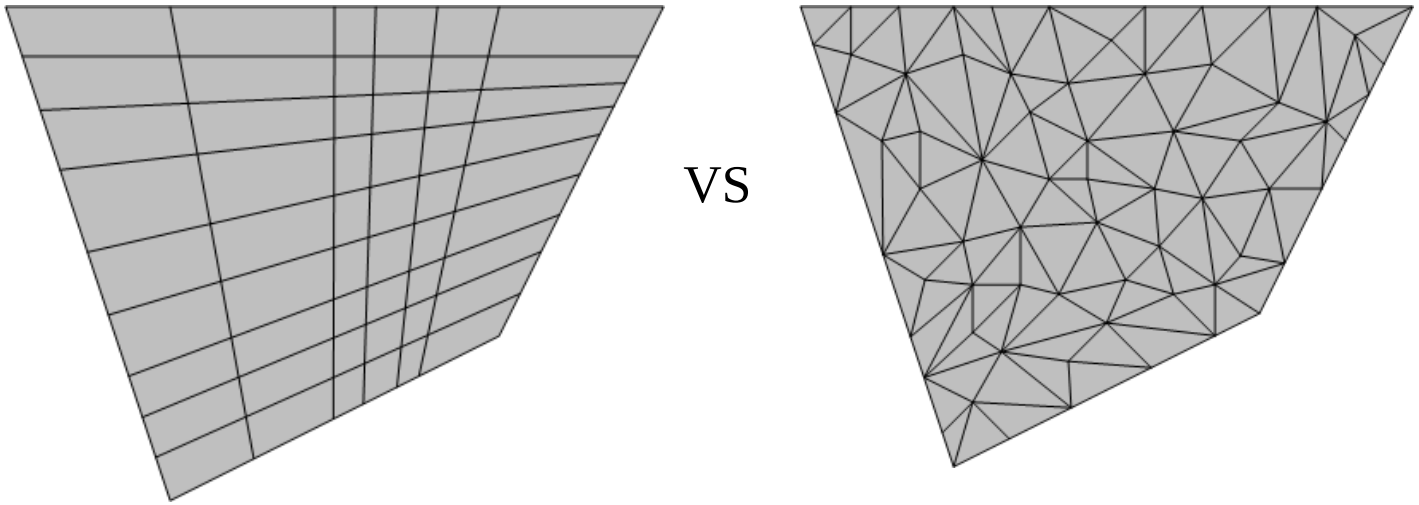
Межведомственный суперкомпьютерный центр Российской академии наук –
филиал Федерального государственного учреждения «Федеральный научный центр
Научно-исследовательский институт системных исследований Российской академии наук»
(МСЦ РАН – филиал ФГУ ФНЦ НИИСИ РАН)

Рыбаков Алексей Анатольевич

Двухуровневое распараллеливание для оптимизации вычислений на суперкомпьютере при расчете задач газовой динамики

Шестая всероссийская конференция
«Вычислительный эксперимент в аэроакустике»
19-24 сентября 2016 года, г. Светлогорск

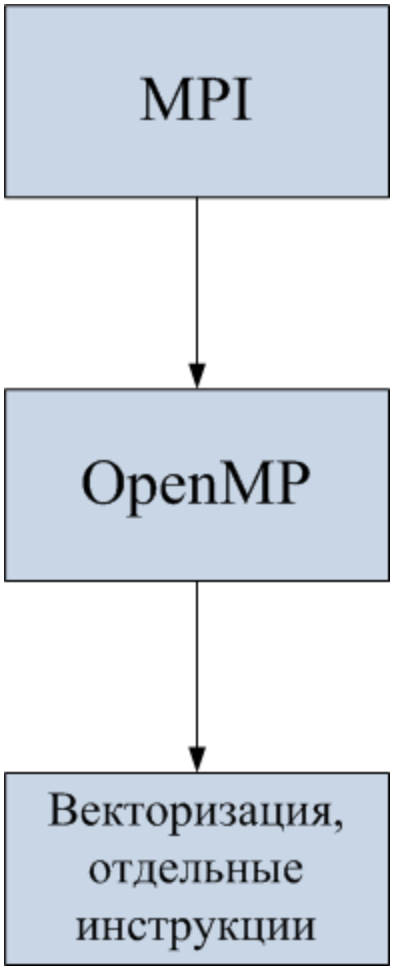
Использование блочно-структурированных сеток в расчетах задач газовой динамики



структурированная сетка неструктурированная сетка

В расчетах задач газовой динамики часто используются блочно-структурированные сетки, состоящие из структурированных блоков. Ячейки упорядочены внутри структурированного блока, что позволяет оптимизировать гнезда циклов по их обработке. Для ускорения расчетов важно уметь равномерно распределять расчеты отдельных блоков между мощностями вычислителя.

Уровни распараллеливания вычислений

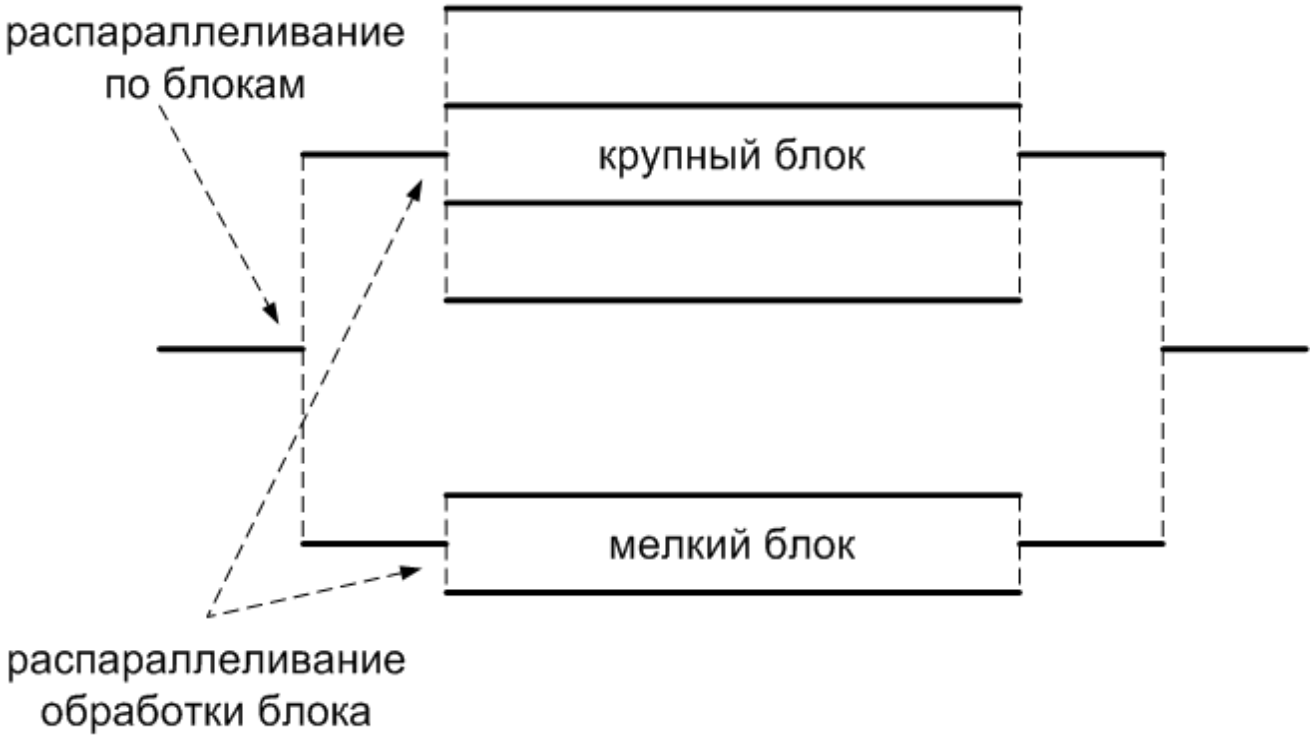


Распараллеливание вычислений между узлами суперкомпьютера с обменом данными с помощью MPI. Каждый узел обрабатывает несколько блоков сетки.

1. Параллельная обработка блоков. Каждый блок обрабатывается в отдельном потоке.
2. Параллельная обработка списков блоков. Блоки разбиваются на списки, каждый список обрабатывается в отдельном потоке.
3. Параллельная обработка блока. Блок обрабатывается несколькими потоками.

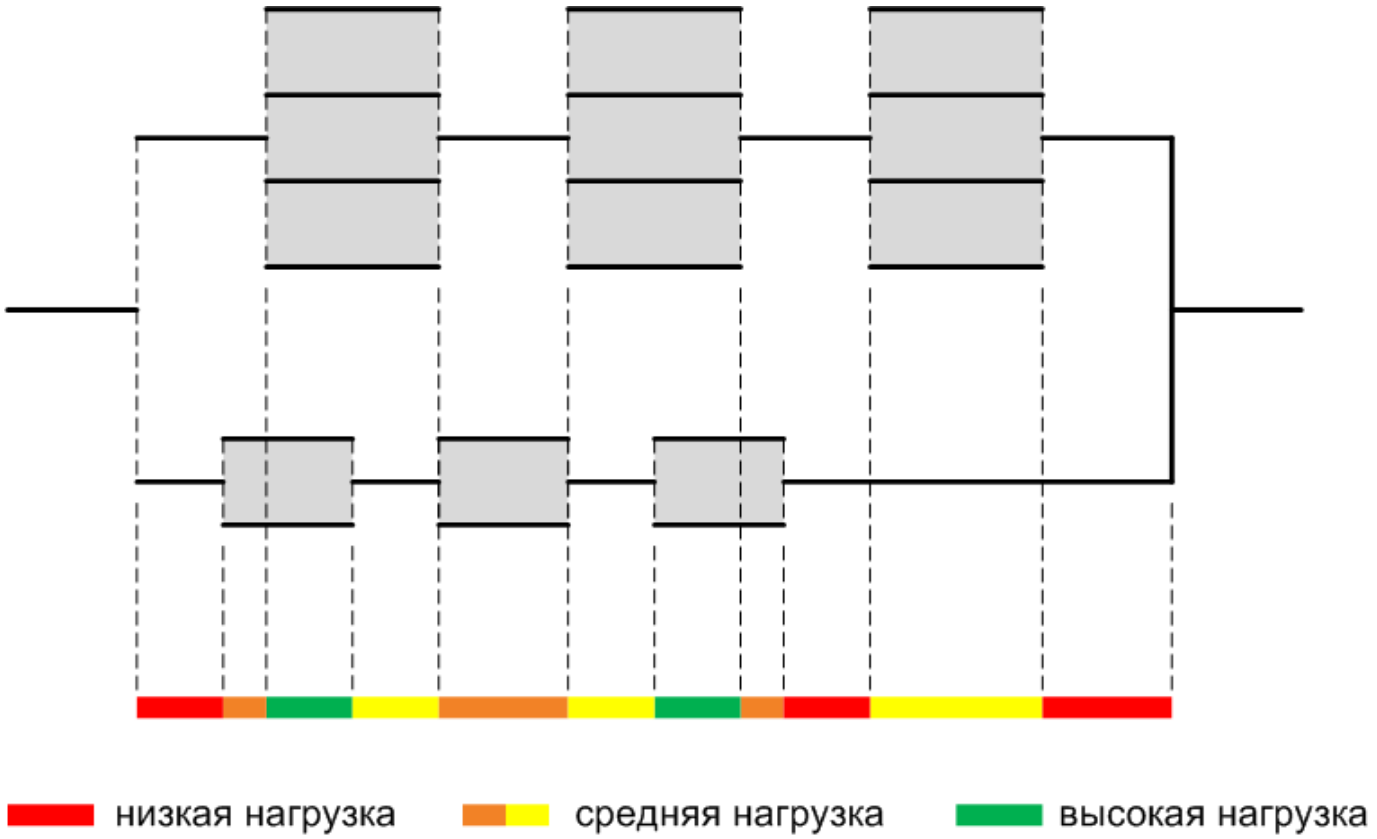
Использование векторных операций, оптимизация кода на уровне отдельных инструкций.

Двухуровневое распараллеливание вычислений внутри одного узла суперкомпьютера



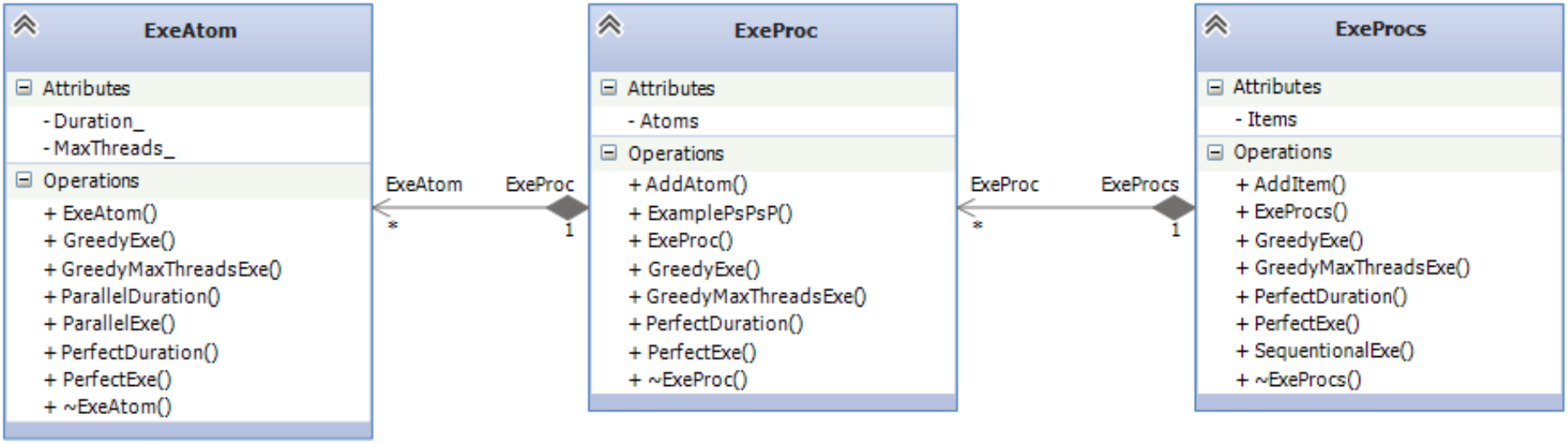
Для сбалансированной по времени обработки блоков сетки внутри одного вычислительного узла степень распараллеливания блока должна зависеть от его размера, то есть блок большего размера должен обрабатываться большим количеством потоков, чем меньший по размеру блок.

Возникновение интервалов с низкой вычислительной нагрузкой внутри узла суперкомпьютера



Бесконтрольное расположение последовательных и параллельных секций обработки двух блоков внутри вычислительного узла приводит в появлению обширных зон низкой загрузки вычислительных мощностей.

Модель управляемой вычислительной процедуры



• Объектом исполнения на вычислителе является набор вычислительных процедур ExeProcs (обработка всех блоков), который состоит из массива ExeProc.

• Вычислительная процедура ExeProc (обработка одного блока) состоит из массива отдельных атомов исполнения ExeAtom.

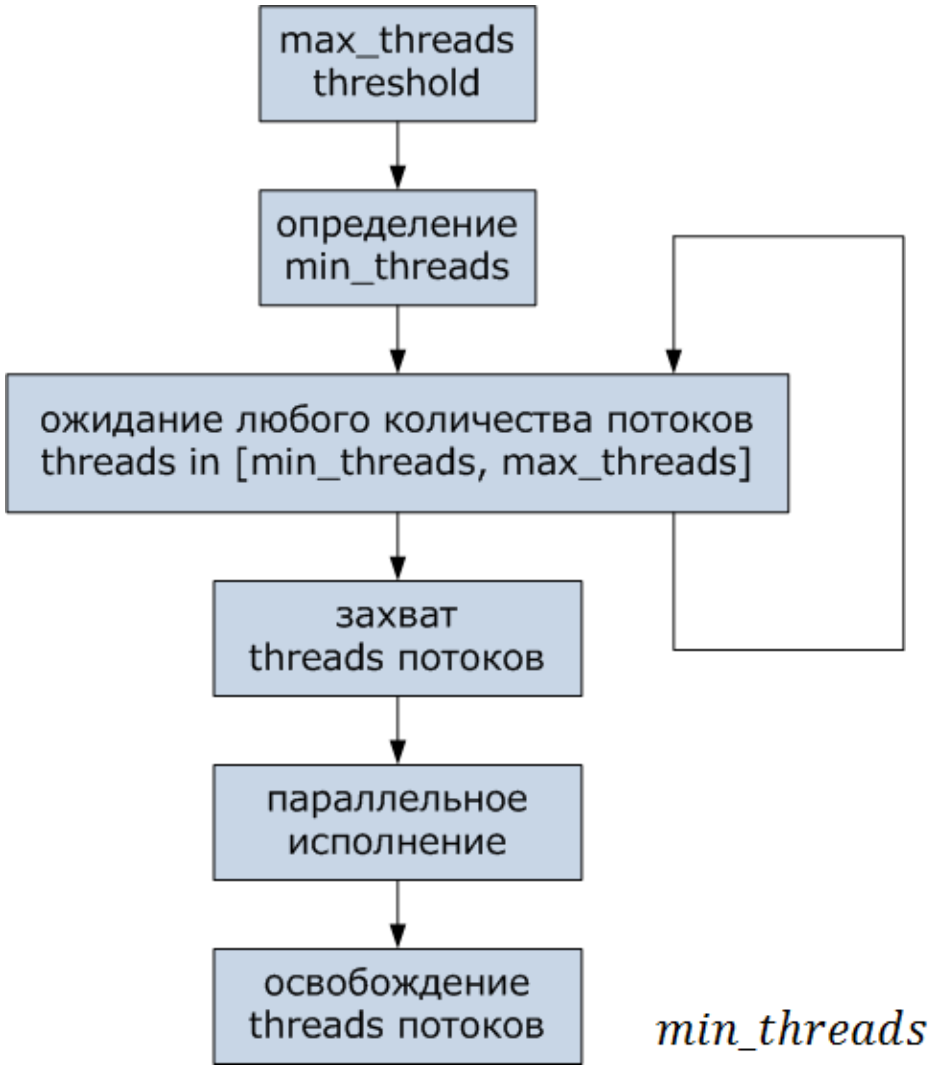
• Атом исполнения является неделимым исполняемым кодом, для которого известна продолжительность исполнения и максимальное полезное количество потоков, которые этот атом может занять.

Механизм жадного захвата потоков исполнения атомом вычислительной процедуры

```
void ExeAtom::GreedyExe(int *free_threads)
{
    int need_threads = <необходимое количество потоков>;
    bool is_start = false;

    while (true)
    {
        while (*free_threads < need_threads) <нулевое ожидание>;

#pragma omp critical
        {
            if (*free_threads >= need_threads)
            {
                is_start = true;
                *free_threads -= need_threads;
            }
        }

        if (!is_start) continue;

        ParallelExe();

#pragma omp critical
        {
            *free_threads += need_threads;
        }

        break;
    }
}
```

1. Общее количество текущих свободных потоков хранится в глобальном счетчике.
2. Атом ожидает пока не освободится достаточное для его исполнения количество потоков.
3. Атом занимает потоки и изменяет глобальный счетчик.
4. Параллельное исполнение атома.
5. Атом освобождает потоки и изменяет счетчик обратно.

Использование порогов снижения количества потоков исполнения

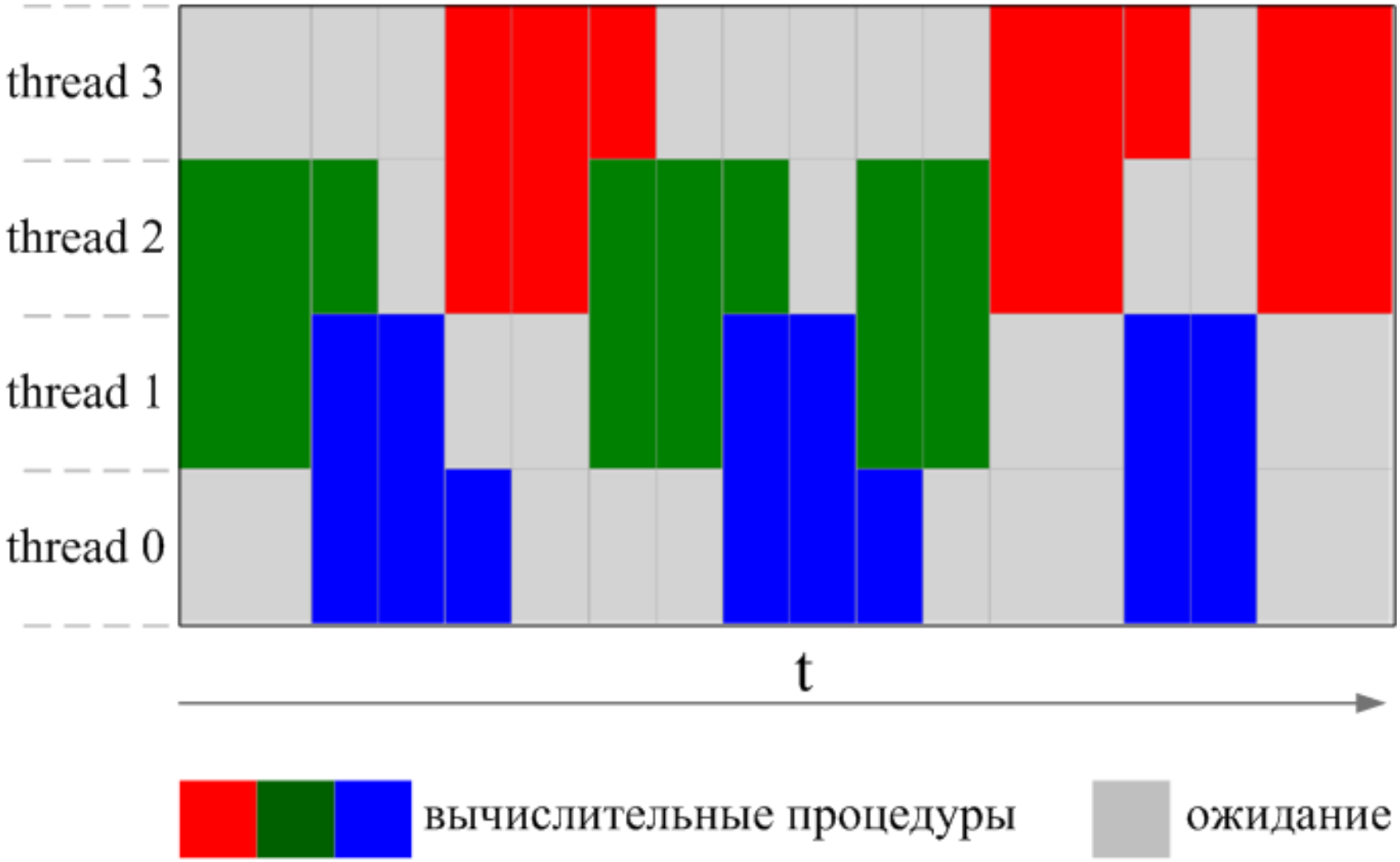


Зачастую невыгодно ждать освобождения полного количества потоков. В этом случае используются пороги снижения количества потоков, которые определяют минимальное количество потоков, которое готов захватить атом.

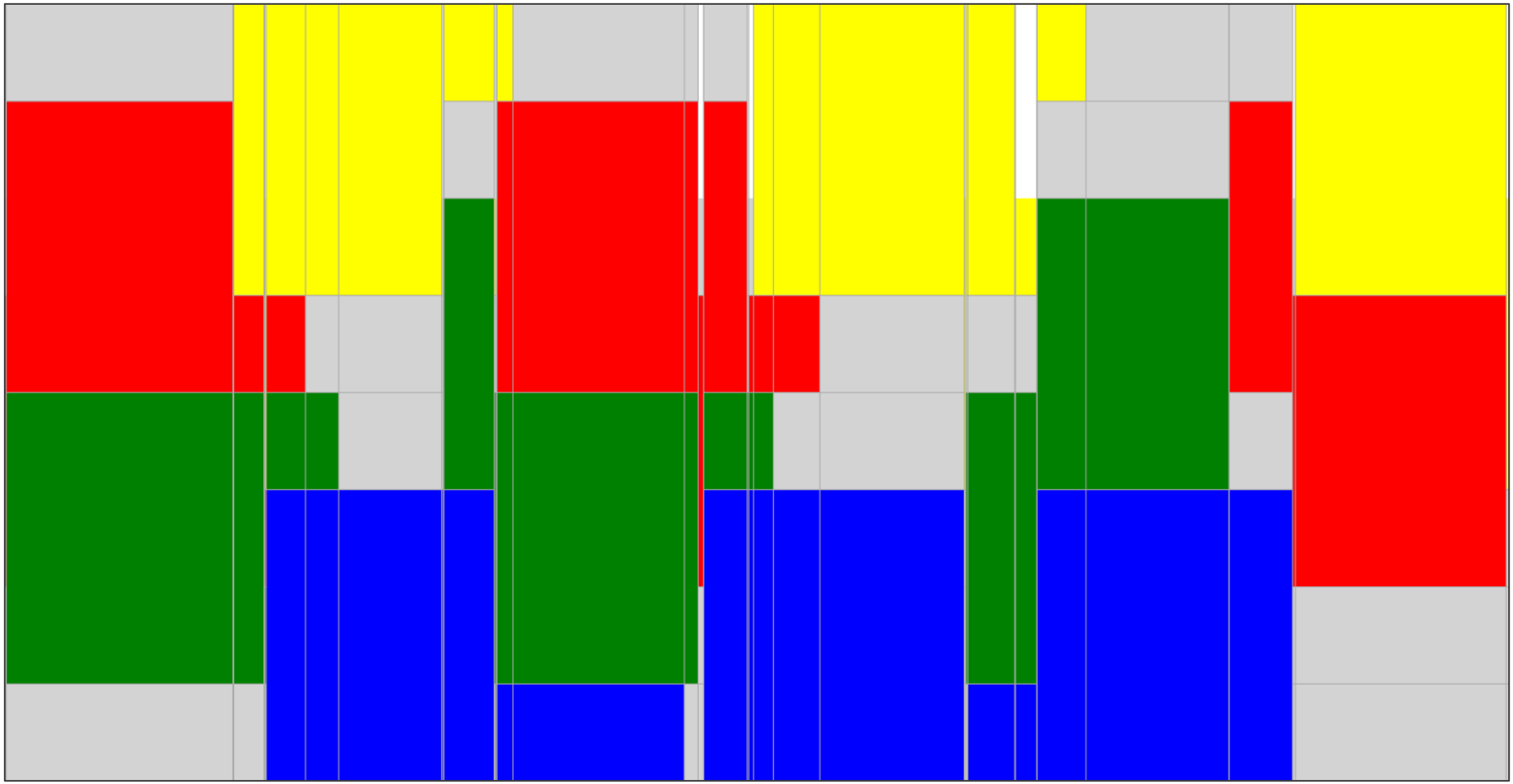
max_threads – максимальное количество потоков, которое может быть захвачено данным атомом,
min_threads – минимальное количество потоков, которое будет захвачено данным атомом,
threads – фактическое количество потоков, захватываемое данным атомом.

$$min\_threads = \max\left(\lfloor max\_threads \times (1 - threshold) \rfloor, 1\right)$$

Распределение вычислений для 4 потоков на модельной задаче

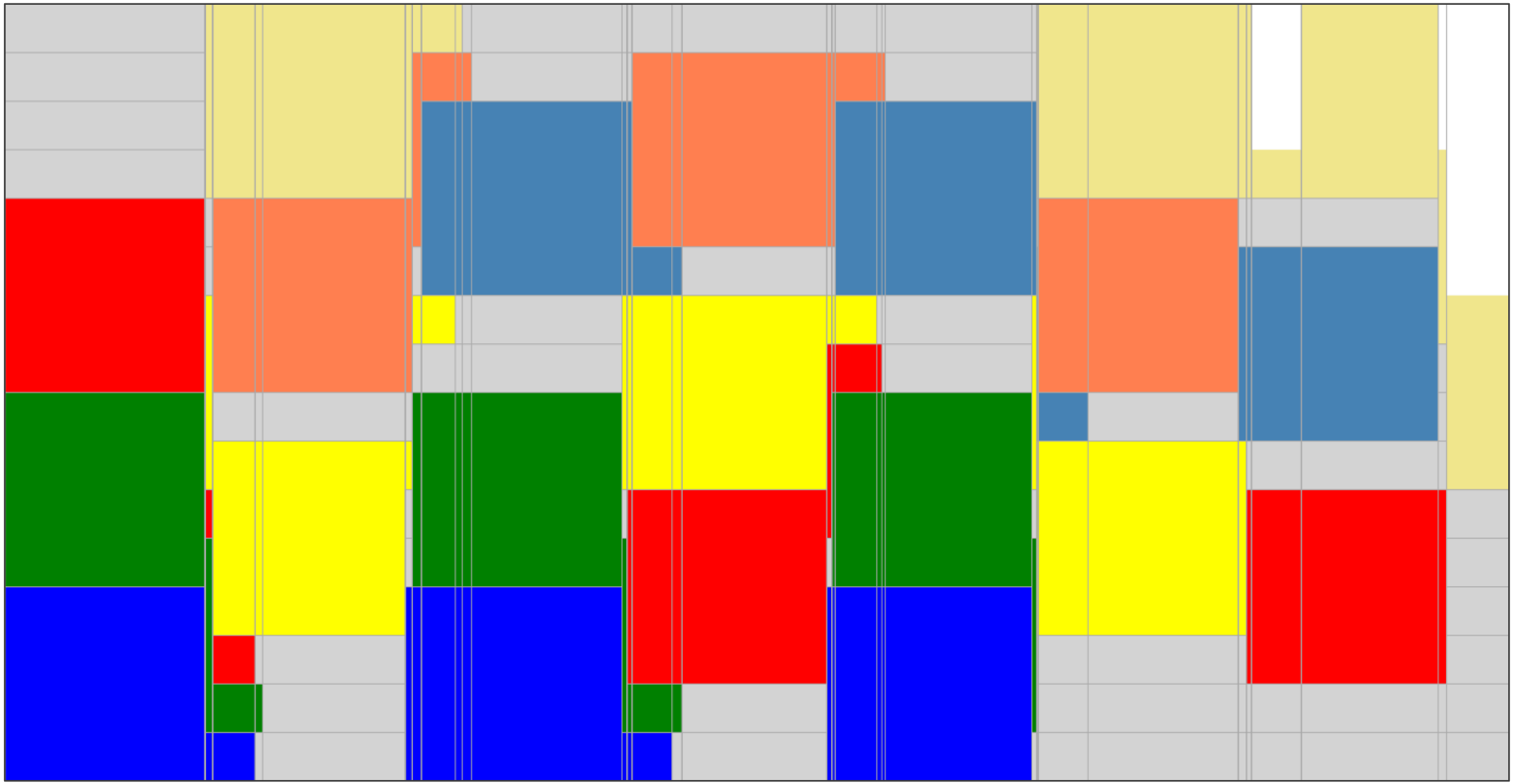


Распределение вычислений для 8 потоков на модельной задаче



8 потоков, 4 вычислительные процедуры с параллельной частью ширины 3

Распределение вычислений для 16 потоков на модельной задаче (Intel Xeon)

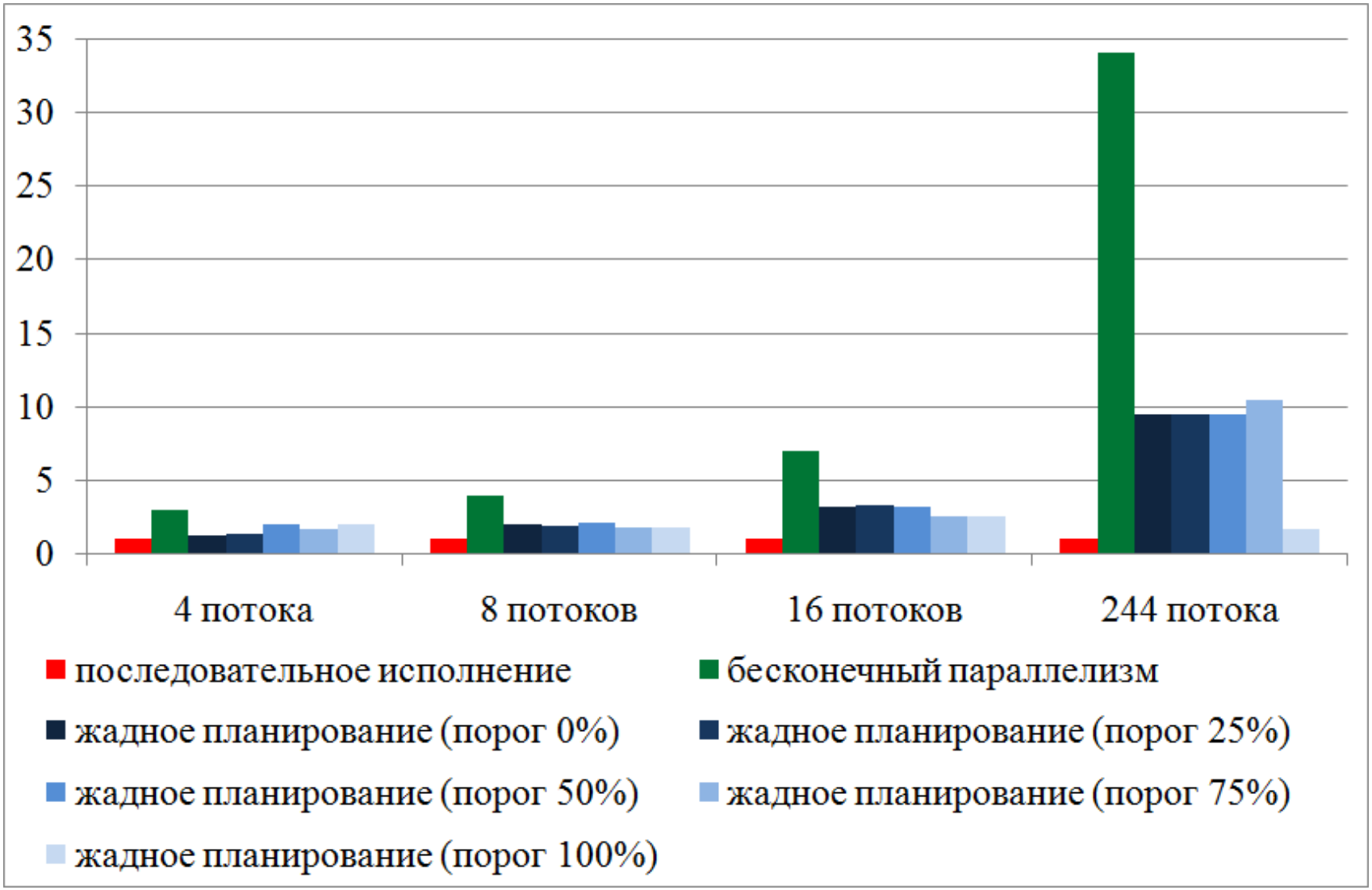


16 потоков, 7 вычислительных процедур с параллельной частью ширины 4

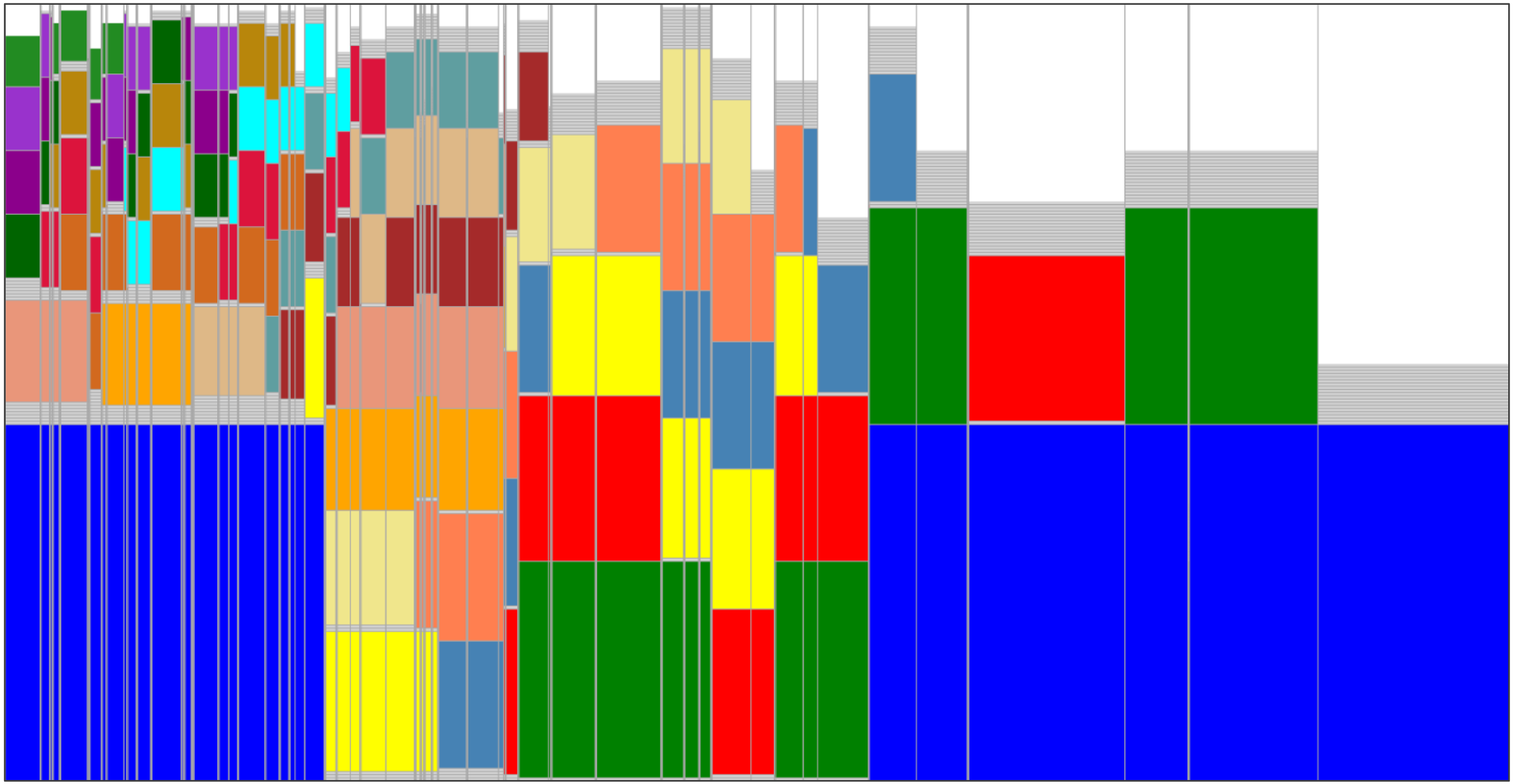
Распределение вычислений для 244 потоков на модельной задаче (Intel Xeon Phi)

244 потока, 34 вычислительные процедуры с параллельной частью ширины 21

Ускорение вычислений на модельных задачах

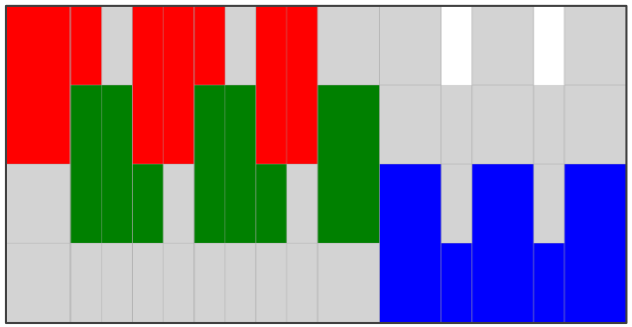


Распределение вычислений для 244 потоков на примере реальной сетки (Intel Xeon Phi, МВС-10П)

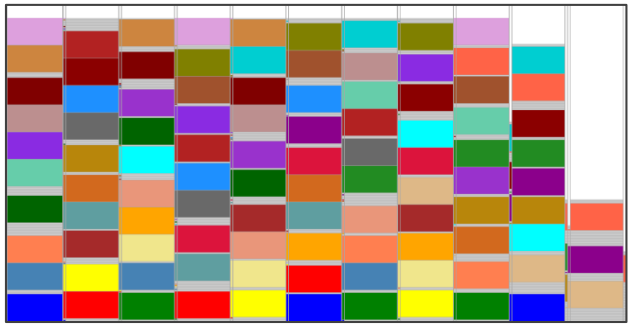


244 потока, 20 вычислительных процедур, присутствуют крупные процедуры

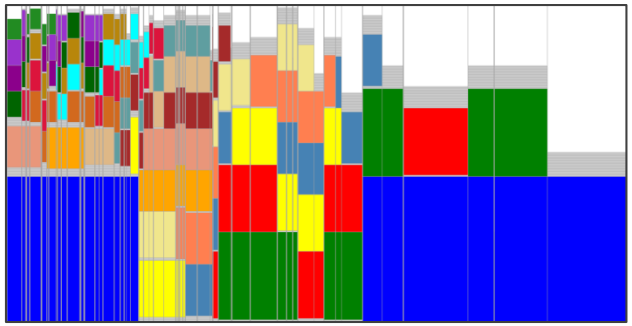
Недостатки метода



1. Из-за случайного захвата потоков вычислительными процедурами возможен простой ресурсов в конце исполнения.



2. При отсутствии порога снижения количества потоков исполнения возможен простой ресурсов из-за отсутствия мелких вычислительных процедур (неоптимальная укладка).



3. Отсутствие механизмов приоритетов приводит к вытеснению тяжелых вычислительных процедур.

Дальнейшие пути развития метода

1. Исследование зависимости времени общего исполнения от значений порогов снижения количества потоков исполнения для вычислительных процедур. Слишком сильное повышение порога может привести к началу исполнения атома со слишком низкой степенью параллельности, и длительным временем исполнения, что может привести к простою других потоков.
2. Разработка механизмов управления приоритетами, которых позволят наиболее тяжелым вычислительным процедурам, формирующим критический путь исполнения, не вытесняться более мелкими многочисленными процедурами.
3. Разработка механизмов циклического распределения нагрузки между вычислительными процедурами одного порядка (Round-robin).
4. Динамический сбор характеристик вычислительных процедур для более эффективного планирования исполнения.
5. Распараллеливание обработки списков блоков с помощью OpenMP для снижения простаивания ожидающих потоков.

Спасибо за внимание.