УДК 519.6

ИСПОЛЬЗОВАНИЕ СИСТЕМЫ НОРМА ДЛЯ РЕШЕНИЯ ВЫЧИСЛИТЕЛЬНЫХ ЗАДАЧ НА МНОГОПРОЦЕССОРНЫХ СИСТЕМАХ С РАСПРЕДЕЛЕННОЙ ПАМЯТЬЮ

А. Н. Андрианов[1], **К. Н. Ефимкин**[1]

Рассматривается опыт решения одной практической задачи из области астрофизики на нескольких параллельных вычислительных системах с распределенной памятью.

Ключевые слова: численные методы, астрофизика, параллельные вычислительные системы, многопроцессорные вычислительные системы, система Норма, message-passing interface, параллельные программы.

1. Введение. Разработке эффективных параллельных программ для решения вычислительных задач на многопроцессорных вычислительных системах в настоящее время посвящено большое количество исследований. Это объясняется тем, что многопроцессорные ЭВМ с распределенной либо общей памятью на данном этапе следует рассматривать как наиболее доступные и перспективные средства для решения задач, требующих большого объема вычислений и памяти.

При создании прикладных программ для подобных систем прикладной специалист, скорее всего, хотел бы: знать как можно меньше об устройстве конкретной системы (но при этом полностью использовать ее возможности); не иметь проблем при переходе с одной системы на другую; избежать детального изучения при программировании моделей параллельности, особенностей компиляторов, организации и использования системных библиотек и т.п.

Реализация такой комфортной модели разработки требует решения трех основных проблем:

— упрощения программирования и отладки;

— обеспечения эффективности параллельных программ;

— обеспечения мобильности параллельных программ.

Различные методы и средства разработки параллельных программ [1] не обеспечивают в полной мере поддержку этой комфортной разработки параллельных прикладных программ. Это связано, видимо, с тем, что у прикладного специалиста возникают трудности при освоении области параллельных вычислений, которые часто осложняются наличием у него опыта “простой” разработки последовательных программ и попытками использовать этот опыт при разработке параллельных программ. Системный программист, обладающий определенными знаниями в области параллельных вычислений, чаще всего не является специалистом в предметной области и не может осуществить математическую постановку задачи и разработать численный метод ее решения. Поэтому проекты, в которых усилия прикладных и системных специалистов объединяются, как правило приводят к положительным результатам.

В данной работе обсуждается опыт решения практической задачи из области астрофизики на различных параллельных вычислительных системах с распределенной памятью. При разработке различных вариантов программы для проведения расчетов использовалась система Норма [2, 3], которая позволила получить выходные программы на языке Фортран с использованием библиотеки MPI (Message-Passing Interface) [4] в модели передачи сообщений. Этот подход является попыткой преодолеть некоторые из отмеченных выше проблем.

Основные вопросы, которые обсуждаются в работе, — некоторые аспекты разработки параллельной программы, эффективность распараллеливания данной задачи для параллельных вычислительных систем с распределенной памятью и мобильность программ, получаемых с помощью системы Норма.

2. Общая характеристика расчетной задачи. Решалась задача из области астрофизики о моделировании трехмерного течения вещества в полуразделенных двойных звездных системах с различными значениями вязкости. Задача принадлежит к классу задач на статических регулярных сетках. Расчетная сетка содержит $61 * 61 * 17$ узлов.

3. Целевые вычислительные системы. Для проведения расчетов использовались следующие вычислительные системы с распределенной памятью.

[1] Институт прикладной математики им. М. В. Келдыша РАН, Миусская пл., 4, 125047, Москва; e-mail: and@a5.kiam.ru, efi@a5.kiam.ru

Вычислительный кластер НИВЦ МГУ (24 процессора). Кластер построен на базе двухпроцессорных узлов с процессорами Pentium III/500MHz с оперативной памятью 256 Mb. Узлы объединены в двумерную решетку с помощью сети SCI. В качестве служебной сети используется Fast Ethernet. Максимальная достигнутая скорость однонаправленных пересылок — 80 Mb/sec. Более подробная информация об архитектуре кластера НИВЦ МГУ изложена в [5].

Вычислительный кластер $AC^3Velocity$ (256 процессоров). Рассматриваемый кластер состоит из 64 SMP-серверов PowerEdge (DELL), каждый из которых включает 4 процессора Pentium III Xeon/500 MHz, 4GB оперативной памяти и 54GB дисковой памяти. Узлы объединены при помощи сетевой технологии cLAN Giganet, расчетная пропускная способность 1,25 Gb/sec. Более подробная информация об архитектуре кластера $AC^3Velocity$ изложена в [6, 7].

Система МВС-1000, СКЦ ИПМ (32 процессора). Основана на использовании микропроцессоров Alpha 21164 (DEC-Compaq)/300 MHz с оперативной памятью 128 Mb. Межпроцессорное взаимодействие осуществляется при помощи коммуникационного процессора TMS320C44 с общей пропускной способностью 80 Mb/сек (4 канала, каждый по 20 Mb/сек).

Система МВС-1000, МСЦ (32 процессора). Основана на использовании микропроцессоров Alpha 21164 (DEC-Compaq)/500 MHz с оперативной памятью 128 Mb. Межпроцессорное взаимодействие осуществляется при помощи TMS320C44 общей пропускной способностью 80 Mb/сек (4 канала, каждый по 20 Mb/сек).

Система МВС-1000, СКЦ ИПМ (8 процессоров). Основана на использовании микропроцессоров Alpha 21164 (DEC-Compaq)/500 MHz с оперативной памятью 128 Mb.

Система МВС-1000М, МСЦ (24 процессора). Кластер построен на базе двухпроцессорных узлов с процессорами Alpha 21264/667MHz и оперативной памятью 1 Gb. Для взаимодействия в процессе вычислений используется SAN Myrinet. Более подробная информация об архитектуре систем семейства МВС изложена в [8, 9, 10].

Кластер ТКС-6, НИЦЭВТ (4 процессора). Кластер построен на базе двухпроцессорных узлов с процессорами Pentium III/733MHz с оперативной памятью 256 Mb. Узлы объединены в кольцо, шина 133 MHz, адаптер Dolphin D320, PCI 64 разряда x 33 MHz.

Кластер ТКС-7, НИЦЭВТ (8 процессоров). Кластер построен на базе двухпроцессорных узлов с процессорами Pentium III/800MHz с оперативной памятью 512 Mb. Узлы объединены в двумерный тор, шина 100 MHz, адаптер Dolphin D311/312, PCI 32 разряда x 33 MHz.

Кластер ТКС-8, НИЦЭВТ (4 процессора). Кластер построен на базе двухпроцессорных узлов с процессорами Alpha 21264/667MHz и оперативной памятью 1 Gb. Узлы объединены в кольцо, адаптер Dolphin D320, PCI 64 разряда x 33 MHz.

4. Разработка Норма-программы. Исходной информацией для разработки Норма-программы являлась последовательная программа на языке Фортран-77, которая рассматривалась как эталонная с функциональной точки зрения.

Эталонная последовательная Фортран-программа вручную была переведена на язык Норма. При этом кроме "чистых" конструкций языка Норма использовалась возможность интерфейса с Фортраном, которая позволила не переписывать часть Фортран-программы (подпрограмм ввода/вывода, использование которых не влияет на параллельное выполнение и обеспечивает привычный для математика интерфейс при подготовке исходных данных и анализе результатов; подпрограммы вычисления скалярных функций). После трансляции такой программы при помощи системы Норма автоматически может быть получена параллельная выходная программа на Фортране-MPI.

Общий объем эталонной программы — 1000 строк. Общий объем Норма-программы — 310 строк. Кроме того, дополнительные Фортран-программы содержат 540 строк.

Специализированный декларативный язык Норма позволяет описывать решение широкого класса задач математической физики и при этом использовать привычные для прикладного специалиста понятия. Описание решения задачи в математических терминах имеет очень высокий уровень абстракции и отражает метод решения, а не его реализацию при конкретных условиях. Такое описание не ориентировано на конкретную архитектуру компьютера, поэтому оно предоставляет большие возможности для выявления естественного параллелизма и организации вычислений.

Программа на языке Норма фактически является представлением математических расчетных формул, полученных прикладным специалистом. Одной из конструкций языка, предназначенных для описания расчетных формул, является оператор

FOR <область> **ASSUME** <соотношение>.

<Соотношение> имеет вид <переменная>=<выражение> и задает правило вычисления значений вели-

чины из левой части по значениям величин из правой части и индексные зависимости между переменными. Величина из левой части должна быть вычислена во всех точках <области>, другими словами, для заданного множества точек индексного пространства. Правило для этого вычисления определено однозначно, однако при этом не требуется немедленного выполнения вычисления в данном месте программы и не задается порядок или способ вычисления. Например, пусть задана область **Matrix**, определяющая точки индексного пространства (i, j), $i, j = 1, \ldots, 5$:

Matrix: ((i=1..5) ; (j=1..5))

и на этой области определена переменная X:

VARIABLE X DEFINED ON Matrix.

Тогда оператор

FOR Matrix ASSUME X = 0

является запросом на обнуление 25-ти элементов величины X. Способ реализации этого запроса (последовательно, параллельно и т.п.) автоматически определяется в процессе трансляции.

Некоторое внешнее сходство с оператором присваивания не должно вводить в заблуждение — Норма является языком с однократным присваиванием (single assignment language) и понятие памяти в нем отсутствует.

Таким образом, программирование на языке Норма не требует написания программы в смысле традиционных универсальных языков программирования, а выходные программы для параллельных ЭВМ с различной архитектурой строятся автоматически при помощи транслятора-синтезатора.

Следует отметить, что ручной перевод последовательной программы на язык Норма является трудоемким процессом, — требуется, по крайней мере, детальный анализ исходной последовательной программы, чтобы понять, а что же необходимо вычислить. При этом приходится "распутывать" информационные зависимости, часто не связанные с существом расчета, а возникшие как результат определенного стиля программирования и богатства возможностей Фортрана (экономия памяти, косвенная адресация, массовое использование COMMON-блоков и т.п.). Этих трудностей можно избежать, если разработку программы на языке Норма вести по расчетным формулам, а не по готовой последовательной программе (отметим, что система Норма позволяет получать по одной и той же Норма-программе как параллельную, так и последовательную реализацию).

Усилия по написанию Норма-программы, по нашему мнению, оправданы — вся работа с этой программой на многопроцессорных вычислительных системах (изменение размеров сетки, изменение числа процессоров, изменение выходного языка) поддерживается системой, поэтому проблемы по синтезу параллельной программы остаются только у компилятора с этого языка, а не у программиста.

5. Некоторые вопросы генерации выходной программы при трансляции с языка Норма. Рассмотрим основные вопросы, которые решаются при трансляции с языка Норма для многопроцессорных вычислительных систем с распределенной памятью. Заметим, что эти вопросы, как правило, возникают и при разработке параллельной программы вручную.

Будем использовать следующую простую модель задачи. Заданы статические регулярные области (сетки), в точках сеток определены расчетные величины (скорость, давление, температура и т.п.), а также вспомогательные величины, которые вводятся по мере необходимости при выборе расчетных формул. Эти формулы для точек, лежащих внутри и на границах сеток, обычно требуют различного количества арифметических операций, однако для простоты будем считать, что в каждой точке сетки проводятся одинаковые (по времени) вычисления.

Задачу распараллеливания для распределенных систем будем рассматривать как задачу статического распределения в модели параллелизма по данным: вычисления выполняются на том процессоре, где находятся вычисляемые данные. Таким образом, необходимо распределить точки сеток (расчетные и вспомогательные величины) между процессорами вычислительной системы и обеспечить доступ к значениям величин, используемым в расчетных формулах.

Будем предполагать также, что каждый процессор должен работать по одной и той же программе (типичная ситуация при использовании библиотеки MPI).

Схемы решения задачи распараллеливания для распределенных вычислительных систем, используемые в компиляторе с языка Норма, основаны на следующих понятиях.

В языке Норма каждая сеточная переменная определяется на некоторой области. Неформально, область состоит из совокупности целочисленных наборов $(k_1, k_2, \ldots, k_n)$, где $n > 0$ — размерность области. Каждый набор задает координаты точки в n-мерном индексном пространстве. С каждой размерностью связывается уникальное имя индекса (имя оси координат индексного пространства). При трансляции на распределенную систему программист сам определяет, точки какого индексного направления будут рас-

пределяться по процессорам системы, и задает число процессоров, на которых будет решаться исходная задача. Направление "разрезания" и число процессоров задается описанием следующего вида:

DISTRIBUTION INDEX i = 1 .. 10; j = 1 .. 5.

Такая запись означает, что все области, в описании которых используются направления i или j, будут распределяться на 10 процессоров по направлению i и на 5 процессоров по направлению j.

Пусть область, на которой определена величина X_k, содержит направление i, по которому должно проводится "разрезание" данной величины, и диапазон по этому направлению имеет вид $i = l^k, \ldots, u^k$. Обозначим $U^i = \max\limits_k u^k$.

Предлагаемая процедура разрезания основывается на размещении постоянного числа компонентов всех массивов в каждом процессоре (по фиксированному направлению). Пусть по направлению i используется m процессоров. Тогда число компонентов массива по направлению i задается значением: ipr $= [U^i/m]$.

Аналогично вводится параметр jpr для направления j. Такое размещение массивов позволяет существенно упростить задачу определения параметров для процедур обмена. Недостатком такого размещения является то, что в случаях значительного расхождения верхних границ диапазонов будут неэффективно использоваться процессоры вычислительной системы.

При определении индексных направлений i, j в конструкции **DISTRIBUTION INDEX** для эффективного распараллеливания обычно используются следующие два критерия:

1) максимальное число точек по направлению (что приводит к распределению между процессорами наиболее "тяжелой работы"),

2) минимальное число вхождений индекса, заданного со смещением, в индексные выражения (это обычно минимизирует число обменов между процессорами, так как если точка с координатой i находится в процессоре с номером P_i, то точка с координатой $i-1$, в зависимости от значения i, может находиться либо в процессоре P_i, либо в процессоре P_{i-1}).

При построении фрагмента распределенной программы, реализующей вычисление на области O некоторой переменной X, определенной на области D, необходимо определить:

1) **номера процессоров,** которые должны выполнять данный фрагмент,

2) **границы изменения параметров циклов,** обеспечивающих обход области O,

3) **описание массива** X,

4) **вид и расположение операторов передачи сообщений (обменов),** которые обеспечивают доступ к значениям величин, используемым в расчетных формулах для вычисления X.

Пусть надо вычислить следующие соотношения:

1. U(i,j)=i+j для i=1..N, j=1..N;

2. W(i,j)=U(i,j)-1 для i=1..N, j=1..N;

3. V(i,j)= U(i-1,j)+1 для i=25..N-25, j=1..N.

Пусть N=100 и используется линейка из 10 процессоров по направлению i. На языке Норма эти вычисления описываются следующим образом:

```
! Описание областей Oi, Oi1, Oj
Oi:(i=1..n).   Oi1:(i=25..n-25).   Oj:(j=1..n).
! Область Oij есть декартово произведение (i=1..n) ( (j=1..n)
Oij:(Oi;Oj).
! Область Oij1 есть декартово произведение (i=25..n-25) ( (j=1..n)
Oij1:(Oi1;Oj).
! Определение параметра областей
DOMAIN PARAMETERS n=100.
! Описание переменных на областях
VARIABLE U,V,W DEFINED ON Oij.
! Число процессоров по направлению i равно 10.
DISTRIBUTION INDEX i=1..10, j=1.
! Расчетные формулы (индексы без смещений можно опускать)
FOR Oij   ASSUME  U=i+j; W=U-1.
FOR Oij1 ASSUME  V=U[i-1]+1.
```

Фрагмент выходной программы представлен ниже:

```
C  Вычисление параметров циклов в зависимости от номера процессора
      IN01 = 1
```

```
      IK01 = 10
      IN02 = 1
      IK02 = 10
      IF(IPR.EQ.3)  THEN   IN02=5
      IF(IPR.EQ.8)  THEN   IK02=5
      DO     2 j=1,100
      DO     2 i=IN01,IK01
C  Выражение i+j в процессоре с номером IPR
      U(i,j)=((IPR-1)*10+i)+j
2 CONTINUE
C  Требуются вычисленные значения из процессоров с номерами  3..7
      IF(IPR.LT.3.OR.IPR.GT.7.OR.JPR.NE.1) GOTO 4
      DO     3 j=1,100
      RM1(j)=U(10,j)
3    CONTINUE
      CALL MPI_SEND(RM1,100,MPI_REAL,ITASKJ(JPR+(IPR+1-1)),2,MPI_COMM_WORLD,IER)
4    CONTINUE
      DO     5 j=1,100
      DO     5 i=IN01,IK01
      W(i,j)=U(i,j)-1
5    CONTINUE
C  Используются вычисленные значения в процессорах с номерами  4..8
      IF(IPR.LT.4.OR.IPR.GT.8.OR.JPR.NE.1) GOTO 7
      CALL MPI_RECV(RM2,100,MPI_REAL,ITASKJ(JPR+(IPR-1-1)),2,MPI_COMM_WORLD,STTS1,IER)
      DO     6 j=1,100
      U(0,j)=RM2(j)
6    CONTINUE
7    CONTINUE
      IF(IPR.LT.3.OR.IPR.GT.8) GOTO 8
      DO     9 j=1,100
      DO     9 i=IN02,IK02
      V(i,j)= U(i-1,j)+1
9    CONTINUE
8    CONTINUE
```

Определение номеров процессоров. Пусть некоторое вычисление необходимо выполнить при значениях переменной i от in до ik. В языке Норма значения in и ik известны на этапе трансляции, поэтому компилятор по значению in (ik) и числу точек, распределенных на процессор, определяет номер процессора Pn (Pk), в котором находится точка in (ik). Таким образом, диапазон процессоров, участвующих в вычислении, задается сегментом [Pn, Pk]. Процессоры, номера которых находятся за пределами этого сегмента, не участвуют в вычислении данного соотношения. В нашем примере операторы, реализующие соотношения 1–2, выполняются во всех 10 процессорах. Соотношение 3 вычисляется при значениях индекса i в диапазоне от 25 до 75. Точка i=25 принадлежит процессору с номером 3, а точка i=75 принадлежит процессору с номером 8. Поэтому оператор, реализующий соотношение 3, выполняется только в процессорах, номера которых лежат в диапазоне [3, 8].

Определение границ изменения параметров циклов. Аналогично тому как определяется диапазон процессоров, реализующих заданное соотношение, определяются начальные и конечные значения параметра цикла i, при которых надо выполнить оператор, реализующий соотношение. В нашем случае операторы 1 и 2 выполняются (по индексу i) от IN01 до IK01, а оператор 3 — от IN02 до IK02. При этом, во всех 10 процессорах, значения IN01=1, IK01 = 10. Более сложный вид имеют значения IN02 и IK02. В процессоре с номером 3 значение IN02 = 5, во всех остальных IN02 = 1. В процессоре с номером 8 значение IK02 = 5, а в остальных IK02 = 10.

Описания массивов. Построение описаний для сеточных переменных проводится достаточно простым образом. В последовательной программе переменным U, V, W соответствуют описания

DIMENSION U(100,100), V(100,100), W(100,100)

При построении распределенной программы учитывается, что эти переменные должны быть распределены в соответствии с конструкцией

DISTRIBUTION INDEX i=1..10, j=1

то есть "разрезаны" на 10 процессоров по частям, состоящим из 10×100 элементов. Кроме этого, поскольку в формуле для вычисления V требуются значения U[i-1], необходимо выделить память ("теневую" грань) для приема значений U[i-1] из соседнего левого процессора. Это делается за счет расширения влево диапазона при описании массива U. Таким образом, описания для распределенной программы переменных U, V, W выглядят так:

DIMENSION U(0:10,100), V(10,100), W(10,100)

Генерация операторов обмена. Интерфейс передачи сообщений MPI [4] предоставляет разнообразные средства реализации взаимодействия в модели передачи сообщений. В рассматриваемой программе взаимодействие между процессорами реализовано при помощи подпрограмм MPI_SEND (отправить сообщение) и MPI_RECV (принять сообщение), которые имеют следующие параметры:

```
CALL MPI_SEND(<адрес посылки>,<длина посылки>,<тип посылаемых элементов>,
     <имя процесса-получателя>,<тэг>,<коммуникатор>,<код возврата>)
CALL MPI_RECV(<адрес приема>,<длина приема>,<тип принимаемых элементов>,
     <имя процесса-отправителя>,<тэг>,<коммуникатор>,<статус>,<код возврата>)
```

Для обеспечения взаимодействия в *каждой* распределенной программе компилятор автоматически определяет и использует следующие структуры данных:

ITASKJ(n,m) — матрица имен параллельных процессов, участвующих в вычислении (ее размеры n и m фактически совпадают с параметрами распределения, заданными в конструкции **DISTRIBUTION INDEX**, поэтому в данном примере описание матрицы имен имеет вид INTEGER ITASKJ(10,1));

IPR, JPR — значения индексов матрицы ITASKJ, определяющие имя каждого из процессов (другими словами, каждый процесс "знает", что его имя — ITASKJ(IPR,JPR)).

Таким образом, вызов

CALL MPI_SEND(RM1,100,MPI_REAL,ITASKJ(JPR+(IPR+1-1)),2,MPI_COMM_WORLD, IER)

приводит к пересылке 100 элементов массива RM1 типа REAL соседу справа. Посылку осуществляет каждый из процессов с номерами JPR=1, IPR=3,. . . , 7. Вызов

CALL MPI_RECV(RM2,100,MPI_REAL,ITASKJ(JPR+(IPR-1-1)),2,MPI_COMM_WORLD, STTS1,IER)

приводит к приему 100 элементов типа REAL в массив RM2 от соседа слева. Прием осуществляет каждый из процессов с номерами JPR=1, IPR=4,. . . , 8.

Определение параметров операторов MPI_SEND и MPI_RECV проводится автоматически на этапе компиляции. Место в выходной программе, где должны быть построены эти операторы, определяется также компилятором из следующих соображений. Оператор MPI_SEND ставится непосредственно за оператором, в котором вычисляется требуемое для передачи значение. Оператор MPI_RECV ставится непосредственно перед оператором, в котором используется отправленное значение. Это позволяет надеяться на перекрытие времени счета и обмена данными. В нашем примере сразу после вычисления значений переменной U стоит оператор записи значений величины U в рабочий массив RM1 и оператор отправки этого массива MPI_SEND. При этом, как отмечено выше, отправляют значения не все процессоры. Соответствующий оператор приема MPI_RECV расположен перед оператором, где используется значение U[i-1,j]. Соответствие обеспечивается наличием уникального тэга 2, причем генерацию тэгов также поддерживает компилятор.

Отметим еще раз, что задачи 1) – 4) решаются автоматически в процессе компиляции Норма-программы, что сокращает объем технической, а часто и интеллектуальной работы по программированию.

Компилятор с языка Норма осуществляет также некоторые оптимизирующие преобразования. В частности, реализован практический прием ускорения вычислений за счет использования кэш-памяти при помощи оптимизации, связанной с изменением порядка выполнения вложенных циклов.

Рассмотрим оптимизацию, связанную с изменением порядка выполнения вложенных циклов, например, заменяющую гнездо циклов вида:

```
do  i  = 1,n
do  j  = 1,m
X(i,j) = ...
Enddo
Enddo
```

на гнездо циклов вида:

```
do  j  =1,m
do  i  =1,n
X(i,j)= ...
enddo
enddo
```

При этом порядок индексных выражений у переменных, входящих в оператор присваивания в правой части, один и тот же: i, j. Конечно, это преобразование может быть использовано только в том случае, когда соблюдаются условия его применимости.

Гнездо первого вида будем называть гнездом с *прямым порядком исполнения*, а гнездо второго вида — гнездом с *обратным порядком исполнения.*

Как известно, в Фортране принято размещение массивов по столбцам. Если массив в Фортране описан как A(N1,N2), то в оперативной памяти последовательно размещаются элементы массива с координатами A(1,1), A(2,1),..., A(N1,1), A(2,1), A(2,2),..., A(N1,N2). Рассмотрим гнездо с прямым порядком исполнения:

```
do  i  = 1,n
do  j  = 1,m
A(i,j) = B(i,j)+1
enddo
enddo
```

Исполнение цикла начинается со значений параметров i=1, j=1. Для выполнения оператора необходимо выбрать из памяти значение B(1,1). Если этот элемент отсутствует в кэш-памяти, то производится выбор элемента из основной памяти в кэш-память и результат передается в процессор. Кроме того, в кэш-память считывается не один элемент данных, а целый блок (кэш-строка) последовательно размещенных в оперативной памяти данных. Размер кэш-строки определяется конкретной ЭВМ.

Для нашего примера в кэш-память попадают элементы B(1,1), B(2,1), B(3,1),. . . и т.д.

Следующее значение параметров цикла i=1, j=2. В этом случае для оператора требуется значение величины B(1,2). При больших значениях n и m с большой вероятностью это значение не попало в кэш-память на предыдущем витке цикла. Таким образом, опять повторится процедура выборки из оперативной памяти. В случае, когда используется гнездо с обратным порядком исполнения, после витка цикла с параметрами (i=1, j=1) для следующего витка с параметрами (i=2, j=1) значение B(2,1) уже находится в кэш-памяти. Более того, и далее, когда требуется элемент B(3,1), он также уже находится в кэш-памяти.

Проведен эксперимент по оценке времени выполнения простого оператора:

```
do i =1,n
do j =1,m
   x(i,j)=y(i,j)+2.0-z(i,j)/v(i,j)
enddo
enddo
```

на одном процессоре компьютера. При этом задавались различные размерности массивов (значения n, m). Задача состояла в том, чтобы определить, насколько можно ускорить вычисления (и можно ли вообще это сделать), изменяя порядок перебора индексных значений.

Ниже в таблице представлены времена выполнения программ на трех компьютерах для различных значений параметров циклов. Каждому варианту расчета соответствует строка в таблице. При этом, в столбцах, содержащих *время счета*, верхнее число указывает время исполнения цикла при прямом порядке выполнения циклов, а нижнее число указывает время, полученное при обратном порядке выполнения цикла. В столбце **ускорение** показан получаемый выигрыш во времени.

Компилятор с языка Норма позволяет строить выходные программы на Фортране как с прямым, так и с обратным порядком выполнения циклов (что было использовано при решении рассматриваемой астрономической задачи).

6. Характеристики времени счета. Для сравнения времени выполнения использовались 3 программы, реализующие один и тот же алгоритм:

1. исходная программа на языке Фортран, предоставленная математиками (программа Et1);

2. фортран-программа, полученная в результате компиляции Норма-программы (с обратным порядком исполнения циклов, программа Pr2);

	НИВЦ МГУ		СКЦ 8		СКЦ 32	
Значения n,m	Время счета	Ускорение	Время счета	Ускорение	Время счета	Ускорение
16×6000	32,3	5,21	45,5	7,46	122,2	10,53
	6,2		6,1		11,6	
6000×16	6,9	1,10	6,0	1,03	12,3	1,11
	6,3		5,8		11,1	
320×300	18,8	3,10	46,4	8,00	118,1	10,64
	6,1		5,8		11,1	

Процессоры		1	2	4	7
ЭВМ	Программа				
	Pr2	2343	1346	669	401
НИВЦ МГУ	Pr3	4994	2527	1375	893
	Et1	2273			
	Pr2	2763	1505	808	475
$AC^3Velocity$	Pr3	4937	3114	1740	808
	Et1	2736			
	Pr2	1545	866	535	340
МВС1000(СКЦ 8)	Pr3	5516	3968	1659	784
	Et1	1822			
	Pr2	2693	1519	1019	645
МВС1000(СКЦ 32)	Pr3	12113	6758	3682	1664
	Et1	3375			
	Pr2	1526	886	536	310
МВС1000(МСЦ -32)	Pr3	5788	3077	1664	755
	Et1	1868			
	Pr2	703	368	199	110
МВС1000М	Pr3	—	—	1215	450
	Et1	—			
	Pr2	1690	973		
ТКС-6	Pr3	4083	2116		
	Et1	1642			
	Pr2	1592	923	470	307
ТКС-7	Pr3	4400	2251	—	—
	Et1	1596			
	Pr2	703	365		
ТКС-8	Pr3	7310	3755		
	Et1	1070			

Рис 1. Сводная таблица времен счета

3. фортран-программа, полученная в результате компиляции Норма-программы (с прямым порядком исполнения циклов, программа Pr3).

Были построены 4 версии программ Pr2 и Pr3. Версии отличаются числом используемых процессоров P = 1, 2, 4 и 7. При этом конфигурация процессоров представляла в каждом случае линейку процессоров вида $P \times 1$.

Сводная таблица времен счета (в секундах) приведена на рис 1.

Ниже на рис. 2 приведена гистограмма времен счета (в секундах) программ Et1, Pr2, Pr3 на одном процессоре разных компьютеров, на рис. 3 — гистограмма времени счета для различного числа процессоров (для программы Pr2 на разных компьютерах).

На рис. 4 приведены графики эффективности распараллеливания и времени счета программ Pr2, Pr3 на МВС-1000 МСЦ.

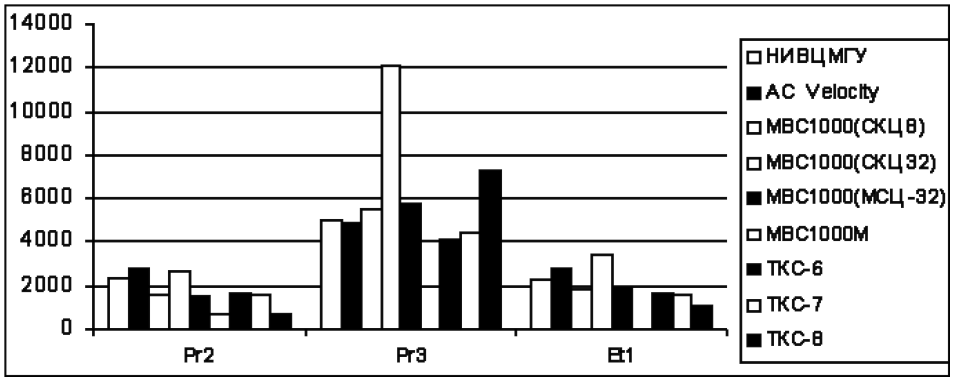

Рис. 2. Гистограмма времен счета программ Et1, Pr2, Pr3 на 1 процессоре

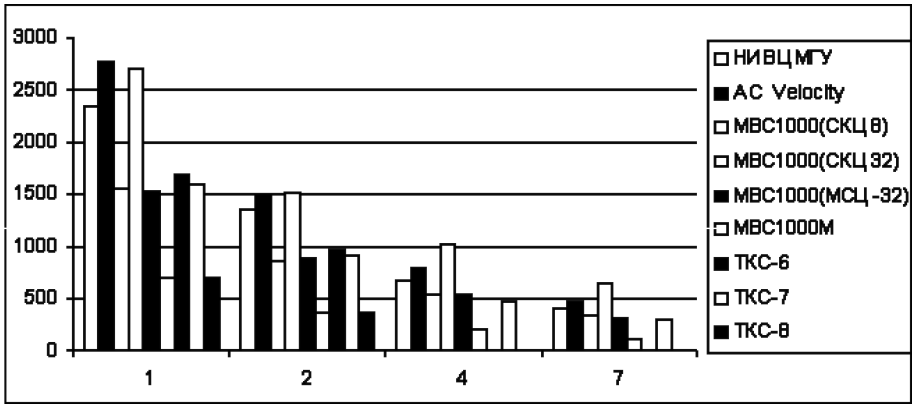

Рис. 3. Гистограмма времен счета программы Pr2 на различных компьютерах

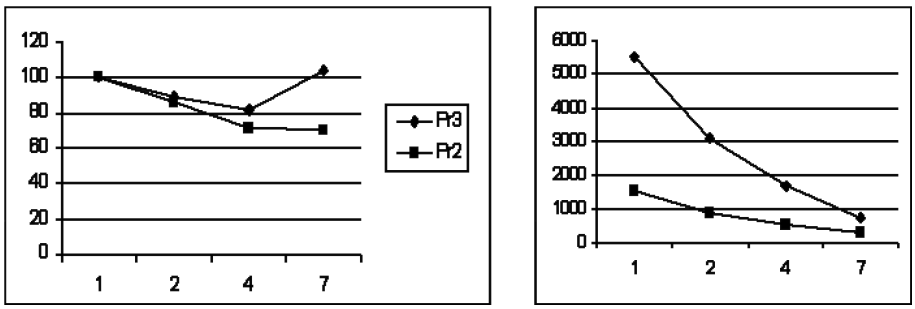

Рис. 4. Эффективность распараллеливания и время счета программ Pr2, Pr3 на МВС-1000 МСЦ

На графиках видно, что программа Pr2 выполняется быстрее, чем программа Pr3, но эффективность распараллеливания программы Pr3 выше эффективности распараллеливания программы Pr2. Это объясняется тем, что в программе Pr3 сами вычисления проводятся медленнее, чем в программе Pr2 (из-за различного порядка циклов), а объемы передаваемой между процессорами информации в этих программах одинаковы.

В заключение отметим, что приведенные данные характеризуют поведение программ решения одной конкретной задачи на различных вычислительных системах. Для задач с другими типами информационных зависимостей картина скорее всего окажется другой. Кроме того, не рассмотрены некоторые параметры, которые также могут оказать существенное влияние на время выполнения программ (например, характеристики используемых компиляторов).

Работа выполнена при финансовой поддержке РФФИ (проекты № 98–01–00987 и 99–01–00842).

СПИСОК ЛИТЕРАТУРЫ

1. Информационно-аналитический центр PARALLEL.RU. http://parallel.ru

2. *Задыхайло И.Б., Ефимкин К.Н.* Содержательные обозначения и языки нового поколения // Информационные технологии и вычислительные системы. 1996. № 2. 46–58.
3. *Андрианов А.Н., Бугеря А.Б., Ефимкин К.Н., Задыхайло И.Б.* Норма. Описание языка. Рабочий стандарт. Препринт ИПМ им. М. В. Келдыша РАН. 1995. № 120.
4. *Gropp W., Lusk E., Skjellum A.* Using MPI. Portable Parallel Programming with the Message-Passing Interface. 2nd ed. Cambridge, 1999.
5. Вычислительный кластер НИВЦ МГУ. http://parallel.ru:81/
6. Корнельский университет устанавливает 256-процессорный кластер на базе SMP-серверов DELL с Windows NT. http://parallel.ru/:81/cluster/configuration.html
7. Advanced Cluster Computing Consortium Established at Cornell. http://www.tc.cornell.edu/er/media/1999/ac3.html
8. *Забродин А.В., Левин В.К.* Опыт разработки параллельных вычислительных технологий. Создание и развитие семейства МВС // Труды Всеросс. научн. конф. “Высокопроизводительные вычисления и их приложения”. Черноголовка, 2000. 3–8.
9. *Савин Г.И., Шабанов Б.М., Забродин А.В., Елизаров Г.С., Каратанов В.В., Корнеев В.В., Левин В.К.* Структура многопроцессорной вычислительной системы МВС-1000М // Труды Всеросс. научн. конф. “Высокопроизводительные вычисления и их приложения”. Черноголовка, 2000. 3.
10. *Забродин А.В., Елизаров Г.С., Каратанов В.В., Корнеев В.В., Левин В.К., Конотопцев В.Н.* Производительность 24-процессорного фрагмента многопроцессорной вычислительной системы МВС-1000М // Труды Всеросс. научн. конф. “Высокопроизводительные вычисления и их приложения”. Черноголовка, 2000. 9–12.

Поступила в редакцию
15.12.2000