Использование языка Fortran DVMH для решения задач гидродинамики на высокопроизводительных гибридных вычислительных системах*

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

ФГБУН Институт прикладной математики им. М.В. Келдыша РАН

В 2011 году для новых гетерогенных и гибридных суперкомпьютерных систем в Институте прикладной математики им. М.В. Келдыша РАН была предложена модель DVMH (DVM for Heterogeneous systems), разработаны языки программирования высокого уровня, представляющие собой стандартные языки Фортран и Си, расширенные директивами отображения программы на параллельную машину, оформленными в виде специальных комментариев (или прагм). В статье анализируется эффективность разработанных на языке Fortran DVMH параллельных программ для решения задач гидродинамики «Каверна» и «Контейнер». Приводятся результаты расчетов при использовании нескольких тысяч ядер и более 1200 GPU-ускорителей.

1. Введение

Будущее высокопроизводительных компьютерных технологий неразрывно связано с массивным параллелизмом и с гетерогенностью. Создаются процессоры, содержащие всё большее количество ядер. Жесткие ограничения по энергопотреблению приводят к тому, что основные вычислительные мощности обеспечиваются многоядерными GPU-ускорителями достаточно специфичной архитектуры, адаптация программного обеспечения к которой - сложная наукоёмкая задача.

Разрыв между существующим программным обеспечением и возможностями новых суперкомпьютеров носит принципиальный характер и является существенной проблемой на пути эффективного использования современной вычислительной техники в научных исследованиях.

Разработанная в Институте прикладной математики им. М.В. Келдыша РАН высокоуровневая модель параллельного программирования - DVMH (DVM for Heterogeneous systems) существенно упрощает разработку параллельных программ для кластеров с гетерогенными узлами, использующих в качестве ускорителей графические процессоры.

2. Модель DVMH. Язык Fortran DVMH

При разработке модели DVMH за основу была взята модель DVM[1], в которую были добавлены следующие возможности:

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.

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). Если конфигурация блока нитей не задана в программе, то она определяется автоматически.

* Исследование выполнено при финансовой поддержке грантов РФФИ № 11-01-00246, 12-01-33003 мол_а_вед, 12-07-31204-мол_а и гранта Президента РФ НШ-4307.2012.9.

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. Для фрагментов программ, которые выполняются на универсальном процессоре (вне вычислительных регионов), имеются специальные средства для задания, какие данные с ускорителя им нужны и какие данные ими были скорректированы.

2.1 Организация вычислений, спецификации потоков данных

Вычислительный регион выделяет часть программы (с одним входом и одним выходом) для возможного выполнения на одном или нескольких вычислителях.

!DVM$ REGION [clause {, clause}]

<region inner>

!DVM$ END REGION

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.

Например, с использованием CUDA-устройства может быть исполнен любой регион без использования операций ввода/вывода, вызовов внешних процедур, рекурсивных вызовов.

Для управления тем, на каких вычислителях регион может исполняться, следует использовать клаузу TARGETS (см. ниже).

Вложенные (статически или динамически) регионы не допускаются.

DVM-массивы распределяются между выбранными вычислителями (с учетом их заданных весов и быстродействия вычислителей), нераспределенные данные размножаются. Витки вложенных в регион параллельных DVM-циклов делятся между выбранными для региона вычислителями в соответствии с правилом отображения параллельного цикла, заданного в директиве параллельного DVM-цикла. Количество и типы используемых каждым MPI-процессом ускорителей можно задать с помощью переменных окружения, а по умолчанию каждым процессом будут использованы все найденные поддерживаемые ускорители.

В качестве клауз может быть задано:

1) 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 - в регионе значение переменной изменяется, причем это изменение может быть использовано далее. INOUT(subarray_or_scalar {, subarray_or_scalar}) - сокращенная запись одновременно двух clause: IN(subarray_or_scalar {, subarray_or_scalar}), OUT(subarray_or_scalar {, subarray_or_scalar}). LOCAL - в регионе значение переменной изменяется, но это изменение не будет использовано далее. INLOCAL(a) - сокращенная запись одновременно двух clause: IN(a), LOCAL(a). Если для переменной указано IN, и не указано OUT или LOCAL, то считается, что в такую переменную в регионе вообще нет записей и она не меняется в процессе его исполнения.

После выбора набора исполнителей региона автоматически определяются и выполняются операции по выделению памяти для подмассивов и скаляров (если отсутствовал представитель или присутствовал не являющийся объемлющим), операции по обновлению входных данных (если не было актуального представителя). По выходу из региона обновления данных не происходят.

Указание всех используемых переменных в регионе не обязательно. При этом используемые, но не указанные в клаузах переменные включаются в регион в автоматическом режиме компилятором Fortran DVMH по правилам:

а) Все используемые массивы считаются используемыми полностью (не выделяются подмассивы);

б) Всякая переменная, которая используется на чтение получает атрибут IN;

в) Всякая переменная, которая используется на запись получает атрибут INOUT;

г) Всякая переменная, направление использования которой не поддаётся определению, получает атрибут INOUT;

д) атрибуты LOCAL и OUT в автоматическом режиме не проставляются.

2) TARGETS(target_name {, target_name})

где target_name это CUDA | HOST

Указание списка типов вычислителей, на которых предполагается исполнять регион. Такая клауза может быть только одна в директиве. Действительное исполнение региона будет происходить на всех используемых конкретным MPI-процессом вычислителях указанных в директиве типов, для которых регион был подготовлен, а если таковых нет, то на хост-системе. Количество и типы используемых каждым MPI-процессом ускорителей можно задать с помощью переменных окружения, а по умолчанию все вычислительные ресурсы каждого узла будут использованы процессами равномерно.

3) 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).

Управление не перейдет на следующий за синхронным регионом оператор, пока текущий регион не закончит исполнение. Управление может перейти на следующий за асинхронным регионом оператор, не дожидаясь его завершения (или даже его старта).

В полный цикл исполнения региона входит:

1) освобождение места для новых переменных на ускорителях (возможна автоматическая актуализация переменных на хосте),
2) выделение памяти для новых переменных на ускорителях,
3) закачка необходимых актуальных данных на вычислители,
4) исполнение исполняемых операторов на вычислителях.

<region inner> - это нуль или более следующих друг за другом конструкций:

1) Параллельный DVM-цикл

Параллельный DVM-цикл - важнейшая часть вычислительного региона.

!DVM$ PARALLEL clause {, clause}

<DVM-loop nest>

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

а) PRIVATE(array_or_scalar {, array_or_scalar})

Объявляет переменную приватной (локальной для каждого витка цикла), при этом ее объявление в объемлющем цикл регионе не обязательно (более того, если по-другому она не используется, то объявление ее в регионе излишне).

б) CUDA_BLOCK(X [, Y [, Z]])

Указание размера 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 полагаться указанной размерностью.

2) Последовательная группа операторов

Каждый оператор последовательной группы операторов исполняется на всех вычислителях, выбранных для исполнения региона, кроме случая модификации в нем распределенных данных - тогда действует правило собственных вычислений.

3) Хост-секция

!DVM$ HOSTSECTION

<hostsection inner>

!DVM$ END HOSTSECTION

Объявляет специального вида секцию исполнения на хосте.

<hostsection inner> - это часть программы с одним входом и одним выходом, которая будет исполняться на хост-системе. Всякие изменения переменных в этой секции могут быть потеряны. Такие секции предлагается использовать в отладочных целях для промежуточного контроля значений переменных по ходу исполнения региона. Операции вывода разрешены, вызовы внешних процедур разрешены.

2.2 Управление перемещением данных, актуальностью

Для фрагментов программ, которые выполняются на хосте (вне вычислительных регионов), управление перемещением данных между оперативной памятью универсального процессора и памятями ускорителей задается при помощи специальных директив актуализации:

!DVM$ GET_ACTUAL[(subarray_or_scalar {, subarray_or_scalar})]

делает все необходимые обновления для того, чтобы в хост-памяти были самые новые данные в указанном подмассиве или скаляре. В случае отсутствия параметров все имеющиеся новые данные с ускорителей переписываются в память хост-системы;

!DVM$ ACTUAL[(subarray_or_scalar {, subarray_or_scalar})]

объявляет тот факт, что указанный подмассив или скаляр самую новую версию имеет в хост-памяти. При этом пересекающиеся части всех других представителей указанных переменных автоматически устаревают и перед использованием будут (по необходимости) обновлены. В случае отсутствия параметров все имеющиеся представители переменных в памятях ускорителей объявляются устаревшими.

Использование директив ACTUAL и GET_ACTUAL без параметров не рекомендуется в силу повышения вероятности ошибок (ACTUAL), а также опасности излишних перемещений данных (GET_ACTUAL).

2.3 Компилятор с языка Fortran DVMH

Компилятор с языка Fortran DVMH преобразует исходную программу в параллельную программу на языке Fortran с вызовами функций системы поддержки времени выполнения (библиотека Lib-DVM). Кроме того, компилятор создает для каждой исходной программы еще два модуля: один - на языке C CUDA[2] и второй - на языке Fortran CUDA[3].

В частности, для параллельного цикла из региона компилятор генерирует функцию-обработчик на языке C CUDA и ядро для вычислений на GPU на языке Fortran CUDA, а также процедуру-обработчик на языке Fortran для выполнения на хост-машине. Обработчик - подпрограмма, осуществляющая обработку части параллельного цикла на конкретном устройстве. Она принимает в качестве аргументов описатель устройства и части параллельного цикла. Обработчик запрашивает порцию для исполнения (границы циклов и шаг), конфигурацию параллельной обработки (количество нитей), запрашивает инициализацию редукционных переменных, а после выполнения порции - передаёт результат частичной редукции в систему поддержки. В случае CUDA-обработчика, он для обработки частей цикла вызывает специальным образом сгенерированное ядро на языке Fortran CUDA. CUDA-ядро выполняется на GPU, производя вычисления, составляющие тело цикла.

По умолчанию, предполагается, что регион может исполняться на всех типах вычислителей и компилятор генерирует обработчики для хост-машины и CUDA-вычислителя. Пользователь может указать посредством клаузы TARGETS директивы REGION, на каких вычислителях предполагается исполнять регион. Согласно его указаниям компилятор генерирует тот или иной обработчик.

Для взаимодействия между узлами система поддержки использует библиотеку MPI.

Основная работа по реализации модели выполнения параллельной программы (например, распределение данных и вычислений) осуществляется динамически. Это позволяет обеспечить динамическую настройку DVMH-программ при запуске (без перекомпиляции) на конфигурацию параллельного компьютера (количество процессоров, ускорителей, их производительность и тип, а также латентность и пропускную способность коммуникационных каналов). Тем самым программист получает возможность иметь один вариант программы для выполнения на последовательных ЭВМ и параллельных ЭВМ различной конфигурации.

2.4 Пример программы Якоби на языке Fortan DVMH

Проиллюстрируем возможности языка Fortan DVMH на примере программы для алгоритма Якоби (см. **Рис. 1**).

```
      PROGRAM JAC
      PARAMETER (L=8, ITMAX=10)
      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
!DVM$ PARALLEL (J,I) ON A(I, J)
!     nest of two parallel loops, iteration (i,j) will
!     be executed on device, 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
        END DO
      END DO
!DVM$ END REGION
      DO IT = 1, ITMAX
         EPS = 0.
!DVM$    REGION
!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)
           END DO
         END DO
!DVM$    PARALLEL (J, I) ON B(I, J), SHADOW_RENEW (A)
         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
           END DO
         END DO
!DVM$    END REGION
!DVM$    GET_ACTUAL(EPS)
         PRINT 200,  IT, EPS
200      FORMAT(' IT = ',I4, '   EPS = ', E14.7)
         IF ( EPS . LT . MAXEPS ) EXIT
      END DO
!DVM$ GET_ACTUAL(B)
      OPEN (3, FILE='JAC.DAT', FORM='FORMATTED', STATUS='UNKNOWN')
      WRITE (3,*)   B
      CLOSE (3)
      END
```

Рис. 1. Программа Якоби на языке Fortan DVMH

В результате выполнения директивы
!DVM$ DISTRIBUTE (BLOCK, BLOCK) :: A

массив A будет распределен между вычислителями. Количество и тип используемых вычислителей задается при запуске программы с помощью переменных окружения и параметров командной строки.

Директива

!DVM$ ALIGN B(I,J) WITH A(I,J)

задает совместное распределение двух массивов A и B. Элементы массива B будут распределены на тот же вычислитель, где будут размещены соответствующие элементы массива A.

Директива

!DVM$ PARALLEL (J,I) ON A(I, J)

задает распределение вычислений. Витки цикла будут выполняться на том вычислителе, где распределены соответствующие элементы массива A.

Клауза REDUCTION (MAX(EPS)) организует эффективное выполнение редукционной операции - глобальной операции с расположенными на различных вычислителях данных (нахождение максимального значения).

Клауза SHADOW_RENEW (A) указывает на необходимость подкачки удаленных данных (теневых граней) с других вычислителей перед выполнением цикла.

Поскольку никакие дополнительные клаузы в директивах REGION не заданы, компилятор определяет направления использования переменных автоматически - INOUT(A,B,EPS).

При выполнении первого вычислительного региона (цикла инициализации) для распределенных частей массивов A и B на ускорителях будет выделена необходимая пямять.

При входе во второй вычислительный регион (в итерационном цикле) осуществляется проверка: присутствуют ли актуальные представители для массивов A и B на вычислителе? Поскольку такие представители уже присутствуют, то никакие дополнительные операции копирования актуальных данных на вычислители не выполняются.

При выходе из вычислительного региона обновление данных в памяти хоста не производится. Перед выводом массива B в файл, требуется скопировать последние изменения массива из памяти вычислителя при помощи директивы GET_ACTUAL(B).

С использованием языка Fortran DVMH были разработаны прикладные программы решения задач гидродинамики.

3. Разработка параллельных программ на языке Fortran DVMH для задач гидродинамики «Каверна» и «Контейнер»

3.1 Задача «Каверна»

Программа «Каверна» предназначена для моделирования циркуляционного течения в плоской квадратной каверне с движущейся верхней крышкой в двумерной постановке в широком диапазоне как параметров задачи, так и параметров численного метода.

Последовательная версия программы занимает 496 строк.

В ходе разработки параллельной программы для данной задачи были проведены следующие действия:

1) Добавлены 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
```

2) Вставлены директивы PARALLEL перед 28-ю гнездами циклов. Из них:
   а) 8 параллельных циклов имеют спецификацию PRIVATE;
   б) 2 цикла спецификацию REDUCTION;
   в) 7 циклов спецификацию SHADOW_RENEW.
3) Вставлены директивы начала и конца вычислительного региона в 7-ми местах программы.
4) Вставлены директивы объявления данных актуальными в 6-ти местах программы.
5) Вставлены директивы запроса актуальных данных в 5-ти местах программы.

6) Вставлена одна директива REMOTE_ACCESS для доступа к удаленным данным (данным, не расположенным на устройстве, которое должно выполнить оператор)

7) Для того чтобы виток цикла целиком мог выполняться на одном устройстве, в 4-х местах программы циклы были разбиты на два.

8) В 4-х местах программы сделаны тесно-гнездовые циклы.

9) Для 6-ти гнезд циклов изменен порядок вычисления витков циклов, что позволило обрабатывать элементы массивов согласно их расположению в памяти ЭВМ.

10) Устранены OUTPUT зависимости между витками 4-х циклов.

Таким образом, было изменено 45 строк (или 9% от количества строк последовательной программы), добавлено 117 строк (или 23,5% от количества строк последовательной программы), текст параллельной программы занимает 613 строк.

3.2 Задача «Контейнер»

Программа Контейнер предназначена 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 в широком диапазоне как параметров задачи, так и параметров численного метода. Последовательная версия программы занимает 828 строки.

В ходе разработки параллельной программы для данной задачи были проведены следующие действия.

1) Добавлены директивы распределения данных:

```
CDVM$ DISTRIBUTE ro(BLOCK,BLOCK,BLOCK)
CDVM$ ALIGN (i,j,k) WITH ro(i,j,k) :: ux, uy, uz, p, E
CDVM$ ALIGN (i,j,k) WITH ro(i,j,k) :: ro1, ux1, uy1,uz1, p1, E1
CDVM$ ALIGN (i,j,k) WITH ro(i,j,k) :: SFro, SFux, SFuy, SFuz, SFE
CDVM$ ALIGN (i,j,k) WITH ro(i,j,k) :: F1x, F2x, F1y, F2y, F1z, F2z
CDVM$ ALIGN (i,j,k) WITH ro(i,j,k) :: F3x, F3y, F3z
```

2) Вставлены директивы PARALLEL перед 21-м гнездом циклов. Из них:
   а) 9 параллельных циклов имеют спецификацию PRIVATE;
   б) 4 цикла спецификацию REDUCTION;
   в) 5 циклов спецификацию SHADOW_RENEW.
3) Вставлены директивы начала и конца вычислительного региона в 5-ти местах программы.
4) Вставлены директивы объявления данных актуальными в 4-х местах программы.
5) Вставлены директивы запроса актуальных данных в 3-х местах программы.
6) Вставлена одна директива REMOTE_ACCESS для доступа к удаленным данным.
7) Для того чтобы виток цикла целиком мог выполняться на одном устройстве, в 3-х местах программы циклы были разбиты на два.
8) В 6-ти местах программы сделаны тесно-гнездовые циклы.
9) Изменен порядок вычисления витков циклов для 12-ти гнезд циклов.
10) Устранены OUTPUT и FLOW зависимости между витками в 1 цикле.

Таким образом, при распараллеливании было изменено 37 строк (или 4,4% от количества строк последовательной программы), добавлено 114 строк (или 13,7% от количества строк последовательной программы), текст параллельной программы занимает 942 строки.

Для разработанных параллельных программ было проведено исследование эффективности.

4. Анализ эффективности разработанных на языке Fortran DVMH параллельных программ при запусках на большом числе узлов и GPU

В следующих подразделах приводятся времена выполнения программ (в секундах), которые были получены на суперкомпьютерном комплексе МГУ «Ломоносов»[5]. Для компиляции кода, выполняемого на хосте, использовались компиляторы Intel версии 13.0, для компиляции кода, выполняемого на ускорителях, использовался компилятор CUDA Fortran компании Port-

land Group версии 12.9 и NVIDIA CUDA C версии 4.0. Для взаимодействия между узлами использовалась библиотека Intel MPI версии 4.0.3.

4.1 Программа «Каверна»

Ускорение выполнения программы «Каверна» на одном GPU по сравнению с выполнением на 1 ядре центрального процессора в зависимости от размера сетки было опубликовано в [4].

В таблицах 1 и 2 приведены времена выполнения 200 итераций программы «Каверна» на сетке 3200x3200 на разном числе ядер и GPU.

Таблица 1. Время выполнения программы «Каверна» на сетке 3200x3200 на разном числе ядер

1	2	4	8	16	32	64	128	256	400	512	1024
1241,83	631,47	332,36	182,95	100,05	75,8	40,20	21,33	11,74	7,11	6,44	3,48

Таблица 2. Время выполнения программы «Каверна» на сетке 3200x3200 на разном числе GPU

1	2	4	8	16	32	64	128	256	400
73,07	39,34	19,94	11,65	7,17	4,80	3,96	3,45	3,32	3,19

При использовании 1024 ядер программа «Каверна» ускорилась в 357 раз по сравнению с выполнением на 1 ядре. При использовании 1 ускорителя программа ускоряется в 17 раз по сравнением с выполнением программы на 1 ядре. Максимальное ускорение, полученное с использованием ускорителей - 390 раз по сравнением с выполнением программы на 1 ядре.

4.2 Программа «Контейнер»

В таблицах 3 и 4 приведены времена выполнения программы «Контейнер» на разном числе ядер и GPU.

Таблица 3. Время выполнения программы «Контейнер» на разном числе ядер

Сетка, количество итераций	4	8	16	32	64	128	256	512	1024	2048
200x200x200 itmax=200	754,01	384,92	206,47	113,64	49,87	29,90	14,52	8,63	5,63	6,01
400x400x400 itmax=100	-	1202,32	630,15	317,36	164,02	85,68	43,10	22,53	13,54	7,66
800x800x800 itmax=50	-	-	-	-	576,16	318,75	151,78	79,68	41,26	21,91
1600x1600x1600 itmax=20	-	-	-	-	-	-	-	235,64	117,88	62,28

Таблица 4. Время выполнения программы «Контейнер» на разном числе GPU

Сетка, количество итераций	1	2	4	8	16	32	64	128	256	512	1024	1280
200x200x200 itmax=200	166,95	86,05	45,77	26,82	14,95	8,99	6,12	3,99	3,26	3,01	3,60	4,32
400x400x400 itmax=100			168,80	89,17	47,15	26,09	14,17	8,39	4,86	3,20	2,88	3,26
800x800x800 itmax=50						92,20	51,80	30,32	13,58	7,56	4,67	4,17
1600x1600x1600 itmax=20									37,38	20,14	10,74	8,95

Для сеток 200x200x200 и 400x400x400 при использовании большого числа графических процессоров задача перестает ускоряться и даже замедляется. Это связано с тем, что при увеличении числа используемых GPU существенно сокращается объем данных, обрабатываемых на одном GPU, что не позволяет полностью загрузить аппаратуру. Накладные расходы на подготовку и запуск вычислительных ядер, копирование теневых граней превышают эффект от распараллеливания программы.

Для сетки 200x200x200 при использовании 4 GPU программа ускоряется 16,47 раз по сравнению с выполнением на 4-х ядрах.

Для сетки 400x400x400 при использовании 8 GPU программа ускоряется 13,48 раз по сравнению с выполнением на 8-х ядрах.

Для сетки 800x800x800 при использовании 64 GPU программа ускоряется 11,12 раз по сравнению с выполнением на 64-х ядрах.

Для сетки 1600x1600x1600 при использовании 512 GPU программа ускоряется 11,7 раз по сравнению с выполнением на 512-х ядрах.

Одним из факторов при выборе задач «Каверна» и «Контейнер» для распараллеливания на языке Fortran DVMH было наличие у этих программ разработанных версий в модели SHMEM/CUDA. Данные об ускорении этих программ, полученные при использовании GPU, были опубликованы еще в 2010 году [6].

Было проведено сравнение эффективности параллельных программ в модели DVMH и модели SHMEM/CUDA. Для этого использовался следующий подход. Осуществлялся запуск исходной задачи на 1-м GPU (сетка 150x150x150), замерялось время ее выполнения. Затем в 2 раза увеличивалась сложность решаемой задачи (размер вычислительной сетки) и задача запускалась на 2 раза большем числе GPU и т.д. В таблицах 5 и 6 приведены времена выполнения 200 итераций SHMEM/CUDA и DVMH-версиий программы «Контейнер» на разном числе GPU.

Таблица 5. Время и эффективность выполнения SHMEM/CUDA-программы «Контейнер» на разном числе GPU

Число GPU	1	2	4	8	16	32	64	128	256	512	1024
время, с	87,12	87,824	88,8	89,296	90,216	90,992	91,496	91,576	91,976	92,468	92,74
Эффективность	100,0%	99,2%	98,1%	97,6%	96,6%	95,7%	95,2%	95,1%	94,7%	94,2%	93,9%

Таблица 6. Время и эффективность выполнения DVMH-программы «Контейнер» на разном числе GPU

Число GPU	1	2	4	8	16	32	64	128	256	512	1024
время, с	71,93	74,77	76,12	76,75	80,56	80,76	82,76	82,91	82,03	90,56	88
Эффективность	100,0%	96,2%	94,5%	93,7%	89,3%	89,1%	86,9%	86,8%	87,7%	79,4%	81,7%

Современные графические процессоры позволяют настраивать режим работы кэша L1 для каждого SM. По умолчанию 16 KB используется для L1, а 48 KB - для общей памяти. В системе поддержки выполнения DVMH-программ задан режим cudaDeviceSetCacheConfig(cudaFuncCachePreferL1), в котором 48 KB используется для кэша L1, а 16 KB - для общей памяти. Разработчики SHMEM/CUDA версии программы не учли эту возможность. В результате DVMH-программа выполняется на 1-ом GPU в 1,2 раза быстрее чем SHMEM/CUDA-программа.

При увеличении числа GPU эффективность DVMH-программы падает. Одна из причин – «лишние» обмены теневыми гранями. Обмен теневыми гранями – это достаточно дорогостоящая операция: необходимо скопировать требуемые теневые грани из памяти ускорителя в память хоста, запустить соответствующие обмены между узлами кластера, а затем скопировать полученные значения в память ускорителя. При определенных условиях можно обновить теневые грани за счет дополнительных вычислений. Такой механизм реализован для DVM-программ (SHADOW_COMPUTE). В настоящее время ведется доработка компилятора Fortran DVMH и системы поддержки выполнения программ для реализации такой возможности при использовании ускорителей.

5. Выводы

Появление новых гетерогенных и гибридных компьютерных архитектур, в частности, на основе многоядерных вычислительных ускорителей, позволило значительно повысить производительность суперкомпьютеров, что сделало актуальным разработку и оптимизацию прикладного программного обеспечения для соответствующих вычислительных систем.

Оценивая современное состояние методов разработки эффективных приложений для высокопроизводительных систем, следует отметить, что имеющиеся средства программирования

являются по своей сути низкоуровневыми и требуют значительных затрат от разработчика, без гарантии достижения требуемого уровня качества создаваемого прикладного обеспечения. Под качеством здесь в первую очередь понимается сокращение времени решения прикладных задач без потери точности их решения, а также простота сопровождения ПО и его переноса на новые архитектуры.

Разработанный в Институте прикладной математики им. М.В. Келдыша РАН подход к созданию прикладного программного обеспечения существенно упрощает создание прикладных программ для суперкомпьютерных систем с ускорителями. Язык Fortran DVMH обеспечивает высокий уровень переносимости прикладного ПО на системы с другими архитектурами графических процессоров, поскольку перенос не требует изменения программы.

Проведенное исследование характеристик разработанных приложений «Каверна» и «Контейнер» показало, что эффективность программ, разработанных в высокоуровневой гибридной модели DVMH, очень мало отличается от эффективности программ, написанных с использованием низкоуровневой технологии CUDA.

Литература

1. DVM-система [Электронный ресурс] - : [web-сайт] - Режим доступа: http://www.keldysh.ru/dvm. - 01.12.2012
2. CUDA [Электронный ресурс] – : [web-сайт] – Режим доступа: http://www.nvidia.ru/object/what_is_cuda_new_ru.html. – 01.12.2012.
3. CUDA Fortran. Programming Guide and Reference. Release 2012. [Электронный ресурс] – : [web-сайт] – Режим доступа: http://www.pgroup.com/lit/whitepapers/pgicudaforug.pdf. – 01.12.2012.
4. Бахтин В. А., Давыдов А. А., Крюков В. А., Четверушкин Б. Н., Шильников Е. В. Расширение DVM-модели параллельного программирования для кластеров с гетерогенными узлами. //ДОКЛАДЫ АКАДЕМИИ НАУК, 2011, том 441, № 6, С. 734–736.
5. Воеводин Вл.В., Жуматий С.А., Соболев С.И., Антонов А.С., Брызгалов П.А., Никитенко Д.А., Стефанов К.С., Воеводин Вад.В. Практика суперкомпьютера "Ломоносов" // Открытые системы. - Москва: Издательский дом "Открытые системы", 2012. – 7.
6. Давыдов А.А., Четверушкин Б.Н., Шильников Е.В. // Моделирование течений несжимаемой жидкости и слабосжимаемого газа на многоядерных гибридных вычислительных системах. Ж. Вычисл. матем. и матем. физ. – 2010. – Т. 50, № 12. – С. 2275–2284.