SPECIALIZED PROJECT REPORT

STUDYING AND DEVELOPING DISTRIBUTED BARRIER ALGORITHMS USING THE HYBRID PROGRAMMING MODEL COMBINING MPI-3 AND C++11

Phạm Võ Quang Minh - 2111762

Vietnam National University - Ho Chi Minh City University of Technology

2025-01-02

Outline

Outline

1.1 Motivation

1.1.1 HPC and its Applications

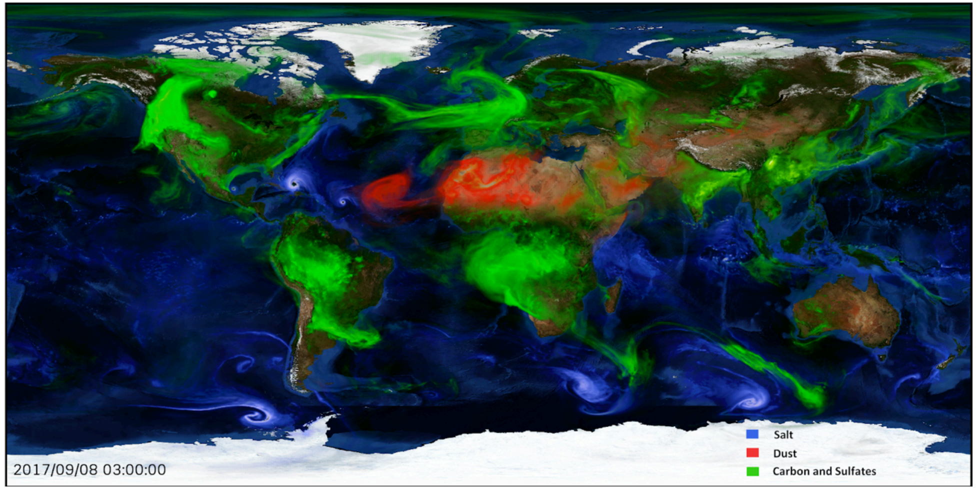


Figure 1: Weather Simulation [1]

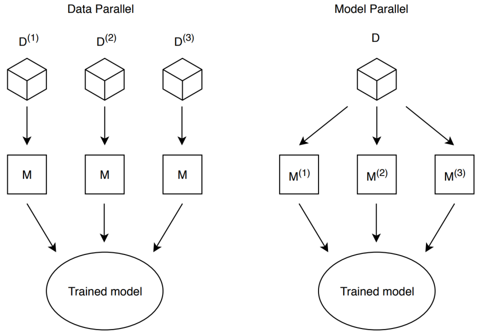


Fig. 2. Parallelism in Distributed Machine Learning. Data parallelism trains multiple instances of the same model on different subsets of the training dataset, while model parallelism distributes parallel paths of a single model to multiple nodes.

Figure 2: Distributed Machine Learning [2]

[1] "NASA Global Weather Forecasting." Accessed: Jan. 01, 2025. [Online]. Available: https://www.nccs.nasa.gov/sci-tech/case-studies/nasa-global-weather-forecasting

[2] "A Survey on Distributed Machine Learning," *ACM Computing Surveys*, vol. 53, doi: 10.1145/3377454.

1.1 Motivation

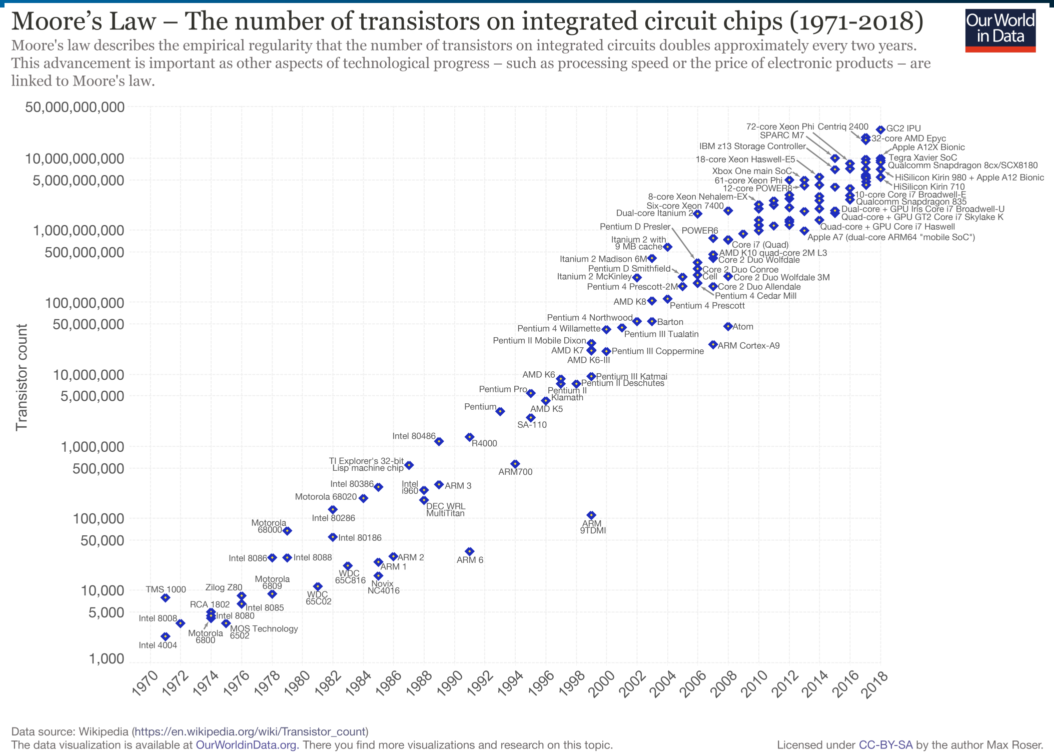


Figure 3: Moore’s Law [1]

[1]“Moore's Law Transistor Count 1971-2018.” Accessed: Jan. 01, 2025. [Online]. Available: https://upload.wikimedia.org/wikipedia/commons/8/8b/Moore%27s_Law_Transistor_Count_1971-2018.png

1.1 Motivation

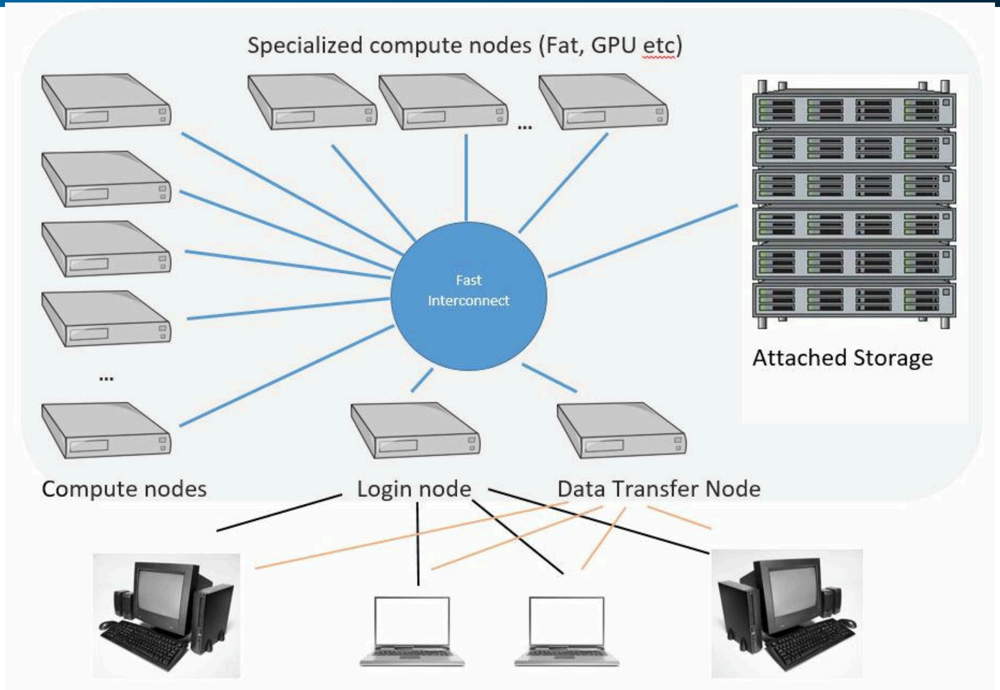


Figure 4: Multiple Computing Nodes connect to each other to form a HPC Cluster [1]

- Nodes are connected via a very fast network (Infiniband, Ethernet, etc.)

[1]"What is an HPC Cluster?." Accessed: Jan. 01, 2025. [Online]. Available: https://www.hpc.iastate.edu/guides/introduction-to-hpc-clusters/what-is-an-hpc-cluster

1.1 Motivation

1.1.2 The Paper

- Quaranta et al [1] proposed to combine MPI-3 and C++11 (hybrid model)
- Only implements a simple barrier algorithm using the hybrid model

→ Implement and benchmark more complex barrier algorithms using the hybrid model

[1] L. Quaranta and L. Maddegedara, "A Novel MPI+MPI Hybrid Approach Combining MPI-3 Shared Memory Windows and C11/C++11 Memory Model," *Journal of Parallel and Distributed Computing*, vol. 157, pp. 125–144, Nov. 2021, doi: 10.1016/j.jpdc.2021.06.008.

1.2 Objectives

- Explore key concepts in high-performance computing (HPC) ✓
- Research and familiarize with the MPI-3 and C++11 programming model ✓
- Research how to combine MPI-3 and C++11 for a hybrid programming model ✓
- Research about many barrier algorithms ✓
- Implement a simple barrier algorithm using MPI-3 ✓

Outline

2.1 Barrier Algorithm

What is a Barrier Algorithm?

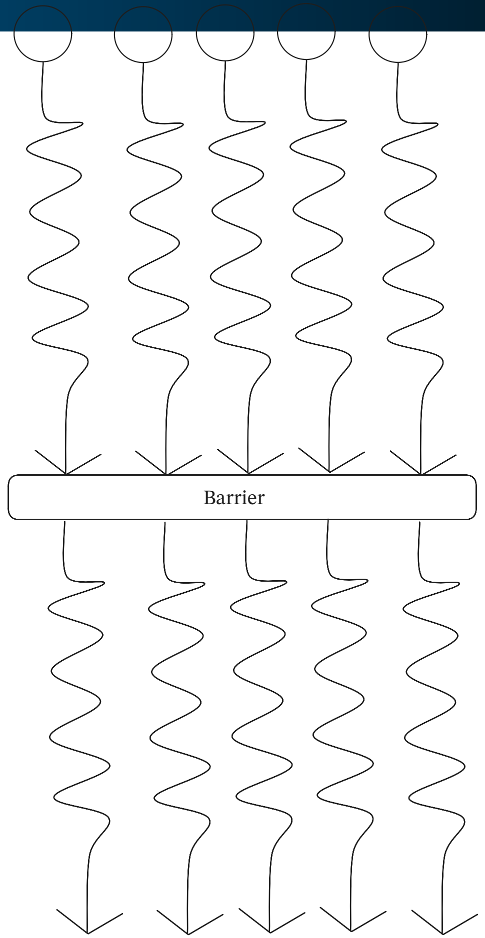
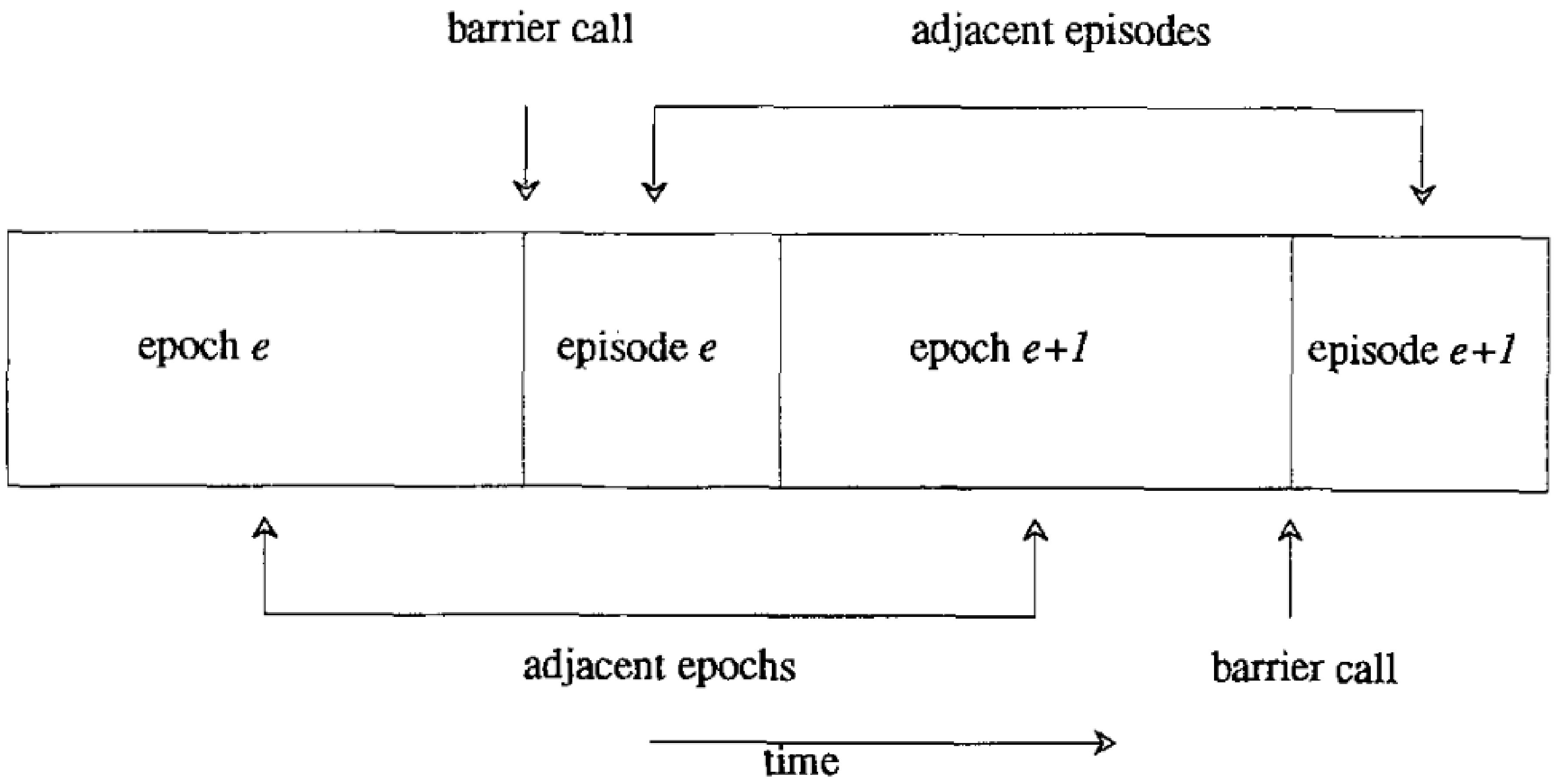


Figure 5: Barrier Algorithm

2.1 Barrier Algorithm



2.2 MPI-3

2.2.1 Message Passing Interface (MPI)

- MPI is a standard for message-passing between nodes in a distributed system
- MPI is optimized for communication between nodes
- Multiple implementations of MPI are available (OpenMPI, MPICH, MVAPICH, etc.)
- Multiple programming languages support MPI (C, Fortran, etc.)

2.2 MPI-3

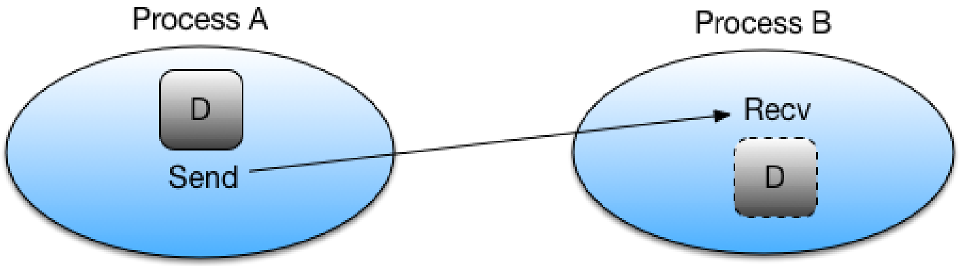


Figure 6: Live program communicating with each other using MPI [1]

[1]"Introduction to MPI - MPI Send and Receive." Accessed: Jan. 01, 2025. [Online]. Available: https://pdc-support.github.io/introduction-to-mpi/03-mpi_send_recv/index.html

2.2 MPI-3

Traditional Message Passing

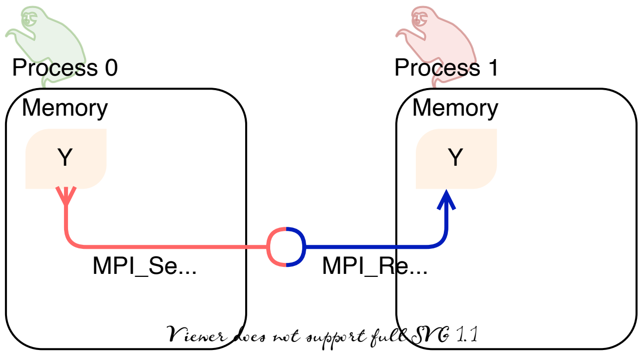


Figure 7: Point-to-point communication[1]

- Point-to-point
- Explicit send and receive

[1]"Introduction to MPI - MPI Send and Receive." Accessed: Jan. 01, 2025. [Online]. Available: https://pdc-support.github.io/introduction-to-mpi/03-mpi_send_recv/index.html

2.2 MPI-3

One-sided Communication

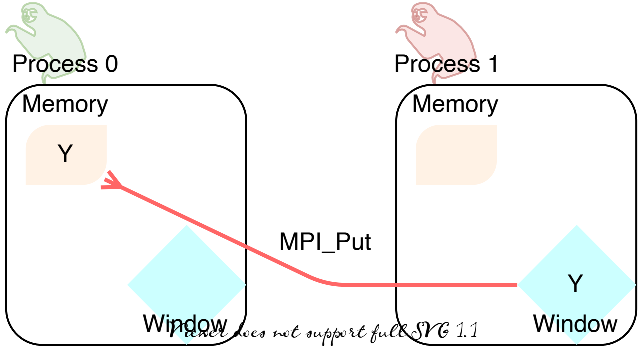


Figure 8: One-sided Communication[2]

- Remote Memory Access
- Handshake is implicit

2.2.2 One-sided Communication

- Introduced in MPI-2
- Share mechanism:
  - Declare a window of memory to be shared
  - read/write without explicit send/receive

[2] "Introduction to MPI - MPI Send and Receive." Accessed: Jan. 01, 2025. [Online]. Available: https://pdc-support.github.io/introduction-to-mpi/03-mpi_send_recv/index.html

2.2 MPI-3

- Simple operations:
  - MPI_Put
  - MPI_Get
  - MPI_Accumulate
- Atomic operations:
  - MPI_Get_accumulate
  - MPI_Fetch_and_op
  - MPI_Compare_and_swap

2.2 MPI-3

2.2.3 New Features in MPI-3

Separate Memory

Unified Memory

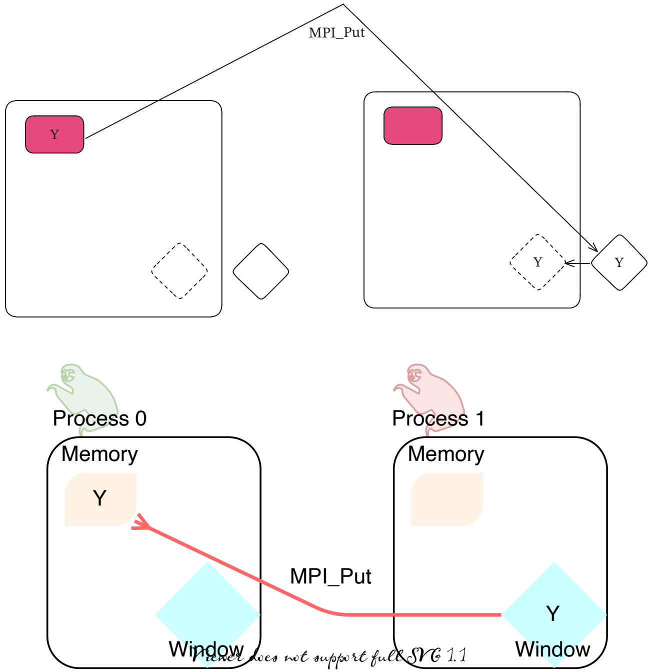


Figure 10: One-sided Communication[1]

[1]"Introduction to MPI - MPI Send and Receive." Accessed: Jan. 01, 2025. [Online]. Available: https://pdc-support.github.io/introduction-to-mpi/03-mpi_send_recv/index.html

2.2 MPI-3

2.2.4 C++11

Supports Multithreading

→ Allows Shared memory programming model instead of Distributed memory programming model

in the STL

→ No reliance on 3rd party libraries for multithreading programming

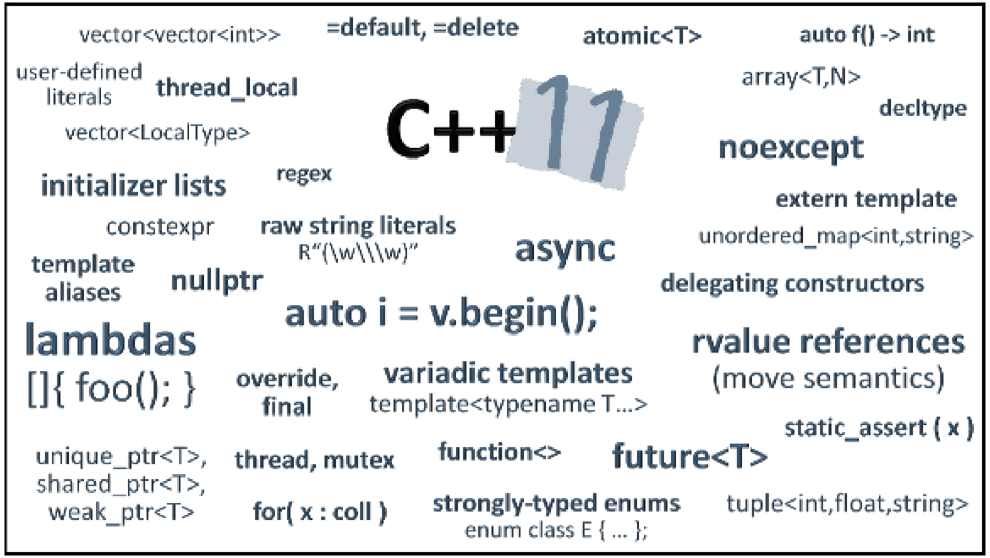


Figure 11: Features introduced in C++11 [1]

[1] "Technical CPP Blog." Accessed: Jan. 01, 2025. [Online]. Available: https://www.cyberplusindia.com/blog/index.php/category/technical/cpp/

Outline

3.1 The Paper

- Quaranta et al[1] proposed a hybrid model of MPI-3 and C++11
- Showed potential of combining MPI-3 and C++11 for a synchronization implementation
- Potential reduction in communication overhead

[1]L. Quaranta and L. Maddegedara, “A Novel MPI+MPI Hybrid Approach Combining MPI-3 Shared Memory Windows and C11/C++11 Memory Model,” *Journal of Parallel and Distributed Computing*, vol. 157, pp. 125–144, Nov. 2021, doi: 10.1016/j.jpdc.2021.06.008.

3.2 Barrier Algorithm Selection

3 school of thoughts:

- Linear
- Tree-based
- Butterfly

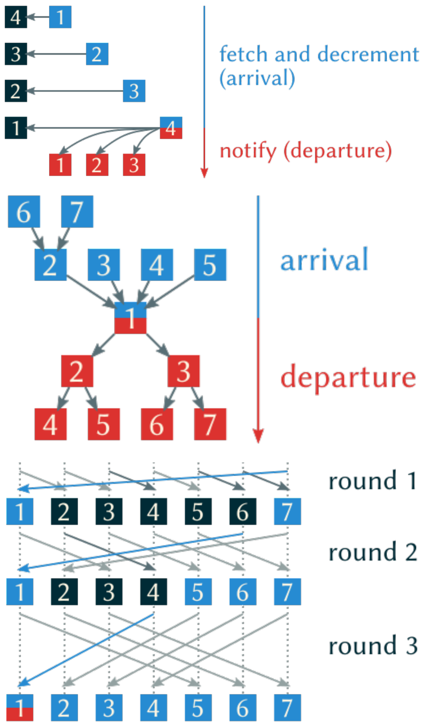


Figure 12: Barrier Algorithm Selection

Outline

4.1 Brook 2 process algorithm

- The basis of many butterfly barrier algorithms
- Barrier for 2 threads of execution

4.1 Brook 2 process algorithm

Step	Process 1	Process 2
1	while SetByProcess1 do wait;	while SetByProcess2 do wait;
2	SetByProcess1 := true;	SetByProcess2 := true;
3	while not SetByProcess2 do wait;	while not SetByProcess1 do wait;
4	SetByProcess2 := false;	SetByProcess1 := false;

4.2 Implementation using RMA Operation

4.2.1 Memory Setup

In Brook's shared memory model:

- Two variables are shared between processes
- Both processes can directly access these variables
- No explicit initialization needed beyond variable declaration

In my MPI implementation:

- Each process owns a local buffer (`exposed_buffer`)
- MPI Window creation establishes remote memory access capability
- Explicit initialization required through `MPI_Win_create`

4.2 Implementation using RMA Operation

4.2.2 Step 1: Wait for Reset

Brook's model:

```
while SetByProcess1 do wait;  // Direct memory access
```

My implementation:

```
while (exposed_buffer) {
    // Busy-waiting loop
}
```

The implementation directly checks the local buffer since each process owns its buffer. This is simpler than Brook's model as we're checking local memory.

4.2 Implementation using RMA Operation

4.2.3 Step 2: Set Flag

Brook's model:

```
SetByProcess1 := true;  // Direct shared memory write
```

My implementation:

```
exposed_buffer = true;  // Local memory write
```

Similar to Step 1, we're working with local memory, making this operation straightforward.

4.2 Implementation using RMA Operation

4.2.4 Step 3: Wait for Other Process

Brook's model:

```
while not SetByProcess2 do wait;  // Direct memory read
```

My implementation:

```
while (!flag_from_other_process) {
    MPI_Win_lock_all(0, win_buffer_handler);
    MPI_Get_accumulate(&flag_from_other_process, 0, MPI_CXX_BOOL,
                       &flag_from_other_process, 1, MPI_CXX_BOOL,
                       target_rank, 0, 1, MPI_CXX_BOOL,
                       MPI_NO_OP, win_buffer_handler);
    MPI_Win_flush_all(win_buffer_handler);
    MPI_Win_unlock_all(win_buffer_handler);
}
```

4.2 Implementation using RMA Operation

4.2.5 Step 4: Reset Other's Flag

Brook's model:

```
SetByProcess2 := false;  // Direct shared memory write
```

My implementation:

```
bool false_value{false};
MPI_Win_lock_all(0, win_buffer_handler);
MPI_Accumulate(&false_value, 1, MPI_CXX_BOOL,
               target_rank, 0, 1, MPI_CXX_BOOL,
               MPI_REPLACE, win_buffer_handler);
MPI_Win_flush(target_rank, win_buffer_handler);
MPI_Win_unlock_all(win_buffer_handler);
```

4.3 Preliminary Result

4.3.1 Test Environment

- Hardware: Apple M1 chip
- Compiler: MPICH's mpic++ compiler
- MPI Implementation: MPICH
- Compilation flags: -std=c++11
- Operating System: macOS

4.3 Preliminary Result

```c
int main(int argc, char **argv) {
  // init the mpi world
  MPI_Init(&argc, &argv);
  MPI_Comm comm = MPI_COMM_WORLD;
  int rank;
  MPI_Comm_rank(comm, &rank);

  brook_2_proc(comm);
  brook_2_proc(comm);
  brook_2_proc(comm);

  printf("Process %d: reached destination\n", rank);
  return MPI_Finalize();
}
```

Listing 1: Main function to test brook_2_proc function

4.3 Preliminary Result

4.3.2 Compilation and Execution

The program was compiled using the following command:

```
mpic++ -std=c++11 src/brook.cpp -o brook
```

Execution was performed using MPICH's mpirun with two processes:

```
mpirun -np 2 ./brook
```

Outline

5.1 Accomplishments

- Explore key concepts in high-performance computing (HPC) ✓
- Research and familiarize with the MPI-3 and C++11 programming model ✓
- Research how to combine MPI-3 and C++11 for a hybrid programming model ✓
- Research about many barrier algorithms ✓
- Implement a simple barrier algorithm using MPI-3 ✓

5.2 Future Works

5.2.1 Plan

- Research C++ and MPI Hybrid model
- Implement more algorithms
- Implement barrier algorithms on C++ shared memory model
- Testing of proposed algorithms on existing computational clusters
- Performance benchmarking to identify and select optimal barrier synchronization strategies
- Integration and contribution to existing applications

5.2 Future Works