Design of Parallel and High-Performance Computing

Fall 2013

Lecture: Introduction

Instructor: Torsten Hoefler & Markus Püschel

TA: Timo Schneider

ETH

Eidgenössische Technische Hochschule Zürich

Swiss Federal Institute of Technology Zurich

Goals of this lecture

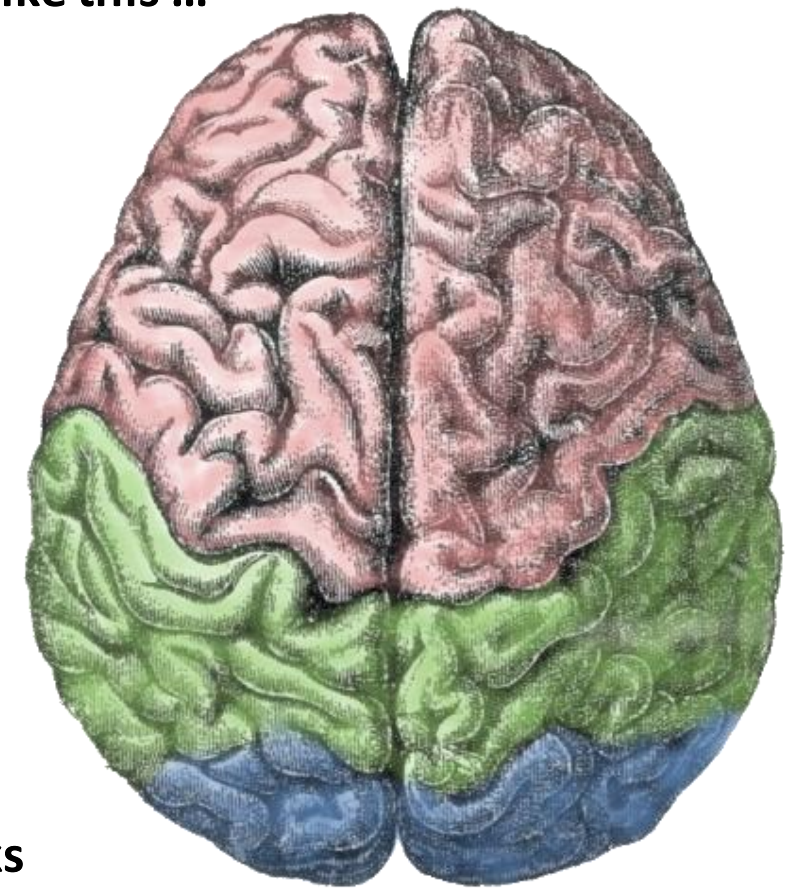
- **Motivate you!**
- **What is parallel computing?**
  - And why do we need it?
- **What is high-performance computing?**
  - What's a Supercomputer and why do we care?
- **Basic overview of**
  - Programming models

    Some examples
  - Architectures

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

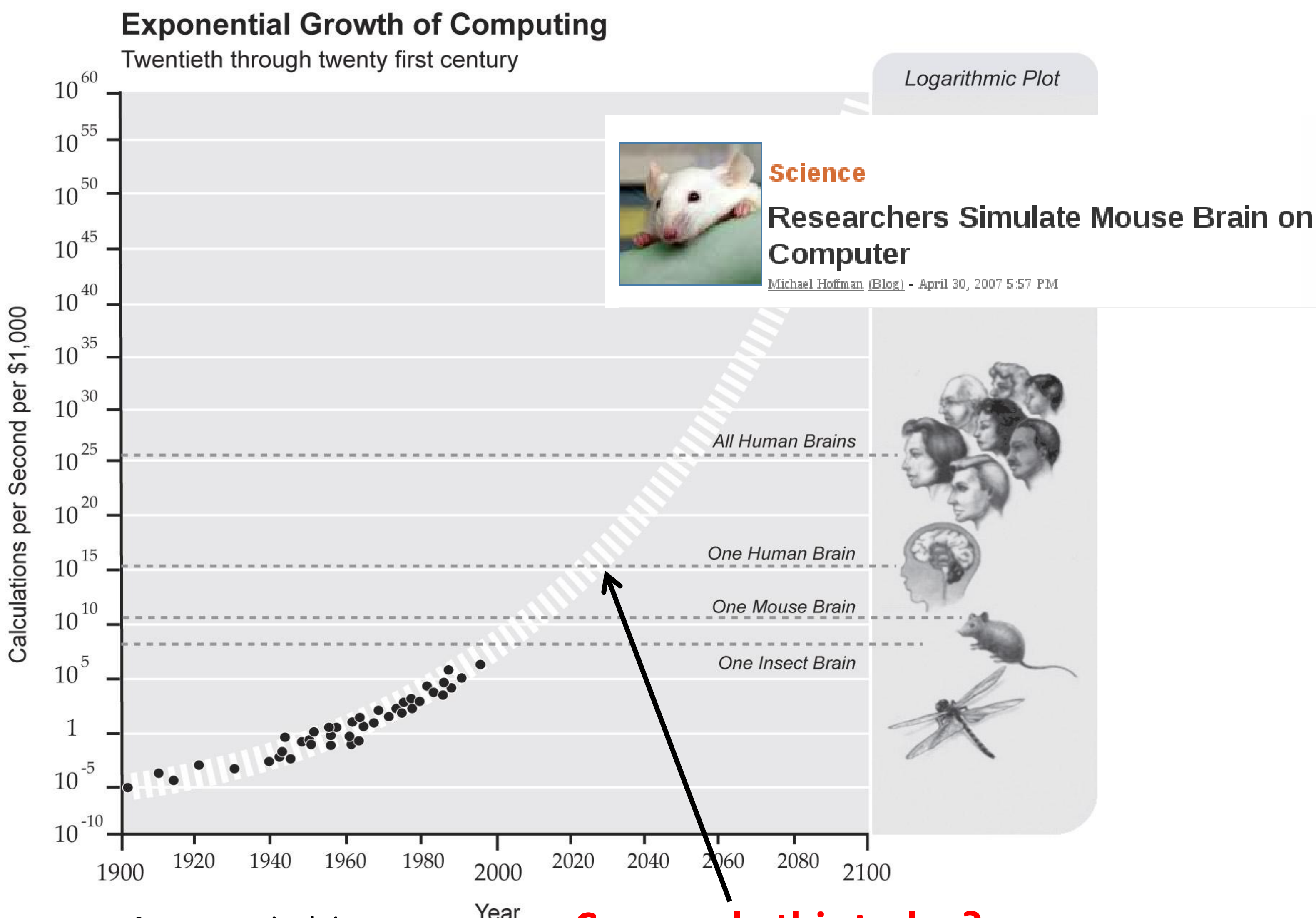
Let us assume …

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



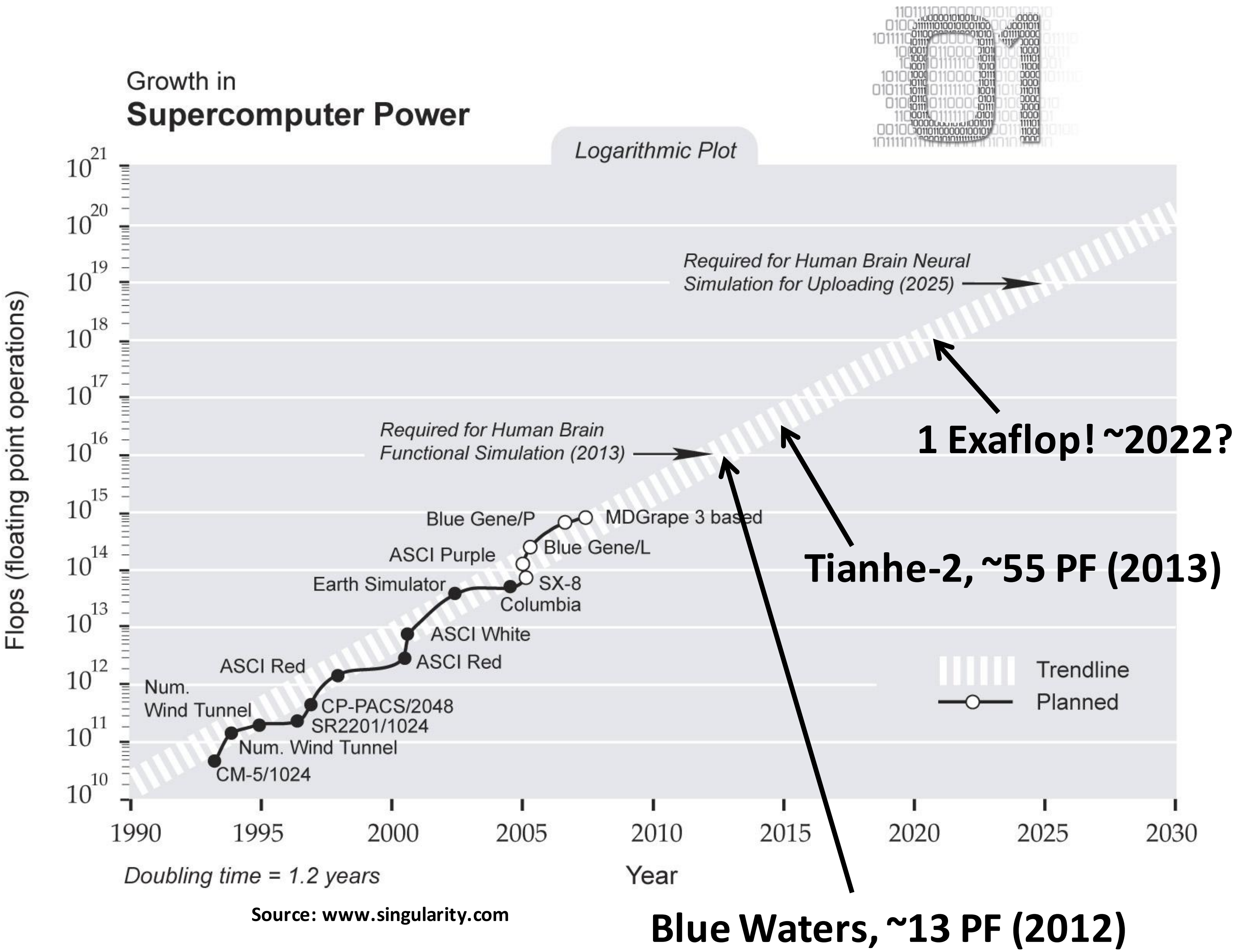
Source: wikipedia

- **… 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

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

Share This Article

436 Like | 123 Tweet | 228 | 108 +1 | 24 Share

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 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 small 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 2013)

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National University of Defense Technology China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3120000	33862.7	54902.4	17808
2	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560640	17590.0	27112.5	8209
3	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1572864	17173.2	20132.7	7890
4	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705024	10510.0	11280.4	12660
5	DOE/SC/Argonne National Laboratory United States	**Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786432	8586.6	10066.3	3945
6	Texas Advanced Computing Center/Univ. of Texas United States	**Stampede** - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P Dell	462462	5168.1	8520.1	4510
7	Forschungszentrum Juelich (FZJ) Germany					
8	DOE/NNSA/LLNL United States	**Vulcan** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	393216	4293.3	5033.2	1972
9	Leibniz Rechenzentrum Germany	**SuperMUC** - iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR IBM	147456	2897.0	3185.1	3423

42	Swiss Scientific Computing Center (CSCS) Switzerland	**Piz Daint** - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect Cray Inc.	35840	626.9	745.5

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.

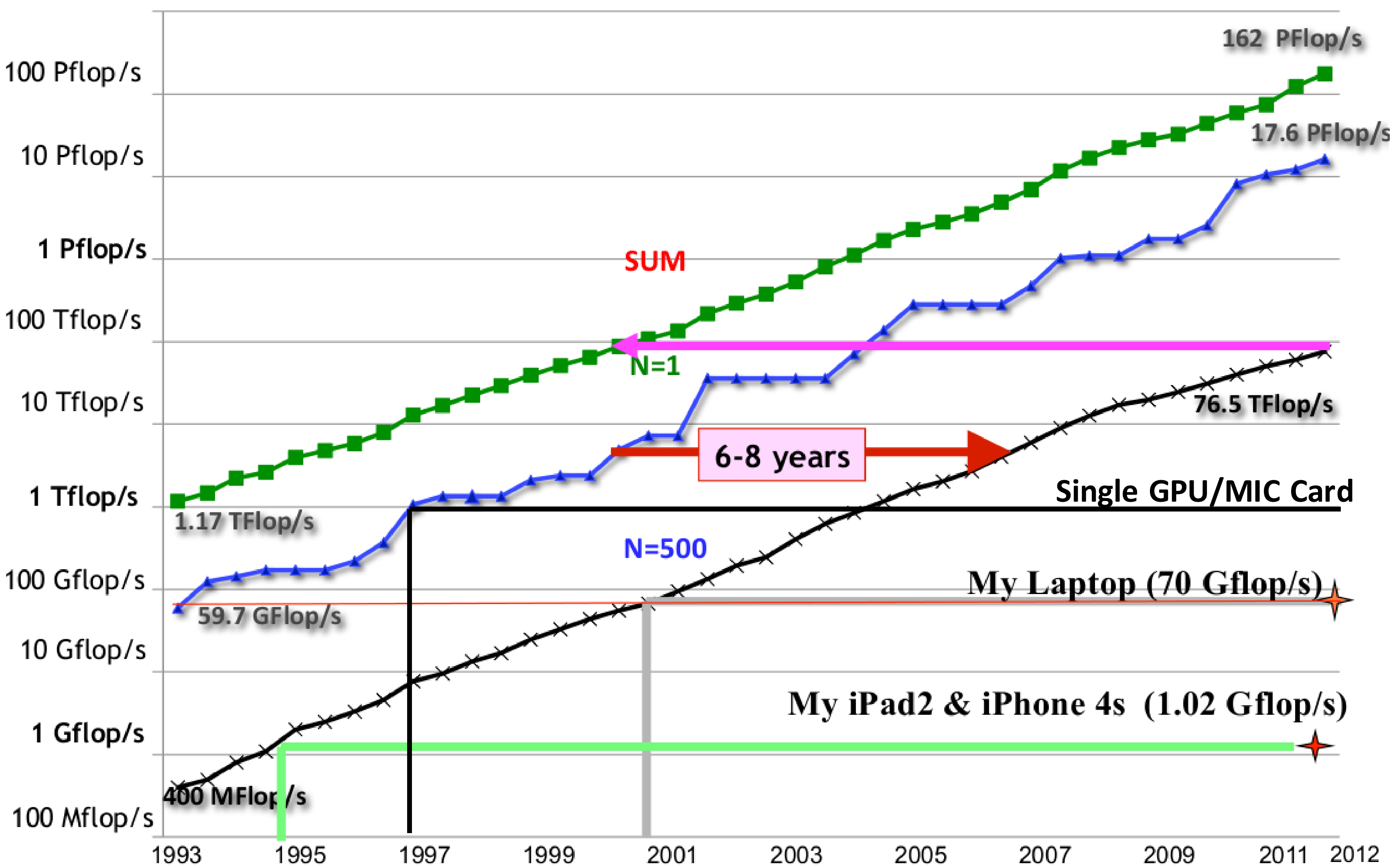
Blue Waters in 2009

Imagine you’re designing a $500 M supercomputer, and all you have is:

This is why you need to understand performance expectations well!

Blue Waters in 2012

History and Trends



Source: Jack Dongarra

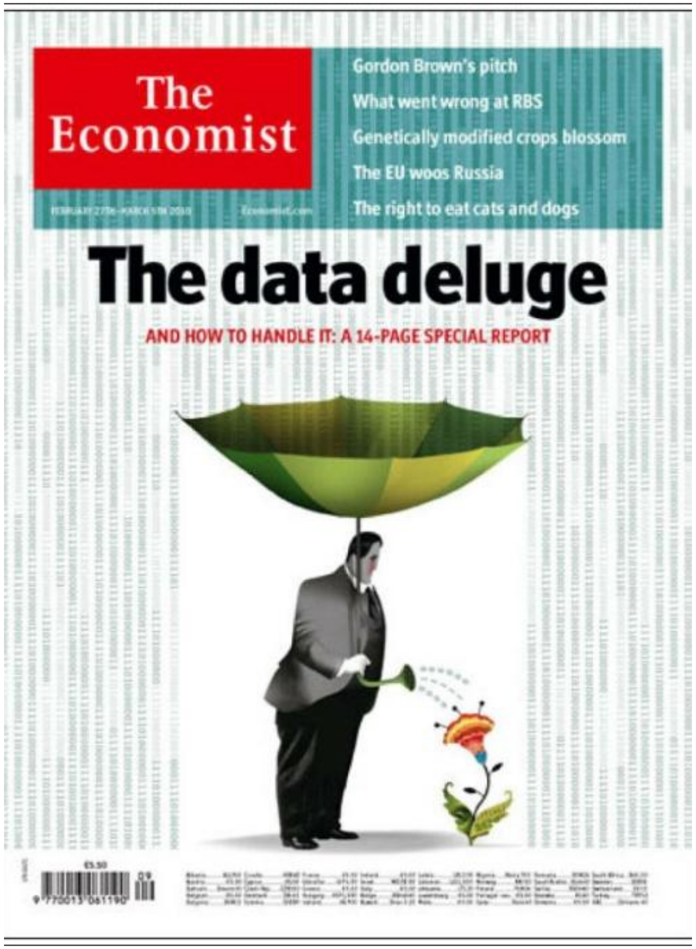
High-Performance Computing grows quickly

- **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."

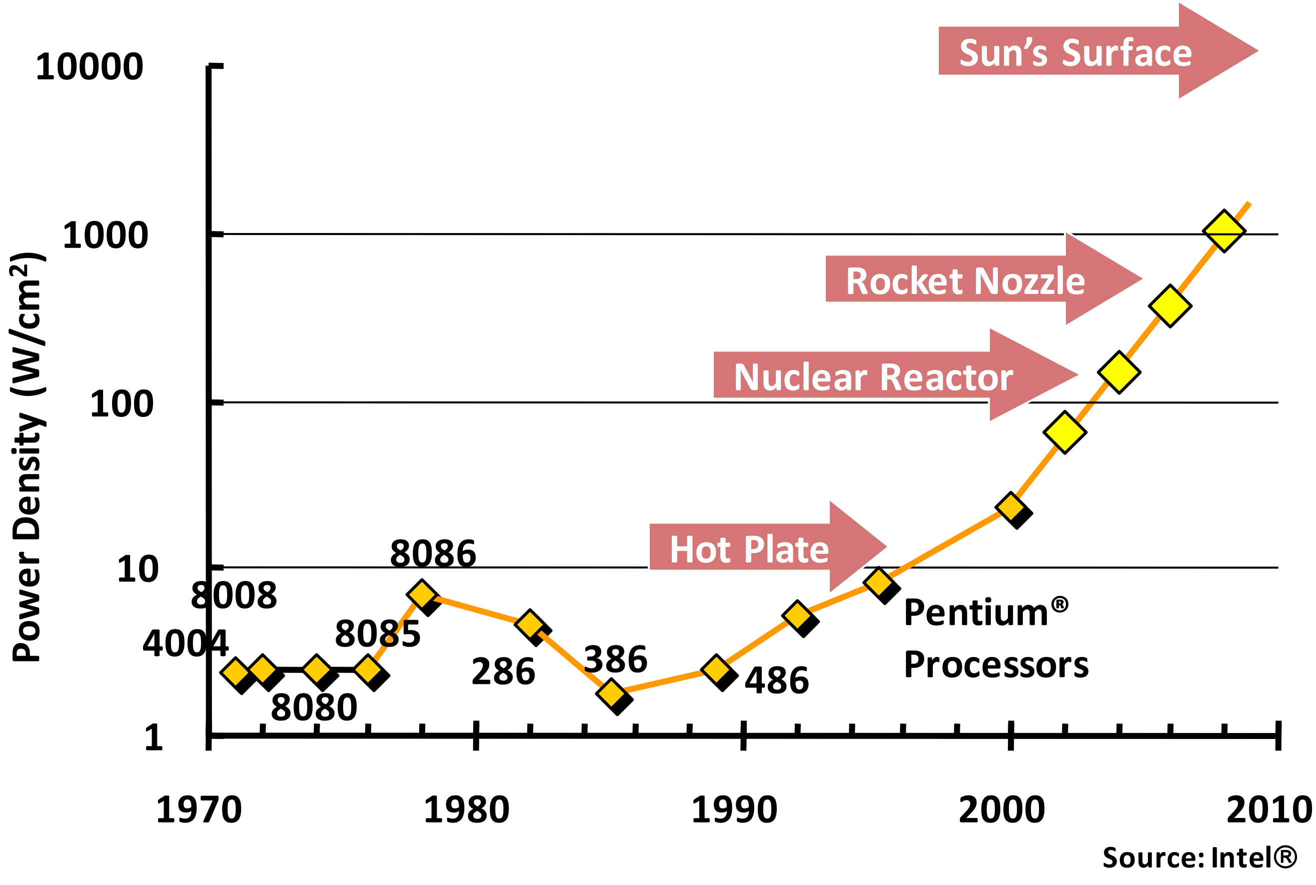
"The non-HPC portion of the server market was actually down 20.5 per cent, to $34.6bn"



Source: The Economist

How to increase the compute power?

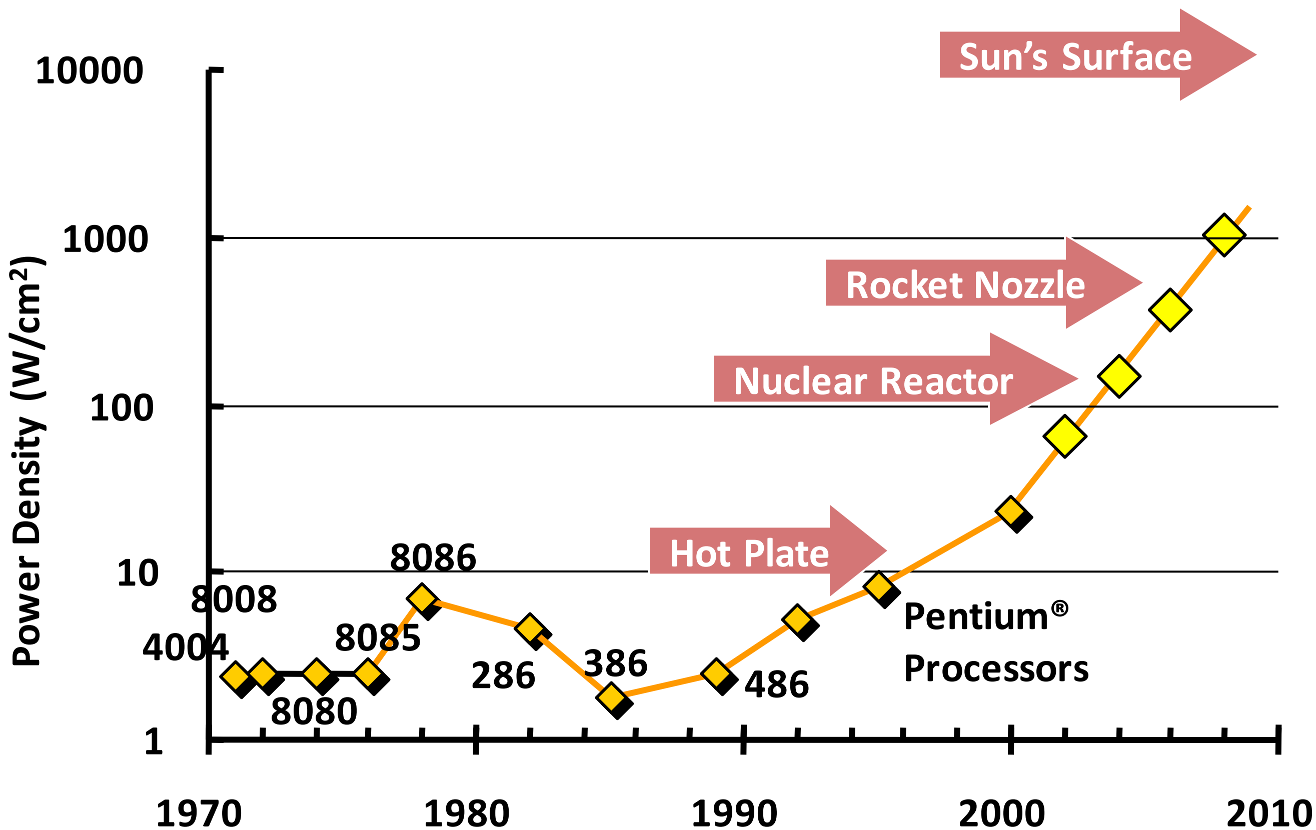
Clock Speed:



How to increase the compute power?

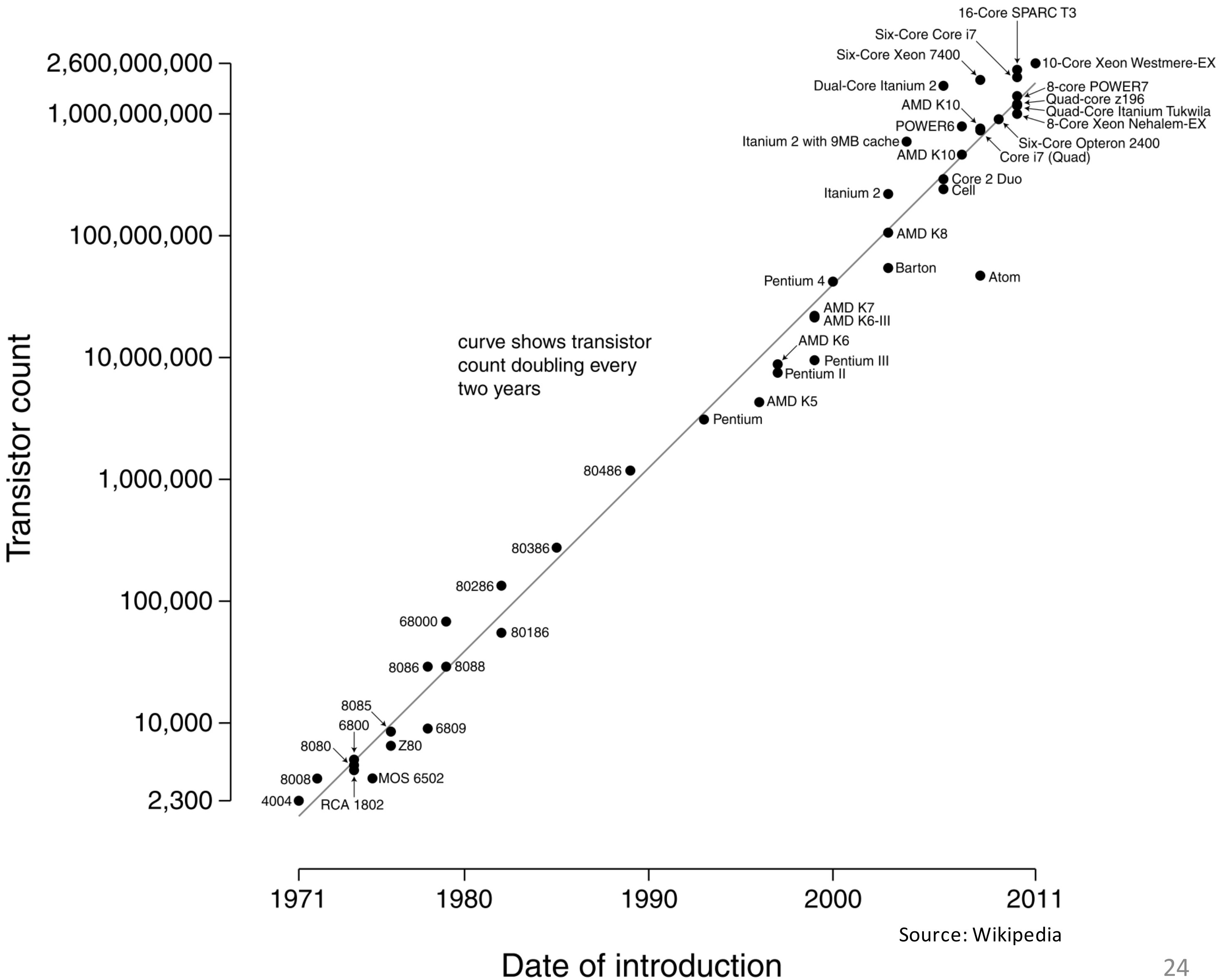
Not an option anymore!

~~Clock Speed:~~



Source: Intel®

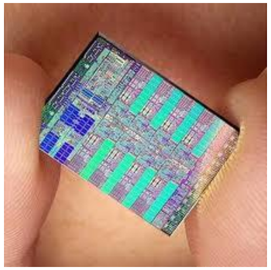
Microprocessor Transistor Counts 1971-2011 & Moore's Law



Source: Wikipedia

So how to invest the transistors?

- **Architectural 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

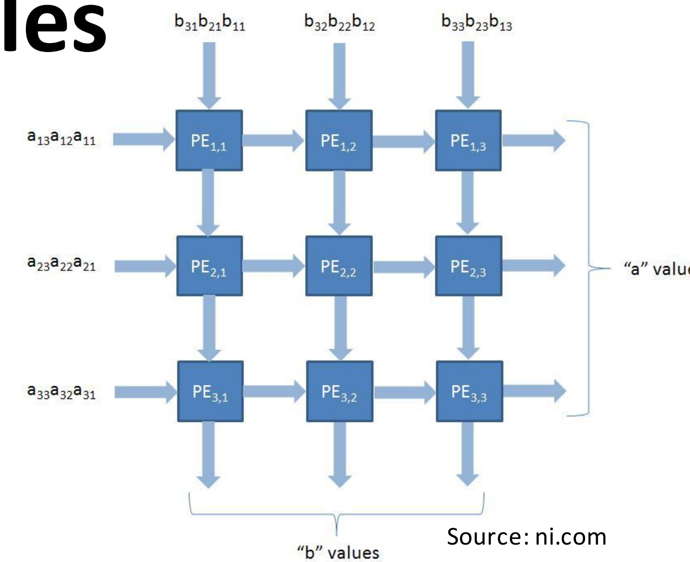
▪ Instruction-level parallelism
  ▪ Pipelining
  ▪ VLIW
  ▪ Superscalar
	▪ Compiler
	  ▪ (inline assembly)
	  ▪ Hardware scheduling

▪ 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

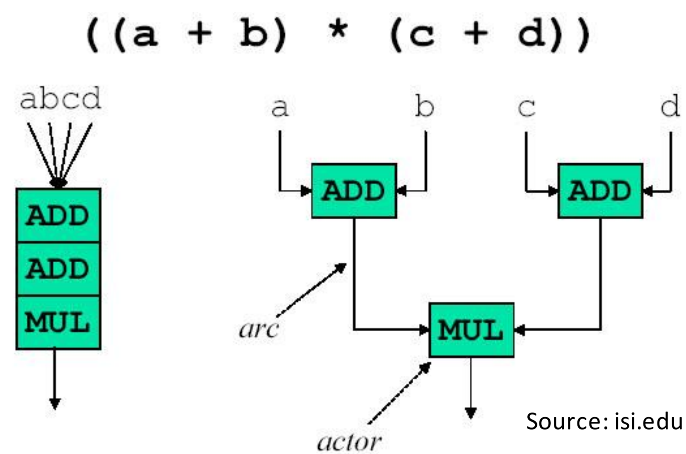
- **Systolic Array**
  - Data-stream driven (data counters)
  - Multiple streams for parallelism
  - Specialized for applications (reconfigurable)



Source: ni.com

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

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



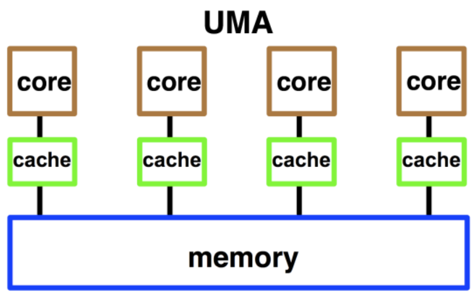
Source: isi.edu

Von Neumann Architecture

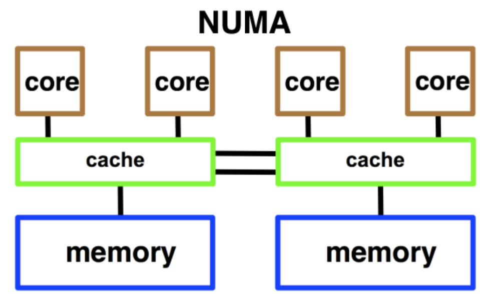
- **Program counter → Inherently serial!**
  Retrospectively define parallelism in instructions and data

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

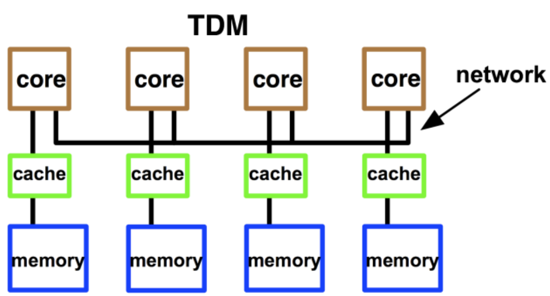
Parallel Architectures 101



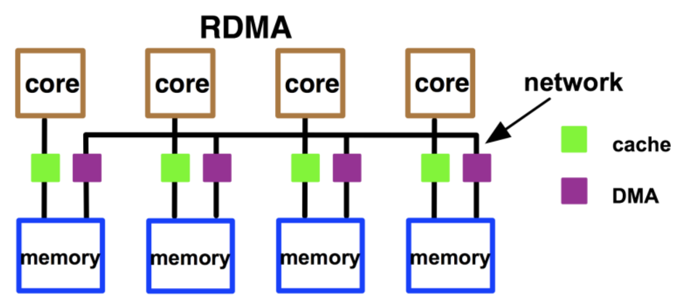
Today's laptops



Today's servers



Yesterday's clusters

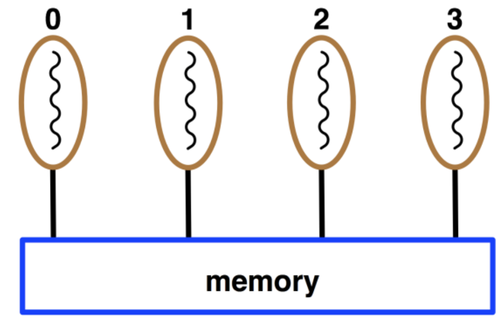


Today's clusters

- … and mixtures of those

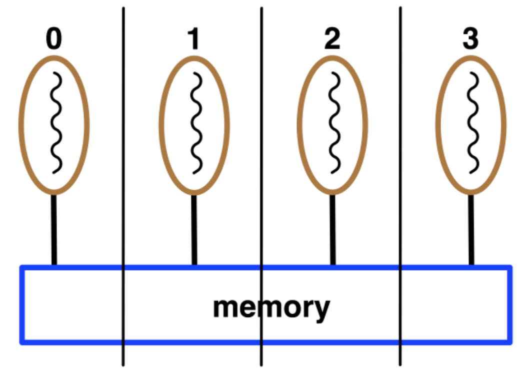
Programming Models

- **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)



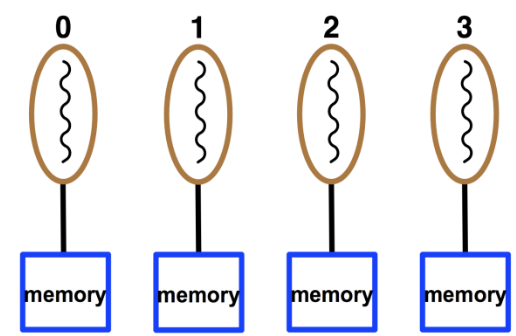
UMA

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



PGAS

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



DM

Shared Memory Machines

- **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?***

Source: IBM

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?***

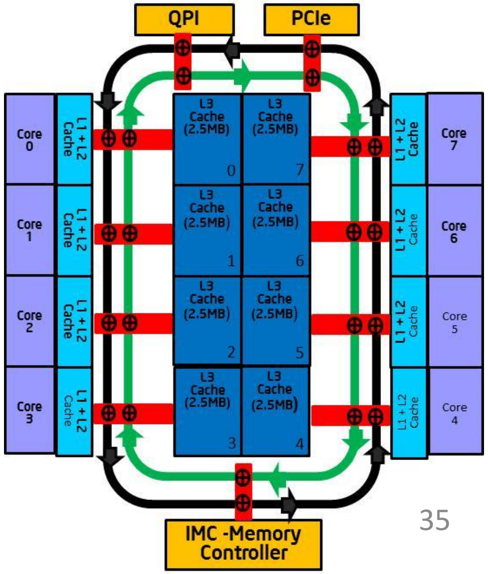
    Answer: *Cost can be cut with multistage connections (butterfly)*
  - "Minicomputer" – bus-based connection

    All traditional SMP systems

    High latency, low bandwidth (cache isimportant)

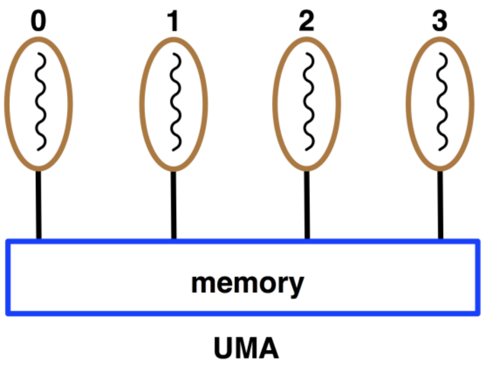
    Tricky to achieve highest performance (contention)

    Low cost, extensible

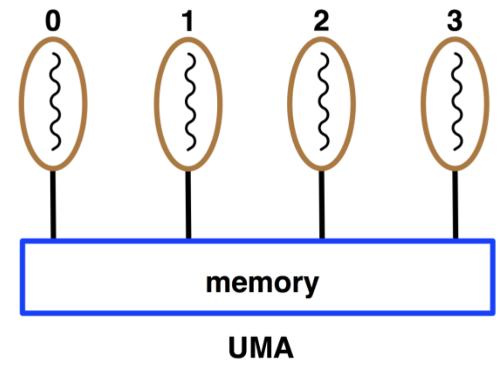


Shared Memory Machine Abstractions

- **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



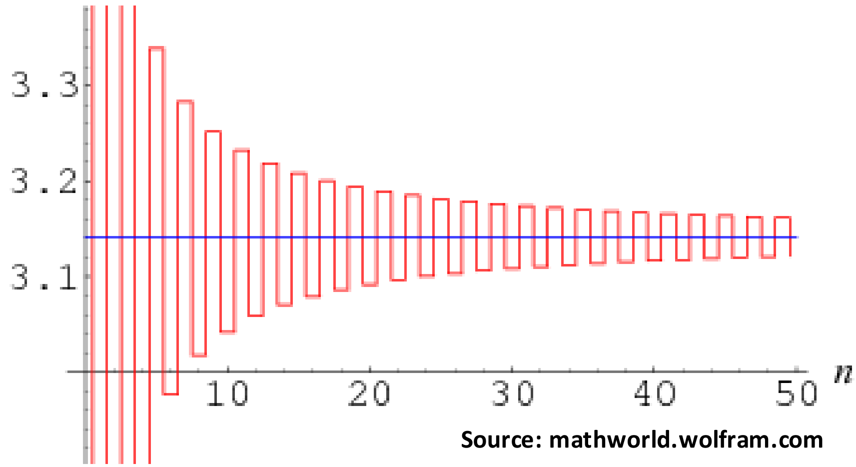
- **Threads or processes**
  - Communication through memory
- **Synchronization through memory or OS objects**
  - Lock/mutex (protect critical region)
  - Semaphore (generalization of mutex (binary sem.))
  - Barrier (synchronize a group of activities)
  - Atomic Operations (CAS, Fetch-and-add)
  - 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));
}
```

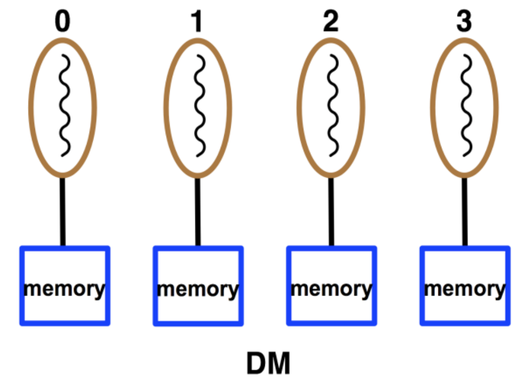
```
int n=10000;
double *resultarr;
int nthreads;

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!***

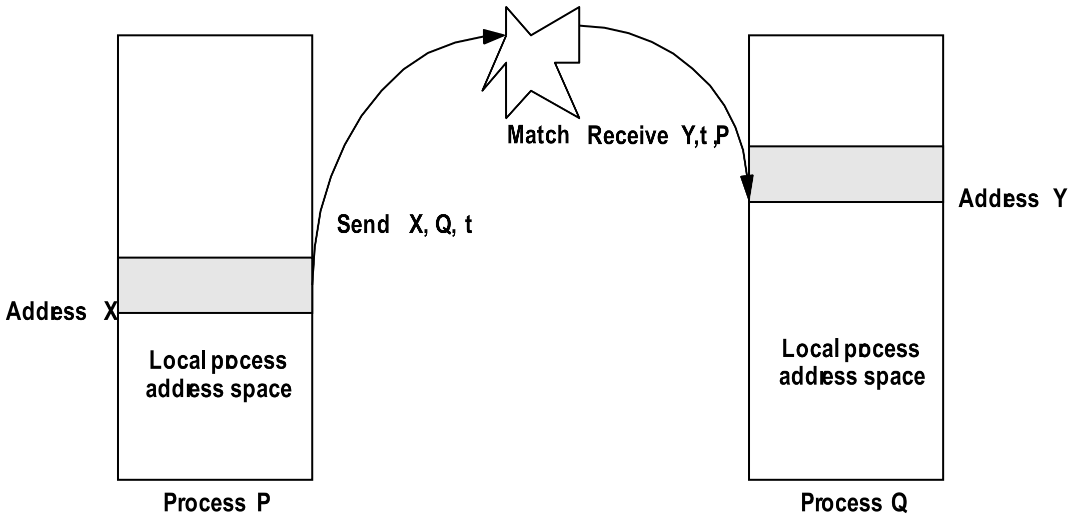
Distributed Memory Machine Programming

- **Explicit communication between PEs**
  - Message passing or channels
- **Only local memory access, no direct access to remote memory**
  - No shared resources (well, the network)

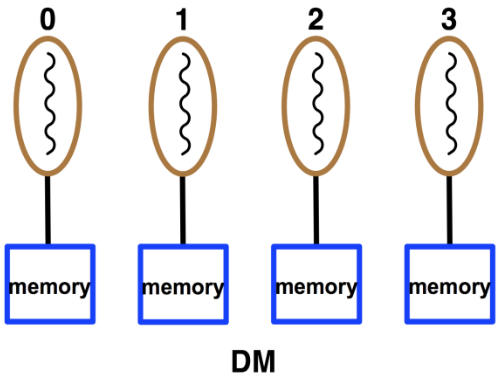


- **Programming model: Message Passing (MPI, PVM)**
  - Communication through messages or group operations (broadcast, reduce, etc.)
  - Synchronization through messages (sometimes unwanted side effect) or group operations (barrier)
  - Typically supports message matching and communication contexts

DMM Example: Message Passing

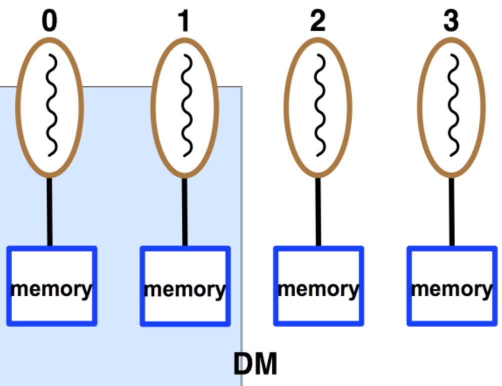


Source: John Mellor-Crummey



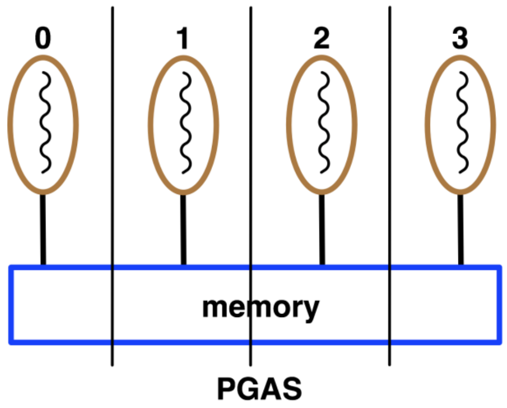
- Send specifies buffer to be transmitted
- Recv specifies buffer to receive into
- Implies copy operation between named PEs
- Optional tag matching
- Pair-wise synchronization (cf. happens before)

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();
}
```

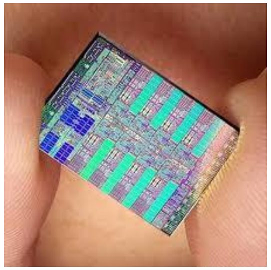
DMM Example: PGAS



- **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, …

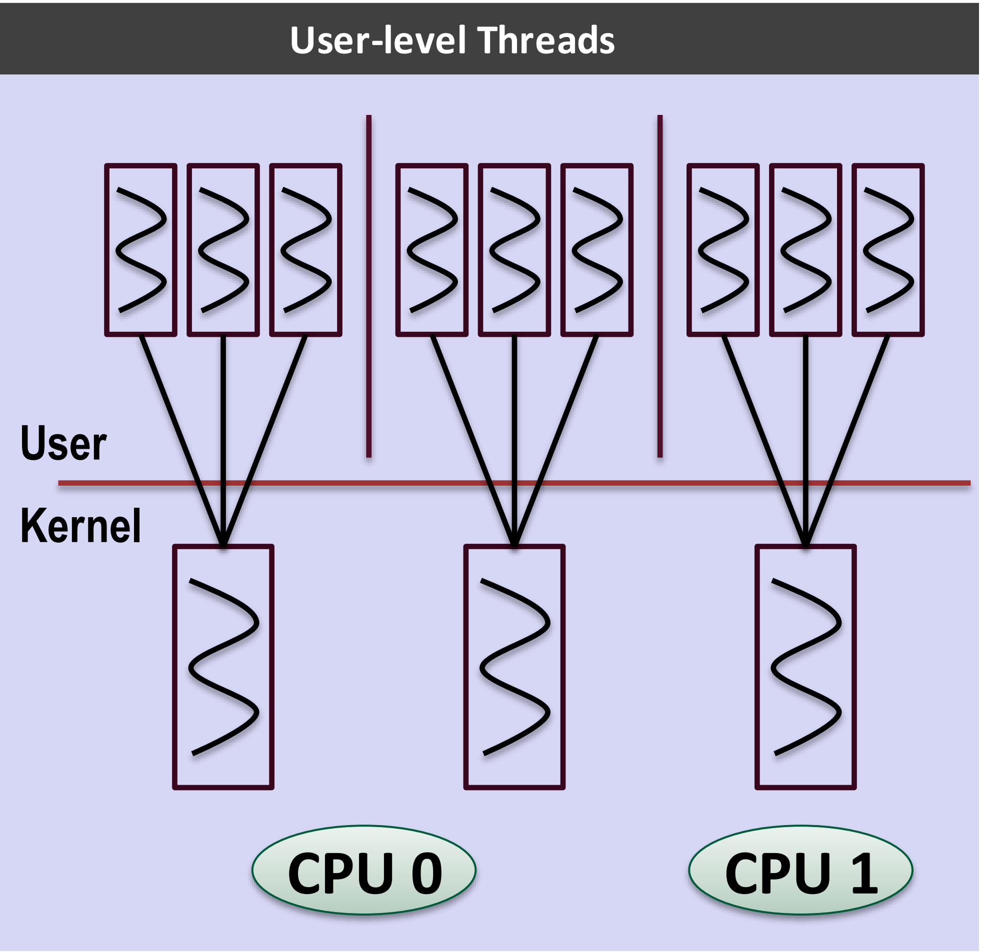
How to Tame the Beast?

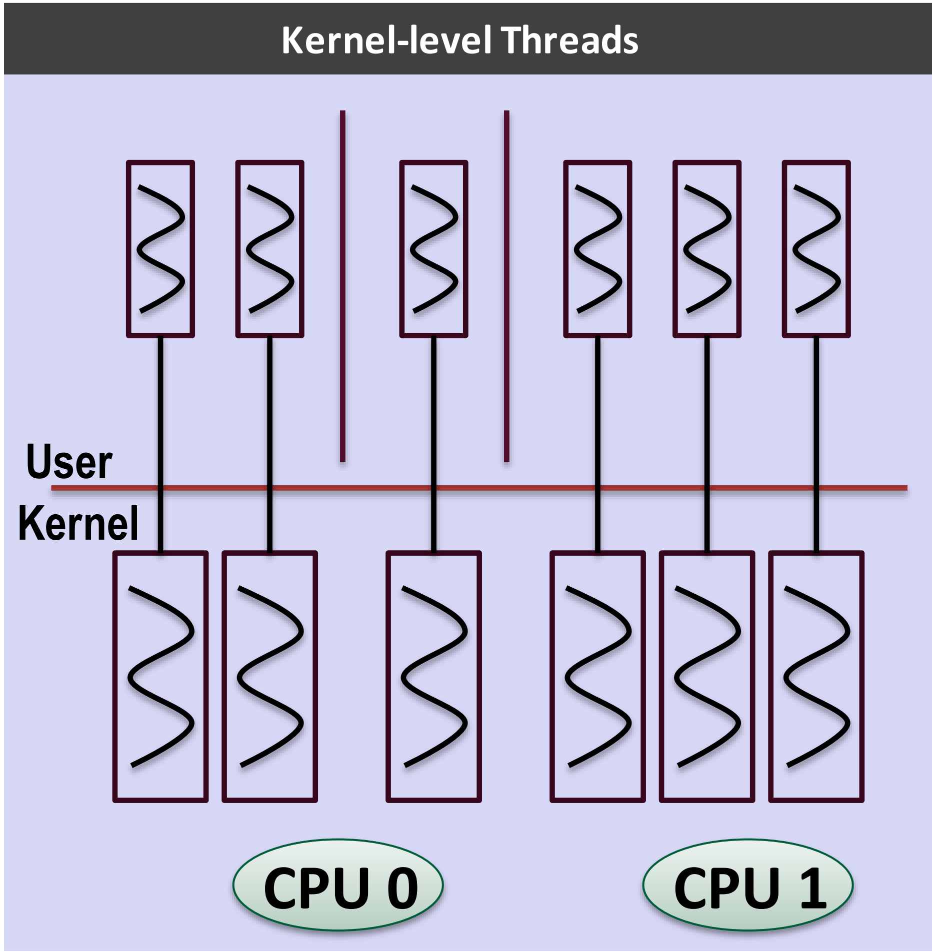
- **How to program large machines?**
- **No single approach, PMs are not converging yet**
  - MPI, PGAS, OpenMP, Hybrid (MPI+OpenMP, MPI+MPI, MPI+PGAS?), …
- **Architectures converge**
  - General purpose nodes connected by general purpose or specialized networks
  - Small scale often uses commodity 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

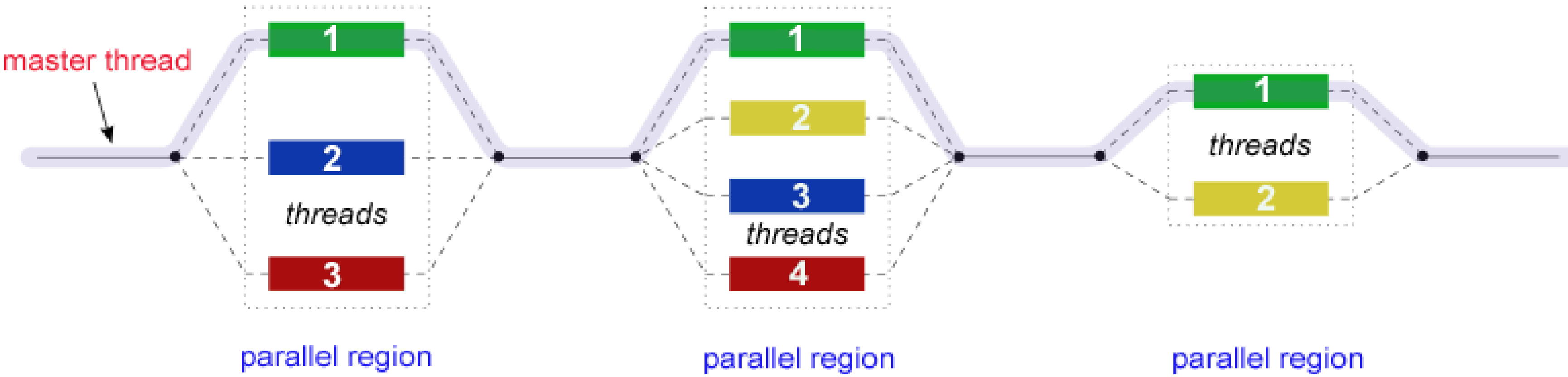




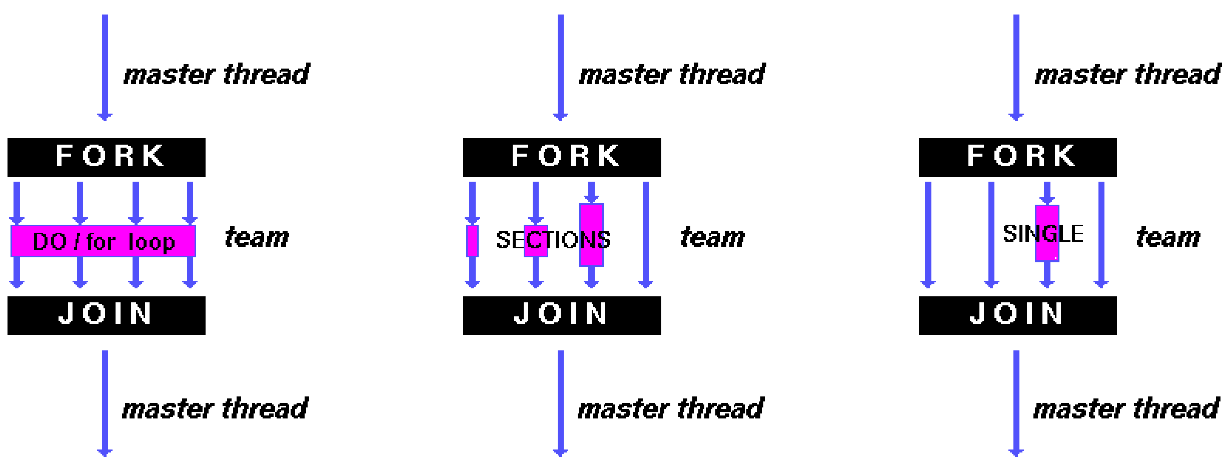
Practical SMM Programming: OpenMP

Source: OpenMP.org

- **Fork-join model**



- **Types of constructs:**



+ Tasks

Source: Blaise Barney, LLNL

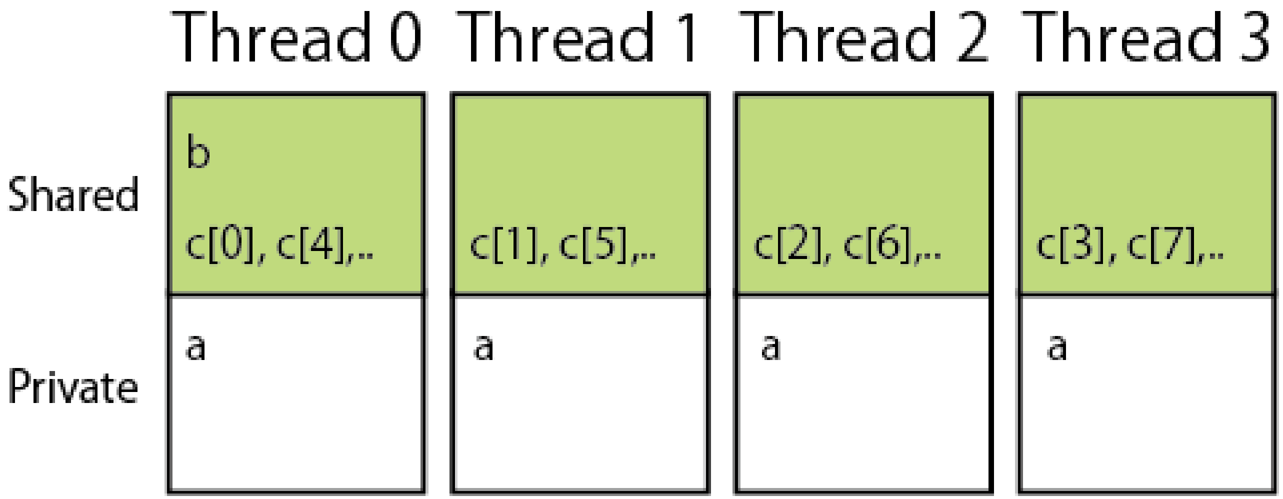
OpenMP General Code Structure

```
#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

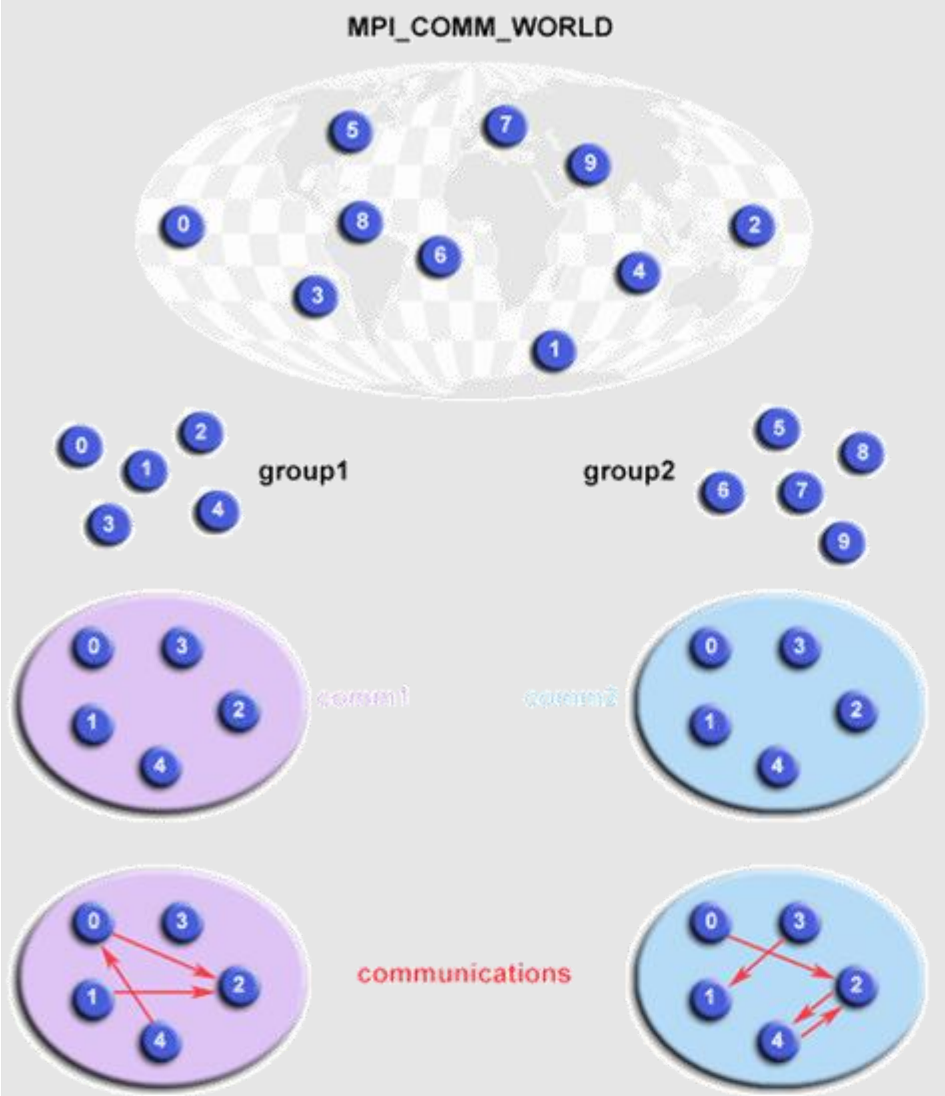
Practical PGAS Programming: UPC

- **PGAS extension to the C99 language**



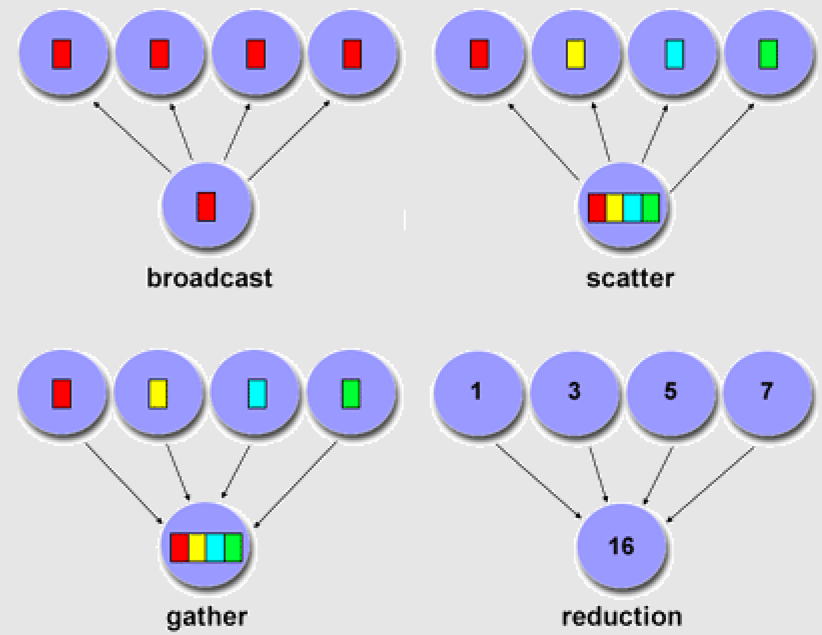
- **Many helper library functions**
  - Collective and remote allocation
  - Collective operations
- **Complex consistency model**

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)

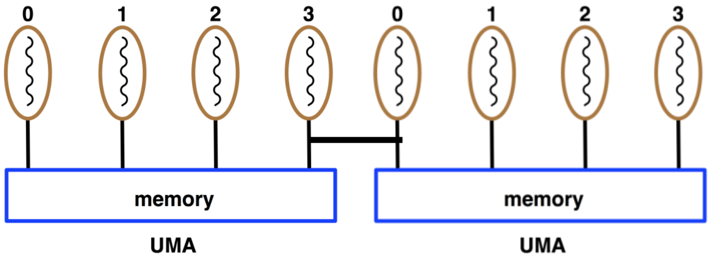
Source: Blaise Barney, LLNL

Complete Six Function MPI-1 Example

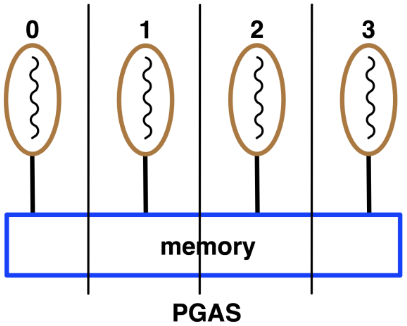
```
#include <mpi.h>

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

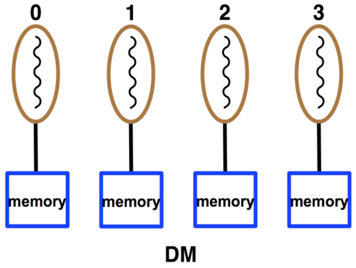
- **Support for shared memory in SMM domains**



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

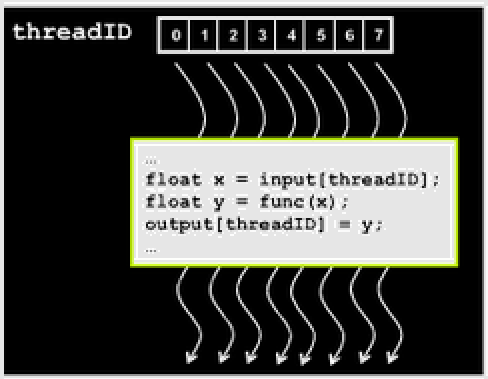


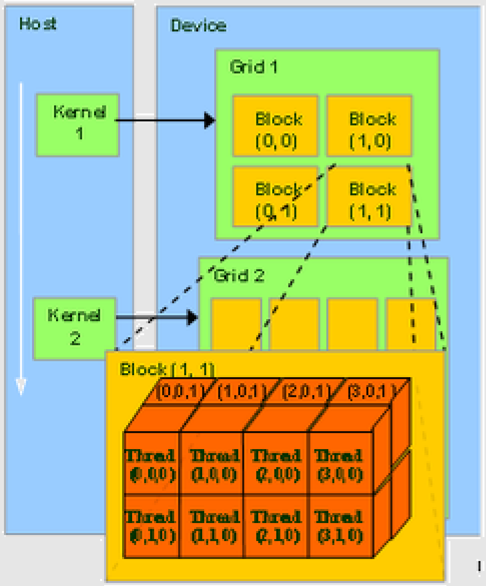
- **Enhanced support for message passing communication**
  - Scalable topologies
  - More nonblocking features
  - … many more



Accelerator example: CUDA

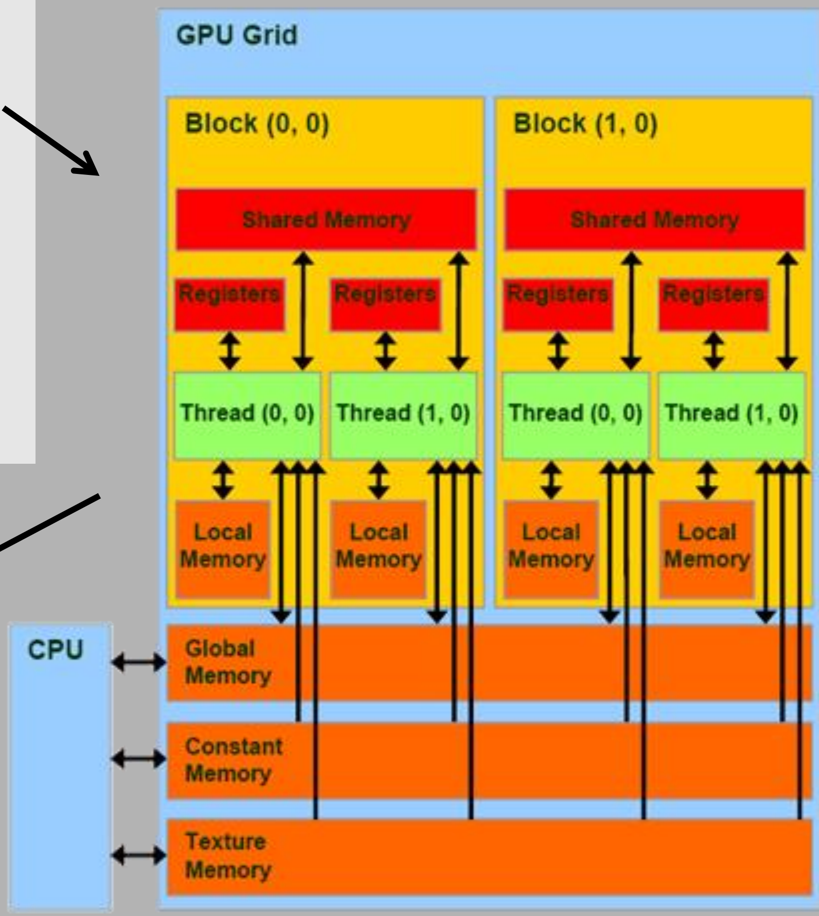
Hierarchy of Threads





Source: NVIDIA

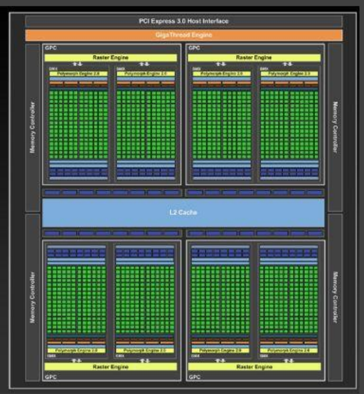
Complex Memory Model



Simple Architecture

Kepler Block Diagram

- 8 SMX
- 1536 CUDA Cores
- 8 Geometry Units
- 4 Raster Units
- 128 Texture Units
- 32 ROP units
- 256-bit GDDR5



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**

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:

DPHPC Overview