Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ”

ОТЧЕТ

О выполнении лабораторной работы № 1
По курсу: «Параллельные алгоритмы анализа и синтеза данных»
«Матричное умножение в OpenMP»

Работу выполнил:
ст. группы С4113 Самарин А. С.

Санкт-Петербург, 2019

ЦЕЛЬ РАБОТЫ

Познакомится с архитектурой параллельных вычислений Shared memory с помощью технологии OpenMP. Рассмотреть ускорение получаемое от параллельных вычислений.

ПОСТАНОВКА ЗАДАЧИ

Реализовать произведение матриц с помощью SPDM паттерна и директив OpenMP. Показать зависимость ускорения выполнения программы от количества входных данных(размера матриц) и от количества потоков исполнения. Сравнить полученные результаты.

КРАТКАЯ ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Задачей выбрана перемножение матриц, так как это достаточно ресурсоемкое вычисление. Теоретическая сложность которого $O(N^3)$, где N — измерение матрицы.

Технология вычислений OpenMP построена на архитектуре с общей памятью, поэтому она имеет все преимущества и недостатки данного подхода.

Достоинства:

- Быстрое доступ к данным и обмен между задачами, так как все находятся в одном адресном пространстве
- Понятная семантика для программирования

Недостатки:

- Плохо масштабируется между вычислительными устройствами
- За синхронизацию, корректный доступ к памяти отвечает программист

Модель вычисления – Fork - Join модель. Когда есть возможность распараллелить задачу, главный поток создает дополнительные потоки (Fork), а по окончанию задачи все созданный потоки соединяются в один главный master поток (Join). Это позволяет добавлять параллелизм пошагово в последовательно исполняющуюся программу. Так же эта модель реализует SPMD паттерн (одна программа много данных). Когда вычислительные средства выполняют одну и ту же программу для разных данных.

OpenMP предоставляет основные интерфейсы для Fortran и C++. API для последнего реализовано как директивы, процедуры и переменные окружения.

Основной единицей исполнения является поток. Поток – самая маленькая единица обработки.

АЛГОРИТМ (МЕТОДИКА) РЕАЛИЗАЦИИ

Мы реализуем 3 версии перемножения матрицы. Последовательный вариант, параллелизм на основе SPMD паттерна с ручным распределением потоков на подзадачи и вариант с использованием директив OpenMP. Опробуем различные варианты для планировщика цикла, предоставляемого OpenMP. Сделаем замеры времени и ускорения программы в зависимости от входных данных и от количества потоков.

РЕЗУЛЬТАТ

Процессор – Intel® Core™ i7-8750H CPU @ 2.20GHz × 12. 6 физических и 12 логических ядер.

При использовании директив для планировщика мы остановились на использовании параметра static. Он циклично распределяет k (параметр директивы) итераций на поток, что позволяет равномерно распределить задачу между потоками. Хорошо подходит, если все подзадачи одинаково ресурсоемки.

Директива с параметрами dynamic, guieded распределяет k (параметр директивы) итераций на поток, причем не известно заранее какие итерации возьмет на себя поток. По окончанию выполнения поток запрашивает еще k итераций. Основное различие между dynamic и guieded, что у первого k фиксировано, а у второго количество запрашиваемых итераций зависит от количества потоков и уменьшается до k. Хорошо подходят для неравномерных задач. Несут накладные расходы во времени выполнения на распределении итераций на потоки.

Практическим путем было установлено, что максимальная производительность наблюдается, когда количество рабочих потоков равно количеству физических ядер. В результатах ниже эта схема соответствует OpenMP with hack.

Данные подавались, как квадратная матрица, поэтому будет указано только одно измерение.

При изменении входных данных матрицы количество потоков равно 6. При изменении количества потоков размер входных матриц равен 1000.

1.1) Матричное умножение с ручной реализацией SPMD

N	T_1	T_p	S_p
500	0.0865	0.0154	5.5941
600	0.1730	0.0320	5.4021
700	0.2787	0.0502	5.5445
800	0.4400	0.0781	5.6311
900	0.6325	0.1443	4.3831
1000	0.9029	0.1772	5.0935
1100	1.2823	0.2445	5.2422
1200	1.6825	0.3049	5.5185
1300	2.8347	0.5091	5.5679
1400	3.5344	0.6745	5.2398
1500	5.3150	0.9675	5.4933

Таблица 1.1

1.2) Матричное умножение с OpenMP

N	T_1	T_p	S_p
500	0.0865	0.0163	5.2983
600	0.1730	0.0324	5.3324
700	0.2787	0.0501	5.5585
800	0.4400	0.0809	5.4379
900	0.6325	0.1268	4.9875
1000	0.9029	0.1911	4.7256
1100	1.2823	0.2528	5.0717
1200	1.6825	0.3425	4.9124
1300	2.8347	0.1554	5.3729
1400	3.5344	0.6793	5.2026
1500	5.3150	0.9795	5.4263

Таблица 1.2

1.3) Матричное умножение с OpenMP with hack

N	T_1	T_p	S_p
500	0.0865	0.0174	4.9642
600	0.1730	0.0307	5.6248
700	0.2787	0.0529	5.2645
800	0.4400	0.0938	4.6899
900	0.6325	0.1369	4.6175
1000	0.9029	0.1900	4.7521
1100	1.2823	0.2549	5.0309
1200	1.6825	0.3418	4.9226
1300	2.8347	0.5287	5.3613
1400	3.5344	0.6662	5.3048
1500	5.3150	0.9953	5.3401

Таблица 1.3

2.1) Матричное умножение с ручной реализацией SPMD

n_threads	T_1	T_p	S_p
2	0.9239	0.4474	2.0651
4	0.9239	0.2590	3.5668
6	0.9239	0.1934	4.7766
8	0.9239	0.2851	3.2406
16	0.9239	0.6071	1.5217
32	0.9239	0.6468	1.4285
50	0.9239	0.6262	1.4754
75	0.9239	0.6369	1.4508
100	0.9239	0.6069	1.5223

Таблица 2.1

2.2) Матричное умножение с OpenMP

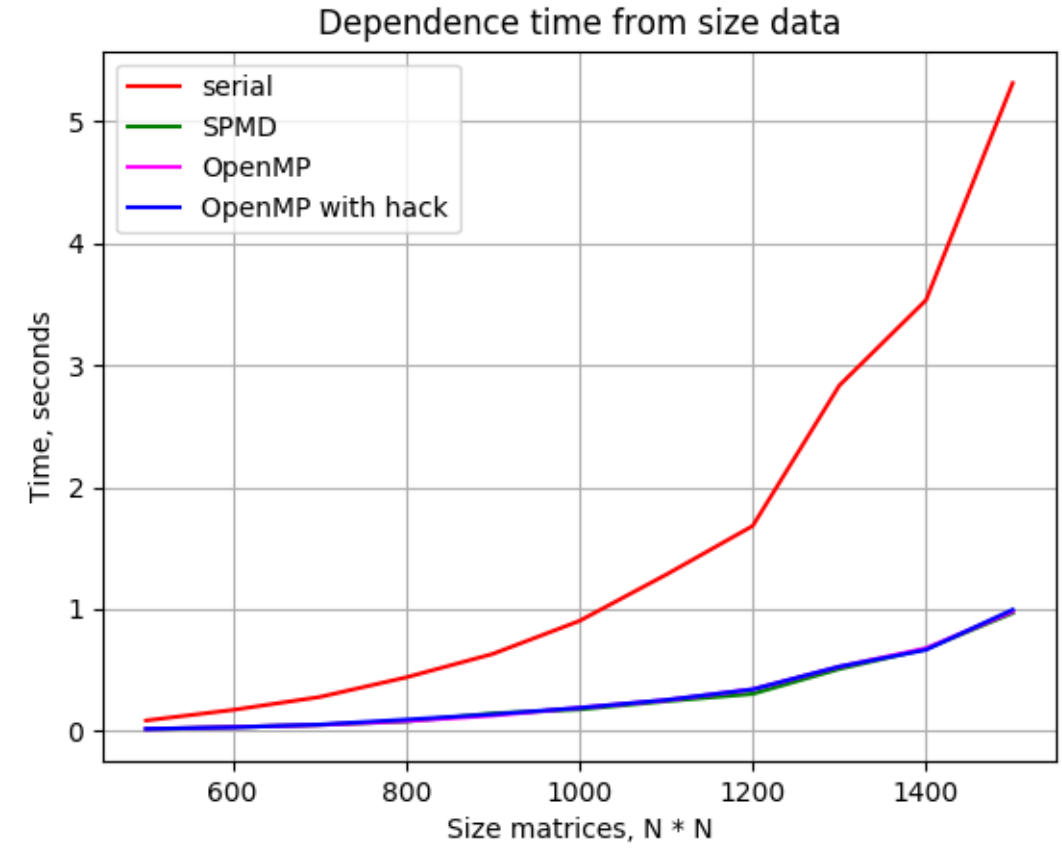
n_threads	T_1	T_p	S_p
2	0.9239	0.4595	2.0108
4	0.9239	0.2528	3.6549
6	0.9239	0.1929	4.7899
8	0.9239	0.2870	3.2196
16	0.9239	0.5909	1.5636
32	0.9239	0.6373	1.4497
50	0.9239	0.6322	1.4613
75	0.9239	0.6109	1.5123
100	0.9239	0.5988	1.5430

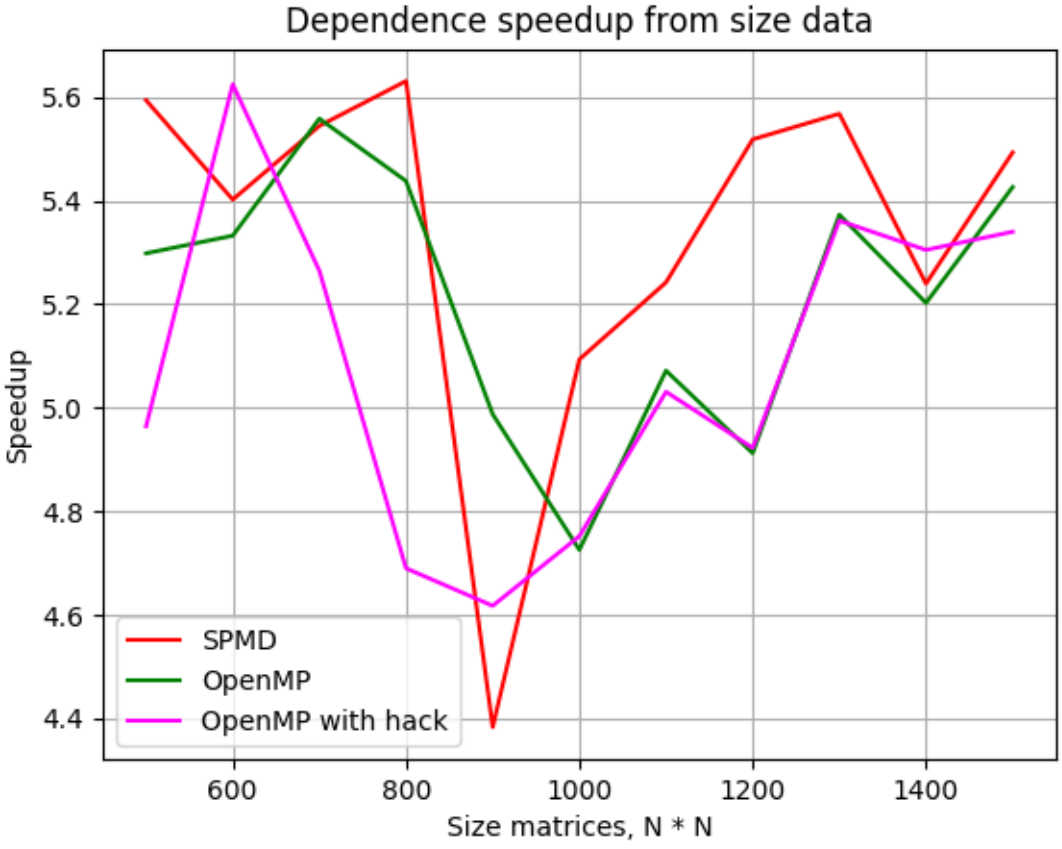
Таблица 2.2

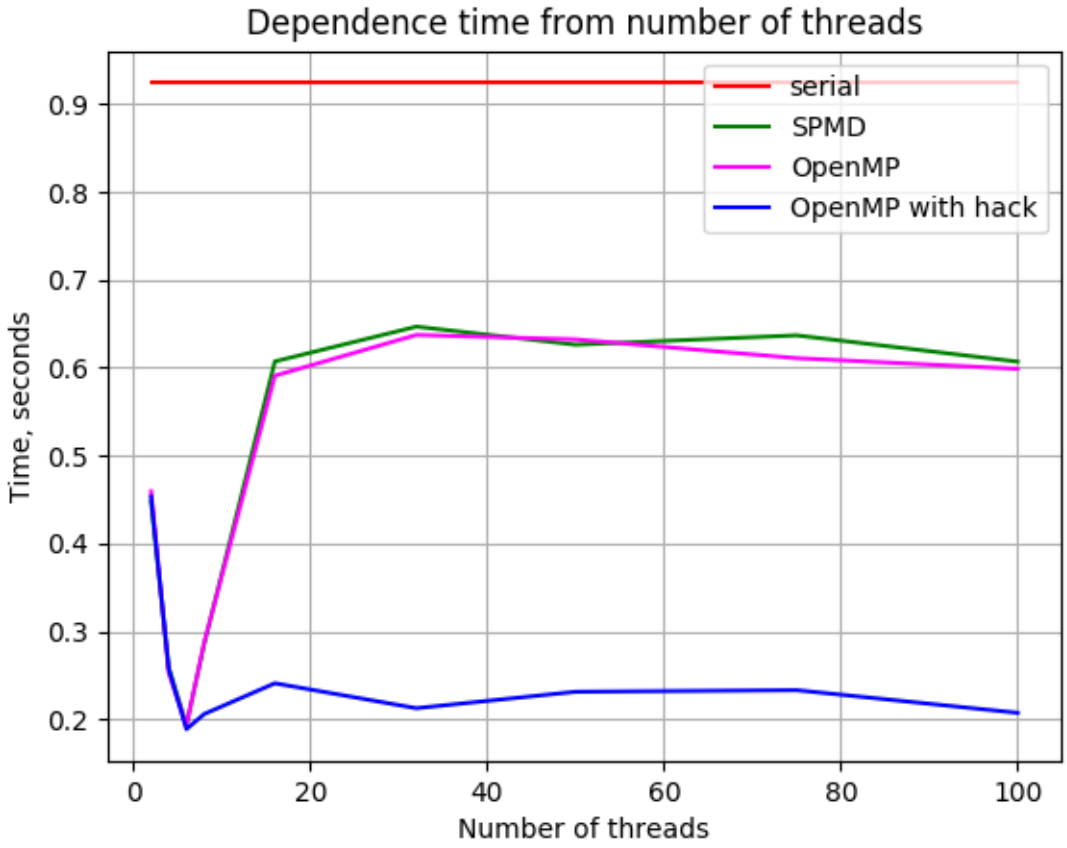
2.3) Матричное умножение с OpenMP with hack

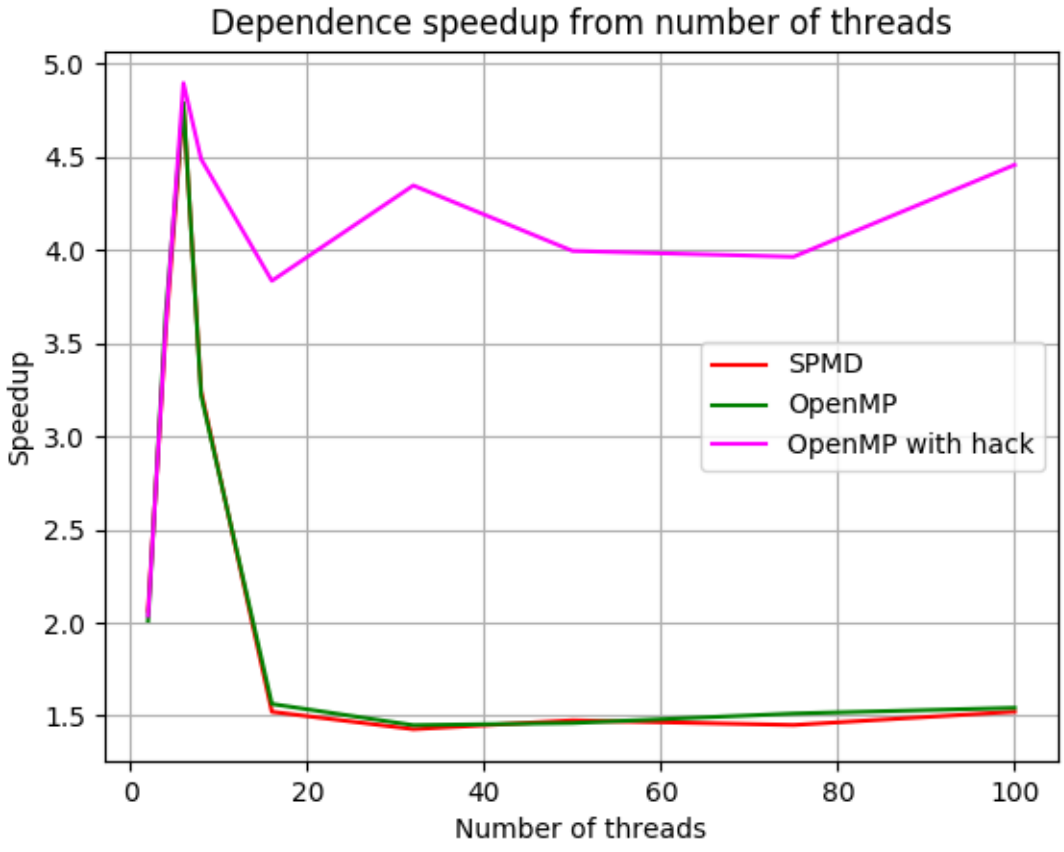
n_threads	T_1	T_p	S_p
2	0.9239	0.4534	2.0378
4	0.9239	0.2569	3.5970
6	0.9239	0.1889	4.8955
8	0.9239	0.2059	4.4873
16	0.9239	0.2410	3.8344
32	0.9239	0.2126	4.3459
50	0.9239	0.2313	3.9948
75	0.9239	0.2331	3.9634
100	0.9239	0.2073	4.4566

Таблица 2.3


Dependence time from size data
serial
SPMD
OpenMP
OpenMP with hack
Time, seconds
0
1
2
3
4
5
600
800
1000
1200
1400
Size matrices, N * N


Dependence speedup from size data
SPMD
OpenMP
OpenMP with hack
Speedup
4.4
4.6
4.8
5.0
5.2
5.4
5.6
600
800
1000
1200
1400
Size matrices, N * N


Dependence time from number of threads
serial
SPMD
OpenMP
OpenMP with hack
Time, seconds
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
20
40
60
80
100
Number of threads


Dependence speedup from number of threads
SPMD
OpenMP
OpenMP with hack
Speedup
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
0
20
40
60
80
100
Number of threads

Исходный код на GitHub

ЗАКЛЮЧЕНИЕ

Мы выполнили поставленное задание, а именно: реализовали 3 подхода к произведению матриц, сделали замеры скорости работы от размера входных данных и количества потоков, визуализировали зависимости для наглядности анализа.

По результатам измерений мы видим, что мы получили ускорение от распараллеливания программы. Но правильно распараллелить не так просто. Необходимо тщательно подбирать количество потоков под каждую систему индивидуально. Так например, хоть мы и имеем сублинейное ускорение, но подобранное количество рабочих потоков дает ускорение больше, чем в других случаях. Так же при ускорении стоит обратить внимание на попадание запроса к памяти в кэш.

Мы познакомились с паттерном SPMD и научились пользоваться технологией OpenMP. Эти навыки помогут нам при увеличении производительности с помощью распараллеливания узких мест программ.