Исследование эффективности выполнения DVMH-программ на кластерах с многоядерными процессорами и ускорителями

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

В докладе анализируется эффективность выполнения на кластерах различной архитектуры (использующих в узлах многоядерные универсальные процессоры, графические ускорители фирмы NVidia и сопроцессоры Intel Xeon Phi) программ на языке Fortran DVMH (далее FDVMH). Сравниваются характеристики программ, разработанных на FDVMH, и их реализаций на других языках. Исследуется влияние на эффективность выполнения DVMH-программы различных оптимизаций, которые могут быть применены при ее отображении на кластеры.

1. Введение

В 2011 году в Институте прикладной математики им. М.В. Келдыша РАН была разработана модель параллельного программирования DVMH (DVM for Heterogeneous systems) для кластеров с ускорителями [2]. Она является расширением модели DVM [1] и позволяет легко перевести DVM-программу для кластера в DVMH-программу для кластера с ускорителями.

В модели DVMH программист определяет фрагменты кода, которые могут быть выполнены на ускорителях. Такие фрагменты называются вычислительными регионами или просто регионами.

Регион может выполняться на одном или нескольких ускорителях одного или разных типов и/или на центральном процессоре. Программмист может указать список типов вычислителей, на которых предполагается выполнять регион.

Фрагменты программы вне регионов всегда выполняются на центральном процессоре.

Для каждого региона указываются данные, необходимые для его выполнения (входные, выходные, локальные).

Перемещения данных между вычислительными регионами осуществляется в основном автоматически в соответствии с имеющейся в описании регионов информацией об используемых ими данных. Для управления перемещениями данных между ускорителями и центральным процессором предусмотрены специальные средства.

В 2014 году реализовано отображение FDVM-программ в OpenMP-программы. Таким образом, компилятор FDVM преобразует FDVM-программы в программы, использующие теперь три стандартные модели параллельного программирования - MPI, OpenMP и CUDA. Это должно повысить эффективность выполнения FDVM-программ на многоядерных универсальных процессорах и сопроцессоре Intel Xeon Phi.

В рамках данного исследования анализируется эффективность выполнения программ на языке Fortran DVMH на кластерах различной архитектуры и исследуются различные оптимизации, которые могут быть применены при отображении FDVMH-программ на кластеры.

Набор FDVMH-программ состоял из небольших вычислительных ядер, тестов NAS NPB 3.3.1 и нескольких прикладных программ, разработанных в ИПМ им.М.В.Келдыша РАН.

Исследование выполнялось на суперкомпьютере К-100 [7] и экспериментальном сервере с процессором Intel Xeon E5-1660 V2 @ 3.7 GHz, оперативной памятью 24 ГБ, сопроцессором Intel Xeon Phi 5110P и ГПУ NVidia GTX Titan (Kepler).

В процессе исследования были проанализированы следующие времена выполнения:

- последовательной версии программы на одном ядре процессора Intel Xeon E5-1660,
- последовательной версии программы на одном ядре сопроцессора Intel Xeon Phi 5110P,
- параллельных версий на разном количестве ядер процессора Intel Xeon E5-1660,
- параллельных версий на разном количестве ядер сопроцессора Intel Xeon Phi 5110P,
- параллельных версий на 1 ГПУ NVidia GTX Titan.

В качестве параллельных версий программ, помимо FDVMH-версий, использовались версии, созданные сторонними разработчиками на базе MPI, OpenMP, CUDA, OpenCL.

Компилятор FDVM преобразует FDVM-программы в программы, использующие стандартные модели параллельного программирования - MPI, OpenMP и CUDA. В процессе этого преобразования и во время выполнения полученных программ производятся различные оптимизации.

Исследовалось влияние следующих оптимизаций:

- выбор количества MPI-процессов и OpenMP-нитей, балансировка их загрузки;
- выбор количества объединяемых (collapse) в OpenMP-программе циклов гнезда;
- добавление в OpenMP-программу указаний о векторизации циклов;
- разбиение цикла для устранения зависимостей между его витками;
- перестановка циклов внутри гнезда;
- перестановка измерений массива;
- другие виды реорганизации данных, приводящих к изменению расположения элементов массива.

2. Оптимизации DVMH-программы

2.1. Выбор количества MPI-процессов и OpenMP-нитей, балансировка их загрузки

В настоящее время в DVMH-программах все распределяемые массивы распределяются блочно: каждое распределенное измерение делится на отрезки. По каждому измерению распределенного массива можно задавать или равномерное блочное распределение, или распределение взвешенными блоками, т.е. с учетом заданного вектора весов. Эти указания непосредственно выполняются при распределении данных между MPI-процессами. Вложенные распределения внутри MPI-процессов строятся при входе в вычислительный регион с использованием этой информации, но, в силу разнородности вычислительных устройств, могут иметь отличающуюся от внешней схему распределения.

Системой поддержки выполнения DVMH-программ обеспечиваются при входе в регион [3] три режима распределения данных и вычислений между вычислительными устройствами:

1. Простой статический режим.
2. Динамический режим с подбором схемы распределения.
3. Динамический режим с использованием подобранной схемы распределения.

Выбор режима распределения данных и вычислений может существенно повлиять на повышение производительности программы. На эффективность выполнения программы в определенном режиме может оказать влияние задание следующих переменных окружения:

DVMH_PPN - количество MPI-процессов, запускаемых на узле/сопроцессоре.

DVMH_NUM_THREADS - количество нитей, работающих на ЦПУ или ядрах сопроцессора Intel Xeon Phi. Существует возможность указать различные значения этого параметра для разных MPI-процессов.

DVMH_CPU_PERF - относительная производительность MPI-процесса. 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.

2.2. Оптимизации обработчиков циклов

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

Для каждого параллельного цикла из региона, предназначенного для выполнения на ГПУ, компилятор генерирует процедуру-обработчик и вычислительное

CUDA-ядро, а также процедуру-обработчик для выполнения этого параллельного цикла на ЦПУ/сопроцессоре.

Обработчик – это подпрограмма, осуществляющая обработку части параллельного цикла на конкретном вычислительном устройстве. Обработчик запрашивает порцию для исполнения (границы цикла и шаг), конфигурацию параллельной обработки (количество нитей), инициализацию редукционных переменных, а после выполнения порции передает результат частичной редукции в систему поддержки.

В случае CUDA-обработчика для обработки частей цикла обработчик вызывает специальным образом сгенерированное ядро. CUDA-ядро выполняется на графическом процессоре, производя вычисления, составляющие тело цикла.

ЦПУ-обработчик использует технологию OpenMP для распределения вычислений цикла между ядрами ЦПУ/сопроцессора.

По умолчанию предполагается, что регион может выполняться на вычислителях всех типов, имеющихся на конкретном кластере, и компилятор генерирует обработчики для ЦПУ, ГПУ и сопроцессора Intel Xeon Phi.

Одной из первых проблем, с которой авторы доклада стокнулись при выполнении данного исследования, было существенное замедление выполнение параллельной DVMH-программы на 1 ядре ЦПУ по сравнению с выполнением последовательной программы (в 1.3 – 2 раза). Причиной такого замедления была линеаризация распределенных массивов, выполняемая компилятором Fortran DVMH. Память для элементов таких массивов выделяется системой поддержки выполнения DVMH-программ. 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+COEFF_ARRAY1*J+COEFF_ARRAY2*K),

где BASE – это база, относительно которой адресуются все распределяемые массивы; ARRAY_OFFSET – смещение начала массива относительно базы; COEFF_ARRAY1, COEFF_ARRAY2 – коэффициенты адресации распределенного массива. Такая программа хуже распознается и оптимизируется стандартными Fortran компиляторами.

Разработанная новая версия DVMH-компилятора лишена данного недостатка. Распределенные массивы передаются в хост-обработчики как параметры — массивы, перенимающие размер (в формальном параметре вместо верхней границы последней размерности указывается звездочка “*”). Такой подход позволяет не линеаризовать обращения к массивам внутри обработчиков и существенно ускорить выполнение программы.

Сопроцессоры Intel Xeon Phi могут иметь до 61 ядра, 244 потоков и обеспечивать производительность до 1,2 терафлопс. Для эффективного использования сопроцессоров Intel Xeon Phi недостаточно одномерного распараллеливания циклов с использованием директивы !$OMP PARALLEL FOR. Например, даже для тестов NAS класса C размеры используемых массивов не превышают 162x162x162x75, что не позволяет использовать более 162 нитей для параллельного выполнения цикла. Клауза COLLAPSE позволяет распределить выполнение нескольких циклов гнезда, что позволяет задействовать все ядра сопроцессора.

В последней версии стандарта OpenMP 4.0 появилась возможность явно указать компилятору на необходимость векторизации кода при помощи конструкции !$OMP SIMD. Это один из наиболее эффективных способов оптимизации производительности ПО, который планируется использовать в компиляторе Fortran DVMH.

Для циклов с регулярной зависимостью по данным компилятор Fortran DVMH организует конвейерное выполнение с использованием средств OpenMP. Однако такое конвейерное выполнение требуется не всегда. Например, при использовании метода переменных направлений, существует цикл в котором есть зависимость по одному измерению массива и цикл, витки которого можно выполнять параллельно.

Новая версия DVMH-компилятора генерирует несколько OpenMP-обработчиков для циклов с зависимостью по данным (конвейерное выполнение, параллельное выполнение самого внешнего цикла, параллельное выполнение вложенного цикла...). Система поддержки выполнения DVMH-программ в динамике определяет какой обработчик требуется вызвать.

2.3. Динамическое переупорядочивание массивов

Для оптимизации доступа к глобальной памяти ГПУ 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 работали 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 располагаются в соседних ячейках памяти, что позволяет применять технику выполнения цикла с зависимостями по гиперплоскостям без значительной потери производительности на операциях доступа к памяти.

Исследуется возможность реализации подобной оптимизации в компиляторе Fortran DVMH и для ЦПУ-обработчиков.

3. Результаты

На рис.1 приводятся времена выполнения различных версий теста SP (DVMH [4] и OpenMP [5]) из пакета NAS NPB, полученные на экспериментальном сервере с 6-ти ядерным 12-ти поточным процессором Intel Xeon E5-1660, сопроцессором Intel Xeon Phi 5110P и ГПУ NVidia GTX Titan.

Tphi – время выполнения программы на сопроцессоре Intel Xeon Phi при использовании до 240 нитей.

Txeon_e5 - время выполнения программы на процессоре Intel Xeon E5-1660 при использовании до 12 нитей.

Tgtx_titan - время выполнения DVMH-программы на ГПУ NVidia GTX Titan.

Tgtx_titan_tautotfm - время выполнения DVMH-программы на ГПУ NVidia GTX Titan при использовании динамического переупорядочивания массивов.

Tphi_collapse_tr - время выполнения DVMH-программы на сопроцессоре Intel Xeon Phi при использовании 240 нитей, клаузы collapse и реализованной вручную перестановки измерений массива.

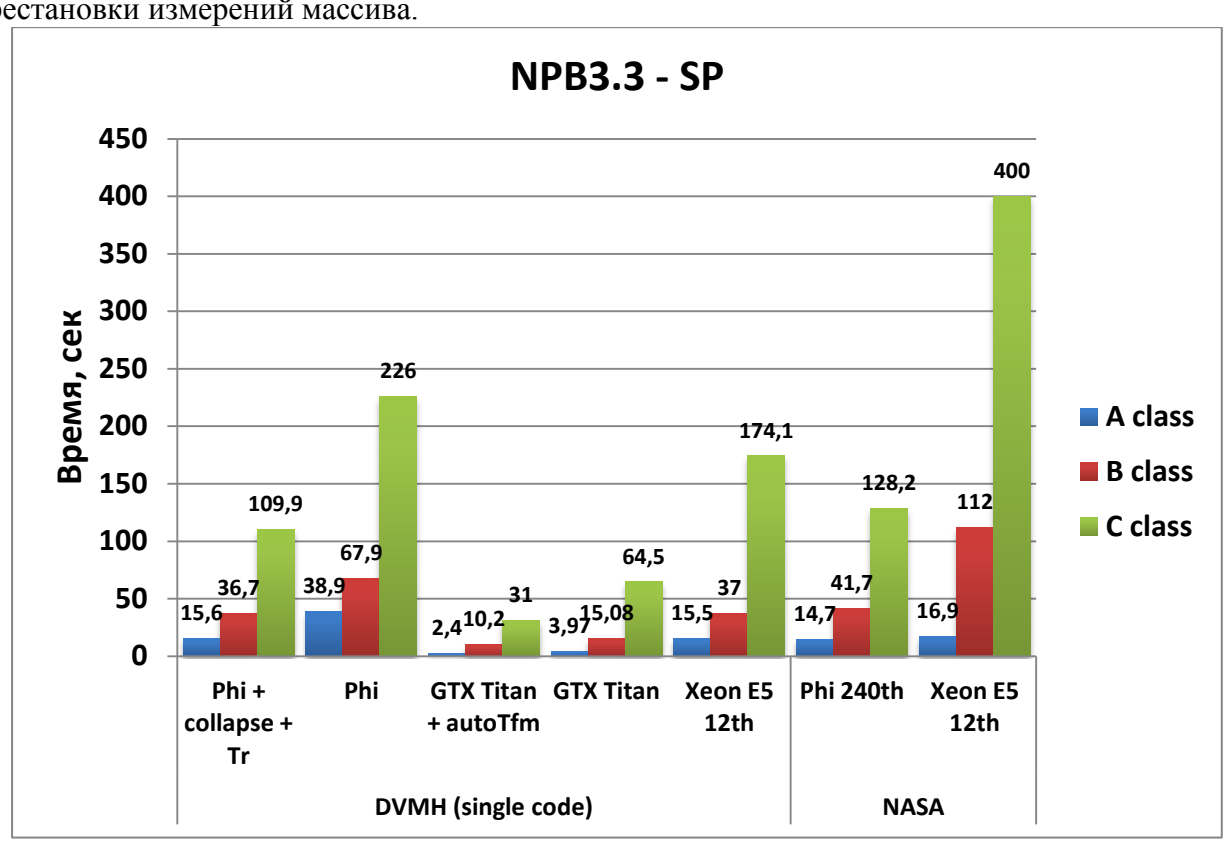

Рис. 1. Влияние различных оптимизаций на время выполнения теста SP

Использование клаузы collapse, а также реализованная вручную перестановка измерений массива позволила для DVMH-версии теста SP получить небольшой выигрыш на сопроцессоре Intel Xeon Phi по сравнению со стандартной OpenMP-версией теста NAS, а также по сравнению с выполнением программы на процессоре Intel Xeon E5-1660. Однако полученная программа выполняется почти в 3 раза медленнее, чем на ГПУ NVidia GTX Titan.

В рамках данного исследования был проведен эксперимент по переписыванию части программы с использованием AVX-инструкций (intrinsic-функций). На рис. 2 показано ускорение одной из основных процедур compute_rhs теста SP при использовании AVX-инструкций. Для класса C процедура compute_rhs ускорилась почти в 6.3 раза. Возможность использования AVX-инструкций в Fortran DVMH компиляторе является предметом для дальнейших исследований.

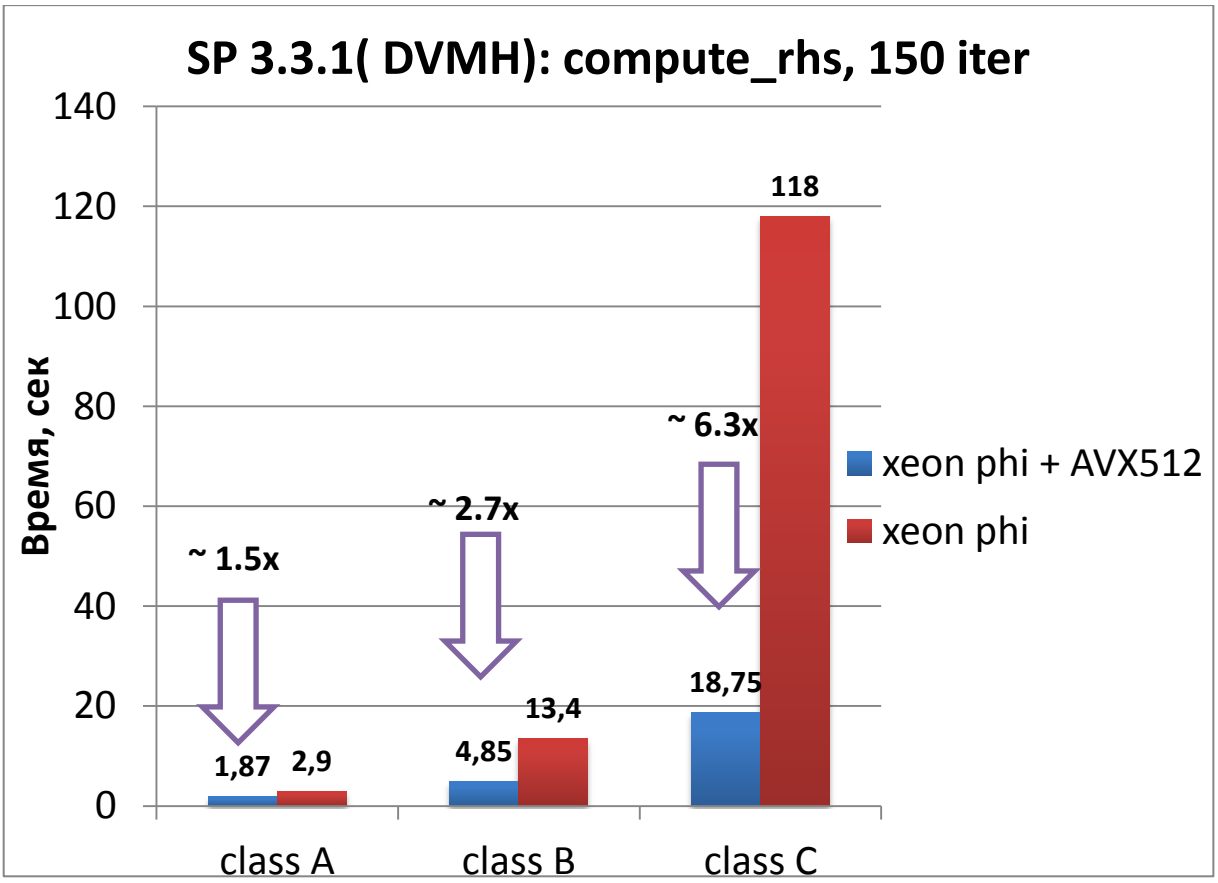

Рис. 2. Ускорение выполнения процедуры compute_rhs теста SP при использовании AVX-инструкций

На рис.3 приводятся времена выполнения DVMH-версии теста EP класса C из пакета NAS NPB, полученные на экспериментальном сервере при одновременном использовании ядер ЦПУ, сопроцессора Intel Xeon Phi и ГПУ NVidia GTX Titan для различных конфигураций запуска программы (числа используемых MPI-процессов, OpenMP-нитей и ГПУ).

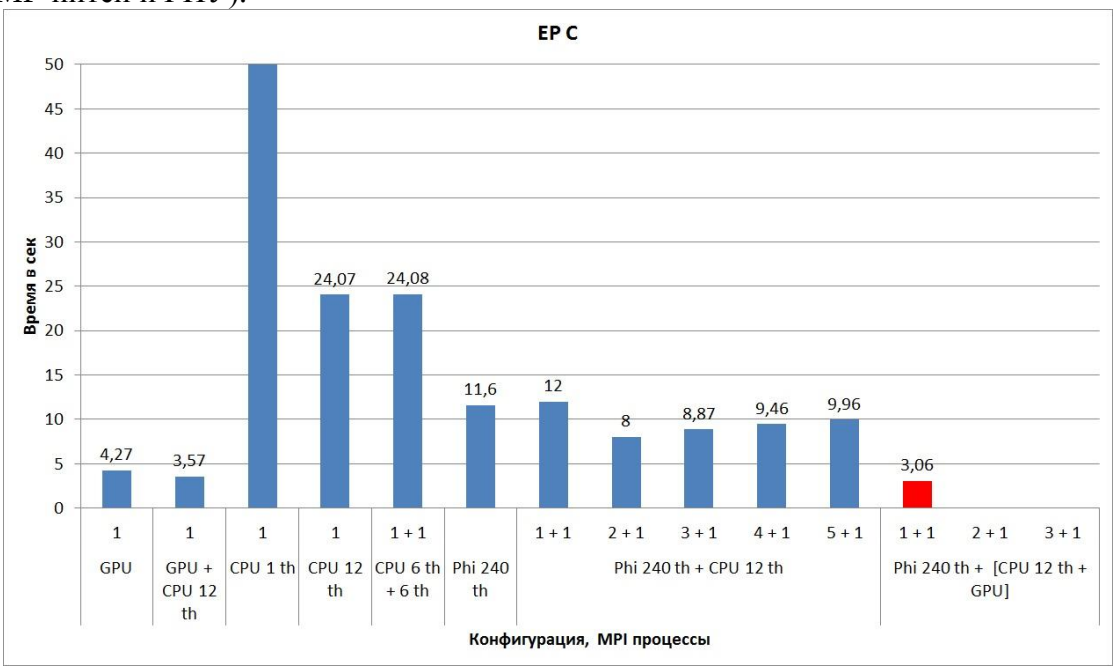

Рис. 3. Времена выполнения теста EP при использовании различных конфигураций запуска

Для теста EP удалось получить выигрыш от одновременного использования всех вычислительных ресурсов узла.

Заключение

Полученные к настоящему времени промежуточные результаты исследований позволяют утверждать, что на языке FDVMH можно создавать программы, которые могут эффективно выполняться на кластерах, использующих в узлах многоядерные универсальные процессоры, графические ускорители фирмы NVidia и сопроцессоры Intel Xeon Phi.

Указанные исследования, безусловно, будут продолжены.

В настоящее время ведутся работы по расширению языка FDVMH для поддержки вычислений с разреженными матрицами и неструктурными сетками. Кроме того, разработан язык CDVMH (директивное расширение языка Си) и планируется в следующем году завершить создание первой версии компилятора с этого языка. Выполнение указанных двух работ существенно расширит возможности эффективного применения модели DVMH и послужит новым импульсом к исследованию ее эффективности.

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.

Литература

1. Н.А. Коновалов, В.А. Крюков, С.Н. Михайлов, А.А. Погребцов. «Fortran DVM - язык разработки мобильных параллельных программ», Ж. "Программирование", № 1, 1995, с. 49-54.
2. 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.
3. Притула М. Н. Отображение DVMH-программ на кластеры с графическими процессорами. Диссертация на соискание ученой степени кандидата физико-математических наук. 2013 год. 105 с.
4. 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 URL: http://omega.sp.susu.ac.ru/books/conference/PaVT2014/full/032.pdf
5. NAS Parallel Benchmarks
URL: http://www.nas.nasa.gov/publications/npb.html (дата обращения 14.10.2014).
6. Pennycook S.J., Hammond S.D., Jarvis S.A., Mudalige G.R. Performance Analysis of a Hybrid MPI/CUDA Implementation of the NAS-LU Benchmark. ACM SIGMETRICS Performance Evaluation Review – Special issue on the 1st international workshop on performance modeling, benchmarking and simulation of high performance computing systems (PMBS 10). 2011. Vol. 38, Issue 4. P. 23–29.

7. Seo S., Jo G., Lee J. Performance Characterization of the NAS Parallel Benchmarks in OpenCL. 2011 IEEE International Symposium on. Workload Characterization (IISWC). 2011. P. 137–148.
8. Гибридный вычислительный кластер K-100 URL: http://www.kiam.ru/MVS/resourses/k100.html (дата обращения 14.10.2014).
9. Arunmoezhi Ramachandran, Jérôme Vienne, Rob F. Van der Wijngaart, Lars Koesterke, Ilya Sharapov: Performance Evaluation of NAS Parallel Benchmarks on Intel Xeon Phi. ICPP 2013: 736-743.