**Решение трудоемких задач многомерной глобальной оптимизации с использованием набора инструментов  
Intel oneAPI [[1]](#footnote-1)1\***

К.А. Баркалов, И.Г. Лебедев, Я.В. Кольтюшкина

Нижегородский государственный университет им. Н.И. Лобачевского

В рамках данной статьи рассматривается решение серии сложно вычислимых задач глобальной оптимизации. В качестве целевой выбирается функция, удовлетворяющая условию Липшица с заранее не известной константой. Инструментом для реализации параллельного алгоритма выбран Intel oneAPI, который позволяет писать один код как для центрального процессора, так и для графического ускорителя. В решении многомерных задач применяется метод, построенный на идее редукции размерности при помощи кривой Пеано, которая непрерывно и однозначно отображает отрезок вещественной оси на гиперкуб.

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

# 1. Постановка задачи

Задачи поиска экстремумов многоэкстремальных функций часто возникают в различных областях науки, таких как химия, физика, машиностроение и т.д. В рамках данной работы решается задача минимизации многомерной функции в гиперинтервале Будем предполагать, что информация о виде самой функции недоступна, целевая функция задается в формате «черного ящика». Дополнительно для целевой функции предполагаем выполнение условия Липшица, причем константа априори неизвестна.

|  |  |  |
| --- | --- | --- |
|  |  | (1) |
|  |  | (2) |
|  |  | (3) |

Сведение многомерной задачи (1) к одномерной проводится за счет использования редукции размерности при помощи кривой Пеано [1, 2] , которая непрерывно и однозначно отображает на *N*-мерный гиперкуб отрезок вещественной оси [0,1]:

|  |  |  |
| --- | --- | --- |
|  |  | (4) |
|  |  | (5) |

У данного способа редукции размерности имеется важное свойство, необходимое для последующих расчетов: ограниченность относительных разностей функции сохраняется, т.е. если в области функция удовлетворяла условию Липшица, то на интервале функция будет удовлетворять равномерному условию Гельдера

|  |  |  |
| --- | --- | --- |
|  |  | (6) |
|  |  | (7) |

Пользуясь этим свойством, можно трактовать исходную задачу, как минимизацию одномерной гельдеровой функции.

Факт того, что функция задана как «черный ящик», сокращает число подходящих алгоритмов. Для исследования выбран алгоритм глобального поиска [3, 4], который является одним из эффективных методов глобальной оптимизации, т.к. при решении многих задач опережает (по числу итераций) другие методы аналогичного назначения [5 – 9]. Алгоритм глобального поиска относится к классу характеристических алгоритмов оптимизации, хорошо масштабируется и отлично подходит для работы на параллельных вычислительных кластерах.

В процессе своей работы алгоритм порождает последовательность точек , в каждой из которых проходит вычисление значения минимизируемой функции . Далее под *испытанием* мы будем понимать подразумевать процесс вычисления значения одномерной функции, в который, помимо прочего, включается построение образа , а *результатом испытания* назовем пару . Процесс распараллеливания построен таким образом, что при выполнении одной итерации метода одновременно проходят испытаний, . Введем обозначение , как общее число испытаний, которые были проведены после выполненных параллельных итераций. Приведем последовательность этапов параллельного алгоритма глобального поиска с модифицированием для решения задач с функциями, удовлетворяющими условию Гельдера:

Подготовительный этап. Изначально проводится испытание в произвольной внутренней точке Когда выполнено итераций, и соответствующее им число испытаний в точках то точки испытаний последующей () итерации будут определять следующим образом:

1 этап. Упорядочить по координате точки уже проведенных испытаний и граничные точки:

|  |  |  |
| --- | --- | --- |
|  |  | (8) |

2 этап. Вычислить текущее значение , оценивающее неизвестную константу Липшица :

|  |  |  |
| --- | --- | --- |
|  |  | (9) |

где заданный параметр метода.

3 этап. Найти значение характеристики для всех интервалов ,

|  |  |  |
| --- | --- | --- |
|  |  | (10) |
|  |  | (11) |
|  |  | (12) |

где .

4 этап. Упорядочить характеристики в порядке невозрастания

|  |  |  |
| --- | --- | --- |
|  |  | (13) |

и выбрать интервалов с номерами , значение характеристики в которых наибольшее.

5 этап. В выбранных интервалах вычислить точки и провести в них новые испытания:

|  |  |  |
| --- | --- | --- |
|  |  | (14) |
|  |  | (15) |

Важно отметить, что в рассматриваемых задачах самый трудоемкий и времязатратный этап алгоритма – это вычисление значения функции в точке. Поэтому параллельное вычисление этих значений положительно влияет на скорость работы алгоритма.

Используемые критерии остановки:

* по длине интервала, как только происходит выполнение условия хотя бы для какого-то номера . Основной критерий остановки, используется в прикладных задачах оптимизации и в тестовых наборах, для которых точка глобального минимума заранее неизвестна.
* по попаданию в окрестность глобального минимума, , где − точка глобального минимума, если ее значение известно. Используется в тестовых задачах, когда известна до начала работы алгоритма.

Как оценка глобально-оптимального решения рассматриваемой задачи (1) выбираются значения

|  |  |
| --- | --- |
|  | (16) |

# 2. 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 процесс развития современных вычислительных средств, включая распараллеливание, предоставляет всё больше различных новых возможностей для решения проблем оптимизации. Однако на фоне этого возникает задача эффективного распараллеливания, а многообразие различных типов современных ускорителей и средств разработки для них дают пользователям немалый выбор [10].

Но перейдя от частной задачи к общей ситуации, разберем, какие предлагаются варианты решения. Для обеспечения высокой производительности вычислений в рамках каких-то новых рабочих задач требуются различные вычислительные архитектуры. Например, компанией Intel предлагается следующая классификация ускорителей: скалярные (CPU), векторные (GPU), матричные и ПЛИС (FPGA). Эти классы отличаются архитектурой, которая прямым образом влияет на их функционал.

Рассмотрим их по порядку. Скалярные (CPU), являются кэш-ориентированными, основаны на оптимизации однопоточной производительности и на решение 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 [11], а не под кэш, как в случае с CPU, поэтому в GPU применима стратегия SIMD[12]. 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 – программируемые логические интегральные схемы (ПЛИС), FPGA [13]. 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 потребности выполнения алгоритма на нескольких типах ускорителях, то при написании кода программы, возникают проблемы связанные с несовместимости разных средств разработки и разными архитектурными особенностиями. 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 С++ 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. Что 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 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 на центральный процессор. Общая схема организации параллельных вычислений приведена на рис. 1.

Параллельное проведение испытаний

Последовательный алгоритм

Буфер точек испытания

**CPU / GPU / FPGA**

Буфер значений функции

**Рис. 1**. Общая схема организации параллельных вычислений.

# 3. Результаты численных экспериментов

В рамках работы были проведены вычислительные эксперименты на персональном компьютере с процессором Intel Core i5-10600 3.3GHz, 32 GB оперативной памяти и встроенной графической картой Intel UHD Graphics 630. Для получения исполняемого программного кода использовался компилятор Intel oneAPI DPC++ 2021.1. Вычислительные эксперименты выполнялись с использованием разработанного программного комплекса Globalizer [14, 15].

Большинство известных тестовых задач из области многомерной глобальной оптимизации характеризуются небольшим временем вычисления значений целевой функции. Из-за этого при проведении экспериментов бывает сложно понять с чем может быть связано относительно большое время выполнения параллельного алгоритма: с появившимися, сравнительно с последовательной версией, накладными расходами или причина в неэффективном распараллеливании. В реальных же задачах вычисление функции самая трудоемкая и время затратная операция, поэтому такой проблемы не возникает.

Нами предлагается вычислительно трудоемкая модификация существующих тестовых задач, расчет целевой функции заключается в интегрировании исходной тестовой функции по части параметров. Для этого изначально порождается задача, размерность которой в два раза больше искомой. При вычислении значения функции, первые координат фиксируются, а по остальным производится численное интегрирование по области определения функции. Для интегрирования используется метод средних прямоугольников.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | | (17) |
|  | |  | (18) |

|  |  |  |
| --- | --- | --- |
|  |  | (19) |

где  *−* количество участков интегрирования по одной координате, а – исходная тестовая функция. Очевидно, чем больше значение , 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 на классической задаче – функции Растригина. Она задается простой формулой, сепарабельна и не имеет ограничений по размерности.

|  |  |
| --- | --- |
|  | (20) |

На рис. 2 изображены линии уровня двухмерной функции Растригина (слева) и интегрированной четырехмерной функции Растригина (справа). Как можно видеть, новая функция не сильно отличается от двухмерного оригинала, и даже глобальный минимум остался прежним – точка (0, 0).

|  |  |
| --- | --- |
|  |  |

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

Число итераций параллельного алгоритма глобального поиска (ПАГП) было ограничено 1000000, точность поиска и параметр метода . Размерность задачи = 4 и 5. При вычислениях на CPU число потоков варьировалось от 1 до 8, а при вычислении на GPU от 256 до 1024. Использовался критерий остановки по попаданию в окрестность. В таблице 1 приведено число итераций ПАГП, в таблице 2 – ускорение по сравнению с однопоточным запуском.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *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 1. Число итераций ПАГП при решении интегрированной функции Растригина.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *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 |

Таблица 2. Ускорение ПАГП при решении интегрированной функции Растригина

Как можно видеть из полученных результатов, использование инструментов Intel oneAPI, при распараллеливании алгоритма глобального поиска, показала хорошие результаты на тестовой функции.

Далее проведем эксперименты на серии задач полученных интегрированием функций из генератора GKLS. Данный генератор описан в работах [16, 17], дает возможность создавать серии задач многоэкстремальной оптимизации и заранее задавать их свойства, такие как: размерность задачи, количество локальных минимумов, размеры их областей притяжения, координата точки глобального минимума, значение функции в ней и т.п. На рис. 3 изображены линии уровня двухмерной функции GKLS номер 9 (слева) и интегрированной функции GKLS номер 9, полученной интегрированием четырехмерной функции по последним двум параметрам (справа).

|  |  |
| --- | --- |
|  |  |

**Рис. 3**. Функции GKLS (слева) и интегрированной функции GKLS (справа).

Число итераций ПАГП было ограничено 100000, точность поиска и параметр метода . Размерность задачи = 4 и 5. При вычислениях на CPU число потоков варьировалось от 1 до 8, а при вычислении на GPU от 256 до 1024. В таблице 3 приведено ускорение по сравнению с однопоточным запуском.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *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 |

Таблица 3. Ускорение ПАГП при решении серии интегрированных функций GKLS.

# Заключение

Написать одну программу, которая в дальнейшем может запуститься на различных устройствах, очевидно проще, чем создавать для каждого инструмента свой проект. При использовании Intel oneAPI, не потребуется портирование программы на другое устройство, и можно будет избежать связанных с ним проблем. В современном мире данный подход приобретает всё большую актуальность, так как быстрый технологический прогресс способствует более частому изменению архитектуры устройств, их усовершенствованию, появлению новых технологий. Всё это только усложняет процесс переноса программ.

Рассмотренный в рамках данной работы параллельный алгоритм решения задач многомерной многоэкстремальной оптимизации был реализован с использованием набора инструментов Intel oneAPI. Для проверки работоспособности параллельного алгоритма была предложена вычислительно трудоемкая модификация существующих тестовых задач. Эксперименты, проведенные на разных архитектурах, подтвердили целесообразность использования инструментов Intel oneAPI для распараллеливания алгоритмов глобальной оптимизации.

# Литература

1. Sergeyev, Ya.D. Introduction to global optimization exploiting space-filling curves / Ya.D. Sergeyev, R.G. Strongin, D. Lera – Springer, 2013. 125 p.
2. Strongin, R.G. Global Optimization with Non-convex Constraints. Sequential and Parallel Algorithms / R.G. Strongin, Ya.D. Sergeyev – Kluwer Academic Publishers, 2000. 704 p.
3. Стронгин, Р.Г. Параллельные вычисления в задачах глобальной оптимизации / Р.Г. Стронгин, В.П. Гергель, В.А. Гришагин, К.А. Баркалов – М.: Издательство Московского университета, 2013. 280 с.
4. Pinter (Ed.), J.D. Global Optimization: Scientific and Engineering Case Studies / J.D. Pinter – Springer, 2006. 546 p.
5. Стронгин, Р.Г. Параллельные методы решения задач глобальной оптимизации / Р.Г. Стронгин, В.П. Гергель, К.А Баркалов, // Известия высших учебных заведений. Приборостроение, 2009. Т. 52. № 10. С. 25 – 33.
6. Захарова, Е. Обзор методов многомерной оптимизации. / Е.М. Захарова, И.К. Минашина, // Информационные процессы, 2014. Т 14. №3. С. 256 – 274.
7. Gaviano, M. Software for generation of classes of test functions with known local and global minima for global optimization/ M. Gaviano, D. Lera, D. E. Kvasov, Y. D. Sergeyev // ACM Transactions on Mathematical Software. – 2003. – Vol. 29. – P. 469-480.
8. Сергеев, Я.Д. Диагональные методы глобальной оптимизации / Я.Д. Сергеев, Д.Е. Квасов – М.: Физматлит, 2008. – 352 c.
9. Gablonsky, J.M. A Locally-Biased Form of the DIRECT Algorithm / J.M. Gablonsky, C.T. Kelley, // Journal of Global Optimization, – 2001, Vol. 21, No. 1, – P. 27–37.
10. Paulavicius, R. Parallel branch and bound for global optimization with combination of Lipschitz bounds / R. Paulavicius, J. Zilinskas and A. Grothey // Optimization Methods & Software, 2011. – Vol. 26, No. 3. P. 487–498.
11. Боресков А.А. Параллельные вычисления на GPU. Архитектура и программная модель CUDA / Боресков А.А., Харламов А.А., Марковский Н.Д., Микушин Д.Н., Мортиков Е.В., Мыльцев А.А., Сахарных Н.А., Фролов В.А. – М.: Издательство Московского университета, 2015. 336 с.
12. Таненбаум, Э. Архитектура компьютера / Э. Таненбаум, Т. Остин; перевод с английского Е. Матвеева. — 6-е изд. — Санкт-Петербург [и др.]: Питер, 2014. 816 с.
13. Комолов Д.А. Системы автоматизированного проектирования фирмы Altera MAX+Plus II и Quartus II. / Комолов Д.А. Мяльк Р.А. Зобенко А.А. Филиппов А.С., Издательство РадиоСофт, 2002. 355 c.
14. Gergel, V.P. A novel supercomputer software system for solving time-consuming global optimization problems / V.P. Gergel, K.A. Barkalov A.V. Sysoyev // Numerical Algebra, Control and Optimization, 2018. Vol. 8, No. 1. P. 47-62.
15. Sysoyev, A., Barkalov, K., Sovrasov, V., Lebedev, I., Gergel, V. AGS NLP solver. URL: https://github.com/sovrasov/ags\_nlp\_solver
16. Сергеев, Я.Д. Диагональные методы глобальной оптимизации / Я.Д. Сергеев, Д.Е. Квасов – М.: Физматлит, 2008. 352 c.
17. Gaviano, M. Software for generation of classes of test functions with known local and global minima for global optimization/ M. Gaviano, D. Lera, D. E. Kvasov, Y. D. Sergeyev // ACM Transactions on Mathematical Software, 2003. Vol. 29. P. 469-480.

1. 1\* 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 № 075-02-2021-1394). [↑](#footnote-ref-1)