Московский Государственный Университет им. М.В. Ломоносова
Факультет Вычислительной Математики и Кибернетики
Кафедра Суперкомпьютеров и Квантовой Информатики

Спецкурс: системы и средства параллельного программирования.

Отчёт № 2.
Анализ блочного умножения матриц.

Работу выполнил
Федоров В. В.

Москва 2020

Постановка задачи и формат данных.

Задача: Реализовать блочный алгоритм матричного умножения и проанализировать следующие параметры:

- Время выполнения рабочего цикла программы (Wall clock)
- Число промахов кэша L1
- Число промахов кэша L2
- Число процессорных тактов
- Число операций над числами с плавающей точкой
- Число промахов TLB

Оптимальный размер блока вычисляется по формуле:

$$3 * b^2 * sizeof(float) = mL$$

Где mL – размер кэша L1 в байтах, b – размер блока. Таким образом, мы умещаем 3 квадратных блока $b \times b$ в кэше L1.

Замеры нужно выполнить для следующих видов умножения:

- Порядок индексов ijk, размер блока стандартный
- Порядок индексов ikj, размер блока стандартный
- Порядок индексов ikj, размер блока оптимальный

Формат командной строки: <имя файла матрицы A> <имя файла матрицы B> <имя файла матрицы C> <режим порядка индексов> <режим размеров блоков> [размеры блоков].

Режимы порядка индексов: 0 – ijk, 1 – ikj.

Режимы размеров блоков: d – стандартные, по 32; o – оптимальные, по формуле (см. выше), c – задаются пользователем следующими тремя параметрами командной строки.

Формат файла-матрицы: Матрица представляются в виде бинарного файла следующего формата:

Тип	Значение	Описание
Число типа size_t	N – натуральное число	Число строк матрицы
Число типа size_t	M – натуральное число	Число столбцов матрицы
Массив чисел типа float	$N \times M$ элементов	Массив элементов матрицы

Элементы матрицы хранятся построчно.

Описание алгоритма.

Математическая постановка: Матрицы A, B и C разбиваются на блоки заданного размера, например:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \cdots & \mathbf{A}_{1t} \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \cdots & \mathbf{A}_{2t} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{s1} & \mathbf{A}_{s2} & \cdots & \mathbf{A}_{st} \end{bmatrix}$$

В таком случае матрицы можно перемножать поблочно:

$$C_{ij} = \sum_{k=1}^{t} A_{ik} B_{kj}$$

Для упрощения вычислений размеры матриц увеличивались до кратных соответствующим размерам блока, дополнительные элементы инициализировались нулями.

Анализ данных программы: Для оценки указанных данных использовались следующие ресурсы:

- Wall clock: замер делался с помощью функции `gettimeofday()` из библиотеки `sys/time.h`
- Число промахов кэша L1: использовалось PAPI-событие `PAPI_L1_DCM`

- Число промахов кэша L2: использовалось PAPI-событие `PAPI_L2_DCM`
- Число тактов процессора: использовалось PAPI-событие `PAPI_TOT_CYC`
- Число операций с плавающей точкой: использовалось PAPI-событие `PAPI_FP_OPS`
- Число промахов TLB: не замерялось, так как соответствующее PAPI-событие `PAPI_TLB_DM` аппаратно не поддерживается на компьютере, на котором проводились замеры

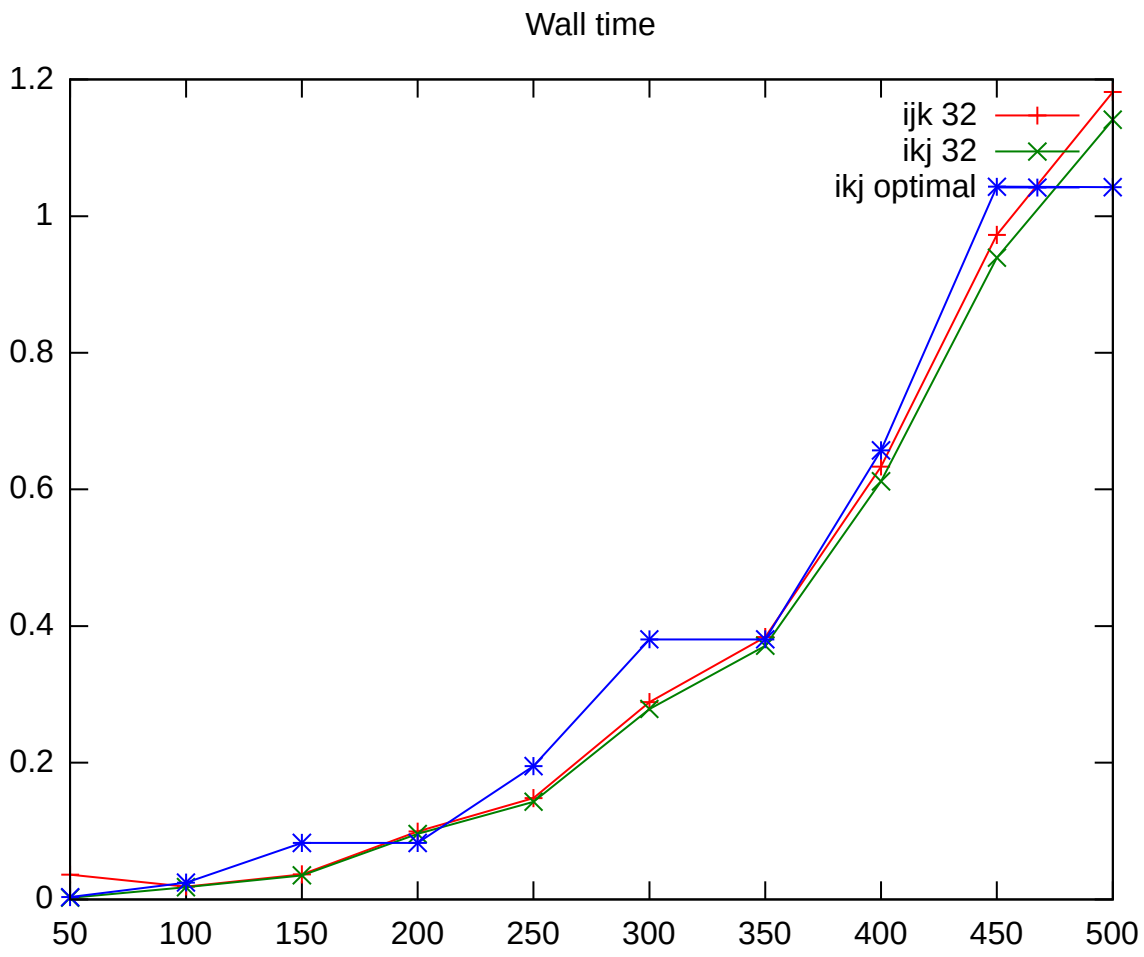
Верификация: Для проверки корректности работы программы использовалось 20 тестовых примеров.

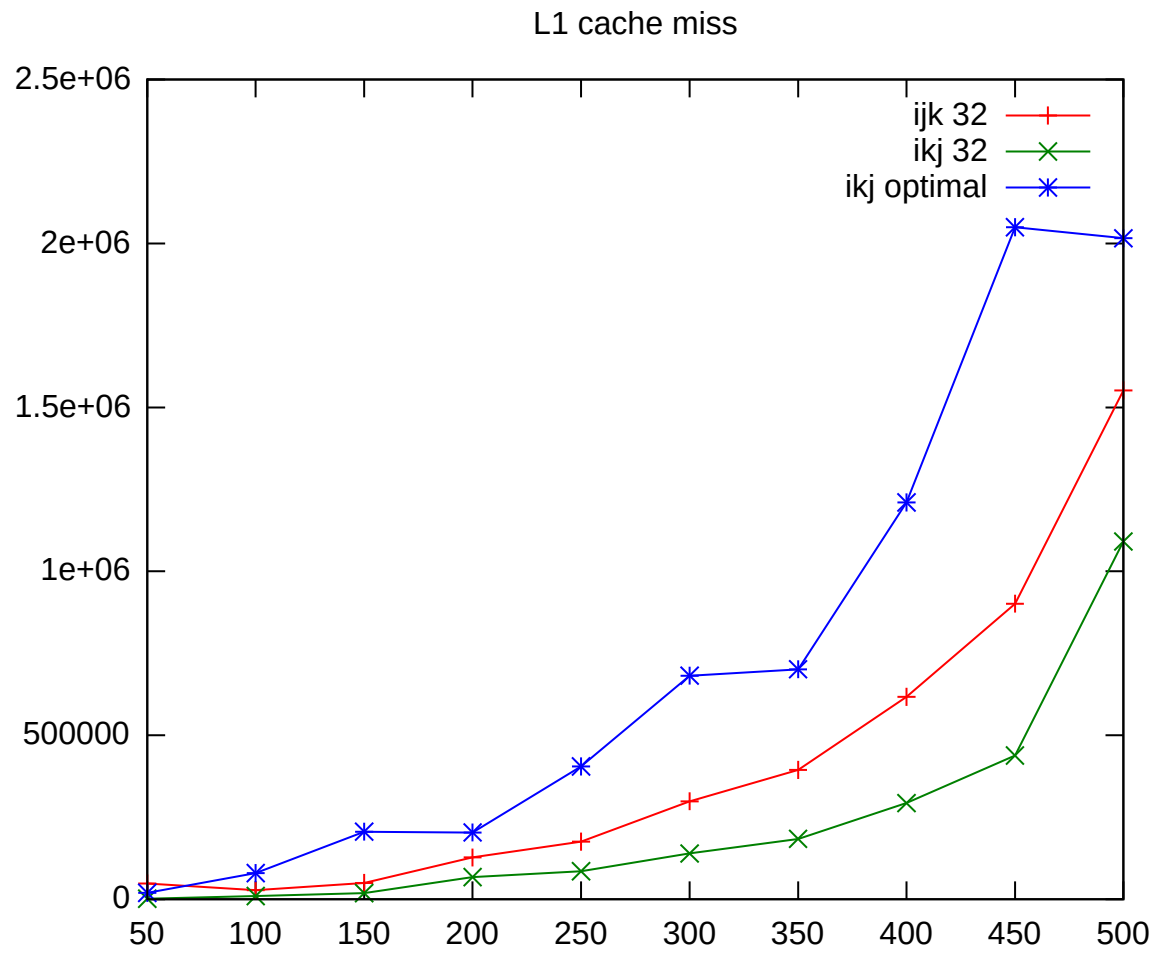
Результаты выполнения.

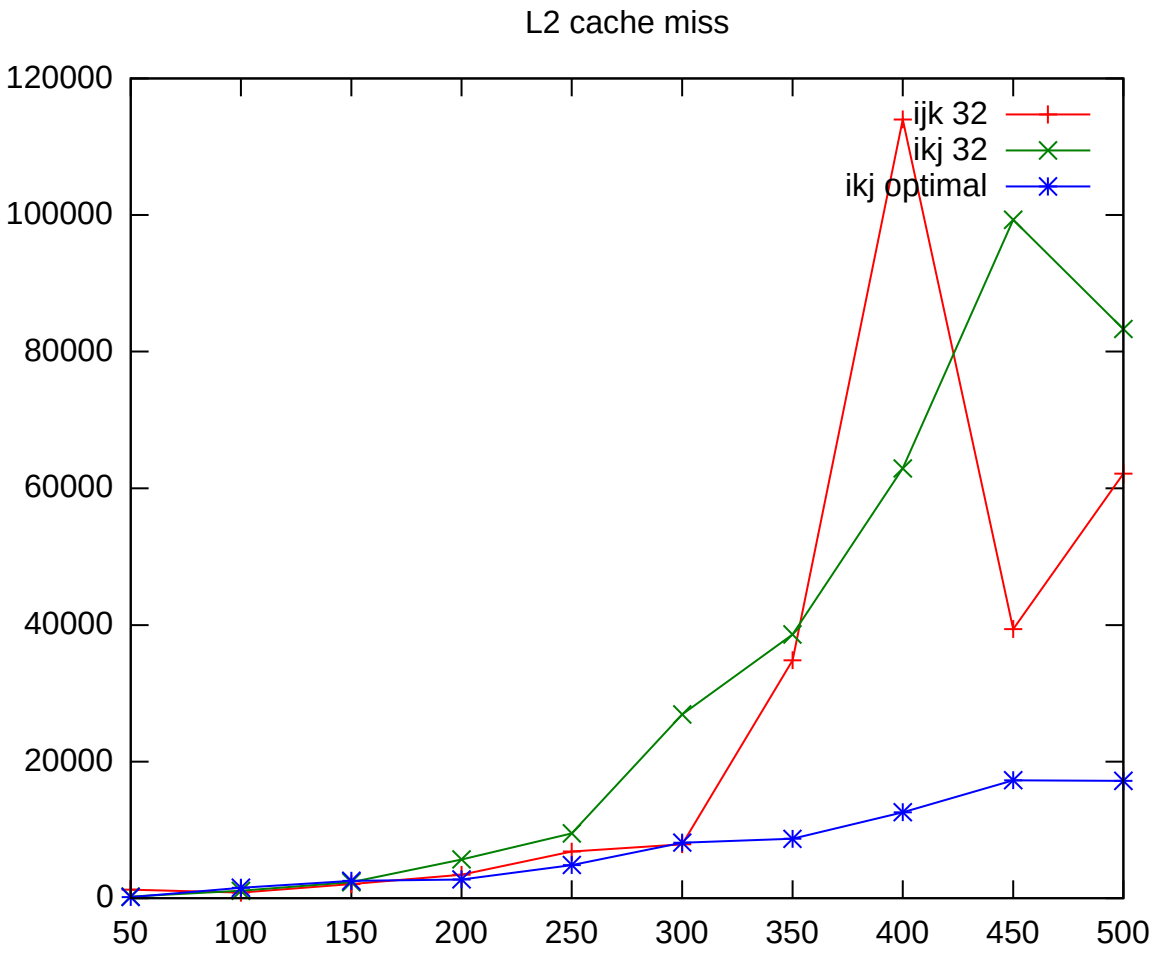
Результаты:

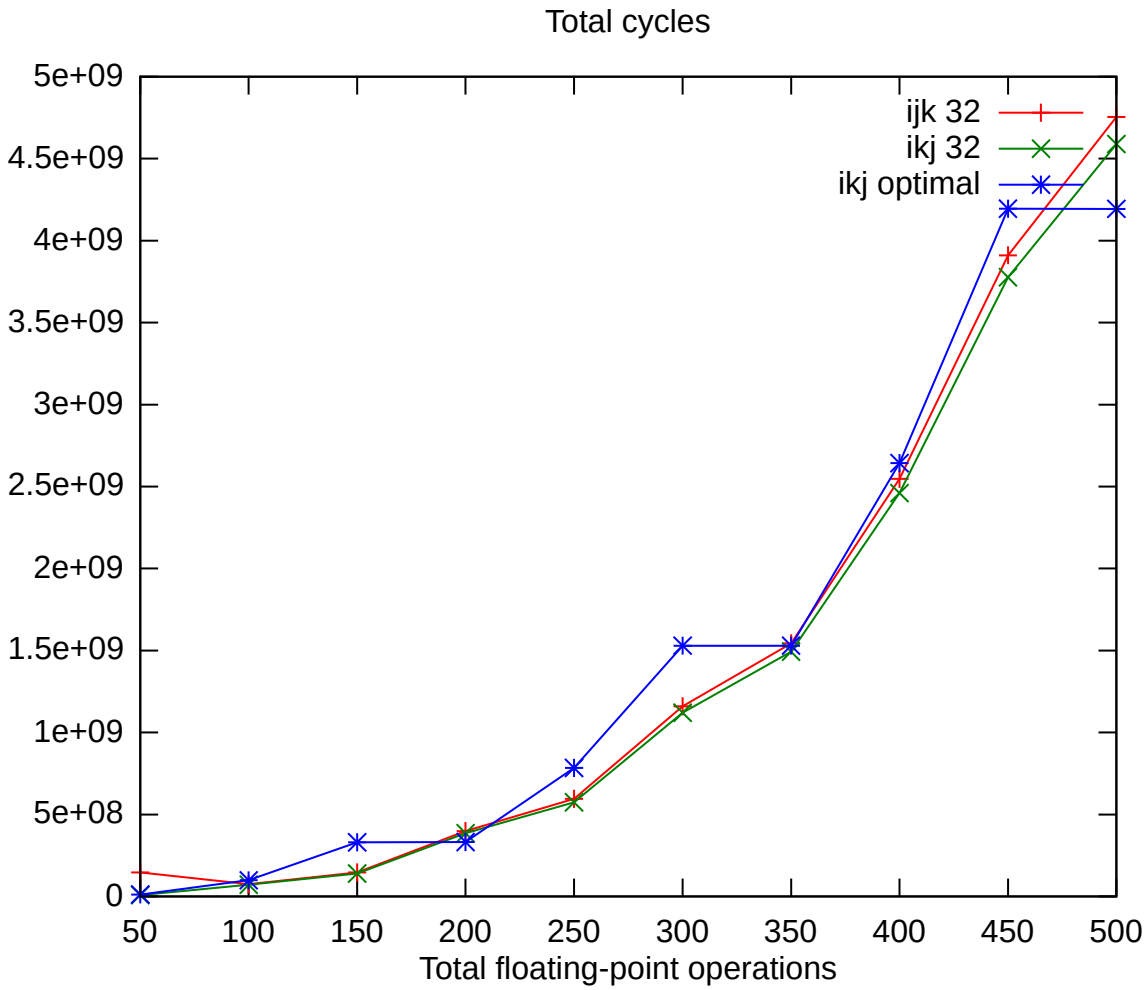
Проводилось перемножение десяти пар квадратных матриц размерами от 50×50 500×500 с шагом в 50. Зависимость данных от размеров матриц и вида умножения представлена на графиках (время — в секундах, остальное — в единицах).

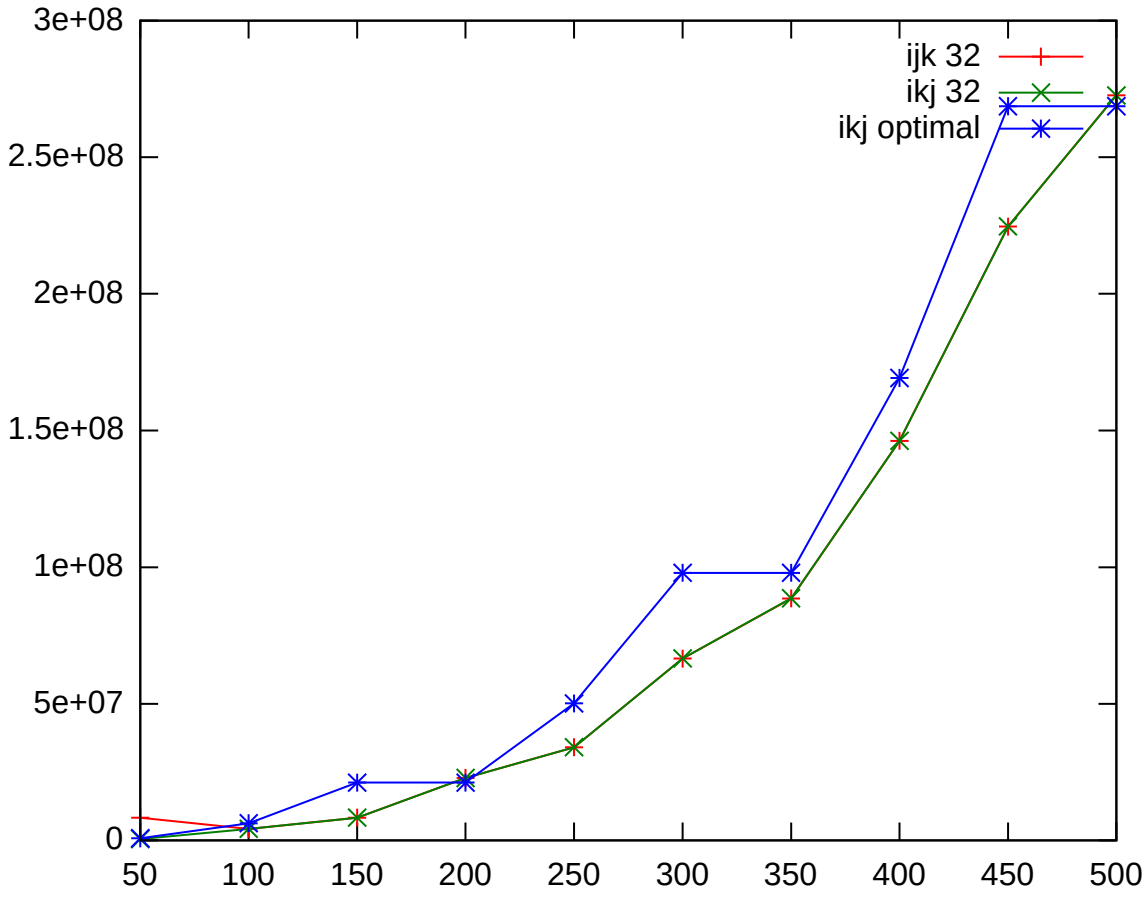
Основные выводы.




L1 cache miss
2.5e+06
2e+06
1.5e+06
1e+06
500000
0
50
100
150
200
250
300
350
400
450
500
ijk 32
ikj 32
ikj optimal


L2 cache miss
120000
100000
80000
60000
40000
20000
0
50
100
150
200
250
300
350
400
450
500
ijk 32
ikj 32
ikj optimal


Total cycles
5e+09
4.5e+09
4e+09
3.5e+09
3e+09
2.5e+09
2e+09
1.5e+09
1e+09
5e+08
0
50
100
150
200
250
300
350
400
450
500
ijk 32
ikj 32
ikj optimal


Total floating-point operations
3e+08
2.5e+08
2e+08
1.5e+08
1e+08
5e+07
0
50
100
150
200
250
300
350
400
450
500
ijk 32
ikj 32
ikj optimal

Исследования показывают, что изменение порядка индексов несколько уменьшило число промахов кэша L1, зависимость же числа промахов кэша L2 для стандартных размеров блоков слишком неясна, чтобы говорить об улучшении или ухудшении. Для оптимального размера блока, как ни странно, число промахов кэша L1 только увеличилось, тогда как число промахов кэша L2 значительно упало. Это свидетельствует о том, что алгоритмы записи данных в кэш и их удаления из него сложнее, чем «хранение трех блоков целиком в кэше» - процессор может преждевременно выбросить элементы блоков из кэша. Более того, при «оптимальном» размере блоков вероятность удаления хотя бы одного элемента при записи всех элементов всех трех блоков почти равна 1, в то время как при намного меньшем стандартном размере блока эта вероятность намного ниже. Уменьшение числа промахов кэша L2 же ожидаемо — он имеет намного больший размер, а значит, что при заданных размерах блоков они будут находиться в L2 целиком почти все время выполнения программы, а значит, увеличение размеров блоков уменьшит обращение к общей памяти за новыми данными. На остальные параметры порядок индексов и увеличение размеров блоков повлияло незначительно, а небольшое увеличение этих параметров для оптимального варианта связано с увеличением числа дополнительных нулевых элементов на краях матриц, которые мы добавили для кратности размеров матриц размерам блоков.