Средства и системы параллельного программирования

кафедра СКИ,
сентябрь – декабрь 2021 г.
Лектор доцент Попова Нина Николаевна

Лекция 2
20 сентября 2021 г.

Параллельное программирование для высокопроизводительных систем

сентябрь – декабрь 2021 г.

Лектор доцент Попова Нина Николаевна

Лекция 3

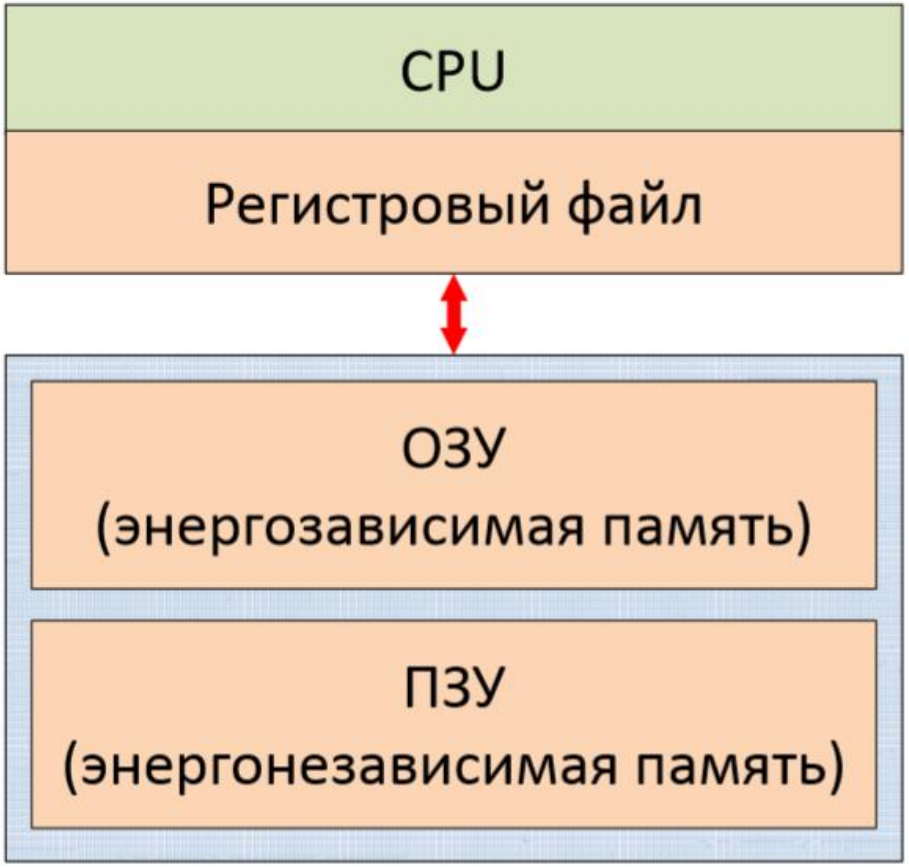
23 сентября 2021 г.

Тема

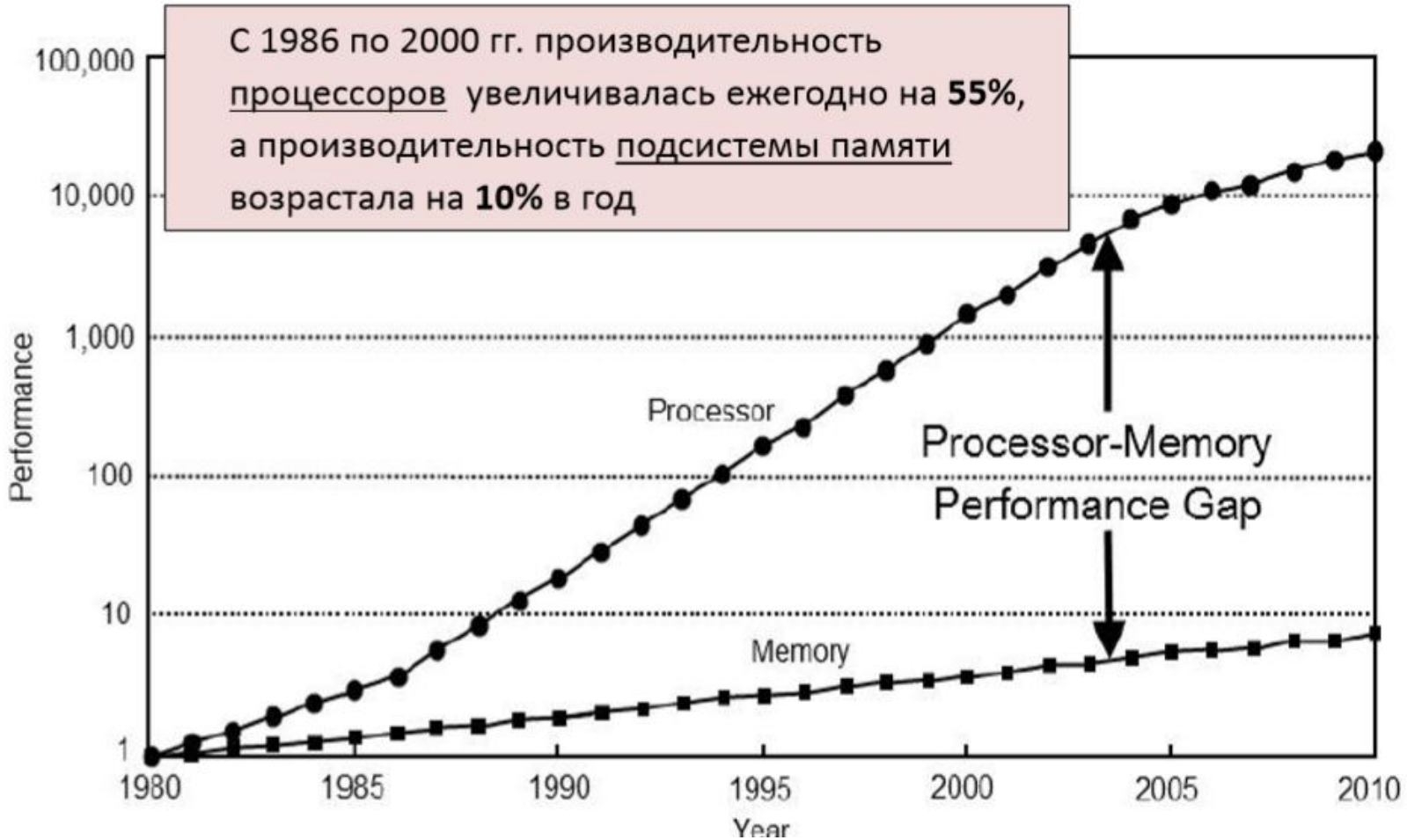
- Архитектура подсистемы памяти
- Алгоритмы матричного умножения. Последовательная реализация.

Архитектура подсистемы памяти

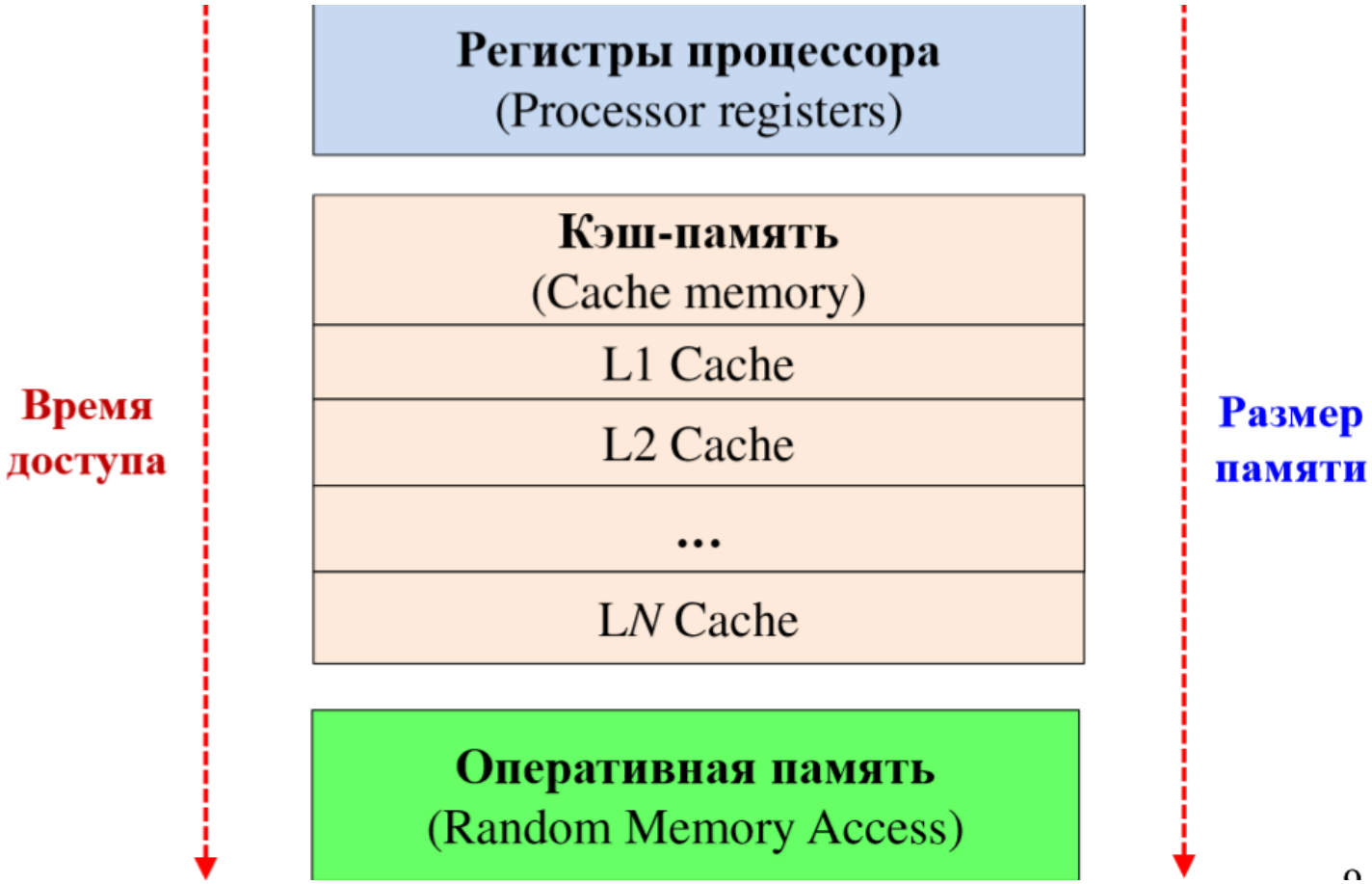
Архитектура подсистемы памяти



Стена памяти (Memory wall)

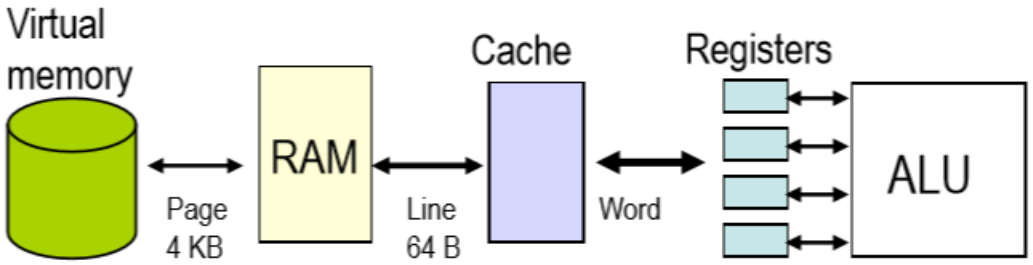


Иерархическая организация памяти



Иерархия памяти

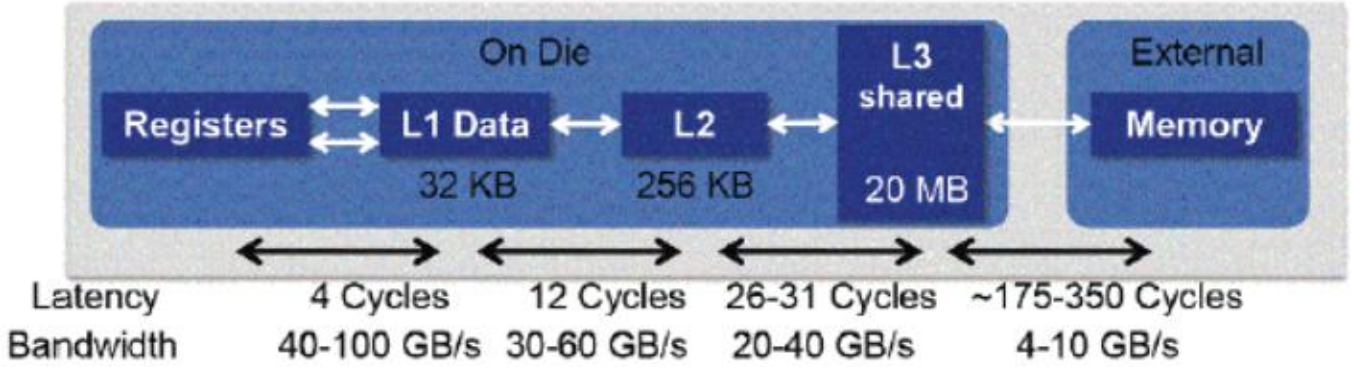
Иерархия памяти:
1. Регистры
2. Кэш
3. ОЗУ
4. Виртуальная память



Типичное времена Latency – Bandwith)
- регистры (0 тактов)
- L1 cache 5 тактов; 2 слова/такт
- L2 cache 15 тактов; 1 слово/такт
- ОЗУ 300 тактов; 0.25 слов/такт

Информация о процессоре

- /proc/cpuinfo summarizes the processor
  - vendor_id : GenuineIntel
  - model name : Intel®Xeon®CPU E5-2680 0 @ 2.70GHz
  - cache size : 20480 KB
  - cpu cores : 8
- processor : 0 through processor : 16
- Detailed information at /sys/devices/system/cpu/cpu*/cache/index*/*



GNU/Linux CPU Cache Information (/proc)

```
$ cat /proc/cpuinfo
processor       : 0
...
model name      : Intel(R) Core(TM) i5-2520M CPU @ 2.50GHz
stepping        : 7
microcode       : 0x29
cpu MHz         : 2975.000
cache size      : 3072 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 2
apicid          : 0
initial apicid  : 0
...
bogomips        : 4983.45
clflush size    : 64
cache_alignment         : 64
address sizes   : 36 bits physical, 48 bits virtual
```

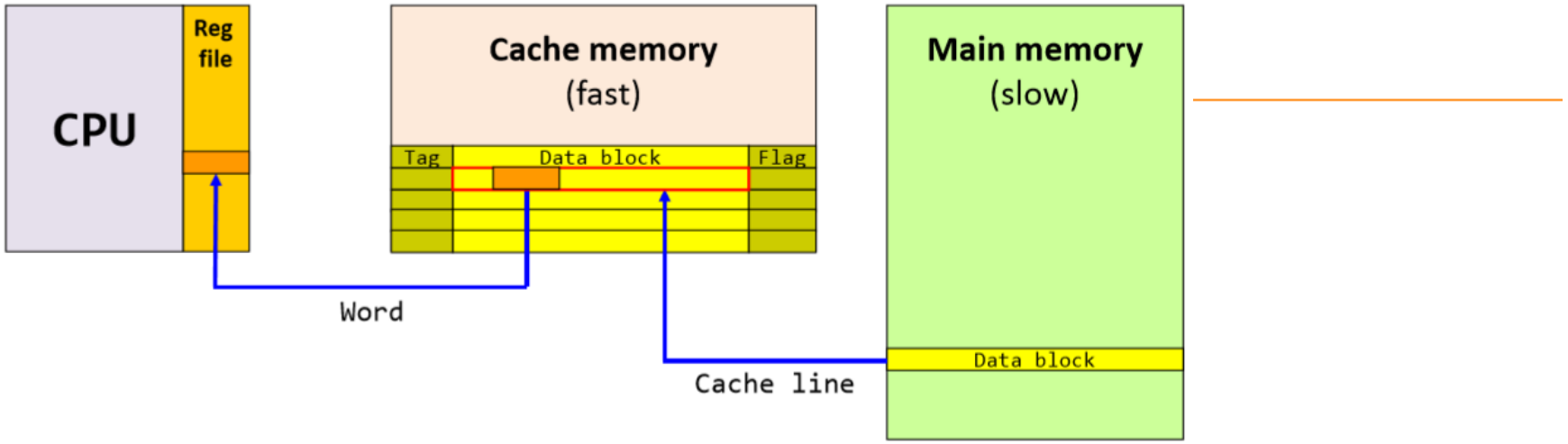
GNU/Linux CPU Cache Information (/sys)

```
/sys/devices/system/cpu/cpu0/cache
    index0/
        coherency_line_size
        number_of_sets
        shared_cpu_list  size
        ways_of_associativity
        level
        physical_line_partition
        shared_cpu_map
        type
    index1/
    index2/
    ...
```

Виды кеш-памяти

- Кеш инструкций (Instruction cache, I-cache)
- Кеш данных(Data cache, D-cache)
- TLB-кеш (TLB –Translation LookasideBuffer) – кеш преобразования виртуальных адресов в физические

Структурная организация кеш-памяти



•Кеш-память (Cache memory)– это массив записей (cache entries), которые содержат тег (tag), данные (data) и флаговые биты (flag)

•Данные между кеш-памятью и оперативной памятью передаются блоками фиксированного размера(cache lines)

•Обращаться к памяти рекомендуется по смежным адресам, это обеспечивает большую вероятность того, что данные будут находиться в кеш-памяти

Показатели производительности кеш-памяти

- **Cache latency** – время доступа к данным в кеш-памяти (clocks)
- **Cache bandwidth** – количество байт передаваемых за такт между кеш-памятью и вычислительным ядром процессора (byte/clock)
- **Cache hit rate** – отношения числа успешных чтений данных из кеш-памяти (без промахов) к общему количеству обращений к кеш-памяти

$$HitRate = \frac{N_{CacheHit}}{N_{Access}}$$

$$\mathbf{0} \leq HitRate \leq \mathbf{1}$$

Закон Литтла (Little’s law)

•Чтения из памяти могут быть зависиммыми:
операнд инструкции может быть результатом предыдущей инструкции

• Два таких обращения к памяти должны быть разделены по крайней величиной latency

•Чтобы канал доступа к памяти был постоянно загружен (использовался весь bandwidth) , одновременно должны выполняться несколько обращений к памяти и они должны быть независимыми

•Little’s law: Concurrency = Bandwidth x Latency

Характеристики кеш-памяти

- Размер кеш-памяти (Cache size)
- Размер строки кэша (Cache line size)
- Метод отображения адресов (Mapping method)
- Алгоритм замены записей в кеше(Replacement policy)
- Политика записи данных в кеш-память (Write policy)
- Протокол поддержания информационно целостности кешей в многопроцессорной систем

Многоуровневая кеш-память

- **Inclusive caches**–данные присутствующие в кеш-памяти L1 обязательно должны присутствовать в кеш-памяти L2

- **Exclusive caches**–те же самые данные в один момент времени могут располагаться только в одной из кешпамяти L1 или L2 (например, AMD Athlon)

- Некоторые процессоры допускают одновременное нахождение данных и в L1 и в L2 (например, Pentium 4)

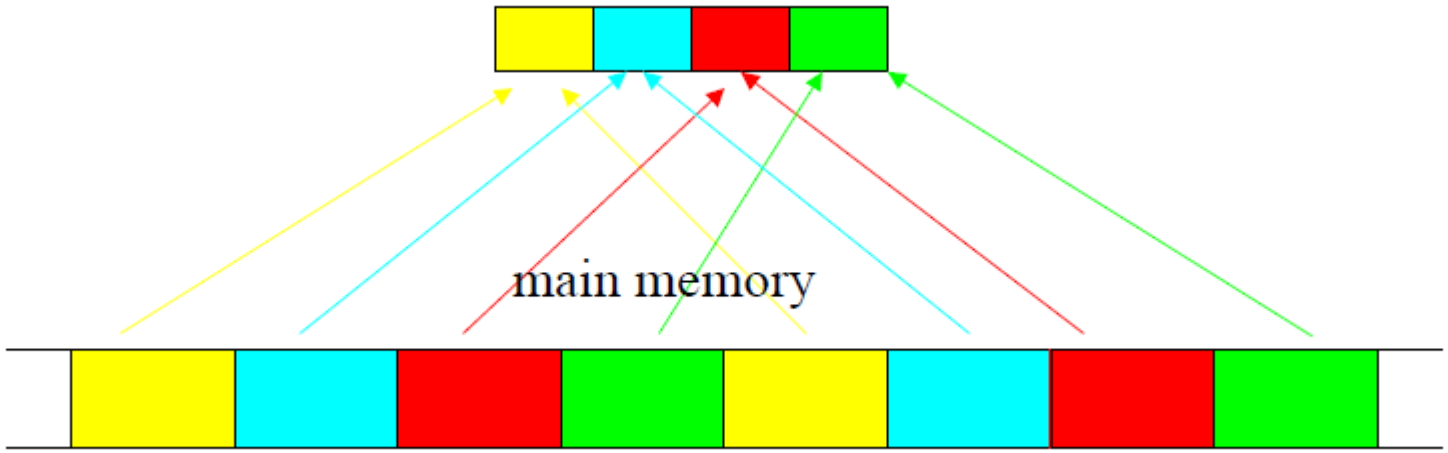
Cache Mapping

Типы:

– кэш прямого отображения (Direct)
– частично или наборно-ассоциативный кэш (Set associative)
– полностью ассоциативный кэш **(**Fully associative)

Кеш прямого отображения

Блок из оперативной памяти может быть помещен только в одно определенное место в кеше.



Кеш прямого отображения

• Если размер кеша *Nc и он делится на k строк,*
то размер строки равен *Nc/k*

• Если размер оперативной памяти равен *Nm, она разделяется на Nm/(Nc/k) блоков, каждый из которых отображается на одну из k cache lines*

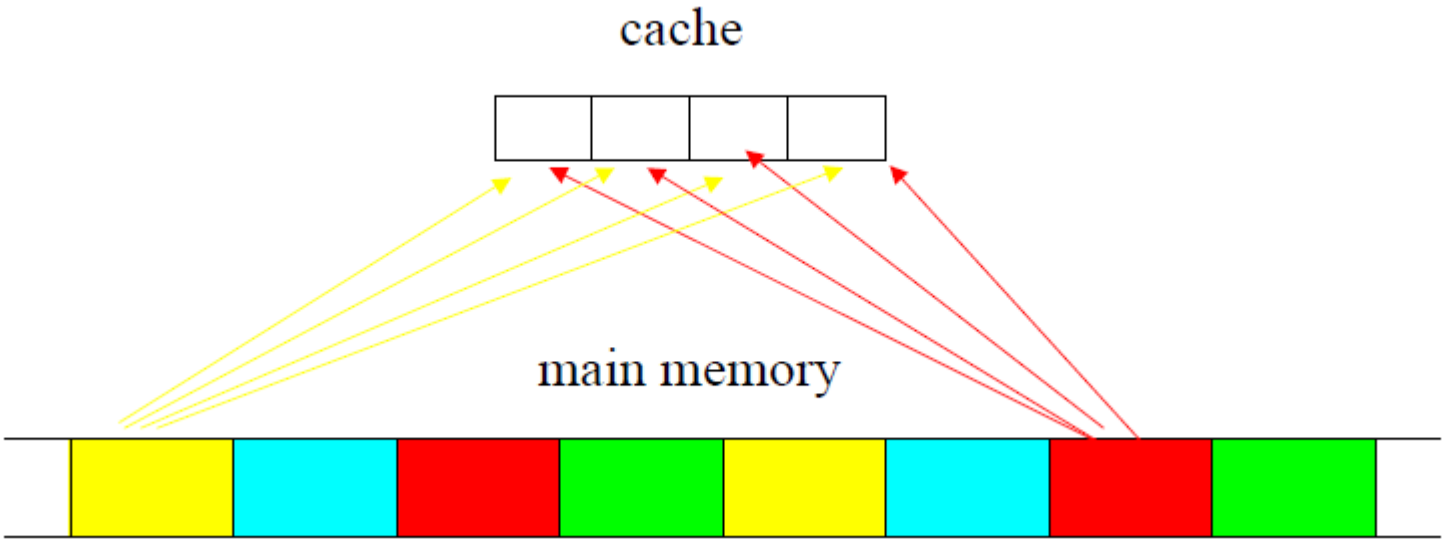
• Это означает, что каждая cache line ассоциирована с конкретной областью памяти

Проблемы с кешем прямого отображения

```
double a[8192],b[8192];
for (i=0; i<n; i++) {
  a[i] = b[i]
}
```

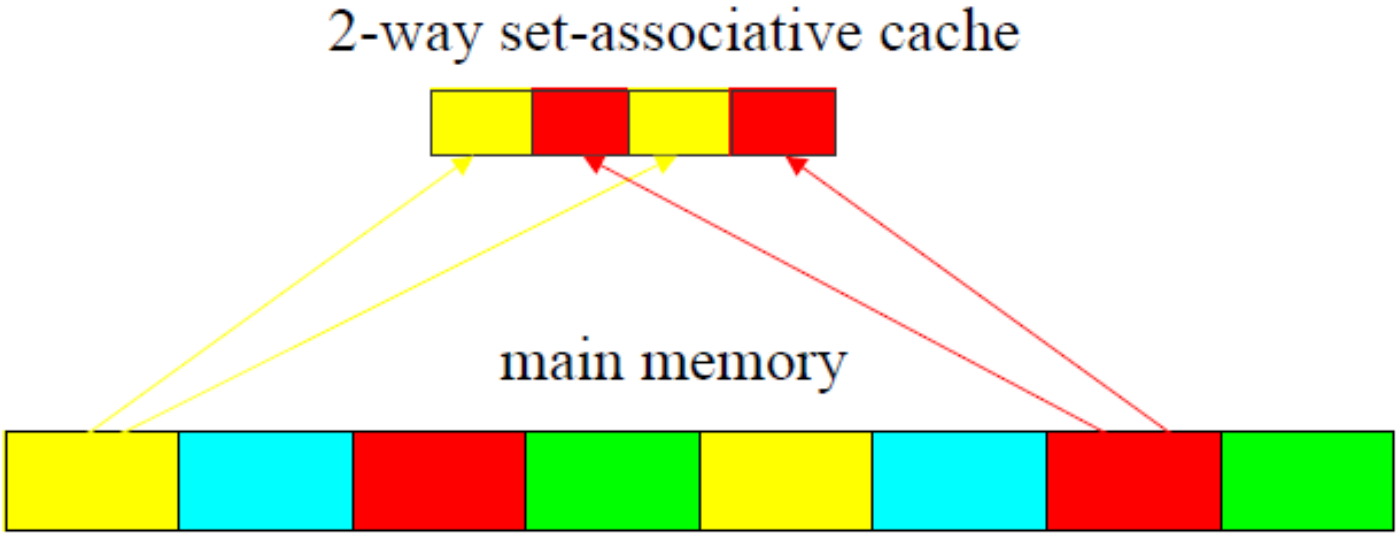
• Пример: cache size 64KByte = 8192 слов (чисел типа double)
• a[0] и b[0] отображаются в один и тот же блок кеша.
• Cache line 4 слова
• **Thrashing**:
– b[0]..b[3] загружаются в кеш, записываются в регистр
– a[0]..a[3] загружаются в кеш, получают новое значение,
Выталкивают b[0]..b[3] из кеша
– b[1] запрашивается, b[0]..b[3] опять загружаются
– a[1] запрашиваются, загружаются, снова выталкивают *b[0..3]*

Полностью ассоциативный кэш



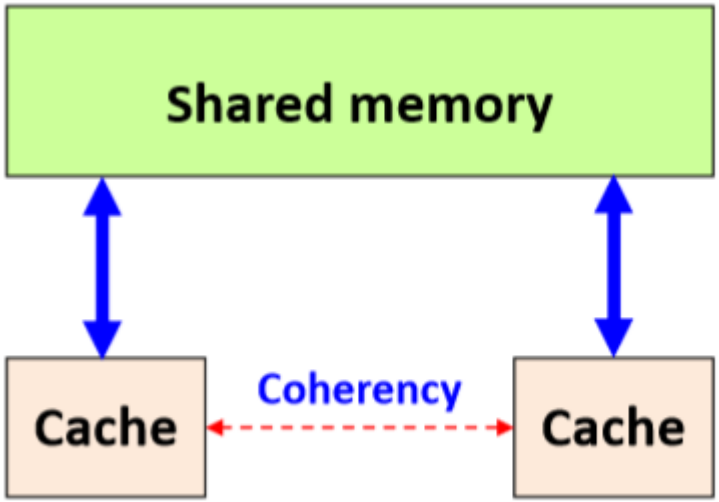
Частично-ассоциативный кэш

Промежуточный уровень между кешем прямого отображения и полностью ассоциативным кешем. В ***n-way set-associative cache*** блок оперативной памяти может размещаться в n (n по крайней мере 2) блоках кеша.



Когерентность кеш-памяти

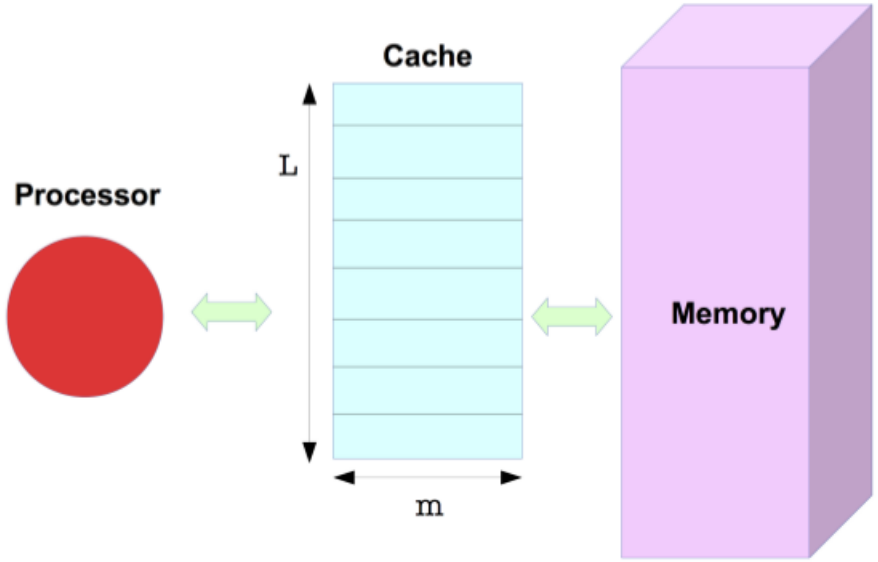
Когерентность кеш-памяти (cache coherence) –свойство кеш-памяти, означающее целостность данных, хранящихся в локальных кешах для разделяемой оперативной памяти



Предварительная загрузка данных в кеш (prefetching)

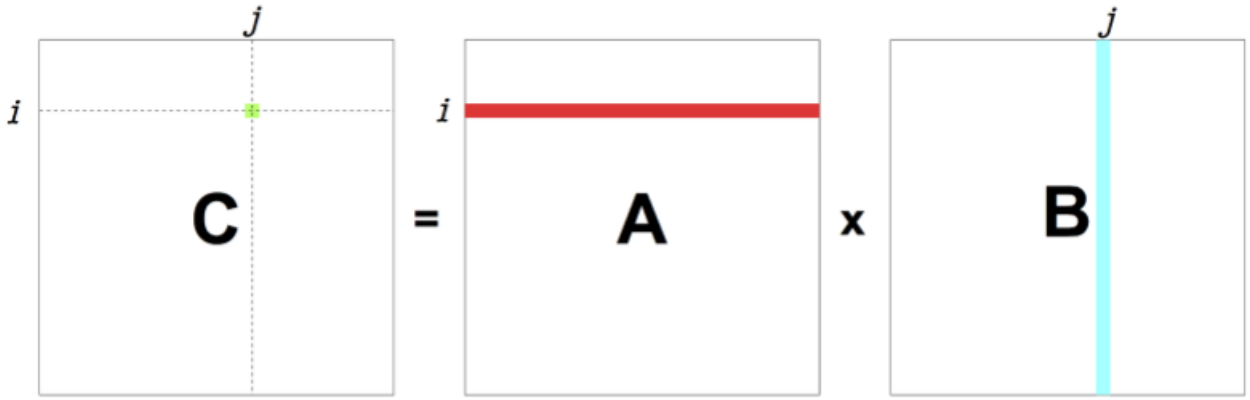
- Аппаратура пытается определить, является ли загрузка из памяти регулярной:
- “prefetch stream”
- Это может выполняться и программно

Пример: организация кэша



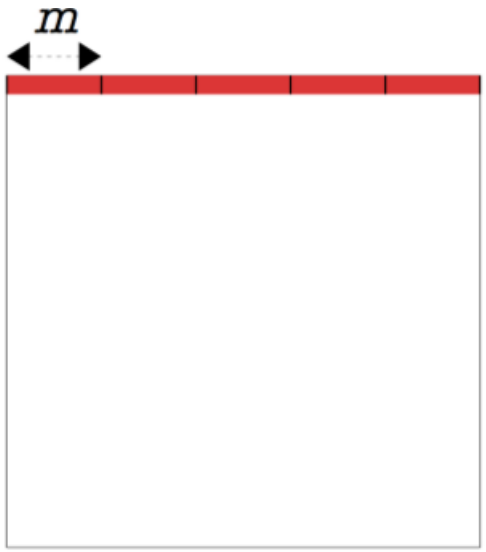
- L строк объемом m вещественных чисел с плав. точкой
- Предположим, что кэш «высокий» : L > m
- Алгоритм : Least Recently Used
- Нет аппаратной реализации предварительной загрузки (prefetching)

Матричное умножение C=AxB



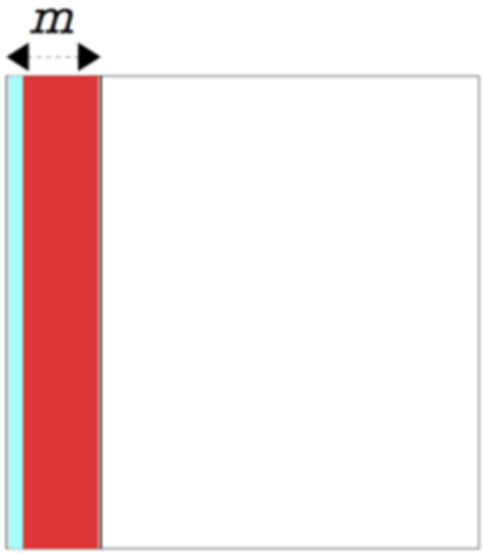
- A, B и C – квадратные матрицы размера n×n
- n достаточно большое, т.е., n > L

Доступ к элементам строки в L1



•Последовательный доступ : доступ к элементам строки требует $\frac{n}{m}$ промахов кэша

Доступ к элементам столбца в L1

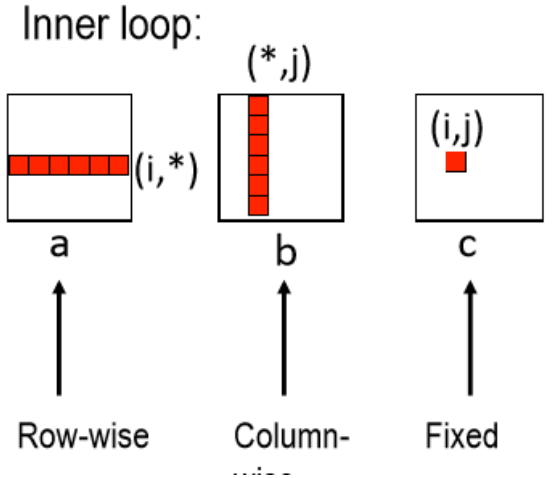


- Доступ с шагом : доступ к столбцу требует n промахов кэша

Матричное умножение

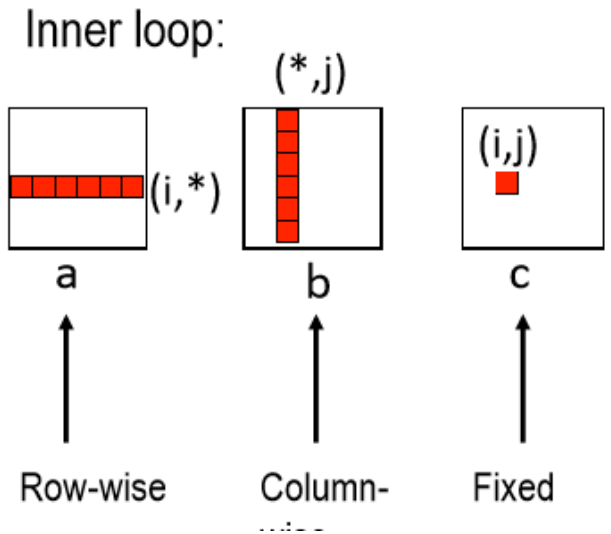
- Матричное умножение – классический пример алгоритма с плохой производительностью кеша
  - время выполнения - O(N3)
  - c[i][j] вычисляется как скалярное произведение строки i на столбец j
- предположим, что строка кеша равна 32 Bytes и размер матрицы достаточно большой (N > 1000)
  - в строке кеша могут размещаться 4 вещественных числа двойной точности

```
for (int i=0; i<N; i++)  {
   for (int j=0; j<N; j++) {
      double sum = 0.0;
      for (int k=0; k<N; k++)
           sum += a[i][k] * b[k][j];
      c[i][j] = sum;
  }
}
```



Матричное умножение

```
for (int i=0; i<N; i++)  {
   for (int j=0; j<N; j++) {
      double sum = 0.0;
      for (int k=0; k<N; k++)
           sum += a[i][k] * b[k][j];
      c[i][j] = sum;
  }
}
```



Исследуем доступ к памяти во вложенном цикле.

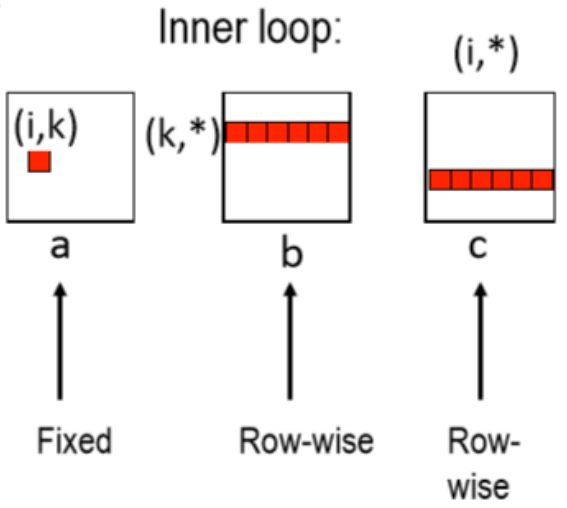
- Элементы матрицы a доступны с шагом 1
  - при чтении очередного элемента в строку кеша записываются 4 элемента., т.е. каждый 4-ый доступ к элементу будет происходить промах кеша (тип промаха – принудительный - compulsory)
- Доступ к элементам матрицы b происходит с шагом N
  - при каждом чтении элемента матрицы b происходит промах кеша
- Значение sum будет находиться на регистре

Промахи кэша

a	b	c
0,25	1,0	0,0

Альтернативная реализация: (i,k,j)

- Что меняется, если меняем порядок вложенных циклов
  - умножаем a[i][k] на все элементы строки k матрицы b и получаем частичную сумму строки i матрицы c
  - доступ к элементам матриц b и c строчный
  - переменная r будет находиться на регистре



Промахи кеша:

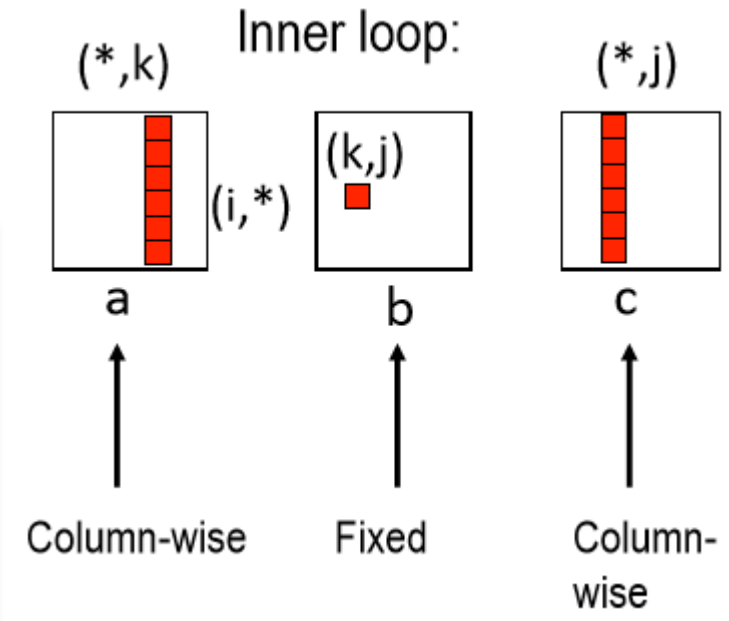
a	b	c
0,0	0,25	0,25

```
for (int i=0; i<N; i++)  {
   for (int k=0; k<N; k++) {
      double r = a[i][k];
      for (int j=0; j<N; j++)
           c[i][j] += r * b[k][j];
  }
}
```

Альтернативная реализация : (j,k,i)

```
for (int j=0; j<N; j++)  {
   for (int k=0; k<N; k++) {
      double r = b[k][j];
      for (int i=0; i<N; i++)
           c[i][j] += a[i][k] * r;
  }
}
```

a	b	c
1,0	0,0	1,0

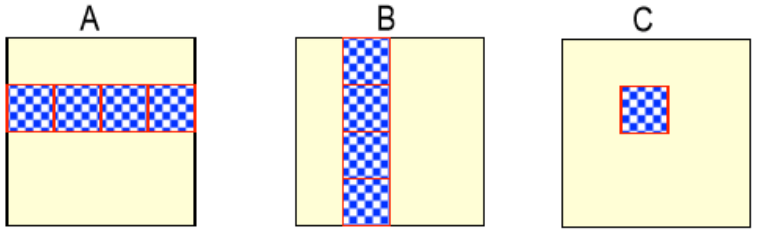


Выводы

- Можно определить 6 вариантов перестановок 3-ех вложенных циклов перестановок индексов трех вложенных циклов при реализации алгоритма матричного умножения
  - варианты, при которых меняется порядок двух внешних циклов, имеют идентичное поведение промахов кэша
- ijk (и jik)
  - 2 loads, 0 stores
  - 1,25 cache misses/iteration
- ikj (и kij)
  - 2 loads, 1 store
  - 0,5 cache misses/iteration
- jki (и kji)
  - 2 loads, 1 store
  - 2,0 cache misses/iteration

Блочное матричное умножение

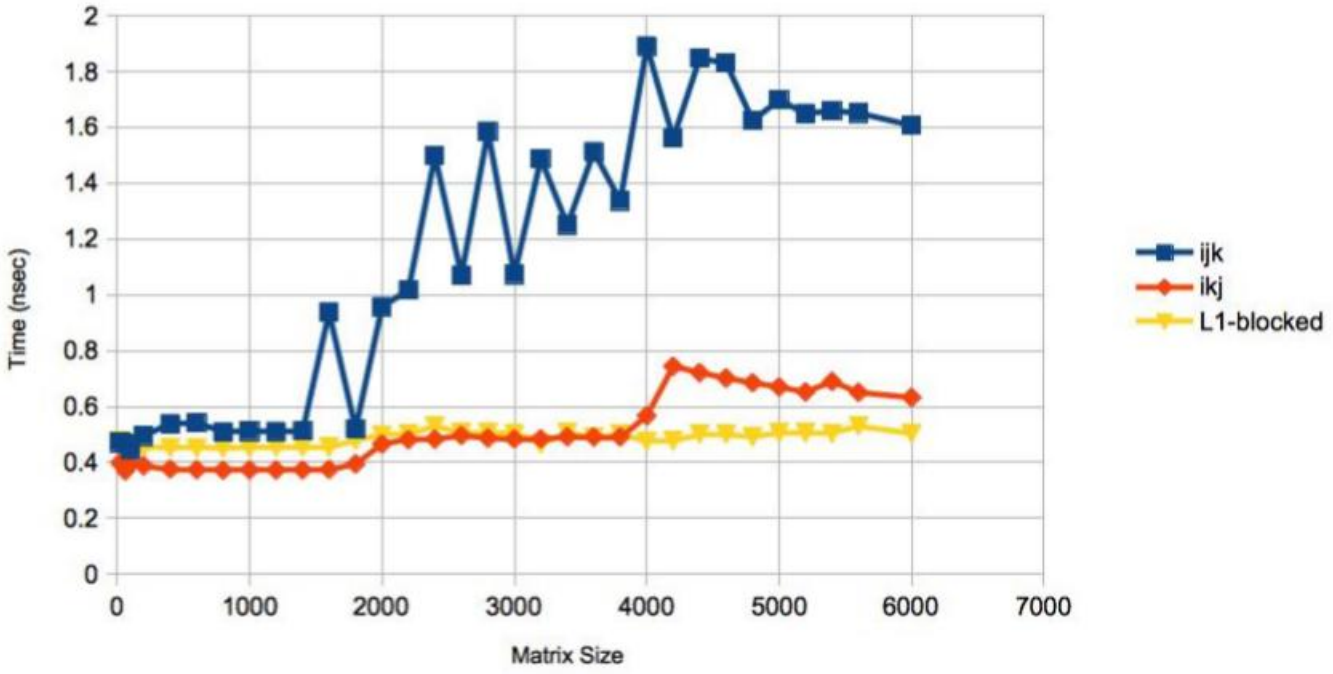
- Оптимизация для данных, которые не укладываются в кеше
- Разделение данных на меньшие блоки размера b*b, которые размещаются в кеше
  - выполнение вычислений над блоками, находящимися в кеше
- Пример: матричное умножение C = A*B
  - матрицы представляются в виде NxN матриц
  - вычисления над блоком выполняются без дополнительного обращения к памяти
- Размер блока выбирается таким образом, чтобы все данные, необходимые для вычисления одного блока, располагались в кэше
  - доступ к элеметам B по-прежнему остается по столбцам, но блок при этом полностью располагается в кеше.



Блочное матричное умножение

```
for (int i = 0; i < N; i+=b)  // b is the blocksize
    for (int j = 0; j < N; j+=b)
        for (int k = 0; k < N; k+=b)
            /* B x B mini matrix multiplications */
            for (i1 = i; i1 < i+b; i++)
                for (j1 = j; j1 < j+b; j++)
                    for (k1 = k; k1 < k+b; k++)
                        c[i1][j1] += a[i1][k1]*b[k1][j1];
```

Сравнение вариантов ijk, ikj и L1-блочный



Задание

- Реализовать последовательный алгоритм матричного умножения (не блочного) и оценить влияние кэша на время выполнения программы.

- Формат командной строки, исследуемые параметры, форма отчета задаются