РОССИЙСКАЯ АКАДЕМИЯ НАУК
ОРДЕНА ЛЕНИНА
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
им.М.В.КЕЛДЫША

УДК 523.16

П.В. Кайгородов

ПРАКТИЧЕСКИЕ ВОПРОСЫ АДАПТАЦИИ ВЫЧИСЛИТЕЛЬНЫХ КОДОВ, ИСПОЛЬЗУЮЩИХ ЯВНЫЕ МЕТОДЫ, К МНОГОПРОЦЕССОРНОЙ АРХИТЕКТУРЕ

Москва – 2002

Аннотация

Рассматриваются различные вопросы технического характера, возникающие при адаптации вычислительных кодов, использующих явные методы, к многопроцессорной архитектуре. Для иллюстрации приводятся фрагменты вычислительного кода, написанного на языке Фортран с использованием библиотеки MPI. Рассмотрены основные проблемы, связанные с сохранением логики работы программы и исключением взаимных блокировок процессов, а так же вопросы производительности (в том числе балансировки нагрузки) и отладки параллельных кодов. Приведенные приемы и примеры программ могут быть использованы при написании и адаптации широкого спектра вычислительных кодов.

Abstract

We consider some technical problems dealing with numerical implementation of explicit methods for computers with multiprocessing architecture. We give few examples written in Fortran and using the MPI library. We also consider problems of processes blocking, increasing of productivity (including processes balancing) and debugging. We believe that these tricks and fragments of the code will be useful for numerical implementation of parallel programs for various problems.

Введение

Проблема адаптации вычислительных кодов к параллельной архитектуре весьма сложна и на сегодняшний день не имеет однозначного решения. Однако, можно выделить и рассмотреть несколько частных проблем, с которыми непременно придется столкнуться в процессе работы. Следует обратить внимание на следующие вопросы:

- Сохранение логики работы программы
- Недопущение взаимных блокировок процессов
- Балансировка нагрузки
- Вопросы производительности
- Вопросы отладки

Данная работа посвящена рассмотрению упомянутых проблем. В качестве примера представлен процесс распараллеливания вычислительного кода [1], написанного на языке Fortran. В качестве коммуникационной библиотеки используется MPI [2].

Рассматриваемый вычислительный код использует явную схему Роу-Ошера. Использование явной схемы предполагает отсутствие глобальной зависимости данных, что позволяет проще адаптировать код к параллельной архитектуре.

Существует несколько общепринятых алгоритмов организации работы параллельных программ. В данной статье будет рассмотрена одна из наиболее простых методик, обычно обозначаемая SIMD (Single Instruction – Multiply Data). Данная модель предполагает, что все параллельные процессы задачи используют один и тот же программный код, обрабатывая, тем не менее, различные участки общего массива данных (см. рис. 1). На рисунке представлена примерная диаграмма работы кода состоящего из

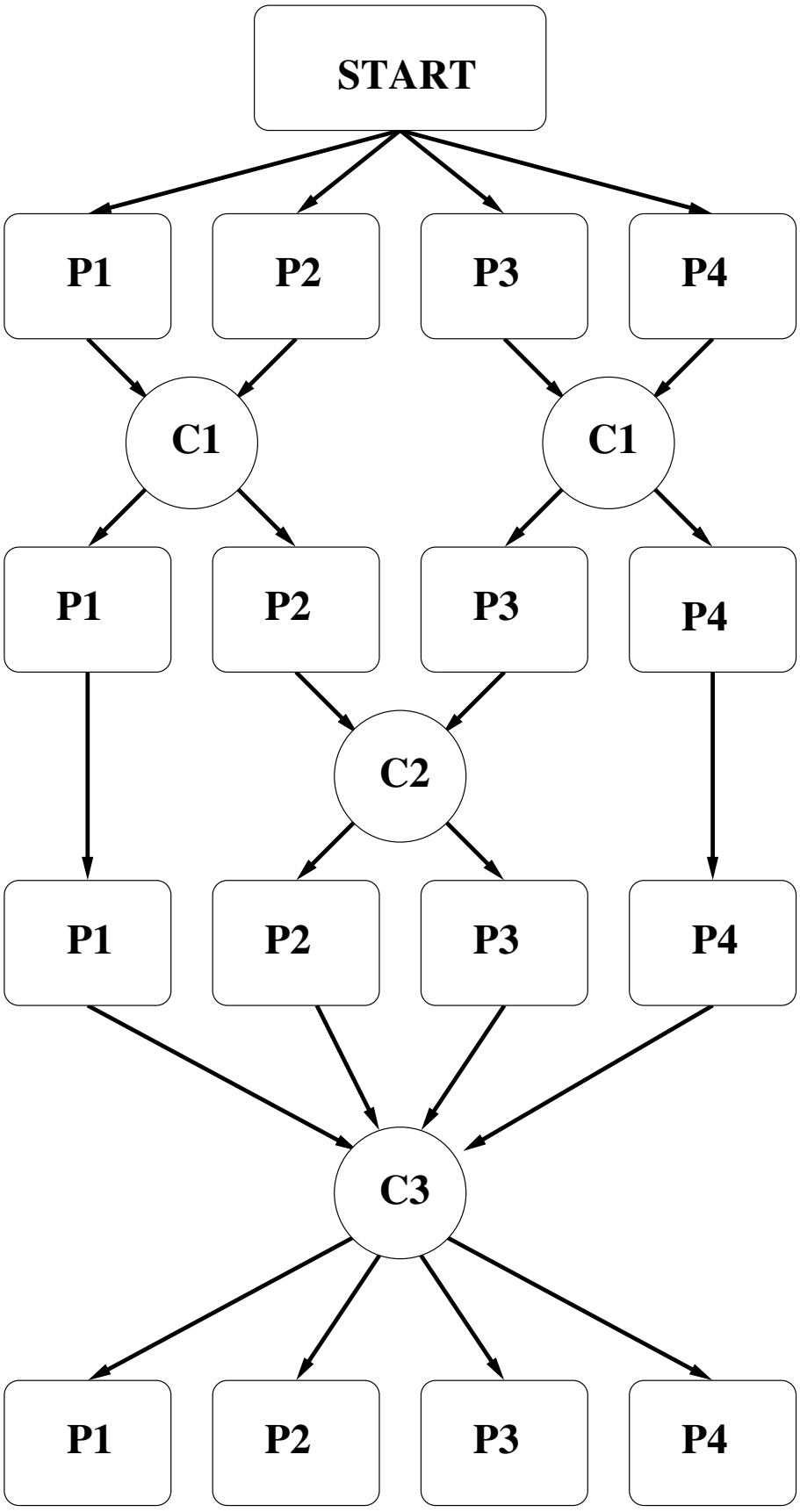

Рис. 1: Логика работы параллельной программы.

четырех процессов (P1,P2,P3,P4). Сначала (START) процессы абсолютно идентичны и выполняют одинаковый код. Затем происходит разделение (специализация) процессов. В дальнейшем процессы взаимодействуют друг с другом, что приводит к необходимости введения точек синхронизации (C1,C2,...).

В модели SIMD можно легко разрабатывать параллельные вычислительные коды, использующие разбиение вычислительной сетки на подобласти [3]. Далее будет рассматриваться именно такой тип распараллеливания.

1. Логика работы параллельной программы

Разделение ролей процессов становится возможным после того, как каждый процесс выяснит свое место в группе, то есть получит свой уникальный номер. В терминологии MPI такой номер называется рангом (rank) процесса.

При использовании MPI ранги всех процессов представляют собой последовательность чисел $(0, 1, 2, 3, \ldots)$, что позволяет на их основе однозначно распределить расчетную подобласть для каждого процесса.

Определить свой собственный ранг и число процессов в группе можно при помощи следующего кода:

```
CALL MPI_INIT(IERR)
CALL MPI_COMM_RANK(MPI_COMM_WORLD,MYID,IERR)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD,NUMPROCS,IERR)
```

Здесь MPI_INIT – процедура инициализации библиотеки MPI (должна быть вызвана перед использованием любых других MPI-процедур), MPI_COMM_RANK – процедура определения ранга, MPI_COMM_SIZE – процедура определения числа параллельных процессов. В результате в переменной MYID окажется число (MYID $\geqslant 0$) характеризующее положение процесса в группе, а в NUMPROCS – общее число процессов. Переменная

MPI_COMM_WORLD является входным служебным параметром для процедур MPI_COMM_RANK, MPI_COMM_SIZE и определена в соответствующем INCLUDE-файле. В случае неудачного завершения любой из этих процедур переменная IERR будет содержать ненулевой код ошибки.

Первым делом получим координаты подобласти в сетке процессов (подобласть считаем прямоугольным параллелепипедом с размерами $N_x \times N_y \times N_z$ ячеек, размеры общей области $N_x^{tot} \times N_y^{tot} \times N_z^{tot}$ ячеек).

Если мы делим область вдоль какой либо оси [3], координаты подобласти процесса MYID можно вычислить тривиальным образом. Единственное затруднение в данном случае возможно при некратности числа ячеек вдоль разбиваемой оси числу процессоров. Решить эту проблему можно двумя способами – выделением процессорам неодинакового числа ячеек, либо выбором разрешения в зависимости от числа процессоров. С точки зрения простоты и надежности второй путь предпочтительнее. Кроме того, он позволяет избегать дисбаланса нагрузки, приводящего к потере производительности (см. раздел “Вопросы производительности”).

Использование (в нашем случае) явной схемы с простым пятиточечным крестообразным шаблоном, приводит к необходимости введения двух фиктивных граничных ячеек с обеих сторон каждой виртуальной границы [3]. Рассмотрим сначала одномерное разбиение (см. рис. 5 из [3] или нижний ряд сетки процессов на рис. 2 данной работы). Пусть каждый процесс обрабатывает подобласть, состоящую из N_x ячеек по оси X. Фиктивные ячейки, окружающие виртуальные границы, должны быть исключены из подсчета ячеек, поэтому N_x^{tot} и N_x связаны как

$$N_x^{tot} = N_x \times P - 4(P-1)\,.$$

С учетом этого, реальные смещения зон можно вычислить по формуле:

$$X_{ofs} = Rank \times (N_x - 4) = Rank \times \frac{N_x^{tot} - 4}{P}\,,$$

где $Rank$ – ранг процесса, P – число процессоров.

Рассмотрим теперь двумерное разбиение [3]. В этом случае реальные смещения зон можно по вычислить (опять же с учетом фиктивных ячеек) по формулам:

$$X_{ofs} = Rank_x \times (N_x - 4) = Rank_x \times \frac{N_x^{tot} - 4}{P_x}\,,$$

$$Y_{ofs} = Rank_y \times (N_y - 4) = Rank_y \times \frac{N_y^{tot} - 4}{P_y}\,,$$

где P_x, P_y – число процессоров по осям X и Y, а $Rank_x$, $Rank_y$ – номера (ранги) процессов в двумерной сетке процессов:

$$Rank_x = Rank \bmod P_x\,, \qquad\qquad Rank_y = \frac{Rank - Rank_x}{P_x}\,.$$

Наглядно расположение зон можно видеть на рисунке 2.

Основываясь на общем числе процессов, размерах сетки и номере текущего процесса, можно определить координаты и размеры подобласти, принадлежащей данному процессу. Предполагается, что разбиение двумерное, число процессов вдоль координаты X хранится в переменной NPROC_X, вдоль координаты Y – соответственно в NPROC_Y:

```
MYID_X=MOD (MYID,NPROC_X)
MYID_Y=INT (MYID/NPROC_X)

ID_IM1=MYID_Y*NPROC_X+MYID_X-1
ID_IP1=MYID_Y*NPROC_X+MYID_X+1

ID_JM1=(MYID_Y-1)*NPROC_X+MYID_X
ID_JP1=(MYID_Y+1)*NPROC_X+MYID_X

IX_OFS=MYID_X*(NX-4)

IY_OFS=MYID_Y*(NY-4)
```

Здесь NX и NY – число ячеек [1)] вдоль осей X и Y, выделяемых каждому процессу. В результате в переменных MYID_X и MYID_Y оказываются координаты процесса в двумерной сетке процессов $Rank_x$, $Rank_y$

[1)] учитывая и фиктивные ячейки.

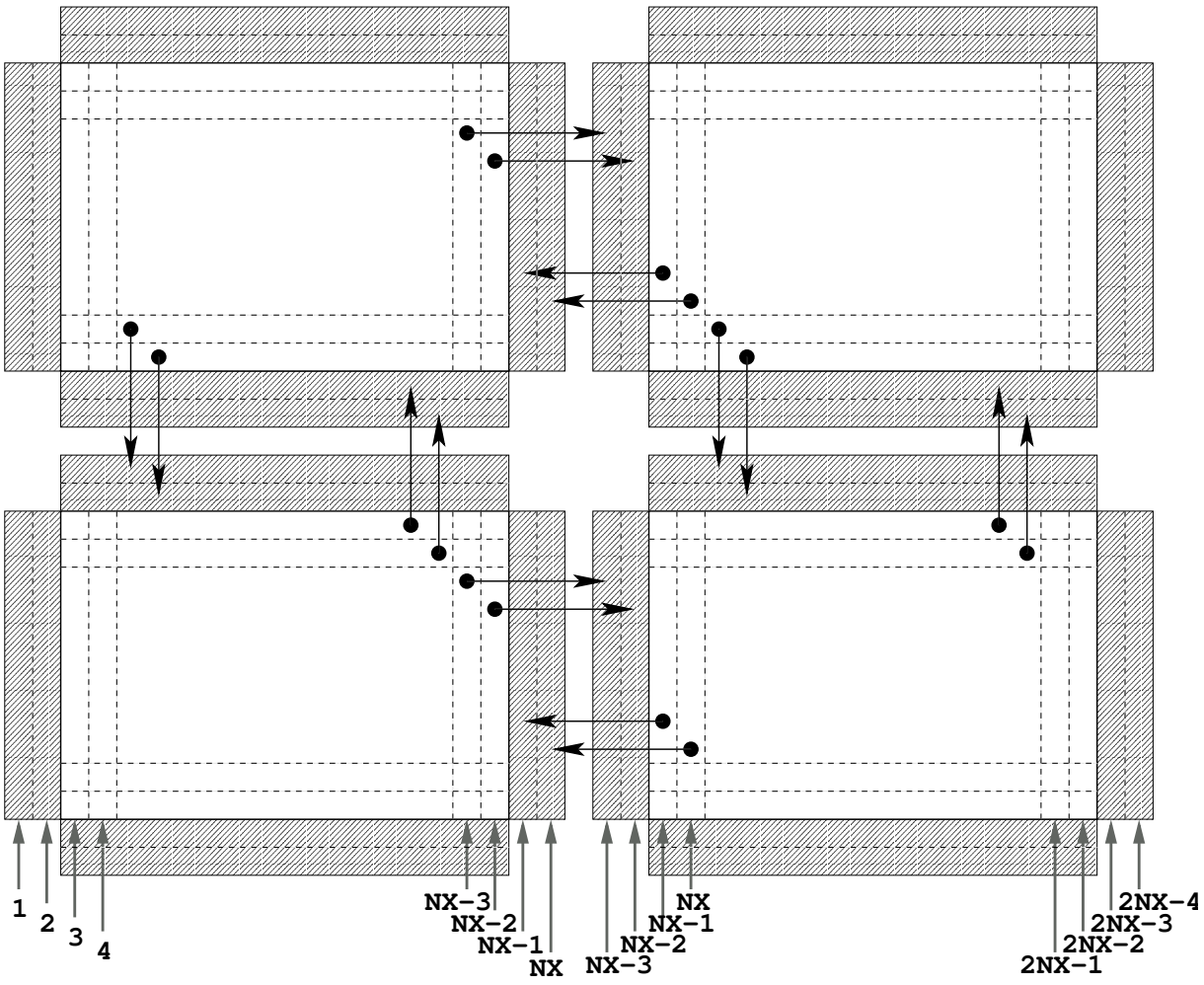

Рис. 2: Разбиение рабочей области.

$(0 \leqslant$ MYID_X $<$ NPROC_X, $0 \leqslant$ MYID_Y $<$ NPROC_Y$)$, а в переменных IX_OFS и IY_OFS – смещения (в ячейках) вычислительной подобласти относительно начала координат. Кроме того, в переменных ID_IM1, ID_IP1, ID_JM1, ID_JP1 оказываются ранги процессов, расположенных, соответственно слева, справа, снизу и сверху от данного процесса. Общее число ячеек результирующей сетки можно вычислить как:

X_TOT=(NX-4)*NPROC_X+4
Y_TOT=(NY-4)*NPROC_Y+4

Такой способ определения размеров сетки сильно упрощает определение координат, гарантирует, что всем процессам будет выделено одина-

ковое (целое) число ячеек, и позволяет статически выделять память под массивы на этапе компиляции.

Иногда геометрия задачи требуют выделения нечетного числа ячеек, в то время как наличие четного числа процессоров мешает это сделать. Например, в центре области должна находиться ячейка с особыми свойствами, причем ее строго центральное положение важно. В некоторых случаях обойти это затруднение можно путем варьирования физических размеров области. Другими словами, можно выделить четное число ячеек (на одну больше чем нужно), но размер ячеек подобрать таким образом, чтобы координаты "особой" ячейки не изменились, несмотря на (возможное) изменение ее номера. В некоторых случаях, можно исключить из расчетов "лишнюю" ячейку на краю области, при этом слегка снизится нагрузка на некоторые процессоры, что не должно отразиться на общей производительности кода.

Коснемся теперь проблемы задания начальных данных. Допустим, что начальные данные считываются из файла. Допустим также, что различным процессам нужны различные начальные данные. Наиболее простым решением данной проблемы является создание файлов с начальными данными отдельно для каждого процесса:

```
      CHARACTER*16 NAME
      WRITE (NAME,100) MYID
100   FORMAT ('DATA.',I2.2)
      OPEN (1,FILE=NAME,FORM='UNFORMATTED')
C     ...READING DATA FROM FILE...
      CLOSE (1)
```

Правда, при расчетах с большим числом процессоров количество файлов может быть весьма большим, что отрицательно скажется на производительности дисковой подсистемы (особенно, если все эти файлы будут считываться/записываться одновременно). Выходом может служить работа всех процессов с одним файлом, но это не всегда является возможным

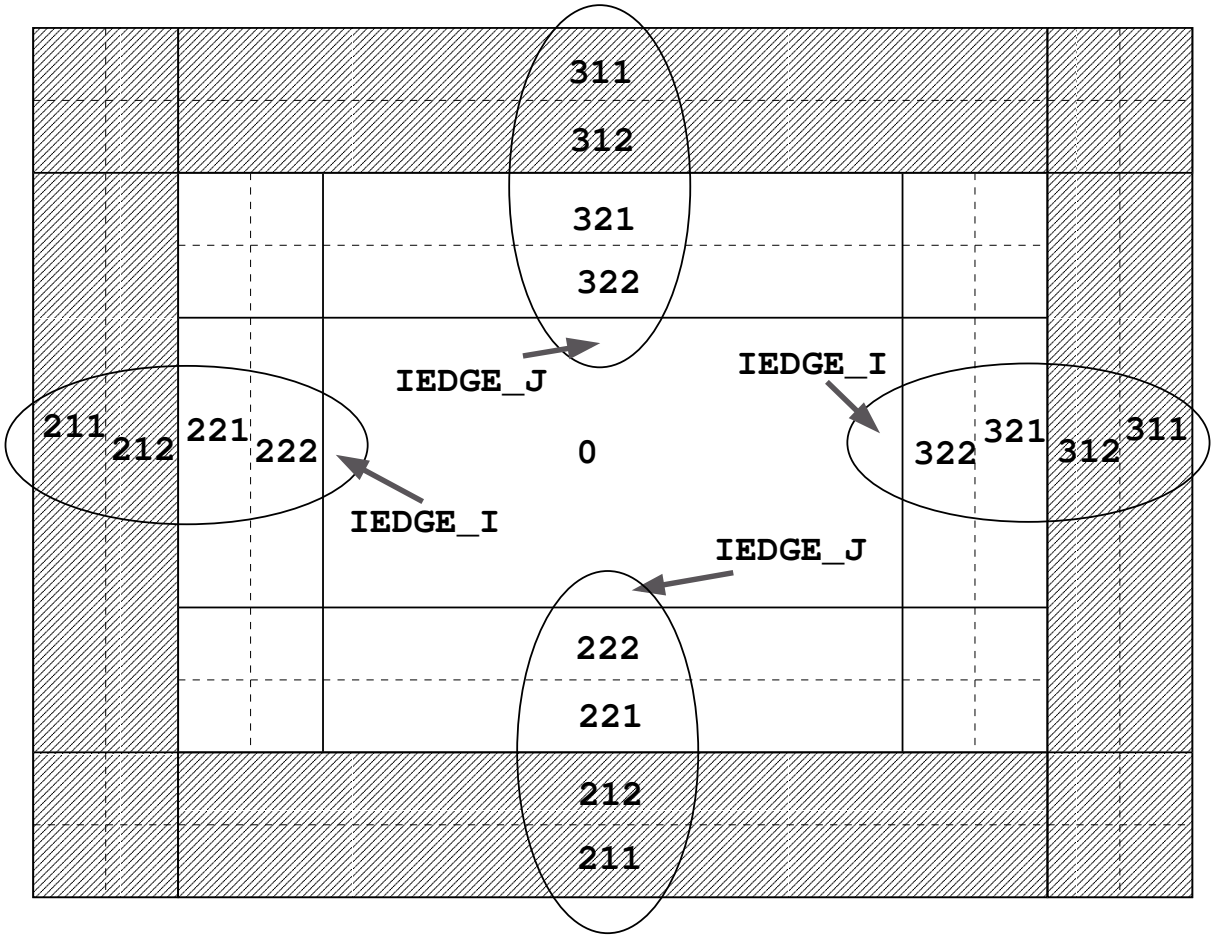

Рис. 3: Возвращаемые значения IEDGE_I, IEDGE_J.

(в частности, для этого дисковая подсистема должна разделяться между всеми узлами кластера).

Кроме начальных данных процессы могут отличаться друг от друга и другими параметрами. В нашем случае существенную роль играют только размеры и координаты подобластей, обсчитываемых отдельными процессорами.

Для сшивки решений на границах подобластей необходимо вводить особый тип граничных условий [3]. Наложение этих условий предполагает пересылку значений из приграничных ячеек соседнего процесса и помещение их в соответствующие “заграничные” ячейки (на рис. 2 эти зоны обозначены серым цветом, а направления пересылок – черными стрелками).

Пример кода, обеспечивающего такую пересылку, приведен в Приложении 1. В приведенном примере описана процедура BVAL и три функции: IEDGE_I, IEDGE_J и IEDGE_K. [2)] В процедуре BVAL переменные I, J и K проходят в цикле все возможные значения координат ячеек внутри подобласти. Для каждой ячейки вызываются функции IEDGE_I, IEDGE_J, IEDGE_K для определения, является ли ячейка "приграничной" или "заграничной", в последнем случае определяется тип границы – реальная, либо межпроцессорная. Данные функции имеют один аргумент – значения координаты вдоль соответствующей оси. Возвращаемые значения определяют тип ячейки: 0 – внутренняя ячейка, 1 – внешняя граница области, остальные значения для IEDGE_I и IEDGE_J показаны на рис. 3. Так как по третьей оси разбиения не производится, IEDGE_K может возвращать только 0 или 1.

При первом прогоне определяются ячейки, которые необходимо передать соседним процессам. Значения переменных, соответствующие данным ячейкам, записываются в промежуточные буфера SEND_BUF[1,2,3,4] для 4-х границ [3)]. Для простоты в данном примере обрабатывается только переменная (массив) RHO, но код можно легко изменить для пересылки нескольких переменных.

Далее происходит обмен при помощи вызовов функций MPI_ISEND и MPI_IRECV. Процессы дожидаются окончания пересылки, вызывая MPI_WAITALL (проверка нужна для того, чтобы корректно мог работать однопроцессорный вариант при NPROC_X=1 и NPROC_Y=1).

После завершения пересылок массивы RECV_BUF[1,2,3,4] содержат принятые от соседних процессов данные, которые нужно поместить в соответствующие ячейки, что и делается во втором вложенном цикле по I, J, K.

[2)] предполагается, что переменные MYID_X, MYID_Y, ID_IM1, ..., ID_IP2 уже определены.

[3)] каждый буфер представляет собой двойной массив (см. первую размерность четырехмерных массивов SEND_BUF, RECV_BUF), что связано с использованием в данном примере пятиточечного шаблона и, следовательно, необходимостью передачи двух слоев данных соседнему процессу.

Кроме обмена приграничными ячейками существует необходимость синхронизации и других значений. Для того, чтобы решения для различных подобластей могли сшиваться, необходимым условием является равенство временно́го шага τ, используемого при вычислениях. Подробно этот вопрос рассматривался в [3].

Для того, чтобы выбор общего τ был эквивалентен выбору в однопроцессорном варианте кода, нужно использовать минимальное значение τ из значений, полученных отдельными процессами, после чего распространить выбранное значение среди всех процессов. В библиотеке MPI это действие может реализовываться при помощи вызова процедуры MPI_ALLREDUCE:

```
      CALL MPI_ALLREDUCE(TAU,TAU1,1,MPI_REAL,
     &                   MPI_MIN,MPI_COMM_WORLD,IERR)
      TAU=TAU1
```

Здесь TAU – входной параметр, поставляемый каждым процессом, TAU1 – выходной параметр, содержащий минимальное по всем процессам значение τ, 1 – длина передаваемого параметра, MPI_REAL, MPI_MIN, MPI_COMM_WORLD – входные служебные параметры, определенные в соответствующем INCLUDE-файле.

2. Взаимные блокировки

Касаясь вопросов работы параллельной программы, нельзя не упомянуть одну из наиболее неприятных ошибок, приводящую к взаимной блокировке процессами друг друга, что влечет за собой остановку работы всей программы.

Ниже приведен пример кода, приводящего к такой блокировке (по крайней мере в реализации $LAM/$MPI [4]):

```
      IF (MYID_X.GT.0) THEN
```

```
      CALL MPI_SEND(SEND_BUF1,IBUF_LEN,MPI_REAL,
&           ID_IM1,308,MPI_COMM_WORLD,IERR)
      CALL MPI_RECV(RECV_BUF1,IBUF_LEN,MPI_REAL,
&           ID_IM1,309,MPI_COMM_WORLD,ISTAT,IERR)
   END IF
   IF (MYID_X.LT.NPROC_X-1) THEN
      CALL MPI_SEND(SEND_BUF2,IBUF_LEN,MPI_REAL,
&           ID_IP1,309,MPI_COMM_WORLD,IERR)
      CALL MPI_RECV(RECV_BUF2,IBUF_LEN,MPI_REAL,
&           ID_IP1,308,MPI_COMM_WORLD,ISTAT,IERR)
   END IF
```

Причина взаимной блокировки в данном случае – одновременный вызов оператора MPI_SEND всеми процессами. Вызов MPI_SEND не может завершиться до окончания передачи данных, для чего необходим вызов MPI_RECV принимающей стороной. Так как процесс-приемник так же находится в ожидании завершения MPI_SEND, все процессы блокируются. Исправить эту ситуацию можно, изменив порядок вызовов MPI_SEND и MPI_RECV в различных процессах:

```
   IF (MYID_X.GT.0) THEN
      CALL MPI_SEND(SEND_BUF1,IBUF_LEN,MPI_REAL,
&           ID_IM1,308,MPI_COMM_WORLD,IERR)
   END IF
   IF (MYID_X.LT.NPROC_X-1) THEN
      CALL MPI_RECV(RECV_BUF2,IBUF_LEN,MPI_REAL,
&           ID_IP1,308,MPI_COMM_WORLD,ISTAT,IERR)
   END IF
   IF (MYID_X.GT.0) THEN
      CALL MPI_RECV(RECV_BUF1,IBUF_LEN,MPI_REAL,
&           ID_IM1,309,MPI_COMM_WORLD,ISTAT,IERR)
   END IF
   IF (MYID_X.LT.NPROC_X-1) THEN
      CALL MPI_SEND(SEND_BUF2,IBUF_LEN,MPI_REAL,
&           ID_IP1,309,MPI_COMM_WORLD,IERR)
   END IF
```

Переменные во этих примерах имеют следующий смысл: SEND_BUF и RECV_BUF – передаваемый/получаемый массив, IBUF_LEN – его длина, ID_IM, ID_IP – ранг (номер) процесса, которому посылается (от которого получается) информация, 308 (309) – тэг (tag) пересылки (произвольное число, но на обеих концах пересылки его значение должно быть одним и тем же), ISTAT – переменная, содержащая дополнительную информацию в случае возникновения ошибки.

Кроме того, можно (и это более предпочтительный способ) использовать асинхронные вызовы MPI:

```
      DIMENSION ISTATS(4),IRESC(4)
      INR=1
      IF (MYID_X.GT.0) THEN
            CALL MPI_ISEND(SEND_BUF1,IBUF_LEN,MPI_REAL,
     &            ID_IM1,308,MPI_COMM_WORLD,ISTATS(INR),IERR)
            INR=INR+1
            CALL MPI_IRECV(RECV_BUF1,IBUF_LEN,MPI_REAL,
     &            ID_IM1,309,MPI_COMM_WORLD,ISTATS(INR),IERR)
            INR=INR+1
      END IF
      IF (MYID_X.LT.NPROC_X-1) THEN
            CALL MPI_ISEND(SEND_BUF2,IBUF_LEN,MPI_REAL,
     &            ID_IP1,309,MPI_COMM_WORLD,ISTATS(INR),IERR)
            INR=INR+1
            CALL MPI_IRECV(RECV_BUF2,IBUF_LEN,MPI_REAL,
     &            ID_IP1,308,MPI_COMM_WORLD,ISTATS(INR),IERR)
            INR=INR+1
      END IF
      IF (INR.GT.1) CALL MPI_WAITALL(INR-1,ISTATS,IRESC,IERR)
```

При каждой пересылке, выполняемой с помощью процедур MPI_ISEND, MPI_IRECV, в массив ISTATS записывается дескриптор пересылки. Процедура MPI_WAITALL приостанавливает работу программы до тех пор, пока все пересылки, чьи дескрипторы записаны в массиве ISTATS, не будут завершены. Массив IRESC содержит коды завершения пересылок.

3. Вопросы производительности

Оценка производительности вычислительного кода является весьма нетривиальной задачей. Чтобы определить реальную производительность, необходимо знать особенности как программной реализации алгоритмов (в том числе и MPI), так и аппаратную часть компьютера.

Для измерения производительности параллельных вычислительных кодов удобно использовать следующие величины:

Ускорение: $S_N = T_N/T_1$

Эффективность: $E_N = S_N/N$

Здесь N – число процессоров, задействованных при счете задачи, T_N – время счета задачи на N процессорах.

Ограничимся рассмотрением вычислительного кластера на базе Intel/Linux/FastEthernet. Особенности процессоров Intel и различных компиляторов для них заслуживают отдельного рассмотрения, далеко выходящего за рамки данной работы. Для оценки производительности наиболее важна такая техническая характеристика процессора как наличие кэш-памяти (cache). Кэш-память представляет собой некую "сверхоперативную" память, используемую для промежуточного хранения данных. Скорость доступа процессора к данным, хранящимся в кэш-памяти в несколько раз выше, чем к основной оперативной памяти. При обращении к памяти процессор сперва проверяет наличие нужных данных в кэш-памяти. В случае отсутствия таковых процессор переходит в состояние ожидания, а данные подгружаются в кэш-память из оперативной памяти.

Влияние кэш-памяти особенно заметно при изменении параметров задачи. Как правило, большинство счетных задач сводится к обработке больших массивов данных, хранящихся в ОЗУ. От размера этих массивов (и, в некоторой степени, от последовательности доступа к их элементам) напрямую зависит эффективность [4)] работы кэш-памяти, что в значи-

[4)] под эффективностью кэш-памяти следует понимать процент попаданий в кэш при запросе данных процессором.

тельной мере может определять быстродействие. При распараллеливании вычислительных кодов методом разбиения вычислительной сетки между процессами размер массивов, обрабатываемых каждым процессом, напрямую зависит от числа процессов. Это означает, что при увеличении числа процессоров (узлов кластера) вдвое, сетка, выделяемая каждому процессу, уменьшится примерно [5)] вдвое. Соответственно, эффективность работы кэш-памяти возрастет, что приведет к росту производительности в целом. Именно влиянием кэш-памяти объясняются некоторые, на первый взгляд парадоксальные, явления, как-то получение значений эффективности, превышающих 100%.

Отдельную сложность представляет распределение нагрузки. Если параллельный код имеет точки взаимной синхронизации процессов (где полностью асинхронный обмен данными невозможен), неизбежны некоторые задержки исполнения, связанные с неодновременностью достижения этих точек. При неравномерной загрузке процессов, участвующих в обмене, наиболее загруженный процесс на каждом цикле будет подходить к точке обмена с опозданием, вынуждая менее загруженные процессы ожидать его, простаивая. Если в обмене участвуют все процессы, общая производительность может быть оценена как производительность наиболее загруженного процесса, умноженная на число процессов. Для многих вычислительных кодов возможно произвести балансировку нагрузки вручную, на этапе компиляции. Например, для вычислительного кода типа [1] достаточно разделить сетку поровну между процессами. В некоторых случаях это невозможно, так как (в общем случае) число ячеек для заданного измерения сетки может быть не кратно соответствующему числу процессоров. В данном случае придется либо подгонять число ячеек, либо мириться с небольшим дисбалансом. Очевидно, можно разделить сетку меду процессами таким образом, что части из них (например, процес-

[5)] надо помнить еще и о “заграничных” ячейках, число которых зависит только от используемой численной схемы.

сам с рангами $0 \ldots \mathrm{mod}(N/P)$ [6]) достанутся фрагменты сетки на одну ячейку (столбец, слой) больше, чем остальным. Соответственно, изменится алгоритм определения координат фрагмента сетки, принадлежащей процессу P_N.

4. Вопросы отладки

Отладка параллельной программы является особенно сложной задачей. Несмотря на то, что существует множество специальных инструментальных средств, позволяющих производить отладку параллельных задач, часто они либо недоступны, либо их использование затруднено (например, по причине медленного соединения с машиной, на которой производится запуск). Кроме того, на многих крупных вычислительных машинах невозможен интерактивный запуск программ пользователями. Таким образом, единственным доступным способом отладки часто является вставка операторов вывода отладочной информации и анализ результатов прогона. Однако, при работе параллельной программы вывод данных будут производить одновременно все процессы, что затрудняет чтение результатов, даже если при выводе указывается ранг процесса:

```
WRITE (*,*) MYID,': SOME OUTPUT'
```

Частой ошибкой является нарушение логики работы некоторых процессов, в результате чего точки обмена данными достигаются существенно неодновременно, либо не в том порядке. Для обнаружения места, где происходит сбой, можно использовать вызов MPI_BARRIER в сочетании с операторами вывода.

```
CALL MPI_BARRIER(MPI_COMM_WORLD,IERR)
WRITE (*,*) MYID,': CHECKPOINT PASSED'
```

[6]) где N и P – соответственно число ячеек и процессоров в разделяемом направлении.

MPI_BARRIER создает дополнительную точку синхронизации и обеспечивает продолжение выполнения только после достижения этой точки всеми процессами. Расставив приведенные выше конструкции в подозрительных частях программы, можно локализовать место ошибки.

Приведенный метод может использоваться и для обнаружения ошибок, вызывающих аварийное завершение одного из процессов. Часто такое завершение влечет за собой остановку системой всех остальных процессов, что затрудняет обнаружение ошибки, так как в тот момент разные процессы могут выполнять разные участки кода.

5. Практический пример

Рассмотрим задачу распространения тепла в кубической области с постоянными граничными условиями. Эта задача описывается уравнением теплопроводности (для простоты коэффициент теплопроводности считается равным 1):

$$\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2},$$

$$T(x,y,z)|_{t=0} = T_0(x,y,z),$$

$$T(x,y,z)|_{bound} = T_0(x,y,z).$$

Для решения этого уравнения используем явную разностную схему

$$\frac{\hat{T} - T}{\tau} = \frac{T^{i+} - 2T + T^{i-}}{h_x^2} + \frac{T^{j+} - 2T + T^{j-}}{h_y^2} + \frac{T^{k+} - 2T + T^{k-}}{h_z^2}$$

(для внутренних ячеек сетки),

$$\hat{T} = T$$

(для граничных ячеек сетки).

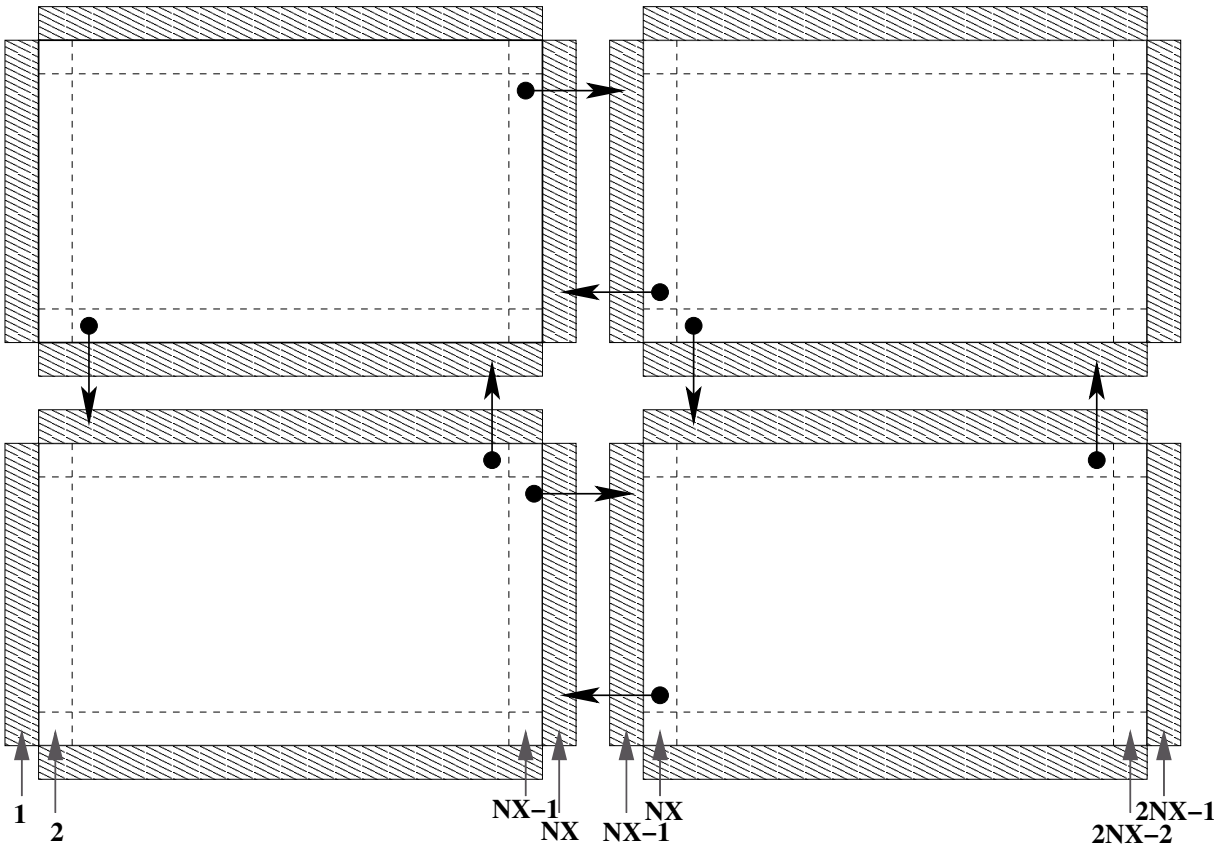

Рис. 4: Разбиение рабочей области для задачи распространения тепла.

Примеры численной реализации этого алгоритма для однопроцессорного компьютера и компьютера с массивно-параллельной архитектурой приведены в Приложениях 2 и 3 (приведена только схема работы программы, в частности, опущены фрагменты кода, связанные с присвоением начальных данных, определением h_x, h_y, h_z, τ, сохранением результатов и т.д.). Для последовательного варианта программы (Приложение 2) размер вычислительной сетки равен NX × NY × NZ. Для параллельного варианта (Приложение 3) эта тройка дает размер части вычислительной сетки, отводимой для каждого процессора, общее же число таких частей равняется NPROC_X × NPROC_Y (см. рис. 4). Отметим также, что в данном примере каждую виртуальную границу окружает с каждой стороны только один слой ячеек. Соответственно, общее число ячеек и смещения зон вычисляются в этом случае по формулам

$$N_x^{tot} = N_x \times P_x - 2(P_x - 1)\,,$$

$$N_y^{tot} = N_y \times P_y - 2(P_y - 1)\,,$$

$$X_{ofs} = Rank_x \times (N_x - 2) = Rank_x \times \frac{N_x^{tot} - 2}{P_x}\,,$$

$$Y_{ofs} = Rank_y \times (N_y - 2) = Rank_y \times \frac{N_y^{tot} - 2}{P_y}\,.$$

Настоящая работа выполнена при поддержке Российского фонда фундаментальных исследований (гранты №№ 02-02-16088, 02-02-17642), грантов Президента РФ №№ 00-15-96722, 02-15-99311, ФЦНТП “Астрономия”, а также Программы Президиума РАН “Математическое моделирование”. Автор признателен О.А.Кузнецову и Д.В.Бисикало за полезные обсуждения.

Приложение 1

```
      SUBROUTINE BVAL
      PARAMETER (NX=...,NY=...,NZ=...,NPROC_X=...,NPROC_Y=...)
      COMMON /ID/     MYID,MYID_X,MYID_Y
      COMMON /ID1/    ID_IM1,ID_IP1,ID_JM1,ID_JP1
      COMMON /D/      RHO(NX,NY,NZ),
      DIMENSION       SEND_BUF1(2,NY,NZ,1),SEND_BUF2(2,NY,NZ,1),
     &                RECV_BUF1(2,NY,NZ,1),RECV_BUF2(2,NY,NZ,1),
     &                SEND_BUF3(2,NX,NZ,1),SEND_BUF4(2,NX,NZ,1),
     &                RECV_BUF3(2,NX,NZ,1),RECV_BUF4(2,NX,NZ,1)
      DIMENSION       IRESC(8),ISTATS(8)
      DO I=1,NX
      DO J=1,NY
```

```
      DO K=1,NZ
            IE_I=IEDGE_I(I)
            IE_J=IEDGE_J(J)
            IE_K=IEDGE_K(K)
            IF ((IE_I.EQ.1).OR.(IE_J.EQ.1).OR.(IE_K.EQ.1)) THEN
C     REAL BOUNDS
            ELSE
                  IF (IE_I.EQ.221) SEND_BUF1(1,J,K,1)=RHO(I,J,K)
                  IF (IE_I.EQ.222) SEND_BUF1(2,J,K,1)=RHO(I,J,K)
                  IF (IE_I.EQ.321) SEND_BUF2(2,J,K,1)=RHO(I,J,K)
                  IF (IE_I.EQ.322) SEND_BUF2(1,J,K,1)=RHO(I,J,K)
                  IF (IE_J.EQ.221) SEND_BUF3(1,I,K,1)=RHO(I,J,K)
                  IF (IE_J.EQ.222) SEND_BUF3(2,I,K,1)=RHO(I,J,K)
                  IF (IE_J.EQ.321) SEND_BUF4(2,I,K,1)=RHO(I,J,K)
                  IF (IE_J.EQ.322) SEND_BUF4(1,I,K,1)=RHO(I,J,K)
            END IF
      END DO
      END DO
      END DO

      INR=1
      IF (MYID_X.GT.0) THEN
       CALL MPI_ISEND(SEND_BUF1,NY*NZ*2,MPI_REAL,
     &       ID_IM1,308,MPI_COMM_WORLD,ISTATS(INR),IERR)
       INR=INR+1
       CALL MPI_IRECV(RECV_BUF1,NY*NZ*2,MPI_REAL,
     &       ID_IM1,309,MPI_COMM_WORLD,ISTATS(INR),IERR)
       INR=INR+1
      END IF
      IF (MYID_X.LT.NPROC_X-1) THEN
       CALL MPI_ISEND(SEND_BUF2,NY*NZ*2,MPI_REAL,
     &       ID_IP1,309,MPI_COMM_WORLD,ISTATS(INR),IERR)
       INR=INR+1
       CALL MPI_IRECV(RECV_BUF2,NY*NZ*2,MPI_REAL,
     &       ID_IP1,308,MPI_COMM_WORLD,ISTATS(INR),IERR)
       INR=INR+1
      END IF
      IF (MYID_Y.GT.0) THEN
       CALL MPI_ISEND(SEND_BUF3,NX*NZ*2,MPI_REAL,
     &       ID_JM1,408,MPI_COMM_WORLD,ISTATS(INR),IERR)
```

```
      INR=INR+1
      CALL MPI_IRECV(RECV_BUF3,NX*NZ*2,MPI_REAL,
     &        ID_JM1,409,MPI_COMM_WORLD,ISTATS(INR),IERR)
      INR=INR+1
    END IF
    IF (MYID_Y.LT.NPROC_Y-1) THEN
      CALL MPI_ISEND(SEND_BUF4,NX*NZ*2,MPI_REAL,
     &        ID_JP1,409,MPI_COMM_WORLD,ISTATS(INR),IERR)
      INR=INR+1
      CALL MPI_IRECV(RECV_BUF4,NX*NZ*2,MPI_REAL,
     &        ID_JP1,408,MPI_COMM_WORLD,ISTATS(INR),IERR)
      INR=INR+1
    END IF
    IF (INR.GT.1) CALL MPI_WAITALL(INR-1,ISTATS,IRESC,IERR)

    DO I=1,NX
    DO J=1,NY
    DO K=1,NZ
          IE_I=IEDGE_I(I)
          IE_J=IEDGE_J(J)
          IE_K=IEDGE_K(K)
          IF ((IE_I.EQ.1).OR.(IE_J.EQ.1).OR.(IE_K.EQ.1)) THEN
          ELSE
                IF (IE_I.EQ.211) RHO(I,J,K)=RECV_BUF1(1,J,K,1)
                IF (IE_I.EQ.212) RHO(I,J,K)=RECV_BUF1(2,J,K,1)
                IF (IE_I.EQ.311) RHO(I,J,K)=RECV_BUF2(2,J,K,1)
                IF (IE_I.EQ.312) RHO(I,J,K)=RECV_BUF2(1,J,K,1)
                IF (IE_J.EQ.211) RHO(I,J,K)=RECV_BUF3(1,I,K,1)
                IF (IE_J.EQ.212) RHO(I,J,K)=RECV_BUF3(2,I,K,1)
                IF (IE_J.EQ.311) RHO(I,J,K)=RECV_BUF4(2,I,K,1)
                IF (IE_J.EQ.312) RHO(I,J,K)=RECV_BUF4(1,I,K,1)
          END IF
    END DO
    END DO
    END DO
    END

    FUNCTION IEDGE_I(I)
    PARAMETER (NX=...,NY=...,NZ=...,NPROC_X=...,NPROC_Y=...)
    COMMON /ID/     MYID,MYID_X,MYID_Y
```

```
IEDGE_I=0
IF (I.EQ.1.AND.MYID_X.EQ.0) IEDGE_I=1
IF (I.EQ.NX.AND.MYID_X.EQ.NPROC_X-1) IEDGE_I=1
IF (I.EQ.1.AND.MYID_X.NE.0) IEDGE_I=211
IF (I.EQ.2.AND.MYID_X.NE.0) IEDGE_I=212
IF (I.EQ.3.AND.MYID_X.NE.0) IEDGE_I=221
IF (I.EQ.4.AND.MYID_X.NE.0) IEDGE_I=222
IF (I.EQ.NX.AND.MYID_X.NE.NPROC_X-1) IEDGE_I=311
IF (I.EQ.NX-1.AND.MYID_X.NE.NPROC_X-1) IEDGE_I=312
IF (I.EQ.NX-2.AND.MYID_X.NE.NPROC_X-1) IEDGE_I=321
IF (I.EQ.NX-3.AND.MYID_X.NE.NPROC_X-1) IEDGE_I=322
END

FUNCTION IEDGE_J(J)
PARAMETER (NX=...,NY=...,NZ=...,NPROC_X=...,NPROC_Y=...)
COMMON /ID/     MYID,MYID_X,MYID_Y

IEDGE_J=0
IF (J.EQ.1 .AND.MYID_Y.EQ.0) IEDGE_J=1
IF (J.EQ.NY.AND.MYID_Y.EQ.NPROC_Y-1) IEDGE_J=1
IF (J.EQ.1.AND.MYID_Y.NE.0) IEDGE_J=211
IF (J.EQ.2.AND.MYID_Y.NE.0) IEDGE_J=212
IF (J.EQ.3.AND.MYID_Y.NE.0) IEDGE_J=221
IF (J.EQ.4.AND.MYID_Y.NE.0) IEDGE_J=222
IF (J.EQ.NY.AND.MYID_Y.NE.NPROC_Y-1) IEDGE_J=311
IF (J.EQ.NY-1.AND.MYID_Y.NE.NPROC_Y-1) IEDGE_J=312
IF (J.EQ.NY-2.AND.MYID_Y.NE.NPROC_Y-1) IEDGE_J=321
IF (J.EQ.NY-3.AND.MYID_Y.NE.NPROC_Y-1) IEDGE_J=322

END

FUNCTION IEDGE_K(K)
PARAMETER (NX=...,NY=...,NZ=...,NPROC_X=...,NPROC_Y=...)
COMMON /ID/     MYID,MYID_X,MYID_Y

IEDGE_K=0
IF (K.EQ.1) IEDGE_K=1
IF (K.EQ.NZ) IEDGE_K=1

END
```

Приложение 2

```
      PARAMETER (NX=...,NY=...,NZ=...)
      DIMENSION TEMP(NX,NY,NZ),TEMP1(NX,NY,NZ)
      DO N=1,NTIM
      DO I=2,NX-1
      DO J=2,NY-1
      DO K=2,NZ-1
      TEMP1(I,J,K)=TEMP(I,J,K)+TAU*(
     &        (TEMP(I+1,J,K)-2*TEMP(I,J,K)+TEMP(I-1,J,K))/HX**2
     &       +(TEMP(I,J+1,K)-2*TEMP(I,J,K)+TEMP(I,J-1,K))/HY**2
     &       +(TEMP(I,J,K+1)-2*TEMP(I,J,K)+TEMP(I,J,K-1))/HZ**2)
      END DO
      END DO
      END DO
      DO I=2,NX-1
      DO J=2,NY-1
      DO K=2,NZ-1
      TEMP(I,J,K)=TEMP1(I,J,K)
      END DO
      END DO
      END DO
      END DO
      END
```

Приложение 3

```
      PARAMETER (NX=...,NY=...,NZ=...,NPROC_X=...,NPROC_Y=...)
      DIMENSION TEMP(NX,NY,NZ),TEMP1(NX,NY,NZ)
      DIMENSION RBND_X1(NY,NZ),RBND_X2(NY,NZ)
      DIMENSION RBND_Y1(NX,NZ),RBND_Y2(NX,NZ)
      DIMENSION SBND_X1(NY,NZ),SBND_X2(NY,NZ)
      DIMENSION SBND_Y1(NX,NZ),SBND_Y2(NX,NZ)
```

```
      DIMENSION ISTATS(8),IRESC(8)

      CALL MPI_INIT(IERR)
      CALL MPI_COMM_RANK(MPI_COMM_WORLD,MYID,IERR)
      CALL MPI_COMM_SIZE(MPI_COMM_WORLD,NUMPROCS,IERR)

      NPROC=NPROC_X*NPROC_Y

      IF (MYID.GE.NPROC) GOTO 999
      IF (NUMPROCS.LT.NPROC) GOTO 9991

      MYID_X=MOD(MYID,NPROC_X)
      MYID_Y=INT(MYID/NPROC_X)

      ID_IM1=MYID_Y*NPROC_X+MYID_X-1
      ID_IP1=MYID_Y*NPROC_X+MYID_X+1
      ID_JM1=(MYID_Y-1)*NPROC_X+MYID_X
      ID_JP1=(MYID_Y+1)*NPROC_X+MYID_X

      DO N=1,NTIM

      DO J=1,NY
      DO K=1,NZ
       SBND_X1(J,K)=TEMP(1,J,K)
       SBND_X2(J,K)=TEMP(NX,J,K)
      END DO
      END DO

      DO I=1,NX
      DO K=1,NZ
       SBND_Y1(I,K)=TEMP(I,1,K)
       SBND_Y2(I,K)=TEMP(I,NY,K)
      END DO
      END DO

      INR=1
      IF (MYID_X.GT.0) THEN
       CALL MPI_ISEND(SBND_X1,NY*NZ,MPI_REAL,
     &        ID_IM1,308,MPI_COMM_WORLD,ISTATS(INR),IERR)
       INR=INR+1
       CALL MPI_IRECV(RBND_X1,NY*NZ,MPI_REAL,
     &        ID_IM1,309,MPI_COMM_WORLD,ISTATS(INR),IERR)
       INR=INR+1
      END IF
      IF (MYID_X.LT.NPROC_X-1) THEN
       CALL MPI_ISEND(SBND_X2,NY*NZ,MPI_REAL,
```

```
&          id_ip1,308,mpi_comm_world,istats(inr),ierr)
     inr=inr+1
     call mpi_irecv(rbnd_x2,ny*nz,mpi_real,
&          id_ip1,309,mpi_comm_world,istats(inr),ierr)
     inr=inr+1
    end if
    if (myid_y.gt.0) then
     call mpi_isend(sbnd_y1,nx*nz,mpi_real,
&          id_jm1,508,mpi_comm_world,istats(inr),ierr)
     inr=inr+1
     call mpi_irecv(rbnd_y1,nx*nz,mpi_real,
&          id_jm1,509,mpi_comm_world,istats(inr),ierr)
     inr=inr+1
    end if
    if (myid_y.lt.nproc_y-1) then
     call mpi_isend(sbnd_y2,nx*nz,mpi_real,
&          id_jp1,508,mpi_comm_world,istats(inr),ierr)
     inr=inr+1
     call mpi_irecv(rbnd_y2,nx*nz,mpi_real,
&          id_jp1,509,mpi_comm_world,istats(inr),ierr)
     inr=inr+1
    end if
    if (inr.gt.1) call mpi_waitall(inr-1,istats,iresc,ierr)

    do j=1,ny
    do k=1,nz
     temp(1,j,k) =rbnd_x1(j,k)
     temp(nx,j,k)=rbnd_x2(j,k)
    end do
    end do

    do i=1,nx
    do k=1,nz
     temp(i,1,k) =rbnd_y1(i,k)
     temp(i,ny,k)=rbnd_y2(i,k)
    end do
    end do

    do i=2,nx-1
    do j=2,ny-1
    do k=2,nz-1
```

```
      temp1(i,j,k)=temp(i,j,k)+tau*(
     &          (temp(i+1,j,k)-2*temp(i,j,k)+temp(i-1,j,k))/hx**2
     &         +(temp(i,j+1,k)-2*temp(i,j,k)+temp(i,j-1,k))/hy**2
     &         +(temp(i,j,k+1)-2*temp(i,j,k)+temp(i,j,k-1))/hz**2)

      end do
      end do
      end do

      do i=2,nx-1
      do j=2,ny-1
      do k=2,nz-1

      temp(i,j,k)=temp1(i,j,k)

      end do
      end do
      end do

      end do

      goto 999

9991   if (myid.eq.0) print *,'error: nprocs < ',nproc
999    call mpi_barrier(mpi_comm_world,ierr)

      call mpi_finalize(0)

      end
```

Литература

[1] A.A.Boyarchuk, D.V.Bisikalo, O.A.Kuznetsov, V.M.Chechetkin, 2002, Mass transfer in close binary stars, London: Taylor & Frances.

[2] Message Passing Interface Forum, "MPI: A Message Passing Interface", in *Proc. of Supercomputing '93*, p. 878, IEEE Computer Society Press, November 1993; Message Passing Interface Forum, "MPI-2", July 1997, http://www.mpi-forum.org/

[3] П.В.Кайгородов, О.А.Кузнецов, Адаптация схемы Роу–Ошера для компьютеров с массивно–параллельной архитектурой. Препринт ИПМ им.М.В.Келдыша, 2002.

[4] G.Burns, R.Daoud, and K.Vaigl, "*LAM*: An open cluster environment for MPI", in *Proceedings of Supercomputing Symposium '94* (J.W.Ross, ed.), p. 379, Univ. of Toronto, 1994

Оглавление