Introduction to Parallel and Distributed Processing

Introduction to MPI

Jaroslaw 'Jaric' Zola
http://www.jzola.org/

Suggested Reading

- Message Passing Interface forum
  http://www.mpi-forum.org/
- MPI tutorial https://computing.llnl.gov/tutorials/mpi/

Limitations of Shared Memory

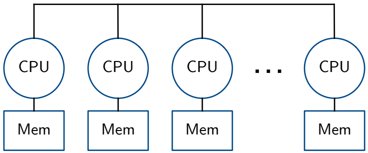
- Shared memory system is not scalable
- As we add more cores/processors, we put more memory pressure
- Memory subsystem is hard to expand (very slow progress)

Distributed Memory Systems

- Combine many individual “nodes” each with its own memory
- Enable them to communicate via fast interconnect
- When needed add more nodes (hence memory) and improve network

Distributed Memory Systems

- How to interconnect nodes – network topology?
- How to program such system – coordination, and communication



Network Topologies

- Overall, you want low-latency, high (bisection) bandwidth
- Static – nodes connect directly point-to-point
  2D/3D/5D mesh or torus, hypercube, etc.
- Dynamic – nodes connected via “switching” element
  often hierarchically, e.g. fat tree

Programming Distributed Memory

- We are always dealing with MIMD (usually SPMD)
- Overall two general strategies:
  - Explicitly describe communication patterns (e.g. MPI)
    tight control and great expressiveness, but laborious
  - Use higher-level model (e.g. Map/Reduce, Pregel)
    easy to use, lack of flexibility
  - Intermediate solutions (e.g. UPC, Chapel)
    dedicated DSLs for improved productivity

Message Passing Interface

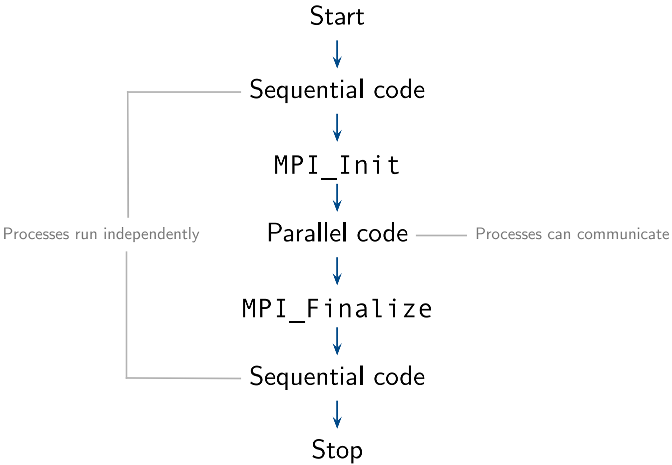
- Standard for parallel programming with messages, support for C/C++ and Fortran
- Meant to be portable, efficient and practical (but no fault tolerance)
- The workhorse of the HPC industry

MPI Support

- Multiple implementations, e.g. MPICH, OpenMPI, each HPC vendor has a version
- MPI standard specifies three components:
  - MPI library with `mpi.h` header
  - Compiler wrapper: `mpicc`, `mpicxx`
  - Runtime system with `mpiexec` command
- Example:

```
mpicxx -std=c++11 -O3 mpiapp.cpp -o mpiapp
mpiexec -machinefile nodeslist.txt -n 65536 mpiapp
```

MPI Program Flow



MPI Environment

- Processes are grouped into a communicator
- Each process within a communicator has a unique rank
- Ranks are integers from $0 \ldots p - 1$
- Initially all ranks belong to `MPI_COMM_WORLD`

MPI Environment

```
// ex01.cpp
#include <iostream>
#include <mpi.h>

int main(int argc, char* argv[]) {
  MPI_Init(&argc, &argv);

  int rank, size;

  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  std::cout << rank << " " << size << std::endl;

  return MPI_Finalize();
} // main
```

Point to Point Communication

- The most basic form of messaging: one rank sends message to some other
- Different variants possible, e.g. blocking/non-blocking

Point to Point Communication

```
// ex02.cpp
#include <mpi.h>

int main(int argc, char* argv[]) {
  MPI_Init(&argc, &argv);

  int rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  const int N = 32;
  int tab[N];

  if (rank == 0) {
    tab[N - 1] = 13;
    MPI_Send(tab, N, MPI_INT, 1, 111, MPI_COMM_WORLD);
  } else if (rank == 1) {
    MPI_Status stat;
    MPI_Recv(tab, N, MPI_INT, 0, 111, MPI_COMM_WORLD, &stat);
  }

  return MPI_Finalize();
} // main
```

Message

- Data part: buffer, count, type
- Message envelope: source, destination, tag, communicator

```
MPI_Send(tab, N, MPI_INT, 1, 111, MPI_COMM_WORLD);
MPI_Recv(tab, N, MPI_INT, 0, 111, MPI_COMM_WORLD, &stat);
```

- Basic data types `MPI_CHAR`, `MPI_INT`, `MPI_DOUBLE`,
  new types can be created when needed

Point to Point Mechanics

- `MPI_Send` and `MPI_Recv` are blocking!
  Will not finish until message is sent/received
- Each send operation must be matched by receive operation
- Messages are selected for receiving based
  on their envelopes
- Messages with the same envelope follow the order
  in which they were sent

Point to Point

Incorrect:

```
// ex04.cpp
#include <mpi.h>

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

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  int x;

  if (rank == 0) {
    x = 13;
    MPI_Send(&x, 1, MPI_INT, 1, 111, MPI_COMM_WORLD);
  } else if (rank == 1) {
    MPI_Status stat;
    MPI_Recv(&x, 1, MPI_INT, 0, 222, MPI_COMM_WORLD, &stat);
  }

  return MPI_Finalize();
} // main
```

Point to Point

Incorrect:

```
// ex05.cpp
#include <mpi.h>

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

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  int x[65536];
  float y[65536];

  if (rank == 0) {
    x[13] = 13;
    y[13] = 0.13;
    MPI_Send(x, 65536, MPI_INT, 1, 111, MPI_COMM_WORLD);
    MPI_Send(y, 65536, MPI_FLOAT, 1, 222, MPI_COMM_WORLD);
  } else if (rank == 1) {
    MPI_Status stat;
    MPI_Recv(y, 65536, MPI_FLOAT, 0, 222, MPI_COMM_WORLD, &stat);
    MPI_Recv(x, 65536, MPI_INT, 0, 111, MPI_COMM_WORLD, &stat);
  }

  return MPI_Finalize();
} // main
```

Useful Trick

- MPI_ANY_TAG and MPI_ANY_SOURCE in MPI_Recv can receive any message from any processor
- MPI_Status stores envelope of the received message

```
int x;
MPI_Status stat;
MPI_Recv(&x, 1, MPI_INT, MPI_ANY_SOURCE, 111,
         MPI_COMM_WORLD, &stat);
int source = stat.MPI_SOURCE;
int tag = stat.MPI_TAG;
```

Blocking Send-Recv

- `MPI_Sendrecv` – handy for cyclic communication like shifts

```cpp
// ex06.cpp
#include <mpi.h>

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

  MPI_Init(&argc, &argv);

  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  int x = rank;
  int y = 0;

  MPI_Status stat;
  for (int i = 0; i < size - 1; ++i) {
    MPI_Sendrecv(&x, 1, MPI_INT, (size + rank - 1) % size, 111,
                 &y, 1, MPI_INT, (rank + 1) % size, 111,
                 MPI_COMM_WORLD, &stat);
    std::swap(x, y);
  }

  return MPI_Finalize();
} // main
```

For Fun

- Implement passing a message in a ring: processor with rank 0 sends to 1, which forwards to 2, which forwards..., and finally p-1 forwards to 0