УДК 519.6

АЛГОРИТМЫ ДЕКОМПОЗИЦИИ НЕРЕГУЛЯРНОЙ СЕТКИ С УЧЕТОМ ВЫЧИСЛИТЕЛЬНОЙ НАГРУЗКИ

О. И. Бутнев, В. А. Пронин
(РФЯЦ-ВНИИЭФ)

Предложены два алгоритма геометрической декомпозиции по процессорам нерегулярной сетки для двумерного случая — по *полосам* и *клеткам*, включающие способ уравновешивания вычислительной нагрузки процессоров в процессе счета на этапе записи-чтения разреза задачи. Алгоритмы протестированы в рамках комплекса программ, реализующего нерегулярную методику МЕДУЗА. Они могут успешно применяться в других методиках, использующих как структурированные, так и неструктурированные сетки.

Постановка задачи и возникающие сложности

В связи с растущей потребностью проведения расчетов в параллельном режиме с использованием нерегулярных сеток все более актуальной становится задача разработки достаточно простых и точных методов декомпозиции таких сеток по процессорам. Особенно актуальной задача становится при существенном увеличении количества рассчитываемых точек. В настоящей работе описаны два алгоритма декомпозиции, которые были протестированы в рамках комплекса программ, опирающегося на основы методики МЕДУЗА [1].

Для того чтобы начать распараллеливание основных модулей комплекса и проверить работоспособность создаваемых программ, необходимо определиться с декомпозицией задачи по процессорам. Существует достаточно много методов декомпозиции с использованием различных критериев. Одним из наиболее распространенных методов является геометрическая декомпозиция. Для методик с регулярной сеткой геометрическая декомпозиция не представляет большой сложности в силу матричной структуры сетки. Для нерегулярных методик это является достаточно сложной задачей. В [2], где приведен обзор методов разрезания графов, подробно разобран спектральный метод [3], предложенный чешским математиком М. Фидлером. Нельзя сказать, что какой-либо метод предпочтительнее другого, так как каждый из них имеет свои области применения. В частности, при использовании спектрального метода наиболее трудоемким этапом является поиск собственных векторов матрицы, что ограничивает на практике размер разрезаемого графа. Для больших графов в работе [2] предложен иерархический алгоритм разрезания. Суть его состоит в том, чтобы подвергать непосредственному разбиению не сам граф, а его грубый макет, имеющий существенно меньшие размеры.

Авторами было решено разработать простые алгоритмы декомпозиции, основанные на методе частичных сумм. В данной работе описан алгоритм *полосы*, позволяющий осуществлять декомпозицию задачи с помощью прямых, параллельных координатным осям, и алгоритм *клетки*, являющийся обобщением предыдущего алгоритма и позволяющий осуществлять декомпозицию в виде клеток. Разработанные алгоритмы были видоизменены таким образом, чтобы набирать точки на каждый процессор с учетом его загруженности. Данная процедура может производиться на этапе записи-чтения разреза задачи.

Одной из проблем, возникающих при геометрической декомпозиции нерегулярной сетки, является определение соседства процессоров (граф соседства) и организация перекрытий параобластей (областей сетки, размещаемых на процессорах). В случае декомпозиции *полосы* соседство процессоров детерминировано, так как каждый процессор граничит с предыдущим (если он не

первый) и следующим (если он не последний). Для более сложных декомпозиций (иерархический алгоритм или алгоритм *клетки*) граф соседства требует дополнительных вычислений.

Разностная схема методики МЕДУЗА такова, что необходимо реализовать перекрытия параобластей на два слоя в каждую сторону. Авторами был реализован алгоритм перекрытий для произвольного планарного графа соседства, возникающего как при декомпозиции с помощью иерархического алгоритма, так и алгоритма *клетки*. В этом алгоритме также была решена задача связывания, т. е. установления соответствия между локальным номером точки текущего процессора с номерами соседних процессоров и локальными номерами образов этой точки на соседних процессорах. Данная проблема является специфической для нерегулярных сеток, так как в регулярном случае эти номера легко вычисляются по номеру процессора и глобальному номеру текущей точки.

Остановимся сначала более подробно на алгоритмах декомпозиции *полосы* и *клетки*, а затем опишем видоизменения этих алгоритмов для балансировки вычислительной нагрузки процессоров.

Декомпозиция *полосы*

Декомпозиция области проводится на основании X или Y-координат ячеек сетки построением полос по X или Y соответственно таким образом, чтобы число ячеек в каждой полосе было примерно одинаковым. Каждая полоса впоследствии ассоциируется с номером процессора. Число полос совпадает с числом процессоров. Опишем без ограничения общности алгоритм набора ячеек в каждую X-полосу.

Сначала производится обход по всем ячейкам задачи и подсчитывается текущее количество $MIPALL$ счетных ячеек (среди всего множества ячеек могут быть не использующиеся). Здесь же вычисляются глобальные максимумы и минимумы X-координат ячеек. Обозначим их через $Xmin$ и $Xmax$. Затем определяется среднее число $AvPt$ ячеек, к которому должно стремиться суммарное количество ячеек для каждого процессора:

$$AvPt = \frac{MIPALL}{numnod}, \qquad (1)$$

где $numnod$ — количество используемых процессоров.

Дальнейший алгоритм похож на аппроксимацию интеграла частичными суммами, где в качестве подынтегральной функции выступает некоторая функция от количества ячеек внутри каждой элементарной полоски. Эти полоски образуются путем разбиения всего отрезка $[Xmax, Xmin]$ на интервалы Dx, количество которых сделаем зависимым от числа используемых процессоров. Обозначим через $Ninx$ количество элементарных интервалов вдоль оси X. Эту величину будем вычислять по формуле

$$Ninx = N \times numnod,$$

где N — некоторый целочисленный параметр, определяющий точность разбиения. Чем больше этот параметр, тем меньше дисбаланс по количеству набранных на процессоры ячеек. В настоящее время этот параметр задан равным 50.

Следующим этапом является инициализация массивов $Xotr[Ninx + 1]$ и $Potr[Ninx]$ (в квадратных скобках указаны размерности массивов). $Xotr$ содержит координаты границ элементарных полосок, а $Potr$ — количество ячеек в каждой элементарной полоске. Будем считать, что ячейка принадлежит k-й полоске, если координата X_i этой ячейки удовлетворяет неравенству

$$Xotr_k < X_i \le Xotr_{k+1}, \quad k = 1, 2, \ldots, Ninx.$$

На последнем этапе происходит набор процессорных полос с использованием информации из массивов $Xotr$ и $Potr$. Зная координаты границ каждой элементарной полоски и количество точек в ней, можно набрать процессорную полосу и узнать координаты границ этой полосы. Для этого устраиваем обход по элементарным полоскам и суммируем количество ячеек в каждой из них. Как только это количество будет примерно равным $AvPt$, процесс набора полосы обрывается и запоминаются координаты начала первой из набранных полосок и конца последней. Здесь же инициализируются массивы $XPol[numnod + 1]$ и $PtPol[numnod]$ координат и количества ячеек в каждой процессорной полосе соответственно. Следующую процессорную полосу будем набирать от границы предыдущей, а число $AvPt_k$, к которому будет стремиться количество ячеек для следующего процессора, пересчитаем по формуле

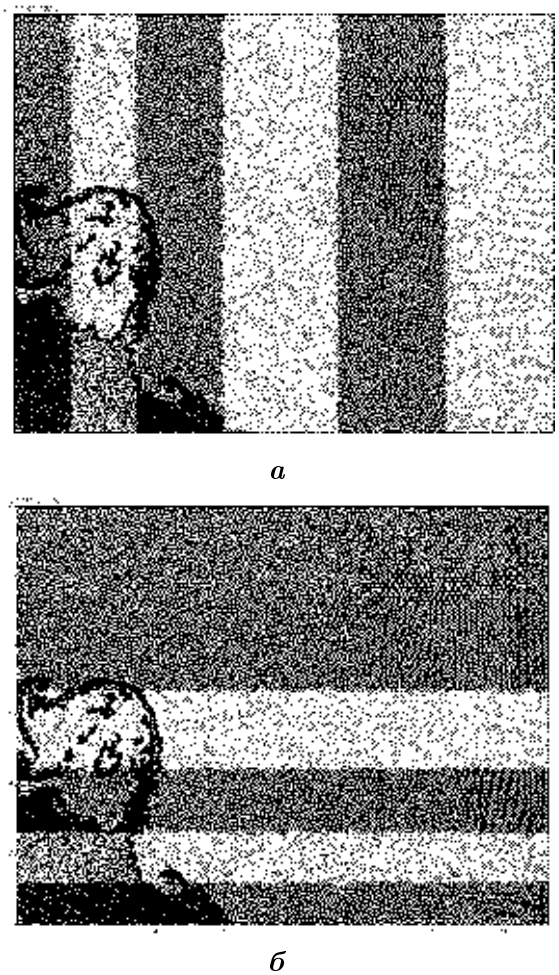
$$AvPt_k = \frac{MIPALL - \sum_{i=1}^{k-1} PtPol\,(i)}{numnod - k + 1}, \qquad (2)$$
$$k = 1, 2, \ldots, numnod - 1.$$

Полоса последнего процессора не набирается, а вычисляется с учетом информации с предыдущих полос. Корректировка значения $AvPt$ по формуле (2) приводит к меньшему дисбалансу точек, распределенных между процессорами.

В качестве тестовой задачи, на которой проверялась работоспособность алгоритма, была выбрана задача о падении плоской ударной волны на заполненную тяжелым газом прямоугольную область. Постановка этой задачи приведена в [4]. На рис. 1, *а,б* изображена декомпозиция задачи в виде полос по X и Y соответственно на 6 процессоров.

В табл. 1 приведено количество точек на каждом процессоре при декомпозиции, изображенной на рис. 1. При этом задача считалась на 107 269 ячейках.

Если посчитать по формуле (1) среднее количество ячеек $AvPt$, к которому должно стремиться количество ячеек для каждого процессора, то получим, что $AvPt \approx 17\,878$ точек. Из табл. 1 видно, что дисбаланс количества точек между процессорами менее 2 %.



а

б

Рис. 1. Декомпозиция задачи в виде полос на 6 процессоров: *а* — вдоль оси X; *б* — вдоль оси Y

Таблица 1

Количество точек на процессорах

Номер процессора	Кол-во ячеек
1	17 950
2	17 599
3	17 810
4	17 875
5	18 154
6	17 881

Декомпозиция *клетки*

Декомпозиция области проводится на основании координат X, Y ячеек построением процессорных клеток таким образом, чтобы число ячеек в каждой клетке было примерно одинаковым. Будем считать заданным число процессоров: $NumProc\_X$ для расчета вдоль оси X и $NumProc\_Y$ — вдоль оси Y. Проведем описание алгоритма с учетом содержания предыдущего раздела.

Основное отличие данного алгоритма от предыдущего состоит в том, что вся область задачи от $Xmin$ до $Xmax$ и от $Ymin$ до $Ymax$ покрывается элементарными прямоугольниками, а не полосками. Применим алгоритм декомпозиции *полосы* в два этапа по каждому направлению. Без ограничения общности будем применять его сначала вдоль X, а затем вдоль Y.

Сначала вычислим величину $AvPtX$, к которой должно стремиться количество точек каждой полосы, по формуле

$$AvPtX = \frac{MIPALL}{NumProc\_X}. \qquad (3)$$

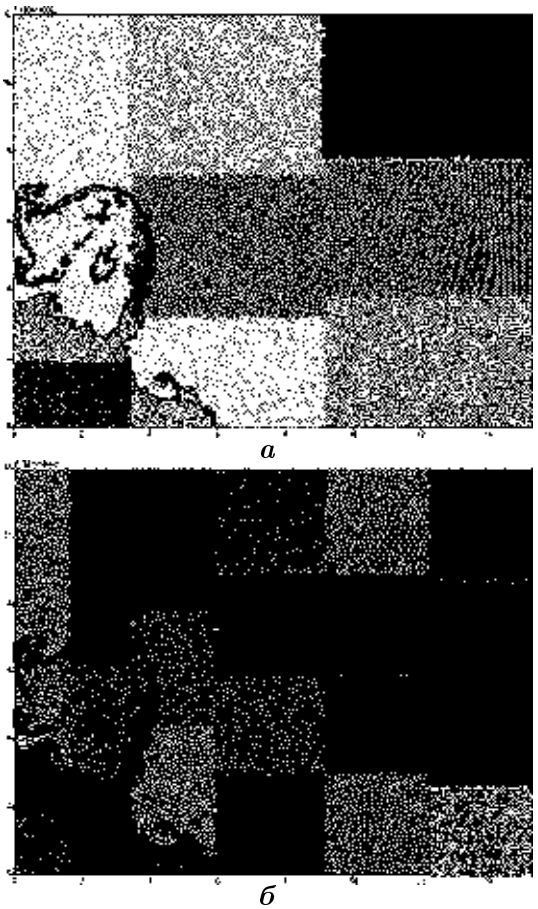
Обозначим через $Ninx$ и $Niny$ количество интервалов вдоль каждого направления. Проинициализируем массивы $Xotr[Ninx + 1]$, $Yotr[Niny + 1]$ и $Potr[NinX, NinY]$ по алгоритму, описанному выше. Будем называть элементарными x_k-полосками с границами $(Xotr_k, Xotr_{k+1})$ объекты в виде объединения элементарных прямоугольников вдоль оси Y с такими же границами, y_m-полосками с границами $(Yotr_m, Yotr_{m+1})$ — объекты в виде объединения элементарных прямоугольников

вдоль оси X с такими же границами. Будем считать, что ячейка принадлежит x_k-полоске и y_m-полоске, если координаты X_i, Y_i этой ячейки удовлетворяют неравенствам

$$\begin{aligned} &Xotr_k < X_i \le Xotr_{k+1}, \quad k = 1, 2, \ldots, Ninx; \\ &Yotr_m < Y_i \le Yotr_{m+1}, \quad m = 1, 2, \ldots, Niny. \end{aligned} \tag{4}$$

Рассматривая алгоритм декомпозиции *полосы* для элементарных x_k-полосок, получаем набор из $NumProc\_X$ полос с уравновешенным количеством точек. Каждую такую полосу будем называть x-полосой. Запоминая границы и количество точек каждой x-полосы, можно применить тот же алгоритм *полосы* вдоль другого направления с разбиением на $NumProc\_Y$ полос. При этом каждую x-полосу будем рассматривать независимо от других. В итоге получим разбиение по клеткам, число которых должно совпадать с количеством используемых процессоров.

На рис. 2, *а,б* представлены декомпозиции области по клеткам на 9 и 24 процессоров соответственно.



а

б

Рис. 2. Декомпозиция задачи по клеткам: *а* — на 9 процессоров (3 по X, 3 по Y); *б* — на 24 процессора (6 по X, 4 по Y)

В табл. 2 приведено количество точек на каждом процессоре при декомпозиции, изображенной на рис. 2.

Если посчитать по формуле (1) среднее количество ячеек $AvPt$, к которому должно стремиться количество ячеек для каждого процессора, то получим, что $AvPt \approx 4\,470$ точек. Из табл. 2 видно, что дисбаланс точек между процессорами менее 3 %.

Таблица 2

Количество точек на процессорах

Номер процессора	Кол-во ячеек	Номер процессора	Кол-во ячеек	Номер процессора	Кол-во ячеек
1	4 500	9	4 402	17	4 572
2	4 400	10	4 422	18	4 520
3	4 600	11	4 491	19	4 497
4	4 450	12	4 495	20	4 565
5	4 416	13	4 500	21	4 500
6	4 416	14	4 456	22	4 472
7	4 416	15	4 419	23	4 402
8	4 351	16	4 500	24	4 507

Балансировка вычислительной нагрузки процессоров на этапе записи-чтения разреза. Таймирование кода

Для любого параллельного комплекса программ актуальна задача балансировки вычислительной нагрузки процессоров в процессе счета задачи. Иногда заранее не известен характер течения рассчитываемой задачи, и поэтому возникают ситуации, когда некоторые процессоры могут оказаться более загруженными вычислениями, чем другие. На основании алгоритмов декомпозиции *полосы* и *клетки* можно утверждать, что набор ячеек на процессоры осуществлялся из предположения, что все ячейки рассчитываются за одно и то же время. Итоговое количество ячеек на процессорах было примерно одинаковым.

На практике достаточно часто приходится иметь дело с различными уравнениями состояния, иногда даже итерационными. Изменение загруженности процессоров возможно и на различных этапах выполнения счетного шага. Например, на каком-либо процессоре может реализовываться ситуация большего количества пересчитываемых точек при поддержании сетки. Это особенно сильно проявляется при расчете

струйных течений. Другой пример — миграция точек между соседними процессорами, которая тоже может занимать различное время в зависимости от количества таких точек.

В связи с этим авторами было предложено ввести весовую функцию каждого процессора, показывающую его загруженность. Чтобы инициализировать такие функции, необходимо создать подсистему таймирования. Для этого реализованный параллельный код комплекса необходимо разделить на блоки, отвечающие за различные этапы. Суммарное время счета каждого процессора можно подразделить на время, непосредственно затраченное на выполнение кода без операций синхронизации и обменов (чистое время), время ожидания процессора при обменах с соседними процессорами и синхронизации (время ожидания) и время, затраченное на сборку-разборку буферов (накладные расходы). Такие засечки времени элементарно реализуются в коде с помощью стандартной процедуры *MPI_ Wtime*() внутри каждого блока. В качестве весовой функции будем использовать чистое время счета процессора t_{pure}.

Основная идея метода балансировки нагрузки процессоров заключается в повторной декомпозиции всей задачи по процессорам с изменением количества счетных точек, принадлежащих процессору. В любом комплексе параллельных программ существует система записи разреза (контрольной точки), с которого в случае необходимости можно продолжить счет. Авторы предлагают видоизменить процедуру записи для реализации метода декомпозиции. Прежде всего необходимо записать чистое время счета каждого процессора в файл в момент записи разреза. В программной реализации использован текстовый файл с именем `Timing_Nstep.txt`, где под *Nstep* понимается номер шага, на котором происходит запись. В первой строке этого файла указывается количество используемых процессоров, а затем следуют строки с указанием значения чистого времени каждого процессора.

Не вдаваясь в подробности организации записи контрольной точки, рассмотрим структуру записи дополнительной информации, необходимой для проведения декомпозиции. Соберем некоторые интегральные данные о координатах точек всей задачи. Для этого будем использовать двоичный файл с именем `CoordPtProc_Nstep.bin`, который содержит координаты всех счетных точек задачи. Возможно, что эти данные уже записаны в разрез задачи. В этом случае процесс их чтения становится функцией самого комплекса. Тем не менее опишем структуру хранения данных в файле `CoordPtProc_Nstep.bin`. В первой строке указывается количество используемых процессоров *Kproc*, затем идут блоки данных о координатах точек каждого процессора. В начале каждого блока указывается номер процессора *Nproc* и количество счетных точек на нем *Npt*, а затем идут пары чисел (Xt, Yt) — координаты точек. Количество таких пар будет равно *Npt*, а количество блоков — *Kproc*. Вследствие того что файл `CoordPtProc_Nstep.bin` является дополнительным, но достаточно большим по объему, можно его сократить путем записи пар (Xt, Yt) в формате `real(4)`.

Наиболее естественным критерием для проведения балансировки нагрузки процессоров является превышение дисбаланса чистого времени счета какого-либо процессора относительно среднего значения чистого времени всех процессоров, рассчитанного как среднее арифметическое. Такой дисбаланс *MaxDisbProc* может быть задан в файле управления счетом в виде числа в процентном отношении. Например, $MaxDisbProc = 20$ будет означать, что если на каком-либо процессоре дисбаланс чистого времени счета превышает среднее время на 20 %, то необходимо произвести балансировку нагрузки процессоров. Пользователь должен минимально заботиться (либо вообще не знать) о том, что необходимо проводить балансировку. Такую работу должна брать на себя программа чтения контрольной точки. Пользователь может лишь определять частоту записи-чтения, во время которых и сможет реализовываться балансировка.

Усовершенствование алгоритмов декомпозиции *полосы* и *клетки*

Сначала подробно остановимся на усовершенствовании алгоритма декомпозиции *полосы*, а затем обобщим его на алгоритм *клетки*. Основная идея усовершенствования заключается в том, чтобы набирать ячейки на процессор не по среднему количеству ячеек *AvPt*, рассчитываемому по формуле (1), а по среднему чистому времени, рассчитываемому как среднее арифметическое всех чистых времен процессоров, записанных в файле `Timing_Nstep.txt`. Обозначим эту величину через T_ave. Будем предполагать, что время счета одной ячейки процессора T_{pt}^{j} вычис-

ляется по формуле

$$T_{pt}^{j} = \frac{T\_pure_j}{Npt_j}, \qquad (5)$$

где T_{pt}^{j} — время расчета одной ячейки j-м процессором; T_pure_j — чистое время расчета j-го процессора; Npt_j — количество ячеек, рассчитываемых j-м процессором. T_pure_j и Npt_j берутся из соответствующих записанных файлов.

Так как в алгоритме декомпозиции *полосы* набор ячеек происходит по элементарным x_k-полоскам, то необходимо инициализировать массив $TPol[NinX]$, в который будем заносить время счета элементарной полоски как сумму времен ячеек, попавших в эту полоску. Набирая такие полоски, будем суммировать времена счета каждой из них (в переменную $Tsum$) и оборвем процесс тогда, когда $Tsum \approx T\_ave$. В процессе набора полосок необходимо знать не только время счета одной ячейки и ее координаты, но и номер j-го процессора, на котором эта ячейка рассчитывалась. Такую информацию также можно получить из файла `CoordPtProc_Nstep.bin`. По мере формирования процессорной полосы можно инициализировать массив $Neigh\_Proc[numnod]$, в котором будут перечислены номера процессоров точек, попавших в набираемую полосу. Эта информация необходима для последующего чтения данных из контрольной точки и обновления массивов текущего процессора.

После проведения декомпозиции форма границ между областями, рассчитываемыми процессорами, становится прямолинейной и дисбаланс чистого времени расчета уменьшается.

Алгоритм декомпозиции *полосы* элементарно обобщается и на декомпозицию *клетки*. Как и ранее, будем применять алгоритм декомпозиции *полосы* в два этапа: сначала по направлению оси X, а затем по направлению оси Y. На первом этапе сформируем полосы с уравновешенным временем счета по X. При этом в каждой полосе будем стремиться ко времени $T\_ave\times$ $\times NumProc\_Y$. Затем, рассматривая каждую полосу независимо от другой и зная границы полос, сформированных на первом этапе, применяем алгоритм декомпозиции *полосы* вдоль другого направления. В данном случае необходимо инициализировать массив $Tcell[NinX, NinY]$, в который будем заносить время счета элементарного прямоугольника как сумму времен расчета ячеек, попавших в этот прямоугольник. Как и ранее, время элементарной полоски составляем из суммы времен элементарных прямоугольников, попавших в эту полоску.

Следует отметить, что алгоритмы декомпозиции *полосы* и *клетки* элементарно обобщаются и на более сложные конфигурации. Например, если задача изначально считалась на одном количестве процессоров, то с помощью декомпозиций, описанных выше, можно продолжить счет на другом количестве процессоров. На рис. 3 приведен пример, когда задача считалась на 5 процессорах, а после декомпозиции *клетки* она будет считаться на 20 процессорах.

В табл. 3, 4 приведен пример использования декомпозиции *полосы* для вышеупомянутой тестовой задачи [4], изображенной на рис. 1. Для таймирования кода была взята контрольная точка на момент времени $t = 1{,}6$ мс. Задача считалась на шести процессорах с алгоритмами поддержания сетки. В зоне перемешивания алгоритмы поддержания работают чаще, вследствие чего появляется несбалансированность вычислительной нагрузки процессоров. Общее количество точек в задаче было равным 23 619. Замеры проводились на 100 временных шагах. В табл. 3 приведены результаты счета с декомпозицией *полосы* в предположении, что все точки считаются с одинаковым весом, а в табл. 4 приведены результаты с таймированием кода и последующей декомпозицией данных на такое же число процессоров.

В результате применения балансировки вычислительной нагрузки время счета 100 шагов задачи уменьшилось с 38,4 до 34,6 с (на 10 %). Это произошло за счет перераспределения точек между процессорами (дисбаланс возрос с 1 до 26 %). В силу того, что балансировку вычислительной нагрузки предполагается осуществлять с определенной периодичностью, связанные с

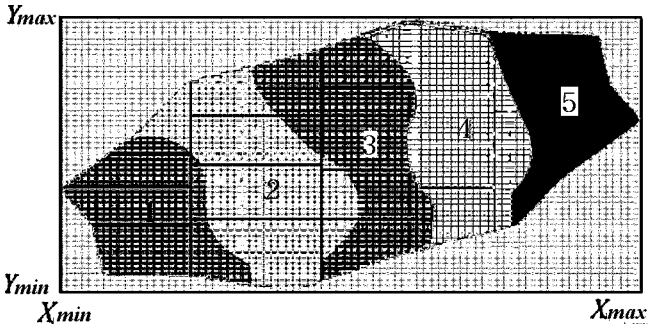


Рис. 3. Декомпозиция *клетки* для продолжения счета на 20 процессорах

Таблица 3

Время счета процессоров без балансировки

Номер процессора	*T_pure*, с	*T_wait*, с	*T_over*, с	Число точек	*T_ave*, с	Отклонение *T_pure* от среднего
1	35,0	3,3	0,1	3 929	26,66	8,34
2	28,8	9,4	0,2	3 895	26,66	2,14
3	28,2	10,0	0,2	3 962	26,66	1,54
4	23,6	14,6	0,1	3 968	26,66	−3,06
5	22,9	15,3	0,1	3 943	26,66	−3,76
6	21,5	16,7	0,1	3 922	26,66	−5,16

Таблица 4

Время счета процессоров с учетом балансировки

Номер процессора	*T_pure*, с	*T_wait*, с	*T_over*, с	Число точек	*T_ave*, с	Отклонение *T_pure* от среднего
1	26,6	7,9	0,1	2 915	25,88	0,72
2	24,7	9,7	0,2	3 404	25,88	−1,18
3	26,3	8,1	0,2	3 679	25,88	0,42
4	24,9	9,6	0,1	4 107	25,88	−0,98
5	26,3	8,2	0,1	4 626	25,88	0,42
6	26,5	8,0	0,1	4 888	25,88	0,62

ней накладные расходы окажутся незначительными.

Список литературы

1. *Глаголева Ю. П., Жогов Б. М., Киръянов Ю. Ф. и др.* Основы методики МЕДУЗА численного расчета двумерных нестационарных задач газодинамики // Числ. методы мех. спл. среды. 1972. Т. 3, № 2. С. 18—55.
2. *Болдырев С. Н., Леванов Е. И., Суков С. А., Якобовский М. В.* Обработка и хранение нерегулярных сеток большого размера на многопроцессорных системах // Сеточные методы для краевых задач и приложения. Мат. IV Всерос. семинара. Казань: Изд-во Казанского мат. общества, 2002. С. 33—39.
3. *Fiedler M.* Algebraic connectivity of graphs // Czechoslovak Mathematical Journal. 1973. No 23. P. 298—305.
4. *Барабанов Р. А., Бутнев О. И., Волков С. Г. и др.* Программа МЕДУЗА-3Д расчета трехмерных задач газовой динамики на нерегулярных сетках. Расчеты неустойчивости Рихтмайера—Мешкова // Сб. докл. Второй науч.-тех. конф. "Молодежь в науке". Саров, 2003. С. 13—18.