Московский государственный университет имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики
Курс «Суперкомпьютеры и Параллельная Обработка Данных»

Задача №62: Разработка параллельной версии программы для задачи “3 Matrix Multiplication” с использованием технологий OpenMP и MPI

Отчет студентки 324 группы
факультета ВМК МГУ
Глуходед Анастасии

Москва, 2021

Содержание

Постановка задачи

Требуется:

1. Улучшить алгоритм.
2. Реализовать параллельный алгоритм умножения матриц с помощью технологии параллельного программирования OMP.
3. Исследовать масштабируемость полученной программы.
4. Построить график зависимости времени её выполнения от числа используемых ядер и объёма входных данных.

Улучшение: реализация блочного умножения матриц.

Описание алгоритма блочного умножения матриц

Алгоритм блочного умножения матриц используют с целью повышения эффективности использования кэш-памяти CPU.

$$C = AB = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \dots & \dots & \dots & \dots \\ A_{n1} & A_{n2} & \dots & A_{nn} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} & \dots & B_{1n} \\ B_{21} & B_{22} & \dots & B_{2n} \\ \dots & \dots & \dots & \dots \\ B_{n1} & B_{n2} & \dots & B_{nn} \end{bmatrix}$$

В котором результирующая матрица

$$C = \begin{bmatrix} C_{11} & C_{12} & \dots & C_{1n} \\ C_{21} & C_{22} & \dots & C_{2n} \\ \dots & \dots & \dots & \dots \\ C_{n1} & C_{n2} & \dots & C_{nn} \end{bmatrix}$$

формируется поблочно с использованием известной формулы

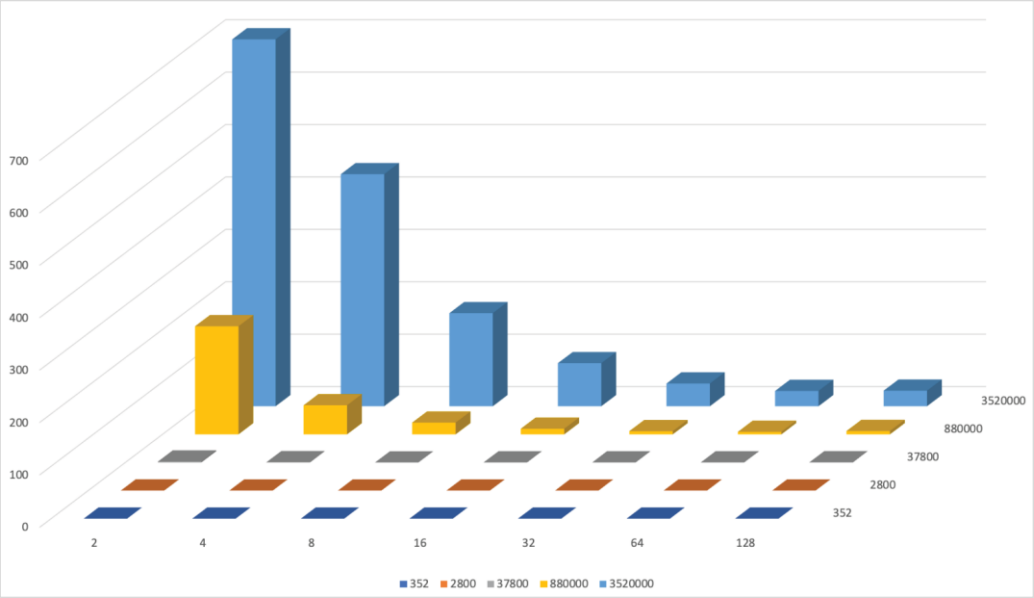
$$C_{ij} = \sum_{k=1}^{n} A_{ik} \times B_{kj}$$

либо её более быстрых аналогов, а размер обрабатываемых данных на каждой итерации не превышает ёмкость кэш-памяти. Размер блока напрямую зависит от архитектуры вычислительной системы и определяет время выполнения умножения.

Результаты замеров времени выполнения программы

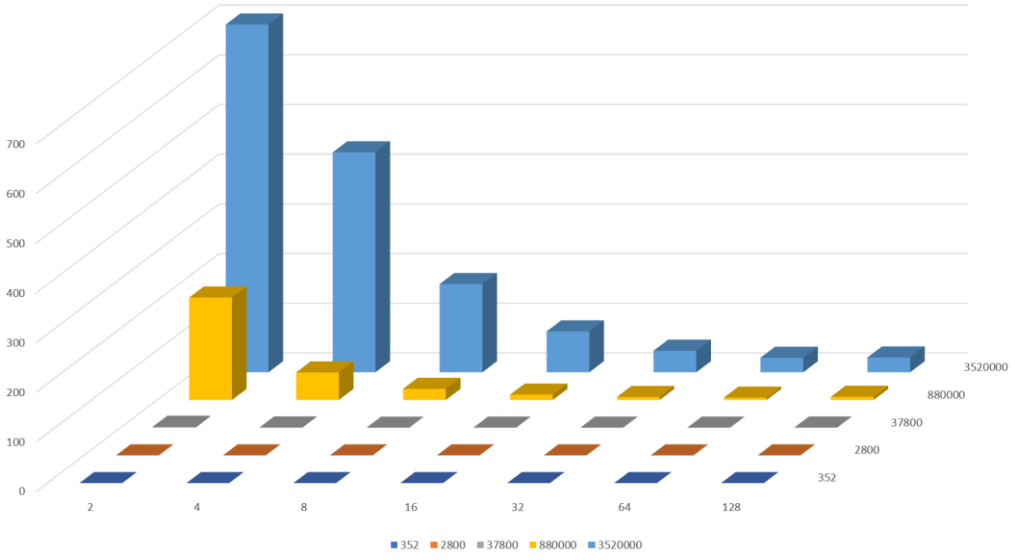
OpenMP

Data_Size \ Amount	1	2	4	8	16	32	64	128
352	0,000038	0,000042	0,000042	0,000043	0,000045	0,000078	0,000146	0,000384
2800	0,000876	0,00087	0,000894	0,000886	0,000884	0,001386	0,002519	0,009382
37800	0,037361	0,020619	0,020482	0,021076	0,021606	0,034161	0,199175	0,35855
880000	4,371283	2,292089	1,267379	0,764135	0,597432	0,930737	2,371948	5,071066
3520000	35,065859	17,539202	9,296192	5,217461	3,41066	3,652303	9,874445	26,349579



MPI

Data_Size \ Amount	2	4	8	16	32	64	128
352	0,001852	0,000771	0,00081	0,001254	0,002299	0,004725	0,009431
2800	0,040739	0,011232	0.005978	0,005232	0,007696	0,013065	0.025947
37800	1,751922	0,469197	0.197647	0,107878	0,091152	0,129028	0,230686
880000	206,20896	55,519542	22,393528	10,669241	6,089453	4,807202	6,358059
3520000	700	442,72179	177,54453	82,172612	43,33087	29,147225	29,701595



Вывод

Блочное умножение матриц дало выигрыш во времени (иногда даже в 2 раза). Однако при использовании данного метода нам нужно подбирать размеры блоков, что не всегда хорошо.

OpenMPI

Распараллеливание программы с помощью технологии OpenMP дало различный выигрыш во времени выполнения в зависимости от числа потоков. Алгоритм с OpenMP довольно прост и понятен в использовании. Хорошо подходит для быстрого запуска параллельной программы и оценки её результатов.

MPI

Распараллеливание программы с помощью технологии MPI требует большое количество узлов для эффективной работы. В случае, когда процессы редко обмениваются данными и выполняют большую часть работы самостоятельно, применения MPI очень выгодно.

Приложение

Код доступен по ссылке: https://github.com/glukhodednastya/3_Matrix_Multiplication.