Автоматическое распараллеливание Фортран-программ на кластер с графическими ускорителями*

В.А. Бахтин, Н.А. Катаев, М.С. Клинов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула

Институт прикладной математики имени М.В. Келдыша РАН

Описывается развитие алгоритмов автоматического распараллеливания Фортран-программ на кластер для использования графических ускорителей в его узлах. Результатом работы алгоритмов является текст программы на высокоуровневом языке Fortran DVMH. Основная задача алгоритмов заключается в нахождении параллельных циклов, которые могут выполняться на ускорителях, определении требуемых этим циклам данных и режимов их использования, объединении таких циклов в более крупные вычислительные регионы с целью сокращения передач данных между ускорителями и центральным процессором.

1. Введение

Программирование высокопроизводительных кластеров и других параллельных систем продолжает оставаться исключительно сложным делом, доступным узкому кругу специалистов и крайне трудоемким даже для них.

Появление кластеров с гетерогенными узлами, использующих в качестве ускорителей графические процессоры (GPU), еще более усложнило разработку программ. Для них требуется использовать помимо технологий MPI или SHMEM для передачи сообщений между узлами кластера еще и низкоуровневую технологию CUDA или OpenCL для программирования вычислений на ускорителе. Программировать, отлаживать, сопровождать, переносить на другие ЭВМ такие программы гораздо сложнее. Например, при переходе от GPU фирмы NVIDIA к GPU фирмы AMD придется заменить CUDA на OpenCL.

Альтернативным вариантом является использование высокоуровневых моделей и соответствующих им языков программирования [1-15] (HPF, OpenMP, DVM, DVMH, HMPP, hiCUDA, PGI APM, OpenACC, Co-Array Fortran, UPC, Titanium, Chappel, X10, Fortress). Особое место занимают модели, реализуемые посредством добавления в программы на стандартных последовательных языках спецификаций, управляющих отображением этих программ на параллельные машины. Эти спецификации, оформляемые в виде комментариев в Фортран-программах или директив компилятору (прагм) в программах на языках Си и Си++, не видны для обычных компиляторов, что значительно упрощает внедрение новых моделей параллельного программирования. Такими моделями для кластеров являются HPF и DVM, а для мультипроцессоров - OpenMP. В последние годы разработаны высокоуровневые модели и для графических процессоров (HMPP, hiCUDA, PGI APM, DVMH, OpenACC). Они имеют между собой много общего, что позволяет говорить о появлении нового подхода к программированию ускорителей типа GPU. Они позволяют программисту, также как в моделях DVM и OpenMP, полностью контролировать отображение данных и вычислений на аппаратуру.

Еще одним вариантом разработки параллельных программ, является использование высокоуровневого инструмента, который автоматически преобразовывал бы последовательную программу на стандартном языке Фортран или Си в параллельную программу для ЭВМ, в том числе с графическими процессорами или потоковыми ускорителями, как в работе [16]. Следует отметить работы по созданию систем автоматизированного распараллеливания для многоядерных кластеров [17-21] (CAPTools/Parawise, FORGE Magic/DM, BERT77, САПФОР, ДВОР), от которых можно ожидать их развития в сторону ЭВМ с графическими процессорами.

* Работа поддержана ФЦП «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007-2013 годы» ГК № 07.514.11.4030, грантами Президента РФ МК-6772.2012.9 и НШ-4307.2012.9, программой Президиума РАН №17 и грантом РФФИ №11-01-00246.

Например, данная работа посвящена такому развитию системы САПФОР.

2. Система САПФОР

Система САПФОР состоит из следующих компонент:

1. Статический анализатор Фортран-программ, который определяет свойства последовательной программы по тексту программы;
2. База данных (БД) осуществляет передачу данных между компонентами системы;
3. Эксперты (DVM-эксперт, OpenMP-эксперт, DVM/OpenMP-эксперт, DVMH-эксперт), которые строят схемы распараллеливания, оценивают их путем прогнозирования характеристик (временных характеристик выполнения), записывают схемы и характеристики в БД;
4. Генератор по схемам из БД строит текст параллельной программы;
5. Диалоговая оболочка визуализирует содержимое БД и соотносит его с текстом программы, запускает компоненты системы, позволяет программисту (пользователю) уточнить результаты анализа.

В последнее время система развивалась в сторону поддержки современного Фортрана, создания DVMH-эксперта для кластеров с графическими ускорителями, улучшения версии диалоговой оболочки для работы с реальными приложениями.

Этап генерации текста параллельной программы получается значительно проще, если использовать в качестве промежуточного языка высокоуровневый язык, для которого уже разработаны соответствующие компиляторы. В терминах этого языка строятся схемы распараллеливания в эксперте, которые представляют собой правила преобразования текста последовательной программы.

В настоящее время известен только один высокоуровневый язык для организации вычислений в кластере с графическими ускорителями. Это язык Fortran DVMH [4], который является развитием Fortran DVM. Именно его использует DVMH-эксперт.

3. Алгоритм автоматического отображения на кластер с графическими процессорами

Работа DVMH-эксперта начинается с запуска алгоритмов DVM-эксперта [22], который осуществляет организацию параллельных вычислений (распределение данных, распределение вычислений и организацию коммуникаций) между узлами, и отбирает наилучшую схему распараллеливания. Потом DVMH-эксперт дополняет эту схему конструкциями, которые необходимы для использования графических ускорителей в узлах кластера:

- Определяет, какие DVM-циклы должны стать вычислительными регионами, что означает, что они будут выполняться на графических ускорителях (есть ограничения, например, в регионе нельзя вызывать внешние процедуры, отличные от встроенных процедур Фортрана), формирует для DVM-циклов спецификацию приватных для него переменных (этих спецификаций не было в DVM-программе);
- Объединяет регионы в более крупные, вставляет для них директивы начала и конца региона;
- Определяет нужные этим регионам данные и режим их использования;
- Вставляет во фрагменты программы, не попавшие в регионы, указания об актуализации данных и их модификации.

В результате DVMH-эксперт дополняет схему распараллеливания, полученную после работы алгоритмов DVM-эксперта, конструкциями, которые необходимы для использования графических ускорителей в узлах кластера.

Рассмотрим работу алгоритма на примере численного решения краевой задачи для уравнения Лапласа в прямоугольной области итерационным методом Якоби (см. Рис. 1).

```
 1          PROGRAM    JAC
 2          PARAMETER    (L=2000,  ITMAX=20)
 3          REAL     A(L,L), EPS, MAXEPS, B(L,L)
 4          MAXEPS  =  0.5E - 7
 5          DO J  =  1, L
 6          DO I  =  1, L
 7              A(I,  J)  =  0.
 8              IF(I.EQ.1 .OR. J.EQ.1 .OR. I.EQ.L .OR. J.EQ.L) THEN
 9                  B(I,  J) = 0.
10              ELSE
11                  B(I,  J)  = ( 1. + I + J )
12              ENDIF
13          ENDDO
14          ENDDO
15
16          DO IT  =  1,  ITMAX
17              EPS  =  0.
18              DO J  =  2, L-1
19              DO I  =  2, L-1
20                  EPS = MAX ( EPS,  ABS( B( I, J) - A( I, J)))
21                  A(I, J)  =  B(I, J)
22              ENDDO
23              ENDDO
24              DO J = 2,  L-1
25              DO I = 2,  L-1
26                  B(I, J) =  (A( I-1, J ) + A( I, J-1 ) +
27       *              A( I+1, J)+ A( I, J+1 )) / 4
28              ENDDO
29              ENDDO
30              PRINT 200,  IT, EPS
31  200         FORMAT(' IT = ',I4, '   EPS = ', E14.7)
32              IF ( EPS . LT . MAXEPS ) GOTO 3
33          ENDDO
34      3   OPEN (3, FILE='JAC.DAT', FORM='FORMATTED', STATUS='UNKNOWN')
35          WRITE (3,*)   B
36          CLOSE (3)
37          END
```

Рис. 1. Программа на языке Fortran численного решения краевой задачи для уравнения Лапласа в прямоугольной области итерационным методом Якоби

Алгоритмы DVM-эксперта строят схемы распараллеливания, которые отличаются вариантами распределения данных, а именно:

Вариант 0

!DVM$ DISTRIBUTE (BLOCK,BLOCK) :: a – означает распределение массива A по двум измерениям, например, при использовании 4 процессоров в конфигурации 2x2 массив A будет разрезан по каждому измерению на 2 части. Во внутреннем представлении программы все переменные обозначены малыми буквами, это объясняет их использование в директиве и отличие от переменных в тексте программы.

!DVM$ ALIGN b(i,j) WITH a(i,j) – означает точное соответствие распределения между элементами массивов A и B. Там где находится элемент A(i,j) будет расположен и B(i,j).

Вариант 1

!DVM$ DISTRIBUTE (*,BLOCK) :: a – означает распределение массива A по второму измерению, например, при использовании 4 процессоров массив A будет разрезан по столбцам на 4 части.

!DVM$ ALIGN b(i,j) WITH a(i,j)

Вариант 2

!DVM$ DISTRIBUTE (BLOCK,*) :: a

!DVM$ ALIGN b(i,j) WITH a(i,j)

Вариант 3

Пусто – означает, что не надо распределять массивы A и B, директивы распределения данных не вставлять

Затем для каждого варианта распределения данных строится вариант распределения вычислений и организация коммуникаций.

Для вариантов 0, 1 и 2

Цикл на строках с номерами 5 и 6 – "!DVM$ PARALLEL (j,i) ON a(i,j)" – означает, что виток с индексами (j,i) будет выполняться на том процессоре, на котором имеется элемент a(i,j)

Цикл на строке 16 не будет распараллелен, потому что в нем есть оператор печати (ввода-вывода)

Цикл на строках с номерами 18 и 19 – "!DVM$ PARALLEL (j,i) ON a(i,j),
DVM$ REDUCTION (MAX(eps))" – означает, что цикл будет распараллелен, а после него будет выполнена операция редукции для переменной eps, копии которой имеются на всех процессорах.

Цикл на строках с номерами 24 и 25 – "!DVM$ PARALLEL (j,i) ON b(i,j),
DVM$ SHADOW_RENEW (a(1:1,1:1))" – означает, что цикл будет распараллелен, а перед его выполнением будет произведена операция обмена между процессорами теневыми гранями массива A. На каждом процессоре будет заведена память не только под локальную часть массива, но и под теневую грань (расширение локальной части массива), в которую будут копироваться значения этого массива с соседних процессоров.

Для варианта 3

Никаких директив не будет вставлено, потому что все массивы размножены по процессорам.

Таким образом, построено 4 схемы. Эксперт оценивает их. При обмене теневыми гранями в случае одномерного распределения данных (вариант 1 и 2), например, для 100 процессоров требуется 198 (99*2) пересылок по 2000 элементов массива, а при двумерном (вариант 0) для конфигурации процессоров 10x10 – 360 пересылок по 200 элементов массива. Для оценки времени передачи данных в системе САПФОР используются параметры, которые соответствуют латентности и времени передачи каждого байта данных. Для нашего примера будем использовать, например, такие параметры Tstart = 7 мкс, Tbyte = 0.004 мкс, и будем считать, что элемент массива занимает 4 байта. Тогда для одномерного случая потребуется 11880 (198*15*4) мкс, а для двумерного 11232 (360*7.8*4) мкс, поэтому лучшей схемой по этим оценкам является схема с номером 0.

Затем DVMH-эксперт строит граф управления с крупными вершинами. Вершина соответствует нескольким операторам программы, и будем называть их блоком. Для него определяются в процессе работы алгоритма все данные, которые изменяются или читаются в блоке.

Для выбранного примера граф управления будет построен в следующем виде (см. Рис. 2). A(*,*) означает использование всего массива.

Номер блока	Строки операторов блока	Пишет данные	Читает данные	Следующий блок	Предыдущие блоки
1	4	MAXEPS		2	
2	5-14	A(*,*), B(*,*), I, J	I, J	3	1
3	16, 17	IT, EPS		4	1-6
4	18-23	A(*,*), EPS, I, J	A(*,*), B(*,*), EPS, I, J	5	1-6
5	24-29	B(*,*), I, J	A(*,*), I, J	6	1-6
6	30-33		IT, EPS, MAXEPS	3, 7	1-6
7	34-36		B(*,*)		1-6

Рис. 2. Граф управления для программы численного решения краевой задачи для уравнения Лапласа в прямоугольной области итерационным методом Якоби

Определяются циклы, которые входят в состав DVM-циклов: для варианта с номером 0 – это циклы на строках 5 и 6, 18 и 19, 24 и 25. Производится проверка возможности их оформле-

ния в виде вычислительных регионов. В данном примере они удовлетворяют ограничениям для регионов. Получается три региона. Приватных для DVM-цикла переменных в них нет.

Затем производится попытка объединить регионы. Цикл на строке 18 и цикл на строке 24 вложены непосредственно в один цикл, который находится на строке 16. Поэтому производится объединение этих регионов. В итоге получается два региона (см. Рис. 3)

Номер региона	Номера блоков графа управления
1	2
2	4, 5

Рис. 3. Набор вычислительных регионов для программы численного решения краевой задачи для уравнения Лапласа в прямоугольной области итерационным методом Якоби

Для них формируется директива языка Fortran DVMH “!DVM$ region”.

Далее происходит определение используемых в регионе данных и типов их использования (входные, выходные, локальные). Рассматривается каждый регион в отдельности.

Регион 1 читает переменные I и J, но они являются итерационными переменными DVM-цикла, работа с ними обеспечивается системой поддержки выполнения программ на графическом ускорителе, поэтому для них не надо указывать тип их использования в регионе. Получается, что нет данных, которые надо пометить как входные для региона. Регион 1 пишет данные A(*,*), B(*,*), I, J. Переменные I и J являются итерационными переменными, а для массивов A и B проверяется, будут ли они использованы далее в программе. Проверка показывает, что будут использоваться, и они помечаются как выходные, иначе были бы локальными. Локальных данных в этом регионе нет.

Регион 2 читает и пишет переменные A(*,*), B(*,*), EPS, I, J. Переменные I и J являются итерационными переменными. Для переменной EPS определяется то, что перед регионом (согласно графу управления) она присваивается, а также используется после региона. То же самое и для массивов A и B, они используются на следующем витке итерационного цикла на строке 16. Поэтому переменные A, B, EPS будут помечены как входные и как выходные для региона, такой тип обозначается как inout. Локальных данных в этом регионе тоже нет.

После этого анализируются операторы программы, которые не включены в регионы, для того чтобы поставить перед ними необходимые операторы актуализации значений переменных. Дело в том, что выходные данные региона не всегда копируются из укорителя на центральный процессор после выполнения региона. Они хранятся на ускорителе с целью повторного их использования в другом регионе.

Операторы на строках 4 и 17 не читают данные, поэтому для них не надо производить подобную актуализацию. Оператор на строке 30 читает переменные IT и EPS, но переменная IT не менялась в предыдущих регионах, поэтому можно использовать ее значение с центрального процессора, копирования дополнительные не нужны. А для переменной EPS нужно копирование. Поэтому перед этим оператором вставляется директива языка Fortran DVMH “!DVM$ get_actual(eps)”. Следующий оператор на строке 32 читает переменные EPS и MAXEPS. Переменная EPS была обновлена оператором на строке 30, и между этими операторами не было регионов, которые выполнялись бы на графическом ускорителе. Поэтому дополнительное копирование для нее не нужно. А переменная MAXEPS не менялась в предыдущих регионах. Оператор на строке 35 читает массив B, который менялся в предыдущих регионах и не был скопирован на центральный процессор. Для него надо вставить директиву “!DVM$ get_actual(b)”.

В результате, получен текст программы на языке Fortran DVMH, который можно скомпилировать и выполнить на кластере с графическими процессорами. Полученный автоматически текст программы близок к тексту программы, написанному вручную, и приведенному в статье [4].

В алгоритме есть еще следующие особенности, которые на данном примере не проявились.

Во-первых, если для операторов цикла, который целиком выполняется на центральном процессоре, надо осуществлять копирования элементов с графического ускорителя, то они выполняются перед циклом, а не в нем. Это позволяет объединить операции копирования к отдельным элементам массива в операции копирования всего массива, и избежать заведомо повторных проверок, связанных с актуализацией одних и тех же данных.

Во-вторых, для массивов хранится множество элементов, которые читаются или изменяются в том или ином фрагменте программы. Несколько подряд идущих элементов объединяются в диапазоны элементов массива. А множество элементов хранится в виде списка, состоящего или из отдельных элементов массива, или из диапазонов. Задание множеств через такие списки позволяет легко находить их пересечения для разных фрагментов программы. Это требуется для определения, надо ли перед чтением элементов массивов на центральном процессоре копировать их из памяти ускорителей в память центрального процессора. Например, для случая, когда массив менялся целиком на графическом ускорителе, потом было несколько присваиваний на центральном процессоре в крайние элементы этого массива, а потом операция чтения не крайних элементов массива, надо скопировать их с графического процессора.

4. Заключение

Появление кластеров с гетерогенными узлами, использующих в качестве ускорителей графические процессоры, поставило вопрос об эффективном автоматическом распараллеливании последовательных программ для них.

Проведенные исследования показали, что такое отображение возможно, если при написании последовательных программ придерживаться определенной дисциплины и предоставить программисту средства для спецификации свойств его программы, недоступных для статического анализа.

Разработанные алгоритмы отображения во многом опираются на разработанные ранее алгоритмы отображения Фортран-программ на многоядерный кластер, существенно используют близость языков Fortran DVM и Fortran DVMH, и возможности компилятора Fortran DVMH.

В настоящее время ведется апробация системы САПФОР и компилятора Fortran DVMH на представительных тестовых программах (среди них тесты NAS) и других реальных приложениях.

Литература

1. High Performance Fortran Forum. High Performance Fortran Language Specification, version 2.0, January 1997. URL: http://hpff.rice.edu/versions/hpf2/index.htm (дата обращения 30.10.2011)
2. OpenMP Application Program Interface. Version 3.1. July 2011. URL: http://www.openmp.org/mp-documents/OpenMP3.1.pdf (дата обращения 30.10.2011)
3. Описание языка Fortran-DVM/OpenMP. Версия 3.0, Октябрь, 2009. URL: http://www.keldysh.ru/dvm/dvmhtm1107/rus/usr/fdvm/fdvmLDr.html (дата обращения 30.10.2011)
4. Бахтин В.А., Клинов М.С., Крюков В.А., Поддерюгина Н.В., Притула М.Н., Сазанов Ю.Л. 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.
5. Dolbeau R., Bihan S., Bodin F. HMPP: A hybrid multicore parallel programming environment.
6. Han T., Abdelrahman T. hiCUDA: A high-level directive-based language for gpu programming. // Proceedings of the 2nd Workshop on General Purpose Processing on Graphics Processing Units, Mar. 2009, P. 52–61.
7. The Portland Group, Inc. The PGI Fortran and C Accelerator Programming Model, March. 2010. URL: htpp://www.pgroup.com/accelerate (дата обращения 30.10.2011).
8. OpenACC. URL: http://www.openacc-standard.org (дата обращения 14.02.2012).
9. Mellor-Crummey J., Adhianto L., Jin G., Scherer W. A New Vision for Coarray Fortran. // The Third Conference on Partitioned Global Address Space Programming Models. Ashburn, VA, October 2009.

10. UPC Consortium. UPC Language Specifications, v1.2 // Lawrence Berkeley National Lab Tech Report LBNL-59208, 2005.
11. Hilfinger P., Bonachea D., Datta K., Gay D., Graham S., Liblit B., Pike G., Su J., Yelick K. Titanium Language Reference Manual. Version 2.20. // U.C. Berkeley Tech Report, UCB/EECS-2005-15.1, August, 2006.
12. Chamberlain B., Callahan D., Zima H. Parallel Programmability and the Chapel Language // International Journal of High Performance Computing Applications, August 2007, 21(3), P. 291-312. URL: http://hpc.sagepub.com/content/21/3/291.abstract (дата обращения 30.10.2011)
13. Milthorpe J., Ganesh V., Rendell A., Grove D. X10 as a Parallel Language for Scientific Computation: Practice and Experience. // IEEE International Parallel and Distributed Processing Symposium, May 2011. URL: http://cs.anu.edu.au/%7EJosh.Milthorpe/publications/Milthorpe2011_IPDPS.pdf (дата обращения 30.10.2011)
14. Cunningham D., Bordawekar R., Saraswat V. GPU Programming in a High Level Language Compiling X10 to CUDA. // ACM SIGPLAN 2011 X10 Workshop, June 2011. URL: http://x10.svn.sourceforge.net/viewvc/x10/external/X10%20Workshop%20%282011%29/papers/cuda.pdf (дата обращения 30.10.2011)
15. Allen E., Chase D., Flood C., Luchangco V., Maessen JW., Ryu S., Steele G.Jr. Project Fortress: A Multicore Language for Multicore Processors // Linux Magazine, September 2007. URL: http://labs.oracle.com/projects/plrg/Publications/linuxMagazine.pdf (дата обращения 30.10.2011)
16. Климов А.В., Окунев А.С. Автоматическое распараллеливание последовательных программ для гибридной системы с ускорителем на основе потока данных.
17. Система ParaWise: сайт.- URL: http://www.parallelsp.com (дата обращения 24.10.2011).
18. Applied Parallel Research. FORGE Magic/DM. URL: http://wotug.ukc.ac.uk/parallel/vendors/apr/ProductInfo/dpf_datasheet.txt (дата обращения 24.10.2011).
19. BERT 77: Automatic and Efficient Parallelizer for FORTRAN. сайт.- URL: http://www.basement-supercomputing.com/content/view/24/50/ (дата обращения 24.10.2011).
20. Бахтин В.А., Бородич И.Г., Катаев Н.А., Клинов М.С., Ковалева Н.В., Крюков В.А., Поддерюгина Н.В. Диалог с программистом в системе автоматизации распараллеливания САПФОР. // Труды Международной научной конференции “Научный сервис в сети Интернет: экзафлопсное будущее”, Новороссийск, сентябрь 2011 – М.: Изд-во МГУ, 2011, С. 67-70.
21. Открытая распараллеливающая система: сайт.- URL: http://www.ops.rsu.ru (дата обращения 24.10.2011).
22. Клинов М.С., Крюков В.А. Автоматическое распараллеливание Фортран-программ. Отображение на кластер. // Вестник Нижегородского университета им. Н.И. Лобачевского, 2009, № 2, С. 128–134.