Design of Parallel and High-Performance Computing

Fall 2016

Lecture: Introduction

Instructor: Torsten Hoefler & Markus Püschel

TA: Salvatore Di Girolamo

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Goals of this lecture

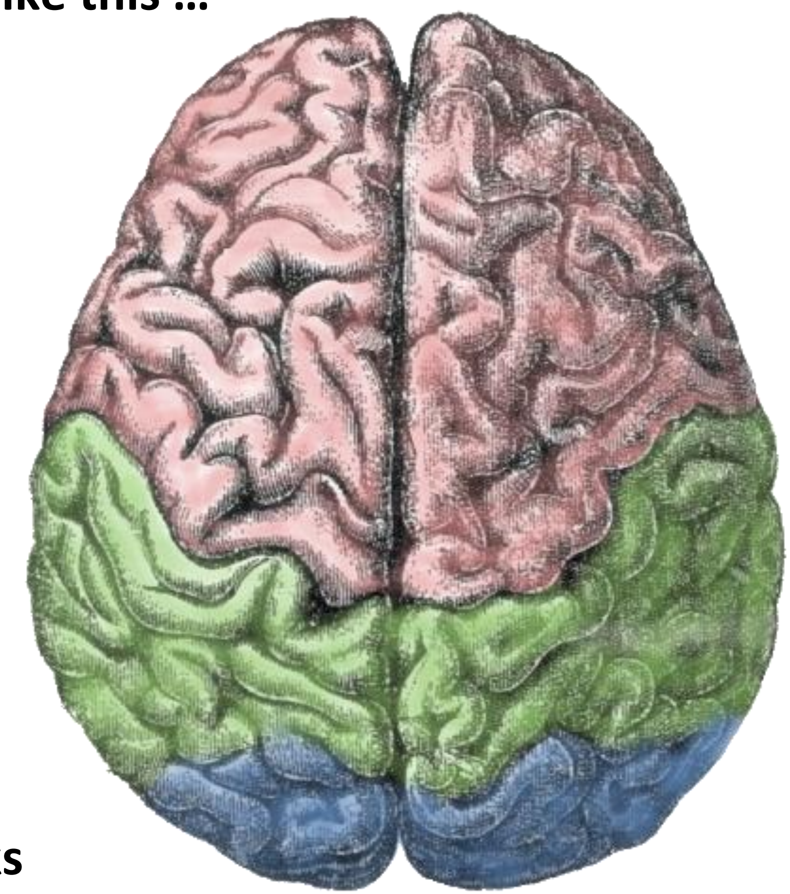
- **Motivate you!**
- **What is parallel computing?**
  - And why do we need it?
- **What is high-performance computing?**
  - What's a Supercomputer and why do we care?
- **Basic overview of**
  - Programming models

    Some examples
  - Architectures

    Some case-studies
- **Provide context for coming lectures**

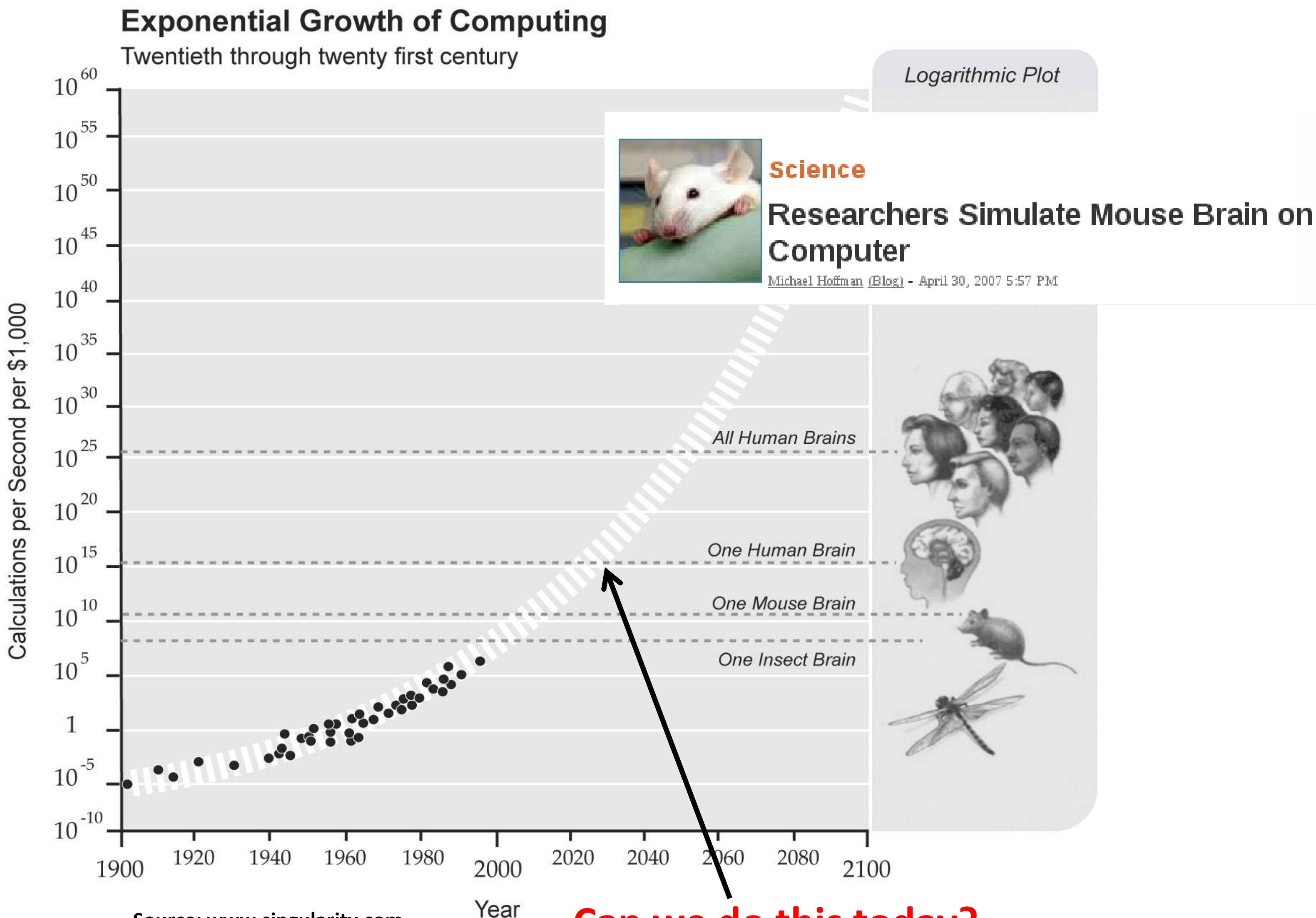
Let us assume …

- **… you were to build a machine like this …**



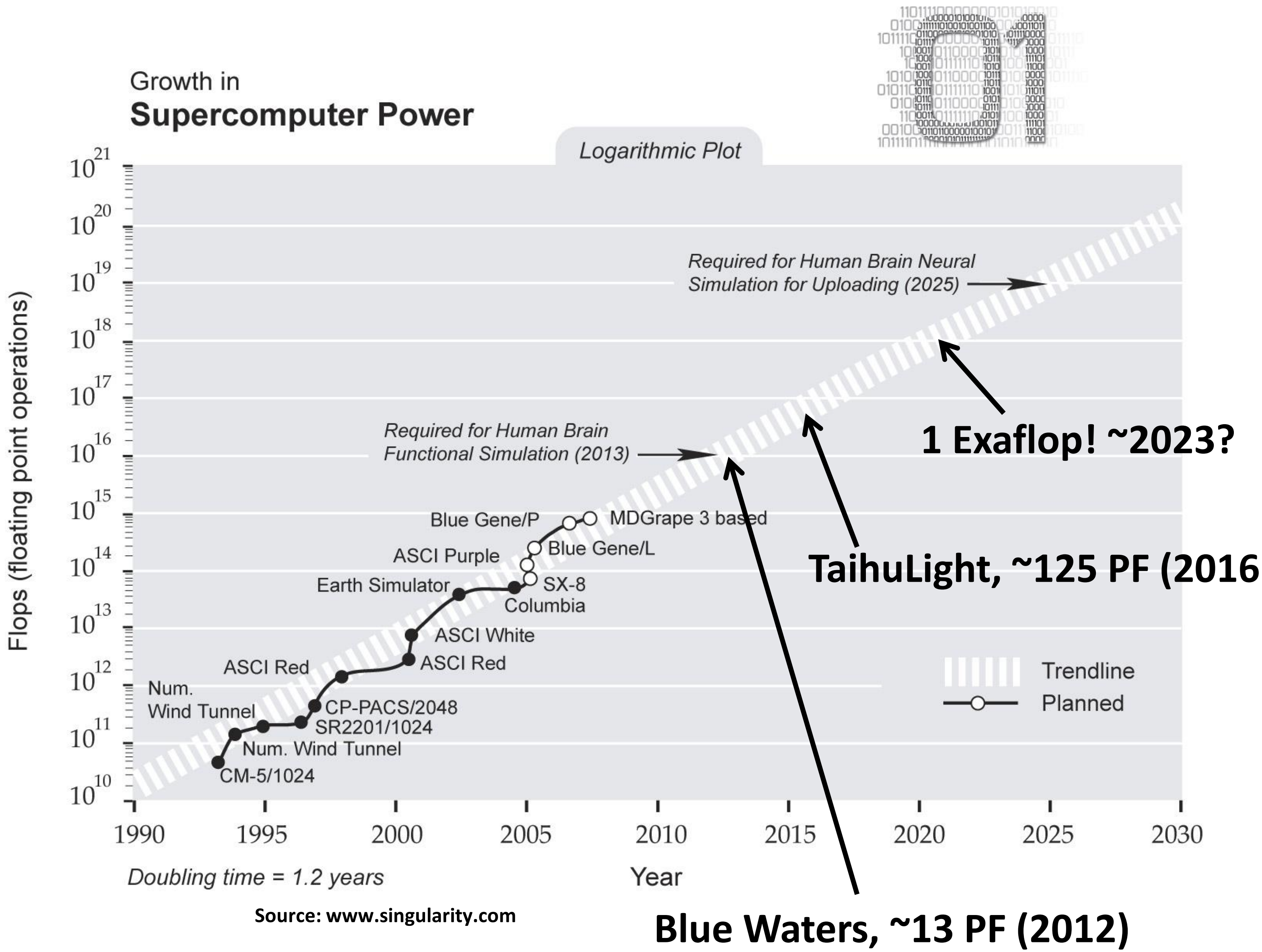
Source: wikipedia

- **… we know how each part works**
  - There are just many of them!
  - Question: How many calculations per second are needed to emulate a brain?



Source: www.singularity.com

Can we do this today?



Source: www.singularity.com


神威
太湖之光
天河
天河二号
North Korea
Seoul
서울
Yellow Sea
Wuxi, Jiangsu
Sunan Shuofang
2 h 20 min
Guangzhou, Guangdong
Laos

Human Brain – No Problem!

- **… not so fast, we need to understand how to program those machines …**

Human Brain – No Problem!

Simulating 1 second of human brain activity takes 82,944 processors

By Ryan Whitwam on August 5, 2013 at 1:34 pm | 21 Comments

The brain is a deviously complex biological computing device that even the fastest supercomputers in the world fail to emulate. Well, that's not entirely true anymore. Researchers at the Okinawa Institute of Technology Graduate University in Japan and Forschungszentrum Jülich in Germany have managed to simulate a single second of human brain activity in a very, very powerful computer.

Scooped!

Source: extremetech.com

Other problem areas: Scientific Computing

- **Most natural sciences are simulation driven or are moving towards simulation**
  - Theoretical physics (solving the Schrödinger equation, QCD)
  - Biology (Gene sequencing)
  - Chemistry (Material science)
  - Astronomy (Colliding black holes)
  - Medicine (Protein folding for drug discovery)
  - Meteorology (Storm/Tornado prediction)
  - Geology (Oil reservoir management, oil exploration)
  - and many more … (even Pringles uses HPC)

Other problem areas: Commercial Computing

- **Databases, data mining, search**
  - Amazon, Facebook, Google
- **Transaction processing**
  - Visa, Mastercard
- **Decision support**
  - Stock markets, Wall Street, Military applications
- **Parallelism in high-end systems and back-ends**
  - Often throughput-oriented
  - Used equipment varies from COTS (Google) to high-end redundant mainframes (banks)

Other problem areas: Industrial Computing

- **Aeronautics (airflow, engine, structural mechanics, electromagnetism)**
- **Automotive (crash, combustion, airflow)**
- **Computer-aided design (CAD)**
- **Pharmaceuticals (molecular modeling, protein folding, drug design)**
- **Petroleum (Reservoir analysis)**
- **Visualization (all of the above, movies, 3d)**

What can faster computers do for us?

- **Solving bigger problems than we could solve before!**
  - E.g., Gene sequencing and search, simulation of whole cells, mathematics of the brain, …
  - The size of the problem grows with the machine power

    → *Weak Scaling*

- **Solve today's problems faster!**
  - E.g., large (combinatorial) searches, mechanical simulations (aircrafts, cars, weapons, …)
  - The machine power grows with constant problem size

    → *Strong Scaling*

High-Performance Computing (HPC)

- a.k.a. “Supercomputing”
- Question: define “Supercomputer”!

High-Performance Computing (HPC)

- **a.k.a. “Supercomputing”**
- **Question: define “Supercomputer”!**
  - “A supercomputer is a computer at the frontline of contemporary processing capacity--particularly speed of calculation.” (Wikipedia)
  - Usually quite expensive ($s and kWh) and big (space)
- **HPC is a quickly growing niche market**
  - Not all “supercomputers”, wide base
  - Important enough for vendors to specialize
  - Very important in research settings (up to 40% of university spending)

    “Goodyear Puts the Rubber to the Road with High Performance Computing”

    “High Performance Computing Helps Create New Treatment For Stroke Victims”

    “Procter & Gamble: Supercomputers and the Secret Life of Coffee”

    “Motorola: Driving the Cellular Revolution With the Help of High Performance Computing”

    “Microsoft: Delivering High Performance Computing to the Masses”

The Top500 List

- **A benchmark, solve Ax=b**
  - As fast as possible! → as big as possible ☺
  - Reflects **some** applications, not all, not even many
  - Very good historic data!
- **Speed comparison for computing centers, states, countries, nations, continents ☹**
  - Politicized (sometimes good, sometimes bad)
  - Yet, fun to watch

The Top500 List (June 2015)

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National Supercomputing Center in Wuxi China	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway NRCPC	10,649,600	93,014.6	125,435.9	15,371
2	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
3	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
4	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890
5	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0	11,280.4	12,660
6	DOE/SC/Argonne National Laboratory United States	**Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786,432	8,586.6	10,066.3	3,945
7	DOE/NNSA/LANL/SNL United States	**Trinity** - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Aries interconnect Cray Inc.	301,056	8,100.9	11,078.9	
8	Swiss National Supercomputing Centre (CSCS) Switzerland	**Piz Daint** - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect , NVIDIA K20x Cray Inc.	115,984	6,271.0	7,788.9	2,325
9	HLRS - Höchstleistungsrechenzentrum Stuttgart Germany	**Hazel Hen** - Cray XC40, Xeon E5-2680v3 12C 2.5GHz, Aries interconnect Cray Inc.	185,088	5,640.2	7,403.5	

Piz Daint @ CSCS

March 19, 2013

Swiss 'GPU Supercomputer' Will Be Fastest in Europe

Tiffany Trader

Page: **1** | 2

The NVIDIA GPU Technology Conference is in full-swing today in San Jose, Calif. The annual event kicked off this morning with a keynote from NVIDIA CEO Jen-Hsun Huang, who revealed that the Swiss National Supercomputing Center (CSCS) is building Europe's fastest GPU-accelerated supercomputer, an extension of a Cray system that was announced last year.

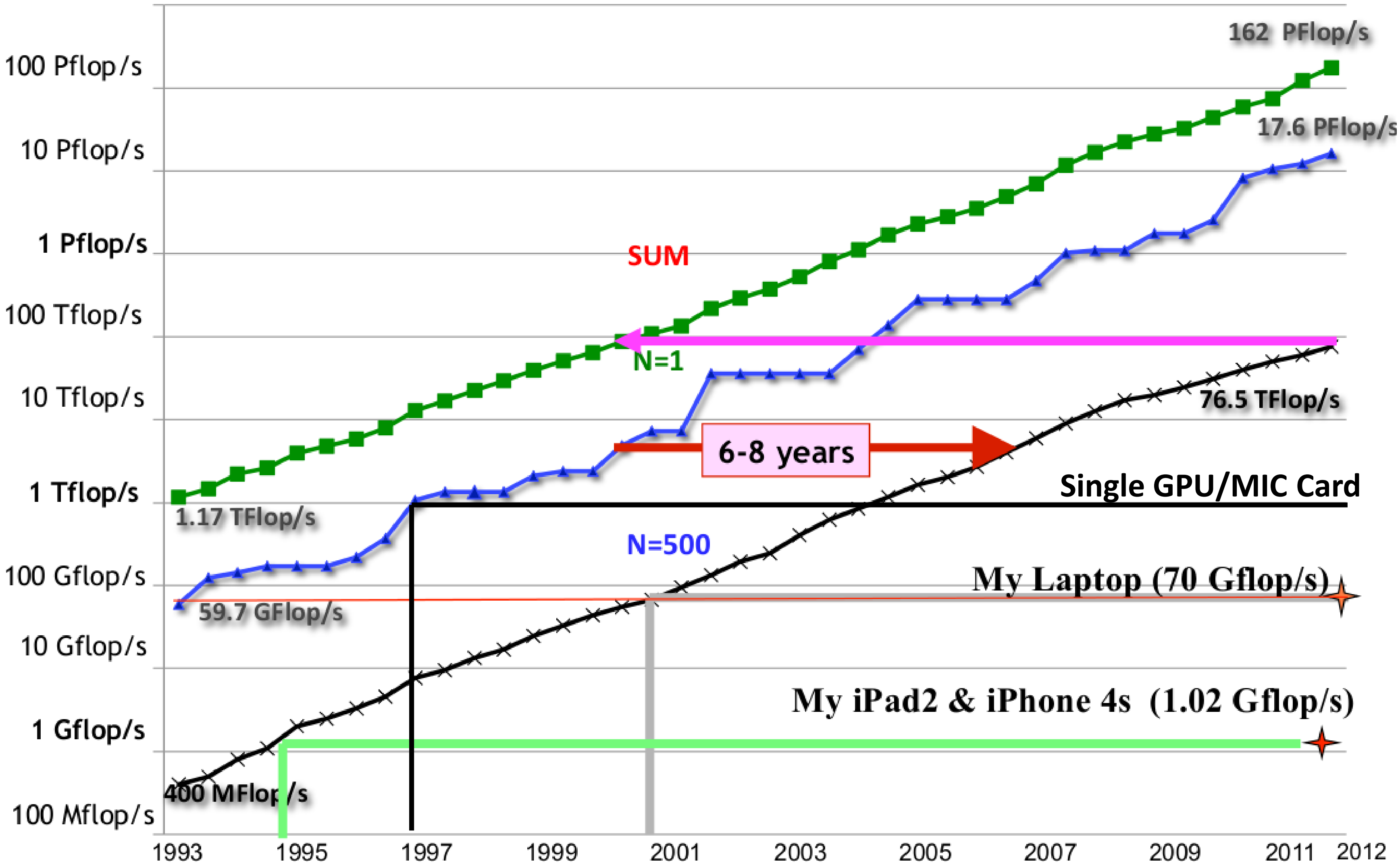
As Cray Vice President, Storage & Data Management Barry Bolding told *HPCwire*, this will be the first Cray supercomputer equipped with Intel Xeon processors and NVIDA GPUs.

CSCS is part of ETH Zurich, one of the top universities in the world and the alma mater of Albert Einstein. The supercomputing center installed phase one of its shiny new Cray XC30 back in December 2012.

Blue Waters in 2009

Blue Waters in 2012

History and Trends



Source: Jack Dongarra

High-Performance Computing grows quickly

- **Computers are used to automate many tasks**
- **Still growing exponentially**
  - New uses discovered continuously

IDC, 2007: *“The overall HPC server market grew by 15.5 percent in 2007 to reach $11.6 billion […] while the same kinds of boxes that go into HPC machinery but are used for general purpose computing, rose by only 3.6 percent to $54.4”*

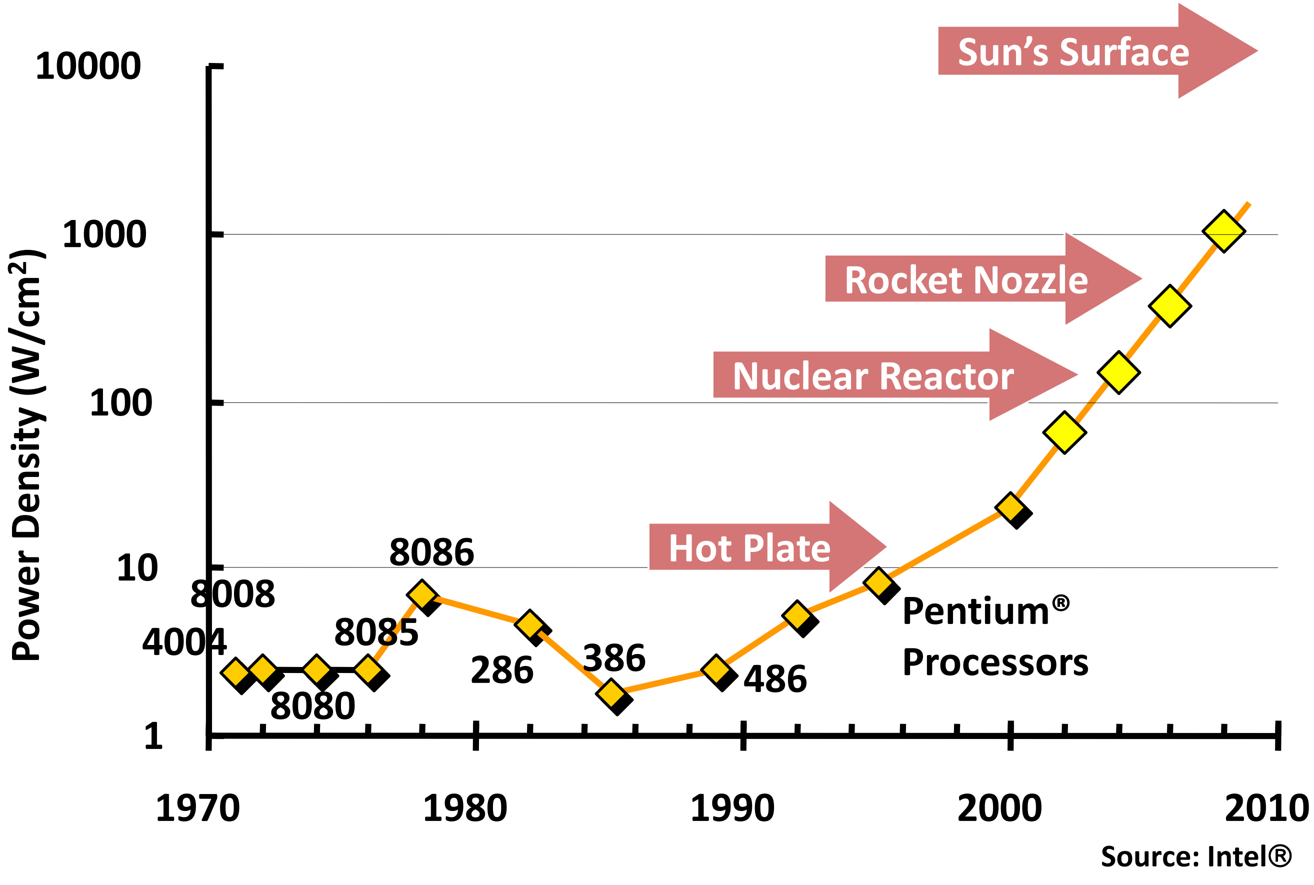
IDC, 2009: “expects the HPC technical server market to grow at a healthy 7% to 8% yearly rate to reach revenues of $13.4 billion by 2015.”

“The non-HPC portion of the server market was actually down 20.5 per cent, to $34.6bn”

Source: The Economist

How to increase the compute power?

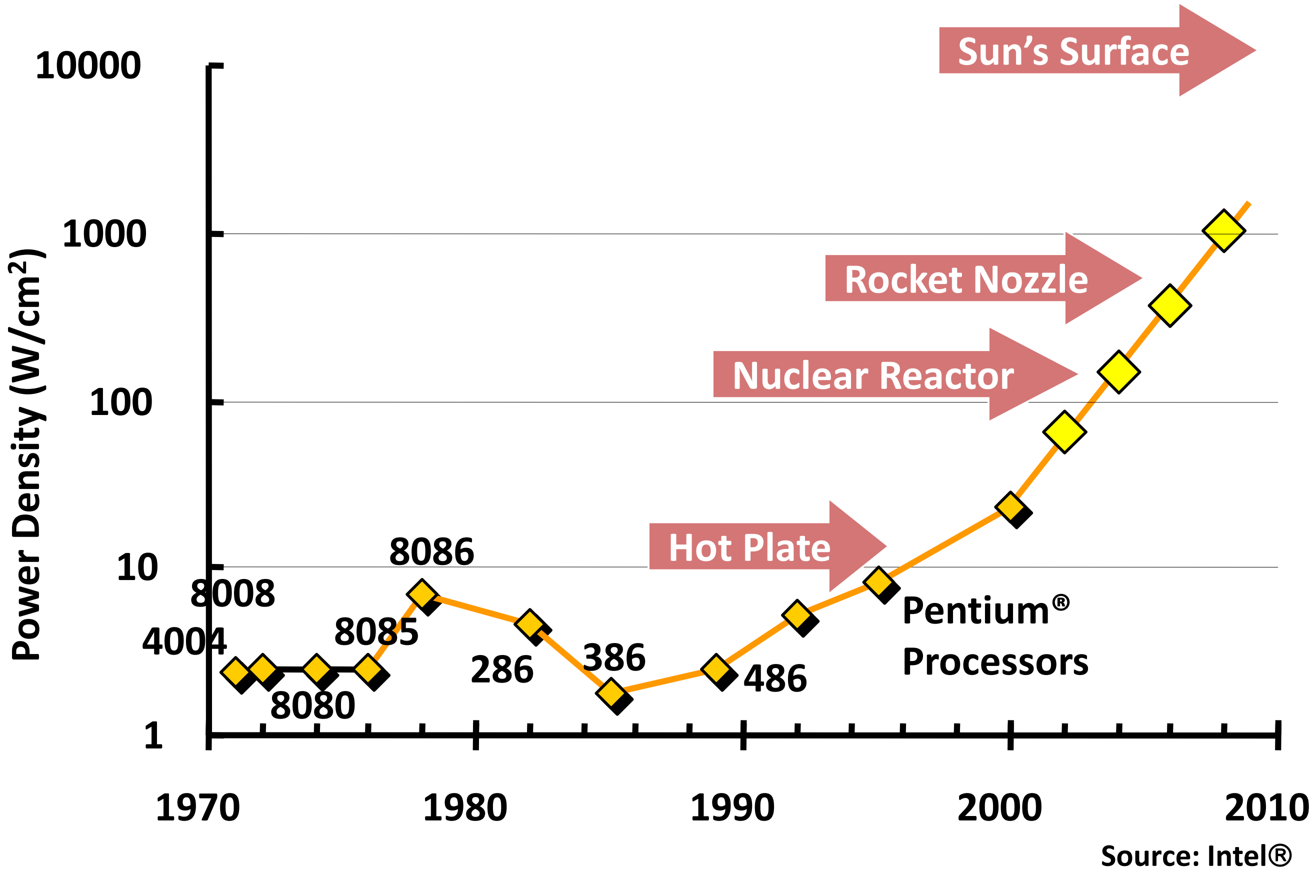
Clock Speed:



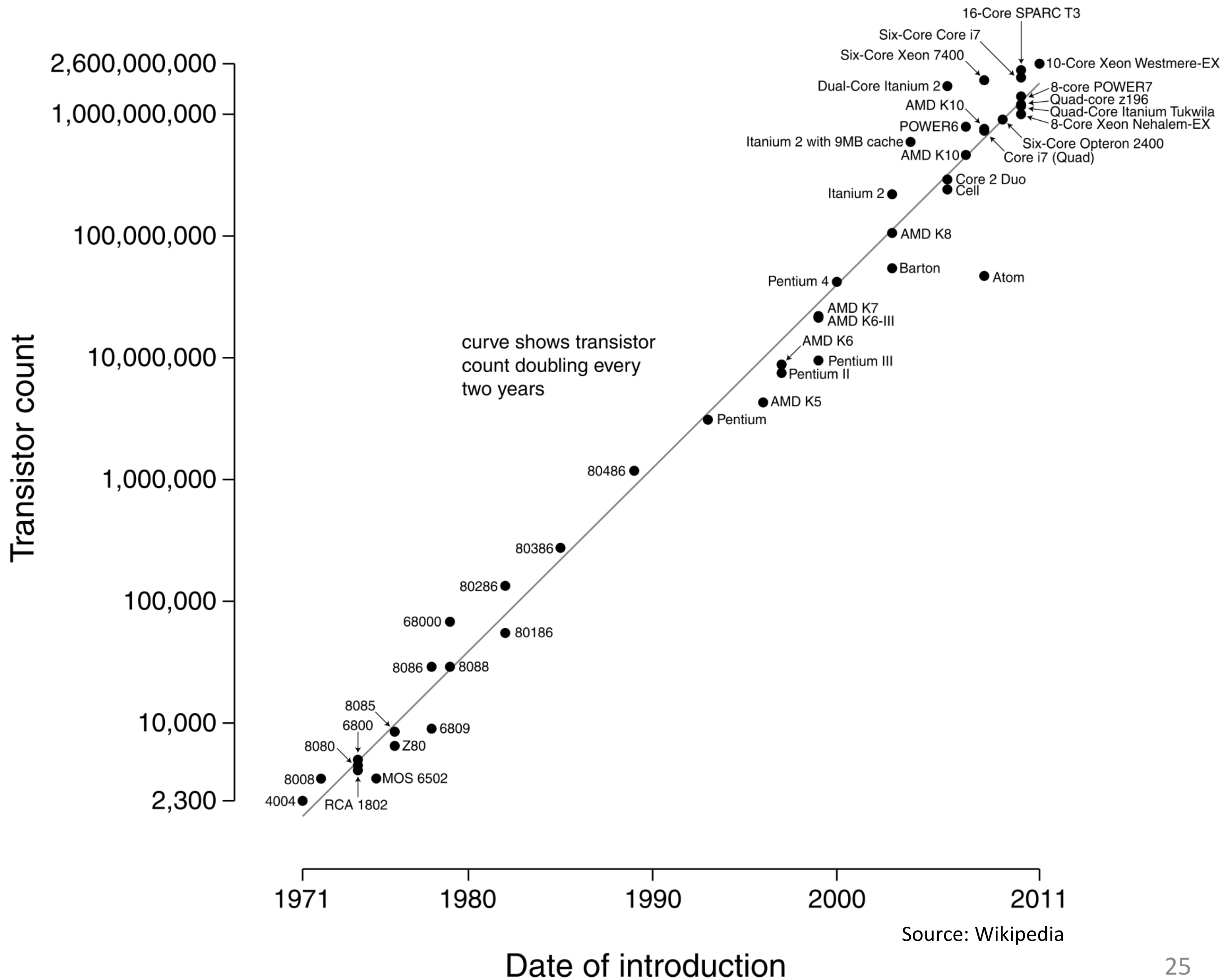
How to increase the compute power?

Not an option anymore!

~~Clock Speed:~~

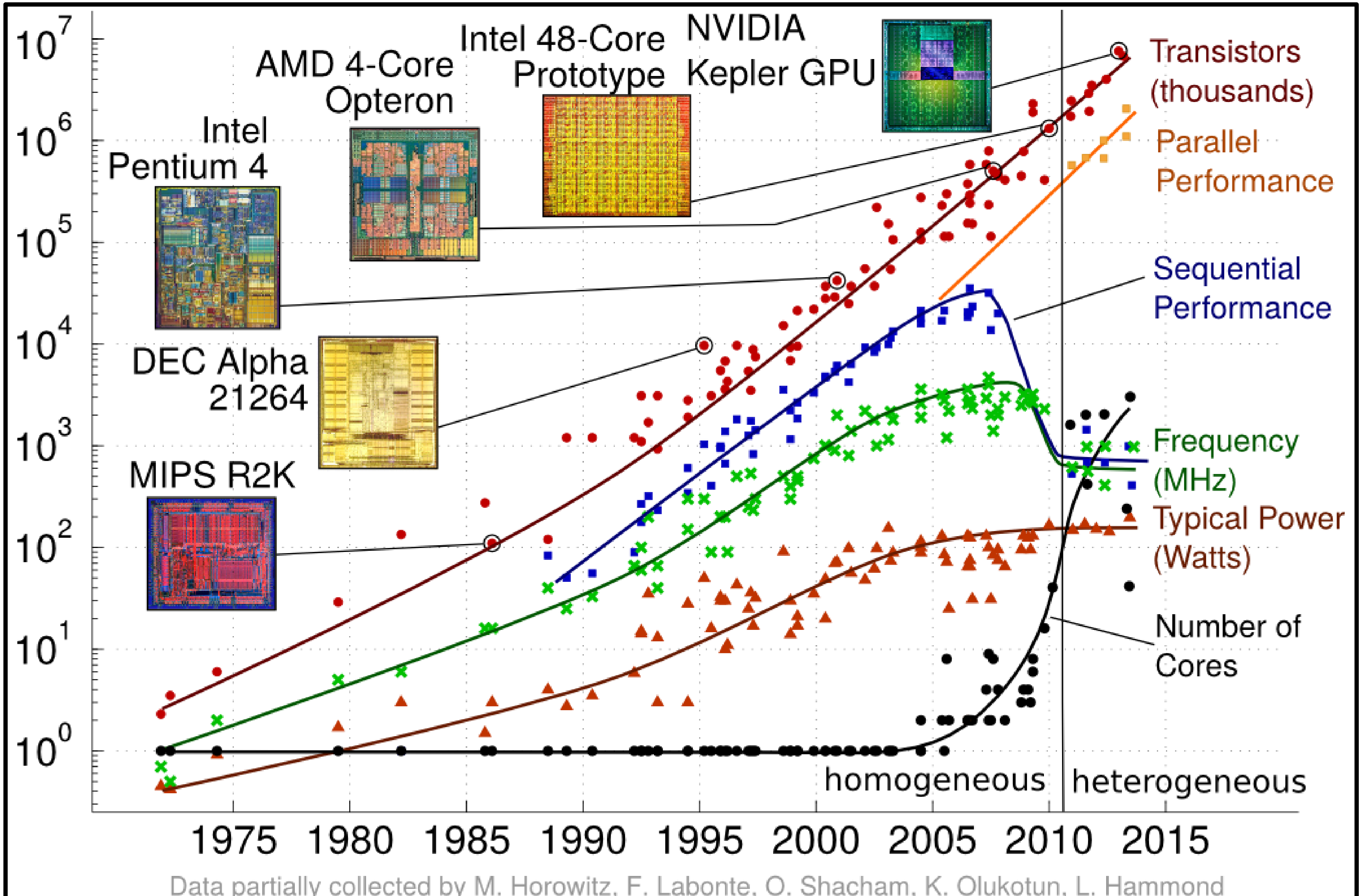


Microprocessor Transistor Counts 1971-2011 & Moore's Law



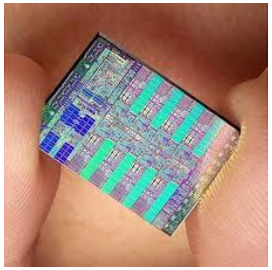
Source: Wikipedia

A more complete view



So how to invest the transistors?

- **Architectual innovations**
  - Branch prediction, Tomasulo logic/rename register, speculative execution, …
  - Help only so much ☹
- **What else?**
  - Simplification is beneficial, less transistors per CPU, more CPUs, e.g., Cell B.E., GPUs, MIC
  - We call this "cores" these days
  - Also, more intelligent devices or higher bandwidths (e.g., DMA controller, intelligent NICs)



Source: IBM

Source: NVIDIA

Source: Intel

Towards the age of massive parallelism

- **Everything goes parallel**
  - Desktop computers get more cores
    2,4,8, soon dozens, hundreds?
  - Supercomputers get more PEs (cores, nodes)
    > 3 million today
    > 50 million on the horizon
    ➢*1 billion in a couple of years (after 2020)*
- **Parallel Computing is inevitable!**

Parallel vs. Concurrent computing
Concurrent activities ***may*** be executed in parallel
Example:
A1 starts at T1, ends at T2; A2 starts at T3, ends at T4
Intervals (T1,T2) and (T3,T4) may overlap!
Parallel activities:
A1 is executed ***while*** A2 is running
Usually requires separate resources!

Goals of this lecture

- **Motivate you!**
- **What is parallel computing?**
  - And why do we need it?
- **What is high-performance computing?**
  - What's a Supercomputer and why do we care?
- **Basic overview of**
  - Programming models

    Some examples
  - Architectures

    Some case-studies
- **Provide context for coming lectures**

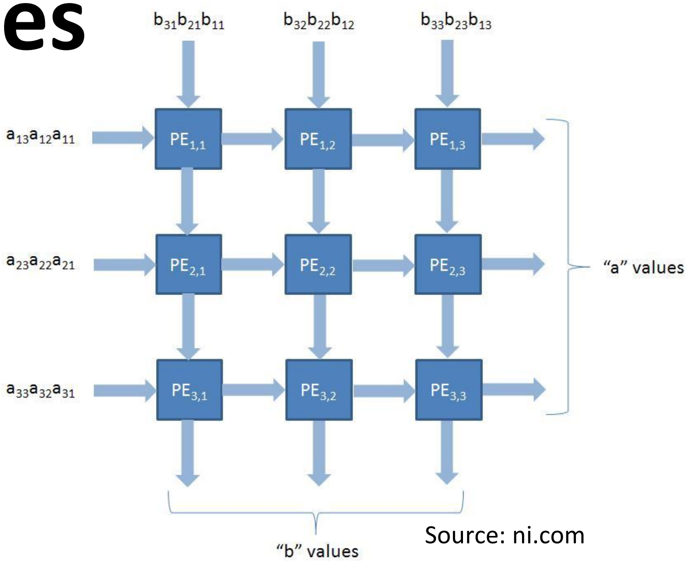
Granularity and Resources

Activities	Parallel Resource
Micro-code instruction	Instruction-level parallelism: Pipelining; VLIW; Superscalar
Machine-code instruction (complex or simple)	SIMD operations: Vector operations
Sequence of machine-code instructions: *Blocks*, *Loops*, *Loop nests*, *Functions*, *Function sequences*	Instruction sequences: Multiprocessors; Multicores; Multithreading

Resources and Programming

Parallel Resource	Programming
▪ Instruction-level parallelism ▪ Pipelining ▪ VLIW ▪ Superscalar	▪ Compiler ▪ (inline assembly) ▪ Hardware scheduling
▪ SIMD operations ▪ Vector operations	▪ Compiler (inline assembly) ▪ Libraries
▪ Instruction sequences ▪ Multiprocessors ▪ Multicores ▪ Multithreading	▪ Compilers (very limited) ▪ Expert programmers ▪ Parallel languages ▪ Parallel libraries ▪ Hints

Historic Architecture Examples

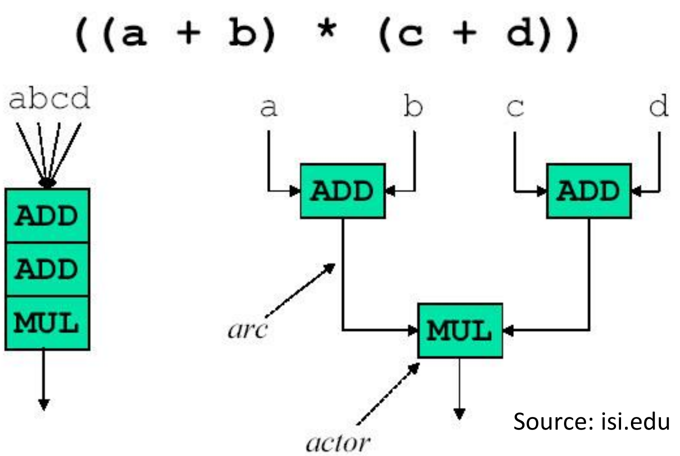


Source: ni.com

- **Systolic Array**
  - Data-stream driven (data counters)
  - Multiple streams for parallelism
  - Specialized for applications (reconfigurable)

- **Dataflow Architectures**
  - No program counter, execute instructions when all input arguments are available
  - Fine-grained, high overheads

    *Example: compute f = (a+b) * (c+d)*



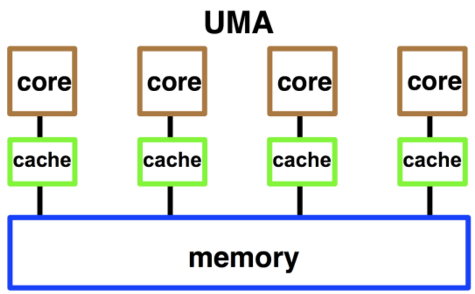
Source: isi.edu

Von Neumann Architecture

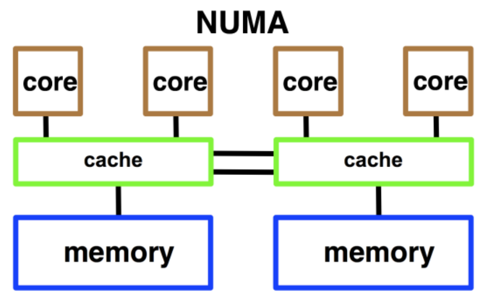
- **Program counter → Inherently serial!**
  Retrospectively define parallelism in instructions and data

SISD **Standard Serial Computer (nearly extinct)**	**SIMD** **Vector Machines or Extensions (very common)**
MISD **Redundant Execution (fault tolerance)**	**MIMD** **Multicore (ubiquitous)**

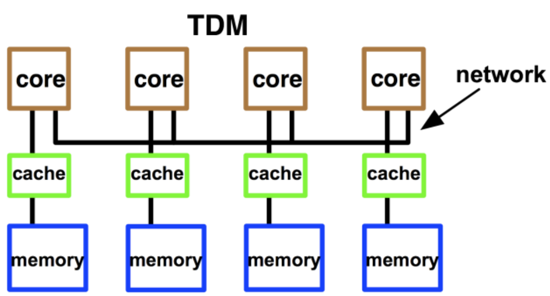
Parallel Architectures 101



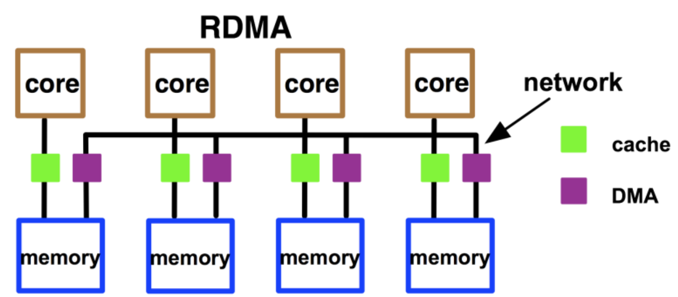
Today's laptops



Today's servers



Yesterday's clusters

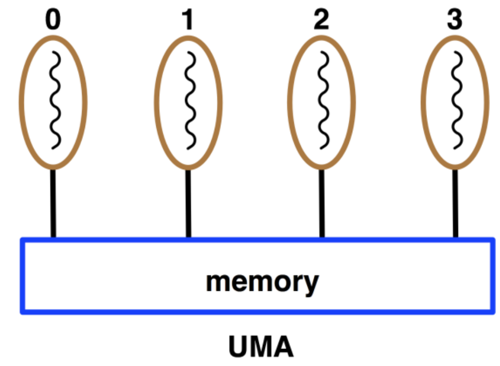


Today's clusters

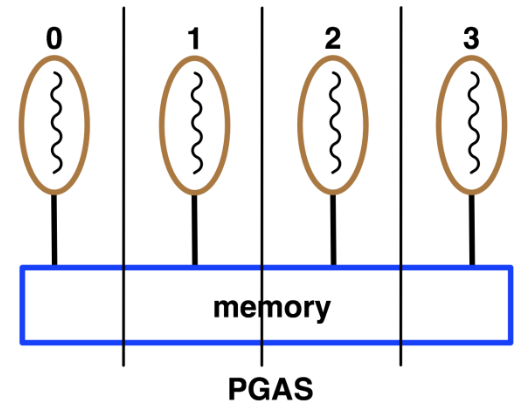
- … and mixtures of those

Programming Models

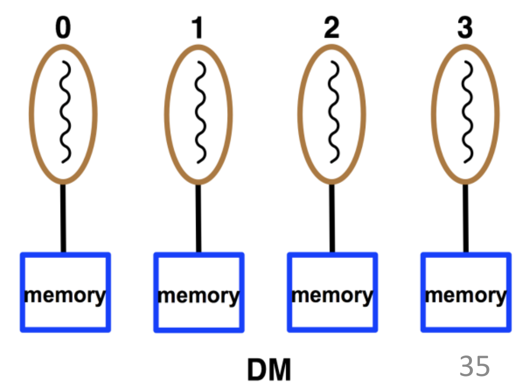
- **Shared Memory Programming (SM/UMA)**
  - Shared address space
  - Implicit communication
  - Hardware for cache-coherent remote memory access
  - Cache-coherent Non Uniform Memory Access (cc NUMA)



- **(Partitioned) Global Address Space (PGAS)**
  - Remote Memory Access
  - Remote vs. local memory (cf. ncc-NUMA)



- **Distributed Memory Programming (DM)**
  - Explicit communication (typically messages)
  - Message Passing



Shared Memory Machines

Source: IBM

- **Two historical architectures:**
  - "Mainframe" – all-to-all connection between memory, I/O and PEs

    Often used if PE is the most expensive part

    Bandwidth scales with P

    PE Cost scales with P, ***Question: what about network cost?***

Shared Memory Machines

Source: IBM

- **Two historical architectures:**
  - "Mainframe" – all-to-all connection between memory, I/O and PEs

    Often used if PE is the most expensive part

    Bandwidth scales with P

    PE Cost scales with P, ***Question: what about network cost?***

    Answer: *Cost can be cut with multistage connections (butterfly)*

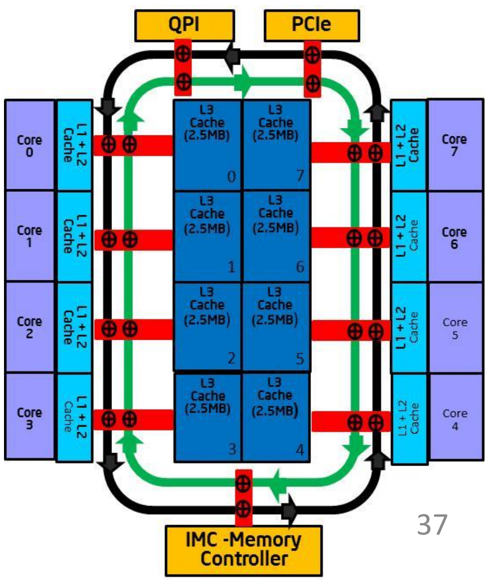
  - "Minicomputer" – bus-based connection

    All traditional SMP systems

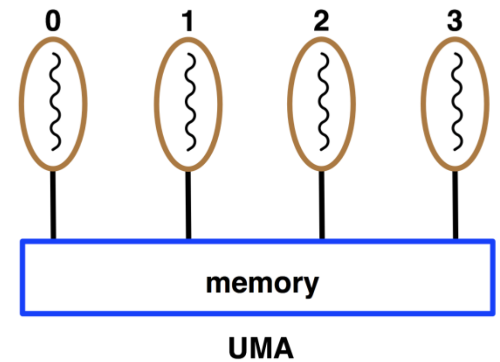
    High latency, low bandwidth (cache is important)

    Tricky to achieve highest performance (contention)

    Low cost, extensible

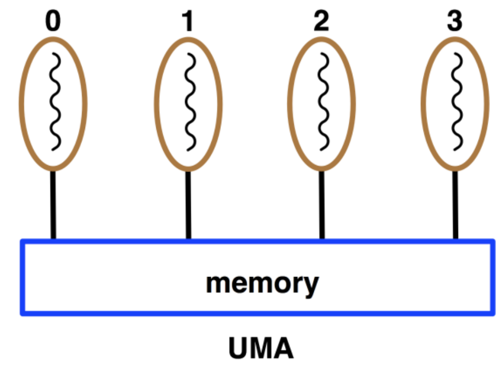


Shared Memory Machine Abstractions



- **Any PE can access all memory**
  - Any I/O can access all memory (maybe limited)
- **OS (resource management) can run on any PE**
  - Can run multiple threads in shared memory
  - Used since 40+ years
- **Communication through shared memory**
  - Load/store commands to memory controller
  - Communication is implicit
  - Requires coordination
- **Coordination through shared memory**
  - Complex topic
  - Memory models

Shared Memory Machine Programming



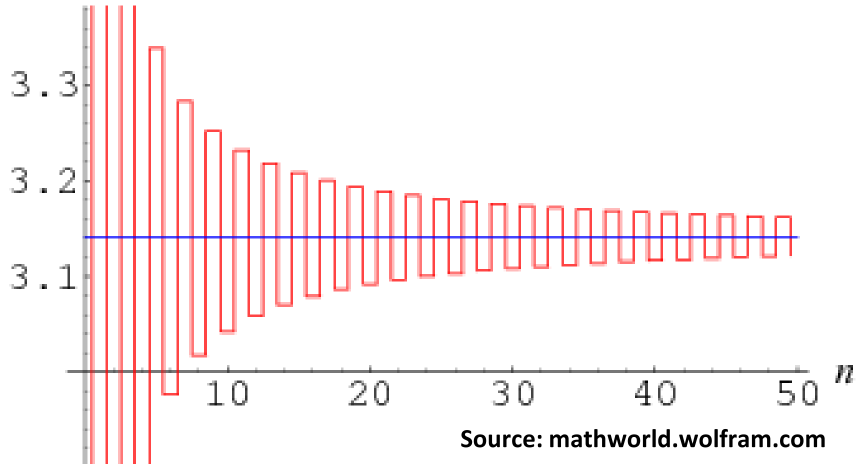
- **Threads or processes**
  - Communication through memory
- **Synchronization through memory or OS objects**
  - Lock/mutex (protect critical region)
  - Semaphore (generalization of mutex (binary sem.))
  - Barrier (synchronize a group of activities)
  - Atomic Operations (CAS, Fetch-and-add)
  - Transactional Memory (execute regions atomically)
- **Practical Models:**
  - Posix threads
  - MPI-3
  - OpenMP
  - Others: Java Threads, Qthreads, …

An SMM Example: Compute Pi

- **Using Gregory-Leibnitz Series:**

$$4\sum_{k=0}^{\infty}\frac{(-1)^k}{2k+1}$$

  - Iterations of sum can be computed in parallel
  - Needs to sum all contributions at the end



Source: mathworld.wolfram.com

Pthreads Compute Pi Example

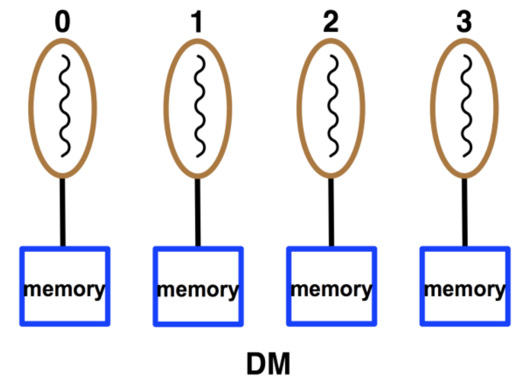
```
int main( int argc, char *argv[] )
{
   // definitions …
   thread_arr = (pthread_t*)malloc(nthreads * sizeof(pthread_t));
   resultarr= (double*)malloc(nthreads * sizeof(double));

   for (i=0; i<nthreads; ++i) {
     int ret = pthread_create( &thread_arr[i], NULL,
                    compute_pi, (void*) i);
   }
   for (i=0; i<nthreads; ++i) {
     pthread_join( thread_arr[i], NULL);
   }
   pi = 0;
   for (i=0; i<nthreads; ++i) pi += resultarr[i];

   printf ("pi is approximately %.16f, Error is %.16f\n",
              pi, fabs(pi - PI25DT));
}
```

```
int n=10000;
double *resultarr;
int nthreads;

void *compute_pi(void *data) {
  int i, j;
  int myid = (int)(long)data;
  double mypi, h, x, sum;

  for (j=0; j<n; ++j) {
    h   = 1.0 / (double) n;
    sum = 0.0;
    for (i = myid + 1; i <= n; i += nthreads) {
      x = h * ((double)i - 0.5);
      sum += (4.0 / (1.0 + x*x));
    }
    mypi = h * sum;
  }
  resultarr[myid] = mypi;
}
```

Additional comments on SMM

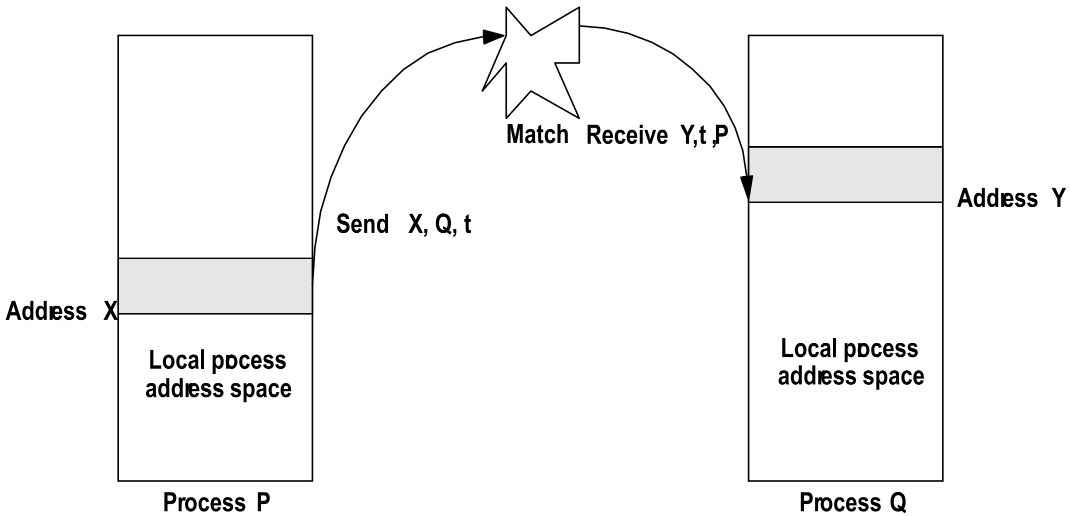
- **OpenMP would allow to implement this example much simpler (but has other issues)**
- **Transparent shared memory has some issues in practice:**
  - False sharing (e.g., resultarr[])
  - Race conditions (complex mutual exclusion protocols)
  - Little tool support (debuggers need some work)
- ***Achieving performance is harder than it seems!***

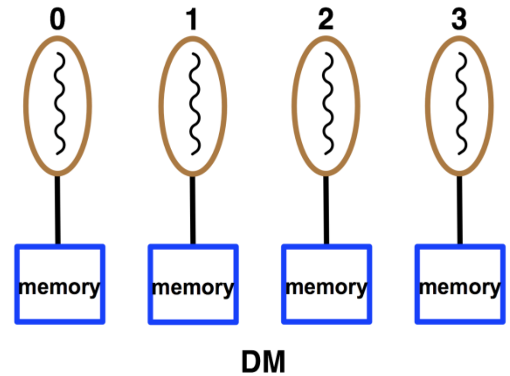
Distributed Memory Machine Programming



- **Explicit communication between PEs**
  - Message passing or channels
- **Only local memory access, no direct access to remote memory**
  - No shared resources (well, the network)
- **Programming model: Message Passing (MPI, PVM)**
  - Communication through messages or group operations (broadcast, reduce, etc.)
  - Synchronization through messages (sometimes unwanted side effect) or group operations (barrier)
  - Typically supports message matching and communication contexts

DMM Example: Message Passing

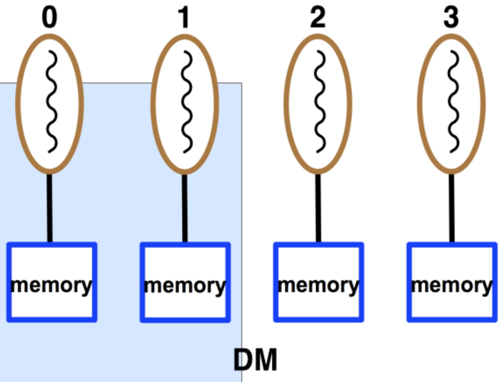




Source: John Mellor-Crummey

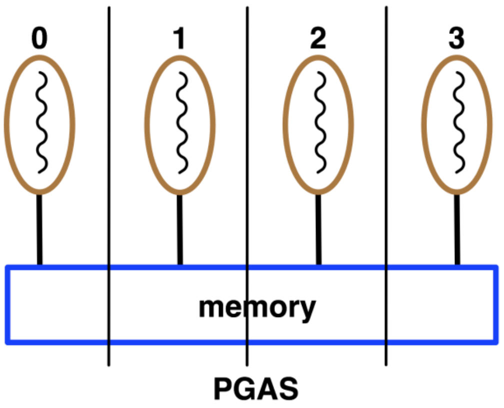
- Send specifies buffer to be transmitted
- Recv specifies buffer to receive into
- Implies copy operation between named PEs
- Optional tag matching
- Pair-wise synchronization (cf. happens before)

DMM MPI Compute Pi Example




```
int main( int argc, char *argv[] ) {
  // definitions
  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myid);

  double t = -MPI_Wtime();
  for (j=0; j<n; ++j) {
    h   = 1.0 / (double) n;
    sum = 0.0;
    for (i = myid + 1; i <= n; i += numprocs) { x = h * ((double)i - 0.5);  sum += (4.0 / (1.0 + x*x)); }
    mypi = h * sum;
    MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
  }
  t+=MPI_Wtime();

  if (!myid) {
    printf("pi is approximately %.16f, Error is %.16f\n", pi, fabs(pi - PI25DT));
    printf("time: %f\n", t);
  }

  MPI_Finalize();
}
```

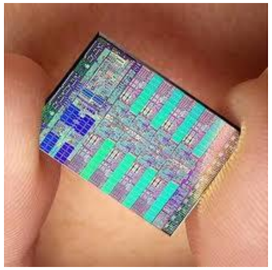
DMM Example: PGAS



- **Partitioned Global Address Space**
  - Shared memory emulation for DMM
    Usually non-coherent
  - "Distributed Shared Memory"
    Usually coherent
- **Simplifies shared access to distributed data**
  - Has similar problems as SMM programming
  - Sometimes lacks performance transparency
    Local vs. remote accesses
- **Examples:**
  - UPC, CAF, Titanium, X10, …

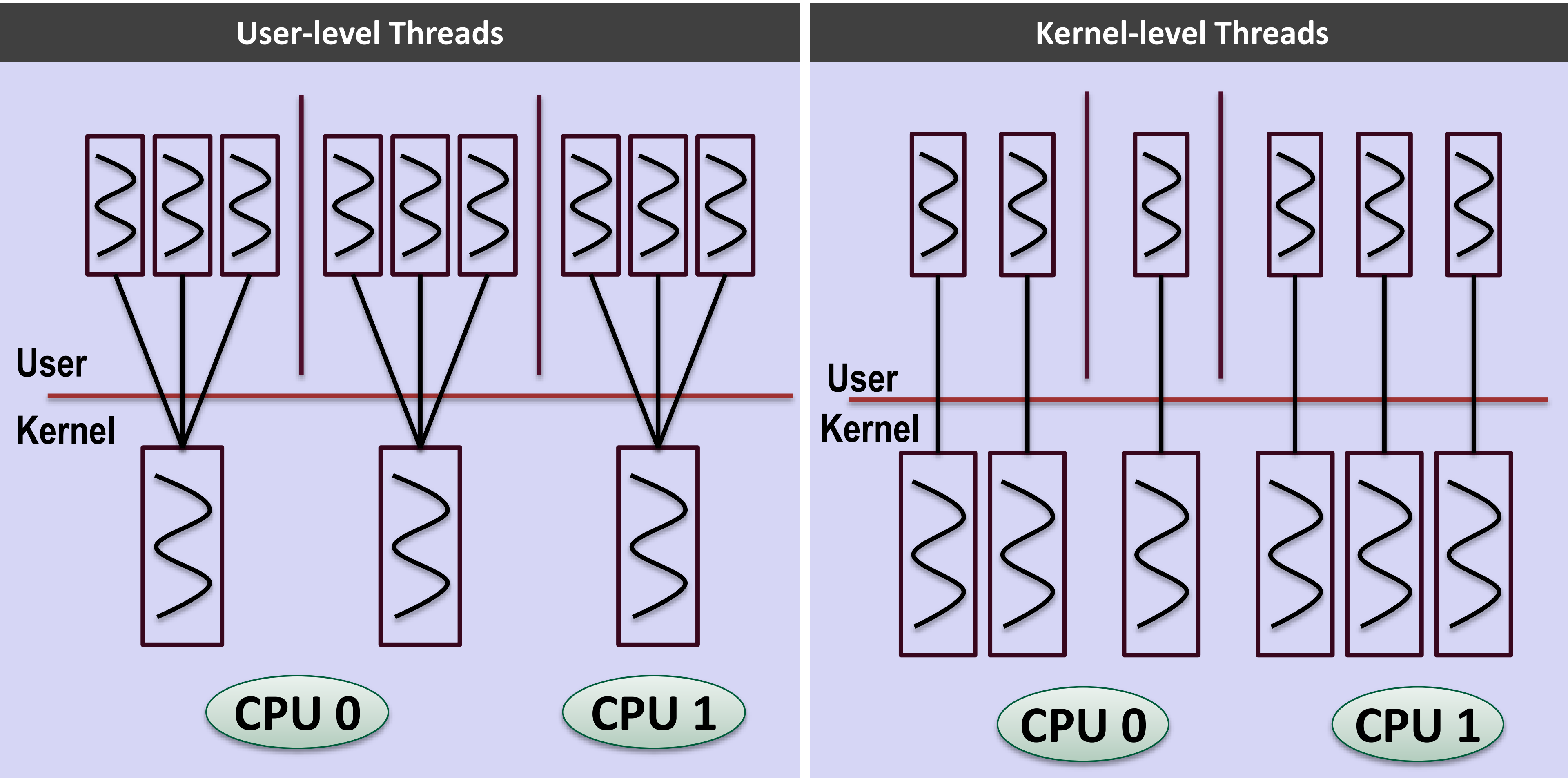
How to Tame the Beast?

- **How to program large machines?**
- **No single approach, PMs are not converging yet**
  - MPI, PGAS, OpenMP, Hybrid (MPI+OpenMP, MPI+MPI, MPI+PGAS?), …
- **Architectures converge**
  - General purpose nodes connected by general purpose or specialized networks
  - Small scale often uses commodity networks
  - Specialized networks become necessary at scale
- **Even worse: accelerators (not covered in this class, yet)**



Practical SMM Programming: Pthreads

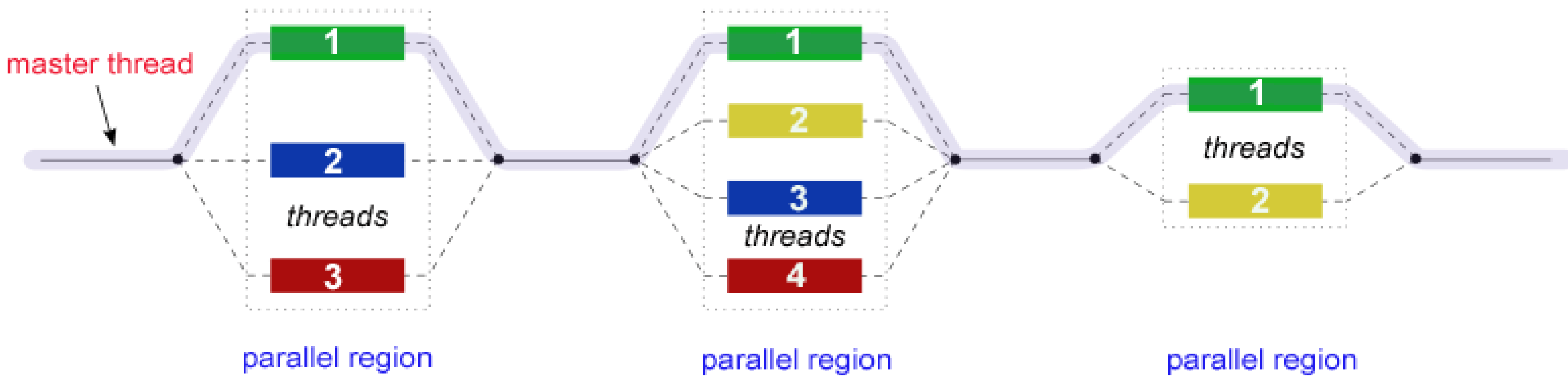
Covered in example, small set of functions for thread creation and management



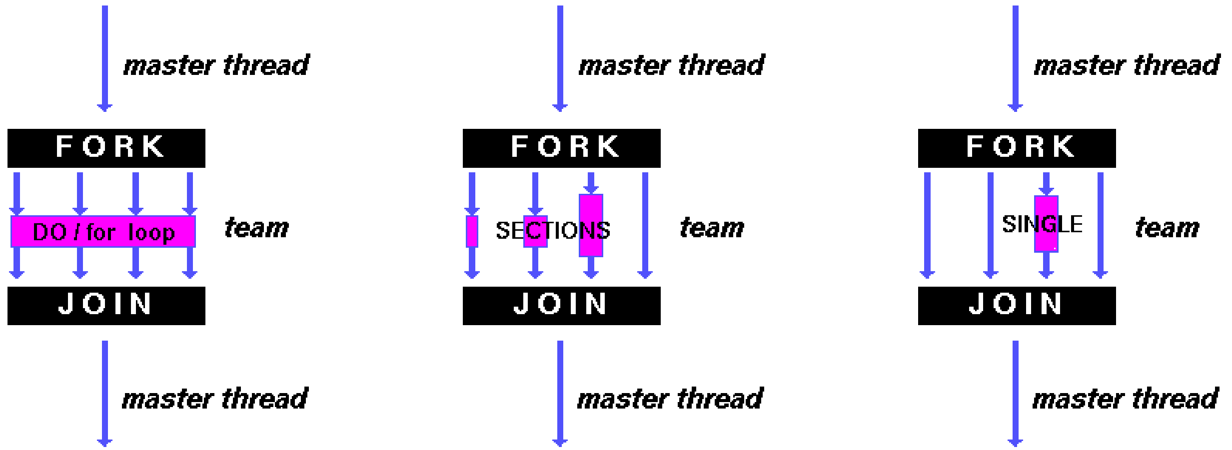
Practical SMM Programming: OpenMP

Source: OpenMP.org

- Fork-join model



- Types of constructs:



+ Tasks

Source: Blaise Barney, LLNL

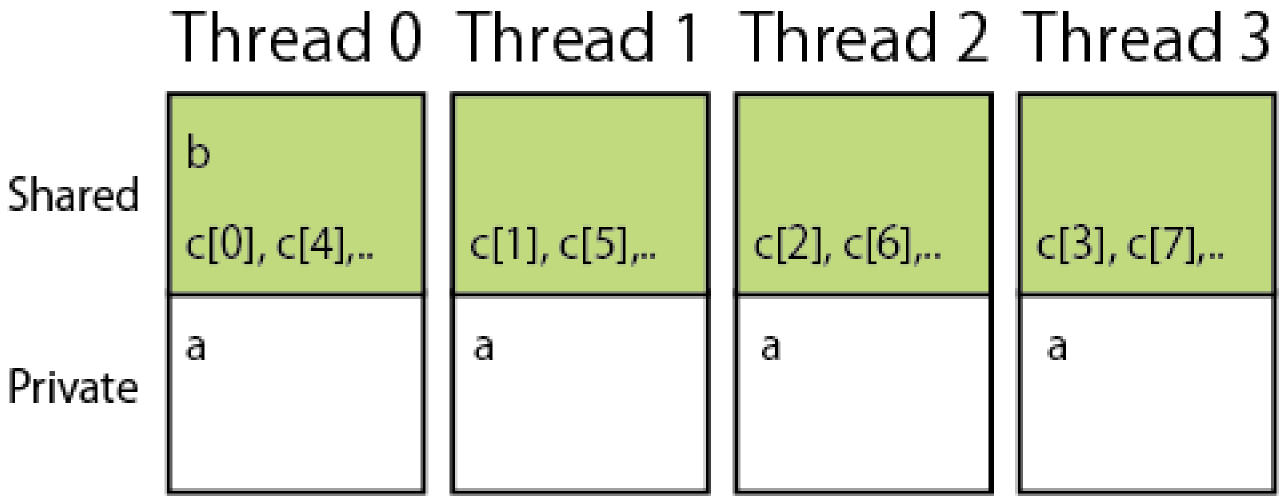
OpenMP General Code Structure

```
#include <omp.h>

main ()  {
        int var1, var2, var3;
        // Serial code

        // Beginning of parallel section. Fork a team of threads. Specify variable scoping
        #pragma omp parallel private(var1, var2) shared(var3)
        {
                // Parallel section executed by all threads
                // Other OpenMP directives
                // Run-time Library calls
                // All threads join master thread and disband
        }
        // Resume serial code
}
```

Source: Blaise Barney, LLNL

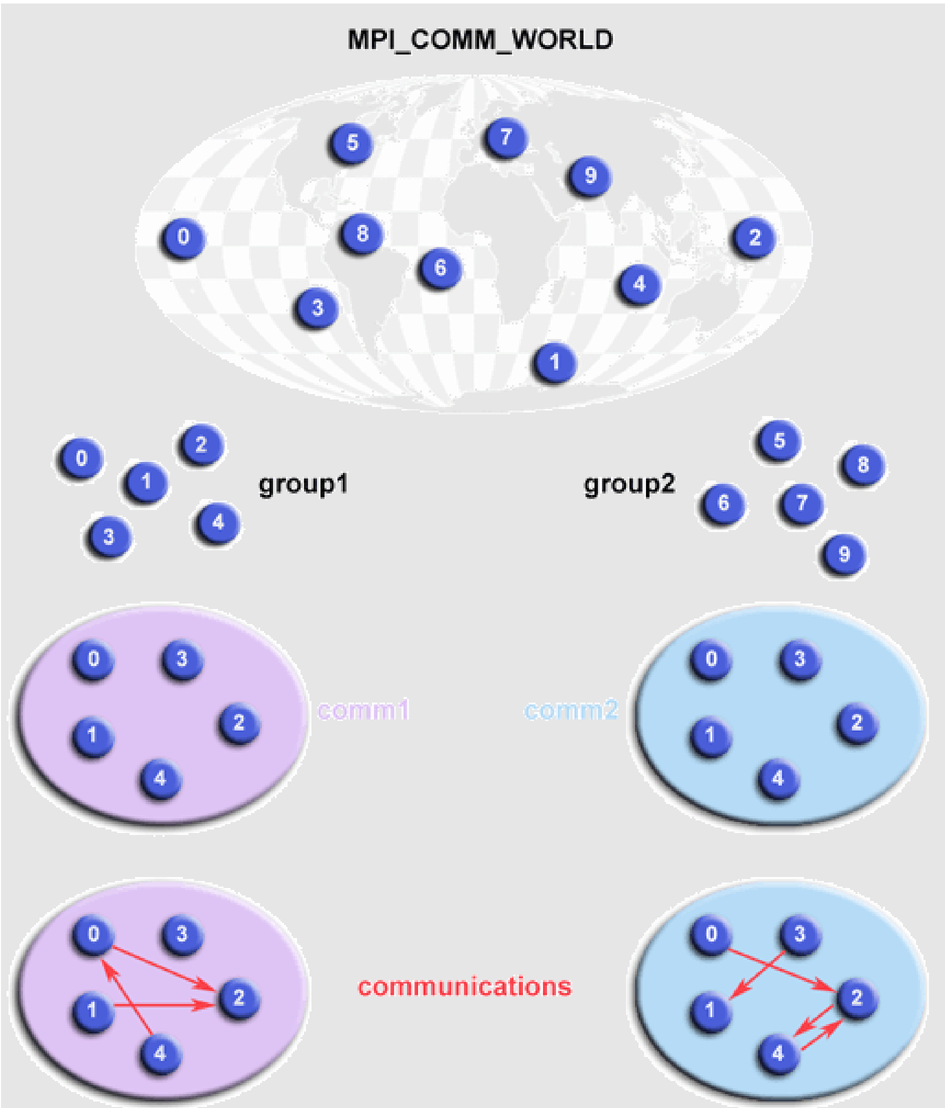
Practical PGAS Programming: UPC

- **PGAS extension to the C99 language**



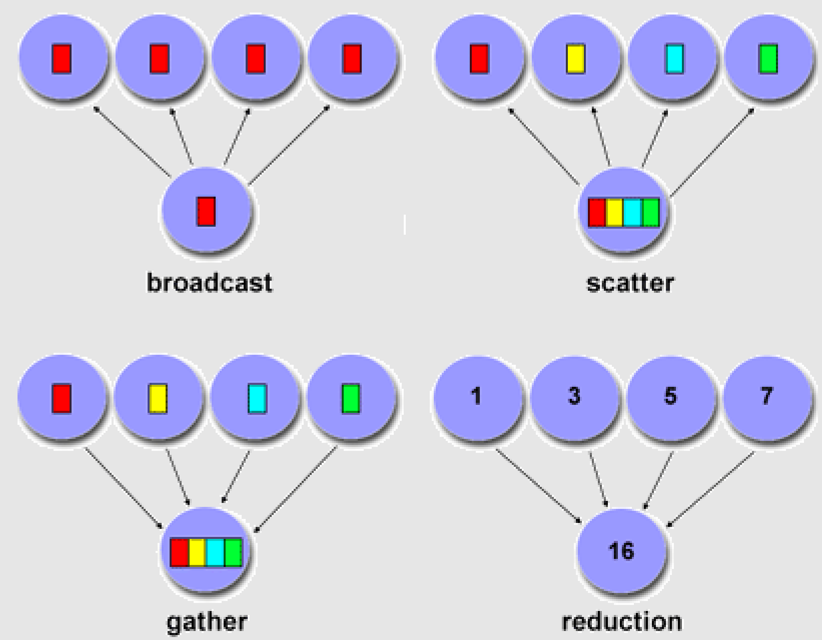
- **Many helper library functions**
  - Collective and remote allocation
  - Collective operations
- **Complex consistency model**

Practical DMM Programming: MPI-1



Collection of 1D address spaces

Helper Functions



0 (0,0)	1 (0,1)	2 (0,2)	3 (0,3)
4 (1,0)	5 (1,1)	6 (1,2)	7 (1,3)
8 (2,0)	9 (2,1)	10 (2,2)	11 (2,3)
12 (3,0)	13 (3,1)	14 (3,2)	15 (3,3)

many more (>600 total)

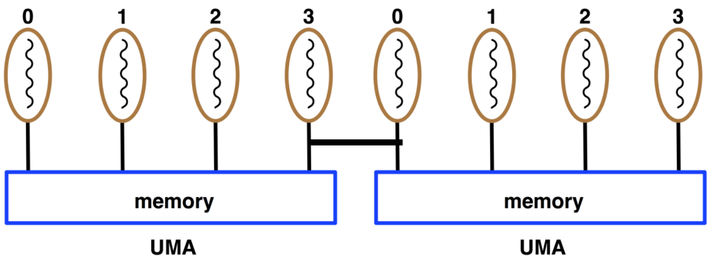
Source: Blaise Barney, LLNL

Complete Six Function MPI-1 Example

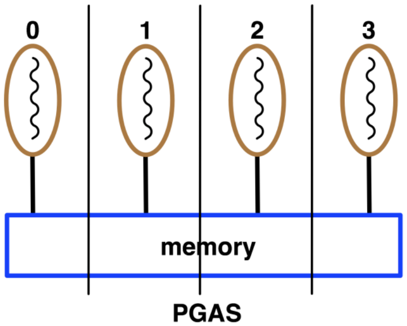
```
#include <mpi.h>

int main(int argc, char **argv) {
  int myrank, sbuf=23, rbuf=32;
  MPI_Init(&argc, &argv);

  /* Find out my identity in the default communicator */
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  if (myrank == 0) {
   MPI_Send(&sbuf,                /* message buffer */
        1,                        /* one data item */
        MPI_INT,                  /* data item is an integer */
        rank,                     /* destination process rank */
        99,                       /* user chosen message tag */
        MPI_COMM_WORLD);          /* default communicator */
  } else {
   MPI_Recv(&rbuf,  MPI_DOUBLE,  0, 99, MPI_COMM_WORLD,  &status);
   printf(“received: %i\n”, rbuf);
  }

  MPI_Finalize();
}
```

MPI-2/3: Greatly enhanced functionality

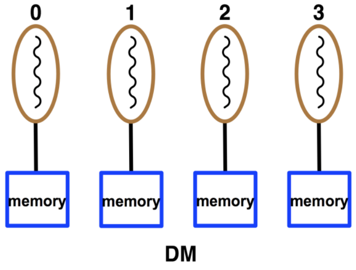
- **Support for shared memory in SMM domains**



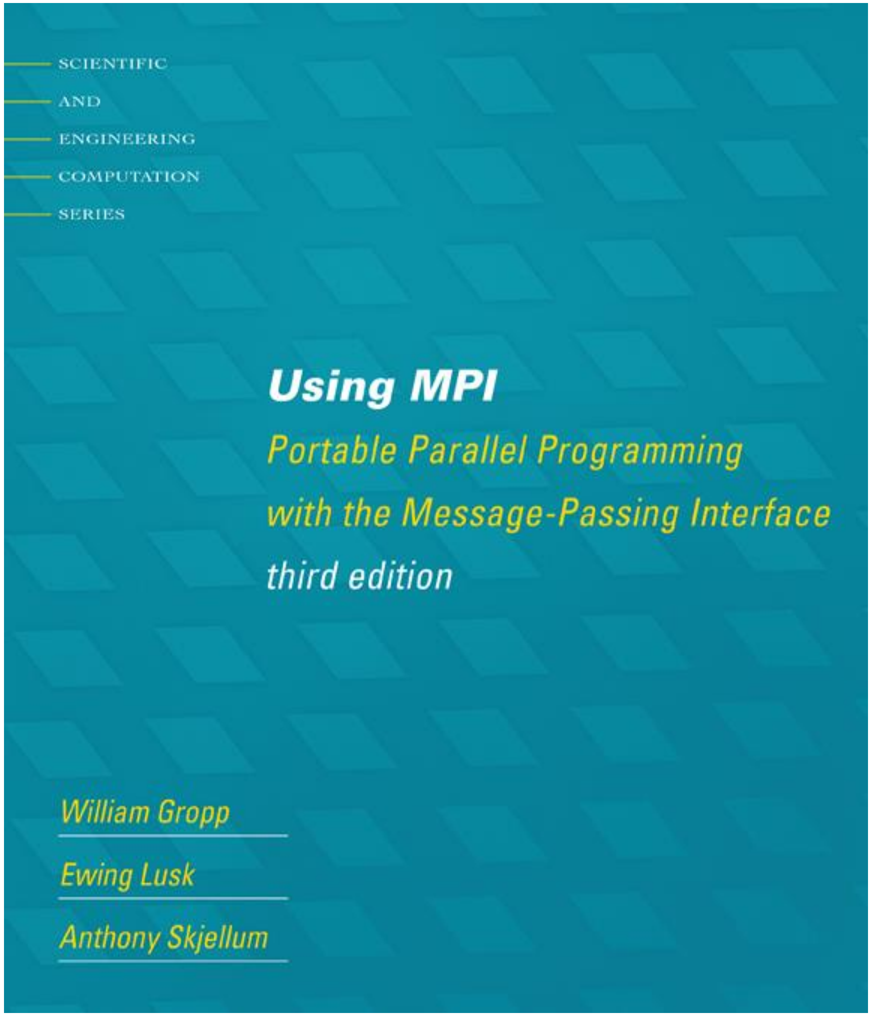
- **Support for Remote Memory Access Programming**
  - Direct use of RDMA
  - Essentially PGAS

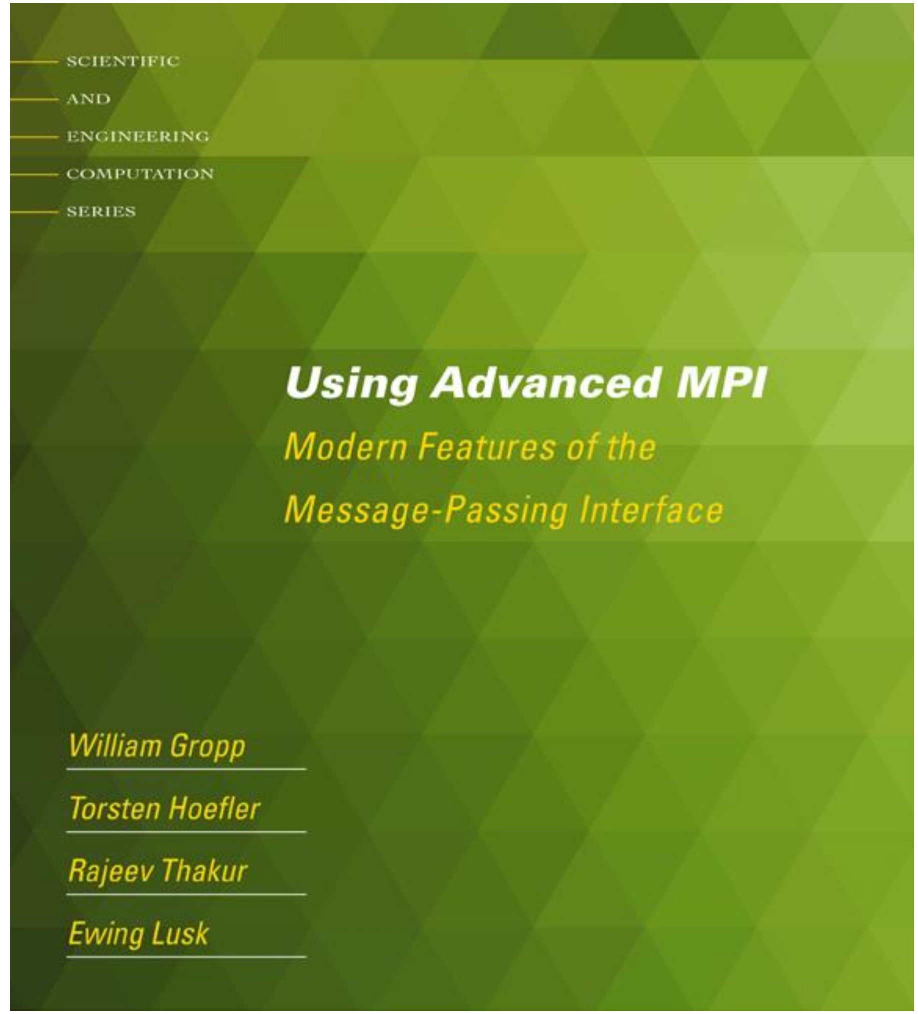


- **Enhanced support for message passing communication**
  - Scalable topologies
  - More nonblocking features
  - … many more



MPI: de-facto large-scale prog. standard



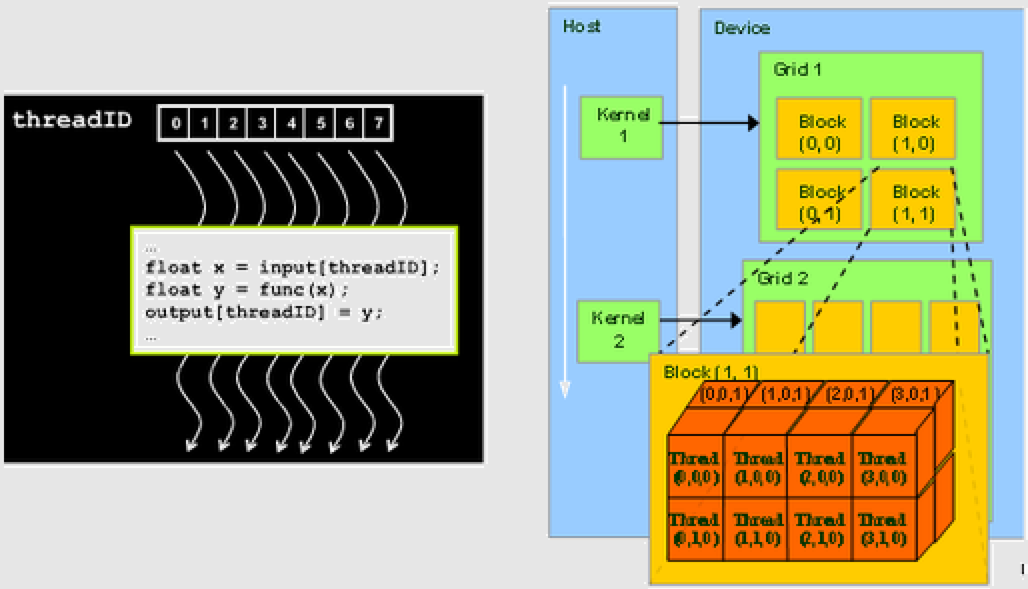


Basic MPI

Advanced MPI, including MPI-3

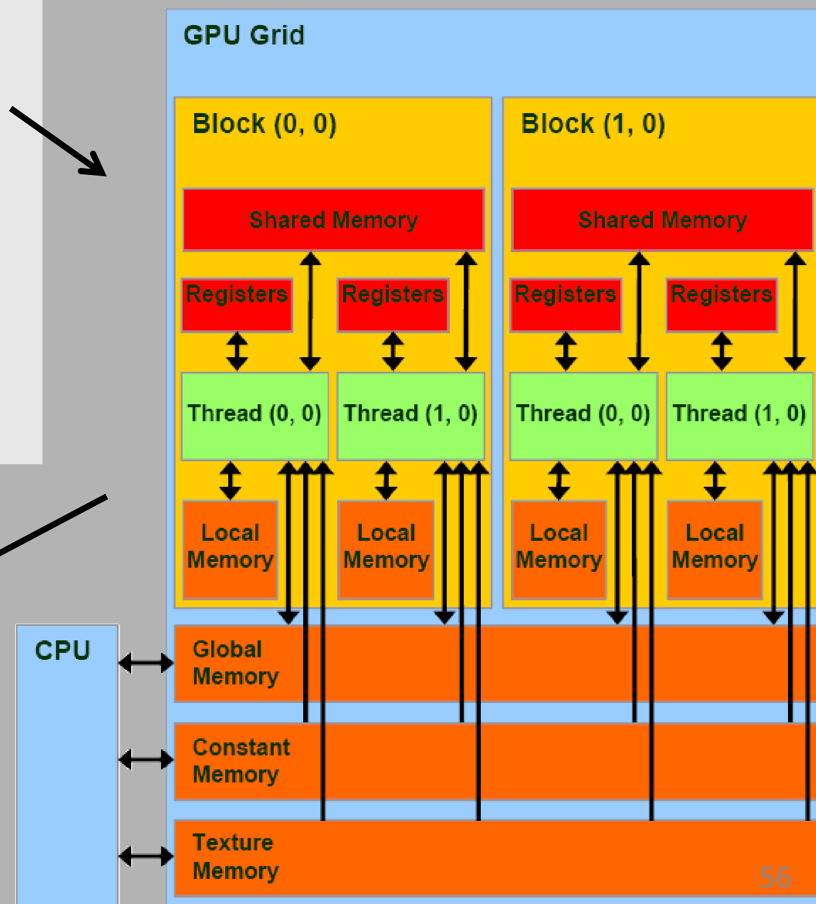
Accelerator example: CUDA

Hierarchy of Threads

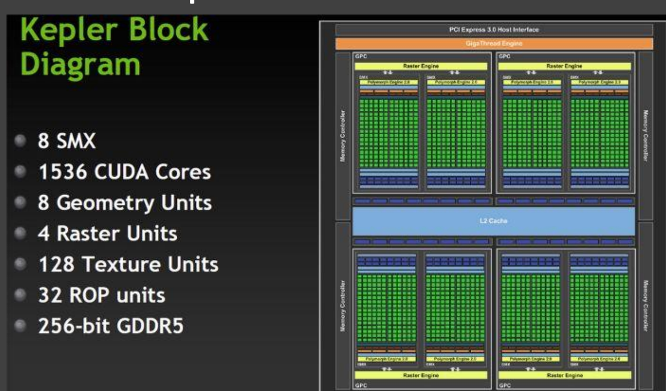


Source: NVIDIA

Complex Memory Model



Simple Architecture



Accelerator example: CUDA

Host Code

```
#define N 10
int main( void ) {
  int a[N], b[N], c[N];
  int *dev_a, *dev_b, *dev_c;
  // allocate the memory on the GPU
  cudaMalloc( (void**)&dev_a, N * sizeof(int) );
  cudaMalloc( (void**)&dev_b, N * sizeof(int) );
  cudaMalloc( (void**)&dev_c, N * sizeof(int) );
  // fill the arrays 'a' and 'b' on the CPU
  for (int i=0; i<N; i++) { a[i] = -i; b[i] = i * i;  }
  // copy the arrays 'a' and 'b' to the GPU
  cudaMemcpy( dev_a, a, N * sizeof(int), cudaMemcpyHostToDevice );
  cudaMemcpy( dev_b, b, N * sizeof(int), cudaMemcpyHostToDevice );
  add<<<N,1>>>( dev_a, dev_b, dev_c );
  // copy the array 'c' back from the GPU to the CPU
  cudaMemcpy( c, dev_c, N * sizeof(int), cudaMemcpyDeviceToHost );
  // free the memory allocated on the GPU
  cudaFree( dev_a ); cudaFree( dev_b ); cudaFree( dev_c );
}
```

The Kernel

```
__global__ void add( int *a, int *b, int *c ) {
  int tid = blockIdx.x;
  // handle the data at this index
  if (tid < N)
    c[tid] = a[tid] + b[tid];
  }
```

OpenACC / OpenMP 4.0

- **Aims to simplify GPU programming**
- **Compiler support**
  - Annotations!

```
#define N 10
int main( void ) {
  int a[N], b[N], c[N];
#pragma acc kernels
  for (int i = 0; i < N; ++i)
    c[i] = a[i] + b[i];
}
```

More programming models/frameworks

- **Not covered:**
  - SMM: Intel Cilk / Cilk Plus, Intel TBB, …
  - Directives: OpenHMPP, PVM, …
  - PGAS: Coarray Fortran (Fortran 2008), …
  - HPCS: IBM X10, Fortress, Chapel, …
  - Accelerator: OpenCL, C++AMP, …
- **This class will not describe any model in more detail!**
  - There are too many and they will change quickly (only MPI made it >15 yrs)
- **No consensus, but fundamental questions remain:**
  - Data movement
  - Synchronization
  - Memory Models
  - Algorithmics
  - Foundations

Goals of this lecture

- **Motivate you!**
- **What is parallel computing?**
  - And why do we need it?
- **What is high-performance computing?**
  - What's a Supercomputer and why do we care?
- **Basic overview of**
  - Programming models

    Some examples
  - Architectures

    Some case-studies
- **Provide context for coming lectures**

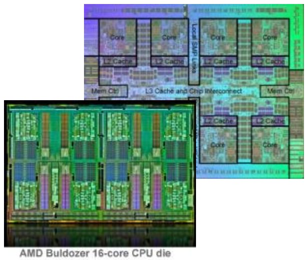
Architecture Developments

<1999

distributed memory machines communicating through messages

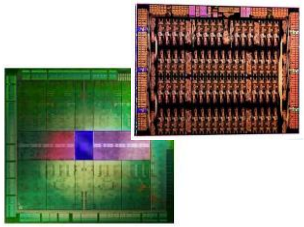
’00-’05

large cache-coherent multicore machines communicating through coherent memory access and messages



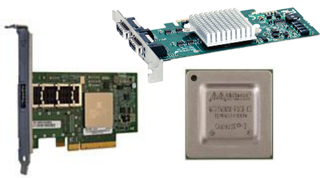
’06-’12

large cache-coherent multicore machines communicating through coherent memory access and remote direct memory access



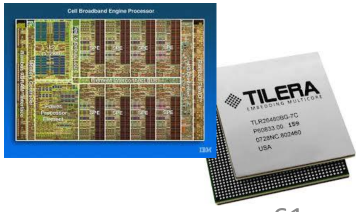
’13-’20

coherent and non-coherent manycore accelerators and multicores communicating through memory access and remote direct memory access



>2020

largely non-coherent accelerators and multicores communicating through remote direct memory access



Sources: various vendors

Computer Architecture vs. Physics

- **Physics (technological constraints)**
  - Cost of data movement
  - Capacity of DRAM cells
  - Clock frequencies (constrained by end of Dennard scaling)
  - Speed of Light
  - Melting point of silicon

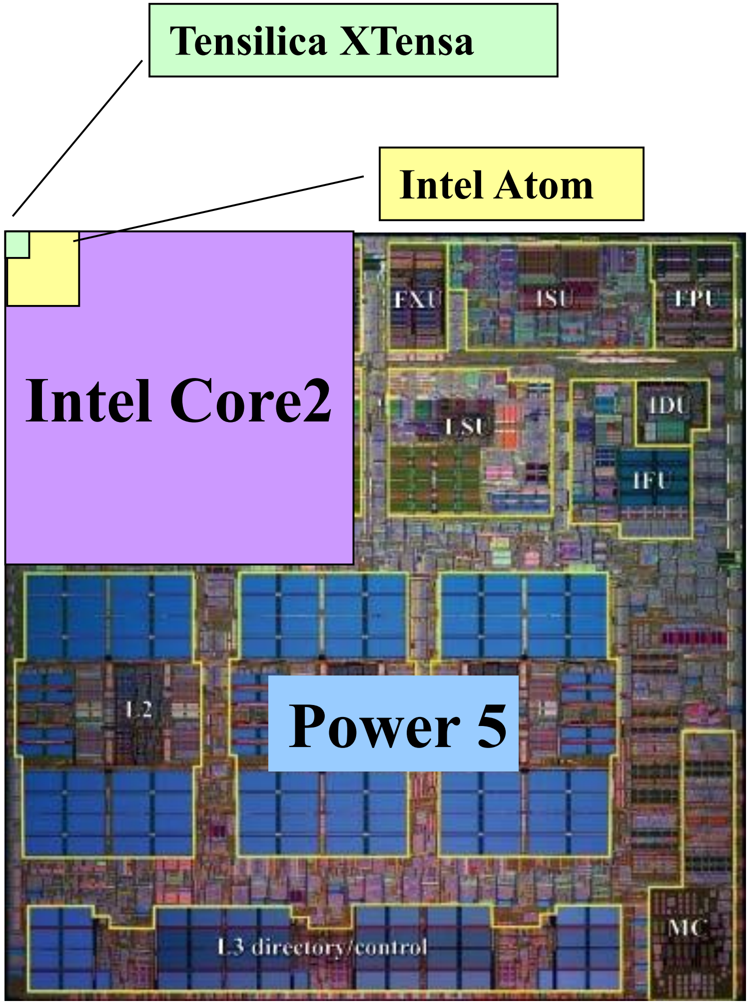
- **Computer Architecture (design of the machine)**
  - Power management
  - ISA / Multithreading
  - SIMD widths

"Computer architecture, like other architecture, is the art of determining the needs of the user of a structure and then designing to meet those needs as effectively as possible within economic and technological constraints." – *Fred Brooks (IBM, 1962)*

Have converted many former "power" problems into "cost" problems

Credit: John Shalf (LBNL)

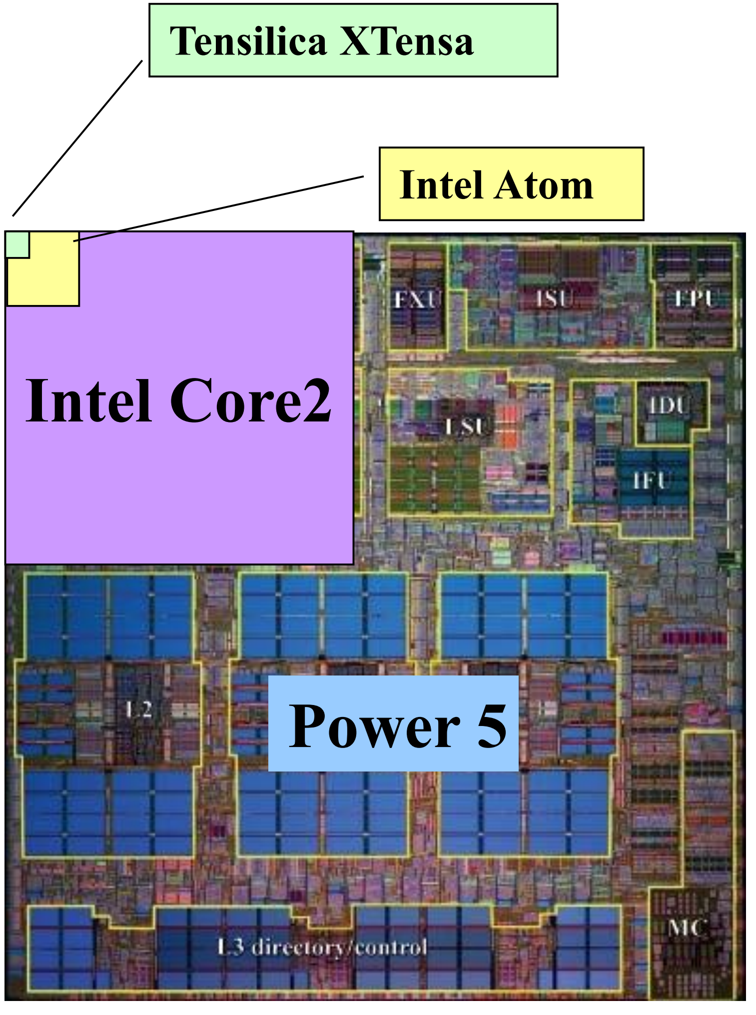
Low-Power Design Principles (2005)



- Cubic power improvement with lower clock rate due to V^2F
- Slower clock rates enable use of simpler cores
- Simpler cores use less area (lower leakage) and reduce cost
- Tailor design to application to **REDUCE WASTE**

Credit: John Shalf (LBNL)

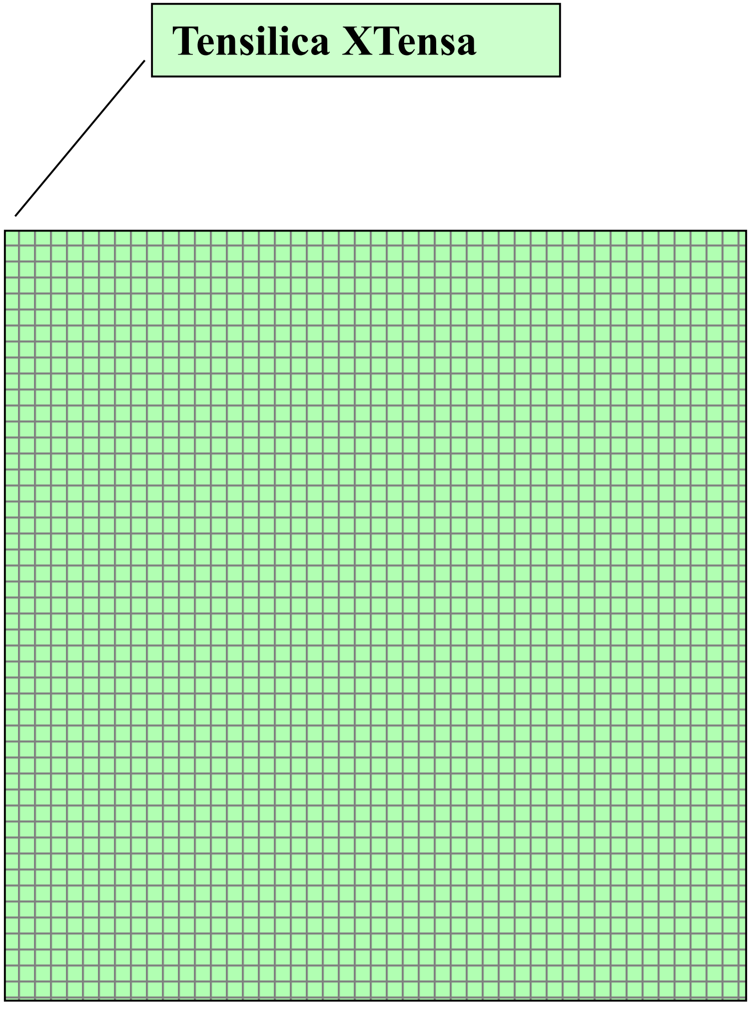
Low-Power Design Principles (2005)



- **Power5 (server)**
  - 120W@1900MHz
  - Baseline
- **Intel Core2 sc (laptop) :**
  - 15W@1000MHz
  - *4x more FLOPs/watt than baseline*
- **Intel Atom (handhelds)**
  - 0.625W@800MHz
  - 80x more
- **GPU Core or XTensa/Embedded**
  - 0.09W@600MHz
  - 400x more *(80x-120x sustained)*

Credit: John Shalf (LBNL)

Low-Power Design Principles (2005)



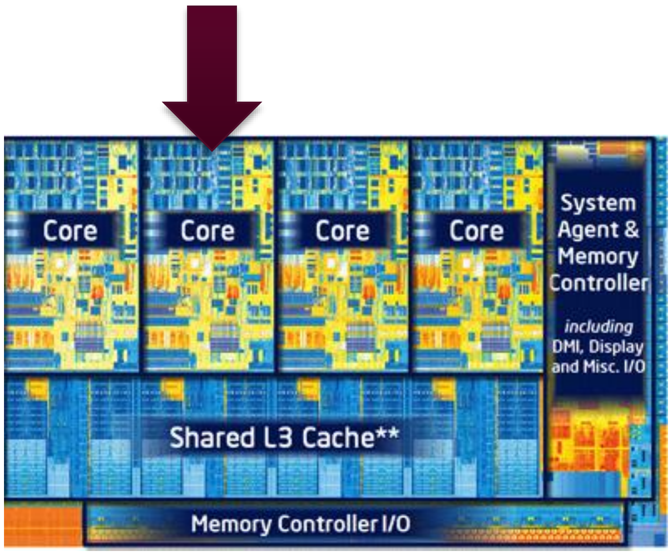
- **Power5 (server)**
  - 120W@1900MHz
  - Baseline
- **Intel Core2 sc (laptop) :**
  - 15W@1000MHz
  - *4x more FLOPs/watt than baseline*
- **Intel Atom (handhelds)**
  - 0.625W@800MHz
  - 80x more
- **GPU Core or XTensa/Embedded**
  - 0.09W@600MHz
  - 400x more *(80x-120x sustained)*

Even if each simple core is 1/4th as computationally efficient as complex core, you can fit hundreds of them on a single chip and still be 100x more power efficient.

Credit: John Shalf (LBNL)

Heterogeneous Future (LOCs and TOCs)

Big cores (very few)



Tiny core

Lots of them!

0.23mm

0.2 mm

Latency Optimized Core (LOC)
Most energy efficient if you don't have lots of parallelism

Throughput Optimized Core (TOC)
Most energy efficient if you DO have a lot of parallelism!

Credit: John Shalf (LBNL)

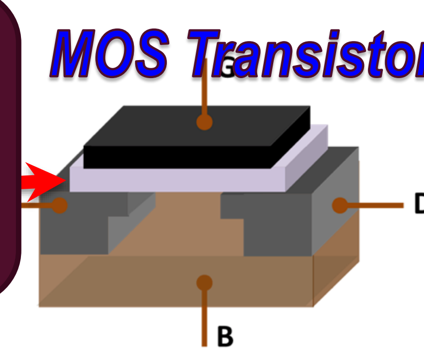
Data movement – the wires

- **Energy Efficiency of copper wire:**
  - **Power = Frequency* Length / cross-section-area**

  - Wire

Photonics could break through the bandwidth-distance limit

- **Energy**
  - Pow
  - Capacitance ~= Area of Transistor
  - Transistor efficiency improves as you shrink it

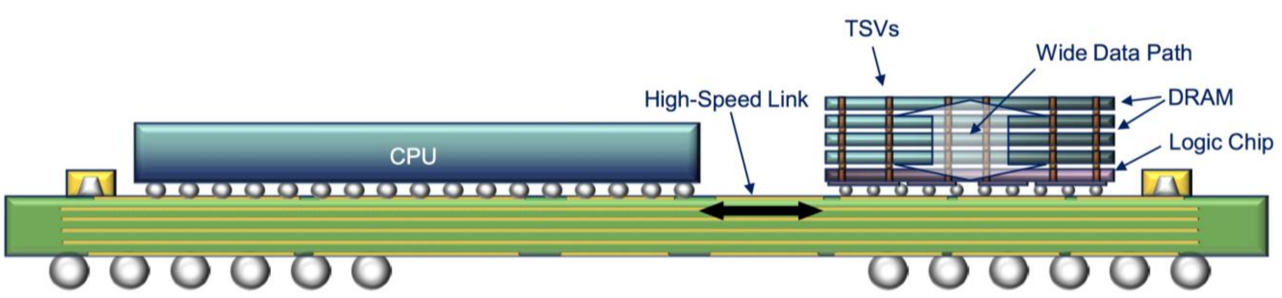


- ***Net result is that moving data on wires is starting to cost more energy than computing on said data (interest in Silicon Photonics)***

Credit: John Shalf (LBNL)

Pin Limits

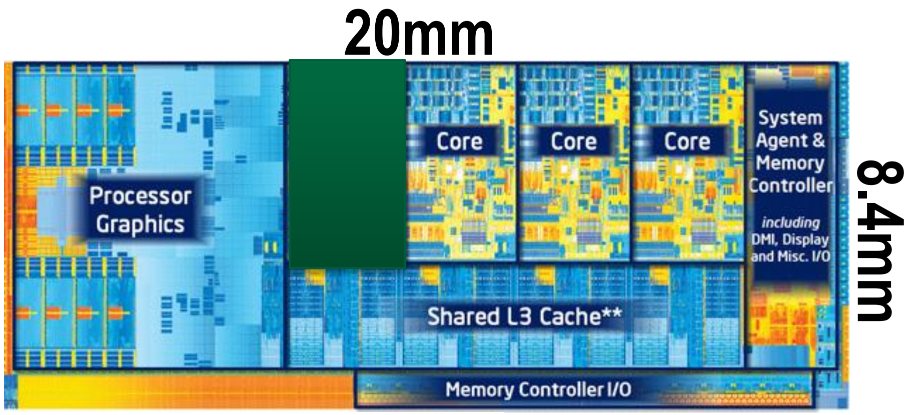
- **Moore's law doesn't apply to adding pins to package**
  - 30%+ per year nominal Moore's Law
  - Pins grow at ~1.5-3% per year at best
- **4000 Pins is aggressive pin package**
  - Half of those would need to be for power and ground
  - Of the remaining 2k pins, run as differential pairs
  - Beyond 15Gbps per pin power/complexity costs hurt!
  - 10Gpbs * 1k pins is ~1.2TBytes/sec
- **2.5D Integration gets boost in pin density**
  - But it's a 1 time boost (how much headroom?)
  - 4TB/sec? (maybe 8TB/s with single wire signaling?)



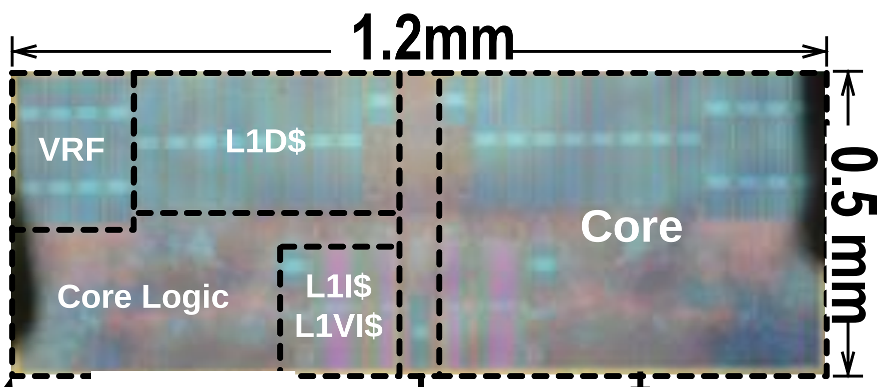
Credit: John Shalf (LBNL)

Die Photos (3 classes of cores)

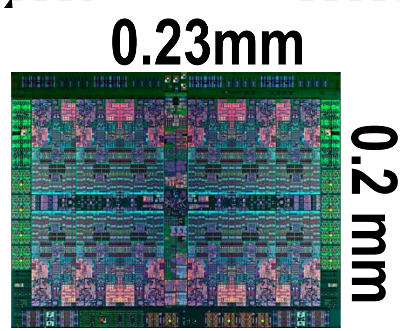
Big



Small

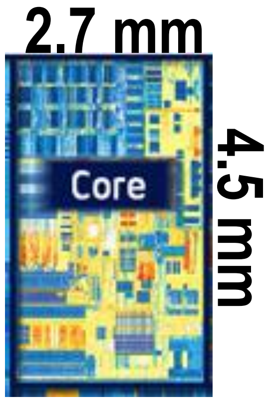


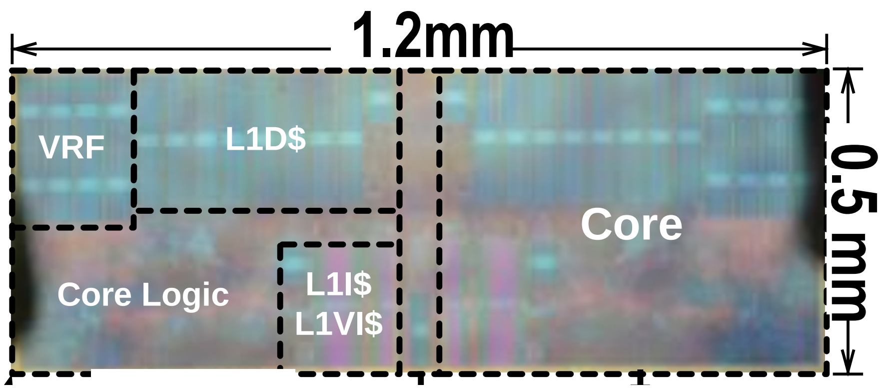
Tiny

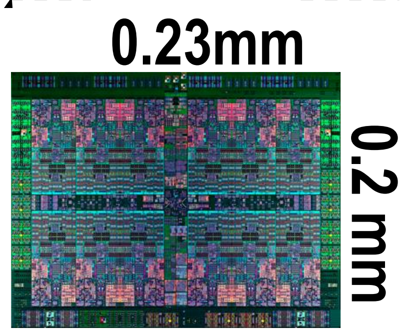


Credit: John Shalf (LBNL)

Strip down to the core







Credit: John Shalf (LBNL)

Actual Size

2.7mm

4.5 mm

0.5mm

1.2 mm

0.23mm

0.2 mm

Credit: John Shalf (LBNL)

Basic Stats

Core Energy/Area est.

2.7mm

4.5 mm

Area: 12.25 mm^2
Power: 2.5W
Clock: 2.4 GHz
E/op: 651 pj

0.5mm 1.2 mm

Area: 0.6 mm^2
Power: 0.3W (<0.2W)
Clock: 1.3 GHz
E/op: 150 (75) pj

0.23mm 0.2 mm

Area: 0.046 mm^2
Power: 0.025W
Clock: 1.0 GHz
E/op: 22 pj

Wire Energy

Assumptions for 22nm
- 100 fj/bit per mm
- 64bit operand

Energy:
- 1mm=~6pj
- 20mm=~120pj

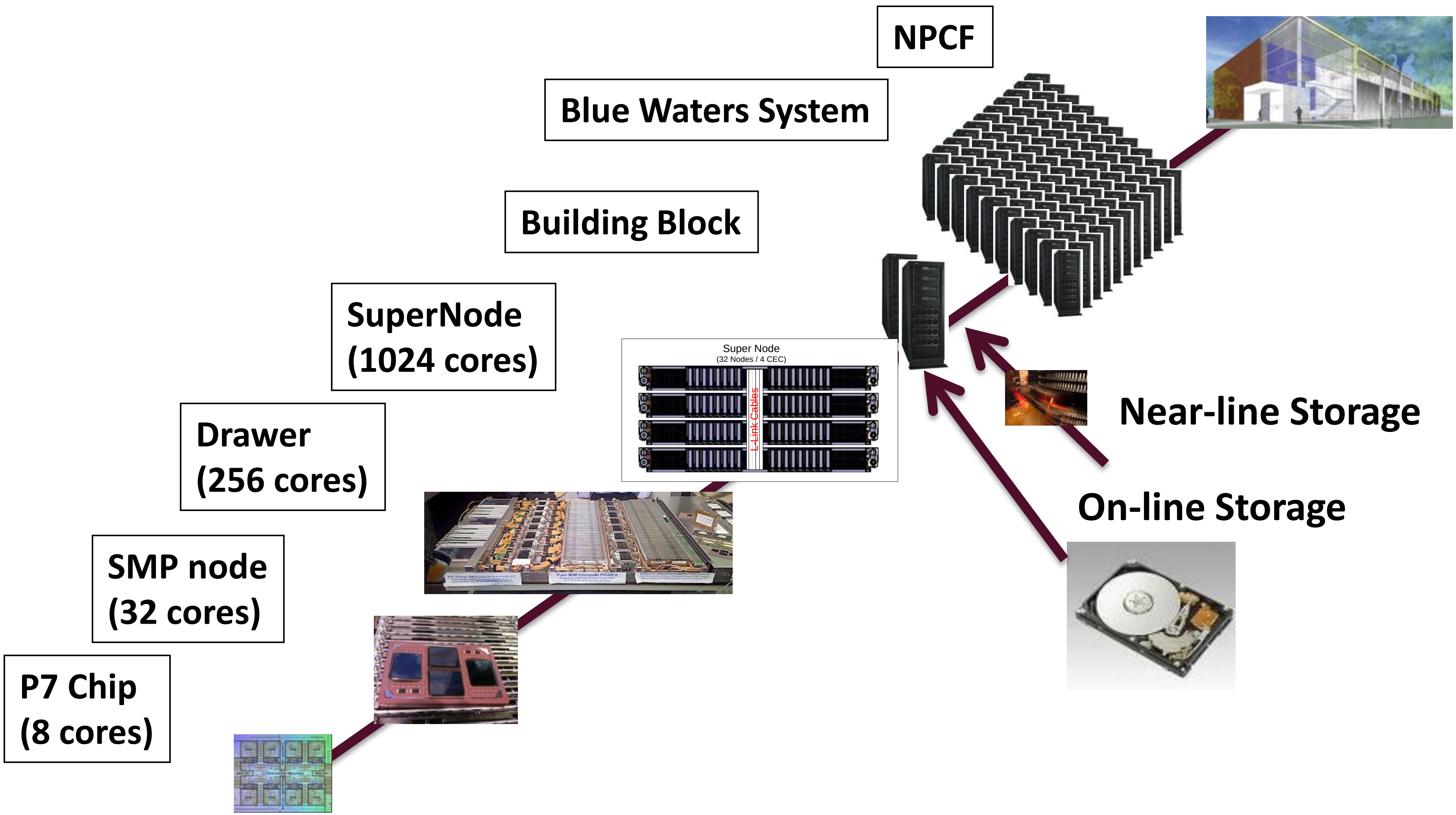
Credit: John Shalf (LBNL)

When does data movement dominate?

	Core Energy/Area est.	Data Movement Cost
intel; 2.7mm; 4.5 mm	Area: 12.25 mm^2 Power: 2.5W Clock: 2.4 GHz E/op: 651 pj	*Compute Op ==* *data movement Energy @* 108mm *Energy Ratio for 20mm* 0.2x
RISC-V; 0.5mm; 1.2 mm	Area: 0.6 mm^2 Power: 0.3W (<0.2W) Clock: 1.3 GHz E/op: 150 (75) pj	*Compute Op ==* *data movement Energy @* 12mm *Energy Ratio for 20mm* 1.6x
tensilica; 0.23mm; 0.2 mm	Area: 0.046 mm^2 Power: 0.025W Clock: 1.0 GHz E/op: 22 pj	*Compute Op ==* *data movement Energy @* 3.6mm *Energy Ratio for 20mm* 5.5x

Credit: John Shalf (LBNL)

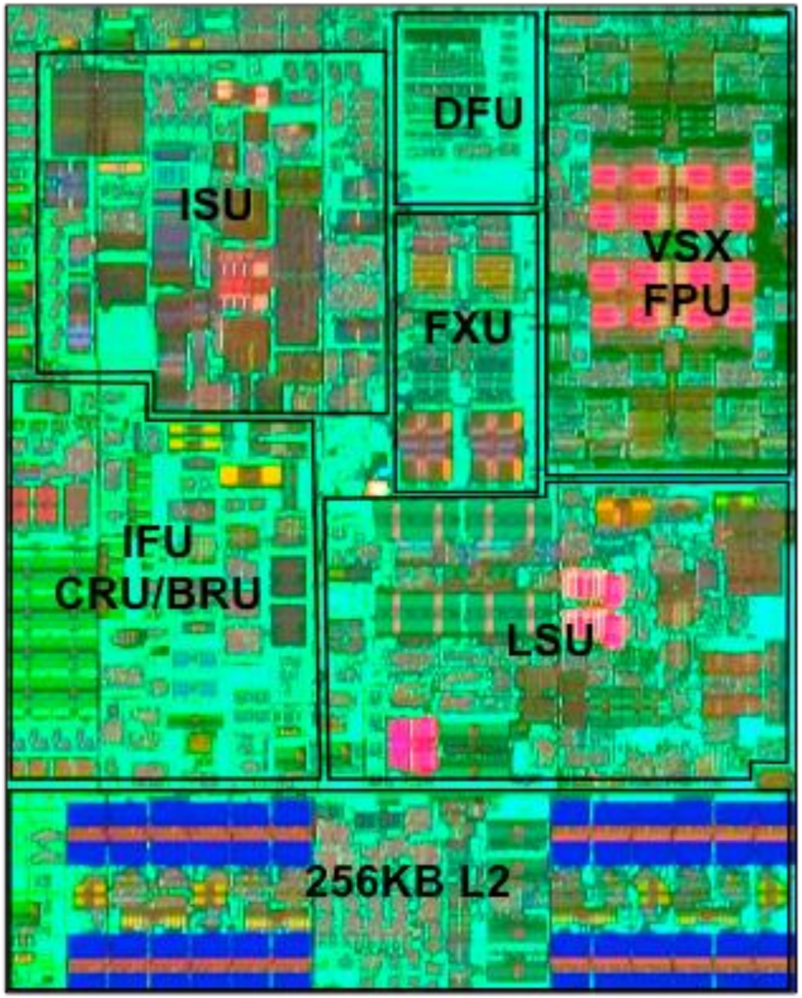
Case Study 1: IBM POWER7 IH (BW)



Source: IBM/NCSA

POWER7 Core

- Execution Units
  - 2 Fixed point units
  - 2 Load store units
  - 4 Double precision floating point
  - 1 Branch
  - 1 Condition register
  - 1 Vector unit
  - 1 Decimal floating point unit
  - 6 wide dispatch
- Recovery Function Distributed
- 1,2,4 Way SMT Support
- Out of Order Execution
- 32KB I-Cache
- 32KB D-Cache
- 256KB L2
  - Tightly coupled to core

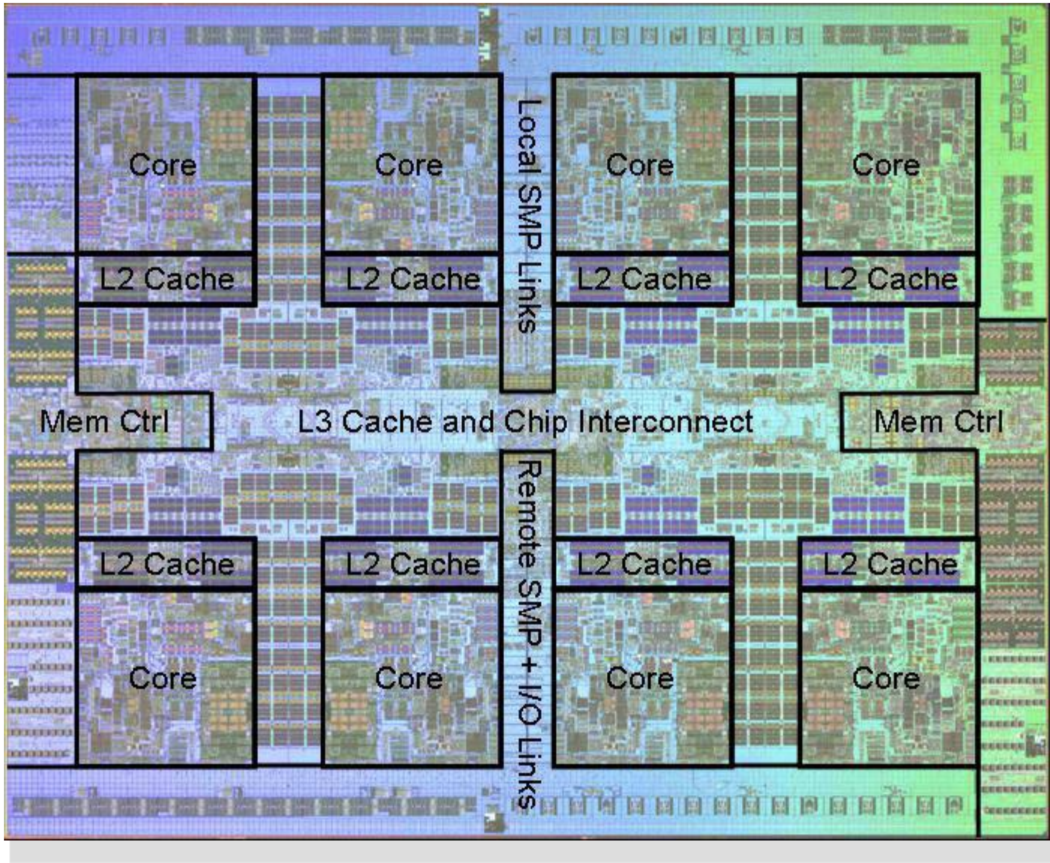


Source: IBM/NCSA

POWER7 Chip (8 cores)

- **Base Technology**
  - 45 nm, 576 mm^2
  - 1.2 B transistors
- **Chip**
  - 8 cores
  - 4 FMAs/cycle/core
  - 32 MB L3 (private/shared)
  - Dual DDR3 memory
    128 GiB/s peak bandwidth
    (1/2 byte/flop)
  - Clock range of 3.5 – 4 GHz

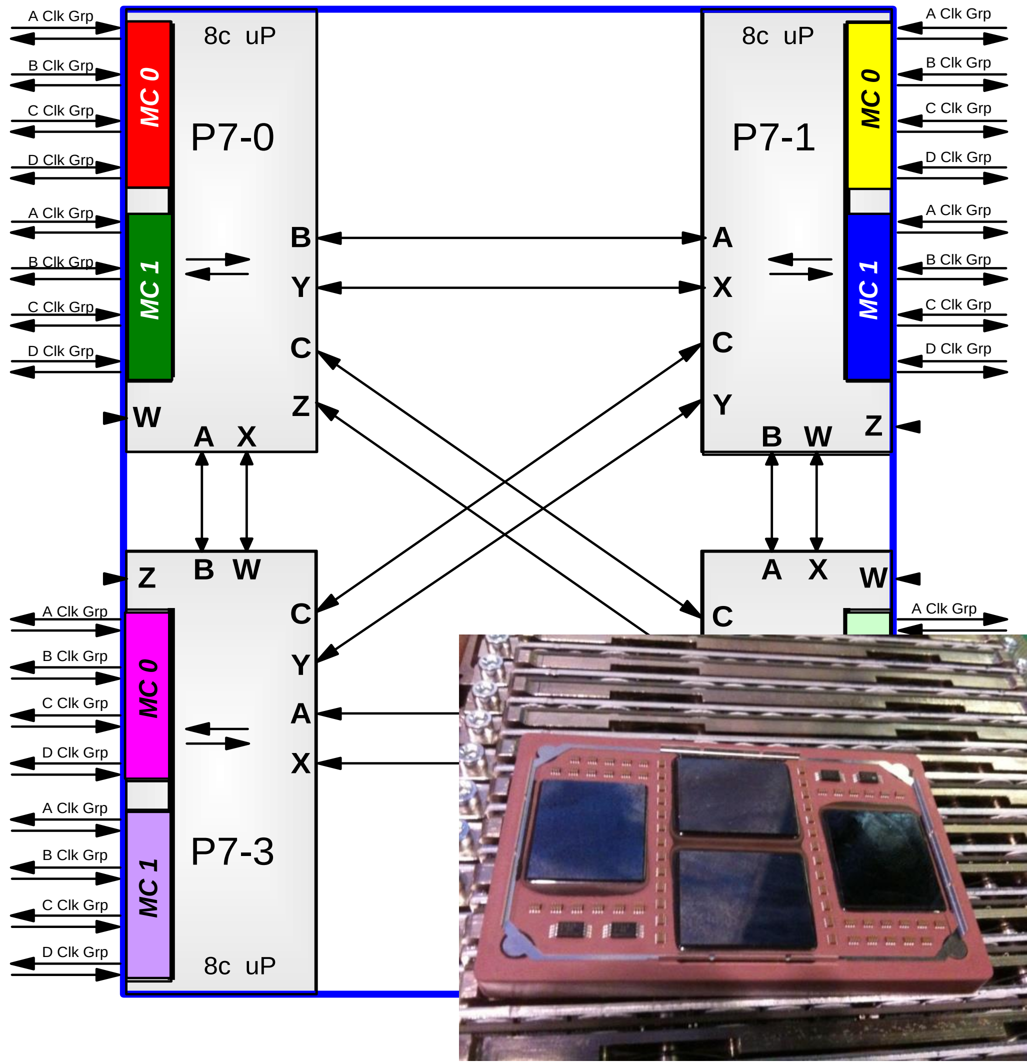
Quad-chip MCM



Source: IBM/NCSA

Quad Chip Module (4 chips)

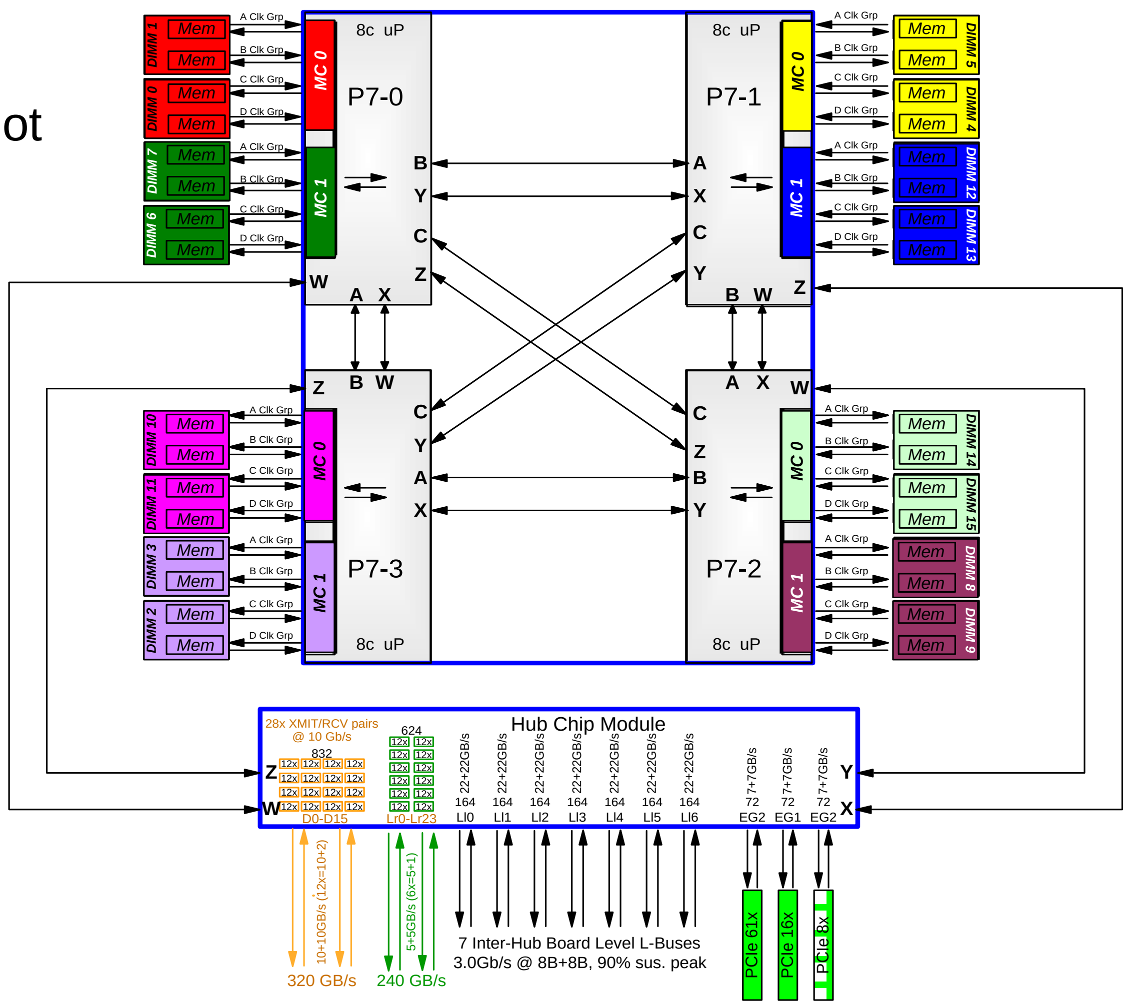
- **32 cores**
  - 32 cores*8 F/core*4 GHz = 1 TF
- **4 threads per core (max)**
  - 128 threads per package
- **4x32 MiB L3 cache**
  - 512 GB/s RAM BW (0.5 B/F)
- **800 W (0.8 W/F)**



Source: IBM/NCSA

Adding a Network Interface (Hub)

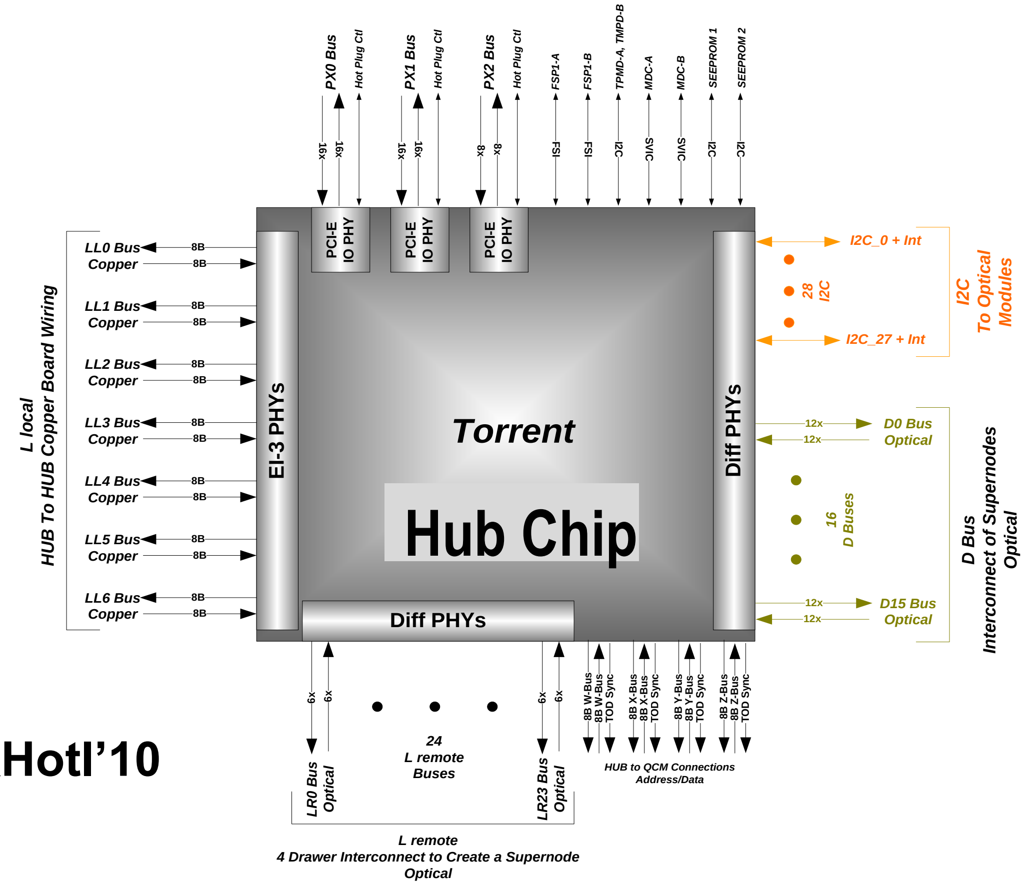
- **Connects QCM to PCI-e**
  - Two 16x and one 8x PCI-e slot
- **Connects 8 QCM's via low latency, high bandwidth, copper fabric.**
  - Provides a message passing mechanism with very high bandwidth
  - Provides the lowest possible latency between 8 QCM's



Source: IBM/NCSA

1.1 TB/s POWER7 IH HUB

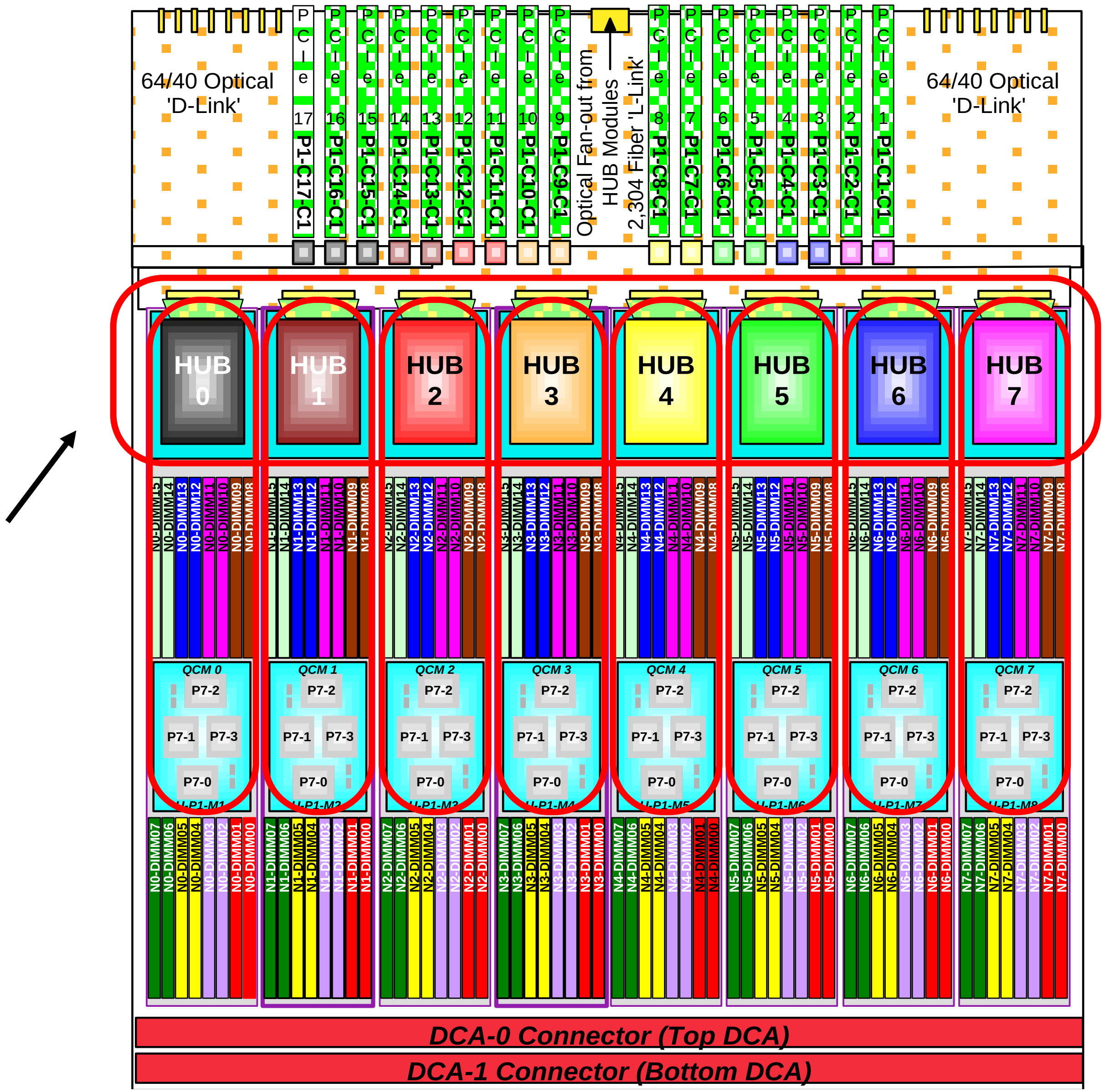
- 192 GB/s Host Connection
- 336 GB/s to 7 other local nodes
- 240 GB/s to local-remote nodes
- 320 GB/s to remote nodes
- 40 GB/s to general purpose I/O
- cf. "The PERCS interconnect" @HotI'10



Source: IBM/NCSA

P7 IH Drawer

- 8 nodes
    - 32 chips
    - 256 cores

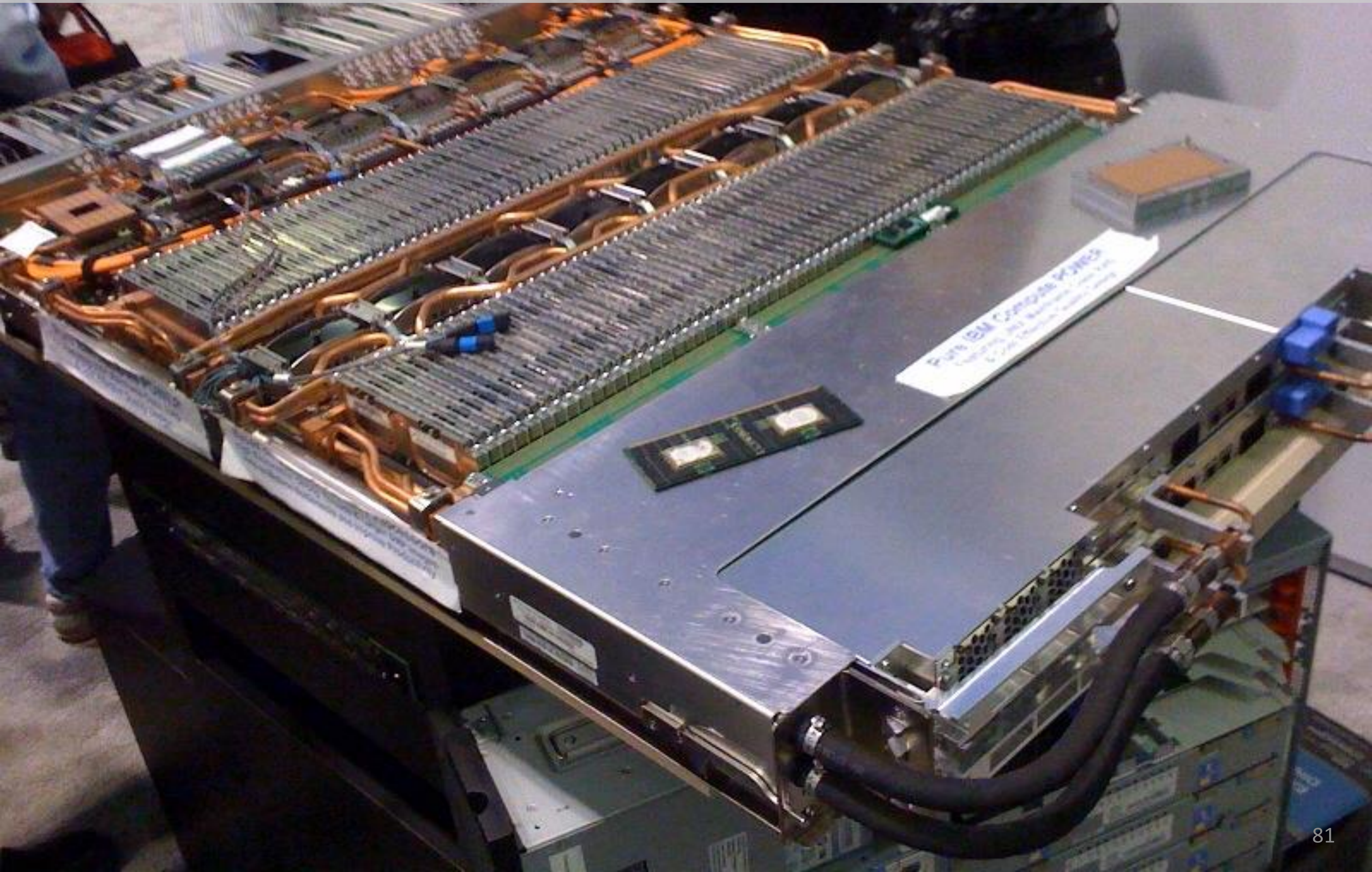


First Level Interconnect

- L-Local
- HUB to HUB Copper Wiring
- 256 Cores

Source: IBM/NCSA

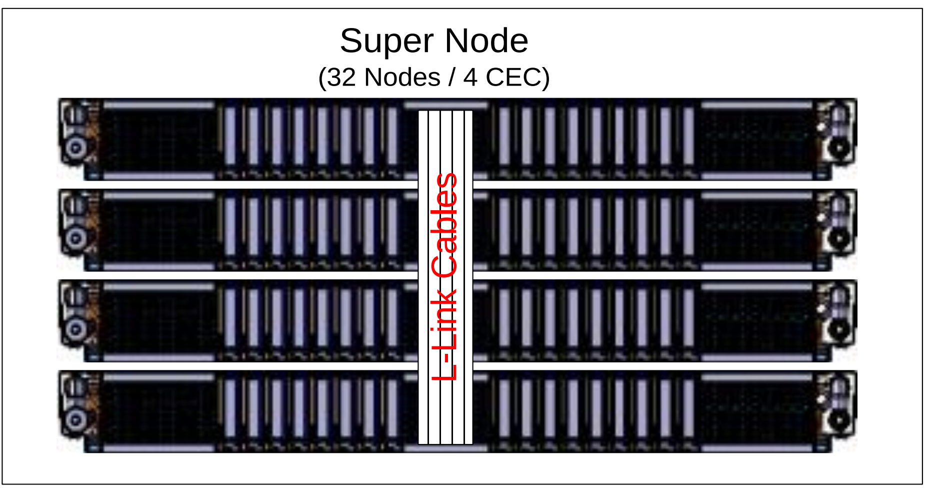
POWER7 IH Drawer @ SC09

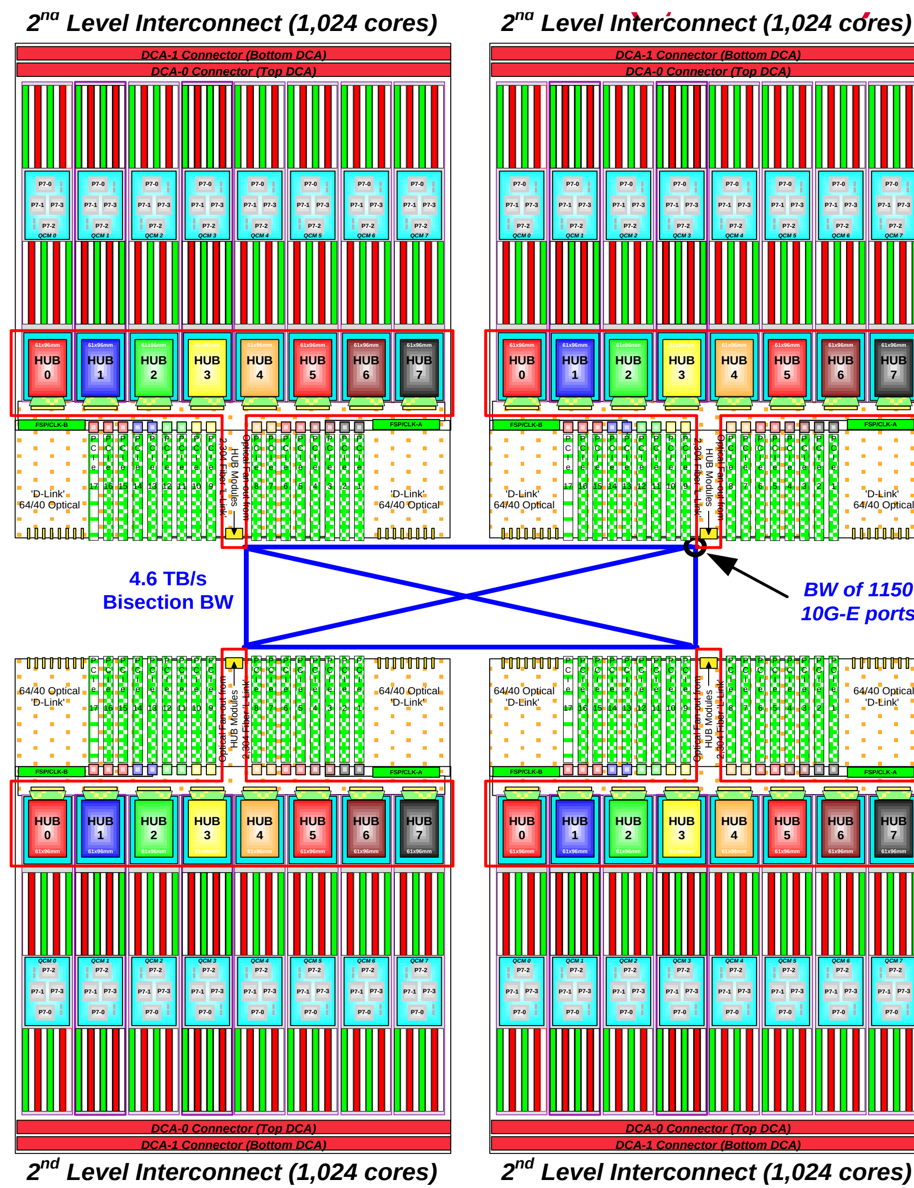


P7 IH Supernode

<u>**Second Level Interconnect**</u>

- Optical 'L-Remote' Links from HUB
- 4 drawers
- 1,024 Cores

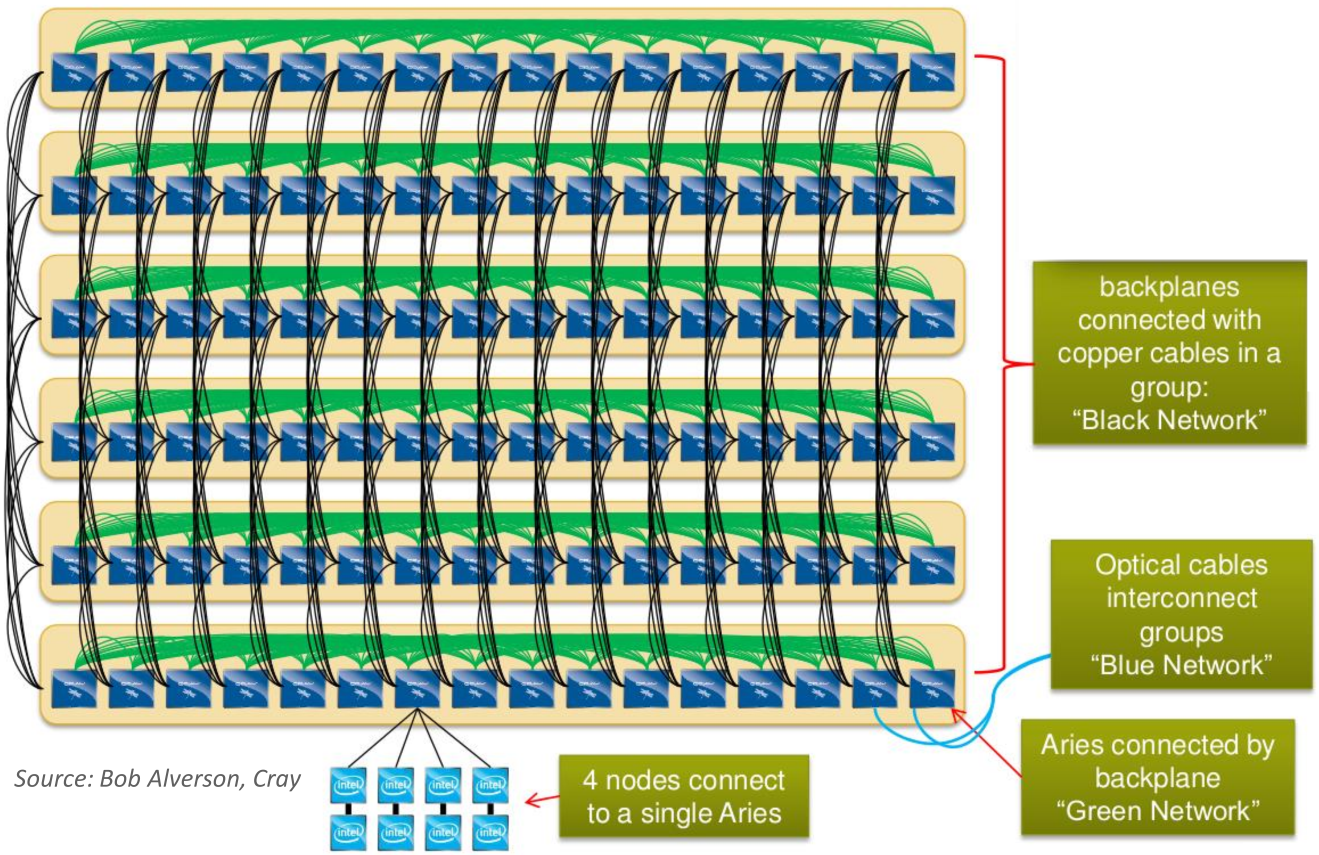




Source: IBM/NCSA

Case Study 2: Cray Cascade (XC30)

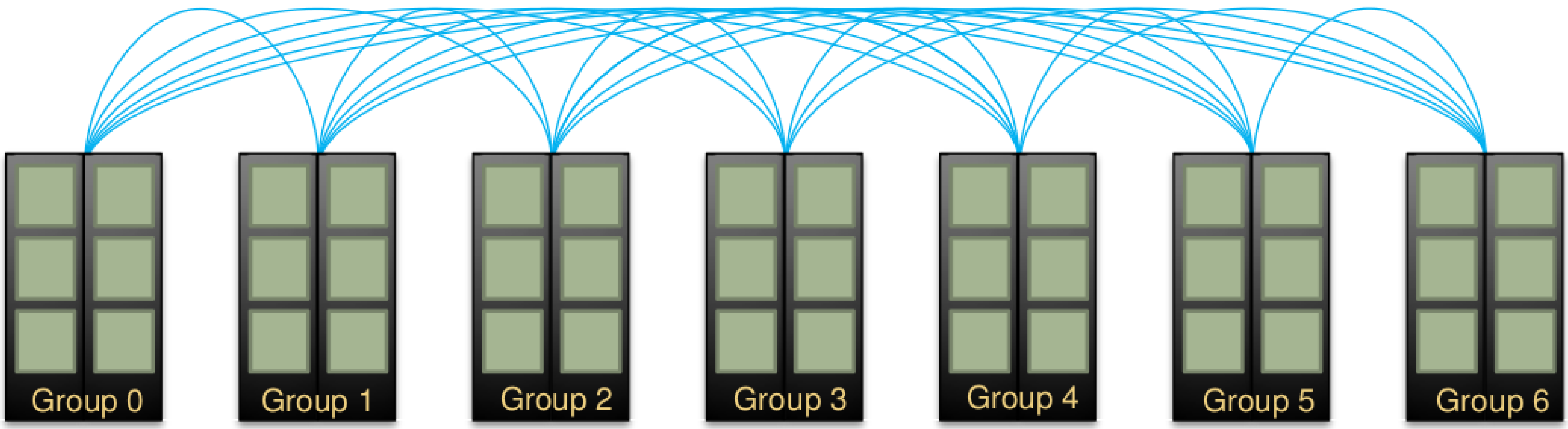
- **Biggest current installation at CSCS! ☺**
  - >2k nodes
- **Standard Intel x86 Sandy Bridge Server-class CPUs**



Source: Bob Alverson, Cray

Cray Cascade Network Topology

- All-to-all connection among groups (“blue network”)



Source: Bob Alverson, Cray

- What does that remind you of?

Goals of this lecture

- **Motivate you!**
- **What is parallel computing?**
  - And why do we need it?
- **What is high-performance computing?**
  - What's a Supercomputer and why do we care?
- **Basic overview of**
  - Programming models
    Some examples
  - Architectures
    Some case-studies
- **Provide context for coming lectures**

DPHPC Lecture

- **You will most likely not have access to the largest machines**
  - But our desktop/laptop will be a “large machine” soon
  - HPC is often seen as “Formula 1” of computing (architecture experiments)
- **DPHPC will teach you concepts!**
  - Enable to understand and use all parallel architectures
  - From a quad-core mobile phone to the largest machine on the planet!

    MCAPI vs. MPI – same concepts, different syntax
  - No particular language (but you should pick/learn one for your project!)

    Parallelism is the future:

Related classes in the SE focus

- **263-2910-00L Program Analysis**
  http://www.srl.inf.ethz.ch/pa.php
  Spring 2017
  Lecturer: Prof. M. Vechev

- **263-2300-00L How to Write Fast Numerical Code**
  http://www.inf.ethz.ch/personal/markusp/teaching/263-2300-ETH-spring16/course.html
  Spring 2017
  Lecturer: Prof. M. Pueschel

- **This list is not exhaustive!**

DPHPC Overview