МИНИСТЕРСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ПО АТОМНОЙ ЭНЕРГИИ

Российский федеральный ядерный центр — ВНИИЭФ

ВОПРОСЫ АТОМНОЙ НАУКИ И ТЕХНИКИ

СЕРИЯ:

Математическое моделирование физических процессов

НАУЧНО-ТЕХНИЧЕСКИЙ СБОРНИК

ВЫПУСК 4

Издается с 1978 г.

Москва — 2002

УДК 519.685.3

РАСШИРЕНИЕ ЯЗЫКА OpenMP FORTRAN ДЛЯ РАСПРЕДЕЛЕННЫХ СИСТЕМ

В.А. Бахтин, Н.А. Коновалов, В.А. Крюков
(ИПМ им. М.В. Келдыша РАН, МГУ им. М.В. Ломоносова)

Отсутствие средств управления локализацией данных ограничивает применение OpenMP многопроцессорными системами с общей памятью или с DSM (Distributed Shared Memory) — распределенной общей памятью. В настоящей работе рассматриваются предложения по расширению OpenMP для языка Fortran директивами распределения данных и доступа к удаленным данным, которые позволяют использовать OpenMP на распределенных системах.

Введение

Стандарт OpenMP [1], появившийся в октябре 1997 г., является обобщением низкоуровневых моделей параллелизма на общей памяти для языков Fortran 77, Fortran 90, C и C++. OpenMP встроен в языки программирования в виде директив, описывающих параллелизм программы.

При использовании OpenMP программист разбивает программу с помощью соответствующих директив на последовательные и параллельные области. В начальный момент времени порождается *основная нить*, которая начинает выполнение программы со стартовой точки. Основная нить и только она исполняет все последовательные области программы. При входе в параллельную область, которая определяется посредством директив PARALLEL и END PARALLEL, порождаются *дополнительные нити*, количество которых задается при помощи функций системы поддержки выполнения OpenMP-программ или при помощи переменных среды. Все нити исполняют один и тот же код, соответствующий параллельной области. При выходе из параллельной области основная нить дожидается завершения остальных нитей, и дальнейшее выполнение программы продолжает только она.

Если в параллельной секции встретился оператор цикла, то согласно общему правилу он будет выполнен всеми нитями, т. е. каждая нить выполнит все итерации данного цикла. Для распределения итераций цикла между различными нитями используется директива DO. Параллелизм на уровне независимых фрагментов оформляется в OpenMP с помощью директивы SECTION. Каждый из таких фрагментов будет выполнен какой-либо одной нитью. Если в параллельной секции какой-то участок кода должен быть выполнен лишь один раз (такая ситуация иногда возникает, например при работе с общими переменными), то его нужно поставить между директивами SINGLE и END SINGLE. Такой участок кода будет выполнен нитью, первой дошедшей до данной точки программы.

Несомненным достоинством OpenMP является высокоуровневая модель параллелизма (высокий уровень интерфейса между пользователем и прикладной программой), которая обеспечивает мобильность (не только между разными мультипроцессорами, но и между разными операционными системами) и достаточную эффективность параллельных программ для многопроцессорных систем с общей памятью (мультипроцессоров).

Одним из недостатков OpenMP является отсутствие средств управления локализацией данных. Современные системы массового параллелизма представляют собой сеть вычислительных узлов. При этом мультипроцессоры могут быть только узлами этой сети, что значительно ограничивает применение OpenMP, так как программа в этом стандарте может выполняться только на одном узле сети.

В последнее время для вычислительных систем (ВС), сочетающих в себе одновременно характеристики архитектур как с общей, так и с распределенной памятью, используется так называемая *гибридная* модель: OpenMP + MPI [2]. При этом программа представляет собой систему взаимодействующих MPI-процессов, а каждый процесс программируется на OpenMP. Используя такую модель, можно добиться хорошей эффективности в том случае, когда в программе есть два уровня параллелизма: 1) параллелизм между подзадачами и 2) параллелизм внутри подзадачи (многоблочные методы). Использование OpenMP на мультипроцессорном узле для некоторых задач (например вычислений на неструктурных сетках) может дать заметный выигрыш в эффективности. Основной недостаток — сложность написания таких программ. Программист должен знать и уметь использовать две разные модели паралле-

лизма и разные инструментальные средства. Более того, прикладному программисту приходится описывать алгоритм не в терминах массива целиком (как это делается на последовательных компьютерах), а манипулировать локальными частями массива, размер которых зависит от числа используемых процессоров (узлов).

Альтернативой такому подходу является использование высокоуровневых спецификаций параллелизма, которые освобождают программиста от рутинной и трудоемкой работы по распределению глобальных массивов на узлы ВС, по управлению передачей сообщений и синхронизации доступа к удаленным данным. Кроме того, эти спецификации позволяют обеспечить динамическую настройку программ при запуске (без перекомпиляции) на параметры приложения (количество и размер массивов данных) и конфигурацию параллельного компьютера.

Далее рассматриваются предложения по расширению OpenMP для языка Fortran директивами распределения данных и доступа к удаленным данным. Рассматривается вычислительная сеть, каждый узел которой является мультипроцессором или отдельным процессором. На каждом узле выполняется параллельная программа OpenMP Fortran. Распределение данных между узлами и доступ к удаленным данным выполняется по модели, аналогичной модели DVM [3].

Используя высокоуровневые директивы распределения данных, программист может отобразить массивы между узлами ВС. После этого вся работа над локальными частями массивов осуществляется средствами OpenMP, при этом распределение вычислений между нитями осуществляется в соответствии с распределением данных: в директивах DO, SECTION и SINGLE должна содержаться информация об узле, на котором будет выполняться соответствующая нить (директива ON). Если в вычислениях участвуют данные, размещенные на других узлах, то все такие данные должны быть описаны с помощью соответствующих спецификаций.

Директивы распределения данных и спецификации удаленных данных взяты из языка Fortran DVM [4].

Распределение данных

Распределение данных между узлами осуществляется директивами DISTRIBUTE и ALIGN. Например, отображение одномерного массива выполняется следующей директивой:

```
CDVM$ DISTRIBUTE AR (<формат>)
```

Здесь <формат> =

BLOCK — отображение равными блоками;

GEN_BLOCK(BS) — отображение блоками разных размеров;

WGT_BLOCK(WB,NWB) — отображение взвешенными (неравными) блоками;

* — отображение целым измерением.

Пусть программа, имеющая следующий фрагмент:

```
        REAL A(12), B(6), WB(6)
        INTEGER BS(4)
CDVM$   DISTRIBUTE A(BLOCK)
CDVM$   DISTRIBUTE B(BLOCK)
```

запущена на NP (в данном случае 4) узлах. При отображении *равными блоками* индексное пространство измерения массивов A и B разделяется на NP блоков (отрезков) и i-й блок отображается на i-й узел (табл. 1).

Распределение *блоками разных размеров* позволяет влиять на балансировку загрузки узлов для алгоритмов, которые выполняют разное количество вычислений на различных участках области данных.

Пусть $BS(1:P)$ — массив целых чисел. Директива

```
CDVM$ DISTRIBUTE A(GEN_BLOCK(BS))
```

разделяет массив A на P блоков. Блок i размером $BS(i)$ распределяется на i-й узел. В табл. 2. показано распределение массива, заданное директивой `DATA BS /2, 4, 4, 2/`.

При отображении *взвешенными блоками* задается вектор весов WB размером NWB для каждой точки (или группы соседних точек) индексного пространства измерения:

```
CDVM$ DISTRIBUTE B(WGT_BLOCK(WB,6))
      DATA WB /1., 0.5, 0.5, 0.5, 0.5, 1./
```

Таким образом заданное распределение массива B приведено в табл. 3.

Таблица 1

Распределение одномерных массивов равными блоками

Массив	Узел 1	Узел 2	Узел 3	Узел 4
A	1,2,3	4,5,6	7,8,9	10,11,12
B	1,2	3,4	5	6

Таблица 2

Распределение одномерного массива блоками разных размеров

Массив	Узел 1	Узел 2	Узел 3	Узел 4
A	1,2	3,4,5,6	7,8,9,10	11,12

Таблица 3

Распределение одномерного массива взвешенными блоками

Массив	Узел 1	Узел 2	Узел 3	Узел 4
B	1	2,3	4,5	6

Измерение массива разделяется на NP блоков так, чтобы минимизировать отклонение веса каждого блока (суммы весов точек блока) от среднего значения. Такое отображение позволяет сбалансировать загрузку узлов.

Отображение *целым измерением* означает, что измерение не будет разделяться между узлами и целиком распределено на каждый узел (локальное, нераспределенное измерение).

Отображение многомерного массива осуществляется посредством задания отображения каждого его измерения. При этом количество распределенных измерений массива не может превышать количества измерений решетки узлов.

Очень часто требуется отобразить несколько массивов согласованно, например для того, чтобы уменьшить количество данных, вычисляемых на одних узлах и используемых на других, доступ к которым требует существенных накладных расходов. Для организации *согласованного* отображения нескольких массивов используется механизм выравнивания одного массива на другой с помощью директивы ALIGN.

Рассмотрим следующий фрагмент программы:

```
          REAL A(N), B(N)
CDVM$     DISTRIBUTE A(BLOCK)
CDVM$     DISTRIBUTE B(BLOCK)
          ...
!$OMP     PARALLEL DO
CDVM$     ON B(I)
          DO I=N1,N2
            B(I)=A(F(I))
          ENDDO
```

Пусть $OWN(B(I))$ обозначает узел, на который распределен элемент $B(I)$. Виток цикла с индексом I будет выполняться на процессоре $OWN(B(I))$. Если элемент $A(F(I))$ будет распределен на узел $OWN(B(I))$, то для каждого витка цикла все данные будут распределены на одном узле. Чтобы обеспечить такую локализацию данных, необходимо вместо директивы `CDVM$ DISTRIBUTE B(BLOCK)` применить директиву выравнивания индексных пространств массивов `CDVM$ ALIGN B(I) WITH A(F(I))`, которая устанавливает соответствие между элементами $B(I)$ и $A(F(I))$. Это означает, что элемент $B(I)$ будет распределен на узел $OWN(A(F(I)))$.

Рассмотрим следующий фрагмент программы:

```
      REAL A(100), B(100), C(100)
      ...
C     распределение массивов A, B и C между
      узлами
      ...
      DO I=2,98
          A(I) = C(I-1) + B(I+1)
      ENDDO
```

Для того, чтобы не было обмена между узлами, необходимо разместить элементы $A(I)$, $C(I-1)$ и $B(I+1)$ на одном узле. Выравнивание массивов C и B на массив A невозможно, так как функции выравнивания $I-1$ и $I+1$ выводят за пределы индексов измерения массива A. В этом случае используется механизм шаблонов. Описывается шаблон $TABC$, на один элемент которого выравниваются элементы массивов A, B и C, которые должны быть размещены на одном узле. Шаблон распределяется с помощью директив DISTRIBUTE и REDISTRIBUTE.

```
          REAL A(100), B(100), C(100)
CDVM$     TEMPLATE TABC(102)
CDVM$     ALIGN B(I) WITH TABC(I)
CDVM$     ALIGN A(I) WITH TABC(I+1)
CDVM$     ALIGN C(I) WITH TABC(I+2)
CDVM$     DISTRIBUTE TABC(BLOCK)
          ...
!$OMP     PARALLEL DO
CDVM$     ON A(I)
          DO I= 2,98
            A(I) = C(I-1) + B(I+1)
          ENDDO
!$OMP     END PARALLEL DO
```

Элементы шаблона не требуют памяти, они указывают узел, на который должны быть распределены элементы выровненных массивов.

При выполнении программы можно динамически изменять распределение данных с помощью директив REDISTRIBUTE и REALIGN. Однако надо понимать, что такое перераспределение может потребовать значительных затрат времени. Если после выполнения директив REDISTRIBUTE и REALIGN перераспределяемые массивы получают новые значения, то перед этими директивами следует указать дополнительную (оптимизирующую) директиву NEW_VALUE. Эта директива отменяет пересылку значений распределенного массива.

Распределение вычислений

Для написания эффективных программ необходимо не только распределить данные между узлами, но и распределить вычисления таким образом, чтобы они осуществлялись на тех узлах, на которых находятся соответствующие данные, используемые в этих вычислениях. Только при выполнении этого условия можно достичь оптимальной производительности.

Распределение вычислений в соответствии с распределением данных осуществляется при помощи директивы ON. Эта директива используется совместно с OpenMP-конструкциями распределения работы между нитями, а именно:

- OMP [PARALLEL] DO;
- SECTION;

– SINGLE.

Директива ON может быть встроена в язык Fortran в виде спецкомментария:

```
CDVM$ ON A(L1,...,Ln),
```

где $Lk = ak \times ij + bk$ — линейная функция от индекса j-го цикла (из m-мерного тесно-гнездового цикла), $0 < k < n+1$, $0 < j < m+1$; A — идентификатор массива данных.

Рассмотрим следующий пример:

```
        REAL A(N,N), B(N,N)
CDVM$   DISTRIBUTE (BLOCK,BLOCK) ::  A,B
        ...
!$OMP   PARALLEL DO
CDVM$   ON A(I,J)
        DO I=1,N
          DO J=1,N
            A(I,J)=B(I,J)
          ENDDO
        ENDDO
!$OMP   END PARALLEL DO
```

Виток цикла будет выполняться на том узле, на который будет отображен элемент массива $A(I,J)$.

Спецификация удаленных данных

Удаленными данными будем называть данные, размещенные на одном узле и используемые на другом.

Рассмотрим следующий оператор:

$$A(inda) = \ldots B(indb) \ldots,$$

где A, B — распределенные массивы; $inda$, $indb$ — индексные выражения.

Ссылка $B(indb)$ не является удаленной ссылкой, если соответствующие ей элементы массива B размещены на узле, на котором находится элемент $A(inda)$. Единственной гарантией этого является выравнивание $A(inda)$ и $B(indb)$ в одну точку шаблона выравнивания. Если выравнивание невозможно или не выполнено, то ссылку $B(indb)$ необходимо специфицировать как удаленную ссылку.

По степени эффективности обработки удаленные ссылки разделены на два типа: SHADOW и REMOTE.

Если массивы A и B выровнены и $inda = indb \pm \pm d$ (d — положительная целочисленная константа), то удаленная ссылка $B(indb)$ принадлежит типу SHADOW. Удаленная ссылка на многомерный массив принадлежит типу SHADOW, если распределяемые измерения удовлетворяют определению типа SHADOW.

Удаленные ссылки, не принадлежащие типу SHADOW, составляют множество ссылок типа REMOTE.

Удаленные данные типа SHADOW. Рассмотрим следующий пример:

```
C$OMP   PARALLEL DO
CDVM$   ON B(I), SHADOW_RENEW (A(D1:D2))
        DO I= D1+1,N-D2
            B(I) = A(I-D1) + A(I+D2)
        ENDDO
```

Требуемые каждому узлу удаленные данные размещены на соседних узлах. Спецификация SHADOW_RENEW указывает, что в цикле используются новые значения удаленных данных из массива A, $D1$ указывает размер требуемых данных с левого соседа, а $D2$ — с правого. Для доступа к удаленным данным используются *теневые грани*, представляющие собой буфер, который является непрерывным продолжением локальной секции массива в памяти процессора. Перед выполнением цикла требуемые удаленные данные пересылаются с соседних узлов в теневые грани массива A на каждом узле. Доступ к этим данным при выполнении цикла производится обычным образом (через ссылки на массив A).

Существует асинхронная форма спецификации, которая позволяет совместить пересылку данных в теневые грани с вычислениями.

Асинхронное обновление теневых граней для именованной группы распределенных массивов описывается следующими директивами:

– определение группы:
  `CDVM$ SHADOW_GROUP (A,B);`
– запуск обновления теневых граней:
  `CDVM$ SHADOW_START AB;`
– ожидание значений теневых граней:
  `CDVM$ SHADOW_WAIT AB.`

Директива SHADOW_START должна выполняться после директивы SHADOW_GROUP. После выполнения директивы SHADOW_GROUP директивы SHADOW_START и SHADOW_WAIT могут выполняться многократно. Новые значения в теневых гранях могут использоваться только после выполнения директивы SHADOW_WAIT.

Рассмотрим следующий пример:

```
        REAL A(100,100), B(100,100),
             C(100,100), D(100,100)
CDVM$   ALIGN (I,J) WITH C(I,J)::A,B,D
CDVM$   DISTRIBUTE (BLOCK,BLOCK)::C
        ...
CDVM$   SHADOW_GROUP AB(A,B)
        ...
CDVM$   SHADOW_START AB
        ...
!$OMP   DO
CDVM$   ON C(I,J), SHADOW_WAIT AB
        DO I=2,99
```

```
        DO J=2,99
          C(I,J) = (A(I-1,J) + A(I+1,J) +
     *          A(I,J-1) + A(I,J+1) ) / 4
          D(I,J) = (B(I-1,J) + B(I+1,J) +
     *          B(I,J-1) + B(I,J+1)) / 4
        ENDDO
      ENDDO
!$OMP ENDDO
```

Между директивой SHADOW_START и директивой SHADOW_WAIT в программе могут стоять вычисления, которые будут перекрывать обмены с соседними узлами.

Удаленные данные типа REMOTE. Иногда доступ к удаленным данным требует заведения специальных буферов и соответствующей замены "удаленных" ссылок на массив, через которые происходят обращения к удаленным элементам массива. Удаленные ссылки типа REMOTE специфицируются директивой REMOTE_ACCESS.

Рассмотрим следующий фрагмент:

```
        DIMENSION A(100,100), B(100,100)
CDVM$   DISTRIBUTE (*,BLOCK) ::  A
CDVM$   ALIGN B(I,J) WITH A(I,J)
        ...
!$OMP   PARALLEL DO
CDVM$   ON A(I,J), REMOTE_ACCESS(B(I,N))
C       рассылка значений B(I,N) по узлам
C       OWN(A(I,J))
        DO I= 1,100
          DO J =1,100
             A(I,J) = B(I,J) + B(I,N)
          ENDDO
        ENDDO
!$OMP   END PARALLEL DO
```

Директива REMOTE_ACCESS в параллельном цикле специфицирует удаленные данные (N-й столбец матрицы B) для всех узлов, на которые распределен массив A (при этом на каждый процессор будет переслана только необходимая ему часть столбца матрицы B).

Если в директиве REMOTE_ACCESS указано имя группы, то выполнение директивы происходит в асинхронном режиме. Для спецификации этого режима необходимы следующие дополнительные директивы:

- описание имени группы:
  CDVM$ REMOTE_GROUP RS;
- директивы PREFETCH и RESET.

Рассмотрим следующий фрагмент многообластной задачи.

```
      REAL A1(M,N1+1), A2(M1+1,N2+1)
CDVM$ DISTRIBUTE (BLOCK,BLOCK) ::  A1,A2
CDVM$ REMOTE_GROUP RS
C$OMP PARALLEL
      DO ITER=1,MIT
CDVM$  PREFETCH RS
       ...
C$OMP  DO
CDVM$  ON A1(I,N1+1),REMOTE_ACCESS(RS:A2(I,2))
       DO 10 I=1,M1
10        A1(I,N1+1) = A2(I,2)
C$OMP  DO
CDVM$  ON A2(I,1), REMOTE_ACCESS(RS:A1(I,N1))
       DO 20 I= 1,M1
20        A2(I,1) = A1(I,N1)
      ...
CDVM$  RESET RS
      ...
       ENDDO
C$OMP END PARALLEL
```

При первом прохождении указанной последовательности операторов директива PREFETCH не выполняется. Директивы REMOTE_ACCESS выполняются в обычном синхронном режиме. При этом происходит накопление ссылок в переменной RS. После выполнения всей последовательности директив REMOTE_ACCESS значение переменной RS равно объединению подгрупп удаленных ссылок $r_i \cup \ldots \cup r_n$.

При втором и последующих прохождениях директива PREFETCH осуществляет упреждающую пересылку удаленных данных для всех ссылок, составляющих значение переменной RS. После директивы PREFETCH и до первой директивы REMOTE_ACCESS с тем же именем группы можно выполнять другие вычисления, которые перекрывают ожидание обработки удаленных ссылок. При этом директивы REMOTE_ACCESS никакой пересылки данных уже не вызывают.

Если характеристики группы удаленных ссылок изменились, то необходимо присвоить неопределенное значение группе удаленных ссылок с помощью директивы RESET, после чего будет происходить новое накопление группы удаленных ссылок.

Выводы

Предложенные расширения OpenMP позволяют OpenMP-программистам писать эффективные программы для распределенных систем. Распределение данных между узлами и доступ к удаленным данным можно выполнять по модели, аналогичной модели DVM.

Основными достоинствами предложенного подхода по расширению OpenMP являются:

- высокоуровневая модель программирования, позволяющая описывать алгоритм в терминах массива целиком, а не требовать от программиста манипулирования локальными частями массива,

размеры которых зависят от числа используемых узлов;

– существование одного варианта программы для эффективного выполнения на мультипроцессорах и распределенных системах, поскольку все директивы распределения данных и доступа к удаленным данным оформляются в виде специальных комментариев.

В ближайшее время планируется реализация экспериментального компилятора (конвертора) с языка OpenMP Fortran, расширенного директивами распределения данных и доступа к удаленным данным, на язык Fortran в рамках DVM-системы [5].

Список литературы

1. *Dagumb L., Menon R.* OpenMP: An industry-standard API for shared memory programming // Computational Science and Engineering. 1998. Vol. 5, № 1.

2. *David W. Walker* The design of a standard message-passing interface for distributed memory concurrent computers // Parallel Computing. 1994. Vol. 20, № 4.

3. *Konovalov N.A., Krukov V.A., Mihailov S.N., Pogrebtsov A.A.* Fortran DVM — a language for portable parallel program development // Proc. of Software for Multiprocessors & Supercomputers: Theory, Practice, Experience. Institute for System Programming, RAS, Moscow, 1994.

4. *Коновалов Н.А., Крюков В.А., Михайлов С.Н., Погребцов А.А.* Fortran DVM — язык разработки мобильных параллельных программ // Программирование. 1995. № 1.

5. *Коновалов Н., Крюков В.* Параллельные программы для вычислительных кластеров и сетей // Открытые системы. 2002. № 3.