Оптимизация для кластера с ускорителями Xeon Phi задачи фильтрации водно-нефтяной смеси через эластичную пористую среду

С.Е. Киреев

Институт вычислительной математики и математической геофизики СО РАН,
Новосибирский государственный университет

На основе разработанного ранее программного комплекса для моделирования многофазных потоков в деформируемой пористой среде был реализован новый параллельный программный комплекс, оптимизированный для запуска на кластере с ускорителями Intel Xeon Phi. Рассмотрены различные способы оптимизации, специфичные для данного ускорителя, и их влияние на время работы программы. Выполнено сравнение различных способов использования ускорителей в составе кластера: симметричного режима и режима offload. Получены оценки ускорения и эффективности при использовании различного числа узлов кластера.

1. Введение

В работах [1, 2] была представлена модель фильтрации многофазной жидкости через деформируемый пористый каркас, и предложен метод, основанный на алгоритме WENO-Рунге-Кутты. На основе этих результатов авторами была создана двумерная реализация данной модели для двух жидкостей для моделирования просачивания водно-нефтяной смеси через пористую породу. В рамках дальнейшей работы задача была реализована с использованием метода Рунге-Кутты 5-го порядка точности по времени. Настоящая работа посвящена параллельной реализации и оптимизации данной задачи для кластера с ускорителями Intel Xeon Phi.

Статья имеет следующую структуру. Второй раздел содержит описание задачи: приводится решаемая система уравнений, используемые методы ее решения и схема численного алгоритма. Перечислены параметры целевой вычислительной системы. Третий раздел содержит описание используемых способов оптимизации и распараллеливания программы. Приведены оценки ускорения, достигнутого на каждом этапе оптимизации и распараллеливания. Выполнено сравнение двух режимов использования сопроцессоров Xeon Phi: симметричного и offload. В заключении приводятся основные результаты работы.

2. Описание задачи

2.1 Численная модель

Система уравнений движения смеси жидкостей через эластичную среду, представленная в [1], выглядит следующим образом:

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u_k}{\partial x_k} = 0,$$

$$\frac{\partial \rho \alpha_n}{\partial t} + \partial_k(\rho \alpha_n u_k) = -\sum_{m=2}^{N} \lambda_{nm}(p_1 - p_m), n = 2, \dots, N,$$

$$\frac{\partial \alpha_n \rho_n}{\partial t} + \partial_k(\alpha_n \rho_n u_k^n) = 0, n = 2, \dots, N,$$

$$\frac{\partial w_k^n}{\partial t} + \partial_k\left(\frac{1}{2}u_i^1 u_i^1 - \frac{1}{2}u_i^n u_i^n + e^1 + \frac{p_1}{\rho_1} - e^n - \frac{p_n}{\rho_n}\right) = e_{kij} u_i \omega_j^n - \sum_{m=2}^{N} \chi_{nm} c_m (u_k - u_k^m),$$

$$\frac{\partial(\rho u_i)}{\partial t} + \partial_k\left(\sum_{n=1}^{N}\alpha_n\rho_n u_i^n u_k^n + \sum_{n=1}^{N}\alpha_n\rho_n - \alpha_1\sigma_{ik}\right) = 0,$$

$$\frac{\partial\rho F_{ij}}{\partial t} + \partial_k\left(\rho F_{ij}u_k - \rho F_{kj}u_i\right) = 0,$$

$$\frac{\partial\rho s}{\partial t} + \partial_k(\rho s u_k) = \frac{1}{T}R,$$

где диссипативная функция:

$$R = \sum_{n,m=2}^{N}\chi_{nm}(u_i - u_i^n)(u_i - u_i^m) + \frac{1}{\rho^2}\sum_{n,m=2}^{N}\lambda_{nm}(p_1 - p_n)(p_1 - p_m).$$

Здесь α_1 – объемная концентрация эластичного каркаса (фаза 1), α_n – объемные концентрации жидкостей (фаза *n*, *n = 2,...,N*, где *N-1* – число жидкостей в смеси, просачивающейся через пористую среду), причем $\alpha_1 + \sum_{n=2}^{N}\alpha_n = 1$. c_1 – массовая концентрация эластичного каркаса (фаза 1), c_n – массовые концентрации жидкостей (фаза n), причем $c_1 + \sum_{n=2}^{N} c_n = 1$. ρ_n – массовая плотность фазы *n*, $\rho = \sum_{n=1}^{N}\alpha_n\rho_n$ – массовая плотность *N*-фазной среды. u_i^n – скорость фазы *n*, *n = 1,…,N*; $w_i^n = u_i^1 - u_i^n$ – относительная скорость жидкой фазы n по отношению к пористой среде. $u_i = \sum_{n=1}^{N} c_n u_i^n$ – скорость *N*-фазной среды, где $c_n = \frac{\alpha_n\rho_n}{\rho}$. F_{ij} – компоненты тензора градиента деформации всей N-фазной среды. s – массовая плотность энтропии всей N-фазной среды. $p_n = \rho_n^2 e_{\rho_n}^n$ – фазовые плотности ($p = \sum_{n=1}^{N}\alpha_n\rho_n$). $\sigma_{ik} = \rho_1 F_{kj} e_{F_{kj}}^1$ – напряжения сдвига. T – температура. Данная система дополнена условием согласованности в форме законов сохранения:

$$e_{ikj}\omega_j^n = \partial_i w_k^n - \partial_k w_i^n, \qquad \frac{\partial\omega_k^n}{\partial t} + \partial_j\left(u_j\omega_k^n - u_k\omega_j^n + \sum_{m=2}^{N} e_{kji}\chi_{nm}E_{w_i^m}\right) = 0, \qquad \partial_k\left(\rho F_{kj}\right) = 0.$$

Закон сохранения энергии:

$$\frac{\partial}{\partial t}\left(\rho\left(E + \frac{1}{2}u_i u_i\right)\right) + \partial_k\left(\sum_{n=1}^{N}\alpha_n\rho_n u_k^n\left(e^n + \frac{1}{2}u_i^n u_i^n + \frac{1}{\rho_n}p_n\right) - u_i\alpha_1\sigma_{ik}\right) = 0.$$

Для решения задачи в двумерном случае система уравнений была представлена в векторной форме [1]:

$$\frac{\partial U}{\partial t} + \frac{\partial F(U)}{\partial x} + \frac{\partial G(U)}{\partial y} = S(U),$$

где U – вектор искомых консервативных переменных, $F(U)$ и $G(U)$ – вектора потоков по направлениям *x* и *y*, $S(U)$ – вектор правых частей.

В данной работе рассматривается решение системы на прямоугольной сетке методом WENO-Рунге-Кутты [1, 3], в котором вычисление потоков через грани ячеек проводится с использованием полиномиального приближения данных внутри ячеек, что обеспечивает 5-й порядок точности по пространству, а для интегрирования уравнений по времени используется метод Рунге-Кутты 5-го порядка точности по времени. Величина каждого очередного шага по времени вычисляется исходя из условия Куранта на основе текущей максимальной скорости звука.

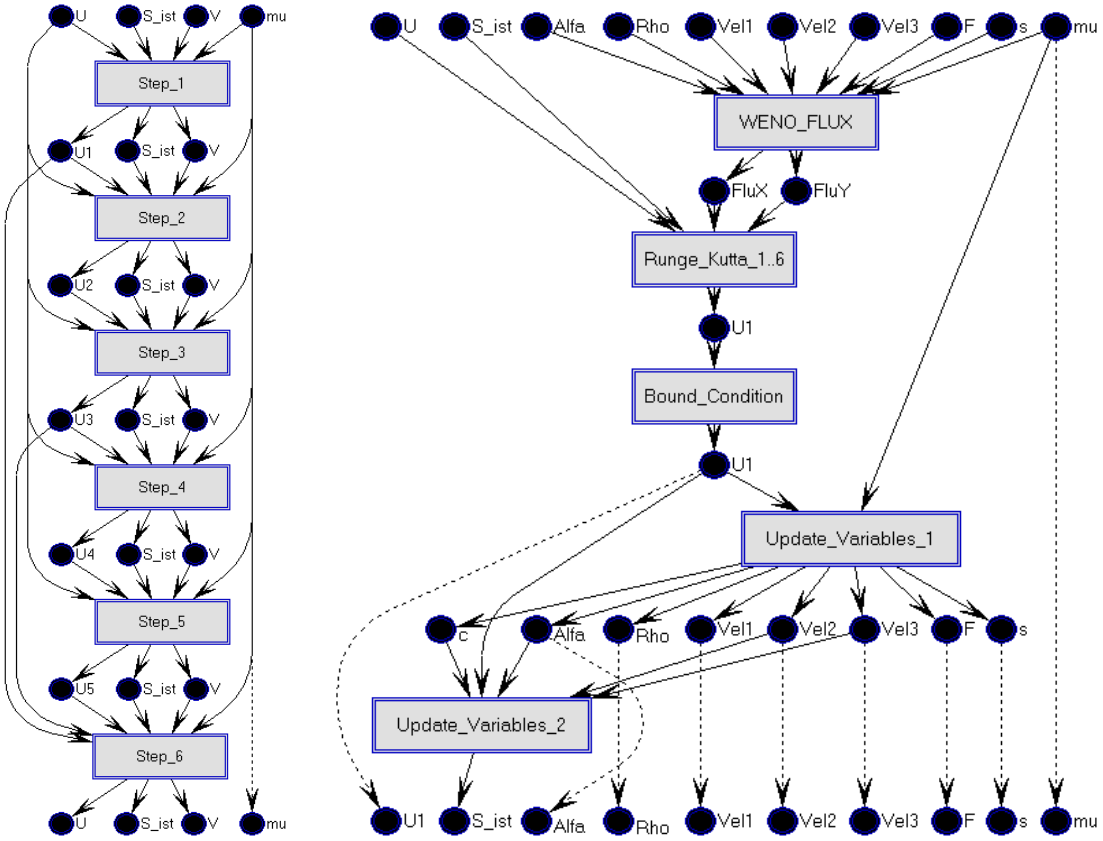
2.2 Схема численного алгоритма

На рис. 1а представлена схема численного алгоритма для одного шага по времени. Схема представлена в виде графа зависимостей с переменными единственного присваивания (черные кружки) и операциями однократного срабатывания (серые прямоугольники). Каждая переменная представляет собой одну или несколько пространственно распределенных характеристик среды, реализуемых в программе как двумерные массивы. Операции отражают порядок вычисления одних массивов из других. Сплошные стрелки обозначают информационные зависимости, штриховые стрелки обозначают тождественность переменных, которые они соединяют.

Шесть операций Step_1…Step_6 соответствуют шести стадиям вычислений в методе Рунге-Кутты 5-го порядка точности. Вычисления на всех шести стадиях почти полностью совпадают и могут быть вынесены в подпрограмму, схема которой представлена на рис. 1б. Переменные

на данной схеме также представляют собой пространственно распределенные характеристики среды, реализуемые в программе двумерными массивами, а каждая из операций реализуется как несколько вложенных циклов (гнездо циклов), выполняющих обход по ячейкам пространственной сетки и вычисляющих значения выходных характеристик для данной ячейки. Операция WENO_Flux вычисляет значения потоков через границы ячеек из значений примитивных переменных. Операция Runge-Kutta_1..6 вычисляет новые значения консервативных переменных на основе значений потоков по формулам, специфичным для данного этапа метода Рунге-Кутты. Операция Bound_Condition обновляет значения консервативных переменных на границах области. Операции Update_Varibles_1 и Update_Varibles_2 вычисляют значения примитивных переменных из консервативных.

Особенностью алгоритма является то, что все вычисления внутри ячеек в пределах каждой отдельной операции алгоритма (гнезда циклов) являются независимыми друг от друга, хотя и используют значения из соседних ячеек. Это позволяет легко распараллелить данный алгоритм методом декомпозиции пространства моделирования. Анализ графа на рис. 1б позволяет выявить наиболее оптимальный этап алгоритма, на котором эффективнее всего будет выполнить обмен граничными значениями в случае декомпозиции пространства.



(а) (б)

Рис. 1. Схема численного алгоритма, реализующего метод Рунге-Кутты 5-го порядка точности (а), и схема одного шага метода Рунге-Кутты (б)

Описанный алгоритм был реализован в виде последовательного программного комплекса на языке Си и послужил основой для настоящей работы.

2.3 Целевая вычислительная система

Целью данной работы является оптимизация программного комплекса для работы на гибридном вычислительном комплексе МВС-10П (МСЦ СО РАН), каждый узел которого кроме основных процессоров (хост-процессоров) содержит два ускорителя архитектуры Intel MIC. В качестве хост-процессоров используются два 8-ядерных процессора Intel Xeon E5-2690 2.9 ГГц

с поддержкой технологии Hyperthreading, что в сумме дает 32 аппаратных потока и 185 GFLOPS пиковой производительности на узел. Объем оперативной памяти узла составляет 64 Гб. В качестве сопроцессоров используются два ускорителя Intel Xeon Phi 7110X 1.1 ГГц, каждый из которых имеет 61 ядро, поддерживающее 4 аппаратных потока, что в сумме дает 244 аппаратных потока и около 1.1 TFLOPS пиковой производительности на ускоритель. Объем оперативной памяти, доступной каждому ускорителю, составляет 8 Гб. Использованное для работы программное обеспечение: Intel C/C++ Compiler v.14.0.1, Intel MPI Library v.4.1.

3. Оптимизация алгоритма для кластера с ускорителями Xeon Phi

3.1 Оптимизация последовательной программы

Основными предпосылками достижения высокой производительности ускорителями Xeon Phi являются наличие у них большого числа ядер и поддержка векторных операций с длинными (64-байтными) векторами. При вещественных вычислениях с двойной точностью, которые используются при решении данной задачи, одна векторная операция заменяет 8 скалярных. Таким образом, векторизация является одним из основных способов оптимизации программ для ускорителей Intel Xeon Phi, включая сюда и выравнивание обращений в память, и потоковую запись в память [4]. Так как архитектура ускорителей близка к архитектуре хост-процессоров, то проводимые оптимизационные преобразования применимы и к ним, хотя и менее существенны – размер вектора у хост-процессора равен всего 16 байт (два вещественных числа с двойной точностью). Поэтому эффект от применяемых к программе оптимизаций будет далее рассмотрен не только для ускорителей, но и для хост-процессоров. Оптимизация выполнялась средствами компилятора Intel путем добавления в код соответствующих директив и, при необходимости, модификации кода. Проверка того, что данная оптимизация успешно выполнена для данного участка кода, осуществлялась путем анализа сообщений компилятора.

С целью векторизации вычислений были выполнены следующие действия:

- Во всех гнездах циклов, обозначающих обход пространственной сетки перед внутренним циклом (по строке) была добавлена директива `#pragma ivdep`, сообщающая компилятору об отсутствии зависимостей между итерациями циклов.
- В операции weno_flux, выполняющей вычисление потоков через границы ячеек, тело цикла оказалось слишком большим, и компилятор отказался его оптимизировать. Данная ситуация была разрешена путем разбиения одной операции weno_flux на шесть операций (гнезд циклов) меньшего размера, что потребовало использования 34-х дополнительных двумерных массивов.
- В операции update_variables_1, выполняющей начальный этап вычисления примитивных переменных из консервативных, для стабилизации давлений в ячейке выполняется коррекция объемных концентраций жидких фаз методом Ньютона. Число итераций метода Ньютона, который реализуется с помощью цикла типа while, зависит от состояния конкретной ячейки. В данном случае цикл по пространству, содержащий в своем теле другой цикл, не мог быть векторизован. С целью оптимизации всё гнездо циклов по пространству было разбито на гнезда меньшего размера, содержащие: 1) операции из двух первых итераций цикла while, которые происходят в любом случае, 2) остальные итерации цикла while, 3) операции за пределами цикла while. Соответственно, первое и третье гнезда циклов были успешно векторизованы.

Каждое из перечисленных действий привело к уменьшению времени счета. В целом векторизация всей программы дала ускорение на Xeon Phi в 2 раза, на хост-процессорах – на 10%. Столь малое ускорение объяснятся большой долей времени, которая тратится на выполнение итераций методом Ньютона.

Выровненный доступ в память означает, что все обращения к массивам данных внутри векторизованного цикла происходят по адресам кратным размеру вектора. Обеспечение этого условия позволяет компилятору применить в цикле более быстрые инструкции выровненного обращения в память. Данное условие было достигнуто путем:

- выровненного выделения памяти с помощью функции _mm_malloc (или posix_memalign);
- выравнивания размера строк всех массивов по размеру, кратному размеру вектора;
- указания компилятору внутри векторизованных циклов с помощью встроенной функции компилятора __assume_aligned, что данный массив правильным образом выровнен в памяти.

В результате выравнивания обращений в память программа ускорилась на Xeon Phi всего на 1%, на хост-процессорах – на 0.2%. Такое малое ускорение объясняется тем, что наиболее значительная доля времени в программе тратится не на обращения в память, а на вычисления, которые данная оптимизация не затронула.

Еще одним способом оптимизации, рекомендуемым для ускорителя Xeon Phi [4], является использование потоковой записи в память. При ее использовании запись вычисляемых в циклах элементов массивов происходит напрямую в оперативную память, минуя кэш-память. Это было достигнуто путем добавления перед векторизованными циклами директивы `#pragma vector nontemporal`. В результате на Xeon Phi программа ускорилась очень незначительно, а на хост-процессорах замедлилась на 30%. Поэтому в итоговой версии программы потоковую запись было решено не выполнять.

Говоря об оптимизации программы нельзя не упомянуть о стандартных способах оптимизации, таких как вынос общих подвыражений и замена долгих операций на быстрые. В большинстве случаев компилятор не имеет права выполнять оптимизирующие преобразования выражений, выполняющих вычисления с вещественными числами. Поэтому такого рода оптимизации остаются на ответственности программиста [5]. С помощью ручного преобразования выражений рассматриваемая программа была ускорена более чем в 2 раза как на Xeon Phi, так и на хост-процессоре.

В таблице 1 приведено сравнение времени работы последовательной программы до и после всего цикла оптимизаций. Видно, что на последовательной оптимизированной программе ускоритель Xeon Phi значительно проигрывает хост-процессору. Это вполне ожидаемый результат, учитывая, что ядра Xeon Phi имеют более простую архитектуру, а один из ключевых способов оптимизации для них – векторизация – на данной задаче дал небольшой выигрыш.

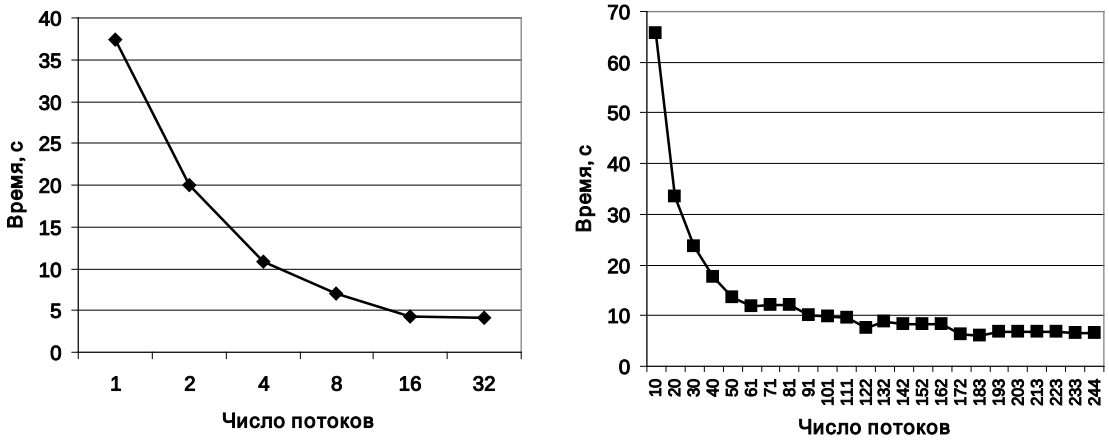
Таблица 1. Время выполнения одного шага по времени для сетки 100×100

	Intel Xeon Phi	**Хост-процессор**
Неоптимизированная программа	28.09 с	1.67 с
Оптимизированная программа	8.71 с	0.74 с
Ускорение	3.2	2.3

3.2 Распараллеливание внутри вычислительного узла

Каждый вычислительный узел кластера без учета сопроцессоров, также как и каждый сопроцессор Xeon Phi в отдельности, может рассматриваться как многоядерная система с общей памятью. Распараллеливание по ядрам является вторым из основных способов оптимизации программ для ускорителей Intel Xeon Phi. Задействовать все ядра такой системы для решения одной задачи можно с помощью технологии OpenMP. В рассматриваемой программе таким способом были распараллелены внешние циклы во всех гнездах циклов, обозначающих обход пространственной сетки. Следует отметить, что для данной задачи число параллельных потоков напрямую зависит от размера задачи, поэтому на задачах малого размера часть ядер Xeon Phi будут неизбежно простаивать.

На основе результатов, приведенных на рис. 2, можно сделать вывод, что на хост-процессорах узла было получено максимальное ускорение в 9 раз на 32 потоках. На отдельно взятом Xeon Phi максимальное ускорение составило 104.5 раза при использовании 183 потоков (по три потока на ядро), учитывая, что при использовании одного потока на Xeon Phi аналогичный временной показатель составил 644.4 секунды.



(а) (б)

Рис. 2. Время выполнения одного шага по времени для сетки размером 300×300 при различном числе потоков OpenMP на хост-процессорах (а) и на одном ускорителе Xeon Phi (б)

3.3 Совместное использование всех вычислительных ресурсов узла

Целью следующего этапа оптимизации программы стала реализация совместного использования всех вычислительных ресурсов одного узла: хост-процессоров и обоих ускорителей Xeon Phi. Существующие средства программирования предоставляют два способа достижения этой цели [4]:

- симметричный режим, при котором различные процессы одной MPI-программы распределяются по ядрам хост-процессоров и по доступным ускорителям;
- режим offload, при котором в основной программе с помощью директив компилятора выделяются данные и вычисления, которые должны быть выгружены на сопроцессоры.

Для рассматриваемой задачи были реализованы оба режима, и выполнено их сравнение.

Чтобы реализовать симметричный режим использования ресурсов узла, а впоследствии запускать программу и на нескольких узлах кластера, программа была распараллелена с использованием библиотеки MPI. Распараллеливание было выполнено методом декомпозиции пространства моделирования по двум измерениям на примерно равные подобласти с теневыми гранями шириной в три ячейки. Число MPI-процессов по каждому измерению определяется библиотекой MPI автоматически при запуске программы исходя из запрашиваемого числа процессов. Путем анализа алгоритма (рис. 1б) был определен этап, на котором минимальное число массивов должно будет обменяться граничными значениями.

На рис. 3 представлены результаты сравнения времени счета при различном соотношении числа MPI-процессов и OpenMP-потоков на различных вычислительных устройствах узла. Число потоков на процесс в каждом случае задается одинаковым для всех процессов как результат деления количества доступных ядер на число процессов. Представленные результаты показывают, что время счета зависит от соотношения процессов и потоков, и позволяет определить для них оптимальные значения: 16 процессов по 2 потока для хост-процессоров и 8 процессов по 30 потоков для ускорителей Xeon Phi.

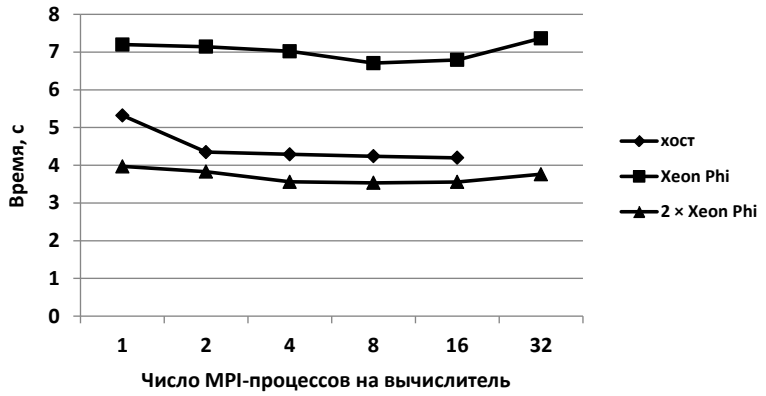
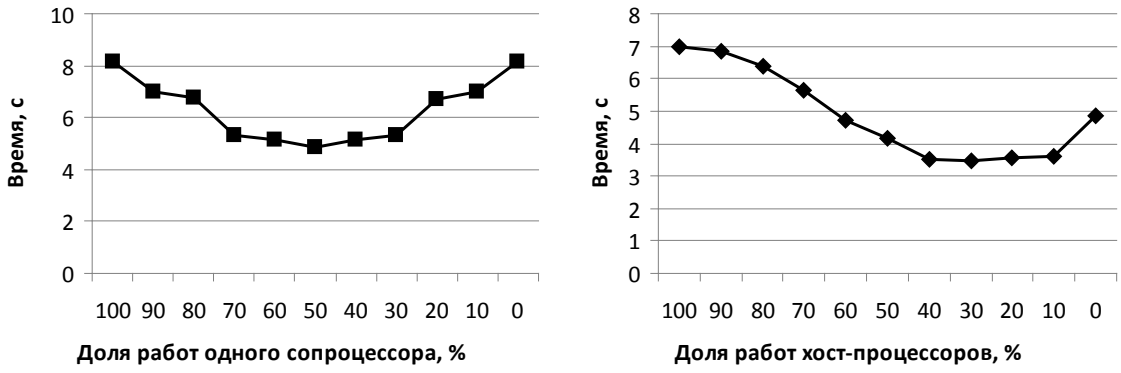


Рис. 3. Время вычисления одного шага по времени для сетки размером 1000×1000 при различном соотношении числа процессов и потоков на различных ресурсах узла

Сравнение времени счета для симметричного режима при использовании различных ресурсов узла можно посмотреть на рис. 5. Видно, что производительность двух сопроцессоров Xeon Phi на данной задаче ненамного превосходит производительность двух хост-процессоров. Поэтому при использовании всех ресурсов узла в симметричном режиме число MPI-процессов в хост-системе и одном ускорителе должно задаваться в соотношении 2:1, что соответствует полученным ранее оптимальным значениям 16 и 8. Совместное использование всех ресурсов узла позволяет ускорить расчет примерно в два раза по сравнению с использованием только хост-процессоров.

Кроме симметричного режима, программа была распараллелена в режиме offload путем декомпозиции пространства моделирования по оси y, причем было исключено излишнее копирование данных между хост-системой и сопроцессорами: используемые на каждом сопроцессоре данные хранятся только на нем, а все вычислители в узле обмениваются только границами. Распределение долей работ между хост-процессорами и сопроцессорами задается статически в качестве параметра при запуске программы. Вычисления на каждом отдельном вычислителе узла распараллеливаются с помощью OpenMP.

На рис. 4 приведены результаты работы программы для различных соотношений работ между вычислителями узла. Результаты подтверждают, что между двумя сопроцессорами узла работу лучше всего делить поровну. При использовании всех ресурсов узла в режиме offload хост-процессорам следует выделять примерно треть от всей работы. Этот результат отличается от симметричного режима, вероятно, вследствие того, что в режиме offload на хост-процессоры накладывается дополнительная задача управления счетом.



(а) (б)

Рис. 4. Время вычисления одного шага по времени для сетки размером 1000×1000 при использовании двух сопроцессоров с различным соотношением работ между ними (а), при использовании всех ресурсов узла с различным соотношением работ между хост-процессорами и сопроцессорами и равномерным разделением работ между сопроцессорами (б)

На рис. 5 представлено сравнение времени счета для симметричного и offload режимов при использовании различных ресурсов узла. По разнице времен видно, что во всех случаях режим offload имеет значительные накладные расходы. Существенные накладные расходы при использовании только хост-процессоров объясняются тем, что даже при указании такого предельного распределения работ программа в общем случае выполняет некоторую инициализацию и взаимодействие между вычислителями узла. Так как целью создания программы является совместная работа всех ресурсов узла, то оптимизация offload-версии программы для случая неиспользования каких-либо ресурсов узла является нецелесообразной.

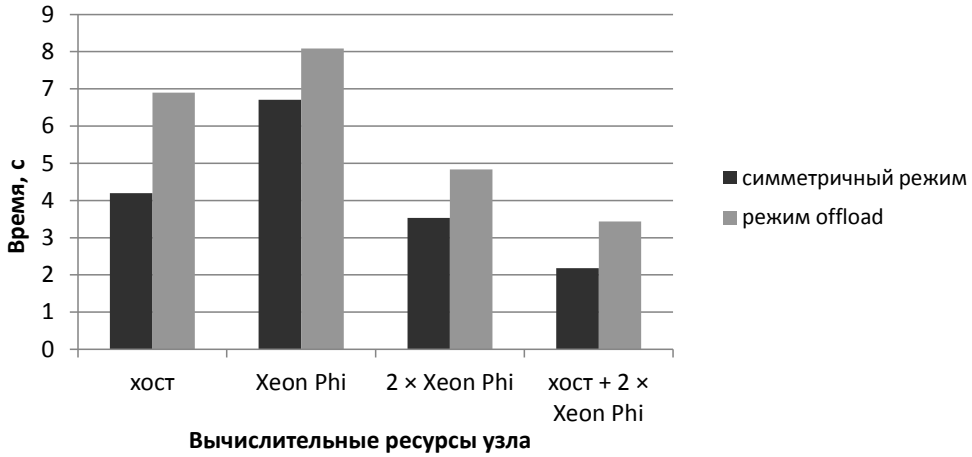


Рис. 5. Время вычисления одного шага по времени для сетки размером 1000×1000 при использовании различных ресурсов узла и различных режимов

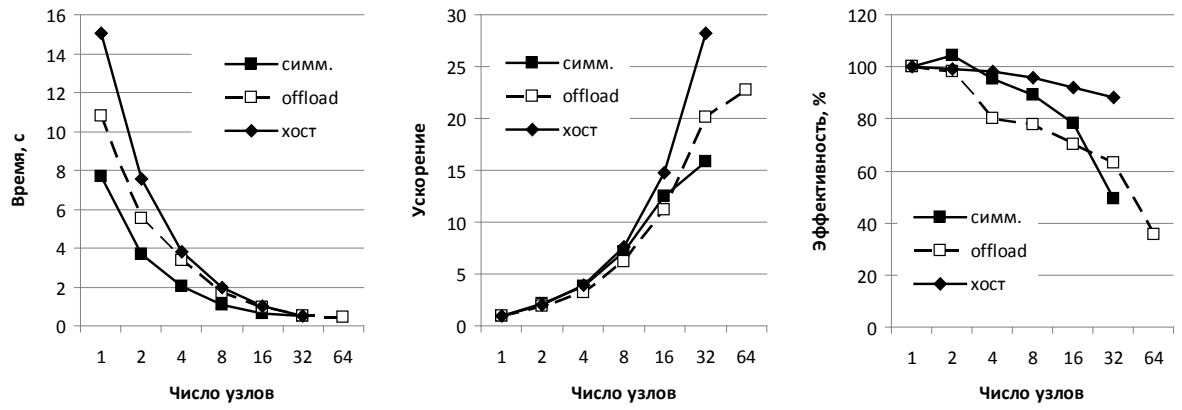
При использовании режима offload предполагается, что программист только расставит в нужных местах директивы компилятора, не сильно модифицируя свою программу. На практике же использование данных директив оказалось не столь удобным. Так необходимость идентификации массивов в параметрах директив по имени массива, а не по адресу, исключила возможность вынесения схемы на рис. 1б в подпрограмму, что привело к ее шестикратному дублированию в коде. Кроме того, по той же причине потребовалось создать отдельные копии вызовов операций алгоритма для хост-процессора и для обоих сопроцессоров. В результате использование режима offload привело в части программы, отвечающей за запуск операций алгоритма, к восемнадцатикратному дублированию кода, тем самым увеличив его объем, ухудшив читаемость и усложнив сопровождение.

Сравнивая два режима использования сопроцессоров Xeon Phi для распараллеливания больших программных комплексов с точки зрения удобства использования и производительности получаемой программы, можно отметить, что режим offload по всем параметрам проигрывает симметричному режиму. Таким образом, режим offload можно рекомендовать использовать только для распараллеливания небольших программ, которые не потребуется запускать более чем на одном вычислительном узле.

3.4 Использование нескольких узлов кластера

Разработанный с использованием MPI, OpenMP и offload-директив программный комплекс позволяет производить вычисления на кластере МВС-10П в симметричном или offload-режиме. На рис. 6 приведено сравнение времени счета, ускорения и эффективности распараллеливания при использовании только хост-процессоров или всех ресурсов узла в различных режимах в зависимости от числа узлов кластера. Параметры распределения процессов и потоков в узле для хост-процессоров и симметричного режима, а также параметры распределения работ в режиме offload выставлены в соответствии с полученными ранее оптимальными значениями. Результаты показывают, что при малом числе узлов соотношение времен повторяет соотношение для одного узла. Однако при использовании только хост-процессоров эффективность распарал-

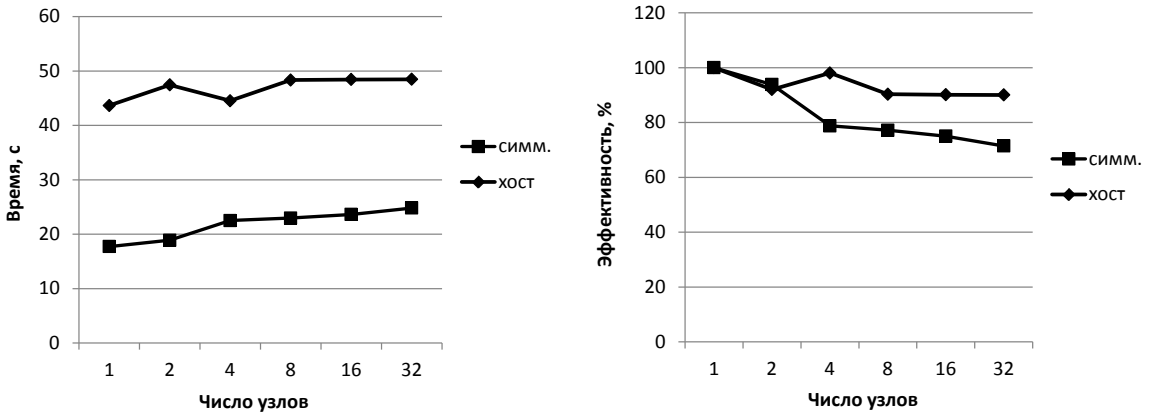
леливания с ростом числа узлов деградирует медленнее, и, начиная с некоторого критического числа узлов использование сопроцессоров для данного размера задачи становится неэффективным. Дополнительные запуски при других размерах задачи показали, что чем больше размер задачи, тем больше это критическое число узлов.



(а) (б) (в)

Рис. 6. Время вычисления одного шага по времени (а), ускорение (б) и эффективность распараллеливания (в) в зависимости от числа узлов кластера для сетки размером 1800×1800 при использовании различных ресурсов узла и различных режимов

На рис. 7 показано время счета при увеличении размера задачи пропорционально числу узлов, а также эффективность распараллеливания в слабом смысле. Видно, что при использовании всех ресурсов узла в симметричном режиме эффективность деградирует ненамного сильнее, чем при использовании только хост-процессоров, тогда как время счета отличается в два раза.



(а) (б)

Рис. 7. Время вычисления одного шага по времени (а) и эффективность распараллеливания в слабом смысле (б) в зависимости от числа узлов кластера для сетки размером 1800×1800 на узел при использовании различных ресурсов узла

4. Заключение

Параллельный программный комплекс для моделирования процесса фильтрации двухфазной жидкости через эластичную пористую среду был реализован и оптимизирован для использования на кластере с ускорителями Intel Xeon Phi. Рассмотрены способы оптимизации, специфичные для данного ускорителя, и их влияние на время работы программы. Комплекс обладает хорошей масштабируемостью относительно числа используемых узлов кластера. Использова-

ние ускорителей Xeon Phi на кластере МВС-10П позволяет уменьшить время расчета по сравнению с использованием только хост-процессоров примерно в два раза.

Опыт применения различных режимов использования ускорителей Xeon Phi показал, что для больших задач более эффективным является симметричный режим, как с точки зрения программирования, так и с точки зрения достигаемой производительности.

Литература

1. Perepechko Yu.V., Romenski E.I., Reshetova G.V. Modeling of multiphase flows in finite-deformed porous media // E. Onate, et al. (eds), Proc. of 11th World Congress on Computational Mechanics (WCCM XI), Spain. 2014. P. 4630-4641.
2. Ю.В. Перепечко, Е.И. Роменский, Г.В. Решетова, Н.Л. Перепечко, В.Э. Малышкин, К.В. Калгин, С.Е. Киреев, М.Б. Остапкевич Нелинейная акустика и режимы фильтрации в пористых средах // Суперкомпьютерные технологии в науке, образовании и промышленности: Альманах / Под редакцией академика В.А. Садовничего, академика Г.И. Савина, чл.-корр. РАН Вл.В. Воеводина. – М.: Издательство Московского университета, 2013. С. 119-126.
3. Доровский В. Н., Роменский Е. И., Федоров А. И., Перепечко Ю. В. Резонансный метод измерения проницаемости горных пород // Геология и геофизика, № 7, 2011. С. 950-961.
4. J. Jeffers, J. Reinders Intel Xeon Phi Coprocessor High-Performance Programming. Morgan Kaufmann, 2013. 432 p.
5. Agner Fog Optimizing software in C++: An optimization guide for Windows, Linux and Mac platforms, 2014. URL: http://www.agner.org/optimize/optimizing_cpp.pdf (дата обращения: 01.12.2014).