Реализация алгоритма Штрассена на Intel(R) Xeon Phi(TM)

Е.С. Сергеев[1], О.В. Шаповалов[1], В.С. Чалышев, Д.И. Крыжановский[1]

ООО Сингулярис Лаб[1]

В данной работе рассматривается реализация параллельной версии алгоритма Штрассена для умножения матриц на архитектуре Intel(R) Xeon Phi(TM). Реализуется комбинированный подход, когда при достижении порогового значения из алгоритма Штрассена вызывается стандартный алгоритм умножения матриц. Целью данной работы является определение наиболее эффективной схемы распараллеливания, определение порогового значения размера матрицы для перехода на стандартный алгоритм, оптимизация потребления памяти, сравнение производительности алгоритмов с Intel(R) Math Kernel Library (Intel(R) MKL) DGEMM, определение зависимости точности вычислений от размеров матриц.

1. Введение

Умножение матриц – это одна из основных вычислительных операций. Вычислительная сложность стандартного алгоритма умножения матриц порядка N составляет $O(N^3)$. Существует более сложное для программирования решение на основе алгоритма Штрассена [1], которое позволяет сократить вычислительную сложность до $O(N^{\log_2 7})$. Рекурсивная природа данного алгоритма представляет большую сложность для эффективного распараллеливания на современных вычислительных системах и использования данных из памяти.

В данной статье мы описываем алгоритм Штрассена и предлагаем варианты его распараллеливания на платформе Intel(R) Xeon Phi(TM). Результаты вычислительных экспериментов сравниваются с результатами экспериментов, проведенных с помощью функции матричного умножения DGEMM из библиотеки Intel(R) MKL [2]. Работа выполнена при поддержки компании Intel Corporation.

2. Алгоритм Штрассена

2.1. Общее описание алгоритма

Рассматривается умножение квадратных матриц размера $N \times N$, где $N = p \cdot 2^k$ и $p < M$. Умножение двух матриц записывается как $C = A \cdot B$, где A, B и C матрицы размера $N \times N$. Метод Штрассена для умножения матриц основан на рекурсивном делении каждой перемножаемой матрицы на 4 подматрицы и выполнения операций над ними. Требование к размеру матриц ($N = p \cdot 2^k$ и $p < M$) нужно, чтобы было возможным разбить каждую матрицу на $4\frac{N}{2} \times \frac{N}{2}$ подматрицы и выполнить умножение по формуле 1. Под M здесь понимается пороговое значение размера матриц, после которого вызывается стандартный алгоритм.

$$C = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \cdot \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix} = \begin{bmatrix} (S_1 + S_4 - S_5 + S_7) & (S_3 + S_5) \\ (S_2 + S_4) & (S_1 - S_2 + S_3 + S_6) \end{bmatrix}, \quad (1)$$

где

$$\begin{aligned} S_1 &= (A_{1,1} + A_{2,2}) \cdot (B_{1,1} + B_{2,2}) \\ S_2 &= (A_{2,1} + A_{2,2}) \cdot B_{1,1} \\ S_3 &= A_{1,1} \cdot (B_{1,2} - B_{2,2}) \\ S_4 &= A_{2,2} \cdot (B_{2,1} - B_{1,1}) \\ S_5 &= (A_{1,1} + A_{1,2}) \cdot B_{2,2} \\ S_6 &= (A_{2,1} - A_{1,1}) \cdot (B_{1,1} + B_{1,2}) \\ S_7 &= (A_{1,2} - A_{2,2}) \cdot (B_{2,1} + B_{2,2}) \end{aligned} \tag{2}$$

Таким образом одна процедура умножения матриц размера $N \times N$ уменьшается до 7 умножений матриц размера $\frac{N}{2} \times \frac{N}{2}$ (при стандартном подходе требуется 8 умножений). Далее происходит разбиение каждого перемножения рекурсивно, до тех пор пока размер матриц не достигнет порогового значения размера матрицы для перемножения с помощью процедуры DGEMM. Алгоритм включает следующие шаги:

1) Рекурсивное деление матриц A и B до тех пор пока не достигнут пороговое значение размера матрицы.

2) Умножение матриц с помощью процедуры DGEMM из библиотеки Intel(R) MKL.

3) Получение результирующей матрицы C по выражениям (1) и (2).

2.2. Потребление памяти

В ходе перемножения матриц с помощью метода Штрассена требуется дополнительная память, выделяемая на каждом уровне рекурсии для хранения вспомогательных матриц. Глубина рекурсии равна $log_2 \frac{N}{M}$, где M - пороговое значение размера матрицы после которого вызывается DGEMM, N - размер перемножаемых матриц. Размер дополнительных матрицы на i-том уровне рекурсии $\frac{N}{2^i} \times \frac{N}{2^i}$. Общее количество требуемой дополнительной памяти:

$$2 \sum_{i=1}^{log_2 \frac{N}{M}} \left(\frac{N}{2^i} \right)^2 \tag{3}$$

Например, для $N = 8192$, $M = 512$ и при учете, что расчет выполняется с двойной точностью, получаем по формуле (3) 340 МБ дополнительной памяти без учета заранее выделенных матриц A, B и C.

3. Распараллеливание для Xeon Phi

3.1. Схема распараллеливания

Была реализована самая простая схема распараллеливания с использованием технологии OpenMP, все 7 умножений подматриц выполняются в отдельных потоках (потоки порождаются на всех уровнях рекурсии). Такая схема требует достаточно много дополнительной памяти, как показано в формуле 4.

$$13 \sum_{i=1}^{log_2 \frac{N}{M}} \left(\frac{N}{2^i} \right)^2 7^{i-1} \tag{4}$$

Например, для $N = 8192$, $M = 512$, получаем по формуле (4) 18.15 ГБ. Поскольку при распараллеливании на 7 потоков алгоритму Штрассена требуется больше памяти, чем есть на устройстве Intel(R) Xeon Phi(TM) SE10P/X (8 ГБ), необходимо сократить объем требуемой памяти за счет уменьшения параллелизма. Для этого была реализована схема,

при которой на каждом уровне рекурсии возможен вызов разных реализаций умножения (с разным количеством используемых потоков). Для поиска оптимальной конфигурации распараллеливания были добавлены разные реализации распараллеливания на одном уровне рекурсии:

- на 8 потоков (блочное умножение),
- на 7 потоков (Штрассена, все умножения матриц происходят в отдельных потоках),
- на 4 потока (первые 3 потока выполняют по 2 умножения, 4-й – одно),
- на 2 потока (первые 3 умножения происходят в 3 потока, затем 2 умножения – в два, и остальные 2 умножения – также в 2 потока)
- однопоточная реализация.

Таблица 1. Варианты распараллеливания

n	e	Описание схемы распараллеливания
8	4	Параллельно выполняются 8 умножений в 8 потоков, затем параллельно выполняются 4 сложения в 4 потока.
7	13	Параллельно выполняются 7 умножений для вычисления $S_1 ... S_7$.
4	5	В первом потоке вычисляются матрицы S_1 и S_2, во втором потоке вычисляются матрицы S_5 и S_6, в третьем потоке вычисляются матрицы S_7 и S_3, в четвертом потоке вычисляется матрица S_4.
2	4	вычисляются матрицы S_5, S_6 и S_7 в три потока, вычисляются матрицы S_1 и S_2 в два потока, вычисляются матрицы S_3 и S_4 в два потока.

В таблице 1 приведены сведения об использованных схемах распараллеливания. В колонке n – количество потоков (а также номер варианта распараллеливания для конфигурации), в колонке e – количество дополнительных матриц размером $\frac{N}{2} \times \frac{N}{2}$, требуемых для распараллеливания на одном уровне рекурсии.

Алгоритм умножения был реализован таким образом, чтобы на каждом уровне рекурсии можно было вызывать любую реализацию распараллеливания из описанных выше. Для подбора оптимальной конфигурации распараллеливания была реализована возможность передачи в программу параметра для управления четырьмя верхними уровнями распараллеливания (далее вызывается последовательная версия). Например, передав в качестве значения параметра “8-7-4-4”, на верхнем уровне будет использоваться 8-поточная реализация, затем семи-, четырех-, на четвертом уровне – четырехпоточая и далее последовательная реализация. В итоге будет порождено $8 \times 7 \times 4 \times 4$ потоков (данная конфигурация приведена для примера, такое большое количество программных потоков будет приводить к деградации производительности).

3.2. Результаты тестирования

Для подбора конфигурации распараллеливания были сгенерированы возможные комбинации из четырех значений (из 8, 7, 4 и 2), и затем с данными комбинациями были запущены тесты на производительность. Результаты тестирования отображены в таблице 2 (показаны только лучшие по времени тесты, которые прошли по объему памяти).

Таблица 2. Время выполнения алгоритма Штрассена на 244 потоках на матрицах размера 8192×8192

Конфигурация	Время, с.
2-4-7-2	6.259
4-4-2-2	6.035
4-4-4-2	6.084
4-2-4-2	6.435
4-2-7-2	6.58
8-2-2-2	5.352

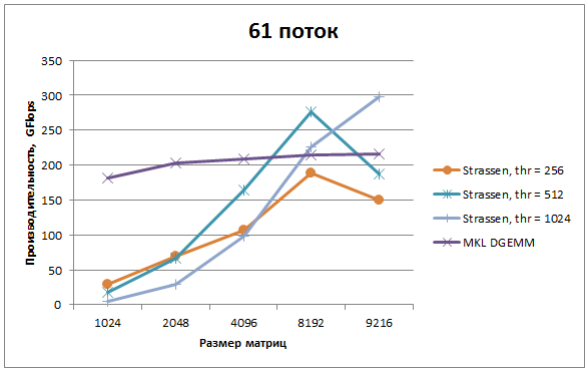
Наиболее эффективной конфигурацией оказалась 8-2-2-2 – на первом уровне рекурсии в 8 потоков выполняется блочное умножение, а затем вызывается алгоритм Штрассена с распараллеливанием на 2 потока.

На рисунке 1 показано потребление памяти для конфигурации 8-2-2-2.

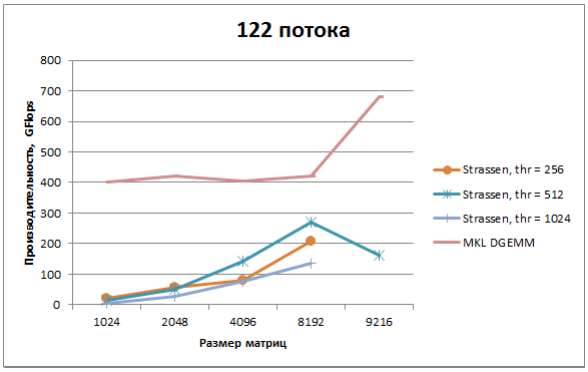
Рис. 1. Потребление памяти для конфигурации 8-2-2-2

Было произведено тестирование реализации алгоритма Штрассена и MKL DGEMM при заданном KMP_AFFINITY=granularity=core,balanced.

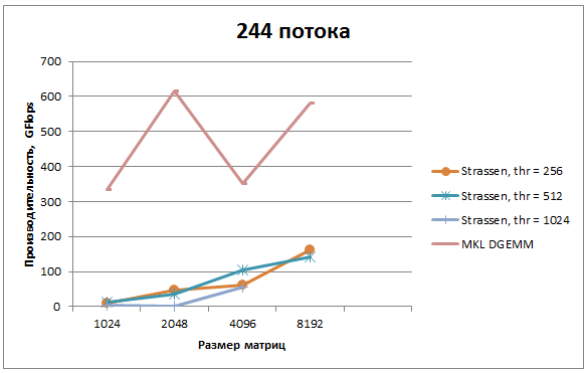
Результаты тестирования для конфигурации 8-2-2-2 приведены на рисунках 2. При данной конфигурации на 61 потоках удается получить результат лучше чем у Intel(R) MKL DGEMM на размерах матриц 8192×8192 и 9216×9216. Тестирование производилось на Intel(R) Xeon Phi(TM) SE10P/X, mpss 2.1.5889-14, компилятор Intel(R) Composer XE 2013.5.192, версия Intel(R) MKL 11.0.5.



(a) для 61 потоков



(b) для 122 потоков



(c) для 244 потоков

Рис. 2. Производительность алгоритма Штрассена для разного размера матриц

Для тестирования точности используется тест, предложенный в статье [5]:

$$A = I + uv^T, B = I - \frac{1}{1 + v^T}, C = I,$$

где u и v заданы как:

$$u_i = \frac{1}{N + 1 - i}, v_i = \sqrt{i}, i = 1 \dots N.$$

Результаты тестирования точности приведены на рисунке 3. Погрешность не зависит от конфигурации и одинакова для последовательной и параллельных версий.

(a) Абсолютная ошибка

(b) Относительная ошибка

Рис. 3. Погрешность алгоритма Штрассена

4. Анализ результатов и выводы

Из графиков, представленных на рисунке 2 видно, что пороговое значение размера матрицы для перехода на Intel(R) MKL DGEMM не сильно зависит от количества потоков. Однако зависит от размеров матриц: при $N \leq 4096$ эффективным является значение $M = 256$, при $N \leq 8192$ эффективным является значение $M = 512$, для $N = 9216 — 1024$. Можно сделать вывод, что при больших значениях N это значение будет увеличиваться.

Алгоритм Штрассена дает преимущество при малом количестве потоков, но плохо подходит для реализации на массивно-параллельной архитектуре Intel(R) Xeon Phi(TM). Во-первых, он требует большое количество дополнительной памяти (особенно при распараллеливании), что ограничивает максимальный размер перемножаемых матриц. Во-вторых, рекурсивная структура алгоритма хуже поддается распараллеливанию, так как схема распараллеливания получается многоуровневой с синхронизацией между этими уровнями, также схема алгоритма Штрассена обладает менее регулярным доступом к памяти по сравнению со стандартным алгоритмом.

Литература

1. V. Strassen. Gaussian elimination is not optimal. Numerische Mathematik, 13:354-356, 1969.
2. Intel(R) Math Kernel Library // https://software.intel.com/en-us/intel-mkl
3. Nicholas J. Higham, Exploiting fast matrix multiplication within the level 3 BLAS. ACM Trans. Math. Softw. 16, 4 (December 1990), 352-368.
4. Junjie Li, Sanjay Ranka, and Sartaj Sahni, Strassen's Matrix Multiplication on GPUs. In Proceedings of the 2011 IEEE 17th International Conference on Parallel and Distributed Systems (ICPADS '11). IEEE Computer Society, Washington, DC, USA, 157-164.
5. Kaporin, I. (1999), A practical algorithm for faster matrix multiplication. Numer. Linear Algebra Appl., 6: 687–700