Реализация параллельного алгоритма поиска глобального экстремума функции на Intel Xeon Phi *

К.А. Баркалов, И.Г. Лебедев, В.В. Соврасов, А.В. Сысоев

Аннотация

Рассмотрен параллельный алгоритм решения задач многоэкстремальной оптимизации. Описывается реализация алгоритма на современных вычислительных системах с использованием сопроцессора Xeon Phi. Рассматриваются два подхода к распараллеливанию алгоритма, учитывающие информацию о трудоемкости вычисления значений оптимизируемой функции. Приводятся результаты вычислительных экспериментов, полученные на суперкомпьютере «Лобачевский». Демонстрируется, что реализация для Xeon Phi опережает версию для CPU. Результаты подтверждают ускорение алгоритма с использованием Xeon Phi по сравнению с алгоритмом, реализованным только на CPU.

Ключевые слова: глобальная оптимизация, многоэкстремальные функции, редукция размерности, параллельные алгоритмы, Intel Xeon Phi.

1 Введение

Рассматриваются задачи многоэкстремальной оптимизации и параллельные методы их решения. Важной особенностью указанных задач является тот факт, что глобальный экстремум есть интегральная характеристика задачи, таким образом, его отыскание связано с построением покрытия области поиска и вычислением значений оптимизируемой функции во всех точках этого покрытия. На сложность решения задач рассматриваемого класса решающее влияние оказывает размерность: вычислительные затраты растут экспоненциально при ее увеличении. Использование простейших способов решения (таких, как перебор на равномерной сетке) является неприемлемым. Требуется применение более экономных методов, которые порождают в области поиска существенно неравномерную сетку, более плотную в окрестности глобального минимума и разреженную вдали от него [1][2][3]. Данная статья продолжает развитие информационно-статистического подхода к построению параллельных алгоритмов глобальной оптимизации, предложенного в ННГУ им. Н.И. Лобачевского, который описан в монографиях [4][5].

В рамках обсуждаемого подхода решение многомерных задач сводится к решению набора связанных подзадач меньшей размерности. Соответствующая редукция основана на использовании разверток единичного отрезка вещественной оси на гиперкуб. Роль таких разверток играют непрерывные однозначные отображения типа кривой Пеано, называемые также кривыми, заполняющими пространство. Еще одним используемым механизмом снижения размерности решаемой задачи является схема вложенной (рекурсивной) оптимизации. Численные методы, позволяющие эффективно использовать аппарат таких отображений, детально разработаны и обоснованы в [4][5].

*Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 16-31-00244 мол_а «Параллельные методы решения вычислительно трудоемких задач глобальной оптимизации на гибридных кластерных системах»

Алгоритмы, развиваемые в рамках информационно-статистического подхода, основаны на предположении липшицевости оптимизируемого критерия, что является типичными для других методов (см., например, [2][3]). Предположение такого рода выполняется для многих прикладных задач, поскольку относительные вариации функций, характеризующих моделируемую систему, обычно не превышают некоторый порог, определяемый ограниченной энергией изменений в системе.

Использование современных параллельных вычислительных систем расширяет сферу применения методов глобальной оптимизации и, в то же время, ставит задачу эффективного распараллеливания процесса поиска. Именно поэтому разработка эффективных параллельных методов для численного решения задач многоэкстремальной оптимизации и создание на их основе программных средств для современных вычислительных систем является актуальной задачей. Особый интерес представляет разработка схем распараллеливания, позволяющих эффективно использовать ускорители вычислений, такие, как сопроцессор Intel Xeon Phi.

В рамках проводимого исследования мы будем предполагать, что время проведения одного испытания (вычисления значения функции в области поиска) может значительно отличаться в различных решаемых задачах. Это определяет разработку разных подходов к распараллеливанию в задачах с «легкими» и «сложными» критериями. В статье приведено описание универсального подхода к распараллеливанию алгоритма глобального поиска, который охватывает оба этих случая. Указанный подход реализован в разработанной в ННГУ им. Н.И. Лобачевского параллельной программной системе решения задач глобальной оптимизации.

2 Параллельный алгоритм глобального поиска

Рассмотрим задачу поиска глобального минимума N-мерной функции $\phi(y)$ в гиперинтервале $D = \{y \in R^N : a_i \leqslant x_i \leqslant b_i, 1 \leqslant i \leqslant N\}$. Будем предполагать, что функция удовлетворяет условию Липшица с априори неизвестной константой L.

$$\phi(y^*) = min\{\phi(y) : y \in D\} \tag{1}$$

$$|\phi(y_1) - \phi(y_2)| \leqslant L||y_1 - y_2||, y_1, y_2 \in D, 0 < L < \infty \tag{2}$$

Существует ряд способов адаптации эффективных одномерных алгоритмов для решения многомерных задач, см., например, методы диагонального [6] или симплексного [7] разбиения области поиска. В данной работе мы будем использовать подход, основанный на идее редукции размерности с помощью кривой Пеано $y(x)$, непрерывно и однозначно отображающей отрезок вещественной оси $[0, 1]$ на n-мерный куб

$$\{y \in R^N : -2^{-1} \leqslant y_i \leqslant 2^{-1}, 1 \leqslant i \leqslant N\} = \{y(x) : 0 \leqslant x \leqslant 1\} \tag{3}$$

Вопросы численного построения отображений типа кривой Пеано и соответствующая теория подробно рассмотрены в [4]. Здесь же отметим, что численно построенная развертка является приближением к теоретической кривой Пеано с точностью порядка 2^{-m}, где m — параметр построения развертки. Использование подобного рода отображений позволяет свести многомерную задачу к одномерной задаче

$$\phi(y^*) = \phi(y(x^*)) = min\{\phi(y(x)) : x \in [0, 1]\} \tag{4}$$

Важным свойством является сохранение ограниченности относительных разностей функции: если функция $\phi(y)$ в области D удовлетворяла условию Липшица, то функция $\phi(y(x))$

на интервале $[0,1]$ будет удовлетворять равномерному условию Гельдера

$$|\phi(y(x_1)) - \phi(y(x_2))| \leqslant H|x_1 - x_2|^{\frac{1}{N}}, x_1, x_2 \in [0,1] \tag{5}$$

где константа Гельдера H связана с константой Липшица L соотношением

$$H = 4Ld\sqrt{N}, d = max\{b_i - a_i : 1 \leqslant i \leqslant N\} \tag{6}$$

Поэтому, не ограничивая общности, можно рассматривать минимизацию одномерной функции $f(x) = \phi(y(x)), x \in [0,1]$, удовлетворяющей условию Гельдера.

Рассматриваемый алгоритм решения данной задачи предполагает построение последовательности точек x_k, в которых вычисляются значения минимизируемой функции $z_k = f(x_k)$. Процесс вычисления значения функции (включающий в себя построение образа $y_k = y(x_k)$) будем называть испытанием, а пару (x_k, z_k) — результатом испытания. Множество пар $\{(x_k, z_k)\}, 1 \leqslant k \leqslant n$ составляют поисковую информацию, накопленную методом после проведения n шагов. В нашем распоряжении имеется $p \geqslant 1$ вычислительных элементов и в рамках одной итерации метода мы будем проводить p испытаний одновременно. Обозначим $k(n)$ общее число испытаний, выполненных после n параллельных итераций.

На первой итерации метода испытание проводится в произвольной внутренней точке x_1 интервала $[0,1]$. Пусть выполнено $n \leqslant 1$ итераций метода, в процессе которых были проведены испытания в $k = k(n)$ точках $x_i, 1 \leqslant i \leqslant k$. Тогда точки $x^{k+1}, \ldots, x^{k+p}$ поисковых испытаний следующей $(n+1)$-ой итерации определяются в соответствии с правилами:

Шаг 1. Перенумеровать точки множества $X_k = \{x^1, \ldots, x^k\} \cup \{0\} \cup \{1\}$, которое включает в себя граничные точки интервала $[0,1]$, а также точки предшествующих испытаний, нижними индексами в порядке увеличения значений координаты, т.е.

$$0 = x_0 < x_1 < \ldots < x_{k+1} = 1 \tag{7}$$

Шаг 2. Полагая $z_i = f(x_i), 1 \leqslant i \leqslant k$, вычислить величины

$$\mu = max\frac{|z_i - z_{i-1}|}{\Delta_i}, M = \tag{8}$$

где r является заданным параметром метода, а $\Delta_i = (x_i - x_{i-1})^{\frac{1}{N}}$.

Шаг 3. Для каждого интервала $(x_{i-1}, x_i), 1 \leqslant i \leqslant k+1$, вычислить характеристику в соответствии с формулами

(9)

(10)

Шаг 4. Характеристики $R(i), 1 \leqslant i \leqslant k+1$, упорядочить в порядке убывания

(11)

и выбрать наибольших характеристик с номерами интервалов $t_j, 1 \leqslant j \leqslant p$.

Шаг 5. Провести новые испытания в точках $x_{k+j}, 1 \leqslant j \leqslant p$, вычисленных по формулам

(12)

Алгоритм прекращает работу, если выполняется условие $\Delta_{t_j} \leqslant \varepsilon$ хотя бы для одного номера $t_j, 1 \leqslant j \leqslant p$; здесь $\varepsilon > 0$ есть заданная точность. В качестве оценки глобально-оптимального решения задачи выбираются значения

(13)

Теоретическое обоснование данного способа организации параллельных вычислений изложено в [5].

3 Обобщенная схема редукция размерности

Одним из подходов к решению многомерных задач глобальной оптимизации является сведение их к одномерным и использование эффективных одномерных алгоритмов глобального поиска к редуцированной задаче. В предыдущем разделе была изложена идея редукции размерности с использованием кривых Пеано. Ниже излагается обобщенный способ редукции размерности, комбинирующий использование разверток и схему вложенной (рекурсивной) оптимизации.

3.1 Рекурсивная схема редукции размерности

Схема рекурсивной оптимизации основана на известном [10] соотношении , (9) которое позволяет заменить решение многомерной задачи (1) решением семейства одно-мерных подзадач, рекурсивно связанных между собой. Введем в рассмотрение множество функций , (10) , . (11) Тогда, в соответствии с соотношением (9), решение исходной задачи сводится к решению одномерной задачи . (12) Однако при этом каждое вычисление значения одномерной функции в некоторой фиксированной точке предполагает решение одномерной задачи минимизации ,и так далее до вычисления согласно (10).

3.2 Блочная рекурсивная схема редукции размерности

Для изложенной выше рекурсивной схемы предложено обобщение (блочная рекурсивная схема), которое комбинирует использование разверток и рекурсивной схемы с целью эффективного распараллеливания вычислений. Рассмотрим вектор y как вектор блочных переменных , где i-я блочная переменная ui представляет собой вектор размерности из последовательно взятых компонент вектора y, т.е. , ,..., , причем . С использованием новых переменных основное соотношение многошаговой схемы (9) может быть переписано в виде , (13) где подобласти , являются проекциями исходной области поиска D на подпространства, соответствующие переменным . Формулы, определяющие способ решения задачи на основе соотношений (13) в целом совпадают с рекурсивной схемой (10)⊠(12). Требуется лишь заменить исходные перемен-ные , на блочные переменные . При этом принципиальным отличием от исходной схемы является тот факт, что в блочной схеме вложенные подзадачи , , (14) являются многомерными, и для их решения может быть применен способ редукции раз-мерности на основе кривых Пеано. Число векторов и количество компонент в каждом векторе являются параметрами блочной многошаговой схемы и могут быть использованы для формирования подзадач с нужными свойствами. Например, если , т.е. , то блочная схема идентична исходной; каждая из вложенных подзадач является одномерной. А если , т.е. , то решение задачи эквивалентно ее решению с использованием единствен-ной развертки, отображающей [0,1] в D; вложенные подзадачи отсутствуют.

4 Реализация на Xeon Phi

В2012 году компания Intel представила первый сопроцессор с архитектурой Intel MIC (Intel® Many Integrated Core Architecture). Архитектура MIC позволяет использовать большое количество вычислительных ядер архитектуры x86 в одном процессоре. В результате для параллельного программирования могут быть использованы стандартные технологии, такие как OpenMP и MPI. Архитектура Intel Xeon Phi поддерживает несколько режимов использования сопроцессора, которые можно комбинировать для достижения максимальной производительно-сти в зависимости от характеристик решаемой задачи. В режиме Offload процессы MPI выполняются только на CPU, а на сопроцессоре про-исходит запуск отдельных функций, аналогично использованию графических ускорителей. В режиме MPI базовая система и каждый сопроцессор Intel Xeon Phi рассматриваются как отдельные равноправные узлы, и процессы MPI могут выполняться на центральных процессорах и сопроцессорах Xeon Phi в произвольных сочетаниях.

4.1 Режим Offload

Вначале рассмотрим ситуацию, когда проведение одного испытания является трудоемкой операцией. В этом случае ускоритель Xeon Phi может быть использован в режиме Offload для параллельного проведения сразу многих испытаний на одной итерации метода. Пересылки данных от CPU к Xeon Phi будут минимальны – требуется лишь передать на сопроцессор координаты точек испытаний, и получить обратно значения функции в этих точках. Функции, определяющие обработку результатов испытаний в соответствии с алгоритмом и требующие работы с большим объемом накопленной поисковой информацией, могут быть эффективно реализованы на CPU. Общая схема организации вычислений с использованием Xeon Phi будет следующий. На CPU выполняются шаги 1 – 4 параллельного алгоритма глобального поиска из п. 2. При этом на каждой итерации происходит накопление координат точек испытания в буфере, и этот буфер передается на сопроцессор. На Xeon Phi выполняется параллельное вы-числение значений функции в этих точках (шаг 5 алгоритма). Используется OpenMP распараллеливание цикла, на каждой итерации которого происходит вычисление значений функции. По завершению испытаний происходит передача вычисленных значений функ-ции на CPU.

4.2 Режим MPI

В случае, если проведение одного поискового испытания является относительно про-стой операцией, параллельное проведение многих итераций в режиме Offload не дает большого ускорения (сказывается влияние накладных расходов на передачу данных). Од-нако здесь можно увеличить вычислительную нагрузку на Xeon Phi, если применить блочную схему редукции размерности из п. 3.2, а для решения возникающих подзадач использовать сопроцессор в режиме MPI. Для организации параллельных вычислений будем использовать небольшое (2-3) число уровней вложенности в блочной схеме, при котором исходная задача большой размерности разбивается на 2-3 вложенные подзадачи меньшей размерности. Тогда, применяя в блочной рекурсивной схеме (13) для решения вложенных подзадач (14) параллельный алгоритм глобальной оптимизации, мы получим схему параллельных вычислений с широ-кой степенью вариативности (например, можно варьировать количество процессоров на различных уровнях оптимизации, т.е. при решении подзадач по различным переменным). Общая схема организации вычислений с использованием нескольких узлов кластера и нескольких сопроцессоров состоит в следующем. Процессы параллельной

программы об-разуют дерево, соответствующее уровням вложенных подзадач, при этом вложенные под-задачи

при i⊠1,...,M–2 решаются только с использованием CPU. Непосредственно в данных подзадачах вычислений значений оптимизируемой функции не происходит: вычисление значения функции ⊠ это решение задачи минимизации следующего уровня. Каждая подзадача решается в отдельном процессе; обмен данными осуществляется лишь между процессами-предками и процессами-потомками. Подзадача последнего (M–1)-го уровня

отличается от всех предыдущих подзадач – в ней происходит вычисление значений оптимизируемой функции, т.к. . Подзадачи этого уровня решаются на сопроцессоре, и каждое ядро сопроцессора будет решать свою подзадачу (M–1)-го уровня в отдельном MPI-процессе. Самый простой вариант использования данной схемы будет соответствовать двухкомпонентному вектору распараллеливания . Здесь будет соответствовать числу MPI-процессов на CPU, а ⊠числу MPI-процессов на Xeon Phi; тем самым общее количество процессов будет определяться как . Отметим, что синхронный параллельный алгоритм глобальной оптимизации, описанный в п.3, в сочетании с блочной схемой редукции размерности обладает существенным недостатком, связанным с возможными простоями всех процессов, кроме корневого. Простои могут возникать в том случае, если часть потомков некоторого процесса закончили решение своих подзадач и отправили данные родителю раньше остальных. Для преодоления данной проблемы был применен предложенный в [12] асинхронный вариант параллельного алгоритма. Конкретные детали реализация блочной схемы в сочетании с асинхронным алгоритмом описаны в [12].

5 Результаты численных экспериментов

Вычислительные эксперименты проводились на узле суперкомпьютера «Лобачевский», установленного в ННГУ им. Н.И. Лобачевского, на сегменте под управлением опе-рационной системы семейства Windows (HPC Server 2008). Узел располагает двумя процессорами Intel Sandy Bridge E5-2660 2.2 GHz, 64 Gb RAM, и двумя сопроцессорами Intel Xeon Phi 5110P. В работе [6] описан GKLS-генератор, позволяющий порождать задачи многоэкстремальной оптимизации с заранее известными свойствами: количеством локальных мини-мумов, размерами их областей притяжения, точкой глобального минимума, значением функции в ней и т.п. Тестовые задачи, порождаемые данным генератором, характеризуются малым временем вычисления значений целевой функции. Поэтому с целью имитации вычислительной трудоемкости, присущей прикладным задачам оптимизации, расчет критерия был усложнен дополнительными вычислениями, не меняющими вид функции и рас-положение ее минимумов.

Рис. 1. Решение двумерной задачи

Для примера на рис. 1 приведены линии уровня двумерной функции, порождаемой GKLS-генератором. Темные точки соответствуют 464 испытаниям, потребовавшимся для решения данной задачи с точностью 0.01 по координате; при этом число итераций составило 58 (на каждой итерации проводилось 8 испытаний). Линии уровня наглядно демонстрируют многоэкстремальность задачи, а расположение точек – неравномерное покрытие, сгущающееся только в районе глобального минимума. Ниже приведены результаты численного сравнения трех последовательных алгоритмов – DIRECT [7], DIRECTl [8] и алгоритма глобального поиска (АГП) (результаты рабо-ты первых двух алгоритмов приводятся по работе [9]). Численное сравнение проводилось на классах функций Simple и Hard размерности 4 и 5 из [9]. Глобальный минимум y* считался найденным, если алгоритм генерировал точку испытания yk в ⊠-окрестности глобального минимума, т.е. . При этом размер окрестности

выбирался (в соответствии с [9]) как , здесь N – размерность решаемой задачи, a и b – границы области поиска D, параметр ⊠=10⊠6 при N=4 и ⊠=10⊠7 при N=5. При использовании метода АГП для класса Simple выбирался параметр r=4.5, для класса Hard ⊠ r=5.6; параметр по-строения кривой Пеано был фиксированный m⊠10. Максимально допустимое число итераций составляло Kmax = 1 000 000. В таблице 1 отражено среднее число итераций kav, которые выполнил метод при решении серии задач из данных классов. Символ «>» отражает ситуацию, когда не все зада-чи класса были решены каким-либо методом. Это означает, что алгоритм был остановлен по причине достижения максимально допустимого числа итераций Kmax. В этом случае значение Kmax=1 000 000 использовалось при вычислении среднего значения числа итераций kav, что соответствует нижней оценке этого среднего значения. Количество нерешенных задач указано в скобках. Таблица 1. Среднее число итераций N Класс функции DIRECT DIRECTl АГП 4 Simple >47282 (4) 18983 11953 Hard >95708 (7) 68754 25263 5 Simple >16057 (1) 16758 15920 Hard >217215 (16) >269064 (4) >148342 (4)

Как видно из таблицы 1, АГП превосходит методы DIRECT и DIRECTl на всех классах задач по среднему числу итераций. При этом в классе 5-Hard каждый из методов решил не все задачи: DIRECT не решил 16 задач, DIRECTl и АГП – по 4 задачи.

5.1 Задачи с трудоемким критерием

Вначале приведем результаты экспериментов, которые соответствуют задачам с большим временем проведения одного поискового испытания. Оценим ускорение параллельного алгоритм глобального поиска (ПАГП), реализованного на CPU, в зависимости от использованного количества ядер p. В таблицах 2 и 3 приведено ускорение по времени S(p) и по итерациям s(p). Ускорение вычислено относительно однопоточного запуска.

Таблица 2. Ускорение по времени S(p) на CPU p N⊠4 N⊠5 Simple Hard Simple Hard 2 2.45 2.20 1.15 1.32 4 4.66 3.90 2.82 2.59 8 7.13 7.35 3.47 5.34

Таблица 3. Ускорение по итерациям s(p) на CPU p N⊠4 N⊠5 Simple Hard Simple Hard 2 2.51 2.26 1.19 1.36 4 5.04 4.23 3.06 2.86 8 8.58 8.79 4.22 6.56

Результаты экспериментов показывают значительное ускорение ПАГП при использовании CPU. Лучшие результаты получены при использовании всех ядер процессора. Далее изложим результаты экспериментов, проведенных с использованием Intel Xeon Phi. Вначале рассмотрим эксперименты на одном Xeon Phi в режиме Offload. Варьировалось количество потоков на сопроцессоре, все остальные параметры совпадают с предыдущими запусками. Приведено (таблицы 4 и 5) ускорение относительно восьмипоточного запуска на центральном процессоре. Таблица 4. Ускорение по времени S(p) на Phi p N⊠4 N⊠5 Simple Hard Simple Hard 60 0.54 1.02 1.07 1.61 120 0.55 1.17 1.05 2.61 240 0.51 1.06 1.07 4.17 Таблица 5. Ускорение по итерациям s(p) на Phi p N⊠4 N⊠5 Simple Hard Simple Hard 60 8.13 7.32 9.87 6.55 120 16.33 15.82 15.15 17.31 240 33.07 27.79 38.80 59.31

Результаты экспериментов показывают, что только на классе Simple при N=4 реализация на Xeon Phi медленнее, чем CPU реализация, на классе Hard при N=4 и на классе Simple при N=5 версии для Xeon Phi и CPU демонстрирует примерно равную эффективность, а на классе Hard при N=5 версия для Xeon Phi существенно превосходит CPU-реализацию. Наибольшее преимущество реализация для Xeon Phi показывает при 120 потоках для четырехмерных задач и при 240 потоках для пятимерных. При этом на всех классах наблюдается значительное ускорение по итерациям, а также практически линейная масштабируемость от числа потоков. Далее рассмотрим эксперименты с запуском сопроцессора в режиме MPI. Число уровней вложенности разбиваемой задачи равно двум. Число процессов, запущенных на со-процессоре, – 30 (таблицы 6 и 7) и 60(таблицы 8 и 9), что соответствует

ситуации, когда у корня дерева имеется соответственно 30 и 60 потомков. Процессы, работающие на сопроцессоре, использовали OpenMP для параллельного вычисления значений функции. Ускорение приведено относительно восьмипоточного запуска на CPU. Таблица 6. Ускорение по времени S(p) на Phi, 30 MPI процессов p N⊠4 N⊠5 Simple Hard Simple Hard 4 3,18 5,14 5,15 14,65 8 3,53 1,49 3,35 10,49 16 1,09 2,77 4,61 14,36 Таблица 7. Ускорение по итерациям s(p) на Phi, 30 MPI процессов p N⊠4 N⊠5 Simple Hard Simple Hard 4 11,19 19,19 9,27 27,38 8 13,97 3,63 8,24 23,85 16 1,29 3,56 8,28 22,12 Таблица 8. Ускорение по времени S(p) на Phi, 60 MPI процессов p N⊠4 N⊠5 Simple Hard Simple Hard 4 2,76 3,20 4,19 16,56 8 3,05 7,73 4,08 14,87 16 0,89 5,21 2,72 13,05 Таблица 9. Ускорение по итерациям s(p) на Phi, 60 MPI процессов p N⊠4 N⊠5 Simple Hard Simple Hard 4 10,36 3,34 6,01 26,04 8 9,04 28,81 7,68 25,02 16 1,27 28,66 7,19 30,38

Результаты экспериментов показывают, что запуск в режиме MPI значительно превосходит Offload режим по времени работы, однако проигрывает по числу итераций. Лучшие результаты для простого класса задач наблюдаются при использовании 30 процессов на сопроцессоре, а для сложного класса, при 60 процессах. Таким образом, можно сделать вывод, что задачи с большим временем работы, для которых важно число итераций мето-да оптимизации, лучше использовать запуск в режиме Offload, а для более быстро решаемых задач, лучше подходит режим MPI.

5.2 Задачи с легко вычисляемым критерием

Рассмотрим теперь решение задач, в которых поисковые испытания проводятся быстро. Вначале приведем результаты OpenMP версии. В таблице 10 приведено ускорение относительно однопоточного запуска. Таблица 10. Ускорение по времени S(p) на CPU p N⊠4 N⊠5 Simple Hard Simple Hard 2 1,09 1,09 0,56 0,57 4 1,10 1,06 0,74 0,66 8 0,90 1,11 0,45 1,77 16 0,66 0,92 0,76 0,66 32 0,29 0,38 0,22 0,55

Эксперименты показывают незначительное ускорение, а во многих случаях – и замедление при вычислениях только на центральном процессоре. Далее рассмотрим ускорение при использовании Xeon Phi в режиме MPI. Результаты приведены относительно однопоточного запуска на CPU. В таблице 11 приведено ускорение по времени S(p) для запуска 30 процессов на Xeon Phi, в таблице 12 для 60 процессов. Таблица 11. Ускорение по времени S(p) на Phi, 30 MPI процессов p N⊠4 N⊠5 Simple Hard Simple Hard 1 0,35 2,28 0,37 1,92 2 0,94 1,95 0,39 1,23 4 1,23 0,43 0,54 2,63 8 0,85 0,56 0,40 1,53

Таблица 12. Ускорение по времени S(p) на Phi, 60 MPI процессов p N⊠4 N⊠5 Simple Hard Simple Hard 1 0,59 2,56 1,22 3,30 2 0,89 0,87 1,29 2,76 4 1,09 1,03 0,38 5,32 8 0,70 0,78 0,69 2,19

Результаты экспериментов показывают ускорение на Xeon Phi, превосходящее ускорение на центральном процессоре. Лучшие результаты получены при использовании 30 процессов на Xeon Phi только для простых четырехмерных задач, все остальные задачи лучше всего решаются при использовании 60 процессов. Теперь перейдем к эксперименту с шестимерными задачами простого класса. При решении шестимерной задачи с использованием одного процесса (вложенные подзадачи отсутствуют) и сохранением точности построения развертки m=10 требуется использовать числа с плавающей запятой расширенной точности, что ведет к значительному увеличению времени работы алгоритма. При этом в случае использования хотя бы одного уровня вложенных подзадач, задачи на каждом уровне будут решаться без подключения расширенной точности, что обеспечивает дополнительное преимущество блочной многошаговой схемы. Рассмотрим результаты эксперимента с использованием Xeon Phi в режиме MPI, таблица 13. Число уровней вложенности разбиваемой задачи равно двум. Приведены результаты экспериментов для разбиений 3:3 (три

переменных на первом уровне, три – на вто-ром) и 4:2 (4 переменных на первом уровне, две – на втором). Число процессов, запущенных на сопроцессоре равно 30 и 60, что соответствует тому, что у корня дерева имеется соответственно 30 и 60 потомков. Процессы, работающие на Xeon Phi, использовали OpenMP для параллельного вычисления значений функции. Ускорение приведено относительно однопоточного запуска на CPU. Таблица 13. Ускорение по времени S(p) на Phi p 30 процессов 60 процессов 3:3 4:2 3:3 4:2 2 7,30 9,87 17,68 18,21 4 5,99 8,18 6,60 13,57 8 3,14 12,74 5,92 25,58 16 3,54 7,61 4,87 11,33 32 3,41 6,85 3,73 6,50

Результаты экспериментов показывают значительное ускорение запуска на сопроцессоре по сравнению с запуском только на центральном процессоре. Лучшее ускорение наблюдается при шестидесяти MPI процессах на Xeon Phi по 8 потоков на каждый процесс, время работы в 25 раз меньше, чем решение задачи только на центральном процессоре.

6 Заключение

В работе рассмотрен параллельный алгоритм глобального поиска, разработанный в рамках информационно статистического подхода. Предложен подход к распараллеливанию данного алгоритма, который может быть эффективен как в задачах с трудоемким критерием оптимизации, так и в задачах с относительно простым критерием. Проведены эксперименты с реализацией предложенного параллельного алгоритма на суперкомпьютере «Лобачевский» с использованием сопроцессоров Intel Xeon Phi. Результаты вычислительных экспериментов подтверждают высокую эффективность распараллеливания для Xeon Phi во всех классах рассмотренных задач.

Список литературы

[1] Евтушенко Ю.Г. *Методы решения экстремальных задач и их применение в системах оптимизации.* М.: Наука, 1982, 432с.

[2] Pinter J. *Global optimization in action (Continuous and Lipschitz optimization: algorithms, imple-mentations and applications).* Kluwer Academic Publishers, 1996, 507 p.

[3] Квасов Д.Е. Сергеев Я.Д. *Диагональные методы глобальной оптимизации.* М.: Физматлит, 2008, 352с.

[4] Sergeyev Ya.D. Strongin R.G. *Global Optimization with Non-convex Constraints. Sequential and Parallel Algorithms.* Kluwer Academic Publishers, 2000, 704p.

[5] Гришагин В.А. Баркалов К.А. Стронгин Р.Г., Гергель В.П. *Параллельные вычисления в задачах глобальной оптимизации.* М.: Издательство Московского университета, 2013, 280с.

[6] Sergeyev Ya.D. Kvasov D.E. Global search based on efficient diagonal partitions and a set of lipschitz constants. *SIAM Journal on Optimization*, 16(3):910–937, 2006.

[7] Žilinskas J. Branch and bound with simplicial partitions for global optimization. *Mathematical Modelling and Analysis*, 13(1):145–159, 2008.