ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В.ЛОМОНОСОВА»

ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ
КАФЕДРА СУПЕРКОМПЬЮТЕРОВ И КВАНТОВОЙ ИНФОРМАТИКИ

ЗАДАНИЕ 2

«АНАЛИЗ БЛОЧНОГО АЛГОРИТМА МАТРИЧНОГО УМНОЖЕНИЯ С ПОМОЩЬЮ СИСТЕМЫ PAPI»

Выполнил студент
группы м118:
Пухов Д. Н.

Москва
2017

Формулировка задачи

Реализовать последовательный алгоритм блочного матричного умножения и оценить влияние кэша на время выполнения программы. Дополнить отчёт результатами сбора информации с аппаратных счётчиков, используя систему PAPI.

Описание алгоритма

Уравнение $C = A \cdot B$, где $A = n \times m$, $B = m \times h$, $C = n \times h$, можно переписать следующим образом:

$$c_{ij} = \sum_{k=0}^{k=m-1} a_{ik} b_{kj}, \quad i = \overline{0, n-1}, \quad j = \overline{0, h-1},$$

что соответствует следующему коду на языке C:

```
for(int i=0; i<n; ++i)
        for(int j=0; j<h; ++j)
                for(int k=0; k<m; ++k)
                        C[i][j] += A[i][k] * B[k][j];
```

Это прямой алгоритм умножения матриц. Идея блочного алгоритма заключается в повышении эффективности работы кэша, а именно в улучшении пространственной локальности данных: в кэш загружаются небольшие блоки умножаемых матриц, с ними проводятся вычисления, следующие блоки загружаюся в кэш и т.д. В данной работе оптимизируется только кэш первого уровня - L1.

Программа производит расчёты на числах одинарной точности для блока размером 32×32 в режимах ijk, ikj и для блока оптимального размера в режиме ijk.

Определение оптимального размера блока

Ядро процессора Intel Core i5 3230M имеет L1d кэш размером 32 KiB. Размер линии кэша равен 64 B. Размер числа одинарной точности равен 4 B. В кэше нужно хранить 3 блока оптимального размера b (измеряем в количестве чисел, а не в байтах).

Таким образом, $3b^2 <= \frac{32*1024}{4}$, откуда $b = 52$.

Используемый код

```
void square_dgemm_ijk_with_call_ijk(float* A, float* B, float* C,
                                    int n, int bsz)
{
    for(int i=0; i<n; i+=bsz)
    {
        const int i_offset = i*n;
        const int M = min(bsz, n-i);
        for(int j=0; j<n; j+=bsz)
        {
            const int N = min(bsz, n-j);
            for(int k=0; k<n; k+=bsz)
            {
                const int K = min(bsz, n-k);
                calculate_block_ijk(A+i_offset+k, B+k*n+j,
                                    C+i_offset+j, M, N, K, n);
            }
        }
    }
}
```

```
static void calculate_block_ijk(float* A, float* B, float* C,
                                int M, int N, int K, int n)
{
    for(int i=0; i<M; ++i)
    {
        const int i_offset = i*n;
        for(int j=0; j<N; ++j)
        {
            float cij = 0.0;
            for(int k=0; k<K; ++k)
                cij += A[i_offset+k] * B[k*n+j];
            C[i_offset+j] += cij;
        }
    }
}
```

Описание программы

Запуск

Программа использует Makefile и не предусматривает различных сценариев запуска. Для обработки данных требуется запустить скрипт report.py.

Формат записи в файл

Запись осуществляется в два файла из соображений удобства записи (а не чтения) в формате csv. Однако файлы имеют расширение txt ("../res/results1.txt"и "../res/results2.txt"). При обработка они сливаются другой сторонней программой в единый csv файл.

Формат записи: indices,n,bsz,events,time,

где indices = ijk или ikj — порядок индексов внутри блока,

n — размер матрицы,

bsz — размер блока,

events — одно или несколько значений счётчиков (зависит от аргумента функции тестирования), time — время в секундах.

Пример первых двух строк:

```
indices,size,bsz,PAPI_L1_DCM,PAPI_L2_DCM,PAPI_TOT_CYC,time
ijk,1000,32,10101668,6251000,9925367478,3.113803
```

Здесь events содержит три счётчика.

Все расчёты проводились без усреднения.

Результаты

Ожидаемое число промахов должно быть одинаковым для ijk, ikj 32×32, поскольку блок матрицы так или иначе полностью загружается в кэш. Число промахов для оптимального размера блока (52 для данного процессора) должно быть больше, чем для блока 32×32!

Я рассуждаю по следующей модели. Поскольку данные загружаются в L1 линиями длиной 64 байта, или 16 чисел одинарной точности, то блок 32×32 требует 32×2 обращений в память, причём линии кэша используются максимально эффективно. Когда блок имеет размер 52×52, требуется 52×4 обращений в память, причём каждое третье обращение загружает линию кэша лишь частично (52-16-16-16=4 числа должны быть загружены, вместе с ними загружается "мусор"длиной 16-4=12 чисел).

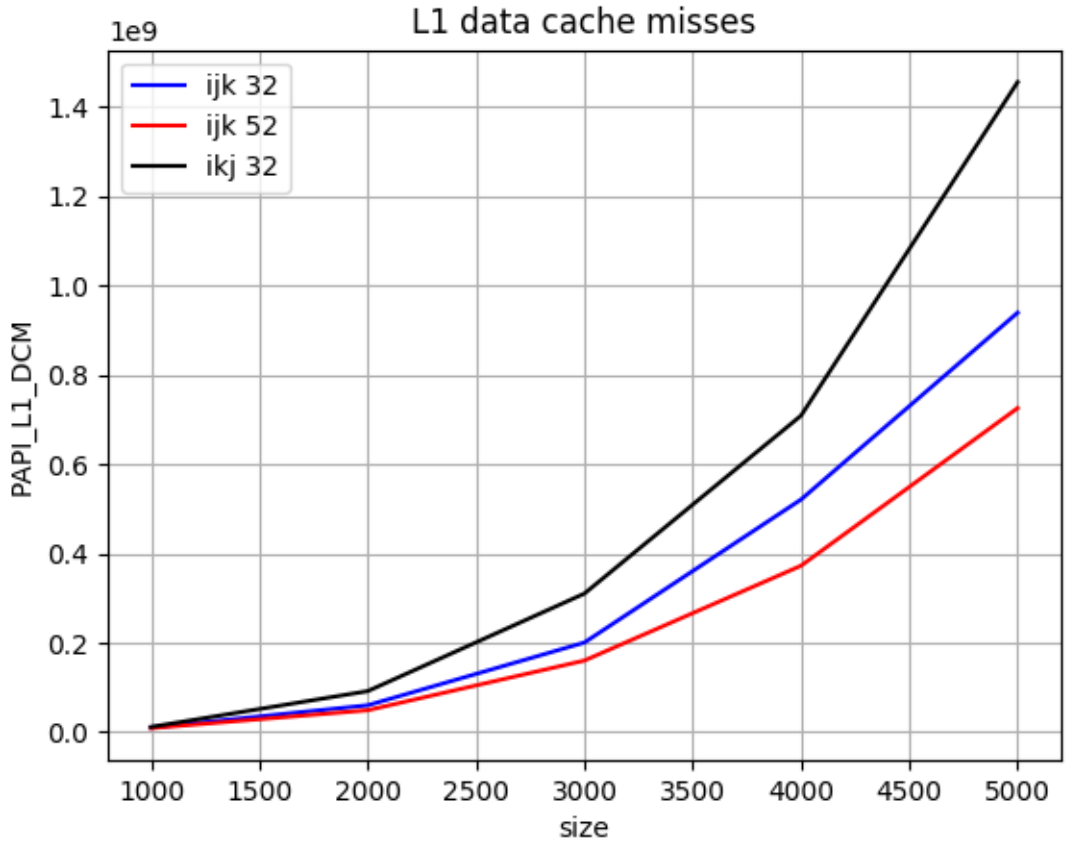
Итоговая формула числа промахов кэша определяется числом матриц, количеством блоков внутри одной матрицы и количеством промахов кэша при загрузке одного блока:

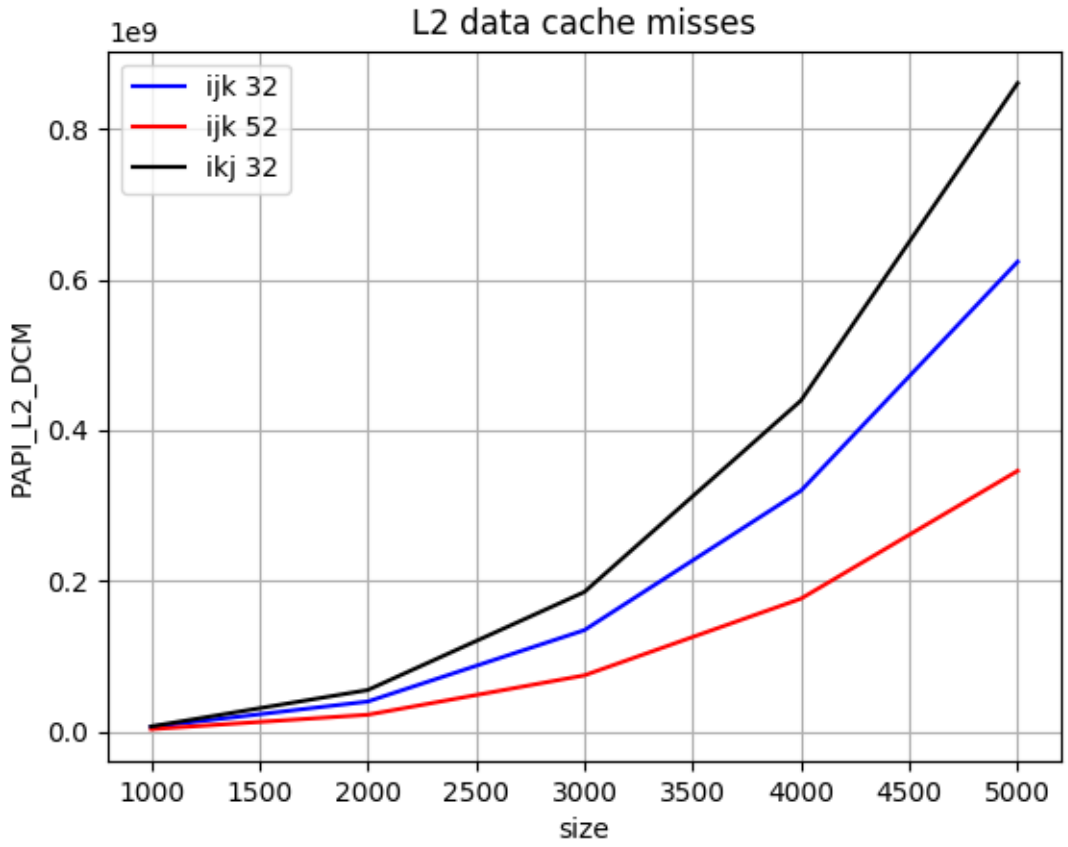
$$L1\_DCM \leq 3 \times \left(\lceil \frac{matrixsize}{blocksize} \rceil \right)^2 \times b \times \lceil \frac{blocksize}{cachelinesize} \rceil.$$

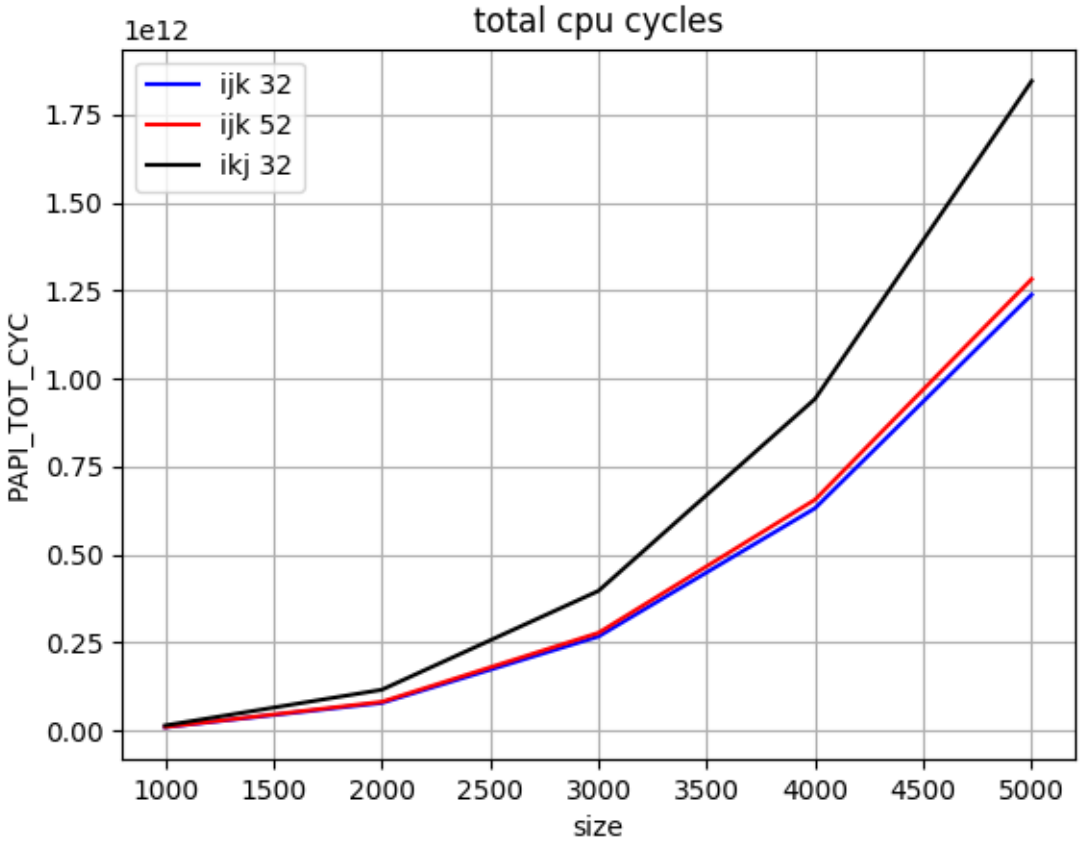
Формула учитывает только холодные промахи кэша, т.е. подразумевается, что все данные помещаются в кэш и не замещаются данными других программ (ОС, например, может что-то хранить в кэше). Первое условие должно выполняться, поскольку мы соответствующим образом вычислили размер блока. Второе условие может быть невыполненным, это нужно дополнительно изучить.

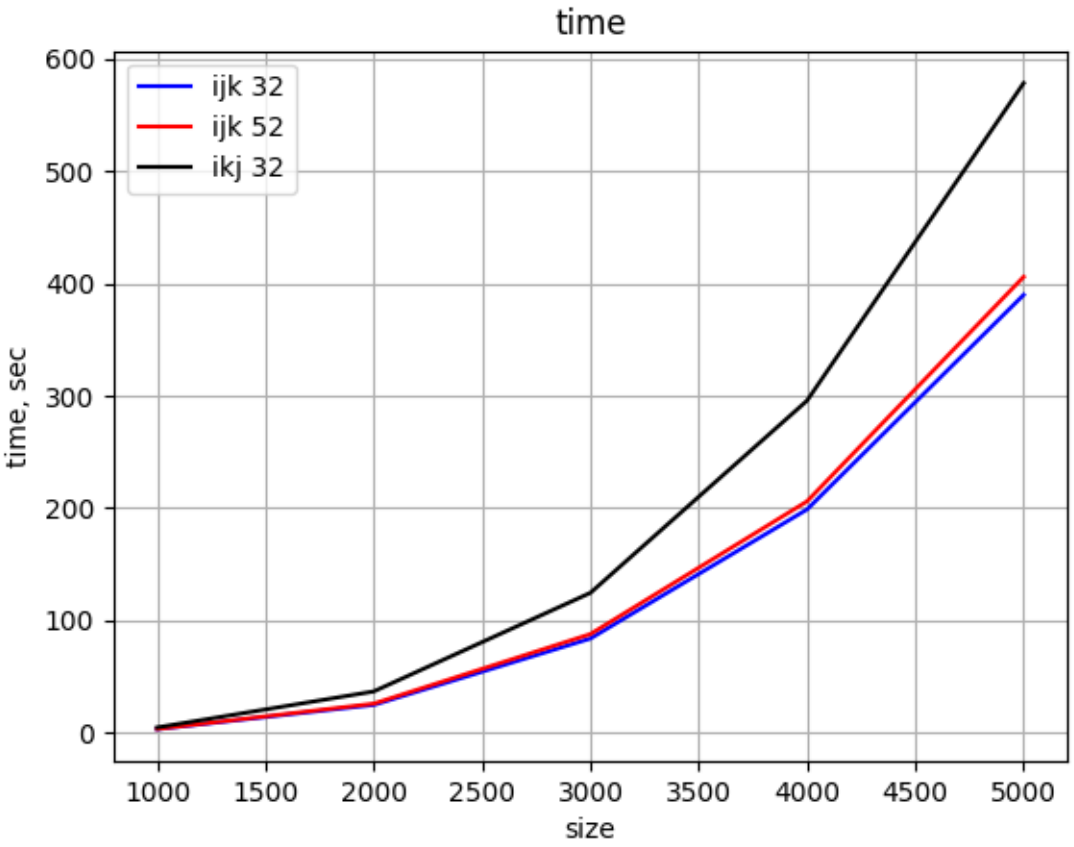
Возможные причины несоответствия графиков формуле: неверные измерения, архитектурные особенности работы кэша, особенности работы счётчиков, неверная обработка данных.

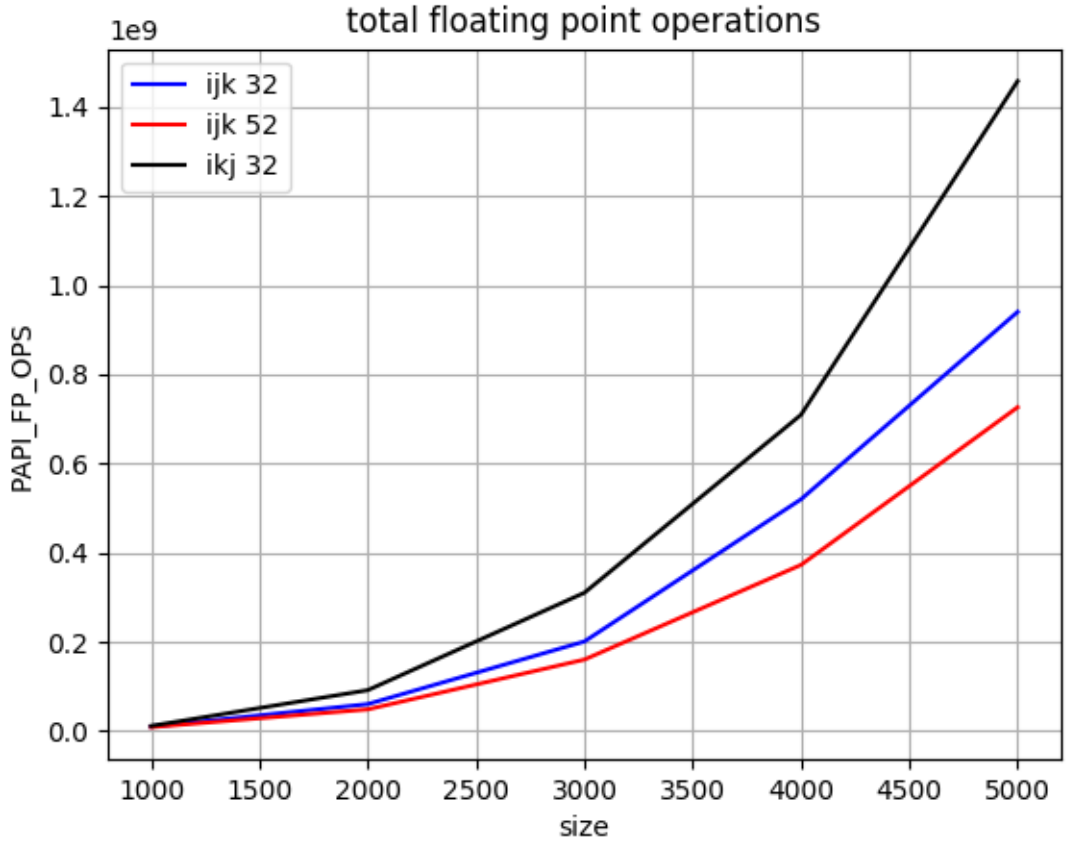
Также глубоко неясно, почему общее число операций с плавающей запятой зависит от порядка индексов и размера блока (операций всё равно n^3).

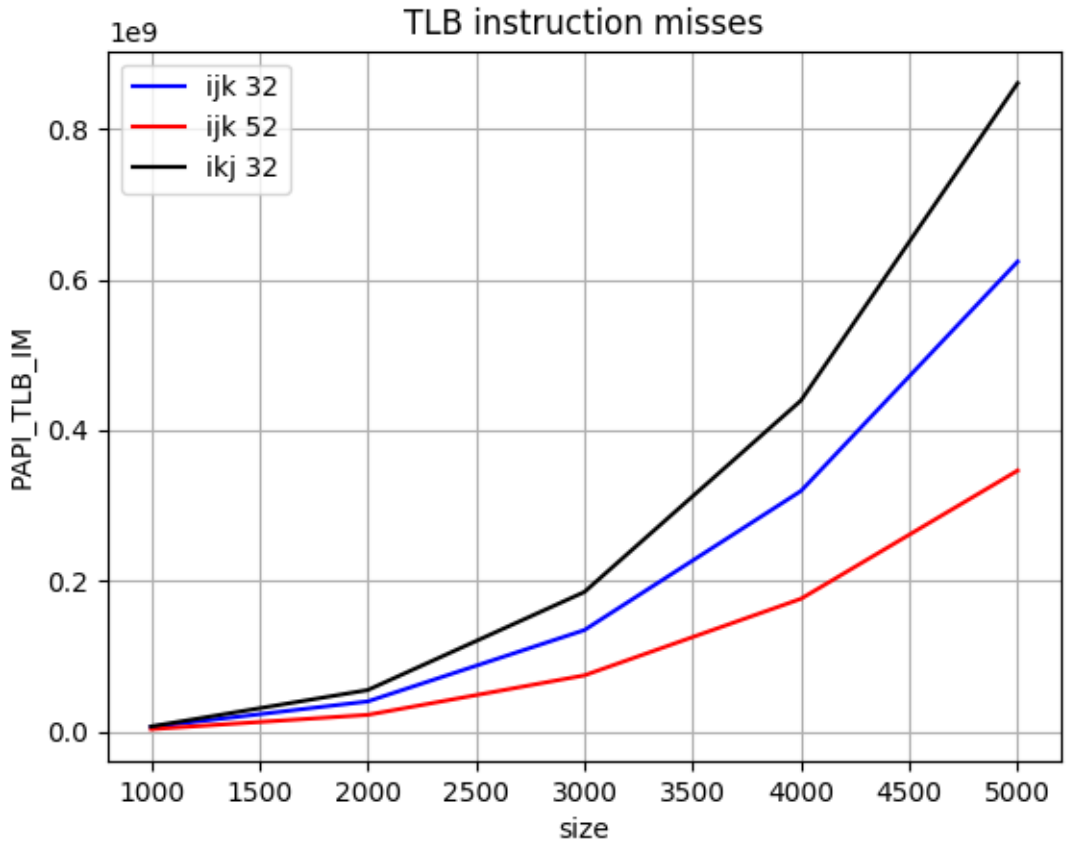


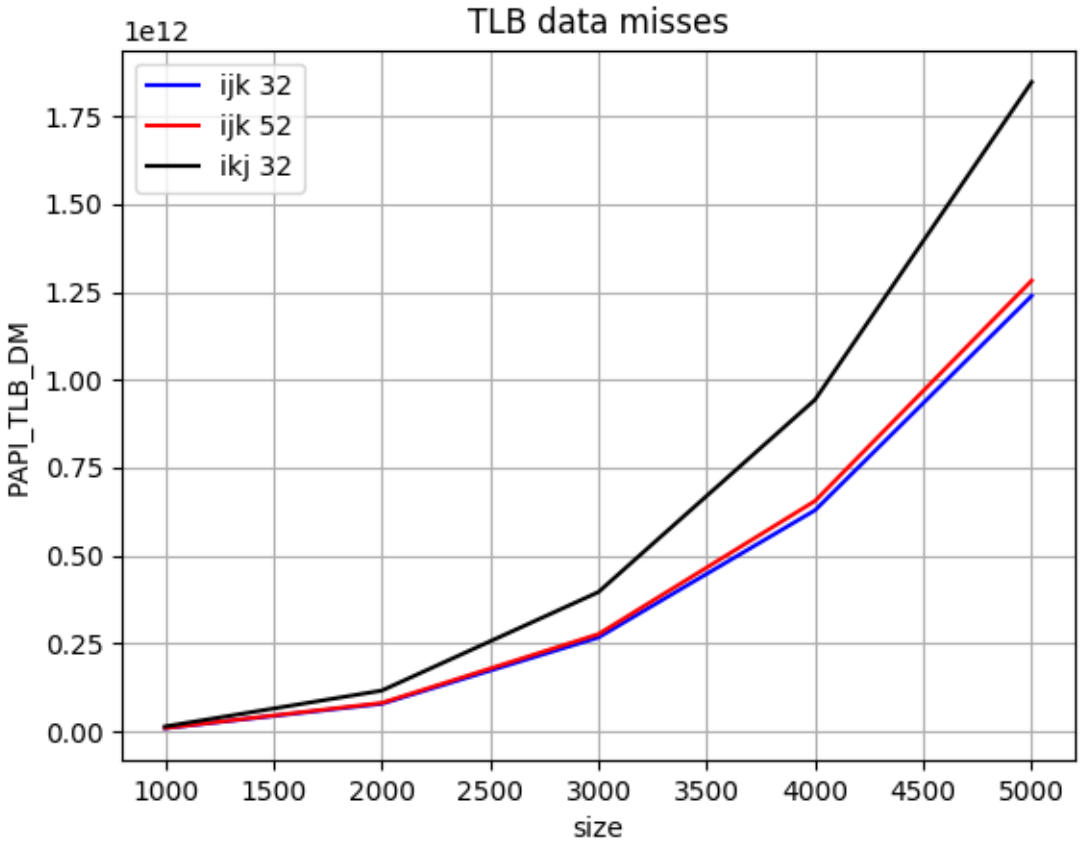

L2 data cache misses
1e9
ijk 32
ijk 52
ikj 32
PAPI_L2_DCM
0.0
0.2
0.4
0.6
0.8
1000
1500
2000
2500
3000
3500
4000
4500
5000
size


total cpu cycles
1e12
ijk 32
ijk 52
ikj 32
PAPI_TOT_CYC
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
1000
1500
2000
2500
3000
3500
4000
4500
5000
size


time
ijk 32
ijk 52
ikj 32
time, sec
size
600
500
400
300
200
100
0
1000
1500
2000
2500
3000
3500
4000
4500
5000


total floating point operations
1e9
ijk 32
ijk 52
ikj 32
PAPI_FP_OPS
size
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
1000
1500
2000
2500
3000
3500
4000
4500
5000


TLB instruction misses
1e9
ijk 32
ijk 52
ikj 32
PAPI_TLB_IM
0.0
0.2
0.4
0.6
0.8
1000
1500
2000
2500
3000
3500
4000
4500
5000
size


TLB data misses
1e12
ijk 32
ijk 52
ikj 32
PAPI_TLB_DM
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
1000
1500
2000
2500
3000
3500
4000
4500
5000
size