Lecture 2: Terminology and Definitions

Abhinav Bhavele, Department of Computer Science
Announcements

- Slides and video from previous class are now posted online
- Assignments, and midterm dates are also online
- Deepthought2 accounts have been mailed out:
Group project timeline

- [Form groups: March 4]
- Finalize project topic: March 11
- Interim report due: April 15
- Project presentations: May 6, 11
- Final project and report due: May 13
Summary of last lecture

- Need for high performance computing
- Parallel architecture: nodes, memory, network, storage
- Programming models: shared memory vs. distributed
- Performance and debugging tools
- Systems issues: job scheduling, routing, parallel I/O, fault tolerance, power
- Parallel algorithms and applications
Cores, sockets, nodes

- CPU: processor
  - Single-core or multi-core
  - Core is a processing unit, multiple such units on a single chip make it a multi-core processor
- Socket: same as chip or processor
- Node: packaging of sockets

https://www.glennklockwood.com/hpc-howtos/process-affinity.html
Job scheduling
Job scheduling

- HPC systems use job or batch scheduling
- Each user submits their parallel programs for execution to a “job” scheduler

Job Queue

<table>
<thead>
<tr>
<th>#Nodes Requested</th>
<th>Time Requested</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>30 mins</td>
</tr>
<tr>
<td>64</td>
<td>24 hours</td>
</tr>
<tr>
<td>56</td>
<td>6 hours</td>
</tr>
<tr>
<td>192</td>
<td>12 hours</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Job scheduling

- HPC systems use job or batch scheduling
- Each user submits their parallel programs for execution to a “job” scheduler
- The scheduler decides:
  - what job to schedule next (based on an algorithm: FCFS, priority-based, ….)
  - what resources (compute nodes) to allocate to the ready job

<table>
<thead>
<tr>
<th>Job Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Nodes Requested</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>
Job scheduling

- HPC systems use job or batch scheduling
- Each user submits their parallel programs for execution to a “job” scheduler
- The scheduler decides:
  - what job to schedule next (based on an algorithm: FCFS, priority-based, ….)
  - what resources (compute nodes) to allocate to the ready job

- Compute nodes: dedicated to each job
- Network, filesystem: shared by all jobs

### Job Queue

<table>
<thead>
<tr>
<th>#Nodes</th>
<th>Time Requested</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>128 30 mins</td>
</tr>
<tr>
<td>2</td>
<td>64 24 hours</td>
</tr>
<tr>
<td>3</td>
<td>56 6 hours</td>
</tr>
<tr>
<td>4</td>
<td>192 12 hours</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>
Compute nodes vs. login nodes

- **Compute nodes**: dedicated nodes for running jobs
  - Can only be accessed when they have been allocated to a user by the job scheduler
- **Login nodes**: nodes shared by all users to compile their programs, submit jobs etc.
Supercomputers vs. commodity clusters

- Supercomputer refers to a large expensive installation, typically using custom hardware
  - High-speed interconnect
  - IBM Blue Gene, Cray XT, Cray XC
- Cluster refers to a cluster of nodes, typically put together using commodity (off-the-shelf) hardware
Serial vs. parallel code

• Thread: a thread or path of execution managed by the OS
  • Share memory

• Process: heavy-weight, processes do not share resources such as memory, file descriptors etc.

• Serial or sequential code: can only run on a single thread or process

• Parallel code: can be run on one or more threads or processes
Scaling and scalable

- Scaling: running a parallel program on 1 to n processes
  - 1, 2, 3, …, n
  - 1, 2, 4, 8, …, n
- Scalable: A program is scalable if it’s performance improves when using more resources
Scaling and scalable

- **Scaling**: running a parallel program on 1 to n processes
  - 1, 2, 3, … , n
  - 1, 2, 4, 8, …, n
- **Scalable**: A program is scalable if its performance improves when using more resources

![Graph showing execution time vs. number of cores]
Weak versus strong scaling

- **Strong scaling**: *Fixed total* problem size as we run on more processes
  - Sorting $n$ numbers on 1 process, 2 processes, 4 processes, …

- **Weak scaling**: *Fixed problem size per process but increasing total* problem size as we run on more processes
  - Sorting $n$ numbers on 1 process
  - $2n$ numbers on 2 processes
  - $4n$ numbers on 4 processes
Speedup and efficiency

- **Speedup:** Ratio of execution time on one process to that on $n$ processes

  \[
  \text{Speedup} = \frac{t_1}{t_n}
  \]

- **Efficiency:** Speedup per process

  \[
  \text{Efficiency} = \frac{t_1}{t_n \times n}
  \]
Amdahl’s law

- Speedup is limited by the serial portion of the code
  - Often referred to as the serial “bottleneck”
- Let’s say only a fraction $f$ of the code can be parallelized on $p$ processes

\[
\text{Speedup} = \frac{1}{(1 - f) + f/p}
\]
Amdahl’s law

- Speedup is limited by the serial portion of the code
  - Often referred to as the serial “bottleneck”
- Lets say only a fraction $f$ of the code can be parallelized on $p$ processes

\[
\text{Speedup} = \frac{1}{(1 - f) + \frac{f}{p}}
\]
Amdahl’s law

- Speedup is limited by the serial portion of the code
  - Often referred to as the serial “bottleneck”
- Lets say only a fraction $f$ of the code can be parallelized on $p$ processes

$$\text{Speedup} = \frac{1}{(1 - f) + \frac{f}{p}}$$
Amdahl’s law

\[ \text{Speedup} = \frac{1}{(1 - p) + p/n} \]

```c
fprintf(stdout,"Process %d of %d is on %s\n", myid, numprocs, processor_name);
fflush(stdout);

n = 10000;    /* default # of rectangles */
if (myid == 0)
    startwtime = MPI_Wtime();

MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

h   = 1.0 / (double) n;
sum = 0.0;
/* A slightly better approach starts from large i and works back */
for (i = myid + 1; i <= n; i += numprocs)
{
    x = h * ((double)i - 0.5);
    sum += f(x);
}
mypi = h * sum;

MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
```

\[ 100 - p = 40 \text{ s on 1 process} \]

\[ \text{Speedup} = \frac{1}{(1 - 0.6) + 0.6/n} \]

\[ p = 60 \text{ s on 1 process} \]
Communication and synchronization

- Each physical node might compute independently for a while
- When data is needed from other (remote) nodes, messaging occurs
  - Referred to as communication or synchronization or MPI messages
- Intra-node vs. inter-node communication
- Bulk synchronous programs: All processes compute simultaneously, then synchronize together
Different models of parallel computation

- SIMD: Single Instruction Multiple Data
- MIMD: Multiple Instruction Multiple Data
- SPMD: Single Program Multiple Data
  - Typical in HPC
Writing parallel programs

- Decide the serial algorithm first
- Data: how to distribute data among threads/processes?
  - Data locality: assignment of data to specific processes to minimize data movement
- Computation: how to divide work among threads/processes?
- Figure out how often communication will be needed
Two-dimensional stencil computation

- Commonly found kernel in computational codes
- Heat diffusion, Jacobi method, Gauss-Seidel method

2D stencil iteration in parallel
2D stencil iteration in parallel

- 1D decomposition
  - Divide rows (or columns) among processes
2D stencil iteration in parallel

- 1D decomposition
  - Divide rows (or columns) among processes
2D stencil iteration in parallel

- 1D decomposition
  - Divide rows (or columns) among processes

- 2D decomposition
  - Divide both rows and columns (2d blocks) among processes
2D stencil iteration in parallel

- **1D decomposition**
  - Divide rows (or columns) among processes

- **2D decomposition**
  - Divide both rows and columns (2d blocks) among processes
N-body problem

N-body problem

- Simulating the movement of N-bodies under gravitational forces

N-body problem

• Simulating the movement of N-bodies under gravitational forces

• Naive algorithm: $O(n^2)$
  • Every body calculates forces pair-wise with every other body (particle)

Data distribution in N-body problems

- Naive approach: Assign $n/k$ particles to each process
- Other approaches?

http://datagenetics.com/blog/march222013/
https://en.wikipedia.org/wiki/Z-order_curve
Data distribution in N-body problems

- Naive approach: Assign $n/k$ particles to each process
- Other approaches?

Space-filling curves

http://datagenetics.com/blog/march22013/
https://en.wikipedia.org/wiki/Z-order_curve
Data distribution in N-body problems

- Naive approach: Assign $n/k$ particles to each process
- Other approaches?

http://datagenetics.com/blog/march222013/
https://en.wikipedia.org/wiki/Z-order_curve

Space-filling curves
Data distribution in N-body problems

- Naive approach: Assign n/k particles to each process
- Other approaches?

Space-filling curves

- http://datagenetics.com/blog/march22013/

Data distribution in N-body problems

- Let us consider a two-dimensional space with bodies/particles in it
Let us consider a two-dimensional space with bodies/particles in it.
Data distribution in N-body problems

- Let us consider a two-dimensional space with bodies/particles in it

Quad-tree: not all nodes are shown
Load balance and grain size

- Load balance: try to balance the amount of work (computation) assigned to different threads/processes
  - Bring ratio of maximum to average load as close to 1 as possible
  - Secondary consideration: also load balance amount of communication

- Grain size: ratio of computation-to-communication
  - Coarse-grained (more computation) vs. fine-grained (more communication)
Questions?

Abhinav Bhatele
5218 Brendan Iribe Center (IRB) / College Park, MD 20742
phone: 301.405.4507 / e-mail: bhatele@cs.umd.edu