УДК 004.432

ОТОБРАЖЕНИЕ НА КЛАСТЕРЫ С ГРАФИЧЕСКИМИ ПРОЦЕССОРАМИ DVMH-ПРОГРАММ С РЕГУЛЯРНЫМИ ЗАВИСИМОСТЯМИ ПО ДАННЫМ[1]

В.А. Бахтин, А.С. Колганов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула

В 2011 г. 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 описываются проблемы и методы отображения циклов с зависимостями на графические процессоры, демонстрируется эффективность разработанных на языке Fortran DVMH параллельных программ с регулярными зависимостями по данным.

Ключевые слова: DVM for Heterogeneous systems, Fortran DVMH, гибридные системы с ускорителями, графические процессоры, CUDA.

Введение

В последнее время появляется много вычислительных кластеров с установленными в их узлах ускорителями. В основном, это графические процессоры компании NVIDIA. В 2012 г. начинают появляться кластеры с ускорителями другой архитектуры – Xeon Phi от компании Intel. Так, в списке Top500 [1] самых высокопроизводительных суперкомпьютеров мира, объявленном в ноябре 2012 г., 62 машины имеют в своем составе ускорители, из них 50 машин имеют ускорители NVIDIA, 7 – Intel, 3 – AMD/ATI, 2 – IBM. Данная тенденция заметно усложняет процесс программирования кластеров, так как требует освоения на достаточном уровне сразу нескольких моделей и языков программирования. Традиционным подходом можно назвать использование технологии MPI для разделения работы между узлами кластера, а затем технологий CUDA (или OpenCL) и OpenMP для загрузки всех ядер центрального и графического процессоров.

С целью упрощения программирования распределенных вычислительных систем были предложены высокоуровневые языки программирования, основанные на расширении директивами стандартных языков такие, как HPF [2], Fortran-DVM [3, 4], C-DVM [3, 5]. Также были предложены модели программирования и соответствующие основанные на директивах расширения языков для возможности использования ускорителей такие, как HMPP [6], PGI Accelerator Programming Model [7], OpenACC [8], hiCUDA [9].

Распараллеливание на GPU циклов без зависимостей, будь то ручное или с использованием высокоуровневых средств, обычно не вызывает больших идеологических трудностей, так как целевая массивно-параллельная архитектура хорошо подходит для их обработки. Циклы с зависимостями могут быть распараллелены с заметно большими трудностями, связанными в частности с моделью консистентности общей памяти, огра-

[1] Статья рекомендована к публикации программным комитетом Международной суперкомпьютерной конференции «Научный сервис в сети Интернет: все грани параллелизма – 2013».

ниченной поддержкой синхронизации потоков выполнения на GPU, большими потерями времени при невыравненном произвольном доступе к памяти.

Статья организована следующим образом. Разделы 1 и 2 статьи содержат описание способа исполнения циклов с регулярными зависимостями на GPU, примененном в языке Fortran-DVMH. Раздел 3 содержит наглядную демонстрацию новых возможностей языка Fortran-DVMH. Раздел 4 содержит описание применения языка Fortran-DVMH для распараллеливания нескольких тестов NAS на GPU. В заключении отмечены полученные результаты и направление для будущих исследований.

1. Язык Fortran-DVMH

В 2011 г. в Институте прикладной математики им. М.В. Келдыша РАН была расширена модель DVM для поддержки кластеров с ускорителями [10]. Это расширение названо DVMH и позволяет с небольшими изменениями перевести DVM-программу для кластера в DVMH-программу для кластера с ускорителями.

Язык Fortran-DVM был дополнен набором директив, обеспечивающих следующие возможности:

- задание вычислительных регионов — частей программы, для которых будет подготовлено исполнение на GPU;
- спецификация дополнительных свойств параллельных циклов;
- управление актуальным состоянием данных в памяти CPU.

Одним из важных аспектов функционирования такой программной модели, как DVMH, является вопрос отображения исходной программы на все уровни параллелизма и разнородные вычислительные устройства. Важными задачами механизма отображения является обеспечение корректного исполнения всех поддерживаемых языком конструкций на разнородных вычислительных устройствах, балансировка нагрузки между вычислительными устройствами, а также выбор оптимального способа исполнения каждого участка кода на том или ином устройстве.

В Fortran-DVM поддерживаются циклы с регулярными зависимостями по данным, для чего существует специальное указание ACROSS в директиве PARALLEL. Такие циклы могут корректно выполняться на кластере параллельно в режиме конвейера или в режиме обработки по гиперплоскостям.

В 2013 г. в язык Fortran-DVMH была добавлена поддержка работы циклов с регулярными зависимостями на GPU NVIDIA с использованием технологии программирования CUDA.

В модели DVM [3] имеется возможность спецификации циклов с регулярными зависимостями и их эффективного выполнения на кластерных архитектурах. При подключении возможности использовать графический ускоритель для таких циклов нужно решить следующий набор задач:

- подкачка данных между соседями, в том числе в режиме конвейера;
- эффективное отображение порции цикла с зависимостями на архитектуру CUDA;
- оптимизация обращений к глобальной памяти GPU.

В режиме конвейера часть цикла, попавшая на конкретный MPI-процесс, разбивается на части для того, чтобы соседние зависимые процессы как можно раньше начали

выполнять этот цикл. Так как стоит задача использовать GPU, то необходимо поддерживать такую подкачку во время обработки цикла.

Для циклов, имеющих более одного зависимого измерения, есть возможность не превращать эти измерения в последовательные, а использовать, например, метод гиперплоскостей для извлечения максимального параллелизма.

Так как обращения к глобальной памяти в случае невыравненного доступа очень медленные, то существует проблема эффективности выполнения циклов в случае изменения порядка обработки витков цикла, которая не решается простой перестановкой измерений массива или цикла на уровне исходного текста, как будет продемонстрировано на примере методов попеременных направлений и последовательной верхней релаксации. Данную проблему призван решать механизм динамического переупорядочивания массивов, реализованный в компиляторе Fortran-DVMH.

2. Алгоритм отображения циклов с зависимостями на GPU

Пусть есть программа, написанная на языке Fortran-DVMH, в которой присутствуют многомерные тесно-гнездовые циклы с регулярными зависимостями по данным. Один из известных алгоритмов, в котором есть зависимости по данным — SOR (Successive Over Relaxation) — метод последовательной верхней релаксации. Для простоты рассмотрим двумерный случай. Тогда для квадратной матрицы порядка N основной цикл выглядит так, как показано на рис. 1.

```
DO J = 2,N-1
  DO I = 2,N-1
    S=A(I,J)
    A(I,J)=(W/4)*(A(I-1,J)+A(I+1,J)+A(I,J-1)+A(I,J+1))+(1-W)*A(I,J)
    EPS=MAX(EPS,ABS(S-A(I,J))
  ENDDO
ENDDO
```

Рис. 1. Основной цикл метода последовательной верхней релаксации

Пространством витков данного цикла назовем множество кортежей всех принимаемых значений индексных переменных цикла. В рассматриваемом цикле есть прямая и обратная зависимость по измерению I и J, следовательно, его пространство витков не может быть отображено на блок нитей GPU, так как все нити исполняются независимо. Следовательно, нужен другой метод отображения.

Одним из известных методов отображения подобных циклов является метод гиперплоскостей. Все элементы, лежащие на гиперплоскости, могут быть вычислены независимо друг от друга. Будем вычислять все пространство витков за несколько итераций: на первой итерации будут вычислены элементы первой гиперплоскости, на второй — элементы второй гиперплоскости и т.д., пока не будут вычислены все элементы. Если применить данный метод к рассматриваемому циклу, то будут вычисляться диагонали, параллельные побочной. Всего будет выполнено порядка $2N$ итераций.

Обобщим предложенный алгоритм на многомерный случай. Пусть есть многомерный тесно-гнездовой цикл размерности k с регулярными зависимостями по данным по всем k измерениям. Тогда все элементы, лежащие на гиперплоскости ранга k-1 могут быть вычислены независимо. Если из k измерений q не имеют зависимости по данным, а p – имеют, где $k = p + q$, то применим предложенный алгоритм к гиперплоскостям ранга p, а остальные q измерений вычислим как независимые.

Как уже упоминалось выше, при таком порядке выполнения витков цикла может возникать проблема эффективного доступа к глобальной памяти в силу того, что параллельно обрабатываются не соседние элементы массивов, что приводит к значительной потери производительности.

Для оптимизации доступа к глобальной памяти в систему поддержки выполнения Fortran-DVMH программ был внедрен механизм динамического переупорядочивания массивов. Данный механизм перед каждым циклом использует информацию о взаимном выравнивании цикла и массива, которая уже имеется в DVM-программе для отображения на кластер и распределения вычислений. Он устанавливает соответствие измерений цикла и измерений массива, после чего переупорядочивает массив таким образом, чтобы при отображении на архитектуру CUDA доступ к элементам осуществлялся наилучшим образом — соседние нити блока работают с соседними ячейками памяти.

Данный механизм осуществляет любую необходимую перестановку измерений массива, а также так называемую подиагональную трансформацию, в результате которой соседние элементы на диагоналях (в плоскости необходимых двух измерений) располагаются в соседних ячейках памяти, что позволяет применять технику выполнения цикла с зависимостями по гиперплоскостям без значительной потери производительности на операциях доступа к глобальной памяти.

3. Примеры программ и характеристики их выполнения

Для иллюстрации описываемых новых возможностей Fortran-DVMH программ рассмотрим две простые программы — реализации метода попеременных направления и метода последовательной верхней релаксации.

Рассмотрим Fortran-DVMH программу для метода попеременных направлений, показанную на рис. 2.

Как видно, в программе есть циклы с зависимостями, причем в каждом из трех циклов зависимость только по одному измерению. Данная особенность позволяет выделить достаточный уровень параллелизма из них, но существенно то, что при любом порядке измерений массива один из этих циклов будет работать значительно медленнее (примерно в 10 раз) остальных вследствие невозможности параллельной обработки в этом цикле элементов массива, лежащих в соседних ячейках памяти. В такой ситуации является чрезвычайно полезным механизм динамического переупорядочивания массивов. Для такой программы достаточно переупорядочивать массив 1 раз на три цикла (одну итерацию метода). Данная программа без дополнительных изменений, скомпилированная Fortran-DVMH компилятором и запущенная на выполнение автоматически обнаружит необходимость такой перестановки и ускорит ее выполнение по сравнению с более традиционным подходом, при котором расположение массива задано жестко и не может изменяться во время работы программы. Была произведена серия запусков на кластере К-100, установленном в ИПМ им. М.В. Келдыша РАН, в котором в вычислительном узле установлены 2 CPU Intel Xeon X5670 и 3 GPU NVIDIA Tesla C2050. Результаты запусков отображены в табл. 1.

```
      PROGRAM ADI
      PARAMETER(NX=400,NY=400,NZ=400,MAXEPS=0.01,ITMAX=100)
      INTEGER NX,NY,NZ,ITMAX
      DOUBLE PRECISION EPS,RELAX,A(NX,NY,NZ)
!DVM$ DISTRIBUTE(BLOCK,BLOCK,BLOCK) :: A
      CALL INIT(A,NX,NY,NZ)
      DO IT = 1,ITMAX
       EPS=0.D0
!DVM$  ACTUAL(EPS)
!DVM$  REGION
!DVM$  PARALLEL (K,J,I) ON A(I,J,K),
!DVM$*  ACROSS (A(1:1,0:0,0:0))
       DO K = 2,NZ-1
        DO J = 2,NY-1
         DO I = 2,NX-1
          A(I,J,K) = (A(I-1,J,K) + A(I+1,J,K)) / 2
         ENDDO
        ENDDO
       ENDDO
!DVM$  PARALLEL (K,J,I) ON A(I,J,K),
!DVM$*  ACROSS (A(0:0,1:1,0:0))
       DO K = 2,NZ-1
        DO J = 2,NY-1
         DO I = 2,NX-1
          A(I,J,K) = (A(I,J-1,K) + A(I,J+1,K)) / 2
         ENDDO
        ENDDO
       ENDDO
!DVM$  PARALLEL (K,J,I) ON a(I,J,K),
!DVM$*  REDUCTION(MAX(EPS)),ACROSS(A(0:0,0:0,1:1))
       DO K = 2,NZ-1
        DO J = 2,NY-1
         DO I = 2,NX-1
          EPS = MAX(EPS, ABS(A(I,J,K) -
     *           (A(I,J,K-1)+A(I,J,K+1)) / 2))
          A(I,J,K) = (A(I,J,K-1)+A(I,J,K+1)) / 2
         ENDDO
        ENDDO
       ENDDO
!DVM$  END REGION
!DVM$  GET_ACTUAL(EPS)
       IF (EPS .LT. MAXEPS) GOTO 3
      ENDDO
3     CONTINUE
      END
```

Рис. 2. Реализация метода попеременных направлений на Fortran-DVMH

На малых размерах видно небольшое замедление в силу недостаточной загруженности GPU, а значит и проигрыш от медленного доступа к памяти не играет такой важной роли.

Рассмотрим Fortran-DVMH программу для метода последовательной верхней релаксации, показанную на рис. 3.

Основной цикл данной программы имеет зависимости по всем своим измерениям, что приводит к значительным трудностям при распараллеливании даже в модели OpenMP. DVM-система позволяет исполнять такие циклы на GPU и получать ускорение за счет двух изложенных выше техник. Стоит отдельно подчеркнуть, что эта программа

не только может эффективно исполняться на GPU, но также она может быть исполнена и на кластере с GPU без каких-либо дополнительных изменений.

Таблица 1

Время работы для метода попеременных направлений (100 итераций)

Линейный размер (nx=ny=nz)	CPU, 1 ядро	GPU, без переупорядочивания массива		GPU, с переупорядочиванием массива	
	Время, с	Время, с	Ускорение	Время, с	Ускорение
64	0,17	0,13	1,31	0,18	0,94
128	1,52	0,92	1,65	0,52	2,92
256	12,66	6,84	1,85	3,16	4,01
400	48,69	26,17	1,86	11,34	4,29

В табл. 2 приведены характеристики эффективности применения алгоритма отображения как с применением переупорядочивания массива, так и без.

```
      PROGRAM  SOR
      PARAMETER(N1=16000,N2=16000,ITMAX=100,MAXEPS=0.5E-6,W=0.5)
      REAL A(N1,N2), EPS, W
      INTEGER  ITMAX
!DVM$ DISTRIBUTE  A(BLOCK, BLOCK)
      CALL INIT(A, N1, N2)
      DO IT = 1,ITMAX
       EPS = 0.
!DVM$  ACTUAL(EPS)
!DVM$  REGION
!DVM$  PARALLEL (J, I) ON A(I, J), PRIVATE (S),
!DVM$*  REDUCTION(MAX(EPS)), ACROSS (A(1:1,1:1))
       DO J = 2,N2-1
        DO I = 2,N1-1
         S = A(I,J)
         A(I,J) = (W/4)*(A(I-1,J)+A(I+1,J)+
     *          A(I,J-1)+A(I,J+1)) + (1-W)*A(I,J)
         EPS = MAX(EPS, ABS(S - A(I,J)))
        ENDDO
       ENDDO
!DVM$  END REGION
!DVM$  GET_ACTUAL(EPS)
       IF (EPS .LT. MAXEPS) GOTO 4
      ENDDO
4     CONTINUE
      END
```

Рис. 3. Реализация метода последовательной верхней релаксации направлений на Fortran-DVMH

Таблица 2

Время работы для метода последовательной релаксации (100 итераций)

Линейный размер (*N1=N2*)	CPU, 1 ядро	GPU, без переупорядочивания массива		GPU, с переупорядочиванием массива	
	Время, с	Время, с	Ускорение	Время, с	Ускорение
2000	2,76	2,23	1,24	1,8	1,53
4000	11,10	5,6	1,98	3,9	2,85
8000	44,50	19,01	2,34	10,76	4,14
16000	178,02	80,24	2,22	32,91	5,41

4. Применение на тестах NPB: BT, SP, LU

В пакете NAS Parallel Benchmarks присутствуют три теста, в алгоритмах которых есть циклы с зависимостью по данным: BT (Block Tridiagonal), SP (Scalar Pentadiagonal) и LU (Lower - Upper). Эти тесты решают синтетическую задачу дифференциальных уравнений в частных производных (трехмерная система уравнений Навье–Стокса для сжимаемой жидкости или газа), используя блочную трехдиагональную схему с методом переменных направлений (BT), скалярную пятидиагональную схему (SP), и метод симметричной последовательной верхней релаксации (SSOR, алгоритм LU при помощи симметричного метода Гаусса–Зейделя).

В тесте BT всего 57 тесно-гнездовых циклов, которые можно вычислить параллельно. Из них 6 циклов имеют зависимость по одному из трех отображаемых измерений, причем зависимое измерение в различных циклах соответствует различным измерениям обрабатываемых массивов. В полученной Fortran-программе 3850 строк в фиксированном формате, 76 из которых — директивы DVMH. Для лучшего доступа к глобальной памяти GPU в исходном тексте была произведена следующая оптимизация — во всех массивах были переупорядочены измерения так, чтобы самое внутреннее измерение цикла соответствовало первому измерению массива, в соответствии с принятым в Fortran способом расположения массива в памяти.

В тесте SP всего 56 тесно-гнездовых циклов, которые можно вычислить параллельно. Из них 12 циклов имеют зависимость по одному из трех отображаемых измерений, причем зависимое измерение в различных циклах соответствует различным измерениям обрабатываемых массивов. В полученной Fortran-программе 3500 строк в фиксированном формате, 215 из которых — директивы DVMH. Никаких оптимизаций на уровне исходных текстов по сравнению с исходной последовательной программой не производилось.

Таблица 3

Эффективность распараллеливания тестов NPB: BT, SP, LU

Задача		CPU, 1 ядро	CPU, 12 ядер		Tesla C2050 (с ECC)		GeForce GTX 560 Ti (без ECC)	
Тест	Класс	Время, с	Время, с	Ускорение	Время, с	Ускорение	Время, с	Ускорение
BT	W	2,67	0,41	6,51	3,81	0,70	3,12	0,86
	A	67	10,7	6,26	27,9	2,40	19,7	3,40
	B	282,3	46	6,14	127	2,22	—	—
SP	W	8,45	1,3	6,50	3,83	2,21	3,02	2,80
	A	70	9,9	7,07	11,9	5,88	8,83	7,93
	B	290	40,8	7,11	45,5	6,37	30,6	9,48
	C	1246	159,5	7,81	161,5	7,72	—	—
LU	W	6,98	1,36	5,13	3,83	1,82	3,48	2,01
	A	48,1	10,02	4,80	7,43	6,47	6,6	7,29
	B	203	42	4,83	22,4	9,06	19,45	10,44
	C	819	163,9	5,00	73,8	11,10	59,51	13,76

В тесте LU всего 107 тесно-гнездовых циклов, которые можно вычислить параллельно. Из них 2 цикла имеют зависимость по трем отображаемым измерениям. В полученной Fortran-программе 4500 строк в фиксированном формате, 171 из которых — директивы DVMH. Для данного теста на уровне исходного текста была сделана такая же оптимизация, как и примененная в тесте BT. Также временные массивы, используемые в двух больших циклах с зависимостью, были удалены из цикла, а их выражения подставлены с целью уменьшения чтений из глобальной памяти GPU и сокращения объема занимаемой памяти.

Тестирование производилось на суперкомпьютере K100, имеющим процессоры Intel Xeon X5670 и GPU NVIDIA Tesla C2050 с включенным ECC и на локальной машине с процессором AMD Phenom II x4 и GPU NVIDIA GeForce GTX 560 Ti без ECC. Данные GPU имеют схожие характеристики, что позволит оценить влияние ECC на время вычисления. Последовательные версии программ были выполнены на суперкомпьютере K100. Также для сравнения были получены времена параллельных DVM программ, исполненных с использованием 12 процессорных ядер (один узел K100).

В табл. 3 и на рис. 4, рис. 5, рис. 6 приведены результаты тестирования производительности для этих тестов (для каждого варианта запуска бралась программа, показывающая лучший результат).

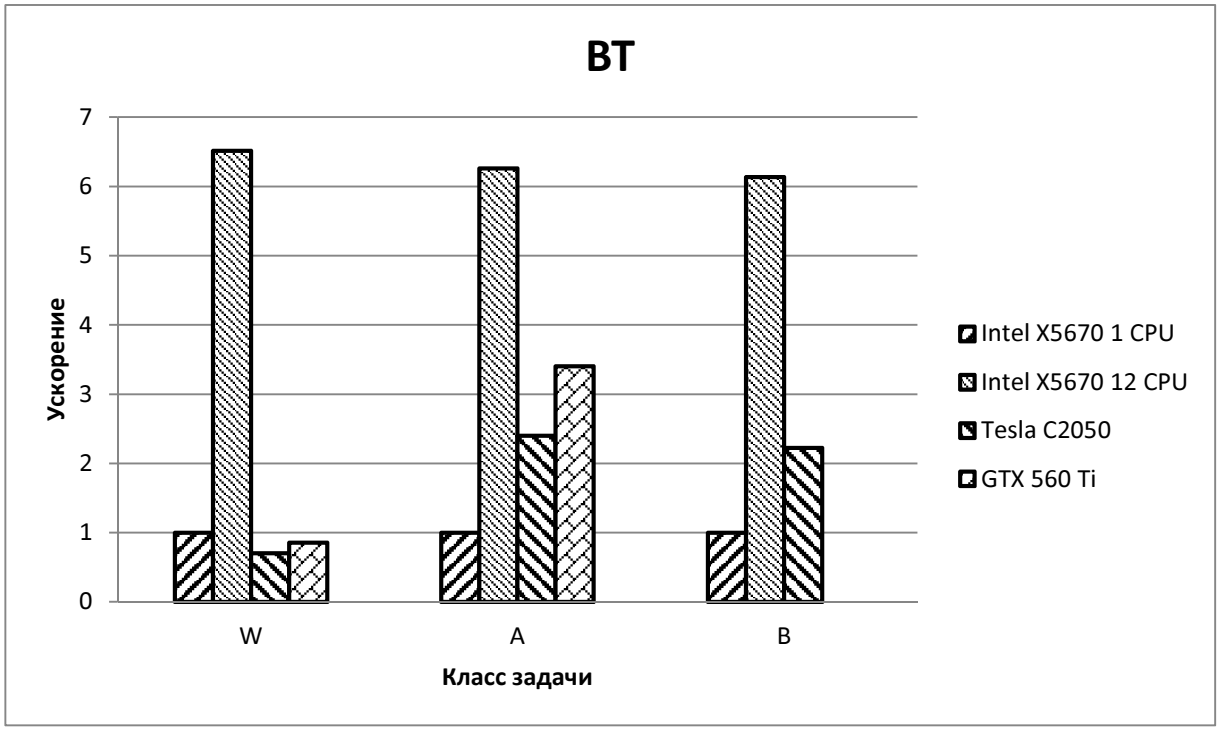

Рис. 4. Эффективность распараллеливания теста BT

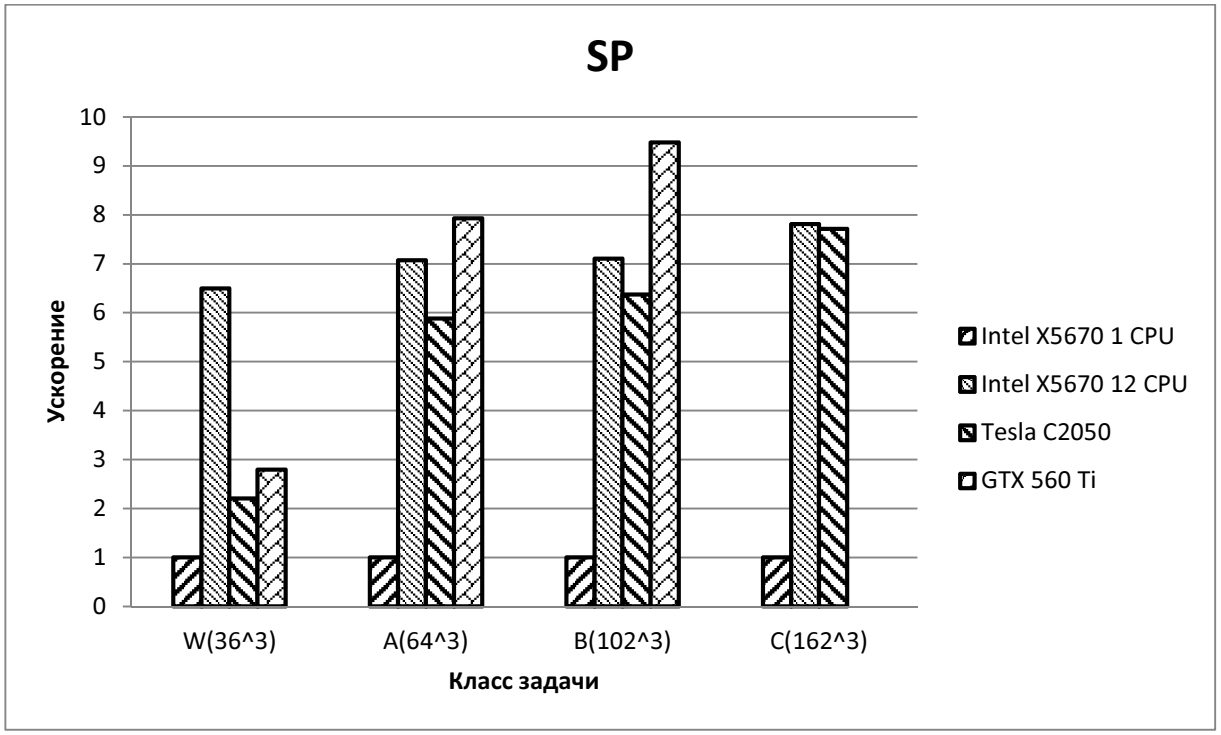

Рис. 5. Эффективность распараллеливания теста SP

Заключение

В статье рассмотрена проблема отображения циклов с зависимостями на графические процессоры. Предложен метод выполнения DVMH-программ с такими циклами на GPU, в том числе механизм автоматического переупорядочивания массивов для ускорения доступа к его элементам в памяти GPU. DVM-система и язык Fortran-DVMH были расширены поддержкой циклов с зависимостями на GPU. Проведена апробация на те-

стах NAS, которая показывает результаты, близкие к результатам оптимизированных вручную вариантов данных тестов, описанным в [11], а также заметный выигрыш в сравнении с [12].

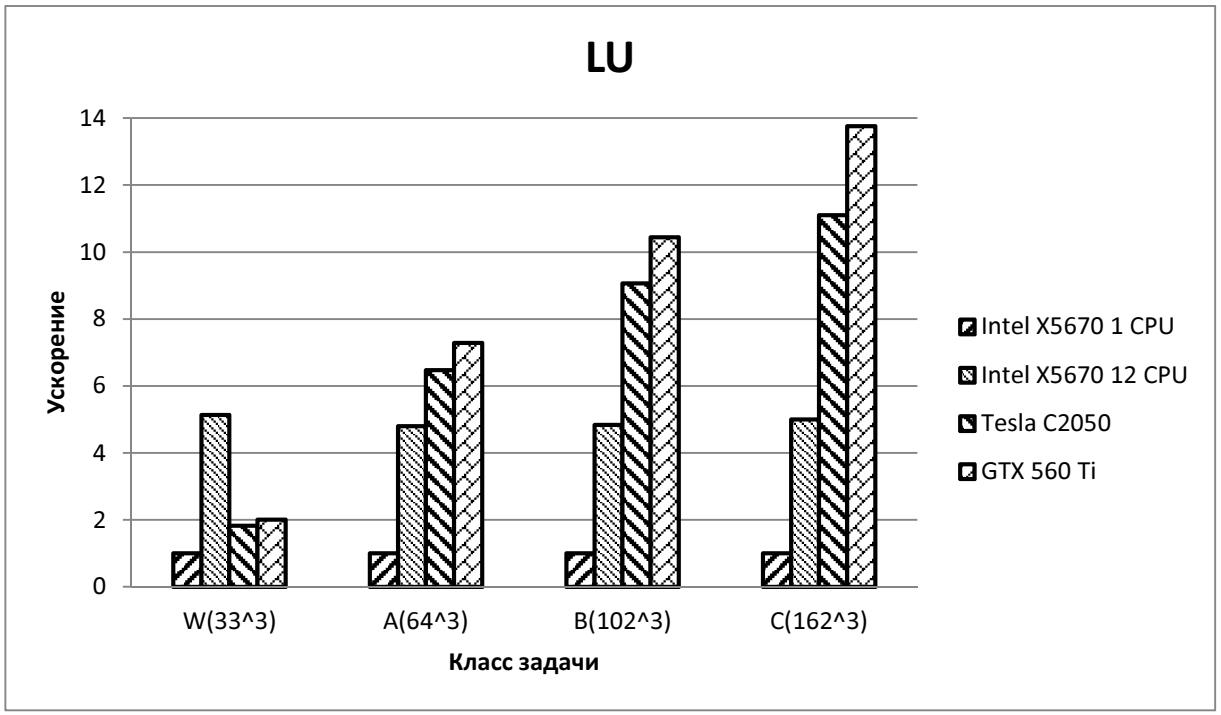

Рис. 6. Эффективность распараллеливания теста LU

В будущих исследованиях планируется повысить эффективность выполнения DVMH-программ с регулярными зависимостями по данным при использовании нескольких графических ускорителей.

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.

Литература

1. Top500 List – November 2012 TOP500 Supercomputer Sites. URL: http://top500.org/list/2012/11/ (дата обращения 01.12.2012).
2. High Performance Fortran. URL: http://hpff.rice.edu/ (дата обращения 01.12.2012).
3. Параллельное программирование в системе DVM. Языки Fortran-DVM и C-DVM / Н.А. Коновалов, В.А. Крюков, А.А. Погребцов и др. // Труды Международной конференции «Параллельные вычисления и задачи управления» (PACO'2001). — Москва, 2001. — С. 140–154.
4. Fortran DVM – язык разработки мобильных параллельных программ / Н.А. Коновалов, В.А. Крюков, С.Н. Михайлов, А.А. Погребцов // Программирование. — 1995. — № 1. — С. 49–54.
5. Коновалов, Н.А. C-DVM – язык разработки мобильных параллельных программ / Н.А. Коновалов, В.А. Крюков, Ю.Л. Сазанов // Программирование. — 1999. — № 1. — С. 54–65.
6. Dolbeau, R. HMPP™: A Hybrid Multi-core Parallel Programming Environment / R. Dolbeau, S. Bihan, F. Bodin. URL: http://www.caps-entreprise.com/wp-

content/uploads/2012/08/caps-hmpp-gpgpu-Boston-Workshop-Oct-2007.pdf (дата обращения 02.12.2012).
7. The Portland Group. PGI Accelerator Programming Model for Fortran & C. URL: http://www.pgroup.com/lit/whitepapers/pgi_accel_prog_model_1.3.pdf (дата обращения 02.12.2012).
8. OpenACC. URL: http://www.openacc-standard.org/ (дата обращения 01.12.2012).
9. Han, T.D. *hi*CUDA: High-Level GPGPU Programming / T.D. Han, T.S. Abdelrahman.// IEEE Transactions on Parallel and Distributed Systems. — 2011. — Vol. 22, No. 3 — P. 78–90.
10. Расширение DVM-модели параллельного программирования для кластеров с гетерогенными узлами / В.А. Бахтин, М.С. Клинов, В.А. Крюков и др. // Вестник Южно-Уральского государственного университета, серия «Математическое моделирование и программирование». — Челябинск: Издательский центр ЮУрГУ, 2012. — Вып. 12 — № 18 (277). — С. 82–92.
11. Pennycook, S.J. Performance Analysis of a Hybrid MPI/CUDA Implementation of the NAS-LU Benchmark / S.J. Pennycook, S.D. Hammond, S.A. Jarvis, G.R. Mudalige // ACM SIGMETRICS Performance Evaluation Review – Special issue on the 1st international workshop on performance modeling, benchmarking and simulation of high performance computing systems (PMBS 10). — 2011. — Vol. 38, Issue 4. — P. 23–29.
12. Seo, S. Performance Characterization of the NAS Parallel Benchmarks in OpenCL / S. Seo, G. Jo, J. Lee // 2011 IEEE International Symposium on. Workload Characterization (IISWC). — 2011. — P. 137–148.

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

Колганов Александр Сергеевич, Институт прикладной математики им. М.В. Келдыша РАН (г. Москва, Российская Федерация), alex-w900i@yandex.ru.

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

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

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

MAPPING DVMH-PROGRAMS WITH REGULAR DEPENDENCIES ONTO CLUSTERS WITH GPU

V.A. Bakhtin, Keldysh Institute of Applied Mathematics Russian Academy of Sciences (Moscow, Russian Federation),

A.S. Kolganov, 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),

In the 2011 year DVMH programming model for new heterogeneous and hybrid supercomputer systems (or DVM for Heterogeneous systems) was introduced in the Keldysh Institute for Applied Mathematics of RAS. The developed high-level programming languages were based on standard Fortran and C programming languages, but extended with the directives for mapping the program onto a parallel computer. The directives are represented as special comments (or pragmas). The paper describes problems and methods for mapping loops, which have dependencies to the GPU. Efficiency of the developed Fortran DVMH parallel programs with regular dependencies is demonstrated.

Keywords: DVM for Heterogeneous systems, Fortran DVMH, hybrid computational systems with accelerators, GPU, CUDA.

References

1. Top500 List – November 2012 TOP500 Supercomputer Sites. URL: http://top500.org/list/2012/11/ (accessed 01.12.2012).
2. High Performance Fortran. URL: http://hpff.rice.edu/ (accessed 01.12.2012).
3. Konovalov N.A., Krukov V.A., Pogrebtsov A.A., Podderyugina N.V., Sazanov Y.L. Parallel'noe programmirovanie v sisteme DVM. Yazyki Fortran-DVM i C-DVM [Parallel programming in the DVM system. Fortran-DVM and C-DVM languages]. Trudy Mezhdunarodnoy konferencii "Parallel'nye vychisleniya i zadachi upravleniya" (PACO'2001) [Proceedings of International conference "Parallel computations and control problems" (PACO'2001)]. Moscow, 2001. P. 140–154.
4. Konovalov N.A., Krukov V.A., Mihailov S.N., Pogrebtsov A.A. Fortran DVM – yazyk razrabotki mobil'nyh parallel'nyh programm [Fortran DVM – a language for mobile parallel programs development]. Programmirovanie [Programming]. 1995. No. 1. P. 49–54.
5. Konovalov N.A., Krukov V.A., Sazanov Y.L. C-DVM – yazyk razrabotki mobil'nyh parallel'nyh programm [C-DVM – a language for mobile parallel programs development]. Programmirovanie [Programming]. 1999. No. 1. P. 54–65.
6. Dolbeau R., Bihan S., Bodin F. HMPP™: A Hybrid Multi-core Parallel Programming Environment URL: http://www.caps-entreprise.com/wp-content/uploads/2012/08/caps-hmpp-gpgpu-Boston-Workshop-Oct-2007.pdf (accessed 02.12.2012).

7. The Portland Group. PGI Accelerator Programming Model for Fortran & C. URL: http://www.pgroup.com/lit/whitepapers/pgi_accel_prog_model_1.3.pdf (accessed 02.12.2012).
8. OpenACC. URL: http://www.openacc-standard.org/ (accessed 01.12.2012).
9. Han T.D., Abdelrahman T.S. *hi*CUDA: High-Level GPGPU Programming. IEEE Transactions on Parallel and Distributed Systems. 2011. Vol. 22, No. 3. P. 78–90.
10. Bakhtin V.A., Klinov M.S., Krukov V.A., Podderyugina N.V., Pritula M.N., Sazanov Y.L. Rasshirenie DVM-modeli parallel'nogo programmirovaniya dlya klasterov s geterogennymi uzlami [Extension of the DVM parallel programming model for clusters with heterogeneous nodes]. Vestnik Yuzhno-Ural'skogo gosudarstvennogo universiteta, seriya "Matematicheskoe modelirovanie i programmirovanie". Chelyabinsk, Publishing of the South Ural State University, 2012. No. 18 (277), Issue 12. P. 82–92.
11. Pennycook S.J., Hammond S.D., Jarvis S.A., Mudalige G.R. Performance Analysis of a Hybrid MPI/CUDA Implementation of the NAS-LU Benchmark. ACM SIGMETRICS Performance Evaluation Review – Special issue on the 1st international workshop on performance modeling, benchmarking and simulation of high performance computing systems (PMBS 10). 2011. Vol. 38, Issue 4. P. 23–29.
12. Seo S., Jo G., Lee J. Performance Characterization of the NAS Parallel Benchmarks in OpenCL. 2011 IEEE International Symposium on. Workload Characterization (IISWC). 2011. P. 137–148.

Поступила в редакцию 18 октября 2013 г.