

УДК 004.65

РАСПАРАЛЛЕЛИВАНИЕ НА ГРАФИЧЕСКИЕ ПРОЦЕССОРЫ ТЕСТОВ NAS NPВ3.3.1 НА ЯЗЫКЕ FORTRAN-DVMH

В. Ф. АЛЕКСАХИН¹, В. А. БАХТИН², О. Ф. ЖУКОВА³, А. С. КОЛГАНОВ⁴, В. А. КРЮКОВ⁵,
Н. В. ПОДДЕРЮГИНА⁶, М. Н. ПРИТУЛА⁷, О. А. САВИЦКАЯ⁸, А. В. ШУБЕРТ⁹

¹valex@keldysh.ru, ²bakhtin@keldysh.ru, ³socol@spp.keldysh.ru, ⁴alexander.k.s@mail.ru, ⁵krukov@keldysh.ru,
⁶konov@keldysh.ru, ⁷pritmick@yandex.ru, ⁸savol@keldysh.ru, ⁹alexei@shubert.ru

¹⁻⁸ФГБУН «Институт прикладной математики им. М. В. Келдыша» РАН

⁹Московский государственный университет им. М. В. Ломоносова

Поступила в редакцию 17 февраля 2015 г.

Аннотация. Описываются преобразования последовательных версий тестов NAS из пакета NPВ3.3.1 (EP, MG, BT, LU, SP) и спецификации их параллельного выполнения посредством DVMH-директив, необходимые для их эффективной работы на кластерах с графическими процессорами. Исследуется влияние различных вариантов распараллеливания на эффективность выполнения программ. Сравниваются характеристики тестов, разработанных на высокоуровневом языке Fortran-DVMH (далее FDVMH), с их реализацией на низкоуровневом языке OpenCL, выполненной исследователями из Сеульского национального университета.

Ключевые слова: DVM; DVMH; CUDA; OpenCL; NASANPB 3.3.

ВВЕДЕНИЕ

В последнее время появляется много вычислительных кластеров с установленными в их узлах ускорителями. В основном, это графические процессоры компании NVIDIA. С 2012 года стали появляться кластеры с ускорителями другой архитектуры – XeonPhi от компании Intel. Так, в списке Top500 [1] самых высокопроизводительных суперкомпьютеров мира, объявленном в ноябре 2013 года, 53 машины имеют в своем составе ускорители, из них 39 машин имеют ускорители NVIDIA, 14 – Intel, 2 – AMD/ATI. 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 HRF [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 графических ускорителях (ГПУ) 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. При распараллеливании циклов с зависимостями возникают проблемы, обусловленные ограниченной поддержкой синхронизации потоков выполнения на ГПУ и моделью консистентности глобальной памяти.

ОБЗОР ПАКЕТА ТЕСТОВ NASA NPВ ВЕРСИИ 3.3

NAS Parallel Benchmarks [10] – комплекс тестов, позволяющий оценивать производительность суперкомпьютеров. Они разработаны и поддерживаются в NASA Advanced Supercomputing (NAS) Division (ранее NASA Numerical Aerodynamic Simulation Program), расположен-

Исследование выполнено при финансовой поддержке грантов РФФИ № 13-07-00580, 14-01-00109 и Программ фундаментальных исследований президиума РАН №№ 15, 16 и 18.

ном в NASA Ames Research Center. В версии 3.3 пакет NPВ включает в себя 11 тестов.

В статье рассматривается распараллеливание на ГПУ тестов EP, MG, BT, LU, SP, для которых ранее были разработаны параллельные версии для кластеров на высокоуровневом языке Fortran-DVM:

MG (Multi Grid) – множественная сетка. Тест вычисляет приближенное решение трехмерного уравнения Пуассона («трехмерная решетка») в частных производных на сетке $N \times N \times N$ с периодическими граничными условиями (функция на всей границе равна 0 за исключением заданных 20 точек). Размер сетки N определяется классом теста.

EP (Embarrassingly Parallel) – чрезвычайно параллельный. Тест вычисляет интеграл методом Монте-Карло – тест «усложненного параллелизма» для измерения первичной вычислительной производительности плавающей арифметики. Этот тест может быть полезен, если на кластере будут решаться задачи, связанные с применением метода Монте-Карло. В алгоритме также учитывается время на форматирование и вывод данных.

BT (Block Tridiagonal) – блочная трехдиагональная схема. Тест решает синтетическую систему нелинейных дифференциальных уравнений в частных производных (трехмерная система уравнений Навье–Стокса для сжимаемой жидкости или газа), используя блочную трехдиагональную схему с методом переменных направлений.

SP (Scalar Pentadiagonal) – скалярный пентадиагональный. Тест решает синтетическую систему нелинейных дифференциальных уравнений в частных производных (трехмерная система уравнений Навье–Стокса для сжимаемой жидкости или газа), используя скалярную пятидиагональную схему.

LU (Lower-Upper) – разложение при помощи симметричного метода Гаусса–Зейделя. Тест решает синтетическую систему нелинейных дифференциальных уравнений в частных производных (трехмерная система уравнений Навье–Стокса для сжимаемой жидкости или газа), используя метод симметричной последовательной верхней релаксации.

Разделим данные тесты на две группы: EP и MG – тесты, в которых не требуется распараллеливать циклы с регулярными зависимостями по данным (т. е. в FDVM-версиях этих тестов нет параллельных циклов со спецификациями ACROSS) и BT, SP, LU – тесты, в которых требуется распараллеливать циклы с регулярными зависимостями по данным (т. е. в FDVM-

версиях этих тестов есть параллельные циклы со спецификациями ACROSS).

Характеристики полученных тестов после преобразования, с использованием FDVMH:

В тесте EP всего 1 параллельный цикл.

В тесте MG всего 15 тесно-гнездовых параллельных циклов.

В тесте BT всего 44 тесно-гнездовых параллельных циклов. Из них 6 циклов имеют зависимость по одному из трех измерений, причем зависимое измерение в различных циклах соответствует различным измерениям обрабатываемых массивов.

В тесте SP всего 25 тесно-гнездовых параллельных циклов. Из них 6 циклов имеют зависимость по одному из трех измерений, причем зависимое измерение в различных циклах соответствует различным измерениям обрабатываемых массивов.

В тесте LU всего 107 тесно-гнездовых параллельных циклов. Из них 2 цикла имеют зависимость по всем трем измерениям.

РЕАЛИЗАЦИЯ И ОСОБЕННОСТИ ТЕСТОВ EP И MG

Тест EP

В тесте порождаются пары псевдослучайных нормально распределенных чисел и вычисляются частоты их попадания в каждый из десяти выбранных полуинтервалов $[k, k+1)$, где k меняется от 0 до 9. Тест содержит один единственный цикл, витки которого можно выполнять независимо друг от друга. При параллельном выполнении этого цикла на кластере никаких обменов между процессорами не требуется, а по окончании его выполнения производится объединение результатов редуцирующей операции суммирования, накопленных на каждом процессоре в массиве из 10 элементов. Для эффективного отображения на ГПУ в DVM-программе параллельный цикл объявлен вычислительным регионом.

Тест MG

Тест MG реализует алгоритм многосеточного метода решения задачи Пуассона. В тесте содержатся два вида циклов, в которых сосредоточена основная вычислительная нагрузка. К первому виду относятся циклы проецирования на более грубую сетку – при спуске по V-циклу и аппроксимирования на уточненную сетку, при восхождении по V-циклу, ко второму – циклы интерполяции решения на основании невязки.

```

!DVM$ PARALLEL (i3,i2,i1) ON u(i1,i2,i3)
do i3=2,n3-1)
do i2=2,n2-1
do i1=2,n1-1
    u(i1,i2,i3) = u(i1,i2,i3) + c(0) * r(i1,i2,i3)
>c(1) * r(i1+1,i2,i3) + r(i1,i2-1,i3) + r(i1,i2+1,i3) + r(i1,i2,i3-1) +
>r(i1,i2,i3+1)) + c(2) * (r(i1,i2-1,i3-1)+ r(i1,i2+1,i3-1) +
>r(i1,i2-1,i3+1) + r(i1,i2+1,i3+1) + r(i1-1,i2-1,i3) + r(i1-1,i2+1,i3) +
>r(i1-1,i2,i3-1) + r(i1-1,i2,i3+1) + r(i1+1,i2-1,i3) + r(i1+1,i2+1,i3)+
>r(i1+1,i2,i3-1) + r(i1+1,i2,i3+1))
enddo
enddo
enddo

```

Рис. 1. Цикл процедуры psinv до преобразований

При вычислении значений в одной сетке по значениям другой сетки требуется согласованное распределение этих сеток между процессорами. Для этого в FDVM-версии проблема решалась с помощью директивы REALIGN, которая позволяет перераспределить уже распределенный массив. Так, как переход от одной сетки к другой сетке происходит очень часто, то их перераспределение ведет к замедлению работы программы. Это замедление при работе на ГПУ оказалось очень значительным из-за низкой скорости обменов между памятью ЦПУ и памятью ГПУ (по сравнению со скоростью вычислений на ГПУ). Поэтому вFDVMH-версии было решено сократить количество операций REALIGN во время вычислений за счет использования дополнительной памяти для хранения нескольких экземпляров сетки с разным распределением ее элементов между процессорами. Для этого были использованы средства языка FDVMH для работы с динамическими массивами и указателями.

Оптимизация циклов теста MG

В тесте четыре основных вычислительных цикла. Каждый из них построен по одинаковому принципу. Поэтому рассмотрим один цикл и его оптимизацию. Директива PARALLEL позволяет сопоставить витки тесно-гнездовых циклов элементам распределенных массивов. В данном случае тесно-гнездовые циклы будут выполняться параллельно (рис. 1).

Не трудно заметить, что, например, для каждой фиксированной итерации по i_2 необходимы данные массива R по оставшимся не фиксированным индексам (i_1, i_3) и всем их комбинациям $(i_1 \pm 1, i_3 \pm 1)$. В итоге, получается, что каждая нить, исполняющая виток цикла по i_2 , будет считывать повторяющиеся данные соседних нитей по индексам i_1 и i_3 . Для повторного

использования данных внесем цикл по i_2 внутрь и изменим директиву следующим образом:

```
!DVM$ PARALLEL (i3, i1) ON u(i1, *, i3).
```

Для того чтобы повторно использовать значения в этом цикле, для каждой фиксированной пары (i_1, i_3) и $(i_1 \pm 1, i_3 \pm 1)$ заведем три переменные, объявленные как приватные – $r_1, r1_p1$ и $r1_m1$, которые будут соответствовать индексам $i_2, i_2 + 1, i_2 - 1$. Далее необходимо считать данные в $r1_m1$ и $r1$ до основного цикла, добавить считывание в $r1_p1$ в цикле до вычислений, заменить соответствующие выражения в цикле на считанные, а после вычислений – сохранить значения $r1$ и $r1_p1$ в переменные $r1_m1$ и $r1$ соответственно для следующей итерации. После всех преобразований, в цикле будет 5 групп (рис. 2).

Тем самым, количество необходимых чтений из глобальной памяти существенно сокращается за счет сохранения уже считанных значений в цикле i_2 при переходе с одной итерации на другую. На классе C время выполнения программы сократилось на 50 % по сравнению с программой без данных оптимизаций, однако время выполнения программы на ЦПУ стало значительно хуже.

РЕАЛИЗАЦИЯ И ОСОБЕННОСТИ ТЕСТОВ VT, SP, LU

Обзор типов зависимостей

Для лучшего представления характера зависимости по данным абстрагируемся от алгоритма каждого из тестов и рассмотрим типовые программы, а именно – метод переменных направлений, который используется в тестах VT и SP, и метод последовательной верхней релаксации (Successive over-relaxation – SOR), который используется в тесте LU.

```

!DVM$ PARALLEL (i3,i1) ON u(i1,*,i3)
!DVM$& ,PRIVATE (i2, r1,r1_m1,r1_p1, r2,r2_m1,r2_p1, r3,r3_m1,r3_p1,
!DVM$& r4,r4_m1,r4_p1, r5,r5_m1,r5_p1)
do i3=2,n3-1
do i1=2,n1-1
    r1_m1 = r(i1,1,i3) ! загрузкапервыхзначенийдоцикла
    r1 = r(i1,2,i3)
    . . . . .
do i2=2,n2-1
    r1_p1 = r(i1,i2+1,i3) ! загрузкаследующихзначений
r2_p1 = r(i1-1,i2+1,i3)
    . . . . .
u(i1,i2,i3) = u(i1,i2,i3)+
                c_0 * r1 + c_1 * (r2 + r3 + r1_m1 + r1_p1 + r4 + r5) +
                c_2 * (r4_m1 + r4_p1 + r5_m1 + r5_p1 + r2_m1 + r2_p1 +
r(i1-1,i2,i3-1) + r(i1-1,i2,i3+1) + r3_m1 + r3_p1 +
r(i1+1,i2,i3-1) + r(i1+1,i2,i3+1))

r1_m1 = r1 ! сохранение считанных значений
r1 = r1_p1
    . . . . .

enddo
enddo
enddo

```

Рис. 2. Цикл процедуры psinv после преобразований

```

programadi
parameter (nx=400,ny=400,nz=400,maxeps=0.01,itmax=100)
integernx,ny,nz,itmax
double precision eps,relax,a(nx,ny,nz)
callinit(a,nx,ny,nz)
do it = 1,itmax
eps=0.D0
do k = 2,nz-1
do j = 2,ny-1
doi = 2,nx-1
    a(i,j,k) = (a(i-1,j,k) + a(i+1,j,k)) / 2
enddo
enddo
enddo
do k = 2,nz-1
do j = 2,ny-1
doi = 2,nx-1
    a(i,j,k) = (a(i,j-1,k) + a(i,j+1,k)) / 2
enddo
enddo
enddo
do k = 2,nz-1
do j = 2,ny-1
doi = 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
enddo
end

```

Рис. 3. Реализация метода переменных направлений

Выше представлен пример программы для метода переменных направлений (рис. 3).

В программе три тесно-гнездовых цикла и в каждом из них есть зависимость по одному

из трех измерений. В первом гнезде циклов самый внутренний цикл может быть выполнен только последовательно, так как существует зависимость по данным массива А. В связи с этим – при выполнении данного цикла на ГПУ – время чтения из памяти будет большим, так как данные для параллельных витков оставшихся двух циклов будут расположены не подряд. Одним из способов решения данной проблемы является перестановка местами двух первых измерений массива А. Но тогда похожая проблема возникнет во втором гнезде циклов. Решить проблему можно аналогичным образом, например, поменяв местами первые два измерения обратно. Следовательно, нельзя подобрать начальное расположение массива таким, чтобы все три гнезда циклов выполнялись максимально эффективно.

Рассмотрим пример другой программы, реализующий метод последовательной верхней релаксации (рис. 4). В данной программе всего одно гнездо циклов, в котором происходят вычисления. В отличие от метода переменных направлений – в данном методе основной цикл имеет зависимость по всем своим измерениям, что приводит к значительным трудностям при распараллеливании даже в модели OpenMP.

Алгоритм отображения циклов с зависимостями в DVM-системе

Для отображения данных циклов в DVM-системе применяется следующий алгоритм [11]. Пространством витков данного цикла назовем множество кортежей всех принимаемых значений индексных переменных цикла.

В рассматриваемом цикле есть прямая и обратная зависимости по измерениям I, J и K, следовательно, его пространство витков не может быть отображено на блок нитей ГПУ, так как все нити исполняются независимо. Одним из известных методов отображения подобных циклов является метод гиперплоскостей. Все элементы, лежащие на гиперплоскости, могут быть вычислены независимо друг от друга.

При таком порядке выполнения витков цикла возникает проблема эффективного доступа к глобальной памяти в силу того, что параллельно обрабатываются не соседние элементы массивов, что приводит к значительной потере производительности (примерно в 10 раз). Причем в отличие от первой рассмотренной программы – задать первоначальное расположение массива будет еще сложнее, так как элементы вычисляются по гиперплоскостям. Наличие в программе циклов с зависимостями более чем по одному измерению и циклов без зависимостей усложняют задачу, так как для эффективного исполнения потребуется разное расположение данных для таких циклов.

Для того чтобы избавить программиста от выше описанных сложностей, в языке FDVMH существуют следующие возможности: поддержка циклов с зависимостями указанием спецификации ACROSS в директиве PARALLEL и динамическое переупорядочивание массивов.

Спецификация ACROSS

В основе реализации спецификации ACROSS – описанный выше метод гиперплоскостей.

```

program sor
parameter (n1=1000,n2=1000,n3=1000,itmax=100,
maxeps=0.5e-6,w=0.5)
real a(n1,n2,n3), eps, w
integer itmax
callinit(a, n1, n2, n3)
do it = 1,itmax
eps = 0.
do k = 2, n3-1
do j = 2, n2-1
doi = 2, n1-1
s = a(i, j, k)
a(i, j, k) = (w/4) * (a(i-1, j, k) + a(i+1, j, k) +
a(i, j-1, k) + a(i, j+1, k) + a(i, j, k-1) + a(i, j, k+1))
+ (1-w) * a(i, j, k)
eps = max(eps, abs(s - a(i, j, k)))
enddo
enddo
enddo
enddo
end

```

Рис. 4. Реализация метода последовательной верхней релаксации

Для того чтобы указать, что цикл имеет зависимость, необходимо добавить в спецификацию ACROSS список массивов, по которым есть зависимость (рис. 5).

Механизм динамического переупорядочивания массивов в DVMH

Для оптимизации доступа к глобальной памяти ГПУ в системе поддержки выполнения DVMH-программ был реализован механизм динамического переупорядочивания массивов, который перед выполнением на ГПУ параллельного цикла проверяет соответствие измерений цикла и измерений, используемых в нем массивов, и при необходимости переупорядочивает некоторые массивы таким образом, чтобы доступ к элементам осуществлялся наилучшим образом – соседние нити блока работали бы с соседними ячейками глобальной памяти.

Данный механизм осуществляет любую необходимую перестановку измерений массива, а также так называемую диагональную 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 ГПУ.

Преобразования исходных текстов программ

В тестах VT и SP используется метод переменных направлений. В тесте LU используется метод SSOR – симметричной последовательной верхней релаксации, состоящий из двух трехмерных циклов с положительными и отрицательными шагами с тремя зависимыми измерениями.

Для того чтобы распараллелить циклы с помощью FDVMH, необходимо преобразовать программу таким образом, чтобы циклы стали тесно-гнездовыми. Для этого, например, в LU, были подставлены процедуры bltsi butsi, чтобы получились трехмерные тесно-гнездовые циклы. В SP и VT были объединены циклы в процедуре compute_rhs для более эффективного выполнения. В процедурах x_solve, y_solve и z_solve, реализующих метод переменных направлений, были расширены временные массивы на одно измерение, чтобы трехмерные циклы можно было выполнять на ГПУ.

Оптимизации полученных программ

Программист, используя высокоуровневый язык FDVMH, оперирует с последовательной версией программы. Следовательно, необходимо уметь писать «хорошую» последовательную программу, чтобы из нее получалась эффективная параллельная программа. Знание конечной архитектуры может быть полезным при выполнении оптимизаций.

```
!пример для метода переменных направлений
!DVM$ PARALLEL (k,j,i) on A(i,j,k), ACROSS(A(1:1,0:0,0:0))
dok = 2,nz-1
do j = 2,ny-1
doi = 2,nx-1
    a(i,j,k) = (a(i-1,j,k) + a(i+1,j,k)) / 2
enddo
enddo
enddo
!пример для метода SOR
!DVM$ PARALLEL (K,J,I) on A(I,J,K), ACROSS(A(1:1,1:1,1:1))
!DVM$& ,PRIVATE(s), REDUCTION(MAX(eps))
do k = 2, n3-1
do j = 2, n2-1
doi = 2, n1-1
    s = a(i,j,k)
a(i,j,k) = (w/4)*(>a(i-1,j,k)+a(i+1,j,k)+>a(i,j-1,k)+a(i,j+1,k)+>a(i,j,k-1)+a(i,j,k+1)+>
    ) + (1-w)*a(i,j,k)
eps = max(eps, abs(s - a(i,j,k)))
enddo
enddo
enddo
```

Рис. 5. Расстановка спецификации ACROSS для двух методов

Современные ЦПУ имеют трехуровневый кэш. Размер кэша первого уровня равен 64Кб и содержится на всех вычислительных ядрах процессора. Размер кэша второго уровня варьируется от 1 до 2 Мб. Кэш третьего уровня является общим для всего ЦПУ и имеет размер порядка 12–15Мб.

Современные ГПУ имеют двухуровневый кэш. Размер кэша первого уровня равен 64Кб. Используется для разделяемой памяти и вытеснения регистров. Для разделяемой памяти доступно не более 48 Кб. Содержится в каждом вычислительном блоке. Максимальный размер кэша второго уровня составляет 1,5 Мб и является общим для всего ГПУ. Используется для кэширования данных, загружаемых из глобальной памяти ГПУ. В современном чипе ГПУ GK110 имеется 15 вычислительных блоков. Получается, что на один блок приходится примерно 48 Кб кэша первого уровня и 102 Кб кэша второго уровня. По сравнению с ЦПУ, это очень мало, поэтому операции чтения из глобальной памяти графического процессора дороже, чем из оперативной памяти центрального процессора. Все оптимизации будут направлены на уменьшение количества чтений из глобальной памяти и на увеличение вычислительной нагрузки, тем самым повышается соотношение количества вычислительных операций к операциям чтения из глобальной памяти ГПУ.

Тест LU

Основная вычислительная нагрузка сосредоточена в четырех циклах процедур `blts` и `bult`. В каждой из процедур сначала происходит инициализация четырех массивов типа `double`

`precision`, размер которых равен $250 \times \text{Class}^3$, где `Class` – размер класса рассматриваемой задачи (например, для класса `C` `Class = 162`). В итоге, для класса `C` необходимо около 3 Гб памяти. Так как после инициализации следующее гнездо циклов имеет зависимость по трем измерениям, то по описанному выше принципу работы DVMH-программы, произойдет диагонализация данных массивов. Так как массивы после инициализации используются только в процедурах `blts` и `bult`, выражения для инициализации элементов массивов были подставлены непосредственно в тело гнезда циклов с зависимостью, а сами массивы удалены, что привело к превышению значений данных массивов на каждом витке. Тем самым для выполнения данного теста на классе `C` требуется на 3 Гб меньше памяти, что позволило запустить эту программу на ГПУ Tesla C2050.

Еще одно преобразование, примененное в данном тесте, – использование приватных переменных для часто используемых элементов массивов, которые отображаются компилятором NVCC на регистры. Элемент массива часто используется, если суммарное количество записей и чтений из глобальной памяти в теле цикла будет больше количества чтений из глобальной памяти для его загрузки на регистры и записью в глобальную память для сохранения. Для пояснений рассмотрим следующий фрагмент программы (рис. 6).

Во фрагменте массив `U` используется на чтение и запись, причем по последнему измерению массив не распределен. Всего делается 15 записей и 45 чтений. А для загрузки на регистры необходимо 5 чтений и столько же потребуется для сохранения результата.

```

!DVM$ PARALLEL (k,j,i) on u(i,j,k,*), PRIVATE (m1,m2,m)
do k = 2,nz-1
do j = 2,ny-1
do i = 2,nx-1
      m1 = 2
      m2 = 3
do m = 1,5
      u(i,j,k,m) = u(i,j,k,m1) + u(i,j,k,m2)
enddo
do m = 1,5
      u(i,j,k,m) = u(i,j,k,m1+1) + u(i,j,k,m2+1)
enddo
do m = 1,5
      u(i,j,k,m) = u(i,j,k,m1-1) + u(i,j,k,m2-1)
enddo
enddo
enddo
enddo

```

Рис. 6. Исходный цикл

```

doi = 2, nx-1
do m=1,5
    u_(m) = u(i, j, k, m)
enddo
    m1 = 2
    m2 = 3
do m = 1,5
    u_(m) = u_(m1) + u_(m2)
enddo
do m = 1,5
    u_(m) = u_(m1+1) + u_(m2+1)
enddo
do m = 1,5
    u_(m) = u_(m1-1) + u_(m2-1)
enddo
do m=1,5
u(i, j, k, m) = u_(m)
enddo
enddo
enddo
enddo

```

Рис. 7. Применение оптимизации – замена массива u массивом u_

В итоге, в тексте программы необходимо добавить два цикла для загрузки и сохранения массива, а также заменить все обращения в теле цикла новой переменной (рис. 7).

Тест VT

В тесте VT, как было описано выше, самая большая вычислительная нагрузка содержится в трех процедурах – x_solve, y_solve, z_solve. Так как данные процедуры отличаются только зависимым измерением (x, y и z соответственно), то оптимизации, применяемые к одной из них, будут так же применены и ко всем остальным. Рассматривая процедуру x_solve, можно заметить, что используется большой временный массив lhs типа double precision, размер которого равен $75 \times \text{Class}^3$, где Class – класс задачи (например, для класса C Class = 162). Тем самым, данный массив на классе C занимает примерно 2,3 Гб памяти. Получаем те же самые проблемы, что и в тесте LU: нехватка памяти и долгое время динамического переупорядочивания. Решить проблему можно описанным выше способом – сокращением длины массива за счет повторной инициализации элементов массива непосредственно в теле цикла процедуры x_solve. Таким образом, в силу специфики задачи, вспомогательный массив можно сократить на одно измерение, а именно в число раз, соответствующее классу задачи. Для класса C массив сократится в 162 раза и будет занимать около 15 Мб.

Тест SP

Так как данный тест отличается от VT алгоритмом, используемым в процедуре x_solve, а характер зависимости такой же, то все описанные оптимизации применяются аналогично. Стоит добавить, что изначально используется временный массив lhs типа double precision, размера $15 \times \text{Class}^3$. Для класса C массив занимает примерно 500 Мб и сократить его в данной программе нельзя.

ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ. СРАВНЕНИЕ С OPENCL

Для оценки эффективности распараллеливания с помощью FDVMH использовались две версии каждой из программ: первоначальный последовательный вариант, в котором никаких преобразований не делалось, и преобразованный и оптимизированный вариант с директивами FDVMH. Тестирование производилось на суперкомпьютере K-100 [12], имеющем процессоры Intel Xeon X5670 и ГПУ NVIDIA Tesla C2050 с архитектурой Fermi, и на сервере с процессорами Intel Core i7 и ГПУ NVIDIA GeForce GTX Titan с самой новой архитектурой Kepler. Для сравнения, были получены времена выполнения этих тестов, реализованных на низкоуровневом языке OpenCL исследователями из Сеульского национального университета [13].

На рис. 8 показано ускорение FDVMH версий тестов по сравнению с OpenCL.

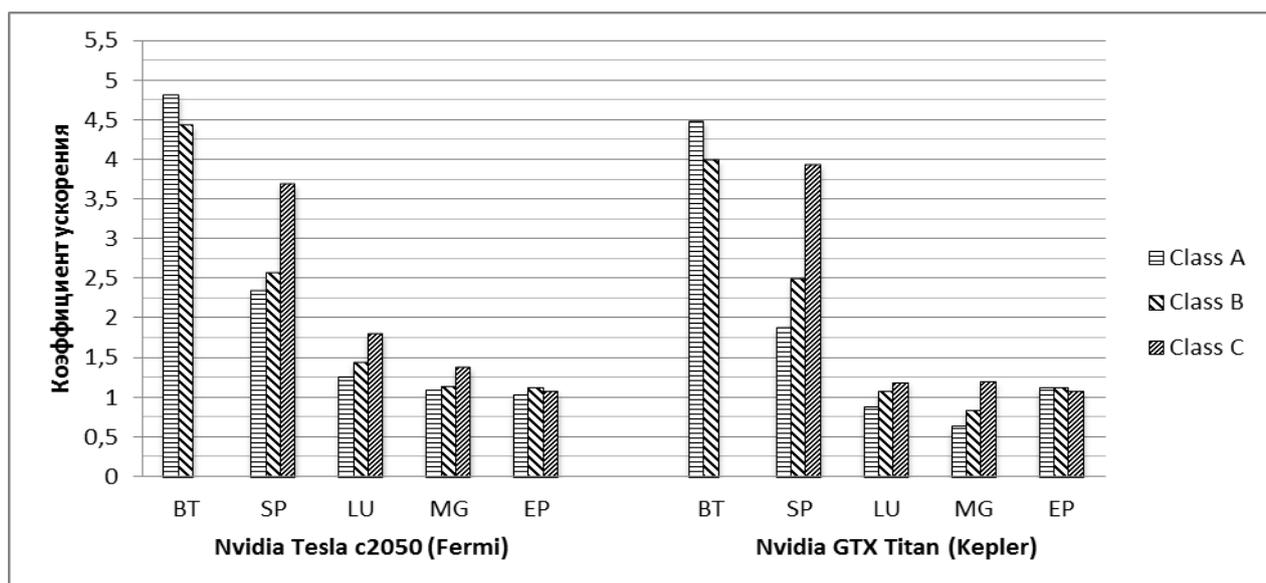


Рис. 8. Ускорение DVHM-программ по сравнению с OpenCL-программами

Таблица

Эффективность распараллеливания тестов

Задача		CPU, Xeon X5670	FORTRAN DVMH				OpenCL			
Тест	Класс		Tesla C2050 (с ECC)		GeForce GTX Titan (без ECC)		Tesla C2050 (с ECC)		GeForce GTX Titan (без ECC)	
		Время, секунд	Время, секунд	Ускорение	Время, секунд	Ускорение	Время, секунд	Ускорение	Время, секунд	Ускорение
BT	A	52,69	15,63	3,37	5	10,54	75,5	0,7	22,41	2,35
	B	221,9	61,74	3,59	18,7	11,87	272,42	0,81	71,7	2,97
	C	951,0	192,29	4,94	56,86	16,73	-	-	-	-
SP	A	36,6	9,82	3,73	3,23	11,33	22,95	1,59	6,09	6,01
	B	154,7	33,4	4,63	11,37	13,61	86,35	1,79	28,53	5,42
	C	637,73	117,55	5,43	35	18,22	433,77	1,47	137,7	4,63
LU	A	40,31	7,0	5,76	5,14	7,84	8,86	4,55	4,49	8,9
	B	170	21,0	8,10	12,51	13,59	30,23	5,62	13,49	12,6
	C	779	70,5	11,05	37,46	20,80	127,95	6,09	44,65	17,4
MG	A	1,41	0,18	7,8	0,11	12,8	0,20	7,05	0,09	20
	B	6,62	0,82	8,07	0,50	13,24	0,94	7,04	0,42	15,7
	C	55,17	5,62	9,8	3,08	17,9	7,83	7,04	3,71	14,8
EP	A	7,97	0,24	33,2	0,38	20,9	0,25	31,88	0,43	18,5
	B	31,85	0,73	43,6	1,12	28,4	0,82	38,8	1,26	25,2
	C	55,17	2,68	20,5	4,14	13,3	2,89	19	4,51	12,2

В таблице указаны времена и ускорения FDVMH и отношению к исходным последовательным версиям тестов, исполненных на одном ядре процессора Intel Xeon X5670. Прочерком в ячейках таблицы обозначены те варианты запусков, для которых не хватило памяти на ГПУ.

Стоит отметить, что реализация теста BT на классе C OpenCL версии требует более 6ГБ памяти (в 4 раза больше, чем требуется FDVMH-программе), что не позволило сравнить OpenCL и FDVMH-версии этого теста.

ЗАКЛЮЧЕНИЕ

В результате проделанной работы были получены оптимизированные и распараллеленные с помощью FDVMH-тесты LU, BT, SP, MG и EP. Трудоемкость распараллеливания с помощью FDVMH и OpenCL можно грубо оценить количеством добавленных строк или слов в исходную программу. Размер полученных DVMH-программ отличается от исходных по-

следовательных не более чем на 45 %. Размер OpenCL-программ отличается в 2–3 раза. В тестах с зависимостями эффективность OpenCL-программ крайне мала. Эффективность распараллеливания FDVMH-версии теста EP незначительно выше эффективности распараллеливания OpenCL-версии этого теста. Эффективность распараллеливания DVMH-версии теста MG выше эффективности распараллеливания OpenCL-версии этого теста на 50 %, если использовать архитектуру Fermi, и на 25 %, если использовать архитектуру Kepler, на классе C. Если рассматривать ресурсоемкие задачи (класс C), то FDVMH-программы показали большую эффективность, чем OpenCL на всех рассматриваемых тестах.

В дальнейшем планируется распараллелить с помощью FDVMH остальные тесты и сравнить их с реализациями на OpenCL и OpenACC. Планируется также исследование эффективности распараллеливания тестов на платформе Intel Xeon Phi.

СПИСОК ЛИТЕРАТУРЫ

1. (2013, Nov. 24). *Top500 List – November 2013 | TOP500 Supercomputer Sites* [Online]. Available: <http://top500.org/list/2013/11/>
2. (2013, Dec. 1). *High Performance Fortran* [Online]. Available: <http://hpff.rice.edu/>
3. **Коновалов Н. А. и др.** Параллельное программирование в системе DVM. Языки Fortran-DVM и C-DVM // Параллельные вычисления и задачи управления RASO-2001: Междунар. конф. (Москва, 2–4 окт. 2001). С. 140–154. [[N. A. Kononov, et al., "Parallel programming in the system DVM. Languages Fortran-DVM and C-DVM," (in Russian), in *Proc. Int. Parallel Computations and Control Problems (RASO-2001)*, Moscow, Russia, 2001, pp. 140-154.]]
4. **Коновалов Н. А., Крюков В. А., Михайлов С. Н., Поребцов А. А.** Fortran DVM – язык разработки мобильных параллельных программ // Программирование. 1995. № 1. С. 49–54. [[N. A. Kononov, V. A. Krukov, S. N. Michailov, A. A. Pgrebzo, "Fortran DVM – language for developing portable parallel programs," (in Russian), in *Programming*, vol. 1, pp. 49-54, 1995.]]
5. **Коновалов Н. А., Крюков В. А., Сазанов Ю. Л.** C-DVM – язык разработки мобильных параллельных программ // Программирование. 1999. № 1. С. 54–65. [[N. A. Kononov, V. A. Krukov, U. L. Sazanov, "C-DVM – language for developing portable parallel programs," (in Russian), in *Programming*, vol. 1, pp. 54-65, 1999.]]
6. **Dolbeau R., Bihan S., and Bodin F.** "HMPPTM: A Hybrid Multi-core Parallel Programming Environment," in *Proc. Workshop on General Purpose Processing on Graphics Processing Units (GPGPU 2007)*.
7. (2013, Nov. 24). *Open ACC* [Online]. Available: <http://www.openacc-standard.org/>
8. **Han T. D., Abdelrahman T. S.** "hiCUDA: High-Level GPGPU Programming," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 1, pp. 78-90, Jan. 2011.
9. **Бахтин В. А., Клинов М. С., Крюков В. А., Поддерюгина Н. В., Пritула М. Н., Сазанов Ю. Л.** Расширение DVM-модели параллельного программирования для кластеров с гетерогенными узлами // Вестник ЮУрГУ. 2012. Т. 12, № 18 (277). С. 82–92. [[V. A. Bakhtin, M. S. Klinov, V. A. Krukov, N. V. Podderugina, M. N. Pritula, U. L. Sazaov, "Extension DVM-parallel programming models for clusters with heterogeneous nodes," (in Russian), in *Vestnik YUrGU*, vol. 12, no. 18 (277), pp. 82-92, 2012]]
10. (2013, Nov. 25). *NAS Parallel Benchmarks* [Online]. Available: <http://www.nas.nasa.gov/publications/npb.html>
11. **Бахтин В. А. и др.** Отображение на кластеры с графическими процессорами циклов с зависимостями по данным в DVMH-программах // Научный сервис в сети Интернет: все грани параллелизма: Междунар. суперком. конф. (Новороссийск, сент. 2013). М.: МГУ, 2013. С. 250–257. [[V. A. Bakhtin, et al., "Mapping loops with dependencies to clusters with graphics processing unit in DVMH programs," (in Russian), in *Superecomp. Proc. Int. Scientific service in the internet: all faces of parallelism*, pp. 250-257, Novorossiisk, Russia, 2013.]]
12. **Гибридный** вычислительный кластер K-100 [Электронный ресурс]. URL: <http://www.kiam.ru/MVS/resourses/k100.html> (дата обращения 25.11.2013). [[(2013, Nov. 25). *Hybrid computing cluster K-100* [Online]. Available: <http://www.kiam.ru/MVS/resourses/k100.html>]]
13. **Seo S., Jo G., Lee J.** "Performance Characterization of the NAS Parallel Benchmarks in OpenCL," in *IEEE Int. Symp. on Workload Characterization (IISWC)*, pp. 137–148, 2011.

ОБ АВТОРАХ

АЛЕКСАХИН Валерий Федорович, ст. науч. сотр. Дипл. математик (МГУ, 1974). Иссл. в обл. языков параллельного программирования.

БАХТИН Владимир Александрович, зав. сект. Дипл. инф.-мат. (МГУ, 1999). Канд. физ.-мат. наук (ИПМ, 2008). Иссл. в обл. автоматического отображения программ на кластеры различной архитектуры в модели DVMH.

ЖУКОВА Ольга Федоровна, науч. сотр. Дипл. мат. (МГУ, 1980). Иссл. в обл. языков параллельного программирования.

КОЛГАНОВ Александр Сергеевич, мл. науч. сотр. Дипл. инф.-мат. (МГУ, 2014). Иссл. в обл. автоматического отображения программ на кластеры различной архитектуры в модели DVMH.

КРЮКОВ Виктор Алексеевич, зав. отделом. Дипл. мат. (МГУ, 1968). Канд. физ.-мат. наук (ИПМ, 1977). Иссл. в обл. операционных систем, языков параллельного программирования.

ПОДДЕРЮГИНА Наталия Викторовна, ст. науч. сотр. Дипл. мат. (МГУ, 1978). Канд. физ.-мат. наук (МГУ, 1983). Иссл. в обл. параллельного программирования.

ПРИТУЛА Михаил Николаевич, науч. сотр. Дипл. инф.-мат. (МГУ, 2010). Канд. физ.-мат. наук (ИПМ, 2013). Иссл. в обл. автоматического отображения программ на кластеры различной архитектуры в модели DVMH.

САВИЦКАЯ Ольга Антониевна, науч. сотр. Дипл. мат. (МГУ, 1978). Иссл. в обл. параллельного программирования.

ШУБЕРТ Алексей Владимирович, асп. Дипл. инф.-мат. (МГУ, 2014), Иссл. в обл. автоматического отображения программ на кластеры в модели DVMH.

METADATA

Title: Parallelization on GPUs of NPB 3.3 NAS tests on language Fortran DVMH.

Authors: V. F. Aleksahin¹, V. A. Bakhtin², O. F. Zhukova³, A. S. Kolganov⁴, V. A. Krukov⁵, N. V. Podderugina⁶, M. N. Pritula⁷, O. A. Savitskaia⁸, A. V. Shubert⁹.

Affiliation:

¹⁻⁸Keldysh Institute of Applied Mathematics of Russian Academy of Sciences (KIAM);

⁹Lomonosov Moscow State University (MSU), Russia.

Email: ¹valex@keldysh.ru, ²bakhtin@keldysh.ru, ³socol@spp.keldysh.ru, ⁴alexander.k.s@mail.ru, ⁵krukov@keldysh.ru, ⁶konov@keldysh.ru, ⁷pritmick@yandex.ru, ⁸savol@keldysh.ru, ⁹alexei@shubert.ru

Language: Russian.

Source: Vestnik UGATU, vol. 19, no. 1 (67), pp. 237-247, 2015. ISSN 2225-2789 (Online), ISSN 1992-6502 (Print).

Abstract: The article describes the transformation of serial versions of the NAS tests package NPB3.3.1 (EP, MG, BT, LU, SP) and the specification of the parallel execution by DVMH-directives necessary for their effective work on clusters with GPUs. The impacts of the various options are examined of parallelization of program performance. The characteristics of tests developed in a high level language Fortran-DVMH (FDVMH), are compared with OpenCL implementation, performed by researchers at Seoul National University.

Key words: DVM; DVMH; CUDA; OpenCL; NASA NPB 3.3.

About authors:

ALEKSAHIN, Valeriy Fedorovich, Master of mathematics (MSU, 1974)

BAKHTIN, Vladimir Aleksandrovich, Master of mathematics and technology (MSU, 1974), Cand. of Physical and Mathematical Sciences (Keldysh Institute of Applied Mathematics, 2008).

ZHUKOVA, Olga Fedorovna, Master of mathematics (MSU, 1980).

KOLGANOV, Alexander Sergeevich, post-graduate student, Master of mathematics and technology (MSU, 2014).

KRUKOV, Victor Alekseevich, Master of mathematics (MSU, 1968), Cand. of Physical and Mathematical Sciences (Keldysh Institute of Applied Mathematics, 1977).

PODDERUGINA, Natalia Viktorovna, Master of mathematics (MSU, 1978), Cand. of Physical and Mathematical Sciences (MSU, 1983).

PRITULA, Mikhail Nikolaevich, Master of mathematics and technology (MSU, 2010), Cand. of Physical and Mathematical Sciences (Keldysh Institute of Applied Mathematics, 2013).

SAVITSKAIA, Olga Antonievna, Master of mathematics (MSU, 1978).

SHUBERT, Aleksei Vladimirovich, post-graduate student, Master of mathematics and technology (MSU, 2014).