

CRAY

The Supercomputer Company

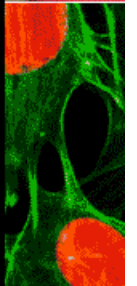
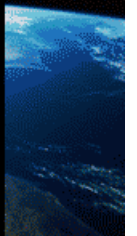


Challenges and Opportunities in the Post Single-Thread-Processor Era

Steve Scott

Chief Technology Officer, Cray Inc.

August 2, 2006



Abstract

Challenges and Opportunities in the Post Single-Thread-Processor Era

- The age of the single thread juggernaut has ended, due to a variety of factors. Multi-core processors are coming on strong, and scaling is being stressed more than ever. This presents a number of architectural, hardware and software challenges. This talk will reflect on these challenges from Cray's perspective in the high performance computing industry.

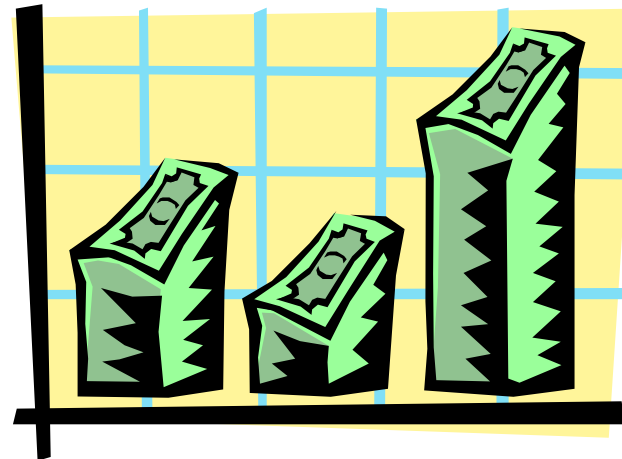
Outline of Talk

- The impact of technology on architecture
- The recent technology inflection point
- Implications on processor designs
- Resulting hardware and software challenges
- Where might we go next?

Some historical background from Cray's perspective

The Early Days of Supercomputing

- Cray 1 (1976)
 - 80 MHz, pipelined, 2 flops/clock \Rightarrow 160 Mflops/sec (64 bit)
 - Contemporary micro's were 1-4 MHz, non-pipelined (16 bit)
 - The Cray-1 was the fastest *scalar* processor in the world
- At the time Cray Research stock peaked it was the **#1 performing stock** on Wall Street over its lifetime

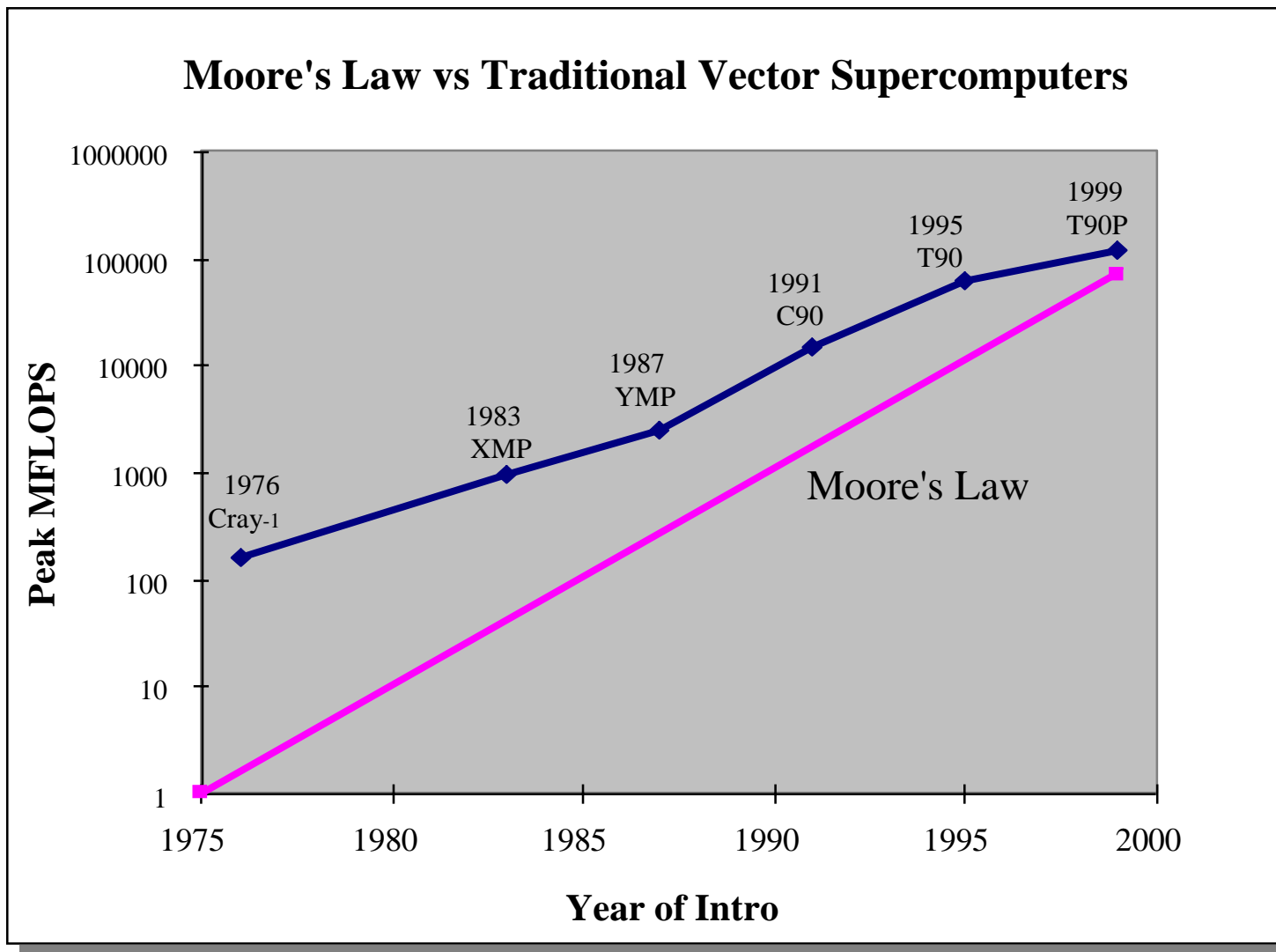


- I joined the company three years later. ☹️

Why Was The Cray-1 So Fast?

- Technology:
 - ECL logic
 - SRAM main memory
 - Only 11 clocks away (< 150 ns)
 - Awesome system engineering (important due to low density)
 - One processor on ~3400 circuit boards
 - “C” shaped for shorter wires
 - Refrigerant cooled
 - More mechanical patents than logic/architecture patents
- Highly pipelined RISC design
 - Really Invented by Seymour Cray
 - Intel introduced pipelining about 15 years later with the '486
- Vector ISA
 - High operation rate with very low control complexity

Classic Vector Systems of CRI



Attack of the Killer Micros



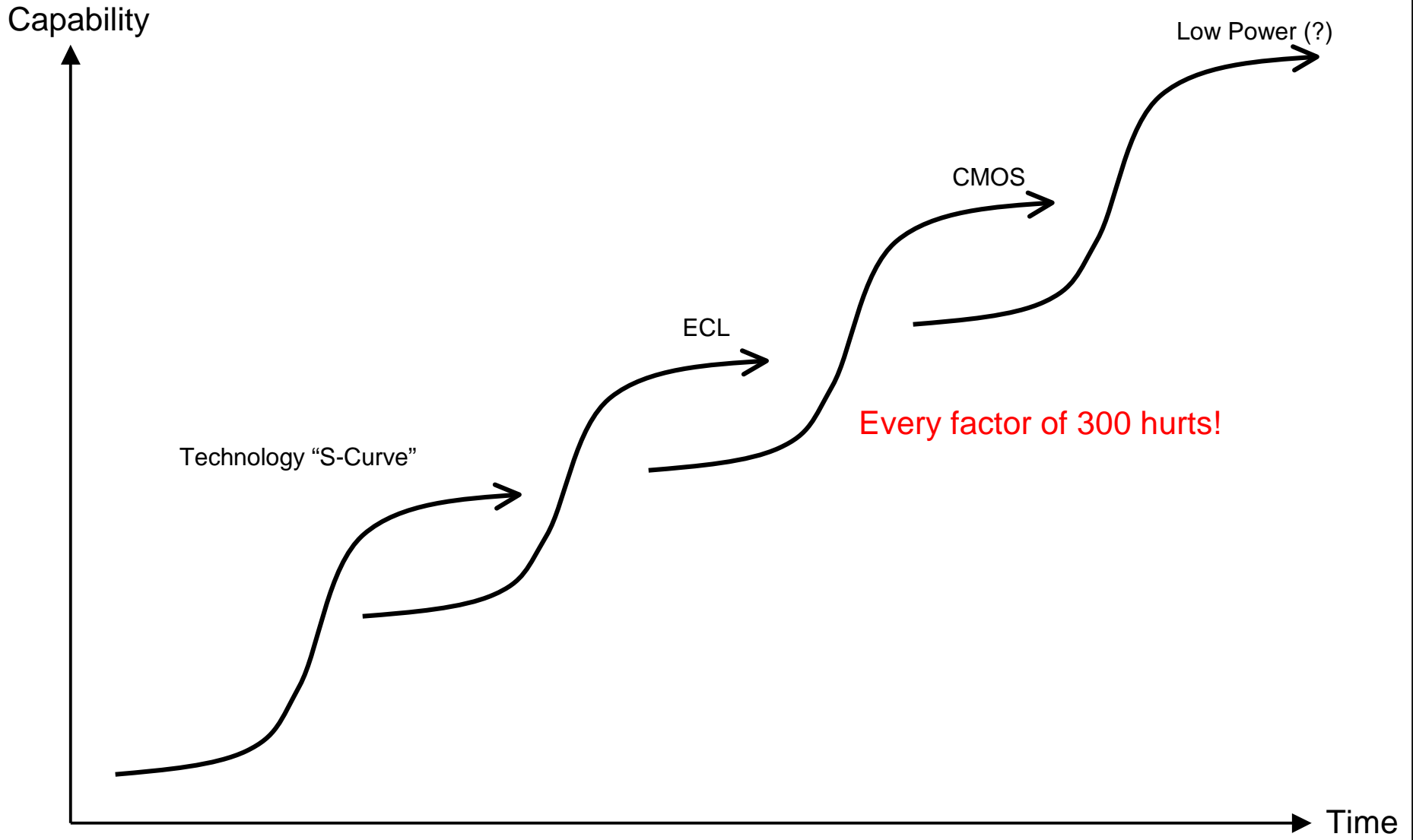
Q: Why the decline of classic vector supercomputers?

A: A few reasons...

Economic Impact

- Economies of scale
- Higher volume means:
 - Cheaper parts
 - Can amortize larger development cost over many sales
 - Mutually-reinforcing community
 - ISV codes, standards, partnerships, tools, peripherals, programmer mindshare, etc.

Technology Impact



Architecture Impact

- Classic vector systems relied upon uniform connectivity to global shared memory (no memory hierarchy).
- But IC *connectivity* doesn't improve as fast as *density*
- Example: DRAM memory chips over same time period

DRAM Memory Chip Performance

1979: Standard DRAM

- 16 Kbit
- 1-bit wide interface
- 5 Mb/s uniform access BW
- 2 Mb/s random access BW



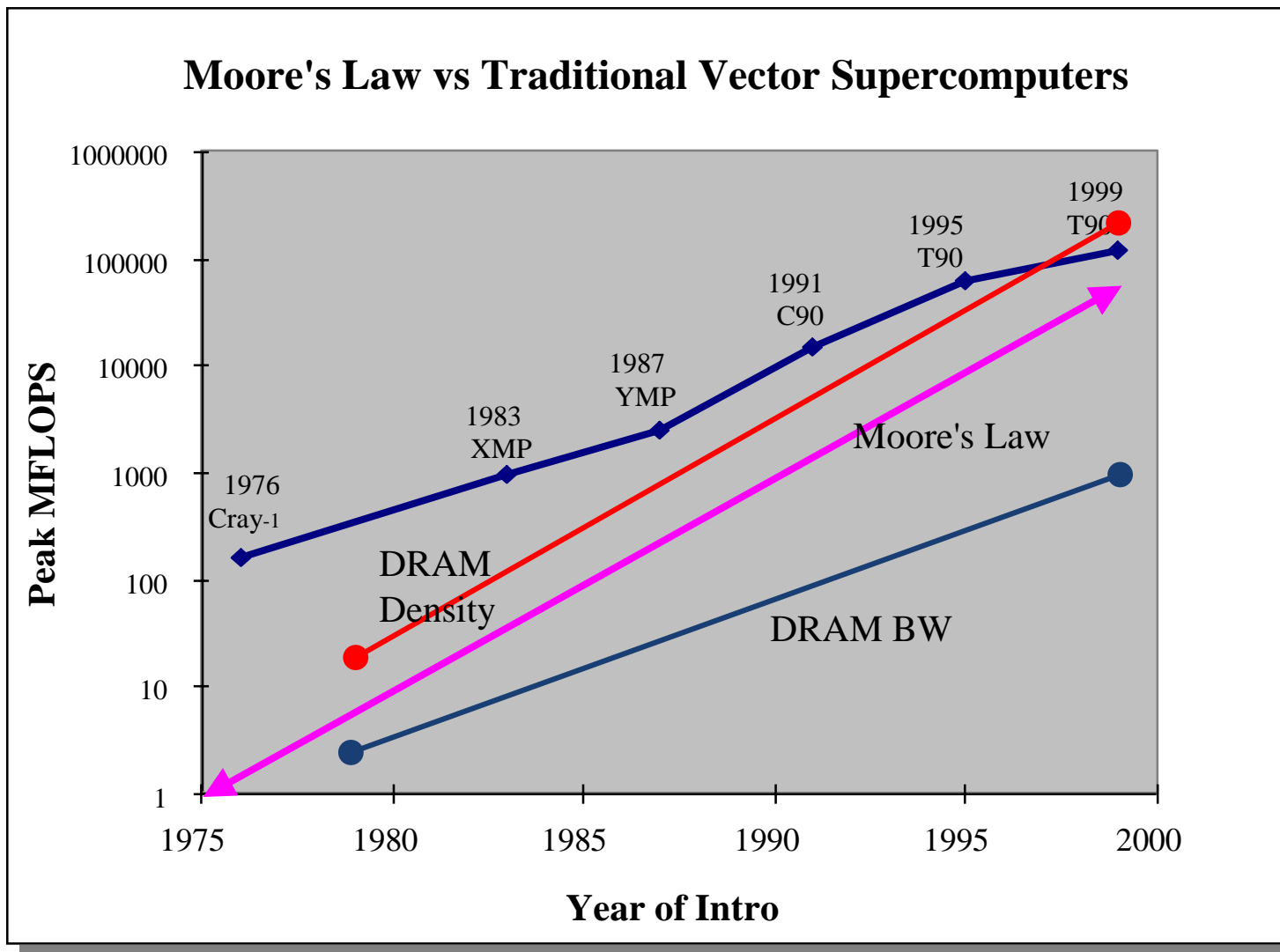
1999: 200 MHz SDRAM

- 256 Mbit
- 16 bit wide interface
- 3200 Mb/s uniform access BW
- 1000 Mb/s random access BW

1979 → 1999:

- 16000X density increase
- 640X uniform access BW increase
- 500X random access BW increase
- 25X *less* per-bit memory bandwidth

Vector Performance Trend



Architectural Implication

It wasn't the processor architecture that held back these vector systems,
it was the *system* architecture.

For vector systems to advance at Moore's Law rate,
they must adopt *hierarchical* memory

- Registers
- Caches
- Local memory
- Remote memory

This was a significant motivation behind the design of the Cray X1 system.

Meta point: technology has a significant impact on architecture

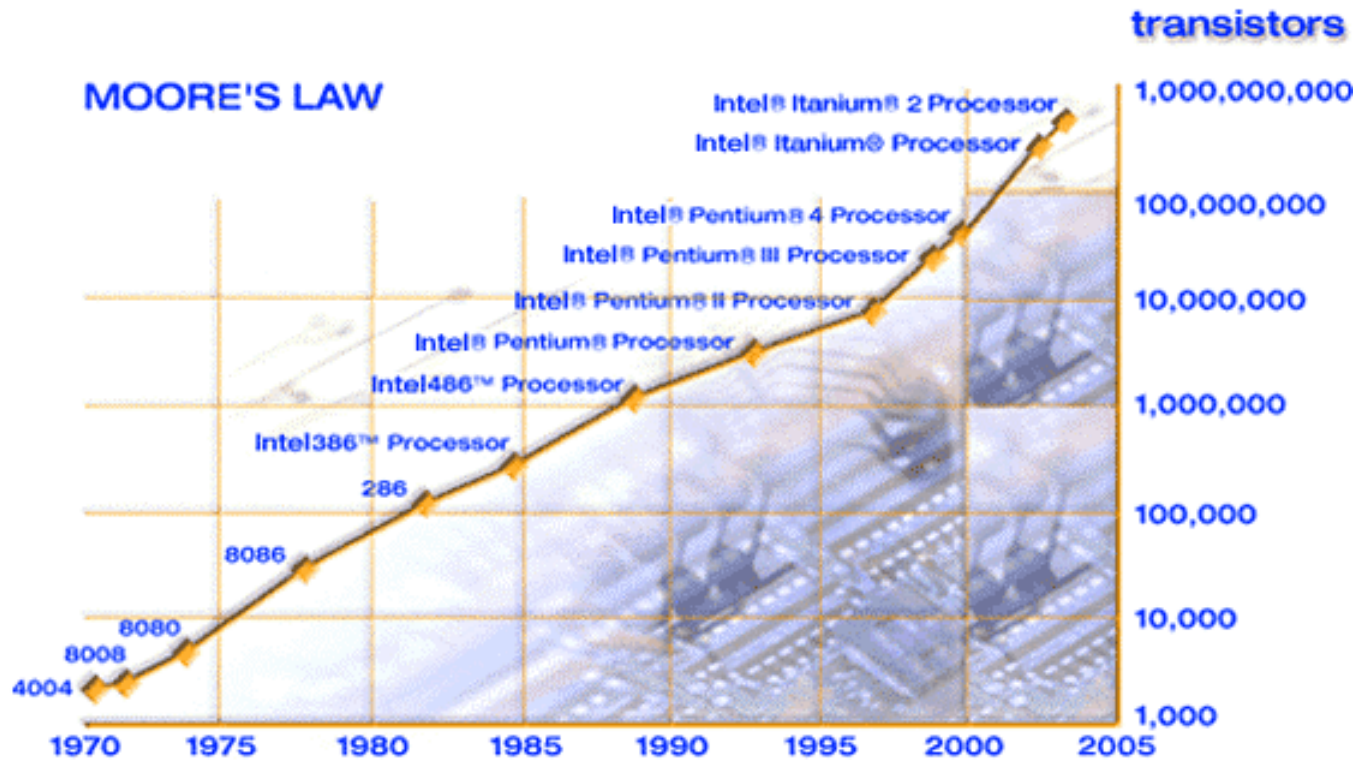
The same questions have different answers as the underlying technology changes.

So what is the technology situation today?



Moore's Law

The number of transistors per chip doubles every 18 months



From Intel web site.

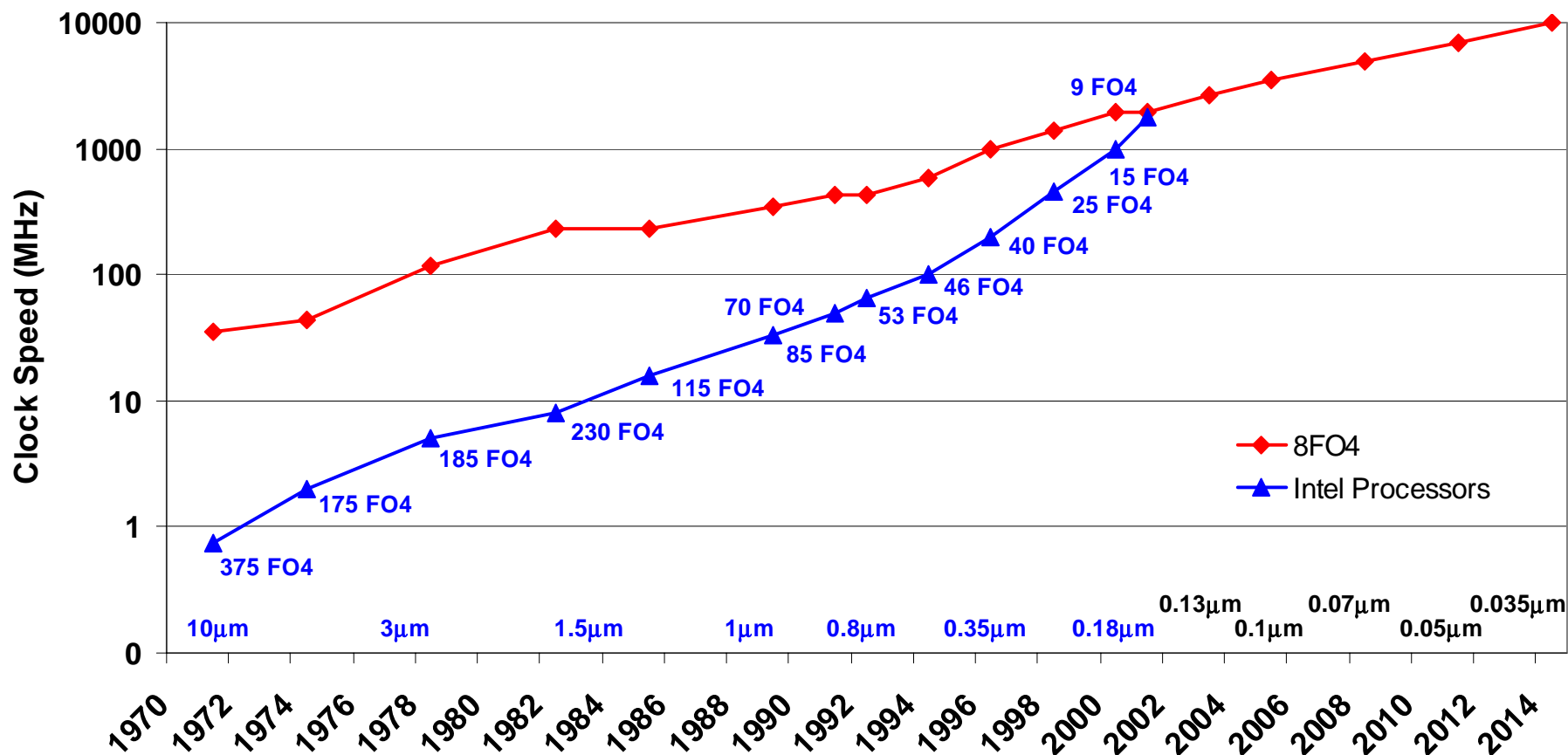


Gordon Moore, "Cramming More Components Onto Integrated Circuits," *Electronics*, April 19, 1965.

Density Has Driven Performance

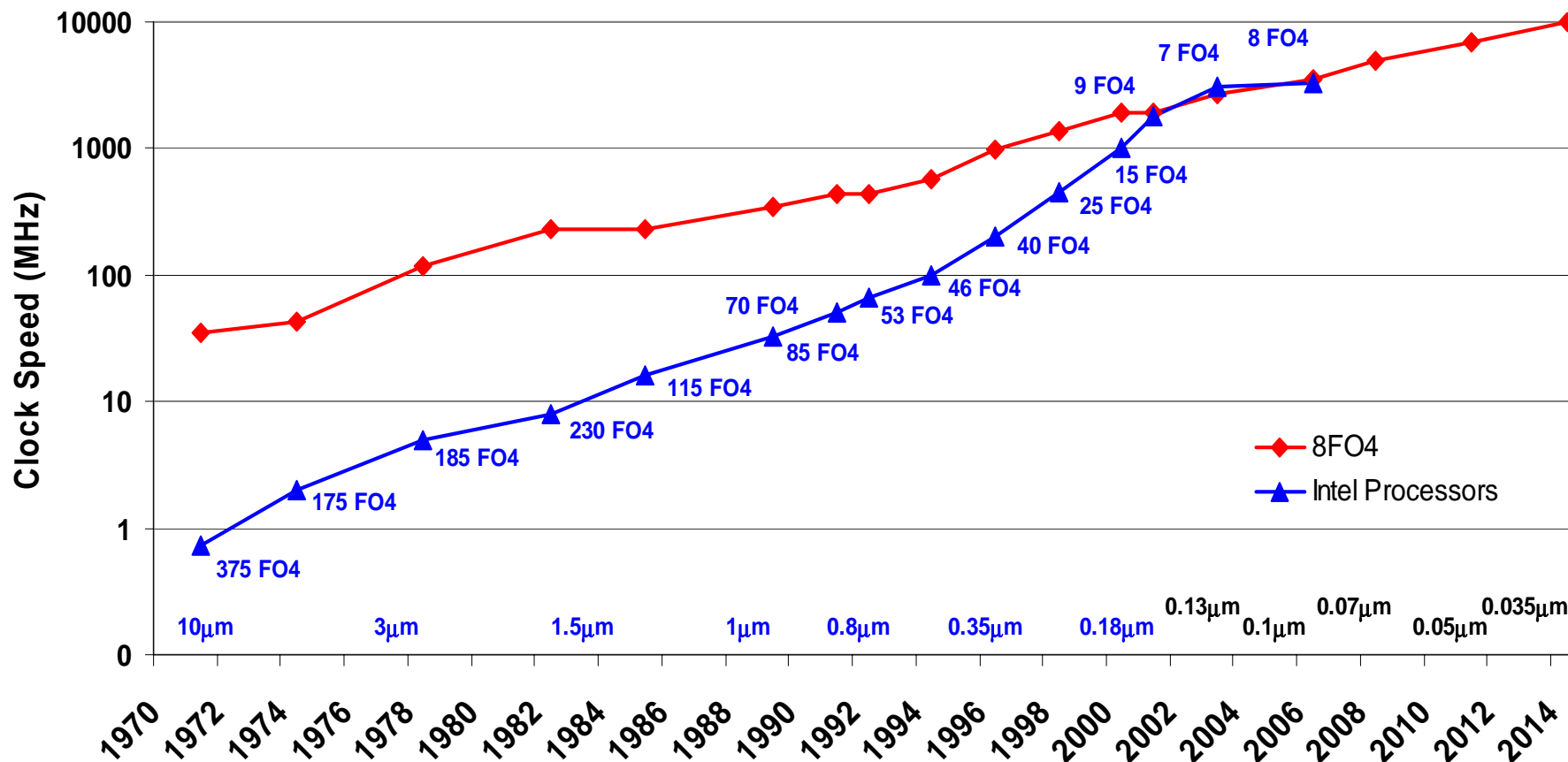
- Moore's Law relates to *density*
 - Transistor switching time ~ proportional to gate length
 - feature size X 0.7 \Rightarrow **density X 2, speed X 1.4**
 - However, we've also used the extra transistors to....
 1. ... design deeper pipelines (fewer levels of logic per clock)
 - \Rightarrow clock rate has been increasing faster than logic transistor speed
 2. ... perform more complicated logic
 - \Rightarrow instructions per clock (IPC) has increased over the years
- \Rightarrow *performance scaled with density for first 30 years of Moore's Law*

Clock Scaling Over Time



Source: ISAT Last Classical Computer study, 2001

Clock Scaling Over Time



Source: ISAT Last Classical Computer study, 2001

Difficult to Extract More ILP

- Instructions per clock (IPC) over the years:
 - Early microprocessors took several clocks to issue one instruction
 - In 1991: all mainstream micros were single way issue
 - By 1993: x86, 68K, Sparc, Alpha, PowerPC and PA-RISC were all 2 (or 3) way issue
 - By 1994: PowerPC, Alpha and MIPS were 4 way, and so were Sparc and PA-RISC by '95 and '96.
 - In 2006: *still 4 to 6 way*
 - Since then, processors have gotten much more complex (aggressive prediction and speculation) to sustain more of the 4-6 way potential
- It's becoming much harder to increase IPC
 - ⇒ **Not effective to keep building fatter sequential processors**

Pentium III vs. Pentium IV

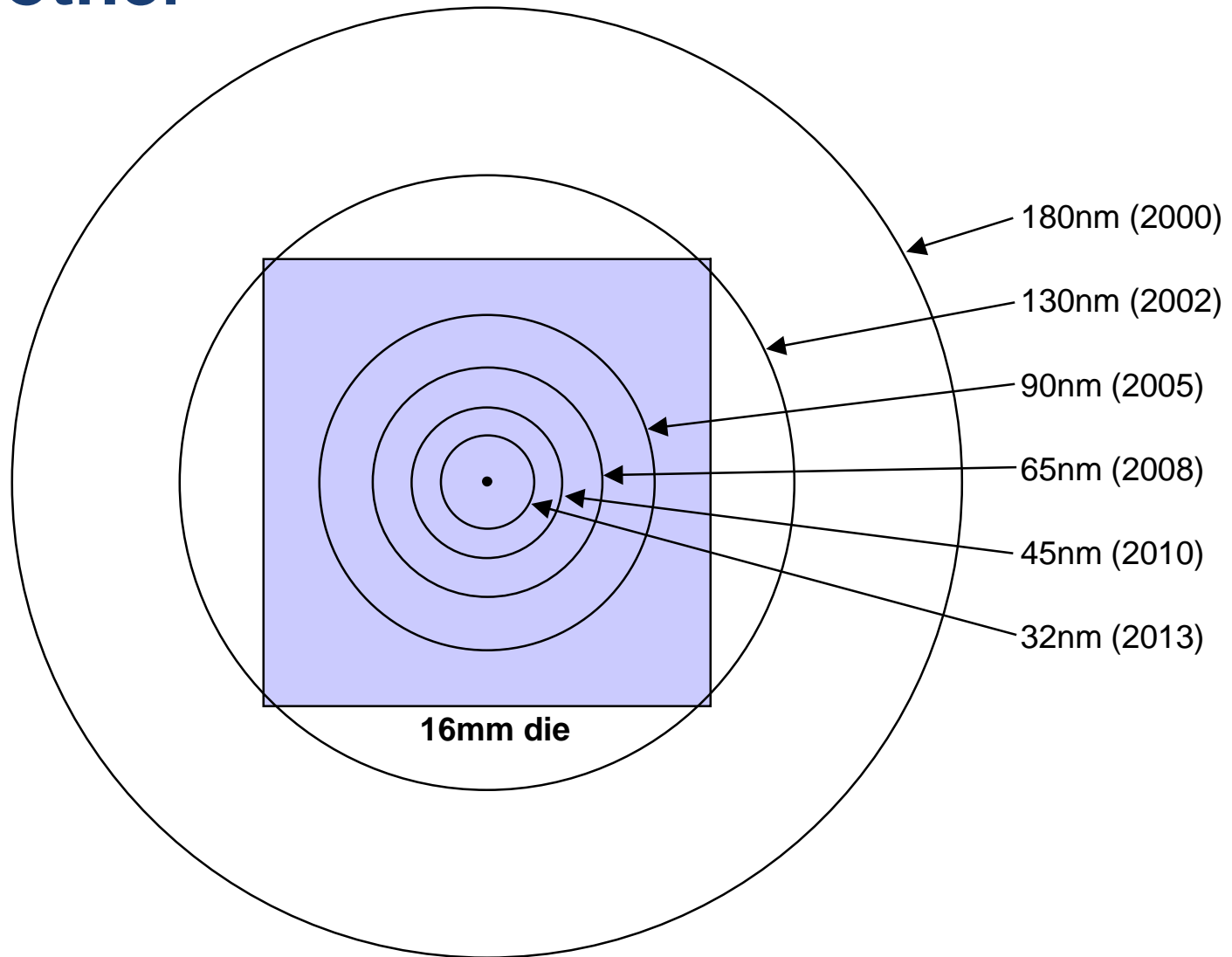
	Pentium III	Pentium IV
Technology	180nm	180nm
Die Size	106mm ²	217mm ²
Transistor Count	24 million	42 million
# Grids	10 ⁸	2x10 ⁸
Pipeline Stages	10	20
Clock Rate	1GHz (15 FO4)	1.5GHz (10.4 FO4)
L1 D\$ Capacity	16KBytes	12KBytes
SpecInt2000	454	524
SpecInt/MHz	0.45	0.35

Source: ISAT Last Classical Computer study, 2001

Another Reason Not to Build Faster Uniprocessors



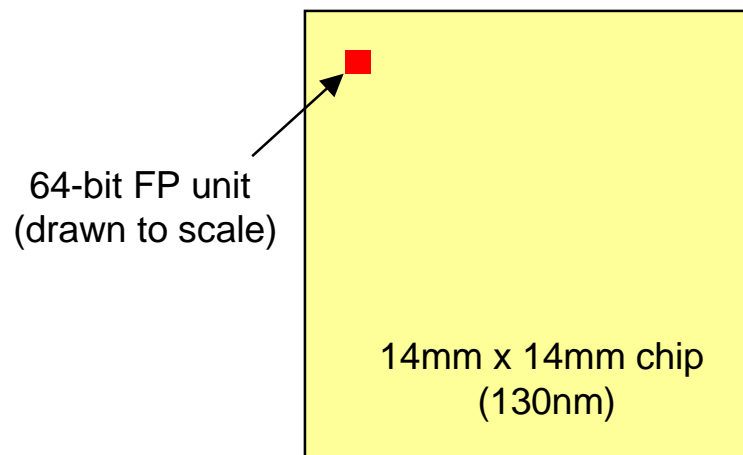
And Another



Signal reach in one clock cycle (8 FO4)

Flops are Cheap, *Communication is Expensive*

- In 0.13um CMOS, a 64-bit FPU is $< 1\text{mm}^2$ and $\cong 50\text{pJ}$
Can fit over 200 on a \$200 14mm x 14mm 1GHz chip



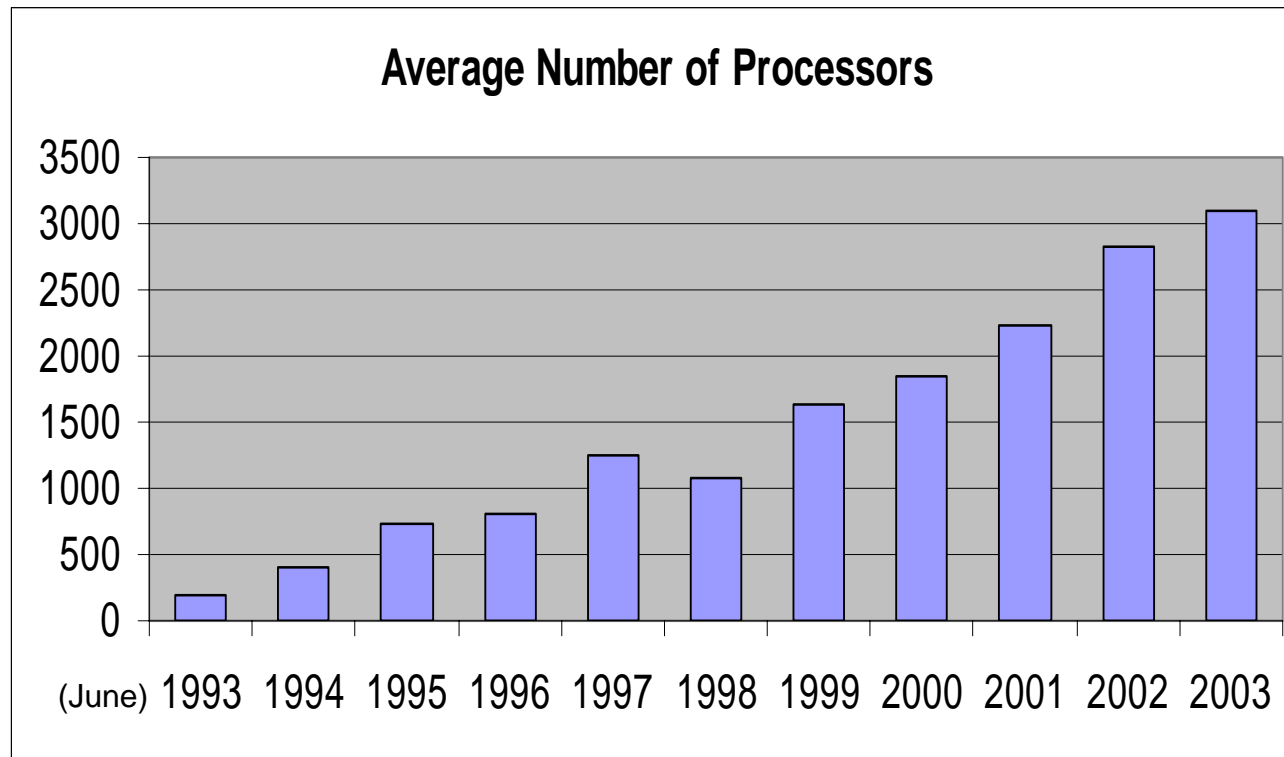
- If fed from small, local register files:
 - 3200 GB/s, 10 pJ/op
 - $< \$1/\text{Gflop}$ (60 mW/Gflop)
- If fed from global *on-chip* memory:
 - 100 GB/s, 1 nJ/op
 - $\sim \$30/\text{Gflop}$ (1W/Gflop)
- If fed from *off-chip* memory:
 - 16 GB/s
 - $\sim \$200/\text{Gflop}$ (many W/Gflop)

Relative cost growing with successive IC generations

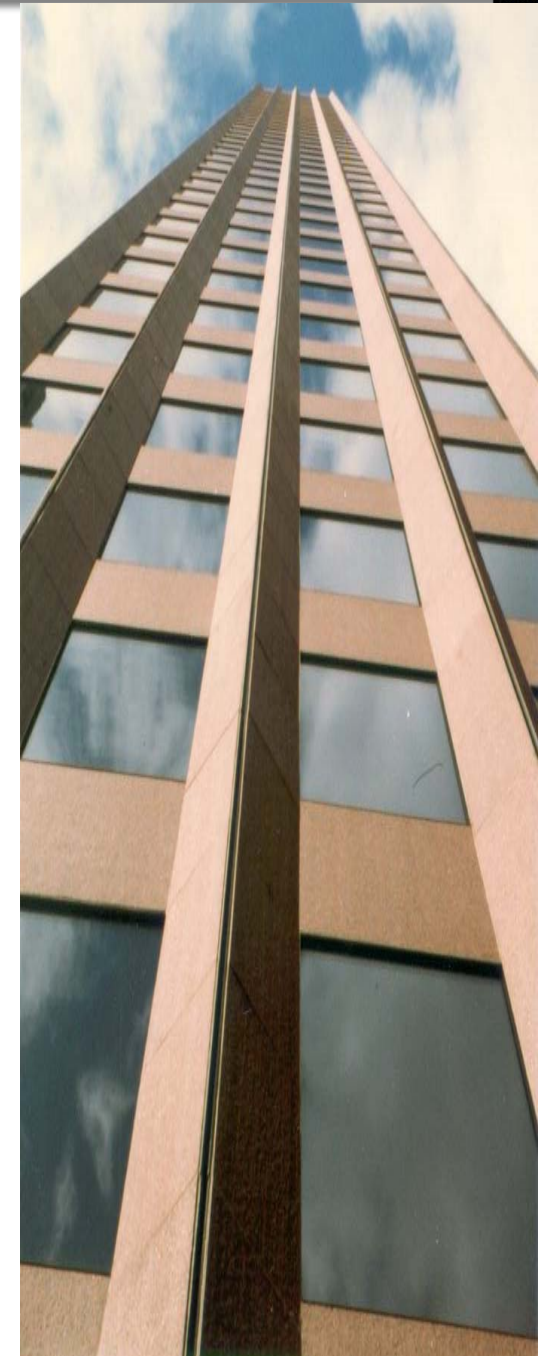
Implications for On-chip Architecture

- Okay to overprovision (cheap) flops to maximum utilization of (expensive) memory bandwidth
- Must exploit locality on chip
 - Try to minimize expensive data movement
- Keep voltage and frequency down
 - Concentrate on parallel performance, not single thread performance
- Reduce complexity and overhead
 - Higher fraction of chip doing *computation*, as opposed to control and orchestration
- Commercial response has been to go multi-core
 - Drives up the number of processors per system

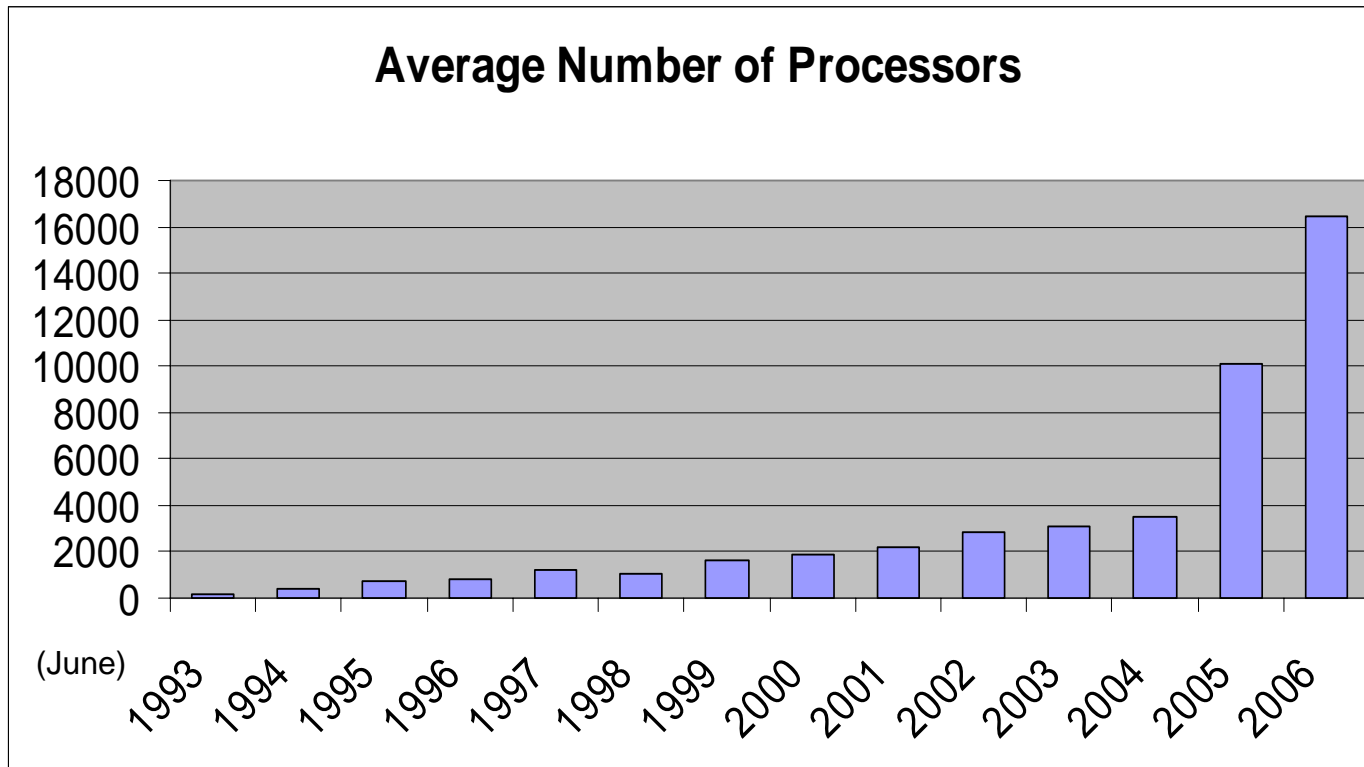
Trends in the Top 500 list (top 20)



- Supercomputing is becoming increasingly about *scalability*
- Expect *rate* of increase to increase
 - ⇒ systems with > 100,000 procs in the next decade



Trends in the Top 500 list (top 20)



- Supercomputing is becoming increasingly about *scalability*
- Expect *rate* of increase to increase
 - ⇒ systems with > **1,000,000** procs in the next decade



How Do We Deal with 1,000,000 Processors?

- System management challenges
 - Operating system scaling
 - File system scaling
 - Job scheduling, launch, and monitoring
 - Configuration management and system boot
- Reliability challenges
 - Basic components are becoming *less* reliable
 - In largest systems, something will always be broken and breaking
 - Need to tolerate failures and keep the system up
 - Need to think about reliable *applications* (checkpointing too expensive)
- Programming challenges
 - Scaling applications – Amdahl's Law, load balancing, communication, etc.
 - Debugging – conventional models simply don't scale (need better tools)
 - Performance tuning – very difficult to understand program behavior at huge scales (need better tools)

Parallel Programming Goes Mainstream

- The revolution has begun: parallel programming for the masses
 - Not just for scientists at national labs anymore...
 - Pretty soon we'll all have tens of processors on our desktops
 - We *really* need to make this easier!

Parallel Programming Complexity

Relative complexity of three programming models
(as measured by the size of their documentation):

- MPI 1 Standard: 239
MPI 2 Standard: 376
Total:

615 Pages
- “intro_shmem” man page: 10 Pages
(we implemented the entire NPB
suite with a shmem_get and a barrier)
- CF90 Co-Array Programming
Manual (SR-3908): 30 Pages
(11 pages for language description)

Random Gather (serial)

```
parameter (n=2**30)
real table(n)
buffer(nelts)      ! nelts << n
...
do i=1,nelts
  buffer(i) = table(index(i))
enddo
```

- Purely synthetic, but simulates “irregular” communication patterns
- We do have customers that do this stuff

Random Gather (Coarray Fortran)

```
parameter (n=2**30)           ! 1 Gigaword
parameter (NPES=2**7)        ! 128 Pes
parameter (eltspe = 2**23)    ! Elements per PE
real table (eltspe)[NPES]
...
do i=1,nelts
  PE =( index(i) + eltspe-1)/eltspe
  offset = mod(index(i)-1,eltspe)+1
  buffer(i) = table(offset)[PE]
enddo
```

Random Gather (MPI)

```

if(my_rank.eq.0)then
! first gather indices to send out to individual PEs
do i=1,nelts
indpe = ceiling(real(index(i))/real(myelts)) - 1
isum(indpe)=isum(indpe)+1
who(isum(indpe),indpe) = index(i)
enddo
! send out count and indices to PEs
do i = 1, npes-1
call MPI_SEND(isum(i),8,MPI_BYTE,i,10,
& MPI_COMM_WORLD,ier)
if(isum(i).gt.0)then
call MPI_SEND(who(1,i),8*isum(i),MPI_BYTE,i,11,
& MPI_COMM_WORLD,ier)
endif
enddo
! now wait to receive values and scatter them.
do i = 1,isum(0)
offset = mod(who(1,0)-1,myelts)+1
buff(i,0) = table(offset)
enddo
do i = 1,npes-1
if(isum(i).gt.0)then
call MPI_RECV(buff(1,i),8*isum(i),MPI_BYTE,i,12,
& MPI_COMM_WORLD,status,ier)
endif
enddo
do i=nelts,1,-1
indpe = ceiling(real(index(i))/real(myelts)) - 1
offset = isum(indpe)
isum(indpe) = isum(indpe) - 1
buffer(i) = buff(offset,indpe)
enddo
else !if my_rank.ne.0
call MPI_RECV(my_sum,8,MPI_BYTE,0,10,
& MPI_COMM_WORLD,status,ier)
if(my_sum.gt.0)then
call MPI_RECV(index,8*my_sum,MPI_BYTE,0,11,
& MPI_COMM_WORLD,status,ier)
do i = 1, my_sum
offset = mod(index(i)-1,myelts)+1
do i = 1, my_sum
offset = mod(index(i)-1,myelts)+1
buffer(i) = table(offset)
enddo
call MPI_SEND(buffer,8*my_sum,MPI_BYTE,0,12,
& MPI_COMM_WORLD,ier)
endif
endif
endif

```

Opportunity for a Better Path

- I don't think we want to inflict MPI style programming on the world
- *Now is the time to establish a better parallel programming model*
 - It's a social/cultural problem, not a technical one
 - Will require government, industry and user cooperation
 - Current models: MPI, OpenMP
 - Transitional models: Coarray Fortran, UPC, Titanium, Global Arrays, etc.
 - DARPA HPCS languages: Chapel, Fortress, X-10
 - Or perhaps something like Matlab?

New Opportunities for Processor Designers

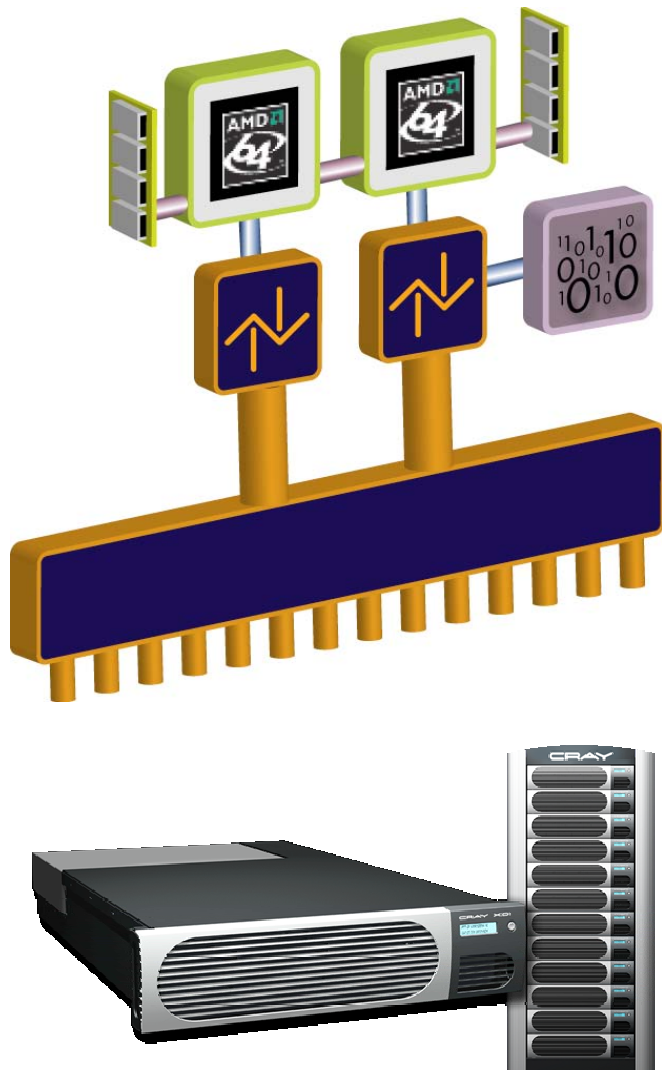
- Slowing single-thread performance may make special-purpose designs more attractive
 - It used to be better to wait for general purpose processors to improve
 - May want to spend extra transistors for specialized functions
 - Possible for niche processors to “catch up” in serial performance
- How to best use all those transistors to extract parallelism?
 - Multi-core (CMP)
 - But still a lot of overhead in traditional scalar processors
 - Exacerbates the “memory wall”
 - Latency continues to grow relative to cycle time
 - Processors are not very latency tolerant
 - Other possibilities
 - Heterogeneous/specialized processors
 - Vectors
 - Multithreading
 - FPGAs
 - Streaming architectures

So, Can We Just Pack Chips with Flops?

- Key is making the system easily programmable
- Must balance peak computational power with generality
 - How easy is it to map high level code onto the machine?
 - How easy is it for computation units to access global data?
- Some examples:
 - XD1 FPGAs
 - Stanford's Streaming Supercomputer project (Merimac)
 - Clearspeed CSX600
 - IBM Blue Gene
 - IBM Cell
- Flop efficiency vs. generality/programmability spectrum:
 - Qualitative only; also influenced by memory system



Cray XD1 FPGA Accelerators

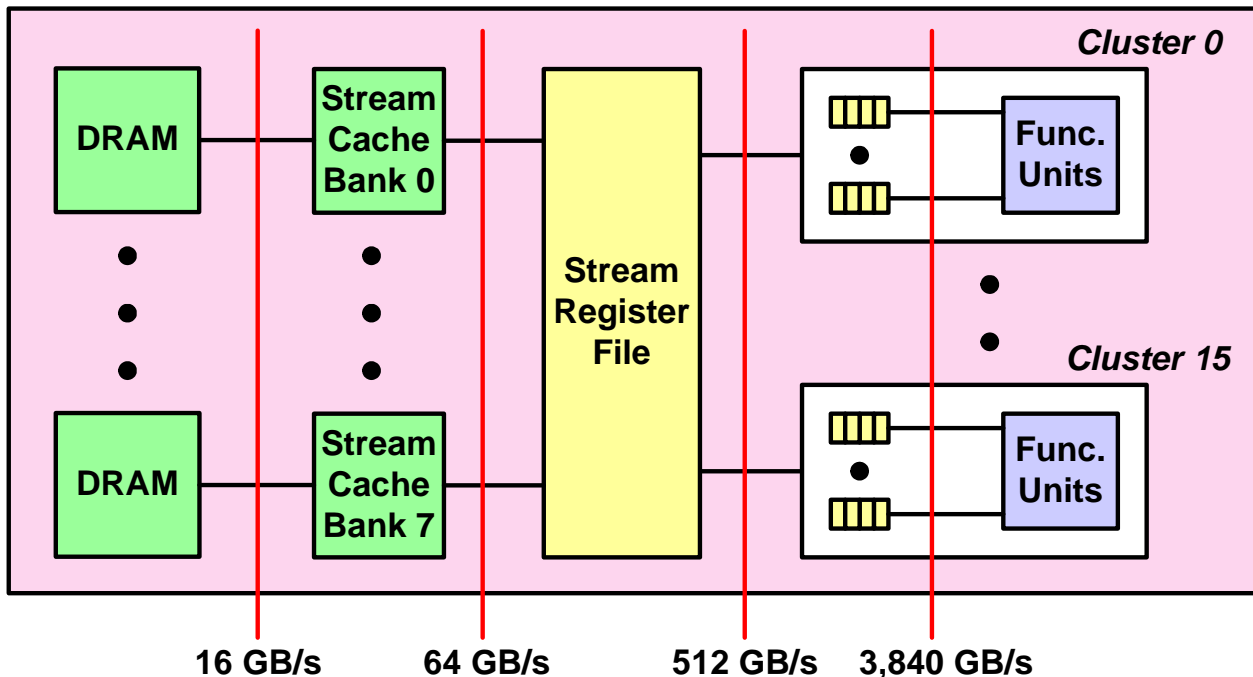


Performance gains from FPGA:

- **RC5 Cipher Breaking**
 - Implemented on Xilinx Virtex II
 - **1000x** faster than 2.4 GHz P4
- **Elliptic Curve Cryptography**
 - Implemented on Xilinx Virtex II
 - **895-1300x** faster than 1 GHz P3
- **Vehicular Traffic Simulation**
 - Implemented on Xilinx Virtex II (XC2V6000) and Virtex II Pro (XC2VP100)
 - **300x** faster on XC2V6000 than 1.7 GHz Xeon
 - **650x** faster on XC2VP100 than 1.7 GHz Xeon
- **Smith Waterman DNA matching**
 - **28x** faster than 2.4 GHz Opteron

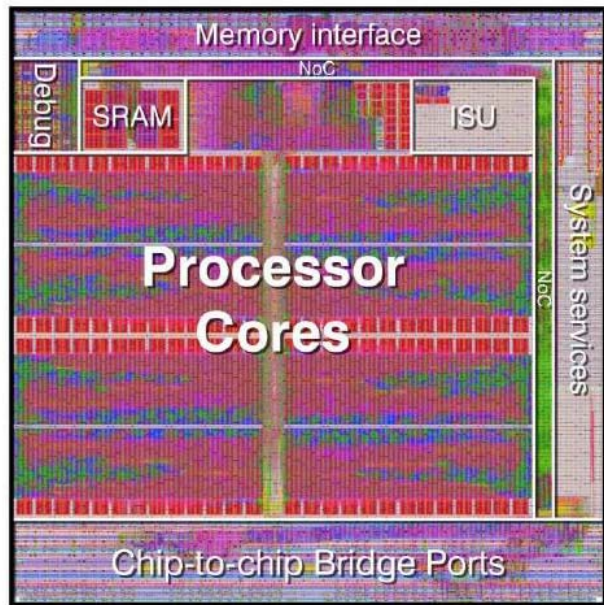
Primary challenge is programming.

Stanford Merrimac Streaming Computer

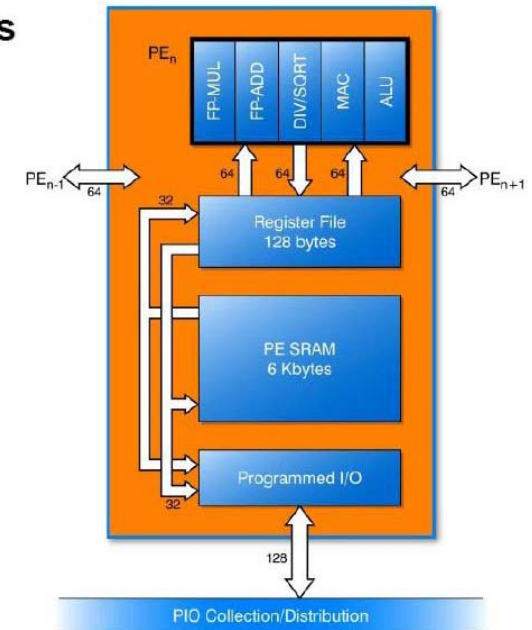


- On-chip memory hierarchy explicitly controlled by software
- Data is staged in large stream register file to maximize reuse
- Arrays of functional units operate independently, mostly from local registers
- *Requires some changes to code structure*

Clearspeed CSX600



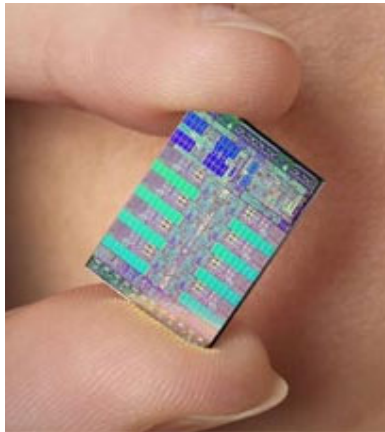
- Array of 96 Processor Elements
- 250 MHz
- IBM 0.13µm FSG process, 8-layer metal (copper)
- 47% logic, 53% memory
 - About 50% of the logic is FPUs
 - Around one quarter of the chip is floating point hardware
- 15 mm x 15 mm die size
- 128 million transistors
- Approx. 10 Watts



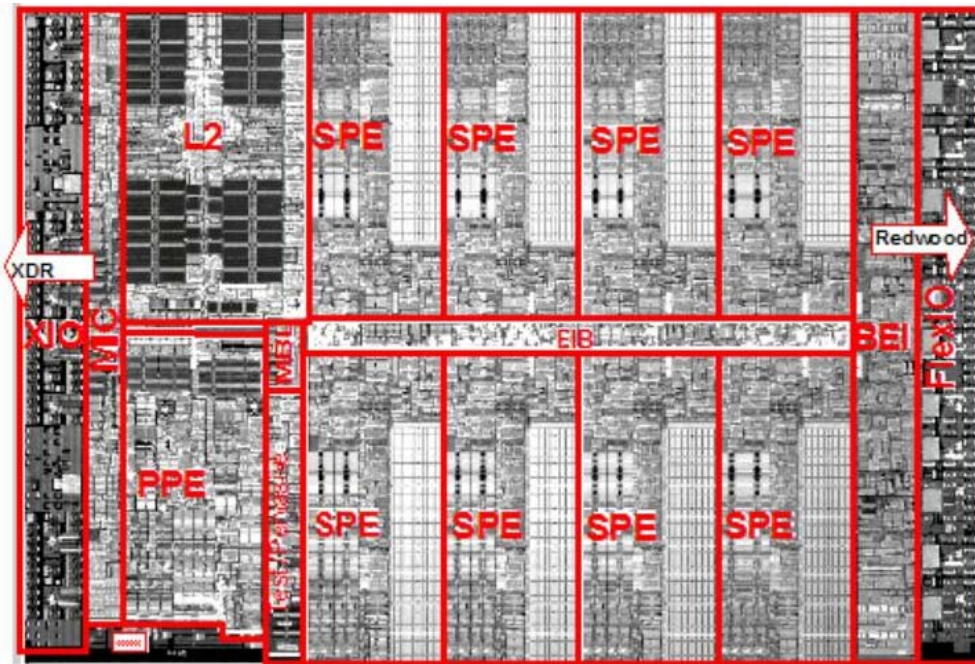
- 50 Gflops on card
- 6 GB/s to on-card local memory (4GB)
- 2+ GB/s to local host memory

*Doesn't share memory with host
Mostly used for accelerating libraries*

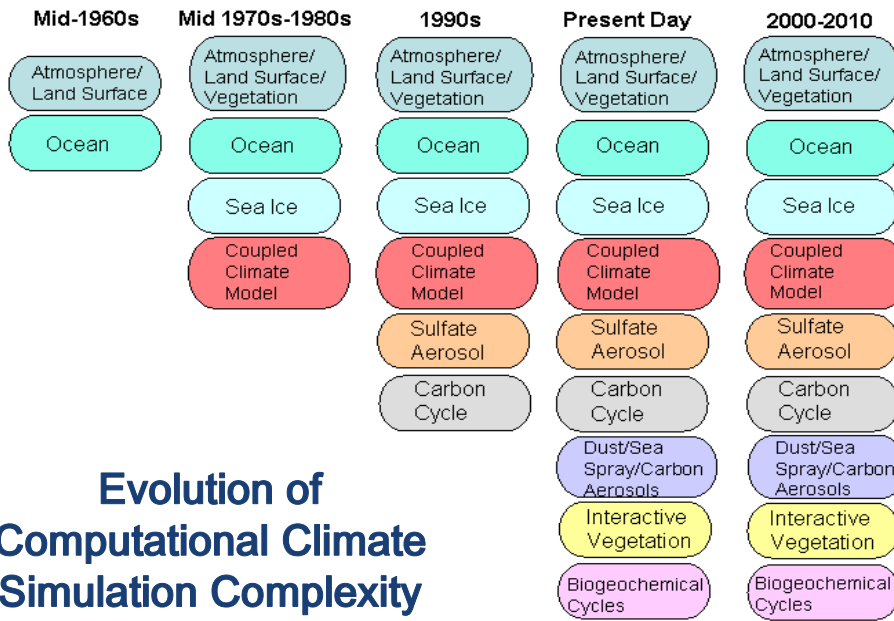
Cell Processor



- Each chip contains:
 - One PowerPC
 - Eight “synergistic processing elements”
- Targeted for:
 - (1) Playstations, (2) HDTVs, (3) computing
- Lots of flops
 - 250 Gflops (32 bit)
 - ~25 Gflops (64 bit)
- 25 GB/s to < 1GB memory
- **Big challenge is programming**
 - SPE’s have no virtual memory
 - Can only access data in local 256 KB buffers
 - Requires alignment for good performance
 - Often called a vector proc, really more a stream proc

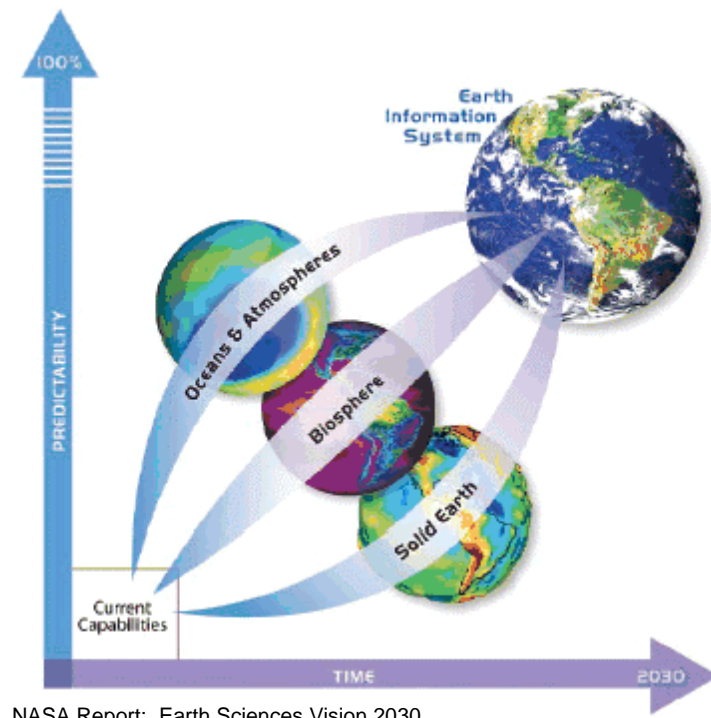


Increasingly Complex Application Requirements Earth Sciences Example



Evolution of Computational Climate Simulation Complexity

International Intergovernmental Panel on Climate Change, 2004, as updated by Washington, NCAR, 2005



NASA Report: Earth Sciences Vision 2030

Increased complexity and number of components lends itself well to a variety of processing technologies

“As scientific computing migrated toward commodity platforms, interconnect technology, both in terms of Bandwidth and latency, became the limiting factor on application performance and continues to be a performance bottleneck.”

-ComputerWorld article 2/6/06
-James Hack (NCAR)

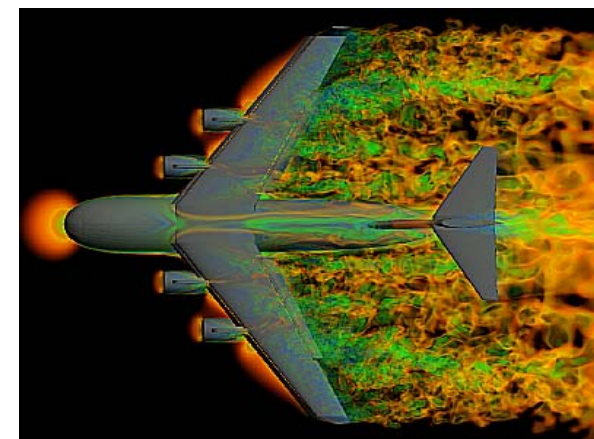
Increasingly Complex Application Requirements

CAE Example

- Industry is pushing the limits on size and complexity
 - Model sizes are currently limited by computational and data storage capabilities
 - Moving to Multi-Physics simulations
 - Modeling real-world behavior
 - Coupling previously independent simulations
 - Multi-Scale Requirements
 - Full system analysis requires different timescales
 - Material behavior in composite materials (micro-scale)
 - Real-time stress-strain behavior (macro-scale)

“The next high-payoff high performance computing grand challenge is to optimize the design of a complete vehicle by simultaneously simulating all market and regulatory requirements in a single, integrated computational model.”

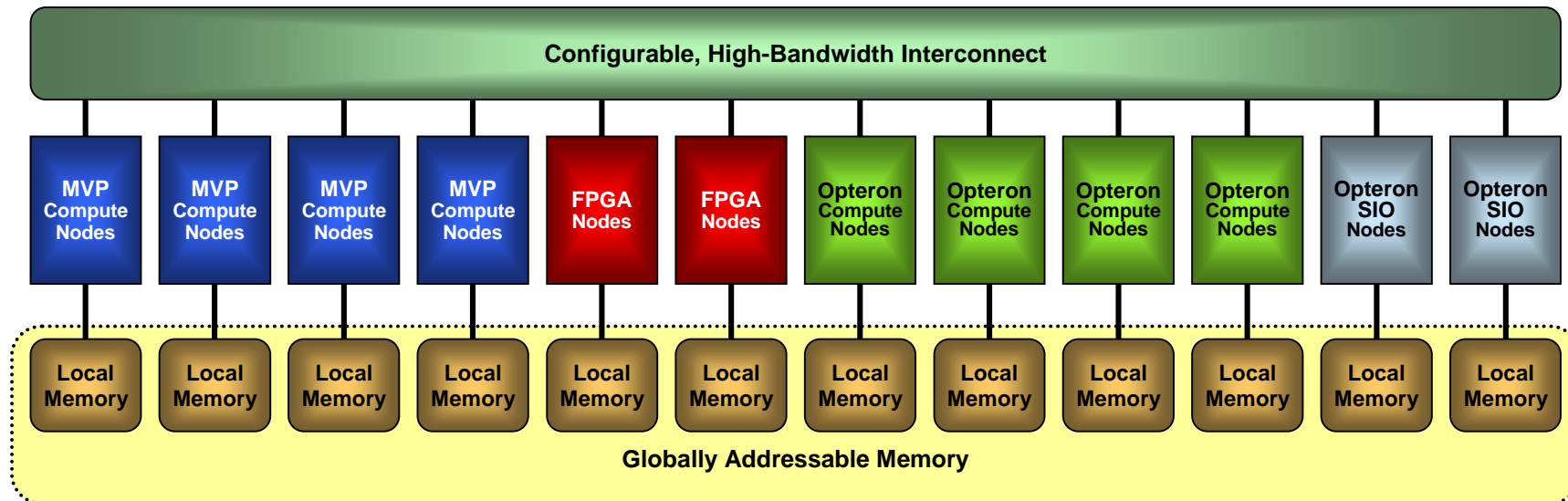
-Grand Challenge Case Study
-High Performance Computing & Competitiveness
-Sponsored by the Council on Competitiveness



Opportunities to Exploit Heterogeneity

- Applications vary considerably in their demands
- Any HPC application must contain some form of parallelism
 - Many HPC apps have rich, SIMD-style *data-level parallelism*
 - Can significantly accelerate via **vectorization**
 - Those that don't generally have rich *thread-level parallelism*
 - Can significantly accelerate via **multithreading**
 - Some parts of applications are not parallel at all
 - Need fast **serial scalar** execution speed (Amdahl's Law)
- Applications also vary in their *memory and network bandwidth* needs
 - Amount of bandwidth
 - Dense vs. sparse
- Opportunity to exploit heterogeneity at the....
workload, workflow, or even loop-nest granularity

Cascade System Architecture

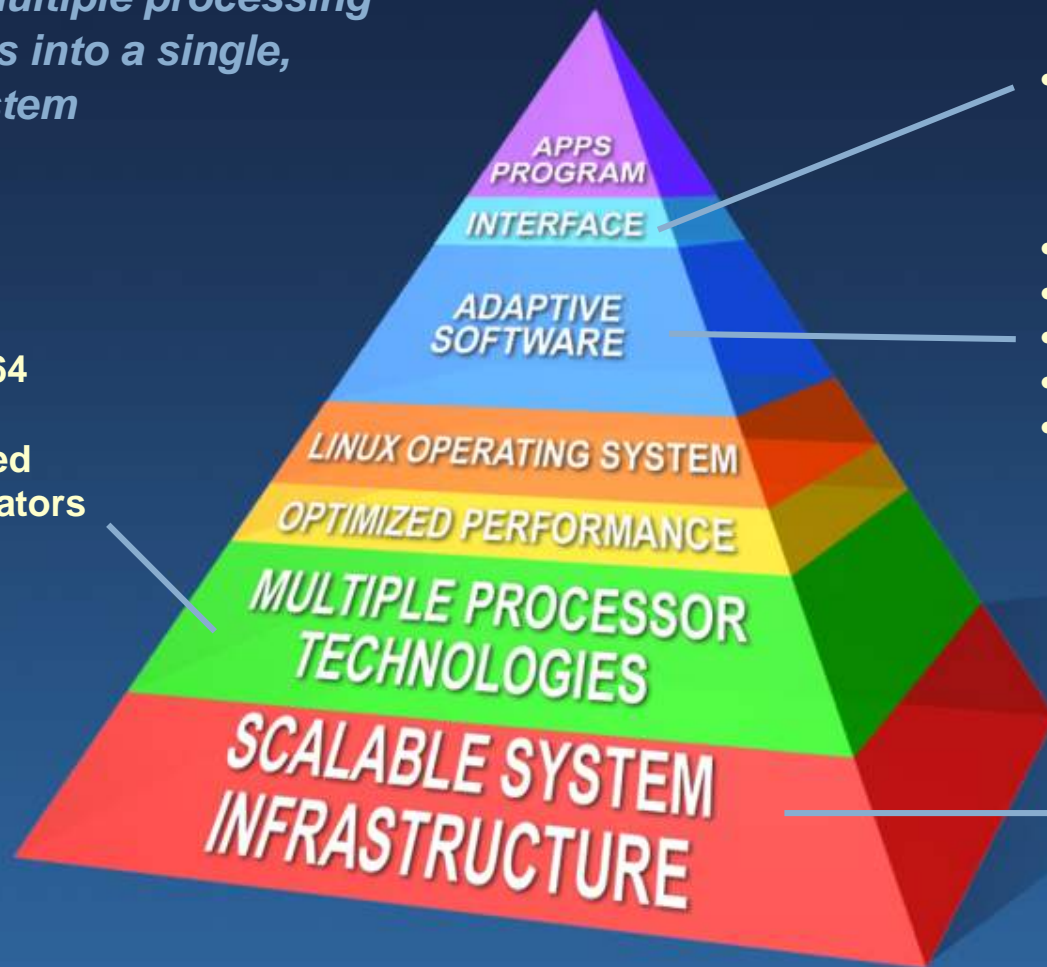


- Globally addressable memory with unified addressing architecture
- Heterogeneous processing across node types
- Adaptive MVP processors that incorporate Opterons + vector and multithreaded acceleration

*Let the machine to adapt to fit the code,
rather than making the code adapt to fit the machine.*

Adaptive Supercomputing

Combines multiple processing architectures into a single, scalable system



- Scalar X86/64
- Vector
- Multithreaded
- HW Accelerators

- Transparent Interface

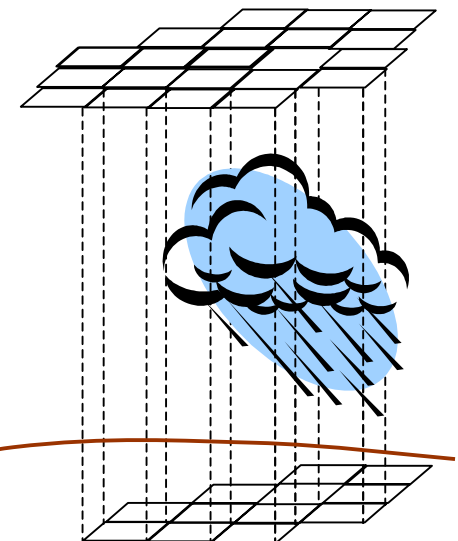
- Libraries Tools
- Compilers
- Scheduling
- System Management
- Runtime

- Interconnect
- File Systems
- Storage
- Packaging

Adapt the system to the application – not the application to the system

Example Application: Weather Research & Forecasting (WRF) Model

- Mesoscale numerical weather prediction system
 - Regional forecast model (meters to thousands of kilometers)
- Operational forecasting, environmental modeling, & atmospheric research
 - Key commercial application for Cray (both vector & scalar MPP systems)
- Accelerating WRF performance on Cascade:
 - Part of the code is serial:
 - Runs on Opteron for best-of-class serial performance
 - Most of the code vectorizes really well
 - Dynamics and radiation physics
 - ⇒ Runs on MVP accelerator in vector mode
 - Cloud physics doesn't vectorize
 - Little FP, lots of branching and conditionals
 - Degrades performance on vector systems
 - Vertical columns above grid points are all independent
 - ⇒ Runs on MVP accelerator in multithreaded mode



Summary of Major Challenges for HPC

- Scaling, scaling, scaling
 - $\frac{d^2}{dt^2}$ (*system_size*) is positive \Rightarrow million processor systems coming...
 - Implications on cost, reliability, management, performance, programming
- At the same time, components are becoming *less* reliable
 - Technology, architecture, system software, applications
- Power is becoming a (*the?*) major issue
 - Cooling, electricity bill (\$1M per MWatt-year!)
- Figuring out the right on-chip parallel architecture
 - Many interesting alternatives...
 - Tension between **efficiency** and **generality/ease of programming**
 - One size doesn't fit all \Rightarrow heterogeneous solutions
- Ease of use continues to be a *major* issue
 - Lack of good, portable parallel programming models and tools
 - We've *got* to be able to do better than MPI
 - Requiring programmers to adapt to machine (vs. the opposite)
 - Need software to make heterogeneous computing transparent to users

CRAY

The Supercomputer Company

Thank you.

Questions?

CRAY

