Распараллеливание на языке Fortran-DVMH для сопроцессора Intel Xeon Phi тестов NAS NPB3.3.1 *

В.Ф. Алексахин, В.А. Бахтин, О.Ф. Жукова, А.С. Колганов, В.А. Крюков, И.П. Островская, Н.В. Поддерюгина, М.Н. Притула, О.А. Савицкая

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

В докладе анализируется эффективность выполнения тестов NAS из пакета NPB 3.3.1 (EP, MG, BT, SP, LU) на узлах кластеров различной архитектуры, использующих многоядерные универсальные процессоры, графические ускорители фирмы NVidia и сопроцессоры фирмы Intel. Сравниваются характеристики тестов, разработанных на высокоуровневом языке Fortran-DVMH (далее FDVMH), и их реализации на других языках. Исследуется влияние различных оптимизаций для FDVMH-версий тестов NAS, необходимых для их эффективной работы на сопроцессоре Intel Xeon Phi. Представлены результаты запусков тестов при одновременном использовании всех ядер центрального процессора, графического процессора и сопроцессора Intel Xeon Phi.

1. Введение

В последнее время все чаще стали появляться вычислительные кластеры, в узлах которых помимо универсальных многоядерных процессоров установлены ускорители разных типов. В основном, это графические процессоры компании NVidia и сопроцессоры Intel Xeon Phi. Так, в списке Top500 [1] самых высокопроизводительных суперкомпьютеров мира, объявленном в ноябре 2014 года, 75 машин имеют в своем составе ускорители, из них 50 машин имеют ускорители NVidia, 25 – Intel. В четырех машинах используется комбинация ускорителей NVidia и Intel Xeon Phi. 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 OpenMP, CUDA, OpenCL или OpenACC для загрузки ядер центрального и графического процессоров. Поэтому при разработке программы для суперкомпьютера приходится точно знать его архитектуру.

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], C-DVM [3]. Также были предложены модели программирования и соответствующие, основанные на директивах, расширения языков для возможности использования ускорителей, такие, как OpenACC [4] и OpenMP 4.0 [5].

Распараллеливание на графических ускорителях (ГПУ) и сопроцессорах 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: ограниченная поддержкой синхронизации потоков выполнения на ГПУ, модель консистентности глобальной памяти на ГПУ, необходимость синхронизации OpenMP-нитей на сопроцессоре для организации конвейера.

*Исследование выполнено при финансовой поддержке грантов РФФИ № 13-07-00580, 14-01-00109, 14-07-31321_мол_а и Программ фундаментальных исследований президиума РАН №15, №16 и №18.

2. Обзор параллельных архитектур

В данной главе рассматриваются следующие архитектуры: центрального процессора на примере Intel Ivy Bridge-EP [6], сопроцессора Xeon Phi (MIC – Many Integrated Cores) [7] на примере Knights Corner и ГПУ Nvidia Kepler [8] на примере GK110.

2.1. Архитектура Ivy Bridge-EP

В наиболее сложной модификации данной архитектуры используются три блока, включающие четыре ядра ЦПУ и фрагмент кэш-памяти третьего уровня объёмом 2.5 МБ на ядро. Сдвоенная кольцевая шина обеспечивает взаимодействие между блоками внутри чипа, а мультиплексоры позволяют довести команды до ядра, которому они адресованы. Внешняя шина QPI (Quick Path Interconnect), использующаяся для соединения процессоров между собой и с чипсетом, работает на скорости до 9.6 ГТ/с. Встроенный контроллер PCI Express обеспечивает работу 40 линий третьего поколения. В 12-ядерном чипе предусмотрено два контроллера памяти, каждый из которых поддерживает работу памяти до DDR3-1866 в двухканальном режиме.

В данной архитектуре присутствуют SSE (Streaming SIMD Extensions) регистры и AVX (Advanced Vector Extensions) регистры. SSE включает в архитектуру процессора восемь 128-битных регистров и набор инструкций, работающих со скалярными и упакованными типами данных. Преимущество в производительности достигается в том случае, когда необходимо произвести одну и ту же последовательность действий над разными данными. В таком случае блоком SSE осуществляется распараллеливание вычислительного процесса между данными. AVX предоставляет различные улучшения, новые инструкции и новую схему кодирования машинных кодов:

- Ширина векторных регистров SIMD увеличивается со 128 до 256 бит. Существующие 128-битные SSE-инструкции используют младшую половину новых 256-битных регистров, не изменяя старшую часть. Для работы с данными регистрами добавлены новые 256-битные AVX-инструкции. В будущем возможно расширение векторных регистров SIMD до 512 или 1024 бит;

- Для большинства новых инструкций отсутствуют требования к выравниванию операндов в памяти. Однако рекомендуется следить за выравниванием на размер операнда, во избежание значительного снижения производительности;

- Набор AVX-инструкций содержит в себе аналоги 128-битных SSE-инструкций для вещественных чисел. При этом, в отличие от оригиналов, сохранение 128-битного результата будет обнулять старшую половину 256-битного регистра. 128-битные AVX-инструкции сохраняют прочие преимущества AVX, такие как новая схема кодирования, трехоперандный синтаксис и невыровненный доступ к памяти.

2.2. Архитектура Knights Corner

В сопроцессоре с данной архитектурой может быть интегрировано до 61 ядра x86 с большими 512-разрядными векторными модулями (содержащие 512-битные AVX-регистры), работающими на частоте более 1 ГГц и обеспечивающими скорость вычислений двойной точности более 1 TFlops. Они расположены на двухслотовой карте PCI Express со специальной прошивкой на базе Linux. Intel Xeon Phi включает до 16 Гбайт памяти GDDR5. Безусловно, таким образом сконструированные сопроцессоры не рассчитаны на обработку основных задач, с которыми справляются процессоры семейства Xeon E. Они преуспевают в параллельных задачах, способных использовать большое количество ядер для максимального эффекта. Основные характеристики:

- Архитектура x86 с поддержкой 64 бит, четыре потока на ядро и до 61 ядра на сопроцессор;
- 512-битные AVX-инструкции, 512 Кбайт кэша L2 на ядро (до 30,5 Мбайт на всю карту Xeon Phi);
- Поддержка Linux (специальная версия для Phi), до 16 Гбайт памяти GDDR5 на карту.

Можно заметить, что даже у самой старшей модели Intel Xeon Phi гораздо меньше ядер, чем у обычного графического процессора. Но нельзя сравнивать ядро MIC с ядром CUDA в соотношении один к одному, так как одно ядро Intel Xeon Phi – это четырёхпоточный модуль с 512-бит SIMD.

2.3. Архитектура GK110

Чип Kepler GK110 был разработан в первую очередь для пополнения модельного ряда Tesla. Его цель – стать самым быстрым параллельным микропроцессором в мире. Чип GK110 не только превышает по производительности чип предыдущего поколения Fermi, но и потребляет значительно меньше энергии. GK110 в максимальной конфигурации состоит из 15 потоковых мультипроцессоров, называемых SMX, и шести 64-битных контроллеров памяти.

Каждый потоковый мультипроцессор SMX содержит 192 cuda-ядра для операций одинарной точности, 64 cuda-ядра для операций двойной точности, 32 специальных функциональных блока и 32 блока для загрузки и сохранения данных, четыре планировщика. Для всех SMX на ГПУ предусмотрен один общий кэш второго уровня, размер которого составляет 1.5 МБ. Также у каждого SMX есть свой собственный кэш первого уровня размером 64 КБ.

Для того чтобы использовать большие мощности ГПУ, необходимо отобразить вычислительно независимые участки кода на группы независимых виртуальных нитей. Каждая группа будет исполнять одну и ту же команду над разными входными данными. Для управления данной группой виртуальных нитей необходимо объединить их в блоки фиксированного размера. Данные блоки в архитектуре CUDA называются cuda-блоками.

Такой блок имеет три измерения. Все блоки объединяются в решетку блоков, которая также может иметь три измерения. В конечном счете, каждый cuda-блок будет обработан каким-либо потоковым мультипроцессором, и каждой виртуальной нити будет сопоставлена физическая. Минимальной единицей параллелизма на ГПУ является warp – группа из 32 независимых нитей, которые исполняются физически параллельно и синхронно.

3. Обзор пакета тестов NAS

NAS Parallel Benchmarks [9] — комплекс тестов, позволяющий оценивать производительность суперкомпьютеров. Тесты разработаны и поддерживаются в NASA Advanced Super-computing (NAS) Division (ранее NASA Numerical Aerodynamic Simulation Program), расположенном в NASA Ames Research Center. Версия 3.3 пакета NPB включает в себя 11 тестов. В данной статье рассматривается распараллеливание на ЦПУ и сопроцессоры в модели DVMH [10] тестов EP, MG, BT, LU, SP, для которых ранее были разработаны параллельные версии для кластеров с использованием языка Fortran-DVM, а также параллельные версии для графических процессоров с использованием языка Fortran-DVMH:

- MG (Multi Grid) – тест вычисляет приближенное решение трехмерного уравнения Пуассона ("трехмерная решетка") в частных производных на сетке NxNxN с периодическими граничными условиями (функция на всей границе равна 0 за исключением заданных 20 точек). Размер сетки N определяется классом теста. Тестирует возможности системы выполнять как длинные, так и короткие передачи данных;

- EP (Embarrassingly Parallel) – чрезвычайно параллельный. Тест вычисляет интеграл методом Монте-Карло; предназначен 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.

4. Отображение программ на разные архитектуры в системе DVM

В 2011 году в Институте прикладной математики им. М.В. Келдыша РАН была разработана высокоуровневая модель программирования для поддержки кластеров с графическими ускорителями. Модель получила название DVMH [10] (DVM for Heterogeneous systems). Она является расширением модели DVM и позволяет с небольшими изменениями перевести DVM-программу для кластера в DVMH-программу для кластера с графическими ускорителями.

В 2014 году с появлением сопроцессоров Intel Xeon Phi возникла необходимость в их поддержке DVM-системой.

Всего существует три варианта выполнения программ на сопроцессоре Xeon Phi [11]:

- Native режим – программа компилируется и выполняется только на сопроцессоре;

- Offload режим – программа компилируется для ЦПУ, некоторые участки кода компилируются и для ЦПУ и для Xeon Phi. Данные участки кода помечаются специальными директивами OpenMP 4.0 (в общем случае #pragma offload). Программа выполняется так же, как и в случае использования ГПУ: последовательная часть – на ЦПУ, параллельная часть со специальными указаниями – на сопроцессоре. При этом возникает та же проблема, что и при использовании ГПУ – загрузка и выгрузка данных в собственную память сопроцессора через PCIe. В случае отсутствия сопроцессора в узле, специальные участки кода будут выполнены на ЦПУ.

- Symmetric режим – одна и та же программа компилируется отдельно для ЦПУ и отдельно для сопроцессора. Скомпилированные программы одновременно запускаются на сопроцессоре и ЦПУ и могут синхронизироваться с помощью технологии MPI.

Основной режим выполнения, который был выбран для реализации в DVMH, – симметричный (symmetric). Данный режим позволяет настраивать баланс между ЦПУ и сопроцессором с помощью технологии MPI и использовать технологию OpenMP для лучшей загрузки ядер внутри каждого устройства. Также можно запустить FDVMH-программу

только на сопроцессоре (native режим), используя, например, один MPI-процесс и максимальное количество OpenMP-нитей, поддерживаемое конкретным сопроцессором.

В итоге, распараллеливая программы с использованием модели DVMH, не нужно выбирать ту или иную архитектуру, будь то графические процессоры, центральные процессоры или сопроцессоры Intel Xeon Phi, так как эта модель поддерживает использование всех перечисленных архитектур как по отдельности, так и одновременно в рамках одной программы.

5. Реализация поддержки Xeon Phi в FDVMH-компиляторе

Вычислительным регионом в DVMH-программе называется часть программы с одним входом и одним выходом для возможного выполнения на одном или нескольких вычислительных устройствах. Регион может содержать несколько параллельных циклов. Данный участок программы помечается директивой REGION.

Регион может быть выполнен на одном или сразу нескольких устройствах: ускорителях, ЦПУ, сопроцессорах. При этом на ЦПУ или сопроцессоре может быть выполнен любой регион. Для выполнения региона на различных ускорителях на регион могут накладываться различные дополнительные ограничения. Например, на CUDA-устройстве может быть выполнен любой регион, в котором нет операторов ввода/вывода или вызовов внешних процедур.

Для каждого региона можно указать тип вычислителя, на котором следует его выполнять. Для этого предназначена спецификация TARGETS. Вложенные (статически или динамически) регионы не допускаются. DVM-массивы распределяются между выбранными вычислителями (с учетом весов и быстродействия вычислителей), нераспределенные данные размножаются. Витки параллельных DVM-циклов внутри региона делятся между выбранными для выполнения региона вычислителями в соответствии с правилом отображения параллельного цикла, заданного в директиве параллельного цикла.

5.1. Генерация обработчиков для параллельных циклов

Компилятор с языка Fortran-DVMH преобразует исходную программу в параллельную программу на языке Fortran с вызовами функций системы поддержки выполнения DVMH-программ (RTS – RunTime System). Кроме того, компилятор создает для каждой исходной программы несколько дополнительных модулей для обеспечения выполнения регионов программы на ГПУ с использованием технологии CUDA и на ЦПУ и сопроцессоре с использованием технологии OpenMP.

Для каждого параллельного цикла из вычислительного региона компилятор генерирует процедуру-обработчик и ядро для вычислений на ГПУ, а также процедуру-обработчик для выполнения этого параллельного цикла на ЦПУ и сопроцессоре. Обработчик — это подпрограмма, осуществляющая обработку части параллельного цикла на конкретном вычислительном устройстве. Аргументами обработчика являются описатель устройства и часть параллельного цикла.

Обработчик запрашивает порцию для выполнения (границы цикла и шаг), конфигурацию параллельной обработки (количество нитей), инициализацию редукционных переменных и иную системную информацию у RTS. CUDA-обработчик для выполнения частей цикла вызывает специальным образом сгенерированное CUDA-ядро с параметрами, полученными во время выполнения от RTS. CUDA-ядро выполняется на графическом процессоре, производя вычисления, составляющие тело цикла. Для обработки частей цикла ЦПУ-обработчик использует технологию OpenMP для распределения вычислений. По умолчанию предполагается, что вычислительный регион может выполняться на архитектурах всех типов, которые присутствуют в узле кластера, и компилятор генерирует обработчики для ЦПУ, сопроцессора и CUDA-устройства.

Одной из основных причин замедления программ, выполняемых на ЦПУ или сопроцессоре, является линеаризация распределенных массивов, выполняемая компилятором Fortran-DVMH. Память для элементов таких массивов выделяется системой RTS. Для локальной секции массива на каждом процессоре память отводится в соответствии с форматом распределения данных и с учетом теневых граней. Все ссылки к элементам распределенных массивов вида ARRAY(I,J,K) заменяются компилятором на ссылки вида

$$BASE(ARRAY\_OFFSET + I + C\_ARRAY1 * J + C\_ARRAY2 * K),$$

где BASE – это база, относительно которой адресуются все распределяемые массивы; ARRAY_OFFSET – смещение начала массива относительно базы; C_ARRAYi – коэффициенты адресации распределенного массива. Такая программа хуже распознается и оптимизируется стандартными Fortran компиляторами.

Для решения данной проблемы компилятор FDVMH по специальной опции может генерировать оптимизированные обработчики, чтобы распределенные массивы передавались в хост-обработчики как массивы, перенимающие размер (assumed shape arrays). Такой подход позволяет не преобразовывать обращения к массивам в линейную форму внутри хост-обработчиков и существенно ускорить выполнение программы на ЦПУ или сопроцессоре.

5.2. Использование клаузы collapse

В отличие от центральных процессоров, количество нитей на которых не превосходит 24 (для последних Intel Xeon E), сопроцессоры Intel Xeon Phi могут иметь 244 потока. Существует множество задач, в которых циклы не только одномерные, но и двухмерные и трехмерные. К таким задачам, в частности, относятся многие тесты из пакета NPB [9]. Директива OpenMP !$*OMP PARALLEL FOR*, генерируемая компилятором FDVMH в хост-обработчиках, распространяется только на самый верхний цикл из всего гнезда. Если количество итераций такого цикла меньше, чем количество доступных нитей, то производительность будет снижаться.

Для оптимизации работы хост-обработчиков на Xeon Phi необходимо генерировать дополнительную клаузу COLLAPSE в OpenMP-директиве. Клауза COLLAPSE позволяет распределить выполнение нескольких циклов гнезда, что позволяет задействовать все ядра сопроцессора. Параметром данной клаузы служит количество вложенных циклов, на которые она распространяется. Распространять данную клаузу на все вложенные циклы не эффективно, так как один и тот же поток будет обрабатывать элементы, не лежащие подряд, и в этом случае будет больше промахов в L2 кэш.

При статическом анализе во время компиляции DVMH-программы не удается определить количество вложенных циклов, на которое может быть распространена клауза COLLAPSE. Поэтому во время компиляции генерируется несколько хост-обработчиков, в каждом их которых расставляется клауза COLLAPSE(N), где $N = 1, 2, 3, ..m$, а m есть количество циклов в гнезде. В момент выполнения программы RTS определяет обработчик, который будет выполнять параллельный цикл.

При компиляции DVMH-программы пользователь может задать опцию компилятору $-collapseN$, где N – целое число, большее 0. Тогда для каждого параллельного цикла в хост-обработчике в OpenMP-директиву будет добавлена клауза COLLAPSE(N).

5.3. Балансировка нагрузки в DVM-системе

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 выполнения всех поддерживаемых языком конструкций на разнородных вычислительных устройствах, балансировка нагрузки между вычислительными

устройствами, а также выбор оптимального способа выполнения каждого участка кода на том или ином устройстве. Параллелизм в DVMH-программах проявляется на нескольких уровнях:

- Распределение данных и вычислений по MPI-процессам. Этот уровень задается директивами распределения и перераспределения данных и спецификациями параллельных подзадач и циклов. На данном уровне происходит отображение программы на узлы кластера. Внутри каждого узла может находиться несколько MPI-процессов, которые могут быть отображены на ядра ЦПУ или ядра сопроцессора;
- Распределение данных и вычислений по вычислительным устройствам при входе в вычислительный регион;
- Параллельная обработка в рамках конкретного вычислительного устройства. Этот уровень появляется при входе в параллельный цикл, находящийся внутри вычислительного региона. На данном уровне происходит отображение вычислений на OpenMP-нити и архитектуру CUDA.

Согласно данному разделению в DVMH-модели существует два уровня балансировки между вычислительными устройствами: задание веса каждого MPI-процесса, отображаемого на ядра ЦПУ или сопроцессора, и задание соотношения вычислительной мощности между ядрами ЦПУ и ГПУ для каждого MPI-процесса.

Для настройки производительности DVMH-программы не требуется ее перекомпиляция. Чтобы определить, как соотносятся веса MPI-процессов, пользователю необходимо указать вектор весов в специальном файле с параметрами запуска DVMH-программы. В соответствии с указанными весами, RTS разделит данные между MPI-процессами во время работы программы. По умолчанию все MPI-процессы считаются одинаковыми.

Для балансировки нагрузки между ЦПУ и ГПУ существуют следующие механизмы. Все настройки осуществляются при помощи переменных окружения. Пользователь может определить количество нитей в терминах OpenMP на ЦПУ (или сопроцессоре), которое нужно использовать. Также можно включить или отключить выполнение параллельного цикла на одном или нескольких ГПУ. Для того, чтобы указать производительность ЦПУ и ГПУ, существуют две переменные окружения:

- DVMH_CPU_PERF – относительная производительность ЦПУ. Для разных MPI-процессов, выполняемых на ядрах ЦПУ или ядрах сопроцессора Intel Xeon Phi, существует возможность указать различные значения этого параметра;
- DVMH_CUDAS_PERF – относительная производительность ГПУ, задается списком вещественных чисел через пробел или запятую. Список является закольцованным. Это позволяет не расписывать производительности всех ГПУ.

Таким образом, DVM-система позволяет эффективно использовать все устройства различной архитектуры, установленные в узлах кластера, путем двухуровневой балансировки: определение веса каждого из MPI-процессов и определение соотношения производительности между OpenMP-нитями и CUDA устройствами внутри каждого из MPI-процессов.

6. Применение AVX-инструкций

Минимальной единицей параллелизма в терминах ГПУ является warp, который состоит из 32-х независимых нитей. Уже при написании параллельной программы с использованием программной модели CUDA прикладной программист сам определяет warp'ы, объединяет их в блоки, а затем в сетки блоков, тем самым указывая, где и что будет исполняться

параллельно и векторно. Можно считать, что минимальный размер вектора в архитектуре ГПУ равен 32-м элементам. Дальнейшие сложности по оптимизации, планированию и загрузке/выгрузке данных берет на себя компилятор NVidia и аппаратура ГПУ.

Минимальной единицей параллелизма одного ядра ЦПУ или сопроцессора можно назвать векторный AVX-регистр, который может обрабатывать 256 или 512 бит данных за одну операцию соответственно. В данном случае прикладной программист работает сначала с последовательным кодом программы, а затем применяет высокоуровневые директивные расширения OpenMP. И в отличие от ГПУ, в программе, ориентированной на ЦПУ или сопроцессор, нет явных указаний о векторизации операций – всю работу по векторизации берет на себя компилятор. Поэтому есть два варианта написания эффективных параллельных программ, использующих векторные регистры AVX:

- использовать директивы компилятора или OpenMP (например, "*omp simd*");
- использовать низкоуровневые команды или AVX-инструкции (на уровне intrinsic-функций).

Первый способ не всегда реализуем, так как компилятор не всегда может справиться с векторизацией, даже если она возможна. Рассмотрим два варианта одного и того же цикла (см. Рис. 1).

```
#if VER1 /*** первый вариант цикла ***/
  double rhs[5][162][162][162], u[5][162][162][162];
  #define rhs(m,i,j,k) rhs[(m)][(i)][(j)][(k)]
  #define u(m,i,j,k)     u[(m)][(i)][(j)][(k)]
#endif
#if VER2 /*** второй вариант цикла ***/
  double rhs[162][162][162][5], u[162][162][162][5];
  #define rhs(m,i,j,k) rhs[(i)][(j)][(k)][(m)]
  #define u(m,i,j,k)     u[(i)][(j)][(k)][(m)]
#endif
#pragma omp parallel for
for(int i = 0; i < Ni; i++)
{
    for(int j = 0; j < Nj; j++)
    {
        for(int k = 0; i < Nk; k++)
        { /*** первая группа операторов ***/
            rhs(0, i, j, k) = F(u(0, i, j, k), u(0, i+1, j, k));
            rhs(1, i, j, k) = F(u(1, i, j, k), u(1, i-2, j, k));
            rhs(2, i, j, k) = F(u(2, i, j, k), u(2, i, j+1, k));
            rhs(3, i, j, k) = F(u(3, i, j, k), u(3, i, j-1, k));
            rhs(4, i, j, k) = F(u(4, i, j, k), u(4, i+1, j, k+1));
          /*** вторая группа операторов ***/
            for (int m = 0; m < 5; m++)
                rhs(m, i, j, k) += F(u(m, i, j, k), u(m, i, j+1, k));
        }
    }
}
```

Рис. 1. Два варианта параллельного цикла

Данные циклы производят одинаковые вычисления, но отличаются расположением данных в памяти. В первом варианте (VER1) компилятор не может векторизовать ни одну группу операторов, потому что самое быстро индексируемое измерение является измерением параллельного цикла. Во втором варианте (VER2) компилятор успешно векторизует вторую группу операторов, так как самое быстро индексируемое измерение не является измерением параллельного цикла и данные в памяти по этому измерению расположены подряд.

Тем не менее, бывают ситуации, когда выгодно использовать расположение данных такое, как в первом варианте (VER1, Рис. 1). Одна из таких ситуаций – перестановка массивов

в предшествующем параллельном цикле программы для лучшей локализации данных. И для того, чтобы не произошло замедления выполнения рассматриваемого цикла, необходимо использовать векторные AVX-инструкции непосредственно в коде программы.

Для первого варианта (VER1) возможна векторизация двух групп операторов, потому что данные в памяти по самому быстрому измерению параллельного цикла лежат подряд. Пример преобразования одного из операторов с применением AVX-инструкций показан на Рис. 2.

```
double rhs[5][162][162][162], u[5][162][162][162];
#pragma omp parallel for
for(int i = 0; i < Ni; i++)
{
  for(int j = 0; j < Nj; j++)
  {
    /*** изменение индексного пространства цикла ***/
    for(int k = 0; i < Nk; k+=8)
    {  /*** rhs[0][i][j][k] = u[0][i][j][k] + u[0][i+1][j][k]; ***/
        _mm512_store_pd(&rhs[0][i][j][k],
                        _mm512_add_pd(
                                _mm512_load_pd(u[0][i][j][k]),
                                _mm512_load_pd(u[0][i+1][j][k]
                                      )
                        );
    }
  }
}
```

Рис. 2. AVX-преобразование

За одну операцию 512-битная AVX-команда может обрабатывать 8 элементов типа double. Использование данного подхода позволяет ускорить программу примерно в 8 раз. Для подтверждения данного факта была реализована одна из вычислительно сложных процедур теста NPB SP на языке C++ с использованием AVX-инструкций и без. Полученные программы компилировались Intel-компилятором с флагом оптимизации -O3.

В результате проведенного эксперимента было достигнуто ускорение порядка 6.3 раз по сравнению с той же процедурой без использования AVX-инструкций (см. Таблицу 1), что говорит о высокой эффективности использования данного подхода. Возможность использования AVX-инструкций в Fortran-DVMH компиляторе является предметом для дальнейших исследований.

Таблица 1. Сравнение времени выполнения разных реализаций процедуры compute_rhs программы SP

	Xeon Phi 240th	Xeon Phi 240th + AVX512	SpeedUp
CLASS A	2,9 sec	1,8 sec	1,55
CLASS B	13,4 sec	4,8 sec	2,76
CLASS C	118,1 sec	18,7 sec	6,31

7. Полученные результаты. Сравнение с MPI и OpenMP версиями тестов NASA

Для оценки эффективности распараллеливания программ с использованием модели DVMH было произведено сравнение времени выполнения тестов NAS (MG, EP, SP, BT и LU из пакета NPB 3.3), написанных на языке Fortran-DVMH, с временем выполнения стандартных версий этих тестов, использующих технологии MPI и OpenMP. Для каждого теста имеется всего один вариант параллельной FDVMH-программы, который может быть скомпилирован под каждую из архитектур: ЦПУ, сопроцессор или ГПУ. Все оптимизации, которые были проделаны над данными тестами, были описаны в статье [12].

Тестирование производилось на сервере с установленными на нем 6-ядерным (12-поточным) процессором Intel Xeon E5-1660v2 с 24Гб оперативной памяти типа DDR3, сопроцессором Intel Xeon Phi 5110P с 8Гб оперативной памяти типа GDDR5 и ГПУ NVidia GTX Titan с 6Гб оперативной памяти типа GDDR5. Основные результаты представлены в Таблице 2.

Таблица 2. Времена выполнения тестов NASA различных версий (в секундах)

Application		NASA					Fortran-DVMH		
Test	Class	Xeon E5 (4-12th)			Xeon Phi (64-240th)		Xeon E5	Xeon Phi	GTX
		Serial	MPI	OpenMP	MPI	OpenMP	12th	240th	Titan
BT	A	40,7	12,13	10,2	11,08	11,5	7,8	7,68	2,84
	B	166,9	54,9	43,07	33,1	32,15	32,2	20,9	9,16
	C	713,3	223,1	176,7	119,4	105,7	125	74	31,05
SP	A	28,6	17,8	14,6	12,03	13,3	15,5	11,6	2,4
	B	116,94	96,8	57,1	33,4	38,7	37	27	10,2
	C	483,24	408,6	425,2	124,03	128,2	174,1	120	31
LU	A	35,07	9,6	8,31	15,05	16,5	18,9	33,75	4,18
	B	148,56	35,2	31,10	47,01	44,5	77,7	89,8	11,69
	C	852,3	291,4	351,9	162,4	134,2	312,5	192,5	34,32
MG	A	1,06	0,57	0,7	0,36	0,22	0,8	0,61	0,13
	B	4,96	2,7	3,22	1,7	1,13	3,8	2,8	0,58
	C	42,3	25,7	34,6	10,9	6,39	29,7	15,5	3,36
EP	A	16,73	1,63	1,76	0,94	0,89	1,5	0,78	0,48
	B	67,33	6,6	7,03	3,94	3,31	5,99	2,99	1,17
	C	266,3	26,1	26,3	14,8	13,31	23,96	11,6	4,27

На Рис.3 показано ускорение теста EP, написанного с использованием языка FDVMH, по сравнению с последовательной версией данной программы, выполненной на одном ядре ЦПУ. Данный тест выполнялся на разных архитектурах по отдельности, а также в следующих комбинациях: ЦПУ + ГПУ, ЦПУ + сопроцессор и сопроцессор + ЦПУ + ГПУ. На Рис.3 для каждой конфигурации запуска указывается количество MPI-процессов и количество OpenMP-нитей внутри каждого из процессов. Красным и сиреневым цветом показаны случаи, когда дополнительно использовалась балансировка нагрузки путем задания соотношения весов всех ядер ЦПУ и ГПУ и соотношения весов MPI-процессов, отображаемых на ЦПУ и сопроцессор.

В итоге, на данном тесте при одновременном использовании ГПУ и ЦПУ удалось достичь производительности, которая на 17% больше производительности выполнения теста на одном ГПУ. При этом соотношение производительности ЦПУ и ГПУ было установлено как 1 : 5,7 соответственно.

При совместном использовании сопроцессора и ЦПУ удалось достичь производительности, которая на 30% больше производительности выполнения теста на одном сопроцессоре. Наиболее выгодное соотношение MPI-процессов следующее: два MPI-процесса на Xeon Phi и один MPI-процесс на Xeon E5, при этом MPI-процессы имеют одинаковый вес, либо по одному MPI-процессу на ЦПУ и сопроцессор, и веса процессов соотносятся как 1 : 2 соответственно.

При использовании всех устройств, установленных в узле, удалось достичь производительности, которая на 28% больше производительности выполнения теста EP на ГПУ и в 3.7 раз превышает производительность выполнения этого теста на сопроцессоре.

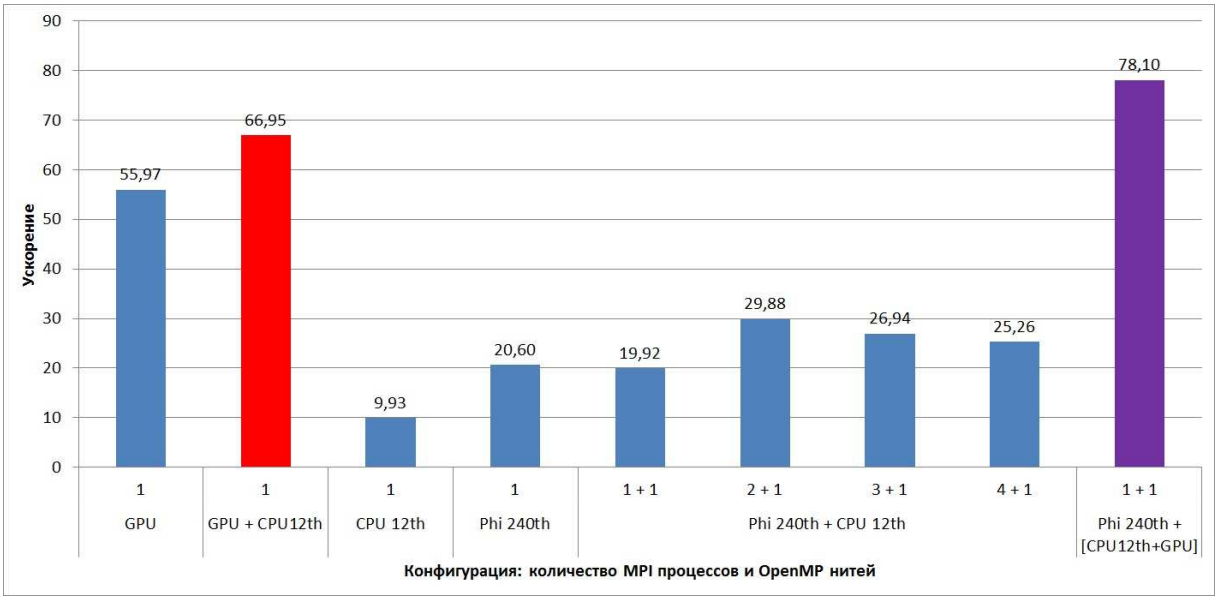

Рис. 3. Использование балансировки нагрузки в DVMH-программе на примере теста EP класса C

8. Заключение

В результате данного исследования было реализовано расширение DVM-системы для поддержки ускорителей Intel Xeon Phi. Таким образом, сделан существенный шаг в направлении обеспечения эффективной переносимости FDVMH-программ, когда одна и та же параллельная программа может эффективно выполняться на кластерах различной архитектуры, использующих многоядерные универсальные процессоры, графические ускорители и сопроцессоры Intel Xeon Phi. Реализованное отображение DVMH-программы на ЦПУ и сопроцессор позволяет применять многие из тех оптимизаций последовательной программы, которые были предложены в статье [12], и позволяющие эффективно отображать данные программы на графический процессор.

Из приведенных выше времен выполнения реализованных FDVMH-программ видно, что эффективность работы на ускорителе Intel Xeon Phi и ЦПУ тестов MG и LU невысока. В тесте LU содержатся циклы с зависимостями более, чем по одному измерению. Вполне возможно, что диагональная схема выполнения вместе с диагональной трансформацией массивов во время выполнения с помощью RTS может дать такой же эффект, как на ГПУ [13]. Данная проблема является предметом для дальнейших исследований.

Литература

1. Top500 List – November 2014 | TOP500 Supercomputer Sites URL: http://top500.org/list/2014/11/ (дата обращения 30.11.2014)
2. High Performance Fortran URL: http://hpff.rice.edu/ (дата обращения 30.11.2014)
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., с. 140-154
4. OpenACC URL: http://www.openacc-standard.org/ (дата обращения 30.11.2014)
5. OpenMP 4.0 Specifications. URL: http://openmp.org/wp/openmp-specifications/ (дата обращения 30.11.2014)
6. Архитектура Intel Ivy Bridge-EP URL: http://www.intel.ru/content/www/ru/ru/secure /intelligent-systems/privileged/ivy-bridge-ep/xeon-e5-1600-2600-v2-bsdl.html (дата обращения 30.11.2014)
7. Архитектура Intel MIC URL: https://software.intel.com/mic-developer (дата обращения 30.11.2014)
8. Архитектура Nvidia Kepler URL: http://www.nvidia.com/content/PDF/kepler/ NVIDIAKepler-GK110-Architecture- Whitepaper.pdf/ (дата посещения 30.11.2014)
9. NAS Parallel Benchmarks URL: http://www.nas.nasa.gov/publications/npb.html (дата обращения 30.11.2014)
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. Intel Xeon Phi programming environment URL: https://software.intel.com/en-us/articles/intel-xeon-phi-programming-environment (дата обращения 30.11.2014)
12. В.Ф. Алексахин, В.А. Бахтин, О.Ф. Жукова, А.С. Колганов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула, О.А. Савицкая, А.В. Шуберт. Распараллеливание на графические процессоры тестов NAS NPB3.3.1 на языке Fortran DVMH. // Труды международной научной конференции "Параллельные вычислительные технологии (ПаВТ'2014)" (Ростов-на-Дону, 31 марта – 3 апреля 2014 г.) – Челябинск: Издательский центр ЮУрГУ, 2014, с. 30-41.
13. Бахтин В.А., Колганов А.С., Крюков В.А., Поддерюгина Н.В., Притула М.Н. Отображение на кластеры с графическими процессорами DVMH-программ с регулярными зависимостями по данным // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2013. Т. 2. № 4. С. 44-56.
14. Arunmoezhi Ramachandran, Jerome Vienne, Rob Van Der Wijngaart, Lars Koesterke, Ilya Sharapov. "Performance Evaluation of NAS Parallel Benchmarks on Intel Xeon Phi" . In Proceedings of the 42nd International Conference on Parallel Processing, 2013, pp. 736-743