Московский Государственный Университет
имени М.В. Ломоносова
Факультет Вычислительной Математики и Кибернетики

Практикум по курсу

«Суперкомпьютеры и Параллельная обработка данных»

Разработка параллельной версии алгоритма тройного перемножения матриц

Отчет о проделанной работе

327 группа,
Махов А.М.

Москва, 2020 г.

Оглавление

1. Постановка задачи

Ставится задача перемножения двух пар матриц с дальнейшем перемножением получившихся результирующих матриц (некий прототип программы был уже дан).
Даны четыре матрицы, результатом конечного вычисления является матрица, которая получается в ходе перемножения произведений двух пар данных матриц – первой со второй и третьей с четвертой. Проще говоря:

Даны: матрицы A, B, C, D
Необходимо получить: E=A*B, F=C*D, G=E*F

В задаче требуется:

1. Улучшить программу с использованием технологии OpenMP, чтобы ускорить вычисление матриц
2. Сравнить скорость вычисления матриц при различном количестве потоков
3. Сравнить скорость вычисления матриц при различном объеме исходных данных
4. Найти оптимальное количество потоков, на которые стоит распараллеливать вычисления на суперкомпьютере Polus

2. Описание использованных алгоритмов

2.1 Простейший вид алгоритма

Наивное перемножение матриц на языке программирования Си выглядит так:

Сложность такого вычисления составляет по определению перемножения матриц $O(n^3)$.

2.2 Параллельные вычисления

Вся модификация OMP программы сводится к добавлению специальных omp-клауз для выполнения циклов параллельно: в случае данной задачи это клаузы *#pragma omp for* и *#pragma omp parallel* с соответствующими параметрами. При выборе клауз была использована документация IBM и лекции курса 2020 года.

Исходный код задачи доступен как в архиве во вложениях на сайте курса или в репозитории автора (более свежие версии). Код имеет базовую документацию, которую можно найти в директории «*Docs*» проекта.

Компиляция программы проводилась с помощью Apple Clang version 12, x86_64-apple-darwin20.1.0 + libomp.dylib 5-й версии из пакета библиотек llvm-11 (на локальной машине) и с помощью gcc version 4.8.5, ppc64le-redhat-linux. Общие опции компиляции: *-fopenmp -std=gnu11 -Wall -Wpedantic*.
Для компиляции под ОС Linux и других Unix-like системах необходимо использовать утилиту make, «*make help*» выводит все возможные опции компиляции. Для компиляции необходим gcc и библиотека openmp (для Mac OS – пакет llvm из репозитория *homebrew*)

3 Результаты замеров времени выполнения

Для запуска тестов был сделан shell-скрипт (*bench-runner.sh*), пересобирающий и запускающий программу для разного размера матриц, каждый тест был сделан с помощью этого скрипта 5 раз и посчитано среднее время выполнения.

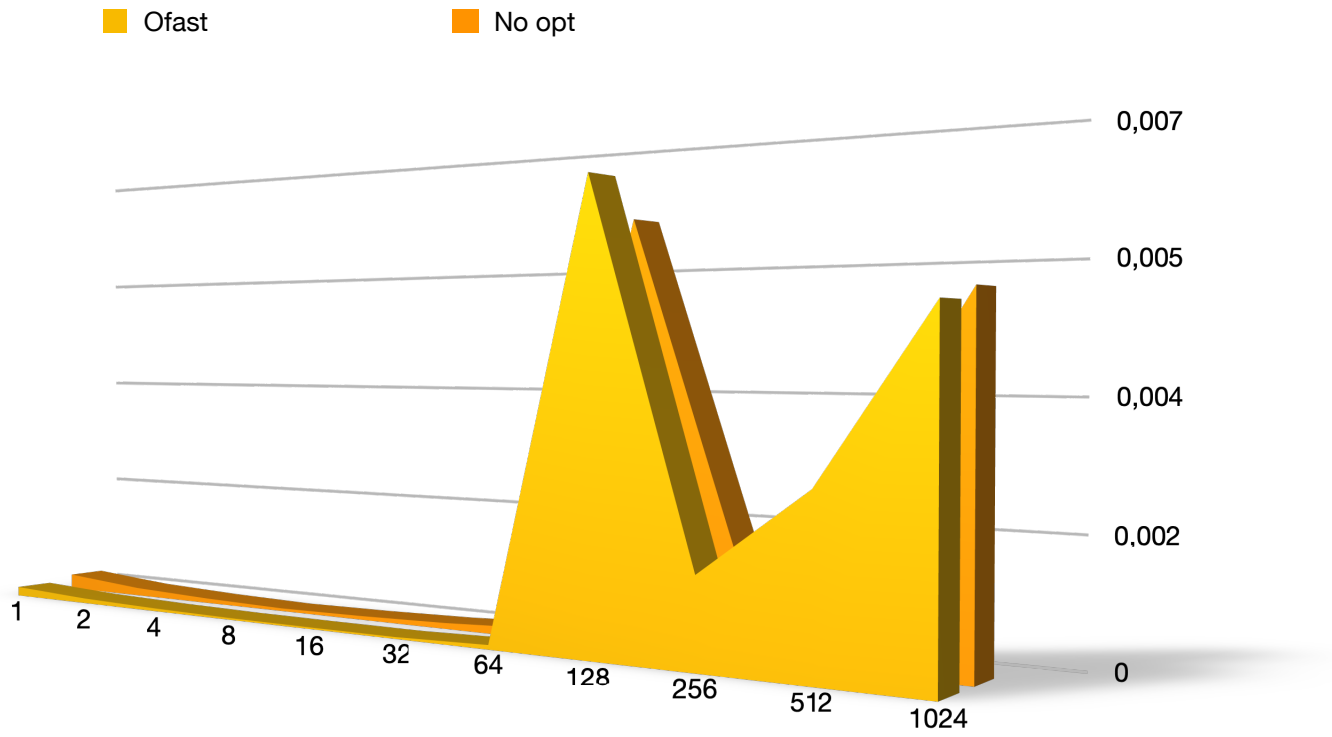
Результаты замеров времени выполнения показаны в следующих таблицах (для самого маленького и самого большого объема данных, первая – без оптимизаций компилятора, вторая – с опцией -ofast), полные измерения доступны по ссылкам: noopt, ofast.

Таблица результатов замеров времени вычисления без оптимизаций компилятора			
Mini-Dataset – размер матриц ~40		Extra-Dataset – размер матриц ~4000	
Потоки	Среднее время	Потоки	Среднее время
1	0,000232	1	242,506471
2	0,000121	2	121,835132
4	0,000070	4	61,475975
8	0,000041	8	31,529308
16	0,000055	16	19,604188
32	0,000086	32	17,869849
64	0,000149	64	17,458169
128	0,005878	128	10,945532
256	0,001297	256	10,273166
512	0,002412	512	9,858072
1024	0,004890	1024	9,666387

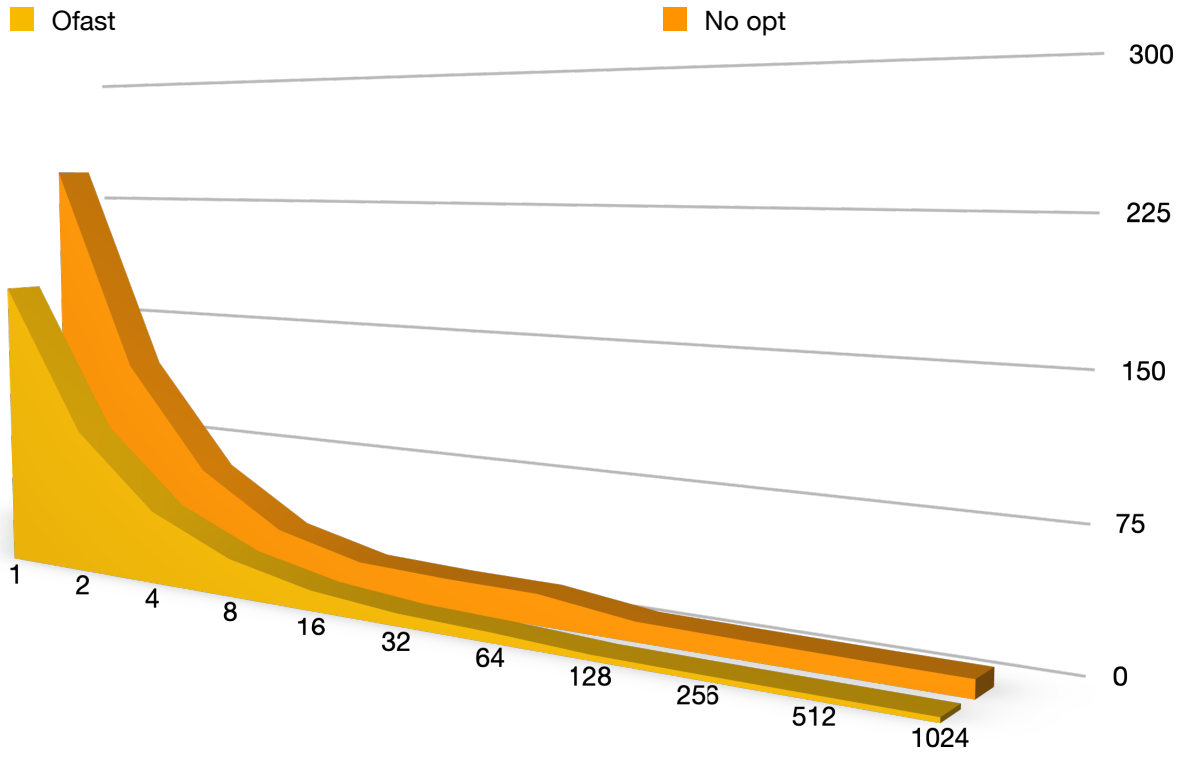
Таблица результатов замеров времени вычисления с оптимизациями компилятора -ofast			
Mini-Dataset – размер матриц ~40		Extra-Dataset – размер матриц ~4000	
Потоки	Среднее время	Потоки	Среднее время
1	0,000140	1	170,659687
2	0,000073	2	85,547864
4	0,000042	4	43,003723
8	0,000026	8	22,007962
16	0,000024	16	11,626688
32	0,000027	32	7,058639
64	0,000064	64	5,627997
128	0,006427	128	3,137430
256	0,001252	256	2,853611
512	0,002409	512	2,752253
1024	0,004704	1024	2,672031

Примерные графики зависимости времени выполнения от количества потоков

Mini dataset:



Extra dataset:



4. Выводы по полученным результатам

- С увеличением числа потоков производительность вычислений для большого объема данных повышается, для маленького объема данных повышение числа потоков с определенного момента ее ухудшает.
- Из-за того, что объем данных статичен и известен на этапе компиляции программы, компилятор справляется с оптимизацией алгоритма в большинстве случаев, что улучшает производительность.
- Оптимальное количество потоков для большого объема данных (размер матрицы более 800) – 128 потоков, при дальнейшем увеличении их числа наблюдалось замедление вычислений в общем случае.
- Оптимальное количество потоков для маленького объема данных (размер матрицы примерно 40) – 32 потока, при дальнейшем увеличении их числа наблюдалось существенное падение производительности.
- Из двух предыдущих пунктов можно сделать вывод, что оптимальное количество потоков для каждого объема данных пропорционально объему этих данных и количеству доступных вычислительных ядер (в ситуациях с самым большим объемом данных оптимальное количество потоков было, по-видимому, ограничено числом доступных ядер).