2022, **23** (4), 339–349. doi 10.26089/NumMet.v23r421

doi 10.26089/NumMet.v23r421 УДК 519.853.4

О реализации параллельного алгоритма глобальной оптимизации с использованием набора инструментов Intel oneAPI

К. А. Баркалов
Нижегородский государственный университет имени Н. И. Лобачевского, Нижний Новгород, Российская Федерация
ORCID: 0000-0001-5273-2471, e-mail: konstantin.barkalov@itmm.unn.ru

И. Г. Лебедев
Нижегородский государственный университет имени Н. И. Лобачевского, Нижний Новгород, Российская Федерация
ORCID: 0000-0002-8736-0652, e-mail: ilya.lebedev@itmm.unn.ru

Я. В. Силенко
Нижегородский государственный университет имени Н. И. Лобачевского, Нижний Новгород, Российская Федерация
ORCID: 0000-0003-4894-2497, e-mail: yanina.silenko@itmm.unn.ru

Аннотация: В статье рассматривается параллельный алгоритм решения задач глобальной оптимизации и обсуждается его реализация с использованием набора инструментов Intel oneAPI. Предполагается, что целевая функция задачи задана как “черный ящик” и удовлетворяет условию Липшица. Изложенный в статье параллельный алгоритм использует схему редукции размерности на основе кривых Пеано, которые непрерывно и однозначно отображают отрезок вещественной оси на гиперкуб. В качестве средства для реализации параллельного алгоритма использован инструментарий Intel oneAPI, который позволяет писать один код как для центрального процессора, так и для графических ускорителей. Приведены результаты вычислительных экспериментов, полученные при решении серии сложных задач многоэкстремальной оптимизации.

Ключевые слова: глобальная оптимизация, многоэкстремальные функции, параллельные вычисления, редукция размерности, графические ускорители, Intel oneAPI.

Благодарности: Работа выполнена при поддержке программы Центра компетенций oneAPI в ННГУ, Министерства науки и высшего образования РФ (проект № 0729–2020–0055) и научно-образовательного математического центра “Математика технологий будущего” (проект № 075–02–2022–883).

Для цитирования: Баркалов К.А., Лебедев И.Г., Силенко Я.В. О реализации параллельного алгоритма глобальной оптимизации с использованием набора инструментов Intel oneAPI // Вычислительные методы и программирование. 2022. **23**, № 4. 339–349. doi 10.26089/NumMet.v23r421.

2022, **23** (4), 339–349. doi 10.26089/NumMet.v23r421

On implementation of the parallel global optimization algorithm with the Intel oneAPI toolkit

Konstantin A. Barkalov
Lobachevsky State University of Nizhny Novgorod, Nizhny Novgorod, Russia
ORCID: 0000-0001-5273-2471, e-mail: konstantin.barkalov@itmm.unn.ru

Ilya G. Lebedev
Lobachevsky State University of Nizhny Novgorod, Nizhny Novgorod, Russia
ORCID: 0000-0002-8736-0652, e-mail: ilya.lebedev@itmm.unn.ru

Yanina V. Silenko
Lobachevsky State University of Nizhny Novgorod, Nizhny Novgorod, Russia
ORCID: 0000-0003-4894-2497, e-mail: yanina.silenko@itmm.unn.ru

Abstract: The paper considers the parallel global optimization algorithm and discusses its implementation with the Intel oneAPI toolkit. We suppose that the objective function is given as a black-box and satisfies the Lipschitz condition. The parallel algorithm presented in the paper uses the scheme of dimensionality reduction employing the Peano curve, which continuously maps an interval of the real axis onto a hypercube. The Intel oneAPI tools, that allows one to write the same code for both the central processor and the graphics accelerator, were used for implementation of the parallel global optimization algorithm. The results of numerical experiments obtained by solving a series of time-consuming multiextremal optimization problems are presented.

Keywords: global optimization, multiextremal functions, parallel computing, reduction of dimensionality, graphics accelerators, Intel oneAPI.

Acknowledgements: The work was supported by the oneAPI Center of Excellence, by the Ministry of Science and Higher Education of the Russian Federation (project No. 0729–2020–0055), and by the Research and Education Mathematical Center (project No. 075–02–2022–883).

For citation: K. A. Barkalov, I. G. Lebedev, and Ya. V. Silenko, "On implementation of the parallel global optimization algorithm with the Intel oneAPI toolkit," Numerical Methods and Programming. **23** (4), 339–349 (2022). doi 10.26089/NumMet.v23r421.

1. Постановка задачи. Задачи поиска минимума многоэкстремальных функций часто возникают в различных областях науки — химии, физике, биологии и т.д. В частности, такие задачи возникают при разработке новых лекарственных препаратов [1], при поиске конфигураций новых химических соединений [2, 3]. Задачи глобальной оптимизации возникают и во многих других приложениях [4]. Традиционной сферой применения методов глобальной оптимизации стала идентификация параметров математических моделей по данным экспериментов. В задачах такого вида требуется провести поиск значений неизвестных параметров модели, при которых результаты расчетов близки к результатам, полученным экспериментально [5].

Мы будем рассматривать задачу глобальной оптимизации в следующей постановке: требуется найти глобальный минимум y^* функции $\varphi(y)$ в гиперинтервале D, т.е.

$$\varphi(y^*) = \min\{\varphi(y) : y \in D\}, \quad D = \{y \in \mathbb{R}^N : \ a_i \leqslant x_i \leqslant b_i, \ 1 \leqslant i \leqslant N\}. \tag{1}$$

Будем предполагать, что информация о структуре целевой функции $\varphi(y)$ неизвестна, а сама функция задается как "черный ящик", т.е. с помощью некоторого реализованного программно алгоритма вычисления ее значений. Также будем предполагать, что $\varphi(y)$ удовлетворяет условию Липшица

$$|\varphi(y_1) - \varphi(y_2)| \leqslant L\|y_1 - y_2\|, \quad y_1, y_2 \in D, \quad 0 < L < \infty,$$

причем константа Липшица L априори неизвестна.

2022, **23** (4), 339–349. doi 10.26089/NumMet.v23r421

Многие известные алгоритмы липшицевой глобальной оптимизации основаны на идее редукции размерности и адаптации одномерных алгоритмов для решения многомерных задач [6, 7]. В данной работе мы будем использовать подход из [8, 9], основанный на идее редукции размерности с помощью кривой Пеано $y(x)$, которая непрерывно и однозначно отображает отрезок вещественной оси $[0,1]$ на N-мерный гиперкуб:

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

С использованием отображения $y(x)$ решение исходной задачи (1) сводится к минимизации одномерной функции на отрезке $[0,1]$, т.е.

$$\varphi(y^*) = \varphi(y(x^*)) = \min\{\varphi(y(x)) : x \in [0,1]\}.$$

Указанный способ редукции размерности обладает следующим важным свойством: сохраняется ограниченность относительных разностей функции, т.е. если в области D функция $\varphi(y)$ удовлетворяла условию Липшица, то на интервале $[0,1]$ функция $\varphi(y(x))$ будет удовлетворять равномерному условию Гельдера

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

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

Пользуясь этим свойством, можно трактовать исходную задачу (1) как задачу минимизации одномерной функции $f(x) = \varphi(y(x))$, $x \in [0,1]$, удовлетворяющей условию Гельдера. Для решения указанной задачи минимизации может быть применен *алгоритм глобального поиска* [10], который является эффективным алгоритмом, так как при решении сложных многоэкстремальных задач опережает (по числу итераций, требующихся для корректного решения задачи с фиксированной точностью) другие методы аналогичного назначения [11]. Алгоритм глобального поиска является весьма гибким и отлично подходит для реализации на параллельных вычислительных системах.

2. Параллельный алгоритм глобального поиска. Рассматриваемый в статье параллельный алгоритм глобального поиска в процессе своей работы порождает последовательность точек $\{x^k\}$, в каждой из которых проходит вычисление значений минимизируемой функции $f(x^k)$. В дальнейшем будем называть процесс вычисления значения одномерной функции в точке x^k *поисковым испытанием*. Испытание включает в себя также построение образа $y^k = y(x^k)$, а результатом испытания является пара $(x^k,\ z^k)$, где $z^k = \varphi(y(x^k))$. Процесс распараллеливания организован таким образом, что при выполнении одной итерации метода одновременно проводятся $p \geqslant 1$ испытаний.

Для описания вычислительной схемы параллельного алгоритма глобального поиска обозначим $k(n)$ общее число испытаний, которые были проведены после выполнения n итераций алгоритма.

На подготовительном шаге параллельно проводятся p поисковых испытаний в произвольных внутренних точках $x^1, \ldots, x^p$ отрезка $[0,1]$, что соответствует первой итерации алгоритма.

Если выполнено $n \geqslant 1$ итераций, которым соответствуют $k = k(n)$ проведенных поисковых испытаний в точках x^i, $1 \leqslant i \leqslant k$, то точки $x^{k+1}, \ldots, x^{k+p}$ испытаний следующей $(n+1)$-й итерации будут определяться следующим образом.

Шаг 1. Перенумеровать (нижним индексом) точки ранее проведенных испытаний x^i, $1 \leqslant i \leqslant k$, а также граничные точки отрезка $[0,1]$ в порядке возрастания координаты:

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

и сопоставить им значения $z_i = f(x_i)$.

Шаг 2. Вычислить текущие нижние оценки M неизвестной константы Гельдера H:

$$\mu = \max\left\{\frac{|z_i - z_{i-1}|}{(x_i - x_{i-1})^{1/N}},\ \ i = 1, \ldots, k\right\}, \quad M = \begin{cases} r\mu, & \mu > 0, \\ 1, & \mu = 0, \end{cases} \tag{2}$$

где $r > 1$ — заданный параметр алгоритма.

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

$$R(1) = 2\Delta_1 - 4\frac{z_1}{M}, \quad R(k+1) = 2\Delta_{k+1} - 4\frac{z_k}{M},$$

2022, **23** (4), 339–349. doi 10.26089/NumMet.v23r421

$$R(i) = \Delta_i + \frac{(z_i - z_{i-1})^2}{M^2\Delta_i} - 2\frac{z_i + z_{i-1}}{M}, \quad 1 < i < k+1,$$

где $\Delta_i = (x_i - x_{i-1})^{\frac{1}{N}}$.

Шаг 4. Упорядочить характеристики $R(i)$, $1 \leqslant i \leqslant k+1$, в порядке невозрастания:

$$R(t_1) \geqslant R(t_2) \geqslant \ldots \geqslant R(t_k) \geqslant R(t_{k+1}),$$

и выбрать p интервалов с номерами t_j, $1 \leqslant j \leqslant p$, с наибольшими значениями характеристики.

Шаг 5. В выбранных интервалах вычислить точки x^{k+j}, $1 \leqslant j \leqslant p$, в соответствии с формулами

$$x^{k+j} = \frac{x_{t_j} + x_{t_j-1}}{2}, \quad t_j = 1, \quad t_j = k+1, \tag{3}$$

$$x^{k+1} = \frac{x_{t_j} + x_{t_j-1}}{2} - \operatorname{sign}\left(z_{t_j} - z_{t_j-1}\right)\frac{1}{2r}\left[\frac{\left|z_{t_j} - z_{t_j-1}\right|}{\mu}\right]^N, \quad 1 < t_j < k+1. \tag{4}$$

Очередные испытания проводятся параллельно, в вычисленных по формулам (3), (4) точках x^{k+j}, $1 \leqslant j \leqslant p$. Отметим, что в прикладных оптимизационных задачах процесс проведения испытания является, как правило, значительно более трудоемким по сравнению с работой вычислительных правил алгоритма, поэтому параллельное проведение испытаний ускоряет работу алгоритма.

Алгоритм останавливает свою работу в случае, если хотя бы для одного значения t_j, $1 \leqslant j \leqslant p$, из (2) выполняется условие $\Delta_{t_j} < \varepsilon$. Данный критерий остановки (наряду с обычным для итерационных методов критерием, ограничивающим число выполненных итераций) используется в задачах оптимизации, в которых точка глобального минимума x^* заранее неизвестна.

При решении тестовых задач, в которых точка глобального минимума x^* является известной, можно использовать также и критерий остановки по попаданию в окрестность глобального минимума. В этом случае метод прекращает работу, если хотя бы для одного значения t_j, $1 \leqslant j \leqslant p$, из (2) выполняется условие $\left|x_{t_j} - x^*\right| < \varepsilon$.

В качестве итоговой оценки глобально-оптимального решения рассматриваемой задачи выбираются значения

$$f_k^* = \min_{1 \leqslant i \leqslant k} f(x_i), \quad x_k^* = \arg\min_{1 \leqslant i \leqslant k} f(x_i).$$

3. Теоретические оценки ускорения параллельного алгоритма. Опишем (в соответствии с [8]) теоретические свойства рассматриваемого параллельного алгоритма. В решаемых задачах время выполнения одного испытания существенно превышает время обработки его результата. Поэтому ускорение параллельного алгоритма можно определить как *ускорение по испытаниям* $s(p)$:

$$s(p) = \frac{n(1) \cdot p}{n(p)},$$

где $n(1)$ — количество испытаний, выполненных последовательным алгоритмом, а $n(p)$ — количество испытаний, выполненных параллельным алгоритмом с использованием p параллельных вычислительных устройств. Ускорение по испытаниям $s(p)$ является ключевой характеристикой эффективности параллельного алгоритма глобального поиска.

Отметим, что количество испытаний $n(p)$ для алгоритмов с разной степенью распараллеливания p будут отличаться друг от друга. Действительно, последовательный алгоритм при выборе точки x^{k+1} следующего $(k+1)$-го испытания имеет полную информацию, полученную на предыдущих k итерациях. Параллельный алгоритм выбирает не одну, а p точек $x^{k+1}, \ldots, x^{k+p}$ на основе той же информации. Это означает, что выбор точек x^{k+j}, $1 < j \leqslant p$, производится без информации о результатах испытаний в точках x^{k+i}, $1 \leqslant i < j$. Только первая точка x^{k+1} будет соответствовать точке, выбранной последовательным алгоритмом. Точки других испытаний могут не совпадать с точками, генерируемыми последовательным алгоритмом. Поэтому будем называть такие испытания *избыточными*, а величину

$$\lambda(p) = \begin{cases} (n(p) - n(1))/n(p), & n(p) > n(1) \\ 0, & n(p) \leqslant n(1) \end{cases}$$

будем называть *избыточностью* метода.

2022, **23** (4), 339–349. doi 10.26089/NumMet.v23r421

Следующее утверждение определяет степень распараллеливания p, которая соответствует безызбыточному (т.е. с нулевой избыточностью) распараллеливанию (доказательство утверждения см. в [8]).

Обозначим испытания, генерируемые последовательным и параллельным алгоритмами при решении одной и той же задачи с $\varepsilon = 0$ в условиях остановки, как $\{x^k\}$ и $\{y^m\}$ соответственно.

Теорема. Пусть x^* — точка глобального минимума, а x' — точка локального минимума функции $f(x)$, и пусть выполнены следующие условия:

1. Выполняется неравенство
$$f(x') - f(x^*) \leqslant \delta, \quad \delta > 0. \tag{5}$$
2. Первые $q(l)$ испытаний последовательного и параллельного алгоритмов совпадают, т.е.
$$\{x^1, \ldots, x^{q(l)}\} = \{y^1, \ldots, y^{q(l)}\},$$
где
$$\{x^1, \ldots, x^{q(l)}\} \subset \{x^k\}, \quad \{y^1, \ldots, y^{q(l)}\} \subset \{y^m\}.$$
3. Существует точка $t^n \in \{y^m\}$, $n < q(l)$, такая что $x' \leqslant y^n \leqslant x^*$ или $x^* \leqslant y^n \leqslant x'$.
4. Для величины M из (2) выполняется неравенство
$$M > 2^{2-1/N} H,$$
где H — постоянная Гельдера для одномерной целевой функции.

Тогда параллельный алгоритм при $p = 2$ будет безызбыточным (т.е. $s(2) = 2$, $\lambda(2) = 0$), пока выполняется условие
$$(x_{t_j} - x_{t_j - 1})^{1/N} > \frac{4\delta}{M - 2^{2-1/N} H}, \tag{6}$$
где t_j определяется в соответствии с (2).

Следствие. Пусть целевая функция $f(x)$ имеет Q локальных минимумов $\{x'_1, \ldots, x'_Q\}$, для которых выполняется условие (5), и пусть существуют точки испытания y^{n_i}, $1 \leqslant i \leqslant Q$, такие, что
$$y^{n_i} \in \{y^1, \ldots y^{q(l)}\},$$
$$\alpha_i \leqslant y^{n_i} \leqslant \alpha_{i+1}, \quad \alpha_i, \alpha_{i+1} \in \{x^*, x'_1, \ldots, x'_Q\}, \quad 1 \leqslant i \leqslant Q.$$

Тогда при выполнении условий теоремы параллельный алгоритм со степенью распараллеливания $Q + 1$ будет безызбыточным (т.е. $s(Q+1) = Q + 1$, $\lambda(Q+1) = 0$) до тех пор, пока выполняется условие (6).

Следствие из теоремы играет особую роль при решении многомерных задач, сводящихся к одномерным с помощью отображения $y(x)$. Отображение $y(x)$, являясь аппроксимацией кривой Пеано, характеризуется эффектом “расщепления” точки глобального минимума $y^* \in D$ на несколько (до 2^N) прообразов на отрезке $[0, 1]$. Применяя параллельный алгоритм для минимизации редуцированной одномерной функции, можно ожидать нулевую избыточность при степени распараллеливания до $2^N + 1$.

4. Реализация с использованием Intel oneAPI. Рассматриваемые многомерные многоэкстремальные задачи обладают высокой трудоемкостью численного решения, поскольку при увеличении размерности задачи наблюдается экспоненциальный рост вычислительных затрат. Вместе с тем быстрый прогресс параллельных вычислительных средств предоставляет все больше различных новых возможностей для решения сложных задач. Однако на фоне этого возникает вопрос эффективного распараллеливания, а многообразие различных типов устройств и средств разработки ставят пользователя перед проблемой выбора одной из них.

В общем случае для обеспечения высокой производительности вычислений в рамках решения новых задач требуются различные вычислительные архитектуры. Например, компанией Intel предлагается следующая классификация ускорителей: скалярные (CPU), векторные (GPU), матричные и программируемые логические интегральные схемы (ПЛИС)[1]. Эти классы отличаются архитектурой, которая прямым образом влияет на их функционал.

[1]Далее в статье вместо ПЛИС используется сокращение FPGA от английского названия этого класса устройств — field-programmable gate array.

2022, **23** (4), 339–349. doi 10.26089/NumMet.v23r421

Рассмотрим их по порядку. Скалярные CPU являются кэш-ориентированными, оптимизированными для достижения максимальной однопоточной производительности и предназначены для решения задач общего назначения. Несмотря на то что большинство процессоров сейчас многоядерные, достаточно много ресурсов тратится на обеспечение их однопоточной производительности. В GPU большая часть ресурсов задействована под вычисления [12], а не под кэш, как в случае с CPU, поэтому в GPU применима стратегия SIMD[13]. Третий класс — это матричные процессоры, рассчитанные на быструю работу с матрицами. Прежде всего это ускорители из области искусственного интеллекта, нейронные процессоры, например процессоры машинного зрения, тензорные и другие. И последний класс — FPGA [14]. Основу структуры составляет матрица логических элементов, функции этих элементов и связи между ними могут модифицироваться (программироваться) в процессе использования. Сфера применения ПЛИС достаточно широка, они используются в бытовой электронике, телекоммуникационном оборудовании, разнообразной робототехнике и при прототипировании микросхем.

Однако использование преимуществ нескольких типов архитектур является сложной задачей для разработчиков. Для каждой архитектуры требуются разные языки, отдельные инструменты, а повторное использование кода ограничено. Это делает разработку сложной, дорогостоящей и трудоемкой.

Например, когда возникает потребность выполнения алгоритма на нескольких типах ускорителей, при написании кода программы возникают проблемы, связанные с несовместимостью разных средств разработки с различными архитектурными особенностями ускорителей. Для более четкого понимания этого проведем небольшой обзор инструментов и средств разработки для программирования ускорителей. Для GPU часто используются графические API и шейдерные языки: DirectX, OpenGL, Vulcan, Metal. Некоторые производители, к примеру NVidia и AMD, создают свои специальные средства, которые подходят под их ускорители (NVidia CUDA, AMD ROCm). Для программирования под FPGA применяются языки описания архитектур, например Verilog и VHDL. Также есть набор общих средств, которые отличаются по применимости, сложности написания кода, текущей поддержке: OpenMP, OpenCL, Python, SYCL и другие.

Как правило, если остановиться на одной из вычислительных архитектур, то для возможности запуска на абсолютно другой может потребоваться адаптация части кода, а может, даже придется написать программу с нуля, применяя другие инструменты. В большинстве случаев будет создан новый проект и необходимо будет поддерживать несколько программ, в которых используются разные технологии.

Для создания универсального кода, работающего на различных устройствах, можно воспользоваться набором инструментов Intel oneAPI. Он прост, открыт и позволяет разработчику обеспечивать высокую производительность в разных архитектурах. А поскольку oneAPI основан на стандартах и открытых спецификациях, риски при переносе снижаются. Это дает возможность один раз написать код и в дальнейшем запустить его на другом устройстве. Также к преимуществам oneAPI можно отнести возможность применения в различных прикладных решениях, например в задачах машинного обучения.

В рамках данной задачи необходимы возможности распараллеливания, которые обеспечиваются за счет включения в oneAPI языка Data Parallel C++ и набора библиотек, облегчающих межархитектурную разработку. Data Parallel C++ основан на широко известном языке C++ и включает в себя SYCL от группы Khronos и расширения от комьюнити.

Все это позволяет повторно использовать код в разных архитектурах и выполнять пользовательскую настройку ускорителей. Это дает разработчикам гибкость, позволяющую отказаться от патентованных подходов, и открывает возможности для использования аппаратных средств, которые ранее были невозможны.

Руководствуясь приведенным ранее алгоритмом, реализована возможность одновременно вычислять сразу несколько значений целевой функции, используя при этом инструменты распараллеливания Intel oneAPI. Также имеется возможность выбрать устройство, на котором будут проходить вычисления значений функции. Остальные этапы рассмотренного алгоритма необходимо проводить последовательно, потому что в процессе их выполнения проходит работа с достаточно большим количеством накопленной ранее поисковой информации, в связи с этим реализуем их на CPU.

Таким образом, первые этапы рассматриваемого алгоритма выполняются на центральном процессоре CPU. Далее, полученные в ходе выполнения одной итерации новые p координат из интервалов с наибольшими характеристиками передаются с помощью промежуточного буфера на устройство (CPU, GPU, FPGA), выбранное для исполнения распараллеленного блока кода, для вычисления значения функции в них. Затем найденные значения функции в этих точках передаются через промежуточный буфер обратно

2022, **23** (4), 339–349. doi 10.26089/NumMet.v23r421

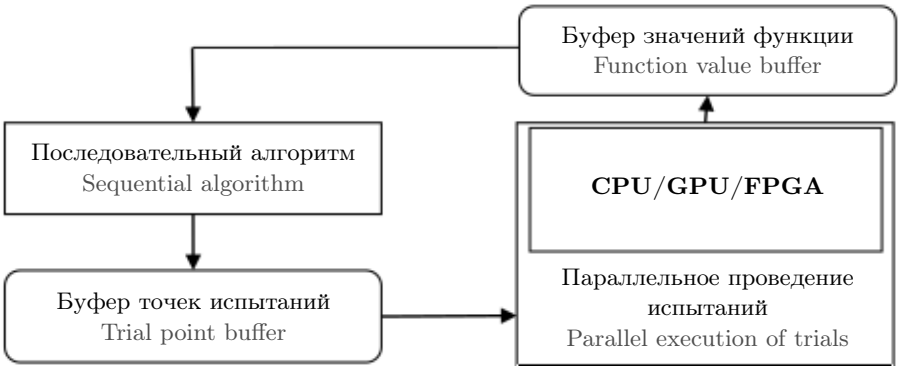

Рис. 1. Общая схема организации параллельных вычислений

Fig. 1. General scheme of parallel computing

на центральный процессор. Общая схема организации параллельных вычислений приведена на рис. 1.

5. Результаты численных экспериментов. Вычислительные эксперименты были проведены на компьютере с процессором Intel Core i5-10600 3.3GHz, 32 GB оперативной памяти и графической картой Intel UHD Graphics 630. Для получения исполняемого программного кода использовался компилятор Intel oneAPI DPC++ 2021.1. Вычислительные эксперименты выполнялись с использованием программной системы Globalizer [15].

Большинство известных тестовых задач из области многомерной глобальной оптимизации характеризуются небольшим временем вычисления значений целевой функции, которое сопоставимо с временем работы расчетных правил алгоритма. В прикладных оптимизационных задачах вычисление значений функции является трудоемкой операцией, здесь время работы алгоритма будет значительное меньше времени проведения одного испытания. В связи с этим нами была предложена модификация существующих тестовых задач, в которых требуется провести интегрирование исходной тестовой функции по части параметров. Для этого изначально порождается функция $\psi(y)$, $y \in \mathbb{R}^{2N}$, размерности $2N$, в которой первые N координат фиксируются, а по остальным производится численное интегрирование по области определения функции. Для интегрирования используется метод средних прямоугольников

$$\varphi\left(y_1,\ldots,y_N\right) = \int\limits_{a_{N+1}}^{b_{N+1}} \int\limits_{a_{N+2}}^{b_{N+2}} \cdots \int\limits_{a_{2N}}^{b_{2N}} \psi\left(y_1,\ldots,y_N,y_{N+1},\ldots,y_{2N}\right)\, dy_{N+1}dy_{N+2}\ldots dy_{2N} \;=$$

$$\sum_{i_1=0}^{M-1}\sum_{i_2=0}^{M-1}\cdots\sum_{i_N=0}^{M-1}\left(\left(\prod_{j=1}^{2N} h_j\right)\psi\left(y_1,y_2,\ldots,y_N,\; y_{(N+1)_{i_1}},y_{(N+2)_{i_2}},\ldots,y_{(2N)_{i_N}}\right)\right),$$

$$D = \left\{y \;\in\; \mathbb{R}^{2N} :\; a_i \leqslant y_i \leqslant b_i,\; 1 \leqslant i \leqslant 2N\right\},$$

$$y_{(N+k)_{i_k}} = a_k + i_k h_k + \frac{h_k}{2},$$

$$h_k = (b_k - a_k)/M.$$

Здесь M — количество участков интегрирования по одной координате, а $\psi(y)$ — исходная тестовая функция. Очевидно, чем больше значение M, тем больше проводится вычислений. Изменяя число узлов в сетке интегрирования по всем координатам или число участков по одной, можно регулировать время выполнения вычислений.

Вначале проведем вычислительные эксперименты с использованием классической тестовой функции Растригина

$$\psi\left(y_1,\ldots,y_{2N}\right) = \sum_{i=1}^{2N}\left(y_i^2 - 10\cos\left(2\pi y_i\right) + 10\right),$$

$$-2.2 < y_i < 1.8, \quad 1 \leqslant i \leqslant 2N.$$

2022, **23** (4), 339–349. doi 10.26089/NumMet.v23r421

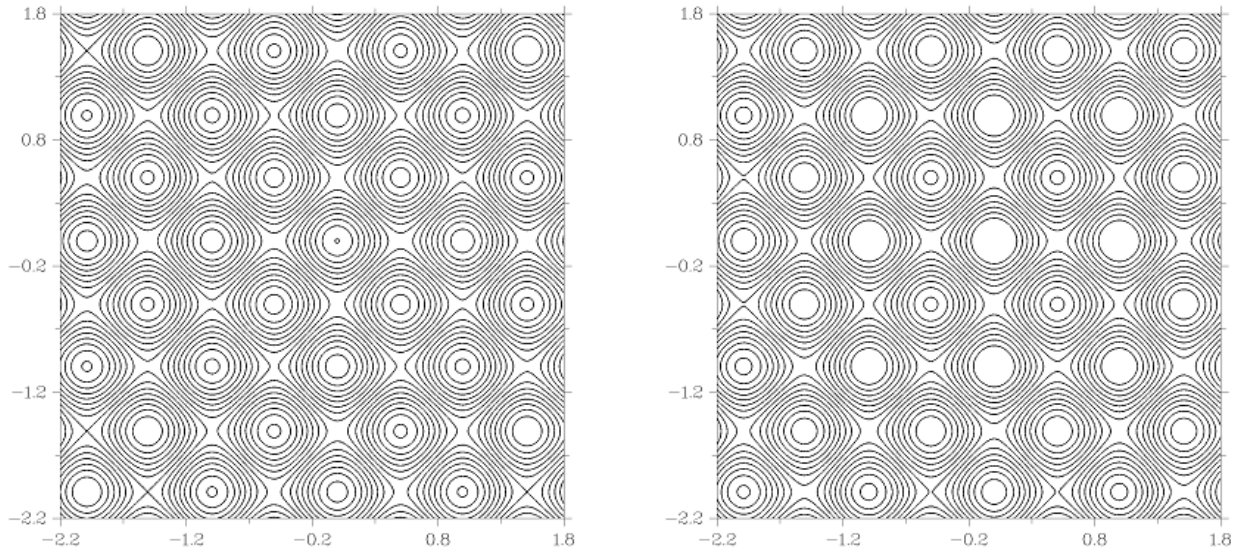

Рис. 2. Линии уровня двумерной функции Растригина (слева) и модифицированной четырехмерной функции Растригина (справа)

Fig. 2. Level lines of the two-dimensional Rastrigin function (left) and the modified four-dimensional Rastrigin function (right)

На рис. 2 изображены линии уровня двумерной функции Растригина (слева) и модифицированной функции Растригина, в которой проводилось интегрирование по двум переменным (справа). Как можно видеть, новая функция не сильно отличается от исходной — ее глобальный минимум остался прежним, в точке $(0, 0)$.

При проведении экспериментов количество испытаний параллельного алгоритма глобального поиска (ПАГП) было ограничено величиной $K_{\max} = 10^6$, точность поиска установлена $\varepsilon = 0.01$, использовался параметр метода $r = 3$. Размерность решаемых задач была $N = 4$ и $N = 5$. При вычислениях на CPU число потоков P варьировалось от 1 до 8, а при вычислении на GPU — от 256 до 1024. Так как решались тестовые задачи, то использовался критерий остановки по попаданию в окрестность известного решения. В табл. 1 приведено число итераций ПАГП, в табл. 2 — его ускорение по сравнению с однопоточным запуском.

Далее были проведены эксперименты на серии задач, полученных интегрированием функций из генератора GKLS. Данный генератор описан в [6]. С его помощью можно порождать задачи многоэкс-

Таблица 1. Число итераций ПАГП при минимизации модифицированной функции Растригина

Table 1. The number of iterations of PAGP when minimizing modified Rastrigin function

N	CPU				GPU		
	$P = 1$	$P = 2$	$P = 4$	$P = 8$	$P = 256$	$P = 512$	$P = 1024$
4	61231	26592	21007	6728	340	301	92
5	703548	328141	258351	99524	4642	2194	995

Таблица 2. Ускорение ПАГП при минимизации модифицированной функции Растригина

Table 2. Acceleration of PAGP when miminimizing modified Rastrigin function

N	CPU			GPU		
	$P = 2$	$P = 4$	$P = 8$	$P = 256$	$P = 512$	$P = 1024$
4	2.1	2.1	5.6	2.8	3.1	9.4
5	1.9	2.0	4.3	2.4	4.9	9.7

2022, **23** (4), 339–349. doi 10.26089/NumMet.v23r421

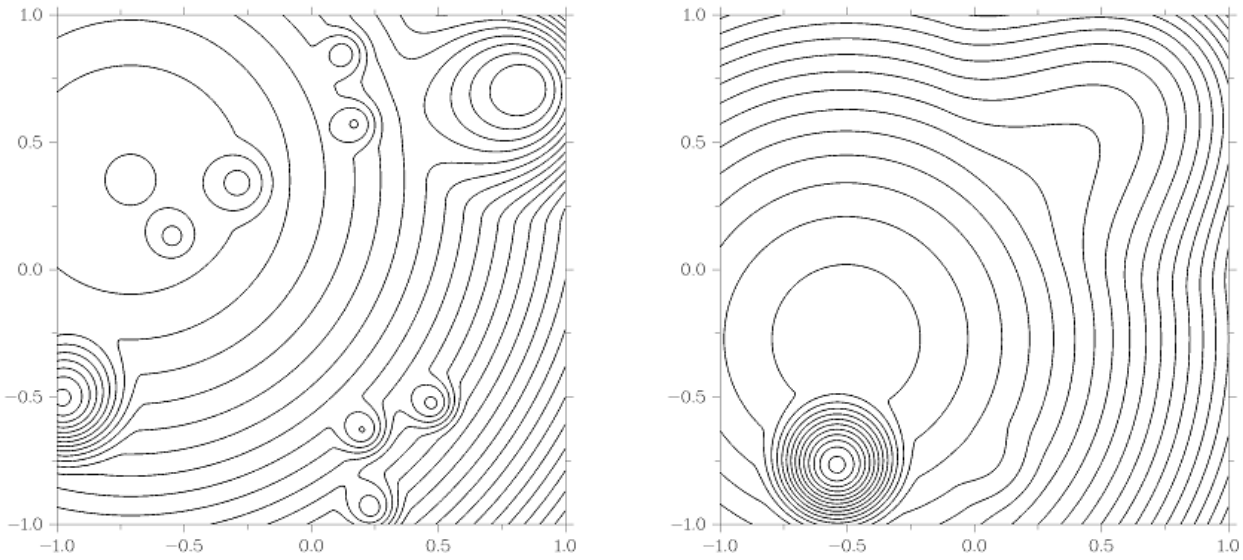

Рис. 3. Линии уровня функции GKLS (слева) и модифицированной функции GKLS (справа)

Fig. 3. Level lines of the GKLS function (left) and modified GKLS function (right)

Таблица 3. Ускорение ПАГП при решении серии модифицированных функций GKLS

Table 3. Acceleration of PAGP when solving a series of modified GKLS functions

N	CPU			GPU		
	$P = 2$	$P = 4$	$P = 8$	$P = 256$	$P = 512$	$P = 1024$
4	1.9	3.0	1.5	1.2	2.3	4.5
5	1.9	3.0	1.6	1.2	2.3	4.5

тремальной оптимизации и варьировать их свойства: размерность, количество локальных минимумов, размеры их областей притяжения, координаты точки глобального минимума, значение функции в ней и т.д. На рис. 3 изображены линии уровня двумерной функции GKLS (слева) и модифицированной функции GKLS, полученной интегрированием четырехмерной функции по последним двум параметрам (справа).

Число итераций ПАГП было ограничено величиной 10^5, точность поиска $\varepsilon = 0.01$, параметр $r = 4$. Было сгенерировано две серии по 100 задач размерности $N = 4$ и $N = 5$. При вычислениях на CPU число потоков P варьировалось от 1 до 8, а при вычислении на GPU — от 256 до 1024. В табл. 3 приведено ускорение по сравнению с однопоточным запуском.

6. Заключение. Полученные результаты подтверждают, что использование инструментов Intel oneAPI для реализации параллельного алгоритма глобального поиска позволяет написать одну версию кода, которая будет показывать хорошее ускорение как при использовании центрального процессора, так и при использовании графической карты.

Тестирование параллельного алгоритма было проведено при решении нескольких серий сложных задач многомерной многоэкстремальной оптимизации.

Работа рекомендована Программным комитетом международной конференции “Суперкомпьютерные дни в России” (26–27 сентября, 2022 г.).

2022, **23** (4), 339–349. doi 10.26089/NumMet.v23r421

Список литературы

1. *Kutov D.C., Sulimov A.V., Sulimov V.B.* Supercomputer docking: investigation of low energy minima of protein–ligand complexes // Supercomputing Frontiers and Innovations. 2018. **5**, N 3. 134–137. doi 10.14529/jsfi180326.
2. *Абгарян К.К., Посыпкин М.А.* Применение оптимизационных методов для решения задач параметрической идентификации потенциалов межатомного взаимодействия // Ж. вычисл. матем. и матем. физ. 2014. **54**, № 12. 1994–2001. doi 10.7868/S0044466914120023.
3. *Евтушенко Ю.Г., Лурье С.А., Посыпкин М.А., Соляев Ю.О.* Применение методов оптимизации для поиска равновесных состояний двумерных кристаллов // Ж. вычисл. матем. и матем. физ. 2016. **56**, № 12. 2032–2041. doi 10.7868/S0044466916120097.
4. *Pintér J.D.* Global optimization: scientific and engineering case studies. New York: Springer, 2006.
5. *Gubaydullin I., Enikeeva L., Barkalov K., Lebedev I.* Parallel global search algorithm for optimization of the kinetic parameters of chemical reactions // Communications in Computer and Information Science. Vol. 1510. Cham: Springer, 2021. 198–211. doi 10.1007/978-3-030-92864-3_16.
6. *Сергеев Я.Д., Квасов Д.Е.* Диагональные методы глобальной оптимизации. М.: Физматлит, 2008.
7. *Paulavicius R., Zilinskas J.* Simplicial global optimization. New York: Springer, 2014. doi 10.1007/978-1-4614-9093-7.
8. *Strongin R.G., Sergeyev Ya.D.* Global optimization with non-convex constraints: sequential and parallel algorithms. Berlin: Springer, 2000.
9. *Sergeyev Ya.D., Strongin R.G, Lera D.* Introduction to global optimization exploiting space-filling curves. New York: Springer, 2013. doi 10.1007/978-1-4614-8042-6.
10. *Стронгин Р.Г., Гергель В.П., Гришагин В.А., Баркалов К.А.* Параллельные вычисления в задачах глобальной оптимизации. М.: Изд-во Моск. ун-та, 2013.
11. *Sovrasov V.* Comparison of several stochastic and deterministic derivative-free global optimization algorithms // Lecture Notes in Computer Science. Vol. 11548. Cham: Springer, 2019. 70–81. doi 10.1007/978-3-030-22629-9_6.
12. *Боресков А.В., Харламов А.А., Марковский Н.Д., Микушин Д.Н., Мортиков Е.В., Мыльцев А.А., Сахарных Н.А., Фролов В.А.* Параллельные вычисления на GPU. Архитектура и программная модель CUDA. М.: Изд-во Моск. ун-та, 2015.
13. *Таненбаум Э., Остин Т.* Архитектура компьютера. Санкт-Петербург: Питер, 2014.
14. *Комолов Д.А., Мяльк Р.А., Зобенко А.А., Филиппов А.С.* Системы автоматизированного проектирования фирмы Altera MAX+Plus II и Quartus II. М.: РадиоСофт, 2002.
15. *Gergel V., Barkalov K., Sysoyev A.* Globalizer: a novel supercomputer software system for solving time-consuming global optimization problems // Numerical Algebra, Control and Optimization. 2018. **8**, N 1. 47–62. doi 10.3934/naco.2018003.

Поступила в редакцию
17 октября 2022 г.

Принята к публикации
2 ноября 2022 г.

Информация об авторах

Константин Александрович Баркалов — д.т.н., профессор; Нижегородский государственный университет имени Н. И. Лобачевского, просп. Гагарина, д. 23, 603022, Нижний Новгород, Российская Федерация.

Илья Генадьевич Лебедев — заведующий лабораторией; Нижегородский государственный университет имени Н. И. Лобачевского, просп. Гагарина, д. 23, 603022, Нижний Новгород, Российская Федерация.

Янина Вадимовна Силенко — лаборант; Нижегородский государственный университет имени Н. И. Лобачевского, просп. Гагарина, д. 23, 603022, Нижний Новгород, Российская Федерация.

2022, **23** (4), 339–349. doi 10.26089/NumMet.v23r421

References

1. D. C. Kutov, A. V. Sulimov, and V. B. Sulimov, "Supercomputer Docking: Investigation of Low Energy Minima of Protein–Ligand Complexes," Supercomput. Front. Innovs. **5** (3), 134–137 (2018). doi 10.14529/jsfi180326.
2. K. K. Abgaryan and M. A. Posypkin, "Optimization Methods as Applied to Parametric Identification of Interatomic Potentials," Zh. Vychisl. Mat. Mat. Fiz. **54** (12), 1994–2001 (2014) [Comput. Math. Math. Phys. **54** (12), 1929–1935 (2014)]. doi 10.1134/S0965542514120021.
3. Yu. G. Yevtushenko, S. A. Lurie, M. A. Posypkin, and Yu. O. Solyaev, "Application of Optimization Methods for Finding Equilibrium States of Two-Dimensional Crystals," Zh. Vychisl. Mat. Mat. Fiz. **56** (12), 2032–2041 (2016) [Comput. Math. Math. Phys. **56** (12), 2001–2010 (2016)]. doi 10.1134/S0965542516120083.
4. J. D. Pintér (Ed.), *Global Optimization: Scientific and Engineering Case Studies* (Springer, New York, 2006).
5. I. Gubaydullin, L. Enikeeva, K. Barkalov, and I. Lebedev, "Parallel Global Search Algorithm for Optimization of the Kinetic Parameters of Chemical Reactions," in *Communications in Computer and Information Science* (Springer, Cham, 2021), Vol. 1510, pp. 198–211. doi 10.1007/978-3-030-92864-3_16
6. Ya. D. Sergeev and D. E. Kvasov, *Diagonal Methods of Global Optimization* (Fizmatlit, Moscow, 2008) [in Russian].
7. R. Paulavicius and J. Zilinskas, *Simplicial Global Optimization* (Springer, New York, 2014). doi 10.1007/978-1-4614-9093-7.
8. R. G. Strongin and Ya. D. Sergeyev, *Global Optimization with Non-Convex Constraints: Sequential and Parallel Algorithms* (Springer, Berlin, 2000).
9. Ya. D. Sergeyev, R. G. Strongin, and D. Lera, *Introduction to Global Optimization Exploiting Space-Filling Curves* (Springer, New York, 2013). doi 10.1007/978-1-4614-8042-6.
10. R. G. Strongin, V. P. Gergel, V. A. Grishagin, and K. A. Barkalov, *Parallel Computations in Global Optimization Problems* (Mosk. Gos. Univ., Moscow, 2013) [in Russian].
11. V. Sovrasov, "Comparison of Several Stochastic and Deterministic Derivative-Free Global Optimization Algorithms," in *Lecture Notes in Computer Science* (Springer, Cham, 2019), Vol. 11548, pp. 70–81. doi 10.1007/978-3-030-22629-9_6.
12. A. V. Boreskov, A. A. Kharlamov, N. D. Markovsky, et al., *Parallel Computing on the GPU. Architecture and Programming Model of CUDA* (Mosk. Gos. Univ., Moscow, 2015) [in Russian].
13. A. S. Tanenbaum and T. Austin, *Structured Computer Organization* (Pearson, New York, 2006; Piter, Saint Petersburg, 2014).
14. D. A. Komolov, R. A. Myalk, A. A. Zobenko, and A. S. Filippov, *Computer-Aided Design Systems from Altera MAX+Plus II and Quartus II* (RadioSoft, Moscow, 2002) [in Russian].
15. V. Gergel, K. Barkalov, and A. Sysoyev, "Globalizer: A Novel Supercomputer Software System for Solving Time-Consuming Global Optimization Problems," Numer. Algebra Control Optim. **8** (1), 47–62 (2018). doi 10.3934/naco.2018003.

Received
October 17, 2022

Accepted for publication
November 2, 2022

Information about the authors

Konstantin A. Barkalov — Dr. Sci., professor; Lobachevsky State University of Nizhny Novgorod, Gagarin prospekt, 23, 603022, Nizhny Novgorod, Russia.

Ilya G. Lebedev — head of laboratory; Lobachevsky State University of Nizhny Novgorod, Gagarin prospekt, 23, 603022, Nizhny Novgorod, Russia.

Yanina V. Silenko — laboratory assistant; Lobachevsky State University of Nizhny Novgorod, Gagarin prospekt, 23, 603022, Nizhny Novgorod, Russia.