Assignment 4

Bjørn Christian Weinbach Thameez Bodhanya

October 13, 2021

Instructions

For compiling the code that was provided in the assignment as well as the code that was edited for solving the exercises, the makefile provided was used.

We have used *gullviva.it.uu.se* for checking that the code runs. The code is structured in the following way:

- Q1
  - hosts
  - Makefile
  - results.text
  - sieve_bcast.cpp
  - sieve.cpp

The makefile has several commands. If you want to compile the program:

*Make PROG=**filename***

Where ***filename*** is the name of the C/C++ file to compile. If you want to run the code locally on one machine. Use the command

*Make run-local PROG=**filename***

If you want to run it on all three machines in the hostfile, use the command

*Make run PROG=**filename***

Question 1.1 - Sieve of Eratosthenes - MPI (Single Machine)

In this question we were asked to rewrite our earlier implementation of the Sieve of Eratosthenes using MPI, while utilizing 'MPI_send' and 'MPI_Recv'. Once complete we were then asked to run the code on a single server and note the speed-up/slow-down when compared to previous implementations.

Process

- Reusing our implementation of the sieve from Assignment 2, we first removed all PThread specific code.
- Proceeded to the add in the required MPI directives.
  - MPI_Init
  - MPI_Comm_rank
  - MPI_Comm_size
  - MPI_Send
  - MPI_Recv
- Once the code was refactored with the correct directives, testing of the code then began.
- Once satisfied that the code still functioned as expected (and produced the correct results) we then began testing and recording the data.

Difficulty

When comparing to the PThread implementation, using MPI was far easier to write. However using OpenMP far exceeds MPI in ease-of-use, readability and code-length.

Results

As seen in figure1, running the MPI implementation on a single machine far outstripped our initial PThread version with about a 50% reduction in time. However when compared to the OpenMP implementation, we see that the MPI version is slightly slower. This seems to be likely due to the manual calculation of the ‘range‘, ‘startNumber‘ for each thread (we assume that that OpenMP’s ‘for‘ directive has a far more refined and optimized algorithm than ours).

Question 1.2 - Sieve of Eratosthenes - MPI (Multi-Machine)

In this question we were asked to rewrite our earlier implementation of the Sieve of Eratosthenes using MPI, while utilizing ‘MPI_send‘ and ‘MPI_Recv‘. Once complete we were then asked to run the code on three server and note the speed-up/slow-down when compared to previous implementations.

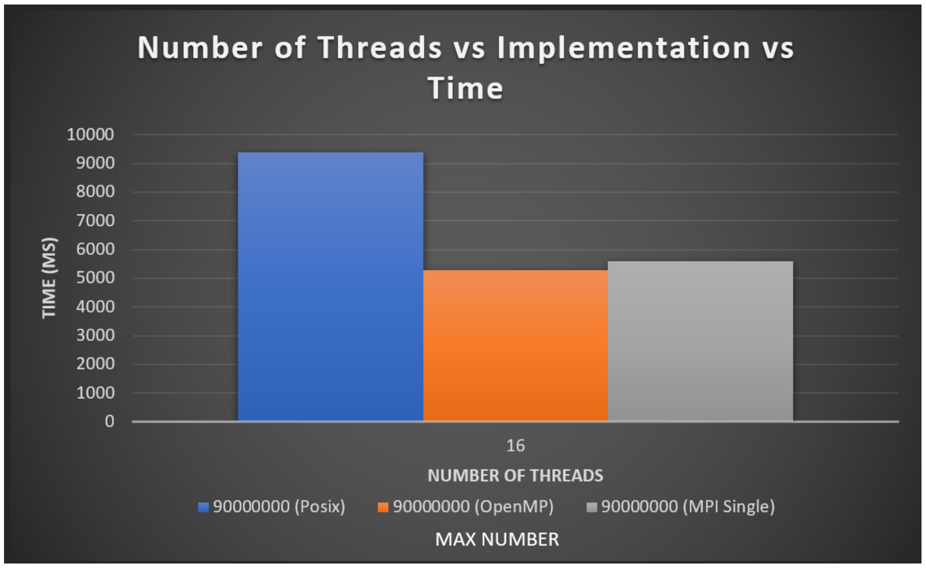

Figure 1: Sieve of Eratosthenes with max number 9×10^7 on different implementations.

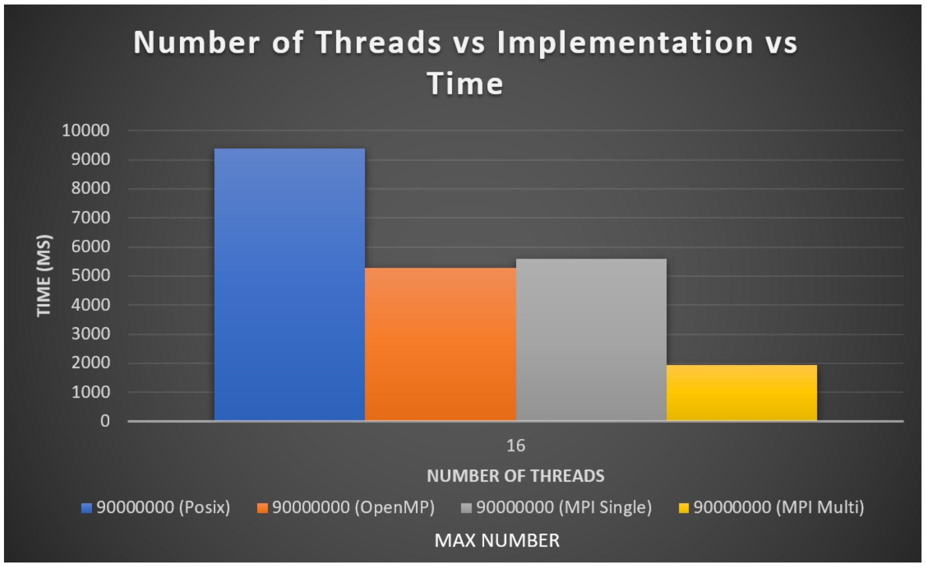

Figure 2: Sieve of Eratosthenes with max number 9×10^7 on different implementations.

Results

As seen in figure 2, running the MPI implementation on multi-machines far outstripped our initial PThread version and OpenMP implementations. This makes sense as we are now exploiting 3x the amount of cores we previously had. Interestingly even though we now utilized 3x the amount of cores, we only saw a 70% improvement in terms of speed. This can be attributed to a number of factors:

- Network Traffic and Latency
- Overhead the MPI adds when running on multiple-machines
- Time spent sending and receiving from the different machines
- The code which is still being run serially in the application

Question 1.3 - Sieve of Eratosthenes - MPI (Multi-Machine using MPI_Reduce)

In this question we were asked to rewrite our earlier implementation of the Sieve of Eratosthenes using MPI, while utilizing ‘MPI_Bcast‘ and ‘MPI_Reduce‘. Once complete we were then asked to run the code on three server and note the speed-up/slow-down when compared to previous implementations.

Process

- We firstly removed any mentions of ‘MPI_Send‘ and ‘MPI_Recv‘ from our implementation.
- We then refactored the code to use ‘MPI_Bcast‘ and ‘MPI_Reduce‘

Difficulty

When comparing to the our initial MPI implementation, using Broadcast and Reduce was far easier to write. However using OpenMP still far exceeds MPI in ease-of-use, readability and code-length.

Results

As seen in figure 3, running the MPI implementation with ‘Broadcast‘ and ‘Reduce‘, we see no noticeable slow-down or speed-up between the two implementations. This is most likely as our application does very little at send/receive or reduce time, therefore there would be no noticeable speed-up between the two algorithms. Therefore due to the slightly cleaner code and easier implementation, we would suggest using ‘MPI_Bcast‘ and ‘MPI_Reduce‘ to replace ‘MPI_Send‘ and ‘MPI_Recv‘ where possible.

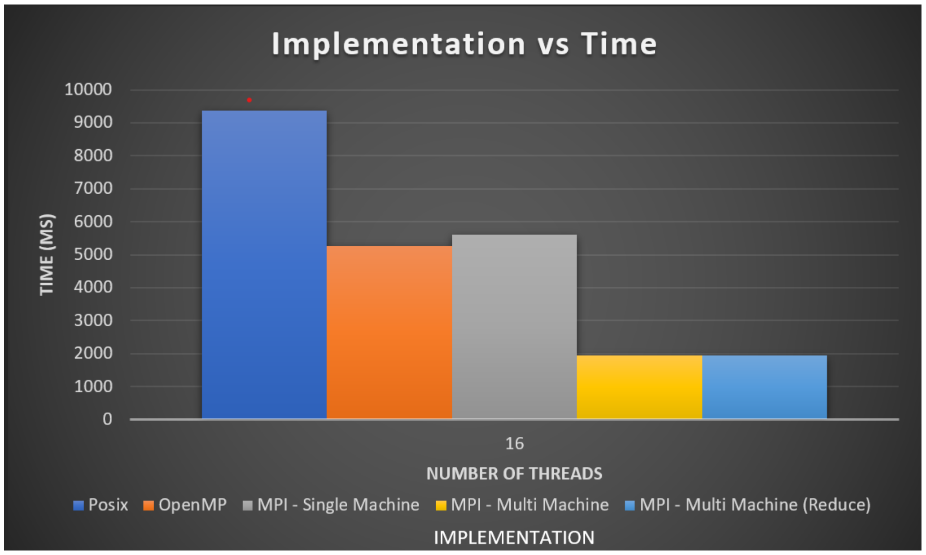

Figure 3: Sieve of Eratosthenes with max number 9×10^7 on different implementations.