Università degli Studi di Napoli Federico II

Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione

Parallel and Distributed Computing submissions

Giuliano Aiello

2024

Contents

Chapter 1

Prolusion

1.1 Goal

A comprehensive overview of a Parallel and Distributed Computing project developed in `C` is presented. The project consists of multiple modules delivered in incremental phases. Every algorithm is implemented within the **Single Program Multiple Data** parallel model.

This report is not intended as a user guide, but rather aims to describe the project's exploration of Parallel and Distributed Computing techniques, leveraging High Performance Computing in certain instances.

1.2 Environment

The project was entirely developed on macOS with the help of Xcode IDE.
Naturally, this will mainly impact the build process.

1.3 Project directory layout

The structure of the project's root directory is outlined below.

```
parallel-distributed-computing/
├── common/
├── hpc/
│   ├── gemm/
│   ├── matmatblock/
│   ├── matmatdist/
│   └── matmatthread/
├── laplace/
├── maxsum/
├── ringsum/
└── parallel-distributed-computing.entitlements
```

`common` package serves as a library of utility functions designed to support and be reused by various modules across the project.

The remaining directories represent the individual project modules, which constitute the deliverables of the project. Within each module, the directory structure follows a standard format:

```
<module>/
├── build/
│   ├── deploy-cluster.pbs
│   └── Makefile
├── src/
│   ├── <module>/
│   │   ├── <module>.c
│   │   └── <module>.h
│   └── main.c
├── config.sh
└── run.sh
```

Most parts of the `main.c` files are provided by the project supervisor.

1.4 Build

The project was primarily compiled using the Clang compiler.
The build process was carried out either through the `Makefile` (some of which support compilers other than Clang) or via Xcode.

Regardless of the build process, every module of the project was compiled with:

- `-O3` optimization flag to maximize performance.

1.4.1 Libraries

The following are the dynamically linked libraries integrated into the project.

- `math.h`
- `mpi.h`
- `omp.h`
- `stdio.h`
- `stdbool.h`
- `stdlib.h`
- `sys/time.h`
- `unistd.h`

1.4.2 Xcode

When it came to build with the Xcode, the development process adhered to the workflow and conventions defined by the chosen IDE, leveraging its built-in tools and features to organize and manage the project. Particularly, this includes:

- Xcode targets
- Xcode schemes
- Xcode `.entitlements` file

1.5 Run

1.5.1 Network Requirement

Running an MPI module with no internet connection, makes the following error occur:

```
[Giulianos-MacBook-Pro.local:05355] ptl_tool: problems getting address for
index 0 (kernel index -1)
--------------------------------------------------------------------------
The PMIx server's listener thread failed to start. We cannot continue.
--------------------------------------------------------------------------
Program ended with exit code: 213
```

Chapter 2

Maxsum

2.1 Problem at a glance

Given a matrix, the objective is to identify the row with the maximum squared root sum.

2.2 Algorithm

2.2.1 Parallelization

The algorithm employs a multithreaded approach, each thread can compute indipendently its local maximum squared root sum. Once a thread has completed its task, a synchronization is needed with the other threads in order to determine the overall maximum squared root sum.

2.2.2 Pseudocode

Input:

- global matrix
- number of thread(s) for parallel execution

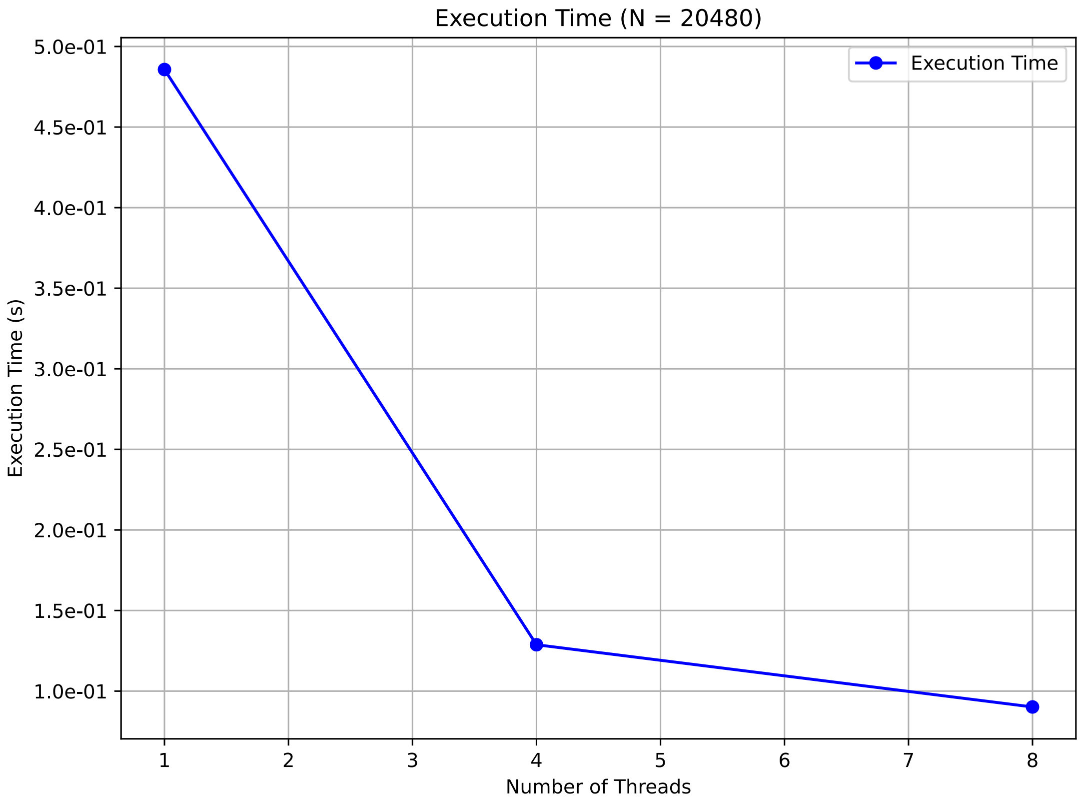
Steps:

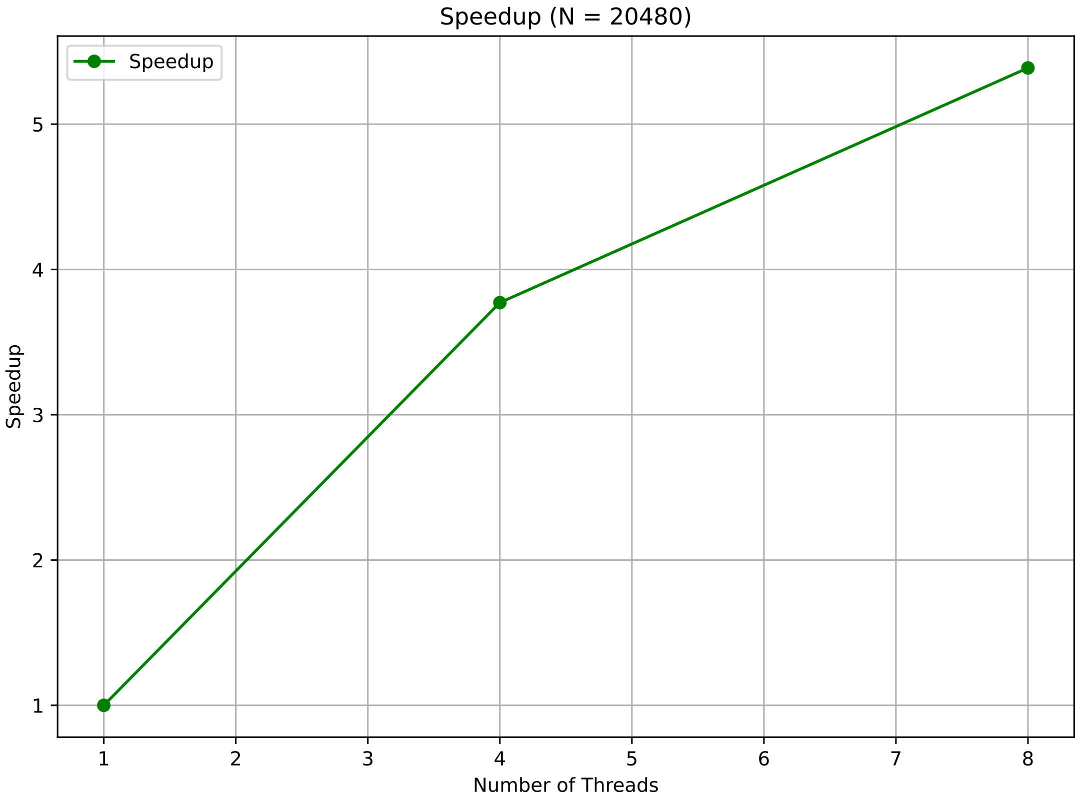
1. Compute the indices of the global matrix corresponding to the local matrix of the thread.
2. Compute the local maximum squared root sum.
3. Perform a thread-safe update of the overall maximum squared root sum variable.

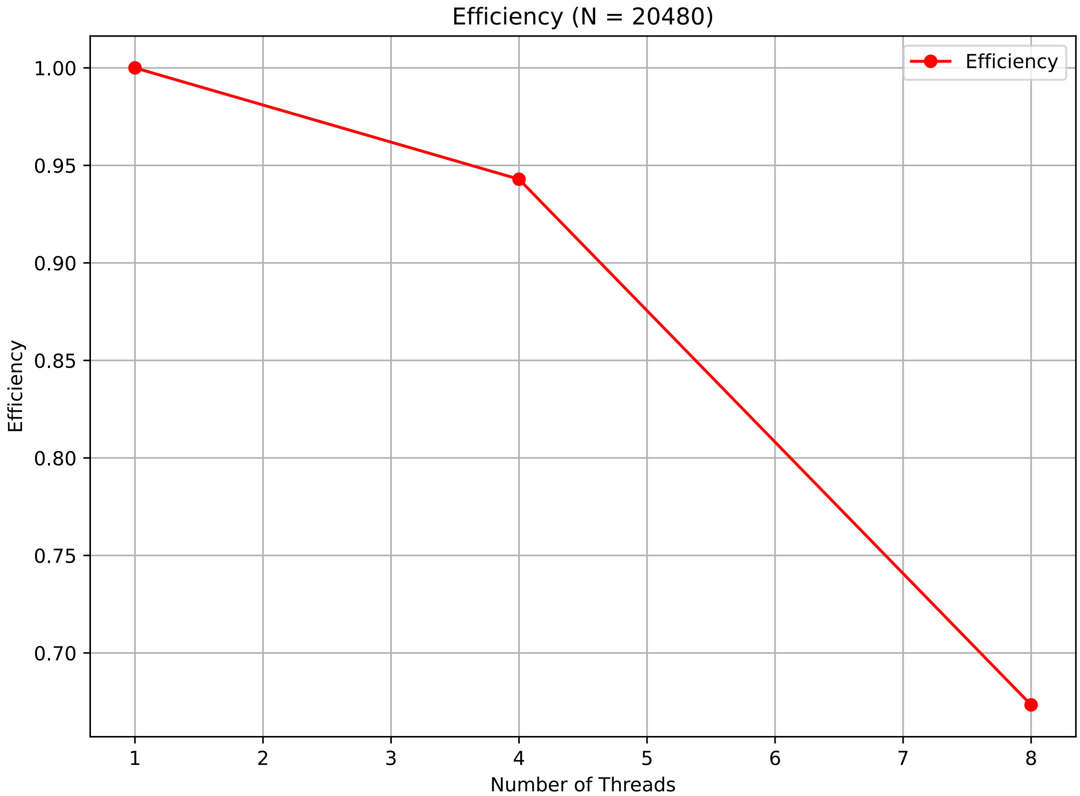
Algorithm 1: Maxsum

```
Input   : matrix, numThreads
Output: maxSumOverall

STARTTHREADS(numThreads)

startRow ← getStartingRowIndex(threadRank)
endRow ← getEndingRowIndex(threadRank)

for currRow ← startRow to endRow do
    currSum ← squareRootSum(currRow)

    if currSum > maxSum then
        maxSum ← currSum
    end

    CRITICAL
    if maxSumOverall < maxSum then
        maxSumOverall ← maxSum
    end
end
```

2.2.3 Performance Analysis




Speedup (N = 20480)
Speedup
Speedup
Number of Threads
1
2
3
4
5
6
7
8


Efficiency (N = 20480)
Efficiency
1.00
0.95
0.90
0.85
0.80
0.75
0.70
Efficiency
1
2
3
4
5
6
7
8
Number of Threads

Chapter 3

Laplace

3.1 Problem at a glance

Given a matrix and a number of iterations, the objective is to compute the Laplacian over the specified iterations.

3.2 Algorithm

3.2.1 Parallelization

The algorithm employs a multiprocess approach. A synchronization between processes is needed from the outset.

The global matrix is distributed among the processes, harnessing the advantages of parallelism, representing a spatial advantage other than a temporal one.

3.2.2 Pseudocode

Input:

- local process matrix
- number of iteration(s)

Steps: For each iteration

1. If $P_i \neq P_1$ Then
   (a) Send to P_{i-1} the local first row
   (b) Receive from P_{i-1} its last row
2. If $P_i \neq P_n$ Then
   (a) Receive from P_{i+1} its first row
   (b) Send to P_{i+1} the local last row
3. Compute the Laplacian considering only the local inner matrix, the matrix boundaries are left out.

4. If $P_i \neq P_1$
   (a) Compute the Laplacian considering only the local top row, with the auxiliary last row from the previous process.
5. If $P_i \neq P_n$
   (a) Compute the Laplacian considering only the local bottom row, with the auxiliary first row from the next process.

An auxiliary matrix is used to facilitate the computation.

Algorithm 2: Laplace

Input : *matrix*, *iterations*
Output: *matrix*

for $iter \leftarrow 1$ **to** *iterations* **do**
  if $P_i \neq P_1$ **then**
    `SEND(`P_{i-1}`,` *localFirstRow*`)`
    `RECV(`P_{i-1}`,` *receivedLastRow*`)`
  end
  if $P_i \neq P_n$ **then**
    `RECV(`P_{i+1}`,` *receivedFirstRow*`)`
    `SEND(`P_{i+1}`,` *localLastRow*`)`
  end
  laplacian(inner(matrix))
  if $P_i \neq P_1$ **then**
    laplacian(matrix, receivedLastRow)
  end
  if $P_i \neq P_n$ **then**
    laplacian(matrix, receivedFirstRow)
  end
end

Acronyms

HPC High Performance Computing 1

IDE Integrated Development Environment 1, 3

MPI Message Passing Interface 4

PDC Parallel and Distributed Computing 1

SPMD Single Program Multiple Data 1