ИПМ им.М.В.Келдыша РАН • Электронная библиотека

Препринты ИПМ • Препринт № 102 за 2022 г.

ISSN 2071-2898 (Print)
ISSN 2071-2901 (Online)

М.А. Корнилина, М.В. Якобовский

Оценка накладных расходов при выполнении расчётов на локально измельчаемых сетках

Рекомендуемая форма библиографической ссылки: Корнилина М.А., Якобовский М.В. Оценка накладных расходов при выполнении расчётов на локально измельчаемых сетках // Препринты ИПМ им. М.В.Келдыша. 2022. № 102. 36 с. https://doi.org/10.20948/prepr-2022-102
https://library.keldysh.ru/preprint.asp?id=2022-102

О р д е н а Л е н и н а
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М.В. Келдыша
Р о с с и й с к о й а к а д е м и и н а у к

М.А. Корнилина, М.В. Якобовский

Оценка накладных расходов при выполнении расчётов на локально измельчаемых сетках

Москва — 2022

Корнилина М.А., Якобовский М.В.

Оценка накладных расходов при выполнении расчетов на локально измельчаемых сетках

Рассматриваются метод представления локально измельчаемых решёток и алгоритмы их обработки при выполнении в последовательных и параллельных вычислительных системах. Приводятся оценки зависимости числа выполняемых операций и уровня накладных расходов от таких параметров, как размер базовой регулярной сетки, максимальная глубина измельчения ячеек и число процессоров. Рассматриваются алгоритмы динамической балансировки нагрузки процессоров, обеспечивающие низкий уровень накладных расходов.

Ключевые слова: суперкомпьютер, балансировка нагрузки процессоров, локально измельчаемые расчётные сетки, ускорение вычислений

Marina Kornilina, Mikhail Yakobovskiy

Estimation of computational overheads for calculations on locally refined meshes

A method for presenting grid data and algorithms for their processing when performing calculations on locally refined adaptive grids is considered. Estimates are obtained of the dependence of the number of operations and computational overheads on such parameters as the size of the base regular grid, the depth of refinement, and the number of processors. Algorithms for dynamic load balancing of processors, which provide a low level of overhead costs, are considered.

Key words: supercomputer, load balancing, adaptive mesh refinement, speedup

Оглавление

Список переменных и ключевых понятий

n	число базовых ячеек, расположенных вдоль ребра решётки базовых ячеек
d	размерность пространства (2 или 3 для двух- и трёхмерного случая соответственно)
N	общее число базовых ячеек, $N = n^d$
p	общее число процессоров
B	число блоков базовых ячеек
A	общее число активных элементарных ячеек
L	общее число активных элементарных ячеек, примыкающих к границам зон измельчения
A_i	число активных элементарных ячеек уровня измельчения i
m	максимальный уровень измельчения базовой ячейки
q	ширина шаблона по направлению
b	ширина буферных зон
LRM	Local Mesh Refinement – локально измельчаемые сетки
Блок ячеек	прямоугольный (в двумерном случае) или параллелепипедный (в трёхмерном случае) фрагмент решётки ячеек
Домен	фрагмент расчётной сетки, обрабатываемый одним процессором или графическим ускорителем

Введение

Характеристики суперкомпьютеров мира списка top500 [1] подтверждают хорошо известный тезис о том, что основной эксплуатируемый сегодня ресурс увеличения производительности экстенсивен. Он базируется на механическом увеличении числа одновременно работающих вычислительных устройств. На рисунке 1 для каждой из пятисот систем списка указано число обеспечивающих её производительность процессорных ядер. Подавляющее большинство систем имеет в своём составе тридцать и более тысяч вычислительных ядер.

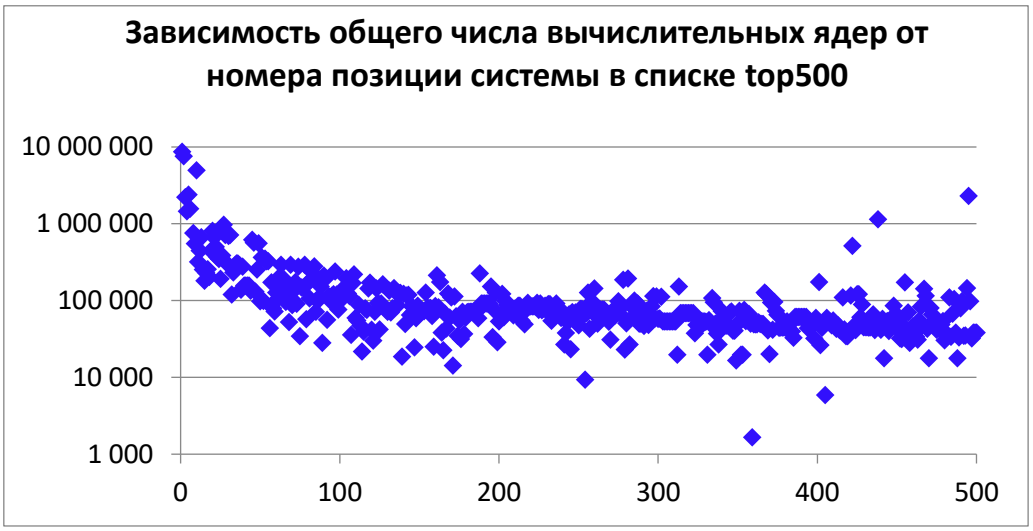


Рис. 1. Число вычислительных ядер суперкомпьютеров top-500, ноябрь 2022

Число вычислительных ядер в каждой из первых пятидесяти систем превышает 130 тысяч. Данное обстоятельство следует учитывать при разработке суперкомпьютерных приложений. Предел масштабируемости алгоритмов, неформально определённый как наибольшее число эффективно параллельно работающих, не мешающих друг другу процессов, должен быть не меньше числа исполнительных устройств. Максимально достижимое за счёт параллельной обработки ускорение S_p можно оценить с помощью соотношения (1).

$$S_p \leq \frac{T_1}{\alpha T_1 + (1+\alpha)\frac{T_1}{p} + t_н(p)} \qquad (1)$$

Здесь T_1 – время выполнения последовательного алгоритма одним процессом, α – доля операций алгоритма, выполняемых исключительно последовательно, p – число процессов, $t_н(p)$ – дополнительные, относительно последовательного алгоритма, расходы времени при выполнении программы на p процессах. В рамках концепции неограниченного параллелизма (при $p \to \infty$)

и в предположении нулевого уровня дополнительных расходов $t_{\text{н}}(p) = 0$, оценка ускорения сверху даётся законом Амдаля (2).

$$S_p \leq \frac{1}{\alpha} \tag{2}$$

Для достижения ускорения, по порядку величины совпадающего с общим числом процессов, следует обеспечить выполнение условия $\alpha < \frac{1}{p}$. Поскольку даже для среднего сегмента списка top500 число процессов превышает 10^4, доля нераспараллеливаемых операций не должна превышать 10^{-4}, и это без учёта дополнительных накладных расходов.

Полученная оценка обосновывает точность, с которой, при выполнении расчётов на суперкомпьютерах, должна быть решена задача равномерного распределения по процессам полезных вычислений. Задача сводится к необходимости выполнить рациональную декомпозицию взвешенной расчётной сетки [12] на домены, относительные веса которых определены производительностями процессоров и ускорителей. При сделанных предположениях веса сформированных доменов не должны отличаться от идеального случая более чем на десятую долю процента. Это жёсткое требование сложно выполнить, но на практике оно ослаблено благодаря тому, что основная задача суперкомпьютеров – решение больших задач. Во многих практически значимых случаях число элементов обрабатываемой на суперкомпьютере расчётной сетки настолько велико, что не позволяет выполнить требуемые вычисления за разумное время на одном процессорном узле. Как следствие, за опорное время T_1 принимают не время выполнения программы на одном процессе (процессорном ядре), а время выполнения параллельной программы на нескольких вычислительных узлах суперкомпьютера.

К примеру, на второй строчке списка top500 [1] располагается суперкомпьютер Fugaku. Общее число процессорных ядер вычислительного поля этого суперкомпьютера равно 7 630 848. Поскольку суперкомпьютер укомплектован 48-ядерными процессорами A64FX 48C, общее число процессоров почти на два порядка меньше, чем число процессорных ядер и составляет 158 976. Для среднего сегмента списка разница между общим числом ядер и числом вычислительных узлов ещё больше, поскольку каждый используемый в этих системах ускоритель содержит сотни мультитредовых устройств. Тем не менее формально требуемая точность решения задачи декомпозиции расчётной сетки остаётся на уровне 10^{-3}, поэтому указанное ослабление требований не отменяет необходимости разработки алгоритмов, эффективно использующих каждое из вычислительных ядер, будь то процессорное ядро или мультитредовый ускоритель графической карты.

В общем случае задача равномерного распределения по доменам весов вершин при одновременной минимизации числа рёбер расчётной сетки,

соединяющих вершины разных доменов, принадлежит классу NP. Поскольку точно решать подобные задачи требуемого размера за разумное время не представляется возможным, для декомпозиции расчётных сеток применяются приближённые методы. Сужение класса рассматриваемых расчётных сеток позволяет повысить качество работы приближённых методов за счёт учёта априорной информации о свойствах сеток выделенных классов.

Препринт посвящен обсуждению способов организации расчётов при использовании локально измельчаемых динамически адаптивных расчётных сеток. Поскольку речь идёт о динамической адаптации, решать задачу декомпозиции [12] необходимо многократно, по мере существенного изменения фактического распределения нагрузки, вызванного актами адаптации сетки (соответствующие вопросы подробнее рассмотрены в разделе «Балансировка нагрузки процессоров»). От качества её решения и от времени, требуемого на её решение, ключевым образом зависит общая эффективность выполнения расчётов. Доступные методы, основанные на иерархических алгоритмах декомпозиции, например [14], [15], [17], или метод геометрической декомпозиции [12] не обеспечивают устойчивости декомпозиции. Под устойчивостью понимается свойство преимущественного сохранения места размещения данных на процессорах – минимизации объёма данных, перемещаемых между процессорами на этапе перехода от одной адаптивной сетки к другой. Устойчивость обеспечивают, например, методы, основанные на равномерно заполняющих многомерное пространство фрактальных кривых [16]. Однако соответствующие им алгоритмы обладают низким запасом внутреннего параллелизма, требуя б***о***льшего времени на декомпозицию относительно метода, рассматриваемого в разделе «Послойный алгоритм декомпозиции множества базовых ячеек».

Максимальное достижимое ускорение вычислений (1) существенно снижается ввиду наличия накладных расходов $t_{\text{н}}(p)$. Дополнительные потери времени обусловлены множеством причин, в том числе: передачей данных между процессами, простоями процессов вследствие дисбаланса нагрузки – неравномерности распределения вычислений по процессам; дополнительными расходами на общее управление вычислительным процессом; дополнительными расходами на обработку нерегулярной сетки. Предыдущие два абзаца являются содержательным примером таких дополнительных накладных расходов: в общее время $t_{\text{н}}(p)$ естественным образом включаются времена, требуемые:

– для формирования новой адаптивной сетки и переноса/интерполяции данных с используемой ранее расчётной сетки на вновь сформированную;

– для решения задачи декомпозиции расчётной сетки;

– для перераспределения данных между процессорами при переходе между расчётными сетками до и после очередной адаптации.

Указанный список исчерпывающим не является, но уже он обращает внимание на необходимость разработки параллельных алгоритмов, эффективно решающих перечисленные задачи.

Рассматриваемый класс расчётных сеток

Широко распространённым способом, используемым при численном моделировании физических процессов с помощью метода конечных разностей [2], является метод регулярных сеток. В соответствии с ним вся расчётная область покрывается регулярной, например декартовой, индексной сеткой простейшей формы (прямоугольник или параллелепипед). К преимуществам метода регулярных сеток относятся: простота генерации сетки; относительная простота и экономичность разностной схемы; высокая скорость выполнения операций за счёт хорошей регулярности доступа к ячейкам оперативной памяти и, как следствие, потенциальная возможность эффективного использования кеш-памяти процессоров, векторизации вычислений и графических ускорителей.

Недостатком использования регулярных решёток является низкая точность описания границ расчётной области или особенностей решения, поскольку декартовая решётка не согласована с криволинейными границами и не согласована с характерными особенностями моделируемых физических полей, например, с контактными границами или ударными волнами. Точность может быть повышена за счёт использования сеток более высокого разрешения, но в этом случае общее число ячеек расчётной сетки, а значит и общее время вычислений, неоправданно возрастает, поскольку регулярная сетка покрывает одинаковыми мелкими ячейками всю область расчёта, а не только фрагменты области, в которых грубая сетка даёт неудовлетворительную точность. Следует отметить и иную альтернативу, а именно метод отображений [19], обеспечивающий для определённых классов задач высокое качество согласования регулярной сетки с границами расчётной области и особенностями решения, но оценка общности применения и трудоёмкости алгоритмов, соответствующих этому методу, выходит за рамки препринта.

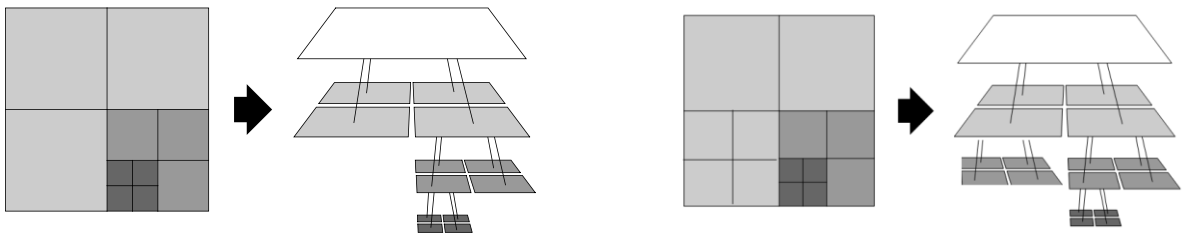
Экономичной альтернативой методу регулярных сеток является метод локально измельчаемых сеток, статически или динамически адаптивных к границам области и к особенностям решения. Некоторые вопросы представления декартовых локально сгущающихся расчётных сеток и алгоритмы выполнения на них расчётов рассмотрены в работах [4], [5], [6], [7]. Доступно описание известных пакетов прикладных программ для работы с декартовыми локально сгущающимися сетками: Chombo [8], p4est [9], AMReX [21] и ряд других. Некоторые проблемы эффективного доступа к памяти и способы их решения описаны в [13].

С одной стороны, использование адаптивных сеток обеспечивает возможность сокращения времени выполнения расчётов за счёт уменьшения общего числа обрабатываемых ячеек расчётной сетки. С другой стороны,

использование адаптивных сеток сопряжено с дополнительными расходами, в том числе обусловленных усложнением вычислительных процедур, необходимостью формирования и обслуживания адаптивных сеток, дополнительными затратами на балансировку нагрузки процессоров. Несмотря на интенсивное использование локально измельчаемых динамически адаптивных сеток, остаются открытыми вопросы оценки фактически достигаемого при их использовании выигрыша как при выполнении расчётов на однопроцессорных системах, так и при использовании многопроцессорных кластерных систем и систем, оснащённых графическими ускорителями.

Далее рассматривается комплексный подход, направленный на повышение эффективности использования адаптивных локально измельчаемых декартовых сеток, и предлагается анализ накладных расходов, снижающих скорость вычисления относительно «идеального» сценария, при котором время вычислений определяется исключительно содержательными операциями над сеточными переменными, описывающими моделируемые процессы.

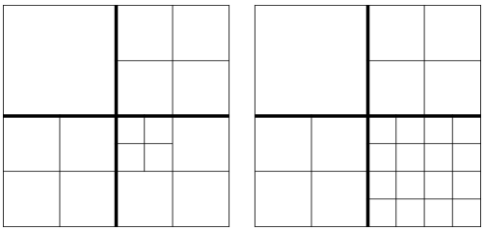
Общий принцип адаптации при использовании локально измельчаемых сеток иллюстрируется рисунком 2а. В соответствии с ним при необходимости детального описания некоторого фрагмента расчётной области описывающие его ячейки делятся на 4 части (в двумерном случае) или на 8 частей (в трёхмерном случае). Некоторые из полученных в результате деления ячеек могут быть подвергнуты дальнейшему измельчению.



а б

Рис. 2. Локальное измельчение ячейки

Ряд методик накладывает дополнительные ограничения на вид измельчения, например, запрещая непосредственное соседство ячеек с линейными размерами, отличающимися более чем в два раза (рис. 2б). Или ещё более строгие ограничения, согласно которым между ячейками уровней измельчения $(level - 1)$ и $(level + 1)$ требуется наличие буферной зоны шириной не менее b ячеек уровня $level$. Последнее требование оправданно и полезно с двух точек зрения. Во-первых, соблюдение подобного условия повышает качество численного решения. Во-вторых, достаточно широкие промежуточные слои позволяют реже выполнять динамическую адаптацию, что не только повышает качество решения (за счёт меньшего общего числа операций интерполяции), но и, при определённых условиях, сокращает общее время расчёта.



а б

Рис. 3. Листовое (а) и однородное (б) измельчение базовых ячеек

Широко используются две основные стратегии измельчения ячеек исходной грубой сетки – базовых ячеек: листовой [3], [9], [10] и блочно-ориентированный [8]. При листовом подходе ячейки измельчаются в соответствии с заданными критериями и, в общем случае, области измельчения одного уровня могут иметь произвольную форму (рис. 3а). В рамках блочно-ориентированного подхода области одного уровня измельчения имеют прямоугольную (параллелепипедную в трёхмерном случае) форму (рис. 3б).

Листовой подход обеспечивает наименьшее общее число ячеек благодаря детальному измельчению, диктуемому исключительно действующим критерием адаптации к границам области и к особенностям решения (физическому критерию) и наложенным дополнительным условиям на соседство ячеек разного уровня измельчения (структурному критерию).

В противоположность ему, блочно-ориентированный подход ограничивает многообразие форм зон измельчения. Как правило, блоки одного уровня измельчения $level$ не пересекаются между собой и геометрически вложены в области, покрытые блоками измельчения уровня $level - 1$. Эти два принципа в общем случае не требуют согласования формируемых зон с границами исходной грубой сетки и оставляют значительную свободу формирования блоков, но, как следствие, вызывают необходимость решать задачу распознавания форм зон измельчения и вычисления координат, покрывающих эти зоны прямоугольников или параллелепипедов.

В рассматриваемом далее подходе наложено дополнительное условие, согласно которому формируемые области измельчения согласованы с ячейками исходной базовой сетки. В соответствии с этим принципом каждая из базовых ячеек измельчается однородно (рис. 3б). Равномерное измельчение базовых ячеек приводит к формированию б***о***льшего общего числа ячеек, но время на обработку дополнительно возникающих ячеек ниже, чем экономия за счёт повышения регулярности обработки. В Таблице 1 приведены данные, показывающие относительное увеличение общего числа ячеек при адаптивном измельчении сетки, описывающей положение кольцевого слоя в квадратной решётке размера n^2 или сферического слоя на кубической решётке размера n^3, выраженное в процентах (двумерный пример с одним уровнем измельчения приведён на рисунке 12). Из таблицы следует, что при адаптивном равномерном измельчении до пятого уровня сетки, содержащей 400^3 базовых

ячеек, в трёхмерном случае формируется всего на 0.74% больше ячеек, чем при адаптивном листовом измельчении.

К недостаткам листового метода следует отнести высокую вычислительную сложность обработки листовых ячеек древовидных структур, неизбежно возникающих при детальном дроблении базовых ячеек. Она выражается, например: – в высокой трудоёмкости операций определения положения элементов, входящих в вычислительный шаблон; – в существенной неоднородности обращений к оперативной памяти. Соответствующие затраты при равномерном измельчении практически отсутствуют.

Таблица 1
Относительное увеличение числа листовых ячеек
за счёт однородного измельчения базовых ячеек (%)

2D					n	3D				
1	2	3	4	5		1	2	3	4	5
0.57	0.80	1.10	0.85	1.10	100	1.40	2.40	2.80	2.90	3.00
0.28	0.49	0.55	0.52	0.52	200	0.70	1.20	1.40	1.40	1.40
0.17	0.32	0.39	0.35	0.44	300	0.47	0.83	0.95	0.98	0.99
0.13	0.24	0.28	0.30	0.33	400	0.35	0.62	0.71	0.73	0.74

Дальнейшее рассмотрение будет посвящено именно блочно-ориентированному подходу. Наиболее близким к рассматриваемому методу является плиточный (tile-based) подход [11], в рамках которого сетка разбивается на блоки одинакового размера. В отличие от него, в предлагаемом подходе фиксируется размер блоков по всем геометрическим направлениям, кроме одного из них. В силу вытянутости блоков вдоль одного выделенного направления будем в дальнейшем называть этот метод *блочным линейно-ориентированным*.

Алгоритм расчёта при использовании регулярных декартовых решёток

Рассмотрим схематичное описание трёх последовательных алгоритмов: неадаптивного; статически адаптивного; динамически адаптивного. Неадаптивный алгоритм потребуется для дальнейшей оценки относительного уровня накладных расходов при использовании локально измельчаемых динамически адаптивных сеток. Статически адаптивный алгоритм потребуется для целей оценки свойств динамически адаптивного алгоритма. Описываемые алгоритмы применимы как для двумерного, так и для трехмерного случая, но приводимые дальше наглядные иллюстрации соответствуют двумерному случаю.

```
sloy0 = 0; sloy1 = 1 – sloy0;

for( i_step=1; i_step < n_steps ; i_step ++ )
{ // Цикл шагов расчёта
  // длительности расчета ячеек одинаковы
  for( i=1; i < n ; i ++ )
  for( j=1; j < n ; j ++ )
    U[sloy1][i][j] = fun1( U[sloy0][i-2...i+2][j],
                           U[sloy0][i][j-2...j+2]);

  // длительности расчета ячеек различны
  for( i=0; i < n ; i ++ )
  for( j=0; j < n ; j ++ )
    if(hard(U[sloy1][i][j]))
        U[sloy1][i][j] = fun2( U[sloy1][i][j] );

  sloy0 = sloy1; sloy1 = 1 – sloy0;
}
```

Алг. 1. Последовательный алгоритм для расчёта на регулярной неадаптивной сетке

Здесь: `n` – число ячеек стороны квадратной решётки расчётной сетки; `n_steps` – число шагов модельного времени; `U[sloy0]`, `U[sloy1]` – векторы сеточных переменных (таких как плотность, концентрации веществ), соответствующие ячейкам`[i][j]` текущего и следующего шага модельного времени, `hard(U[sloy1][i][j])` – бинарная функция, определяющая необходимость дополнительной обработки соответствующей ячейки.

Пусть время выполнения функции `fun1` не зависит ни от номера обрабатываемой ячейки, ни от значения сеточных функций и равно τ_c. Подобная ситуация характерна для выполнения расчётов, например, по явным разностным схемам.

Пусть время выполнения функции `fun2` зависит от значений сеточных функций и равно $\tau_{d,i,j}$. Подобная ситуация складывается, например, при моделировании фронтов горения с помощью методик, предполагающих решение в каждой из ячеек жёстких систем обыкновенных дифференциальных уравнений [18]. В качестве второго примера можно указать определение термодинамических параметров в ячейках, содержащих несколько веществ многокомпонентного течения. Решение жёстких систем ОДУ или нелинейных систем для их определения сопряжено с использованием итерационных процессов, число итераций в которых может сильно и непредсказуемо меняться

в зависимости от числа компонент в ячейке и конкретных значений других сеточных переменных. Общее время выполнения одного шага алгоритма 1 можно оценить согласно (3).

$$T_1 = \tau_c n^2 + \sum_{\text{i,j: hard(U[sloy1][i][j])==1}} \tau_{d,i,j} \qquad (3)$$

Время T_1 в дальнейшем рассматривается как наилучшее. Относительно него оценивается уровень накладных расходов. Оценка числа ячеек (9), для которых `hard(U[sloy1][i][j])==1`, обсуждается на странице 22.

В рамках данной работы предлагаемые оценки времени выполнения фактически являются оценками не *времени выполнения*, а *числа выполняемых укрупнённых операций*. В этих оценках не принимается во внимание влияние на скорость выполнения вычислений кеш-памяти процессора, возможности спекулятивного выполнения и другие программные и аппаратные особенности современных процессоров, процессорных узлов и компиляторов. Аналитическая оценка влияния этих факторов крайне затруднена. Их более-менее достоверный учёт возможен на основе выполнения объёмных апостериорных экспериментальных исследований. Целью же настоящей работы является выбор и обоснование перспективной технологии на основе априорных аналитических оценок. Тем не менее рассматриваемые методы косвенно ориентированы на эффективное использование существенных архитектурных особенностей современных суперкомпьютеров, в том числе на эффективное использование кеш-памяти процессоров и векторизации вычислений.

Общее описание линейно-ориентированного метода

Основные структуры данных

Прежде чем переходить к описанию алгоритмов работы с локально измельчаемыми сетками, рассмотрим основные применяемые структуры данных. Сокращённое описание их основных полей приведено в Приложении 1.

Базовая ячейка – ячейка исходной сетки, равномерно разбиваемая на элементарные ячейки одинакового размера. Это исключительно логическое понятие. Соответствующий тип данных отсутствует, поскольку в оперативной памяти матрица базовых ячеек не хранится. Тем не менее понятие базовой ячейки является ключевым, оно используется при описании алгоритмов. Базовые ячейки не хранятся, но хранится описание их групп – блоков базовых ячеек LRM_BASE_BLOCK.

Блок базовых ячеек LRM_BASE_BLOCK – прямоугольный блок базовых ячеек одинакового уровня измельчения. Блок базовых ячеек задаётся путём указания индексов начала блока по каждому из направлений и числа ячеек в блоке по каждому из направлений.

Элементарная ячейка VCELL – ячейка, описывающая один элемент физического объёма. Именно эта структура содержит «физические» величины. Линейные размеры элементарной ячейки определяются уровнем измельчения соответствующей базовой ячейки. Эти ячейки логически соответствуют «листовым» элементам расчётной сетки, используемой в рамках листового подхода.

Блок элементарных ячеек LRM_VC_BLOCK – прямоугольный блок элементарных ячеек одного уровня измельчения. Некоторые блоки элементарных ячеек описывают внутренние зоны расчётной области, некоторые прилегают к её физическим границам, некоторые соответствуют теневым граням, а некоторые могут не участвовать в расчёте, поскольку описывают фрагменты, покрытые ячейкам иных уровней измельчения (рис. 10).

Задание LRM_TASK – одно задание на выполнение вычислений или дополнительных действий (инициализация, копирование, интерполяция, вычисление, пересылка и тому подобное). Именно задание является единицей планирования работы на уровне MPI или графических ускорителей. На уровне OpenMP или posix каждое задание может одновременно обрабатываться несколькими процессорными ядрами.

Блок ячеек задания LRM_TASK_BLOCK – блок элементарных ячеек одного уровня измельчения, описывающий область ячеек источника или приёмника данных для задания LRM_TASK. Эти блоки геометрически вложены в блоки элементарных ячеек.

Блоки базовых и элементарных ячеек

Единицами описания фрагментов расчётной области являются блоки базовых ячеек и блоки элементарных ячеек. Единицами планирования и обработки являются задания. Каждое задание содержит сведения о требуемом действии, целевом блоке элементарных ячеек и, при необходимости, о блоке ячеек источника данных. Область источника данных может отсутствовать, например, при инициализации исходных данных. Она может быть вычислена автоматически на основе используемого шаблона, например, при выполнении шага расчёта по явной разностной схеме. Однако в ряде случаев сведения об источнике данных целесообразно вычислить однократно (при формировании заданий) и в явном виде включить в задания. В качестве характерного примера можно указать задания интерполяции данных на внутренних границах между блоками базовых ячеек с разными уровнями измельчения. Такое решение дополнительно сокращает общее время выполнения расчёта, что является основной целью рассматриваемого подхода.

Рассмотрим пример расположения блоков базовых и элементарных ячеек для случая базовой сетки, содержащей $N = 16 = 4 \cdot 4$ базовых ячеек (рис. 6). Для определённости предположим, что используемый для выполнения расчётов шаблон разностной сетки является локальным и захватывает по каждому направлению не более двух соседних ячеек (рис. 5).

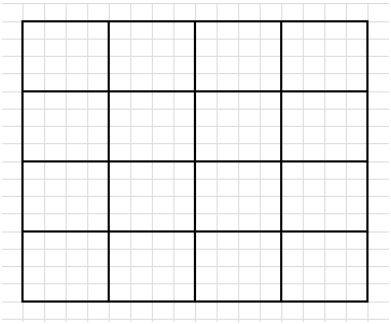


Рис. 4. Исходная сетка базовых ячеек

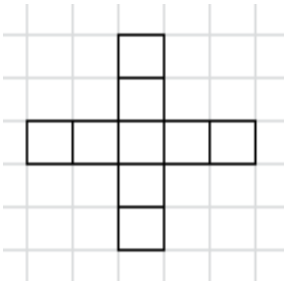


Рис. 5. Вид шаблона, $q = 2$

Алгоритм формирования адаптивной сетки предполагает выполнение следующих шагов:

1. вычисление уровней измельчения базовых ячеек;
2. распределение базовых ячеек по блокам базовых ячеек;
3. распределение блоков базовых ячеек по блокам элементарных ячеек;
4. указание расположения блоков базовых ячеек в блоках элементарных ячеек.

На рисунке 6 представлены базовые ячейки двух уровней измельчения – первого и второго.

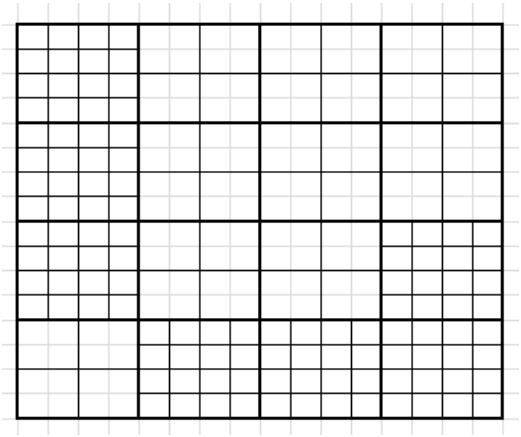


Рис. 6. Пример измельчения ячеек

Пусть базовые ячейки распределены по блокам базовых ячеек в соответствии с рисунком 7. Предлагается использовать именно такой, легко автоматизируемый принцип разбиения, в рамках которого все блоки имеют высоту 1 и выделяются последовательно по полосам. Данный принцип позволяет существенно сократить накладные расходы в момент формирования заданий, поскольку в его рамках:

- тривиальным образом с трудоёмкостью $O(N)$ решается задача распознавания множества прямоугольников, покрывающих заданные области измельчения;

- алгоритм определения соседства базовых блоков, используемый при формировании заданий, обладает низкой, не превышающей $O(n)$, оценкой числа операций.

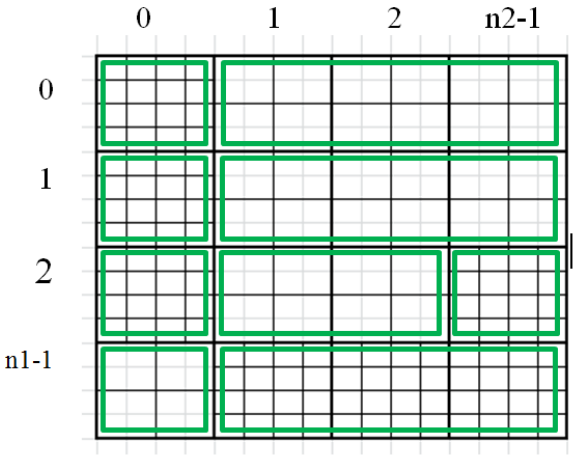


Рис. 7. Блоки базовых ячеек

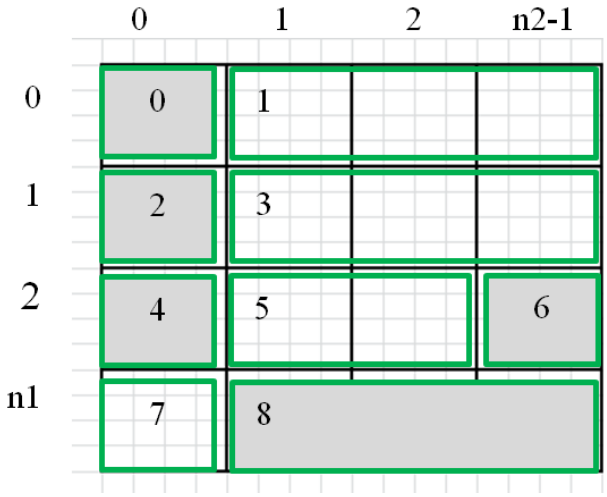


Рис. 8. Номера блоков базовых ячеек и уровни измельчения

Всего на рисунке 7 показано 9 блоков базовых ячеек. Их номера указаны на рисунке 8, там же серым цветом отмечены блоки с уровнем измельчения, равным 2. Каждая базовая ячейка такого уровня разбита на 16 элементарных ячеек. Остальные базовые ячейки тоже имеют одинаковый уровень измельчения, равный единице, – каждая из них разбита на 4 элементарные ячейки.

Одно из возможных распределений девяти блоков базовых ячеек на группы, определяющих выделение памяти элементарных ячеек, приведено на рисунке 9. Для всех блоков базовых ячеек уровня 1 (блоки базовых ячеек с номерами 1, 3, 5 и 7) выделен один блок элементарных ячеек, его номер 0. Этот блок, как и следующие два, показан красным цветом. Для блоков базовых ячеек с номерами 0, 2 и 4 выделен блок элементарных ячеек с номером 1. Для блоков базовых ячеек с номерами 6 и 8 выделен блок элементарных ячеек с номером 2.

Рисунок 10 иллюстрирует состав блока элементарных ячеек с номером 2. В этом блоке хранятся элементарные ячейки уровня измельчения 2 (номер блока и уровень измельчения совпали случайно). Кроме активных элементарных ячеек, соответствующих моделируемой области, измельчённой до соответствующего уровня, распределена память для дополнительных граничных ячеек, для внутренних теневых граней и для неиспользуемых элементарных ячеек. Элементарные ячейки белого цвета (рис. 10) являются активными. Значения соответствующих им сеточных функций вычисляются при обработке блоков базовых ячеек с номерами 6 и 8. Значения в элементарных ячейках, отмеченных фиолетовым цветом, будут вычислены при

измельчении элементарных ячеек, вычисленных при обработке блока 5 базовых ячеек, хранящихся в блоке 0 физических ячеек (уровня измельчения 1).

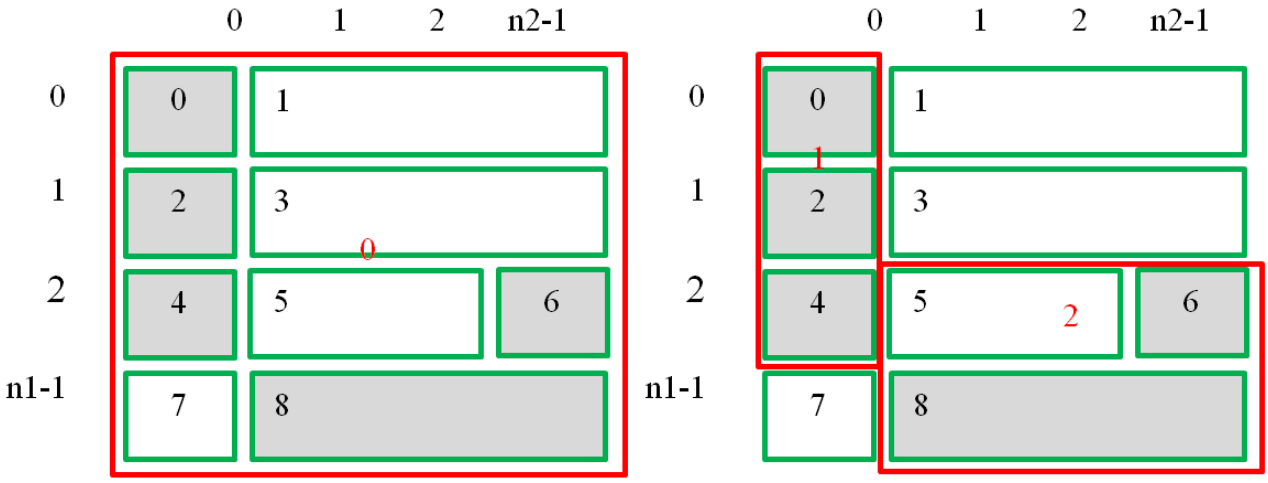


Рис. 9. Блоки элементарных ячеек уровня 1 (слева) и 2 (справа)

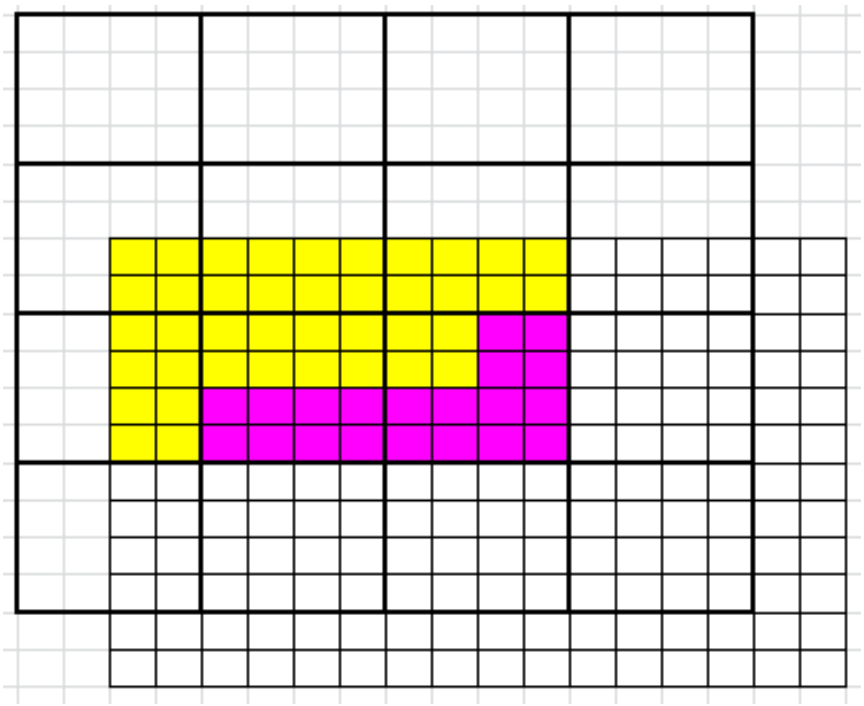


Рис. 10. Блок номер 2 элементарных ячеек

Таким образом, область данных, выделенная для ячеек фиолетового цвета, будет использована при расчёте величин в граничных ячейках 6-го и 8-го блоков базовых ячеек, в соответствии с видом и размером шаблона (рис. 5). Элементарные ячейки, отмеченные жёлтым цветом, не являются активными и не будут обработаны. Они не будут использованы в расчёте, но избыточное выделение оперативной памяти оправданно, поскольку существенно упрощает индексацию и снижает накладные расходы, сокращая число операций

копирования данных. Например, в данном конкретном случае при выполнении вычислений не потребуется дополнительного копирования данных между блоками базовых ячеек с номерами 6 и 8.

Алгоритм расчёта на статически адаптивной локально измельчаемой сетке

Рассмотрим основные этапы алгоритма выполнения расчёта на статически адаптивной локально измельчаемой сетке (алг. 2).

```
1. Инициализация расчётной сетки
2. Формирование списка заданий (spisok_init) для
   заполнения элементарных ячеек начальными значениями
   физических переменных
3. Выполнение заданий списка
   do_tasks(spisok_init)
4. Формирование списка заданий (spisok_step) для
   выполнения одного шага расчёта
5. Выполнение шагов расчёта
   for( i_step=1; i_step < n_steps ; i_step ++ )
   {
    do_tasks(spisok_step); // выполнение заданий шага
    sloy0 = sloy1;         // смена номера
    sloy1 = 1 - sloy0;     // активного слоя
   }
```

Алг. 2. Статически адаптивный последовательный алгоритм

В описание задания входят название выполняемой функции (алг. 3) и фактические значения параметров `VC_FUN_HEAD_PARAM` (приложение 1), в том числе координаты и размеры целевого блока и блока источника элементарных ячеек.

```
void vc_fun_calc_inner (VC_FUN_HEAD_PARAM)
{
  sloy_tar = mesh.sloy; sloy_src = 1 - sloy_tar;

  vc_tar = mesh.vc_block[id_tar_vc_block].vc[sloy_tar];
  vc_src = mesh.vc_block[id_src_vc_block].vc[sloy_src];

  for( i=i_tar_off; i<i_n ;i++)
```

```
  for( j=j_tar_off; j<j_n ;j++)
  {
     vc_tar.U[i][j] = fun1(    vc_src.U[i-2...i+2][j],
                               vc_src.U[i][j-2...j+2]    );
  }
}
```

Алг. 3. Пример функции задания

Общее число блоков базовых ячеек B зависит от особенностей решаемой задачи. В худшем случае $B = O(N)$. Однако для характерного примера – системы колец (рис. 11) – число блоков базовых ячеек в строке не превышает $2m - 1$, поскольку в этом примере присутствует не более одной зоны максимального измельчения в любой строке решётки. Соответственно, в этом и во многих других практически значимых случаях справедлива оценка (4).

$$B = O\left(m\frac{N}{n}\right) \tag{4}$$

```
for( b=0; b< B; b++ ) // цикл по всем базовым блокам
{
  Формирование задания обработки внутренних ячеек();
  Формирование заданий обработки границ базового блока();
}
```

Алг. 4. Формирование списков заданий

Список заданий содержит задания, обрабатывающие внутренние ячейки, и задания, обрабатывающие граничные ячейки (алг. 4). Трудоёмкость формирования заданий первого типа оценивается как $O(B)$. Число заданий второго типа зависит от числа блоков базовых ячеек, соседних с каждым из них. В худшем случае блок, занимающий всю строку решётки, будет соседствовать с n блоками верхней строки и n блоками нижней. Соответствующее число границ, примыкающих к этому блоку, можно оценить как $2 + 2(d - 1)n$. Однако на практике это число ограничено константой. В самом трудоёмком случае число заданий, обрабатывающих границы, не может превышать число граней базовых ячеек, равное $6N$. Таким образом, трудоёмкость формирования заданий обработки границ блоков даже в худшем случае оценивается как $O(N)$, а на практике как $O\left(\frac{N}{n}\right)$.

Для соблюдения указанных оценок трудоёмкости необходимо обеспечить возможность определения соседей каждого из базовых блоков за время $O(1)$. Данное условие выполняется благодаря наличию поля `id_bb_ph` элементарных ячеек (Приложение 1). Его значение соответствует номеру блока

базовых ячеек (достаточно установить значения `id_bb_ph` только в граничных элементах блока элементарных ячеек). Использование этого поля в совокупности с тем фактом, что блоки пронумерованы сплошной нумерацией в пределах каждой строки решётки, обеспечивает возможность получения номера очередного соседнего блока за время $O(1)$.

Оценка времени выполнения

Пусть $level_i$ – уровень измельчения блока i базовых ячеек; k_i – число базовых ячеек в блоке с номером i (с учётом того, что блоки имеют единичный размер по всем направлениям, кроме одного, число базовых ячеек в блоке равно его длине). Тогда общее число активных элементарных ячеек A определяется выражением (5).

$$A = \sum_{i}^{B} \left((k_i \cdot 2^{level_i} + 2q)(2^{level_i} + 2q)^{d-1} - 2^d q^d \right) \qquad (5)$$

Пусть L – число активных элементарных ячеек, примыкающих к границам между зонами разного уровня измельчения, полученное исходя из длины границ, степени измельчения и ширины вычислительного шаблона (рис. 5). Именно для этих ячеек потребуется выполнять дополнительные операции по интерполяции данных. Знание величин A и L позволит оценить уровень накладных расходов, связанных с использованием локально измельчаемых сеток. Выражение (5), в силу громоздкости, мало что даёт для содержательного анализа. Аналитическое выражение для значений L в общем случае имеет ещё более неудобный вид. В связи с этим дальнейший анализ выполняется на примере упрощённой постановки задачи и опирается на аналитические и численные результаты, полученные для квадратной сетки базовых ячеек (9).

Пусть T_{reg} – время выполнения расчёта на регулярной сетке, соответствующей максимальному уровню измельчения m, T_{LMR} – время выполнения расчёта на LRM сетке, τ_c – время выполнения функции `fun1` (алг. 1, 2), τ_{int} – время интерполяции данных между ячейками разных уровней измельчения, τ_{dif} – среднее время выполнения функции `fun2` (алг. 1, 2). Выполнение функции `fun2` обусловлено, например, необходимостью согласования термодинамических переменных после шага интерполяции консервативных переменных. Второе слагаемое в оценке (6) времени выполнения алгоритма 1 обусловлено необходимостью выполнения дополнительных действий в области наибольшего сгущения сетки, – например, в силу наличия в ней многокомпонентного течения.

Оценка сверху ускорения S, обеспечиваемого применением LMR сетки относительно расчёта на полностью измельчённой сетке (7), может быть получена в предположениях $\tau_d = 0$ и незначительности затрат на интерполяцию и согласование термодинамических переменных. В первом

приближении, как и следовало ожидать, она определяется выигрышем за счёт уменьшения общего числа элементарных ячеек.

$$T_{reg} = \tau_c n^d 2^{dm} + \tau_{dif} A_m$$
$$T_{LMR} = \tau_c A + \tau_{dif} A_m + \tau_{int} L \quad (6)$$

$$S = \frac{T_{reg}}{T_{LMR}} = \frac{n^d 2^{dm}}{A + \frac{\tau_{int}}{\tau_c} L} < \frac{n^d 2^{dm}}{A} \quad (7)$$

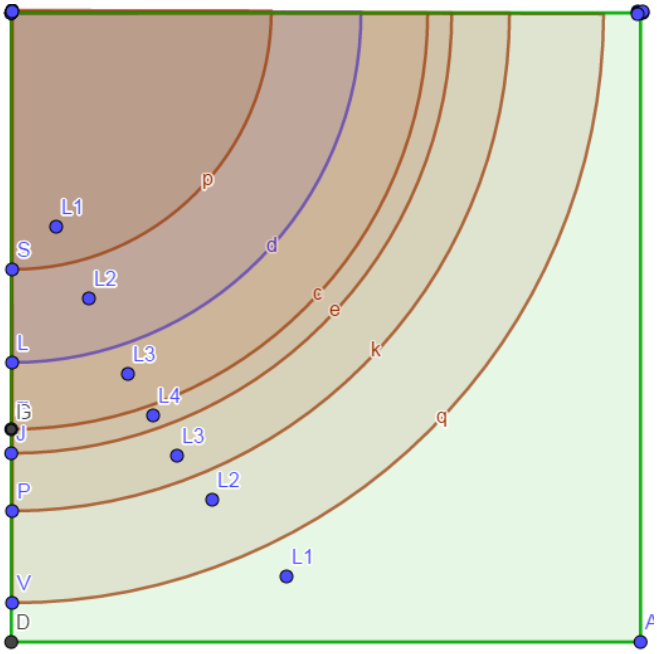


Рис. 11. Пример зон измельчения ячеек

Рассмотрим, без существенной потери общности, задачу эволюции некоторой простой области максимального измельчения сетки. На рисунке 11 ей соответствует тонкое кольцо, помеченное меткой L_4. К кольцу L_4 прилегают два кольца меньшего уровня измельчения L_3. К каждому из них, в свою очередь, примыкает область L_2 и так далее. Метки L_i соответствуют -му уровню измельчения базовых ячеек сетки. Радиусы границ колец меняются в ходе вычислений и определяют перемещение зон измельчения.

Будем полагать заданными значения только двух радиусов, определяющих границы кольца максимального уровня измельчения m: r_m – внутренний и R_m – внешний радиусы кольца, центр которого расположен в вершине квадратной области моделирования. В рассматриваемом примере $m = 4$, соответствующее кольцо помечено меткой L_4. Учитывая ограничения, накладываемые

структурным критерием, остальные радиусы границ колец зон измельчения определяются с учётом требуемого числа элементарных ячеек b в каждой из буферных зон уровня дробления i, разделяющих зоны с уровнями дробления $i-1$ и $i+1$. Примем за 1 длину ребра базовой ячейки. Тогда длина стороны квадратной области моделирования равна $R_0 = n$, а радиусы внутренних и внешних колец уровней измельчения $i = 1, \dots, m-1$ будут равны $r_i = r_{i+1} - \left\lceil \frac{b}{2^i} \right\rceil$ и $R_i = R_{i+1} + \left\lceil \frac{b}{2^i} \right\rceil$.

Введём обозначения $Q = \frac{r_m + R_m}{2}$, $a = \frac{R_m - r_m}{2}$, $\beta = \frac{\tau_{dif}}{\tau_c}$ – отношение времён основной и дополнительной обработки, $\delta = \frac{\pi Q a}{n^2}$ – доля площади, занимаемой максимально измельчёнными ячейками. Для числа активных элементарных ячеек A_i, для числа элементарных ячеек L_i на границах зоны уровня измельчения i и для общего числа таких ячеек L справедливы оценки (8) и (9).

$$A_0 = n^2 - \pi Q(a + 2b) \qquad A_i = \pi Q b 2^{2(i-1)} \qquad A_m = \pi Q a 2^{2m}$$
$$A = n^2 + \frac{\pi}{3} Q\left(3a(4^m - 1) + b(4^{m-1} - 7)\right) \tag{8}$$
$$A \approx n^2 + 4^m \pi Q \left(a + \frac{b}{12}\right)$$

$$L_i = \pi Q b 2^i \qquad L \approx 2^{m+1} \pi Q b \tag{9}$$

С учётом (6), (8) и (9) оценка сверху потенциально достижимого ускорения имеет вид (10).

$$S = \frac{T_{reg}}{T_{LMR}} = \frac{1 + \delta\beta}{\delta\,(\beta + 1)} \in \left(1, \frac{1}{\delta}\right) \tag{10}$$

Алгоритм расчёта на динамически адаптивной локально измельчаемой сетке

При использовании динамически адаптивных сеток выделяются следующие циклически повторяющиеся этапы:

0) `current_mesh = 0; next_mesh = 1;`
1) выполнение шага или нескольких шагов расчёта (алг. 2);
2) проверка необходимости построения новой адаптивной сетки;
3) перестроение расчётной сетки; `current_mesh <-> next_mesh;`

4) копирование и/или получение значений сеточных переменных ячеек новой сетки путём интерполяции значений сеточных переменных ячеек старой сетки;

5) проверка необходимости выполнения динамической балансировки нагрузки;

6) выполнение динамической балансировки нагрузки процессоров (вычисление нового распределения элементов расчётной сетки между процессорами и перемещение между процессорами соответствующих данных).

Обратим внимание на то, что в общем случае используются четыре набора данных, каждый из которых определяется значениями величин `current_mesh`, `next_mesh`, `sloy0`, `sloy1`. Эти величины связаны соотношениями `current_mesh = 1 - next_mesh` и `sloy0 = 1 - sloy1` (алг. 1).

С точки зрения точности выполняемых расчётов целесообразно сокращать число шагов, выполняемых на этапе 1, но с точки зрения сокращения времени вычисления – увеличивать. Соответствующий компромисс достигается благодаря введению достаточно широких буферных зон между различными уровнями измельчения.

Массовые операции этапа 2, поскольку они сопряжены с выполнением коротких логических операций, могут быть совмещены с этапом 1 без увеличения, а как правило, наоборот, с уменьшением общего времени выполнения расчёта (за счёт лучшего использования кеш-памяти).

На этапах 1, 3-4 при листовом измельчении (рис. 3) выполняются затратные, с точки зрения требований к вычислениям и к передаче данных, операции. К плюсам листового разбиения относится тот факт, что общее число листовых ячеек меньше, чем при равномерном. Минусы заключаются в том, что при листовом измельчении:

- необходимо хранить и обрабатывать информацию о структуре разбиения;
- усложнены процедуры вычисления адресов ячеек, имеющих общие грани, но принадлежащих разным базовым ячейкам;
- сложнее, с вычислительной точки зрения, выполнять балансировку нагрузки процессоров, если выполнять её с точностью до листовой ячейки.

В отличие от листового измельчения (рис. 3, слева), в рамках однородного измельчения (рис. 3, справа) удаётся совместить этапы 1-4 и, благодаря блочно-ориентированному подходу, дополнительно снизить трудоёмкость выполнения этапов 5 и 6. Потенциально это позволяет свести накладные расходы до уровня, обеспечивающего возможность выполнения динамической адаптации на каждом шаге вычислений.

На рисунке 12 приведён пример измельчения расчётной сетки для трёх моментов модельного времени (левые рисунки) и соответствующего ему распределения базовых ячеек по блокам базовых ячеек (правые рисунки). На каждом из рисунков правой колонки разными оттенками серого цвета обозначены разные блоки базовых ячеек.

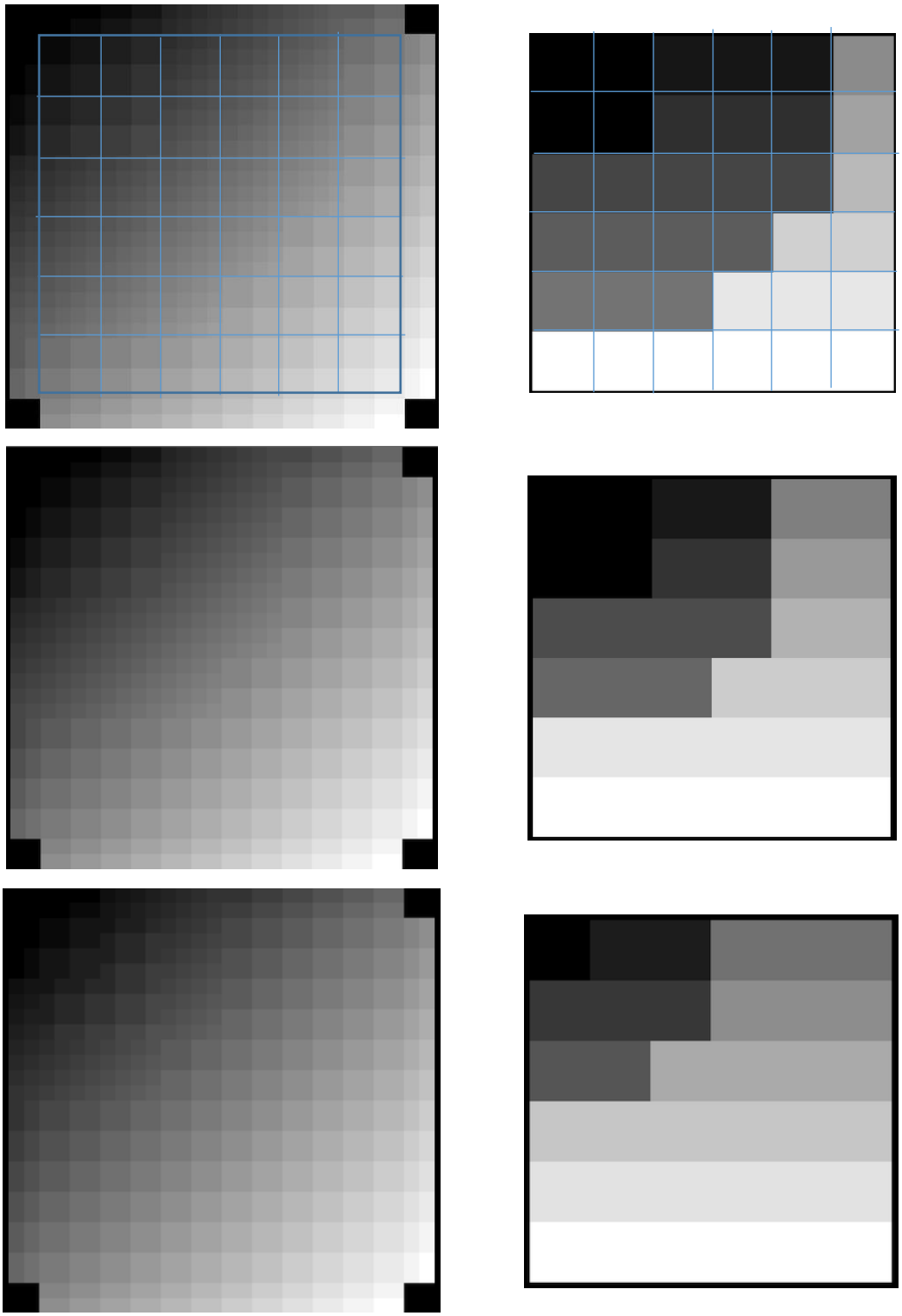


Рис. 12. Измельчение сетки (слева) и распределение ячеек по блокам базовых ячеек (справа) для разных моментов модельного времени, на рисунках первой строки нанесена сетка базовых ячеек

Вычислительные операции (им соответствуют задания на расчёт и интерполяцию блоков элементарных ячеек) могут выполняться на

произвольных процессорных ядрах или графических ускорителях. Поскольку каждый блок элементарных ячеек имеет вид прямоугольника (параллелепипеда в трёхмерном случае), обработка блока сводится к выполнению двух или трёх вложенных циклов, что позволяет при вычислении каждого элементарного задания непосредственно использовать базовые технологии, подобные OpenMP или CUDA. Использование графических ускорителей предполагает предварительное выделение и назначение на обработку заданий высокой трудоёмкости, например, средствами DVMH [22, 23]. Операции управления расчётом при рассматриваемом подходе выполняются на центральных процессорах.

Балансировка нагрузки процессоров

Классические методы декомпозиции

На рисунке 13 представлены домены, сформированные в результате выполнения иерархического алгоритма декомпозиции (а,б) пакетом Metis [15] и при декомпозиции вдоль кривой Гильберта (в,г) для некоторых двух последовательных этапов адаптации сетки [16]. Второй алгоритм далее будем называть фрактальным. Рисунки 13а и 13в соответствуют одному моменту адаптации сетки, а рисунки 13б и 13г – другому. В начале расчёта сетка содержала 64 базовые ячейки, каждая из которых была разделена на 16 элементарных ячеек. Общее число элементарных ячеек 1024, согласно методике [16], примерно сохранялось в течение всего расчёта.

Качество декомпозиции приемлемо в обоих случаях, но следует обратить внимание на объёмы данных, перераспределяемых между процессорами в случае иерархического и фрактального алгоритмов. При переходе от декомпозиции (рис. 13а) к декомпозиции (рис. 13в) объёмы перераспределяемых данных больше, чем при аналогичном переходе в случае применения декомпозиции вдоль фрактальной кривой от (рис. 13б) к (рис. 13г). Наглядно этот факт проиллюстрирован на рисунках 14а и 14б. Цвета исходных доменов (красный, жёлтый, зелёный, цианový и синий) на рисунке 14б занимают б***о***льшую площадь, чем на рисунке 14а, – соответственно, большая часть сеточных данных остаётся на тех процессорах, на которых они уже были размещены. Перемещение данных во втором случае, по построению, затрагивает только пары процессоров, имеющих соседние номера, а в первом случае – произвольные пары процессоров.

Меньшие затраты времени на вычисление доменов при использовании фрактальных кривых обеспечивают возможность более частого вызова процедуры перебалансировки, дополнительно снижая объёмы передаваемых при каждой балансировке данных. Отметим, что кроме кривой Гильберта есть множество других кривых, заполняющих пространство (рис. 15). Например, популярна кривая Мортона, отличающаяся простотой вычисления топологии, но не сохраняющая, в отличие от кривой Гильберта, близость в физическом

пространстве точек, имеющих соседние номера на самой кривой. Тем не менее методы на основе фрактальных кривых не лишены недостатков, поскольку при их использовании затруднено вычисление отношения соседства фрагментов, расположенных в разных частях кривой.

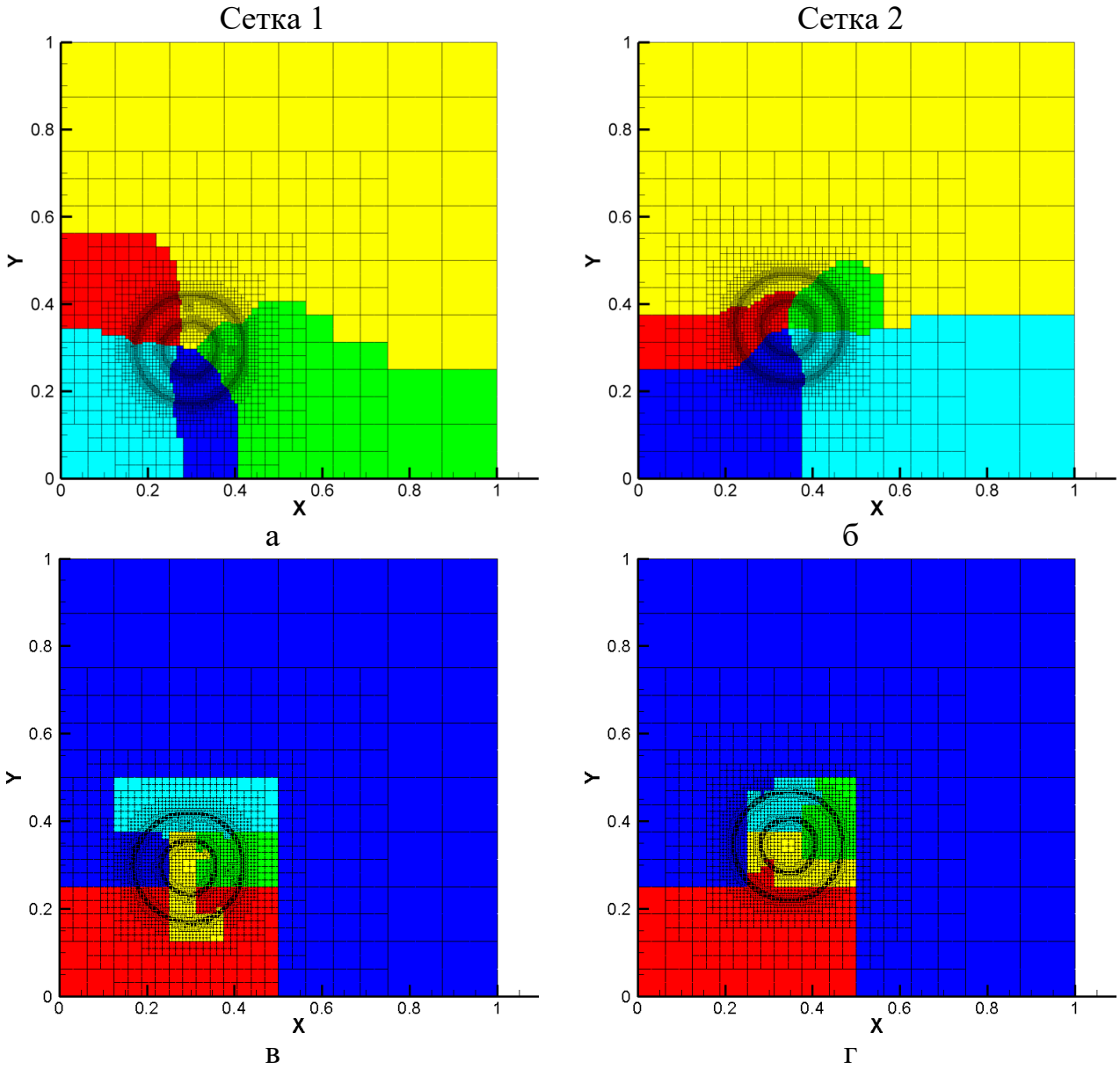


Рис. 13. Домены, сформированные с помощью иерархического метода (а,б) и вдоль кривой Гильберта (в,г) [16]

Напомним, что при оценке трудоёмкости T_1 выполнения вычислений на регулярной сетке (3) рассматривались элементарные ячейки двух типов. Трудоёмкость τ_c операций при обработке ячеек первого типа (содержащих только одно вещество) не зависит от конкретных значений сеточных функций в таких ячейках. Трудоёмкость обработки ячеек второго типа (содержащих

несколько разных веществ) существенно больше первой трудоёмкости и существенно зависит от текущих значений сеточных функций в ячейке. Вычисления в таких ячейках выполняются в две стадии.

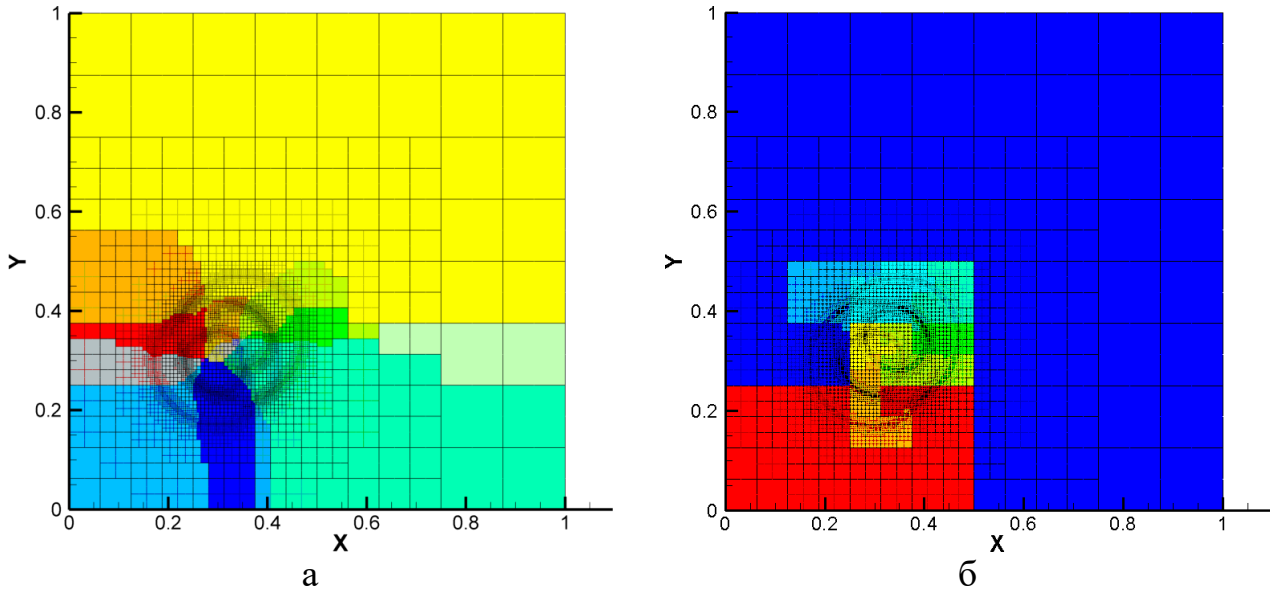


Рис. 14. Пересечение доменов, полученных с помощью иерархического метода (а) и с помощью кривой Гильберта (б)

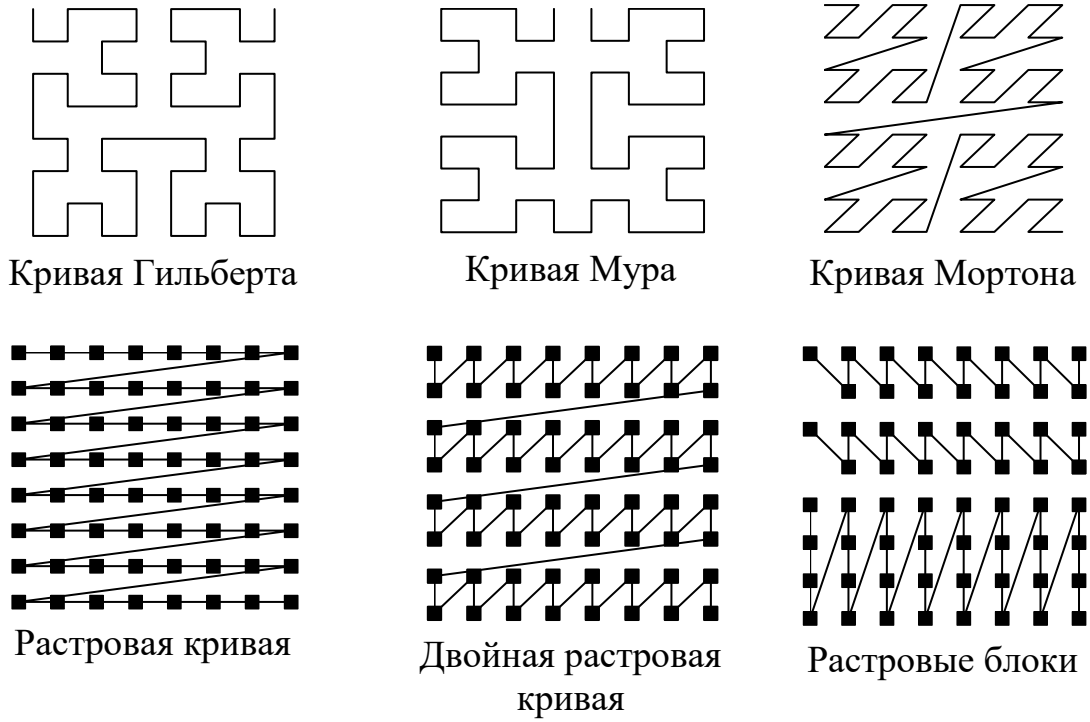


Рис. 15. Кривые, заполняющие пространство

На первой выполняются точно такие же операции, как и в ячейках первого типа. Для их выполнения требуются данные из соседних ячеек согласно вычислительному шаблону (рис. 5). Их трудоёмкость равна τ_c для каждой

элементарной ячейки и, соответственно, $\tau_c 2^{d \cdot level}$ для каждой базовой ячейки. На второй стадии, например, для согласования термодинамических переменных, используется итерационный процесс, сходящийся в разных ячейках за разное число итераций, поэтому в разных ячейках значения $\tau_{d,i,j}$ могут многократно отличаться друг от друга. Но при выполнении этих вычислений в элементарной ячейке не требуются данные из соседних элементарных ячеек.

Таким образом, для организации эффективного расчёта при выполнении вычислений первой стадии следует применять один вид разбиения расчётной области на домены, а при выполнении второй стадии – другой. Поскольку τ_c могут рассматриваться как константы, то для декомпозиции применимы методы, аналогичные методам, используемым при статической балансировке нагрузки, такие как рекурсивные бисекции, инкрементный, диффузный алгоритмы и некоторые другие. Указанные методы не применимы для декомпозиции операций второй стадии, поскольку заранее неизвестны трудоёмкости обработки той или иной элементарной ячейки.

Похожая проблема возникает при моделировании на основе метода суммарной аппроксимации фронтов горения [18]. Решение жёстких систем обыкновенных дифференциальных уравнений, описывающих кинетику горения, сопряжено с выполнением плохо прогнозируемого числа итераций. Во-первых, это приводит к сильным дисбалансам вычислительной нагрузки процессоров. Во-вторых – к слабой предсказуемости номеров процессоров, на которых вычислительная нагрузка будет существенно выше, чем на других.

Рассматриваемая сейчас задача проще, поскольку вероятность достоверного априорного выявления элементарных ячеек с высоким временем обработки достаточно высока, что позволяет предложить следующий метод динамической балансировки нагрузки процессоров:

- вычисление декомпозиции множества базовых ячеек, в предположении, что трудоёмкость их обработки пропорциональна числу содержащихся в каждой базовой ячейке элементарных ячеек;
- разбиение доменов на группы и формирование расписания перераспределения данных в пределах каждой группы для выполнения второй стадии вычислений.

Послойный алгоритм декомпозиции множества базовых ячеек

Рассмотрим алгоритм, обеспечивающий распределение по доменам базовых ячеек, каждая из которых считается неделимой единицей обработки, независимо от того, на сколько уровней она измельчена. Для ускорения вычисления декомпозиции решётки базовых ячеек предлагается послойный метод, оперирующий блоками базовых ячеек, соответствующий декомпозиции с использованием кривой «Растровые блоки» (рис. 15).

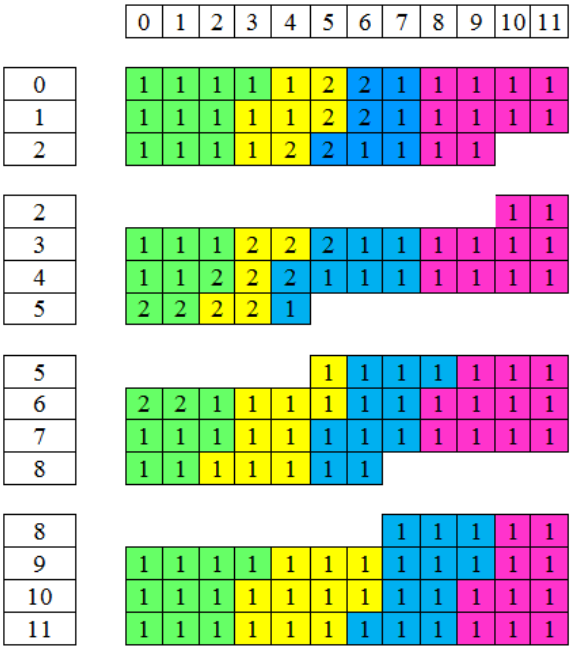


Рис. 16. Домены

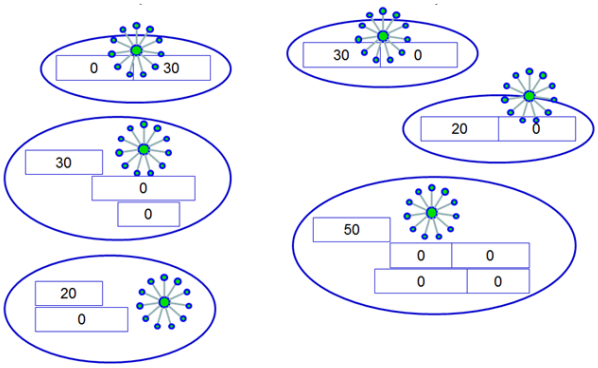


Рис. 17. Группы доменов

В его основе лежит идея поэтапного формирования доменов. Рассмотрим двумерный случай. Предполагается, что число доменов p представимо в виде $p = p_1 p_2$. На первом этапе всё множество базовых ячеек разбивается на p_1 полос примерно одинакового веса, определяемого первой стадией вычислений (рис. 16). Для каждой базовой ячейки указан условный вес, определяемый уровнем измельчения. Следовало вместо веса 2 указать вес 4, что соответствовало бы числу элементарных ячеек при однократном дроблении базовой ячейки, но тогда снизилась бы общая наглядность примера. На втором этапе в каждой полосе формируется p_2 доменов примерно равного веса. Домены каждой полосы показаны разными цветами. Всего в примере сформировано 16 доменов ($p = 4 \cdot 4$). Ячейки одинакового цвета в разных полосах соответствуют разным доменам. В трёхмерном случае домены формируются аналогично, но в три этапа: в предположении, что $p = p_1 p_2 p_3$ формируется p_1 плоский слой, затем в каждом слое p_2 колонн, после чего – в каждой колонне p_3 домен примерно среднего веса. Важно, что данный подход позволяет выполнять формирование требуемых доменов одновременно и независимо, после обмена предварительной общей информацией ограниченного объёма порядка $O\left(\frac{n^{d-1}}{p}\right)$, на каждом из процессоров. С учётом того, что на каждом из процессоров i требуются, кроме сведений о конфигурации домена с номером i, сведения о конфигурации не всех доменов, а только о тех, что соприкасаются с доменом i, весь процесс расчёта, благодаря поэтапной организации, может выполняться независимо на каждом из процессоров.

Рассматриваемый алгоритм формирования доменов обеспечивает возможность вычисления границ домена с любым заданным номером за время, пропорциональное $O\left(\frac{n}{p_1}\right)$, где n – число базовых ячеек вдоль ребра всей исходной сетки. Указанная оценка времени не включает этапа предварительной подготовки описания структуры измельчения, трудоёмкость которой не превышает, при последовательной обработке, $O(n^d)$, где d – размерность пространства. В общем случае приведённая оценка является неулучшаемой, поскольку для вычисления требуемых уровней измельчения необходим просмотр каждой из базовых ячеек. В частном случае, при наличии априорной информации о характере течения, оценка может быть улучшена благодаря использованию блочного описания до уровня $O(n^{d-1})$. Следует отметить, что при использовании листового измельчения соответствующее время пропорционально числу листовых ячеек, которое многократно превышает количество базовых. Параллельная обработка позволяет дополнительно сократить оценку трудоёмкости расчёта декомпозиции до величины $O\left(\frac{n^{d-1}}{p}\log p\right)$. Базовые ячейки распределяются между процессорными узлами в соответствии с вычисленными доменами. На рисунке 18 приведены суммарные трудоёмкости обработки ячеек доменов на первой стадии.

Балансировка на этапе определения термодинамических параметров

Рассмотрим способ организации вычислений на второй стадии, учитывающий имеющееся распределение для первой стадии (рис. 16).

10	10	10	10
11	10	10	10
10	9	10	11
10	10	11	10

Рис. 18. Однородный расчёт

0	30	30	0
30	50	20	0
20	0	0	0
0	0	0	0

Рис. 19. Динамическая балансировка

Пусть предполагаемые суммарные трудоёмкости обработки ячеек доменов на второй стадии соответствуют указанным на рисунке 19. Суммарная по всем доменам трудоёмкость второй стадии равна 180, а средняя трудоёмкость обработки одного домена, при условии равномерного распределения вычислений по всем процессорам, равна 11.

В результате выполнения послойного алгоритма декомпозиции базовых ячеек уже получено распределение базовых ячеек по доменам. Далее на его основе определяются группы доменов для эффективного выполнения расчётов второй стадии.

– Для каждого домена i вычисляются априорные трудоёмкости w_i определения термодинамических параметров в смешанных ячейках (рис. 19).

– Вычисляется средняя трудоёмкость выполнения этапа $\overline{w} = \frac{\sum_i^p w_i}{з}$. В рассматриваемом примере $\overline{w} = 11$.

– Домены упорядочиваются по возрастанию вычисленной трудоёмкости обработки доменов w_i. В рассматриваемом примере, при нумерации доменов слева направо и сверху вниз, массив выглядит следующим образом: {0,3,7,9,10,11, 12,13,14,15,6,8,1,2, 4,5}.

– Всё множество доменов с помощью жадного алгоритма разбивается на группы. Каждому домену k с высокой трудоёмкостью $w_k > \beta\,\overline{w}$ (они расположены в конце упорядоченного массива) приписывается один или несколько доменов с низкой трудоёмкостью $w_q \ll \overline{w}$ (они в начале массива). Число приписываемых доменов определяется отношением $\frac{w_k}{\overline{w}}$. В результате каждой группе (рис. 17) приписан один домен с трудоёмкостью, превышающей среднюю не менее чем в β раз, и один или несколько доменов с низкой трудоёмкостью. Рассматриваемому примеру соответствуют группы {0,3,7,9,5} {10,11,4} {12,2} {13,1} {14,8} {15,6}.

– В каждой из сформированных групп выполняется расчёт второй стадии.

Фактически всё множество вычислительных узлов (каждому из них соответствует один домен) разбивается на два класса: вычислительные узлы, обрабатывающие домены с большим числом ячеек, требующих выполнения второй стадии, и остальные вычислительные узлы. На рисунке 17 представлено разбиение процессоров на группы, в каждой из которых есть один из процессоров первого множества и один или несколько – из второго. Далее в пределах каждой группы задания на обработку трудоёмких ячеек распределяются равномерно либо на основе метода коллективного решения. Предлагаемый алгоритм эффективен при условии превышения времени вычислений второй стадии передаваемой группы ячеек над временем передачи данных, определяемым латентностью (порядка микросекунд) и пропускной способностью интерконнекта.

Заключение

Предложенные алгоритмы реализованы в виде программного комплекса, предварительное тестирование которого подтверждает перспективность выбранного подхода. В частности, при моделировании с помощью явной разностной схемы гидродинамической двумерной задачи получены следующие результаты. При равномерно измельчённой сетке с общим числом ячеек $K_{\text{рег}} = 1280^2 = 1\,638\,400$ время счёта на одном процессоре составило $T_{\text{рег}} = 516$ секунд. При адаптивном измельчении расчётной сетки, покрывающей ту же геометрическую область, общее число элементарных ячеек снизилось до величины $K_{\text{адапт}} = 14\,860$, а время счёта составило $T_{\text{адапт}} = 4.98$ секунды.

Размеры наименьших ячеек совпадали в обеих сетках. Таким образом, достигнутое фактическое ускорение $S = \frac{T_{\text{рег}}}{T_{\text{адапт}}} = 104$ отличается от максимально ожидаемого ускорения $S_{max} = \frac{K_{\text{рег}}}{K_{\text{адапт}}} = 110$ на величину порядка 5%, что подтверждает незначительность накладных расходов, обусловленных дополнительными операциями обработки адаптивной сетки. В указанные 5% входят неснижаемые расходы на интерполяцию данных в ячейках, примыкающих к границам зон измельчения разного уровня.

Авторы благодарят Д.А. Захарова за значительную помощь в разработке программного комплекса, обсуждение которого выходит за рамки данного препринта.

Библиографический список

1. Рейтинг суперкомпьютеров. URL: http://top500.org
2. Самарский А.А. Теория разностных схем. – «Наука», Глав. ред. физико-математической лит-ры, 1989.
3. Афендиков А.Л., Давыдов А.А., Луцкий А.Е. и др. Адаптивные вейвлетные алгоритмы для решения задач гидро и газовой динамики на декартовых сетках. М.: ИПМ им. М.В. Келдыша Москва, 2016.
4. Абалакин И.В., Жданова Н.С., Суков С.А. Реконструкция геометрии объекта на элементах неструктурированной сетки при использовании метода погруженных границ // Математическое моделирование. – 2016. – Т. 28. – №. 6. – С. 77-88.
5. Сухинов А.А. Построение декартовых сеток с динамической адаптацией к решению // Математическое моделирование. – 2010. – Т. 22. – №. 1. – С. 86-98
6. Dubey A. et al. A survey of high level frameworks in block-structured adaptive mesh refinement packages // Journal of Parallel and Distributed Computing. – 2014. – Т. 74. – №. 12. – С. 3217-3227.
7. Aftosmis M.J. Solution adaptive Cartesian grid methods for aerodynamic flows with complex geometries // VKI Lecture Series. – 1997. – Т. 2. – С. 1997.
8. Adams M. et al. Chombo software package for amr applications-design document. – 2015. – №. LBNL6616E.
9. Burstedde C., Wilcox L. C., Ghattas O. p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees // SIAM Journal on Scientific Computing. – 2011. – Т. 33. – №. 3. – С. 1103-1133
10. FlowVision Адаптации расчётной сетки и тонкости их применения, 20.07.2020. URL: https://flowvision.ru/ru/support-menu-header-ru/blog-ru/adaptation?showall=1

11. Adaptive Mesh Refinement. Los Alamos National Laboratory. URL: https://www.lanl.gov/projects/codesign/codesign-summer-school/research-areas/adaptive-mesh-refinement.php
12. Якобовский М.В. Введение в параллельные методы решения задач. М.: МГУ. – 2013.
13. Горобец А.В., Суков С.А., Триас Ф.Х. Проблемы использования современных суперкомпьютеров при численном моделировании в гидродинамике и аэроакустике // Ученые записки ЦАГИ. – 2010. – Т. 41. – №. 2., – С. 65-71
14. Hendrickson B. , Leland R. A Multilevel Algorithm for Partitioning Graphs // Supercomputing '95 Proceedings. – San Diego, CA, 1995. http://www.leonidzhukov.net/hse/2016/networks/papers/MultilevelAlgorithmPartitioningGraphs.pdf
15. Karypis George. Family of Graph and Hypergraph Partitioning Software. URL: http://glaros.dtc.umn.edu/gkhome/views/metis/
16. Сухинов А.А. Математическое моделирование процессов переноса примесей в жидкостях и пористых средах. Автореферат диссертации на соискание учёной степени кандидата физико-математических наук, специальность 05.13.18 — Математическое моделирование, численные методы и комплексы программ, Москва — 2009. URL: https://keldysh.ru/council/3/D2058/sukhinov.pdf
17. Головченко Е.Н. Кандидатская диссертация «Декомпозиция расчетных сеток для решения задач механики сплошной среды на высокопроизводительных вычислительных системах». [Электронный ресурс]. URL: https://keldysh.ru/council/3/D00202403/golovchenko_diss.pdf
18. Kornilina M.A., Yakobovskii M.V. Modeling of Evolution of Complicated Nonlinear Systems on Multiprocessor Computational Complexes // Russian Journal of Physical Chemistry A, Pleiades Publishing, Ltd (Road Town, United Kingdom), 1995, № 69 (8), p. 1390-1393.
19. Лисейкин В. Д. Разностные сетки. Теория и приложения. ISBN: 978-5-7692-1364-9 Издательство: СО РАН. 2014. – 256 с. URL: https://www.rfbr.ru/rffi/ru/books/o_1920731
20. Лисейкин В.Д. Методы построения разностных сеток. – Новосибирск: Ред.-издат. центр НГУ, 2014. – 208 с.
21. Zhang W., Myers A., Gott K., Almgren A., Bell J. AMReX: Block-structured adaptive mesh refinement for multiphysics applications. The International Journal of High Performance Computing Applications. 2021;35(6):508-526. doi:10.1177/10943420211022811

22. Бахтин В.А., Крюков В.А. DVM-подход к автоматизации разработки параллельных программ для кластеров // Программирование. 2019. № 3. С. 43-56. URL: https://doi.org/10.1134/S0132347419030038

23. Описание DVM системы URL: http://dvm-system.org

Приложение 1. Основные структуры данных

Блок базовых ячеек LRM_BASE_BLOCK – параллелепипедный (прямоугольный) блок базовых ячеек одинакового уровня измельчения. Блок базовых ячеек задаётся указанием номеров левой верхней ячейки и числа ячеек по направлениям.

```
typedef struct LRM_BASE_BLOCK // Описание одного куба базовых ячеек
{
  int level2; // 2^level число дочерних ячеек прилегающих к ребру одной базовой
  int i_bb, j_bb; // начало блока базовых ячеек
                  // одинакового уровня измельчения
  int m1, m2;     // размеры блока базовых ячеек
                  // одинакового уровня измельчения

  int id_of_vc_block;  // номер физического блока, хранящего физические данные
  int i_vc_offset_vc_block; // смещение в физическом блоке данных
  int j_vc_offset_vc_block; // в единицах элементарных ячеек ( прибавлено nb )
}
  LRM_BASE_BLOCK;
```

Элементарная ячейка VCELL – описание элемента физического объёма, содержащего сеточные (физические) величины. Его размер зависит от уровня измельчения. Именно эти ячейки являются листовыми элементами расчётной сетки

```
typedef struct VCELL // сеточные физические величины одной элементарной ячейки
{
  double   rho;  // плотность
  double   T;    // температура
  double   c[3]; // концентрации
  // ...
  int id_bb_ph;  // номер базового блока
}
  VCELL;
```

Блок элементарных ячеек LRM_VC_BLOCK – параллелепипедный (прямоугольный) блок физических ячеек одного уровня измельчения.

Некоторые блоки элементарных ячеек описывают физические границы области или теневые грани, а некоторые могут не участвовать в расчёте, поскольку описывают область, покрытую иным уровнем измельчения.

```
typedef struct LRM_VC_BLOCK   // эта структура оперирует
{                             // адресами элементарных ячеек,
                              // но масштаба базовых
  int i_bc,j_bc; // номер угловой базовой ячейки

  int level2;     // число элементарных ячеек в одной базовой ячейке
  int m1,m2;      // фактические размеры куба физических
                  // данных элементарных ячеек

  LPLPVCELL vc[2]; // указатели на начала строк физических данных
                   // элементарных ячеек для опорного и нового слоя по времени
}
  LRM_VC_BLOCK;
```

Задание LRM_TASK – одно задание на выполнение вычислений или подготовительных действий (копирование, интерполяция, вычисление, пересылка, …). Именно задание является единицей планирования работы на уровне MPI. На уровне OpenMP задание может обрабатываться несколькими процессорными ядрами.

```
typedef struct LRM_TASK // Описание одного задания
{
  enum LRM_TASK_TYPE type;     // тип задания ( копирование, вычисление, ... )
  LRM_TASK_BLOCK block_target; // целевой блок ячеек
  LRM_TASK_BLOCK block_source; // блок ячеек источника
}
  LRM_TASK;
```

Блок ячеек задания LRM_TASK_BLOCK – прямоугольный (параллелепипедный) блок физических ячеек одного уровня измельчения, описывающий область источника или цели одного задания

```
typedef struct LRM_TASK_BLOCK // Описание одного блока элементарных ячеек
{
  int id_of_base_block; // номер базового блока

  int i_vc_off;   // фактическое, никаких смещений потом не добавят,
  int j_vc_off;   // положение первой элементарной активной ячейки этого блока
                  // в сетке соответствующего vc блока

  int m1; // фактические размеры блока активных элементарных ячеек подлежащих
  int m2; // обработке. Интерпретация зависит от типа задания. Допустимы
          // обращения за пределы этого блока (к теневым граням)
}
  LRM_TASK_BLOCK;
```

Основная структура описания локально измельчаемой сетки LRM

```
typedef struct LRM // описание адаптивной сетки
{
  int n1,n2;      // размеры базовой сетки
  int nb;         // размер одностороннего шаблона

  int set;  // номер набора блоков, описывающих активную сетку (0 или 1)
  int sloy; // номер активного текущего слоя (0 или 1)
            // всего используется четыре набора для хранения блоков
            // базовых и элементарных ячеек

  int             n_base_block; // число блоков базовых ячеек активного слоя
  LRM_BASE_BLOCK  *base_block; // блоки базовых ячеек

  int             n_base_block_new; // число блоков базовых ячеек целевого слоя
  LRM_BASE_BLOCK  *base_block_new;  // блоки базовых ячеек

  int           n_vc_block; // число выделенных блоков элементарных ячеек
  LRM_VC_BLOCK  *vc_block; // блоки элементарных ячеек. Каждый vc_block
                           // может обеспечивать несколько base_block

  int       n_task; // число заданий
  LRM_TASK   *task; // задания (инициализация, счёт, интерполяцию, пересылка, …)
}
  LRM;
```

Параметры функции задания

```
VC_FUN_HEAD_PARAM:
      LRM &mesh,                                  // ссылка на описание сетки
      enum LRM_TASK_TYPE type_fun,                // уточнение типа операции
      int id_tar_vc_block,                        // номер целевого блока
      int id_src_vc_block,                        // номер блока источника
      int in, int jn,                             // размеры обрабатываемого блока
      int i_tar_off, int j_tar_off,               // начало целевого блока
      int i_src_off, int j_src_off,               // начало блока источника
      double              *p_reduction,           // редукционные переменные
      LRM_REDUCTION_TYPE   type_of_reduction,
      LRM_REDUCTION_START  start_of_reduction
```