РАСШИРЕНИЕ DVM-МОДЕЛИ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ ДЛЯ КЛАСТЕРОВ С ГЕТЕРОГЕННЫМИ УЗЛАМИ

В.А. Бахтин, М.С. Клинов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула, Ю.Л. Сазанов

В статье будут рассмотрены принципы расширения DVM-модели, принципы построения языка Fortran DVMH и компилятора, перечислены новые возможности языка и новые функции системы поддержки параллельного выполнения программ. Будут приведены экспериментальные данные об эффективности выполнения тестовых программ на графических процессорах кластера К-100.

1. Введение

Появление кластеров с гетерогенными узлами, использующих в качестве ускорителей графические процессоры (ГПУ), серьезно усложнило разработку программ, поскольку потребовало использовать, помимо низкоуровневых технологий MPI и SHMEM, еще и низкоуровневую технологию CUDA или OpenCL. На подходе новые процессоры с большим количеством ядер (например, 48-ядерный Intel SCC процессор, который стал недавно доступен для широкого круга исследователей [1]), для эффективного использования которых потребуются новые модели программирования.

Разработчики новых языков параллельного программирования явно не успевают отслеживать архитектурное разнообразие многоядерных процессоров. Очень маловероятно, что в течение 5-10 лет появится и широко распространится новый высокоуровневый язык. Поэтому в ближайшие годы программистам придется использовать гибридные языки, объединяющие разные модели параллельного программирования.

2. Использование гибридных моделей и языков

В настоящее время при разработке программ для высокопроизводительных вычислений на современных кластерах широко используются три модели программирования - MPI, OpenMP и CUDA. При этом пока вполне можно обходиться комбинацией MPI/OpenMP или MPI/CUDA, поскольку GPU используют для тех программ, для которых они значительно эффективнее, чем многоядерные процессоры, и, следовательно, неполной загрузкой многоядерных процессоров можно пренебречь. Если будут появляться программы, в которых только часть вычислений выгодно производить на GPU, а оставшиеся вычисления лучше оставить на универсальном многоядерном процессоре, то придется использовать и комбинацию из трех перечисленных моделей. Технически объединять низкоуровневые модели программирования, реализованные через библиотеки, проще, чем высокоуровневые модели, реализуемые посредством языков и соответствующих компиляторов. Но 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. Поэтому важно использовать высокоуровневые модели и языки программирования.

Среди высокоуровневых моделей программирования 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 [2], а для мультипроцессоров - OpenMP.

Аналогичные модели (HMPP [3], hiCUDA [4], PGI_APM [5]) разработаны в последние годы и для GPU. Все эти три модели имеют между собой много общего, что позволяет говорить о появлении нового подхода к программированию ускорителей типа GPU. Эти модели объединяет с моделями DVM и OpenMP также и то, что программист может полностью контролировать отображение данных и вычислений на аппаратуру. Поэтому, вполне естественно использовать предложенный подход для расширения моделей DVM и OpenMP для упрощения и ускорения разработки программ для кластеров с гетерогенными узлами.

Такие проекты расширения OpenMP уже появились - одни авторы [6] предлагают взять за основу модель HMPP, другие [7] - PGI_APM.

В данной работе описываются принципы расширения модели DVM. Расширение модели и реализация соответствующих гибридных языков и компиляторов с них не только упростит разработку программ для кластеров с гетерогенными узлами, но и ускорит создание для них автоматически распараллеливающих компиляторов. Эти языки будут использоваться в качестве некоторого промежуточного представления распараллеленной программы, на котором при необходимости программист сможет проводить дополнительную ручную оптимизацию программы.

3. Принципы расширения DVM-модели, построения языка Fortran DVMH и компилятора

При разработке гибридной модели DVM/OpenMP [8] были исследованы три подхода к объединению моделей DVM и OpenMP:

1. Расширение языка Fortran OpenMP директивами для распределения данных (DISTRIBUTE и ALIGN) и специальными клаузами для уточнения OpenMP-директив распределения вычислений (OpenMP + подмножество DVM).
2. Расширение языка Fortran DVM при помощи OpenMP-спецификаций, которые позволили бы автоматически преобразовывать существующие DVM-программы в программы, использующие MPI для взаимодействия между узлами и OpenMP для распределения вычислений внутри узла (DVM + подмножество OpenMP).
3. Полное объединение языков Fortran DVM и Fortran OpenMP (полностью DVM + полностью OpenMP). Наличие реальных приложений на языке Fortran DVM и на языке Fortran OpenMP говорило в пользу этого подхода, который и был реализован.

Преимущества выбранного подхода:

a) имеющиеся программы на языке Fortran DVM являются и программами на новом языке Fortran DVM/OpenMP;
b) имеющиеся программы на языке Fortran OpenMP являются и программами на новом языке Fortran DVM/OpenMP;
c) параллельная программа на новом языке Fortran DVM/OpenMP будет и параллельной программой на стандартном языке Fortran OpenMP;
d) параллельная программа на новом языке Fortran DVM/OpenMP будет и программой на уже достаточно отработанном языке Fortran DVM.

Если бы сейчас появился бы новый стандарт OpenMP с расширениями для использования ускорителей, то было бы естественно продолжить следование этому подходу. Но такого стандарта пока нет, модели HMPP и PGI_APM хотя и обретают популярность, но стандартами вряд ли станут в обозримое время. Появление таких кластеров с ГПУ, как К-100 (ИПМ им. М.В. Келдыша РАН) и "Ломоносов" (НИВЦ МГУ) остро поставило вопрос о расширении модели DVM - разработке модели DVMH (DVM for Heterogeneous systems).

С целью ускорения реализации расширения DVM-системы и упрощения адаптации DVM-программ для работы на кластерах с ГПУ оптимальным решением было бы добавить в язык Fortran DVM минимально необходимые дополнительные возможности. Однако необходимо учитывать следующие тенденции развития архитектуры ЭВМ:

- появление ускорителей разных типов, эффективность которых может сильно различаться для разных программ или разных фрагментов программ;
- расширение возможностей встраивания в ЭВМ все большего числа ускорителей разных типов при ограничении одновременной их работы из-за требований энергопотребления.

Учет этих тенденций требует от модели DVMH гибкости управления распределением вычислений и перемещением данных внутри узла.

Упомянутые выше модели HMPP, hiCUDA, PGI_APM предоставляют программисту следующие основные возможности для задания отображения программ на ускорители:

- определение фрагментов программы, которые следует выполнять на том или ином ускорителе, и требуемых им данных, которые должны быть расположены в памяти этого ускорителя;
- задание свойств цикла и правил отображения витков цикла на ускоритель;
- управление перемещением данных между оперативной памятью универсального процессора и памятью ускорителей.

Однако эти модели сильно различаются способами реализации этих возможностей и распределением работы между компиляторами и системами поддержки выполнения параллельных программ.

В модели DVMH предложены следующие методы реализации этих возможностей.

1. **Определение фрагментов программы, которые следует выполнять на том или ином ускорителе**

Такими фрагментами программ (называемых *вычислительными регионами*, или просто *регионами*) могут быть отдельные DVM-циклы или их последовательность. Для них можно задать перечень типов ускорителей и условия выбора каждого из них. Если для выполнения региона нет подходящего ускорителя, то он выполняется на универсальном процессоре.

2. **Определение требуемых регионам данных**

Для каждого региона указываются требуемые ему данные и вид их использования (входные, выходные, локальные).

3. **Задание свойств цикла и правил отображения витков цикла на ускоритель**

Для каждого DVM-цикла можно задать конфигурацию блока нитей (в терминологии CUDA).

4. **Управление перемещением данных между оперативной памятью универсального процессора и памятями ускорителей**

Перемещение данных осуществляется, в основном, автоматически в соответствии с запусками регионов на ускорителях и информацией об используемых ими данных. Однако, поскольку между выполнениями регионов могут выполняться на универсальном процессоре другие фрагменты программы, не специфицированные как регионы (а, следовательно, отсутствует информация об использовании ими данных), то имеются специальные средства для задания, какие данные им нужны и какие данные ими скорректированы.

Для реализации модели DVMH язык Fortran DVM расширен новыми директивами. Компилятор переводит программу на расширенном языке (Fortran DVMH) в программу на языке PGI Fortran CUDA. Новые директивы требуют от компилятора не только добавления в программу обращений к новым функциям системы поддержки DVM программ (Lib-DVM), но и преобразования параллельных циклов в соответствии с требованиями языка Fortran CUDA.

4. **Новые директивы языка Fortran DVMH**

Язык Fortran DVM расширен следующими директивами:

1. **Определение вычислительного региона**

!dvm$ region clause {, clause}
<structured block>
!dvm$ end region

Внутри вычислительного региона не может быть других вычислительных регионов.

В качестве клауз может быть задано:

a) **in**(subarray_or_scalar), **out**(subarray_or_scalar), **inout**(subarray_or_scalar), **local**(subarray_or_scalar)

Указание характеристики использования подмассивов и скаляров в регионе.

out - по выходу из региона значение переменной изменяется, причем это изменение может быть использовано далее,

in - при входе в регион нужны актуальные данные,

inout(subarray_or_scalar) - сокращенная запись двух клауз - **in** и **out**,

local - копирование данных между ЦПУ и ГПУ не требуется.

a) **targets**(target_name:condition {, target_name:condition})

Указание условий для вычислителей. При выборе вычислителя для региона будет сначала проанализирован список указанных вычислителей и в порядке указания будут вычислены условия. Первый, прошедший тест условия, будет выбран как исполнитель. Если ни один из вычислителей не прошел тест условия, регион будет выполнен на хосте (ЦПУ). Если клауза **targets** отсутствует, то выбор будет осуществляться из всех доступных вычислителей автоматически. Набор доступных на данный момент вычислителей: **HOST**, **CUDA**.

После выбора исполнителя региона автоматически будет определены и выполнены операции по запросу памяти для подмассивов и скаляров (у которых не было представителя на выбранном вычислителе) и операции по обновлению входных данных.

2. **Задание свойств цикла и правил отображения витков цикла на ускоритель**

!dvm$ do clause {, clause}
<DVM-loop nest>

В качестве клауз может быть задано:

a) а) **private**(array_or_scalar)
   Объявляет переменную приватной (локальной для каждого витка цикла).
b) б) **firstprivate**(array_or_scalar)
   Объявляет переменную, как локальную для каждого витка цикла, инициализированную значением, которое она имела перед параллельным циклом.
c) в) **cuda_block**(dim3)

Задает размер блока нитей для вычислителя CUDA. Может указываться целочисленное выражение - тогда блок полагается одномерным, может указываться два или три целочисленных выражения через запятую - соответственно блок будет иметь указанную размерность.

3. **Указание об актуализации данных на ЦПУ**

!dvm$ get_actual(subarray_or_scalar) делает все необходимые обновления для того, чтобы в памяти ЦПУ были актуальные данные в указанном подмассиве или скаляре (при этом, возможно, ничего и не будет перемещено).

4. **Подтверждение актуальности данных на ЦПУ**

!dvm$ set_actual(subarray_or_scalar) объявляет тот факт, что указанный подмассив или скаляр актуальную версию имеет в хост-памяти. При этом все другие представители указанного подмассива или

скаляра в памяти ускорителей автоматически устаревают и перед использованием будут обновлены.

5. **Регистрация данных, которые могут быть удалены из памяти ГПУ**

!dvm$ free(subarray_or_scalar)

5. Новые функции системы поддержки выполнения Lib-DVMH

Библиотека Lib-DVM расширена следующими функциями:

- определение состава ускорителей в момент запуска программы;
- регистрация региона и информации об используемых им данных;
- выделение памяти на ГПУ под данные, используемые в регионе, выполнение которого будет происходить на ГПУ;
- копирование входных данных региона из памяти ЦПУ в память ГПУ, если они там отсутствуют;
- запуск параллельных циклов региона на ГПУ. Если конфигурация блока нитей для цикла не задана в программе, то она определяется автоматически;
- копирование выходных данных региона из памяти ГПУ в память ЦПУ, если они там требуются для выполнения программ на ЦПУ;
- регистрация данных, которые могут быть удалены из памяти ГПУ;
- освобождение памяти на ГПУ в случае ее нехватки для размещения требуемых данных.

6. **Результаты экспериментов**

Продемонстрируем основные возможности языка Fortran DVMH на примере алгоритма Якоби.

```
      PROGRAM JAC_GPU
      PARAMETER (L=4096, ITMAX=1000)
      REAL A(L,L), EPS, MAXEPS, B(L,L)
!DVM$ DISTRIBUTE (BLOCK, BLOCK) :: A
!DVM$ ALIGN B(I,J) WITH A(I,J)
!     arrays A and B  with block distribution
      PRINT *,  '**********  TEST_JACOBI   **********'
      MAXEPS  =  0.5E - 7
!DVM$ REGION OUT(A), OUT(B), IN(L)
!DVM$   PARALLEL (J,I) ON A(I, J)
!       nest of two parallel loops, iteration (i,j) will be executed on
!       processor, which is owner of element A(i,j)
        DO J = 1, L
          DO I = 1, L
            A(I,  J) = 0.
            IF(I.EQ.1 .OR. J.EQ.1 .OR. I.EQ.L .OR. J.EQ.L) THEN
              B(I, J) = 0.
            ELSE
              B(I, J) = (1. + I + J)
            ENDIF
          ENDDO
        ENDDO
!DVM$ END REGION
      DO  IT  =  1,  ITMAX
!DVM$    REGION IN(A(2:L-1, 2:L-1), OUT(A), INOUT(B(2:L-1, 2:L-1)), OUT(EPS)
           EPS  =  0.
!DVM$      PARALLEL (J, I) ON A(I, J), REDUCTION (MAX(EPS))
!          variable EPS is used for calculation of maximum value
           DO J =  2, L-1
             DO I =  2, L-1
               EPS = MAX(EPS, ABS(B( I, J) - A( I, J)))
               A(I, J) = B(I, J)
             ENDDO
           ENDDO
!DVM$      PARALLEL (J, I) ON B(I, J), SHADOW_RENEW (A)
!          Copying shadow elements of array A from
!          neighbouring processors before loop execution
           DO J = 2, L-1
             DO I = 2, L-1
               B(I, J) =  (A( I-1, J) + A(I, J-1) + A(I+1, J) + A(I, J+1)) / 4
```

```
              ENDDO
            ENDDO
!DVM$   END REGION
!DVM$   GET_ACTUAL (EPS)
        PRINT 200, IT, EPS
200     FORMAT(' IT = ',I4, ' EPS = ', E14.7)
        IF (EPS .LT. MAXEPS) GO TO 3
      ENDDO
3     CONTINUE
!DVM$ GET_ACTUAL(B)
      OPEN (3, FILE='JAC.DAT', FORM='FORMATTED', STATUS='UNKNOWN')
      WRITE (3,*)   B
      CLOSE (3)
      END
```

В таблице 1 приводятся времена выполнения вариантов программы JAC_GPU на одном узле кластера К-100 (использовался компилятор PGI с опцией -O2).

В узле кластера К-100 имеются два 6-ядерных процессора Intel Xeon X5670 и 3 графических процессора NVIDIA Tesla C2050. Поскольку одно ядро выделено для специальных целей, то на узле можно запустить 11 MPI-процессов или 11 нитей OpenMP.

Программа на языке Fortran DVMH была запущена как последовательная (столбец Serial), как DVM-программа (столбец DVM) и как DVMH-программа (столбец DVMH). Кроме того, были созданы и пропущены еще три варианта программы - на языке PGI Fortran APM, на языке PGI Fortran CUDA и на языке Fortran OpenMP.

Таблица 1. Времена выполнения программы JAC_GPU (сек) на кластере К-100.

Число ядер	Serial	DVM	DVMH (32x16x1)	PGI_APM	Fortran CUDA (32x16x1)	OpenMP
1	46.22	51.13	7.68	13.68	5.52	46.04
2		25.70	4.18			23.15
3		18.65	3.12			19.62
4		15.29				13.84
8		10.55				11.70
11		12.26				11.01

7. **Заключение**

Появление кластеров с гетерогенными узлами, использующих в качестве ускорителей графические процессоры, остро поставило вопрос о соответствующем расширении модели DVM. При этом необходимо было учесть следующие тенденции развития архитектуры ЭВМ:

- появление ускорителей разных типов, эффективность которых может сильно различаться для разных программ или разных фрагментов программ;
- расширение возможностей встраивания в ЭВМ все большего числа ускорителей разных типов.

Были исследованы высокоуровневые модели HMPP, hiCUDA и PGI_APM, разработанные в последние годы для GPU, и предложена новая модель DVMH (DVM for Heterogeneous systems) для кластеров с гетерогенными узлами. Сформулированы расширения языка Fortran DVM и системы поддержки параллельного выполнения программ Lib-DVM.

Первая версия библиотеки Lib-DVMH уже разработана и проверена на небольших тестовых программах (включая и приведенную в главе 6), скомпилированных вручную. В настоящее время завершается реализация первой версии компилятора Fortran DVMH. Ее появление позволит проводить эксперименты с представительными тестовыми программами (например, тесты NAS) и реальными приложениями. Для успешного применения разработанного языка необходимо обеспечить удобные средства функциональной отладки и анализа эффективности выполнения программ на ГПУ.

Работа поддержана программами Президиума РАН №14, №15 и №17, грантами РФФИ №10-07-00211 и №11-01-00246, грантом Президента РФ МК-3324.2010.9.

ЛИТЕРАТУРА:

1. Timothy G. Mattson, Rob F. Van der Wijngaart, and other. "The 48-core SCC processor: the programmer's View"/ Proceedings of the 2010 ACM/IEEE conference on Supercomputing, SC10, New Orleans, Louisiana, Nov. 2010

2. Konovalov N.A., Krukov V.A., Mihailov S.N. and Pogrebtsov A.A. Fortran DVM - a Language for Portable Parallel Program Development. Proceedings of Software For Multiprocessors & Supercomputers: Theory, Practice, Experience. Institute for System Programming, RAS, Moscow, 1994
3. Dolbeau, R., Bihan, S., Bodin, F.: HMPP: A hybrid multi-core parallel programming environment. In: First Workshop on General Purpose Processing on Graphics Processing Units (October 2007)
4. T.D. Han and T.S. Abdelrahman, "hiCUDA: a high-level directive-based language for GPU programming," in GPGPU-2, 2009, pp. 52-61.
5. WOLFE, M. Design and implementation of a high level programming model for GPUs. In Workshop on Exploiting Parallelism using Hardware Assisted Methods (Seattle, Wa., Mar. 2009)
6. E. Ayguade, R.M. Badia, D. Cabrera, A. Duran, M. Gonzalez, F. Igual, D. Jimenez, J. Labarta, X. Martorell, R. Mayo, J.M. Perez, and E.S. Quintana-Ortiz, "A proposal to extend the openmp tasking model for heterogeneous architectures," in 5th International Workshop on OpenMP, IWOMP 2009.
7. Alistair Hart, Harvey Richardson, Alan Gray, Karthee Sivalingham. Accelerator directives: a user's perspective. (http://www.cse.scitech.ac.uk/events/GPU_2010/12_Hart.paper.pdf)
8. В.А. Бахтин, Н.А. Коновалов, В.А. Крюков. Расширение языка OpenMP Fortran для распределенных систем. // Вопросы атомной науки и техники. сер. Математическое моделирование физических процессов. 2002 г. Вып.4. стр.65-70