L17: Design Review and 6-Function MPI

Administrative

- Organick Lecture: TONIGHT
  - David Shaw, "*Watching Proteins Dance: Molecular Simulation and the Future of Drug Design*", 220 Skaggs Biology, Reception at 6:15, talk at 7:00PM
  - Round-table with Shaw in the Large Conference Room (**MEB 3147) beginning TODAY at 3:30pm (refreshments!**
  - **Technical talk TOMORROW** "*Anton: A Special-Purpose Machine That Achieves a Hundred-Fold Speedup in Biomolecular Simulations*", 104 WEB, Reception at 11:50, talk at 12:15PM

Design Reviews

- Goal is to see a solid plan for each project and make sure projects are on track
  - Plan to evolve project so that results guaranteed
  - Show at least one thing is working
  - How work is being divided among team members
- Major suggestions from proposals
  - Project complexity – break it down into smaller chunks with evolutionary strategy
  - Add references – what has been done before? Known algorithm? GPU implementation?

Design Reviews

- Oral, 10-minute Q&A session (April 4 in class, plus office hours if needed)
  - Each team member presents one part
  - Team should identify "lead" to present plan
- Three major parts:
  - I. Overview
  - Define computation and high-level mapping to GPU
  - II. Project Plan
  - The pieces and who is doing what.
  - What is done so far? (Make sure something is working by the design review)
  - III. Related Work
  - Prior sequential or parallel algorithms/implementations
  - Prior GPU implementations (or similar computations)
- Submit slides and written document revising proposal that covers these and cleans up anything missing from proposal.

Final Project Presentation

- Dry run on April 18
  - Easels, tape and poster board provided
  - Tape a set of Powerpoint slides to a standard 2'x3' poster, or bring your own poster.
- Poster session during class on April 23
  - Invite your friends, profs who helped you, etc.
- Final Report on Projects due May 4
  - Submit code
  - And written document, roughly 10 pages, based on earlier submission.
  - In addition to original proposal, include
    - Project Plan and How Decomposed (from DR)
    - Description of CUDA implementation
    - Performance Measurement
    - Related Work (from DR)

Let's Talk about Demos

- For some of you, with very visual projects, I encourage you to think about demos for the poster session
- This is not a requirement, just something that would enhance the poster session
- Realistic?
  - I know everyone's laptops are slow …
  - … and don't have enough memory to solve very large problems
- Creative Suggestions?
  - Movies captured from run on larger system

Message Passing and MPI

- Message passing is the principle alternative to shared memory parallel programming, predominant programming model for supercomputers and clusters
  - Portable
  - Low-level, but universal and matches earlier hardware execution model
- What it is
  - A library used within conventional sequential languagess (Fortran, C, C++)
  - Based on Single Program, Multiple Data (SPMD)
  - Isolation of separate address spaces
    - + no data races, but communication errors possible
    - + exposes execution model and forces programmer to think about locality, both good for performance
    - - Complexity and code growth!

Like OpenMP, MPI arose as a standard to replace a large number of proprietary message passing libraries.

Message Passing Library Features

- All communication, synchronization require subroutine calls
  - No shared variables
  - Program runs on a single processor just like any uniprocessor program, except for calls to message passing library
- Subroutines for
  - Communication
    - Pairwise or point-to-point: A message is sent from a specific sending process (point a) to a specific receiving process (point b).
    - Collectives involving multiple processors
      - Move data: Broadcast, Scatter/gather
      - Compute and move: Reduce, AllReduce
  - Synchronization
    - Barrier
    - No locks because there are no shared variables to protect
  - Queries
    - How many processes? Which one am I? Any messages waiting?

MPI References

- The Standard itself:
  - at http://www.mpi-forum.org
  - All MPI official releases, in both postscript and HTML
- Other information on Web:
  - at http://www.mcs.anl.gov/mpi
  - pointers to lots of stuff, including other talks and tutorials, a FAQ, other MPI pages

Slide source: Bill Gropp

Compilation

wrapper script to compile

source file

```
mpicc -g -Wall -o mpi_hello mpi_hello.c
```

produce debugging information

create this executable file name (as opposed to default a.out)

turns on all warnings

Execution

mpiexec -n <number of processes> <executable>

---

mpiexec -n 1 ./mpi_hello

run with 1 process

mpiexec -n 4 ./mpi_hello

run with 4 processes

Hello (C)

```
#include "mpi.h"
#include <stdio.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( "Greetings from process %d of
              %d\n", rank, size );
    MPI_Finalize();
    return 0;
}
```

Hello (C++)

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

int main( int argc, char *argv[] )
{
    int rank, size;
    MPI::Init(argc, argv);
    rank = MPI::COMM_WORLD.Get_rank();
    size = MPI::COMM_WORLD.Get_size();
    std::cout << "Greetings from process " << rank << "
         of " << size << "\n";
    MPI::Finalize();
    return 0;
}
```


Execution

```
mpiexec -n 1 ./mpi_hello

    Greetings from process 0 of 1 !
```

```
mpiexec -n 4 ./mpi_hello

    Greetings from process 0 of 4 !
    Greetings from process 1 of 4 !
    Greetings from process 2 of 4 !
    Greetings from process 3 of 4 !
```


MPI Components

- MPI_Init
  - Tells MPI to do all the necessary setup.

```
int MPI_Init(
      int*      argc_p  /* in/out */,
      char***   argv_p  /* in/out */);
```

- MPI_Finalize
  - Tells MPI we're done, so clean up anything allocated for this program.

```
int MPI_Finalize(void);
```


Basic Outline

```
. . .
#include <mpi.h>
. . .
int main(int argc, char* argv[]) {
   . . .
   /* No MPI calls before this */
   MPI_Init(&argc, &argv);
   . . .
   MPI_Finalize();
   /* No MPI calls after this */
   . . .
   return 0;
}
```

MPI Basic (Blocking) Send

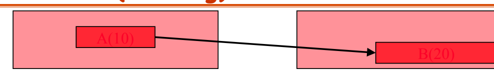
MPI_Send(A, 10, MPI_DOUBLE, 1, …)

MPI_Recv(B, 20, MPI_DOUBLE, 0, …)

MPI_SEND(start, count, datatype, dest, tag, comm)

- The message buffer is described by (**`start, count, datatype`**).
- The target process is specified by **`dest`**, which is the rank of the target process in the communicator specified by **`comm`**.
- When this function returns, the data has been delivered to the system and the buffer can be reused. The message may not have been received by the target process.

MPI Basic (Blocking) Receive



MPI_Send(A, 10, MPI_DOUBLE, 1, …) MPI_Recv(B, 20, MPI_DOUBLE, 0, …)

MPI_RECV(start, count, datatype, source, tag, comm, status)

- Waits until a matching (both **`source`** and **`tag`**) message is received from the system, and the buffer can be used
- **`source`** is rank in communicator specified by **`comm`**, or **`MPI_ANY_SOURCE`**
- **`tag`** is a tag to be matched on or **`MPI_ANY_TAG`**
- receiving fewer than **`count`** occurrences of **`datatype`** is OK, but receiving more is an error
- **`status`** contains further information (e.g. size of message)

MPI Datatypes

- The data in a message to send or receive is described by a triple (address, count, datatype), where
- An MPI datatype is recursively defined as:
  - predefined, corresponding to a data type from the language (e.g., MPI_INT, MPI_DOUBLE)
  - a contiguous array of MPI datatypes
  - a strided block of datatypes
  - an indexed array of blocks of datatypes
  - an arbitrary structure of datatypes
- There are MPI functions to construct custom datatypes, in particular ones for subarrays

A Simple MPI Program

```
#include "mpi.h"
#include <stdio.h>
int main( int argc, char *argv[])
{
  int rank, buf;
  MPI_Status status;
  MPI_Init(&argv, &argc);
  MPI_Comm_rank( MPI_COMM_WORLD, &rank );

  /* Process 0 sends and Process 1 receives */
  if (rank == 0) {
    buf = 123456;
    MPI_Send( &buf, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
  }
  else if (rank == 1) {
    MPI_Recv( &buf, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
              &status );
    printf( "Received %d\n", buf );
  }

  MPI_Finalize();
  return 0;
}
```

Six-Function MPI

- Most commonly used constructs
- A decade or more ago, almost all supercomputer programs only used these
  - MPI_Init
  - MPI_Finalize
  - MPI_Comm_Size
  - MPI_Comm_Rank
  - MPI_Send
  - MPI_Recv
- Also very useful
  - MPI_Reduce and other *collectives*
- Other features of MPI
  - Task parallel constructs
  - Optimized communication: non-blocking, one-sided

MPI_Reduce

```
int MPI_Reduce(
      void*          input_data_p   /* in  */,
      void*          output_data_p  /* out */,
      int            count          /* in  */,
      MPI_Datatype   datatype       /* in  */,
      MPI_Op         operator       /* in  */,
      int            dest_process   /* in  */,
      MPI_Comm       comm           /* in  */);
```

```
MPI_Reduce(&local_int, &total_int, 1, MPI_DOUBLE, MPI_SUM, 0,
      MPI_COMM_WORLD);
```

```
double local_x[N], sum[N];
. . .
MPI_Reduce(local_x, sum, N, MPI_DOUBLE, MPI_SUM, 0,
      MPI_COMM_WORLD);
```


Count 6s in MPI?