Parallel Computing Tutorials

Xingfu Wu
Department of Computer Science & Engineering
Texas A&M University
IAMCS, Feb 2012
I. Parallel Programming

Focus: how to program in parallel
Session 1: Overview of PP (today)
Session 2: MPI + Matlab/Octave/R

II. Parallel Performance Evaluation
Methodology and Techniques

Focus: how to evaluate para. programs
Session 1: Methodology and Tech.
Session 2: Efficient parallelization
Introduction to Parallel Programming
(Session 1: Overview of PP)

Xingfu Wu

Department of Computer Science & Engineering
Texas A&M University

IAMCS, Feb 10, 2012
Survey

- Who is mainly programming in:
  - Fortran, C or C++
  - Matlab/Octave, R, Java

- Who used linux/unix systems before?

- Who has experience with parallel processing?
  - If yes, who has used MPI?
  - If yes, who has used OpenMP?
  - Any other parallel programming model(s) used?

- Who plans to use parallel processing in the future?
Outline

- Introduction
- Parallel Architectures
- Parallel Programming
  - MPI and OpenMP
  - MPI + Matlab/Octave
  - MPI + R
- IAMCS Clusters: Brazos and Hurr
- Try to parallelize your own code
- Q&A
Ground Motion of the 2008 Ms 8.0 Wenchuan earthquake (China)
Scientific Simulation Process

Physical modeling → Integrated parallel meshing, partitioning, solving, and visualizing → Scientific understanding

Jaguar: World’s most powerful computer
Designed for science from the ground up

Source: T. Tu et al., From mesh generation to scientific visualization: an end-to-end approach to parallel supercomputing, SC2006
## World Top 10 Supercomputers (11/2011)

<table>
<thead>
<tr>
<th>Rank</th>
<th>Site</th>
<th>Computer/Year Vendor</th>
<th>Cores</th>
<th>R&lt;sub&gt;max&lt;/sub&gt;</th>
<th>R&lt;sub&gt;peak&lt;/sub&gt;</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RIKEN Advanced Institute for Computational Science (AICS) Japan</td>
<td>K computer, SPARC64 VIIIxf 2.0GHz, Tofu interconnect / 2011 Fujitsu</td>
<td>705024</td>
<td>10510.00</td>
<td>11280.38</td>
<td>12659.9</td>
</tr>
<tr>
<td>2</td>
<td>National Supercomputing Center in Tianjin China</td>
<td>NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050 / 2010 NUDT</td>
<td>186368</td>
<td>2566.00</td>
<td>4701.00</td>
<td>4040.0</td>
</tr>
<tr>
<td>3</td>
<td>DOE/SC/Oak Ridge National Laboratory United States</td>
<td>Cray XT5-HE Opteron 6-core 2.6 GHz / 2009 Cray Inc.</td>
<td>224162</td>
<td>1759.00</td>
<td>2331.00</td>
<td>6950.0</td>
</tr>
<tr>
<td>4</td>
<td>National Supercomputing Centre in Shenzhen (NSCS) China</td>
<td>Dawning TC3600 Blade System, Xeon X5650 6C 2.66GHz, Infiniband QDR, NVIDIA 2050 / 2010 Dawning</td>
<td>120640</td>
<td>1271.00</td>
<td>2984.30</td>
<td>2580.0</td>
</tr>
<tr>
<td>5</td>
<td>GSIC Center, Tokyo Institute of Technology Japan</td>
<td>HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows / 2010 NEC/HP</td>
<td>73278</td>
<td>1192.00</td>
<td>2287.63</td>
<td>1398.6</td>
</tr>
<tr>
<td>6</td>
<td>DOE/NNSA/LANL/SNL United States</td>
<td>Cray XE6, Opteron 6136 8C 2.40GHz, Custom / 2011 Cray Inc.</td>
<td>142272</td>
<td>1110.00</td>
<td>1365.81</td>
<td>3980.0</td>
</tr>
<tr>
<td>7</td>
<td>NASA/Ames Research Center/NAS United States</td>
<td>SGI Altix ICE 8200EX/8400EX, Xeon HT QC 3.0/Xeon 5570/5670 2.93 Ghz, Infiniband / 2011 SGI</td>
<td>111104</td>
<td>1088.00</td>
<td>1315.33</td>
<td>4102.0</td>
</tr>
<tr>
<td>8</td>
<td>DOE/SC/LBNL/NERSC United States</td>
<td>Cray XE6, Opteron 6172 12C 2.10GHz, Custom / 2010 Cray Inc.</td>
<td>153408</td>
<td>1054.00</td>
<td>1288.63</td>
<td>2910.0</td>
</tr>
<tr>
<td>9</td>
<td>Commissariat a l’Energie Atomique (CEA) France</td>
<td>Bull bullix super-node S6010/S6030 / 2010 Bull</td>
<td>138368</td>
<td>1050.00</td>
<td>1254.55</td>
<td>4590.0</td>
</tr>
<tr>
<td>10</td>
<td>DOE/NNSA/LANL United States</td>
<td>BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 GHz / Opteron DC 1.8 GHz, Voltaire Infiniband / 2009 IBM</td>
<td>122400</td>
<td>1042.00</td>
<td>1375.78</td>
<td>2345.0</td>
</tr>
</tbody>
</table>

**Source:** [www.top500.org](http://www.top500.org) (R: Tflops  Power: KW)
Parallel Computing

- Parallel computing benefits
  - decrease the wall-time taken by program execution of a problem:
    - Several processors work together to solve the problem,
    - Total CPU time does not decrease
  - increase the size of the problem that can be solved

- Parallel programming is more difficult than sequential programming
Why is Parallel Programming harder

- Developing a parallel algorithm
  - Complexity of specifying and coordinating concurrent activities

- Parallel software development
  - Lack of standardized & effective development tools, programming models, and environments

- Rapid pace of change in architecture
  - Today’s parallel algorithms may not be suitable for tomorrow’s systems
Parallelization

- Parallelization is simply another optimization technique to get your results faster
- To this end, more than one processor is used to solve the problem
- The latter is a difference with serial optimization, where one makes better use of existing resources i.e. the cost comes down
Outline

- **Introduction**
- **Parallel Architectures**
- **Parallel Programming**
  - MPI and OpenMP
  - MPI + Matlab/Octave
  - MPI + R
- **IAMCS Clusters: Brazos and Hurr**
- Try to parallelize your own code
- **Q&A**
IBM Quad-core PPC450
AMD Hex-core Opteron

HyperTransport™ technology links provide up to 57.6 GB/s peak bandwidth per processor.

12.8 GB/s @ DDR2-800
ORNL Cray XT4/5 Supercomputer

Jaguar: World’s most powerful computer
Designed for science from the ground up

Source: ORNL National Center for Computational Science
Classifying Parallel Architectures

- Ongoing debate about labeling a system
  - Hard to classify
  - Most share some characteristics, not all

- Classifying systems based on main memory
  - Shared or Distributed?
    - Can all processors access all of memory or only a subset?
  - Memory access time
    - Uniform or non-uniform?
Memory is shared resource for CPUs
Memory access time for all CPUs is uniform
UMA: Uniform Memory Access
Easy to use but not-scalable
IBM Quad-core PPC450
AMD Hex-core Opteron

HyperTransport™ technology links provide up to 57.6 GB/s peak bandwidth per processor*.
Distributed memory (NUMA)

- Memory is not shared resource for all CPUs
- Memory access time for all CPUs is non-uniform
- NUMA: Non-uniform memory access
- Hard to use but scalable
Cluster of Multi-/Many-Core

- Distributed shared memory: UMA within a node and NUMA across nodes
- Multiple levels of parallelism, Scalable

Ethernet, Infiniband, etc
ANL BlueGene/P

Source: ANL Leadership Computing Facility
ORNL Cray XT4/5 Supercomputer

Jaguar: World’s most powerful computer
Designed for science from the ground up

Source: ORNL National Center for Computational Science
ORNL Cray XT4/5 Supercomputer

Source: ORNL National Center for Computational Science
Outline

- Introduction
- Parallel Architectures
- Parallel Programming
  - MPI and OpenMP
  - Matlab/Octave + MPI
  - MPI + R
- IAMCS Clusters: Brazos and Hurr
- Try to parallelize your own code
- Q&A
How to Program a Parallel Computer

The most well-known ones are:

- **Distributed Memory**
  - PVM - Parallel Virtual Machine (obsolete)
  - MPI - Message Passing Interface (de-facto standard)

- **Shared Memory**
  - Posix Threads (standardized, low level)
  - OpenMP (de-facto standard)
  - Automatic Parallelization (compiler does it for you)
Distributed Memory Programming Model

Transfer Mechanism
Distributed Memory Programming Model

Serial

message-passing

Process 0
Node 1

Process 1
Node 2

Process 2
Node 3

Process 3
Node 4

Data transmission over the interconnect
MPI: Message Passing Interface

- MPI: a standard library for message-passing that can be used to develop portable message-passing programs using either C, C++ or Fortran.
- The MPI standard defines both the syntax as well as the semantics of a core set of library routines.
- Vendor implementations of MPI are available on almost all commercial parallel computers.
- See http://www.mpi-forum.org/
# MPI: Message Passing Interface

The minimal set of MPI routines

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Init</td>
<td>Initializes MPI.</td>
</tr>
<tr>
<td>MPI_Finalize</td>
<td>Terminates MPI.</td>
</tr>
<tr>
<td>MPI_Comm_size</td>
<td>Determines the number of processes.</td>
</tr>
<tr>
<td>MPI_Comm_rank</td>
<td>Determines the label of calling process.</td>
</tr>
<tr>
<td>MPI_Send</td>
<td>Sends a message.</td>
</tr>
<tr>
<td>MPI_Recv</td>
<td>Receives a message.</td>
</tr>
</tbody>
</table>
Example: MPI + C

```
#include <stdio.h>
#include "mpi.h"

int main( int argc, char **argv )
{
    int rank, size;

    MPI_Init( &argc, &argv );
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf( "Hello world from process %d of %d\n", rank, size );
    MPI_Finalize();
    return 0;
}
```

Output: Hello world from process 0 of 2
Hello world from process 1 of 2
Example: MPI + Fortran

program main

include "mpif.h"

integer rank, size

call MPI_INIT( ierr )
call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierr)
print *, "Hello world from process ", rank, " of ", size
call MPI_FINALIZE()
stop
end

Output: Hello world from process 0 of 2
Hello world from process 1 of 2
Shared Memory Programming Model
Shared Memory Programming Model

Single thread

```
<table>
<thead>
<tr>
<th>S1</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
</table>
```

Multi-thread

```
<table>
<thead>
<tr>
<th>S1</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
</table>
```

- fork
- join

Thread

Shared address space

Process
Shared Memory Programming Model

Fork and Join Model

- Master Thread
- Worker Threads
- Parallel region
- Synchronization
OpenMP: a Standard for Directive Based Parallel Programming (http://openmp.org)

Three components:

- **Set of compiler directives for**
  - creating threads
  - sharing the work among threads
  - synchronizing the threads

- **Library routines** for setting and querying thread attributes

- **Environment variables** for controlling runtime behavior of the parallel program
OpenMP Programming Model

- OpenMP is a directive-based API that can be used with FORTRAN, C, and C++.
- OpenMP directives in C and C++ are based on the \#pragma compiler directives.
- A directive consists of a directive name followed by clauses.
  \#pragma omp directive [clause list]
- OpenMP programs execute serially until they encounter the parallel directive, which creates a group of threads.
  \#pragma omp parallel [clause list]
  /* structured block */
- The main thread that encounters the parallel directive becomes the master of this group of threads and is assigned the thread id 0 within the group.
Example: OpenMP + C

```c
#include <stdio.h>
#include <omp.h>

int main( int argc, char **argv )
{
    int rank = 0, size = 1;

    #pragma omp parallel private(rank) shared(size)
    {
        #if defined (_OPENMP)
            size = omp_get_num_threads();
            rank = omp_get_thread_num();
        #endif
        printf("Hello from thread %d of %d \n", rank, size);
    }
}

Output: Hello world from thread 0 of 2
        Hello world from thread 1 of 2
```
MPI + X Programming Models

- MPI + C, C++, Fortran
- MPI + Matlab/Octave (next session)
- MPI + R (Rmpi, snow)
- MPI + Java

Hybrid Parallel Programming

- MPI + OpenMP
- MPI + CUDA (Support GPU)
- MPI + OpenMP + CUDA
Hybrid Parallel Programming Models

Message-passing

Process 0

Node 1

Process 1

Node 2
Outline

- Introduction
- Parallel Architectures
- Parallel Programming
  - MPI and OpenMP
  - MPI + Matlab/Octave
  - MPI + R
- IAMCS Clusters: Brazos and Hurr
- Try to parallelize your own code
- Q&A
IAMCS Computing Resources

- Brazos, a linux cluster with 1376 cores (http://brazos.tamu.edu)
- Hurr, a linux cluster with 1024 cores
- Shaheen, an IBM BlueGene/P system with 65,600 cores from KAUST
IAMCS Clusters: Brazos/Hurr

- Total of 300 nodes, 2400 cores

- Brazos:
  - 172 nodes, each with two quad-core Intel Xeon (Harpertown) processors with 2.5GHz and 16GB per node

- Hurr
  - 128 nodes, each with two quad core AMD Opteron (Shanghai) processors with 2.5GHz and 32GB per node
Remote Login Brazos

- ssh username@brazos.tamu.edu
- See brazos.tamu.edu for the details
- Setup Environments (.bashrc)
  - Compilers
  - MPI library
  - Matlab/Octave, R
- Storage: /home/wuxf, /data/wuxf
- Examples:
  cp –r /data/wuxf/examples
Compiling and Linking

- **MPI + X codes**
  - `mpicc -o hello hello.c`
  - `mpicxx -o hello hello.cc`
  - `mpif77 -o hello hello.f`

- **OpenMP + C code**
  - `cc -ffopenmp -o hello hello.c`
Running the Code

- **Interactive Mode**
  - what most of us are used to
  - Like we are "interacting" with the computer
  - Use command line to run the program immediately

- **Batch Mode**
  - The programs (or jobs) are not performed immediately, but instead, put onto a queue to execute later
  - Use a scheduler to schedule the job to run
  - Use PBS (Portable Batch Scheduler) for a job
Running the Code (interactive)

- **MPI + X codes**
  - `mpirun -np 2 hello`
  - Run the program and get wall-time:
    ```
    time mpirun -np 2 hello
    ```

- **OpenMP + C code**
  - Set environment variable for the number of threads:
    - `tcsh`
      ```
      setenv OMP_NUM_NUM_THREADS 2
      ```
    - `sh/bash`
      ```
      export OMP_NUM_THREADS=2
      ```
  - Run program and get wall-time:
    ```
    time hello
    ```
Running MPI Code (batch mode)

```bash
#!/bin/sh

#PBS -l walltime=00:10:00
#PBS -q iamcs
#PBS -l nodes=2:ppn=8
#PBS -j oe
#PBS -N hellompi

Nprocs=16

source ~/.bashrc

### go to your working directory

cd $HOME/examples/mpi/hello

### run the program

time mpirun -np $Nprocs ./hello
```
Running OpneMP Code (batch)

```
#!/bin/sh
#PBS -l walltime=00:10:00
#PBS -q iamcs
#PBS -l nodes=1:ppn=8
#PBS -j oe
#PBS -N helloomp
Nprocs=8
###Set the number of OpenMP threads
export OMP_NUM_THREADS=4
### go to your working directory
cd $HOME/examples/omp
### run the program
time ./hello
```
Submit a batch job

- TORQUE Resource Manager provides control over batch jobs and distributed computing resources.

- It is an advanced open-source product based on the OpenPBS.

Submit a batch job

- `qsub file.pbs`

Look at the job status

- `qstat -u username`
- `qdel jobid` (delete the job with jobid)
Outline

- Introduction
- Parallel Architectures
- Parallel Programming
  - MPI and OpenMP
  - MPI + Matlab/Octave
  - MPI + R
- IAMCS Clusters: Brazos and Hurr
- Try to parallelize your own code
- Q&A
Introduction to Parallel Programming
(Session 2: MPI + Matlab/Octave/R)

Xingfu Wu

Department of Computer Science & Engineering
Texas A&M University

IAMCS, Feb 2012 (Next Week)