CMPEN 431
Computer Architecture
Fall 2018

Introduction to Message Passing Multiprocessors

Jack Sampson(www.cse.psu.edu/~sampson)

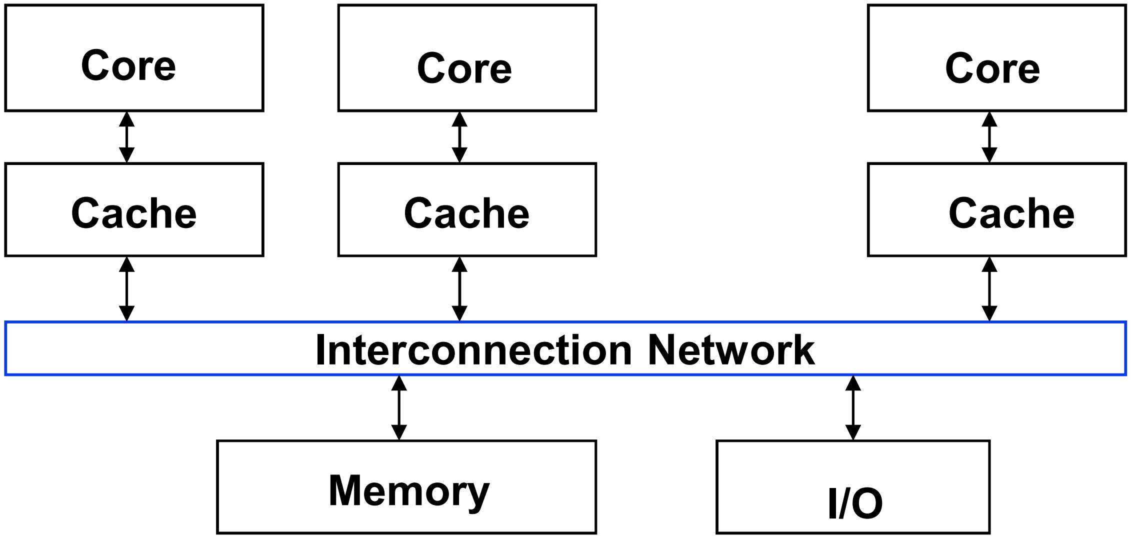
[Slides adapted from work by Mary Jane Irwin, in turn adapted from *Computer Organization and Design, Revised 4th Edition*,

Patterson & Hennessy, © 2011, Morgan Kaufmann and *5th Edition*,

Patterson & Hennessy, © 2014, MK]

Review: Shared Memory Multiprocessors (SMP)

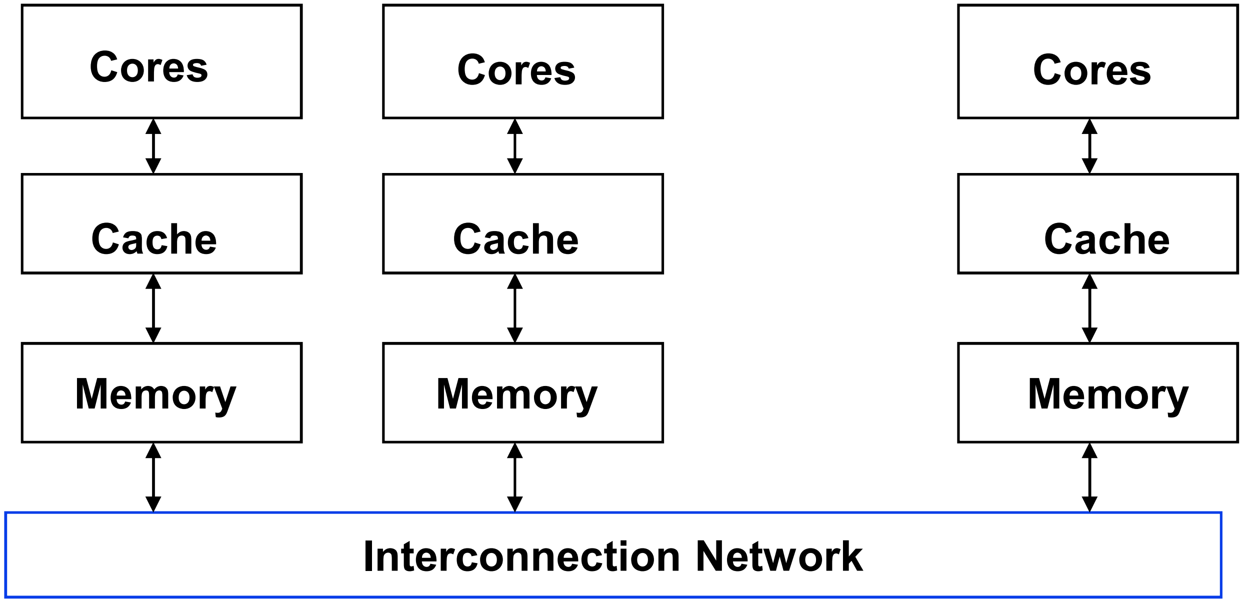
- Q1 – Single address space shared by all cores
- Q2 – Cores coordinate/communicate through shared variables in memory (via loads and stores)
  - Use of shared data must be coordinated via synchronization primitives (locks) that allow access to data to only one core at a time



- SMPs come in two styles
  - Uniform memory access (UMA) multiprocessors
  - Nonuniform memory access (NUMA) multiprocessors

Message Passing Multiprocessors (MPP)

- Each core has its own private address space
- Q1 – Cores share data by *explicitly* sending and receiving information (message passing)
- Q2 – Coordination is built into message passing primitives (message send and message receive)



Communication in Network Connected Multi's

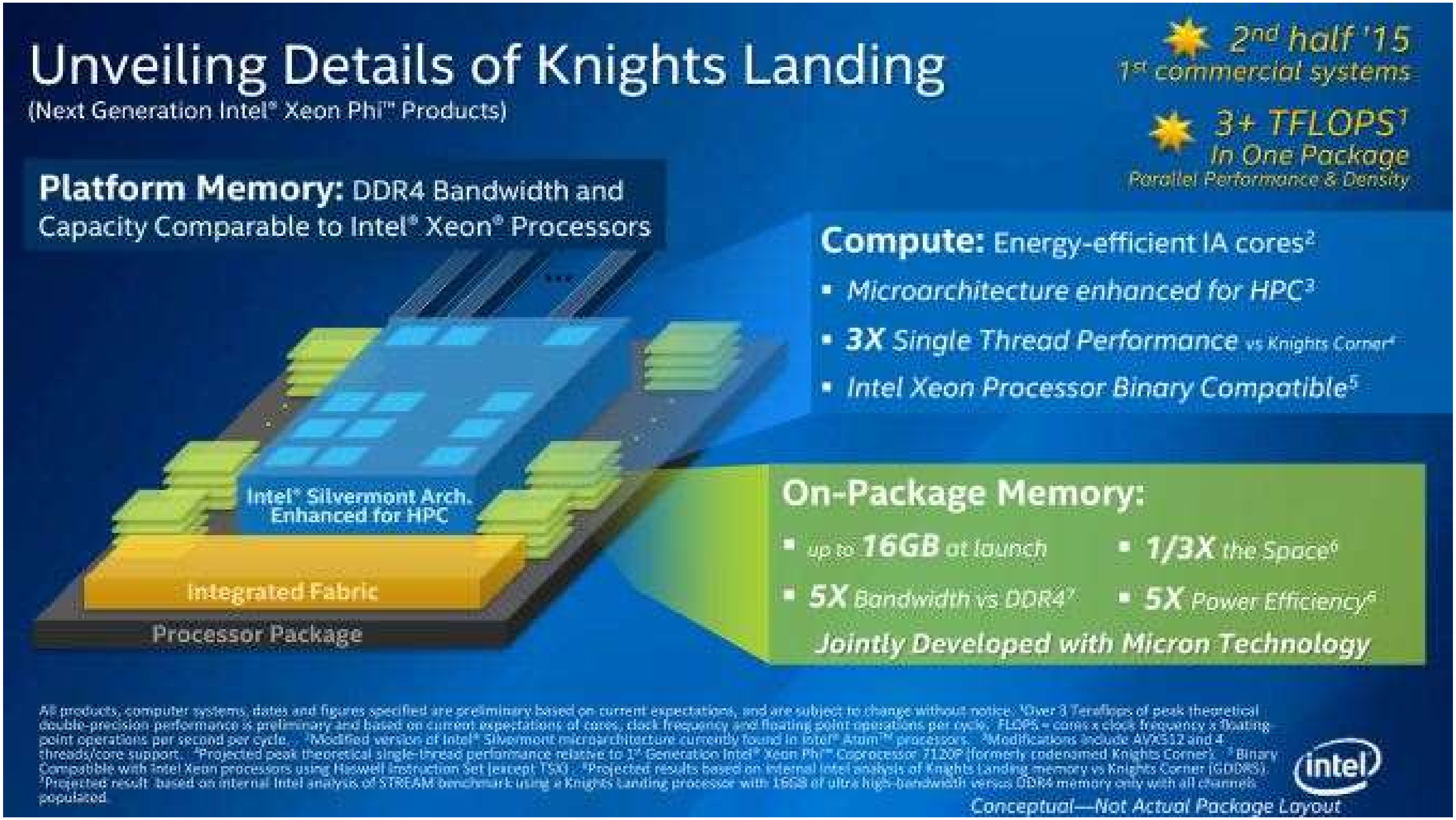
- Implicit communication via loads and stores (SMP)
  - hardware architects have to provide coherent caches and process (thread) synchronization primitives (like `ll` and `sc`)
  - lower communication overhead
  - harder to overlap computation with communication
  - more efficient to use an address to get remote data when *needed* rather than to send for it in case it *might* be needed

- Explicit communication via sends and receives (MPP)
  - simplest solution for hardware architects
  - higher communication overhead
  - easier to overlap computation with communication
  - easier for the programmer to optimize communication

SMP/MPP Example – Intel Xeon Phi Coprocessor

- Intel’s Many Integrated Core Architecture (MIC, Mike)
  - Up to 8 coprocessors (72 cores each) per host server, SMP within a coprocessor, MPP between coprocessors
- Three generations: Knight’s Ferry (3100), Knight’s Corner (5100), Knight’s Landing (7100)
  - Knight’s Landing in 14nm FinFETs, 2nd + quarter 2015
  - 72 Atom (Silvermont) cores, static 2-way superscalar, 4 threads per core (FGMT) so 288 threads, 1.238GHz (1.33GHz Turbo mode)
  - Each core has two 512-bit vector units and supports AVX-512F SIMD instructions
  - On-chip interconnect, mesh NoC ?
  - Up to 384GB of DDR4DRAM and 16GB of stacked 3D MCDRAM
  - 3+ TeraFLOPS per coprocessor
  - TDP of 300W (estimated 15W per core, so how ??)
- Programming tools: OpenMP, OpenCL, Cilk, and specialized versions of Intel's Fortran, C++ and scientific libraries

Xeon Phi Knight’s Landing



Unveiling Details of Knights Landing

(Next Generation Intel® Xeon Phi™ Products)

2nd half '15
1st commercial systems

3+ TFLOPS[1]
In One Package
Parallel Performance & Density

Platform Memory: DDR4 Bandwidth and Capacity Comparable to Intel® Xeon® Processors

Compute: Energy-efficient IA cores[2]

- *Microarchitecture enhanced for HPC*[3]
- *3X Single Thread Performance* vs Knights Corner[4]
- *Intel Xeon Processor Binary Compatible*[5]

On-Package Memory:

- up to **16GB** at launch
- **5X** Bandwidth vs DDR4[7]
- **1/3X** the Space[6]
- **5X** Power Efficiency[6]

Jointly Developed with Micron Technology

All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice. [1]Over 3 Teraflops of peak theoretical double-precision performance is preliminary and based on current expectations of cores, clock frequency and floating point operations per cycle. FLOPS = cores x clock frequency x floating point operations per second per cycle. [2]Modified version of Intel® Silvermont microarchitecture currently found in Intel® Atom™ processors. [3]Modifications include AVX512 and 4 threads/core support. [4]Projected peak theoretical single-thread performance relative to 1st Generation Intel® Xeon Phi™ Coprocessor 7120P (formerly codenamed Knights Corner). [5]Binary Compatible with Intel Xeon processors using Haswell Instruction Set (except TSX). [6]Projected results based on internal Intel analysis of Knights Landing memory vs Knights Corner (GDDR5). [7]Projected result based on internal Intel analysis of STREAM benchmark using a Knights Landing processor with 16GB of ultra high-bandwidth versus DDR4 memory only with all channels populated.

Conceptual—Not Actual Package Layout

http://www.anandtech.com/show/8217/intels-knights-landing-coprocessor-detailed

Summing 100,000 Numbers on 100 Core MPP

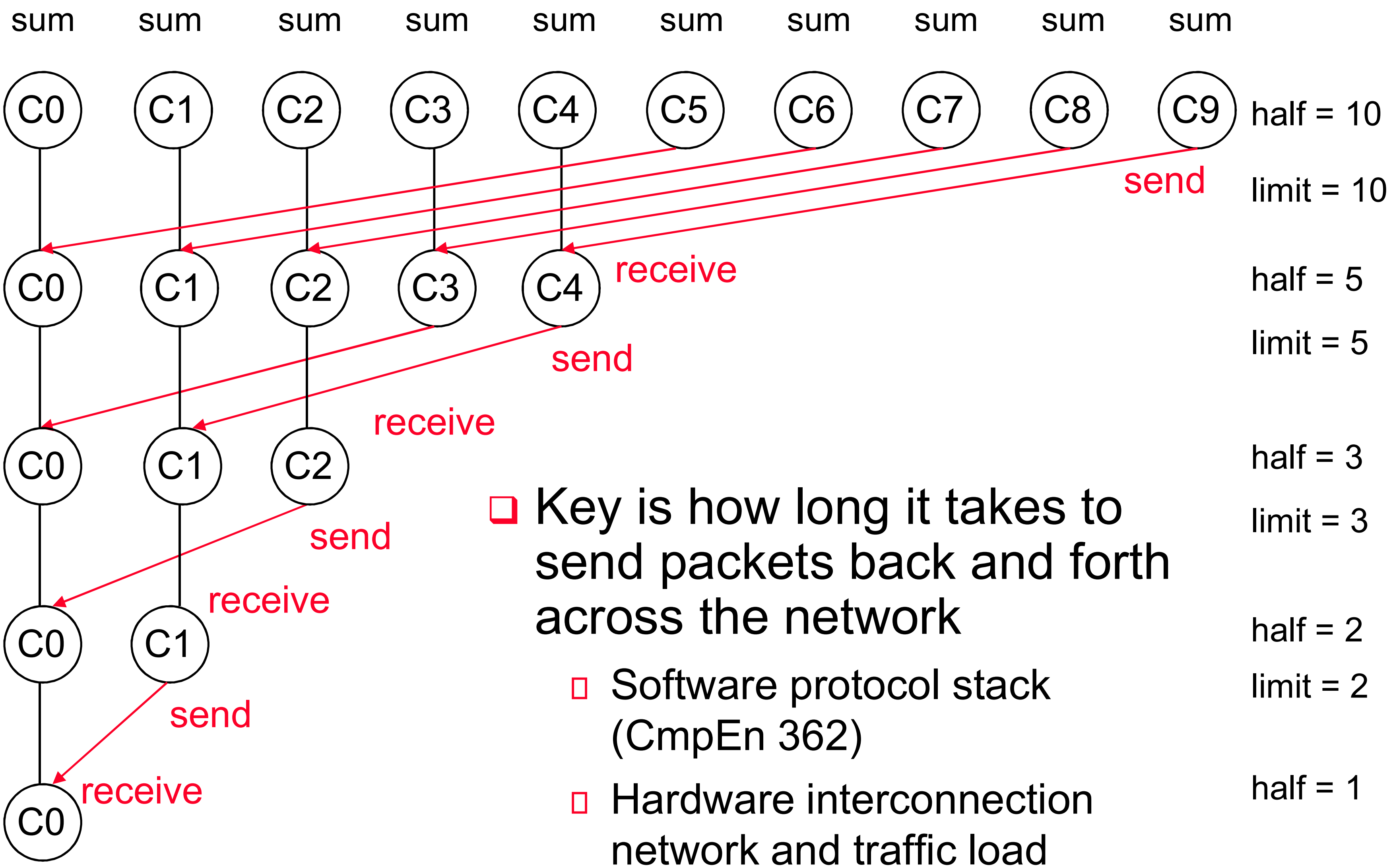
- Start by distributing 1000 elements of vector `A` to each of the local memories and summing each subset in parallel

```
sum = 0;
for (i = 0; i<1000; i = i + 1)
  sum = sum + Al[i];      /* sum local array subset
```

- The cores then coordinate in adding together the sub sums (`Cn` is the number of cores, `send(x,y)` sends value `y` to core `x`, and `receive()` receives a value)

```
half = 100;
limit = 100;
repeat
  half = (half+1)/2;     /*dividing line
  if (Cn>= half && Cn<limit) send(Cn-half,sum);
  if (Cn<(limit/2)) sum = sum + receive();
  limit = half;
until (half == 1);       /*final sum in C0's sum
```

An Example with 10 Cores



- Key is how long it takes to send packets back and forth across the network
    - Software protocol stack (CmpEn 362)
    - Hardware interconnection network and traffic load

Pros and Cons of Message Passing

- Message sending and receiving is *much* slower than addition, for example
- But message passing multiprocessors are much easier for hardware architects to design
  - Don’t have to worry about cache coherency for example
- The advantage for programmers is that communication is explicit, so there are fewer “performance surprises” than with the implicit communication in cache-coherent SMPs
  - Message passing standard MPI-2.2 (www.mpi-forum.org)
- However, its harder to port a sequential program to a message passing multiprocessor since every communication must be identified in advance
  - With cache-coherent shared memory the hardware figures out what data needs to be communicated

Aside: Quick Summary of MPI

- The MPI Standard describes
  - point-to-point message-passing
  - collective communications
  - group and communicator concepts
  - process topologies
  - environmental management
  - process creation and management
  - one-sided communications
  - extended collective operations
  - external interfaces
  - I/O functions
  - a profiling interface
- Language bindings for C, C++ and Fortran are defined

http://www.mpi-forum.org/docs/docs.html

Concurrency and Parallelism

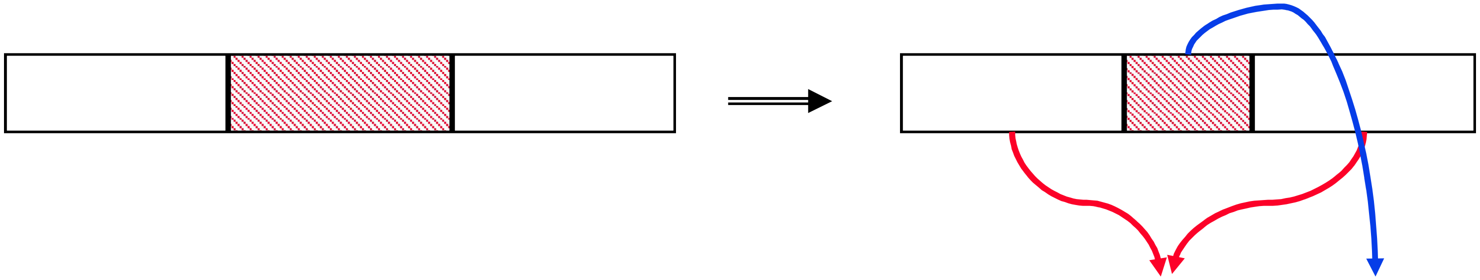
- Programs are designed to be sequential or concurrent
  - Sequential – only one activity, behaving in the “usual” way
  - Concurrent – multiple, simultaneous activities, designed as independent operations or as cooperating threads or processes
    - The various parts of a concurrent program need not execute simultaneously, or in a particular sequence, but they do need to coordinate their activities by exchanging information in some way
- A key challenge is to build parallel (concurrent) programs that have high performance on multiprocessors as the number of cores increase – programs that scale
  - Problems that arise
    - Scheduling threads on cores close to the memory space where their data primarily resides
    - Load balancing threads on cores and dealing with thermal hot-spots
    - Time for synchronization of threads
    - Overhead for communication of threads

Encountering Amdahl's Law

- Speedup due to enhancement E is

$$\text{Speedup w/ E} = \frac{\text{Exec time w/o E}}{\text{Exec time w/ E}}$$

- Suppose that enhancement E accelerates a fraction F (F <1) of the task by a factor S (S>1) and the remainder of the task is unaffected



ExTime w/ E = ExTime w/o E × ((1-F) + F/S)

Speedup w/ E = 1 / ((1-F) + F/S)

Example 1: Amdahl’s Law

Speedup w/ E = 1 / ((1-F) + F/S)

- Consider an enhancement which runs 20 times faster but which is only usable 25% of the time.

  Speedup w/ E = 1/(.75 + .25/20) = 1.31

- What if its usable only 15% of the time?

  Speedup w/ E = 1/(.85 + .15/20) = 1.17

- Amdahl’s Law tells us that to achieve linear speedup with 100 cores (so 100 times faster), none of the original computation can be scalar!

- To get a speedup of 90 from 100 cores, the percentage of the original program that could be scalar would have to be 0.1% or less

  Speedup w/ E = 1/(.001 + .999/100) = 90.99

Example 2: Amdahl's Law

Speedup w/ E = 1 / ((1-F) + F/S)

- Consider summing 10 scalar variables and two 10 by 10 matrices (matrix sum) on 10 cores

  Speedup w/ E = 1/(.091 + .909/10) = 1/0.1819 = 5.5

- What if there are 100 cores?

  Speedup w/ E = 1/(.091 + .909/100) = 1/0.10009 = 10.0

- What if the matrices are 100 by 100 (or 10,010 adds in total) on 10 cores?

  Speedup w/ E = 1/(.001 + .999/10) = 1/0.1009 = 9.9

- What if there are 100 cores?

  Speedup w/ E = 1/(.001 + .999/100) = 1/0.01099 = 91

Multiprocessor Scaling

- To get good speedup on a multiprocessor while keeping the problem size fixed is harder than getting good speedup by increasing the size of the problem
    - Strong scaling – when good speedup is achieved on a multiprocessor without increasing the size of the problem
    - Weak scaling – when good speedup is achieved on a multiprocessor by increasing the size of the problem proportionally to the increase in the number of cores and the total size of memory

- But Amdahl was an optimist – you probably will need extra time to patch together parts of the computation that were done in parallel

Multiprocessor Benchmarks

	Scaling?	Reprogram?	Description
LINPACK	Weak	Yes	Dense matrix linear algebra
http://www.top500.org/project/linpack/			
SPECrate	Weak	No	Parallel SPEC programs for job-level parallelism
SPLASH 2	Strong	No	Independent job parallelism (both kernels and applications, from high-performance computing)
NAS Parallel	Weak	Yes (c or Fortran)	Five kernels, mostly from computational fluid dynamics
PARSEC	Weak	No	Multithreaded programs that use Pthreads and OpenMP. Nine applications and 3 kernels – 8 with data parallelism, 3 with pipelined parallelism
Berkeley Patterns	Strong or Weak	Yes	13 design patterns implemented by frameworks or kernels

DGEMM (Double precision GEneral MM) Example

- DGEMM: A BLAS (Basic Linear Algebra Subprograms) routine; part of LINPACK used for performance measurements

```
                     C = C + A * B
```

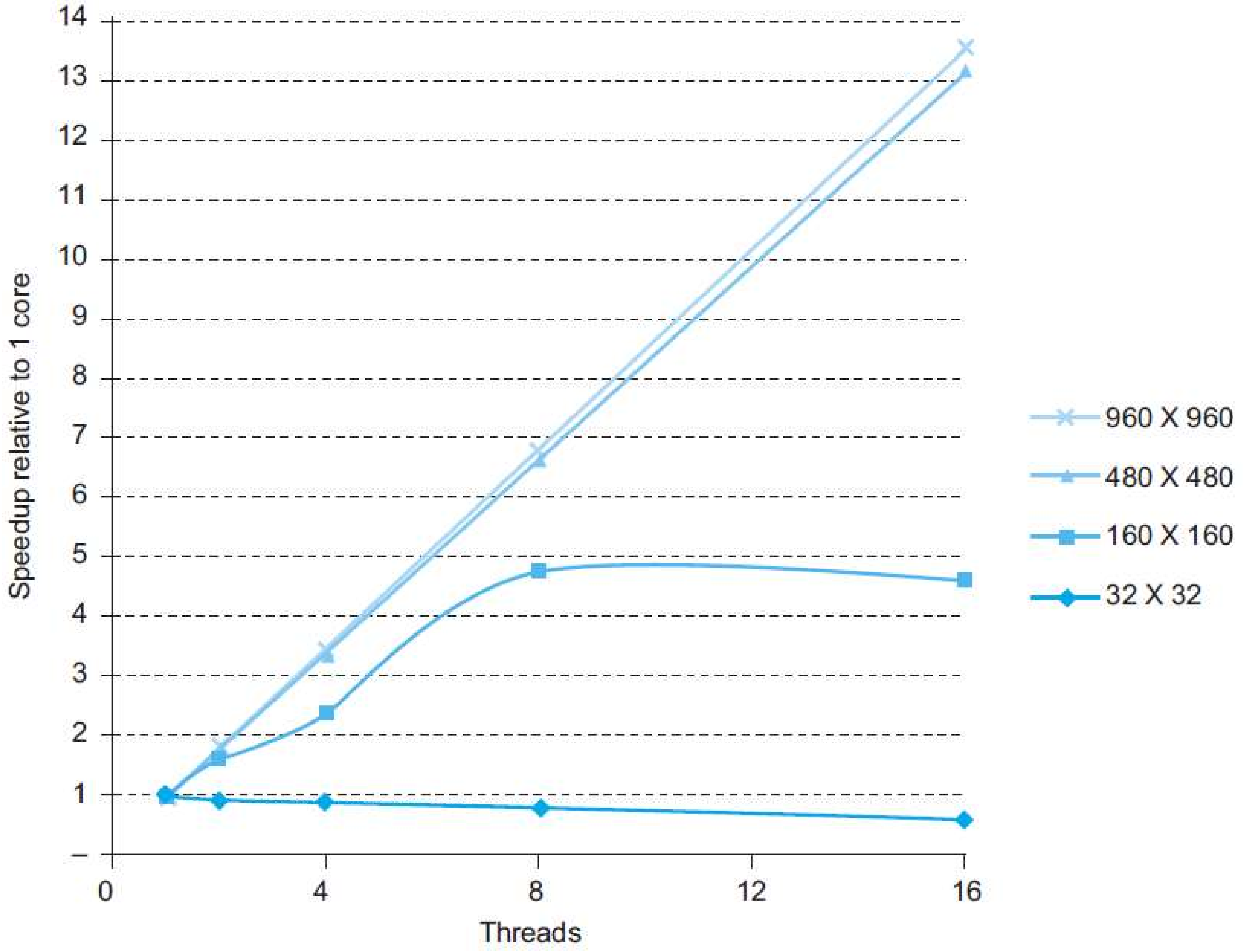
```
void dgemm (int n, double* A, double* B, double* C)
{ for (int i = 0; i < n; ++i)
  for (int j = 0; j < n; ++j)
  { double cij = C[i+j*n]; /* cij=C[i][j] */
    for (int k = 0; k < n; ++k)
       cij += A[i+k*n] * B[k+j*n];
                  /* cij += A[i][j] * B[i][j] */
       C[i+j*n] = cij; /*C[i][j] = cij */
   }
}
```

Multithreaded, Blocked OpenMP DGEMM

- `#pragma` OpenMP code makes the outmost `for` loop operate in parallel

```
.
.
.
void dgemm (int n, double* A, double* B, double* C)
{
#pragma omp parallel for
 for ( int sj = 0; sj < n; sj += BLOCKSIZE )
  for ( int si = 0; si < n; si += BLOCKSIZE )
   for ( int sk = 0; sk < n; sk += BLOCKSIZE )
    do_block(n, si, sj, sk, A, B, C);
```

DGEMM Scaling: Thread Count, Matrix Size



Multiprocessor Basics

- Q1 – How do they share data?
  - A single physical address space shared by all cores or message passing
- Q2 – How do they coordinate?
  - **Through atomic operations on shared variables in memory (via loads and stores) or via message passing**
- Q3 – How scalable is the architecture? How many cores?

			# of Cores
Communication model	Message passing		8 to 2048 +
	SMP	NUMA	8 to 256 +
		UMA	2 to 32
Physical connection	Network		8 to 256 +
	Bus		2 to 8

Architectural taxonomies for parallelism

- An alternate classification for < ILP, TLP, DLP >

		Data Streams	
		Single	Multiple
Instruction Streams	Single	SISD: Intel Core i7	**SIMD: SSE Instr’s of x86**
	Multiple	MISD: No examples today	MIMD: SMPs (IBM Power 8); MPPs (Intel Phi)

- SPMD: Single Program Multiple Data
  - A parallel program running on a MIMD computer
  - With conditional code for different cores

SIMD Processors

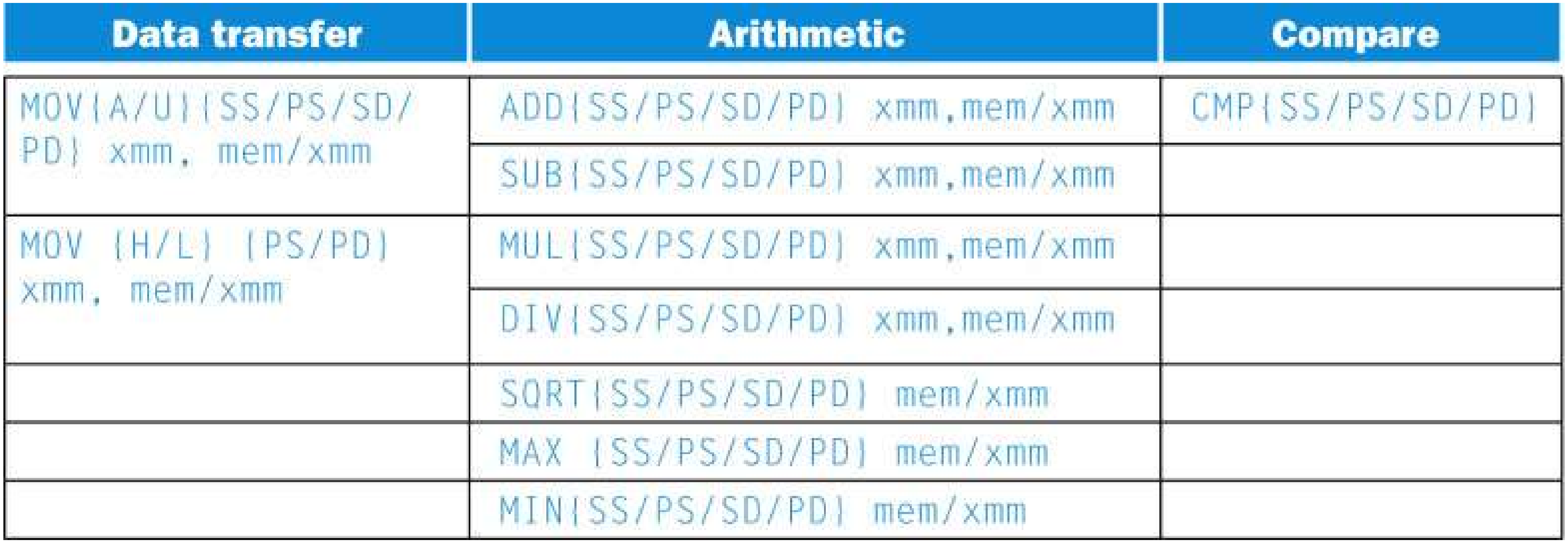
- All processing elements execute the same instruction at the same time which operate element-wise on “vectors” of data
  - Each with different data addresses, etc.
- Works best for highly data-parallel applications
  - Simplifies synchronization
  - Amortized instruction control hardware

- Example 1: MMX, SSE, and AVX instruction in x86
- Example 2: Illiac IV http://en.wikipedia.org/wiki/ILLIAC_IV
- Example 3: GPUs like NVIDIA’s GTX

Subword Parallelism – SIMD (on the cheap)

- By partitioning the carry chain in a 128-bit adder, a core can perform simultaneous operations on short "vectors" of sixteen 8-bit operands, eight 16-bit operands, four 32-bit operands, or two 64-bit operands at low additional cost
- MMX (MultiMedia eXtensions) added to x86 – 1997
  - Added 57 instr's to accelerate multimedia and communication app's that operated on subword data elements (four 8-bit data per 32-bit register, operated on in parallel by a 32-bit adder)
- SSE (Streaming SIMD eXtensions) – 1999, 2001
  - Eight SSE 128-bit SSE registers, so four 32-bit (later two 64-bit) floating point op's could be performed in parallel
  - Cache prefetch load and streaming store instructions
- AVX (Advanced Vector eXtensions) – 2011
  - Register and ALU extensions to support four 64-bit (later eight 64-bit) FP ops in parallel

SSE/SSE2 x86 Instructions



Data transfer	Arithmetic	Compare
MOV{A/U}{SS/PS/SD/PD} xmm, mem/xmm	ADD{SS/PS/SD/PD} xmm,mem/xmm	CMP{SS/PS/SD/PD}
	SUB{SS/PS/SD/PD} xmm,mem/xmm	
MOV {H/L} {PS/PD} xmm, mem/xmm	MUL{SS/PS/SD/PD} xmm,mem/xmm	
	DIV{SS/PS/SD/PD} xmm,mem/xmm	
	SQRT{SS/PS/SD/PD} mem/xmm	
	MAX {SS/PS/SD/PD} mem/xmm	
	MIN{SS/PS/SD/PD} mem/xmm	

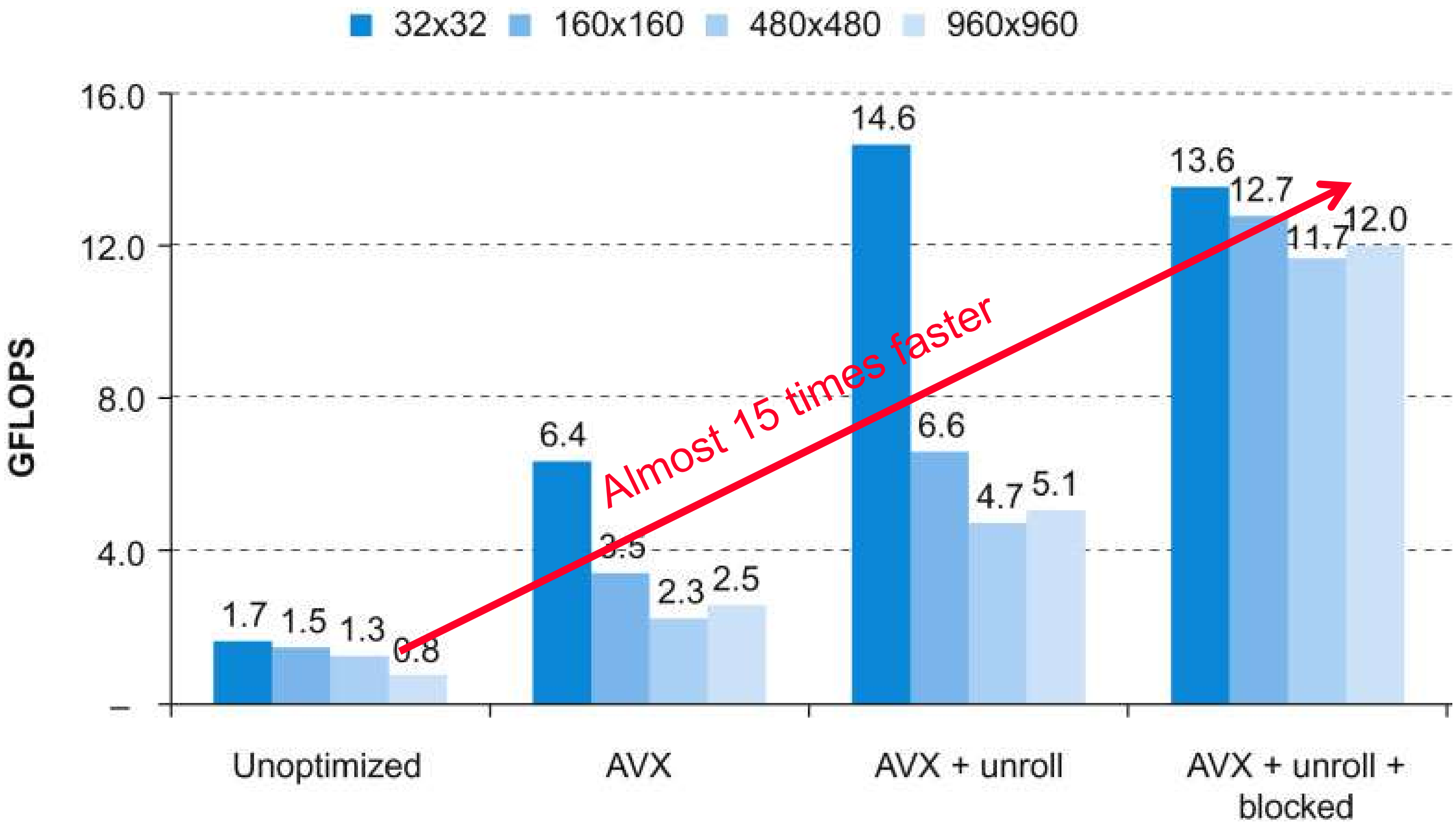
- For example (in x86)

```
mulpd %xmm0, %xmm4
```

performs two 64-bit floating-point (packed double) multiplies in parallel on operands stored in 128-bit SSE registers

DGEMM Using AVX

- Taking advantage of subword parallelism (AVX), instruction level parallelism (compiler loop unrolling), and caches (matrix blocking)



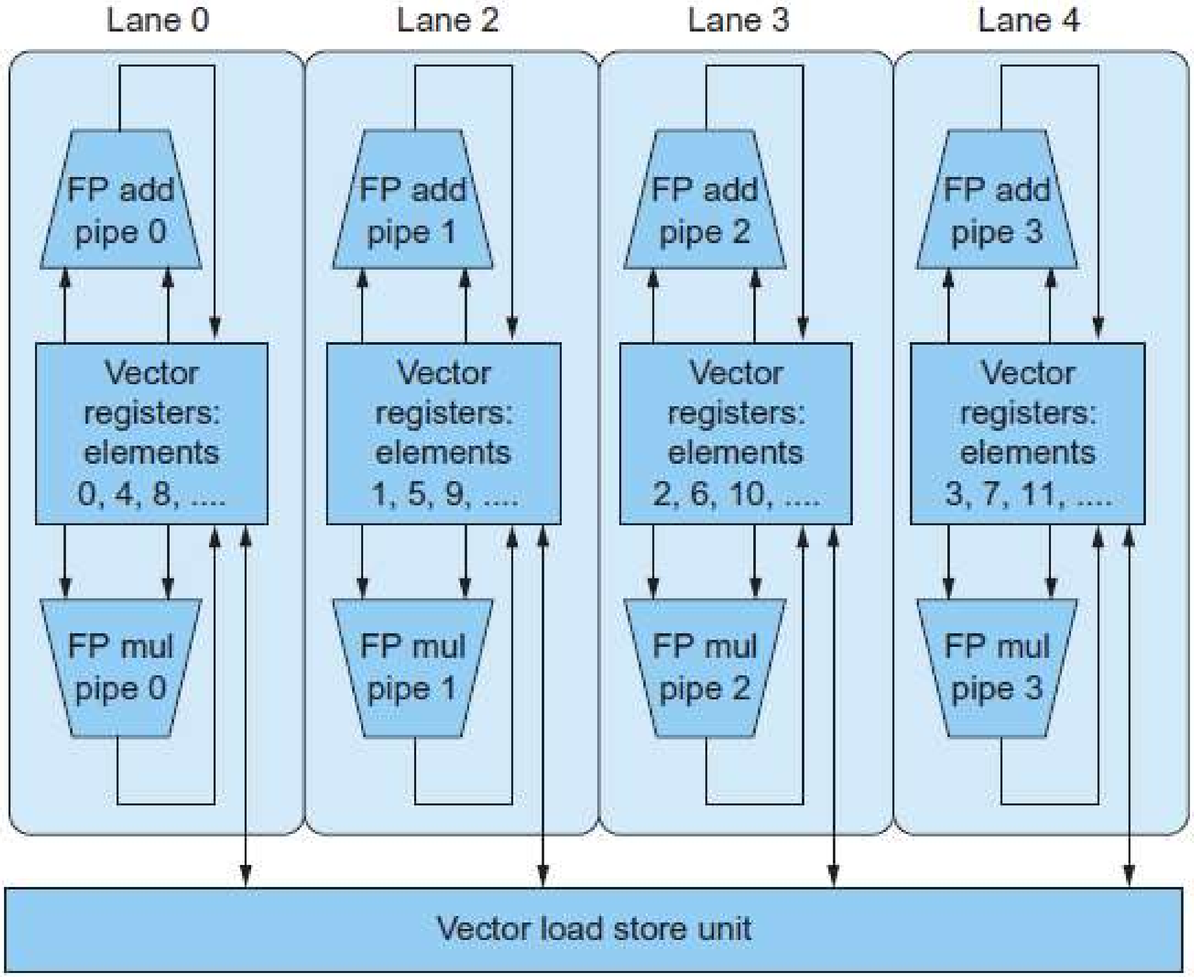
Almost 15 times faster

Vector Processors ("old-school") SIMD

- Highly pipelined functional units
- Stream data from/to vector registers and functional units
  - Data collected (gather) from memory into vector registers
  - Results stored (scatter) from vector registers to memory
  - http://en.wikipedia.org/wiki/Vectored_I/O
- Example 1: Vector extensions to MIPS
  - 32 x 64-element registers (64-bit elements)
  - Vector instruction examples
    - `lv, sv`: load/store vector
    - `addv.d`: add vectors of double
    - `addvs.d`: add scalar to each element of vector double
- Example 2: Cray-1, Cray-2, and Cray-3
  http://en.wikipedia.org/wiki/Cray

A Vector Unit with Four Pipelines (Lanes)

- Vector-register elements are interleaved across the pipeline lanes
  - Each lane holds every 4th vector element
- Pipelined FUs can complete 8 ops per cycle

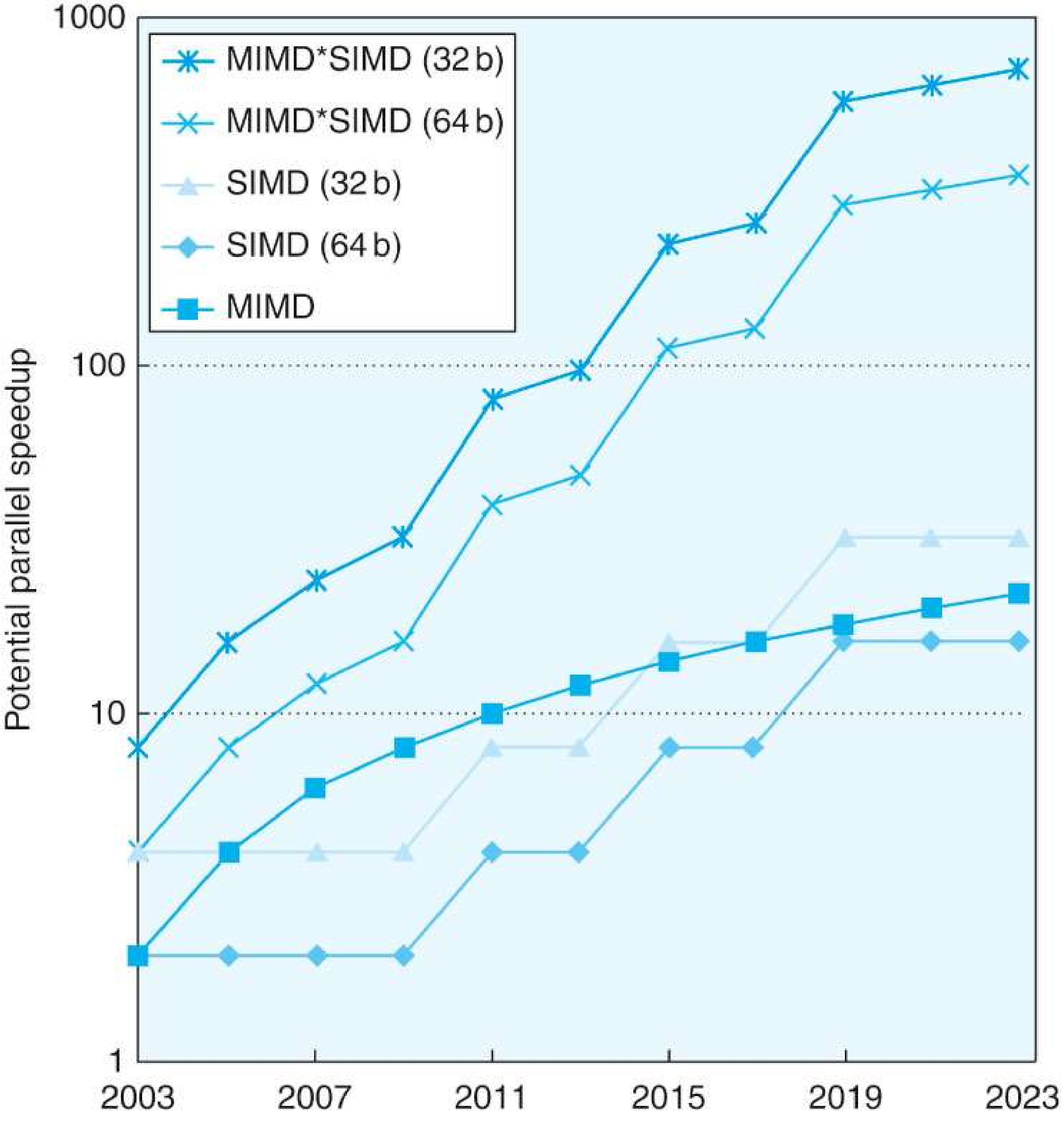

Lane 0
Lane 2
Lane 3
Lane 4
FP add pipe 0
FP add pipe 1
FP add pipe 2
FP add pipe 3
Vector registers: elements 0, 4, 8, ...
Vector registers: elements 1, 5, 9, ...
Vector registers: elements 2, 6, 10, ...
Vector registers: elements 3, 7, 11, ...
FP mul pipe 0
FP mul pipe 1
FP mul pipe 2
FP mul pipe 3
Vector load store unit

Vector Processor versus Multimedia Extensions

- Vector processing is more general than ad-hoc multi-media extensions (such as MMX, SSE)
  - Vector instr's have a variable vector width, MM extensions have a fixed width
  - Vector instr's support strided access, MM does not
  - Vector FUs can be a combination of pipelined and arrayed FUs
- And are a better match with compiler technologies
  - Simplifies data-parallel programming
  - Significantly reduces instruction-fetch bandwidth
  - Avoids control hazards by avoiding loops
  - Explicit statement of absence of loop-carried dependencies
    - Reduced data dependency hardware checking
  - Regular access patterns benefit from interleaved and burst DRAMs

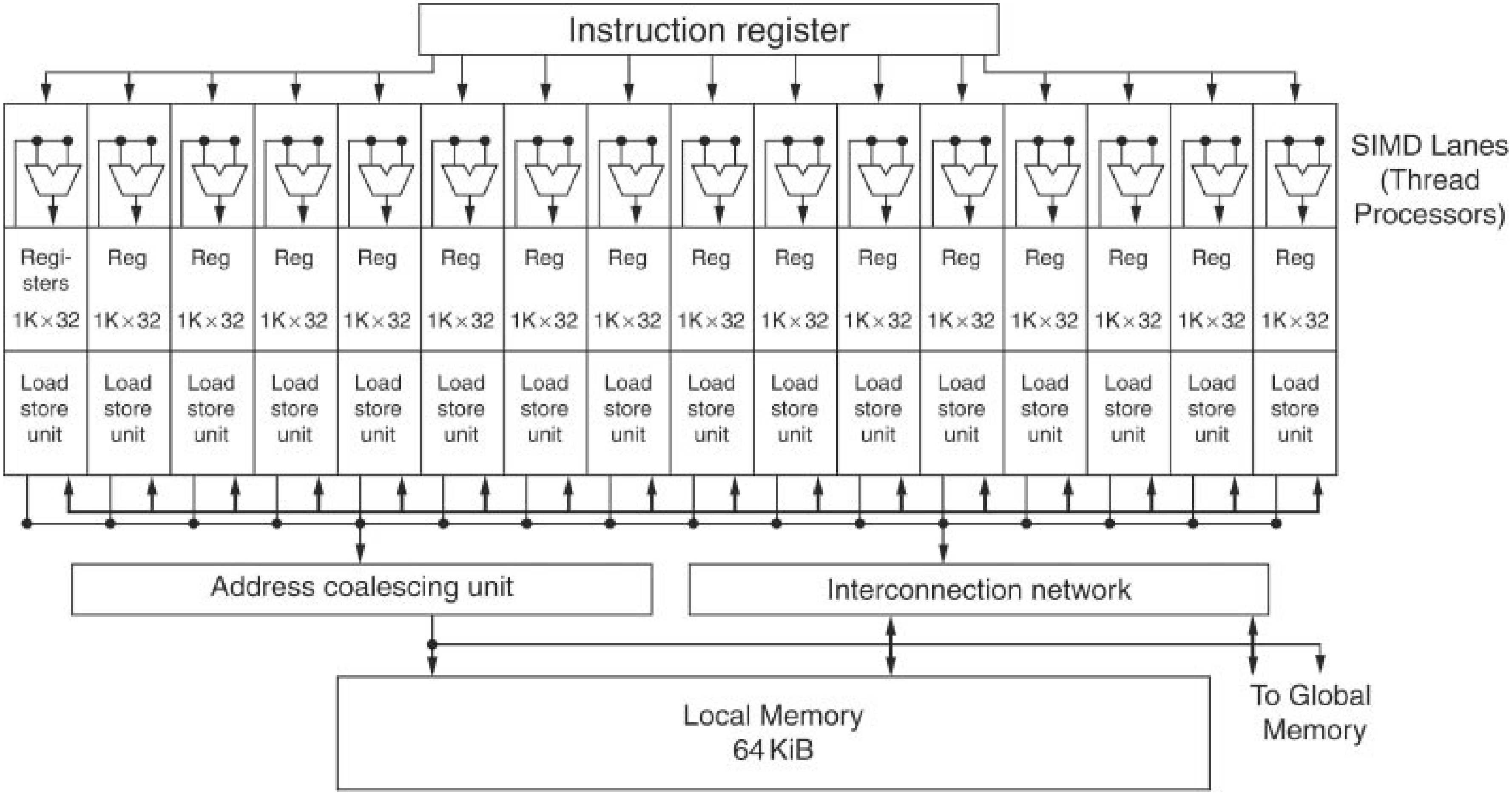
SIMD, MIMD, both?

- Scalability limit study (using x86 baseline values): <u>What would happen if:</u>
  - MIMD: 2 cores per chip added every two years
  - SIMD: SIMD widths (registers and ALUs) will double every four years



A Multithreaded SIMD Processor

- 16 SIMD “lanes” (ALU, Register File, Load/store unit)



- A GPU – a collection of multithreaded SIMD processors

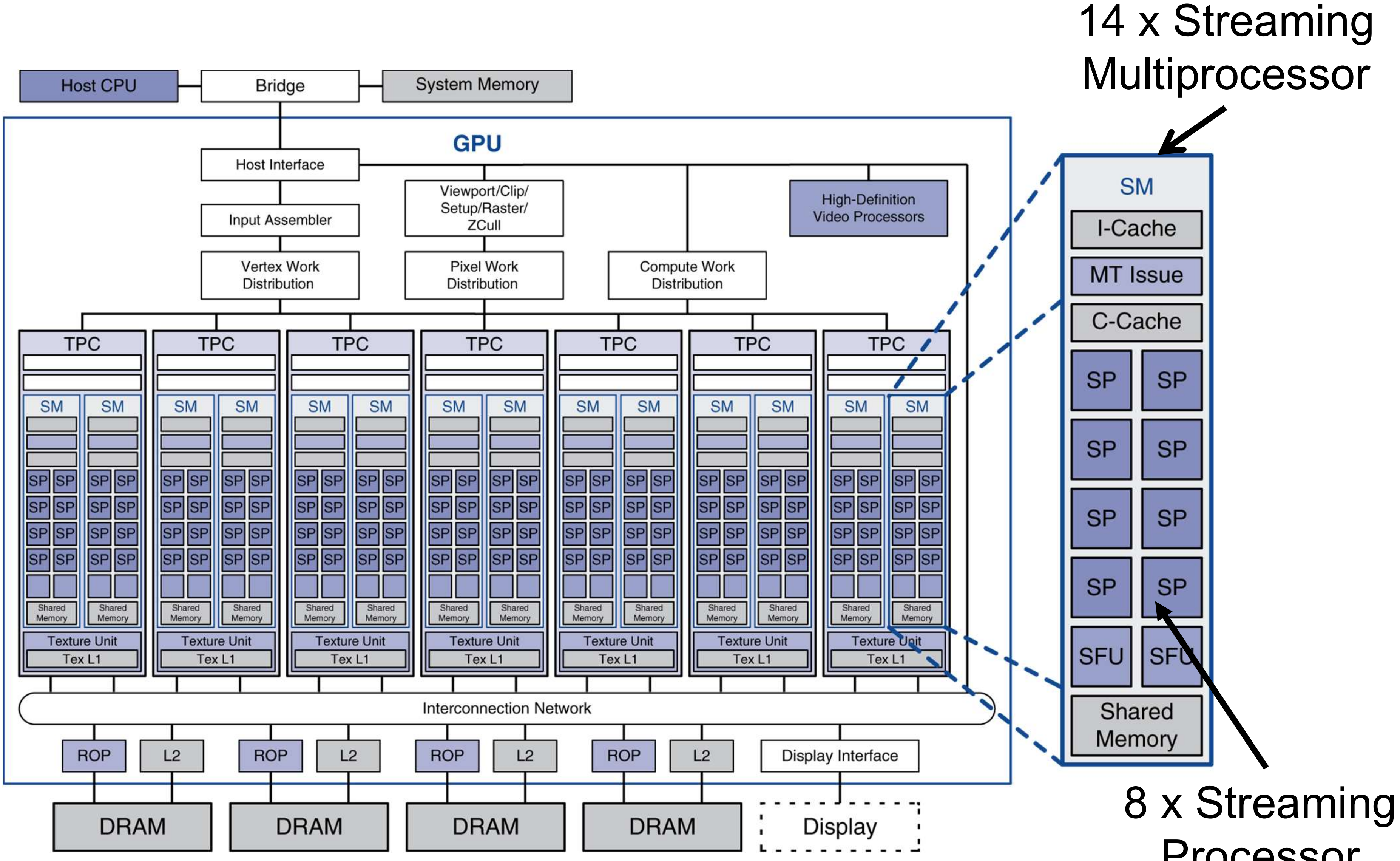
GPU Architectures

- GPUs are highly multithreaded (data-parallel)
  - Use thread switching to hide memory latency (~FGMT)
    - Less reliance on multi-level caches
  - Graphics memory is wide and high-bandwidth

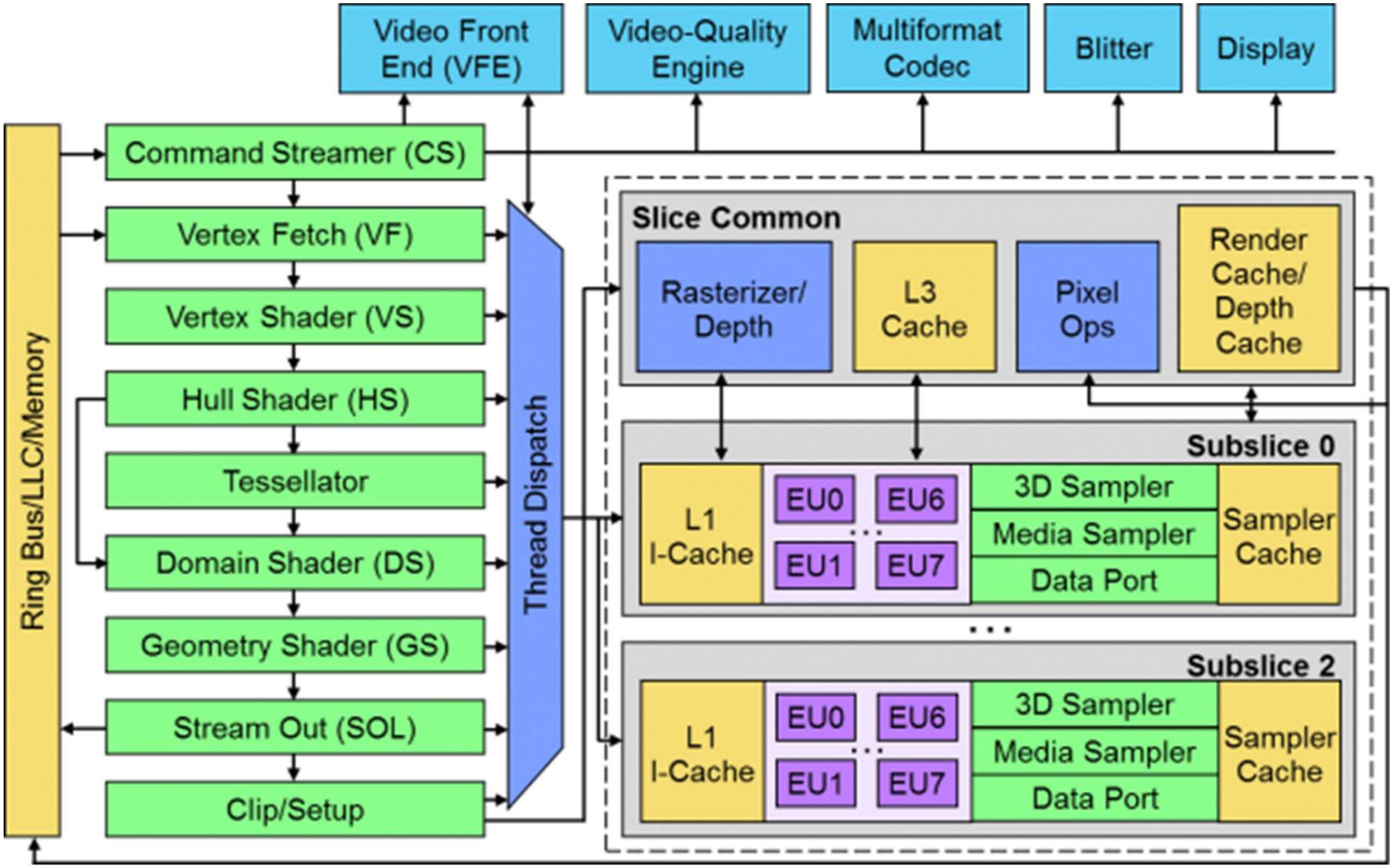
- Trend is toward general purpose/peer GPUs
  - Heterogeneous CPU/GPU systems
  - CPU for sequential code, GPU for parallel code

- Programming languages/APIs
  - DirectX, OpenGL
  - C for Graphics (Cg), High Level Shader Language (HLSL)
  - NVIDIA's Compute Unified Device Architecture (CUDA)

NVIDIA Tesla



Intel’s Gen8 GPU Architecture



From MPR, Sep 2014

Classifying GPUs

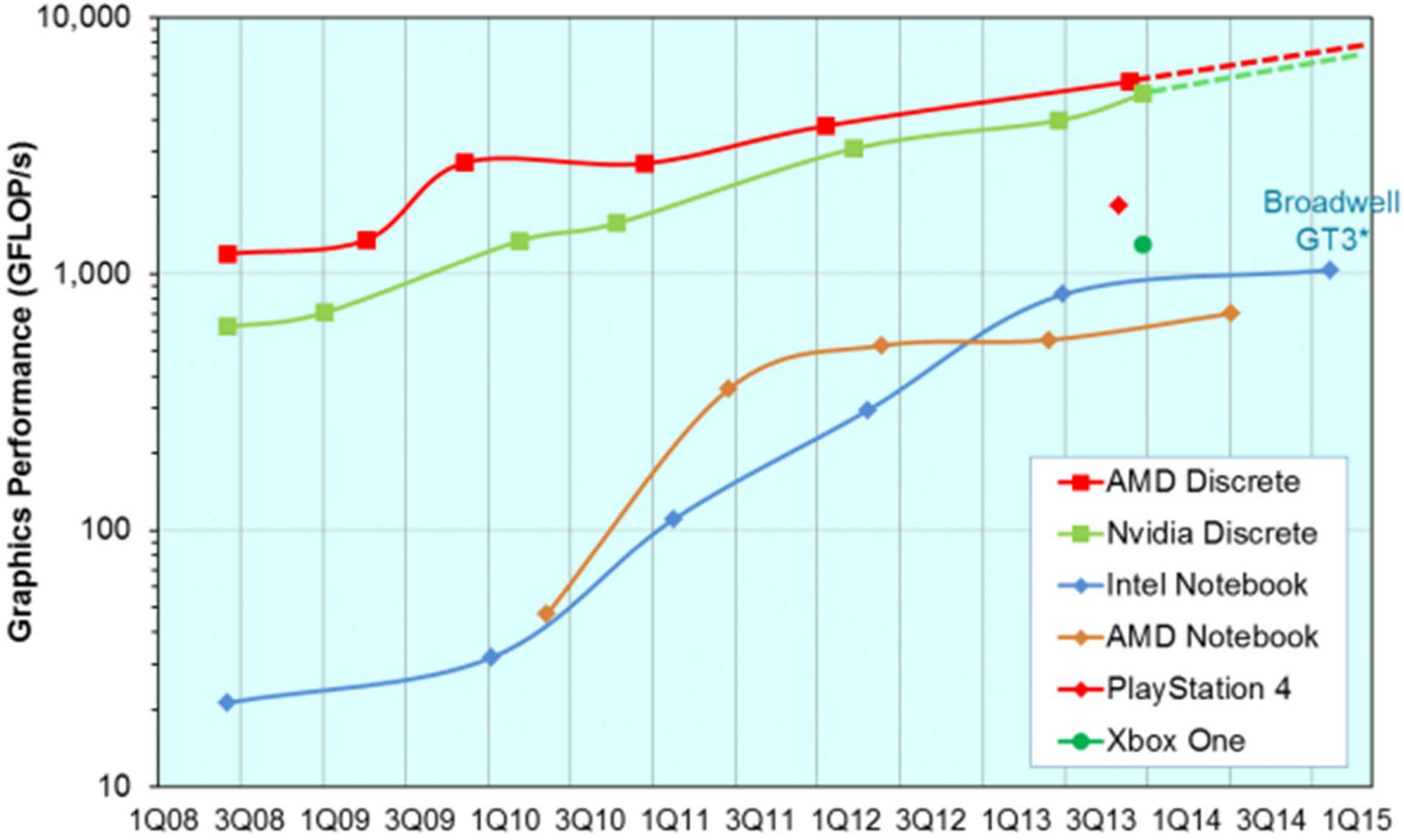
- ❑ Don’t fit nicely into the SIMD/MIMD model
  - Conditional execution in a thread allows an illusion of MIMD
    - But with performance degradation
    - Need to write general purpose code with care

	Static: Discovered at Compile Time	Dynamic: Discovered at Runtime
Instruction-Level Parallelism	VLIW	Superscalar
Data-Level Parallelism	SIMD or Vector	**Tesla Multiprocessor**

Multimedia SIMD Extensions Versus GPUs

Feature	Multicore with SIMD	GPU
SIMD processors	4 to 8	8 to 16
SIMD lanes/processor	2 to 4	8 to 16
Multithreading hardware support for SIMD threads	2 to 4	16 to 32
Typical ratio of single precision to double-precision performance	2:1	2:1
Largest cache size	8 MB	0.75 MB
Size of memory address	64-bit	64-bit
Size of main memory	8 GB to 256 GB	4 GB to 6 GB
Memory protection at level of page	Yes	Yes
Demand paging	Yes	No
Integrated scalar processor/SIMD processor	Yes	No
Cache coherent	Yes	No

Graphics Performance



From MPR, Sep 2014

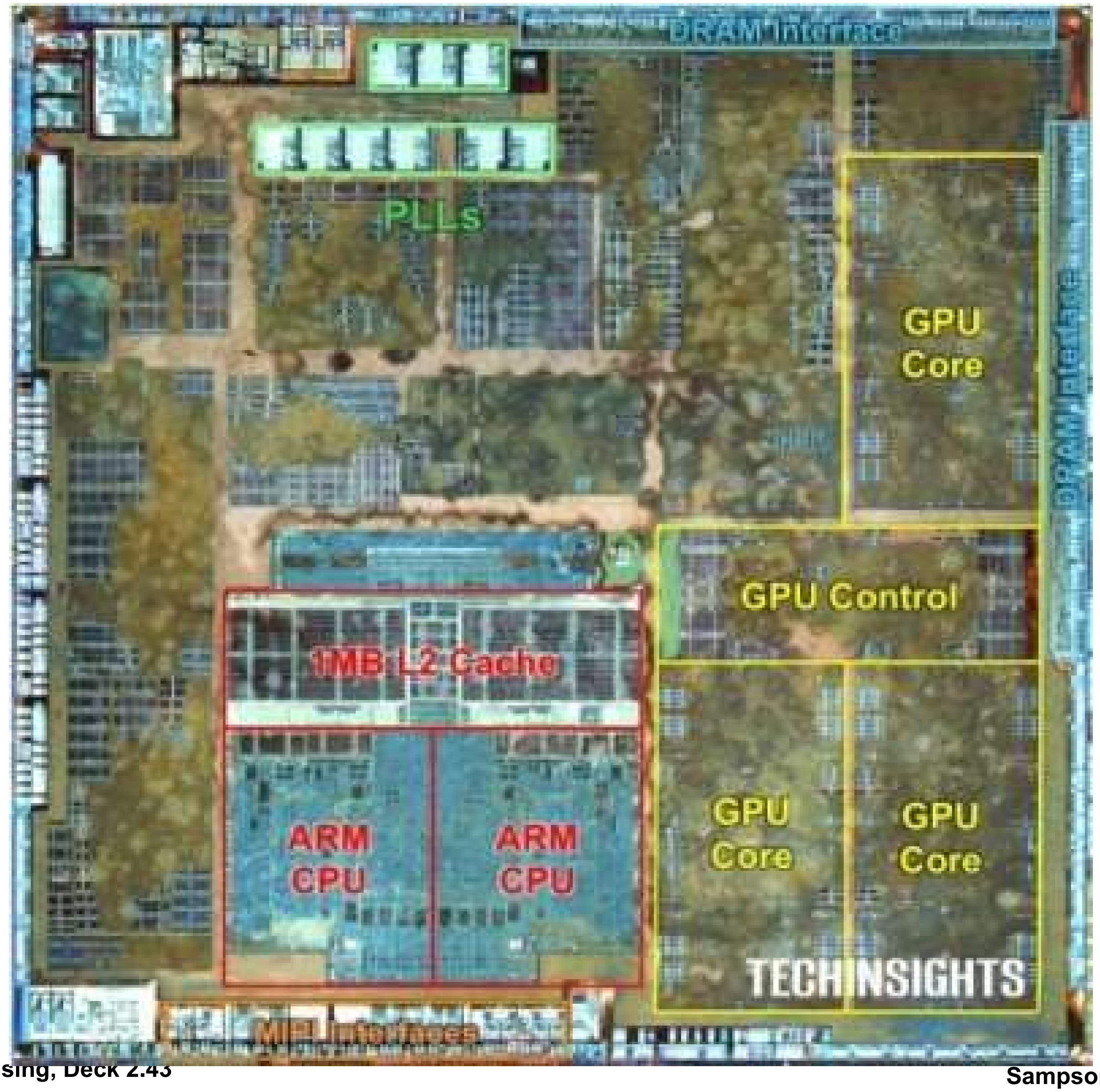
Parallel processing in practice

- As you’ve seen, don’t need to (or want to) pick just one degree of parallelism to exploit!
  - However, picking all of them could be expensive
  - Need to prioritize based on expected workload properties

- Many (majority?) of high performance processors today are SoCs
  - Combine together CPUs, GPUs, and ASICs
  - Mixes different in smartphones and servers, but same principle

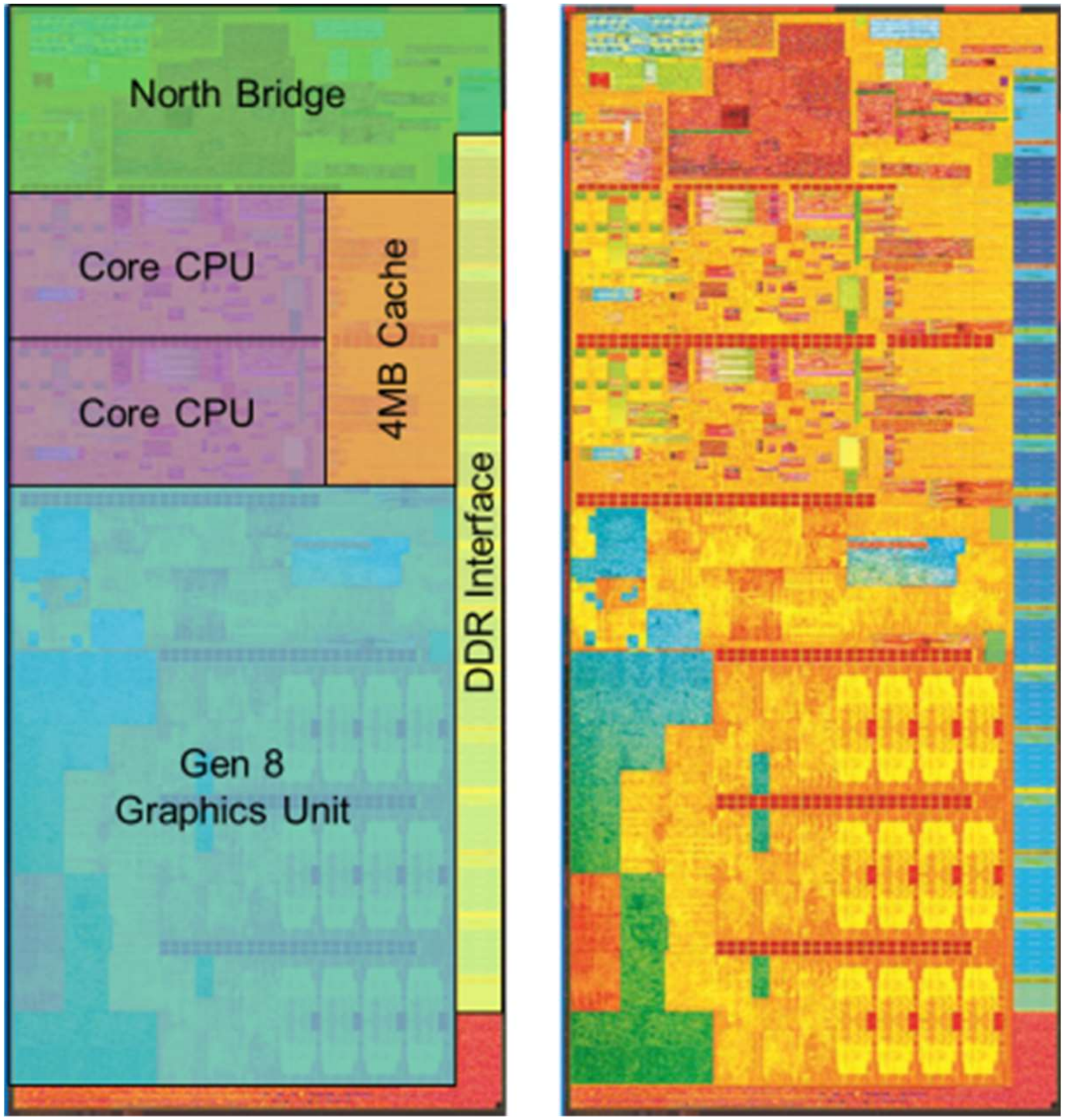
- Can also consider post-manufacturing parallel systems:
  - Discrete CPU + discrete GPU
  - FPGA + NIC + CPU (now deployed in MS datacenters)
  - Datacenters themselves as “parallel computers” (more in a few slides)

Apples A6 Processor (2 CPUs, 3 GPU cores)



Intel’s Broadwell-L

- Tock node at 14nm FinFET
    - 82mm2 die
    - 4.5W TDP
- Haswell’s “tick” partner
    - IPC 5% faster
    - Larger ROB
    - Larger BP tables
    - Larger L2 TLB (1K to 1.5K entries)
    - New L2 TLB for large pages (16 entries for 1GB pages)



From MPR, Sep 2014

Loosely Coupled Parallel Architectures (Clusters)

- Multiple off-the-shelf computers (each with its own private address space and OS) connected via a local area network (LAN) functioning as a “single” multiprocessor
  - Search engines, Web servers, email servers, databases, …
  - The current trend is toward grid or cloud computing, using wide- or global-area networks, perhaps borrowing from other participants, or selling a service as a subset of an existing corporate network
- N OS copies limits the memory space for applications
- Improved system availability and expandability
  - easy to replace a machine without bringing down the whole system
  - allows rapid, incremental expandability
- Economy-of-scale advantages with respect to costs

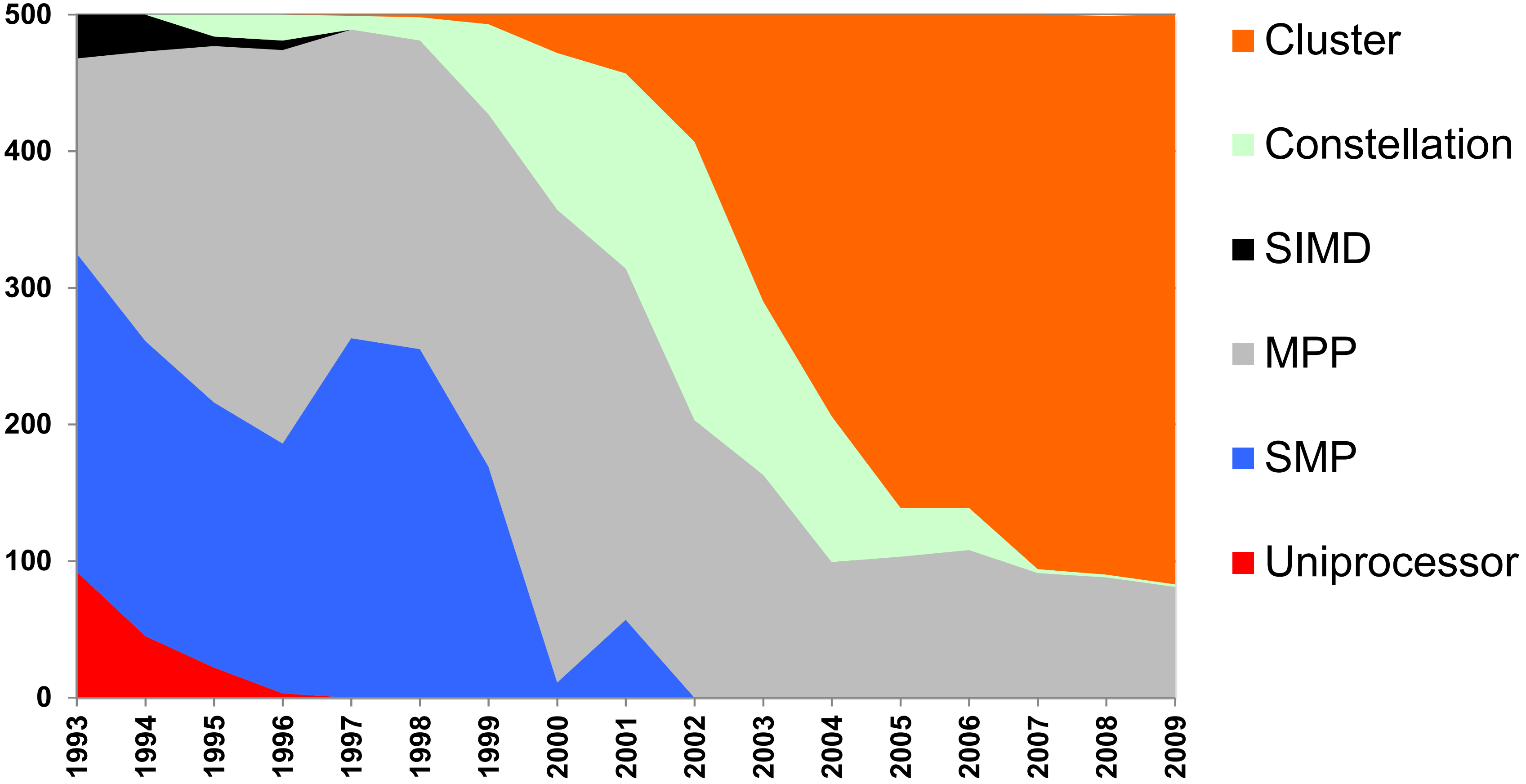
Top500 Architecture Styles

- Uniprocessor – one core only
  - On the Top500 list, a vector processor
- SIMD – one control core, many compute units
- SMP – Shared Memory multiProcessors
- MPP – Message Passing multiProcessors
- Constellation – a collection of network connected SMPs
- Clusters – collection of independent, network connected PCs (commodity processors and memory), either
  - Separate (not coherent) address spaces, communicate via message passing, commodity network
  - Single (coherent) address space, communicate via direct inter-node memory access, custom inter-node network

Top500 Architecture Styles, 1993-2009

http://www.top500.org/

Nov 2009 data



- Uniprocessors and SIMDs disappeared while Clusters and Constellations grew from <3% to 80%. As of 2017, it is 100% Clusters and MPPs. (Clusters >87%)

From the Top500 Lists and Earlier …

- First Megaflop (10^6) system – CDC 7600, 1971
  - in today's vocabulary, a superscalar processor

http://en.wikipedia.org/wiki/CDC_7600

- First Gigaflop (10^9) system – Cray-2, 1986
  - a vector processor, 4 CPUs, 2 GB memory

http://en.wikipedia.org/wiki/Cray-2

- First Teraflop (10^{12}) system – Intel ASCI Red, built for Sandia National Laboratories, Albuquerque, 1996
  - 7,264 Pentium Pro processors (200 MHz), later increased to 9,632 Pentium II OverDrive processors (333 MHz)
- First Petaflop (10^{15}) system – IBM Roadrunner, built for Los Alamos National Laboratory, 2008
  - 12,960 IBM Cell + 6,480 AMD Opteron processors, all multicore
- First Exaflop (10^{18}) system – 2020?

http://en.wikipedia.org/wiki/IBM_Roadrunner

- Most recent list – https://www.top500.org/lists/2017/11/