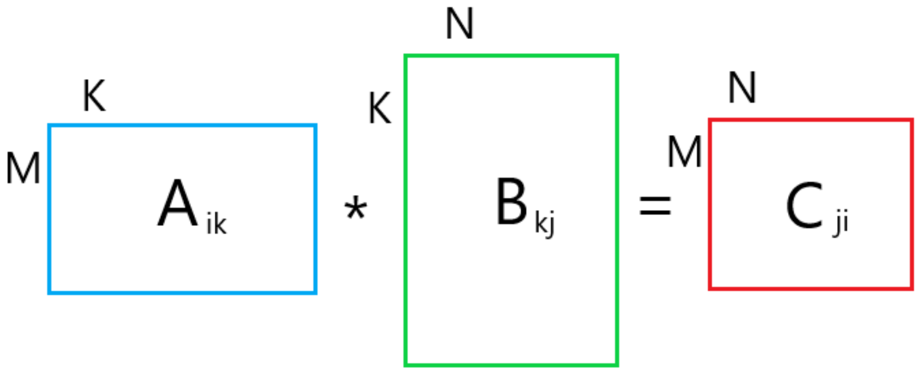
Умножение плотных матриц

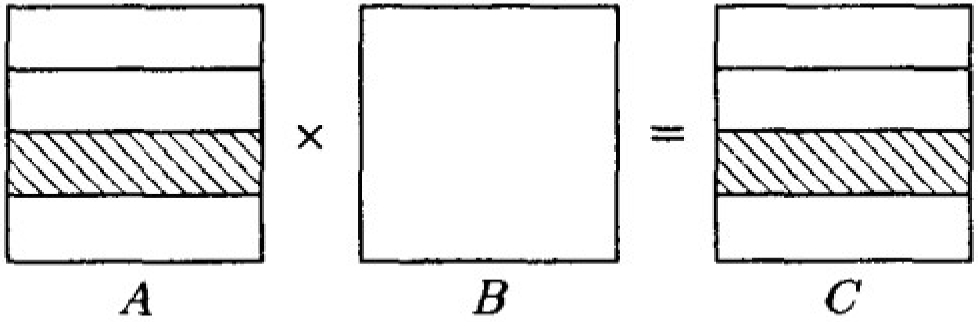
Комаров Артем Б05-872б

Описание

- (7) Умножение двух плотных матриц (A*B=C).
- Плотные квадратные матрицы со случайными элементами.
- Строчно-столбцовое распределение данных по процессорам
- Без дублирования начального и конечного распределения данных, например, A и C по блочным строкам, B - по столбцам.



Простейший алгоритм



Матрица A делится и распределяется по процессам.
Матрица B полностью копируется во все процессы.

Алгоритм не оптимален с точки зрения использования памяти процессами

Алгоритмы

1)

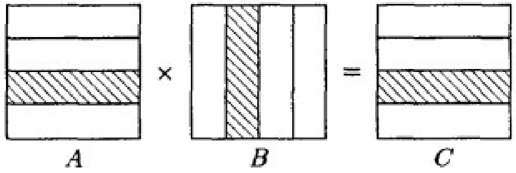


Рис. 2.1. Разрезание данных для параллельного алгоритма произведения двух матриц при вычислении на кольце компьютеров. Выделенные полосы расположены в одном компьютере

2)

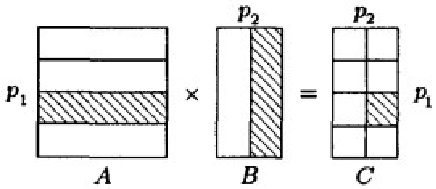


Рис. 2.3. Разрезание данных для параллельного алгоритма произведения двух матриц при вычислении в 2D решетке компьютеров. Выделенные данные расположены в одном компьютере

3)

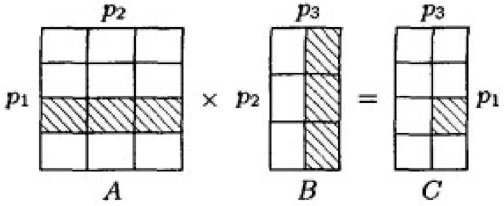


Рис. 2.5. Разрезание данных для параллельного алгоритма произведения двух матриц при вычислении в 3D решетке компьютеров

Алгоритмы

1)

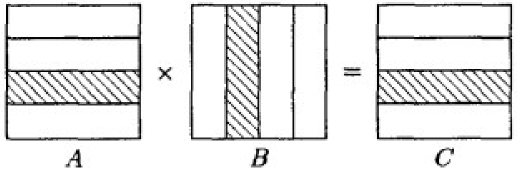


Рис. 2.1. Разрезание данных для параллельного алгоритма произведения двух матриц при вычислении на кольце компьютеров. Выделенные полосы расположены в одном компьютере

Выбран данный алгоритм поскольку:

1) Оптимален с точки зрения использования процессов (линейно от количества кусков)
2) Оптимален с точки зрения использования памяти по причние (1)
3) Интереснее эксперимент (используются больше пересылок)
4) Для равного количества процессов из-за большого числа взаимодействий алгоритм 1 будет работать хуже 2 и 3, НО:
если нас НЕ интересует, сколько памяти будут использовать процессы, то используйте простейший алгоритм параллелизма!

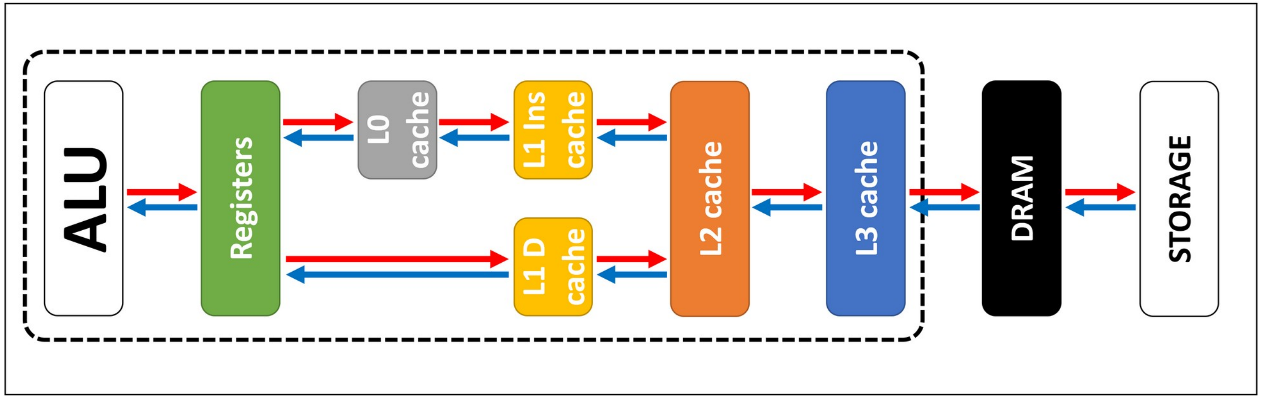
Другой алгоритм/дополнение: Блочное умножение матриц

Для максимальной скорости работы программы следует построить ее так, чтобы было как можно меньше «cache miss».

Для этого можно организовать матрицу в блоки. Размер блока можно подобрать по размеру какого-то уровня кэша.

По этой же причине матрицу «B» разумнее формировать как «column major»

$$C = AB = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \dots & \dots & \dots & \dots \\ A_{n1} & A_{n2} & \dots & A_{nn} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} & \dots & B_{1n} \\ B_{21} & B_{22} & \dots & B_{2n} \\ \dots & \dots & \dots & \dots \\ B_{n1} & B_{n2} & \dots & B_{nn} \end{bmatrix}$$



Теоретическое ускорение

$$S = S(p) = T(1)/T(p) = T_a/(T_a/p + T_c/p) = pT_a/(T_a + T_c)$$
$$= p/(1 + T_c/T_a) = p/(1 + (\tau_c L_c)/(\tau_a L_a)) = p/(1 + \tau L).$$

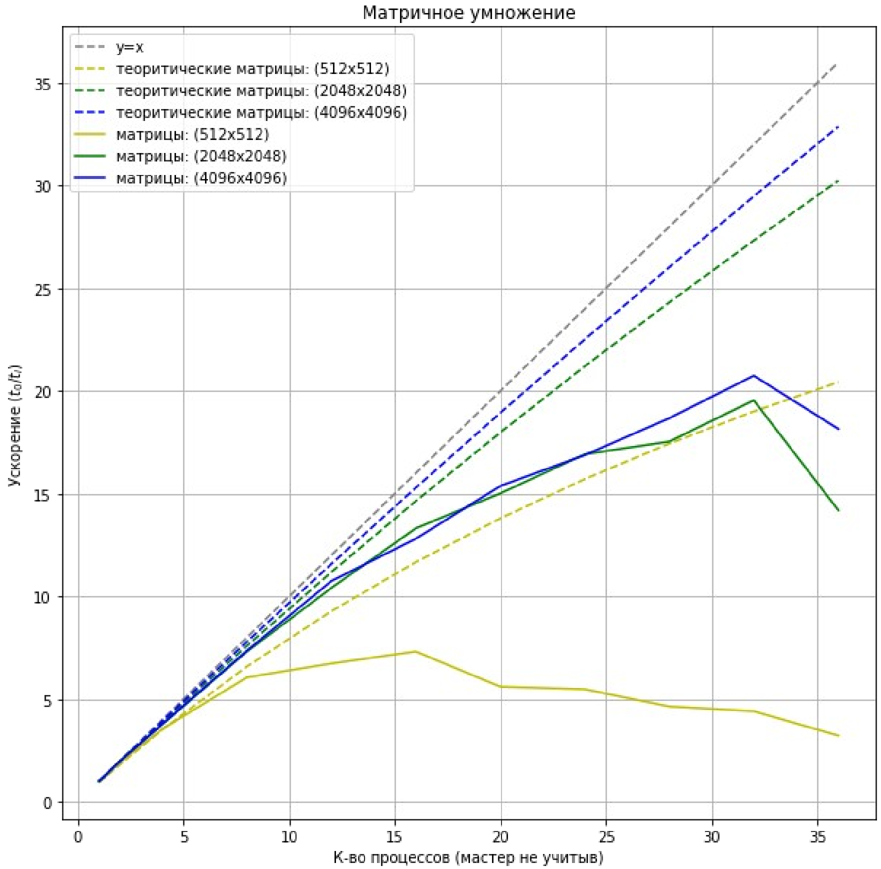
$L = L_c/L_a,$ — отношение общей длины сообщений к числу арифметических операций

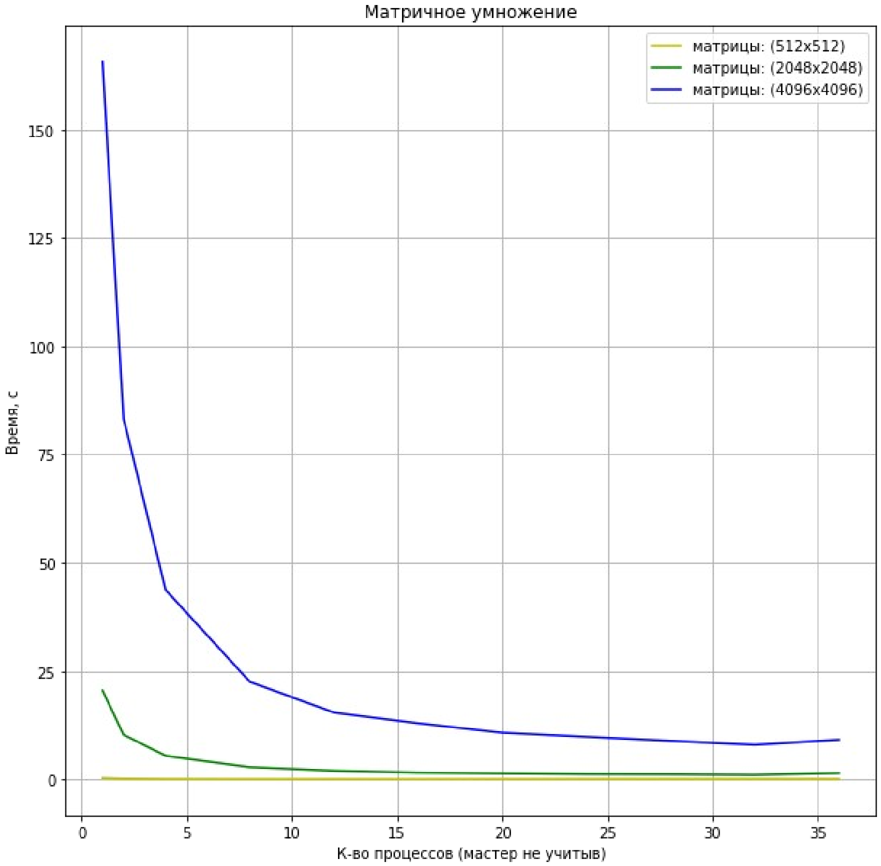
$\tau = \tau_c/\tau_a,$ — отношение времени передачи одного сообщения к времени исполнения одной арифметической операции

Для матриц A(n x m), B(m x k) примерное теоретическое ускорение:

S = p / (1 + 10 * (2 * n * m / p + m * k) / ((n / p) * m * k))
= (если n=m=k)
= p / (1 + (10 * (2 + p)) / n)

Произвдение матриц разных размеров (Алгоритм 1)





Выводы

- Для произведения небольших матриц, с небольшим числом процессоров используйте простейший алгоритм
- Важным источником ускорения программ является эффективное использование кэша
- Воспроизведен и проверен алгоритм 1 параллельного матричного произведения. Эксперимент показал эффективность алгоритма с точки зрения производительности и потребления памяти
- Посчитано теоретическое ускорение, но оно далеки от реальности из-за огромного влияния особенности вычислений

Спасибо за внимание!