ИПМ им.М.В.Келдыша РАН • Электронная библиотека

Препринты ИПМ • Препринт № 31 за 2014 г.

Жуков В.Т., Краснов М.М., Новикова Н.Д., Феодоритова О.Б.

Параллельный многосеточный метод: сравнение эффективности на современных вычислительных архитектурах

Рекомендуемая форма библиографической ссылки: Параллельный многосеточный метод: сравнение эффективности на современных вычислительных архитектурах / В.Т.Жуков [и др.] // Препринты ИПМ им. М.В.Келдыша. 2014. № 31. 22 с. URL: http://library.keldysh.ru/preprint.asp?id=2014-31

О р д е н а Л е н и н а
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М.В.Келдыша
Р о с с и й с к о й а к а д е м и и н а у к

В.Т.Жуков, М.М.Краснов,
Н.Д.Новикова, О.Б.Феодоритова

Параллельный многосеточный метод: сравнение эффективности на современных вычислительных архитектурах

Москва — 2014

Жуков В.Т., Краснов М.М., Новикова Н.Д., Феодоритова О.Б.

Параллельный многосеточный метод: сравнение эффективности на современных вычислительных архитектурах

Целью работы является демонстрация на современных процессорных архитектурах работоспособности многосеточного метода, предназначенного для численного решения трехмерных параболических и эллиптических дифференциальных уравнений с существенно разномасштабными разрывными коэффициентами на неравномерных сетках. Показано, что компьютерный код обеспечивает решение сложных задач и сохраняет эффективность с ростом числа процессоров при функционировании программы на современных процессорных архитектурах: на многоядерных процессорах традиционной архитектуры, графических ускорителях Nvidia и сопроцессорах Xeon Phi архитектуры Intel MIC.

Ключевые слова: гибридные многопроцессорные системы**,** многосеточный метод, параллельная реализация

Victor Timofeevich Zhukov, Mikhail Mikhailovich Krasnov, Nataliya Dmitrievna Novikova, Olga Borisovna Feodoritova.

Parallel multigrid: effectiveness on modern numerical technologies

The goal of the work is to demonstrate the capability of multigrid method on the modern processors architectures. The method is used to numerical solution of 3D parabolic and elliptic differential equations with essential large-scale discontinuous coefficients on non-uniform grids. It is shown that the computer code solves complex problems and keeps effectiveness with increasing processors number on the modern processor architectures: on multi-cores processors of traditional architecture, graphics processors Nvidia and coprocessor Xeon Phi architecture Intel MIC.

Key words: hybrid multiproccessor system, multigrid, parallel implementation

Работа выполнена при поддержке Российского фонда фундаментальных исследований, проект 13-01-12079-офи_м и Программы № 25 фундаментальных исследований Президиума РАН

1. Введение

Целью работы является демонстрация работоспособности на современных процессорных архитектурах многосеточного алгоритма и компьютерного кода [1−4], предназначенного для численного решения трехмерных параболических и эллиптических дифференциальных уравнений с существенно разномасштабными разрывными коэффициентами на неравномерных сетках. Практическая необходимость в эффективном решении таких задач вызвана их широким распространением в математических моделях описания многомасштабных процессов в неоднородных областях с существенно различающимися свойствами. Многосеточный алгоритм изначально сконструирован для эффективной работы на суперкомпьютерах современных архитектур и обеспечивает масштабируемое моделирование на расчетных сетках с миллиардом (и более) узлов. Перспективная цель состоит в обеспечении успешной работы на машинах эксафлопсной производительности. Многосеточный алгоритм является методом сквозного счета и не требует априорного выделения поверхностей разрыва коэффициентов и согласования с этими поверхностями координатных сеточных плоскостей. В данной работе показано, что компьютерный код обеспечивает решение сложных задач и сохраняет эффективность с ростом числа процессоров при функционировании программы на современных процессорных архитектурах: на многоядерных процессорах традиционной архитектуры, графических ускорителях Nvidia и сопроцессорах Xeon Phi архитектуры Intel MIC.

Проверка работоспособности схем на современных архитектурах проводится в разных аспектах. Тенденции развития суперкомпьютеров на ближайшее пятилетие в основном определились. К концу 2013 года свыше 30 суперкомпьютеров преодолели петафлопсный барьер. Доля решений Intel среди всех суперкомпьютеров составляет более 80%. Теперь она растёт не только за счёт многоядерных процессоров традиционной архитектуры, но и за счёт ускорителей Xeon Phi архитектуры Intel MIC. Их использование стало новой тенденцией: уже тринадцать систем из рейтинга Топ-500 применяют их вместо ускорителей Nvidia и AMD. В 2013 году июньский и ноябрьский листы Топ-500 возглавляет Tianhe-2 («Млечный Путь») Оборонного университета Китая. Эта модель содержит 16 тысяч вычислительных узлов, в каждом из которых расположено по два процессора Intel Xeon E5-2692 и по три сопроцессора Intel Xeon Phi 31S1P. На каждый процессор выделяется по 32 ГБ оперативной памяти, а на каждый сопроцессор — по 8 ГБ памяти. Суммарный объём всех модулей памяти составляет тысячу терабайт. Отказ от ускорителей Nvidia в пользу Intel Xeon Phi объясняется тем, что хотя Intel MIC значительно уступает

Nvidia Kepler по теоретическим показателям производительности, объединять решения от одного разработчика гораздо проще.

Считается, что для ускорителя Xeon Phi можно написать более эффективный программный код без типичных для CUDA Nvidia промежуточных преобразований. Поэтому реальные научные программы, запускаемые на Xeon Phi, могут требовать меньше интеллектуальных ресурсов и затрат рабочего времени на перенос программ при сопоставимых временных затратах на их выполнение.

В разделе 2 описываются вычислительные системы, на которых производится тестирование алгоритмов. В разделе 3 приведены результаты тестовых расчетов. Кроме двух примеров, взятых из стандартного набора тестов, основное внимание уделено демонстрации сравнительной эффективности параллельного кода многосеточного метода [1−4], разработанного для решения эллиптических и параболических задач.

2. Вычислительные средства

Тестирование проведено на гибридной вычислительной системе К-100 ИПМ им. М.В. Келдыша РАН [5] и на сервере SuperMicro с установленным сопроцессором Intel Xeon Phi [6].

Суперкомпьютер К-100 состоит из 64 вычислительных узлов. На каждом узле установлены по два 6-ядерных процессора Intel Xeon X5670, т.е. задаче пользователя на одном узле доступно 12 ядер. Общая оперативная память узла составляет 96 Гб. Тактовая частота процессора Intel Xeon X5670 равна 2.93 ГГц. Каждый узел содержит по 3 графических ускорителя nVideo Fermi C2050 с 448 ядрами и собственной памятью в 2.5 Гб. Сразу отметим, что относительно небольшая оперативная память графического ускорителя является для наших задач серьезным ограничением. Доступные программные средства включают в себя системы распараллеливания вычислений OpenMP, MPI, CUDA, которые и представляют для нас интерес.

Вычислительный сервер SuperMicro с установленным сопроцессором Intel Xeon Phi 5100 (в дальнейшем будем для краткости писать сопроцессор Xeon Phi) имеет следующую конфигурацию. В качестве управляющей хост-машины используется 4-ядерный процессор Intel Xeon с 64 Гбайт оперативной памяти. Сопроцессор Xeon Phi имеет 60 ядер с тактовой частотой 1,1 ГГц. В каждом ядре могут выполняться 4 потока, всего 240 потоков, объём оперативной памяти сопроцессора Xeon Phi – 8 Гбайт. Сопроцессор использует архитектуру MIC и позволяет опираться на такие широко используемые среди вычислителей системы параллельного программирования как OpenMP и MPI.

Заметим, что тактовая частота каждого ядра сопроцессора Xeon Phi почти в три раза ниже (в 2.79 раза) тактовой частоты ядра базового процессора Xeon X5670 суперкомпьютера К-100, но практически совпадает

с тактовой частотой графического ускорителя NVIDIA, установленного на К-100.

Базовый процессор суперкомпьютера К-100 не является новым, но он относится к группе серверных процессоров, параметры которых достигли экстремальных значений достаточно давно и практически не меняются. Поэтому сравнение сопроцессора новой архитектуры Xeon Phi и серверного процессора Xeon X5670 вполне разумно.

3. Результаты тестирования

Будем использовать при тестировании популярные средства параллельных вычислений, такие как стандарт для многопоточного программирования с общей памятью OpenMP, технология программирования в среде с разделенной памятью MPI, различные гибридные варианты MPI+OpenMP, MPI+CUDA.

В качестве характеристик масштабируемости используем, во-первых, отношение времени выполнения на одном процессе t_1 к фактическому времени выполнения t_n $A(n) = \frac{t_1}{t_n}$, где t_n – время счета программы на n процессорах. Будем называть эту величину ускорением. Вторая характеристика близко связана с первой и оценивает эффективность программной реализации по формуле $E(n) = \frac{t_1}{t_n\, n}$.

Интересно сравнить производительность одного ядра сопроцессора Xeon Phi с одним ядром процессора Intel Xeon хост-машины, исследовать масштабируемость сопроцессора Xeon Phi при увеличении числа потоков (процессов), сравнить общую производительность сопроцессора Xeon Phi и ускорителя NVIDIA с использованием на этом ускорителе системы программирования CUDA (см [7]).

Для работы на сопроцессоре Xeon Phi существуют два режима: offload и native. В offload режиме программа запускается на управляющей машине и соответственно все данные тоже находятся на хост-машине, но в процессе счета могут быть обращения к сопроцессору Xeon Phi. При этом необходимые для вычислений данные копируются с хост-машины на сопроцессор, там с ними проводятся вычисления, затем результирующие данные копируются обратно на хост.

В native режиме программа копируется на сопроцессор Xeon Phi (с помощью программы scp копирования файлов между двумя компьютерами под управлением Linux или Unix) и запускается на сопроцессоре. Соответственно все данные располагаются в памяти сопроцессора. Все тесты, представленные ниже, запускались на сопроцессоре в native режиме. Здесь же приведем пример работы в режиме offload. В этой программе массив v копируется на MIC, а результат (переменная s) обратно на хост.

Компилировать такую программу нужно без опции «-mmic» и запускать на хосте.

```
#include<iostream>
int main(int argc, char *argv[])
{
        float v[10];
        for(int i = 0; i < 10; ++i)
                v[i] = i + 1;
        float s = 0.f;
        #pragma offload target(mic) in(v)
        {
                for (int i=0; i<10; ++i)
                        s += v[i];
        }
        std::cout << s << std::endl;
}
```

Тест 1. В качестве первого взят тест, предложенный компанией Intel. В этом простейшем тесте методом Монте Карло вычисляется число π. Описание алгоритма, а также текст программы на языке С можно найти на сайте компании Intel (см. [8]).

Особенность этого теста – отсутствие как интенсивного обмена с памятью так и больших массивов данных. Вычисления между ядрами Xeon Phi распределяются с использованием MPI инструкций. Количество параллельных процессов менялось от 1 до 240. В расчетах до 60 процессов включительно рост производительности являлся практически линейным, затем он полностью остановился, а при приближении к 240 процессам даже несколько замедлился. Приведём таблицу 1 результатов.

Таблица 1

Зависимость времени счета на сопроцессоре Intel Xeon Phi от числа активных процессов. Коммуникации с помощью MPI инструкций. Тест 1

Процессы	1	2	4	8	16	32	64	128	240
Время, с	2255	1130	566	280	141	70	35	24	28

Для сравнения расчет проведен на одном ядре хост-процессора Intel Xeon и время счета составило 224 сек, что в 10 быстрее, чем на одном ядре сопроцессора Intel Xeon Phi.

На рис. 1 и 2 показаны ускорение и эффективность как функции числа MPI процессов (зеленая линия) и идеальная производительность (красная линия).

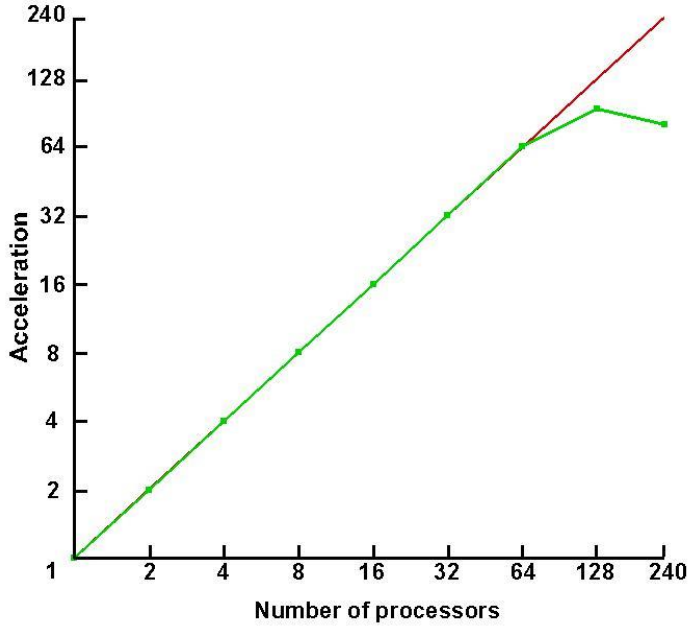

Рис. 1. Ускорение *A(n)* как функция числа MPI – процессов. Тест 1

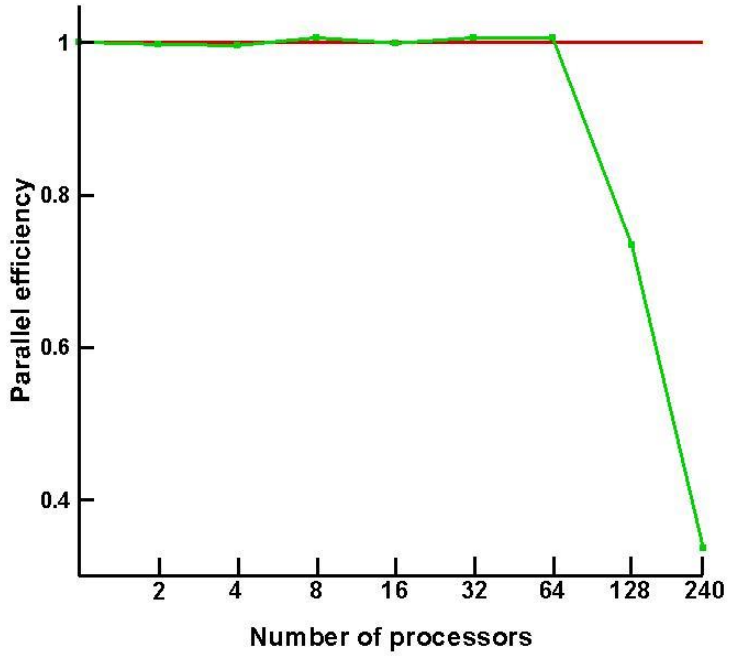

Рис. 2. Эффективность *E(n)* как функция числа MPI – процессов. Тест 1

Тест 2. Он представляет собой реализацию классического многосеточного метода и называется MG в сборнике тестов NAS Parallel Benchmarks (см. [9]). В тесте взят класс задачи B (сетки 256^3 узлов, число многосеточных итераций − 20). Это стандартный вариант многосеточного

метода для решения трехмерного уравнения Пуассона с периодическими краевыми условиями на равномерной декартовой сетке. Используются линейный оператор интерполирования, точечный метод Якоби в качестве сглаживающей процедуры и набор вложенных сеток с коэффициентом 2 прореживания узлов по каждому направлению. Операторы на каждой сетке строятся по формулам 27-точечной разностной аппроксимации.

Тест реализован с помощью операторной сеточной библиотеки *grid_math* [10]. Данная библиотека предназначена для упрощённой записи вычислений на языке C++ на трёхмерных индексных сетках. При ее написании ставились две цели. Первая – приблизить за счёт использования операторов внешний вид программ к формулам в теоретических работах, и вторая – скрыть внутреннюю реализацию библиотеки и за счёт этого максимально облегчить программирование для нетрадиционных архитектур, таких, как Nvidia CUDA и Intel Xeon Phi. В данной библиотеке «оператор» (например, оператор Лапласа) превращён из абстрактного теоретического понятия в конкретный программный объект. Например, если A, B и C – сеточные функции, а L – оператор Лапласа, то в программе можно написать такую строку: A = L (B + C). При этом в правой части оператора присваивания может быть выражение любой сложности, но вычисления будут запущены только один раз в операторе присваивания; таким образом, в библиотеке реализована концепция «ленивых» вычислений. Если программа, написанная с использованием данной библиотеки, скомпилировать «обычным» компилятором, то будет сгенерирован последовательный код, содержащий три вложенных цикла по трём координатам индексной сетки, если же программу скомпилировать с помощью компилятора nvcc для Nvidia CUDA, то оператор присваивания будет выполняться параллельно путём вызова ядра (kernel) в графическом процессоре. Данные при этом также будут располагаться в памяти графического процессора. Таким образом, отладив программу в последовательном режиме, её можно будет очень легко перенести на графический процессор (практически просто перекомпилировав другим компилятором). В последовательной версии библиотеки перед тремя вложенными циклами стоит прагма OpenMP (#pragma omp parallel for) (см. [11]), которая позволяет распараллелить эти циклы в том случае, если компилятор поддерживает эту прагму. В частности, именно таким образом осуществляется распараллеливание для Intel Xeon Phi. В настоящее время появилась новая версия данной библиотеки (matrix_math), работающая с двумерными матрицами вместо трёхмерных сеточных функций. В ней, например, есть встроенный оператор умножения матриц и оператор транспонирования, таким образом, в программе можно написать, например, такую строку: A=~B*C. Здесь A, B и C – матрицы, а ~ − оператор транспонирования матриц. Матрице A присваивается результат умножения транспонированной матрицы B на матрицу C.

Возвращаясь к тесту 2, отметим, что написаны последовательная, OpenMP и Nvidia CUDA версии этого теста. Для первых двух версий фактически исходный текст является общим и включает инструкции «pragma» OpenMP, просто последовательная версия компилируется без опции компилятора -openmp, а версия для OpenMP – с этой опцией. MPI – инструкции в этом тесте не используются. CUDA-версия реализована на суперкомпьютере К-100 на графических ускорителях Nvidia.

Последовательная версия – один поток на одном ядре сопроцессора Xeon Phi – считается 293,08 секунды, в то время как последовательная версия этого же теста на одном ядре хост-машины Intel Xeon выполняется 25 секунд и это практически в 12 раз быстрее, чем время работы на сопроцессоре. Для сравнения укажем, что такой же расчет на суперкомпьютере К-100 на одном графическом ускорителе NVIDIA Fermi C2050 (у которого на каждом ускорителе 448 GPU и 2,5 Гбайт памяти) занимает 2,52 секунды.

Результаты расчетов с помощью OpenMP–версии представлены в таблице 2. Графики ускорения и эффективности показаны на рис. 3–4.

Таблица 2

Зависимость времени счета на сопроцессоре Xeon Phi от числа активных процессов. Коммуникации с помощью OpenMP. Тест 2

Процессы	1	2	4	8	16	32	64	128	240
Время, с	584.8	298.2	150.7	78.31	40.65	23.42	15.67	14.36	15.17

Один из неожиданных результатов состоял в том, что версия для OpenMP на одном потоке работает (time=584.8 с) в два раза медленнее, чем просто последовательная версия (time=293.08 с). Объяснить это можно, по всей видимости, накладными расходами на распределение индексов циклов между потоками даже в том случае, если поток всего один.

На рис. 3 и 4 видно, что отклонение от идеальной прямой (красная линия), в отличие от предыдущего теста, начинается уже на восьми потоках, а на 60 потоках ускорение счета составляет всего 40 раз вместо 60.

Результаты всех тестов, начиная с третьего, относятся к параллельному коду многосеточного метода [1–4], показавшего хорошую работоспособность в сложных задачах, в том числе в расчетах реальных сложных течений на основе модели Навье-Стокса динамики несжимаемой среды. Именно об этом коде шла речь во введении.

Охарактеризуем очень коротко алгоритмические особенности предлагаемого метода. Алгоритм многосеточного метода использует чебышевские итерации при решении уравнений на самой грубой сетке, а также на этапах сглаживания.

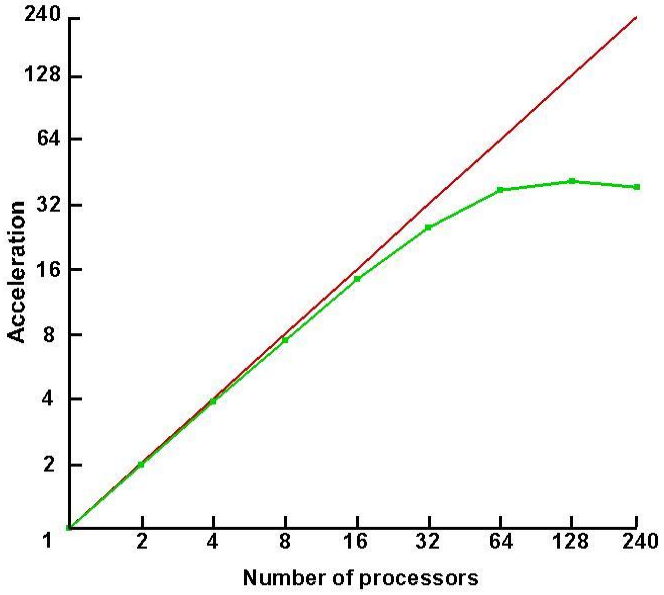

Рис. 3. Ускорение *A(n)* как функция числа активных процессов, версия OpenMP. Тест 2

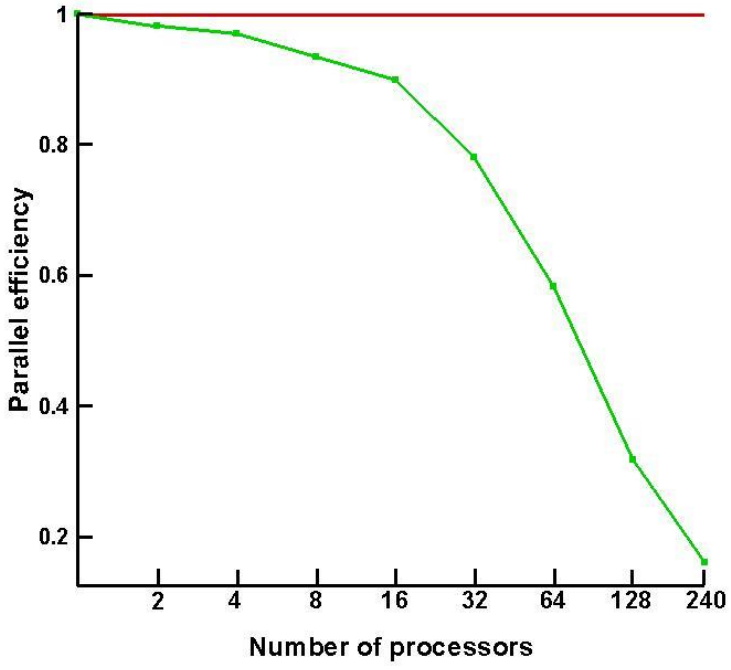

Рис. 4. Эффективность *E(n)* как функция числа активных процессов, OpenMP. Тест 2

По указанию в качестве сглаживателя может использоваться и специальная схема ЛИ-М в модификации, обеспечивающей сглаживание невязки в интервале частот, определяемом двумя спектральными

параметрами: условной границей раздела спектра на низкочастотную и высокочастотную части и оценкой верхней границы спектра. Сам процесс многосеточных итераций обеспечивает информацию для проведения коррекции первого из двух спектральных параметров. Выбранные системно-алгоритмические решения обеспечивают эффективную параллельную реализацию многосеточного метода. Помимо стандартного оператора трилинейной интерполяции грубосеточных переменных используются операторы межсеточных переходов в проблемно-зависимой форме, что обеспечивает работоспособность многосеточного метода для случая разрывных коэффициентов диффузии.

Тест 3. Сравним эффективность решения эллиптического уравнения с постоянными коэффициентами в прямоугольном параллелепипеде $G = [-22{:}22]\times[-22{:}22]\times[-500:6280]$

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2}=f(r),\quad r\in G\,.$$

На верхней грани расчетной области задано условие Дирихле, а на всех остальных гранях – условие Неймана. Правая часть $f(r)$ и граничные значения определены по точному решению $u=x^2+y^2$.

Расчетная сетка выбрана равномерной по координатным направлениям OX и OY. По оси OZ сетка разбита на два подмножества со сгущением к точке z= 0. Минимальный шаг в этой точке равен 0.2. Общее число узлов сетки равно $64\times 64\times 4096\approx 1.7\times 10^7$.

Задача решалась многосеточным методом с тремя сеточными уровнями до уменьшения нормы начальной невязки в $\varepsilon^{-1}=10^7$ раз, т.е. с относительной точностью ε=10^{-7}; заданная точность достигнута за 11 итераций. В качестве сглаживающей процедуры использован чебышевский сглаживатель. С помощью чебышевских итераций (но с другим набором чебышевских итерационных параметров) решается и уравнение на самой грубой сетке с относительной точностью ε_{coarse}=10^{-3}. Операторы межсеточных переходов простейшие: оператор интерполяции линейный, оператор сборки ему сопряжен.

Цель данного теста состоит в сравнение эффективности выполнения многосеточного параллельного кода в варианте с сеточной библиотекой *grid_math* [10] на вычислительной системе К-100 и на ускорителе Xeon Phi архитектуры Intel MIC.

В расчетах выбрана процессорная топология – "линейка" процессоров в z-направлении. Проверялась так называемая сильная масштабируемость, т.е. полный объем обрабатываемых данных (определяемый сеткой) оставался неизменным при увеличении числа участвующих в вычислениях процессов, объем данных в каждом процессоре (ядре) уменьшался пропорционально их

количеству. На вычислительной системе К-100 расчет проводился с опцией ppn=4, т.е. на каждом узле работали 4 ядра.

Как известно, сопроцессоры Xeon Phi позволяют использовать привычные языки программирования (C, C++, Fortran) и параллельные вычислительные средства OpenMP, MPI и пр. Это дает возможность пользоваться уже имеющимся параллельным кодом без дополнительного изучения моделей программирования, привязанных к определенному аппаратному обеспечению. В данном тесте код, написанный на языке C++ и использующий директивы MPI, без каких-либо изменений был запущен на сопроцессоре Xeon Phi. Расчет на Xeon Phi занял в ~11 раз большее время, чем расчет на кластере К100 (см. табл. 3).

Таблица 3

Сравнение эффективности параллельного многосеточного кода, выполненного в MPI-технологии. MIC vs. K-100. Тест 3

Процессорная сетка	MPI, MIC *t*, с	MPI, K-100, ppn=4 *t*, с
1 × 1 × 1	30190	2667
1 × 1 × 2	15480	1253
1 × 1 × 4	7867	714.9
1 × 1 × 8	4023	342.8
1 × 1 × 16	2146	232.2
1 × 1 × 32	1159	120.2
1 × 1 × 64	740.4	65.95
1 × 1 × 128	552.1	36.77

Вспомним, что тактовая частота сопроцессора Xeon Phi (1.053 Ггц) почти в 3 раза меньше тактовой частоты процессора Intel Xeon X5670 (2.93 Ггц), используемого на К-100 в качестве базового. Это, однако, не может объяснить одиннадцатикратное отставание по времени. Обратимся дополнительно к информации об архитектуре Intel Xeon Phi (см. [12]). На странице 23 мы находим указание на то, что в одном ядре на двух последовательных тактах не могут выбираться инструкции одного и того же потока, для полной загрузки ядра необходимо выполнять на нём по крайней мере два потока одновременно. Это означает, что если поток всего один, то он выполняется на половинной тактовой частоте.

При этом эффективность параллельного кода на сопроцессоре Xeon Phi сравнима с эффективностью кода на К-100, по крайней мере, в пределах 60 процессоров (ядер) и является вполне удовлетворительной, что видно из рис. 5 и 6, где представлены характеристики масштабируемости.

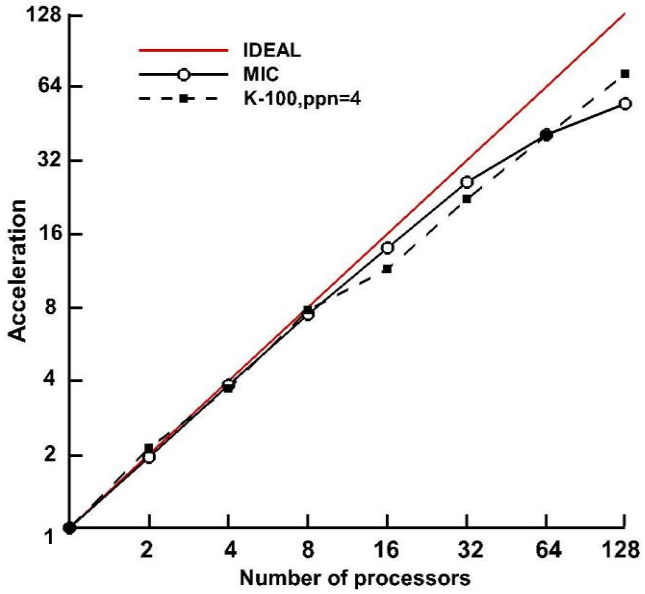

Рис. 5. Ускорение *A(n)* как функция числа активных процессов, MPI. Тест 3

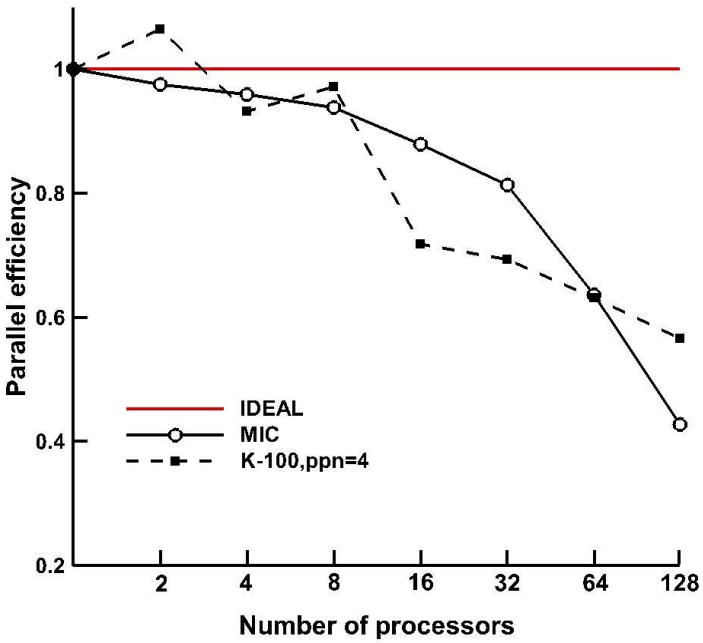

Рис. 6. Эффективность *E(n)* как функция числа активных процессов, MPI. Тест 3

Тест 4. Математическая формулировка задачи, выбранной для теста 4, отличается от задачи предыдущего теста только краевым условием – это задача Дирихле. Вычислительная постановка осталась без изменений.

Цель данного теста состоит в сравнении эффективности выполнения многосеточного параллельного кода в библиотечном варианте *grid_math* на вычислительной системе К-100 как с использованием графического ускорителя, так и без него, а также на ускорителе Xeon Phi архитектуры Intel MIC. При этом на ускорителе Xeon Phi рассматриваются два варианта. Первый вариант основан на технологии MPI, в то время как второй вариант использует директивы OpenMP. В случае многопоточного подхода OpenMP распределение вычислений по нитям касается только трехмерных циклов, входящих в итерационную процедуру решения уравнений на самой грубой сетке, процедуры сглаживания, интерполяции и сборки. Следует заметить, что процедура сглаживания является наиболее трудоемкой частью используемого алгоритма, особенно для задач с сильной анизотропией или задач с разрывными коэффициентами (свыше 90% всего времени счета).

Таблица 4

Сравнение эффективности многосеточного метода для различных параллельных систем. Тест 4

Процессорная сетка	CUDA+MPI K-100, ppn=3 t, с	MPI K-100, ppn=3 t, с	MPI MIC $t_{MPI\ MIC}$, с	OpenMP MIC $t_{OpenMP\ MIC}$, с
1 × 1 × 1	317.12	2129.98	52085.83	92632.49
1 × 1 × 2	204.91	998.68	26561.84	49221.61
1 × 1 × 4	153.78	549.2	13409.40	27030.82
1 × 1 × 8	135.46	250.98	6863.35	15615.75
1 × 1 × 16	120.87	135.48	3613.32	9836.28
1 × 1 × 32	125.34	71.78	1951.71	6864.12
1 × 1 × 64	134.47	51.56	1523.34	6380.29
1 × 1 × 128	131.52	33.38	1193.53	6042.12

Стратегия параллельного счета не отличалась от предыдущего случая (см. тест 3) – вычислительная задача отображается на линейку процессоров. Проверяется сильная масштабируемость.

Заметим также, что, так как в состав каждого вычислительного узла К-100 входят 3 графических ускорителя, запуск тестов, как для варианта с CUDA+MPI, так и для варианта, в котором используется только MPI, производился с ключом ppn=3. Это означает, что на каждом узле берется только 3 ядра.

Результаты сравнения сведены в табл. 4 и дополнительно представлены на рис. 7, 8 . Первый столбец в таблице "Процессорная сетка" для случая "OpenMP MIC" указывает число используемых потоков, например, $1 \times 1 \times 16$ означает 16 потоков.

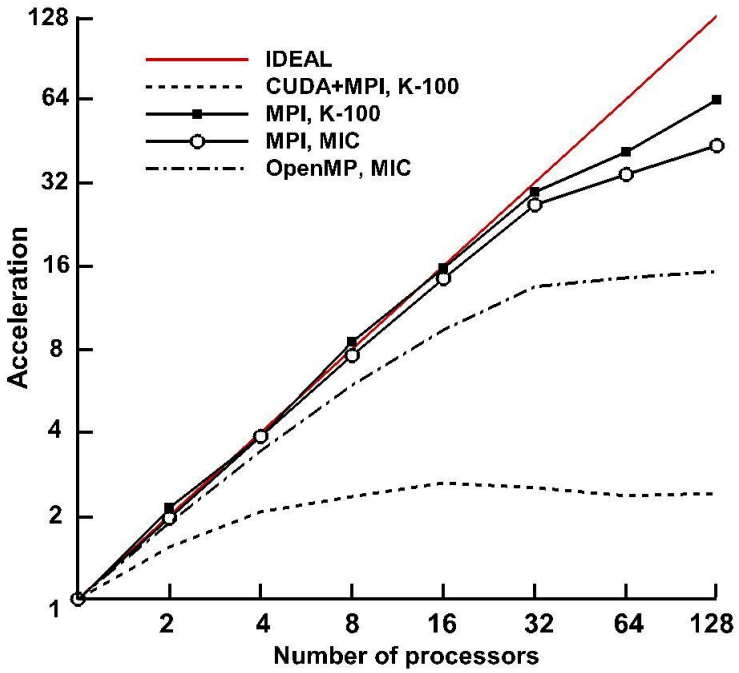

Рис. 7. Ускорение *A(n)* как функция числа активных процессов. Тест 4

Как и в предыдущем тесте, вариант "MPI MIC" показывает близкую с вариантом "MPI K-100" масштабируемость в пределах 60 процессоров (ядер), существенно отставая по времени исполнения. Что касается варианта "OpenMP MIC", то расчет довольно быстро выходит на асимптотику, что связано с наличием последовательных участков в программе. Этот эффект ожидаем (распараллеливание по потокам коснулось только операторов сглаживания, интерполирования и сборки) и согласуется с законом Амдала

$$A(n) \le \frac{1}{s + (1-s)n^{-1}}.$$

Здесь s − суммарная доля последовательных блоков в параллельной программе, n − число процессов. Тот факт, что вариант расчета в комбинации «OpenMP MIC» занимает больше времени, чем расчет «MPI

MIC» можно объяснить наличием последовательных блоков в программе. Добиться лучших характеристик счета можно, по-видимому, тщательной переработкой алгоритма. Неожиданным является факт значительной разницы счета последовательного варианта (процессорная сетка $1 \times 1 \times 1$) в режимах MPI MIC и OpenMP MIC: $t_{MPI\ \ MIC}$=52085.83 и $t_{OpenMP\ \ MIC}$=92632.49, $\frac{t_{OpenMP\ \ MIC}}{t_{MPI\ \ MIC}} = 1.8$.

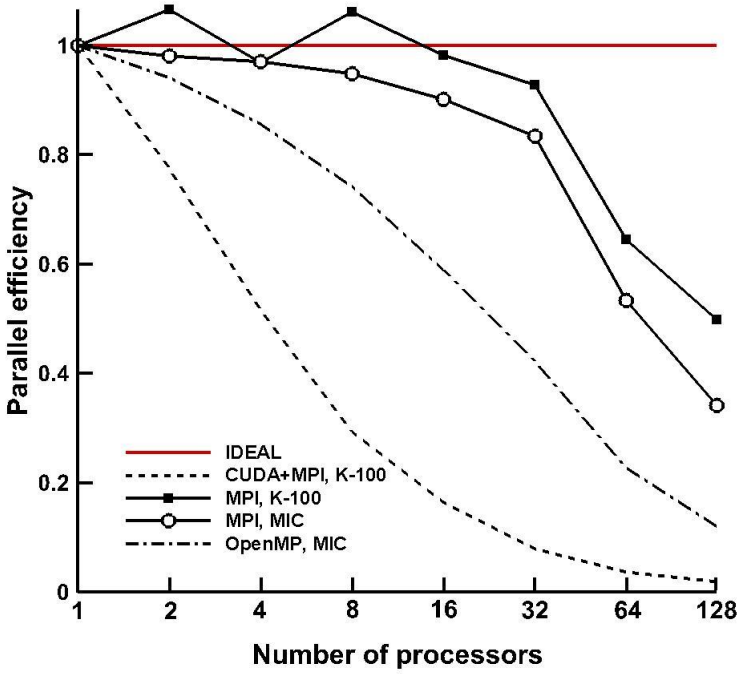

Рис. 8. Параллельная эффективность *E(n)* как функция числа активных процессов. Тест 4.

Тест 5. Исследуем слабую масштабируемость. В этом тесте решаем задачу Дирихле для уравнения Пуассона. Рассмотрим куб $G = [0{:}L]^3$ с длиной ребра L и введем равномерную сетку с числом узлов N по каждому координатному направлению. Ограничимся тремя сетками с $N = 64, 128, 256$, так что самая подробная сетка содержит $N^3 \approx 1.7\ 10^7$ узлов. Одновременно с увеличением числа процессоров будем увеличивать число узлов сетки N^3 таким образом, чтобы объем данных, обрабатываемых в каждом процессоре оставался неизменным. Пропорционально будем увеличивать длину ребра L, чтобы сохранить величину шага $h = L/N$ и обеспечить одинаковую обусловленность задач на разных сетках; тогда все процедуры многосеточного метода при фиксированном числе сеточных уровней будут иметь одинаковую вычислительную сложность.

Задача решается многосеточным методом с тремя многосеточными уровнями с заданной точностью $\varepsilon = 10^{-7}$. Чебышевские итерации используются как для сглаживания, так и для решения линейной системы на самом грубом уровне. Точность решения грубосеточной линейной системы $\varepsilon_{coarse} = 10^{-3}$. Оператор интерполяции − проблемно-зависимый. По вычислительным затратам и параллельной эффективности он практически такой же как оператор линейной интерполяции. В указанной задаче оба оператора дают тождественный результат, что дополнительно служит проверкой корректности реализации достаточно сложной ветви кода построения проблемно-зависимого оператора интерполяции и ему сопряженного. Полное число многосеточных итераций и времена счета приведены в табл. 5. В идеальном варианте при условии, что временем межпроцессорных обменов можно пренебречь, время счета должно оставаться неизменным. В данных, представленных в табл. 5, время счета растет, и полного понимания причин такого роста пока нет.

Таблица 5

Сравнение эффективности параллельного многосеточного метода на параллельных системах MPI MIC (сопроцессор Intel Xeon Phi) и MPI (K-100, ppn=3)

Процессорная сетка	Расчетная сетка	Число MG итераций	MPI MIC $t_{MPI\ MIC}$, с	MPI K100 $t_{MPI\ K\text{-}100}$, с	Ускорение, $\frac{t_{MPI\ MIC}}{t_{MPI\ K-100}}$, с
$1 \times 1 \times 1$	$64 \times 64 \times 64$	9	6.09	0.41	14.85
$2 \times 2 \times 2$	$128 \times 128 \times 128$	9	6.74	0.62	10.87
$4 \times 4 \times 4$	$256 \times 256 \times 256$	9	11.69	1.15	10.17

Тест 6. Вернемся к постановке задачи, описанной в тесте 3. Рассмотрим решение уравнения Пуассона в прямоугольном параллелепипеде $G = [-22{:}22] \times [-22{:}22] \times [-500 : 6280]$ с комбинированными краевыми условиями: на верхней грани расчетной области задано условие Дирихле, а на всех остальных гранях − условие Неймана. Правая часть $f(r)$ и граничные значения определены по точному решению $u = x^2 + y^2$. Вычислительная постановка задачи также совпадает с тестом 3 за исключением размера задачи. Полное число расчетных точек равнялось $32 \times 32 \times 2048 \approx 2 \times 10^6$.

Рассмотрим, к чему приводит использование комбинации MPI+OpenMP. В этом варианте вычислительная задача разбивается на отдельные MPI-процессы, внутри каждого из которых часть работы

распараллеливается по нитям с доступом к общей памяти с использованием OpenMP-инструкций.

Таблица 6

Время счета параллельного варианта многосеточного метода на сопроцессоре Xeon Phi с использованием средств MPI+OpenMP

Число нитей OpenMP	Число MPI процессов			
	1	16	32	64
1	4359.14 с	307.52 с	167.40 с	139.03 с
2	2361.34 с	166.95 с	133.01 с	107.39 с
4	1315.46 с	142.32 с	109.16 с	135.54 с

Таблица 7

Время счета параллельного варианта многосеточного метода на суперкомпьютере К-100 (ppn=3) с использованием средств MPI+OpenMP

Число нитей OpenMP	Число MPI процессов			
	1	16	32	64
1	315.60 с	23.71 с	13.53 с	9.21 с
2	169.13 с	14.59 с	9.07 с	6.77 с
4	94.10 с	9.63 с	6.60 с	5.34 с

Расчеты на К-100 проводились с опцией «ppn=3», что позволяет при создании четырех потоков (это максимальное заказанное число в экспериментах) остаться в рамках одного узла. Заметим, что на К-100 каждое ядро содержит две нити, но они насильственно отключены, поэтому включение нитей на некоторых участках вычислительной работы просто приводит к активизации дополнительных ядер.

На сопроцессоре Xeon Phi в случае, если произведение числа MPI-процессов на число OpenMP-потоков превышает число ядер ($\geq$ 60), эффективность кода падает. Это вполне согласуется с уже представленными выше результатами в предыдущих тестах.

В дополнении приведем совсем свежие сравнительные результаты тестирования многосеточного алгоритма MG из сборника тестов NAS

Parallel Benchmark (уже использованного в тесте 2) на макете вычислительного узла, в котором установлены четыре 12-ти ядерных процессора Intel E7-4860 v2 (см. [13]) с тактовой частотой 2.6 Ггц (3.2.Ггц в режиме boost). Каждый процессор допускает 24 потока, пропускная способность памяти до 85 ГБ/с; кэш–память 50 Мб, максимальный объем памяти до 1536 Гб, максимальная электрическая мощность 130 ватт. В первом эксперименте решается уравнение Пуассона на сетке $512 \times 512 \times 512$ и делается 20 многосеточных итераций.

Таблица 8

Время счета MG-теста суперкомпьютере К-100 и макете

К-100, 1 процесс	$t_{K100(1)}$ = 243.6 секунды
К-100, 12 процессов (через boost)	$t_{K100(12)}$ = 27.7 с $\left(\frac{t_{K100(1)}}{t_{K100(12)}} = 8.8\right)$
Макет 1 процесс	$t_{m(1)}$ = 174.7 с $\left(\frac{t_{K100(1)}}{t_{m(1)}} = 1.4\right)$
Макет 96 процессов (через boost)	$t_{m(96)}$ = 14.88 с $\left(\frac{t_{m(1)}}{t_{m(96)}} = 11.7\right)$
Макет 96 процессов (через OpenMP)	$t_{m(96)\_OpenMP}$ =19.8с $\left(\frac{t_{m(1)}}{t_{m(96)\_OpenMP}} = 8.8\right)$

Во втором эксперименте решается та же задача на большей сетке $1024 \times 1024 \times 1024$ за 50 многосеточных итераций.

Таблица 9

Время счета MG-теста на суперкомпьютере К-100 и макете

К-100 / 12 процессов (через boost)	t = 547.9 с
Макет / 96 процессов (через boost)	t = 261 с
Макет / 96 процессов (через OpenMP)	t = 288.8 с

Базовый элемент макета – процессор Intel E7-4860 v2 (на 1 потоке) примерно в 1.4 раза быстрее, чем на К-100, что неплохо. Интересно, что использование всех возможных потоков (96) ускоряет вычисления примерно в 12 раз. Для сравнения: на К-100 использование 12 потоков даёт ускорение примерно в 8-9 раз. Заметим, что есть еще процессор 8893 из той же линейки (самый быстрый и самый дорогой), но он нам пока недоступен.

4. Заключение

Общие результаты тестирования следующие:

− Последовательная версия тестов (один поток на одном ядре) на сопроцессоре Xeon Phi примерно в 11 раз медленнее, чем на обычном серверном процессоре Intel Xeon.

− Задачи, в которых ведутся интенсивные вычисления, относительно хорошо масштабируются на Intel Xeon Phi при увеличении числа потоков до 60, после этого рост производительности практически останавливается, а при дальнейшем увеличении числа потоков даже немного уменьшается, т.е. на 240 потоках задача может считаться чуть медленнее, чем на 60 потоках. Этот эффект объясняется тем, что технология hyperthreading даёт реальный эффект только тогда, когда у потоков есть простои (например, на ввод-вывод). Если простоев нет, то фактически потоки на одном ядре считаются в режиме разделения времени.

− Масштабирование задачи (до 60 потоков) зависит от интенсивности обмена с оперативной памятью. Если обмен с памятью практически отсутствует (как в первом тесте), то масштабирование близко к идеальному, т.е. при 60 потоках задача может считаться примерно в 60 раз быстрее, чем при одном потоке. Если же ведётся интенсивный обмен с памятью (как во втором тесте), то масштабирование существенно хуже (во втором тесте на 60 потоках всего в 40 раз быстрее, чем на одном потоке). Объясняется это тем, что при интенсивном обмене с памятью узким местом становится шина ввода-вывода, которая не справляется со столь интенсивным обменом данными, и потокам приходится ждать завершения обмена данными по шине.

− Тактовая частота базовых процессоров суперкомпьютера К-100 равна 2.93Ггц. В то же время, тактовая частота сопроцессора Intel Xeon Phi – 1.053 Ггц, что почти в три раза меньше. И это одна из причин более скромных результатов, демонстрируемых на вычислительном комплексе Intel Xeon.

− Уже существующий программный код, написанный, например, с использованием MPI-инструкций, можно автоматически перевести на Intel Xeon Phi. Однако трудно надеяться на получение хорошей эффективности параллельной реализации без дополнительных усилий по оптимизации вычислений. По крайней мере, на нынешнем этапе скоростей вычислений и обменов с памятью.

Список литературы

1. Жуков В.Т., Новикова Н.Д., Феодоритова О.Б. Параллельный многосеточный метод для разностных эллиптических уравнений. Часть I. Основные элементы алгоритма // Препринты ИПМ им. М.В.Келдыша. 2012. № 30. 32 с. URL: http://library.keldysh.ru/preprint.asp?id=2012-30

2. Жуков В.Т., Новикова Н.Д., Феодоритова О.Б. Параллельный многосеточный метод для разностных эллиптических уравнений. Анизотропная диффузия // Препринты ИПМ им. М.В.Келдыша. 2012. № 76. 36 с. URL: http://library.keldysh.ru/preprint.asp?id=2012-76

3. Жуков В.Т., Новикова Н.Д., Феодоритова О.Б. Параллельный многосеточный метод для разностных эллиптических уравнений уравнений // Матем. моделирование, 2014, т.26, № 1, с. 55−68.

4. Жуков В.Т., Новикова Н.Д., Феодоритова О.Б. Многосеточный метод для анизотропных уравнений диффузии на основе адаптации сглаживателей // Матем. моделирование, 2014, в печати.

5. Гибридный вычислительный кластер K-100 URL: http://www.kiam.ru/MVS/ resourses/k100.html

6. Intel Xeon Phi. URL: http://www.intel.com/content/www/us/en/processors/ xeon/ xeon-phi-detail.html

7. Nvidia CUDA. URL: http://www.nvidia.com/object/cuda_home_new.html

8. Using the Intel MPI Library on Intel Xeon Phi Coprocessor Systems https://software.intel.com/articles/using-the-intel-mpi-library-on-intel-xeon-phi-coprocessor-systems

9. NAS Parallel Benchmarks. URL: https://www.nas.nasa.gov/publications/npb.html

10. Краснов М.М., Феодоритова О.Б. Операторная библиотека для решения трёхмерных сеточных задач математической физики с использованием графических плат с архитектурой CUDA // Препринты ИПМ им. М.В.Келдыша. 2013. № 9. 32 с. URL: http://library.keldysh.ru/preprint.asp?id=2013-9

11. OpenMP. URL: http://www.openmp.org

12. Линев А.В. Программирование для Intel Xeon Phi http://itprojects.narfu.ru/grid/material2014/2014-GRID5-XeonPhi-Architecture.pdf

13. Intel Xeon Processor E7-4860 v2 http://ark.intel.com/ru/products/75249/Intel-Xeon-Processor-E7-4860-v2-30M-Cache-2_60-GHz

Оглавление