Переразмещение матриц к блочному виду компилятором языка Си с минимизацией использования дополнительной памяти

Семионов Станислав
Михаил Юрушкин

Проблема доступа к данным

- Данные, используемые в программе, находятся “далеко” от процессора.
- Несложно создать процессор с большим количеством ядер (Tile64 с 64 ядрами, 2007 г.), но сложно успевать подавать на этот процессор данные.
- Появление кеш-памяти процессора и блочных программ (Monica Lam, Michael Wolf, 1992 г.)

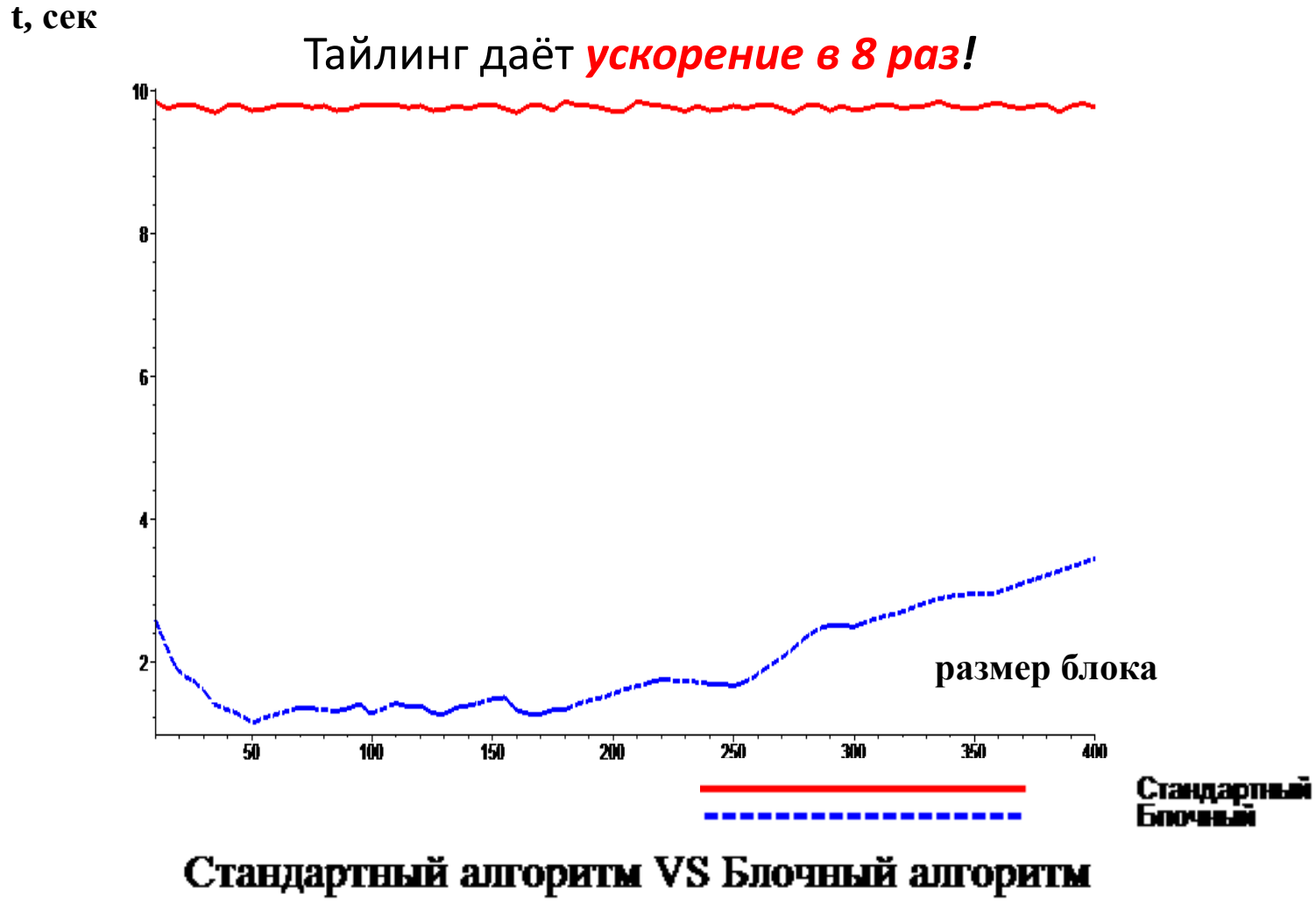
Тайлинг – создание блочных программ

- Разбиение пространства итераций на “тайлы”
- Изменение порядка обхода матрицы
- Уменьшается количество промахов в кеш-память процессора

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Числа означают порядок размещения элементов матрицы (размещение по строкам)

Преимущество тайлинга на примере умножения матриц (N = 2000)



Блочное размещение матрицы в памяти

- Матрица хранится в виде блоков, блоки размещаются стандартно
- В дополнение к тайлингу позволяет добиться дополнительного увеличения эффективности работы с кеш-памятью ЦП, в том числе с TLB-кешем (Буфер Ассоциативной Трансляции)

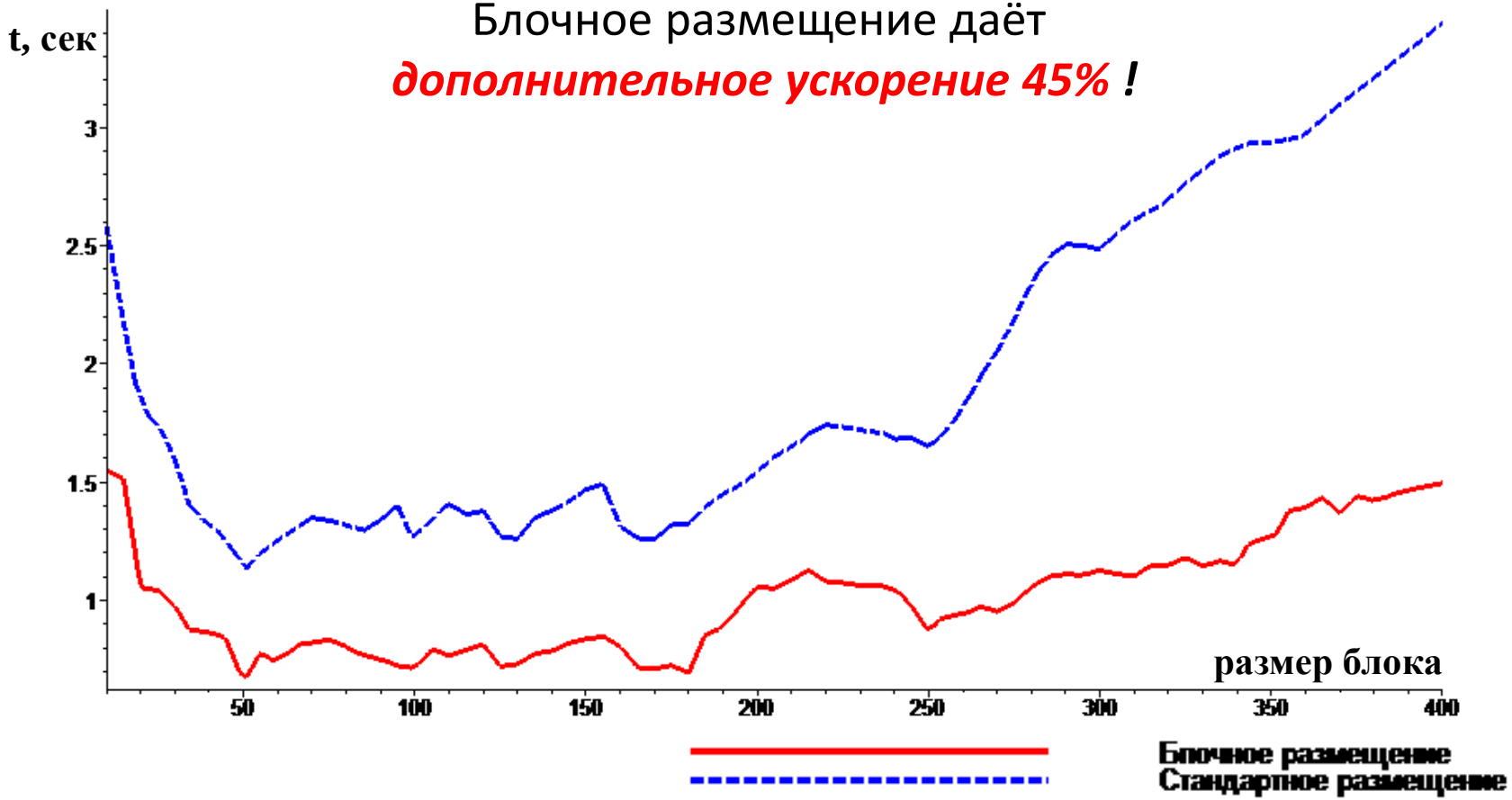
0	1	2	3	16	17	18	19
4	5	6	7	20	21	22	23
8	9	10	11	24	25	26	27
12	13	14	15	28	29	30	31
32	33	34	35	48	49	50	51
36	37	38	39	52	53	54	55
40	41	42	43	56	57	58	59
44	45	46	47	60	61	62	63

Address(X[0,4])=16

Порядок размещения элементов матрицы в памяти по блокам 4 x 4

Преимущество блочного размещения на примере умножения матриц (N = 2000)

Блочное размещение даёт
дополнительное ускорение 45% !



Тайлинг со стандартным размещением VS Тайлинг с блочным размещением

Двойное блочное размещение матрицы в памяти

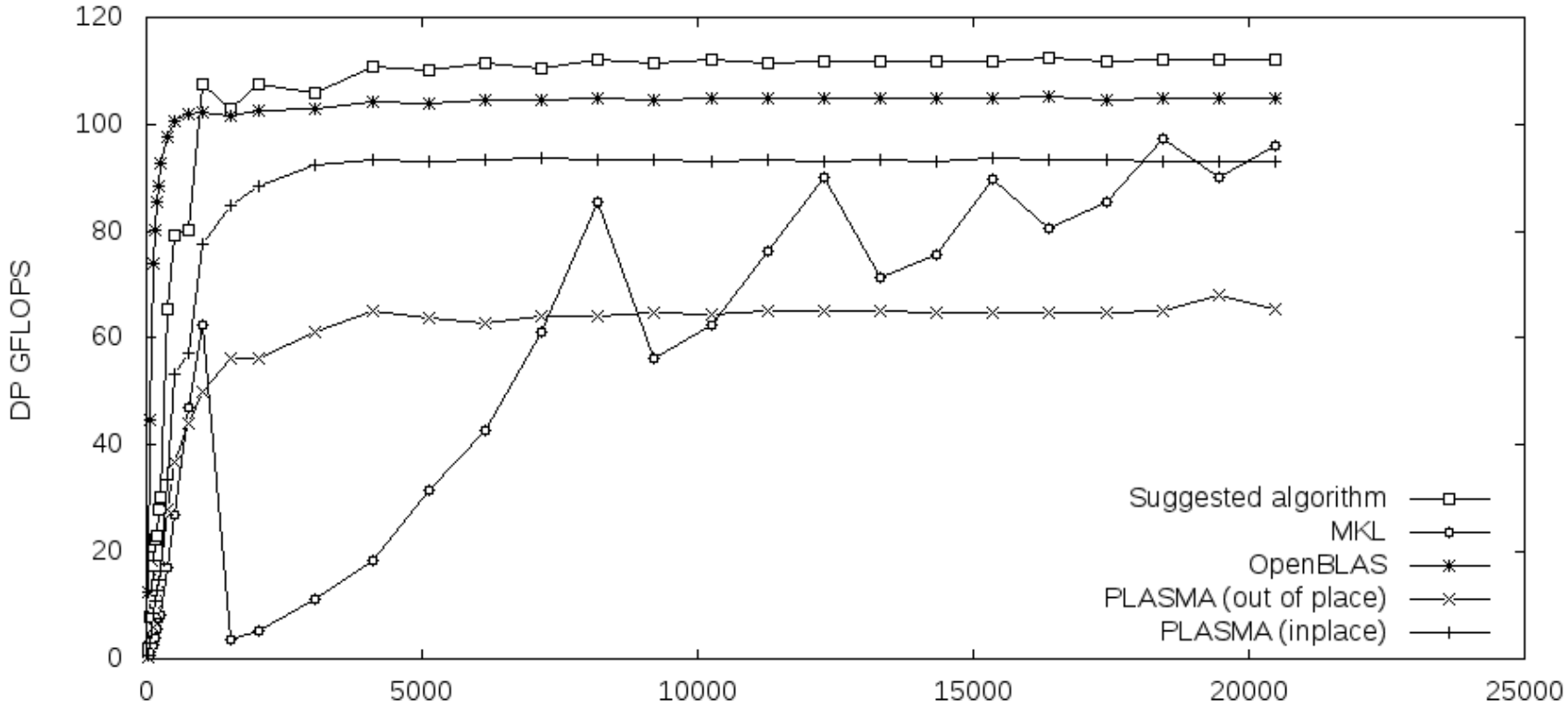
- Два уровня блочности – “большие” блоки хранятся в виде “малых” блоков
- Позволяет оптимизировать использование различных уровней кеш-памяти процессора, а также векторных регистров

0	1	4	5	16	17	20	21
2	3	6	7	18	19	22	23
8	9	12	13	24	25	28	29
10	11	14	15	26	27	30	31
32	33	36	37	48	49	52	53
34	35	38	39	50	51	54	55
40	41	44	45	56	57	60	61
42	43	46	47	58	59	62	63

Порядок размещения элементов матрицы в памяти по блокам 4 x 4 и 2 x 2

Умножение матриц рекордной производительности

Использование двойного блочного размещения позволяет разрабатывать алгоритмы рекордной производительности



Сравнение производительности алгоритма, использующего двойное блочное размещение, а также пакетов MKL, OpenBLAS и PLASMA

Блочное размещение массивов на этапе компиляции в системе OPC

```
#pragma ops array declare(A, N1, N2, d1, d2)
double *A;
...
#pragma ops array allocate(A)
A = malloc(sizeof(double)*N1*N2);
...
#pragma ops array release(A)
free(A);
```

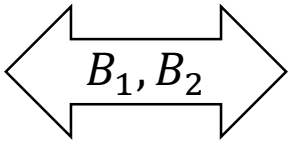
Название алгоритма	Размер матриц	Размер блока	Время работы алгоритма без директив (сек.)	Время работы алгоритма с директивами (сек.)	Ускорение (%)
Двумерное быстрое преобразование Фурье	4096x4096	256x256	17.9	10.54	41%
Блочный алгоритм Флойда	2048x2048	256x256	47.49	41.17	13.3%
Блочное QR-разложение матрицы	2048x1024	256x256	19.6	17.11	12.7%
Блочное LU-разложение матрицы	2048x2048	256x256	27.4	14.44	47%
Блочное умножение квадратных матриц	2048x2048	256x256	14.93	11.2	25%
Блочное возведение матрицы в квадрат	2048x2048	256x256	81.37	17.36	78.6%

Поддержка быстрых блочных переразмещений на этапе выполнения

- ***Требуется способ преобразования размещения матрицы в памяти между стандартным и блочным видами***

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Размещение матрицы по строкам



0	1	4	5	8	9	12	13
2	3	6	7	10	11	14	15
16	17	20	21	24	25	28	29
18	19	22	23	26	27	30	31
32	33	36	37	40	41	44	45
34	35	38	39	42	43	46	47
48	49	52	53	56	57	60	61
50	51	54	55	58	59	62	63

Размещение матрицы по блокам 2 x 2

Способы получения блочного размещения

- Копирование в буфер
  - ✓ Требует $O(N_1 N_2)$ памяти – возможен свопинг
  - ✓ Нерегулярные обращения к памяти

```
double* buf, *source;
//***
for (i = 0; i < N1*N2; i++)
{
    buf[f(i)] = source[i]
}
```

- Копирование каждого блока
  - ✓ Требует $O(B_1 B_2)$ памяти
  - ✓ Избыточные копирования

- ***In-place переразмещение*** – предлагаемый метод
  - ✓ Требует $O(1)$ памяти (в идеале)
  - ✓ Нерегулярные обращения к памяти

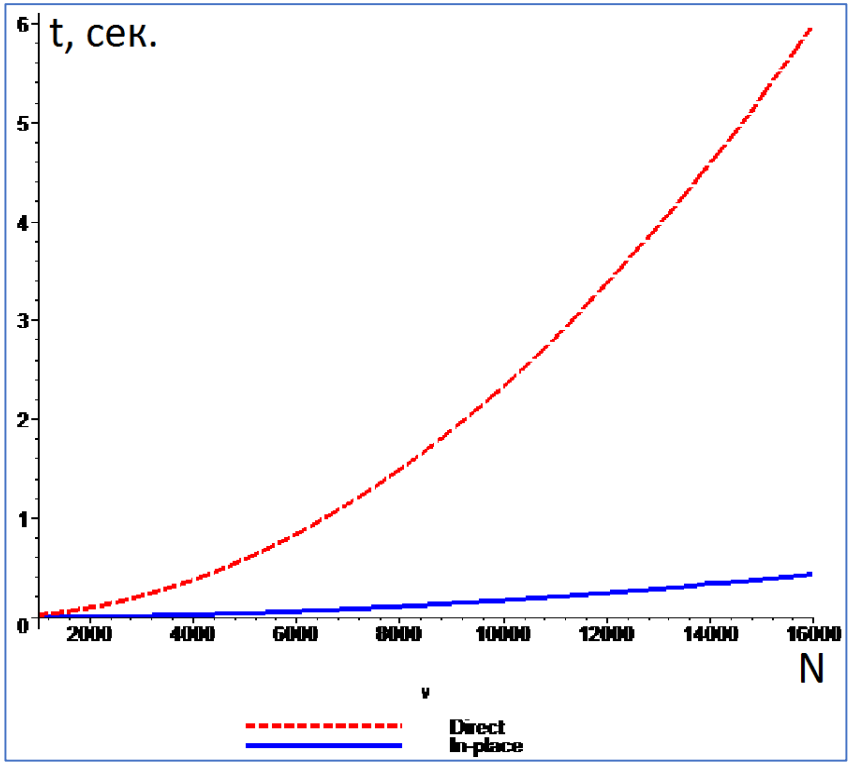
Сложность предлагаемого алгоритма in-place переразмещения матрицы

- Временная:
  - ✓ В среднем $\boldsymbol{O}\left(\boldsymbol{N_1N_2} + \frac{\boldsymbol{B_1N_2}}{\boldsymbol{w}}\boldsymbol{ln}\left(\frac{\boldsymbol{B_1N_2}}{\boldsymbol{w}}\right)\right), w = \text{НОД}(B_2, N_2)$
  - ✓ В худшем $\boldsymbol{O}\left(\boldsymbol{N_1N_2} + \left(\frac{\boldsymbol{B_1N_2}}{\boldsymbol{w}}\right)^{\boldsymbol{2}}\right)$
  - ✓ В лучшем $\boldsymbol{O}(\boldsymbol{N_1N_2})$
- Пространственная:
  - ✓ $\boldsymbol{O}\big(\boldsymbol{max}(\boldsymbol{B_1}, \boldsymbol{B_2})\big)$

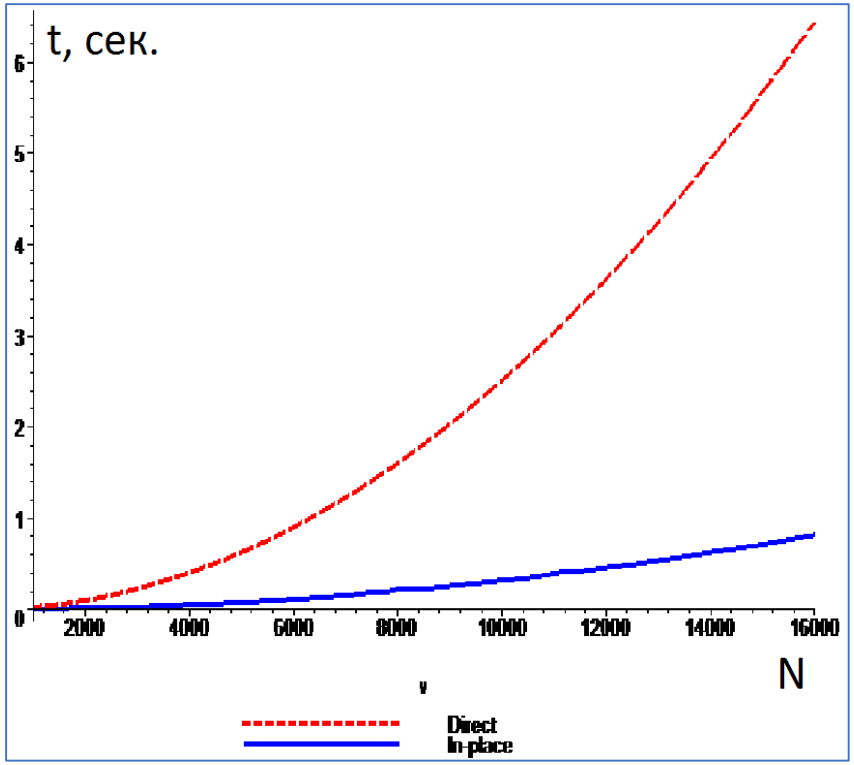
Параметры размещения матрицы:
$\boldsymbol{N_1}(\boldsymbol{N_2})$ - число строк (столбцов) в матрице
$\boldsymbol{B_1}(\boldsymbol{B_2})$ - число строк (столбцов) в блоке

Эффективность предлагаемого алгоритма in-place переразмещения матрицы

Сравнение с ***наивным способом*** перехода к блочному размещению (зависимость времени работы алгоритма от размера матрицы): in-place переразмещение работает ***быстрее на порядок!***



Переход к блочному размещению



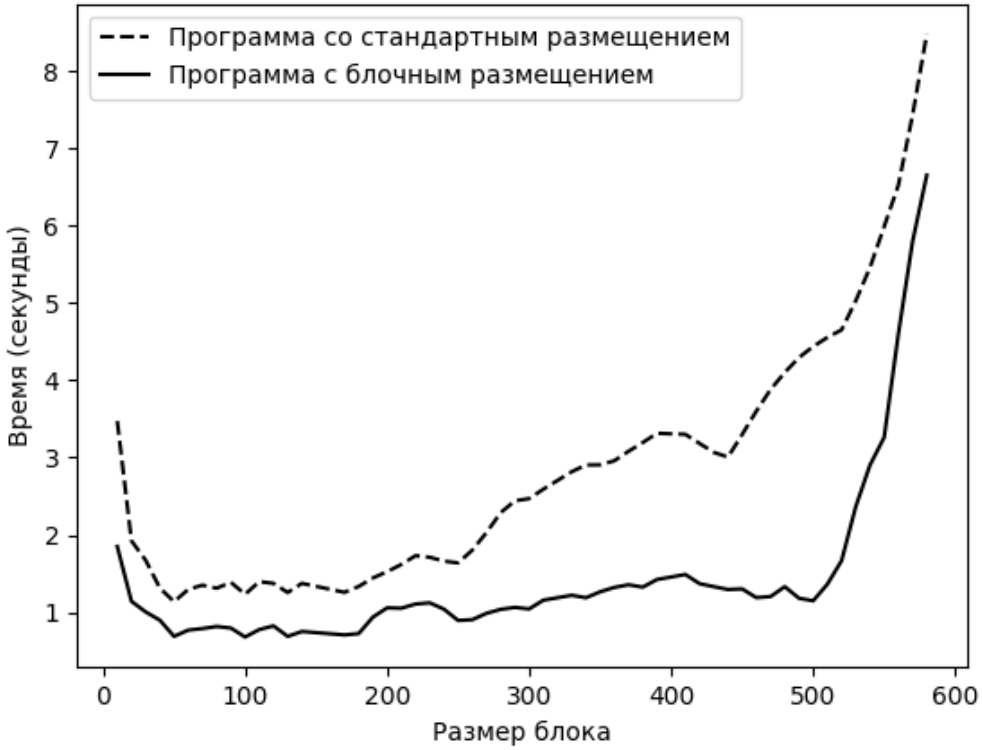
Переход к двойному блочному размещению

Размер блока первого уровня взят 250, блока второго уровня - 50

Применение предлагаемого метода in-place переразмещения в реальных задачах

Сравнение производительности версий ***блочного алгоритма умножения матриц*** со стандартным и с блочным размещениями *с учётом времени на переразмещение*

Версия с блочным размещением и in-place переразмещением
быстрее на 44%,
переразмещение заняло ***2%*** от времени вычислений



Блочное умножение матриц (N = 2000)

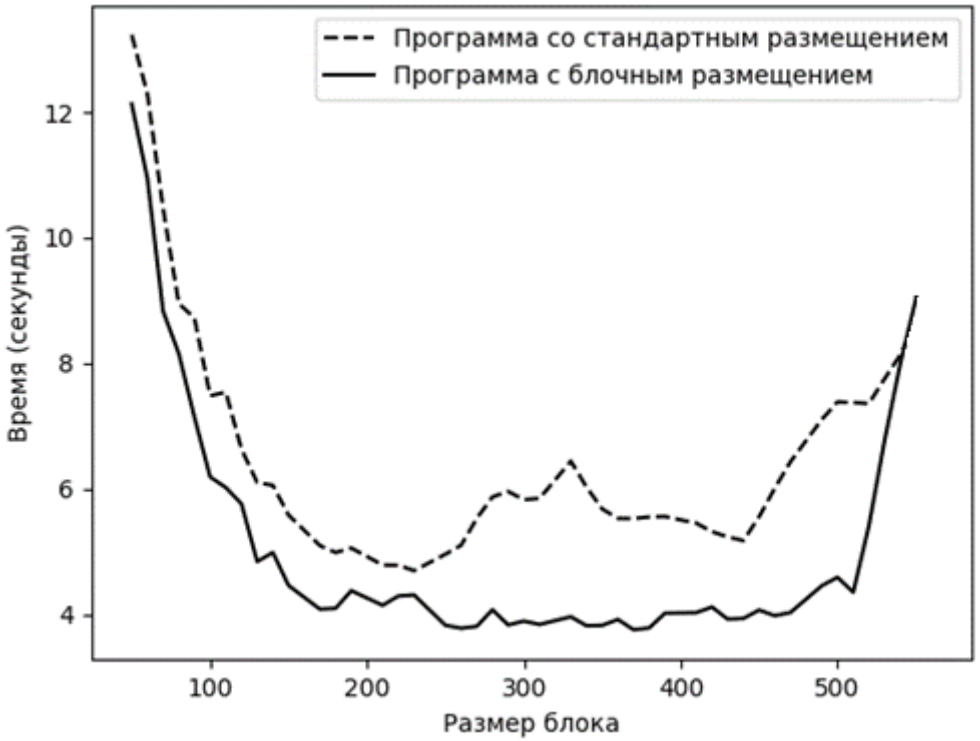
Сравнение производительности версий ***блочного алгоритма умножения матриц*** со стандартным и с двойным блочным размещениями *с учётом времени на переразмещение*

Версия с двойным блочным размещением и in-place переразмещением ***быстрее на 33%,***
переразмещение заняло ***4%*** от времени вычислений

Двойное блочное умножение матриц (N = 2000, B = 100)

Сравнение производительности версий ***блочного алгоритма QR разложения матрицы*** со стандартным и с блочным размещениями *с учётом времени на переразмещение*

Версия с блочным размещением и in-place переразмещением ***быстрее на 19%,***
переразмещение заняло ***0.5%*** от времени вычислений



Блочное QR-разложение (N = 2000)

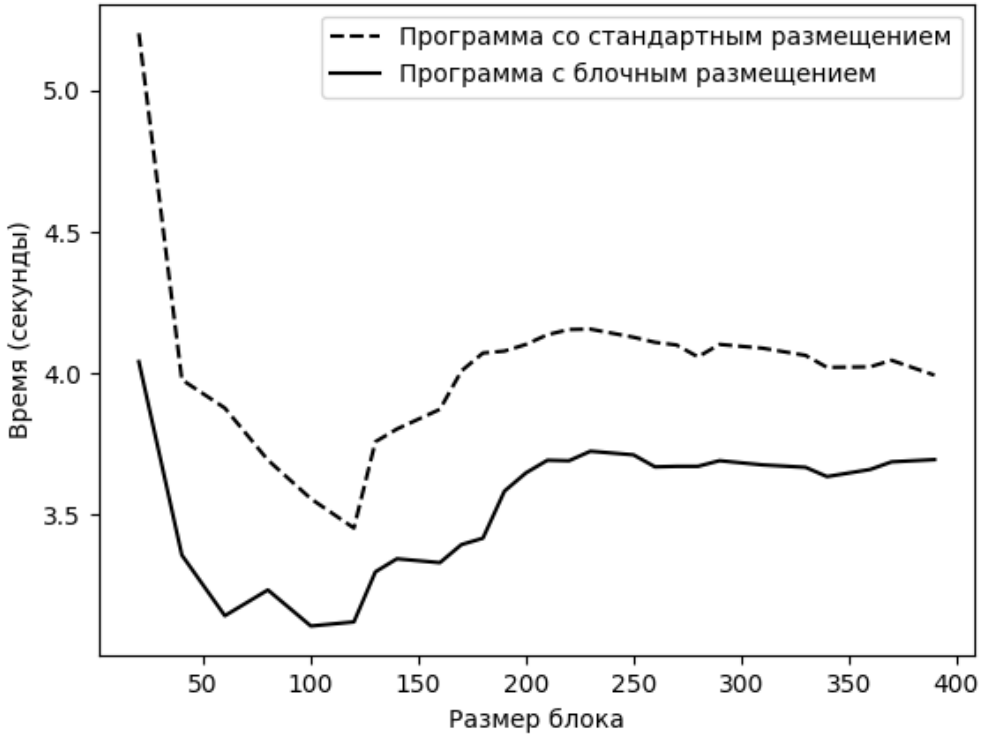
Сравнение производительности версий ***блочного алгоритма QR разложения матрицы*** со стандартным и с двойным блочным размещениями *с учётом времени на переразмещение*

Версия с двойным блочным размещением и in-place переразмещением ***быстрее на 15%,***
переразмещение заняло ***1%*** от времени вычислений

Двойное блочное QR-разложение (N = 2000, B = 475)

Сравнение производительности версий ***блочного алгоритма Флойда*** со стандартным и с блочным размещениями *с учётом времени на переразмещение*

Версия с блочным размещением и in-place переразмещением ***быстрее на 9%,***
переразмещение заняло ***0.3%*** от времени вычислений



Блочный алгоритм Флойда (N = 2000)

Интеграция в ОРС

- Проект реализован в виде runtime-библиотеки компилятора языка Си Оптимизирующей Распараллеливающей Системы
- В целях упрощения использования планируется реализация поддержки переразмещений в виде директив компилятора

```
void some_function(double* A, int n1, int n2)
{
   int b1 = …, b2 = …;
   /*
      Работа со стандартно размещённой матрицей …
   */
   // Переразмещение к блочному виду
# pragma ops reallocateSTtoB(A, n1, n2, b1, b2)
   /*
      Вычисления с блочной матрицей …
   */
   // Переразмещение обратно к стандартному виду
# pragma ops reallocateBtoST(A, n1, n2, b1, b2)
   /*
      Работа со стандартно размещённой матрицей …
   */
}
```

Спасибо за внимание!

Дополнительные слайды

Получение двойного блочного размещения*

Используя предлагаемый метод можно получить двойное блочное размещение. Для этого нужно:

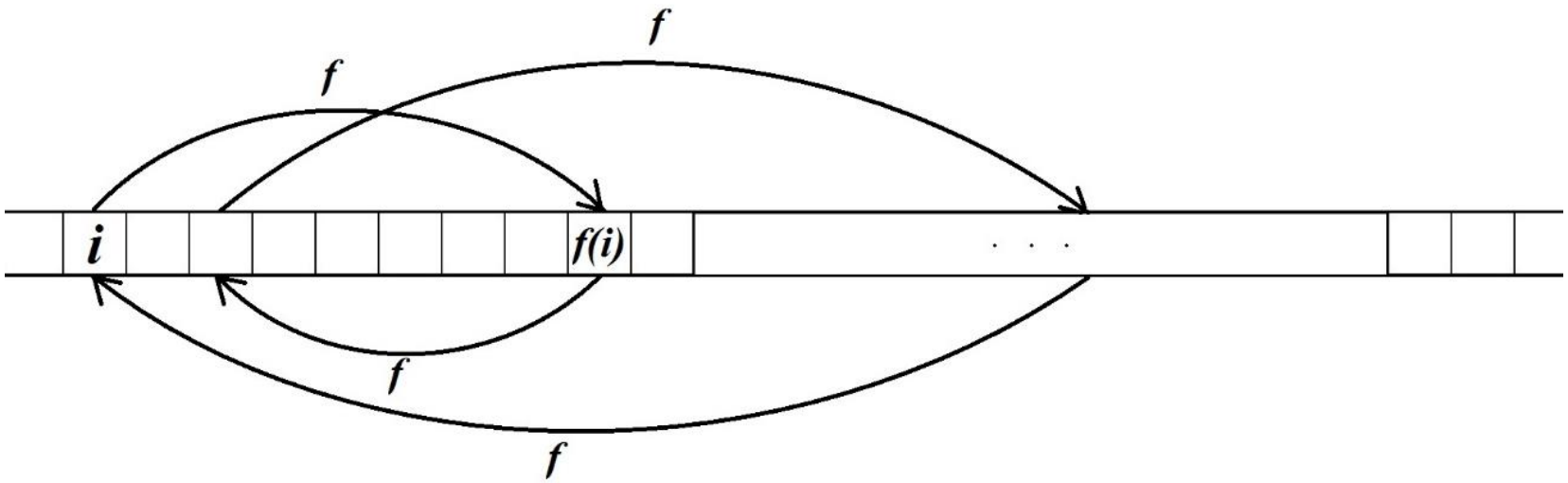
- Выполнить блочное переразмещение с параметрами N_1, N_2, B_1, B_2
- Для каждого получившегося блока выполнить блочное переразмещение с параметрами B_1, B_2, D_1, D_2

0	1	4	5	16	17	20	21
2	3	6	7	18	19	22	23
8	9	12	13	24	25	28	29
10	11	14	15	26	27	30	31
32	33	36	37	48	49	52	53
34	35	38	39	50	51	54	55
40	41	44	45	56	57	60	61
42	43	46	47	58	59	62	63

Алгоритм in-place переразмещения*

Основан на обходе циклов перестановки, порождённой отображением f из строчного в блочное размещение:

$$i \to f(i) \to f\big(f(i)\big) \to \cdots \to f(\ldots f(i) \ldots) \to i$$

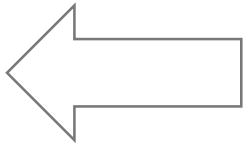


Возврат к исходному размещению*

- Нужно сохранить список порождающих адресов циклов
- Используя его, выполнить процедуру, аналогичную прямому переразмещению, но вместо $f(i)$ использовать $f^{-1}(i)$

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Размещение матрицы по строкам



0	1	4	5	8	9	12	13
2	3	6	7	10	11	14	15
16	17	20	21	24	25	28	29
18	19	22	23	26	27	30	31
32	33	36	37	40	41	44	45
34	35	38	39	42	43	46	47
48	49	52	53	56	57	60	61
50	51	54	55	58	59	62	63

Размещение матрицы по блокам 2 x 2

Поиск циклов*

- Циклов обычно более одного → имеется задача отличения одного цикла от другого
- Она решается нахождением порождающих адресов для всех циклов