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

2025-01-03

Outline

Outline

1.1 Motivation

1.1.1 HPC and its Applications

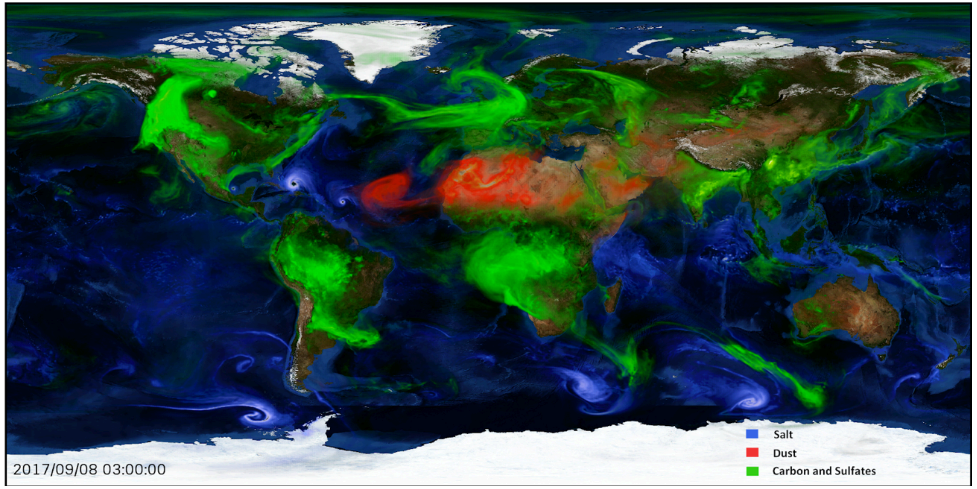


Figure 1: Weather Simulation [1]

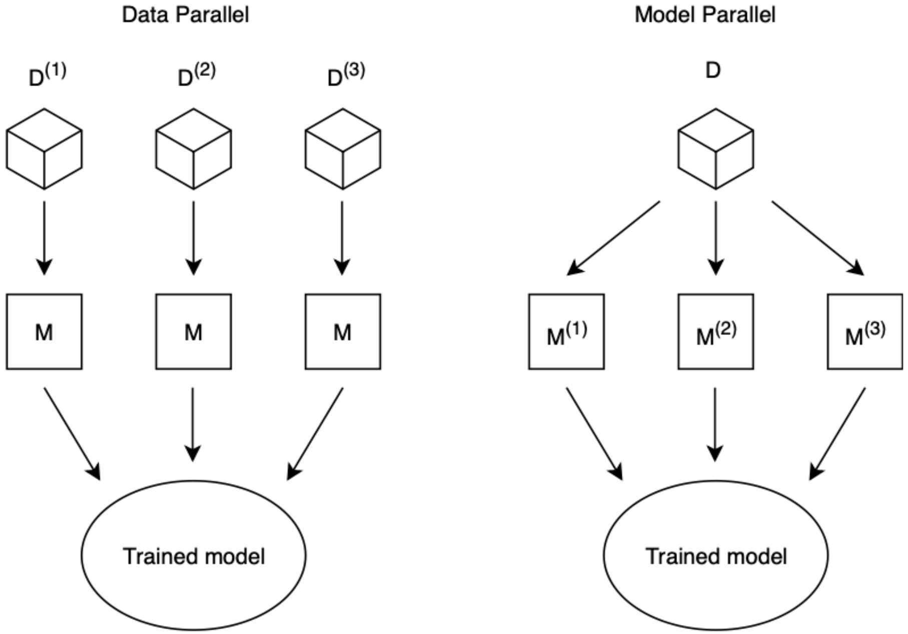


Figure 2: Distributed Machine Learning [2]

[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

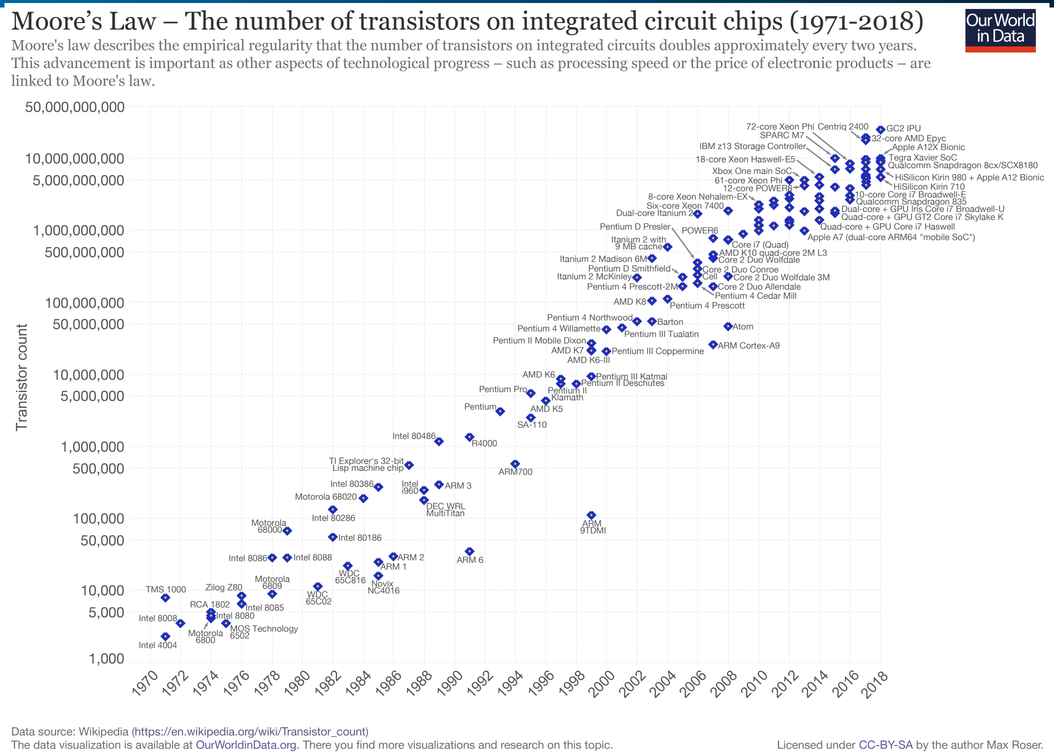


Figure 3: Moore's Law [1]

[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

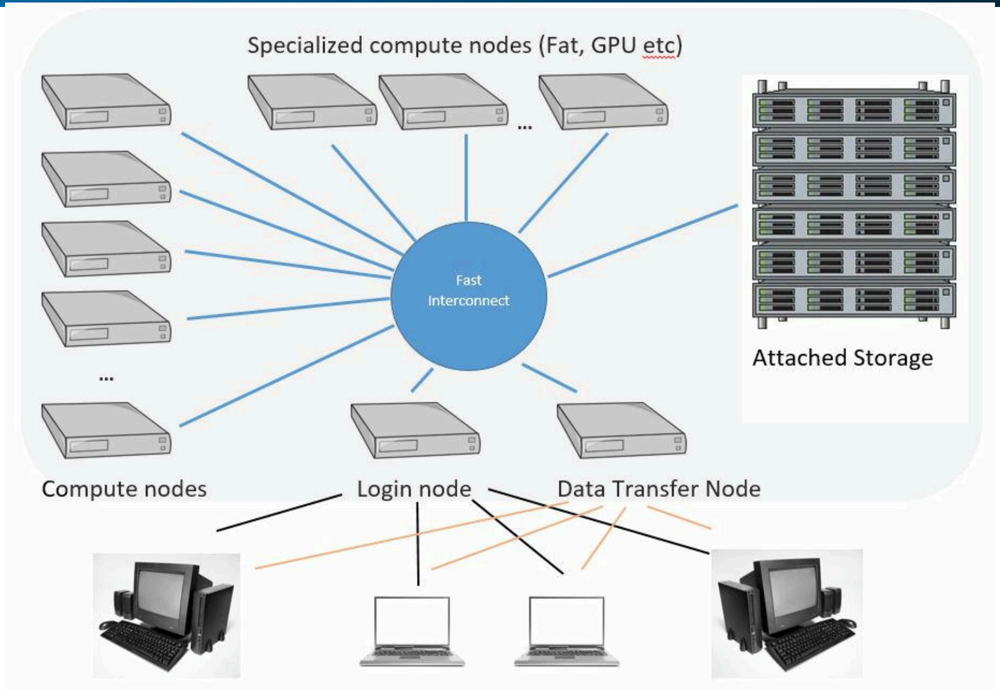


Figure 4: Multiple Computing Nodes connect to each other to form a HPC Cluster [1]

- 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

What is a Barrier Algorithm?

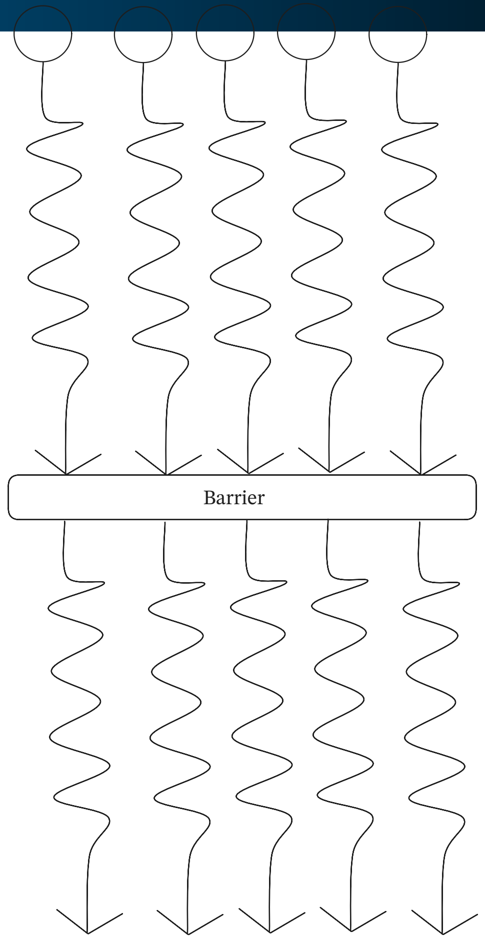


Figure 5: Barrier Algorithm

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

- Survey existing barrier synchronization algorithms
- Implements variants of barrier synchronization algorithms in the hybrid programming model.
- **Compare the performance of the implemented barrier algorithms with existing algorithms.**
- Propose methods to deploy selected barrier algorithms on the current HPC system.

1.3 Scope

- Explore key concepts in high-performance computing (HPC).
- Study MPI-3 and its programming model.
- Investigate C++11 threading and concurrency features.
- Research the hybrid programming model combining MPI-3 and C++11.
- Survey existing barrier synchronization algorithms.
- Implement a simple barrier synchronization algorithm with the hybrid model.

Outline

2.1 MPI-3

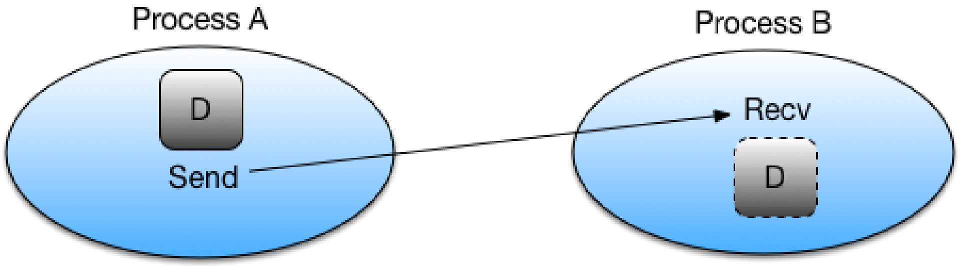


Figure 6: Live program communicating with each other using MPI [1]

[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

2.1 MPI-3

Traditional Message Passing

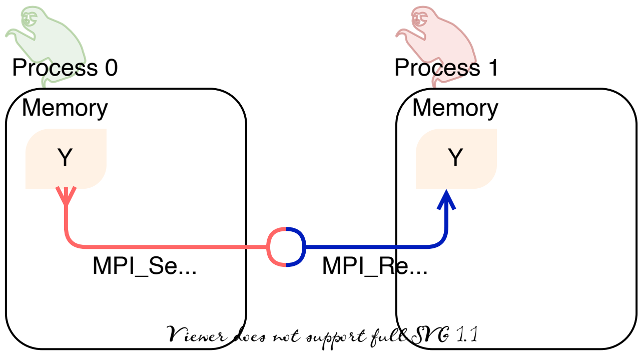


Figure 7: Point-to-point communication[1]

- 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

2.1 MPI-3

One-sided Communication

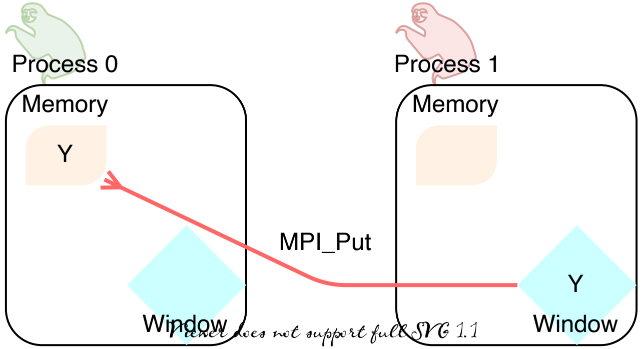


Figure 8: One-sided Communication[2]

- Remote Memory Access
- Handshake is implicit

2.1.1 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

2.1 MPI-3

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

2.1 MPI-3

2.1.2 New Features in MPI-3

Separate Memory

Unified Memory

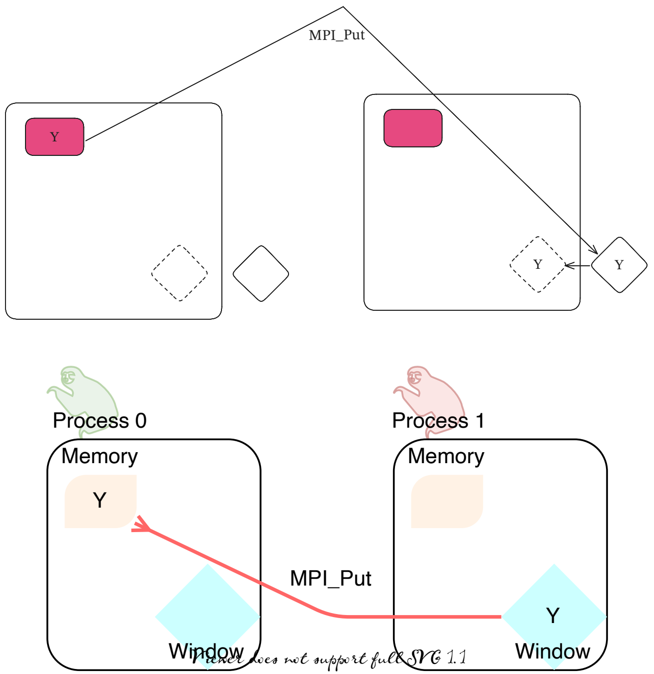


Figure 10: One-sided Communication[1]

[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 C++11

Supports Multithreading

$\longrightarrow$ Allows Shared memory programming model instead of Distributed memory programming model

in the STL

$\longrightarrow$ No reliance on 3rd party libraries for multithreading programming

2.2 C++11

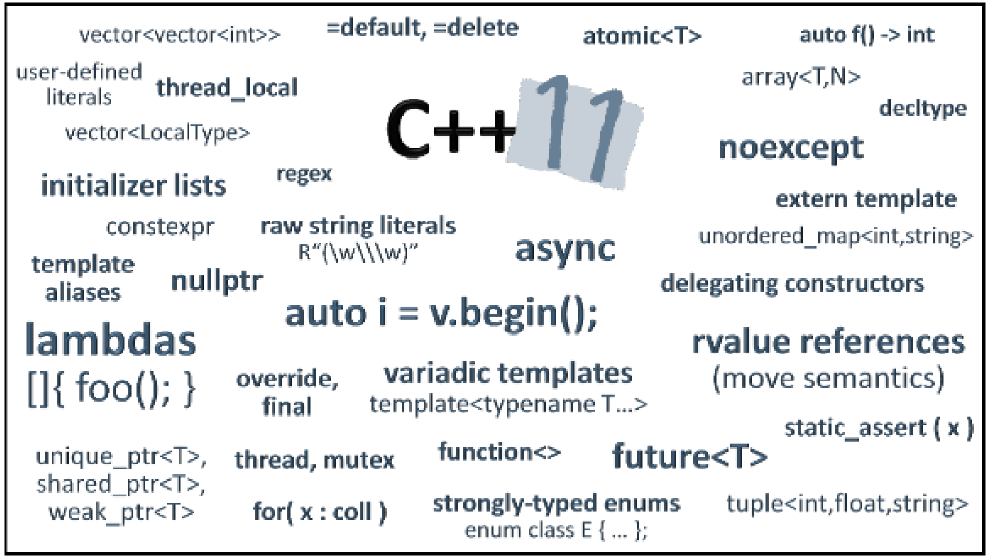


Figure 11: Features introduced in C++11 [1]

[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

2 schools of thoughts:

- Linear
- Butterfly

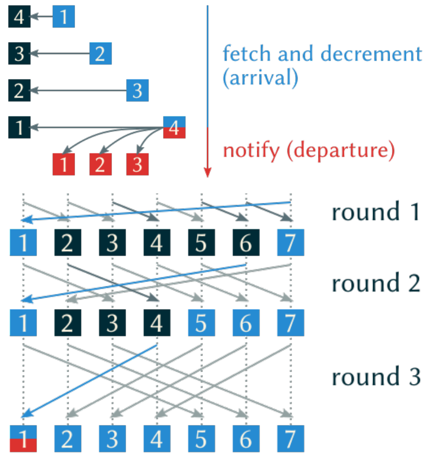


Figure 12: Barrier Algorithm Selection

Outline

4.1 Brook 2 process algorithm

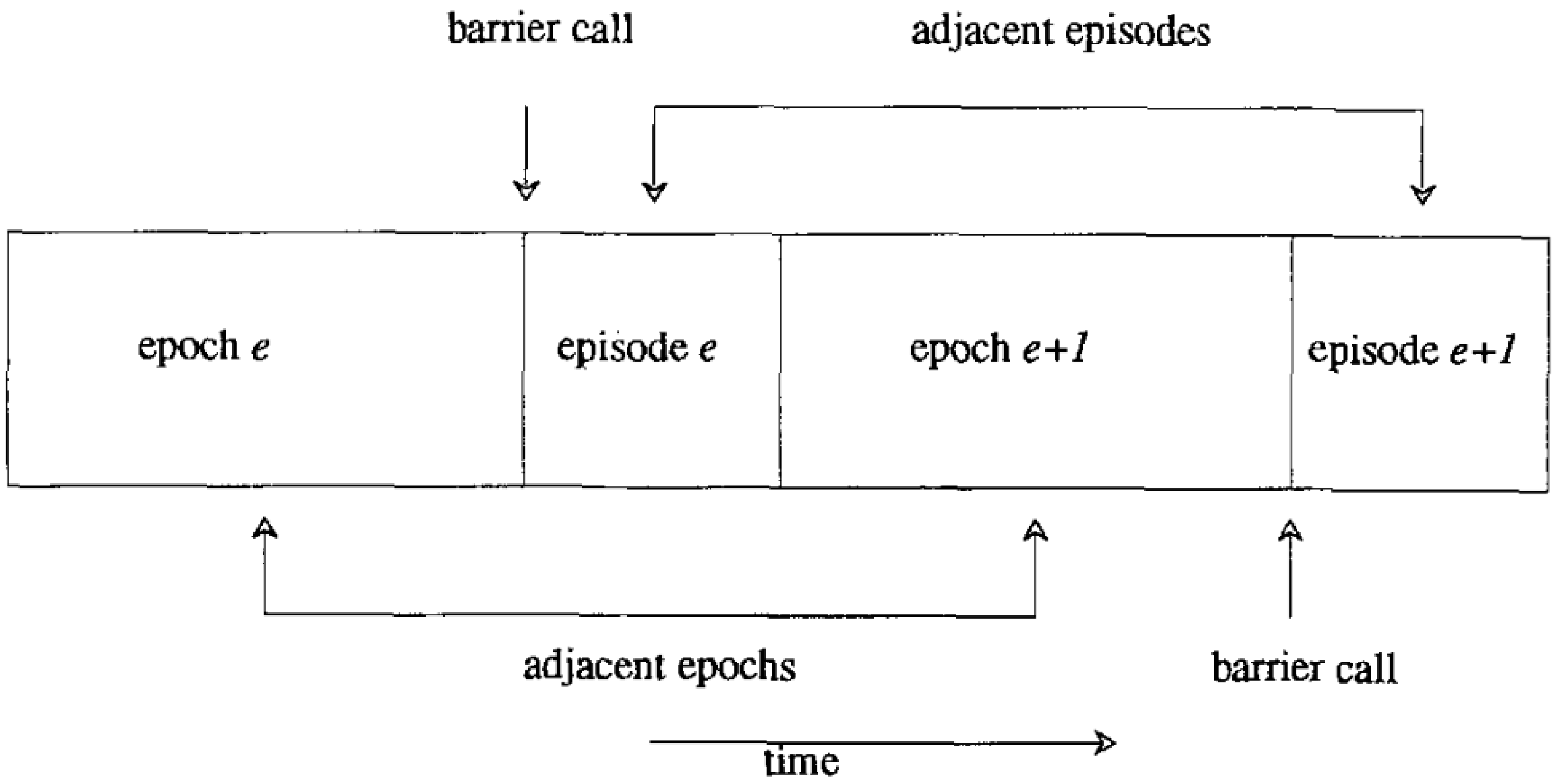
- The basis of many butterfly barrier algorithms
- Barrier for 2 threads of execution

BROOK-BARRIER-ALGORITHM

---

procedure Brook's-Barrier(SetByMyProcess, SetByTargetProcess)
  while SetByMyProcess **is** true **do** wait
  SetByMyProcess $\leftarrow$ true
  while SetByTargetProcess **is** false **do** wait
  SetByTargetProcess $\leftarrow$ false
end procedure

4.1 Brook 2 process algorithm



4.1 Brook 2 process algorithm

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

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

DISTRIBUTED-MEM-BROOK-BARRIER

```
procedure Brook's-Barrier(exposed_flag, win, target_rank)
    while exposed_flag is true do wait //step 1
    exposed_flag ← true //step 2
    //step 3
    while target_flag is false do wait
        MPI_WIN_LOCK(win)
        MPI_GET_ACCUMULATE(&target_flag, 0, BOOL, &target_flag, 1, BOOL,
          target_rank, 0, 1, BOOL, MPI_NO_OP, win);
        MPI_WIN_FLUSH(win)
        MPI_WIN_UNLOCK(win)
```

4.2 Implementation using RMA Operation

DISTRIBUTED-MEM-BROOK-BARRIER

```
    end while
    //step 4
    false_value ← false
    MPI_WIN_LOCK(win)
    MPI_ACCUMULATE(&false_value, 1, BOOL, target_rank, 0, 1, BOOL,
      MPI_REPLACE, win);
    MPI_WIN_FLUSH(win)
    MPI_WIN_UNLOCK(win)
end procedure
```

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

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

  // get number of ranks
  int n_ranks;
  MPI_Comm_size(comm, &n_ranks);
```

4.3 Preliminary Result

```
// get current rank
int rank;
MPI_Comm_rank(comm, &rank);

if (n_ranks != 2) {
  if (rank == 0) {
    fprintf(stderr, "This program requires exactly 2 processes.\n");
  }
  MPI_Abort(comm, 1);
}

// set the target
int target_rank = 1 - rank;

// create a window
bool exposed_bool{false};
```

4.3 Preliminary Result

```
  MPI_Win win_buffer_handler;
  MPI_Win_create(&exposed_bool, sizeof(bool), sizeof(bool), MPI_INFO_NULL,
comm,
                 &win_buffer_handler);

  barrier_brook(exposed_bool, win_buffer_handler, target_rank);
  barrier_brook(exposed_bool, win_buffer_handler, target_rank);
  barrier_brook(exposed_bool, win_buffer_handler, target_rank);
  barrier_brook(exposed_bool, win_buffer_handler, target_rank);

  printf("Process %d: reached destination\n", rank);

  return MPI_Finalize();
}
```

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

- Studied and familiarized myself with MPI and its One-Sided Communication techniques
- Explored C++11 threading and concurrency features
- Successfully implemented the two-process Brooks barrier algorithm utilizing MPI's Remote Memory Access (RMA) Operations

5.2 Challenges and Learnings

5.2.1 Initial Challenges:

- Faced a steep learning curve with parallel programming concepts.
- Extensive effort required to comprehend the MPI standard documentation and its historical context.

5.2.2 Key Learnings:

- Gained a robust understanding of memory models and synchronization techniques.
- Developed insight into one-sided communication mechanics in MPI.

5.2.3 Adaptation Effort:

- Adapting Brook's barrier algorithm to distributed memory was straightforward but required a comprehensive system understanding.

5.2 Challenges and Learnings

5.2.4 Outcomes:

- Established a solid foundation in distributed memory systems and synchronization strategies.
- Successfully adapted the two-process barrier algorithm to distributed memory.
- Future Opportunities:
- Yet to implement a fully functional barrier algorithm for hybrid memory models.
- Room for further exploration and development in this domain.

5.3 Future Works

5.3.1 Plan

- Understanding of C++ and MPI Memory model to implement C++ and MPI Hybrid model
- Adapting and Implementing barrier algorithms using MPI's One-Sided Communication
- Testing of proposed algorithms on existing clusters
- Performance benchmarking

5.3 Future Works