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 технологии OpenMP

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

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

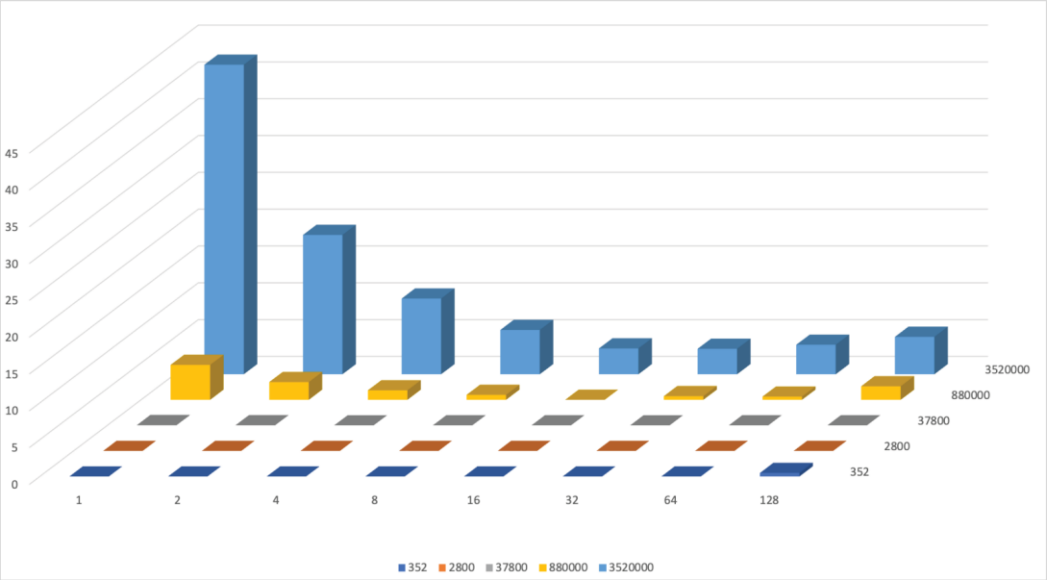
формируется поблочно с использованием известной формулы

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

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.

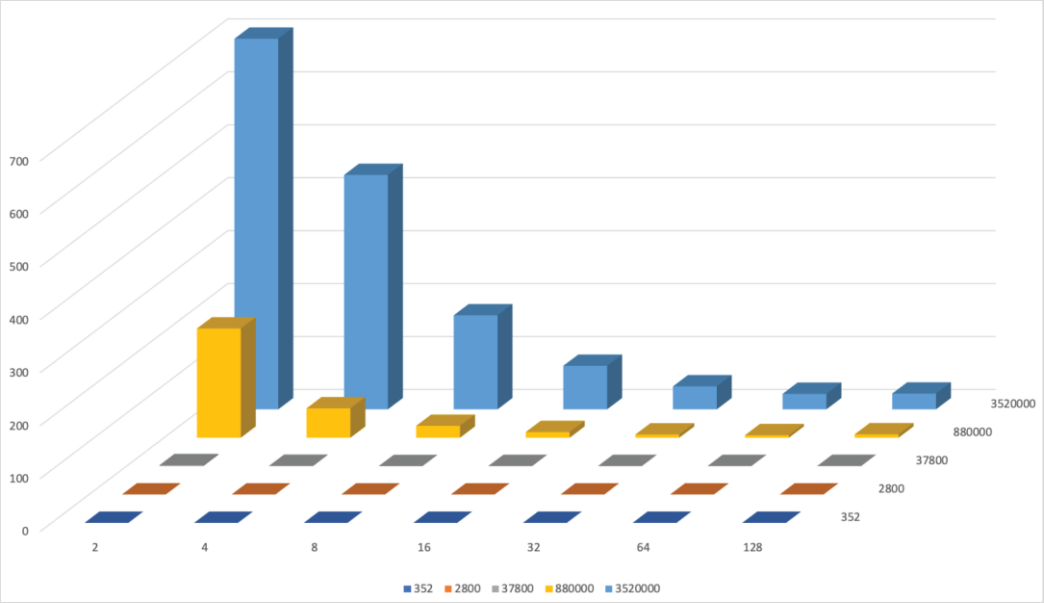
Результаты замеров времени выполнения программы

Amount / Data_Size	1	2	4	8	16	32	64	128
352	0,00003	0,000022	0,00002	0,000017	0,000016	0,000017	0,000027	0,480015
2800	0,000821	0,00042	0,000222	0,000126	0,000089	0,00009	0,000078	0,000273
37800	0,037444	0,018971	0.009501	0,004851	0,002552	0,001816	0,002309	0,003414
880000	4,743801	2,408195	1,291012	0,673047	0.544065	0,497707	0,434879	1,840034
3520000	42,042967	18,926778	10,287609	6,020176	3,513236	3,459082	4,009457	5,08872



OpenMP

Amount / Data_Size	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



Вывод

Блочное умножение матриц дало выигрыш во времени (иногда 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.