**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 данной статье будет рассмотрено решение серии сложно вычислимых задач глобальной оптимизации. Целевая функция является липшицевой с заранее неизвестной константой. В качестве инструментов для реализации параллельного алгоритма был выбран Intel oneAPI, позволяющий писать один код как для центрального процессора, так и для графического ускорителя. Для решения многомерных задач используется подход, основанный на идее редукции размерности с помощью кривой Пеано, непрерывно и однозначно отображающей отрезок вещественной оси на *N*-мерный гиперкуб.

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

# 1. Постановка задачи

Задача многомерной многоэкстремальной оптимизации может быть определена как проблема поиска наименьшего значения действительной функции в -мерном пространстве. В рамках данной работы рассматривается задача поиска глобального минимума -мерной функции в гиперинтервале Будем предполагать, что мы не знаем ничего о самом виде функции, она задана для нас по принципу «черного ящика». Дополнительно для целевой функции предполагаем выполнение условия Липшица, причем константа априори неизвестна.

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

Чтобы свести многомерную задачу (1) к одномерной, будет использована редукция размерности с помощью кривой Пеано [1, 2] , которая непрерывно и однозначно отображает отрезок вещественной оси [0,1] на *N*-мерный гиперкуб:

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

Редукция размерности с помощью кривой Пеано обладает необходимым для дальнейших выкладок качеством сохранения ограниченности относительных разностей функции: если функция в области удовлетворяла условию Липшица, то функция на интервале будет удовлетворять равномерному условию Гельдера

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

Пользуясь этим свойством, можем трактовать исходную задачу, как минимизацию одномерной функции удовлетворяющей условию Гельдера.

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

В процессе работы рассматриваемого алгоритма строится последовательность точек , в каждой из которых вычисляется значение минимизируемой функции . В дальнейшем будем называть *испытанием* процесс вычисления значения одномерной функции, включающий в себя, в том числе, построение образа , а пару будем называть *результатом испытания*. Распараллеливание организовано следующим образом: в ходе выполнения одной итерации работы метода проводятся испытаний одновременно, где . Обозначим − суммарное число испытаний, проведенных после параллельных итераций. Рассмотрим этапы параллельного алгоритма глобального поиска, модифицированного для решения задач с гёльдеровыми функциями:

Подготовительный этап. Изначально провести испытание в произвольной внутренней точке После выполнения итераций, и соответствующего им числа испытаний в точках точки испытаний следующей () итерации определяются следующим образом:

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

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

2 этап. Вычислить оценку для неизвестной константы Липшица :

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

где заданный параметр метода.

3 этап. Вычислить характеристику для каждого интервала ,

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

где .

4 этап. Упорядочить характеристики в порядке убывания

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

и выбрать наибольших из них с номерами интервалов .

5 этап. В точках провести новые испытания:

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

Используемые критерии остановки:

* по длине интервала, как только выполняется условие хотя бы для одного номера . Основной критерий остановки, используется 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 она известна нам заранее. Используется в тестовых задачах, когда известна до начала работы алгоритма.

В качестве оценки глобально-оптимального решения задачи (1) выбираются значения

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

# 2. Реализация с использованием Intel oneAPI

Для задач многоэкстремальной оптимизации свойственна высокая трудоёмкость численного решения, связанная с экспоненциальным ростом вычислительных затрат при увеличении размерности. Однако быстрое развитие вычислительных средств, в частности, распараллеливание, предоставляет новые возможности для решения оптимизационных проблем. Вместе с этим появляется задача эффективного распараллеливания и разнообразие типов современных ускорителей, а также средств разработки для работы с ними дает пользователям немалый выбор.

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 [6], а не под кэш, как в случае с CPU, поэтому в GPU применима стратегия SIMD (Single Instruction, Multiple Data) [7]. 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 [8]. Основу структуры составляет матрица логических элементов, функции этих элементов и связи между ними могут модифицироваться – программироваться, в процессе использования. FPGA широко применяются в разных устройствах: потребительской электронике, оборудовании телекома, платах-ускорителях для применения в дата-центрах, различной робототехнике, а также 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, если есть необходимость запуска одного и того же проекта на различных ускорителях, то при его создании можно столкнуться с проблемами несовместимости разных средств разработки и разными особенностями архитектуры. 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.

В большинстве случаев, если мы остановимся на какой-то одной вычислительной архитектуре, то для запуска на абсолютно другой сначала потребуется адаптировать код, а, возможно, и написать его с нуля, используя иные инструменты. В общем случае, появится еще один проект, и поддерживать необходимо будет сразу несколько программ, использующих разные технологии.

Одним из вариантов решения этой проблемы является использование 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 позволяет написать код один раз и в дальнейшем запускать его на различных устройствах. 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, реализуем возможность проведения одновременного вычисления сразу нескольких значений целевой функции, воспользовавшись инструментами распараллеливания Intel oneAPI. Причем для запуска можно выбрать, на каком устройстве будут происходить вычисления. Оставшиеся этапы алгоритма должны проводиться последовательно, ко всему прочему для их выполнения необходима работа с большим объемом уже накопленной поисковой информации, эти шаги реализованы на CPU.

В результате, первые четыре этапа, представленного в постановке задачи алгоритма, выполняются на CPU. Полученные в ходе работы координат с наибольшими характеристиками для вычислений новых точек испытания передаются через промежуточный буфер на выбранное для исполнения параллельной части кода устройство (CPU, GPU, FPGA). После проведения вычислений в этих точках, соответствующие им значения функций через промежуточный буфер передаются на CPU. 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

Вычислительные эксперименты проводились 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 [9].

Известные тестовые задачи из области многомерной глобальной оптимизации характеризуются малым временем вычисления значений целевой функции. 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 задача.

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 ограничим 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 числи 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. Генератор GKLS, описанный в работах [10, 11], позволяет порождать серии задач многоэкстремальной оптимизации с заранее известными свойствами, такими как: размерность задачи, количество локальных минимумов, размеры их областей притяжения, точка 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 ограничим 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, можно избежать портирования и 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.

В рамках данной работы был рассмотрен параллельный алгоритм решения задач многомерной многоэкстремальной оптимизации и его реализация с использованием инструментов Intel oneAPI. Данный алгоритм разработан в рамках информационно-статистического подхода. Вычислительные 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. 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.
5. Захарова, Е. Обзор методов многомерной оптимизации. / Е.М. Захарова, И.К. Минашина, // Информационные процессы, 2014. Т 14. №3. С. 256 – 274.
6. 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.
7. Таненбаум, Э. Архитектура компьютера / Э. Таненбаум, Т. Остин; перевод с английского Е. Матвеева. — 6-е изд. — Санкт-Петербург [и др.]: Питер, 2014. 816 с.
8. 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.
9. 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.
10. Сергеев, Я.Д. Диагональные методы глобальной оптимизации / Я.Д. Сергеев, Д.Е. Квасов – М.: Физматлит, 2008. 352 c.
11. 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.
12. Pinter (Ed.), J.D. Global Optimization: Scientific and Engineering Case Studies / J.D. Pinter – Springer, 2006. 546 p.
13. 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.
14. Barkalov, K.A. A global optimization technique with an adaptive order of checking for constraints / K.A. Barkalov, R.G. Strongin // Computational Mathematics and Mathematical Physics, 2002. – Vol. 42, No. 9. P. 1289-1300.
15. Gergel, V.P. A method of using derivatives in the minimization of multiextremum functions / V.P. Gergel // Computational Mathematics and Mathematical Physics, 1996. Vol. 36, No. 6. P. 729 – 742.
16. Gergel, V.P. A global optimization algorithm for multivariate functions with lipschitzian first derivatives / V.P. Gergel // Journal of Global Optimization, 1997. Vol. 10, No. 3. P. 257-281.

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)