

**VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING**



LTW

BTL LTW

Major: Computer Science

THESIS COMMITTEE: 12
MEMBER SECRETARY: NGUYỄN QUANG HÙNG

SUPERVISORS: THOẠI NAM
DIỆP THANH ĐĂNG

—o0o—

STUDENTS: PHẠM VÕ QUANG MINH -
2111762
VÕ KIỀU MY - 6969696

HCMC, 04/2025

Supervisor's Signature

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri.

Ho Chi Minh City, 05/04/2025 Ho Chi Minh City, 05/04/2025

Assoc. Prof. Thoai Nam

MS. Diep Thanh Dang

Disclaimers

I hereby declare that this specialized project is the result of my own research and experiments. It has not been copied from any other sources. All content presented and implemented within this document reflects my own hard work, dedication, and honesty, conducted under the guidance of my supervisors, Mr. Thoại Nam and Mr. Diệp Thanh Đăng, from the Faculty of Computer Science and Engineering, Ho Chi Minh City University of Technology.

All data, references, and sources have been legally cited and are explicitly mentioned in the footnotes and references section.

I accept full responsibility for the accuracy of the claims and content in this specialized project and am willing to face any consequences or penalties should any violations or misconduct be identified.

Acknowledgements

First, I want to express my gratitude to my school and teachers for providing me with a strong foundation in the field of computer science and software engineering. Their guidance has helped me refine my skills and develop my mindset as an engineer and scientist. These experiences have not only supported me in this specialized project but will also guide me throughout my lifelong journey.

I am especially thankful to my mother and brother for their unwavering support during both good times and bad. They have always been there for me, offering emotional and financial support and being the pillars I can rely on when times are tough.

I would also like to extend my deepest gratitude to Mr. Diệp Thanh Đăng for inspiring me and guiding me toward this research topic. His mentorship helped me discover my true passion and led me to choose this meaningful topic for my thesis.

Contents

Chapter I:	Introduction	8
1.1	Motivation	9
1.2	Objectives	9
1.3	Scope	9
1.4	Project report structure	9
Chapter II:	Background	11
2.1	Message Passing Interface (MPI)	12
2.2	C++11 Multithreading	12
Chapter III:	Related Works	13
3.1	Barrier Synchronization Algorithm Selection	14
3.2	Hybrid Parallelization Approaches	14
Chapter IV:	Adaptation from shared memory to distributed memory	15
4.1	Brook Algorithm	16
4.2	My proposed implementation of Brook's algorithm	16
4.2.1	Notes on the adapted algorithm	17
Chapter V:	Preliminary Results	18
5.1	Test Environment	19
5.2	Test Implementation	19
5.3	Compilation and Execution	19
5.4	Results	19
Chapter VI:	Conclusions and Future Works	20
6.1	Accomplishments This Semester	21
6.2	Challenges and Learnings	21
6.3	Planned Research Trajectory	21
References	22

List of Listings

Listing 1	Barrier signature	17
Listing 2	My adaptation of Brook's two-process barrier using MPI primitives	17
Listing 3	Ran the barrier four times to prove the algorithm is reusable.	19

List of Images

Figure 1 Test results on Apple M1	19
---	----

Chapter I: Introduction

Chapter 1 introduces the project's research topic, focusing on distributed barrier synchronization algorithms in high-performance computing.

Section 1.1 explores the motivation behind the research, highlighting the growing importance of high-performance computing across scientific domains, including large language model training and complex simulations. Particularly, the section emphasizes the critical role of barrier algorithms in synchronizing computational processes across different nodes.

Section 1.2 details the project objectives, which encompass a comprehensive exploration of high-performance computing concepts, including MPI-3 and C++11 threading technologies. The research aims to investigate a hybrid programming model that combines the communication strengths of MPI with the synchronization capabilities of C++11. The objectives include surveying existing barrier synchronization algorithms and proposing methods for their deployment on current high-performance computing systems.

Section 1.3 narrows down the project's focus, the scope includes exploring key HPC concepts, studying MPI-3 and C++11 threading features, and implementing a simple barrier synchronization algorithm using the hybrid programming model. This aims to equip the author with new concepts before tackling the research objectives.

The project report structure, outlined in Section 1.4, provides a clear roadmap for the research. The document is organized into six chapters, progressing from foundational concepts to detailed algorithm proposals, experimental results, and future research directions. Each chapter builds upon the previous one, creating a comprehensive narrative of the research journey from introduction to conclusion.

1.1 Motivation

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aequi doleamus animo, cum corpore dolemus, fieri.

1.2 Objectives

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aequi doleamus animo, cum corpore dolemus, fieri.

1.3 Scope

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aequi doleamus animo, cum corpore dolemus, fieri.

1.4 Project report structure

The rest of this report is organized as follows:

Chapter II: Background

This chapter establishes the theoretical and technical foundations of the research through three main sections:

Section 2.1 Message Passing Interface (MPI-3)

- Evolution and significance in distributed computing
- Advanced communication mechanisms, focusing on One-Sided Communications
- Remote Memory Access (RMA) operations and memory models
- Collective operations and parallel I/O capabilities

Section 2.2 Modern C++11 Multithreading

- Native threading support and platform independence
- Thread management and synchronization primitives
- Memory model and atomic operations
- Asynchronous programming features and their implications
- Integration of shared and distributed memory paradigms
- Hybrid programming models in modern HPC

Chapter III: Related Works

Surveys existing research and approaches to barrier synchronization algorithms.

Chapter IV: Adaptation from shared memory to distributed memory

This chapter introduces an adaptation of Brook's barrier algorithm, originally designed for shared memory models, to distributed memory systems using MPI's Remote Memory Access (RMA) operations.

This implementation leverages MPI RMA primitives for efficient communication in distributed memory systems, preserving the core synchronization logic of Brook's original algorithm.

Chapter V: Preliminary Results

Presents the implementation outcomes, experimental results, and performance analysis.

Chapter VI: Conclusions and Future Works

Summarizes accomplishments, outlines future research directions, and provides a timeline for planned activities.

Chapter II: Background

Chapter 2 provides a comprehensive background on the fundamental technologies underlying the research: Message Passing Interface (MPI) and C++11 Multithreading. The chapter explores these technologies within the context of high-performance computing, laying the groundwork for understanding the hybrid programming model proposed in the thesis.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri.

2.1 Message Passing Interface (MPI)

This section explores Message Passing Interface, its various paradigms for message passing. The section will then shift its focus into MPI's One-Sided Communication mechanics.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aequi doleamus animo, cum corpore dolemus, fieri.

2.2 C++11 Multithreading

C++11 (ISO/IEC 14882:2011) introduced native support for multithreading directly in the Standard Template Library (STL) [1]. This eliminated the previous reliance on platform-specific threading libraries like POSIX thread (pthread) and the Microsoft Windows API, offering a more portable and standardized approach to parallel computing. The introduction of these features marked a significant advancement in C++ development, providing developers with powerful, standardized tools for concurrent and parallel programming.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aequi doleamus animo, cum corpore dolemus, fieri.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aequi doleamus animo, cum corpore dolemus, fieri.

Chapter III: Related Works

This section examines two key areas relevant to my implementation: barrier synchronization algorithms and hybrid parallelization approaches.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri.

3.1 Barrier Synchronization Algorithm Selection

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aequi doleamus animo, cum corpore dolemus, fieri.

3.2 Hybrid Parallelization Approaches

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aequi doleamus animo, cum corpore dolemus, fieri.

Chapter IV: Adaptation from shared memory to distributed memory

This chapter presents my proposed adaptation of Brook's barrier algorithm within the context of distributed memory models.

Section 4.1 presents Brook's barrier algorithm within the context of the shared memory model.

Section 4.2 provides a straightforward implementation of Brook's two-process barrier algorithm in a distributed memory programming model using MPI's Remote Memory Access (RMA) operations.

4.1 Brook Algorithm

Brook [2] bases the n-process barrier on a two-process barrier using two shared variables. The algorithm is as follows:

Brook-Barrier-Algorithm

```
1 procedure Brook's-Barrier(SetByMyProcess, SetByTargetProcess)
2   while SetByMyProcess is true do wait
3   SetByMyProcess  $\leftarrow$  true
4   while SetByTargetProcess is false do wait
5   SetByTargetProcess  $\leftarrow$  false
6 end procedure
```

We can visualize the algorithm as follows:

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 My proposed implementation of Brook's algorithm

We can extend the concept of Brooks' two-process barrier, where two processes share memory through shared variables, to the one-sided communication model.

In this model, one process can directly access the variables of another process. Instead of storing the shared variables in a common memory segment, each process maintains its own copy, allowing the other process to read and modify it using one-sided communication primitives.

This is my simple adaptation of Brook's two-process barrier algorithm to a distributed memory model using MPI RMA operations:


```
MPI_BROOK_BARRIER(exposed_flag, win, target_rank)
```

INPUTS:

```
exposed_flag: BOOL
```

```
win: MPI_WIN
```

```
target_rank: INT
```

OUTPUTS:

```
None
```

Listing 1: Barrier signature

Distributed-Mem-Brook-Barrier

```
1 procedure Brook's-Barrier(exposed_flag, win, target_rank)
2   while exposed_flag is true do wait
3     exposed_flag ← true
4   while target_flag is false do wait
5     MPI_WIN_LOCK(win)
6     MPI_GET_ACCUMULATE(&target_flag, 0, BOOL, &target_flag, 1, BOOL,
7       target_rank, 0, 1, BOOL, MPI_NO_OP, win);
8     MPI_WIN_FLUSH(win)
9     MPI_WIN_UNLOCK(win)
10  end while
11  false_value ← false
12  MPI_WIN_LOCK(win)
13  MPI_ACCUMULATE(&>false_value, 1, BOOL, target_rank, 0, 1, BOOL,
14    MPI_REPLACE, win);
15  MPI_WIN_FLUSH(win)
16  MPI_WIN_UNLOCK(win)
17 end procedure
```

Listing 2: My adaptation of Brook's two-process barrier using MPI primitives

4.2.1 Notes on the adapted algorithm

The implementation uses MPI_GET_ACCUMULATE and MPI_ACCUMULATE instead of MPI_GET and MPI_PUT for atomicity guarantees. The MPI_WIN_LOCK and MPI_WIN_UNLOCK functions are used to ensure atomicity of the operations.

Chapter V: Preliminary Results

This chapter showcases the preliminary test result of the Brook's two-process barrier algorithm using MPI's RMA operations.

5.1 Test Environment

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

5.2 Test Implementation

The test program was designed to verify the basic functionality of the barrier synchronization:

```
int main(int argc, char **argv) {  
    std::cout << "hello world";  
    return 0;  
}
```

Listing 3: Ran the barrier four times to prove the algorithm is reusable.

5.3 Compilation and Execution

The program was compiled using the following command:

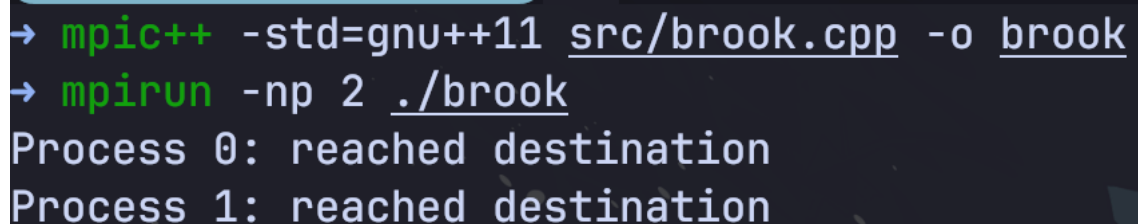
```
mpic++ -std=c++11 src/brook.cpp -o brook
```

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

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

5.4 Results

The test results demonstrated successful barrier synchronization between two processes:



```
→ mpic++ -std=gnu++11 src/brook.cpp -o brook  
→ mpirun -np 2 ./brook  
Process 0: reached destination  
Process 1: reached destination
```

Figure 1: Test results on Apple M1

Chapter VI: Conclusions and Future Works

6.1 Accomplishments This Semester

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aequi doleamus animo, cum corpore dolemus, fieri.

6.2 Challenges and Learnings

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aequi doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguere possit, augeri amplificarique non possit.

6.3 Planned Research Trajectory

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aequi doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguere possit, augeri amplificarique non possit. At.

References

- [1] A. Williams, *C++ Concurrency in Action*, 2nd ed. Manning, 2019, pp. 1–172.
- [2] E. D. Brooks, “The butterfly barrier,” *International Journal of Parallel Programming*, vol. 15, no. 4, pp. 295–307, Aug. 1986, doi: [10.1007/BF01407877](https://doi.org/10.1007/BF01407877).