ОТОБРАЖЕНИЕ НА ГРАФИЧЕСКИЙ ПРОЦЕССОР ЦИКЛОВ С ЗАВИСИМОСТЯМИ ПО ДАННЫМ В КОМПИЛЯТОРЕ С ЯЗЫКА FORTRAN-DVMH*

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

Проблемы программирования GPU

В последнее время появляется много вычислительных кластеров с установленными в их узлах ускорителями. В основном, это графические процессоры компании NVIDIA. В 2012 году 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 года, 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, ограниченной поддержкой синхронизации потоков выполнения на GPU, большими потерями времени при невыравненном произвольном доступе к памяти.

Язык Fortran-DVMH

В 2011 году 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:

- для задания вычислительных регионов — частей программы, для которых будет подготовлено исполнение на GPU;
- для спецификации дополнительных свойств параллельных циклов;
- для управления актуальным состоянием данных в памяти ЦПУ.

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 — или в режиме конвейера, или в режиме обработки по гиперплоскостям.

В Fortran-DVMH для циклов с указанием ACROSS не было эффективной реализации в случае работы на GPU.

Расширение возможностей языка Fortran-DVMH

В 2013 году 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 набор задач:

*Работа выполнена при финансовой поддержке гранта Президента РФ НШ-4307.2012.9; грантов РФФИ №11-01-00246, 12-01-33003-мол_а_вед, 12-07-31204-мол_а; программ фундаментальных исследований президиума РАН №15 и №16.

- 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 как можно раньше начали выполнять этот цикл соседние зависимые процессы. Так как стоит задача использовать GPU, то необходимо поддерживать такую подкачку во время обработки цикла.

Для циклов, у которых 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.

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 следующим образом (Листинг 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
```

Листинг 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 2 * N 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.

Механизм динамического переупорядочивания массивов

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.

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 (Листинг 2)

```
      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
```

Листинг 2. Реализация метода попеременных направлений на Fortran-DVMH

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 в следствие невозможности параллельной обработки в этом цикле элементов массива, лежащих в сосдених 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 её 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 ЦПУ Intel Xeon X5670 и 3 GPU NVIDIA Tesla C2050. Результаты запусков отображены в Таблице 1.

Таблица 1. Времена работы для метода попеременных направлений (100 итераций)

Линейный размер (nx=ny=nz)	ЦПУ, 1 ядро	GPU, без переупорядочивания массива		GPU, с переупорядочиванием массива	
	Время, секунд	Время, секунд	Ускорение	Время, секунд	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

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 (Листинг 3)

```
      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
```

Листинг 3. Реализация метода последовательной верхней релаксации на Fortran-DVMH

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.

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

Таблица 2. Времена работы для метода последовательной релаксации (100 итераций)

Линейный размер (N1=N2)	ЦПУ, 1 ядро	GPU, без переупорядочивания массива		GPU, с переупорядочиванием массива	
	Время, секунд	Время, секунд	Ускорение	Время, секунд	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

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 44 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 3450 строк в свободном формате, 70 из которых — директивы DVMH. Для лучшего доступа к глобальной памяти GPU в исходном тексте была произведена следующая оптимизация — во всех массивах первое измерение стало последним, чтобы улучшить чтение из глобальной памяти GPU, так как данное измерение не отображается ни на одно измерение циклов. Так же в 3х циклах в зависимостью временные массивы, используемые для инициализации, были распределены на регистры, тем самым уменьшив количество обращений к глобальной памяти GPU.

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 соответсвует 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.

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 2700 строк в свободном формате, 171 из которых — директивы DVMH. Для данного теста на уровне исходного текста была сделана такая же оптимизация изменения порядка измерений массивов, как и примененная в тесте BT. Также временные массивы, инициализируемые до процедуры SSOR, 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 рабочей станции с процессором Intel Core i7-3770 и GPU NVIDIA GeForce GTX TITAN без ECC. Данные GPU имеют различную архитектуру, что позволит оценить их применимость к подобным задачам. 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).

Ниже (Таблица 3; Рис. 1, 2, 3) приведены результаты тестирования производительности для этих тестов (для каждого варианта запуска бралась программа, показывающая лучший результат).

Таблица 3. Эффективность распараллеливания тестов NPB: BT, SP, LU

Задача		ЦПУ, 1 ядро	ЦПУ, 12 ядер		Tesla C2050 (с ECC)		GeForce GTX TITAN (без ECC)	
Тест	Класс	Время, секунд	Время, секунд	Ускорение	Время, секунд	Ускорение	Время, секунд	Ускорение
BT	W	2,67	0,41	6,51	2	1,34	1,61	1,66
	A	67	10,7	6,26	13,4	5,0	5,11	13,11
	B	282,3	46	6,14	59,32	4,76	18,6	15,18
	C	1217	220,5	5,52	202,8	6	63,53	19,16
SP	W	8,45	1,3	6,50	3,83	2,21	2,79	3,03
	A	70	9,9	7,07	11,2	6,25	5,69	12,3
	B	290	40,8	7,11	42,4	6,84	17,67	16,41
	C	1246	159,5	7,81	149,6	8,33	52,6	23,69
LU	W	6,98	1,36	5,13	3,4	2,05	3,24	2,15
	A	48,1	10,02	4,80	7	6,87	5,14	9,36
	B	203	42	4,83	21	9,67	13,23	15,34

	C	819	163,9	5,00	70,5	11,62	38,9	21,05

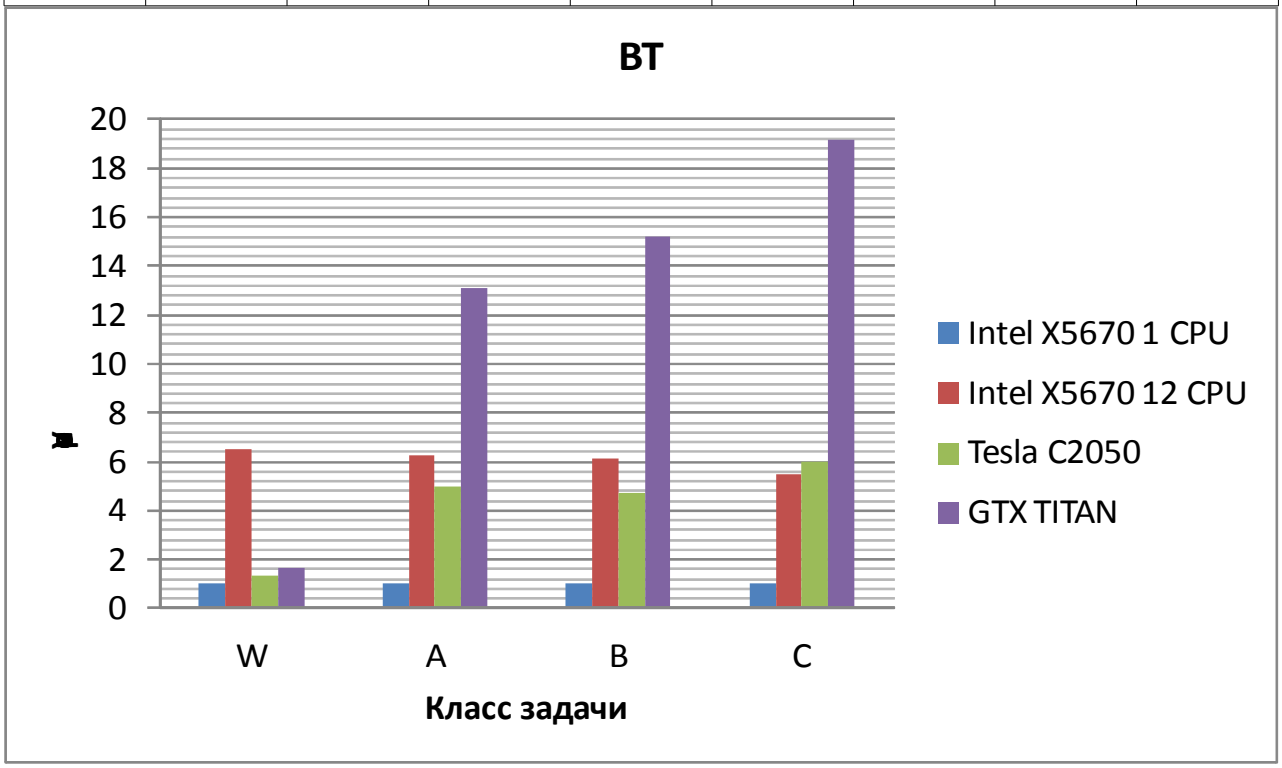

Рис. 1. Эффективность распараллеливания теста BT

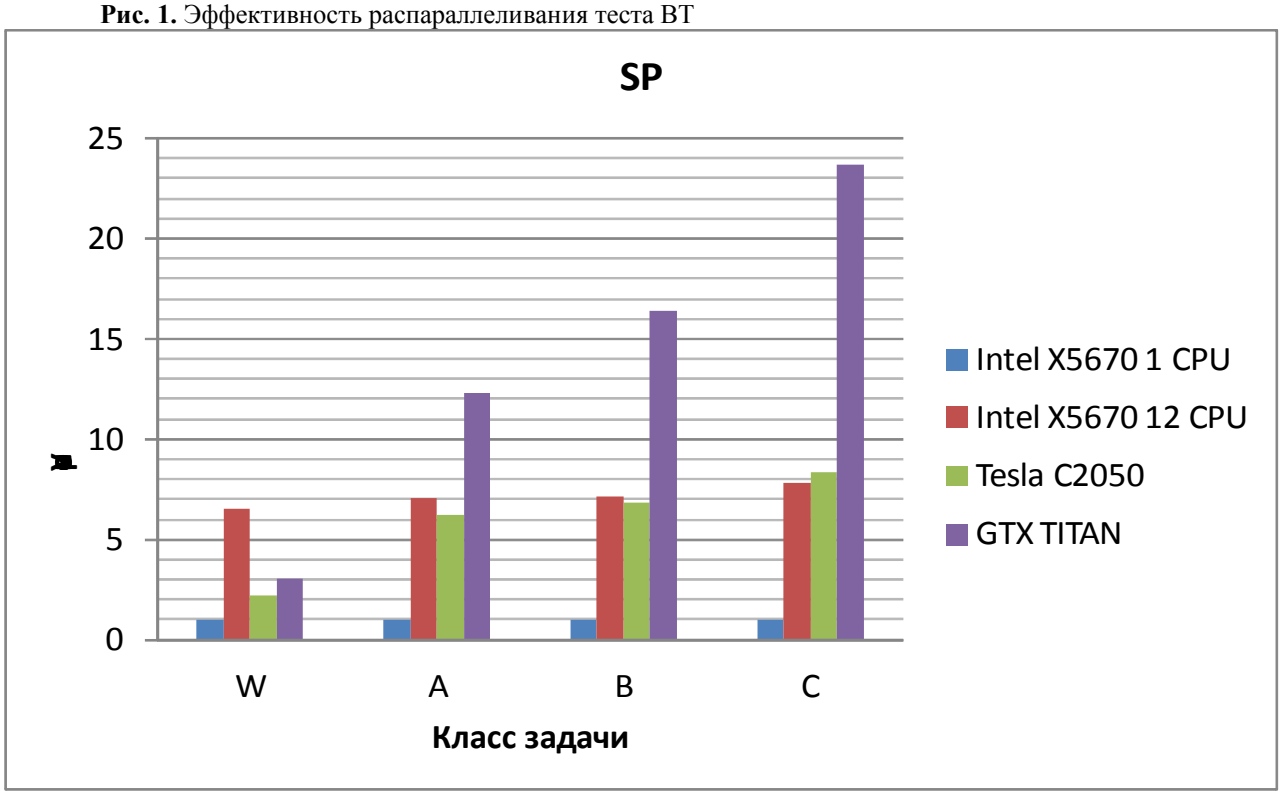

Рис. 2. Эффективность распараллеливания теста SP

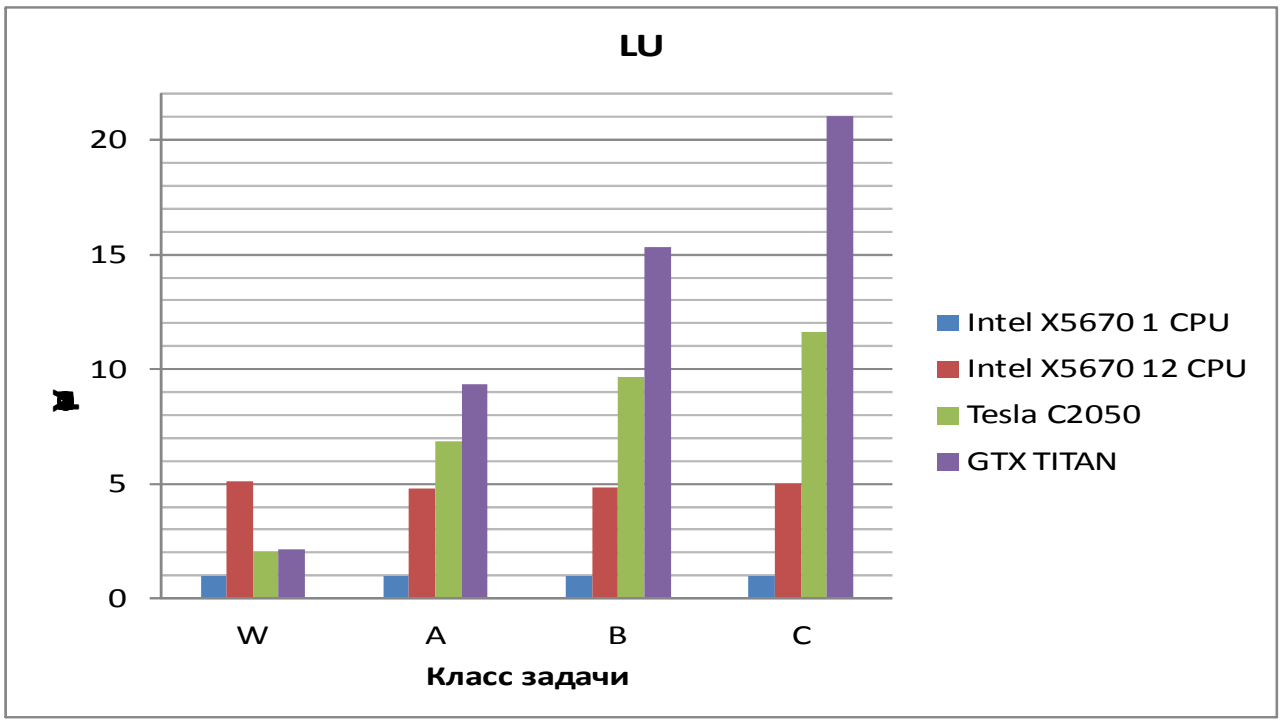

Рис. 3. Эффективность распараллеливания теста LU

Заключение

DVM-система и язык Fortran-DVMH были расширены поддержкой циклов с зависимостями на GPU, проведена апробация на тестах NAS, которая показывает результаты, близкие к результатам оптимизированных вручную вариантов данных тестов, описанным в [11], а также заметный выигрыш в сравнении с [12].

ЛИТЕРАТУРА:

1. Top500 List – November 2012 | TOP500 Supercomputer Sites [Электронный ресурс] – : [web-сайт] – Режим доступа: http://top500.org/list/2012/11/ – 01.12.2012
2. High Performance Fortran [Электронный ресурс] – : [web-сайт] – Режим доступа: http://hpff.rice.edu/ – 01.12.2012
3. Н.А. Коновалов, В.А. Крюков, А.А. Погребцов, Н.В. Поддерюгина, Ю.Л. Сазанов. Параллельное программирование в системе DVM. Языки Fortran-DVM и C-DVM. Труды Международной конференции "Параллельные вычисления и задачи управления" (PACO'2001) Москва, 2-4 октября 2001 г., 140-154 с.
4. Н.А. Коновалов, В.А. Крюков, С.Н. Михайлов, А.А. Погребцов. "Fortran DVM - язык разработки мобильных параллельных программ", "Программирование", № 1, 1995, стр 49-54.
5. Н.А. Коновалов, В.А. Крюков, Ю.Л. Сазанов. C-DVM - язык разработки мобильных параллельных программ. "Программирование", № 1, 1999, стр 54-65.
6. Romain Dolbeau, Stéphane Bihan, and François Bodin. 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 (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 [Электронный ресурс] – : [web-сайт] – Режим доступа: http://www.openacc-standard.org/ – 01.12.2012
9. T.D. Han and T.S. Abdelrahman. *hi*CUDA: High-Level GPGPU Programming. IEEE Transactions on Parallel and Distributed Systems, vol. 22, no. 1, pp. 78-90, Jan. 2011
10. В.А. Бахтин, М.С. Клинов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула, Ю.Л. Сазанов. Расширение DVM-модели параллельного программирования для кластеров с гетерогенными узлами. – Вестник Южно-Уральского государственного университета, серия "Математическое моделирование и программирование", №18 (277), выпуск 12 – Челябинск: Издательский центр ЮУрГУ, 2012, с. 82-92
11. S.J. Pennycook, S.D. Hammond, S.A. Jarvis, G.R. Mudalige. 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. Sangmin Seo, Gangwon Jo, Jaejin Lee. Performance Characterization of the NAS Parallel Benchmarks in OpenCL. — 2011 IEEE International Symposium on. Workload Characteriza-tion (IISWC). — 2011. — P. 137–148.