Национальный исследовательский ядерный университет «МИФИ»

(Московский Инженерно-Физический Институт)

Кафедра №42 «Криптология и кибербезопасность»

Лабораторная работа №5

«Технология MPI. Введение»

Тимин Александр Б21-515 (2023г.)

Рабочая среда:

- Процессор: AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz, 8 ядер (16 логических)
- Оперативная память: 16.0 GB DDR4 3200 МГц
- ОС: Windows 10 Pro 22H2 64-bit operating system, x64-based processor
- Среда разработки: Microsoft Visual Studio 2022 (v143)
- Версия OpenMP: 200203 (/openmp:llvm)

Алгоритм

Программа запускается на **P** процессах, на каждом из которых (для экономии времени разработки) генерируется и заполняется псевдослучайными числами 10 массивов размера **N**. Для каждого массива вызывается функция поиска максимального значения. Главный процесс (rank = 0) разделяет свой массив на **P** частей (поровну кроме одной, если **N** нацело не делится на **P**). **P**-1 части отсылаются по другим процессам, последняя часть остается у главного процесса. Каждый процесс выполняет поиск максимального элемента в своей части массива. Найденные максимальные элементы возвращаются на главный процесс, проходя через оператор **MAX**, выявляющего максимальный из максимальных элемент.

Сложность алгоритма O(**N**).

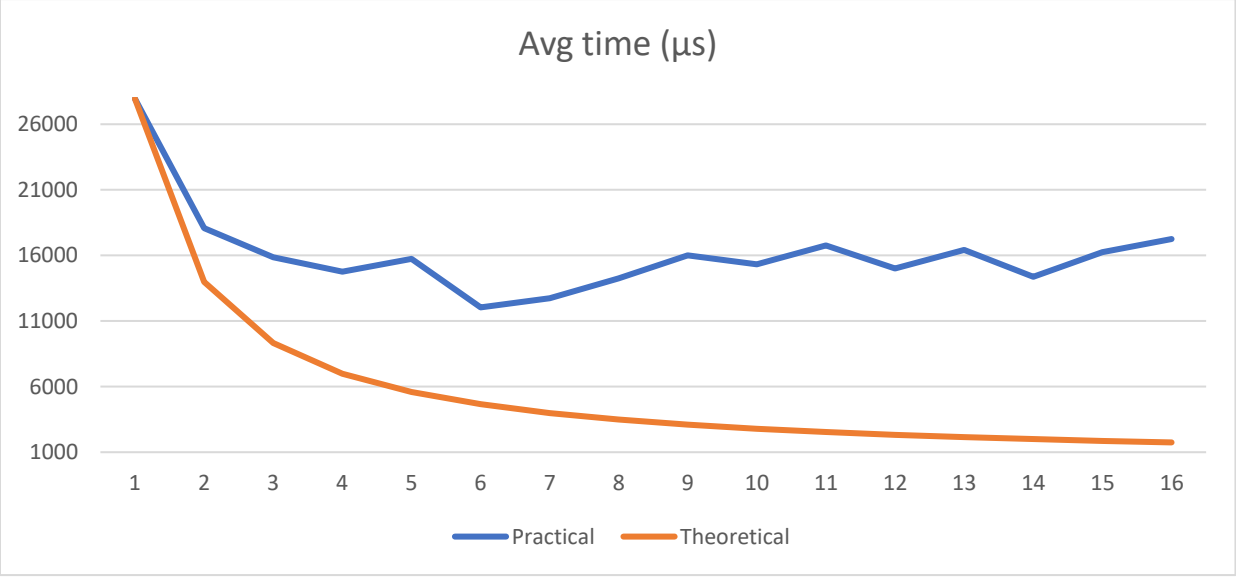
Ход работы

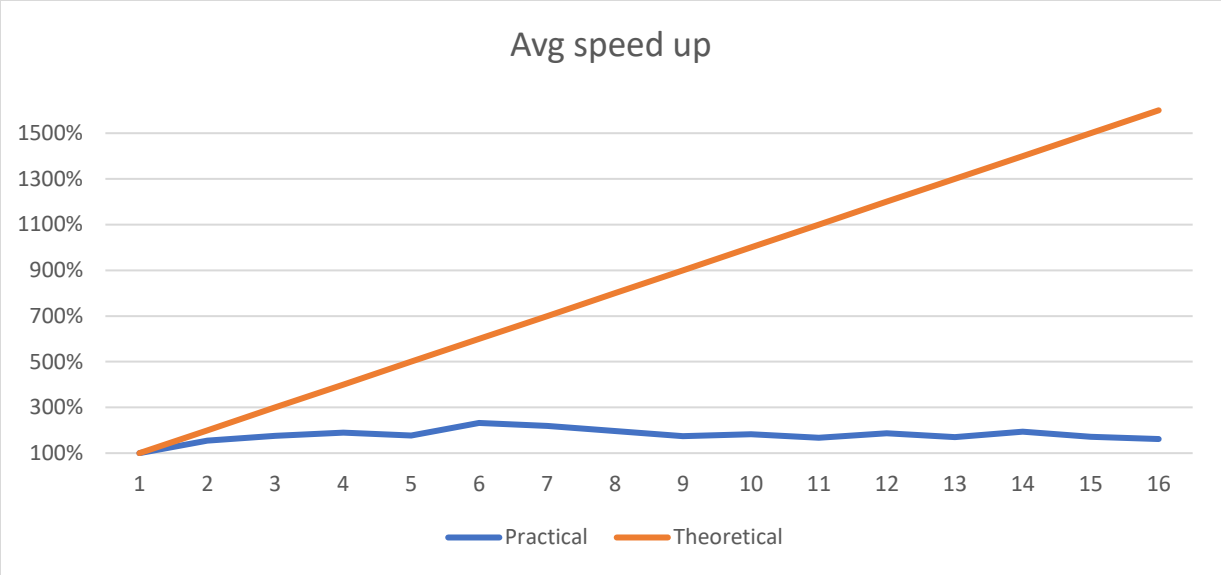
Была модифицирована программа из первой лабораторной работы Таким образом, чтобы использовать возможности библиотеки mpi.h там, где ранее использовалась технология OpenMP для параллельной работы алгоритма. Программа была запущена 16 раз на разном количестве процессов (от 1 до 16), временные характеристики записывались в файл для дальнейшей обработки. На основании продолжительности выполнения алгоритма были вычислены следующие характеристики: среднее время, среднее ускорение и средняя эффективность. По полученным данным были построены соответствующие графики.

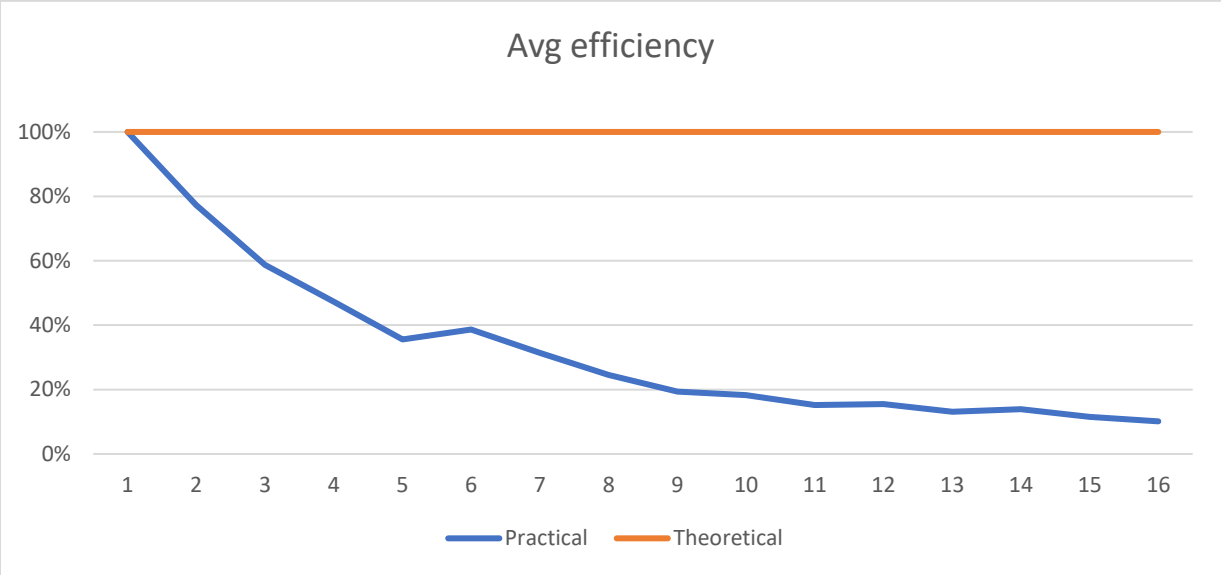
Данные

Procs	Avg time (pr)	Avg time (th)	Avg speed up (pr)	Avg speed up (th)	Avg efficiency (pr)	Avg efficiency (th)
1	27941.46	27941.46	100.00%	100.00%	100.00%	100.00%
2	18067.21	13970.73	154.65%	200.00%	77.33%	100.00%
3	15862.56	9313.82	176.15%	300.00%	58.72%	100.00%
4	14760.42	6985.37	189.30%	400.00%	47.32%	100.00%
5	15727.24	5588.29	177.66%	500.00%	35.53%	100.00%
6	12038.92	4656.91	232.09%	600.00%	38.68%	100.00%
7	12725.80	3991.64	219.57%	700.00%	31.37%	100.00%
8	14246.39	3492.68	196.13%	800.00%	24.52%	100.00%
9	16012.06	3104.61	174.50%	900.00%	19.39%	100.00%
10	15317.70	2794.15	182.41%	1000.00%	18.24%	100.00%
11	16766.13	2540.13	166.65%	1100.00%	15.15%	100.00%
12	14993.66	2328.46	186.36%	1200.00%	15.53%	100.00%
13	16423.13	2149.34	170.13%	1300.00%	13.09%	100.00%
14	14368.68	1995.82	194.46%	1400.00%	13.89%	100.00%
15	16238.68	1862.76	172.07%	1500.00%	11.47%	100.00%
16	17238.11	1746.34	162.09%	1600.00%	10.13%	100.00%

Графики







Заключение

За счет распараллеливания алгоритма удалось сократить время его исполнения по сравнению с последовательной версией, однако прирост ускорения с использованием библиотеки mpi.h оказался меньше, по сравнению с применением технологии OpenMP. Такой эффект объясняется тем, что для обмена сообщениями и данными между процессами требуется дополнительное время. Тем не менее такой подход к разработке параллельной программы тоже имеет место быть, например для функционирования в сети серверов для обработки большого количества данных.

Приложение

- https://github.com/KATEHOK/par_prog-5/blob/main/src/lab.c – исходный код программы;
- https://github.com/KATEHOK/par_prog-5/blob/main/report/data.txt – «сырые» данные (текст);
- https://github.com/KATEHOK/par_prog-5/blob/main/report/data.xlsx – «обработанные» данные (таблица и графики);
- https://github.com/KATEHOK/par_prog-5/blob/main/report/report.docx – отчет (docx);
- https://github.com/KATEHOK/par_prog-5/blob/main/report/report.pdf – отчет (pdf).