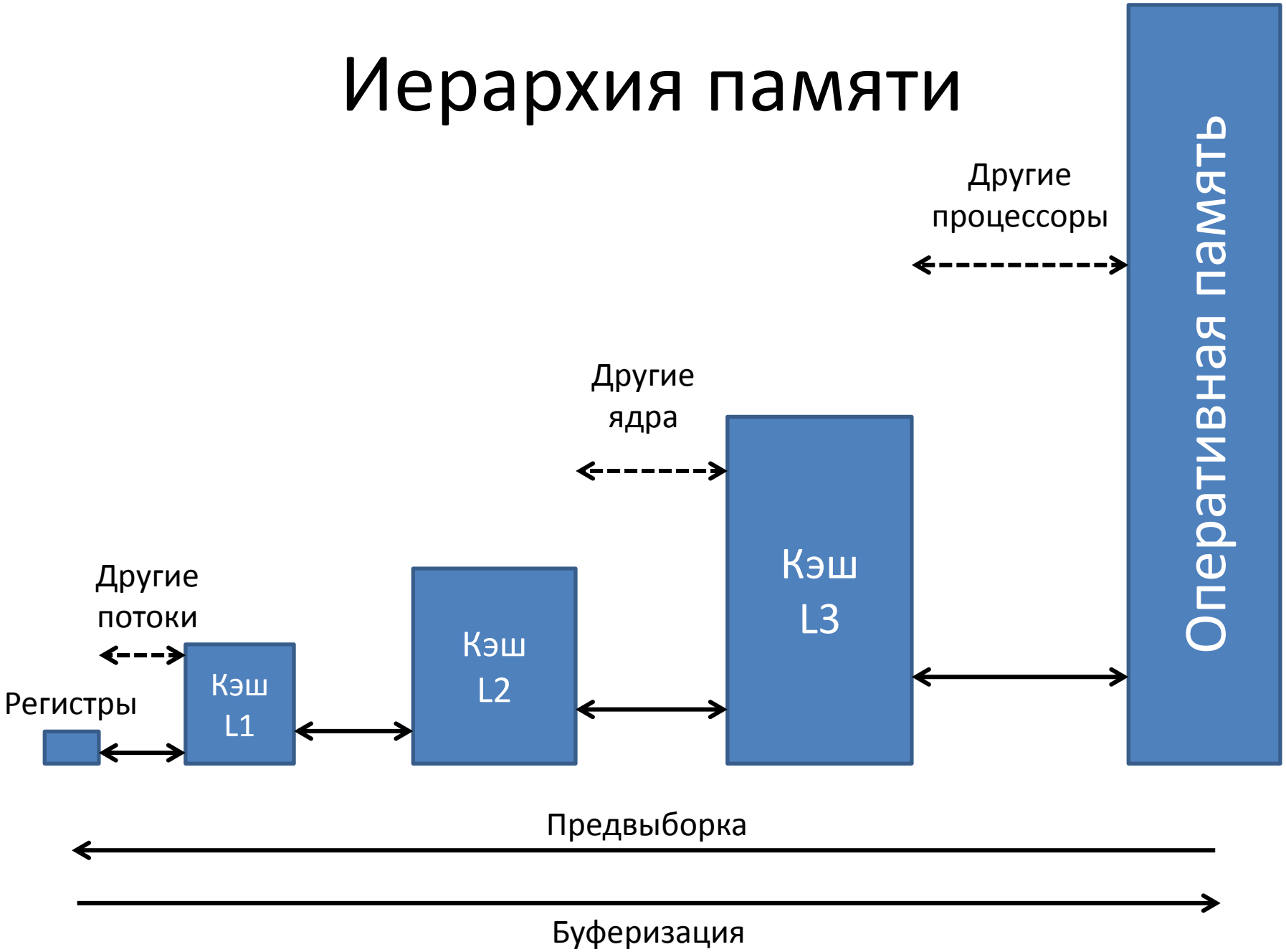
Эффективная работа с памятью

Иерархия памяти



На что следует обращать внимание при работе с памятью

- Выравнивание данных
- Плотность размещения данных
- Объём данных
- Порядок обхода данных
- Доступ к памяти нескольких потоков

Выравнивание данных

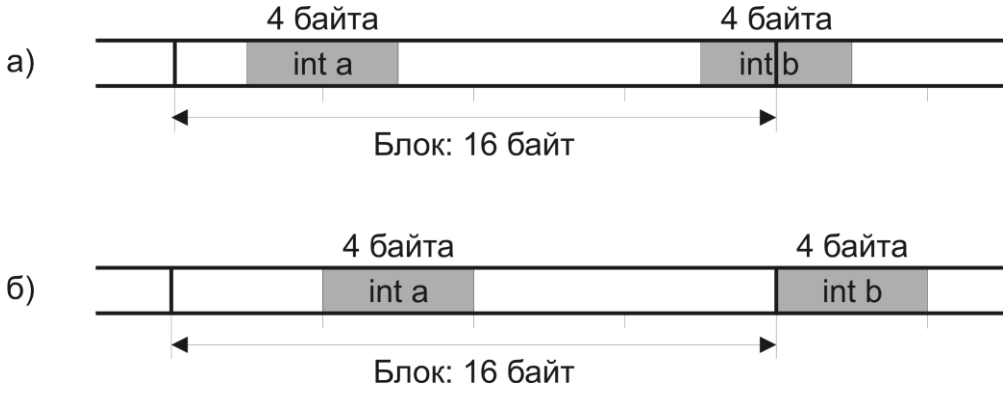
- Память делится на блоки

  в соответствии с размером страницы, кэш-строки, сектора кэша, шины данных, …

- Элементы данных могут пересекать границы блоков

  Это приводит к нескольким операциям чтения/записи вместо одной.

- Пример:



Выравнивание данных

- Естественное выравнивание
  - По размеру элемента данных (до машинного слова)
  - Применяется компилятором автоматически
- Выравнивание вручную

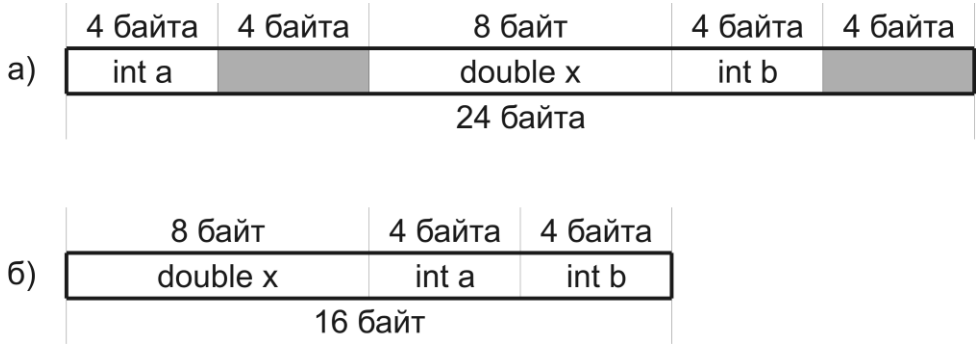
```
int x[N] __attribute__((aligned(64)));
ptr = malloc(size + 64);
ptra = (ptr % 64) ? (ptr & 0xffffffc0) + 64 : ptr;

err = posix_memalign(&ptr, 64, size);

ptr = _mm_malloc(size, 64);
```

Выравнивание данных

- Выравнивание элементов структур:

  struct S1 { int a; double x; int b; };

  struct S2 { double x; int a; int b; };



- Управление выравниванием в компиляторе

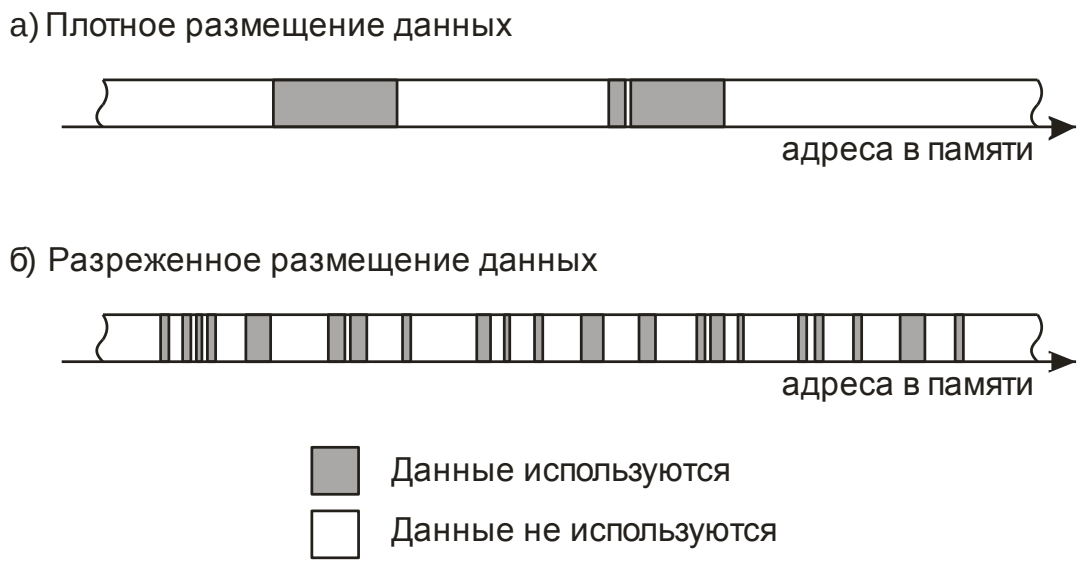
  #pragma pack(…)

Плотное и разреженное размещение данных

- Память делится на блоки

  в соответствии с размером страницы, кэш-строки, сектора кэша, шины данных, …

- Загрузка блока занимает время
- Плотно размещенные данные используют меньше блоков



Плотное и разреженное размещение данных

- Пример:

Вариант A	**Вариант B**
struct point { float x, y; };	float x[N];
struct point points[N];	float y[N];

а)

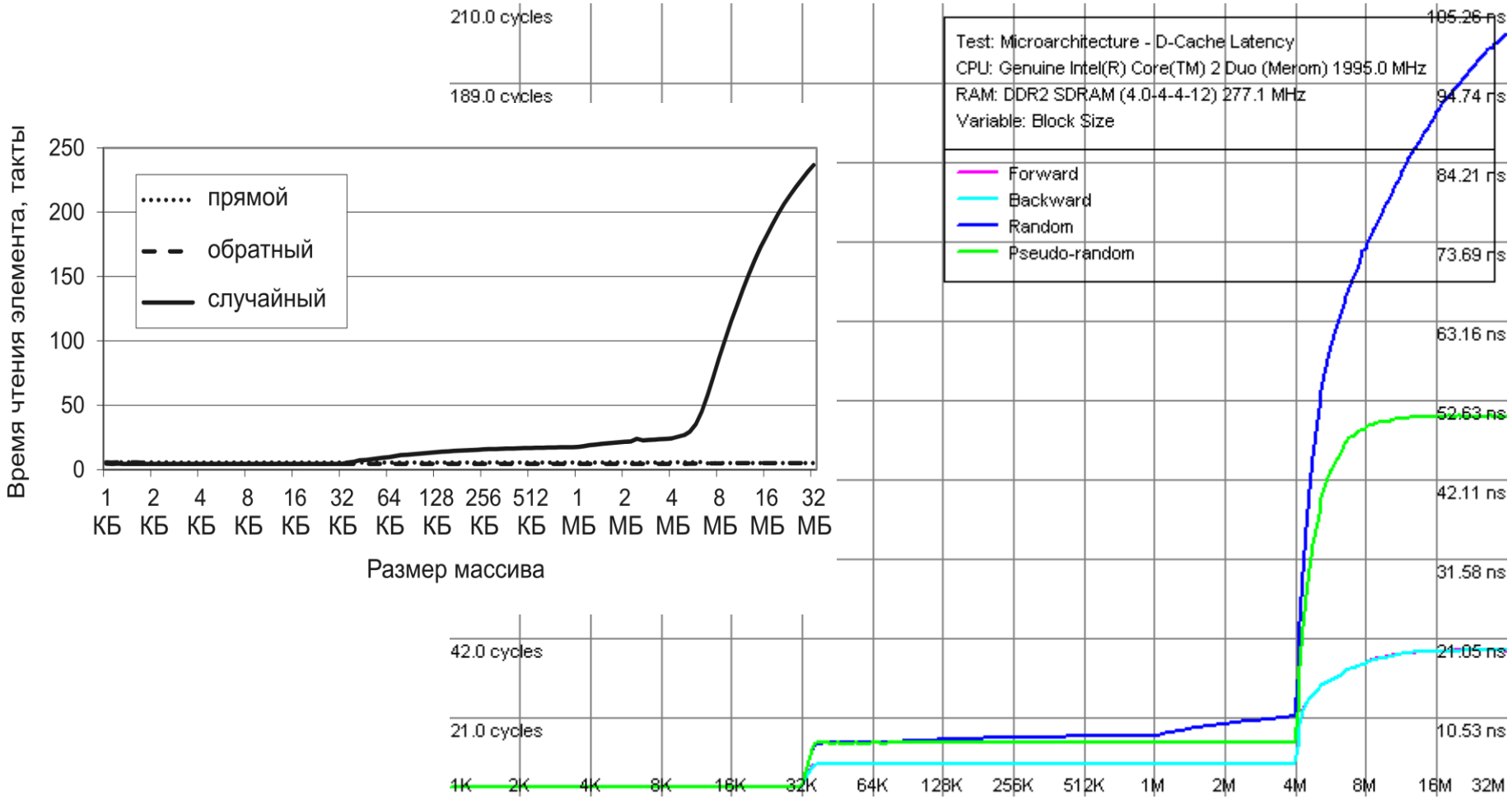
б)


```
for (i=0;i<N;i++)
  process(x[i],y[i])
```

```
for (i=0;i<N;i++) process(x[i]);
for (i=0;i<N;i++) process(y[i])
```

Объём данных

- Среднее время обращения к данным зависит от объема обрабатываемых данных:

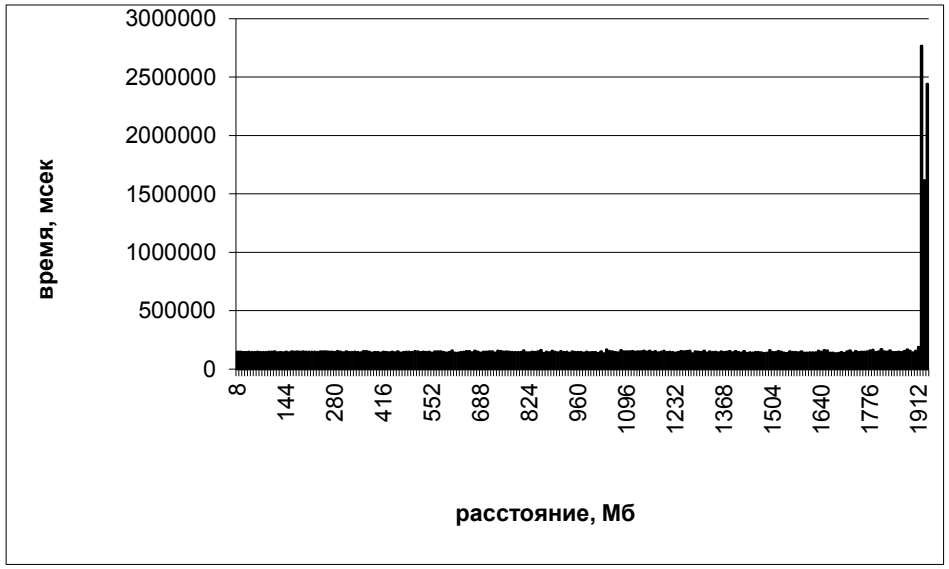


Объём данных

Доступ к памяти в многопроцессорных системах

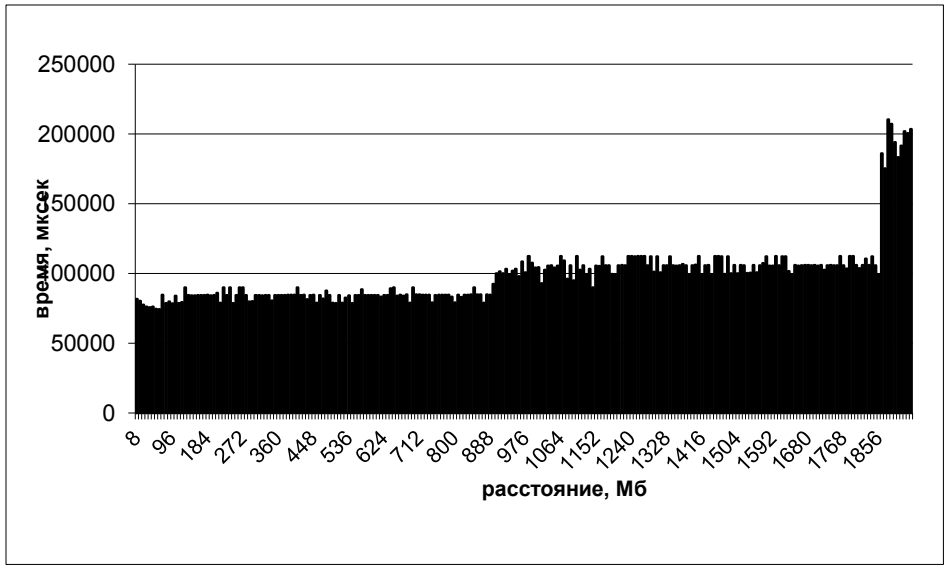
- Тест: копирование данных на различное расстояние

SMP – однородный доступ к памяти



2 x Alpha 21264

NUMA – неоднородный доступ к памяти

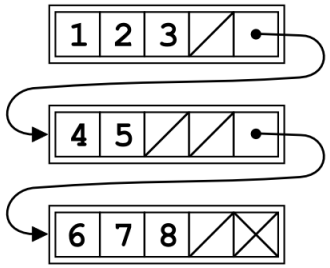


2 x Opteron 244

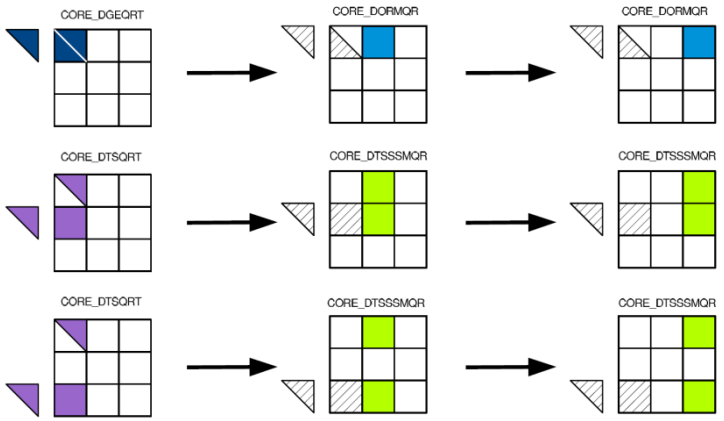
Объём данных

- Для увеличения скорости работы с памятью используются специальные алгоритмы
  - Cache-oblivious algorithms, Блочные (Tiled), …
  - Параметр – размер блока, зависит от размера кэша
- Идея: то, что загрузили, использовать по максимуму

Развернутый связный список

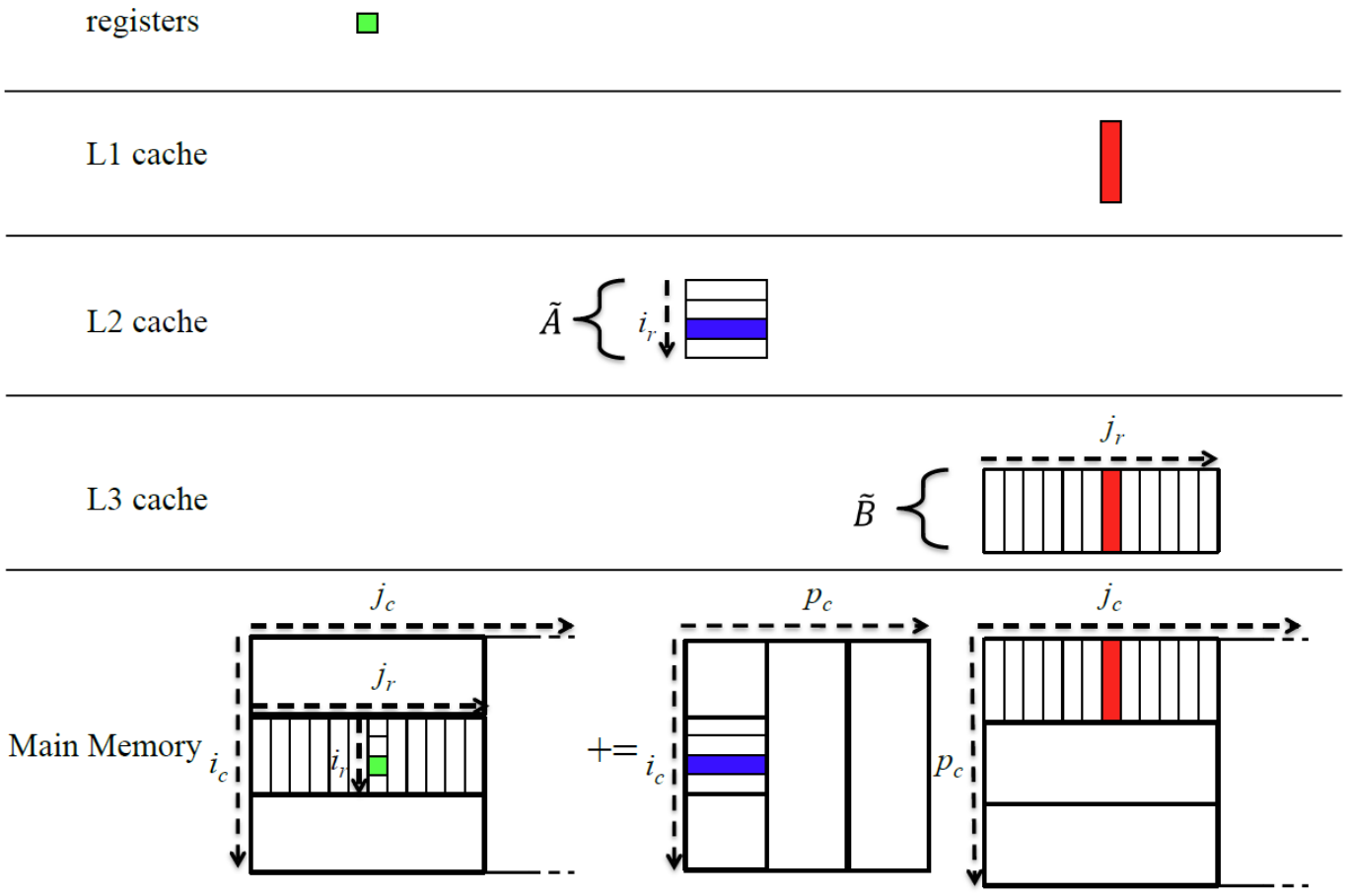


Блочные матричные алгоритмы



Объём данных

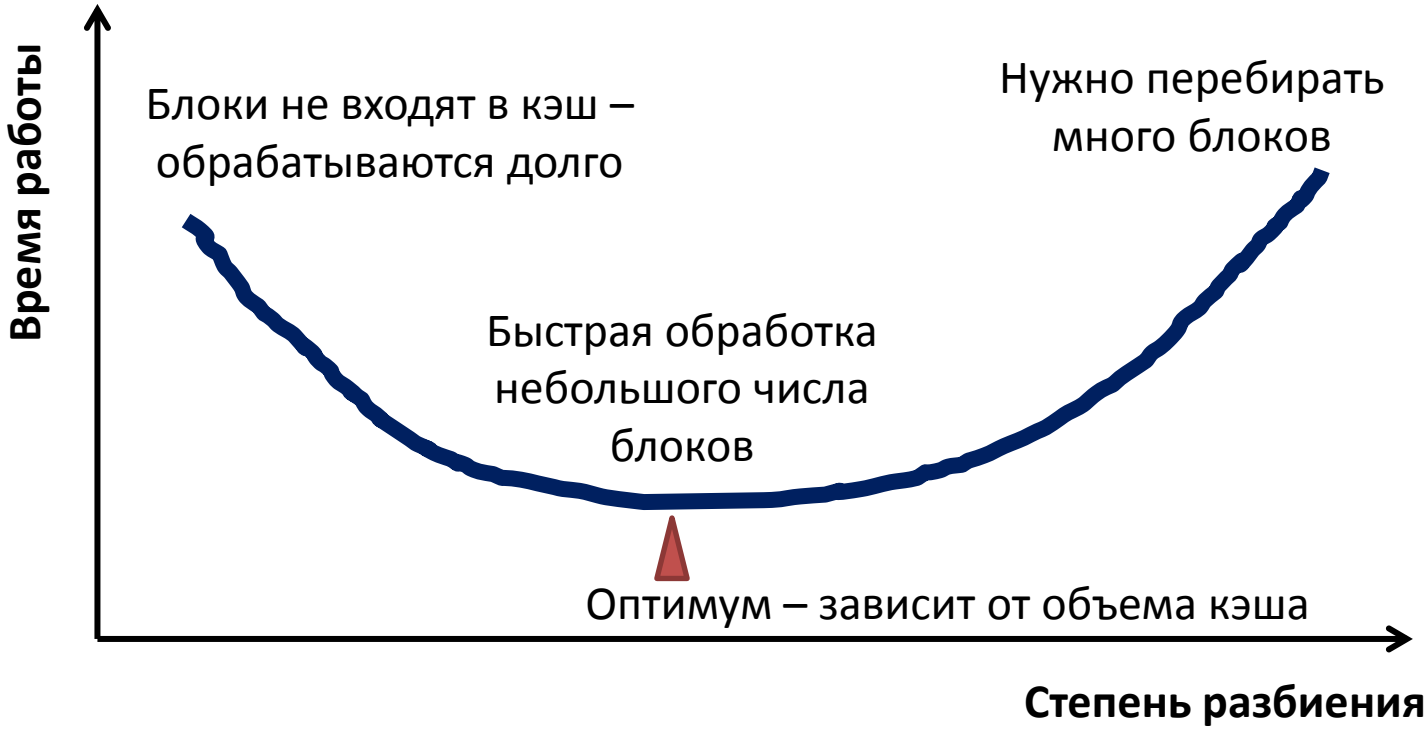
Планирование размещения данных в иерархии памяти для умножения матриц



Kazushige Goto, Robert A. van de Geijn. *Anatomy of High-Performance Matrix Multiplication.* // ACM Trans. Math. Softw., Vol. 34, No. 3., 2008.

Объём данных

- Типичная зависимость времени работы от степени разбиения задачи на блоки:



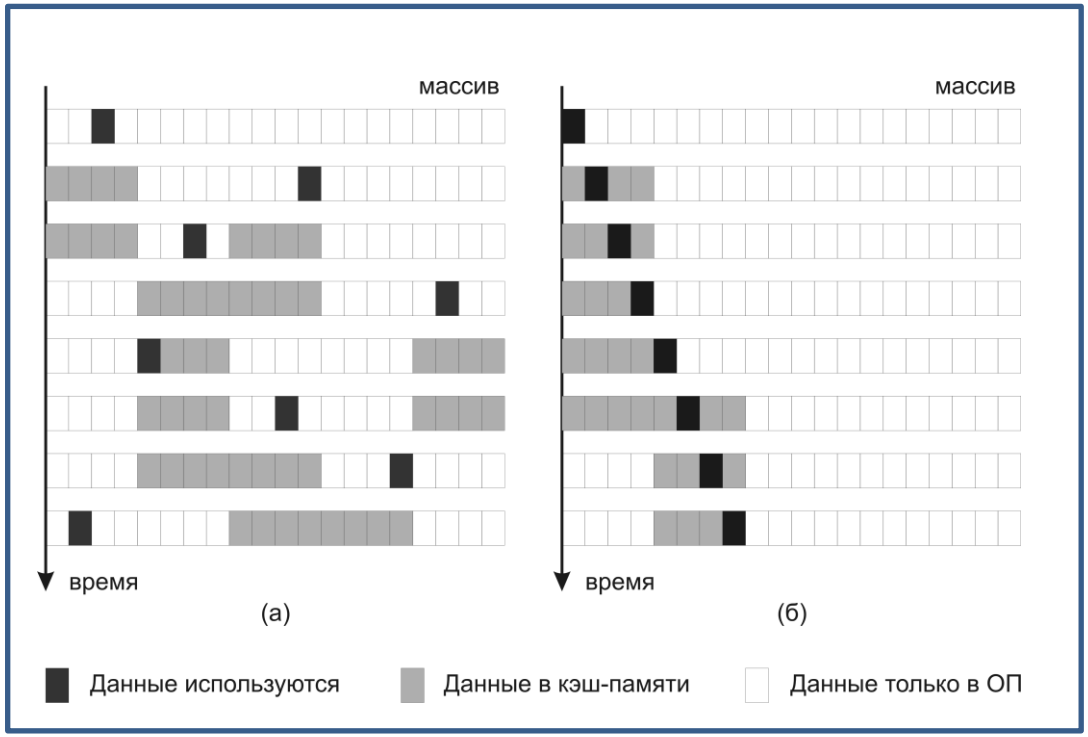
Объём данных

- Особенности виртуальной памяти
  - Память разделена на страницы (стандарт – 4 КБ)
  - Есть таблица страниц (в памяти)
  - Есть TLB – кэш таблицы страниц
    - множественно-ассоциативный – есть буксование
    - многоуровневый (L1/L2, данных/команд)
  - Обращение к новой странице (блоку в 4 КВ) очень долгое (сравнимо со временем чтения всей страницы)
- Как учитывать виртуальную память
  - Обращаться к данным компактно
  - Использовать большие страницы (huge pages)

Порядок обхода данных

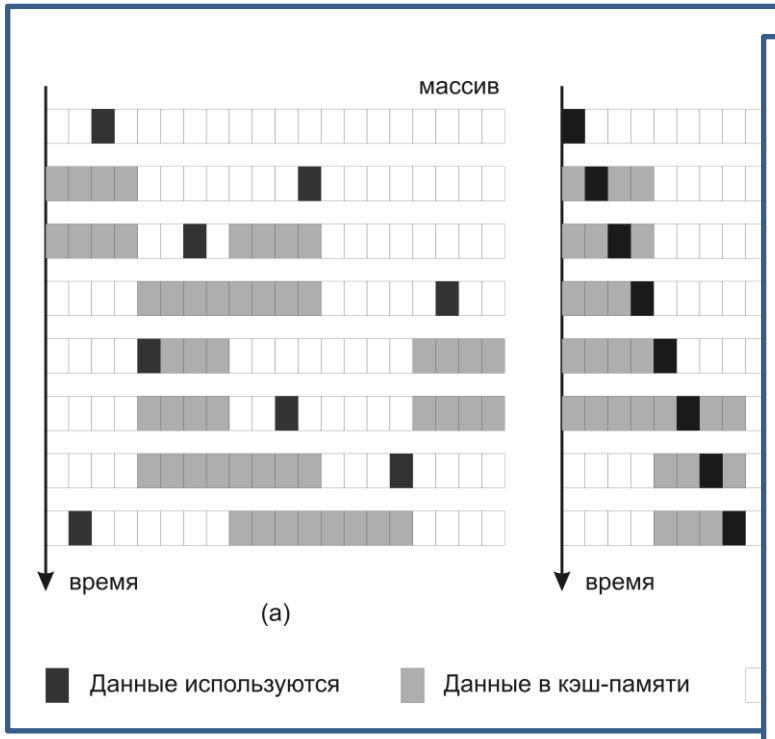
- Подсистема памяти оптимизирована для последовательного обхода
  - Блочное хранение и передача данных
  - Аппаратная предвыборка данных

Порядок обхода данных

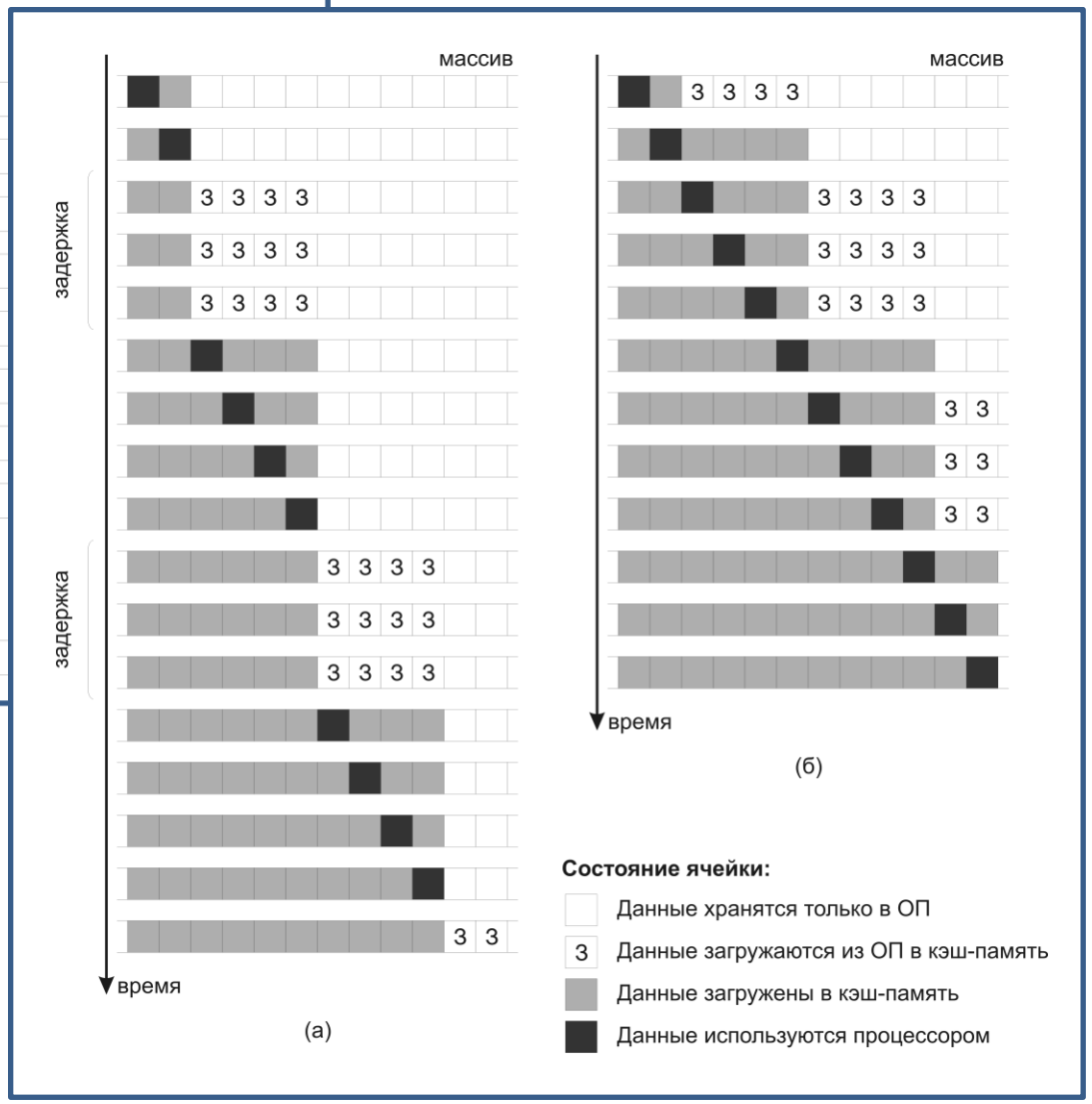


Блочное хранение и
передача данных

Порядок обхода данных



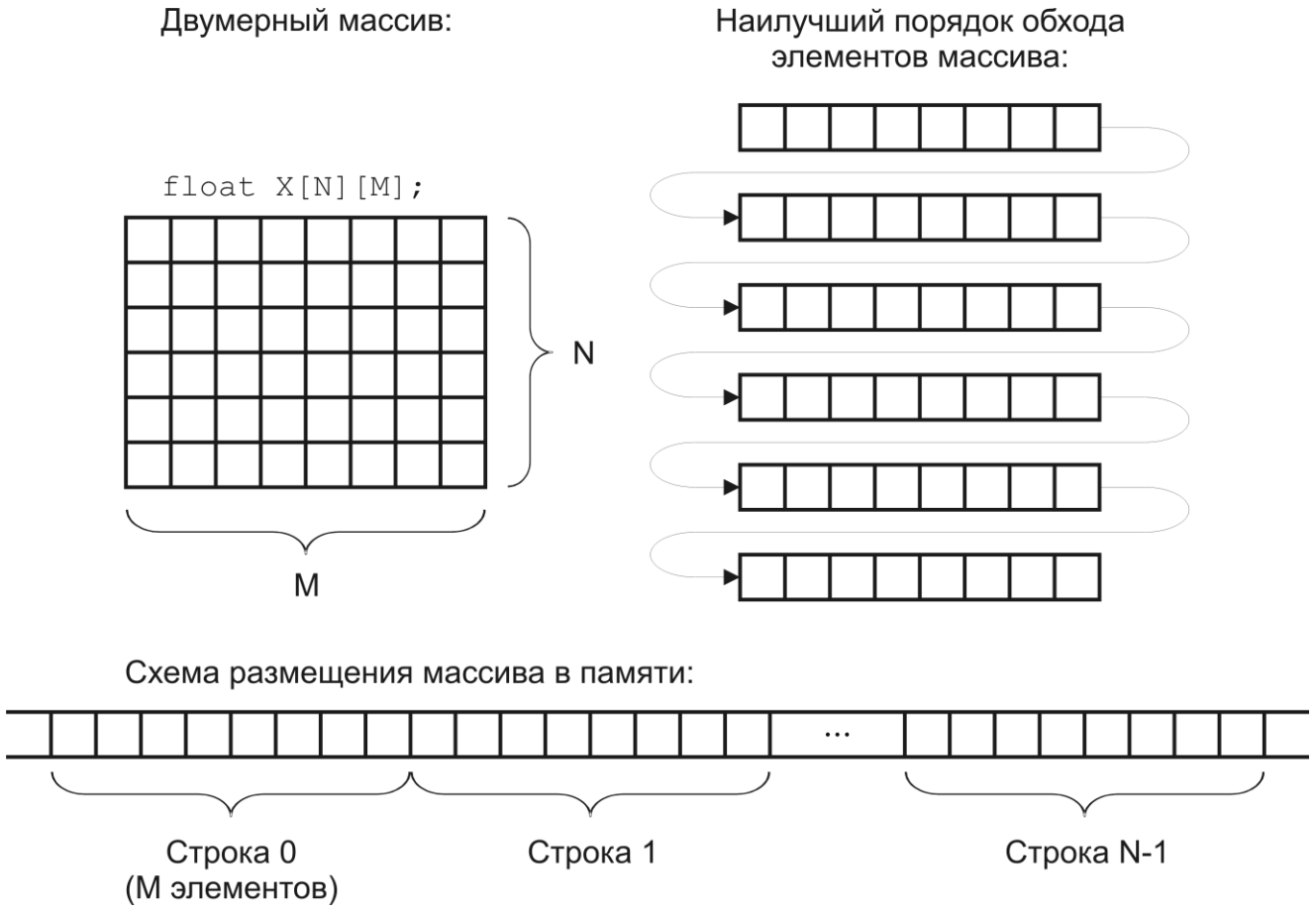
Блочное хранение и передача данных



Аппаратная предвыборка данных

Порядок обхода данных

- Расположение данных в памяти

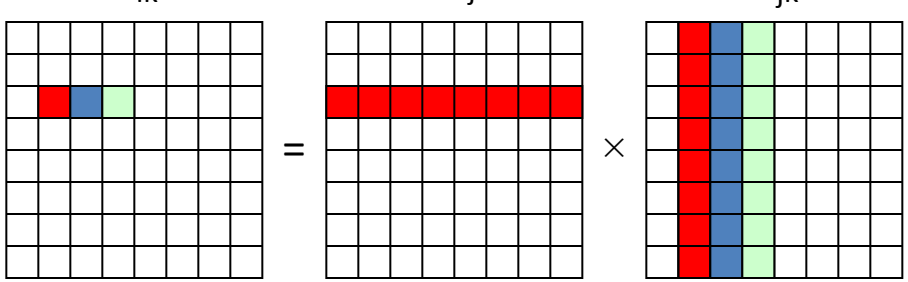


Порядок обхода данных

Пример: умножение матриц

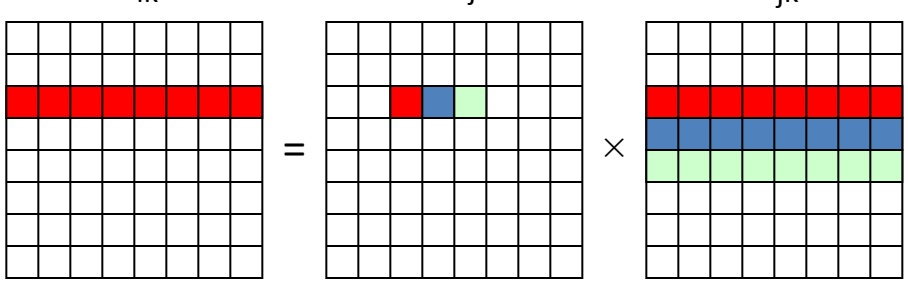
по формуле: $C_{ik} = \sum_{j=0}^{Ny-1} A_{ij} B_{jk}$

```
for(i=0;i<Nx-1;i++)
 for(k=0;k<Nz-1;k++)
  for(j=0;j<Ny-1;j++)
   C[i][k]+=A[i][j]*B[j][k];
```



переставим циклы

```
for(i=0;i<Nx-1;i++)
 for(j=0;j<Ny-1;j++)
  for(k=0;k<Nz-1;k++)
   C[i][k]+=A[i][j]*B[j][k];
```

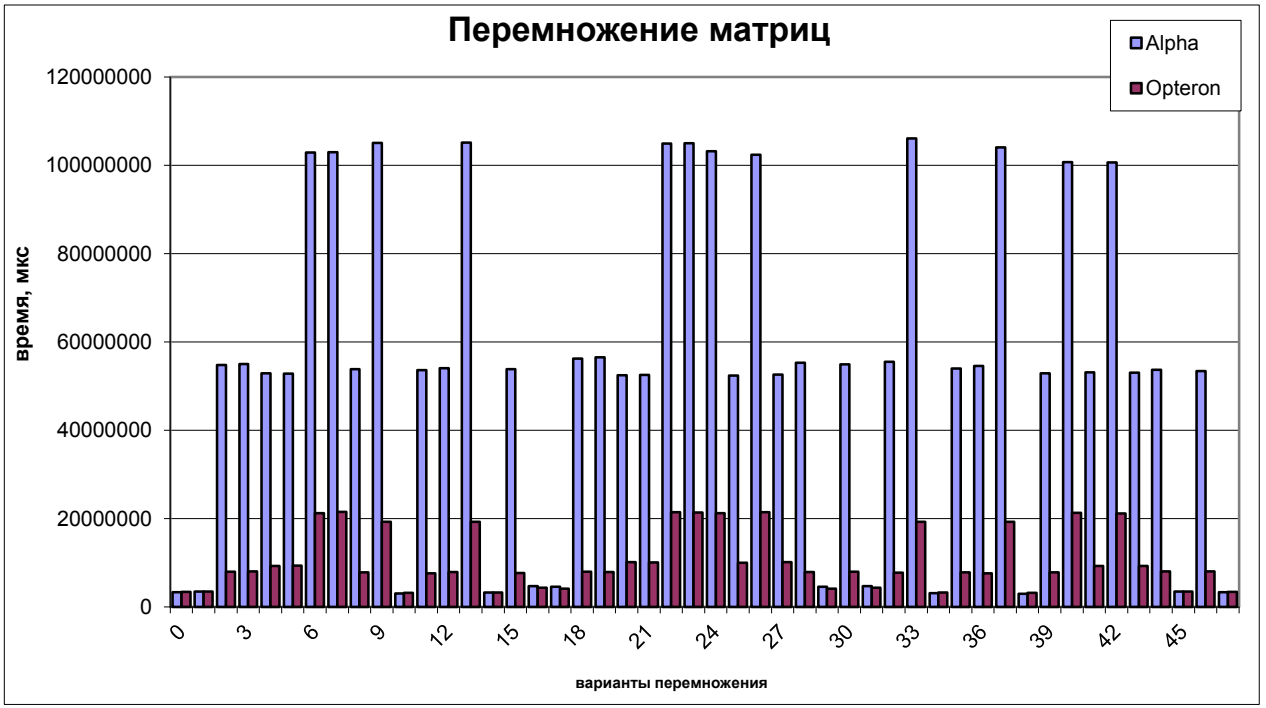


Время перемножения матриц 1000×1000		
120.67 с	**Alpha**	6.24 с
16.67 с	**Opteron**	6.3 с

Порядок обхода данных

- Пример: умножение матриц

```
for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    for (k=0; k<N; k++)
      C[i][j] += A[i][k] * B[k][j];
```



Порядок обхода данных

- Специальный случай обхода:

  кэш-буксование (cache-thrashing)

  Обход элементов с шагом кратным

  размеру банка = размер / степень ассоциативности

Порядок обхода данных

- Простой пример 1 (один массив):

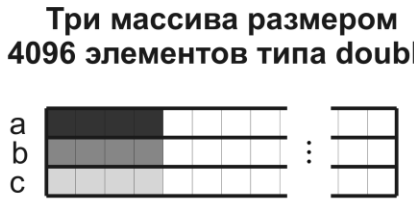
```
double a[4096000], sum[4096];
int i, j;
for (i=0; i<4096; i++) {
  sum[i]=0;
  for(j=0; j<1000; j++) sum[i] += a[i+j*4096];
}
```

- Простой пример 2 (несколько массивов):

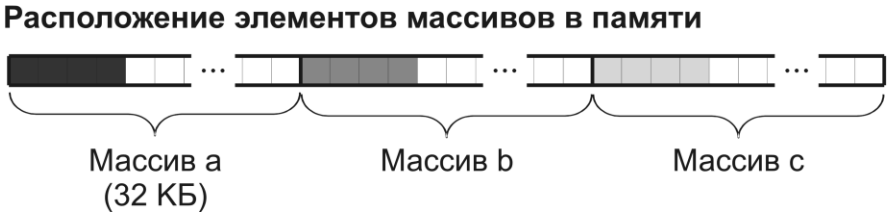
```
double a[4096], b[4096], c[4096];
int i;
for (i=0; i<4096; i++) c[i]=a[i]+b[i];
```

Порядок обхода данных

- Стандартные способы избежать кэш-буксования
  - Изменить порядок обхода
  - Изменить расстояние между элементами
- Пример:

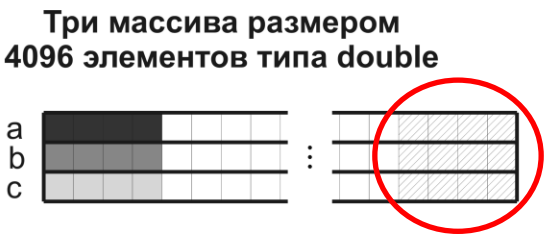



```
double a[4096], b[4096], c[4096];
int i;
for (i=0; i<4096; i++) c[i]=a[i]+b[i];
```

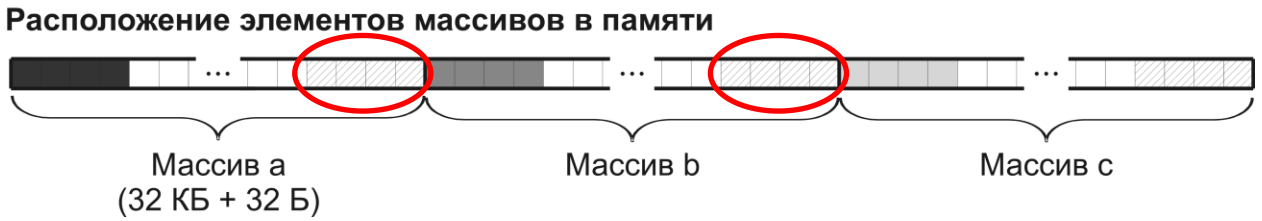


Порядок обхода данных

- Стандартные способы избежать кэш-буксования
  - Изменить порядок обхода
  - Изменить расстояние между элементами
- Пример:



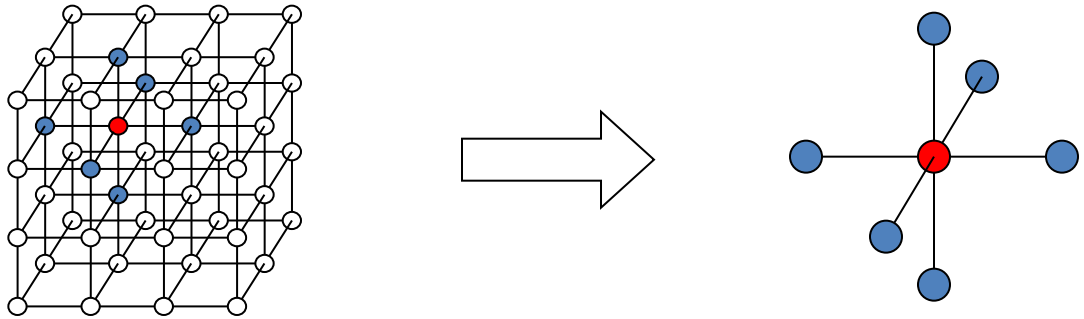

```
double a[4096 +4], b[4096 +4], c[4096 +4];
int i;
for (i=0; i<4096; i++) c[i]=a[i]+b[i];
```



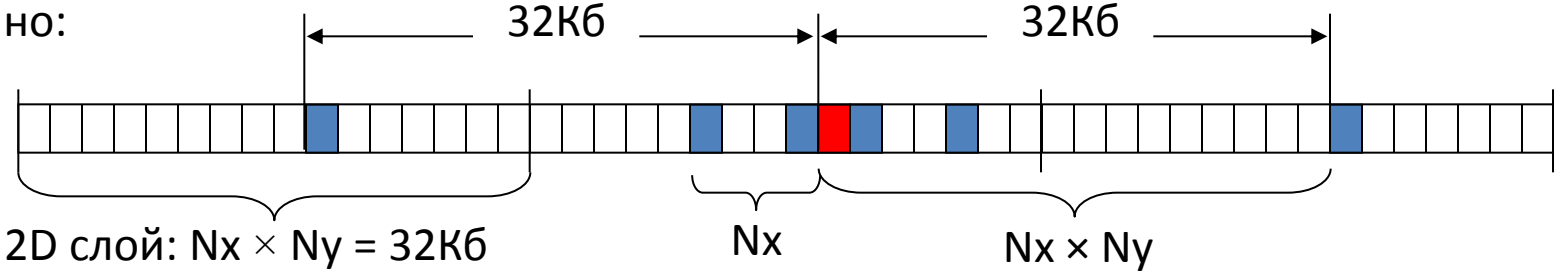
Порядок обхода данных

Пример: «буксование» кэша – семиточечная схема

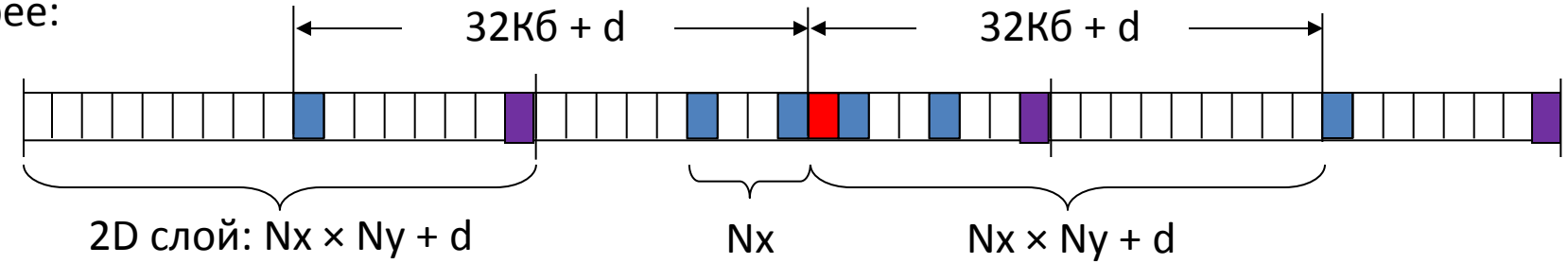
```
X[Nz][Ny][Nx];
```



Медленно:



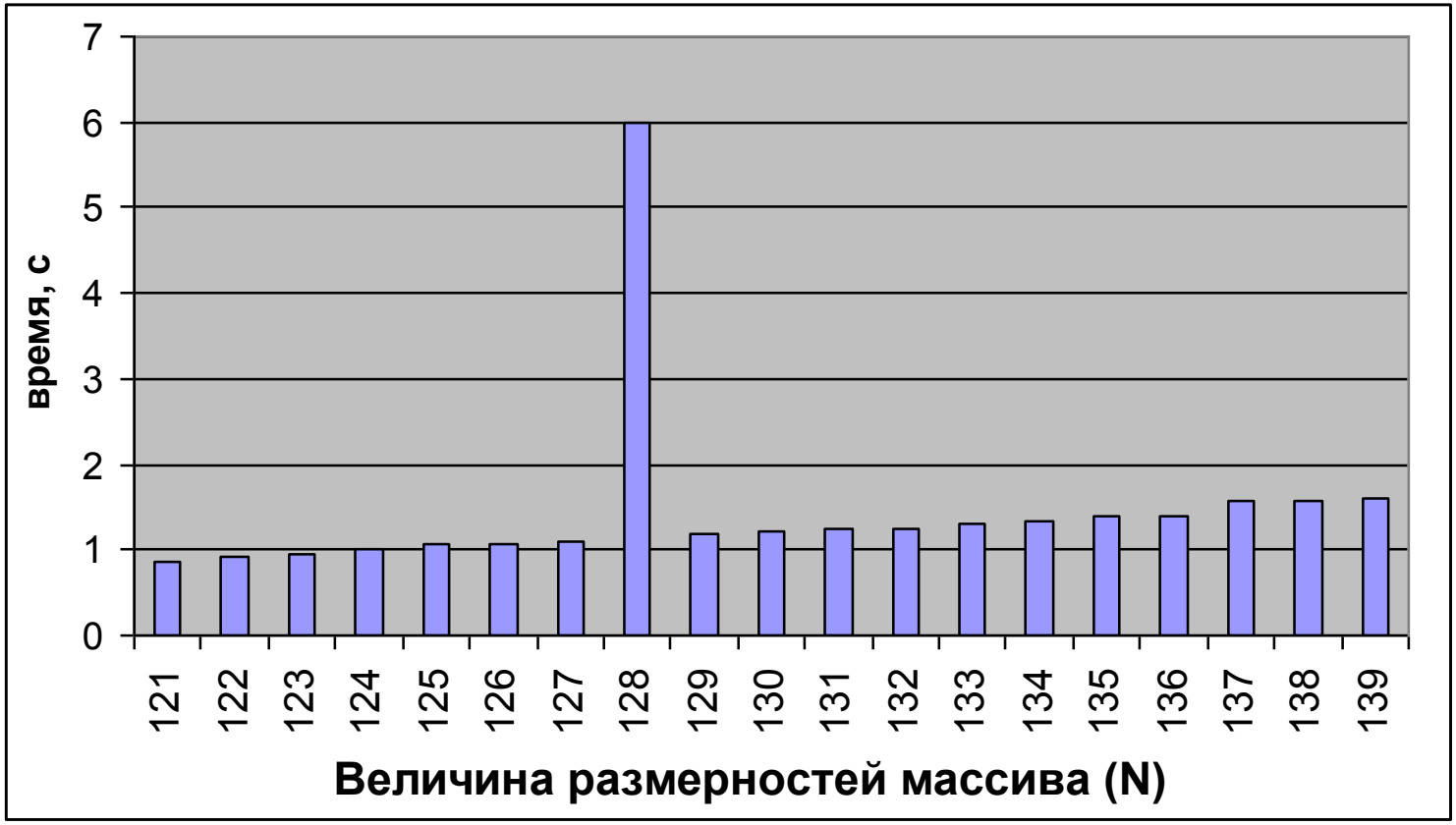
Быстрее:



Обращение к памяти

Пример: «буксование» кэша – семиточечная схема

Размер 3D массива: N × N × N



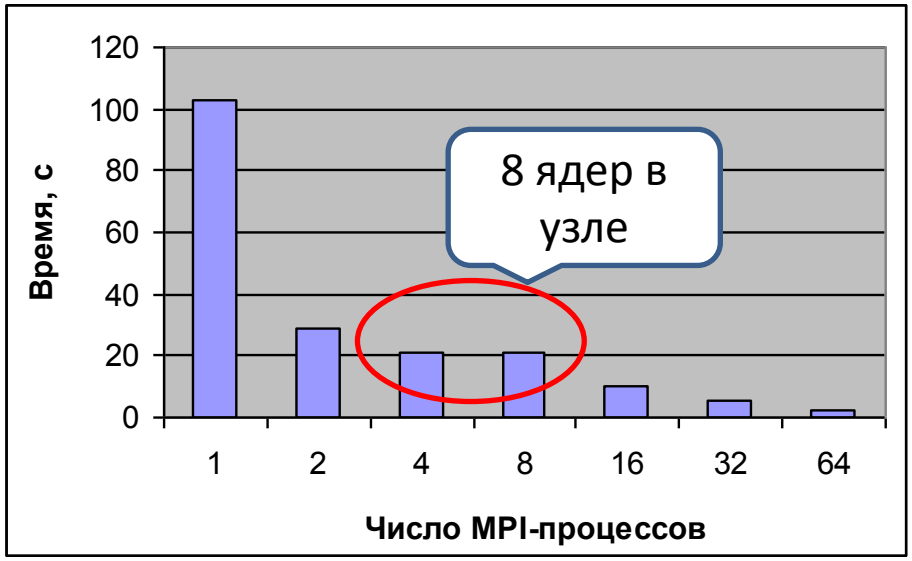
Размер 2D слоя для N=128: 128×128 = 16384 элементов = 64 Кб

Доступ к памяти нескольких потоков

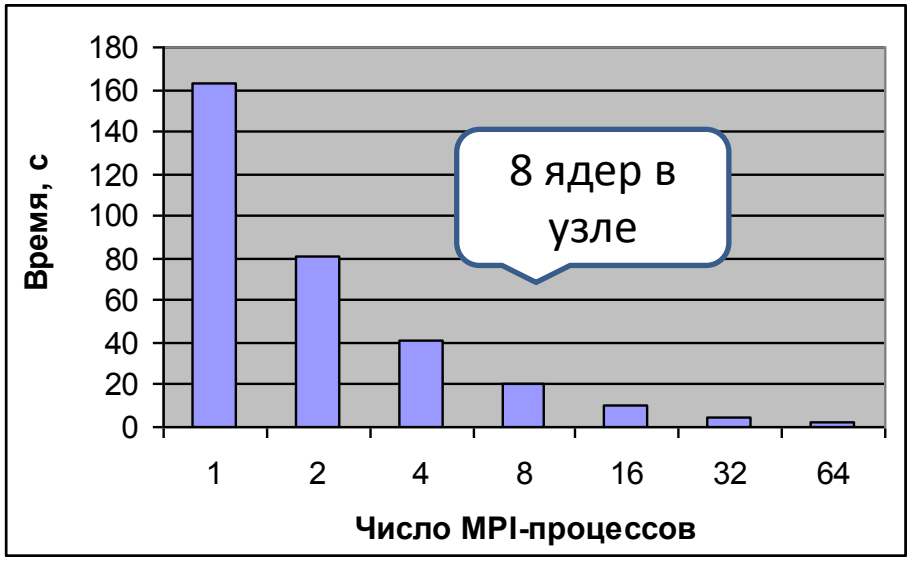
- Совместное использование ядрами кэш-памяти
  - Доступный потоку объем кэш-памяти меньше
- Совместное использование каналов доступа к памяти
  - Доступ потока к данным в памяти медленнее
- Поддержка когерентности кэш-памяти
  - Доступ потока к совместным данным медленнее
  - Возможно ложное разделение кэш-строк

Доступ к памяти нескольких потоков

- Совместное использование канала доступа к памяти
  - Доступ потока к данным в памяти медленнее



Xeon X5365 3.0 GHz



Itanium2 1.6 GHz

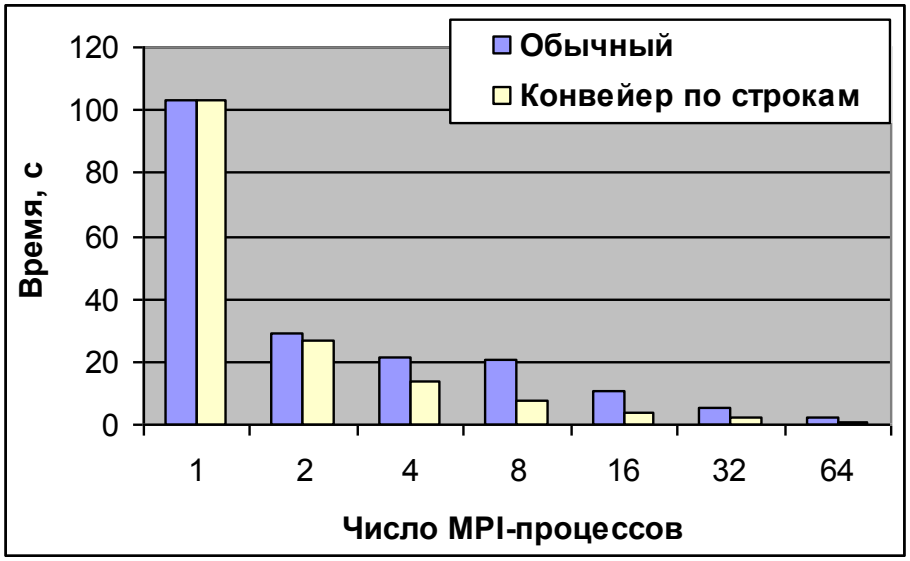
Доступ к памяти нескольких потоков

- Совместное использование канала доступа к памяти
  - Доступ потока к данным в памяти медленнее
- Решение
  - Реже обращаться в оперативную память – использовать кэш-память более эффективно

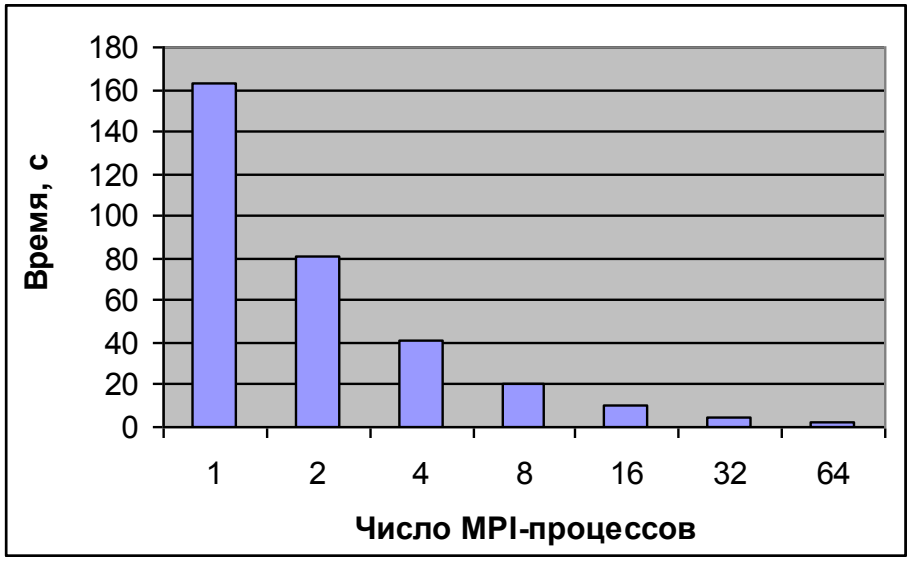
    (блочные алгоритмы…)

Доступ к памяти нескольких потоков

- Совместное использование канала доступа к памяти
  - Доступ потока к данным в памяти медленнее



Xeon X5365 3.0 GHz



Itanium2 1.6 GHz

Доступ к памяти нескольких потоков

- Поддержка когерентности кэш-памяти
  - требует постоянных обновлений данных в памяти при обращении нескольких потоков к одним и тем же данным – кэш работает неэффективно
- Пример:

```
int i, s;
#pragma omp parallel for
for (i=0; i<N; i++)
#pragma omp atomic
  s += i;
```

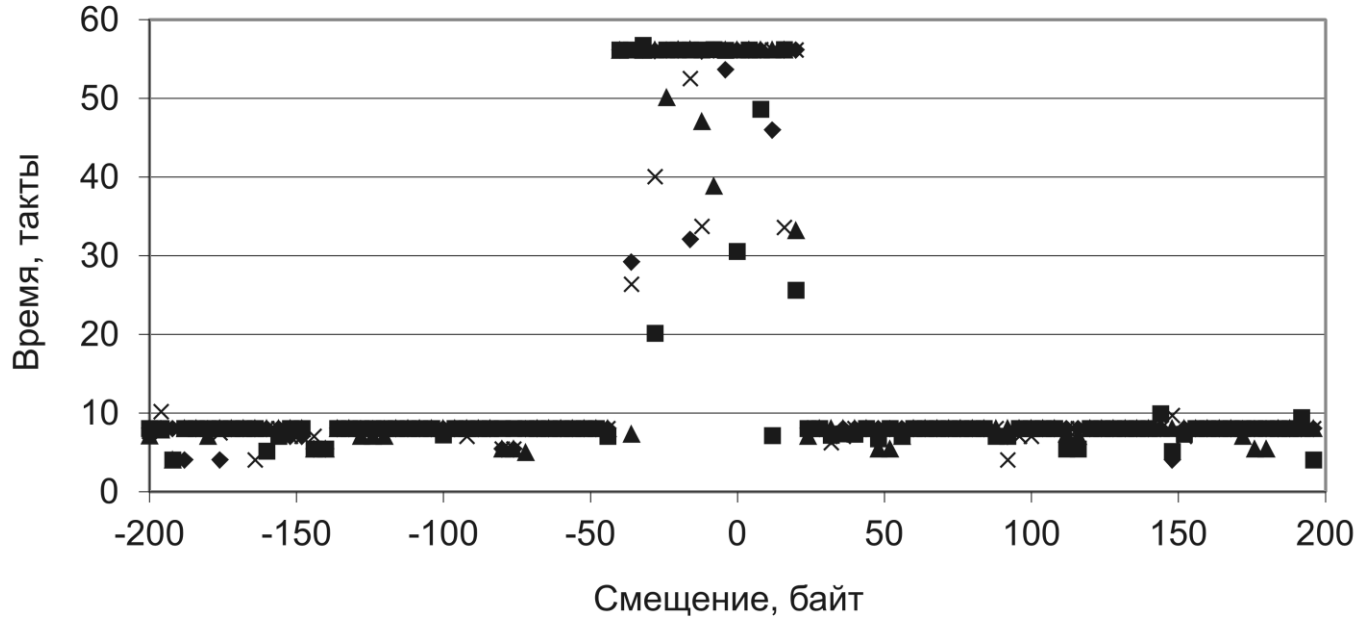
Доступ к памяти нескольких потоков

- Ложное разделение кэш-строк
  - Синхронизация данных происходит кэш-строками
  - Различные данные, размещенные в одной кэш-строке, будут тоже синхронизироваться
- Пример:

```
int i, s[nth];
#pragma omp parallel for num_threads(nth)
for (i=0; i<N; i++)
#pragma omp atomic
  s[ omp_get_thread_num() ] += i;
```

Доступ к памяти нескольких потоков

- Тест: доступ двух потоков к данным
  - Время работы одного из потоков в зависимости от расстояния между данными потоков:



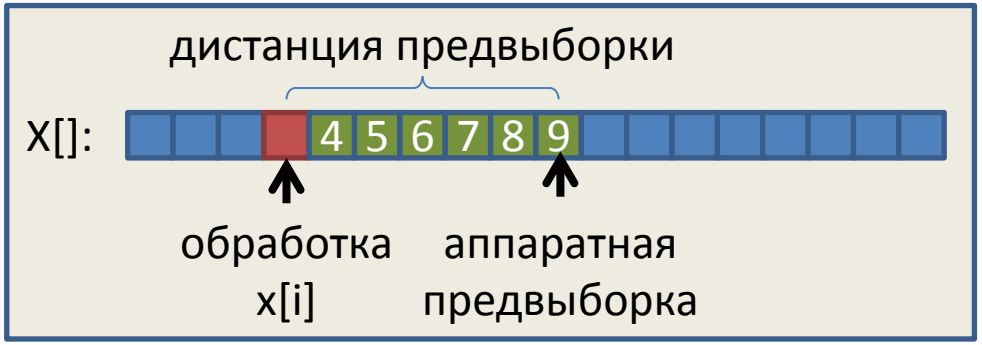
Средства «продвинутой» работы с памятью

- Программная предвыборка данных в кэш
  - _mm_prefetch(ptr, _MM_HINT_NTA);
  - _MM_HINT_T0, _MM_HINT_T1, _MM_HINT_T2, _MM_HINT_NTA
- Запись в память без записи в кэш
  - _mm_stream_ps(ptr, v128);

Средства «продвинутой» работы с памятью

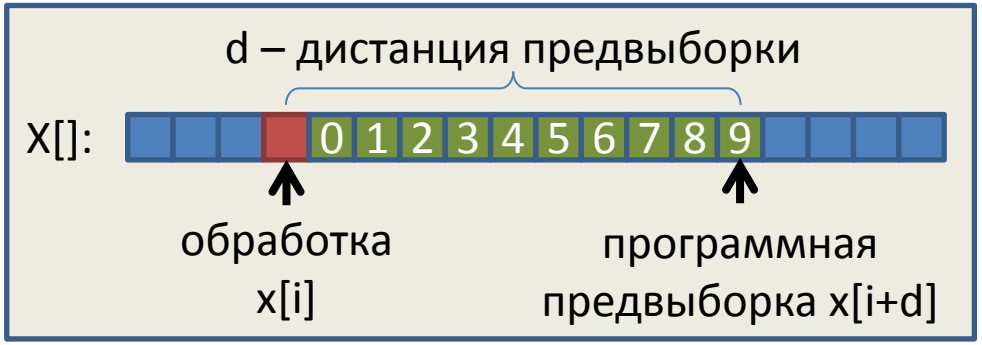
- Аппаратная предвыборка данных

```
for (i=0; i<N; i++)
{
  process(x[i]);
}
```



- Программная предвыборка данных

```
for (i=0; i<N; i++)
{
  _mm_prefetch(&x[i+d], 0);
  process(x[i]);
}
// d – дистанция предвыборки
```



Средства «продвинутой» работы с памятью

- Запись в память без записи в кэш
- Пример: копирование массива

```
for (i=0; i<N; i++)
{
  _mm_prefetch(&x[i+d],0);
  _mm_stream_ps(&y[i], x[i]);
}
```