Московский физико-технический институт (государственный университет)
Физтех-школа радиотехники и компьютерных технологий
Кафедра информатики и вычислительной техники

# Реализация алгоритма быстрого умножения матриц для архитектуры Эльбрус

Студент: Карпочев Б. А.
Научный руководитель: Логинов В. Е.

Москва, 2023

Области применения умножения матриц:

- Численные методы
- Машинное обучение и нейронные сети

Матричное умножение эффективно реализовано для CPU, GPU

Многие алгоритмы стараются свести к матричному умножению

Во многих областях приходится умножать матрицы больших размеров(миллионы элементов), где алгоритмы быстрого умножения матриц более эффективны

# Введение

## Библиотека EML

Цели:

- Ускорение процесса разработки конечного пользователя
- Увеличение эффективности пользовательских программ

Принципы:

- Низкие накладные расходы
- Простая структура
- Общие и базовые операции
- Атомность
- Самодостаточность

EML – высокопроизводительная мультимедийная библиотека

Аналоги: MKL, IPP, OpenBLAS, PLASMA

# Введение

## Методы быстрого умножения матриц

- Обычное матричное умножение GEMM: $O(n^3)$
- Алгоритм Винограда GEMMW: $O(n^{2.81})$

В данном алгоритме 7 умножений, против 8 при обычном блочном умножении

Можем рекурсивно редуцировать размерность

# Цель и задачи работы

Цели:

- Реализация алгоритма Винограда быстрого умножения матриц для архитектуры Эльбрус

Задачи:

- Реализация однопоточной версии алгоритма
- Реализация многопоточной версии алгоритма
- Сравнение различных версий алгоритма

# Алгоритм Винограда

$$
\begin{aligned}
Q_1 &= (A_{11} + A_{22}) \cdot (B_{11} + B_{22}) \\
Q_2 &= (A_{21} + A_{22}) \cdot B_{11} \\
Q_3 &= A_{11} \cdot (B_{12} - B_{22}) \\
Q_4 &= A_{22} \cdot (B_{21} - B_{11}) \\
Q_5 &= (A_{11} + A_{12}) \cdot B_{22} \\
Q_6 &= (A_{21} - A_{11}) \cdot (B_{11} + B_{12}) \\
Q_7 &= (A_{12} - A_{22}) \cdot (B_{21} + B_{22})
\end{aligned}
$$

$$
\begin{aligned}
C_{11} &= Q_1 + Q_4 - Q_5 + Q_7 \\
C_{12} &= Q_3 + Q_5 \\
C_{21} &= Q_2 + Q_4 \\
C_{22} &= Q_1 - Q_2 + Q_3 + Q_6
\end{aligned}
$$

$$
\begin{matrix} n/2 \\ n/2 \end{matrix}
\left\{ \begin{matrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{matrix} \right.
=
\begin{matrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{matrix}
\cdot
\begin{matrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{matrix}
$$

$$F(n) = 7 \cdot F(n/2) + O(n^2)$$

$$F(n) = \Theta\left(n^{\log_2 7}\right)$$

$$\log_2 7 \approx 2.81$$

# Реализация однопоточной версии

## Проблема нечетномерных матриц

Стандартный алгоритм работает с квадратными матрицами размерности $n = m \times k$, где $m$ и $k$ — четные

По свойствам блочного умножения матриц:

$$\begin{pmatrix} A & O \\ O & O \end{pmatrix} \begin{pmatrix} B & O \\ O & O \end{pmatrix} = \begin{pmatrix} AB & O \\ O & O \end{pmatrix}$$

где $O$ нулевая матрица

Матрицы произвольного размера можно дополнить до квадратных матриц размера $n = m \times k$

С такими матрицами алгоритм умеет работать

# Реализация однопоточной версии

## Проблема нечетномерных матриц

Заметим, что дополнять матрицы нулями можно до разных размеров

Существует оптимальный размер, который был получен в статье М. Шульца об оптимальном окаймлении матриц

Алгоритм нахождения:

$$m_0 = n,\ m_{k+1} = \lceil m_k \times 2^{-1} \rceil,\ \text{повторять пока } m_k \neq 1, 3, 5, 7, 9, 13, 17, 21$$

Позволяет уменьшить потребление памяти, по сравнению с более простыми алгоритмами

# Реализация однопоточной версии

## Проблема нечетномерных матриц

С другой стороны, вместо дополнения нулями можно воспользоваться следующим правилом:

$$\begin{pmatrix} A_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} B_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11} + a_{12}b_{21} & A_{11}b_{12} + a_{12}b_{22} \\ a_{21}B_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$

Если на вход подается нечетномерная матрица, выделяем из нее четномерную, а оставшиеся столбец/строку обрабатываем отдельно

# Реализация однопоточной версии

## Проблема нечетномерных матриц

Перепишем уравнения в следующем виде:

$$C_{11} = A_{11}B_{11} + a_{12}b_{21}$$

$$c_{12} = \begin{pmatrix} A_{11} & a_{12} \end{pmatrix} \begin{pmatrix} b_{12} \\ b_{22} \end{pmatrix}$$

$$\begin{pmatrix} c_{21} & {}_{22} \end{pmatrix} = \begin{pmatrix} a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} B_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

Первую операцию можно выполнить функцией GER, оставшиеся две с помощью GEMV

Плюсы подхода:

- Не требуется дополнительная память как в случае дополнения нулями
- Больше производительность из-за отсутствия необходимости копировать матрицы

# Реализация однопоточной версии

## Точка выхода из рекурсии

Полностью рекурсивные алгоритмы на практике имеют меньшую производительность, по сравнению со своими комбинированными версиями

Возникает потребность выбора оптимального размера для выхода из рекурсии, после которого вызывается GEMM

Данный размер на практике ищется эмпирически

# Реализация однопоточной версии

## Точка выхода из рекурсии

Тесты показали, что данный размер достаточно большой $N \approx 1000$

Отказ от рекурсии, всегда делаем один шаг алгоритмом Винограда, затем вызывается GEMM
Плюсы:

- Более понятная реализация
- Больше возможностей для оптимизаций компилятора
- В случае нечетномерных матриц операция дополнения/разрезания выполняется один раз

Этот алгоритм называется одношаговым алгоритмом Винограда, аналогично можно сделать двушаговый, трехшаговый и т. д.
Больше шагов — больше размер для вызова GEMM

# Реализация многопоточной версии

Многопоточная версия была реализована для систем с общей памятью на OpenMP

Был использован механизм task'ов(задач). Каждый рекурсивный вызов объявляется задачей и выполняется параллельно

Тесты проводились на восьмиядерном Эльбрус-8С. Ускорение и эффективность равны соответственно:

$$S = \frac{T_{\text{послед}}}{T_{\text{паралл}}} = 6 \qquad P = \frac{S}{n_{\text{потоков}}} = 0.75 \qquad N = 4096$$

Низкая эффективность и ускорение связаны с накладными расходами на синхронизацию задач в процессе исполнения

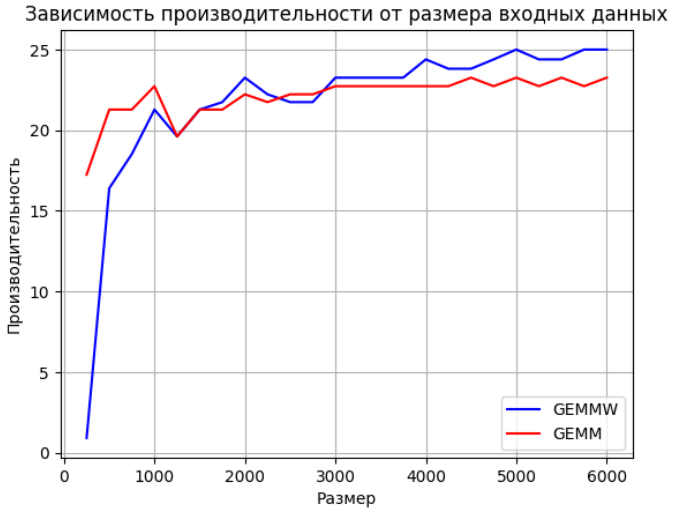
# Сравнение различных версий алгоритма

Используется следующая модель производительности:

$$\text{effective GFLOPS} = \frac{2mnk}{\text{time(sec)}} \cdot 10^{-9}$$

# Сравнение различных версий алгоритма

Эльбрус-8С



# Сравнение различных версий алгоритма

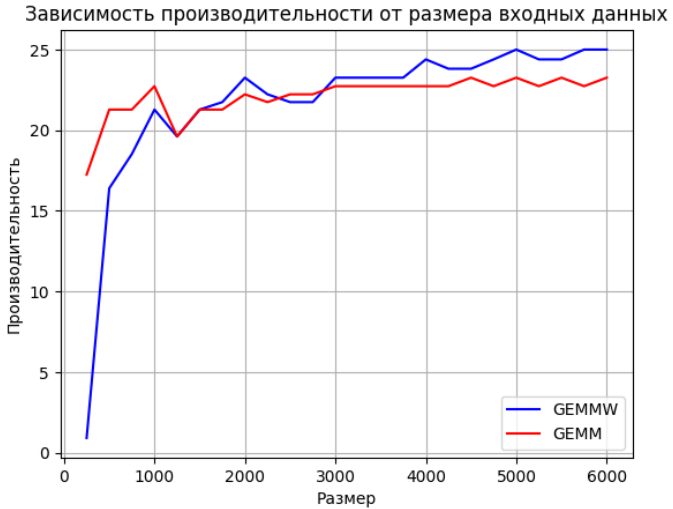
Эльбрус-8С, режим защищенных вычислений

# Сравнение различных версий алгоритма

Сравнение подходов дополнения нулями и разрезания матриц

# Сравнение различных версий алгоритма

Эльбрус-16С



# Результаты

Результаты:

- Реализована однопоточная версия алгоритма Винограда, вошедшая в библиотеку EML
- Полная поддержка интерфейса функции GEMM
- Найден оптимальный размер для вызова GEMM
- На процессоре Эльбрус-8С получено ускорение 2%, на матрицах размерности 1500, на 5%, начиная с матриц размерности 2000
- Реализована параллельная версия алгоритма, эффективность на матрицах $\sim 4000$ $P = 6$