Казанский федеральный
УНИВЕРСИТЕТ
ВЫСШАЯ ШКОЛА
информационных технологий
и информационных систем

Оптимизация алгоритмов с использованием OpenMP

Эдуард Храмченков

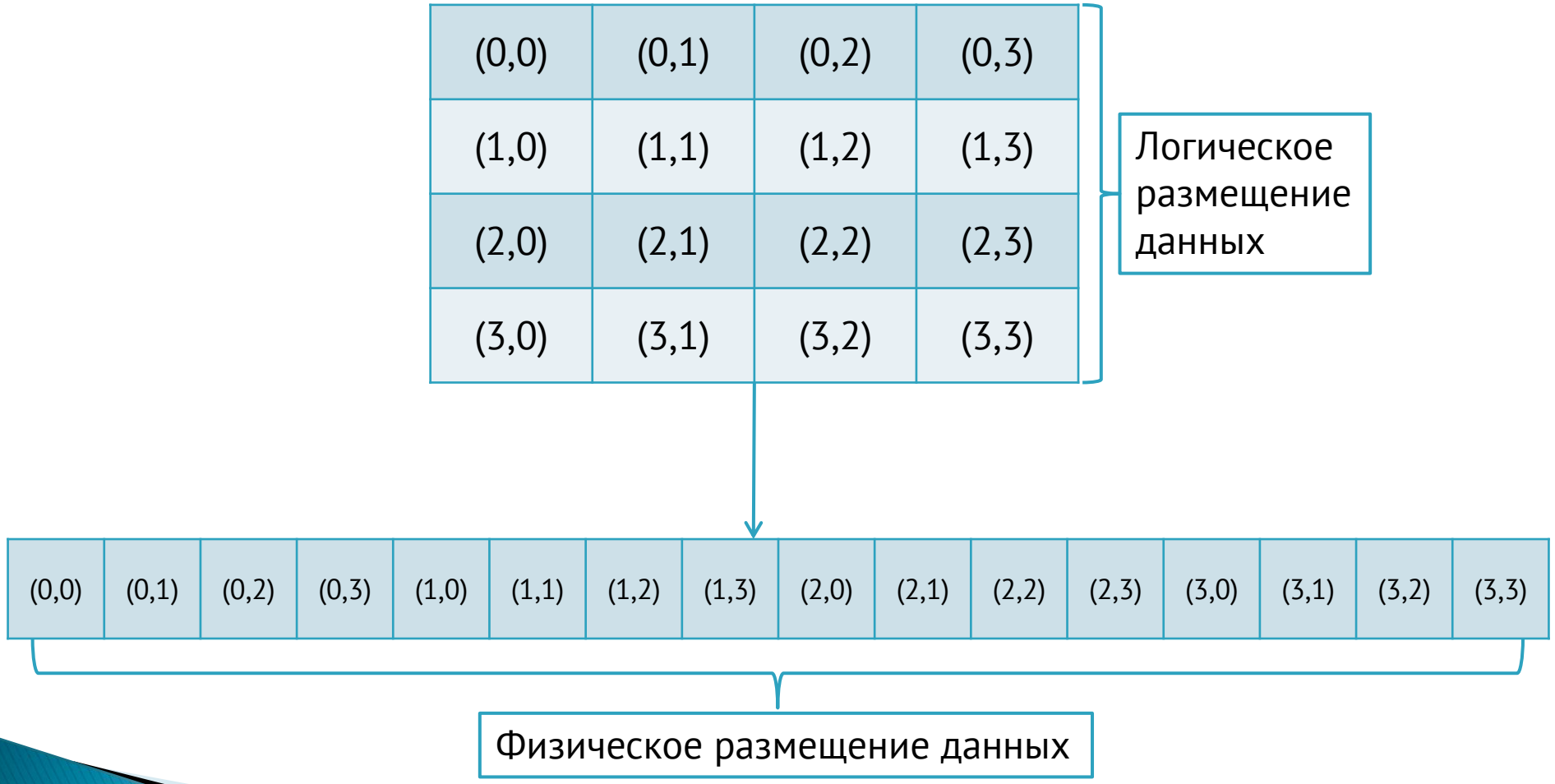
Доступ к памяти

- Основная память – медленная
- Часть данных которые требуются программе попадают в кэш
- Если данных нет в кэше, программа обращается в основную память (cache miss – промах мимо кэша)
- Данные размещаются в кэше кусками, которые называются блоками
- Каждый блок занимает одну строку кэша

Доступ к памяти

- Необходимо по возможности минимизировать число промахов
- Как правило нет возможности управлять тем, какие данные попадут в кэш
- Используемые данные должны максимально попадать в кэш
- Пример – в C/C++ 2-мерные массивы хранятся в памяти последовательно по рядам

Доступ к памяти



Доступ к памяти

- Если программа обращается к элементу (0, 0) в кэш попадет целая строка массива
- Доступ к элементам массива необходимо осуществлять построчно, а не столбцами

```
for (int i=0; i<n; i++)
        for (int j=0; j<n; j++)
                sum+=a[i][j];
```

```
for (int j=0; j<n; j++)
        for (int j=0; j<n; j++)
                sum+=a[i][j];
```

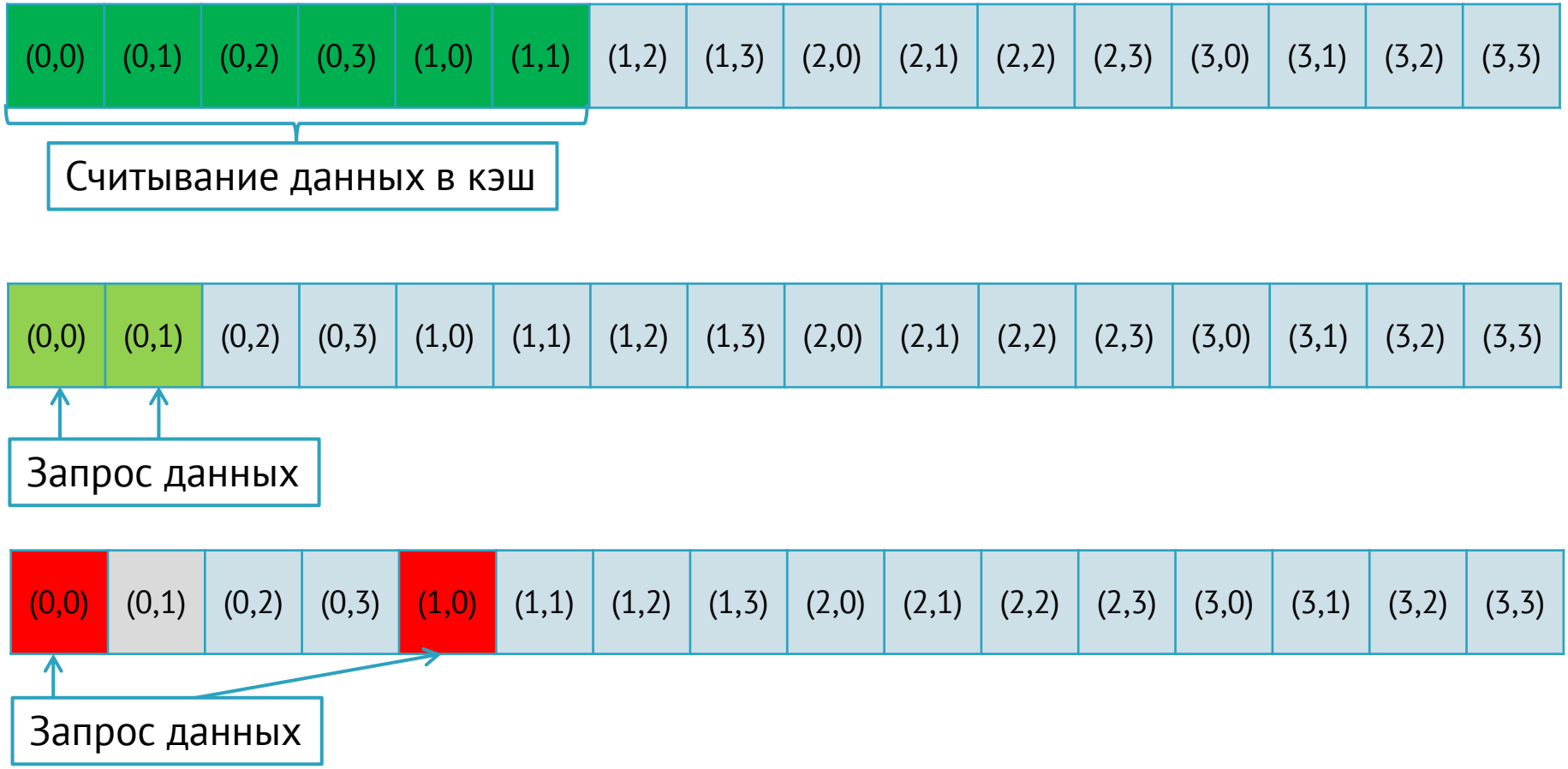
Доступ к памяти

- Если программа обращается к элементу (0, 0) в кэш попадет целая строка массива
- Доступ к элементам массива необходимо осуществлять построчно, а не столбцами

```
for (int i=0; i<n; i++)
        for (int j=0; j<n; j++)
                sum+=a[i][j];
```

~~
```
for (int j=0; j<n; j++)
        for (int i=0; i<n; i++)
                sum+=a[i][j];
```
~~

Доступ к памяти



Доступ к памяти

- В C/C++ использовать многомерные массивы/векторы невыгодно с точки зрения производительности (или boost/eigen/etc)
- 2- и 3-мерные массивы разворачиваются в одномерные, где элементы соотносятся как:

$$a[i][j] = \bar{a}[i * N_x + j]$$

$$a[i][j][k] = \bar{a}[i * N_x + j * N_y + k]$$

Оптимизация циклов

- Переупорядочивание цикла помогает улучшить паттерн доступа к памяти
- Переупорядочивание не должно изменять порядок доступа к одному и тому же месту в памяти
- Переупорядочивание способно также улучшить порядок выполнения команд на конвейере процессора и увеличить параллельные регионы

Оптимизация циклов

```
for (int j=0; j<n; j++)
        for (int i=0; i<m; i++)
                a[i][j+1] = a[i+1][j] + b;
```

```
for (int i=0; i<n; i++)
        for (int j=0; j<m; j++)
                a[i][j+1] = a[i+1][j] + b;
```

Оптимизация циклов

- Слияние циклов – объединение нескольких циклов в один для снижения издержек и увеличения вычислений на итерацию
- Слияние не должно изменять порядок доступа к одному и тому же месту памяти
- Иногда необходимо изменение кода итерации объединенного цикла для обеспечения корректности вычислений

Оптимизация циклов

```
for (int i=0; i<n; i++)
        a[i] = b[i] * 2;
for (int i=0; i<n; i++)
{
        x[i] = 2 * x[i];
        c[i] = a[i] + 2;
}
```

```
for (int i=0; i<n; i++)
{
        a[i] = b[i] * 2;
        c[i] = a[i] + 2;
        x[i] = 2 * x[i];
}
```

Оптимизация циклов

- Разделение цикла – операция по разбиению итерации цикла на несколько итераций в разных циклах
- Полезна в случаях если данные итераций не умещаются в кэш или если разные части итерации требуют разной оптимизации
- Пример – переупорядочивание исходного цикла невозможно из-за c[i]

Оптимизация циклов

```
for (int i=0; i<n; i++)
{
        c[i] = exp(i/n) ;
        for (int j=0; j<m; j++)
                a[j][i] = b[j][i] + d[j] * c[i];
}
```



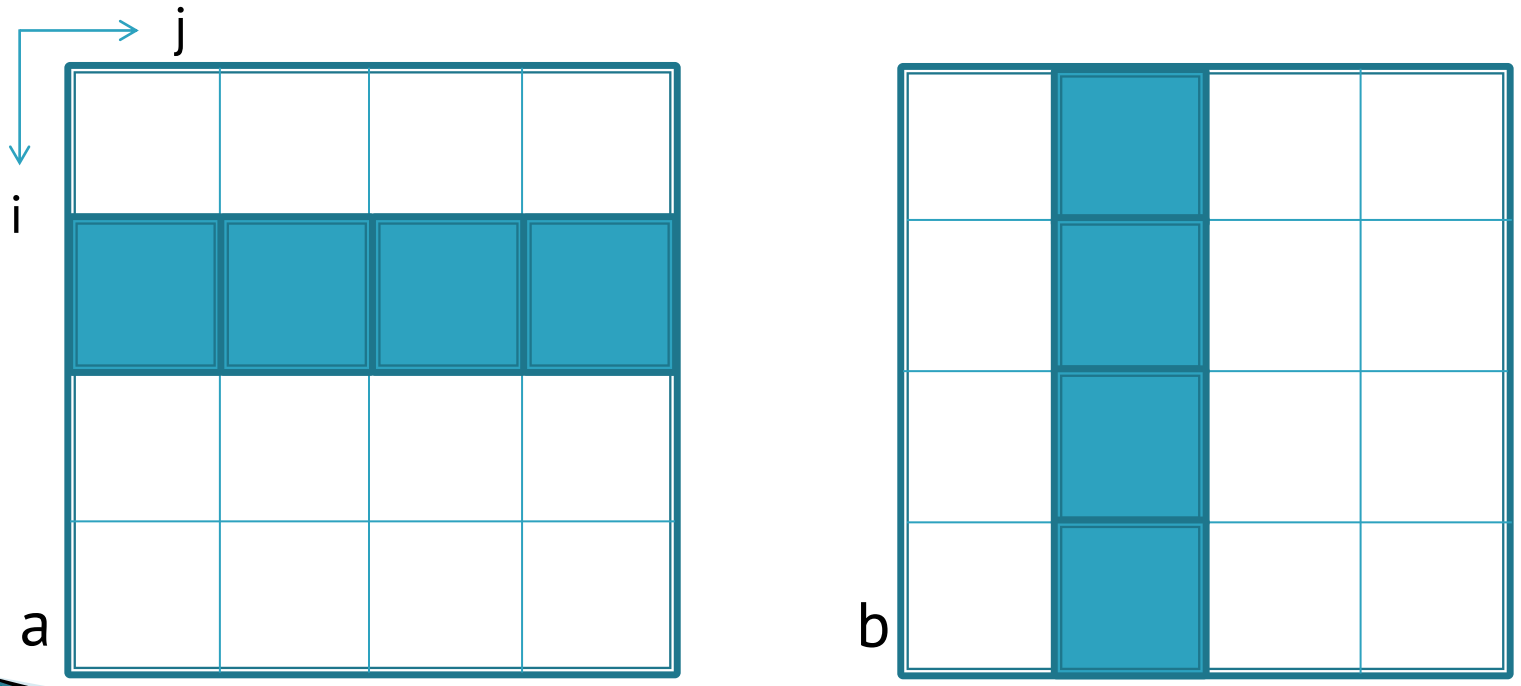
```
for (int i=0; i<n; i++)
        c[i] = exp(i/n) ;

for (int j=0; j<m; j++)
        for (int i=0; i<n; i++)
                a[j][i] = b[j][i] + d[j] * c[i];
```

Оптимизация циклов

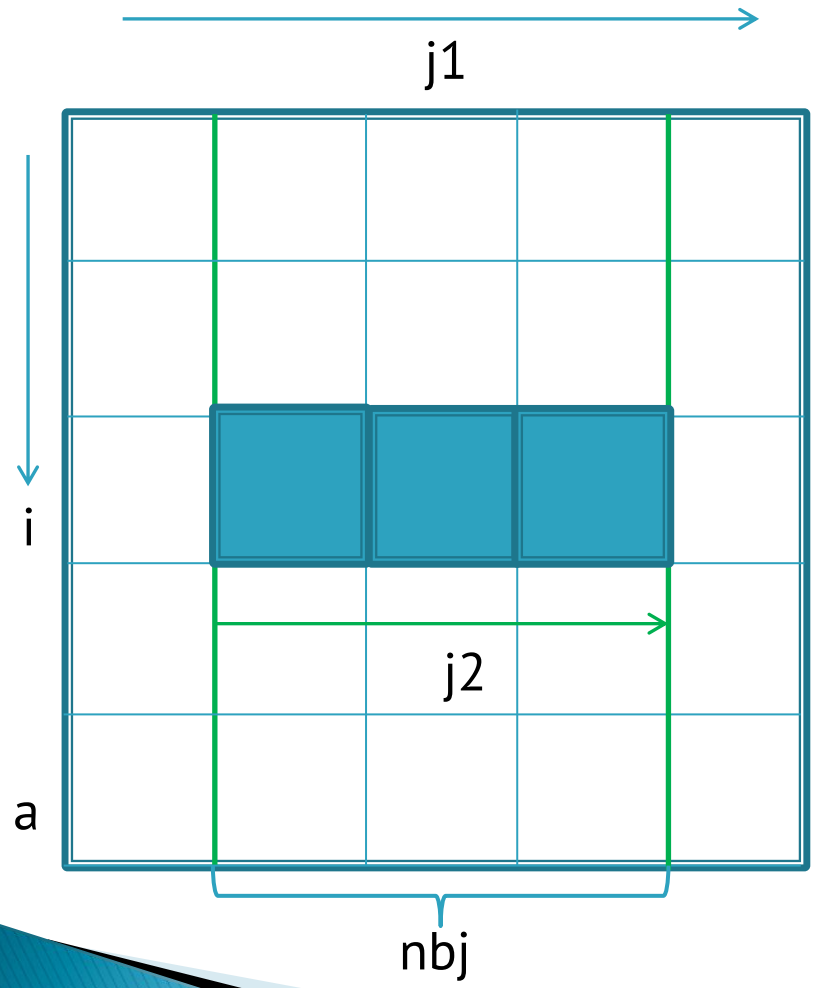
- Разбиение на блоки – эффективная техника для подгонки данных цикла под попадание в кэш
- Применима в случаях когда данные имеют слишком большой объем, когда паттерн доступа к памяти неэффективен, когда итерации цикла зависят друг от друга
- Заменяет исходный цикл на несколько, обрабатывающих каждый свои данные

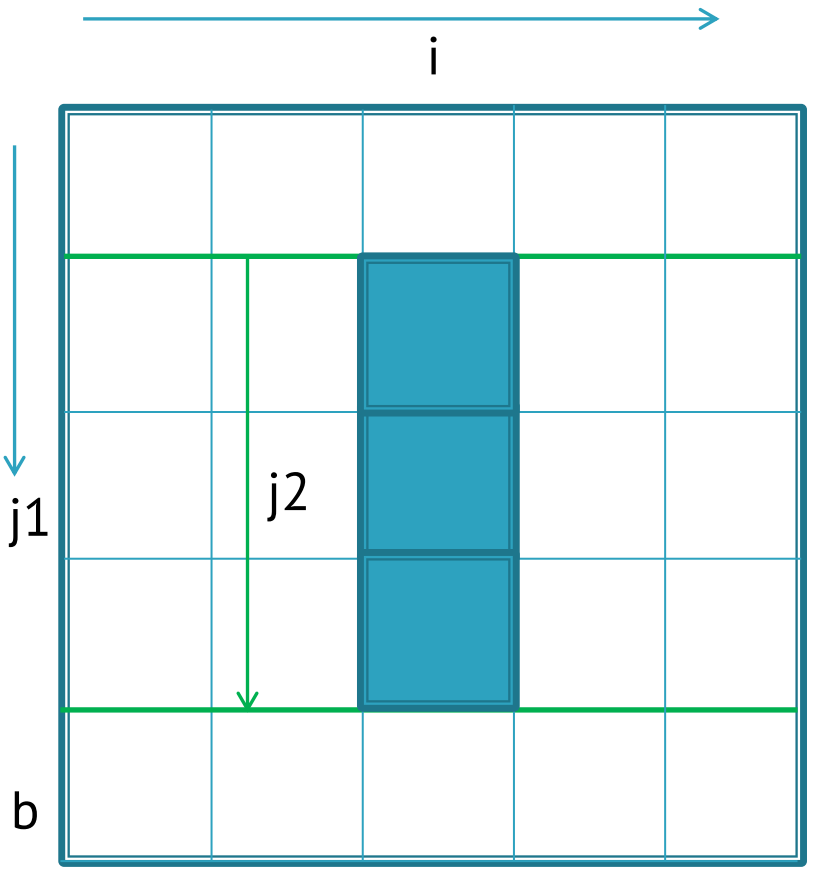
Пример 1

- Транспонирование матрицы – плохой паттерн доступа к памяти у матрицы b



Пример 1



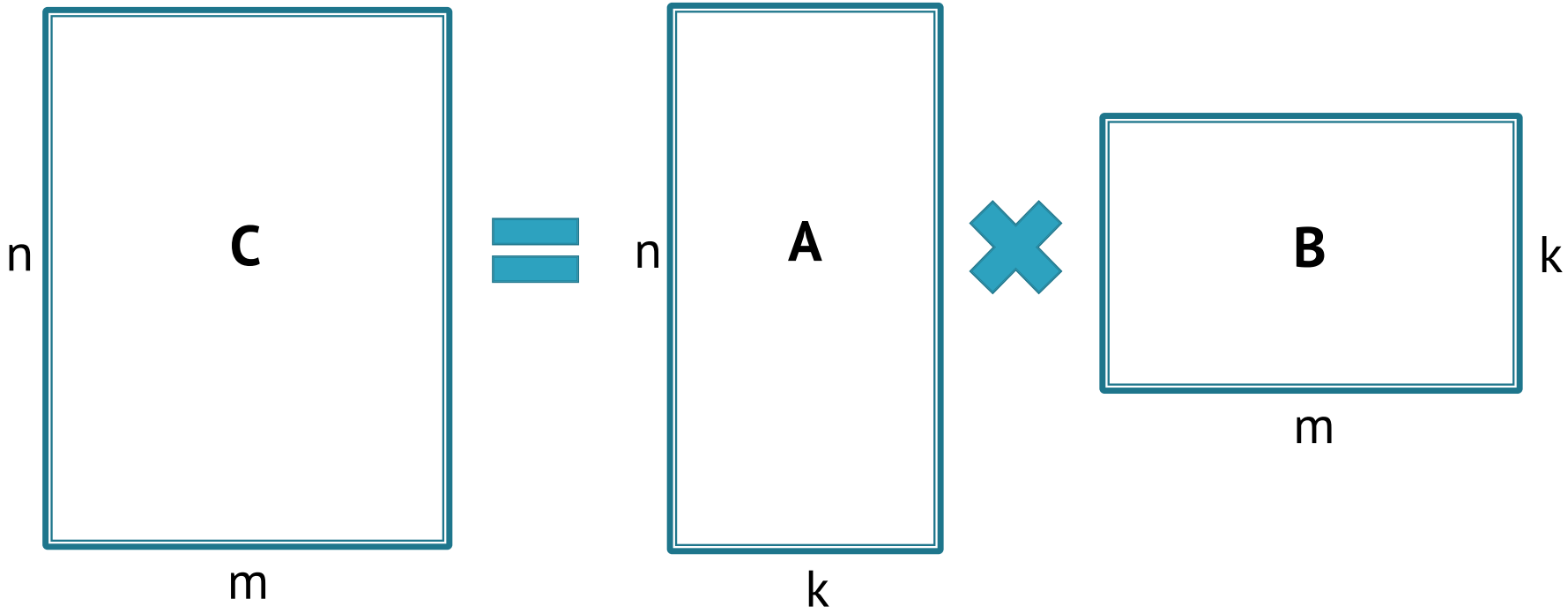


Пример 2

- Перемножение двух матриц – часто используемая операция линейной алгебры

$$c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} * b_{k,j}$$

Пример 2



Пример 2

- Блочный алгоритм – перемножение матриц с помощью перемножения подматриц, с целью уместить данные в кэш
- Транспонирование матрицы b перед умножением, чтобы улучшить паттерн доступа к памяти

Казанский федеральный
УНИВЕРСИТЕТ
ВЫСШАЯ ШКОЛА
информационных технологий
и информационных систем

Вопросы

ekhramch@kpfu.ru