ПРОГРАММИРОВАНИЕ

УДК 004.432

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 рассматриваются принципы расширения DVM-модели и построения языка Fortran DVMH для кластеров с гетерогенными узлами. Новые возможности языка позволяют: определить фрагменты программы, которые следует выполнять на том или ином ускорителе; определить данные, необходимые для выполнения заданного фрагмента программы; задать правила отображения витков цикла на ускоритель; управлять перемещением данных между оперативной памятью универсального процессора и памятью ускорителей. Описываются принципы построения компилятора с языка Fortran DVMH и новые функции системы поддержки параллельного выполнения программ Lib-DVMH. Приводятся экспериментальные данные об эффективности выполнения тестовых программ на графических процессорах кластера K-100. Появление компилятора с языка Fortran DVMH не только упростит разработку программ для кластеров с гетерогенными узлами, но и ускорит создание для таких кластеров автоматически распараллеливающего компилятора с языка Fortran, использующего язык Fortran DVMH в качестве выходного языка, на котором программист сможет проводить дополнительную ручную оптимизацию программы.

Ключевые слова: языки параллельного программирования, гибридные многопроцессорные вычислительные системы, графические ускорители.

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.

1. 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[2] и DVM [3], а для мультипроцессоров – OpenMP.

Аналогичные модели (HMPP [4], hiCUDA [5], PGI APM [6]) 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 [7] предлагают взять за основу модель HMPP, другие [8] – 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.

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

При разработке гибридной модели DVM/OpenMP [9] 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 програм-

мы, 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:

- имеющиеся программы на языке Fortran DVM являются и программами на новом языке Fortran DVM/OpenMP;
- имеющиеся программы на языке Fortran OpenMP являются и программами на новом языке Fortran DVM/OpenMP;
- параллельная программа на новом языке Fortran DVM/OpenMP будет и параллельной программой на стандартном языке Fortran OpenMP;
- 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 (называемыми *вычислительными регионами*, или просто *регионами*) могут быть отдельные DVM-циклы или их последовательность. Регион может быть исполнен на одном или сразу нескольких ускорителях и/или на хост-системе, при этом на хост-системе может быть исполнен любой регион, а на возможность использования каждого типа ускорителей могут накладываться свои дополнительные ограничения на содержание региона.

2. **Определение данных, необходимых региону**

   Для каждого региона указываются данные, необходимые для его исполнения, 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 памятью 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 (для них отсутствует информация об использовании 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 расширенной системы поддержки DVM программ (Lib-DVMH), но и преобразования параллельных циклов в соответствии с требованиями языка Fortran CUDA.

3. Новые директивы языка Fortran DVMH

Язык Fortran DVM расширен следующими директивами:

1. **Определение вычислительного региона**

   !DVM$ REGION *clause* {, *clause*}

   <structured block>

   !DVM$ END REGION

   Вложенные (статически или динамически) регионы не допускаются. В качестве клауз может быть задано:

a) **in**(*subarray_ or_ scalar* {, *subarray_ or_ scalar*}), **out**(*subarray_ or_ scalar* {, *subarray_ or_ scalar*}), **inout**(*subarray_ or_ scalar* {, *subarray_ or_ scalar*}), **local**(*subarray_ or_ scalar* {, *subarray_ or_ scalar*}), **inlocal**(*subarray_ or_ scalar* {, *subarray_ or_ scalar*})

Указание направления использования подмассивов и скаляров в регионе. **in** – по входу в регион нужны актуальные данные. **out** – по выходу из региона значение переменной изменяется, причем это изменение может быть использовано далее. **inout(a)** – сокращенная запись одновременно двух клауз: **in(a), out(a)**. **local** – по выходу из региона значение переменной изменяется, но это изменение не будет использовано далее. **inlocal(a)** – сокращенная запись одновременно двух клауз: **in(a), local(a)**. Если для переменной указано **in**, и не указано **out** или **local**, то считается, что она не меняется в процессе исполнения.

b) **targets**(*target_ name* {, *target_ name* }), где *target_ name* – это CUDA | HOST | OpenCL | OpenMP

Задание типов вычислителей (ускорителей, хост-системы) или соответствующих им технологий программирования, для которых компилятор Fortran DVMH должен сгенерировать программы данного региона. Действительное же выполнение региона может происходить только на доступных DVMH-программе ускорителях (или на хост-системе), количество и типы которых задаются при ее запуске с помощью переменных окружения.

Определение конкретных исполнителей региона (из числа доступных вычислителей, для которых были сгенерированы программы региона) производится во время выполнения программы автоматически.

После определения исполнителей региона на них создаются представители всех требуемых региону подмассивов и скаляров (если такие представители отсутствовали). Кроме того, обновляются представители входных данных региона на тех вычислителях, где эти представители были только что созданы или уже устарели.

c) **async** – указание возможности асинхронного исполнения региона. При запуске региона в любом режиме (синхронный, асинхронный) ожидание завершения ранее запущенного региона возникает, если клаузами **in, out, local, inout, inlocal** задается необходимость изменить данные, используемые этим (ранее запущенным) регионом или необходимость использовать (запись или чтение) данные, изменяемые этим (ранее запущенным) регионом (**out, inout, local, inlocal**).

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

!DVM$ PARALLEL *clause* {, *clause*}

<DVM-loop nest>

В качестве клауз кроме клауз DVM-цикла могут быть также заданы:

a) **private**(*array_ or_ scalar* {, *array_ or_ scalar*}) Объявляет переменную приватной (локальной для каждого витка цикла).

b) **CUDA_block**(*X* [, *Y* [, *Z*]]) Указание размера блока нитей для вычислителя CUDA. Может указываться одно, два или три целочисленных выражения через запятую – соответственно блок будет полагаться указанной размерности.

3. Указание об актуализации данных на ЦПУ

!DVM$ GET_ACTUAL[(*subarray_or_scalar* {, *subarray_or_scalar*})] делает все необходимые обновления для того, чтобы на хост-памяти были самые новые данные в указанном подмассиве или скаляре (при этом, возможно, ничего и не следует перемещать). В случае отсутствия параметров все имеющиеся новые данные с ускорителей переписываются в память хост-системы.

4. **Подтверждение актуальности данных на ЦПУ**

   !DVM$ ACTUAL[(*subarray_or_scalar* {, *subarray_or_scalar*})] объявляет тот факт, что указанный подмассив или скаляр самую новую версию имеет в хост-памяти. В случае отсутствия параметров все имеющиеся представители переменных в памяти ускорителей объявляются устаревшими.

4. 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;
- выполнение редукционных операций на ГПУ;
- копирование выходных данных региона из памяти ГПУ в память ЦПУ, если они там требуются для выполнения программ на ЦПУ;
- освобождение памяти на ГПУ в случае ее нехватки для размещения требуемых данных.

5. 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)
!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 приводятся времена выполнения вариантов программы JAC_GPU на одном узле кластера K-100 (использовался компилятор PGI с опцией -O2).

В узле кластера K-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

Времена выполнения программы JAC_GPU (сек) на кластере K-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

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 5). 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 рекомендована программным комитетом международной суперкомпьютерной конференции «Научный сервис в сети Интернет: экзафлопсное будущее», поддержана ФЦП «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007 – 2013 годы»ГК № 07.514.11.4030, программой Президиума РАН № 17 и грантом РФФИ № 11-01-00246.

Литература

1. The 48-core SCC processor: the programmer's View / T.G. Mattson, R.F. Van der Wijngaart, M. Riepen, T. Lehnig, P. Brett, W. Haas, P. Kennedy, J. Howard, S. Vangal, N. Borkar, G. Ruhl, S. Dighe // SC'10 Proceedings of the 2010 ACM/IEEE International Conference for HPC, Networking, Storage and Analysis. – Washington, DC, 2010. – P. 1 – 11.
2. High Performance Fortran Forum. High Performance Fortran Language Specification, version 2.0. – 1997. – URL: http://hpff.rice.edu/versions/hpf2/index.htm (дата обращения: 30.10.2011).
3. Fortran DVM – a Language for Portable Parallel Program Development / N.A. Konovalov, V.A. Krukov, S.N. Mihailov, A.A. Pogrebtsov // Proceedings of Software For Multiprocessors & Supercomputers: Theory, Practice, Experience. – Moscow, 1994.
4. Bodin, F. Heterogeneous multicore parallel programming for graphics processing units / F. Bodin, S. Bihan // Scientific Programming – Software Development for Multi-core Computing Systems. – 2009. – V. 17, № 4. – P. 325 – 336.
5. Abdelrahman, T.S. hiCUDA: a high-level directive-based language for GPU programming / T.S. Abdelrahman, T.D. Han // GPGPU-2 Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units. – NY., 2009. – P. 52 – 61.
6. Wolfe, M. Design and implementation of a high level programming model for GPUs / M. Wolfe // GPGPU'10 Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units. – N.Y., 2010. – P. 43 – 50.
7. A proposal to extend the openmp tasking model for heterogeneous architectures / E. Ayguade, R.M. Badia , D. Cabrera , A. Duran, M. Gonzalez, F. Igual, D. Jimenez, J. Labarta, X. Martorell, R. Mayo, J.M. Perez, E.S. Quintana-Orti // IWOMP'09 Proceedings of the 5th International Workshop on OpenMP: Evolving OpenMP in an Age of Extreme Parallelism. – B.: Springer-Verlag, 2009. – P. 154 – 167.
8. Accelerator directives: a user's perspective / A. Gray, A. Hart, H. Richardson, K. Sivalingham. – 2010. – URL: http://www.cse.scitech.ac.uk/events/GPU_2010/12_Hart.paper.pdf (дата обращения: 11.03.2012).
9. Бахтин, В.А. Расширение языка OpenMP Fortran для распределенных систем / В.А. Бахтин, Н.А. Коновалов, В.А. Крюков // Вопросы атомной науки и техники. Сер. «Математическое моделирование физических процессов». – 2002. – Вып. 4. – С. 65 – 70.

Владимир Александрович Бахтин, кандидат физико-математических наук, Институт прикладной математики им. М.В. Келдыша РАН (г. Москва, Российская Федерация), bakhtin@keldysh.ru.

Максим Сергеевич Клинов, кандидат физико-математических наук, Институт прикладной математики им. М.В. Келдыша РАН (г. Москва, Российская Федерация), klinov@keldysh.ru.

Виктор Алексеевич Крюков, доктор физико-математических наук, профессор, Институт прикладной математики им. М.В. Келдыша РАН (г. Москва, Российская Федерация), krukov@keldysh.ru.

Наталия Викторовна Поддерюгина, кандидат физико-математических наук, Институт прикладной математики им. М.В. Келдыша РАН (г. Москва, Российская Федерация), konov@keldysh.ru.

Михаил Николаевич Притула, Институт прикладной математики им. М.В. Келдыша РАН (г. Москва, Российская Федерация), pritmick@yandex.ru.

Юрий Львович Сазанов, Институт прикладной математики им. М.В. Келдыша РАН (г. Москва, Российская Федерация), szn@keldysh.ru.

MSC 68N15

Extension of DVM Parallel Programming Model for Clusters with Heterogeneous Nodes

V.A. Bakhtin, Keldysh Institute of Applied Mathematics Russian Academy of Sciences (Moscow, Russian Federation),
M.S. Klinov, Keldysh Institute of Applied Mathematics Russian Academy of Sciences (Moscow, Russian Federation),
V.A. Krukov, Keldysh Institute of Applied Mathematics Russian Academy of Sciences (Moscow, Russian Federation),
N.V. Podderyugina, Keldysh Institute of Applied Mathematics Russian Academy of Sciences (Moscow, Russian Federation),
M.N. Pritula, Keldysh Institute of Applied Mathematics Russian Academy of Sciences (Moscow, Russian Federation),
Y.L. Sazanov, Keldysh Institute of Applied Mathematics Russian Academy of Sciences (Moscow, Russian Federation)

The principles of DVM model extension and the principles of Fortran DVMH language for clusters with heterogeneous nodes are presented. New language features allow you: to define program fragments to be executed on a particular accelerator, to define the data required for execution of a particular program fragment, to set the rules for mapping loop iterations to an accelerator, to control data movement between CPU memory and accelerators memory. We describe the principles of Fortran DVMH compiler implementation and new Lib-DVMH runtime system functions, and present efficiency characteristics for test programs on GPUs of K-100 cluster. The implementation of Fortran DVMH compiler will not only simplify programming for clusters with heterogeneous nodes, but also accelerate the implementation for such clusters automatically parallelizing compiler for Fortran programs that uses Fortran DVMH language as the output language on which the programmer can perform additional manual program optimization.

Keywords: parallel programming languages, hybrid multiprocessor computer systems, graphic accelerators.

References

1. Mattson T.G., Van der Wijngaart R.F., Riepen M., Lehnig T., Brett P., Haas W., Kennedy P., Howard J., Vangal S., Borkar N., Ruhl G., Dighe S. The 48-core SCC Processor: the Programmer's View. *SC'10 Proceedings of the 2010 ACM/IEEE International Conference for HPC, Networking, Storage and Analysis*. Washington, DC, 2010, pp. 1 – 11.
2. *High Performance Fortran Forum. High Performance Fortran Language Specification, Version 2.0* (1997). Available at: http://hpff.rice.edu/versions/hpf2/index.htm (accessed 30 October 2011).
3. Konovalov N.A., Krukov V.A., Mihailov S.N., Pogrebtsov A.A. Fortran DVM – a Language for Portable Parallel Program Development. *Proceedings of Software For Multiprocessors & Supercomputers: Theory, Practice, Experience*. Moscow, 1994.
4. Bodin F., Bihan S. Heterogeneous Multicore Parallel Programming for Graphics Processing Units. *Scientific Programming – Software Development for Multi-core Computing Systems*. 2009, vol. 17, no. 4, pp. 325 – 336.

5. Abdelrahman T.S., Han T.D. HiCUDA: a High-Level Directive-Based Language for GPU Programming. *GPGPU-2 Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*. NY, 2009, pp. 52 – 61.
6. Wolfe M. Design and Implementation of a High Level Programming Model for GPUs. *GPGPU'10 Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*. N.Y., 2010, pp. 43 – 50.
7. Ayguade E., Badia R.M., Cabrera D., Duran A., Gonzalez M., Igual F., Jimenez D., Labarta J., Martorell X., Mayo R., Perez J.M., Quintana-Orti E.S. A Proposal to Extend the Openmp Tasking Model for Heterogeneous Architectures. *IWOMP'09 Proceedings of the 5th International Workshop on OpenMP: Evolving OpenMP in an Age of Extreme Parallelism*. Berlin: Springer-Verlag, 2009, pp. 154 – 167.
8. Gray A., Hart A., Richardson H., Sivalingham K. *Accelerator directives: a user's perspective*. Available at: http://www.cse.scitech.ac.uk/events/GPU_2010/12_Hart.paper.pdf (accessed 11 March 2012).
9. Bakhtin V.A., Konovalov N.A., Krukov V.A. Extension of OpenMP Fortran Language for Distributed Systems [Rasshirenie jazyka OpenMP Fortran dlja raspredelennyh sistem]. *Voprosy atomnoj nauki i tehniki. Ser. Matematicheskoe modelirovanie fizicheskih processov [Problems of Atomic Science and Technology. Vol. Mathematical Modeling of Physical Processes]*. 2002, no. 4, pp. 65 – 70.

Поступила в редакцию 15 ноября 2011 г.