РАСПАРАЛЛЕЛИВАНИЕ С ПОМОЩЬЮ DVM-СИСТЕМЫ НЕКОТОРЫХ ПРИЛОЖЕНИЙ ГИДРОДИНАМИКИ ДЛЯ КЛАСТЕРОВ С ГРАФИЧЕСКИМИ ПРОЦЕССОРАМИ

В.А. Бахтин, И.Г. Бородич, Н.А. Катаев, М.С. Клинов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула, Ю.Л. Сазанов

В статье рассматривается аспект использования DVM-системы, связанный с тем, что исходные Fortran-программы необходимо преобразовывать, а не только вставлять директивы языков Fortran-DVM и Fortran-DVMH. А также производится сравнение с преобразованиями, необходимыми для эффективной работы автоматически распараллеливающего компилятора, который расставляет директивы автоматически. Приведены экспериментальные данные об эффективности выполнения программ на графических и универсальных процессорах кластера К-100.

В качестве задач гидродинамики были выбраны двумерная и трехмерная задачи о течении несжимаемой жидкости или слабосжимаемого газа около прямоугольной выемки, в которых в качестве исходной математической модели используется гиперболический вариант квазигазодинамической системы. Они называются Каверна и Контейнер соответственно [1].

Программа Каверна предназначена для моделирования на многопроцессорных системах циркуляционного течения в плоской квадратной каверне с движущейся верхней крышкой в двумерной постановке в широком диапазоне как параметров задачи, так и параметров численного метода. Контейнер предназначен для численного моделирования течения вязкой тяжелой жидкости под действием силы тяжести в прямоугольном контейнере с открытой верхней стенкой и отверстием в одной из боковых стенок в трехмерной постановке также в широком диапазоне параметров задачи и параметров численного метода.

1. Распараллеливание приложений при помощи директив языка Fortran DVMH

Разработка параллельных программ состояла в преобразовании текста последовательных программ и добавлении директив языка Fortran DVMH [2, 3].

Последовательная версия Каверны занимает 496 строк. Для получения параллельной программы были проведены следующие действия.

- Заведены вспомогательные массивы

```
tmp1(0:nx1,0:ny1), tmp2(0:nx1,0:ny1)
```

и

```
IDX(2)
```

- Разбит каждый из 4-х циклов с присваиваниями вида

```
do i = 1,nx
   ro(i,0) = …
   …
   ro(i,ny1) = …
   …
enddo
```

на два цикла

```
do i = 1,nx
   ro(i,0) = …
   …
enddo
do i = 1,nx
   ro(i,ny1) = …
   …
enddo
```

для того, чтобы виток цикла целиком мог выполняться на одном процессоре. Перенос одного оператора в другой цикл здесь и далее считаем за операцию изменения одной строки кода

- Заменены операторы присваивания в отдельные элементы массивов на циклы в 1 месте программы для 20-ти операторов. Это потребовалось сделать из-за того, что в текущей версии DVMH в регион могут входить только параллельные циклы.
- Сделаны циклы в 4-х местах программы

```
do j = 1,ny
   hy4 = hy(j+1)+2.d0*hy(j)+hy(j-1)
   do i = 0,nx
      …
   enddo
```

```
enddo
```
тесно-вложенными циклами
```
do j = 1,ny
   do i = 0,nx
      hy4 = hy(j+1)+2.d0*hy(j)+hy(j-1)
      …
   enddo
enddo
```
Это позволяет в процессе выполнения программы один раз отобразить витки двумерного цикла, а не делать это для цикла по i в каждом витке цикла по j

- Изменен порядок вычисления витков циклов для 6-ти гнезд циклов

```
do i = 1,nx
   do j = 1,ny
      SFro(i,j) = …
      …
   enddo
enddo
```
на цикл
```
do j = 1,ny
   do i = 1,nx
      SFro(i,j) = …
      …
   enddo
enddo
```
Это позволяет обрабатывать элементы массивов согласно их расположению в памяти ЭВМ и должно было бы сделано в исходной программе

- Разбиты циклы с зависимостями между витками типа OUTPUT (запись-запись) в 4-х местах программы. Например, такой цикл

```
do j = 1,ny
   do i = 0,nx
      …
      ro1(i,j) = ro1(i,j) + Frox
      ro1(i+1,j) = ro1(i+1,j) - Frox
      …
   enddo
enddo
```
был заменен на два цикла
```
do j = 1,ny
   do i = 0,nx
      …
     tmp1(i,j) = Frox
     ro1(i,j) = ro1(i,j) + Frox
     …
   enddo
enddo
do j = 1,ny
   do i = 0,nx
      ro1(i+1,j) = ro1(i+1,j) - tmp1(i,j)
      …
   enddo
enddo
```
массив `tmp1` использовался для переноса значения переменной `Frox` из одного цикла в другой, чтобы не перевычислять его по громоздкой формуле

- Заменен цикл

```
do j = 1,ny
   do i = 1,nx
      …
      imax = i
      jmax = j
      …
   enddo
enddo
```

на цикл и операторы присваивания

```
do j = 1,ny
   do i = 1,nx
      …
      IDX(1) = i
      IDX(2) = j
      …
   enddo
enddo
imax = IDX(1)
jmax = IDX(2)
```

Это сделано для того, чтобы сохранять значения итерационных переменных цикла, при которых достигается максимум некоторой функции, в элементах массива. Это необходимое требования для того, чтобы реализовать параллельное выполнение программы при помощи одной директивы языка Fortran DVMH

- Заменен оператор вывода распределенного массива на оператор вывода скалярных переменных и предварительного копирования из массива в скалярные переменные, поскольку DVM накладывает ограничения на использование распределенных массивов в операторах ввода-вывода

```
write(20,703) xx,yy,RO1(i,j),Ux1(i,j),Uy1(i,j),
     #                      p1(i,j),E1(i,j)
```

на

```
pr_v1=RO1(i,j)
pr_v2=Ux1(i,j)
pr_v3=Uy1(i,j)
pr_v4=p1(i,j)
pr_v5=E1(i,j)
write(20,703) xx,yy,pr_v1,pr_v2,pr_v3,pr_v4,pr_v5
```

А также ряд действий, которые связаны с добавлением DVMH-директив:

- Вставлены директивы распределения данных

```
CDVM$ DISTRIBUTE ro(BLOCK,BLOCK)
CDVM$ ALIGN (i,j) WITH ro(i,j):: ux, uy, p, E, ro1, ux1, uy1, E1, p1
CDVM$ ALIGN (i,j) WITH ro(i,j):: SFro, SFux, SFuy, SFE, tmp1, tmp2
CDVM$ ALIGN (i) WITH ro(*,*):: hx, hy
```

- Вставлены директивы PARALLEL перед 28-ю гнездами циклов
- 8 параллельных циклов имеют спецификацию private
- 2 цикла спецификацию REDUCTION
- 7 циклов спецификацию SHADOW_RENEW
- Вставлены директивы начала и конца вычислительного региона в 7-ми местах программы

```
!DVM$ region
```

и

```
!DVM$ end region
```

- Вставлены директивы объявления данных актуальными в 6-ти местах программы

```
!DVM$ actual
```

- Вставлены директивы запроса актуальных данных в 5-ти местах программы

```
!DVM$ get_actual
```

- Вставлена одна директива REMOTE_ACCESS для доступа к удаленным данным (данным, не расположенным на процессоре, который должен выполнить оператор)

Последовательная версия программы Контейнер занимает 828 строки. Для нее были проведены схожие с Каверной действия.

- Заведены вспомогательные массивы
- Разбит каждый из 3-х гнезд циклов на два гнезда циклов для того, чтобы виток цикла целиком мог выполняться на одном процессоре
- Сделаны тесно-вложенные циклы в 6-ти местах программы
- Изменен порядок вычисления витков циклов для 12-ти гнезд циклов
- Разбит цикл с зависимостями между витками типа OUTPUT (запись-запись) и FLOW (запись-чтение)

```
pbeg = …
do jj = 1,nz
   k = nz+1-jj
```

```
   do i = 1,nx
      do j = 1,ny
         …
         p(i,j,k) = pbeg
         …
      enddo
   enddo
   pbeg = pbeg - grav*hz(k)
enddo
```

на два цикла

```
pbeg = …
pbeg_arr(nz+1)=pbeg
do jj = 1,nz
   k = nz+1-jj
   pbeg = pbeg - grav*hz(k)
   pbeg_arr(k) = pbeg
enddo
do k = 1,nz
   do j = 1,ny
      do i = 1,nx
         …
         p(i,j,k) = pbeg_arr(k+1)
         …
      enddo
   enddo
enddo
```

Один из них содержит зависимости, а другой (с присваиваниями в основные массивы программы) нет. К тому же для второго 3-х мерного цикла, поскольку он теперь без зависимостей, изменен порядок обращения к элементам массива p по 3-ему измерению с обратного на прямой.

- Изменен один 3-х мерный цикл, чтобы сохранять значения итерационных переменных цикла, при которых достигается максимум некоторой функции, в элементах массива
- Заменен оператор вывода распределенного массива на оператор вывода скалярных переменных и предварительного копирования из массива в скалярные переменные
- Перенесен оператор из вычислительного региона за его пределы в 4-х местах программы, чтобы не разбивать регион на два региона

```
!dvm$   region in(hx,hy,hz)
…
dt = beta/um
…
!dvm$   end region
```

на

```
!dvm$   region in(hx,hy,hz)
…
!dvm$   end region
!dvm$ get_actual(um)
dt = beta/um
```

это перенос за вычислительный регион

и в случае замены такого фрагмента программы

```
!dvm$   region in(hx,hy,hz),inout( SFux,SFuy,SFuz,ux,uy,uz)
…
outf = 0.d0
…
!dvm$   end region
```

на такой фрагмент

```
outf = 0.d0
!dvm$ actual(uu,IDX,outf,Slid,coef)
!dvm$   region in(hx,hy,hz),inout( SFux,SFuy,SFuz,ux,uy,uz)
…
!dvm$   end region
```

имеет место перенос оператора на место перед вычислительным регионом

А также вставлены строки, содержащие DVMH-директивы:

- Вставлены директивы распределения данных
- Вставлены директивы PARALLEL перед 21-м гнездом циклов
- 9 параллельных циклов имеют спецификацию private
- 4 цикла спецификацию REDUCTION
- 5 циклов спецификацию SHADOW_RENEW
- Вставлены директивы начала и конца вычислительного региона в 5-ти местах программы
- Вставлены директивы объявления данных актуальными в 4-х местах программы
- Вставлены директивы запроса актуальных данных в 3-х местах программы
- Вставлена одна директива REMOTE_ACCESS для доступа к удаленным данным

2. Распараллеливание приложений при помощи автоматически распараллеливающего компилятора

Рассмотрим процесс распараллеливания Каверны с использованием автоматически распараллеливающего компилятора (АРК) [4, 5].

Последовательная Fortran-программа для Каверны (496 строк) подается на вход АРК, в процессе работы которого создается файл с информацией о ходе распараллеливания. В нем представлены, в том числе, и причины, по которым циклы остались не распараллелены. Последовательная программа изменяется программистом и снова подается на вход АРК.

Во многом причины устраняются способом, похожим на ручное распараллеливание и описанное выше, но есть специфические преобразования текста последовательной программы (изменение строк и добавление новых), которые необходимы для АРК. Но как мы покажем далее, они являются на порядок проще.

В частности, для одного из циклов АРК обнаружил наличие OUTPUT зависимости по переменной imax, и не определил что она редукционная (поиск максимума с сохранением значений итерационных переменных цикла). Программист должен специфицировать наличие такой зависимости в данном цикле при помощи спецкомментария, начинающегося с “CPRG”. Требуется добавление 1 строки.

В случае доступа к соседним элементам (например, с индексами ny и ny+1) через косвенную переменную (например, ny1, которая на самом деле равна ny+1), надо подставить явно выражение для этой переменной в текст программы в местах ее использования (при доступе к элементу массива и при описании размеров циклов). Без такой подстановки в программе возникают операции копирования лишних элементов массивов, и сами операции в некоторых местах программы являются лишними (массив не менялся, буфер для элементов массива с других процессоров можно не обновлять).

Характеристики полученных и исходных программ приведены в таблице 1. Под программой с обозначением Fortran АРК подразумевается программа на языке Fortran, подаваемая на вход АРК. Эта программа имеет спецкомментарий, который является комментарием с точки зрения языка Fortran и описывает свойство последовательной программы. Процентное соотношение приведено относительно исходной Fortran-программы. В процессе работы АРК директивы языка Fortran-DVM вставляются автоматически.

Таблица 1. Характеристики программ для Каверны и Контейнера

Программы	Характеристики	Fortran	Fortran-DVMH	Fortran АРК
Каверна	общее число строк	496	613 (123.5%)	554 (111.6%)
	добавлено строк с директивами	-	87 (17.5%)	-
	добавлено строк с не директивами	-	30 (6%)	58 (11.6%)
	изменено строк	-	45 (9%)	87 (17.5%)
Контейнер	общее число строк	828	942 (113.7%)	864 (104.3%)
	добавлено строк с директивами	-	80 (9.6%)	-
	добавлено строк с не директивами	-	34 (4.1%)	36 (4.3%)
	изменено строк	-	37 (4.4%)	46 (5.5%)

Отметим, что трудоемкость вставки директив вручную программистом значительно выше, чем трудоемкость преобразования им текста последовательной программы, а наиболее сложные преобразования одинаковы для ручного распараллеливания и для подготовки программы для АРК. Преобразования, необходимые только для АРК, выполняются существенно проще.

3. Эффективность параллельных программ для выбранных приложений гидродинамики

Все запуски программ проводились на суперкомпьютере К-100 (ИПМ им. М.В. Келдыша РАН) [6]. Для всех запусков в программе Каверна размеры массивов были 1600x1600 и выполнялось 200 итераций, в Контейнере размеры массивов 120x120x120 и 200 итераций.

Для запуска на одном узле без использования графических ускорителей программы компилировались как Fortran-DVM-программы, при выполнении которых используется библиотека передачи сообщений MPI. Для запуска с использованием графических ускорителей программы компилировались как Fortran-DVMH-программы, при выполнении которых используются средства MPI и CUDA. Замерялись времена основного

итерационного цикла, без циклов инициализации массивов начальными значениями и без вывода результирующих массивов в файл. При этом использовался компилятор PGI Fortran CUDA 11.5 с опцией компиляции -O3. Времена выполнения приведены в таблицах 2 и 3.

Таблица 2. Времена выполнения основного итерационного цикла для Каверны и Контейнера на суперкомпьютере К-100 с использованием одного вычислительного узла без графических ускорителей (в секундах)

Программы		Число ядер					
		1	2	4	8	10	11
Каверна	Fortran	228.15					
	Fortran-DVMH	257.16	131.89	72.93	43.72	38.46	35.85
Контейнер	Fortran	778.73					
	Fortran-DVMH	886.39	410.88	211.52	110.85	90.24	82.26

Таблица 3. Времена выполнения основного итерационного цикла DVMH-программы для Каверны и Контейнера на суперкомпьютере К-100 с использованием графических ускорителей (в секундах)

Число графических ускорителей	1	2	3	4	6	9	12
Каверна	19.94	10.34	7.13	5.45	4.00	2.87	2.28
Контейнер	47.61	25.46	19.22	15.80	11.99	9.32	8.36

Для Каверны использование одного вычислительного узла с 11 ядрами универсальных процессоров и без использования графических ускорителей позволяет ускорить выполнение основного итерационного цикла в 6.3 раза по сравнению с последовательной программой, а с использованием графических ускорителей в 32 раза. Использование 4 вычислительных узлов (по 3 графических ускорителя в каждом узле) позволяет получить ускорение в 100 раз ровно.

Для Контейнера использование одного вычислительного узла с 11 ядрами универсальных процессоров и без использования графических ускорителей позволяет ускорить выполнение в 9.4 раза, а с использованием графических ускорителей в 40.5 раза. Использование 4 вычислительных узлов позволяет получить ускорение в 93.1 раза.

Эффективность автоматического распараллеливания измерялась путем сравнения времени выполнения программ, полученных при помощи АРК из последовательных программ и полученных при распараллеливании этих же программы вручную при помощи директив языка Fortran-DVM.

В результате работы АРК получается по одному варианту параллельной программы. Для программы Каверна вариант автоматического распараллеливания основан на двумерном распределении массивов, также как и вариант ручного распараллеливания этой программы. А для программы Контейнер вариант автоматического распараллеливания основан на двумерном распределении массивов, в то время как вариант ручного распараллеливания этой программы основан на трехмерном распределении массивов. Запуски проводились с использованием компилятора Intel Fortran с опцией оптимизации по умолчанию.

Времена запусков приведены в таблице 4. Обозначение 1s означает компиляцию и запуск на 1 процессе в режиме игнорирования DVM-директив (dvm f –s <имя программы>). Таким образом, программа запускается не как Fortran-DVM-программа, а как Fortran-программа.

Таблица 4. Времена выполнения программы Каверна и Контейнер на кластере К-100 (в секундах)

Варианты программ	Число процессоров			
	1s	1	64	256
Каверна АРК	133.09	257.59	5.72	2.23
Каверна ручная	133.57	257.09	5.71	1.92
Контейнер АРК	427.94	838.92	14.19	3.87
Контейнер ручной	429.10	864.69	15.16	4.54

Компилятор Intel Fortran хорошо оптимизирует Fortran-программы без обращений к функциям библиотеки системы поддержки, без использования для обращения к элементам массивов специальных буферов и сложной индексации по ним. Это проявляется при запуске, обозначенным как 1s.

АРК позволяет ускорить выполнение последовательной Fortran-программы в 59 раз на 256 процессорах для программы Каверна, и в 110 раз на 256 процессорах для программы Контейнер. Времена выполнения программ, полученных через АРК, сравнимы с программами, написанными вручную. А в случае программы Контейнер даже лучше. Мы это связываем с использованием при автоматическом распараллеливании двумерного распределения данных, а при ручном распараллеливании трехмерного распределения и связанного с этим большего количества коммуникационных обменов, в частности операций обмена теневыми гранями массивов между процессорами.

Для всех параллельных программ (написанных вручную при помощи директив языка Fortran-DVMH, директив языка Fortran-DVM или полученных автоматически через АРК) была проверена корректность их

работы путем сравнения результатов работы. Отличия незначительны и они обусловлены измененным порядком выполнения арифметических операций.

Заключение

Преобразования последовательных программ, необходимые при ручном распараллеливании или при подготовке программ для автоматически распараллеливающего компилятора, преимущественно были простыми, за исключением разбиения циклов с зависимостями по данным на несколько циклов. Разбиение цикла либо устраняет зависимость (в программе Каверна), либо позволяет уменьшить время его выполнения (в программе Контейнер). Цикл с зависимостью не будет распараллелен, а цикл с выделенными из него вычислениями будет распараллеленным.

Определение того, какие циклы можно сделать параллельными, не содержат ли они нераспараллеливаемых вычислений с зависимостями по данным (некоторые вычисления с зависимостями по данным, например регулярные, можно распараллелить, организуя конвейерное выполнение), какие данные в них читаются, какие данные являются приватными, какие данные редукционными – все это свойственно параллельному программированию на любом языке, но задать их через язык Fortran DVMH гораздо проще, чем при использовании низкоуровневых средств MPI и CUDA.

В настоящий момент производится развитие автоматически распараллеливающего компилятора для распараллеливания класса программ, включающего выбранные задачи гидродинамики, на графические процессоры.

Работы по расширению модели DVM и автоматизации распараллеливания последовательных программ поддержаны ФЦП “Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007-2013 годы” ГК № 07.514.11.4030, программами президиума РАН №15, №16 и №18, грантами РФФИ № 10-07-00211 и № 11-01-00246, грантами Президента РФ МК-6772.2012.9 и НШ-4307.2012.9.

ЛИТЕРАТУРА:

1. А.А. Давыдов, Б.Н. Четверушкин, Е.В. Шильников “Моделирование течений несжимаемой жидкости и слабосжимаемого газа на многоядерных гибридных вычислительных системах” // Журнал “Вычислительная математика и математическая физика”, 2010, Т. 50, № 12, С. 2275-2284
2. В.А. Бахтин, М.С. Клинов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула, Ю.Л. Сазанов “Расширение DVM-модели параллельного программирования для кластеров с гетерогенными узлами” // Труды Международной научной конференции “Научный сервис в сети Интернет: экзафлопсное будущее”, Новороссийск, сентябрь 2011, М.: Изд-во МГУ, 2011, С. 310-315
3. DVM система: сайт. URL: http://www.keldysh.ru/dvm (дата обращения 31.05.2012)
4. В.А. Бахтин, М.С. Клинов, В.А. Крюков, Н.В. Поддерюгина “Автоматическое распараллеливание последовательных программ для многоядерных кластеров” // Труды Всероссийской научной конференции “Научный сервис в сети Интернет: суперкомпьютерные центры и задачи”, Новороссийск, сентябрь 2010, М.: Изд-во МГУ, 2010, С. 12-15
5. В.А. Бахтин, Н.А. Катаев, М.С. Клинов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула. “Автоматическое распараллеливание Фортран-программ на кластер с графическими ускорителями” // Сборник трудов Международной научной конференции “Параллельные вычислительные технологии” (ПаВТ’2012), Новосибирск, март 2012, С. 373-379
6. ЭВМ К-100. URL: http://www.kiam.ru/MVS/resourses/k100.html (дата обращения 31.05.2012)