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

Human Brain – No Problem!

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

Exponential Growth of Computing

Can we do this today?
Other problem areas: Scientific Computing
- Most natural sciences are simulation driven and 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”!

Human Brain – No Problem!
Simulating 1 second of human brain activity takes 82,944 processors

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

Other problem areas: Scientific Computing
- Most natural sciences are simulation driven and 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)
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)

- National University of Defense Technology, China
  - Piz Daint @ CSCS
  - Blue Waters in 2009
  - This is why you need to understand performance expectations well!
  - Imagine you’re designing a $500 M supercomputer, and all you have is:
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.”

How to increase the compute power?

Microprocessor Transistor Counts 1971-2011 & Moore's Law

Not an option anymore!
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)

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

Resources and Programming

- Parallel Resource
  - Instruction-level parallelism
    - Pipelining
    - VLIW
    - Superscalar
  - SIMD operations
    - Vector operations
  - Instruction sequences
    - Multiprocessors
    - Multicores
    - Multithreading
- Programming
  - Compiler
    - (inline assembly)
    - Hardware scheduling
  - Compiler (inline assembly)
    - Libraries
  - 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)
- Dataflow Architectures
  - No program counter, execute instructions when all input arguments are available
  - Fine-grained, high overheads
  - Example: compute f = (a+b) * (c+d)
Von Neumann Architecture

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

<table>
<thead>
<tr>
<th>SISD</th>
<th>SIMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Serial Computer (nearly extinct)</td>
<td>Vector Machines or Extensions (very common)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MISD</th>
<th>MIMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redundant Execution (fault tolerance)</td>
<td>Multicore (ubiquitous)</td>
</tr>
</tbody>
</table>

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

- 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?
  - “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

---

10
22
31
32
33
34
35
36
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, ...

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., 
  - 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

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

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!

DMM Example: Message Passing

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

```c
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));
        }
        mpi = h * sum;
        MPI_Reduce(&mpii, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    }
    t+=MPI_Wtime();
    if (!myid) {
        printf("pi is approximately %.16f, Error is %.16f
", pi, fabs(pi - PI25DT));
        printf("time: %f
", t);
    }
    MPI_Finalize();
}
```

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

- Fork-join model
- Types of constructs:
  - Fork
  - Join
  - Tasks

OpenMP General Code Structure

```c
#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
}
```
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

- Collection of 1D address spaces
- Helper Functions
  - many more (>600 total)
- Source: Blaise Barney, LLNL

Complete Six Function MPI-1 Example

```c
#include <mpi.h>

int main(int argc, char **argv) {
  int myrank, ...
  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 */
  ...}

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

```c
#define N 10
int main( void ) {
  int a[N], b[N], c[N];
  int *dev_a, ...
  if (tid < N) 
    c[tid] = a[tid] + b[tid]; 
  }
}
```

Accelerator example: CUDA

```c
#define N 10
int main( void ) {
  int a[10], b[10], c[10];
  int *dev_a, *dev_b, *dev_c;
  __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];
  }
```

The Kernel
OpenACC / OpenMP 4.0

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

```c
#define N 10
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

- locals
  - cache hierarchy
  - shared memory
  - distributed memory
- memory
  - cache coherence
  - models
  - distributed algorithms
  - locks
  - group communications
  - wait free
  - linearizability
- Amdahl’s and Gustafson’s law
  - PRAM
  - LogP
- I/O complexity
  - balance principles I
  - balance principles II
  - Little’s Law
  - scheduling