Московский Государственный Университет им. М. В. Ломоносова

Кафедра Вычислительных Технологий и Моделирования

Отчёт о практической работе по курсу "Суперкомпьютеры и параллельная обработка данных": Исследование параллельной масштабируемости метода вращений Гивенса QR-разложения матрицы с использование технологий OpenMP и MPI

Студент: Валиахметов Б. И.
Преподаватели: Воеводин Вл. В.
Антонов А. С.

Группа: 403

Москва
2021

1 Постановка задачи

Необходимо построить QR-разложение квадратной вещественной матрицы A методом вращений Гивенса. Математическое описание алгоритма и его последовательная версия приведены на портале AlgoWiki: по ссылке. Страница отчета на портале AlgoWiki: ссылка.

Отметим, что по сути, после применения вращений к исходной матрице мы получаем матрицу R и для получения матрицы Q необходимо выполнить обратные вращения над единичной матрицей. Однако на практике это не нужно, так как в большинстве случаев требуется умножать на Q или Q^*, а для этого достаточно применить подсчитанные ранее вращения. Также этот подход дает меньшую численную ошибку и выполняется быстрее.

Наибольший интерес представляет параллельная версия алгоритма, описанная в последующем разделе.

2 Параллельный алгоритм

Вычислительным ядром исследуемого метода являются вращения строк исходной матрицы. Нами был выбран блочный подход к реализации метода.

Для удобства будем полагать, что всевозможные размеры и параметры таковы, что все распределения производятся нацело (например, степени 2).

Зададим размер блока k и размер подблока m. Пусть нам доступны p процессов по t нити в каждом. Разобьем матрицу A на блоки размера k и распределим блочные столбцы по процессам так: столбец с номером j хранится на процессе с номером $j \mod p$, чтобы распределение количества вращений было более равномерным и узлы меньше простаивали в ожидании пересылок. Также это способствует активному использованию кэш-памяти. Такое распределение легко произвести, если матрица задается как функция вычисления ее элемента, иначе необходимо провести некоторые пересылки (их мы не учитываем при подсчете времени работы алгоритма). На каждом процессе блоки по блочным строкам (а сами блоки внутри - по стодбцам), так они задействуются совместно на шагах алгоритма.

Далее будем по очереди обнулять поддиагонали диагональных блоков и блоки ниже диагональных. Процесс, на котором хранится блок (ведущий), сначала вычисляет необходимые вращения для данного блока (процедура DLARTG библиотеки LAPACK), а затем рассылает их по остальным процессам (процедурой MPI_IBcast стандарта MPI). Во время неблокирующей пересылки ведущий процесс производит вращения на оставшиеся блоки в блочной строке и приступает к вычислению вращений для последующих блоков в столбце.

Как проводятся вращения на блоке, который сейчас не обнуляется? Каждый процесс (в том числе и ведущий), получив набор вращений, применяет их в той же блочной строке, что и процесс-отправитель. Каждый блок дополнительно разбивается на блочные подстолбцы ширины m и данные подблоки статически распределяются по нитям на процессе (директива omp parallel for библиотеки OpenMP), которые и производят вращения (процедура DROT библиотеки BLAS). Этим дополнительным разделением мы задействуем дополнительный ресурс параллелизма и, что не менее важно, эффективно используем кэш-память высокого уровня.

3 Реализация

Реализация алгоритма выполнена на языке C++11 с использованием стандартных библиотек языка, а также пакетов OpenMPI и OpenBLAS. Визуализация для результатов исследования параллельных свойств выполнена на языке Python3 с использованием библиотек

numpy и matplotlib в среде Jupyter Notebook. Код программы находится в репозитории по ссылке.

4 Численные эксперименты

Эксперименты по установлению параллельных свойств алгоритма проводились на кластере ИВМ РАН. Характеристики узлов кластера (полное описание по ссылке):

- Compute Node Arbyte Alkazar R2Q50 G5.
- 40 ядер (два 20-ядерных процессора Intel Xeon Gold 6230@2.10ГГц).
- Оперативная память: 256/384 Гб.
- Дисковая память: 480 Гб SSD.
- Операционная система: SUSE Linux Enterprise Server 15 SP2 (x86_64).

Были выбраны следующие параметры запусков.

1. На предварительных запусках были установлены оптимальные параметры блочного разбиения: $k = 128$, $m = 16$.
2. Размеры матриц выбирались в виде степеней 2: $N = 2^s, s \in \overline{8, 16}$.
3. Число процессов выбиралось в виде степеней 2: $P = 2^l, l \in \overline{0, 6}$, при условии $P \geq \frac{N}{k}$.
4. Число нитей выбиралось из набора: $T \in \{1, 2, 4\}$.
5. Для каждой конфигурации параметров проводилось 3 запуска, измеренное время работы усреднялось.

В таблицах 1, 2, 3 приведены результаты времени работы программы в зависимости от параметров. Некоторые запуски не проводились ввиду их слишком большой потенциальной длительности или слишком малого необходимого значения размера блока.

P \N	256	512	1024	2048	4096	8192	16384	32768	65536
1	0.0402	0.3432	2.824	29.84	498.4	N/A	N/A	N/A	N/A
2	N/A	0.0459	0.2231	1.372	9.505	99.99	807.177	N/A	N/A
4	N/A	0.03902	0.1529	0.7388	4.615	34.43	250.7	1555.2	N/A
8	N/A	N/A	0.3030	0.6821	3.046	15.33	89.11	588.2	N/A
16	N/A	N/A	N/A	0.6065	2.502	10.68	49.22	254.7	N/A
32	N/A	N/A	N/A	N/A	2.445	9.951	41.64	181.14	N/A
64	N/A	N/A	N/A	N/A	N/A	10.24	39.56	161.6	675.9

Таблица 1: Время работы в сек., $T = 1$ нить

Видим, что ускорение уже на 2 процессах существенно больше чем в 2 раза по сравнению с последовательной программой.

Ниже на Рисунках 1, 2, 3 приведены зависимости из таблиц выше.

На Рисунке 4 приведены графики зависимости времени работы программы от количества процессов при фиксированном размере матрицы $N \in \{4096, 81921\}$ и количество потоков $T = 1$. Видим, что на малом числе процессов ускорение почти линейное, затем

P \N	256	512	1024	2048	4096	8192	16384	32768	65536
2	N/A	0.08189	0.2260	1.274	8.117	56.26	416.2	N/A	N/A
4	N/A	0.06885	0.1688	0.7482	3.926	22.35	146.8	990.4	N/A
8	N/A	N/A	0.1630	0.6467	2.744	13.14	67.82	374.315	N/A
16	N/A	N/A	N/A	0.9567	3.663	14.84	64.07	294.9	N/A
32	N/A	N/A	N/A	N/A	3.785	14.70	57.52	239.5	N/A
64	N/A	N/A	N/A	N/A	N/A	14.54	55.85	223.2	911.6

Таблица 2: Время работы в сек., $T = 2$ нити

P \N	256	512	1024	2048	4096	8192	16384	32768	65536
2	N/A	0.1687	0.6239	2.654	13.17	76.14	487.3	N/A	N/A
4	N/A	0.1299	0.4685	1.905	8.141	40.02	214.4	1154.6	N/A
8	N/A	N/A	0.2759	0.9752	3.870	16.59	75.56	374.2	N/A
16	N/A	N/A	N/A	1.058	3.783	15.13	62.32	269.9	N/A
32	N/A	N/A	N/A	N/A	3.915	14.96	59.65	242.9	N/A
64	N/A	N/A	N/A	N/A	N/A	15.28	59.28	236.4	955.5

Таблица 3: Время работы в сек., $T = 3$ нитей

достигается некоторая граница и время начинает несколько расти из-за увеличения накладных расходов. Также это может быть связано с тем, что процессы располагаются на большем числе отдельных узлов и скорость передачи сообщений между ними меньше, чем внутри одного узла.

На Рисунке 5 приведены графики зависимости времени работы программы от количества процессов при фиксированном размере матрицы $N = 81921$ и различном количестве потоков $T \in \{1, 2, 4\}$. Можем заметить, что использование двух потоков дает практически двукратный выигрыш во времени по сравнению с одним, однако 4 потока работают уже дольше. Вероятно, это связано с накладными расходами на их синхронизацию и менее равномерным распределением задач.

5 Выводы

В рамках практической работы мы реализовали параллельную версию алгоритма QR-разложения квадратной вещественной матрицы методом вращений Гивенса и изучили масштабируемость написанной программы. Из результатов проделанной работы можем сделать следующие выводы:

1. Реализованная нами программа выполняет поставленную задачу.
2. Параллельная программа по сравнению с последовательной версией ускоряется в большее число раз, чем число процессов - это говорит о хороших параллельный свойствах алгоритма.
3. На малом (от 2 до 8) количестве процессов имеет место почти линейное ускорение программы.
4. При большом количестве процессов (≥ 16) или потоков (4) на исследованных размерах матриц программа заметно не ускоряется, так как накладные расходы и несбалансированность "перевешивают"ускорение.

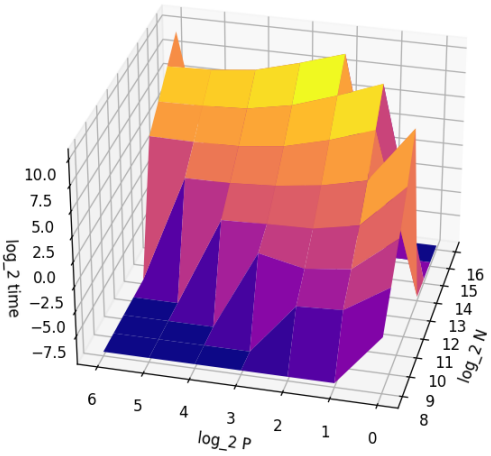


Рис. 1: $T = 1$

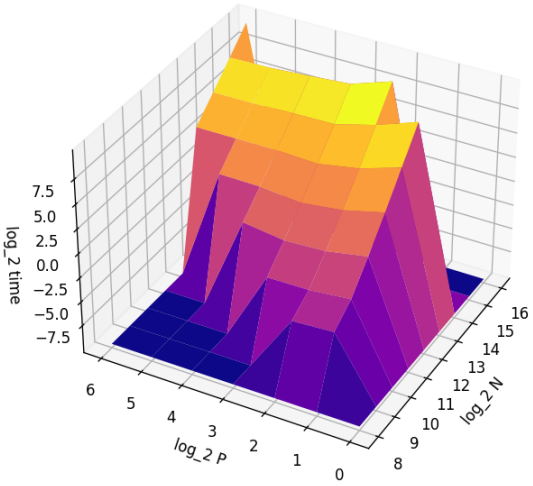


Рис. 2: $T = 1$

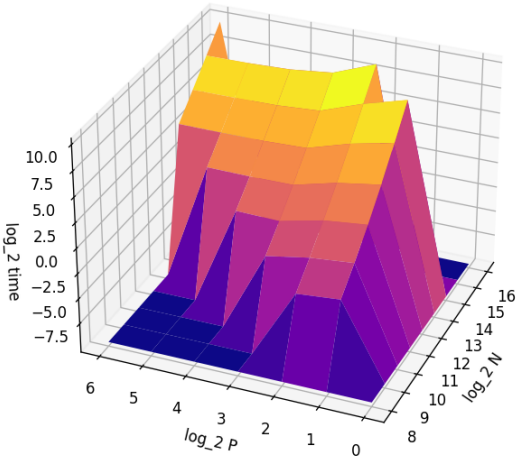
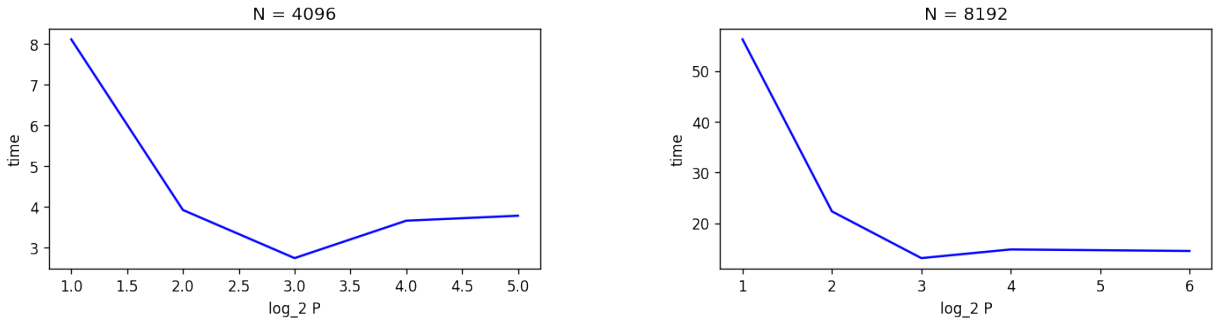


Рис. 3: $T = 1$



(a) $N = 4096$ (b) $N = 8192$

Рис. 4: $time(P)$

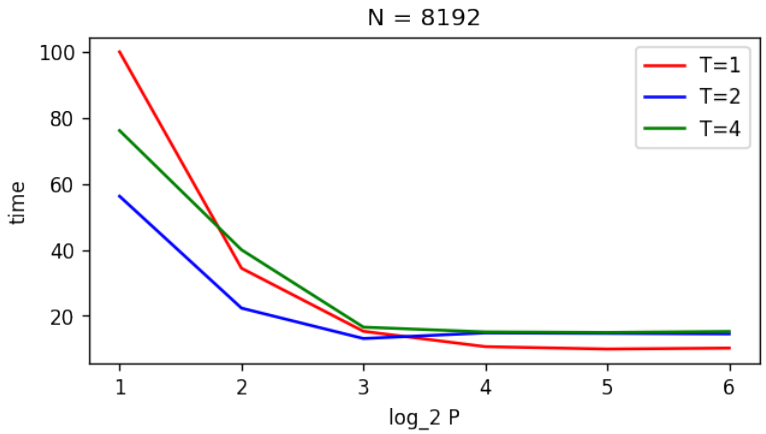


Рис. 5: $N = 8192$