



# **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***

**Оптимизация вычислений, аспекты параллелизма,  
балансировка нагрузки.  
Задача о вычислении простых чисел**

# Содержание

---

- ❑ Цель работы
- ❑ Постановка задачи
- ❑ Различные подходы к распределению вычислительной нагрузки
  - Разделение множества чисел на одинаковые части по числу потоков
  - Разделение множества чисел по схеме Round-robin
  - Разделение множества чисел на небольшие группы
  - Динамическое распределение вычислительной нагрузки
  - Распределение нагрузки средствами OpenMP

# Цели

---

- ❑ Рассмотреть на примере задачи разложения чисел на простые множители некоторые вопросы, возникающие при распараллеливании алгоритмов на системах с общей памятью
- ❑ Получить навыки анализа и сравнения различных подходов к распараллеливанию с помощью инструментов Intel® oneAPI

# Постановка задачи...

---

- ❑ Задача: разложить на простые множители (факторизовать) числа из диапазона от 1 до  $N$
- ❑ Используется алгоритм, который основан на попытке деления факторизируемого числа на каждое из меньших его чисел:
  - Если остаток от деления равен нулю, то очередной множитель запоминается, после чего производится повторная попытка деления на это же число
  - При нахождении каждого множителя факторизируемое число делится на него
  - Алгоритм завершает работу, когда частное от очередного деления становится равным единице

# Постановка задачи

□ Пример:  $12 = ? * ? * \dots * ?$

$12 / 2 = 6$ ; // пробуем разделить на 2 раз

$6 / 2 = 3$ ; // пробуем разделить на 2 еще раз

$3 / 2 = 1.5$ ; // берем следующий делитель

$3 / 3 = 1$ ; // СТОП! – получили единицу

□ Результат:  $12 = 2 * 2 * 3$

□ Замечание: данный алгоритм факторизации использован только для демонстрации возможностей пакета Intel® oneAPI, а также демонстрации методов распределения вычислительной нагрузки.

– На практике используют алгоритмы Полларда, Диксона и др.

# Программная реализация...

## □ Факторизация заданного числа:

```
void factorization_i(int number, vector<vector<int> > &divisors)
{
    int idx = number;
    for (int j = 2; j < idx / 2; j++)
    {
        if (number == 1) break;
        int r;
        r = number % j;
        if (r == 0)
        {
            divisors[idx].push_back(j);
            number /= j;
            j--;
        }
    }
}
```

- код факторизации будет неизменным как для последовательной, так и для параллельных версий алгоритмов

# Программная реализация

## □ Последовательное решение задачи:

```
void factorization(vector<vector<int> > &divisors)
{
    for (int i = 1; i < NUM_NUMBERS; i++)
    {
        int number = i + 1;
        factorization_i(number, divisors);
    }
}
```

```
int main()
{
    vector<vector<int> > divisors(NUM_NUMBERS + 1);
    double time = omp_get_wtime();
    factorization(tid, numThreads, divisors);
    time = omp_get_wtime() - time;
    cout << "Time :" << time << endl;
    return 0;
}
```



# Вычислительная инфраструктура

## □ Вычислительная инфраструктура:

Процессоры	Intel(R) Xeon(R) Platinum 8260L CPU @ 2.40GHz
Число процессоров	2
Общее число ядер	48
Память	512 Gb
Операционная система	Linux CentOS 7
Компилятор, профилировщик, отладчик	Intel® oneAPI

# Результаты вычислительных экспериментов

---

- ❑ Решаемая задача:

- Факторизовать числа [2..1000000]

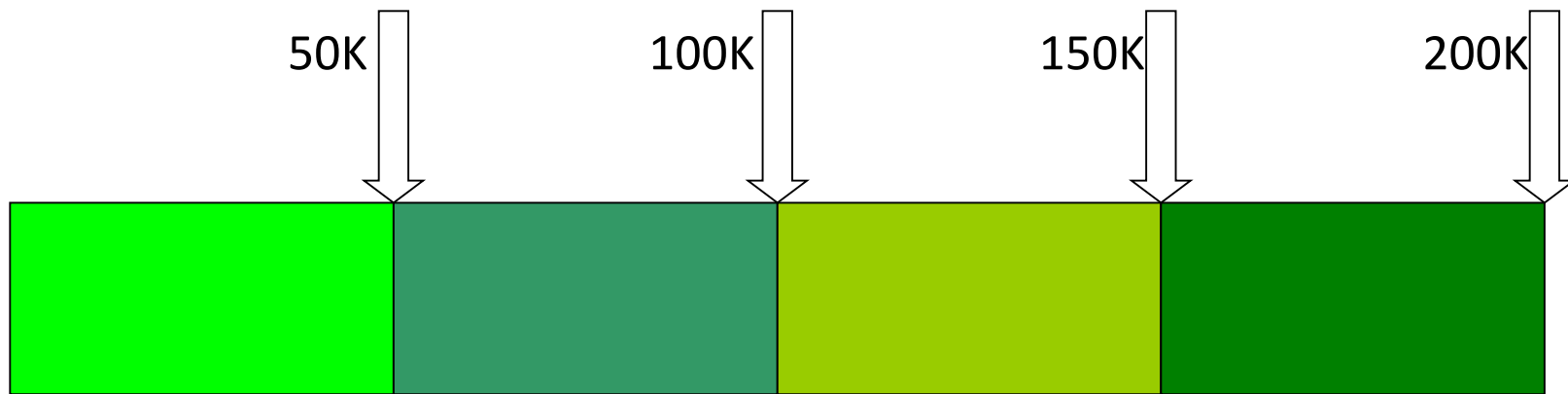
- ❑ Время работы последовательного алгоритма: 174.15 с.

- ❑ Дальнейшие шаги оптимизации программы:

- Реализация различных схем распределения вычислительной нагрузки между ядрами
  - Замер показателей производительности – время вычислений и ускорение
  - Профилирование полученных реализаций

# Подход #1: разделение множества чисел на одинаковые части по числу потоков

- ❑ Стратегия распределения нагрузки между потоками (полная реализация в папке 01\_EqualPartition):



// tid - идентификатор потока

```
1. start ← (NUM_NUMBERS / NUM_THREADS) * tid;  
2. end   ← (NUM_NUMBERS / NUM_THREADS) * (tid + 1);
```

# Подход #1: разделение множества чисел на одинаковые части по числу потоков

## □ Программная реализация:

```
void factorization(int tid, int numThreads, vector<vector<int> > &divisors)
{
    int start, end;

    start = (NUM_NUMBERS / numThreads) * tid;
    end    = (NUM_NUMBERS / numThreads) * (tid+1);

    if(tid+1 == numThreads)
    {
        end = NUM_NUMBERS;
    }
    for (int i = start; i < end; i++)
    {
        int number = i + 1;
        factorization_i(number, divisors);
    }
}
```

# Подход #1: разделение множества чисел на одинаковые части по числу потоков

## □ Программная реализация:

```
int main()
{
    vector<vector<int> > divisors(NUM_NUMBERS + 1);
    double time;

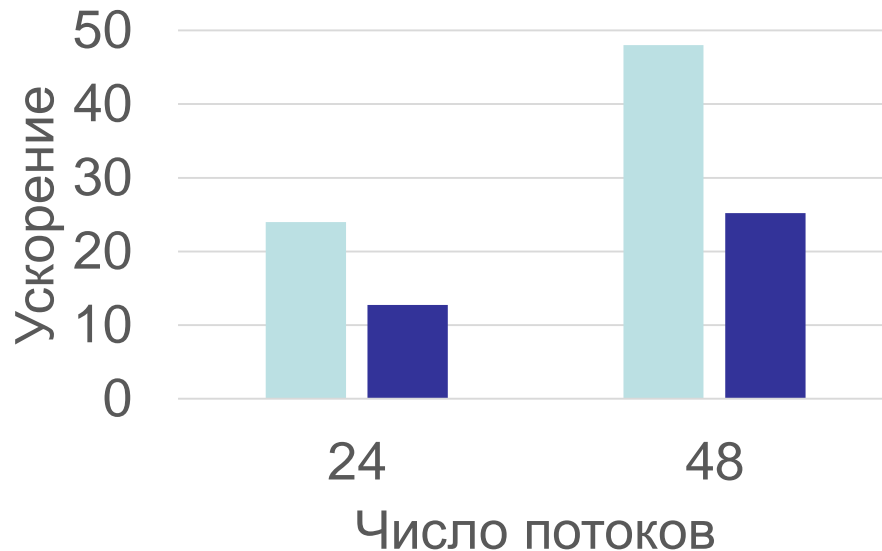
    time = omp_get_wtime();
    #pragma omp parallel
    {
        int numThreads = omp_get_num_threads();
        int tid = omp_get_thread_num();
        factorization(tid, numThreads, divisors);
    }
    time = omp_get_wtime() - time;

    cout << "Time :" << time << endl;
```

# Результаты вычислительных экспериментов

□ Чем обусловлена низкая эффективность?

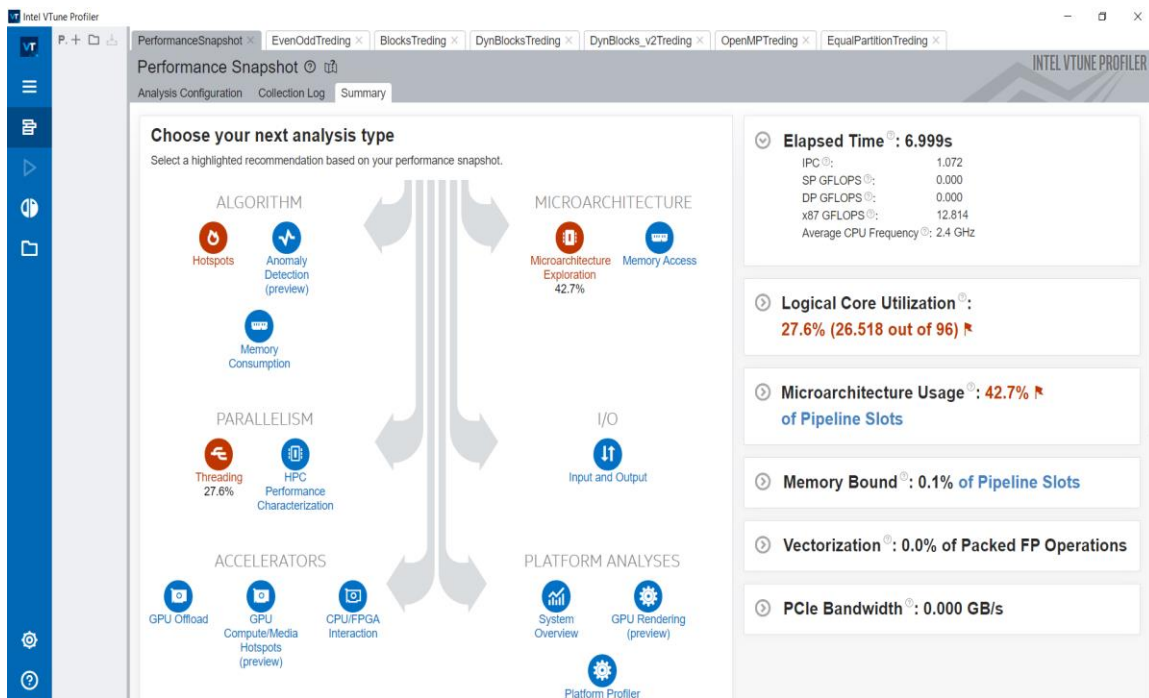
Число потоков	Время, с.	Ускорение
1	174.2	1
24	13.7	12.7
48	6.9	25.2



# Профилировка программной реализации средствами Intel® oneAPI: общий анализ производительности

vtune -collect performance-snapshot -r PerformanceSnapshot --  
./01\_EqualPartition/EqualPartition

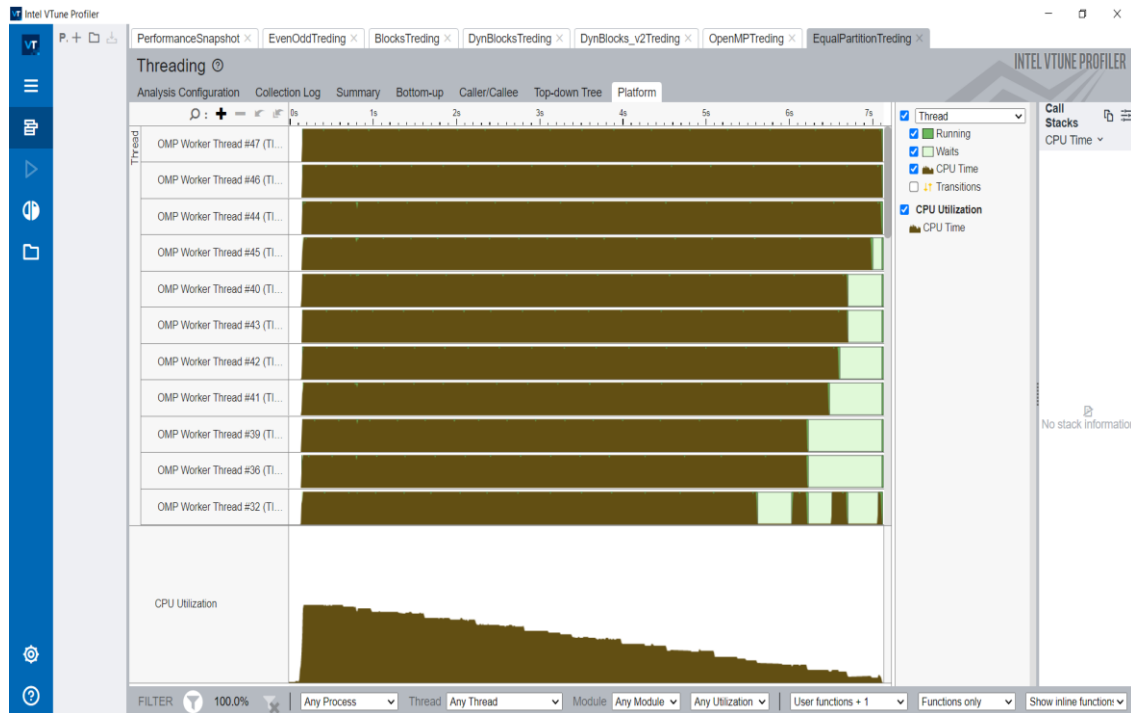
- В дальнейшем рекомендуется собрать статистику инструмента Hotspots, Threading и Microarchitecture exploration
- Обратим более пристальное внимание на инструмент Threading



# Профилировка программной реализации средствами Intel® oneAPI

vtune -collect threading -r EqualPartitionTreading -- ./01\_EqualPartition/EqualPartition

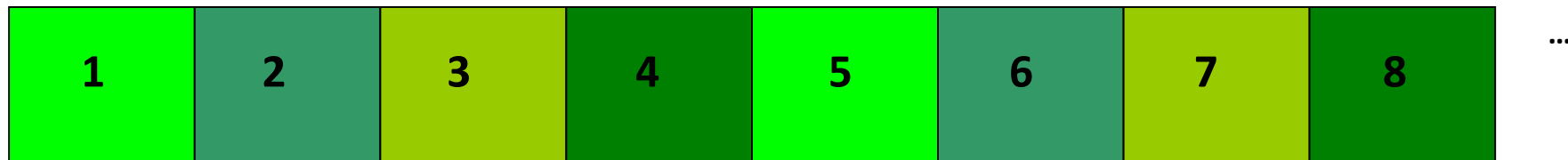
- Из графика справа видно, что у потоков существенно разная загрузка вычислениями.
  - Необходима балансировка нагрузки.
- Возможные варианты:
  - Статическая балансировка
  - Динамическая балансировка





## Подход #2: разделение множества чисел по схеме Round-robin

- ❑ Стратегия распределения нагрузки между потоками (полная реализация в папке 02\_EvenOdd):



// tid - идентификатор потока

// i - индекс числа в наборе чисел, факторизуемых потоком

1. number = i \* NUM\_THREADS + tid + 1;

## Подход #2: разделение множества чисел по схеме Round-robin

### □ Программная реализация:

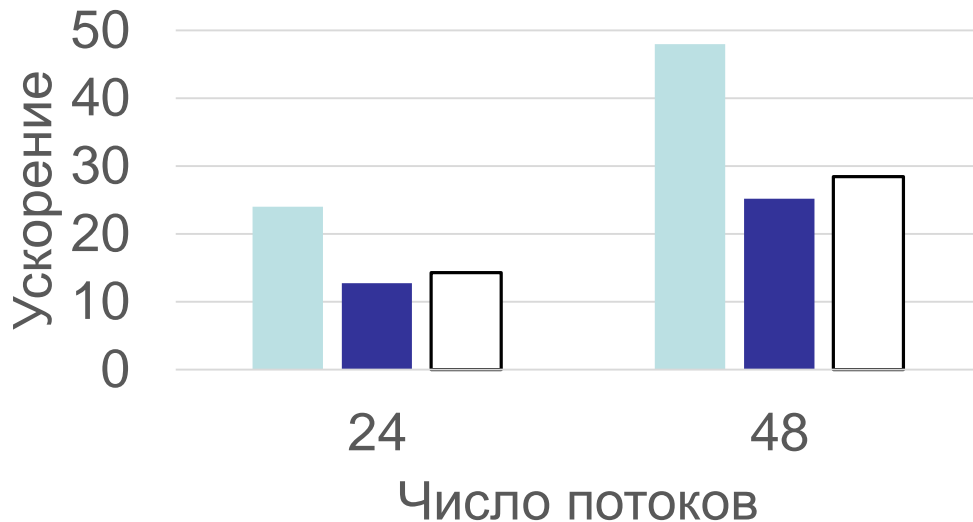
```
void factorization(int tid, int numThreads, vector<vector<int> > &divisors)
{
    for (int i = tid; i < NUM_NUMBERS; i+=numThreads)
    {
        int number = i + 1;
        factorization_i(number, divisors);
    }
}
```

```
int main()
{
    ...
    #pragma omp parallel
    {
        int numThreads = omp_get_num_threads();
        int tid = omp_get_thread_num();
        factorization(tid, numThreads, divisors);
    }
}
```

# Результаты вычислительных экспериментов

□ Почему эффективность выросла незначительно?

Число потоков	Время, с.	Ускорение
1	174.2	1
24	12.2	14.3
48	6.1	28.4



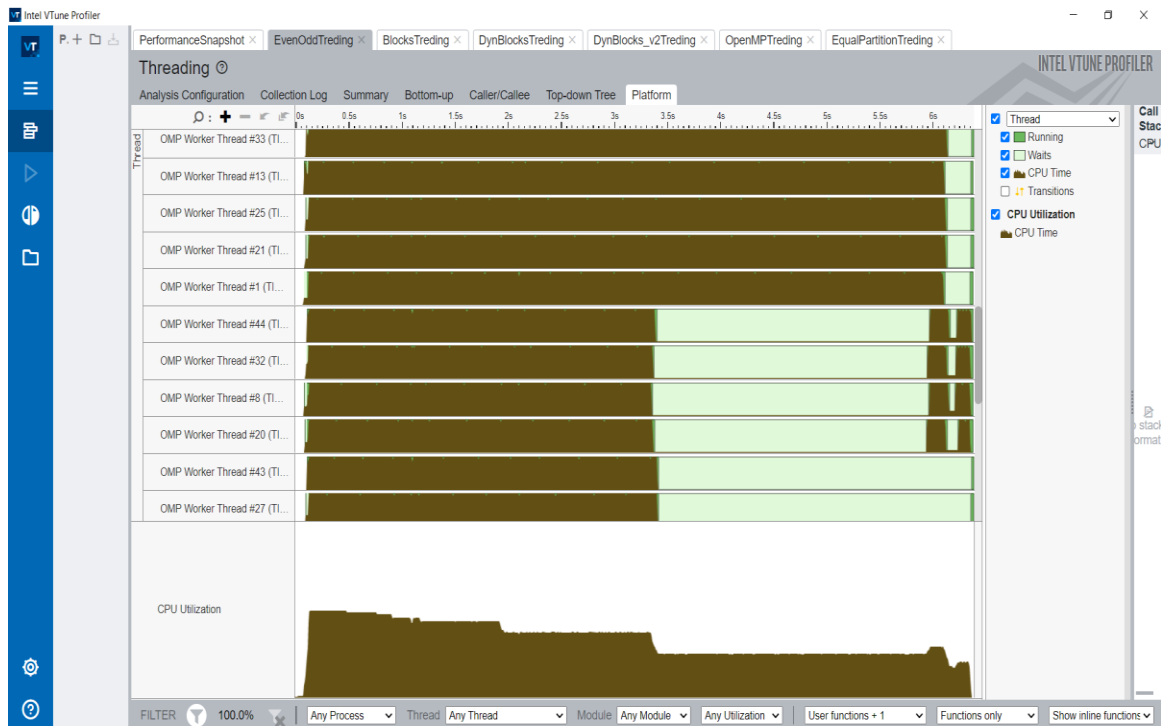
■ Linear Speedup ■ Equal Partition  
□ EvenOdd

# Профилировка программной реализации средствами Intel® oneAPI

vtune -collect threading -r EvenOddTreding -- ./02\_EvenOdd/EvenOdd

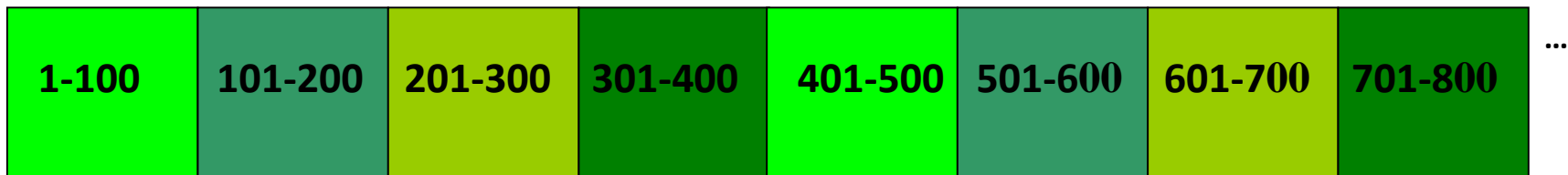
□ Загрузка по прежнему  
не равномерная.  
Почему?

– Ряд чисел кратны  
степени 2



## Подход #3: разделение множества чисел на небольшие группы

- Стратегия распределения нагрузки между потоками (полная реализация в папке 03\_Blocks):



// tid - идентификатор потока

```
1. numberOfGrains = NUM_NUMBERS / NUM_THREADS / GRAIN_SIZE;  
2. for i = 0 to numberOfGrains  
3.     begin = (NUM_THREADS * i + tid) * GRAIN_SIZE + 1;  
4.     end   = (NUM_THREADS * i + tid + 1) * GRAIN_SIZE + 1;
```

## Подход #3: разделение множества чисел на небольшие группы

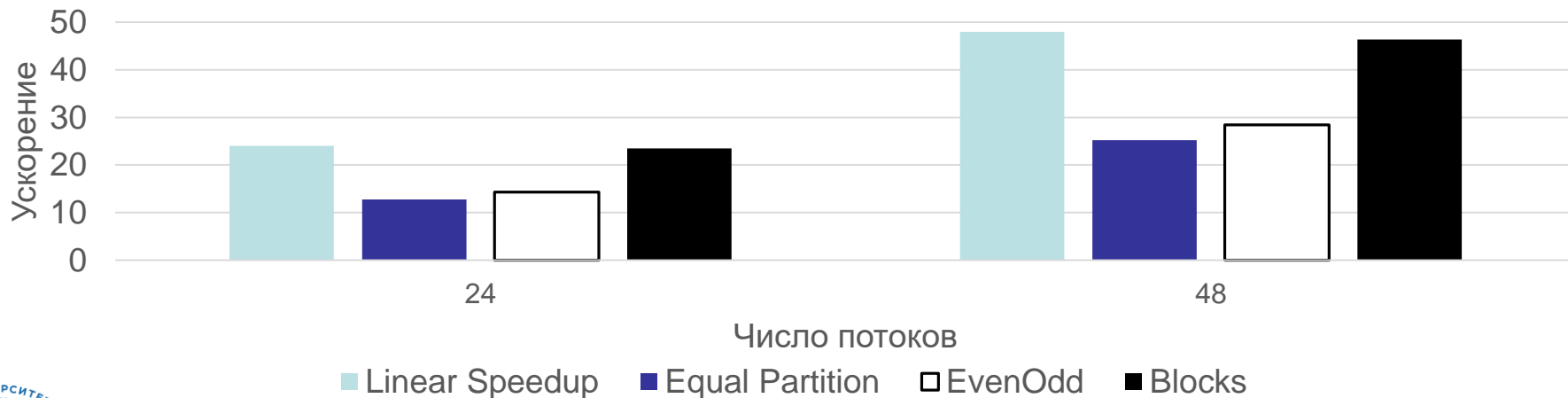
### □ Программная реализация:

```
void factorization(int tid, int numThreads, vector<vector<int> > &divisors)
{
    for (int i = tid * GRAIN_SIZE; i < NUM_NUMBERS; i+=numThreads* GRAIN_SIZE)
    {
        int end = i + GRAIN_SIZE;
        if(NUM_NUMBERS < i + GRAIN_SIZE)
        {
            end = NUM_NUMBERS;
        }
        for (int j = i; j < end; j++)
        {
            int number = j + 1;
            factorization_i(number, divisors);
        }
    }
}
```

# Результаты вычислительных экспериментов

□ Эффективность значительно выросла. Будем использовать размер блока 50.

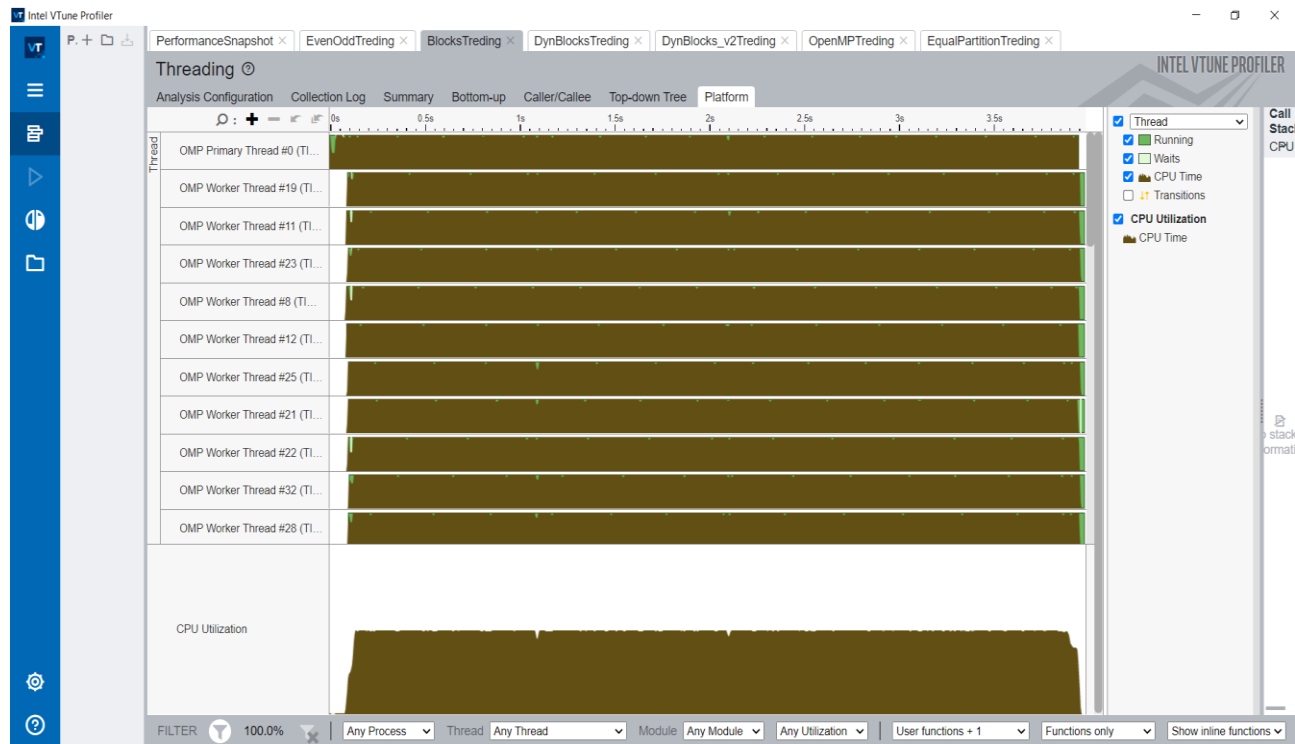
	Размер блока 50		Размер блока 100		Размер блока 500		Размер блока 1000	
Число потоков	Время	Ускорение	Время	Ускорение	Время	Ускорение	Время	Ускорение
1	174.2	1	174.2	1	174.2	1	174.2	1
24	7.4	23.5	7.4	23.5	8.3	21.1	8.3	20.9
48	3.8	46.4	3.8	46.4	4.2	41.2	4.3	40.8



# Профилировка программной реализации средствами Intel® oneAPI

vtune -collect threading -r BlocksTreding -- ./03\_Blocks/Blocks

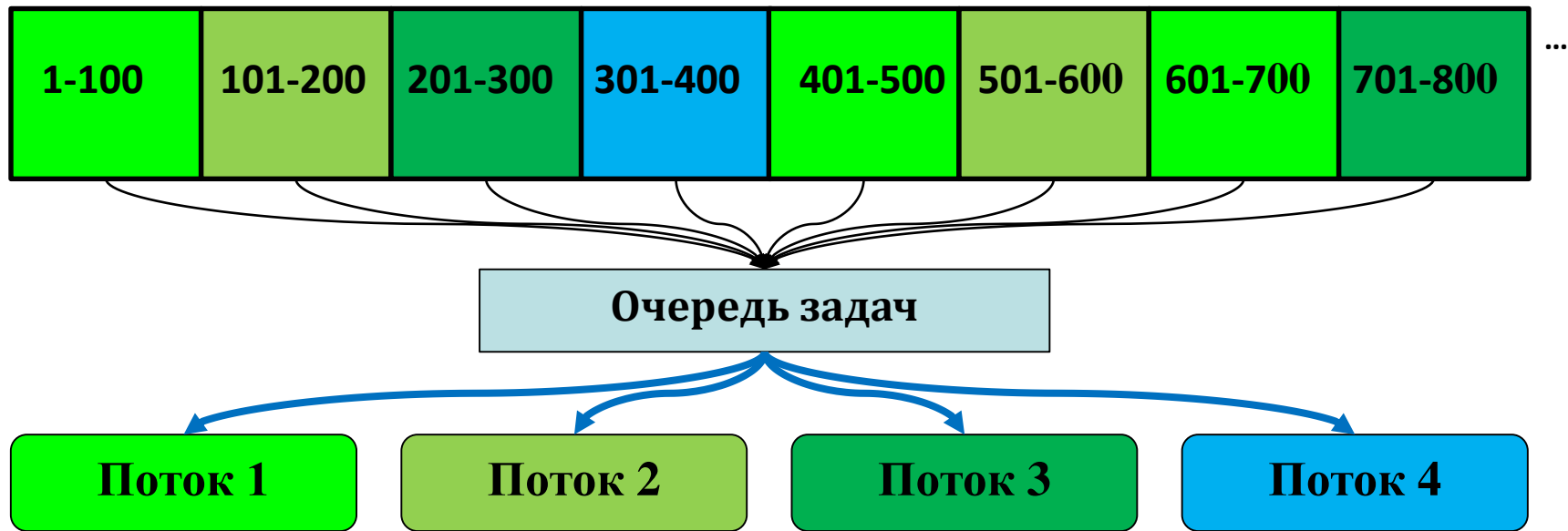
- ❑ Вычисления  
    распределены  
    равномерно
- ❑ Попробуем увеличить  
    ускорение путем  
    динамического  
    планирование





## Подход #4: разделение множества чисел на небольшие группы. Динамическое планирование.

- Стратегия распределения нагрузки между потоками (полная реализация в папке 04\_DynBlocks):



## Подход #4: разделение множества чисел на небольшие группы. Динамическое планирование.

### □ Программная реализация:

```
time = omp_get_wtime();
queue<pair<int, int>> tasks;
for(int i = 0; i < NUM_NUMBERS / GRAIN_SIZE; i++)
{
    tasks.push({i * GRAIN_SIZE, (i + 1) * GRAIN_SIZE});
}
#pragma omp parallel
{
    factorization(tasks, divisors);
}
time = omp_get_wtime() - time;

cout << "Time :" << time << endl;
```

## Подход #4: разделение множества чисел на небольшие группы. Динамическое планирование.

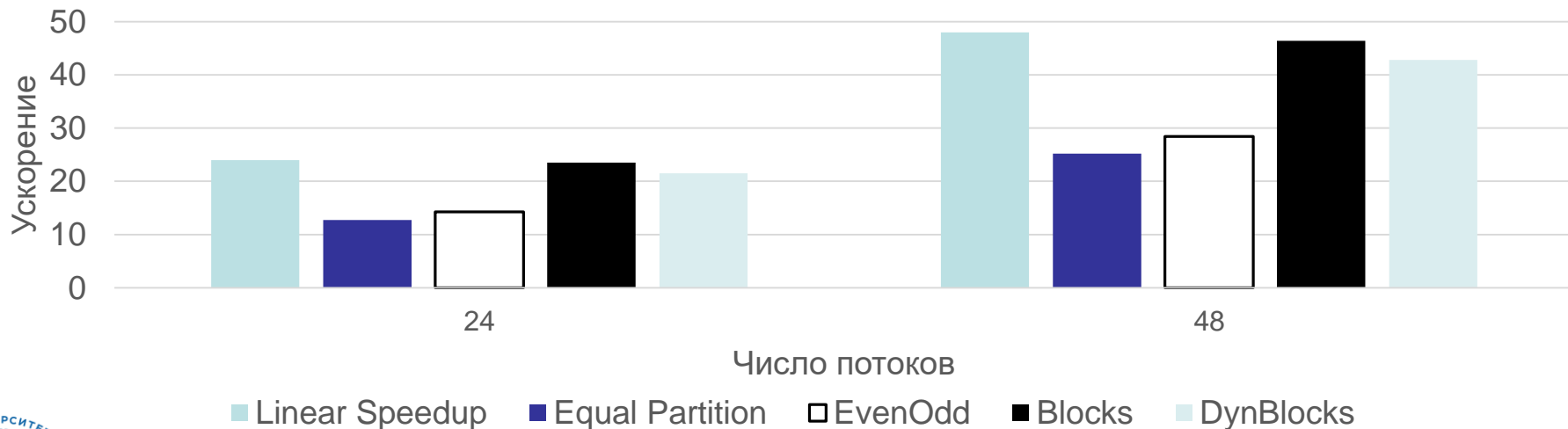
### □ Программная реализация:

```
void factorization(queue<pair <int, int> > &tasks, vector<vector<int> > &divisors)
{ int start = 0, end = 10;
  while (start < end)
  {
#pragma omp critical
  {
    if(!tasks.empty())
    { pair<int, int> task = tasks.front();
      tasks.pop(); start = task.first; end = task.second;
    } else { start = end = 0; }
  }
  for (int i = start; i < end; i++)
  {
    int number = i + 1; factorization_i(number, divisors);
  }
}
```

# Результаты вычислительных экспериментов

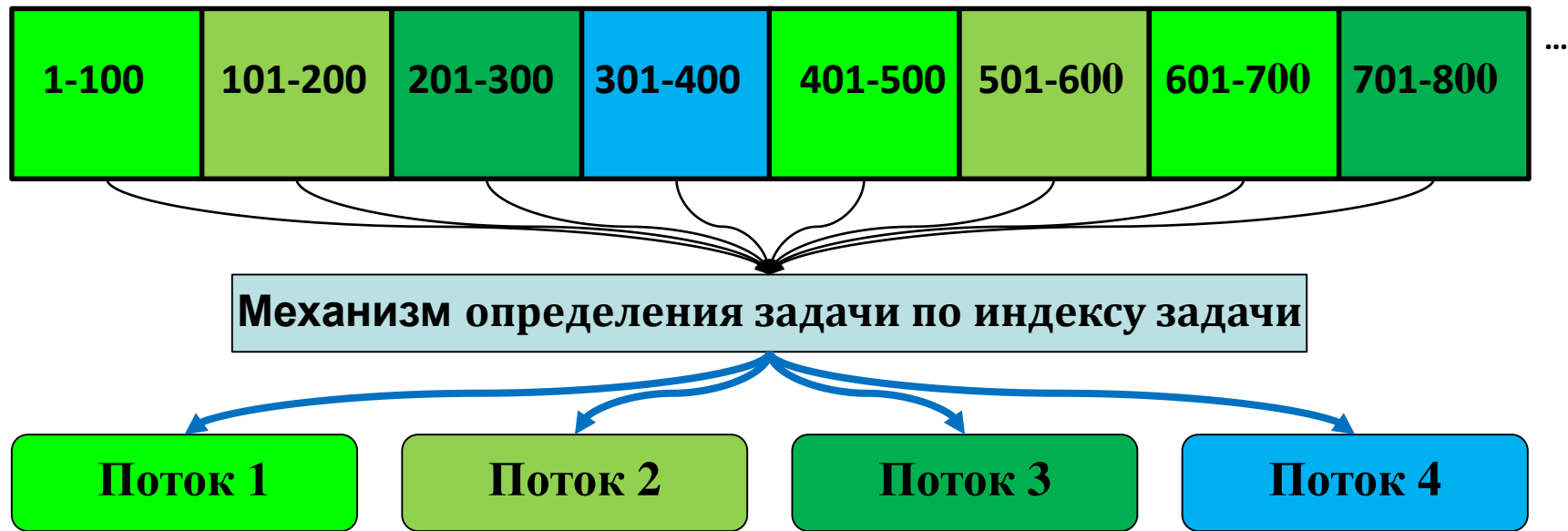
□ Эффективность высокая, но ниже чем при статическом планировании.

	Размер блока 50		Размер блока 100		Размер блока 500		Размер блока 1000	
Число потоков	Время	Ускорение	Время	Ускорение	Время	Ускорение	Время	Ускорение
1	174.2	1	174.2	1	174.2	1	174.2	1
24	8.1	21.5	8.1	21.5	8.2	21.3	8.3	21.1
48	4.1	42.8	4.1	42.7	4.2	41.9	4.3	41.0



## Подход #5: Динамическое планирование без физической очереди

- Стратегия распределения нагрузки между потоками (полная реализация в папке 05\_DynBlocks\_v2):



## Подход #5: Динамическое планирование без физической очереди

### □ Программная реализация:

```
time = omp_get_wtime();  
int taskId = 0; // Разделяемый индекс решаемой задачи  
#pragma omp parallel  
{  
    factorization(taskId, divisors);  
}  
time = omp_get_wtime() - time;  
  
cout << "Time :" << time << endl;
```

## Подход #5: Динамическое планирование без физической очереди

### □ Программная реализация:

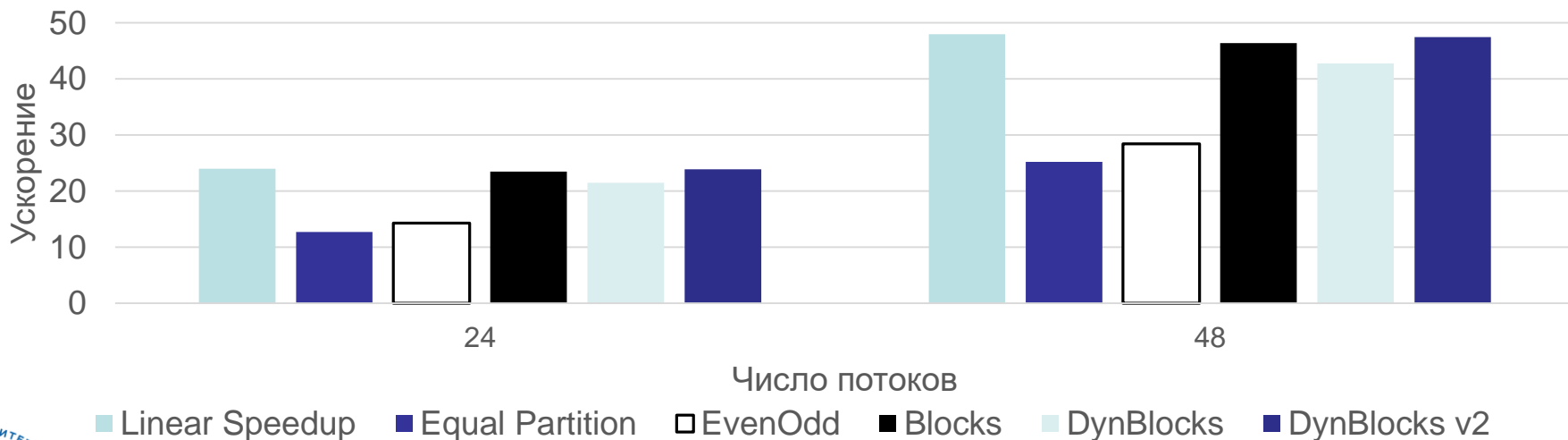
```
void factorization(int &taskId, vector<vector<int> > &divisors)
{ int start = 0, end = 10;
  while (start < end)
  {
#pragma omp critical
  {
    if(taskId < NUM_NUMBERS / GRAIN_SIZE)
    { start = taskId * GRAIN_SIZE;
      end   = (taskId + 1) * GRAIN_SIZE;
    } else { start = end = 0; }
    taskId++;
  }

  for (int i = start; i < end; i++)
  { int number = i + 1;
    factorization_i(number, divisors);
  }
}
```

# Результаты вычислительных экспериментов

□ Эффективность выросла за счет уменьшения времени критической секции

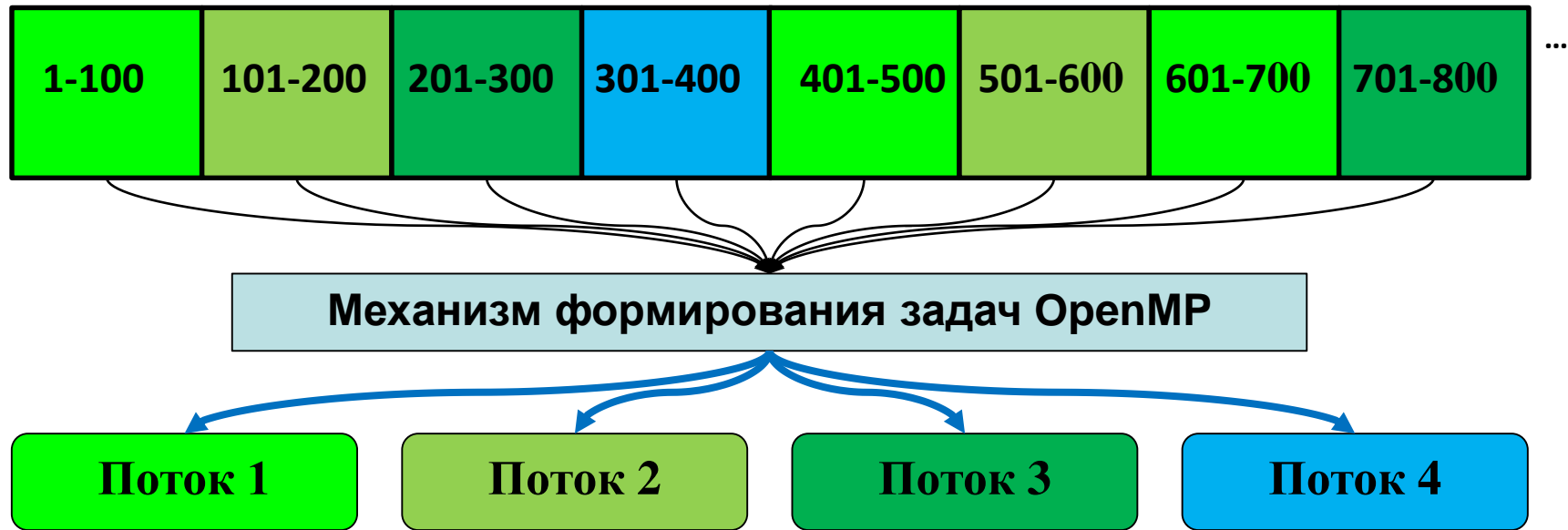
	Размер блока 50		Размер блока 100		Размер блока 500		Размер блока 1000	
Число потоков	Время	Ускорение	Время	Ускорение	Время	Ускорение	Время	Ускорение
1	174.2	1	174.2	1	174.2	1	174.2	1
24	7.3	23.9	7.3	23.9	8.2	21.3	8.3	21.1
48	3.7	47.5	3.7	47.4	4.2	41.9	4.3	41.0





## Подход #6: Динамическое планирование средствами OpenMP

- Стратегия распределения нагрузки между потоками (полная реализация в папке 06\_OpenMP):



## Подход #6: Динамическое планирование средствами OpenMP

### □ Программная реализация:

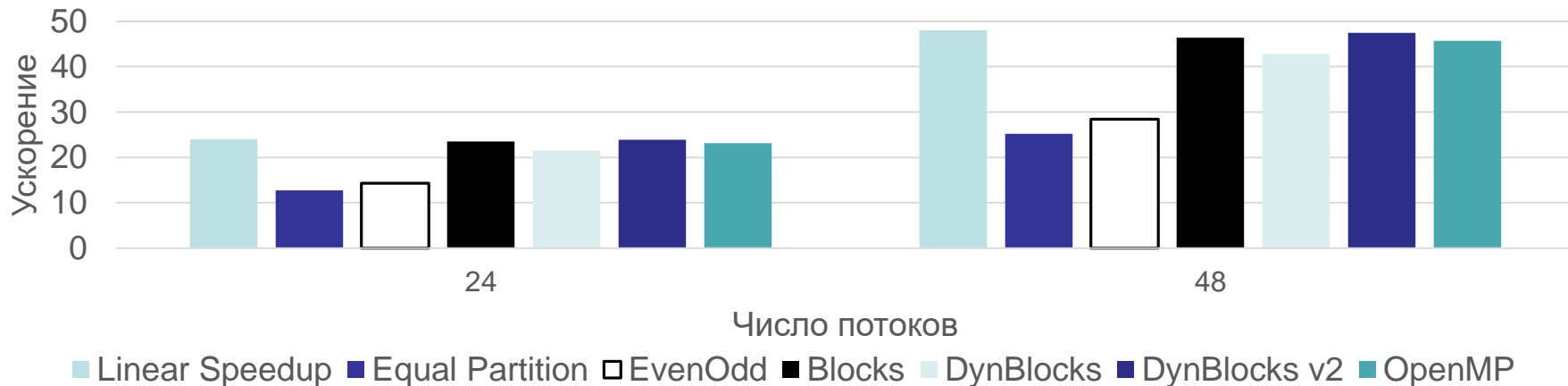
```
time = omp_get_wtime();
int tasksId = 0;
#pragma omp parallel for schedule(dynamic, GRAIN_SIZE)
for (int i = 0; i < NUM_NUMBERS; i++)
{
    int number = i + 1;
    factorization_i(number, divisors);
}
time = omp_get_wtime() - time;

cout << "Time :" << time << endl;
```

# Результаты вычислительных экспериментов

□ Производительность OpenMP версии сравнима со статическим планированием.

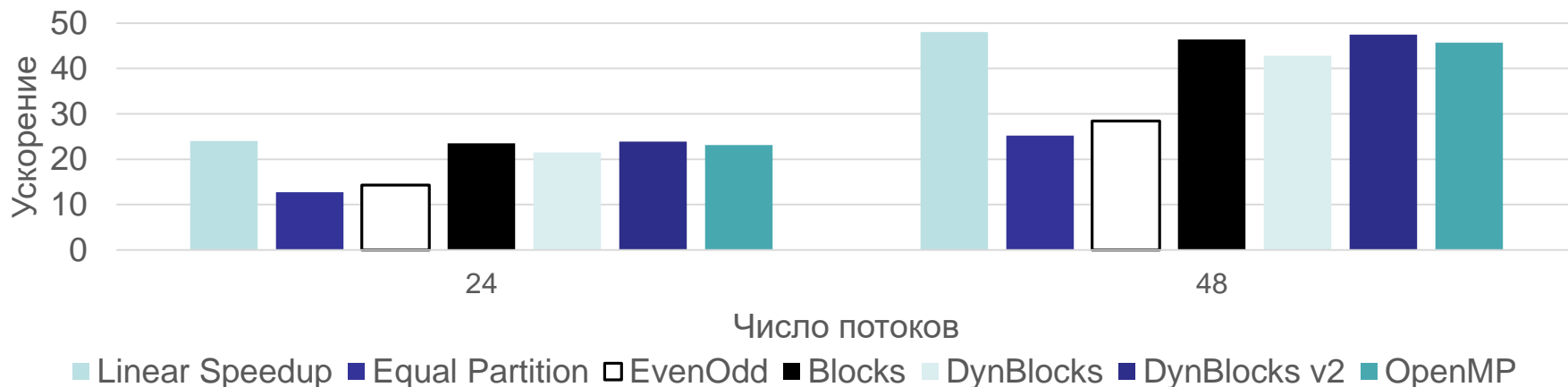
	Размер блока 50		Размер блока 100		Размер блока 500		Размер блока 1000	
Число потоков	Время	Ускорение	Время	Ускорение	Время	Ускорение	Время	Ускорение
1	174.2	1	174.2	1	174.2	1	174.2	1
24	7.5	23.1	7.5	23.1	7.7	22.7	7.8	22.4
48	3.8	45.7	3.8	45.4	3.9	44.8	4.0	43.8



# Результаты вычислительных экспериментов

□ Сводная таблица вычислительных экспериментов:

Число потоков	Ускорение						
	Linear Speedup	Equal Partition	EvenOdd	Blocks	DynBlocks	<b>DynBlocks v2</b>	OpenMP
24	24	12.7	14.3	23.5	21.5	<b>23.9</b>	23.1
48	48	25.2	28.4	46.4	42.8	<b>47.5</b>	45.7



# Литература

---

1. Василенко О.Н. Теоретико-числовые алгоритмы в криптографии. 2003
2. Кнут Д. Искусство программирования. 2007.
3. Черемушкин А.В. Лекции по арифметическим алгоритмам в крипто-графии. 2002.
4. Timothy G. Mattson, Berna Massingill and Beverly Sanders Parallel Programming Patterns: Working with Concurrency in OpenMP, MPI, Java, and OpenCL. 2018.
5. Robert Cook An Introduction to Parallel Programming with OpenMP, PThreads and MPI. 2011

# 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
- ❑ Козинов 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>