Chapter 1: Perspectives

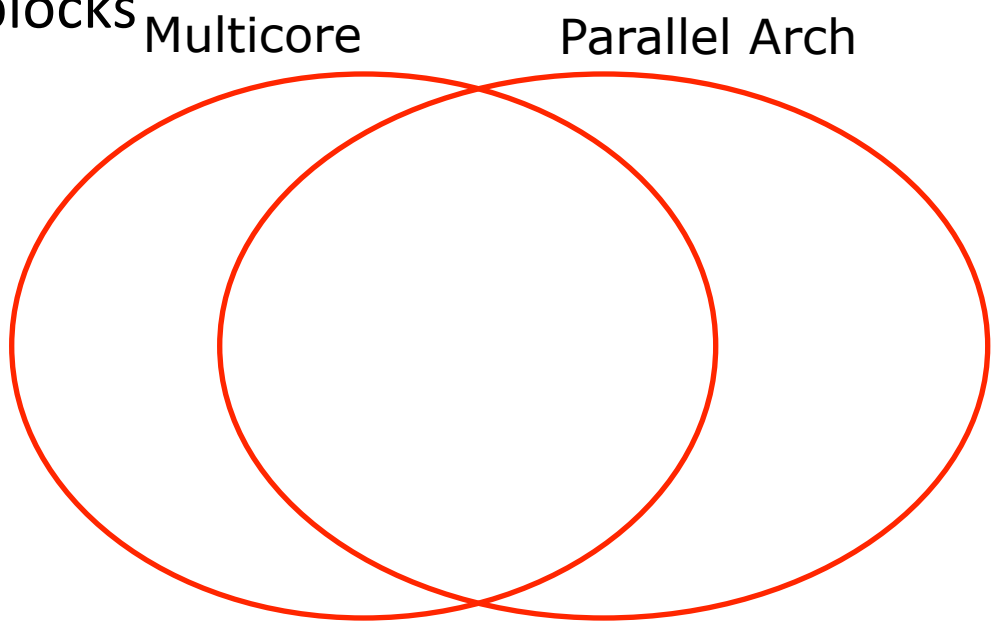
Module 1.1: The Switch to Multicore

Definitions

- Parallel Architecture = an architecture where CPUs are tightly integrated enough that they can work to solve a single problem
- What is a CPU (also known as a core)?
  - A CPU typically refers to a processing element capable of independently fetching and executing instructions from at least one instruction stream.
  - A core typically includes logic such as instruction fetch unit, program counter, instruction scheduler, functional units, register file, etc.
  - Does it include the L1 caches? Yes and **no**
  - Does it include the L2 cache? Yes and **no**

What is a Multicore?

- Multicore = a chip with multiple CPUs/cores integrated on a single die
- Most multicores are parallel architectures, parallel architectures typically have multicores as the smallest building blocks



Recent Multicores

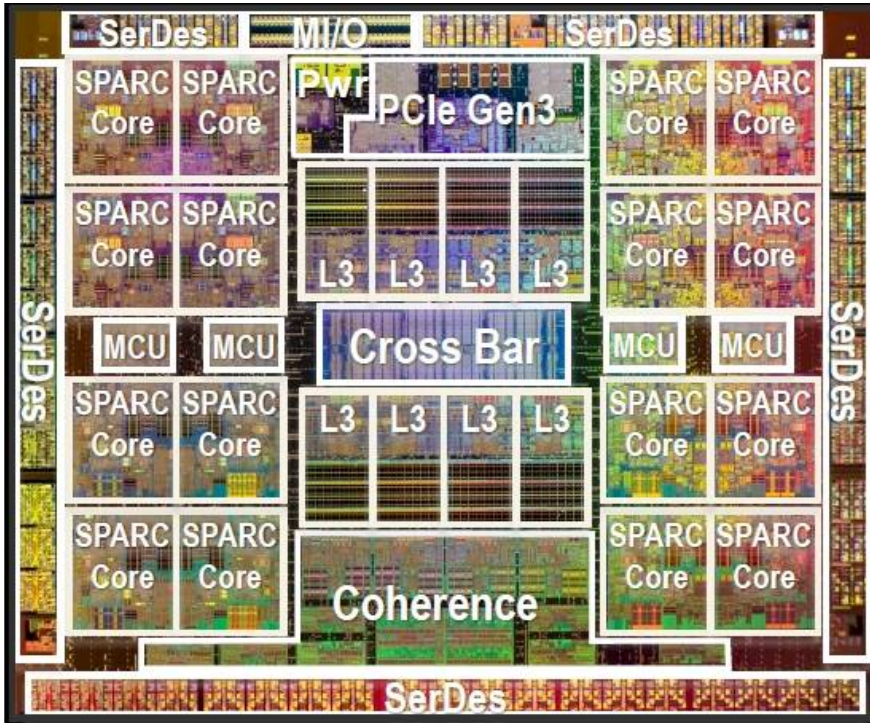
Table 1.2: Examples of recent multicore processors.

Aspects	**Intel IvyBridge**	**Oracle Sparc T5**	**IBM Power7+**
# Cores	6 superscalar 2-way SMT cores	12 2-issue superscalar cores	8 superscalar, 4-way SMT, cores, plus accelerators
Technology	22nm	28nm	32nm, 13 metal layers, 567mm^2, 2.1B transistors
Clock Freq	up to 4.0GHz	3.6GHz	up to 4.4GHz
Caches	64KB L1, 256KB L2 (per core), up to 15MB L3 (shared)	16KB L1 (per core), 128KB L2 (per core), 8MB L3 (shared)	256KB L2 (per core), 80MB eDRAM L3 (shared)

- Ranging from 6-16 cores, often capable of running multiple threads per core
- 3 levels of cache hierarchy, L3 cache 8MB-80MB

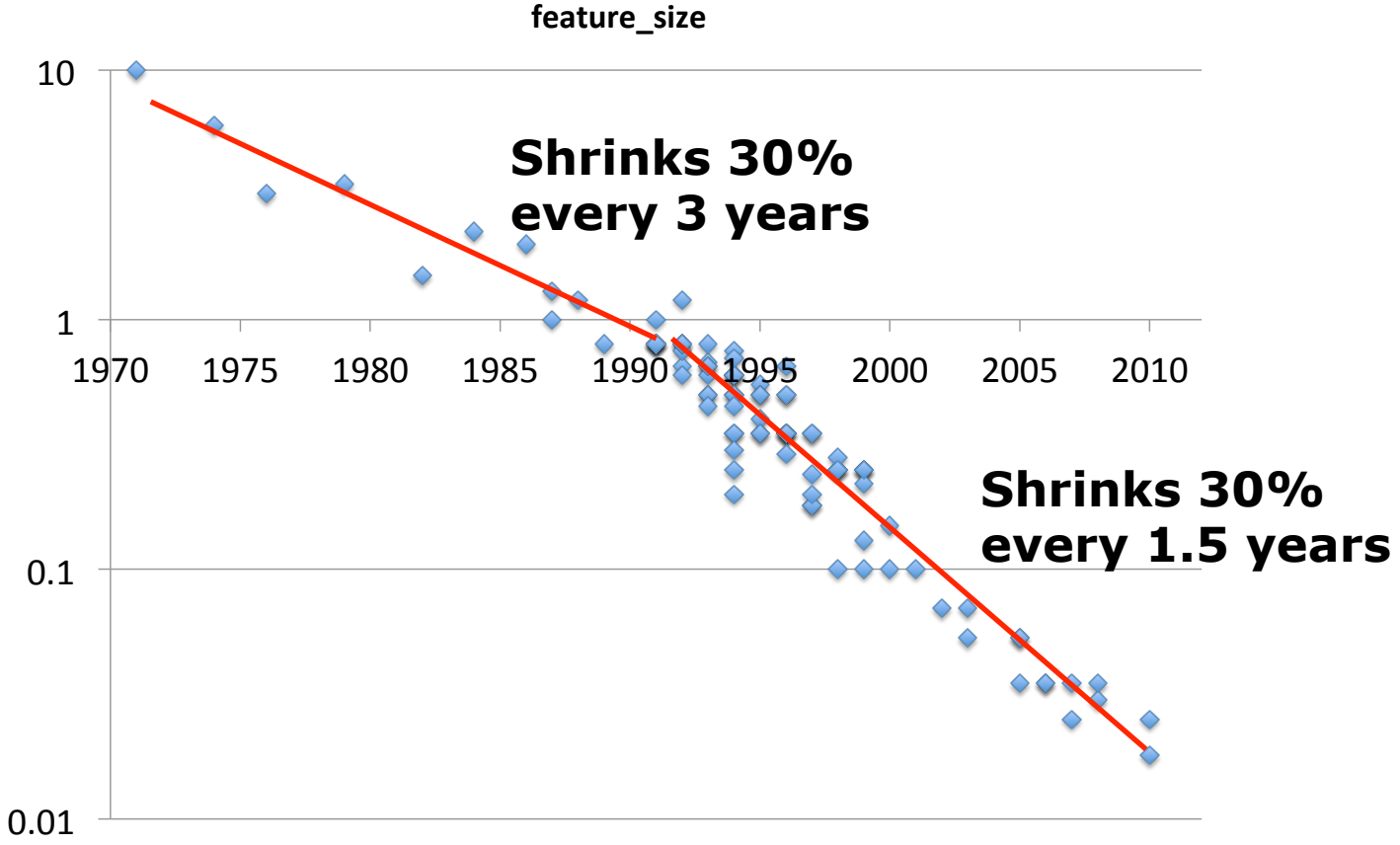
A Closer Look: Oracle T5

- 16 cores
- 8 L3 cache banks
- L1 and L2 caches are “implied” as parts of a core



- Starting 2001-2005, chipmakers turned to multicore design approach. Why?

Feature Size Scaling



- Result of scaling: 2,300 transistors (1971) to 2.3 billion transistors (2012)

Evolution in Microprocessors

Table 1.1: Evolution of Intel Processors. Other chipmakers also enjoy a similar transistor integration progress scaling according to Moore's law.

Year	Processor	Specifications	New Features
1971	4004	740 KHz, 2300 transistors, 10μm, 640B addressable memory, 4KB program memory	
1978	8086	16-bit, 5–10MHz, 29,000 transistors at 3μm, 1MB addressable memory	
1982	80286	8–12.5MHz	Virtual memory and protection mode
1985	386	32-bit, 16-33MHz, 275K transistors, 4GB addressable memory	Pipelining
1989	486	25-100MHz, 1.2M transistors	FPU integration
1993	Pentium	60–200MHz	on chip L1 caches, SMP support
1995	Pentium Pro	16KB L1 caches, 5.5M transistors	out-of-order execution
1997	Pentium MMX	233-450MHz, 32KB L1 cache, 4.5M transistors	dynamic branch prediction, MMX instruction sets
1999	Pentium III	450-1400MHz, 256KB L2 cache on chip, 28M transistors	SSE instruction sets
2000	Pentium IV	1.4-3GHz, 55M transistors	hyper-pipelining, SMT
2006	Pentium Dual-Core	64-bit, 2GHz, 167M transistors, 4MB L2 cache on chip	Dual core, virtualization support
2011	Sandy Bridge (i7)	64-bit, up to 3.5GHz, 2.37B transistors, up to 20MB L3 cache	graphics processor, TurboBoost

Discussion

- Increasingly more and more components and features can be integrated on a single chip
- "Perhaps" transition from single core to multicore was just a matter of time
- But why did it happen in that sequence?
  - Why did pipelining, branch prediction, superscalar issue, and caches came BEFORE multicore?

- My Answer: **Low Hanging Fruit Theory**
  - The farmer picks up the low hanging fruits first because they require less efforts to pick up
  - … that is until they run out of low hanging fruits

Parallel Multicore is Not Low Hanging

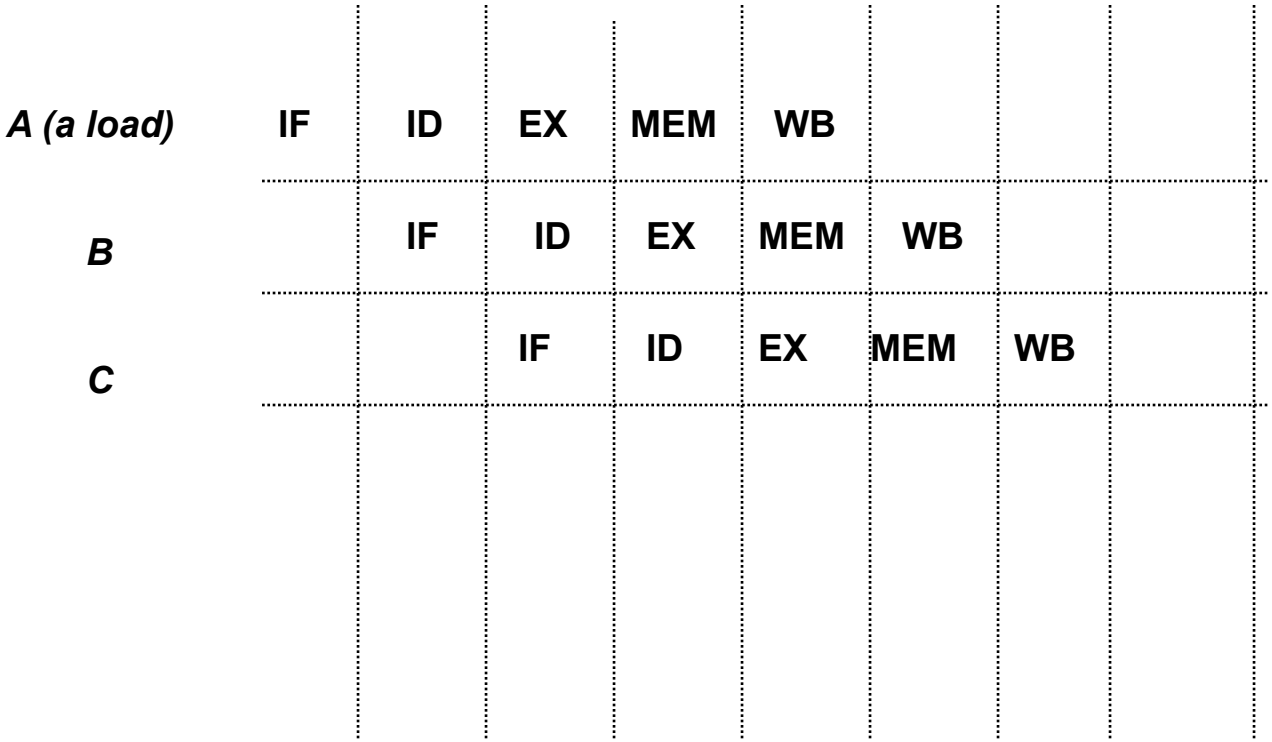
- For the most part, benefiting from multicore requires parallel programming
- Parallel programming requires additional effort over sequential programming
  - Program split into multiple instruction streams (threads)
  - Co-ordinate threads using synchronization
  - Communicate data across threads
  - Result: Thread Level Parallelism (TLP)
- vs. Instruction Level Parallelism (ILP)
  - No (or little) programming efforts
  - Programmers can use sequential paradigm, the way our brain likes it

- So why the switch from ILP to TLP? It must be because we run out of low hanging ILP fruits!

ILP Fruits

- Instruction Level Parallelism (ILP)
  - Pipelining: RISC, CISC with RISC backend
  - Superscalar
  - Out of order execution
- Memory hierarchy (Caches)
  - Exploiting spatial and temporal locality
  - Multiple cache levels

ILP Technique: Pipelining



ILP Technique: Superscalar

- Original:

```
LD    F0, 34(R2)
ADDD  F4, F0, F2
LD    F7, 45(R3)
ADDD  F8, F7, F6
```

- Schedule as:

```
LD    F0, 34(R2)   | LD   F7, 45(R3)
ADDD  F4, F0, F2   | ADDD F8, F0, F6
```

- Moderate degree of parallelism (theoretically ~50, but realistically ~2 inst/cycle)
- Requires fast communication (register level)

TLP Technique: loop parallelism

- Each iteration can be computed independently

```
for (i=0; i<8; i++)
  a[i] = b[i] + c[i];
```

- Each iteration cannot be computed independently, thus does not have loop level parallelism

```
for (i=0; i<8; i++)
  a[i] = b[i] + a[i-1];
```

- Very high parallelism > 1K
- Often easy to achieve load balance
- Usu. requires splitting into multiple instruction streams
- Some loops are not parallel
- Some apps do not have many loops

TLP Technique: function parallelism

- Arbitrary code segments in a single program
- Across loops:

```
…
for (i=0; i<n; i++)
  sum = sum + a[i];
for (i=0; i<n; i++)
  prod = prod * a[i];
…
```

- Subroutines:

```
Cost = getCost();
A = computeSum();
B = A + Cost;
```

- Threads: e.g. editor: GUI, printing, parsing
- Larger granularity => low overheads, communication

- Low degree of parallelism
- Hard to balance

Program level parallelism

- Various independent programs execute together
- gmake:
  - gcc –c code1.c // assign to proc1
  - gcc –c code2.c // assign to proc2
  - gcc –c main.c // assign to proc3
  - gcc main.o code1.o code2.o

+ no communication

- Hard to balance
- Few opportunities

No Longer Low Hanging?

- Increasing *pipeline depth* counter-productive [Hartstein & Puzak, 2002]
  - Latching overheads becoming significant
  - Too little time in a pipeline stage to do useful logic
  - Critical loop sizes increase proportionally to pipeline depth (branch misprediction loop, data dependence loop, cache miss loop)
- Increasing *pipeline width* incurs quadratic increase in complexity [Palacharla & Smith, 1996]
  - Dependence checking logic, wakeup & select logic, register ports, bypass logic, etc.
- Power efficiency declines significantly

Module 1.2: Power Wall

Basics of Power and Energy

- Energy = the ability of a physical system to do work on other physical systems (unit: joule)
  - 1 joule = energy to lift an apple to shoulder height
  - 4 joules = energy to heat a teaspoon of water by 1 degree Celcius
- Power = rate at which energy is transferred
  - Unit: 1 watt = 1 joule delivered in 1 second
  - Power = V * I
- For a capacitor
  - Energy_stored = 0.5 C V^2
  - V is voltage, C is capacitance
- If capacitor is drained and charged at a frequency of f per second
  - Power = Energy/second = 2 * 0.5 CV^2 * f = CV^2f
  - The above is referred to as *dynamic power consumption*

Energy Efficiency

- Energy efficiency
  - how much energy required to perform a work
    - Unit: energy unit/work unit,
    - e.g. EPI (Energy per Instruction)
  - ratio of how fast energy is consumed vs. how fast work gets done
    - Unit: power unit/throughput unit
      - E.g. Watt/IPC, Watt/IPS, Watt/FLOPS
    - Watt/IPS = (Joule/sec)/(Inst/sec) = Joule/Inst = EPI
    - Watt/thruput = (Joule/sec)/(work/sec) = Joule/work = amount of energy required to perform work

Other Metrics

- Two other metrics common in publications
- Energy Delay Product (EDP): what is it?
  - Energy/work most relevant in battery-powered systems
  - But performance matters, e.g. would you choose system A that gets work done at 0.5 watt in 2 seconds, or system B that gets work done at 1 watt in 1 second? Both use 1 joule of energy, but most will prefer the latter.
  - Hence, weight energy with delay by multiplying them.
- Energy Delay2 Product (ED2P): what is it?
  - Put even more emphasis on delay, favoring reduction in delay more than reduction in energy.

Power Wall Problem

- Power Consumption = Dynamic + Static
- Determined by packaging constraints (e.g. 65W for desktop)
- Dynamic Power
  - Power lost from charging/discharging capacitive loads

$$Pdyn = ACV^2 f$$

  - A: fraction of gates actively switching
  - C: total capacitance of all gates
  - V: supply voltage, f: frequency of switching
- Static Power
  - Power lost from leakage

$$Pstatic = VI_{leak}$$

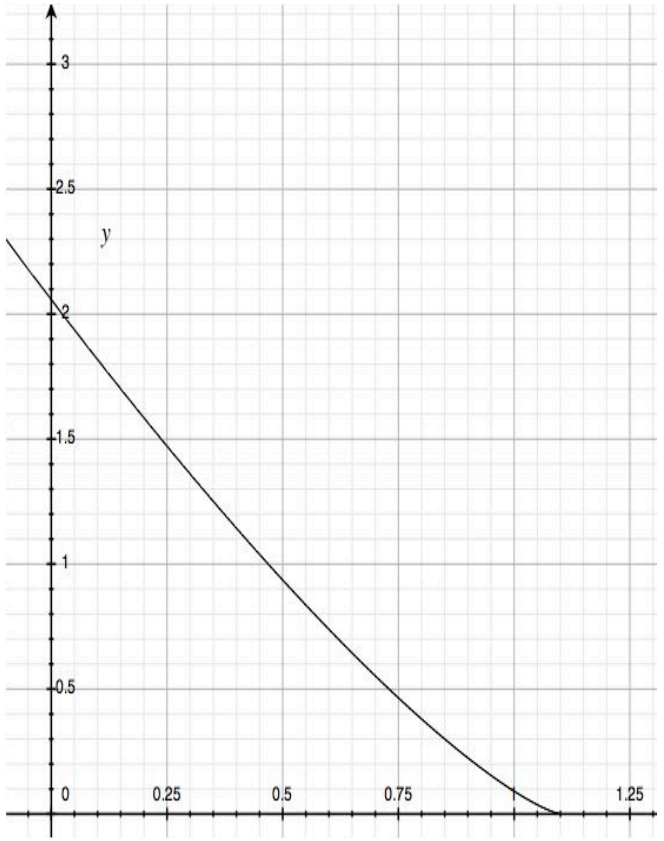
  - I_{leak}: leakage current

Power Wall Fundamentals

- Max frequency vs. threshold voltage

$$f_{\max} \approx c\frac{(V - V_{thd})^{1.3}}{V}$$

  - V_{thd}: threshold voltage, minimum Voltage to make transistor conducive

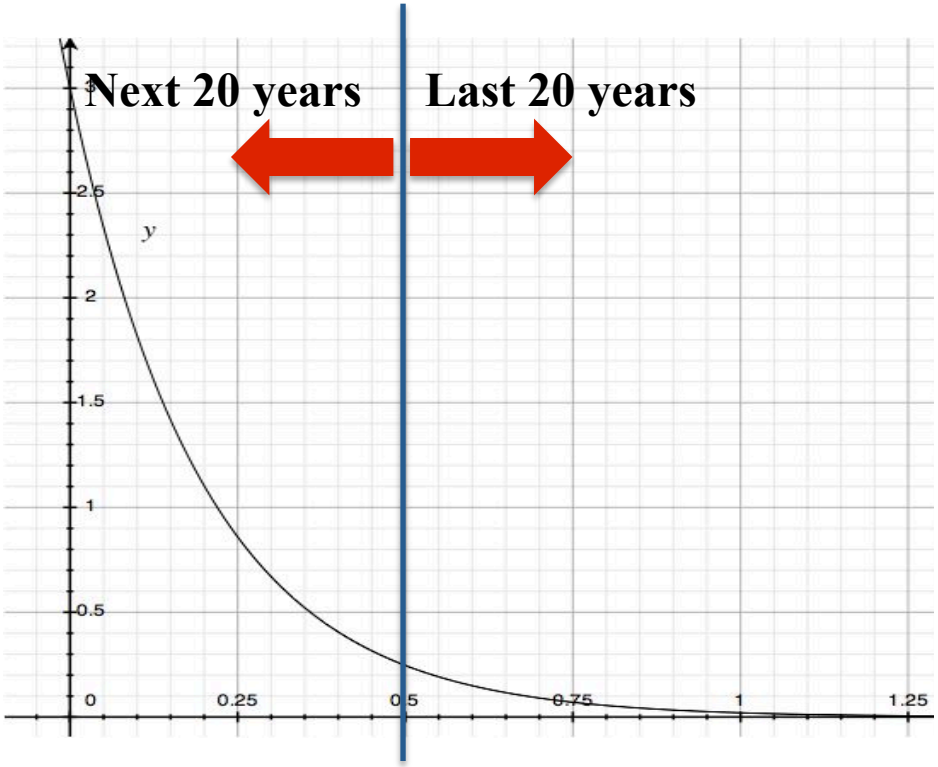


Reducing threshold voltage increases maximum frequency almost linearly

Power Wall Fundamentals

- Static Power Consumption
  - Leakage current:

$$I_{leak} > e^{-\frac{qV_{thd}}{kT}}$$



Reducing threshold voltage increases static power consumption exponentially

Threshold Voltage Dilemma

- Dilemma
  - Decreasing Vthd improves f_{max} (and hence, performance)
  - Decreasing Vthd increases leakage current (hence static power consumption) exponentially

Old Scaling Rule

- Gate length (transistor size) scales by S = 0.7
- Capacitance scales by S = 0.7
- Original area scales by S^2 = 0.5,
- number of transistors scales by $1/S^2$ = 2
- Supply voltage (V) scales by S = 0.7
- Frequency (f) scales by 1/S = 1.4
- Dynamic power of chip remains constant:

$$DynP = ACV^2 f$$

$$DynP' = A'C'V'^2 f'$$

$$DynP' = (2A)(0.7C)(0.7V)^2(1.4f)$$

$$DynP' = ACV^2 f = DynP$$

New Scaling Rule

- Capacitance scales by S = 0.7
- Number of transistors scales by $1/S^2 = 2$
- Supply voltage (V) cannot scale without also scaling Vthd
  - but scaling Vthd increases static power exponentially
- Frequency (f) scales by 1/S = 1.4
- Dynamic power of chip doubles every generation!

$$DynP = ACV^2 f$$

$$DynP' = A'C'V'^2 f'$$

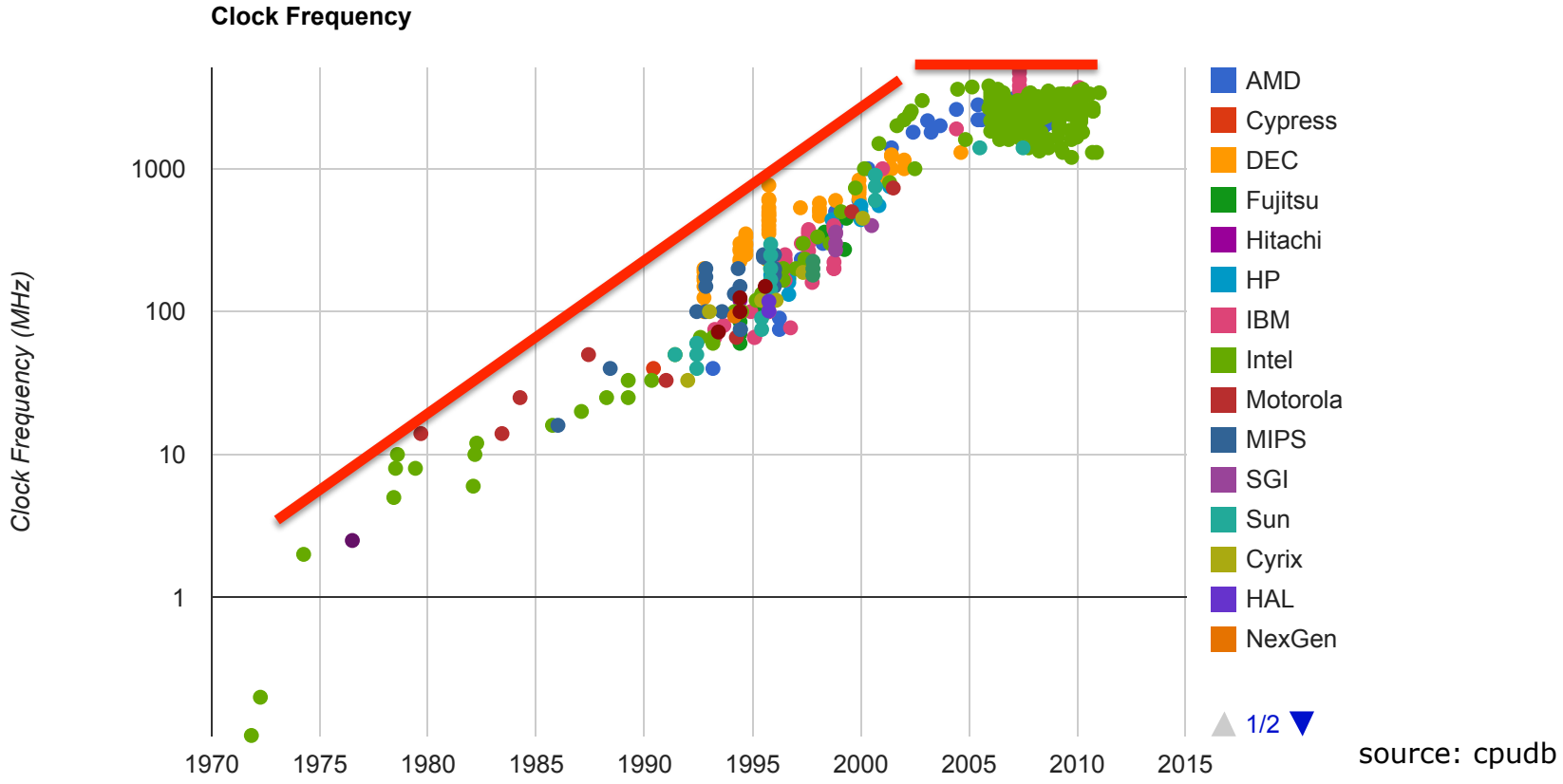
$$DynP' = (2A)(0.7C)(V)^2(1.4f)$$

$$DynP' = 2 \cdot DynP$$

Implications for Architecture Design

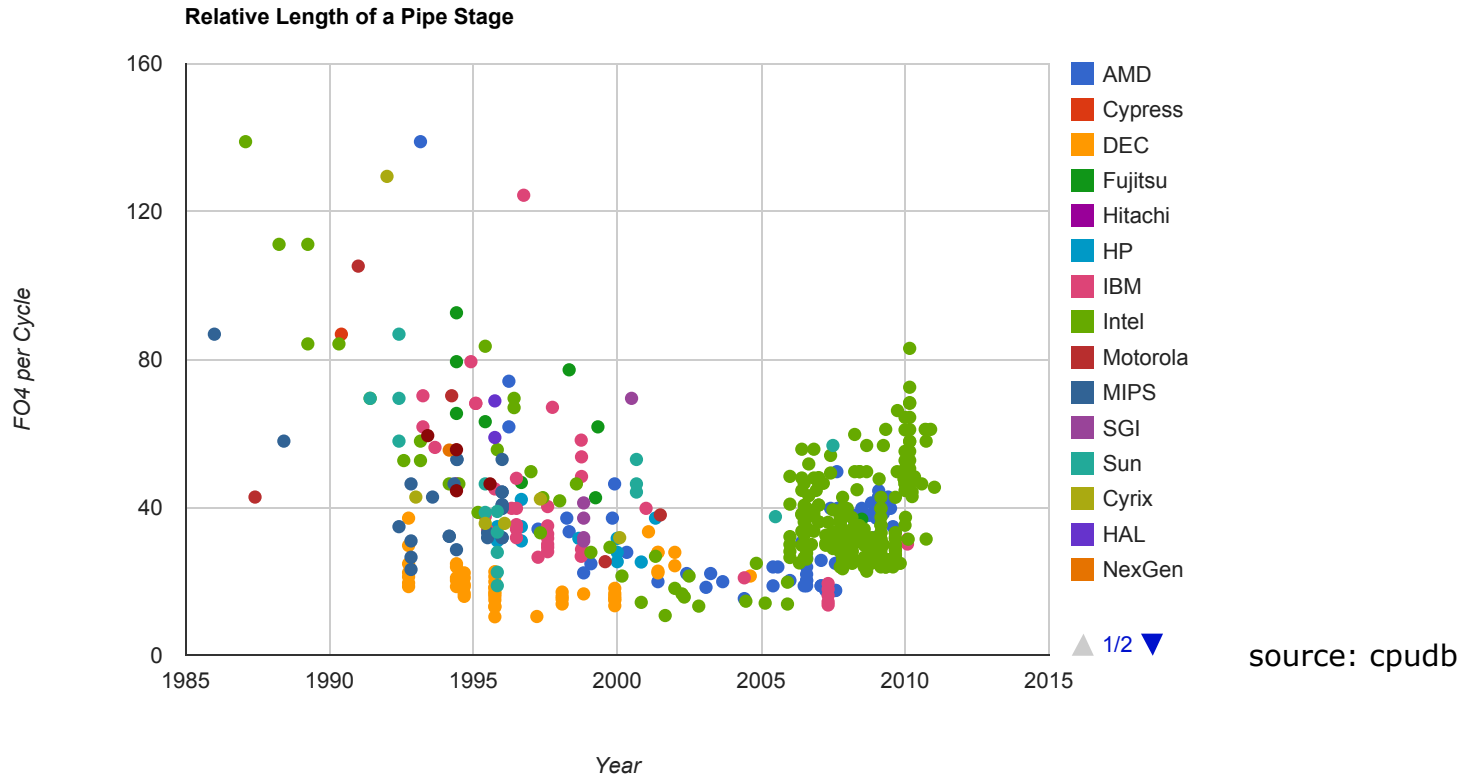
- Increasing clock frequency has hit a dead end
- Area is abundant but power is scarce
- Declining power budget per core
- New architecture must improve energy efficiency
- Dominant metric: performance/watt or ops/joule

Clock Frequency Stagnant?



- 2001-2005: same time as transition to multicore

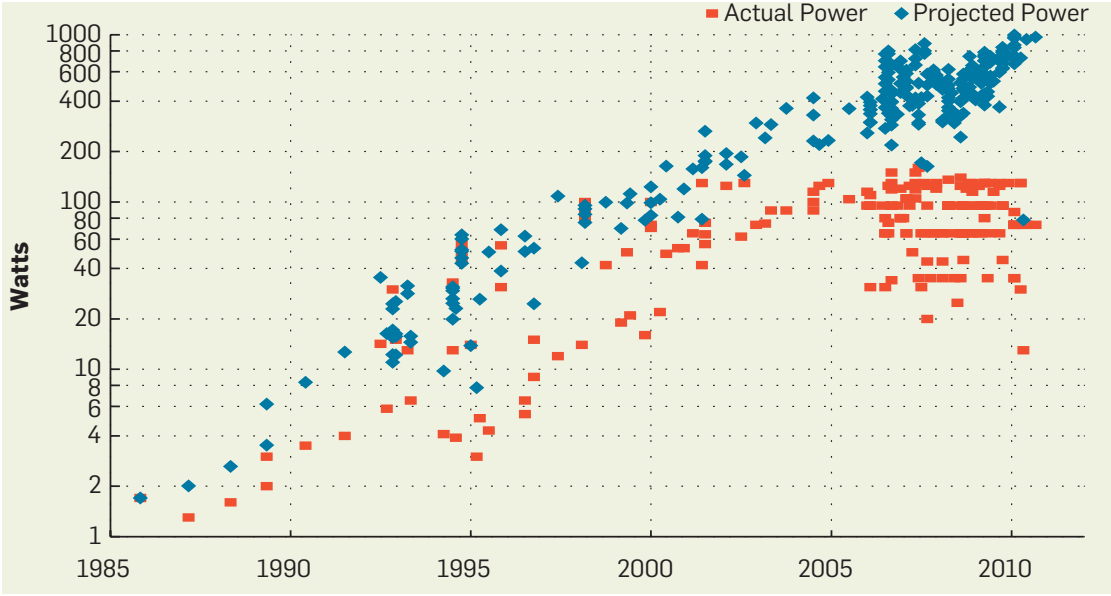
Computation per Pipeline Stage



source: cpudb

- 2001-2005: pipelining started to *reverse* (in terms of amount of logic per pipeline stage

Power Saving Mechanisms



source: cpudb

- Keeping clock frequency constant is insufficient
- Other power saving techniques were employed to contain power: clock gating, etc.

Exercise

- 1. Suppose that instead of progressing at a ratio of 0.7, Moore's law slows down and transistor gate length scales at a ratio of 0.8 instead. Find the dynamic power consumption under unlimited and limited scaling for the next process generation.
- 2. With limited voltage scaling, suppose that we want to keep the dynamic power consumption constant in the next generation by keeping frequency constant and reduce die area. How much should we reduce die area to achieve that?

Answer (Q1)

- Old Scaling Rule
  - Gate length (transistor size) scales by S = 0.8
  - Capacitance scales by S = 0.8
  - Original area scales by S2 = 0.64,
  - allowing the number of transistors to scale by 1/S2 = 1.56
  - Supply voltage (V) scales by S = 0.8
  - Frequency (f) scales by 1/S = 1.25
  - Dynamic power of chip remains constant:

$$DynP = ACV^2 f$$

$$DynP' = A'C'V'^2 f'$$

$$DynP' = (1.56A)(0.8C)(0.8V)^2(1.25f)$$

$$DynP' = ACV^2 f = DynP$$

Answer (Q1)

- Under leakage-limited scaling:
  - Capacitance scales by S = 0.8
  - Number of transistors scales by 1/S2 = 1.56
  - Supply voltage (V) cannot scale without also scaling Vthd
    - but scaling Vthd increases static power exponentially
  - Frequency (f) scales by 1/S = 1.25
  - Dynamic power of chip increases 56% each generation

$$DynP = ACV^2 f$$

$$DynP' = A'C'V'^2 f'$$

$$DynP' = (1.56A)(0.8C)(V)^2(1.25f)$$

$$DynP' = 1.56 \cdot DynP$$

Answer (Q2)

- Limited Voltage Scaling
  - Gate length (transistor size) scales by S = 0.7
  - Capacitance scales by S = 0.7
  - Original area scales by $S^2 = 0.5$,
  - Supply voltage (V) and frequency (f) are constant
  - Dynamic power of chip must remain constant:

$$DynP = ACV^2 f$$

$$DynP' = A'C'V'^2 f'$$

$$ACV^2 f = A'(0.7C)V^2 f$$

$$A = 0.7A'$$

  - Die area shrinks by 1-1.4/2.0 = 30%

Module 1.3: Parallel Computers

What is a Parallel Architecture?

"A parallel computer is a collection of processing elements that can communicate and cooperate to solve a large problem fast."

-- Almasi & Gottlieb

Parallel computers

- A parallel computer is a **collection of processing elements** that can **communicate** and **cooperate** to **solve a large problem fast**. [Almasi&Gottlieb]
- "collection of processing elements"
  - What is a processing element? Logic that fetches instructions from a single program counter and each instruction operates on a single set of data items.
  - How many? How powerful each? Scalability?
  - Few very powerful (e.g., Altix) vs. many small ones (BlueGene)
- "communicate"
  - How do PEs communicate? (shared memory vs. msg passing)
  - Interconnection network (bus, multistage, crossbar, …)
  - Choices determine cost, latency, throughput, scalability, and fault tolerance

- "and cooperate"
  - Synchronization allows sequencing of operations to ensure correctness
  - Issues: granularity and mechanisms
  - Choices affect latency, load balance, scalability, and fairness
- "solve a large problem fast"
  - General vs. special purpose machine?
  - Any machine can solve certain problems well

Usage of Parallel Computers

What domains?

– Highly (embarassingly) parallel apps
  - Many scientific codes

– Medium parallel apps
  - Many engineering apps (finite-elements, VLSI-CAD)

– Not parallel apps
  - Compilers, editors

Why parallel computers?

- Absolute performance: Can we afford to wait?
  - Folding of a single protein takes years to simulate on the most advanced microprocessor. It only takes days on a parallel computer
  - Weather forecast: timeliness is crucial
- Cost-adjusted performance or Power-adjusted performance
  - Especially for small systems with 2-8 chips
  - The smallest system is a multicore processor
  - Large parallel computers are rarely cost-performance effective

Why parallel computers were not mainstream until multicore

- Low hanging ILP fruits mean it was easy to scale performance through ILP
- This made parallel computers unattractive in terms of cost-performance
  - Expensive (cost and time) to design scalable parallel systems
  - Performance from parallelism eclipsed by ILP in just a few years
- This went on until ILP fruits were higher hanging than TLP

Illustration

- 100-processor system with perfect speedup
- Compared to a single processor system
  - Year 1: 100x faster
  - Year 2: 62.5x faster
  - Year 3: 39x faster
  - …
  - Year 10: 0.9x faster
- Single processor performance catches up in just a few years!
- Even worse
  - It takes longer to develop a multiprocessor system
  - Low volume means prices must be very high
  - High prices delay adoption
  - Perfect speedup is unattainable

Flynn Taxonomy

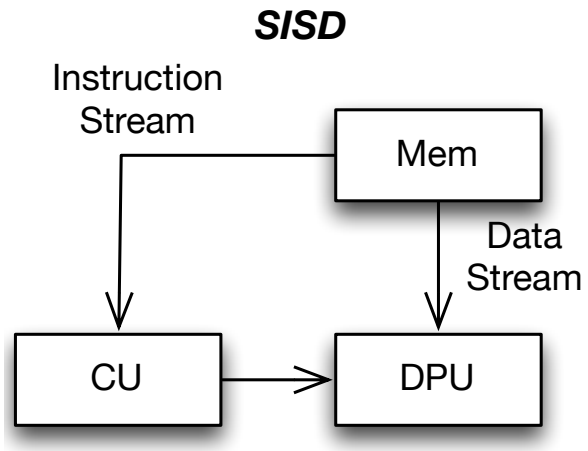
Table 1.3: Flynn's Taxonomy of Parallel Computers

		Number of Data Streams	
		Single	Multiple
Number of Instruction Streams	Single	***SISD***	***SIMD***
	Multiple	***MISD***	***MIMD***

- Instruction stream = instructions fetched from a single program counter
- Data stream = data items accessed by a single instruction

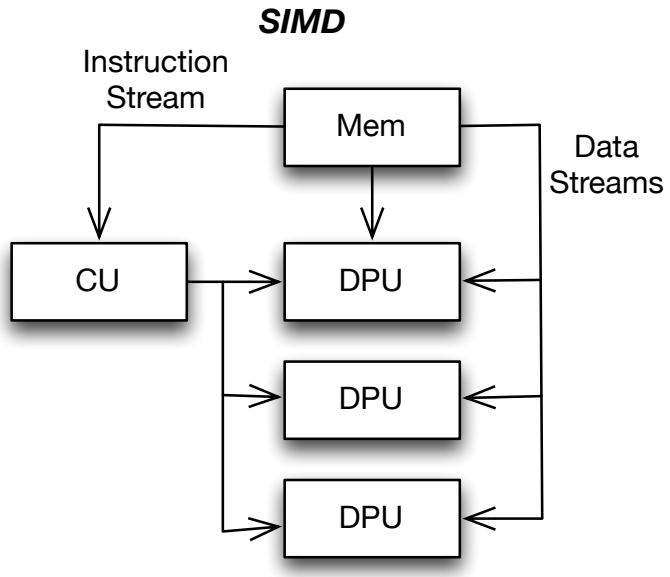
SISD

- SISD = Single Instruction Stream, Single Data Stream
  - Only one program counter (fetched by Control Unit/CU)
  - Each instruction accesses one set of data (e.g. 2 operands and 1 result)
  - Data processed by Data Processing Unit (DPU)



SIMD

- SIMD = Single Instruction Stream and Multiple Data Streams
- Examples: Vector processors, multimedia extension



- Multimedia extension use SIMD-style execution
  - Rather than operating on many sets of data items, data items are packed into a single set of regular registers

Example SSE Code

SISD style

```
for (i=0;i<4;i++)
  C[i] = A[i] + B[i]
```

SIMD style

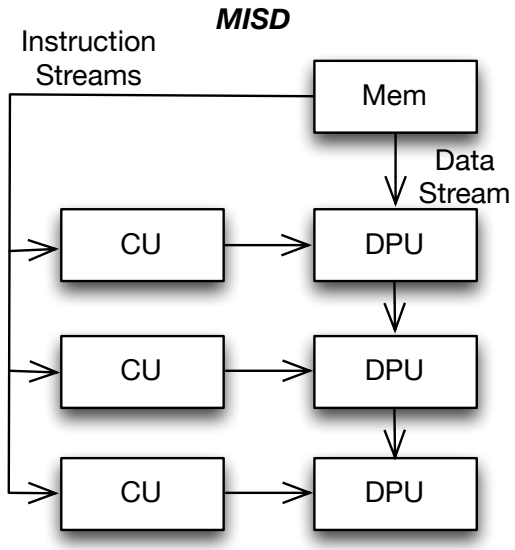
```
C = A + B
```

Example SSE Code with 4 32-bit data packed into 128-bit register:

```
Vector4 SSE_Add ( const Vector4 &Op_A, const Vector4 &Op_B )
{
        Vector4 Ret_Vector;

        __asm
        {
                MOV EAX Op_A                        // Load pointers into CPU regs
                MOV EBX, Op_B

                MOVUPS XMM0, [EAX]                  // Move unaligned vectors to SSE regs
                MOVUPS XMM1, [EBX]

                ADDPS XMM0, XMM1                    // Add vector elements
                MOVUPS [Ret_Vector], XMM0      // Save the return vector
        }

        return Ret_Vector;
}
```

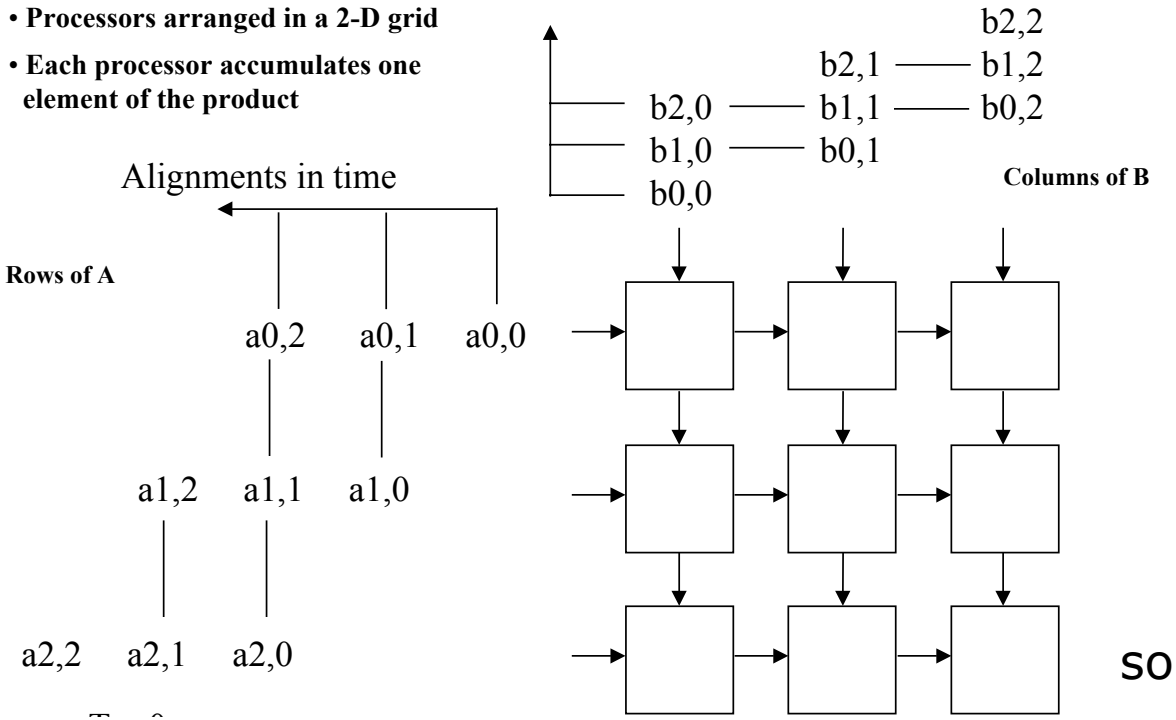
MISD

- MISD = Multiple Instruction Streams and Single Data Stream
- e.g. Systolic arrays



Systolic Arrays (contd.)

Example: Systolic array for Matrix Multiplication

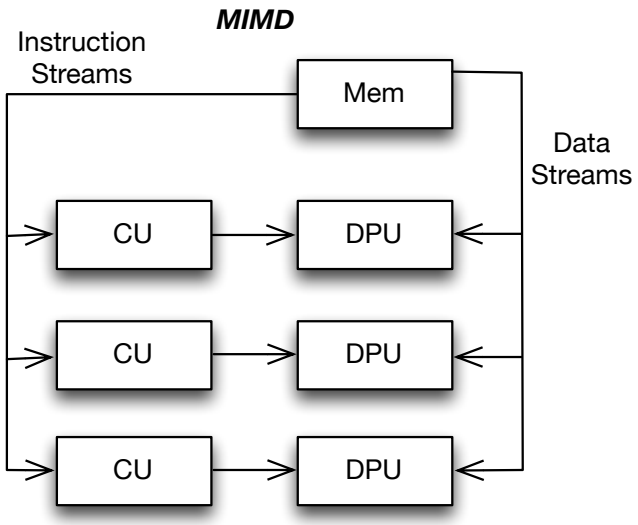


source: Shaaban

– Intel manufactured iWARP, using general processors with dedicated interconnect channels for transferring data from register to register

MIMD

- Independent processing elements connected together
  - Each PE fetches from its own instruction stream and processes its own data stream
- Most parallel computers today are MIMD



Variants of MIMD

- Variants of MIMD differ in physical organization: which memory hierarchy level is shared?

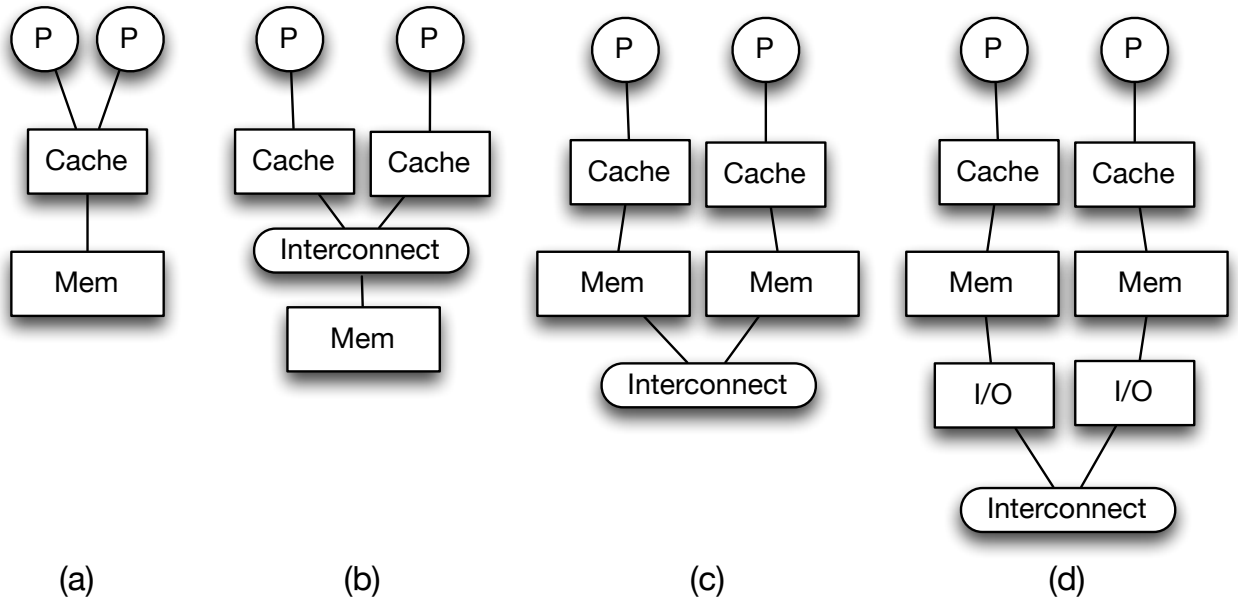


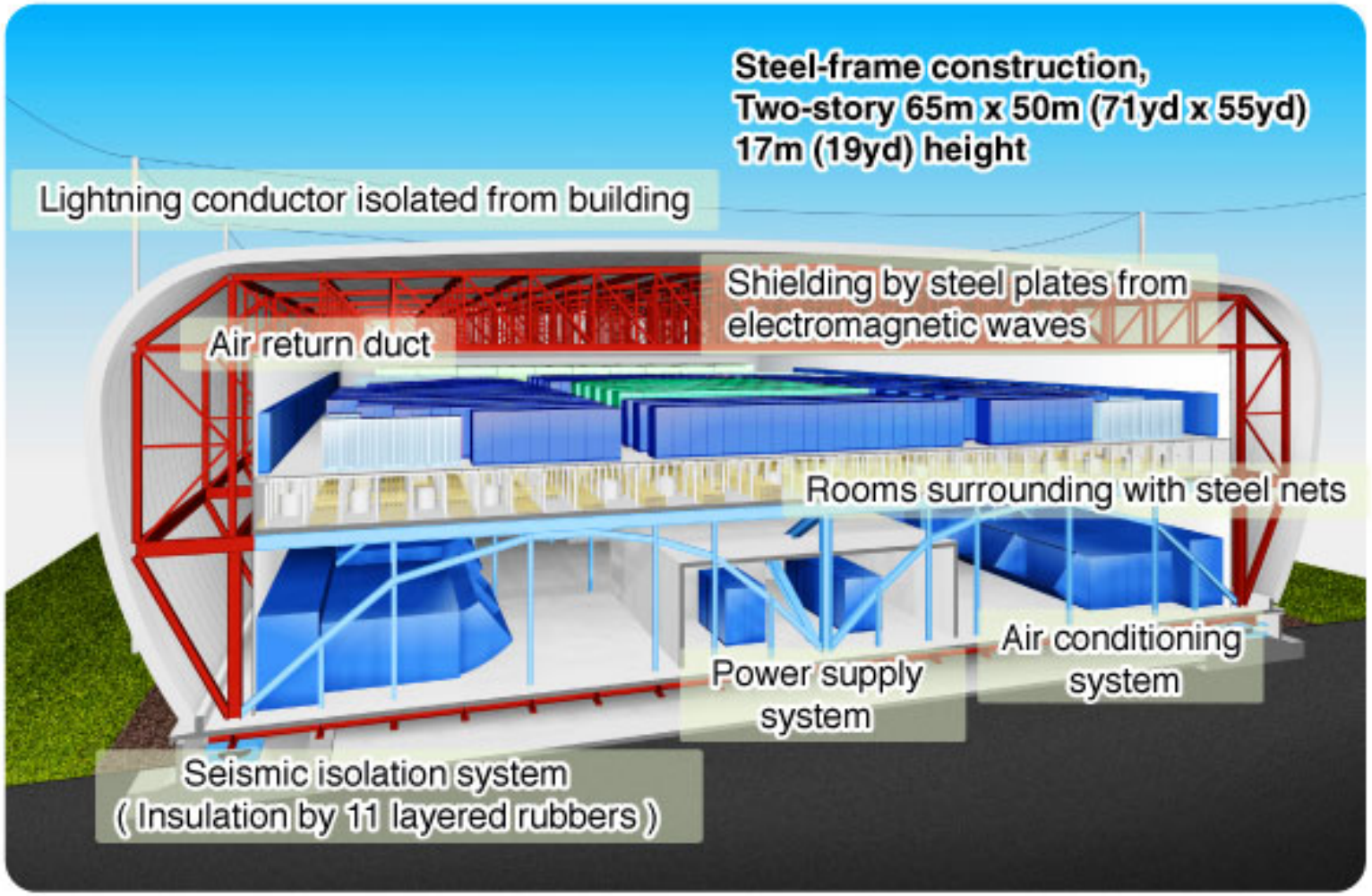
Figure 1.7: Classes of MIMD parallel computers: shared cache (a), uniform memory access (b), non-uniform memory access (c), and distributed system (d).

Top 500 Parallel Computers

- Maintained at http://www.top500.org
- Fast changing
  - Earth Simulator #1 in 2004 became #10 two years later
- Engineering marvel, consuming as much electricity as a city
- Building, plumbing, and cooling often co-designed with the computers

Earth Simulator (2004)

- “The machine room sits at approximately 4th floor level. The 3rd floor level is taken by hundreds of kilometers of copper cabling, and the lower floors house the air conditioning and electrical equipment. The structure is enclosed in a cooling shell, with the air pumped from underneath through the cabinets, and collected to the two long sides of the building. The aeroshell gives the building its "pumped-up" appearance. The machine room is electromagnetically shielded to prevent interference from nearby expressway and rail. Even the halogen light sources are outside the shield, and the light is distributed by a grid of scattering pipes under the ceiling. The entire structure is mechanically isolated from the surroundings, suspended in order to make it less prone to earthquake damage. All attachments (power, cooling, access walkways) are flexible. “


Steel-frame construction,
Two-story 65m x 50m (71yd x 55yd)
17m (19yd) height
Lightning conductor isolated from building
Shielding by steel plates from electromagnetic waves
Air return duct
Rooms surrounding with steel nets
Air conditioning system
Power supply system
Seismic isolation system
(Insulation by 11 layered rubbers)

- Linpack performance: 40 TFlops, 80% peak
- Real world performance: 33-66% peak (vs. less than 15% for clusters)
- Cost? Hint: starts with 4
- Maintenance $15M per year
- Failure one processor per week
- Distributed Memory Parallel Computing System which 640 processor nodes interconnected by Single-Stage Crossbar Network

Fastest (#1 as of June 2013)

- Tianhe-2 (National University of Defense Technology of China)
- 33.8 tera floating point operations per second (TFlops) with Linpack
- 3,120,000 cores, organized into 16,000 nodes
- Each node has 195 cores (2 Intel Xeon IvyBridge + 3 Xeon Phi)
- ~18 MW of electricity