



# **НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. Н.И. ЛОБАЧЕВСКОГО**

## **ЦЕНТР КОМПЕТЕНЦИЙ ONEAPI В ННГУ**

**ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МАТЕМАТИКИ И МЕХАНИКИ**



**НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. Н.И. ЛОБАЧЕВСКОГО**  
**ЦЕНТР КОМПЕТЕНЦИЙ oneAPI в ННГУ**  
**ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МАТЕМАТИКИ И МЕХАНИКИ**

***ВВЕДЕНИЕ В АНАЛИЗ ПРОИЗВОДИТЕЛЬНОСТИ  
И ОПТИМИЗАЦИЮ ПРОГРАММ***

**Рациональное использование  
иерархии памяти**

# Содержание

---

- ❑ Цель работы
- ❑ Тестовая задача
- ❑ Пошаговая оптимизация
- ❑ Самостоятельная работа

# ЦЕЛЬ РАБОТЫ

# Цель работы

---

- Научиться использовать инструментарий для повышения производительности при работе с подсистемой памяти в процессе пошаговой оптимизации кода

## Задачи:

- Рассмотреть основные возможности Intel® VTune
- На примере тестовой задачи выполнить оптимизацию кода с использованием инструментария

# ТЕСТОВАЯ ЗАДАЧА

# Тестовая задача и метод решения

- Вычисляется результат транспонирования матрицы в саму себя (inplace):

$$A = A^T$$

где  $A \in \mathbb{R}^{N \times N}$

- Параметры тестовой задачи:

- $N = 8192, 16384, 32768$
- $A$  – случайные матрицы
- Измеряется время одного транспонирования матрицы

# Код программы

```
#include <chrono>
#include <iostream>
#include <random>

#include "mkl.h"
#include <mkl_trans.h>

void transpose_baseline(double* mat, int n) {
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            std::swap(mat[i * n + j], mat[j * n + i]);
}

void transpose_baseline_omp(double* mat, int n) {
#pragma omp parallel for
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            std::swap(mat[i * n + j], mat[j * n + i]);
}

void transpose_mkl(double* mat, int n) {
    mkl_dimatcopy('R', 'T', N, N, 1.0, mat, N, N);
}
```

```
void run_clock(void (*fun)(double*, int), double* mat, int n,
const char* name) {
    auto start = std::chrono::steady_clock::now();
    fun(mat, n);
    auto end = std::chrono::steady_clock::now();
    std::chrono::duration<double> elapsed_sec = end - start;
    std::cout << name << " elapsed time: " << elapsed_sec.count()
<< " s\n";
}

void ccheck(double * mat_t, double * mat, int n) {
    mkl_dimatcopy('R', 'T', N, N, 1.0, mat, N, N);
    int flag = 0;
    for (int i = 0; i < N * N; i++)
        if (mat[i] != mat_t[i])
            flag = 1;
    if (flag)
        std::cout << "Incorrect" << std::endl;
    else
        std::cout << "Correct" << std::endl;
}
```



# Код программы (2)

```
int main(int argc, char*argv[]) {
    unsigned int N = 1024;
    if (argc > 1)
        N = atoi(argv[1]);
    std::cout << "Matrix size: " << N << std::endl;
    double* matrix = new double[N * N];
    double* matrix_trans = new double[N * N];
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_real_distribution<> dis(-2.0, 2.0);

    for (int i = 0; i < N * N; i++)
        matrix_trans[i] = matrix[i] = dis(gen);

    int mode = 0;
    if (argc > 2)
        mode = atoi(argv[2]);

    if (mode == 0)
        run_clock(transpose_baseline, matrix_trans, N, "Baseline");
    if (mode == 1)
        run_clock(transpose_baseline_omp, matrix_trans, N, "BaselineOMP");
    if (mode == 10)
        run_clock(transpose_mkl, matrix_trans, N, "MKL");
    if (argc > 3)
        check(matrix_trans, matrix, N);
}
```

# Вычислительная инфраструктура

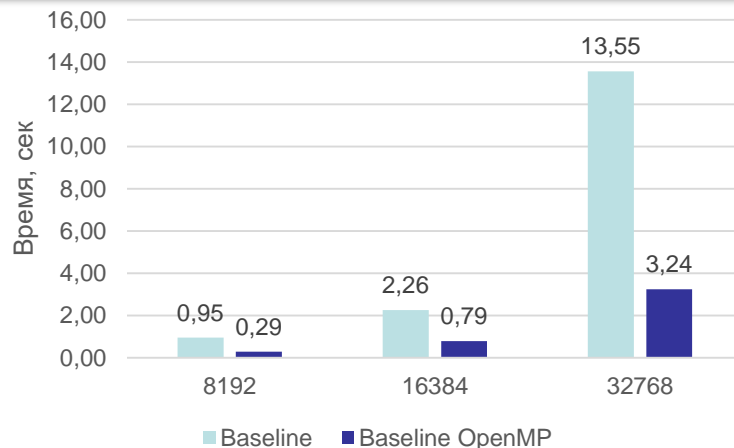
Процессоры	Intel® Xeon Silver 4310T CPU @ 2.30GHz
Число процессоров	2
Общее число ядер	20
Память	64 Gb
Операционная система	Linux CentOS 7
Компилятор, профилировщик, отладчик	Intel® oneAPI 2023.0 (Intel® C++/DPC++ Compiler, Intel® VTune)

# ПОШАГОВАЯ ОПТИМИЗАЦИЯ

# Базовая версия

- ❑ В качестве основной версии используется наивная параллельная реализация
- ❑ Рассматриваемые далее оптимизации последовательной и параллельной версий в данной задаче одинаковые, но в других задачах оптимизации могут быть различными и надо начинать с соответствующей реализации
- ❑ Для проверки корректности происходит сравнение результатов с аналогичной функцией из библиотеки Intel® MKL

```
void transpose_baseline(double* mat, int n) {  
    for (int i = 0; i < n; i++)  
        for (int j = i + 1; j < n; j++)  
            std::swap(mat[i * n + j], mat[j * n + i]);  
}  
  
void transpose_baseline_omp(double* mat, int n) {  
    #pragma omp parallel for  
    for (int i = 0; i < n; i++)  
        for (int j = i + 1; j < n; j++)  
            std::swap(mat[i * n + j], mat[j * n + i]);  
}
```



# Анализ производительности (базовая версия)

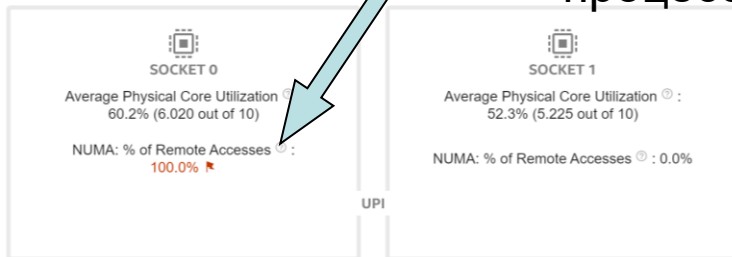
Memory Access Memory Usage  
Analysis Configuration Collection Log Summary Bottom-up Platform

✓ Elapsed Time<sup>Ⓢ</sup>: 17.116s

CPU Time <sup>Ⓢ</sup> :	53.796s	
Memory Bound <sup>Ⓢ</sup> :	N/A*	of Pipeline Slots
L1 Bound <sup>Ⓢ</sup> :	0.0%	of Clockticks
L2 Bound <sup>Ⓢ</sup> :	0.0%	of Clockticks
L3 Bound <sup>Ⓢ</sup> :	8.2%	of Clockticks
DRAM Bound <sup>Ⓢ</sup> :	82.9%	of Clockticks
Store Bound <sup>Ⓢ</sup> :	10.3%	of Clockticks
NUMA: % of Remote Accesses <sup>Ⓢ</sup> :	30.3%	
Loads:	20,226,074,567	
Stores:	12,361,803,679	
LLC Miss Count <sup>Ⓢ</sup> :	332,808,850	
Local DRAM Access Count <sup>Ⓢ</sup> :	57,043,904	
Remote DRAM Access Count <sup>Ⓢ</sup> :	13,521,092	
Remote Cache Access Count <sup>Ⓢ</sup> :	0	
Average Latency (cycles) <sup>Ⓢ</sup> :	175	
Total Thread Count:	46	
Paused Time <sup>Ⓢ</sup> :	0s	

\*N/A is applied to metrics with undefined value. There is no data to calculate the metric.

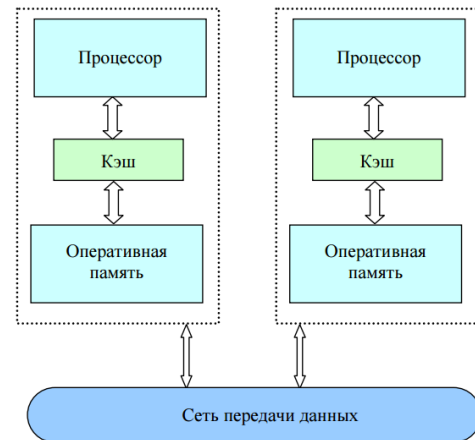
✓ Platform Diagram



- В некоторых системах Intel® VTune может собирать не все сведения из-за отсутствия специального драйвера (N/A)

Большое количество удаленного доступа

Все данные находятся в памяти одного процессора из двух



# NUMA-friendly оптимизация

- В случае проблем с NUMA-доступом есть несколько основных способов решения:

- Использовать first-touch policy:

Физически память выделяется при первом обращении к ней, при этом память выделяется в памяти текущего процессора

- При многократном использовании одной и той же памяти с другого процессора система автоматически перенесет ее в другую часть оперативной памяти

- Использовать ресурсы одного процессора: numactl -H

Для запуска программы:

numactl --physcpubind=0-9,20-29 ./MyProg

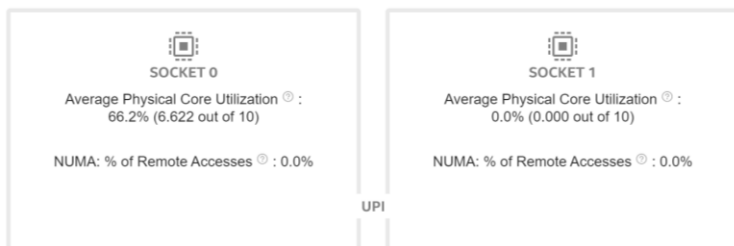
```
numactl -H
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9
20 21 22 23 24 25 26 27 28 29
node 0 size: 31879 MB
node 0 free: 19150 MB
node 1 cpus: 10 11 12 13 14 15 16
17 18 19 30 31 32 33 34 35 36 37
38 39
node 1 size: 32236 MB
node 1 free: 17297 MB
node distances:
node  0  1
 0:  10  20
 1:  20  10
```

# Анализ производительности (NUMA)

Memory Access		
Memory Usage		
Analysis Configuration Collection Log Summary Bottom-up Platform		
Elapsed Time: 15.235s		
CPU Time: 31.749s		
Memory Bound:	N/A*	of Pipeline Slots
L1 Bound:	0.8%	of Clockticks
L2 Bound:	0.0%	of Clockticks
L3 Bound:	7.1%	of Clockticks
DRAM Bound:	77.4%	of Clockticks
Memory Bandwidth:	42.5%	of Clockticks
Store Bound:	13.8%	of Clockticks
NUMA: % of Remote Accesses:	0.0%	
Loads:	20,260,182,778	
Stores:	12,482,901,509	
LLC Miss Count:	599,266,558	
Local DRAM Access Count:	410,697,064	
Remote DRAM Access Count:	0	
Remote Cache Access Count:	0	
Average Latency (cycles):	97	
Total Thread Count:	30	
Paused Time:	0s	

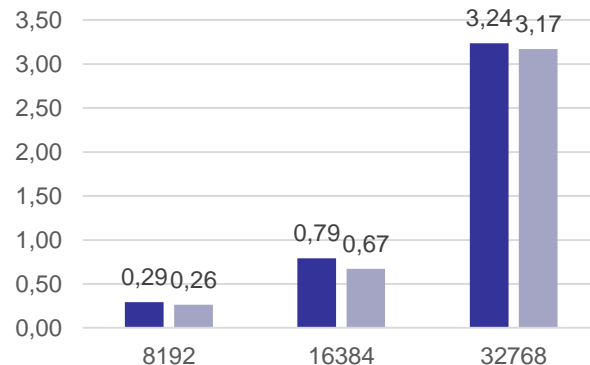
\*N/A is applied to metrics with undefined value. There is no data to calculate the metric.

## Platform Diagram



□ Полностью исчез удаленный доступ

□ Время почти не изменилось, зато задействованные вычислительные ресурсы сократились в 2 раза.  
**Почему?**



■ Baseline OpenMP ■ Baseline OpenMP + NUMA

# Анализ производительности (NUMA) (2)

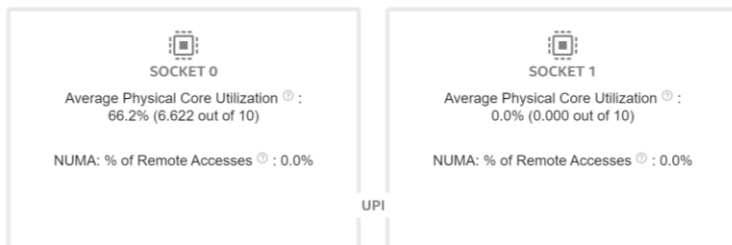
Memory Access Memory Usage  
Analysis Configuration Collection Log Summary Bottom-up Platform

Elapsed Time: 15.235s

CPU Time	31.749s	
Memory Bound	N/A*	of Pipeline Slots
L1 Bound	0.8%	of Clockticks
L2 Bound	0.0%	of Clockticks
L3 Bound	7.1%	of Clockticks
DRAM Bound	77.4%	of Clockticks
Memory Bandwidth	42.5%	of Clockticks
Store Bound	13.8%	of Clockticks
NUMA: % of Remote Accesses	0.0%	
Loads	20,260,182,778	
Stores	12,482,901,509	
LLC Miss Count	599,266,558	
Local DRAM Access Count	410,697,064	
Remote DRAM Access Count	0	
Remote Cache Access Count	0	
Average Latency (cycles)	97	
Total Thread Count	30	
Paused Time	0s	

\*N/A is applied to metrics with undefined value. There is no data to calculate the metric.

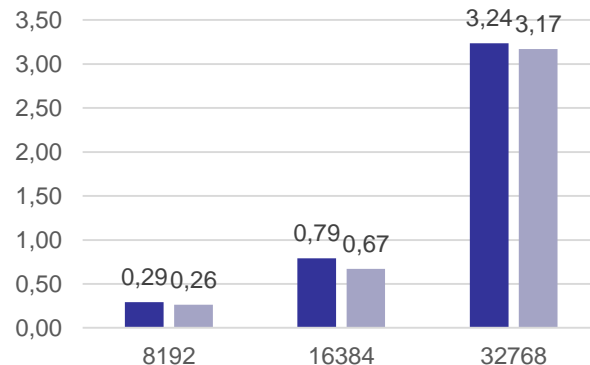
Platform Diagram



□ Полностью исчез удаленный доступ

□ Время почти не изменилось, зато задействованные вычислительные ресурсы сократились в 2 раза.

Конвейер  
останавливается из-за  
подкачки данных из  
DRAM



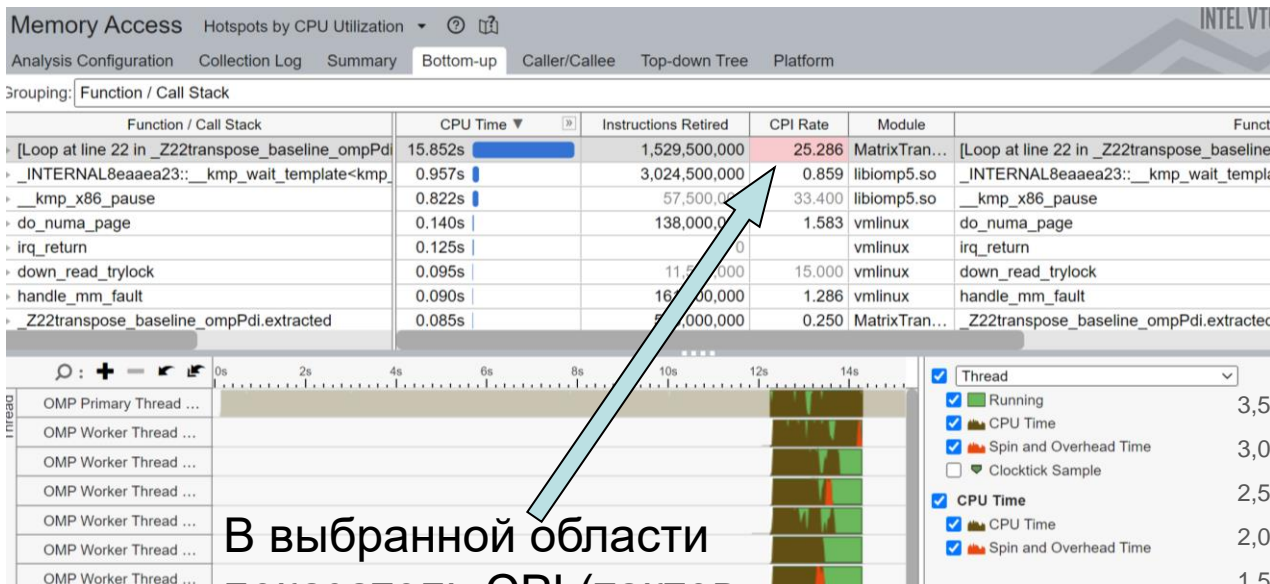
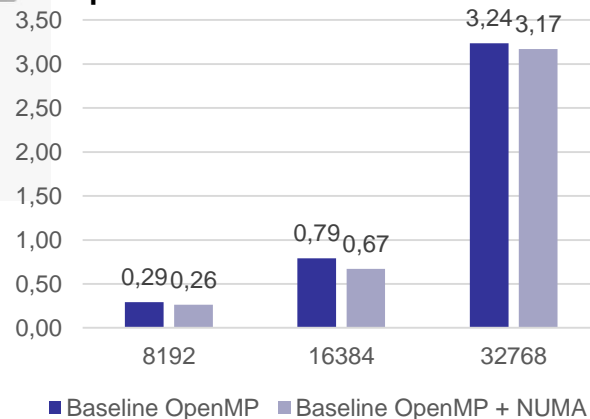
■ Baseline OpenMP ■ Baseline OpenMP + NUMA



# Анализ производительности (NUMA) (3)

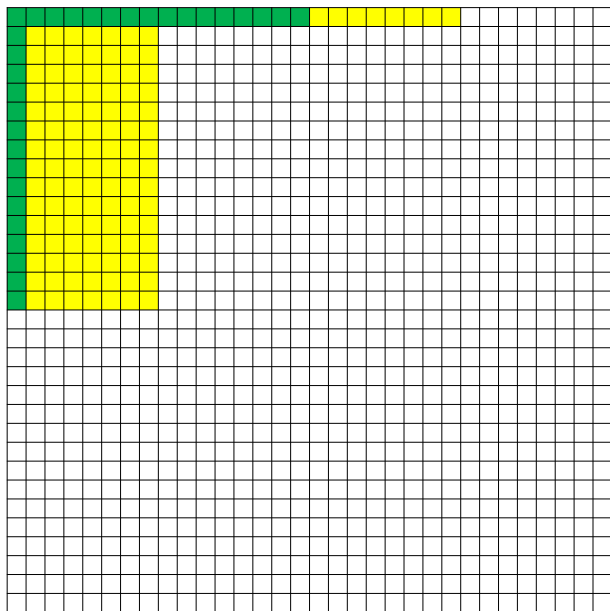
В рамках задачи происходит большая подготовительная работа, для ее исключения в Intel® VTune можно использовать фильтр по временной шкале

В выбранной области показатель CPI (тактов на инструкцию) имеет огромное значение, что часто свидетельствует о том, что «узкое горлышко» программы – работа с памятью

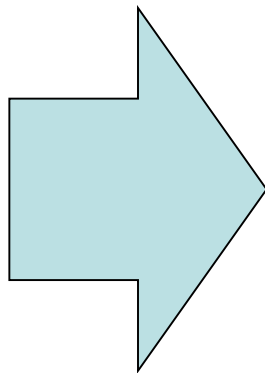
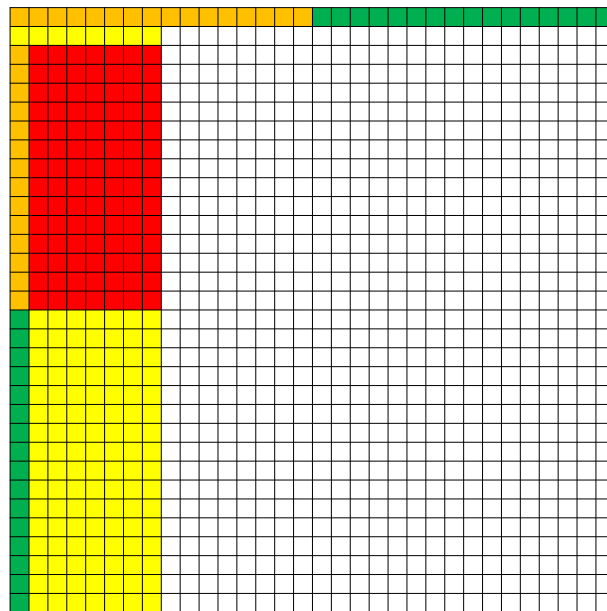


# Базовая версия

I)



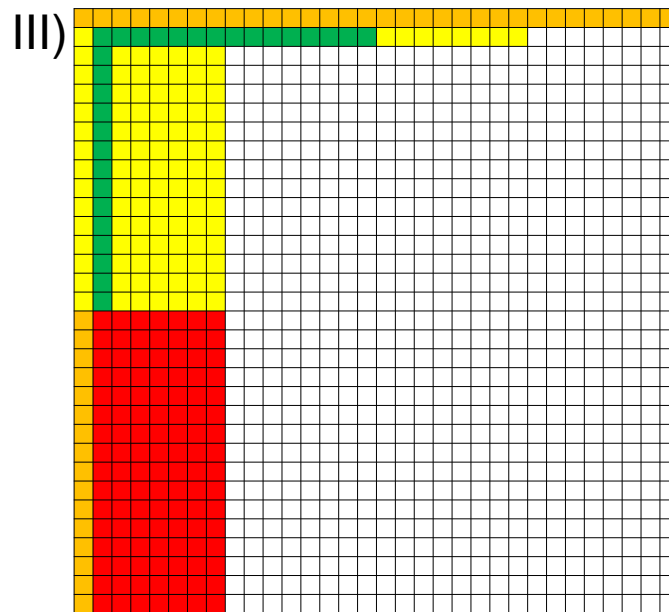
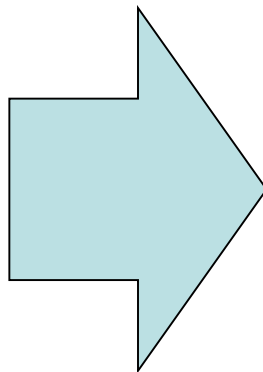
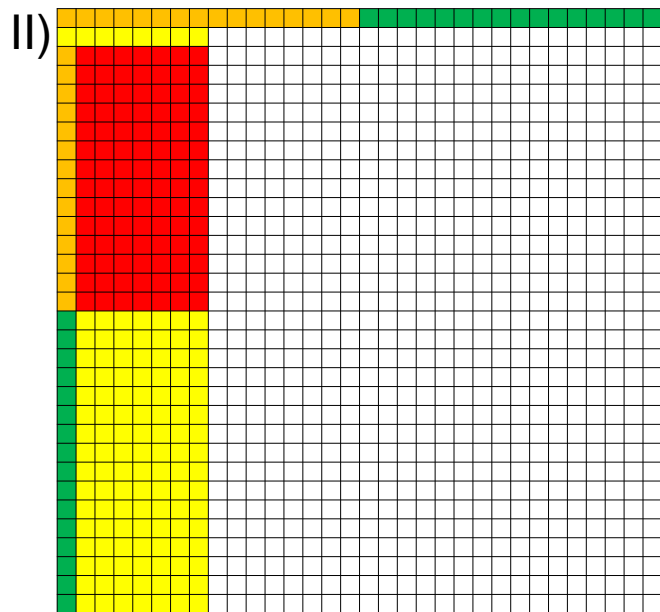
II)



■ - загружено в кэш и было использовано  
■ - было вытеснено из кэша использованным

■ - загружено в кэш  
■ - было вытеснено из кэша неиспользованным

# Базовая версия



■ - загружено в кэш и было использовано  
■ - было вытеснено из кэша использованным

■ - загружено в кэш

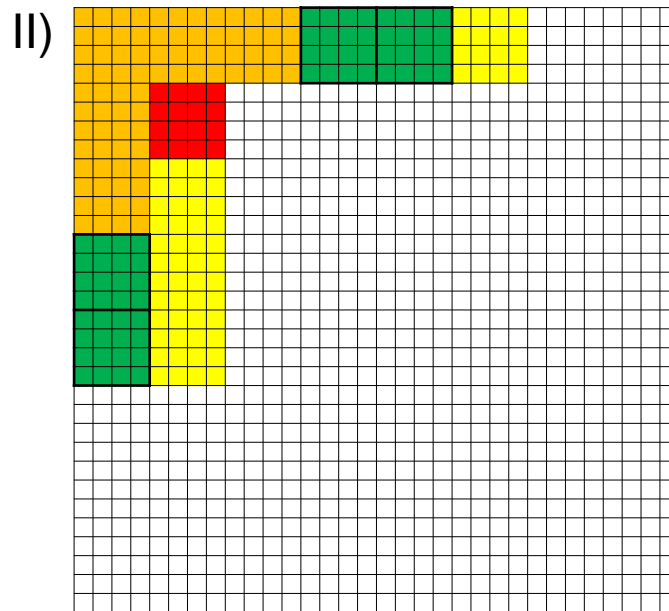
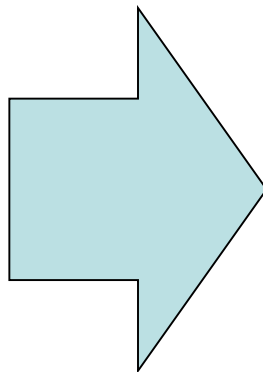
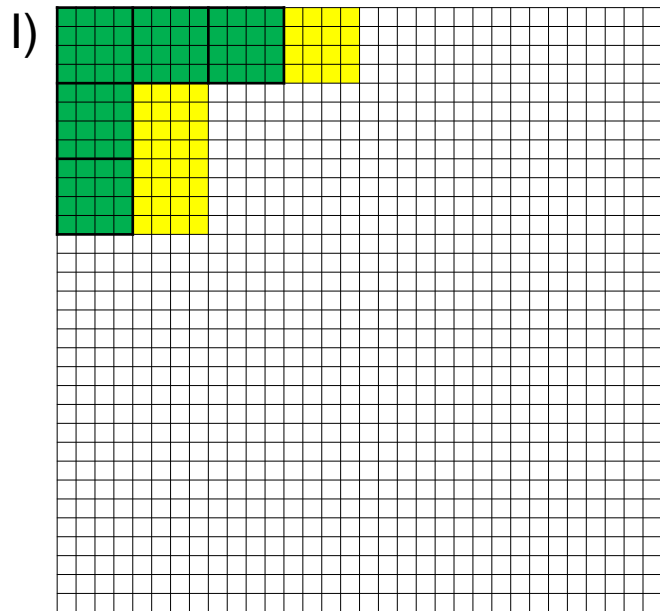
■ - было вытеснено из кэша  
неиспользованным

□ Проблема загрузки в кэш большого количества неиспользуемой памяти

# Cache-friendly оптимизация

- ❑ Проблема загрузки в кэш большого количества неиспользуемой памяти
- ❑ Базовая реализация постоянно загружает память из DRAM (~77% остановки конвейера)
- ❑ В базовой версии происходит много вытеснений еще необходимой памяти из кэша
- ❑ Основная идея блочной схемы в обходе данных таким способом, чтобы уменьшить излишние обращения в DRAM (или кэш высокого уровня) и максимально использовать загруженную в кэш память

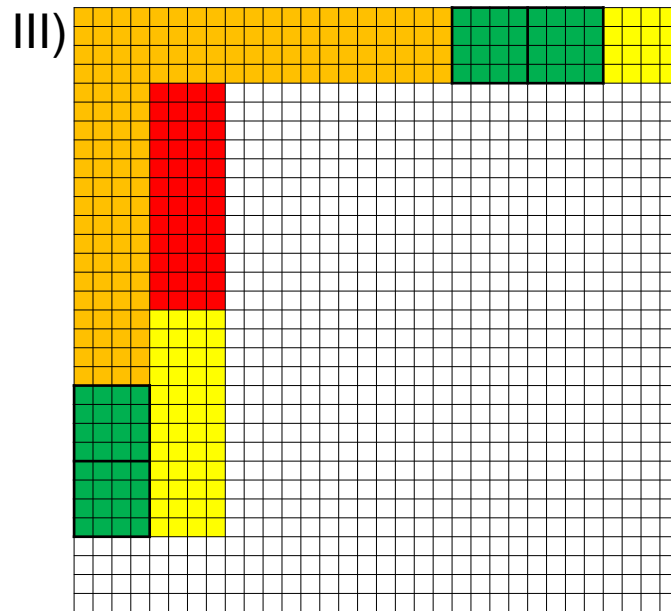
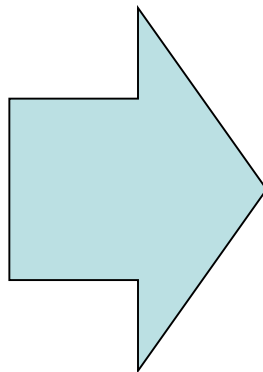
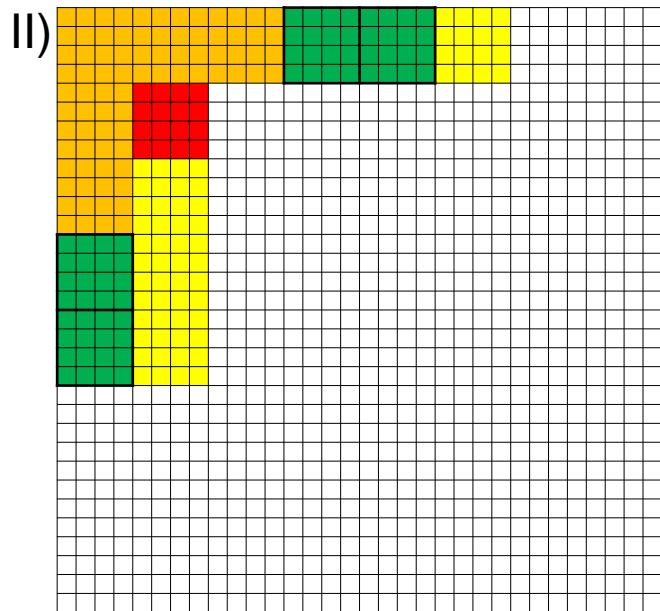
# Блочная версия



■ - загружено в кэш и было использовано  
■ - было вытеснено из кэша использованным

■ - загружено в кэш ■ - было вытеснено из кэша  
■ - было вытеснено из кэша неиспользованным

# Блочная версия



■ - загружено в кэш и было использовано  
■ - было вытеснено из кэша использованным

■ - загружено в кэш

■ - было вытеснено из кэша  
■ - неиспользованным

□ Неиспользуемая память вытесняется меньше, почти все переиспользуется

# Модификация кода

```
void transpose_baseline(double* mat, int n) {
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            std::swap(mat[i * n + j], mat[j * n + i]);
}

void transpose_baseline_omp(double* mat, int n) {
    #pragma omp parallel for
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            std::swap(mat[i * n + j], mat[j * n + i]);
}
```

```
void transpose_block_omp(double* mat, int n) {
    const int bl_size = 64;
    #pragma omp parallel for
    for (int bi = 0; bi < n / bl_size; bi++) {
        int block_start = bi * bl_size;
        for (int i = block_start; i < block_start + bl_size; i++)
            for (int j = i + 1; j < block_start + bl_size; j++)
                std::swap(mat[i * n + j], mat[j * n + i]);
        int bj = block_start + bl_size;
        for (; bj + bl_size < n; bj += bl_size)
            for (int i = block_start; i < block_start + bl_size; i++)
                for (int j = bj; j < bj + bl_size; j++)
                    std::swap(mat[i * n + j], mat[j * n + i]);

        for (int i = block_start; i < block_start + bl_size; i++)
            for (int j = bj; j < n; j++)
                std::swap(mat[i * n + j], mat[j * n + i]);
    }
    int block_index = n / bl_size;
    for (int i = block_index * bl_size; i < n; i++)
        for (int j = i + 1; j < n; j++)
            std::swap(mat[i * n + j], mat[j * n + i]);
}
```

# Анализ производительности (блочная версия)

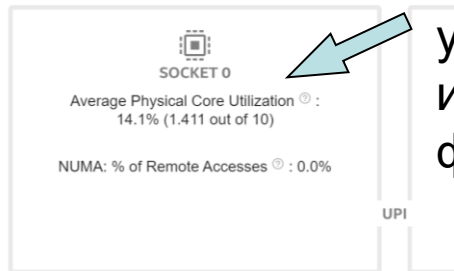
Memory Access Memory Usage  
Analysis Configuration Collection Log Summary Bottom-up Platform

Elapsed Time: 14.224s

CPU Time	22.578s	
Memory Bound	N/A*	of Pipeline Slots
L1 Bound	5.7%	of Clockticks
L2 Bound	5.7%	of Clockticks
L3 Bound	4.4%	of Clockticks
DRAM Bound	41.3%	of Clockticks
Memory Bandwidth	23.5%	of Clockticks
Store Bound	20.1%	of Clockticks
NUMA: % of Remote Accesses	0.0%	
Loads	21,047,282,203	
Stores	13,181,816,983	
LLC Miss Count	102,411,798	
Average Latency (cycles)	36	
Total Thread Count	28	
Paused Time	0s	

\*N/A is applied to metrics with undefined value. There is no data to calculate the metric.

Platform Diagram



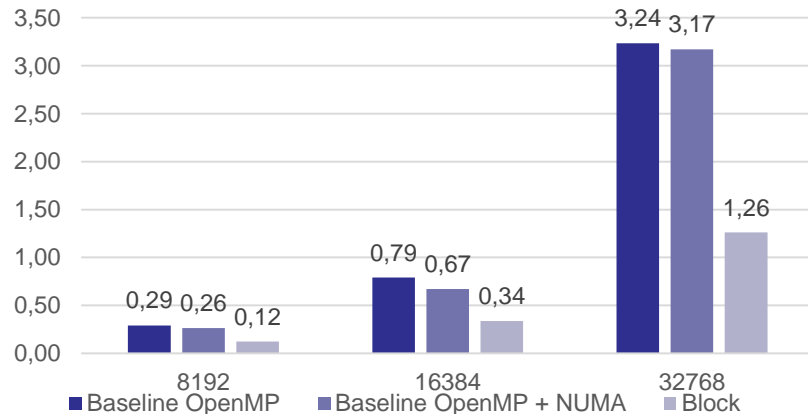
Улучшенный доступ памяти

Сократилась общая проблема с DRAM (было ~77%)

Сократилось кол-во промахов LLC (было 600 млн), как и средняя латентность (было 97).

Но все равно осталось большое количество промахов LLC

Сильно уменьшилось использование физических ядер





# Анализ производительности (блочная версия) (2)

Доступ к памяти с фильтром только по функции транспонирования

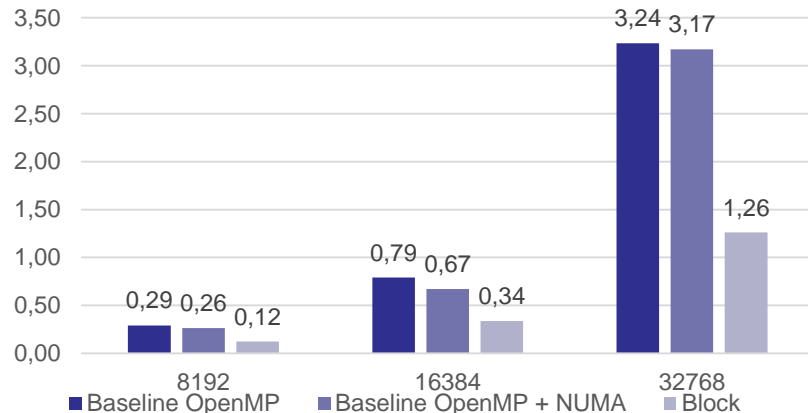
Module / Function / Call Stack	CPU Time ▼	Memory Bound					Loads	Stores	LLC Miss Count	Average Latency (cycles)
		L1 Bound	L2 Bound	L3 Bound	DRAM Bound	Store Bound				
MatrixTranspose	6.621s	0.0%	29.3%	10.8%	100.0%	9.4%	2,796,884,247	1,843,867,589	94,893,224	151
▶ [Loop at line 6	6.420s	0.0%	29.1%	12.1%	100.0%	7.7%	2,714,270,572	1,813,557,939	94,893,224	155
▶ _Z19transpose	0.135s	67.4%	37.5%	0.0%	100.0%	100.0%	24,390,277	0	0	13
▶ [Loop at line 6	0.025s	0.0%	100.0%	0.0%	0.0%	0.0%	13,129,540	0	0	0
▶ [Loop at line 2	0.020s	0.0%	0.0%	0.0%	0.0%	0.0%	7,515,643	0	0	0
▶ main	0.015s	0.0%	0.0%	0.0%	0.0%	0.0%	37,578,215	22,546,929	0	7
▶ [Loop at line 6	0.005s	0.0%	0.0%	100.0%	0.0%	0.0%	0	7,762,721	0	0

Проблемы:

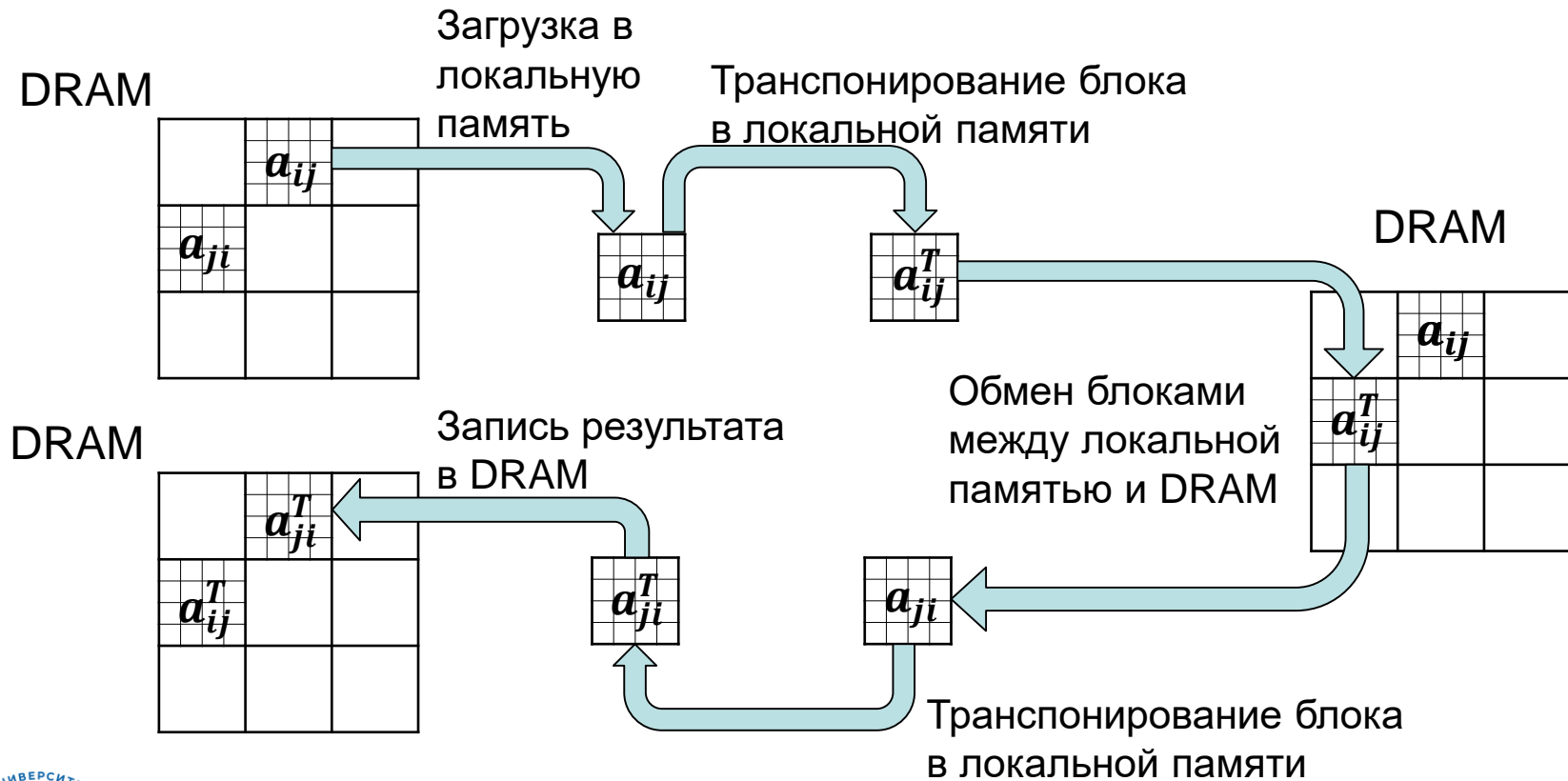
- ❑ 100% DRAM Bound
- ❑ Почти все LLC промахи в транспонировании
- ❑ Огромное количество загрузок и записей

Возможные модификации:

- ❑ Разделить работу с памятью и транспонирование внутри блока



# Оптимизация загрузки



# Модификация кода

```
void transpose_block_omp(double* mat, int n) {
    const int bl_size = 64;
    #pragma omp parallel for
    for (int bi = 0; bi < n / bl_size; bi++) {
        int bl_start = bi * bl_size;
        for (int i = bl_start; i < bl_start + bl_size; i++)
            for (int j = i + 1; j < bl_start + bl_size; j++)
                std::swap(mat[i * n + j], mat[j * n + i]);
        int bj = bl_start + bl_size;
        for (; bj + bl_size < n; bj += bl_size)
            for (int i = bl_start; i < bl_start + bl_size; i++)
                for (int j = bj; j < bj + bl_size; j++)
                    std::swap(mat[i * n + j], mat[j * n + i]);

        for (int i = bl_start; i < bl_start + bl_size; i++)
            for (int j = bj; j < n; j++)
                std::swap(mat[i * n + j], mat[j * n + i]);
    }
    int block_index = n / bl_size;
    for (int i = block_index * bl_size; i < n; i++)
        for (int j = i + 1; j < n; j++)
            std::swap(mat[i * n + j], mat[j * n + i]);
}
```

```
void transpose_cache_omp(double* mat, int n) {
    const int bl_size = 64;
    #pragma omp parallel for
    for (int bi = 0; bi < n / bl_size; bi++) {
        int bl_start = bi * bl_size;
        ...
        int bj;
        double mat_a[bl_size * bl_size];
        for (bj = bl_start + bl_size; bj + bl_size < n; bj += bl_size) {
            for (int j = bj, k1 = 0; j < bj + bl_size; j++, k1++)
                for (int k2 = 0; k2 < bl_size; k2++)
                    mat_a[k1 * bl_size + k2] = mat[j * n + bl_start + k2];

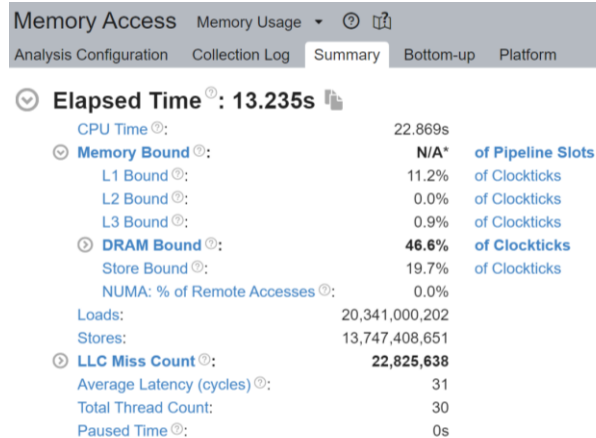
            for (int k1 = 0; k1 < bl_size; k1++)
                for (int k2 = k1 + 1; k2 < bl_size; k2++)
                    std::swap(mat_a[k1 * bl_size + k2],
                               mat_a[k2 * bl_size + k1]);

            for (int i = bl_start, k2 = 0; i < bl_start + bl_size; i++, k2++)
                for (int k1 = 0; k1 < bl_size; k1++)
                    std::swap(mat_a[k2 * bl_size + k1],
                               mat[i * n + bj + k1]);

            for (int k1 = 0; k1 < bl_size; k1++)
                for (int k2 = k1 + 1; k2 < bl_size; k2++)
                    std::swap(mat_a[k1 * bl_size + k2],
                               mat_a[k2 * bl_size + k1]);

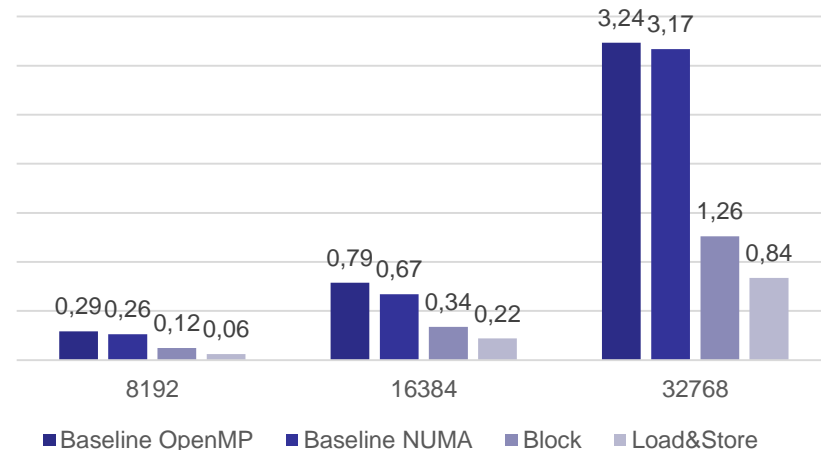
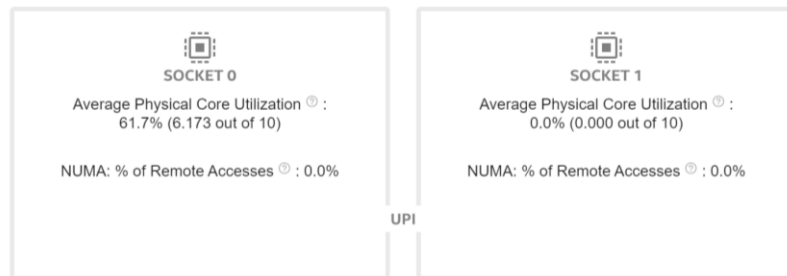
            for (int j = bj, k1 = 0; j < bj + bl_size; j++, k1++)
                for (int k2 = 0; k2 < bl_size; k2++)
                    mat[j * n + bl_start + k2] = mat_a[k1 * bl_size + k2];
        }
        ...
    }
}
```

# Анализ производительности (Load&Store)



Разделение работы с памятью и транспонирования блоков с помощью локального массива, позволило достигнуть почти ручного управления кэшем

## Platform Diagram



# Анализ производительности (Load&Store) (2)

## Блочная версия

Module / Function / Call Stack	CPU Time ▾	Memory Bound					Loads	Stores	LLC Miss Count	Average Latency (cycles)
		L1 Bound	L2 Bound	L3 Bound	DRAM Bound	Store Bound				
MatrixTranspose	6.621s	0.0%	29.3%	10.8%	100.0%	9.4%	2,796,884,247	1,843,867,589	94,893,224	151
▶ [Loop at line 6	6.420s	0.0%	29.1%	12.1%	100.0%	7.7%	2,714,270,572	1,813,557,939	94,893,224	155
▶ Z19transpose	0.135s	67.4%	37.5%	0.0%	100.0%	100.0%	24,390,277	0	0	13
▶ [Loop at line 6	0.025s	0.0%	100.0%	0.0%	0.0%	0.0%	13,129,540	0	0	0
▶ [Loop at line 2	0.020s	0.0%	0.0%	0.0%	0.0%	0.0%	7,515,643	0	0	0
▶ main	0.015s	0.0%	0.0%	0.0%	0.0%	0.0%	37,578,215	22,546,929	0	7
▶ [Loop at line 6	0.005s	0.0%	0.0%	100.0%	0.0%	0.0%	0	7,762,721	0	0

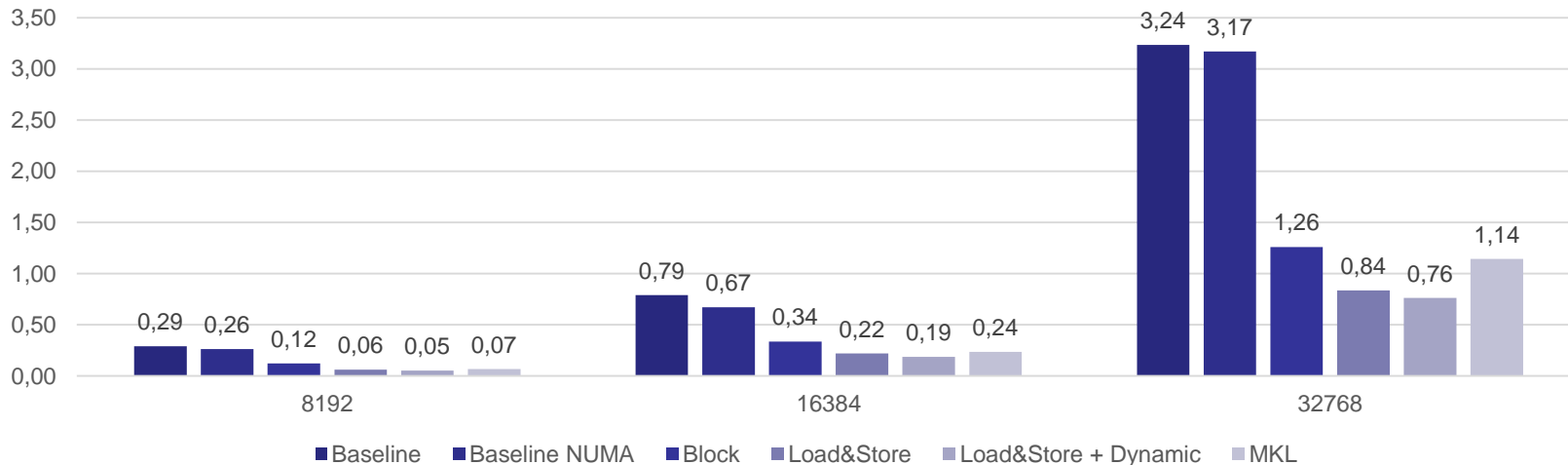
## Кэш версия

Module / Function / Call Stack	CPU Time ▾	Memory Bound					Loads	Stores	LLC Miss Count	Average Latency (cycles)
		L1 Bound	L2 Bound	L3 Bound	DRAM Bound	Store Bound				
MatrixTranspose2.out	7.733s	0.0%	8.4%	0.0%	100.0%	9.5%	2,265,308,162	2,284,012,802	7,608,546	205
▶ __intel_avx_rep_memcpy	3.954s	0.0%		0.0%		8.2%	632,199,736	272,300,913	7,608,546	620
▶ Z19transpose_cache_on	3.157s	0.0%	8.3%	32.0%	97.2%	7.5%	530,202,918	623,815,526	0	294
▶ [Loop at line 127 in _Z19t	0.301s	0.0%	72.7%	0.0%	0.0%	60.5%	631,013,853	706,583,924	0	17
▶ [Loop at line 138 in _Z19t	0.286s	22.6%		15.2%		0.0%	458,321,469	681,312,439	0	15
▶ [Loop at line 116 in _Z19t	0.030s	56.0%	0.0%	12.9%	100.0%	0.0%	13,570,386	0	0	0
▶ [Loop at line 147 in _Z19t	0.005s	100.0%	0.0%	0.0%	0.0%	0.0%	0	0	0	0

- ❑ Само транспонирование не использует DRAM-память
- ❑ Почти все промахи LLC происходят в цикле загрузки и выгрузки данных в локальный массив
- ❑ Физические ядра задействованы хорошо (6.2 против 1.4 из 10)

# Другие оптимизации

- ❑ В качестве оптимизации параллельного цикла можно использовать динамическое планирование задач, так как присутствует дисбаланс в самой задаче (количество блоков в начале верхне-треугольной матрицы больше, чем в конце)



- ❑ Последние две версии обходят по времени аналогичную операцию в MKL

# Возможные оптимизации в задаче

- ❑ Иные оптимизации, не рассмотренные в ходе работы, и, которые могут помочь еще ускорить время выполнения:
  - Специальный обход элементов внутри блока
  - Специальный обход между блоками
  - Изменение размера блока, в том числе, неквадратный вариант блока
  - Ручное балансирование параллельных вычислений

# САМОСТОЯТЕЛЬНАЯ РАБОТА



# ЗАКЛЮЧЕНИЕ

# Заключение

---

## □ В ходе работы:

- Была рассмотрена задача транспонирования матрицы
- Была выполнена пошаговая оптимизация работы с памятью (ускорение в 4 раза по сравнению с базовой параллельной реализацией)
- Были рассмотрены возможности Intel® VTune в анализе производительности подсистемы памяти

# Литература

---

1. Intel® VTune Profiler: <https://software.intel.com/en-us/vtune>
2. Intel® Advisor: <https://software.intel.com/en-us/advisor>
3. Intel® C++ Compiler Developer Guide and Reference:  
<https://software.intel.com/en-us/cpp-compiler-developer-guide-and-reference>
4. Intel® MKL:  
<https://www.intel.com/content/www/us/en/develop/documentation/get-started-with-mkl-for-dpcpp/top.html>

# Авторский коллектив

---

- ❑ Мееров Иосиф Борисович, к.т.н., доцент, зам. зав. каф. МОСТ
- ❑ Сысоев Александр Владимирович, к.т.н., доцент каф. МОСТ
- ❑ Линев Алексей Владимирович, зав. лаб. интернета вещей, каф. ПРИН
- ❑ Волокитин Валентин Дмитриевич, программист лаборатории СТиВВ, каф. МОСТ
- ❑ Козинев Евгений Александрович, к.т.н., преподаватель каф. МОСТ
- ❑ Панова Елена Анатольевна, инженер лаборатории СТиВВ, каф. МОСТ

# Контакты

---

Нижегородский государственный университет

<http://www.unn.ru>

Центр компетенций oneAPI в ННГУ

<http://hpc-education.unn.ru/ru/центр-компетенций-oneapi-в-ннгу>

Институт информационных технологий, математики и механики

<http://www.itmm.unn.ru>