numbers of ALUs

Stream Architecture

- Basically need four parts:
  - Function units (Add,Mult)
  - Registers to hold the data
  - DMA channels to load/spill the registers
  - Something to sequence instructions

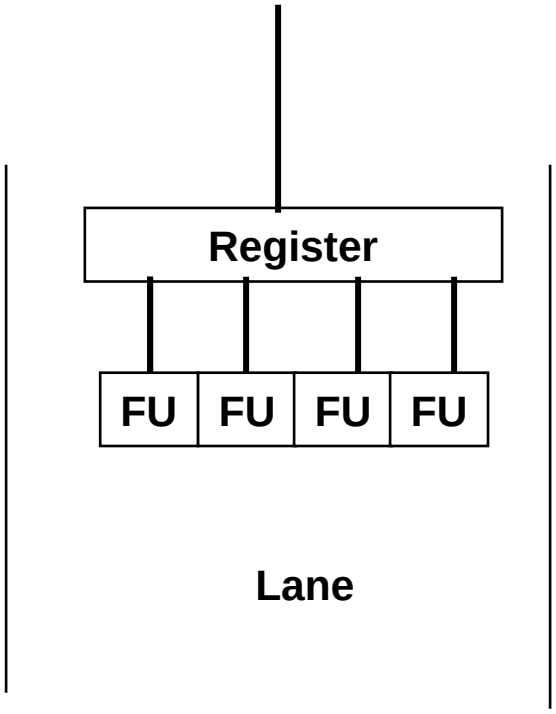
- For efficient operation
  - Want to decouple load/store from execute
    - Need lots of registers
  - Can optimize registers/memory for different functions

Simple Stream Machine

- Vector machines were early stream machines
- Scalar processor was the sequencer
- Two levels of storage for the “registers”
  - Vector registers
  - Main memory

- Vector registers are often a limitation
  - #ports = 3*#FU
  - #regs = Latency

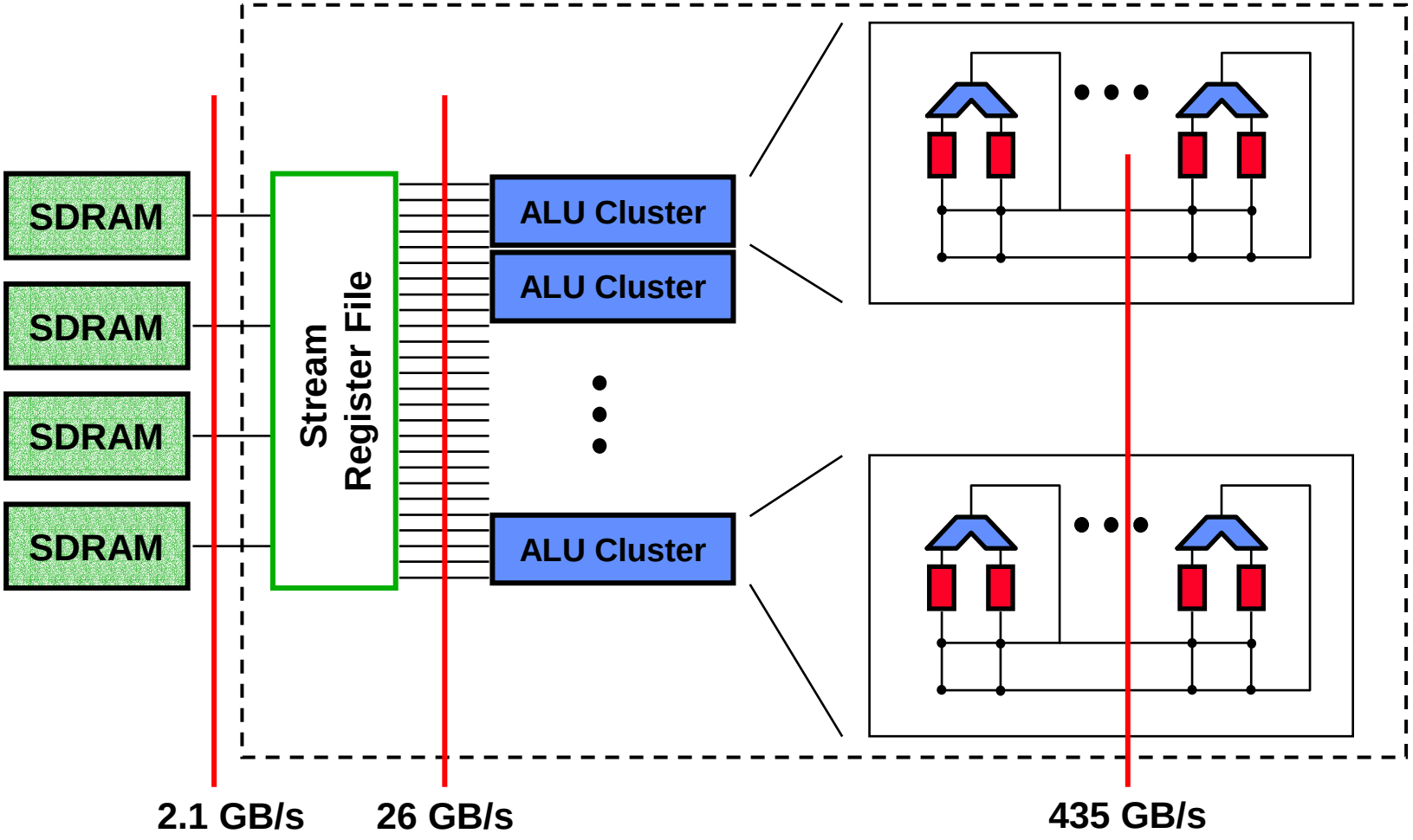
- Graphics use this resource model
  - Slice the computation into stream



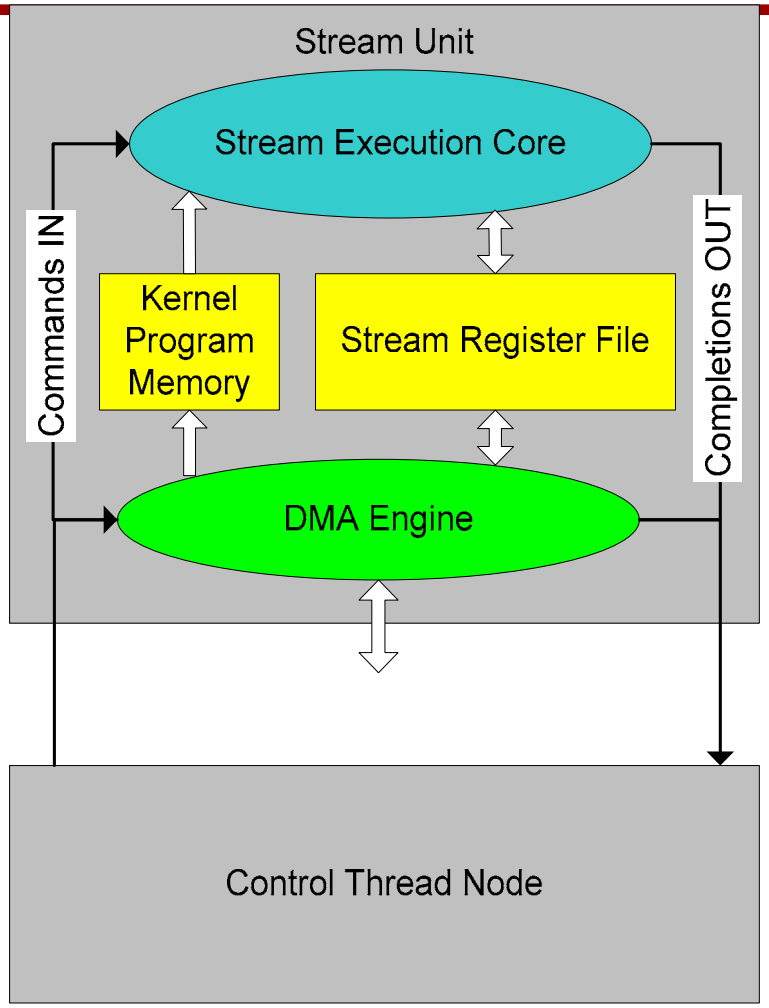
More Refined Model

- Split the registers into two functions
  - Hold temporaries in the computation
  - Input / Output of the kernel
- Temps don’t need to be global for all function units
  - Although it is easier for the compiler if they are
    - Localized registers will need move instructions
  - The don’t need to hide the memory latency either
    - Generated from function units
- Input /Output registers are only for queuing
  - Number of ports does not need to grow with #FU
  - Very regular access out of these registers
  - Access can be longer
  - Build dense memory for these

Imagine Stream Machine



Stream Virtual Machine



- Really three threads of control
- Control thread is master
  - Issues non-blocking instructions for core and DMA
  - Mechanism for respecting dependencies
    - Explicitly specify what each non-blocking instruction depends on
- DMA and Core threads
  - Have a queue of work
  - Run when constraints satisfied
- Machine dependent resources
  - Graphics SRF = memory

Many Possible Stream Hardware Implementations

- We think (hope) all will fall into the general SVM model
- Need to have some parameters that characterize hardware
  - Less is more in this situation
  - Want to export only key issues

- Goal is partition compilation process into two parts
  - High-level compiler
    - Work with parallelization of code
    - Data blocking and placement
    - Platform independent
  - Low-level (node) compiler
    - Code scheduling
    - Detailed resource allocation

Streaming Summary

- Uniprocessor scaling will change soon
  - Highly constrained space
  - Performance gains are generally modest
  - Hard to build any really interesting machines
- VLSI wants you to build parallel modular machines
  - Streams are a programming model that matches machine
- Stream programming model is familiar to some programmers
  - It is the model that they prototype SP applications in
  - Easy to explain to someone
- Potentially large improvement possible in Op/sec and Ops/Joule