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

Практикум по курсу

«Распределенные системы»

Разработка отказо-устойчивой параллельной версии алгоритма тройного перемножения матриц

Отчет о проделанной работе

427 группа,
Махов А.М.

Москва, 2021 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 ранее требовалось:

1. Улучшить программу с использованием технологий OpenMP и MPI, 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 и оптимальную конфигурацию ядер на суперкомпьютере Bluegene

Новая задача подразумевает разработку отказоустойчивой версии решения

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 (СКиПОД, 2020)

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.

При написании MPI версии были использованы mpi-функции по отправке данных другим процессам и получению результатов вычислений (MPI_Send и MPI_Recv), функция барьерной синхронизации (MPI_Barrier), а также стандартные инициализирующие и завершающие mpi-функции. Исходные инициализированные матрицы были разбиты на строки и столбцы, согласно

количеству рабочих процессов; данные отправляются каждому рабочему процессу и главный процесс ждет от них результаты вычислений. Для вычисления на одном ядре используется базовый предоставленный алгоритм без каких либо модификаций.

Исходный код задачи доступен как в архивах во вложениях на сайте курса или в ветках репозитория автора. Код имеет базовую документацию для OMP ветки, которую можно найти в директории «*Docs*» проекта.

Компиляция OMP-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*.

Компиляция MPI-программы проводилась с помощью Apple Clang Version 12, x86_64-apple-darwin20.2.0 + libmpi.40.dylib – библиотека MPI версии 4.0.5 и адаптированных для MPI стандартных заголовочных файлов, поставлявшихся вместе с пакетом open-mpi для Mac OS (на локальной машине). На Bluegene программа компилировалась с помощью mpicc-обертки: с помощью gcc версии 4.1.2, powerpc-bgp-linux, Linux fen1 2.6.16.60-0.54.5-ppc64.
Общие опции компиляции MPI-программы на локальной и удаленной машине: -std=c99 -Wall -O3.

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/clang и библиотеки openmp/mpi (для Mac OS – пакет llvm и пакет *open-mpi* из репозитория *homebrew*)

3. Результаты замеров времени выполнения (СКиПОД, 2020)

Для запуска тестов OpenMP-программы 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.

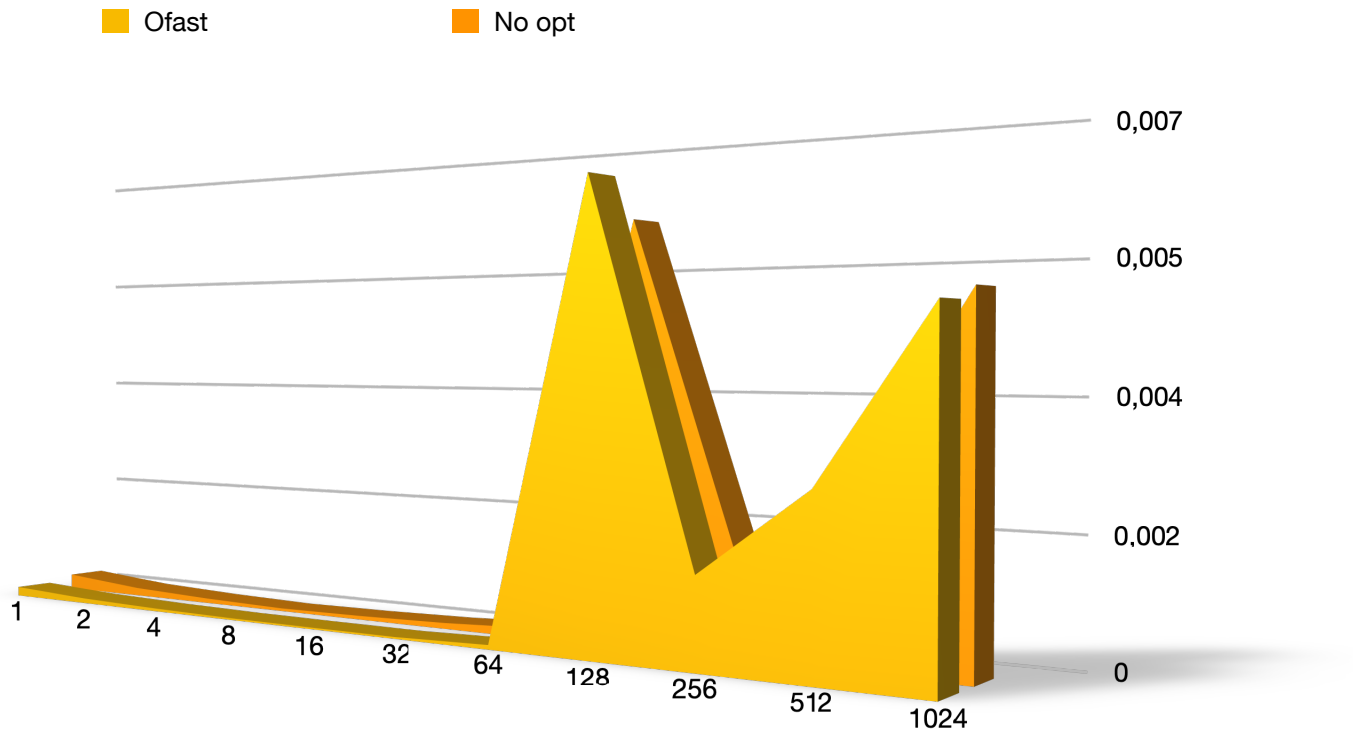
Для запуска (отправки заданий в очередь) был аналогичным образом составлен похожий shell-скрипт (*mpi-runner.sh*), также пересобирающий программу и

последовательно отправляющий каждую новую собранную программу в общую очередь на выполнение с задержкой в 20 секунд (чтобы не переполнялась очередь), результаты работы попадают в директорию ‘benchmarks/opt/‘.

Результаты замеров времени выполнения показаны в следующих таблицах (openmp: для самого маленького и самого большого объема данных, первая – без оптимизаций компилятора, вторая – с опцией -ofast; mpi: ввиду образовавшейся масштабной очереди на выполнение и ограниченное количество доступных конфигураций машин, замеры были выполнены для конфигураций от 32 до 512 ядер с опцией -O3, полные измерения доступны по ссылкам: noopt, ofast – openmp; fen – mpi).

Примерные графики зависимости времени выполнения от количества потоков для OpenMP-программы:

Mini dataset:



Extra dataset:

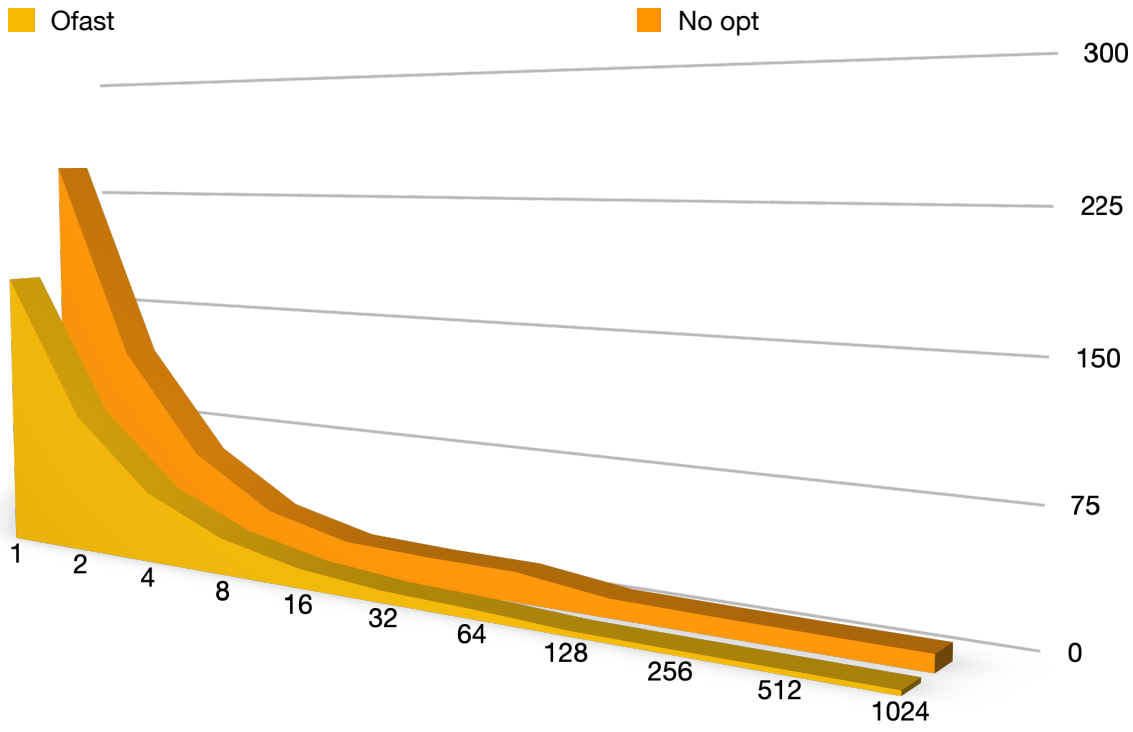
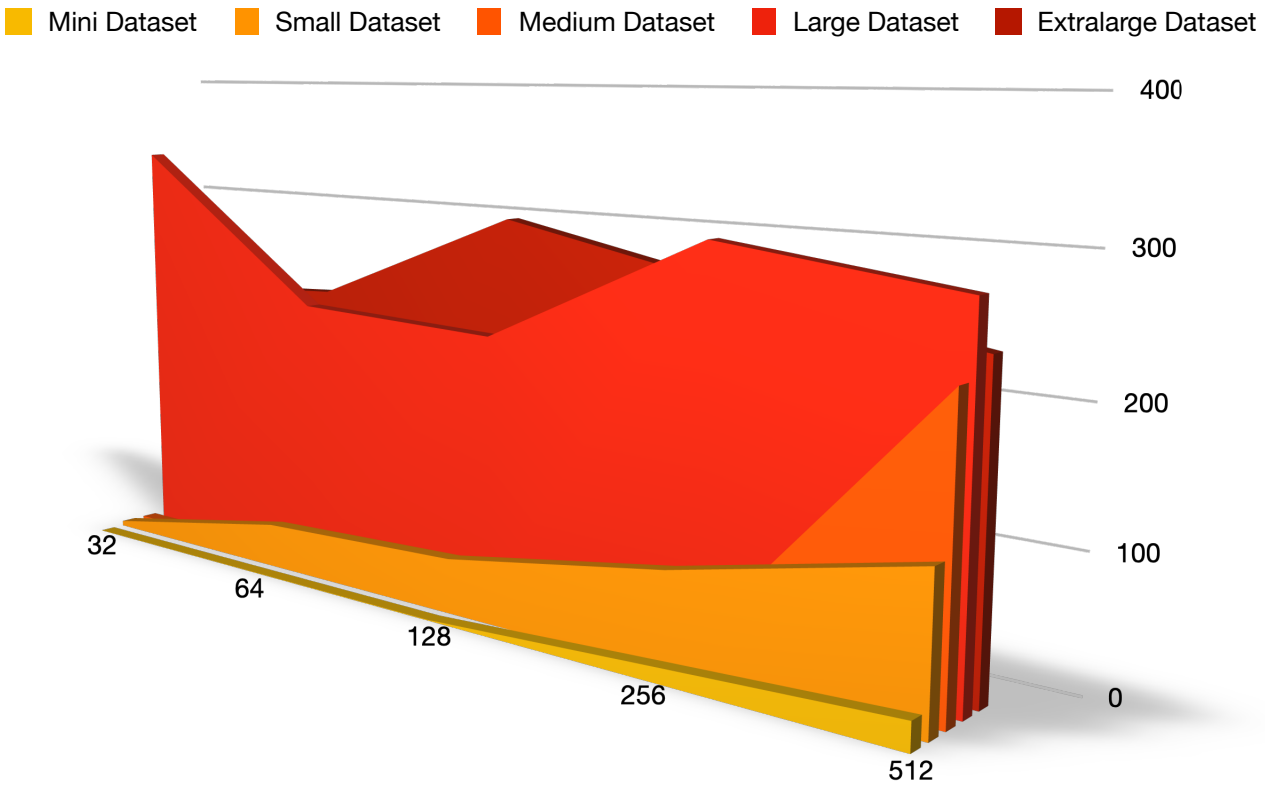


Таблица результатов замеров времени вычисления (OMP) без оптимизаций компилятора			
Mini-Dataset – размер матриц ~40		Extra-Dataset – размер матриц ~4000	
Потоки	Среднее время	Потоки	Среднее время
1	0,000232	1	242,506471
2	0,000121	2	121,835132
4	0,000070	4	61,475975
8	0,000041	8	31,529308
16	0,000055	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 (OMP) 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 для MPI-программы:



Для наглядности, результаты замеров также приведены в таблице:

	Mini	Small	Medium	Large	Extralarge
32	0,138902	4,094233	4,094238	335,493930	215,224397
64	0,266844	35,105109	3,991418	215,224695	225,114333
128	0,531212	42,916219	5,888566	209,357017	298,260861
256	11,057591	72,583579	10,801571	297,260897	263,780292
512	21,235282	113,363242	222,157217	275,495930	235,816025

4. Отказоустойчивость (MPI-FT, 2021)

Отказоустойчивость алгоритма обеспечивается:

1. Заменой всех блокирующих операций на их неблокирующие аналоги с последующим активным ожиданием завершения этих операций.
2. Созданием обработчика ошибок, который находит процессы, которые (в данном тестовом случае) были убиты и порождает новые вместо них.
3. Восстановлением данных из мастер-процесса (процесс с ранком 0) для новых процессов (в этой реализации – вычисления начинаются заново с определенного момента, каждый пройденный этап запоминается мастер-процессом)
4. Успешным завершением вычислений и отправкой данных обратно мастер-процессу.

Особенности реализации:

1. Добавление таймаутов по неблокирующим операциям заметно сказалось на общей производительности
2. Добавление восстановления после сбоя делает обязательным повторную отправку данных новым процессам (в данном решении отправка делается заного всем процессам, так как нельзя утверждать, что умер только один процесс)

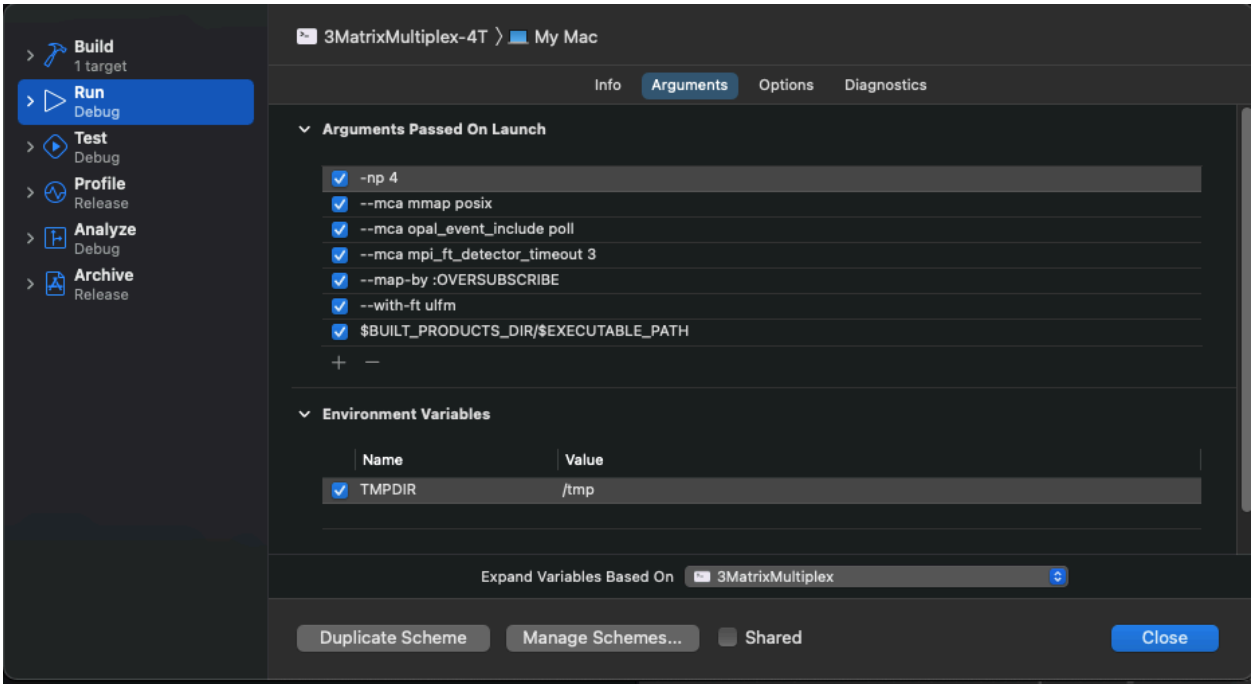
Платформа и версия MPI, на которой программа запускалась:

OS: 21.3.0 Darwin Kernel Version 21.3.0: Wed Dec 8 00:40:29 PST 2021; root:xnu-8019.80.11.111.1~1/RELEASE_X86_64 x86_64

Clang: Apple clang version 13.0.0 (clang-1300.0.29.30), Xcode-13 bundled version, Xcode Version 13.2.1 (13C100)

MPI: 4.1.2, ICLDiscko's fork, commit, Compiled with: --with-hwloc=internal --with-libevent=internal —with-ft=mpi using gcc: gcc version 11.2.0 (Homebrew GCC 11.2.0_3)

Параметры запуска с помощью обертки mpirun:



Примечание: —mca mpi_ft_detector_timeout 3 был использован для отладки операций по восстановлению работоспособности (MPI_Comm_spawn и работе с коммуникаторами)

Примерный лог работы программы можно посмотреть в файле *log.txt.*

5. 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 матриц более 800) – 128 потоков, при дальнейшем увеличении их числа наблюдалось замедление вычислений в общем случае на обоих машинах.
- Оптимальное количество потоков для маленького объема данных (размер матриц 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, в ситуации mpi – ограничено доступными наборами конфигураций машины Bluegene).
- Распараллеливание с помощью OMP оказалось быстрее чем распараллеливание с помощью MPI, как с точки зрения затраченного времени на разработку, так и с точки зрения скорости вычислений; скорость вычислений пострадала скорее всего из-за модели пересылки данных, используемой в MPI-версии – добавляются дополнительные расходы на ожидание и отправку данных. Также, возможно, машина Bluegene менее мощная, чем Polus. В замерах времени MPI-версии наблюдаются некоторые непоследовательные моменты, которые также могут быть связаны с задержкой отправки и получения данных между процессами.
- Добавление отказоустойчивости не только трудоемкая задача, но и серьезно ухудшает производительность программы в случае возникновения сбоя, но благодаря этому программа способна правильно выполнить поставленную задачу.