Introduction to Parallel and Distributed Processing

One-sided communication

Jaroslaw 'Jaric' Zola

http://www.jzola.org/

Problem with Point-to-Point Communication

- Point-to-point communication in MPI (send/recv) requires that both ranks participate
- Hence, the delay on one end slows down the rank on the other end
- This becomes problematic, especially when dealing with irregular communication patterns

One-sided Communication Idea (RMA)

- Each rank exposes part of its memory to other ranks
- Other ranks can directly read/write to this memory
- No (explicit) synchronization required!

One-sided Communication Advantage

- Certain communication patterns much easier to express (e.g. recall shifting)
- Modern networks (e.g. IB) support RDMA, and hence provide great performance

Exposing Memory

- Ranks collectively declare window: memory that will be accessible to all ranks in the communicator
- Memory must be properly aligned (helper API provided)

```
#include <mpi.h>

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

  int* buf;
  MPI_Alloc_mem(1024 * sizeof(int), MPI_INFO_NULL, &buf);

  MPI_Win win;
  MPI_Win_create(buf, 1024 * sizeof(int), sizeof(int),
                 MPI_INFO_NULL, MPI_COMM_WORLD, &win);

  // ...

  MPI_Win_free(&win);
  MPI_Free_mem(buf);

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

Moving Data

- Primitives for putting, getting and modifying data in a window
- We talk about origin and target for data
  (instead of sender/receiver)
- Some important caveats
  - No guaranteed ordering of basic operations
  - Basic concurrent writes to the same location undefined

Getting Data

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

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

  int* buf;
  MPI_Alloc_mem(1024 * sizeof(int), MPI_INFO_NULL, &buf);

  MPI_Win win;
  MPI_Win_create(buf, 1024 * sizeof(int), sizeof(int),
                 MPI_INFO_NULL, MPI_COMM_WORLD, &win);

  int x[2];
  MPI_Win_lock(MPI_LOCK_EXCLUSIVE, 0, 0, win);
  // get from buf + 1021 at target 0
  MPI_Get(x, 2, MPI_INT, 0, 1021, 2, MPI_INT, win);
  MPI_Win_unlock(0, win);

  MPI_Win_free(&win);
  MPI_Free_mem(buf);

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

Putting Data

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

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

  int* buf;
  MPI_Alloc_mem(1024 * sizeof(int), MPI_INFO_NULL, &buf);

  MPI_Win win;
  MPI_Win_create(buf, 1024 * sizeof(int), sizeof(int),
                 MPI_INFO_NULL, MPI_COMM_WORLD, &win);

  int x[2];
  MPI_Win_lock(MPI_LOCK_EXCLUSIVE, 0, 0, win);
  // put from x into buf + 1021 at target 0
  MPI_Put(x, 2, MPI_INT, 0, 1021, 2, MPI_INT, win);
  MPI_Win_unlock(0, win);

  MPI_Win_free(&win);
  MPI_Free_mem(buf);

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

Other Goodies

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

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

  int* buf;
  MPI_Alloc_mem(1024 * sizeof(int), MPI_INFO_NULL, &buf);

  MPI_Win win;
  MPI_Win_create(buf, 1024 * sizeof(int), sizeof(int),
                 MPI_INFO_NULL, MPI_COMM_WORLD, &win);

  MPI_Win_fence(0, win);
  MPI_Accumulate(buf, 1024, MPI_INT, 0, 0, 1024, MPI_INT, MPI_SUM, win);
  MPI_Win_fence(0, win);

  MPI_Win_free(&win);
  MPI_Free_mem(buf);

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

For Fun

- Implement a simple distributed hash table using one-sided communication.