

## СОДЕРЖАНИЕ

Введение .....	7
1 Обзор методов глобальной оптимизации .....	12
1.1 Постановка общей задачи глобальной оптимизации .....	12
1.2 Свойства методов глобальной оптимизации .....	14
1.3 Некоторые методы глобальной оптимизации.....	16
1.4 Оценка сложности методов глобальной оптимизации.....	20
1.5 Разработка параллельных алгоритмов глобальной оптимизации.....	20
2 Визуальное программирование и PGRAFH.....	23
2.1 Способы разработки параллельных программ .....	23
2.1.1 Явный параллелизм и автоматическое распараллеливание .....	23
2.1.2 Графические модели параллельных процессов .....	25
2.2 Концептуальная модель организации параллельных вычислений в технологии ГСП .....	26
2.3 Управление вычислительными процессами. Граф-машина.....	29
2.4 Межмодульный интерфейс параллельного обмена данными.....	31
2.4.1 Стандарт хранения и использования данных в ГСП .....	31
2.4.2 Способ реализации общей памяти в ГСП .....	32
2.4.3 Диспетчер памяти .....	33
2.4.4 Обзор класса TPOData.....	33
2.4.5 Доступ к данным из актора .....	34
2.4.6 Ограничения использования свойств .....	36
2.5 Программное средство моделирования и разработки алгоритмов параллельных вычислений.....	36
2.5.1 Архитектура программного комплекса моделирования и анализа алгоритмов параллельных вычислений .....	36
2.5.2 Программный комплекс моделирования и анализа алгоритмов параллельных вычислений PGRAFH 2.0 .....	39
2.5.3 Генерация исходных текстов параллельных программ на языке C++.....	41
2.5.4 Компилятор данных .....	42
3 Модифицированный метод половинных делений.....	43
3.1 Постановка задачи и основные определения .....	43
3.2 Метод половинных делений .....	44
3.3 Выбор критического параллелепипеда .....	46

3.4	Двухфазный алгоритм метода половинных делений (ДАМПД) .....	48
3.5	Алгоритм построения множества точек начальных приближений для алгоритма локальной оптимизации.....	49
3.6	Параллельная версия двухфазного алгоритма метода половинных делений.....	52
4	Исследование эффективности и работоспособности алгоритма глобальной оптимизации .....	56
4.1	Условия тестирования и экспериментов по оценке эффективности ...	56
4.2	Первая версия алгоритма .....	56
4.3	Вторая версия алгоритма .....	58
4.4	Третья версия алгоритма.....	59
4.5	Рекорд.....	63
4.6	Выбор оптимальных параметров гасителя пульсаций .....	65
4.7	Описание гасителя пульсаций давлений.....	65
4.8	Математическая модель гасителя пульсаций .....	66
4.8.1	Акустическая мощность, излучаемая клапаном и N шайбами ....	67
4.8.2	Мощность шума клапана .....	67
4.8.3	Мощность шума i –й шайбы, i = 2..N-1 .....	68
4.8.4	Мощность шума N –й шайбы ( $x_N = n$ ) .....	69
4.8.5	Вычисление сечений шайб.....	69
4.9	Постановка задачи глобальной оптимизации для гасителя пульсаций давлений.....	71
4.10	Результаты вычислительных экспериментов.....	72
	Заключение .....	78
	Список использованных источников .....	80

## **ВВЕДЕНИЕ**

### **Актуальность темы выпускной работы**

Оптимизация в широком смысле слова находит применение в науке, технике и в любой другой области человеческой деятельности. Оптимизация - целенаправленная деятельность, заключающаяся в получении наилучших результатов при соответствующих условиях. Поиски оптимальных решений привели к созданию специальных математических методов и уже в 18 веке были заложены математические основы оптимизации (вариационное исчисление, численные методы и др). Однако до второй половины 20 века методы оптимизации во многих областях науки и техники применялись очень редко, поскольку практическое использование математических методов оптимизации требовало огромной вычислительной работы, которую без ЭВМ реализовать было крайне трудно, а в ряде случаев – невозможно. Трудности численного решения оптимизационных задач во многом связаны с видом оптимизируемой целевой функции и количеством ее аргументов. Целевая функция может быть невыпуклой, недифференцируемой, негладкой, многоэкстремальной. Кроме того, каждое вычисление значений целевой функции может требовать значительных вычислительных ресурсов.

В настоящее время глобальная оптимизация (ГО) широко востребованное и интенсивно развивающееся направление вычислительной математики. В различных областях науки выдвигается всё больше задач, сводящихся к поиску именно глобального оптимума. Это закономерно привело к росту интереса к проблемам ГО и выделению ей в отдельную ветвь математического программирования.

Подходы ГО существенно отличаются от техники стандартных методов поиска локальных оптимумов функции (часто неспособных найти глобальное решение рассматриваемых многоэкстремальных задач) и характеризуются высокой вычислительной трудоемкостью. Проблематика моделей, методов и программных средств решения задач оптимизации является областью активных научных исследований, в которой результаты советских и российских ученых

имеют широкое признание в стране и за рубежом. Известны работы В.П. Гергеля, Ю.Г. Евтушенко, А.Г. Жилинскаса, С.А. Пиявского, Я. Д. Сергеева, Р.Г. Стронгина, А. А. Жиглявского и др. Среди зарубежных ученых можно указать Р. Брента, П. Пардалоса, Я. Пинтера, Х. Туя, П. Хансена, Р. Хорста и др. При этом техники решения задач одномерной ГО исследованы достаточно глубоко, в то время как построение эффективных алгоритмов многомерной оптимизации, имеющих большое практическое значение, продолжает привлекать большое внимание исследователей.

Важнейшим полученным результатом в теории многоэкстремальной оптимизации является обоснование того факта, что в общем случае, поиск глобального экстремума оптимизируемой функции сводится к построению некоторого покрытия (сетки) в области глобального поиска. При этом данные покрытия должны быть существенно неравномерными для обеспечения эффективности вычислений – эти сетки должны быть достаточно плотными в окрестности глобального оптимума и более разреженными вдали от искомого решения. Построение таких оптимальных покрытий обеспечивается при повышении сложности самих численных методов глобального поиска.

Возможность построения адаптивных схем поиска наилучшего, то есть глобального, решения многоэкстремальных многомерных задач, отличных от переборных схем, предполагает наличие неких априорных предположений о свойствах задачи. Такие предположения служат математическим инструментом для получения оценок глобального решения задачи на основе проведенных испытаний целевой функции и играют существенную роль при построении эффективных алгоритмов глобального поиска. Для многих практических задач (таких как, например, решение нелинейных уравнений и неравенств; регулирование сложных нелинейных систем; оптимизация иерархических моделей, связанных с задачами размещения, системами обслуживания и т.п.) типичным является предположение о липшицевости функций, поскольку относительные вариации функций, характеризующих моделируемую систему, обычно не могут превышать некоторый порог, определяемый ограниченной

энергией изменений в системе. Разработкой теории и методов численного решения задач подобного типа занимается липшицева глобальная оптимизация (ЛГО). Важность данной подобласти ГО объясняется как наличием большого числа прикладных задач, моделируемых при помощи липшицевых функций, так и обширностью класса таких функций.

Во многих задачах, возникающих в практике оптимизации, требуется не просто приближённое численное решение, но ещё и гарантия его близости к идеальному математическому оптимуму, а также часто гарантия того, что найденный оптимум действительно является глобальным. Подобные постановки задач обычно характеризуют термином доказательная ГО, и они являются чрезвычайно трудными.

Существенное продвижение в решении задач ЛГО достигнуто благодаря использованию алгоритмов, сочетающих методы ГО и локальные методы оптимизации. По сравнению с глобальными методами оптимизации, локальные имеют меньшую вычислительную сложность и сходятся быстрее. Данный подход позволяет успешно решать задачи с обязательным нахождением глобального оптимума.

Развитие методов ГО стимулируется не только актуальностью или сложностью этих задач, но и развитием электронно-вычислительных средств. В настоящее время параллельные и векторные суперкомпьютеры рассматриваются как один из основных инструментов для проведения исследований в различных научных и прикладных дисциплинах. В данном случае большое значение приобретают средства автоматизации разработки моделей и кодов параллельных алгоритмов, тестирования и сопровождения программных приложений (в частности алгоритмов оптимизации). Но поскольку технология программирования (особенно в параллельном исполнении) должна быть понятна конечному пользователю, специалисту в предметной области, то на первое место выступают визуальные средства автоматизации программирования параллельных приложений.

Несмотря на явный прогресс в этой области за последние два десятилетия и существенный рост возможностей вычислительной техники, существующие алгоритмы ГО недостаточно вычислительно эффективны, что не позволяет решать множество актуальных задач.

Учитывая практическую важность задач ЛГО, в том числе доказательной, и существующие сложности на пути их решения, представляются актуальными исследования по разработке эффективных алгоритмов решения подобных задач, чему и посвящена данная выпускная работа.

**Целью исследования** является разработка параллельной версии алгоритма ГО функций многих переменных модифицированным методом половинных делений (ММПД), основанном на использовании информационно-статистической стратегии оптимизации и рациональной организации вычислений за счет распределения вычислительной нагрузки между фазами глобальной и локальной оптимизации. А также средства автоматизации построения моделей параллельных алгоритмов.

### **Задачи исследования**

1. Аналитический обзор состояния дел в области параллельных алгоритмов ГО и методов автоматизации построения моделей параллельных алгоритмов.
2. Разработка параллельного алгоритма глобальной оптимизации модифицированным методом половинных делений.
3. Разработка графического редактора, позволяющего создавать модели параллельных алгоритмов.
4. Исследование эффективности алгоритма глобальной оптимизации модифицированным методом половинных делений.
5. Выбор оптимальных параметров гасителя пульсаций давлений.

**Объектом исследования** является алгоритм ГО на основе алгоритма половинного деления, который может быть использованы для успешного решения сложных многомерных и многоэкстремальных задач в физике, химии, биологии, приборостроении и пр.

## **Научная новизна исследования**

1. Предложена новая двухфазная схема организации параллельных вычислений для задачи ГО функций многих переменных, позволяющая организовать рациональное распределение вычислительной нагрузки между фазами глобальной и локальной оптимизации.
2. Разработаны 3 модификации метода половинных делений.
3. В рамках технологии графосимволического программирования разработан графический редактор, позволяющий создавать модели параллельных алгоритмов и строить по ним программы для MPI.
4. Экспериментально доказана эффективность предложенного алгоритма по сравнению с другими существующими алгоритмами ГО.
5. Реализован поиск рациональных параметров гасителя пульсаций давлений.

**Практическая значимость.** Предложенный усовершенствованный параллельный алгоритм ГО ММПД использован при решении актуальной практической задачи – выбора оптимальных параметров гасителя пульсаций давлений. Данная задача решалась в СГАУ на 2 факультете. Была показана применимость реализованного алгоритма. На основе предложенных алгоритмов разработаны современные программные системы, которые позволяют успешно решать задачи оптимизации.

Предложенный алгоритм и средство разработки параллельных алгоритмов PGRAPH 2.0 используются в учебном процессе при проведении лабораторных работ в СГАУ по курсу «Методы и средства визуального программирования», «Автоматизация программирования». Была сформирована документация, содержащая подробное описание алгоритма с описанием всей необходимой информацией для проведения сравнительных исследований алгоритмов ГО по эффективности.

# 1 Обзор методов глобальной оптимизации

## 1.1 Постановка общей задачи глобальной оптимизации

Без ограничения общности, рассмотрим задачу глобальной минимизации (ГО):

$$F(x) \rightarrow \min_{x \in X},$$

где многоэкстремальная функция цели  $F(\cdot)$  является невыпуклой и для различных алгоритмов имеет различные свойства гладкости,  $X \in \mathbf{R}_k$  - допустимое множество задачи,  $\mathbf{R}_k$  -  $k$ -мерное Евклидово пространство.

Множества

$$\begin{aligned} X^{opt} &= \operatorname{Arg} \min_{x \in X} F(x), \\ X^{lopt} &= \{x \in X | \exists \varepsilon > 0 : F(x) \leq F(x'), \forall x' \in B_\varepsilon(x) \cap X\} \end{aligned}$$

множество оптимальных решений и множество локально-оптимальных решений соответственно и  $B_\varepsilon(x) = \{x' \in \mathbf{R}_k | |x' - x| < \varepsilon\}$ . Таким образом, задача ГО состоит в отыскании точки  $x \in X^{opt}$ , причем в случае, когда  $X^{lopt} \neq X^{opt}$ .

В силу широты класса многоэкстремальных функций задача ГО в общем случае является неразрешимой, т.е. нельзя гарантировать, что решение задачи будет получено за конечное число шагов. В работе [1] была получена нижняя оценка стохастической сложности для многоэкстремальной задачи оптимизации  $k$  раз непрерывно дифференцируемой функции  $n$  переменных

$$\tilde{N}(\varepsilon) \geq c(n, k) \left( \frac{1}{\varepsilon} \right)^{n/k}, \quad (1)$$

где  $\varepsilon$  – заданная погрешность решения оптимизационной задачи по координате,  $c(n, k)$  – некоторый коэффициент, зависящий от свойств функции.

В работе [2] для задачи безусловной ГО гладкой липшицевой функции была получена более оптимистическая оценка сложности:

$$N(n) = 2 \cdot \left( \frac{1}{4\varepsilon} \right)^n - 1. \quad (2)$$

Тем не менее, для решения задачи ГО функции 100 переменных для гладких липшицевых функций потребуется приблизительно  $N(100) \approx 7 \cdot 10^{150}$  обращений к целевой функции. Из выражения (1) и (2) ясно, что при заданной точности увеличение размерности задачи оптимизации приводит к катастрофическому росту ее сложности. Даже некоторые из разрешимых задач могут попасть в класс неразрешимых, если число шагов, необходимых для получения решения, окажется чрезмерно большим. Для того чтобы решить задачу, помимо непрерывности функции цели, необходимы некоторые дополнительные условия ее гладкости. Таким образом, специфика задачи ГО заключается в многоэкстремальности функции цели и неразрешимости в общем случае.

В настоящей работе основное внимание уделяется *безусловным задачам глобальной оптимизации*, т.е. задачам, в которых глобальное решение достигается во внутренней точке допустимого множества [3, 4, 5]. В связи с этим предполагается простая структура допустимого множества, обычно это многомерный гиперкуб, без ограничения общности, единичный  $X = \bigotimes_{n=1}^k [0,1]$ .

Для получения точного решения безусловных задач ГО было предложено множество методов. Точные методы либо полагаются на априорную информацию о том, насколько быстро изменяется функция (например информация о константе Липшица функции), либо требуют аналитически сформулировать функцию цели (например, в методе интервалов). Статистические методы, как правило, используют технику разбиения, чтобы разделить область поиска, но такие методы также пользуются априорной информацией или некоторыми предположениями о том, как функция цели может быть промоделирована. Основное предположение заключается в том, что каждая конкретная целевая функция принадлежит классу функций, которые промоделированы конкретной стохастической функцией. Информация из предыдущей выборки целевой функции может быть использована для оценки параметров стохастической функции, и эта усовершенствованная модель

впоследствии может быть использована для смещения при отборе точек в исследуемой области.

Одновременно с возникающими задачами и потребностями в их решении ведутся работы по разработке новых и усовершенствованию существующих специальных инструментов, позволяющих решать эти проблемы, используя весь накопленный опыт и современные вычислительные средства. Проблемам ГО, разработке новых методов и другим важным исследованиям в ГО посвящено немало работ. Интернет-сайт по проблемам ГО – <http://www.mat.univie.ac.at/~neum/glopt.html>.

## **1.2 Свойства методов глобальной оптимизации**

Процедуру (или алгоритм) решения задачи оптимизации можно представить в виде итеративного процесса, который порождает последовательность точек в соответствии с предписанным набором правил, включающим критерий окончания счета. Обычно, глобальное решение задачи оптимизации предлагается найти, перебрав все ее локальные решения. Такая задача, как правило, оказывается трудоемкой. Другой подход - перебрать часть локальных решений и показать, что оставшиеся локальные минимумы не влияют на точность решения. Таким образом, идея всех методов ГО - оценить значения целевой функции  $F(\cdot)$  на некотором множестве точек  $x_1, \dots, x_N$  из допустимого множества  $X$ , и различие методов заключается в способах выбора этих точек.

Поскольку мы не знаем, где именно в  $X$  можно найти глобальные минимумы, то необходимо применить некоторую стратегию, чтобы «разбросать» эти точки по множеству  $X$ . Любую такую стратегию называют *глобальной техникой*. Далее, вполне возможно, что в окрестности выбранной точки  $x_n$  существует лучшее значение функции цели  $F(\cdot)$ , чем  $F(x_n)$ . Для получения лучшего значения функции  $F(\cdot)$  к полученной точке  $x_n$  применяют алгоритм локального спуска. Метод, осуществляющий такой локальный спуск, называют *локальной техникой*. Почти все методы ГО используют методы

локальной оптимизации, по крайней мере, для того чтобы улучшить уже найденную оценку глобального решения, поэтому независимо от применяемой глобальной техники, использование локальной техники является важной частью любого метода ГО.

Другим важным свойством алгоритма оптимизации является сходимость генерируемой им последовательности точек к глобальному оптимальному решению. Однако в большинстве случаев получаются менее благоприятные результаты: невыпуклость функций, большая размерность задачи или другие трудности вынуждают останавливать алгоритм, если получена точка, принадлежащая некоторому множеству приближенных решений. Другими словами, для любого численного алгоритма необходимы условия остановки. Это ключевой момент, т.к. без какой-либо дополнительной информации или предположений о задаче невозможно сделать выводы о точности решения, полученного за некоторое фиксированное число шагов алгоритма. Таким образом, для четкого определения условий остановки алгоритма необходима дополнительная информация или предположения, или же условия остановки должны быть эвристическими. Это подтверждает высказанный ранее факт, что задача ГО в общем случае неразрешима, и мы должны быть готовы принять полученное с помощью численного метода приближение за решение задачи. Таким образом, получить численное решение задачи ГО означает получить некую точку

$$x \in X^{opt}(\varepsilon) = \{x \in X | F(x) \leq F_{opt} + \varepsilon\},$$

где

$$F_{opt} = \min_{x \in X} F(x),$$

а величина  $\varepsilon$  неизвестна.

Любой численный метод имеет свои преимущества и недостатки, в связи, с чем возникает вопрос о сравнении различных методов. Существует ряд факторов, которые следует учитывать при оценке эффективности алгоритмов и их сравнении. Так, универсальность алгоритма определяется тем классом задач,

для решения которых он предназначен, а также рамками требований, предъявляемых алгоритмом к задачам данного класса. Другими важными характеристиками алгоритма является его надежность (или устойчивость), точность, чувствительность к параметрам и исходным данным, затраты на предварительную обработку и вычисления.

Кроме того, с появлением параллельных вычислительных систем возникает другой интересный вопрос: как сравнивать параллельный и последовательный алгоритмы. Не вызывает сомнения факт, что сравнивать параллельные алгоритмы необходимо по-своему. Для оценки эффективности параллельных алгоритмов используют различные подходы, наиболее распространенным из которых является показатель *ускорения*. Ускорение, получаемое при работе алгоритма на  $p$  процессорах - это отношение времени работы алгоритма на одном процессоре ко времени работы того же алгоритма на  $p$  процессорах. Закон Амдала [6] позволяет вычислить верхнюю границу ускорения, которое можно ожидать от параллельной реализации алгоритма.

### **1.3 Некоторые методы глобальной оптимизации**

К сожалению, для решения задачи ГО не существует универсального по эффективности алгоритма. Поэтому при разработке специфических методов ГО в первую очередь учитывают свойства целевой функции  $F(\cdot)$  и допустимого множества  $X$  рассматриваемого класса задач, для которых разрабатывается метод.

Все известные методы ГО можно разделить на две категории: *детерминированные* [7] и *стохастические* [8]. Детерминированные методы получают глобальное решение посредством исчерпывающего поиска на всем допустимом множестве. Поэтому большинство детерминированных методов теряют эффективность и надежность с возрастанием размерности задачи. Более того, чтобы гарантировать успех, такие методы требуют выполнения дополнительных предположений, наложенных на функцию цели. Детерминированные методы не используют стохастики.

Стохастические алгоритмы позволяют уйти от проблем детерминированных алгоритмов. Здесь стохастический подход присутствует не только в разработке и анализе алгоритма, но и используется в решении базовых проблем, например, при определении условия остановки. Большинство стохастических методов оценивают значение функции цели в случайных точках допустимого множества с последующей обработкой выборки. Как следствие, стохастические методы не гарантируют успех.

Ниже приведены некоторые примеры современных методов ГО.

Очень широкий обзор современных достижений в области ГО приведен в статье [2]. Описание методов оптимизации также можно найти в [5, 7, 9, 10, 11, 12, 13, 14, 15, 16] [17, 18, 19].

Исторически первым методом ГО является *метод Монте-Карло*, на базе которого был создан *метод мультистарта* [17]. В методе мультистарта из множества  $X$  случайно или детерминировано выбирается некоторое подмножество из  $N$  точек. Последовательно из каждой точки запускается алгоритм локального спуска, и из полученного множества локальных решений выбирается наилучшее. В чистом виде метод мультистарта не является эффективным, т.к. одно и то же локальное решение может быть найдено не один раз. Мультистарт - это обобщенный подход: большинство эффективных методов ГО основано на идее метода мультистарта - запуска стандартных локальных алгоритмов из множества точек, равномерно распределенных на множестве  $X$ . Таким образом, метод мультистарта можно назвать прототипом таких методов.

*Методы группировок* [18] являются одной из модификаций метода мультистарта. Здесь предпринята попытка устранения главного недостатка мультистарта путем тщательного отбора точек, из которых запускается локальный поиск. Рассматривается некоторая выборка точек, например, равномерно распределенных на  $X$ . Затем, пока не выполнится некоторое условие остановки, последовательно выполняются следующие три шага:

1. из каждой точки запускается алгоритм локального спуска, в результате будет получен набор локальных решений;
2. используя специальную технику группировки, определяются группы точек;
3. в качестве новой выборки точек рассматривается каждая  $m$ -тая точка из группы, и осуществляется переход к первому шагу.

Таким образом, решения находятся посредством локальных алгоритмов спуска из наилучших точек каждой группы.

*Методы деления пополам и методы интервалов* [4, 9, 19, 20] гарантируют, что решение будет получено с заданной точностью. Эти методы в ГО также носят название *методов покрытий* или *методов ветвей и границ*. Цена предлагаемой методами гарантии – некоторая *a priori* известная информация о функции цели. Так, для метода деления пополам необходимо знание константы Липшица функции цели, а для метода интервалов функция цели обязана быть дважды непрерывно дифференцируемой, и первая и вторая производные должны иметь конечное число нулей. Большинство методов интервалов используют *стратегию ветвей и границ* [20]. Такие алгоритмы разделяют область поиска на набор многомерных кубиков, на которых нижняя граница функции цели вычисляется с помощью интервальной техники. Используя интервальную арифметику на каждом шаге, получаем набор последовательно уменьшающихся интервалов, который содержит глобальное решение исходной задачи. Алгоритм останавливается, когда размер интервалов достигает заранее заданного значения. Методы интервалов требуют, чтобы функция цели была задана явно, т. к. это выражение используется интервальной техникой для вычисления границ. Методы, использующие технику ветвей и границ, в силу древовидной структуры являются объектом исследований в контексте параллельных вычислений. Некоторые разработки параллельных алгоритмов ветвей и границ приводятся в [20].

*Эволюционные алгоритмы* [21] являются поисковыми методами, основная идея которых заимствована из биологического процесса

естественного отбора и процесса выживания. Такие алгоритмы отличаются от традиционных методов оптимизации тем, что поиск производится из «популяции» решений, а не из одной точки. Каждая итерация метода производит «естественный отбор», который отсеивает неподходящие решения. Решения с высокой пригодностью («биологической реакцией на естественный отбор») «скрещиваются» с другими решениями путем обмена частей одних решений на другие. Решения могут «муттировать» из-за небольших замен одного элемента решения. Скрещивания и мутации генерируют новые решения, которые «генетически» настроены на области допустимого множества, для которых уже было обнаружено хорошее решение. Существует несколько различных типов эволюционных поисковых алгоритмов: *алгоритмы генетического программирования; алгоритмы эволюционного программирования; алгоритмы эволюционных стратегий; генетические алгоритмы*. Эволюционные алгоритмы обладают слабой сходимостью к глобальному решению, но вместе с тем, хорошо обрабатывают сильно зашумленные функции с большим числом незначительных локальных решений, не «прилипают» к локальным экстремумам и способны получить глобальное решение.

Методы *редукции размерности* позволяют свести решение многомерных оптимационных задач к семейству задач одномерной оптимизации. Один из наиболее общих методов редукции размерности состоит в применении *многошаговой схемы редукции размерности* [22], согласно которой решение многомерной задачи оптимизации может быть получено посредством решения последовательности «вложенных» одномерных задач. Другим общим способом редукции размерности основывается на известном фундаментальном свойстве, согласно которому  $N$ -мерный гиперпараллелепипед и отрезок вещественной оси  $[0,1]$  являются равнomoщными множествами и отрезок может быть однозначно и непрерывно отображен на гиперпараллелепипед. Отображения такого рода обычно называют развертками или кривыми Пеано. В большинстве случаев в методах редукции ставка делается на эффективность одномерных

алгоритмов ГО, например, на информационно-статистический подход Р.Г. Стронгина [23].

#### **1.4 Оценка сложности методов глобальной оптимизации**

Согласно работе [2] в таблицу 1 сведены оценки сложности задачи ГО функции многих переменных для разных методов ее решения и свойств самой функции.

Таблица 1 – Оценка вычислительной сложности задачи глобальной оптимизации

$n$	2	3	4	5	6
$N(n)$ для непрерывной функции	2047	65535	2097151	67108863	2,15E+09
$N(n)$ липшицевой функции, $\alpha=0.5$	127	3412	102984	3215632	1,02E+08
$N(n)$ метод редукции, $\alpha=0.75$	862	14371	273379	5499292	1,14E+08
$N(n)$ метод редукции, $\alpha=0.6$	80	388	2163	12818	78622

Из таблицы видно, что даже при очень благоприятных условиях, связанных либо со свойствами функции, либо эффективными алгоритмическими решениями, когда  $\alpha$  равно 0,5, сложность задачи ГО остается экспоненциальной. Этот факт позволяет разрабатывать приемлемые по сложности алгоритмы распараллеливания задачи ГО для сравнительно небольших размерностей. Метод редукции также принципиально не изменяет экспоненциальный характер роста сложности задачи ГО. Однако, при даже незначительном уменьшении коэффициента «прореживания»  $\alpha$  существенно расширяется диапазон размерности задачи ГО, когда за счет распараллеливания можно добиться ощутимых конечных результатов.

#### **1.5 Разработка параллельных алгоритмов глобальной оптимизации**

Как мы отмечали ранее, в общем случае, методы оптимизации относятся к классу нерешаемых задач. В этом смысле, уже невозможно надеяться на создание универсального алгоритма оптимизации в равной степени эффективно решающего любую задачу. Выход из сложившейся ситуации можно найти на пути разработки частных алгоритмов оптимизации, учитывающих особенности

математических моделей объектов исследования и специфику постановки задачи, а также применение технологий параллельных вычислений. Но для полномасштабного применения этого подхода необходимы специальные программные средства, позволяющие оперативно разрабатывать эффективные частные алгоритмы оптимизации. Более того, частные алгоритмы оптимизации должны опираться на современную парадигму параллельных алгоритмов.

В данном случае большое значение приобретают средства автоматизации разработки моделей и кодов параллельных алгоритмов, тестирования и сопровождения программных приложений (в частности алгоритмов оптимизации). Но поскольку технология программирования (особенно в параллельном исполнении) должна быть понятна конечному пользователю, специалисту в предметной области, то на первое место выступают визуальные средства автоматизации программирования параллельных приложений.

Методология разработки параллельных алгоритмов в корне отличается от традиционных методов создания последовательных кодов программ. В настоящее время более распространёнными и доступными являются вычислительных систем с распределенной памятью, в которых для организации информационного взаимодействия между процессорами в большинстве случаев используется интерфейс передачи данных MPI. Необходимость профессионального владения технологиями программирования на языке высокого уровня (C++, C#) и MPI еще дальше отдаляет «конечных» пользователей – специалистов в предметных областях, основных потребителей суперкомпьютерных технологий, от возможности участия в разработке параллельных алгоритмов. Одним из выходов из сложившейся ситуации является применение средств автоматизации проектирования и разработки параллельных программ с использованием выразительных визуальных, графических моделей описания алгоритмов.

Для неспециалистов MPI не знакомых с тонкостями устройством кластера, в инструменте разработки параллельных программ в первую очередь необходимо иметь механизм автоматизированного управления памятью и

синхронизацией. С другой стороны применение графической технологии программирования сокращает время на модификацию не только параллельных алгоритмом, но и последовательных разработок. Дополнительные модули тестирования и проверки алгоритмов необходимы для предотвращения типичных ошибок параллельного программирования, предварительной оценки эффективности разрабатываемого алгоритма и оптимизации под конкретную аппаратную платформу вычислительной системы.

Средствам и способам разработки параллельных алгоритмов посвящена следующая глава.

## **2 Визуальное программирование и PGRAFH**

### **2.1 Способы разработки параллельных программ**

Существует множество графических моделей описания алгоритмов параллельных вычислений. В области описания вычислительных алгоритмов наиболее удобной представляется форма, близкая описанию блок-схемам, которая реализована в параллельной версии технологии ГСП [24].

Технология графосимволического программирования GRAFH, созданная на кафедре программных систем СГАУ, является одним из способов наглядного представления алгоритмов программы в виде графа управления. При разработке и реализации двухфазного алгоритма глобальной оптимизации модифицированным методом половинных делений использовалась средство моделирования параллельных вычислений PGRAFH 2.0.

#### **2.1.1 Явный параллелизм и автоматическое распараллеливание**

Работы в области моделирования и построения параллельных алгоритмов можно разделить на два больших направления.

- **Неявный параллелизм.** Это направление изучает методы автоматической генерации параллельных алгоритмов на основе их последовательных прототипов (автоматического распараллеливания последовательных алгоритмов).
- **Явный параллелизм.** Разработка методов организации вычислений, изначально ориентированных для реализации на ЭВМ с параллельной архитектурой.

Исследования в области автоматического распараллеливания вычислительных алгоритмов необходимы в связи с наличием большого объема ранее разработанных методов, алгоритмов и программ для решения различных задач на последовательных ЭВМ. Их реализация на параллельных ЭВМ требует модификации, связанной с распределением данных и вычислений по узлам параллельной ЭВМ, а также с адаптацией под особенности архитектуры конкретной ЭВМ. Этот процесс важно автоматизировать, чтобы максимально сократить его длительность и избавить исследователей - специалистов в

различных областях, которые часто являются авторами и пользователями вычислительных программ, от знакомства со спецификой конкретной ЭВМ, на которой программа будет исполняться.

Автоматическое распараллеливание имеет большое значение и при создании новых вычислительных программ. Последовательные алгоритмы удобны и естественны для человека в силу того, что люди привыкли думать и действовать последовательно. Вместе с тем, любая современная ЭВМ обладает определенной степенью параллелизма.

Наряду с несомненными достоинствами, такими как возможность использования ранее разработанных и хорошо отлаженных последовательных программ, сохранение привычного для человека последовательного стиля разработки вычислительных алгоритмов, обеспечение переносимости программ, автоматическое распараллеливание обладает недостатками. Основным недостатком является ограниченная область применения. К сожалению, не все последовательные алгоритмы допускают эффективное распараллеливание. Иногда сам численный метод, на основе которого построен алгоритм, не допускает распараллеливания.

Для достижения максимальной производительности необходимо уже на этапе разработки алгоритма учитывать параллелизм и явно выделять участки, которые должны выполняться одновременно. Более того, необходимо учитывать архитектуру и особенности конкретной параллельной ЭВМ, на которой будет исполняться программа.

Основные сложности, с которыми сталкиваются исследователи в области построения параллельных алгоритмов с явным параллелизмом, в первую очередь связаны с наглядным представлением алгоритма.

Текстовая нотация, традиционно используемая в математике и программировании, удобна для представления последовательных процессов. Однако последовательная природа самого текста значительно затрудняет восприятие текстового описания параллельных вычислений. На первый план выдвигаются графические способы описания параллелизма.

### 2.1.2 Графические модели параллельных процессов

Основой подавляющего большинства графических способов представления параллельных процессов является форма представления в виде графа, то есть совокупности вершин (узлов), соединенных между собой дугами (ребрами). В отличие от текстовой формы записи, в которой объекты (символы и слова) образуют *последовательность*, а каждый объект связан только с левым и правым «соседом», графовая форма позволяет наглядно изображать более сложные *взаимосвязи*, поскольку в ней каждый объект может соединяться с несколькими другими объектами. В этом смысле текстовая форма одномерна, в то время как графовая форма – многомерна. Возможность варьировать геометрические размеры, форму и цвет вершин, внешний вид и толщину дуг, изменять взаимное расположение вершин без изменения топологии графа значительно увеличивают выразительные возможности графовой формы представления.

Графические модели обычно являются ориентированными графами, в которых дуги определяют направление передачи данных или зависимость между вершинами. Вершины и дуги обычно снабжаются текстовыми аннотациями, которые именуют их, перечисляют их содержимое или свойства. Различные графические модели отличаются друг от друга семантикой вершин и дуг. Основные классы графических моделей параллельных процессов:

- *сети Петри* [25];
- *графы переходов*, используются в SWITCH-технологии [26] и графическом языке Statecharts [27];
- *диаграммы потоков данных*, используются в системах CODE 2.0 [28], Paralex [29];
- *диаграммы потоков управления*, примерами моделей этого типа являются графический язык системы HeNCE [30], технология графо-символического программирования.

Графическая нотация является более наглядной и компактной, по сравнению с текстовым описанием. За счет использования графических

моделей удается не только сократить время разработки параллельных алгоритмов, но и повысить их качество, т.к. графическая нотация допускает формальное математическое описание модели, по которому может быть проведена ее автоматическая верификация и оптимизация.

## **2.2 Концептуальная модель организации параллельных вычислений в технологии ГСП**

Технология ГСП в работе [24] определена как технология проектирования и кодирования алгоритмов программного обеспечения (ПО), базирующаяся на графическом способе представления программ, преследующую цель полной или частичной автоматизации процессов проектирования, кодирования и тестирования ПО.

Модель представляется четверкой  $\langle D, F, P, G \rangle$ , где  $D$  – множество данных некоторой предметной области,  $F$  – множество операторов, определенных над данными предметной области,  $P$  – множество предикатов, действующих над структурами данных предметной области,  $G = \{A, \Psi, \Phi, R\}$  – ориентированный помеченный граф.  $A = \{A_1, A_2, \dots, A_n\}$  – множество вершин графа. Каждая вершина  $A_i$  помечена локальным оператором  $f_i(D) \in F$ . На графике задано множество дуг управления  $\Psi = \{\Psi_{1i1}, \Psi_{1i2}, \dots, \Psi_{jm}\}$  и множество дуг синхронизации  $\Phi = \{\Phi_{1i1}, \Phi_{1i2}, \dots, \Phi_{jl}\}$ .  $R$  – отношение над множествами вершин и дуг, определяющее способ их связи. Дуга управления, соединяющая любые две вершины  $A_i$  и  $A_j$ , имеет три метки: предикат  $p_{ij}(D) \in P$ , приоритет  $k(\Psi_{ij}) \in N$  и тип дуги  $T(\Psi_{ij}) \in N$ . Каждая дуга синхронизации  $\Phi_{ij}$  помечена сообщением  $\mu_{ij} \in N$ . В классической (последовательной) модели вычислительного процесса, используемой в технологии ГСП, отсутствуют дуги синхронизации, которые вводятся в модель параллельного вычислительного процесса для решения задач синхронизации между его различными участками.

Развитие вычислительного процесса, описываемого моделью, ассоциируется с переходами из вершины в вершину по дугам управления. При этом переход по дуге управления возможен лишь в случае истинности

предиката, которым она помечена. Если несколько предикатов, помечающих исходящие из вершины дуги, одновременно становятся истинными, переход осуществляется по наиболее приоритетной дуге. Функционирование модели начинается с выполнения оператора  $f_0(D)$ , помечающего начальную вершину  $A_0$ .

Тип дуги  $\Psi_{ij}$  определяется как функция  $T(\Psi_{ij}) \in \{1,2,3\}$ , значения которой имеют следующую семантику:

$T(\Psi_{ij})=1$  – *последовательная дуга* (описывает передачу управления на последовательных участках вычислительного процесса);

$T(\Psi_{ij})=2$  – *параллельная дуга* (обозначает порождение нового параллельного вычислительного процесса);

$T(\Psi_{ij})=3$  – *терминирующая дуга* (завершает параллельный вычислительный процесс).

Для описания параллелизма вводится понятие *параллельной ветви*  $\beta$  – подграфа графа  $G$ , начинающегося параллельной дугой (тип этой дуги  $T(\Psi_{ij}) = 2$ ) и заканчивающейся терминирующей дугой (тип этой дуги  $T(\Psi_{ij}) = 3$ ).  $\beta = \langle A_\beta, \Psi_\beta, R_\beta \rangle$ , где  $A_\beta$  – множество вершин ветви,  $\Psi_\beta$  – множество дуг управления ветви,  $R_\beta$  – отношение над множествами вершин и дуг ветви, определяющее способ их связи.

Дуги, исходящие из вершин параллельной ветви  $\beta$ , принадлежат также ветви  $\beta$ . При кодировании алгоритма, описанного с помощью предлагаемой модели, каждая параллельная ветвь порождает отдельный *процесс* – совокупность подпрограмм, исполняемых последовательно на одном из процессоров параллельной вычислительной системы.

Графическая модель обычно содержит несколько параллельных ветвей, каждая из которых образует отдельный процесс. В этом смысле модель параллельных вычислений можно представить как объединение нескольких параллельных ветвей:

$$G = \bigcup \beta_k,$$

где  $\beta_k$  – параллельные ветви графа G.

Таким образом, распараллеливание вычислений возможно только на уровне граф-модели. Вычисления в пределах любого актора выполняются последовательно.

Число параллельных ветвей в модели фиксируется при ее построении, при этом максимальное количество ветвей не ограничивается. Каждая ветвь имеет ровно один вход и один выход, для обозначения, которых в граф-модели используются два типа дуг: параллельная дуга и терминирующая дуга. На рисунке 1 показано два типа дуг.

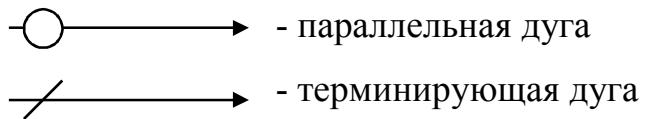


Рисунок 1 – Обозначение параллельных и терминирующих дуг в графической модели параллельных вычислений

Для описания правил построения граф-модели введем следующую систему обозначений:

$To(A_i)$  – список дуг, входящих в вершину  $A_i$ ,

$Fr(A_i)$  – список дуг, исходящих из вершины  $A_i$ ,

$H(\beta_j)$  – начальная вершина ветви  $\beta_j$ ,

$|L|$  – число элементов в списке L.

Переход по параллельной дуге начинает работу параллельной ветви, переход по терминирующей дуге – заканчивает ее работу. Параллельная дуга не содержит предиката, т. е. переход по ней происходит безусловно.

$$\forall \Psi_{ij} \in \Psi, T(\Psi_{ij}) = 2 : P_{ij} \equiv 1.$$

Из вершины может исходить не менее двух параллельных дуг. Если из вершины исходит параллельная дуга, то дуги других типов (обычная или терминирующая) не могут исходить из данной вершины.

$$T(\Psi_{ij0}) = 2 \rightarrow \forall j \neq j0 : T(\Psi_{ij}) = 2, |Fr(A_i)| \geq 2.$$

Входящие в вершину дуги не могут быть одновременно параллельными и терминирующими:

$$T(\Psi_{ij0}) = 2 \rightarrow \forall i : T(\Psi_{ij0}) \neq 3.$$

Функционирование модели начинается с запуска единственной ветви, называемой *мастер-ветвью* (мастер-процессом). Обозначим мастер-ветвь  $\beta_0$ . В вершинах мастер-ветви, имеющих исходящие параллельные дуги, порождаются новые параллельные ветви. Вершины этих ветвей также могут иметь параллельные дуги, таким образом, допускается вложенность параллельных ветвей. Ветви, породившиеся в одной вершине некоторой ветви, должны терминироваться также в одной вершине этой же ветви:

$$\begin{aligned} \forall k : H(\beta_k) = A_{k0}, \Psi_{ik0} \in Fr(A_i), A_i \in \beta_1 \rightarrow \\ \rightarrow \Psi_{kN} \in T_0(A_j), T(\Psi_{kN}) = 3, A_j \in \beta_1. \end{aligned}$$

В ветви  $\beta_1$  управление из вершины  $A_i$  после запуска ветвей  $\beta_k$ ,  $k = k_1, k_2, \dots, K$ , передается вершине  $A_j$ . Вершина  $A_j$  запускается на выполнение после завершения работы ветвей  $\beta_{k1} \dots \beta_K$ . Таким образом, в каждый момент времени в любой ветви выполняется ровно одна вершина.

Переход между двумя вершинами, принадлежащими различным параллельным ветвям, возможен только по параллельным и терминирующими дугам, то есть, запрещены условные переходы между вершинами различных параллельных ветвей:

$$\forall i, j, A_i \in \beta_1, A_j \in \beta_2 \rightarrow T(\Psi_{ij}) \neq 1.$$

Если некоторая ветвь породила новые параллельные ветви, то вычисления в ней приостанавливаются до завершения работы порожденных ветвей. Таким образом, вложенные параллельные ветви выполняются последовательно относительно друг друга.

### **2.3 Управление вычислительными процессами. Граф-машина**

В технологии ГСП для объектов – агрегатов используется мониторная схема организации вычислений. В основу способа положено централизованное

управление процессом вычислений, осуществляемое специальной программой – *граф-машиной*.

Граф-машина универсальна для любого алгоритма. Исходной информацией для граф-машины служит, описанная выше, модель графа управления вычислительным процессом. Анализируя его графическую модель, представленную в виде структур на смежной памяти [31], она выполняет в соответствующем порядке акторы и агрегаты, вычисляет значения предикатов и управляет синхронизацией. Для каждой параллельной ветви запускается по одному экземпляру граф-машины, которая представляет собой отдельный процесс в вычислительной системе.

Работа граф-машины начинается с выполнения актора в корневой вершине. Затем строится список дуг, исходящих из текущей вершины. Этот список просматривается граф-машиной последовательно, начиная с самой приоритетной дуги. Вычисляется значение предиката, помечающего дугу, и в случае его истинности, происходит переход к обработке следующей вершины. В результате обработки параллельной дуги в отдельном процессе запускается другая граф-машина, обрабатывающая порождаемую данной дугой параллельную ветвь. После запуска всех параллельных ветвей происходит переход в вершину, в которой они терминируются. Родительская граф-машина ожидает завершения выполнения всех дочерних граф-машин, если не задано альтернативное условие.

Централизация функций управления в рамках одной программы (граф-машины) на самом деле очень удобное решение, поскольку позволяет:

- контролировать вычислительный процесс в целом. И, в случае непредвиденных ситуаций, принимать системные решения;
- реализовать сбор статистической информации о характеристиках надежности каждого из модулей; вычислительной сложности модулей; маршрутах развития вычислительного процесса и т.п.;
- выполнять пошаговую отладку программы в реальном времени и симулировать вычисления.

## **2.4 Межмодульный интерфейс параллельного обмена данными**

Проблема передачи информации от одной программы к другой традиционно представляет собой сложную проблему, из-за которой возникают серьезные трудности в процессе автоматизации порождения кодов программ. Особенно остро эта проблема стоит в системах с распределенной памятью, таких как суперкомпьютерные кластеры. Для сохранения очевидных преимуществ, возникающих при использовании модели общей памяти, в системе PGRAPH последняя эмулируется за счет использования возможностей объектно-ориентированной парадигмы программирования: понятия класса языка C++.

### **2.4.1 Стандарт хранения и использования данных в ГСП**

В технологии ГСП используется стандарт при организации межмодульного информационного интерфейса. Стандарт обеспечивается выполнением семи основных правил.

1. Вводится единое для всей предметной области программирования (ПОП) хранилище данных, актуальных для всей области. Полное описание данных размещено в словаре данных ПОП. Любые переменные, не описанные в словаре данных, считаются локальными данными для тех объектов ГСП, где они используются.
2. В пределах ГСП описание типов данных размещается централизовано в архиве типов данных.
3. Данные, актуальные для формируемого программного приложения, объединяются в единую универсальную структуру – класс TPOData.
4. В базовых модулях в качестве механизма доступа к данным допускается только передача параметров по адресу, ссылающемуся на универсальную структуру данных.
5. Привязка данных объектов ПОП к формальным параметрам базовых модулей реализована в паспортах объектов ПОП.
6. В технологии ГСП не рекомендуется использовать иные способы организации межпрограммных связей по данным.

7. Данные в ПОП могут быть общими и локальными. Память под общее данное выделяется в менеджере памяти, и все процессоры имеют доступ к этой переменной. Память под локальную переменную выделяется на каждом процессоре, и только этот процессор может читать и изменять её значение.

Данных подход к организации межмодульного информационного интерфейса приводит к тому, что формируемые (автоматически или автоматизировано) программные коды и информационные связи «пространственно» отделены друг от друга. Модификация любого из объектов (актора, предиката или агрегата) не требует переделки кодов других объектов, входящих в ПОП. Физически данные ПОП хранятся в общей области памяти базового компьютера. Параллельный вариант граф модели при использовании технологии MPI предполагает распределенное размещение данных на разных компьютерах некоторого кластера. Для соблюдения вышесказанных условий в параллельной версии технологии ГСП программно реализована модель общей памяти для аппаратной архитектуры с распределенной памятью.

#### **2.4.2 Способ реализации общей памяти в ГСП**

В среде выполнения программы выбирается машина, на которой будет запущен процесс, отвечающий за хранение переменных ПОП. Учитывая аппаратные особенности и топологию ВС, это может быть узел с наибольшим объемом оперативной памятью или центральный узел, имеющий минимальное время доступа от любого из остальных узлов кластера. Преимущество данного подхода в том, что значительно экономится ресурс памяти на вычислительных узлах, т.к. на узлах память выделяется только под те переменные, которые используются.

Описанная идея организации хранения и обмена данными между параллельными процессами ориентирован на модель передачи сообщений, в которой каждый процесс работает с локальными данными. Например, стандарт MPI подразумевает, что процессы обмениваются данными только в результате пересылки их в виде сообщений.

Предлагаемый способ обмена данными требует введения понятия диспетчера данных – подпрограммы, выполняющей функции хранения, чтения и модификации данных предметной области.

#### **2.4.3 Диспетчер памяти**

Диспетчер данных исполняется в отдельном процессе параллельной программы. Он порождает объект, описываемый классом TPOData, который хранит значения данных предметной области. В каждом из процессов, содержащих параллельные ветви граф-модели, также порождается объект класса TPOData. Однако функции доступа к членам-данным у объекта диспетчера данных и у объектов параллельных ветвей различаются. Диспетчер данных хранит все данные в локальной памяти и для обращения к ним использует обычные указатели. На остальных процессах используется ленивая инициализация памяти под переменную при первом доступе.

В параллельных ветвях граф-модели для чтения или записи некоторого данного осуществляется обращение к диспетчеру памяти с помощью совокупности сообщений. В первом сообщении пересыпается запрос на чтение или запись конкретного данного. Каждая переменная из ПОП получает уникальный номер, по которому диспетчер памяти может ее идентифицировать. В случае чтения параллельная ветвь переходит к ожиданию ответа от диспетчера данных. При записи во втором сообщении пересыпается новое значение переменной. Диспетчер данных циклически принимает и обрабатывает запросы параллельных ветвей. Принцип работы менеджера памяти схематично показан на рисунке 2.

#### **2.4.4 Обзор класса TPOData**

Для сохранения очевидных преимуществ, возникающих при использовании модели общей памяти, в системе PGGRAPH последняя эмулируется за счет использования возможностей объектно-ориентированной парадигмы программирования: класса TPOData.

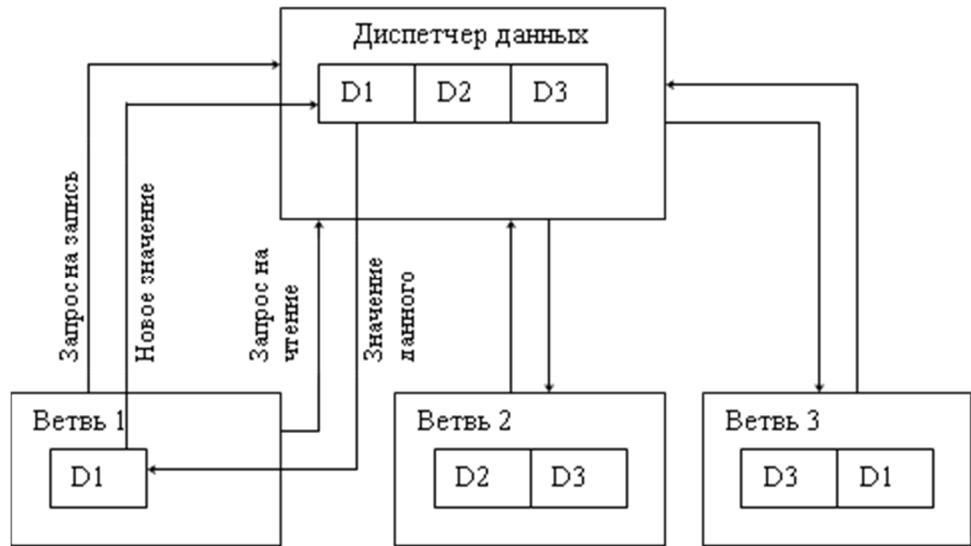


Рисунок 2 – Модель общих данных системы PGGRAPH

Класс TPOData – это ядро механизма хранения и передачи данных в технологии ГСП. Класс TPOData инкапсулирует все данные ПОП и предоставляет доступ к ним через поля-свойства. Свойство – способ доступа к внутреннему состоянию объекта, имитирующий переменную некоторого типа. Обращение к свойству объекта выглядит так же, как и обращение к полю объекта, но, в действительности, реализовано через вызов функции. При попытке задать значение данного свойства вызывается один метод (*setter*), а при попытке получить значение данного свойства – другой (*getter*). Полей ровно столько, сколько переменных в ПОП, и их названия совпадают с названиями переменных. Для переменных простых типов в классе TPOData описано по одному методу доступа для чтения и установки значения. Для массивов определено по два метода чтения и установки: доступ ко всему массиву и к элементу по индексу.

#### 2.4.5 Доступ к данным из актора

Класс TPOData и экземпляр класса напрямую недоступен для разработчика граф-программ и программист не обязан знать о его существовании. Когда программист обращается к переменной ПОП, на самом деле он обращается к полям-свойствам объекта D.

Рассмотрим использование свойств на примере создания inline-актора. Описание данных и переменных ПО приведено в таблице 2 и 3

Таблица 2 – Описание типов данных ПО

Название	Определение	Описание
int		Встроенный тип целых чисел
double		Встроенный тип чисел с двойной точностью
array	typedef int array[100]	Массив целых чисел длины 100

Таблица 3 – Описание данных ПО

Название	Тип
A	int
B	double
C	array

Пример доступа к переменным показан в листинге 1.

Листинг 1

```
a = 100; //устанавливает значение а в 100
double _b = b; //читает значение b в локальную переменную _b
b+=1; //инкрементирует значение b
for (int i = 0; i < c.length; i++)
    c[i] = i; //инициализирует каждый элемент массива
```

С помощью компилятора объектов ГСП данный код перед сохранением, скрытно от пользователя будет транслирован в код, представленный в листинге 2.

Листинг 2

```
D->a = 100; //устанавливает значение а в 100
double _b = D->b; //читает значение b в локальную переменную _b
D->b+=1; //инкрементирует значение b
for (int i = 0; i < D->c.length; i++)
    D->c[i] = i; //инициализирует каждый элемент массива
```

В примере показана дополнительная возможность массивов – это получение их длины, как значения поля `length`.

#### **2.4.6 Ограничения использования свойств**

В целом, с плеч программиста снимается тяжелый груз забот об управлении данными с помощью функций MPI, и обмен данными между процессами становится “похож” на обмен данными между нитями внутри одного процесса. Однако реализованный механизм полей-свойств накладывает ряд ограничений на использование данных.

1. Переменная ПОП не может использоваться в качестве параметра функции с переменным числом аргументов (например `printf(const char *, ...)`) и не может передаваться по указателю в функцию (например `scanf(const char *, void *)`). В подобных случаях необходимо создать локальную буферную переменную, через которую читать и писать в переменную ПОП. Это ограничение касается использования свойств только в `inline`-модулях.

2. Переменные в общей памяти не могут иметь тип “указатель”. Следствием этого является то, что в общей памяти нельзя создавать массивы переменной длины.

3. В качестве переменных предметной области нельзя использовать многомерные массивы, как следствие запрета 2. Необходимо представить многомерный массив в виде одномерного.

### **2.5 Программное средство моделирования и разработки алгоритмов параллельных вычислений**

#### **2.5.1 Архитектура программного комплекса моделирования и анализа алгоритмов параллельных вычислений**

На рисунке 3 приведена структура программного комплекса моделирования и анализа алгоритмов параллельных вычислений.

Ядром системы является информационный фонд, который содержит информацию о переменных предметной области, объектах и созданных пользователем системы граф-моделях. Эта информация поступает в базу данных в результате работы пользователя с редакторами подсистемы

редактирования. Исходный текст базовых модулей и предикатов создается в редакторе текстов, который представляет собой обычный текстовый редактор, дополненный средствами проверки синтаксиса базовых модулей.

Редактор объектов предназначен для создания акторов и предикатов на основе базовых модулей с помощью операции паспортизации. Агрегаты создаются пользователем в редакторе графов – специализированном графическом редакторе, оперирующем визуальными объектами модели, такими как вершины и дуги.

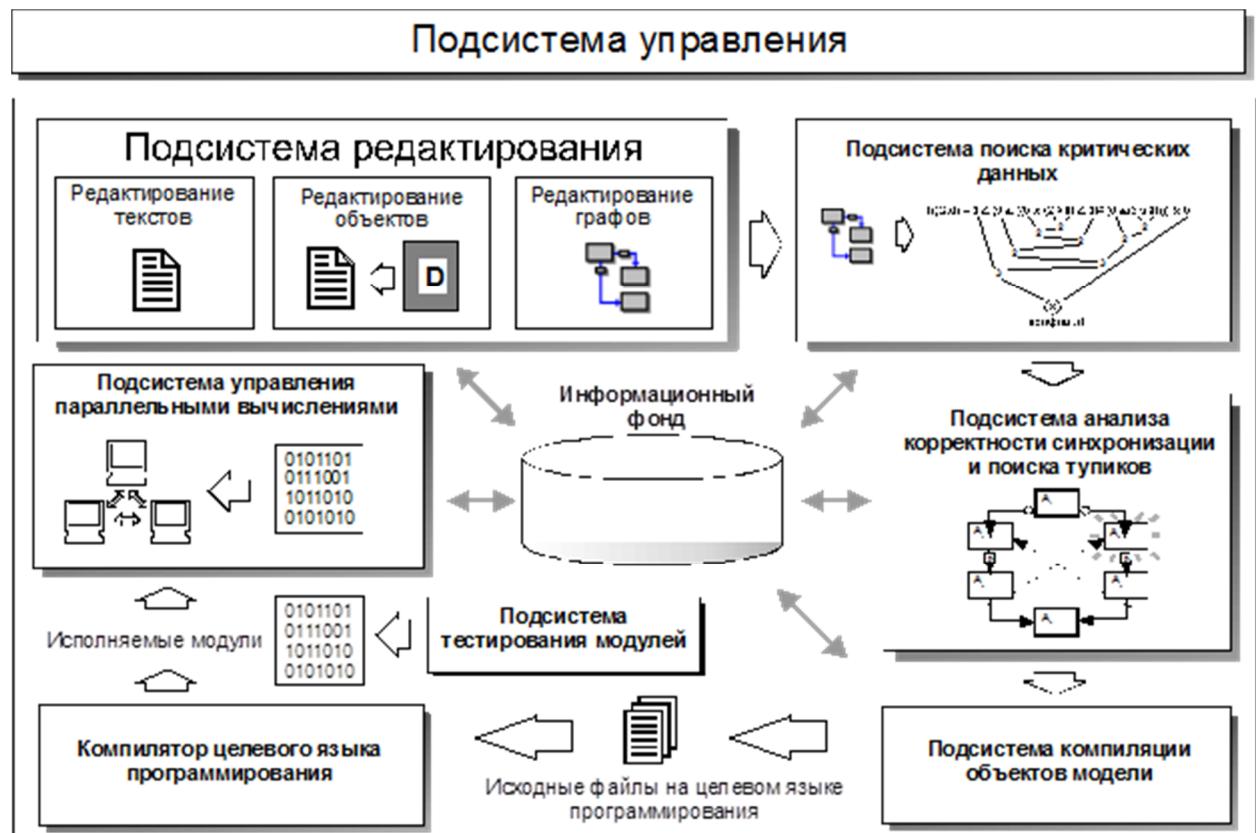


Рисунок 3 – Структура программного комплекса моделирования и анализа алгоритмов параллельных вычислений

После создания графической модели алгоритма, перед запуском процесса вычислений, осуществляется проверка корректности модели.

Информация о разработанных пользователем акторах, предикатах и агрегатах используется подсистемой компиляции объектов модели. На основе данных информационного обеспечения она строит исходные файлы на некотором целевом языке программирования. В настоящее время реализована

поддержка языка С и С++, однако систему можно расширить для работы с другими языками программирования. Таким образом, пользователю предоставляется возможность использования наиболее удобного для данной задачи языка или языка, наиболее известного пользователю, в котором он имеет большую библиотеку написанных и отлаженных БМ.

Для работы с каждым целевым языком программирования создается отдельный компилятор объектов и при желании – отдельный редактор текстов, в качестве которого может применяться, например, специализированный редактор среды программирования на целевом языке.

Для преобразования исходных текстов в исполнимый модуль используется компилятор целевого языка программирования. Как правило, это программный продукт, созданный сторонним разработчиком и входящий в состав среды программирования на целевом языке. Такой подход придает системе гибкость, повышает переносимость и надежность создаваемых программ. Исполняемые модули по желанию пользователя системы могут генерироваться для различных платформ и операционных систем (например, для SMP-компьютеров под управлением операционной системы Windows 2007 или Unix-кластеров).

Подсистема управления параллельными вычислениями работает с исполняемыми модулями и выполняет такие действия, как распределение программ по узлам вычислительной системы, запуск программ на выполнение и сбор статистических данных об их работе. В состав подсистемы управления параллельными вычислениями могут входить, например, средства анализа трассировочной информации о выполнении программы для оптимизации ее производительности. Чаще всего эта подсистема входит в состав операционной системы или является сторонним продуктом, предустановленным на кластере, например, система управления заданиями Torque [32].

## **2.5.2 Программный комплекс моделирования и анализа алгоритмов параллельных вычислений PGRAPH 2.0**

В рамках настоящей работы разработан и реализован программный комплекс PGRAPH 2.0, предназначенный для визуального построения граф-моделей параллельных алгоритмов и автоматической генерации программ на основе этих моделей.

Программный комплекс ориентирован на работу в модели передачи сообщений. Генерируемые программы могут исполняться как на вычислительных системах с общей памятью, так и в распределенных системах. Механизм передачи сообщений между параллельными процессами базируется на технологии MPI (Message Passing Interface), которая представляет стандартизованные средства передачи сообщений в различных операционных системах на различных аппаратных платформах.

Программный комплекс универсален и может работать под управлением операционной системы Windows или Linux (с установленной графической оболочкой). Генерируемые параллельные программы могут работать в любой операционной системе, для которой имеется реализация MPI. Например, для создания программы, ориентированной на Linux-кластер, необходимо лишь наличие библиотеки MPI и компилятора для соответствующей версии Linux. Выбор языка программирования для описания базовых модулей также ограничен лишь наличием реализации MPI для этого языка. В настоящее время реализована версия программного комплекса PGRAPH 2.0, использующая для написания базовых модулей язык C++.

Архитектура программного комплекса соответствует архитектуре, представленной на рисунке 3. Для взаимодействия с пользователем используется графический оконный интерфейс. Подсистема управления параллельными вычислениями интегрирована с графическим редактором граф-моделей. Благодаря этому, при выполнении различных действий в системе, пользователь всегда видит текущий агрегат, над которым эти действия выполняются. На рисунке 4 показано главное окно граф-редактора.

Информационный фонд программного комплекса хранится в базе данных, поддерживаемой СУБД MySQL [33]. Выбор этой СУБД обусловлен тем, что она имеет реализации для различных операционных систем, поддерживает язык структурированных запросов к базам данных SQL (Structured Query Language) и распространяется свободно.

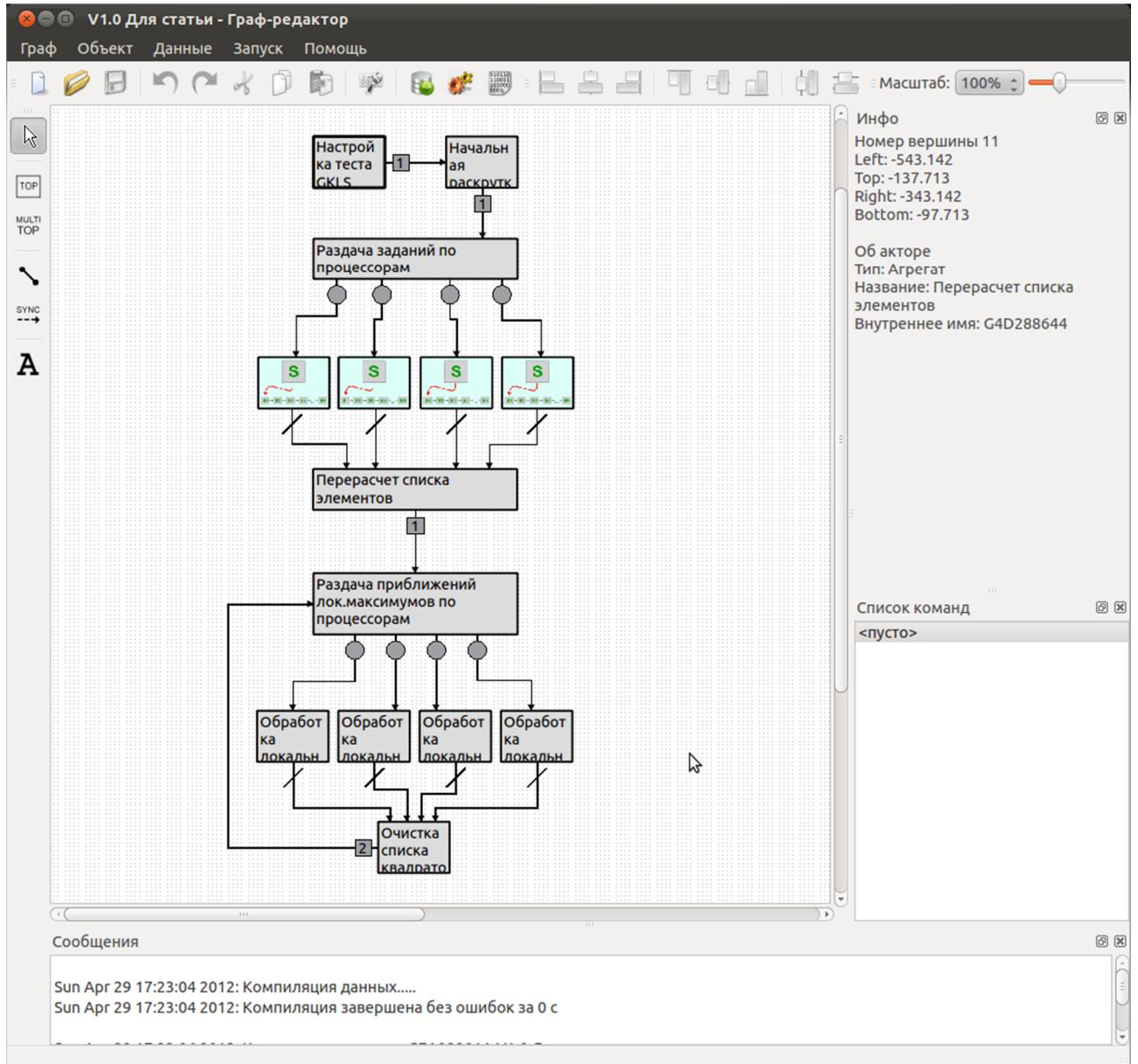


Рисунок 4 – Главное окно программного комплекса моделирования и анализа алгоритмов параллельных вычислений PGGRAPH 2.0

В систему интегрирован редактор текстов, предназначенный для написания исходных текстов базовых модулей и выполняющий простейшие операции по проверке их синтаксической корректности, а также редактор

объектов технологии ГСП. Компилятор объектов выполнен в виде отдельного модуля. Он позволяет на основании описания объектов, хранящегося в информационном фонде системы, генерировать исходные тексты на языке C++. В качестве компилятора исходных текстов используется компилятор для целевой платформы, поддерживающий конкретную библиотеку MPI. Для Linux кластеров имеется большое число различных реализаций MPI и компиляторов. Для Windows систем наиболее типичен вариант сочетания библиотеки MPICH2 и компилятора Microsoft Compiler.

### 2.5.3 Генерация исходных текстов параллельных программ на языке C++

Для построения исполняемых файлов программный комплекс генерирует исходный текст программы на языке C++. В процессе генерации используется описание объектов технологии ГСП, хранящееся в информационном фонде. Полученные исходные тексты подаются на вход компилятора C++, который автоматически вызывается программным комплексом. Генерируемая программа на языке C++ состоит из нескольких модулей, связь между которыми схематично изображена на рисунке 5.

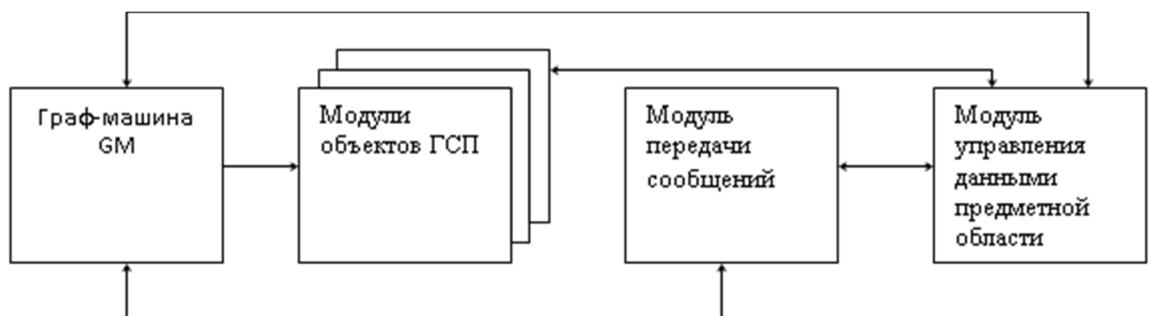


Рисунок 5 – Структура программы, автоматически генерируемой на основе граф-модели алгоритма

Для каждого объекта технологии ГСП компилятор объектов генерирует отдельный файл с исходным текстом. Для агрегатов этот файл состоит из двух частей:

1. заголовка, содержащего структуры данных, описывающих граф-модель;
2. вызова отдельной процедуры, реализующей работу граф-машины.

Для акторов и предикатов генерируется файл с исходным текстом реализующего их базового модуля, в котором все обращения к данным предметной области заменены на специальные конструкции в соответствии с межмодульным информационным интерфейсом параллельной программы.

#### **2.5.4 Компилятор данных**

Компилятор данных собирает из типов и данных ПОП класс TPOData. В основном при создании класса TPOData решается задача определения типа переменной.

Выходные файлы строятся компилятором на основе шаблонов. В шаблоне определены теги, которые заменяются компилятором на соответствующие конструкции. Всего имеется три файла шаблона:

- utypes.h.template – шаблон, в котором описаны типы данных;
- tpodata.h.template – заголовочный файл класса TPOData;
- tpodata.cpp.template – исходный текст класса TPOData.

По количеству шаблонов создается три выходных файла. Для каждого определенного типа данных в файле utypes.h имеется запись. Если речь идет о программе на MPI, то в файле utypes.h создается соответствующий тип MPI для каждого пользовательского типа. В соответствии со стандартом MPI каждый новый пользовательский тип должен быть зарегистрирован в программе перед его использованием. Код для регистрации нового типа генерируется автоматически.

Для каждой переменной из ПОП в классе TPOData создается поле-свойство, для доступа к переменной и метод установки и получения значений, работающие с этим свойством. Класс TPOData описываются в файле tpodata.h, а его реализация, соответственно, помещается в файл tpodata.cpp.

### 3 Модифицированный метод половинных делений

#### 3.1 Постановка задачи и основные определения

Без ограничения общности, рассмотрим задачу безусловной глобальной оптимизации (ГО) непрерывной функции  $f : \mathbf{R}^n \rightarrow \mathbf{R}$ , заданной на допустимом множестве  $X \in \mathbf{R}^n$  в следующей постановке:

$$f_* = \operatorname{glob} \min_{x \in X} f(x) = f(x_*). \quad (3)$$

Положим, что глобальный минимум  $x_*$  принадлежит множеству  $X_*$ ,

причем  $X_* \subset X$ , а  $X = \bigotimes_{i=1}^n [0, 1]$ , является многомерным единичным

гиперкубом. Далее используются условные обозначения и основные определения, заимствованные из работы [34].

Определим множество  $\varepsilon$ -решений задачи (3) следующим образом:

$$X_*^\varepsilon = \{x \in X : f(x) \leq f_* + \varepsilon\}.$$

Нахождение приближенного решения задачи (3) заключается в поиске хотя бы одной точки множества  $X_*^\varepsilon$ .

Впервые метод неравномерных покрытий был предложен Ю.Г. Евтушенко в 1971 году [35]. Кратко опишем его основные положения, поскольку предлагаемый в данной работе модифицированный метод половинных делений во многом основывается на идеях этого метода.

В методе неравномерных покрытий строится последовательность  $\{X_i\} = X_1, \dots, X_k$  подмножеств множества  $X$  и точек  $x_1, \dots, x_k$  ( $x_i \in X_i$ ). В каждой точке  $x_i$  вычисляется значение функции  $f(x)$  и определяется её рекордное значение  $f_r$ :

$$f_r = \min_{1 \leq i \leq k} f(x_i) = f(x_r), \quad 1 \leq r \leq k. \quad (4)$$

На каждом из подмножеств  $X_i$  вводятся *миноранты*, т.е. такие функции  $G_i(x)$ , что  $\forall x \in X_i \quad f(x) \geq G_i(x)$ . Тогда последовательности подмножеств  $\{X_i\}$

можно поставить в соответствие совокупность покрывающих подмножеств  $\{Z_i\} = Z_1, \dots, Z_k$ , определяемых из условия:

$$Z_i = \{x \in X_i : G_i(x) \geq f_r - \varepsilon\}, \quad r \leq i.$$

Множество  $Z_i$  строится таким образом, что  $\forall x \in Z_i$  выполняется неравенство  $f(x) \geq f_r - \varepsilon$ , из чего следует, что глобальный минимум функции  $f(x)$  на множестве  $Z_i$  не может быть меньше рекорда  $f_r$  более чем на  $\varepsilon$ . Таким образом, в процессе поиска глобального минимума подмножество  $Z_i$  можно исключить из рассмотрения и продолжить оптимизацию на множестве  $X_i \setminus Z_i$ . Алгоритм оптимизации останавливается, когда допустимое множество оказывается покрытым подмножествами  $Z_i$ , т.е.

$$X \subseteq Z_1 \cup Z_2 \cup \dots \cup Z_k.$$

Алгоритм неравномерных покрытий, предложенный в работах [6, 40], гарантирует нахождение  $\varepsilon$ -оптимального решения, что подтверждается основополагающей теоремой [40].

### 3.2 Метод половинных делений

Идея метода половинных делений [35] заключается в организации непропорционального деления исходной области поиска (обычно – единичного гиперкуба) на гиперпараллелепипеды, меньшей размерности. Чаще всего реализуется дихотомическое деление параллелепипедов по наибольшему ребру [34, 36, 37]. При этом (для двухмерного пространства) единичный куб делится на две прямоугольные области, которые затем делятся на квадраты четвертной размерности т.д. как показано на рисунке 6. В результате в пространстве поиска формируется нерегулярная сетка испытаний исследуемой функции. Возможен вариант непропорционального деления параллелепипеда по его главной диагонали [38].

Пусть допустимое множество  $X$  делится на прямоугольные параллелепипеды  $X_i$  с центрами  $c_i$  и главными диагоналями  $2R_i$ , где  $R_i$  –

радиус параллелепипеда  $X_i$  ( $R_i = \max_{x \in X_i} \|x - c_i\|$ ). Метод половинных делений

[35] сводится к построению последовательности  $\{X_i\} = X_1, \dots, X_k$  параллелепипедов подмножеств множества  $X$  и точек  $x_1, \dots, x_k$  ( $x_i \in X_i$ ). В каждой точке  $x_i$  вычисляется значение функции  $f(x)$  и определяется её рекордное значение  $f_r$  по формуле (4).

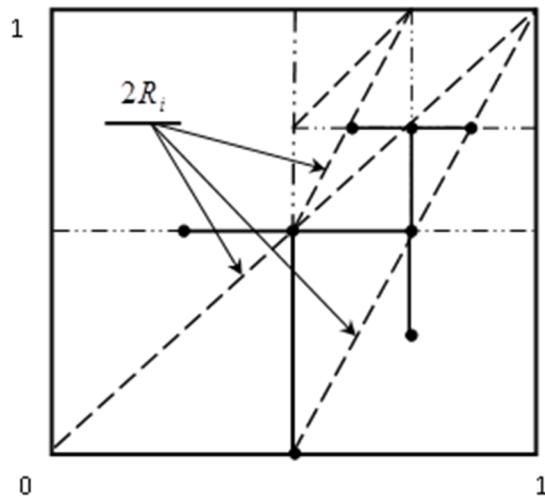


Рисунок 6 - Деление допустимого множества

Для функций, удовлетворяющих условию Липшица, на каждом шаге алгоритма выбирается «критический» параллелепипед, для которого

$$\varphi_s = \min_{1 \leq i \leq k} f(c_i) - 2LR_i.$$

Именно он подвергается дальнейшему делению на две части.

Используя условие Липшица, параллелепипеды, для которых  $\varphi_j \geq f_r$  исключаются из дальнейшего рассмотрения. Здесь  $f_r$  - текущий рекорд при поиске минимума функции.

Для остановки алгоритма можно использовать условие:

$$R_i \leq \varepsilon.$$

### 3.3 Выбор критического параллелепипеда

Сохранив схему двоичного деления параллелепипедов [36], изменим правило выбора «критического» параллелепипеда, подвергающегося дальнейшему двоичному делению. Произвольный выбор «критического» параллелепипеда [36], или ориентация на параллелепипед, имеющий минимальное текущее значение функции в точке  $c_i$  [37], может привести к чрезмерно детальному «изучению» окрестностей локальных минимумов функции и «отсрочить» нахождение её нового рекорда  $f_r$ , т.е. увеличить трудоемкость алгоритма.

В работе [23] для одномерных функций предложена одна из самых эффективных стратегий многоэкстремальной оптимизации, основанная на использовании приближенного апостериорного распределения вероятностей расположения глобального экстремума, формируемого в процессе испытаний функции, что реализует более сбалансированную организацию поиска глобального минимума функции. Данная стратегия настолько эффективна, что её часто переносят с одномерного случая на случай оптимизации функций многих переменных. Например, в работе [39] многомерная задача оптимизации, с помощью кривых Пеано, редуцируется к одномерной задаче ГО, в которой используется характеристическая схема оптимизации Стронгина Р. Г. Аналогичные идеи используются в диагональном методе ГО Я.Д. Сергеева [38]. Попытаемся использовать эту стратегию и для метода половинных делений.

Если в процессе двоичных делений соединять центры параллелепипедов предшественников и потомков, то получится семейство непрерывных непересекающихся линий – траекторий перемещения центров параллелепипедов, на рисунке 7 отмечены жирной линией, для которых можно вычислять характеристики соответствующих участков кривых по аналогии с одномерным случаем [23]. Оценки характеристик, полученных для участков траекторий перемещения центров параллелепипедов, припишем соответствующим параллелепипедам  $X_i$  с центрами в точках  $c_i$ .

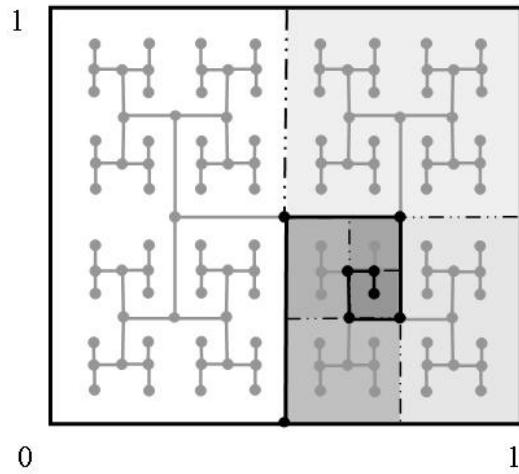


Рисунок 7 – Траектория перемещения центров параллелепипедов

Таким образом, каждый параллелепипед  $X_i$  свяжем с набором параметров  $D_i = (K_i, c_i, f_i, h_i, R_i)$ . Здесь  $c_i$  - центр параллелепипеда,  $f_i$  - значение функции в точке  $c_i$ ,  $R_i$  - радиус параллелепипеда,  $h_i$  - расстояние между текущим параллелепипедом и его предшественником,  $K_i$  - характеристика  $i$ -го параллелепипеда, которую в соответствии с можно вычислить по формуле:

$$K_i = wh_i + \frac{(f_i - f_p)^2}{wh_i} - 2(f_i + f_p),$$

где  $f_p$  - значение функции, вычисленное в предшествующем параллелепипеде в точке  $c_p$ ;  $w$  - оценка константы Липшица. Последняя, адаптивно вычисляется в процессе вычислений функции:

$$w = \begin{cases} 1, & L = 0, \\ rL, & L > 0, \end{cases} \quad L = \max_{1 < i \leq k} \frac{|f_i - f_p|}{h_i},$$

где  $r > 1$  - коэффициент.

Из наборов характеристик  $D_i$  сформируем упорядоченный в направлении убывания характеристики  $K_i$  список  $D = \{D_{i_1}, D_{i_2}, \dots, D_{i_k}\}$ , где  $K_{i_1} \geq K_{i_2} \geq \dots \geq K_{i_k}$ ,  $1 \leq i_j \leq k$ .

Теперь в методе половинных делений в качестве «критического» параллелепипеда можно выбрать первый параллелепипед из списка  $D$ , т.е. параллелепипед, для которого выполняется условие:  $K_1 = \max_{1 \leq i \leq k} K_i$ .

Следует отметить, что в списке  $D$  на любом этапе работы модифицированного алгоритма половинных делений (МАПД) содержатся параллелепипеды, ещё не подвергнутые двоичному делению. Естественно, что «критические» параллелепипеды каждые раз удаляются из списка. Работа МАПД заканчивается, когда список  $D$  пуст.

### **3.4 Двухфазный алгоритм метода половинных делений (ДАМПД)**

Как любой алгоритм ГО, алгоритм половинных делений имеет экспоненциальный характер роста сложности в зависимости от числа переменных. Одной из эффективных стратегий совершенствования алгоритмов ГО является использование *локальной техники* [40], когда стратегия глобального поиска удачно сочетается с методами локальной оптимизации.

В этом случае основной задачей на этапе ГО является определение областей притяжения локальных экстремумов функции, из которых можно запустить алгоритмы локальной оптимизации, поэтому его можно производить достаточно грубо. При ограниченном количестве минимумов функции  $f(x)$  области притяжения  $S_i$  имеют значительные размеры. Предположим, что относительно оптимизируемой функции известно количество локальных минимумов –  $r$  (включая глобальный) и размеры областей притяжения:  $\rho_1, \dots, \rho_r$ . Введем понятие гарантированного радиуса области притяжения минимумов функции, под которым будем понимать  $\rho_m = \min_{1 < i \leq r} \rho_i$ .

Дополнительная информация о размерах областей притяжения локальных минимумов функции позволяет построить двухфазную схему алгоритма ГО, в которой выделим фазы глобальной и локальной оптимизации. Идея разделения фаз алгоритма показана на рисунке 8.

В ДАМПД фаза ГО реализуется с помощью двоичного деления, с той лишь разницей, что деление параллелепипедов в МАПД происходит до достижения заданных размеров их радиусов  $R_i$ . Величина  $R_i$  определяется размерами гарантированного радиуса  $\rho_m$  области притяжения глобального минимума. На рисунке 8 пунктирными линиями отмечены гарантированные области притяжения минимумов функции. Заштрихованные прямоугольники являются прямоугольниками, «отбракованными» с использованием константы Липшица. В итоге, исходя из условия  $R_i \leq \rho_m$ , фаза ГО завершилась, породив, для приведенного на рисунке 7 примера, 24 прямоугольника заданного радиуса.

В фазе локальной оптимизации из точек, принадлежащих областям притяжения локальных минимумов, организуется поиск минимумов функции с помощью локальных алгоритмов оптимизации. В нашем случае использовался метод деформированных многогранников [41, 42].

Такая схема организации процедуры поиска глобального минимума функции существенно сокращает трудоемкость фазы ГО.

### **3.5 Алгоритм построения множества точек начальных приближений для алгоритма локальной оптимизации**

Для больших размерностей вектора независимых переменных оптимизируемой функции в фазе ГО порождается значительное количество параллелепипедов заданного размера. В этом случае объем вычислений в фазе локальной оптимизации становится чрезмерно большим. Рассмотрим следующий адаптивный алгоритм формирования списка точек начальных приближений областей притяжения локальных минимумов оптимизируемой функции, осуществляющий сжатие количества точек начальных приближений, используемых во второй фазе алгоритма.

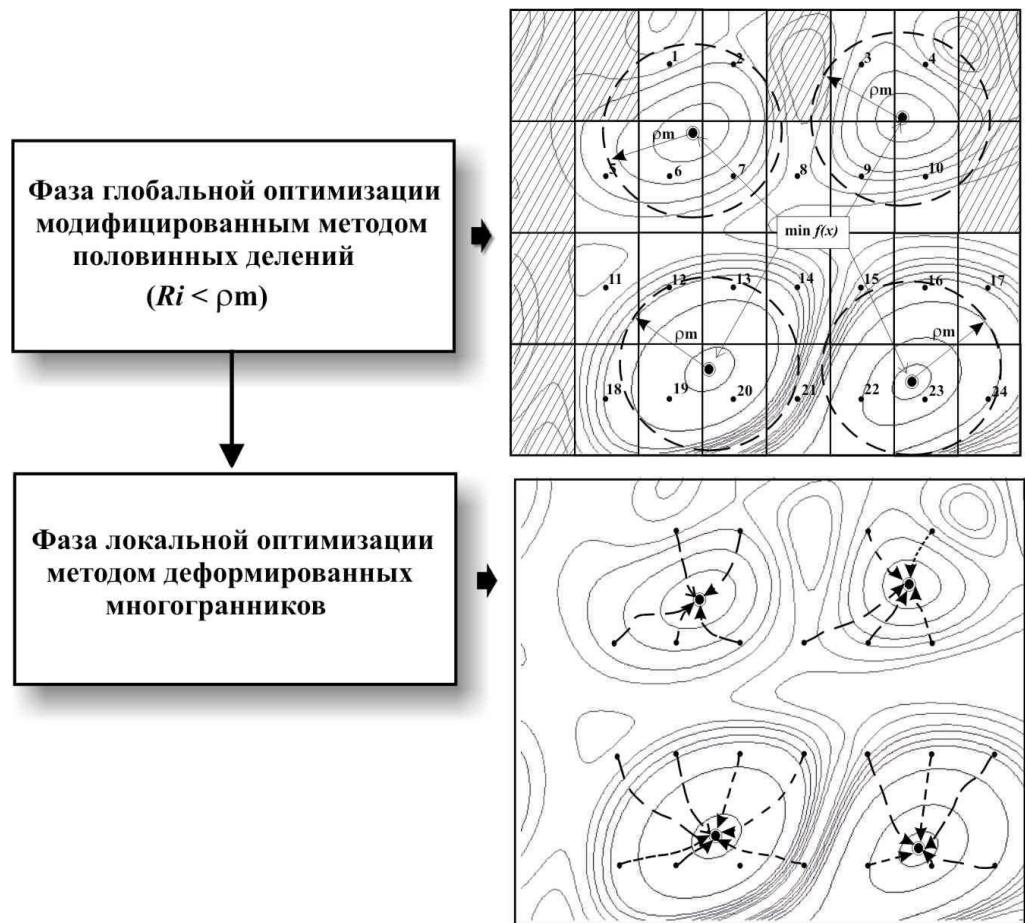


Рисунок 8 – Двухфазный алгоритм метода половинных делений

Будем считать, что область притяжения  $S_i$  определена, если она содержит, по крайней мере, одну точку из множества  $C = \{c_1, \dots, c_k\}$ .

Пусть  $r$  – количество зон притяжения минимума функции;  $\rho_m$  – гарантированный радиус области притяжения минимума функции, обеспечивающий нахождение всех локальных минимумов.

Сформируем список областей притяжения локальных минимумов  $V = \{V_1, V_2, \dots, V_m\}$ , элементами которого являются структуры  $V_i = (\tilde{c}_i, \tilde{f}_i)$ , где  $\tilde{c}_i$  – координаты «представителя»  $i$ -й области притяжения, имеющей наилучшую достигнутую для этой области оценку  $\tilde{f}_i$ . Вектор  $\tilde{c}_i$  условно считается центром  $i$ -ой области притяжения.

Первоначально список  $V$  – пуст. По мере вычислений функции он наполняется элементами, но в конце этапа глобального поиска не может

содержать больше  $m$  элементов ( $m$  – заданный размер списка,  $m \geq r$ ). Размер списка является эмпирическим параметром  $m$  и зависит от свойств оптимизируемой функции и выбирается из соображений попадания в него зоны притяжения глобального минимума функции. Элементы списка  $V$  упорядочены таким образом, что  $\tilde{f}_1 < \tilde{f}_2 < \dots < \tilde{f}_m$ .

Пусть в процессе работы алгоритма ГО произведено очередное испытание  $f_k = f(c_k)$ . Эволюция содержимого списка  $V$  происходит по следующим простым правилам:

1. Проверяется принадлежность центра очередного параллелепипеда  $c_k$  окрестностям одной из имеющихся областей притяжения  $V_i$   $i=1,\dots,l$  ( $l$  – текущий размер списка).

1.1. Если выполняется условие

$$c_k \in \{x \in X : \|(x - \tilde{c}_i)\| \leq \rho_m\}. \quad (5)$$

1.1.1. То при  $f_k < \tilde{f}_i$  содержимое элемента  $V_i$  обновляется:

$$\tilde{c}_i = \frac{1}{n_i} \sum_{j=1}^{n_i} c_j, \quad \tilde{f}_i := f_k, \text{ где } c_j \text{ - центры параллелепипедов}$$

«попавших» в область  $V_i$ . Далее выполняется действие 1.1.3.

1.1.2. Иначе уточняется только значение  $\tilde{c}_i$ . Выполняется действие 1.1.3.

1.1.3. Список  $V$  упорядочивается в порядке возрастания  $\tilde{f}_i$ .

1.2. Если условие (5) не выполняется, то проверяем правило 2.

2. Определяется возможность включения нового элемента в список  $V$ .

2.1. Если  $(f_k < \tilde{f}_1)$ , то элемент  $V_k$  записывается в голову списка  $V$ .

2.2. Если  $(\tilde{f}_j < f_k \leq \tilde{f}_{j+1})$ , то элемент  $V_k$  записывается между  $V_j$  и  $V_{j+1}$  элементами списка.

2.3. Если  $(f_k > \tilde{f}_l)$ , то элемент  $V_k$  записывается в конец списка  $V$ .

3. При превышении предельного числа элементов списка, из списка исключается последний элемент списка.

Предложенный алгоритм является эвристическим, однако, вычислительные эксперименты с тестовыми функциями показали его высокую эффективность. Более того, как правило, в первой сотне элементов списка содержится начальное приближение глобального минимума функции. На рисунке 9 представлены элементы списка областей притяжения. Нумерация соответствует весу элемента. Из рисунка видно, что каждый элемент списка сформировался, агрегируя большое количество точек вычисления функции. Например, элемент  $V_1$  агрегировал девять точек,  $V_2$  – восемь, и т.д.

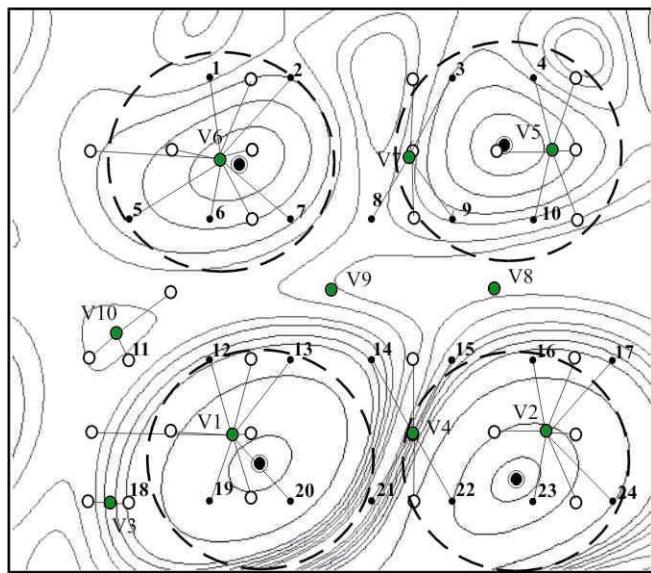


Рисунок 9 – Формирование списка областей притяжения

### 3.6 Параллельная версия двухфазного алгоритма метода половинных делений

Очевидно, что затраты времени на решение задачи ГО можно, в известной степени, сократить за счет распараллеливания вычислительного процесса. Однако архитектура современных кластеров с распределенной памятью значительно затрудняет распараллеливание алгоритмов ГО, поскольку вынужденно приходится заниматься организацией эффективной передачи данных между процессорами, а не непосредственным распараллеливанием

вычислений. При этом обычно используется «тяжеловесный», но естественный для подобных систем стандарт MPI. Как показала практика, наилучшие результаты по критериям оценки качества распараллеливания вычислений (ускорению и эффективности) получаются, когда удается свести к минимуму обмен информацией между процессорами.

Для двухфазного алгоритма метода половинных делений для первой фазы ГО за каждым процессором можно закрепить один параллелепипед из списка  $D = \{D_{i_1}, D_{i_2}, \dots, D_{i_k}\}$ , построенного МАПД. Очевидно, что параллельные вычисления будут эффективны, если параллелепипеды будут иметь одинаковый радиус. В этом случае, если не учитывать «отбраковку» параллелепипедов по критерию Липшица, на каждом процессоре будет произведено одинаковое количество вычислений до достижения заданного уровня разбиения параллелепипедов на части. Из свойств самого алгоритма очевидно, что этого можно добиться, если допустимую область разбивать на  $2^n$  равных частей.

Алгоритм ГО ММПД состоит из двух этапов. Реализация алгоритма в рамках нотации ГСП представлена на рисунке 10.

В вершине 1 устанавливаются значения параметров теста GKLS и его инициализация [42]. В первый этап алгоритма входят вершины 2-5. В вершине 2 вычисляется первое значение функции в точке  $x[i]=0,5$ ,  $i=0..N$ ,  $N$  - размерность пространства и в характеристический список  $D$  добавляется первый гиперкуб, охватывающий все пространство. В этой же вершине с помощью последовательного варианта алгоритма двоичного деления, описанного выше, формируется начальный характеристический список  $D$  с числом элементов не менее числа процессоров. При начальном делении областей прореживание выключено и на каждом шаге мы точно можем подсчитать количество элементов и состав списка. Очевидно из самого алгоритма половинного деления, что через  $2^n$  шагов список будет состоять из

областей равного размера. Это уравнивает начальные условия для последующего параллельного деления.

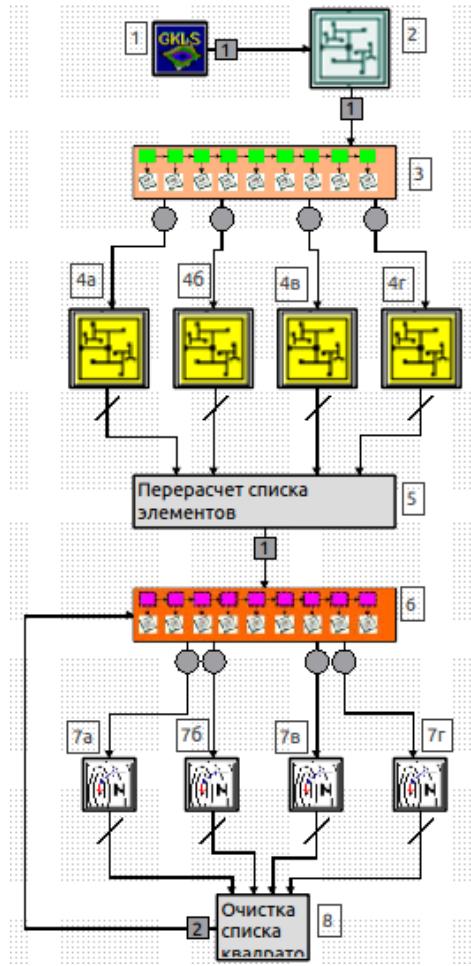


Рисунок 10 – Граф-модель алгоритма ГО ММПД

Первый этап алгоритма осуществляет параллельное половинное деление и глобальный поиск в вершинах 4a-4г. На рисунке 10 приведён вариант алгоритма для четырех процессоров, но алгоритм легко масштабируется на произвольное число процессоров. В вершине 3 происходит раздача заданий по процессорам. Каждый процесс получает один параллелепипед. Вершины 4a-4г имеют одинаковые агрегаты и запускаются каждая на своём процессоре. На каждом из процессоров с помощью МАПД реализуется покрытие частных параллелепипедов (в нашем примере  $X_1, X_2, X_3, X_4$ ), причем их дробление производится до достижения определенных размеров. При этом формируются

списки начальных приближений зон притяжения локальных минимумов. В вершине 5 эти списки объединяются.

Второй этап осуществляет поиск локального минимума из точек определённых во втором этапе. Этап состоит из вершин 6-8. В вершине 6 мастер ветвь раздает начальные точки для локального поиска. В вершинах 7а-7г запускается локальный поиск максимального значения функции с помощью метода деформированных многогранников. В вершине 9 происходит подсчёт количества найденных минимумов. Выходом из второго этапа служит опустошение списка начальных областей или выполнение необходимого минимума запусков локального поиска. Если имеются непроверенные области локальных минимумов, то перед завершением программы они удаляются.

## **4 Исследование эффективности и работоспособности алгоритма глобальной оптимизации**

### **4.1 Условия тестирования и экспериментов по оценке эффективности**

Для вычислительных экспериментов был выбран класс липшицевых функций, моделируемых известным генератором GKLS. Генератор тестовых задач GKLS порождает три класса тестовых функций: недифференцируемых, непрерывно дифференцируемых и дважды непрерывно дифференцируемых. Для каждого класса формируется 100 тестовых функций. В GKLS очень просто задать сложность генерируемых функций, определяя количество локальных минимумов, размеры областей притяжения и многое другое.

Тестирование алгоритма ДАМПД проводилось на наиболее сложном для реализации алгоритмов поиска глобального минимума, классе недифференцируемых функций. Для всех классов задач: число экстремумов равно 10; глобальный минимум равен минус 1; радиус притяжения глобального оптимума – 0,33. Эксперименты проводились на суперкомпьютерном кластере СГАУ «Сергей Королев». Кластер построен на базе линейки оборудования IBM BladeCenter с использованием блейд-серверов HS22 и обеспечивает пиковую производительность более 10 триллионов операций с плавающей точкой в секунду. Общее число процессоров/вычислительных ядер: 272/1184. Глобальный минимум вычислялся с точностью  $\varepsilon = 1.0 \cdot 10^{-8}$  (по аргументам функции).

### **4.2 Первая версия алгоритма**

Сначала был проверен базовый вариант двухфазного алгоритма глобальной оптимизации (ГО) описанный в главе 3. Результаты первого эксперимента приведены в таблице 4.

Таким образом, общее ускорение составило 141,121. На рисунках 11 и 12 приведены гистограммы, показывающие количество обращений к функции,

выполненное на каждом процессоре. Стоит отметить, что этап ЛО повторяется несколько раз, а гистограмма на рисунке показана только для одной итерации.

Таблица 4 – Результаты эксперимента с базовой версией алгоритма ГО ММПД

Время работы алгоритма, сек	88,46
Время накладных расходов, сек (% от общего времени)	3,34 (3,75)
Число обращений к функции на этапе ГО (суммарно на всех процессорах)	21152 (2937912)
Число обращений к функции на этапе ЛО (суммарно на всех процессорах)	21095 (3024009)

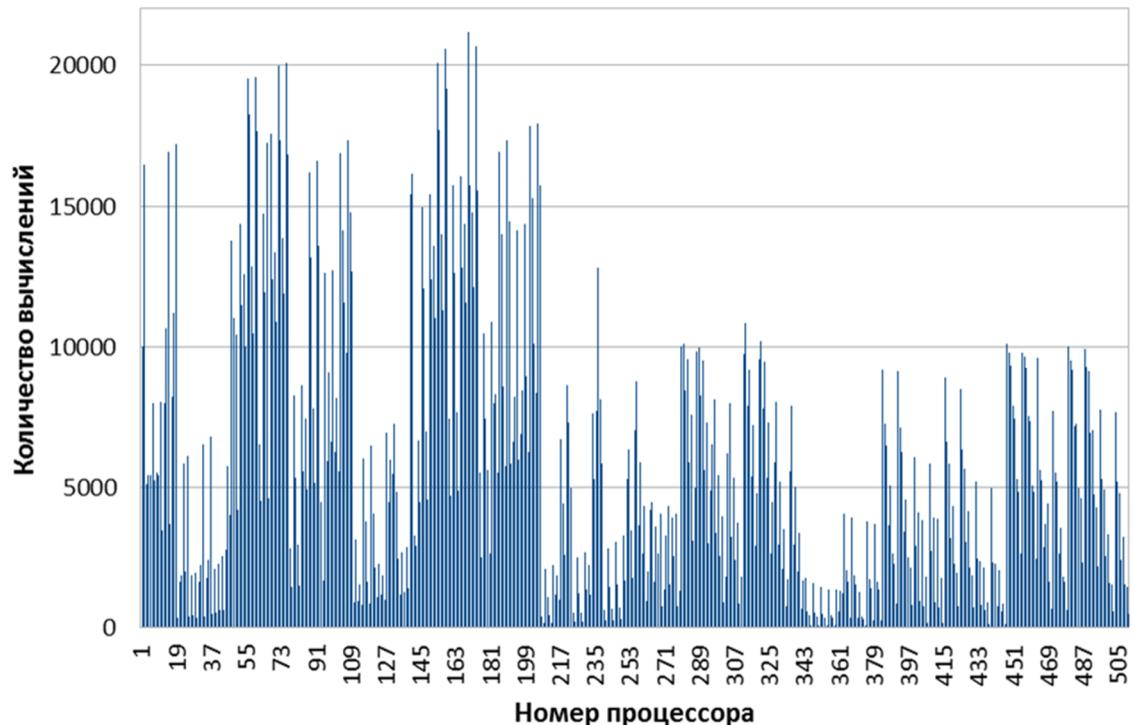


Рисунок 11 – Загрузка процессоров на этапе ГО в базовой версии

Из рисунков видно, что общая эффективность алгоритма невысока как на этапе ГО та и на этапе ЛО. Из-за прореживания на этапе ГО большая часть процессоров завершают деление своего начального параллелепипеда досрочно и вынуждены ждать завершения самого «нагруженного» процесса. Неравномерность распределения нагрузки на этапе ЛО обусловлена особенностью используемого алгоритма деформированных многогранников.

Алгоритм сходится к локальному минимуму быстрее на более крутом участке и «долго ползет» на более пологом. Во-вторых, удаленность начальной точки от локального оптимума определяет различное число вычислений функции на этапе ЛО.

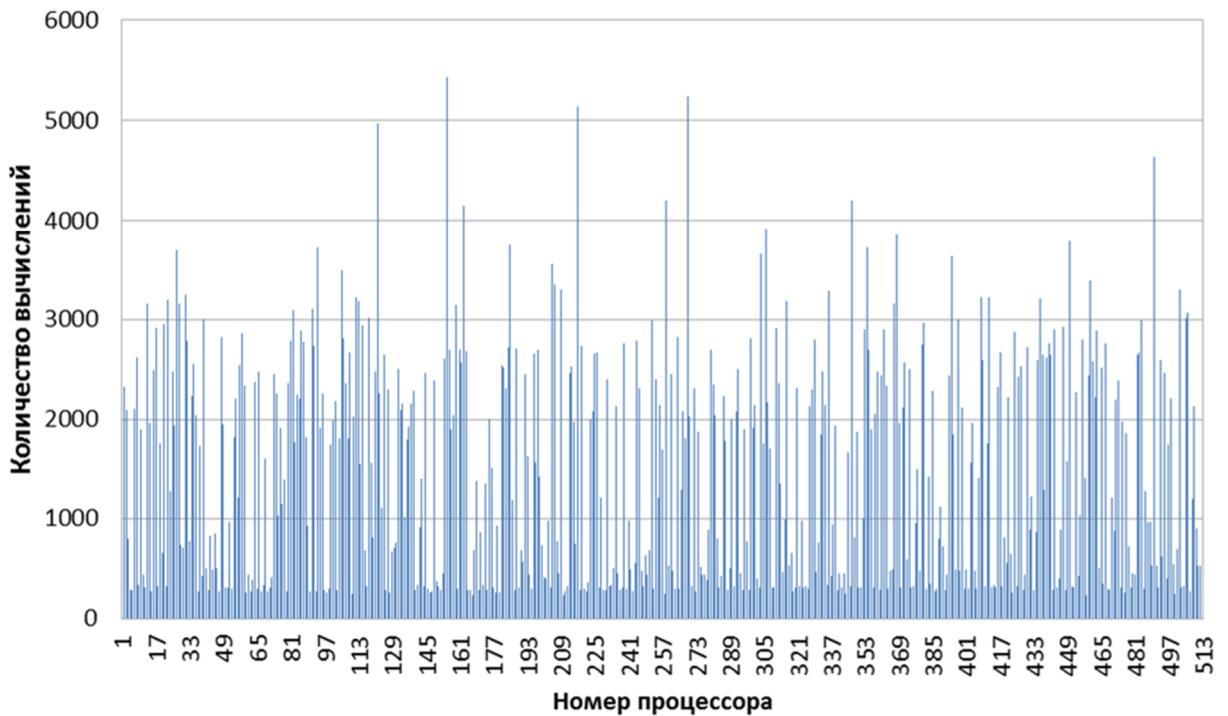


Рисунок 12 – Загрузка процессоров на этапе ЛО в базовой версии

#### 4.3 Вторая версия алгоритма

Во второй версии параллельного алгоритма ГО с целью повышения эффективности вычислений, каждому из процессоров доступна информация о найденном текущем максимальном значении функции. Совместно с локальной оценкой константы Липшица, знание глобального значения максимума функции значительно сокращает количество проверяемых параллелепипедов по критерию прореживания, описанному в главе 3. С помощью технологии ГСП, модификация алгоритма из первой версии во вторую сводится к замене типа переменной, хранящей значение глобального минимума, с локальной на общую. Результаты второго эксперимента приведены в таблице 5.

Во втором эксперименте общее ускорение составило 102,76. Гистограмма распределение нагрузки на процессорах для этапа ГО показана на рисунке 13.

По сравнению с первой версией алгоритма, как и ожидалось, сократилось общее число вычислений, но при этом эффективность еще более уменьшилась. Общий вид гистограммы для ЛО не поменялся.

Таблица 5 – Результаты эксперимента со второй модификацией алгоритма ГО ММПД

Время работы алгоритма, сек	82,34
Время накладных расходов, сек (% от общего времени)	3,54 (4,29)
Число обращений к функции на этапе ГО (суммарно на всех процессорах)	16728 (840718)
Число обращений к функции на этапе ЛО (суммарно на всех процессорах)	21443 (3081723)

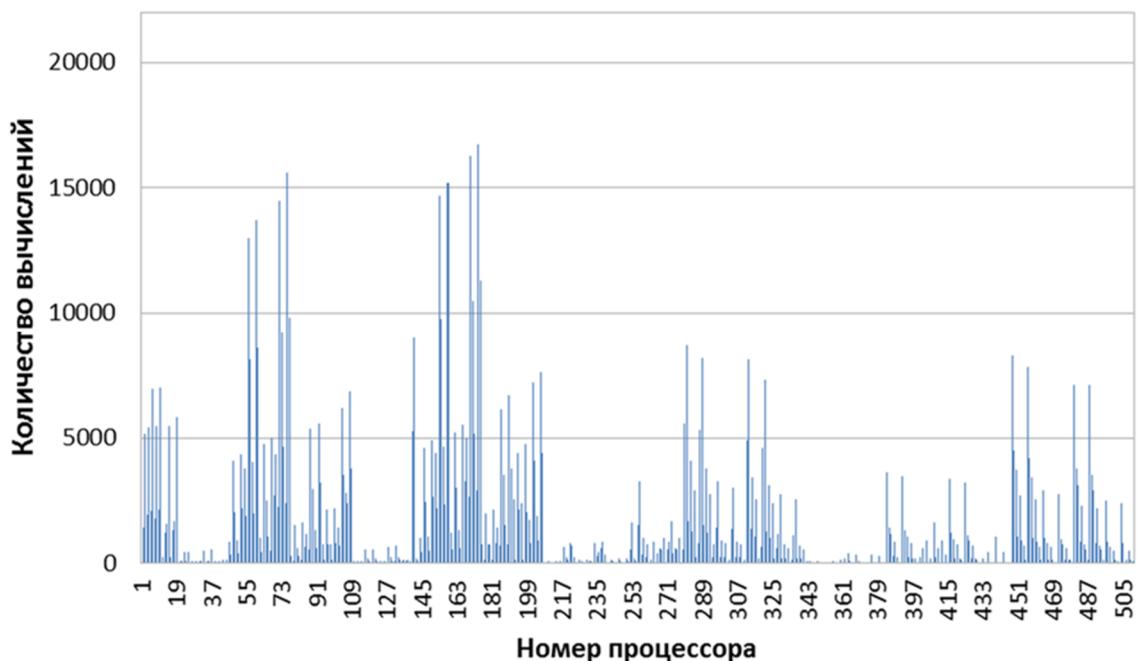


Рисунок 13 – Загрузка процессоров на этапе ГО во второй модификации алгоритма

#### 4.4 Третья версия алгоритма

Диаграммы распределения нагрузки на процессоры в эксперименте 2 говорят о еще меньшей рациональности алгоритма с точки зрения использования процессорного времени. Как и в первом случае это возникает в

связи с неравномерностью прореживания параллелепипедов. В первой случае используется только 30% потенциальной вычислительной мощности, во втором случае менее 20%.

Для повышения эффективности алгоритма необходимо обеспечивать «быстрые» процессоры дополнительным заданием, пока остальные процессоры еще не закончили вычисления. Решением является переход от синхронной модели параллельных вычислений к асинхронной. Сначала попробуем асинхронную модификацию на этапе локального поиска.

Изначально готовится список локальных областей притяжения, больший, чем число процессоров. Все процессоры одновременно запускаются на поиск локального минимума каждый из своей точки. После завершения своего поиска ветвь-менеджер переходит в режим диспетчера и ждет сообщения о завершении от остальных ветвей. После приёма сообщения от любой параллельной ветви, ветвь-менеджер выдает новую точку отработавшей ветви и запускает её. Как только список локальных областей притяжения заканчивается, или проверено необходимое минимальное количество областей, ветвь-менеджер выходит из цикла приёма сообщений и все ветви просто завершают работу.

Для реализации данного подхода требуется либо написания более низкоуровневого актора, использующего функции MPI для ручного управления процессами, либо введение дуг синхронизации в модель алгоритма. Изначально модель параллельных вычислений в технологии ГСП является синхронной, но с помощью дуг синхронизации возможно моделирование асинхронной работы. Пример реализации для четырех процессоров с использованием дуг синхронизации показан на рисунке 14. Данная схема асинхронного управления часто называется менеджер-исполнитель, клиент-сервер и т.д.

Как показали эксперименты, применение асинхронного управления параллельными процессами позволяет значительно повысить ускорение и эффективность этапа поиска локального максимума. На рисунке видно более

равномерное распределение вычислений по процессорам. Так эффективность этапа локального поиска возросла до 72%.

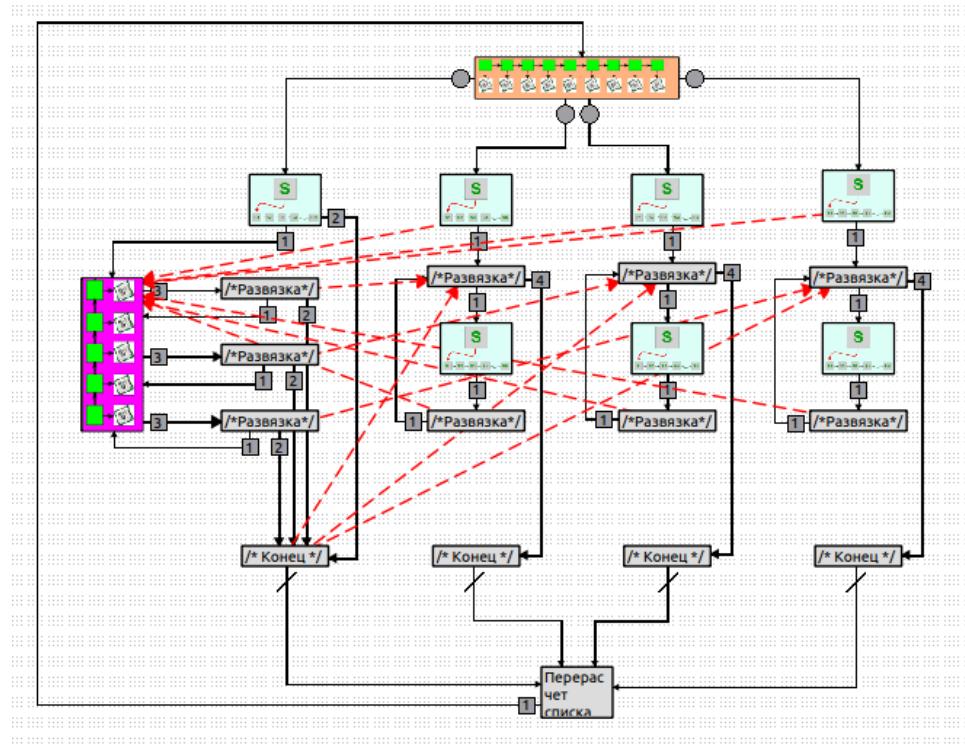


Рисунок 14 – Этап локальной оптимизации в режиме менеджер-исполнитель с синхронизацией

Хорошо показавший себя для этапа локального поиска асинхронный режим управления можно применить и для этапа глобального половинного деления. Для этого, во-первых, на этапе подготовки начального списка параллелепипедов необходимо подготовить параллелепипедов больше, чем число процессоров. Затем с использованием аналогичной схемы вычислений – менеджер-исполнитель – выполнять раздачу новых параллелепипедов уже закончившим работу процессам.

Результаты эксперимента с третьей версией алгоритма приведена в таблице 6.

Таким образом, общее ускорение составило 274,204. На рисунках 15 и 16 приведены гистограммы, показывающие количество обращений к функции, выполненных на каждом процессоре. На рисунках мы видим более

равномерное распределение нагрузки между процессорами. Здесь, в отличие от экспериментов 1 и 2, гистограмма для ЛО показывает общее число вызовов функции на каждом процессоре.

Таблица 6 – Результаты эксперимента с третьей модификацией алгоритма ГО ММПД

Время работы алгоритма, сек	74,4
Время накладных расходов, сек (% от общего времени)	7,87 (10,57)
Число обращений к функции на этапе ГО (суммарно на всех процессорах)	8170 (1949324)
Число обращений к функции на этапе ЛО (суммарно на всех процессорах)	10607 (3199403)

Стоит отметить, что ощутимы результат увеличения эффективности (как в последнем эксперименте) для модели менеджер-исполнитель, достигается для «тяжелых» функций, т.е. время вычисления которых много больше времени передачи параллелепипеда. В основном, такие функции чаще всего встречаются в прикладных задачах науки и техники. Тестовая функция GKLS такой не является, поэтому мы специально увеличивали время ее вычисления.

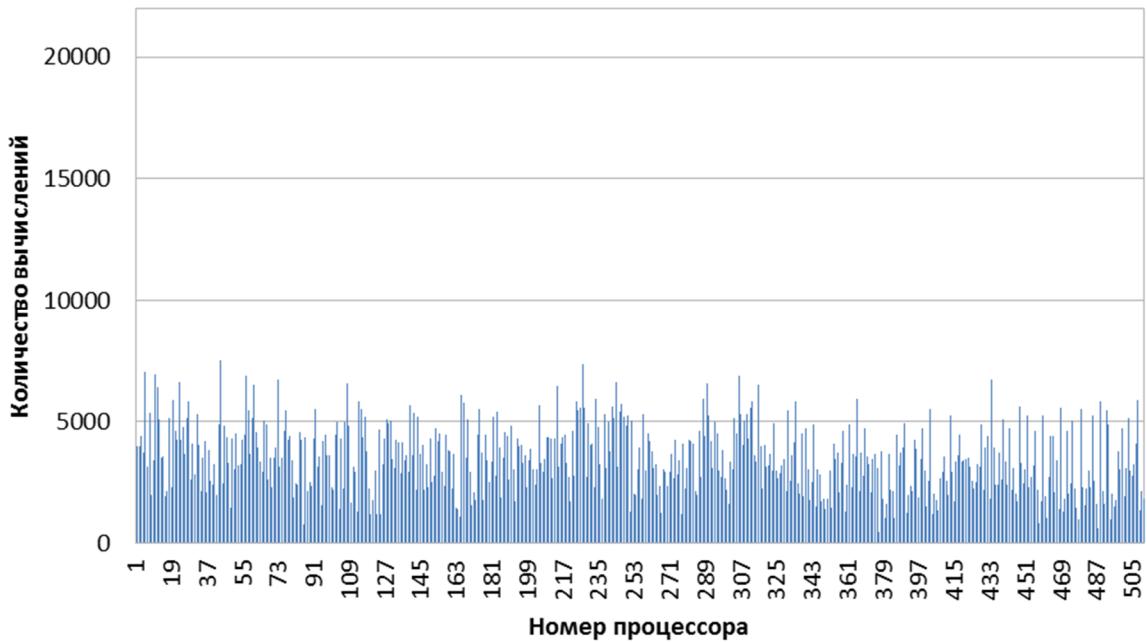


Рисунок 15 – Загрузка процессоров на этапе ГО в третьей модификации

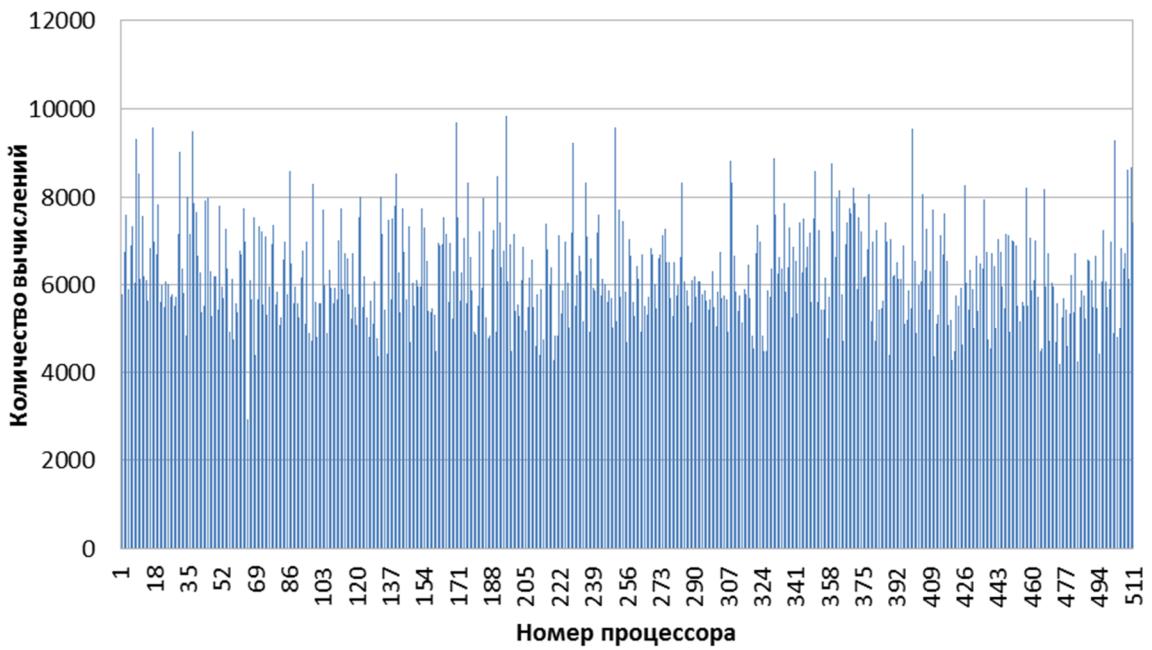


Рисунок 16 – Загрузка процессоров на этапе ЛО в третьей модификации

#### 4.5 Рекорд

В качестве результата проделанных улучшений и настройки алгоритма приведем результаты эксперимента по нахождению глобального минимума для задачи с размерностью 15 той же тестовой функции. Практически это максимальная размерность, достигнутая для данной тестовой функции на суперкомпьютере «Сергей Королев». Параметры радиуса сходимости глобального максимума и количество локальных максимумов функции не поменялось. На рисунке 17 показано как меняется радиус параллелепипеда от шага деления. Оценка количества расчётов для размерности 15 при делении до параллелепипеда со стороной равной 0,25 дает число, порядка  $2 \cdot 10^9$ . С учётом прореживания это число может уменьшиться [2], но ожидать, что оно уменьшится более чем на два порядка не стоит. При такой стороне параллелепипеда его диагональ составляет 0,968, а, следовательно, радиус – 0,484.

Условия эксперимента, параметры метода оптимизации и результаты приведены в таблице 7.

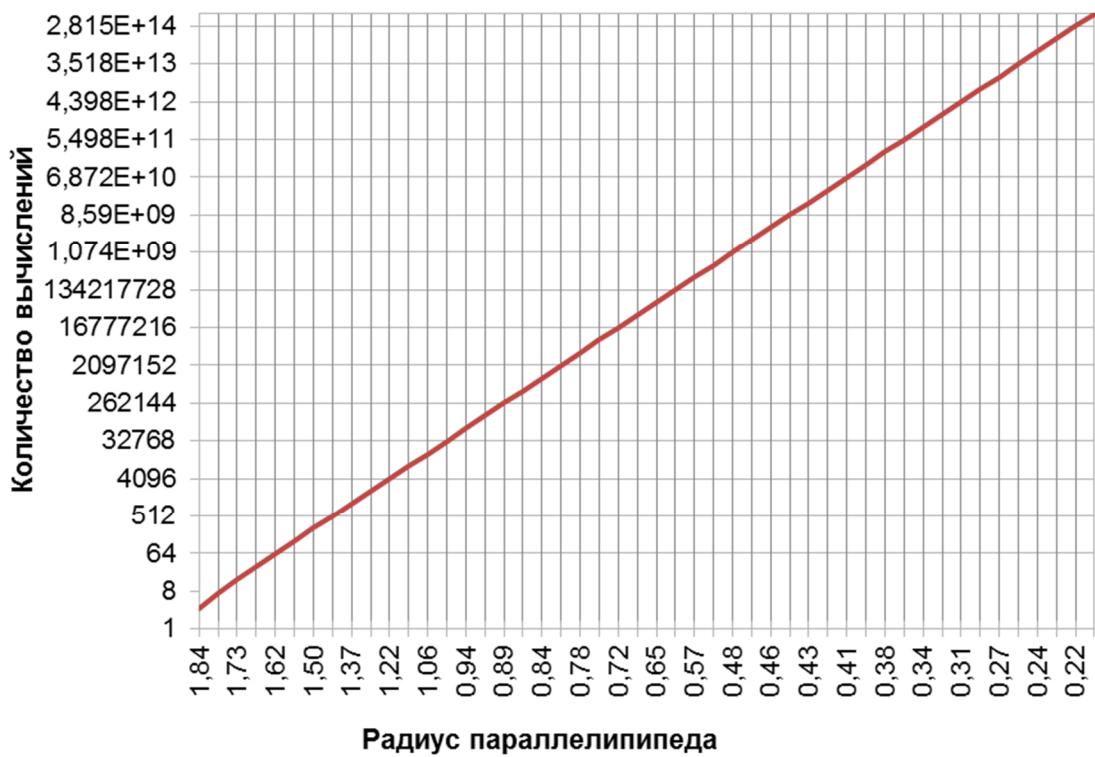


Рисунок 17 – Зависимость числа вычислений функции от диагонали параллелепипеда

Таблица 7 – Исходные данные эксперимента для размерности с 15 переменными

Количество процессоров	512
Радиус зоны притяжения локального максимума	0,2
Номер и тип функции GKLS	15 ND
Глобальный минимум найден	4 раза
Общее число вызовов функций	983770
Общее ускорение	286,18 сек
Ускорение глобального этапа	310,34 сек
Ускорение локального этапа	218,08 сек
Общее время работы алгоритма	2413,79 сек

## **4.6 Выбор оптимальных параметров гасителя пульсаций**

Исследование эффективности работы нового алгоритма, предназначенного для решения практических задач науки и техники, будет неполным без примера решения с помощью этого алгоритма одной из таких задач. Для проверки применимости разработанного алгоритма ГО была решена реальная техническая задача выбора оптимальных параметров гасителя пульсаций давления (ГПД) по критерию оценки среднего уровня акустической мощности. Задача была поставлена на втором факультете СГАУ.

## **4.7 Описание гасителя пульсаций давлений**

Внешний вид гасителя пульсаций давлений (ГПД), установленного на регулятор давления, представлен на рисунке 18. ГПД – это устройство, предназначенное для сглаживания пульсаций и вибраций жидкости и газа, предохраняющее от удара и позволяющее сохранять оборудование в рабочем состоянии более длительное время. Основную конструкцию ГПД составляет специальный клапан, выдерживающий необходимое давление на выходе. Основной шум от работы гасителя формируется на этом клапане.



Рисунок 18 – Внешний вид ГПД

#### 4.8 Математическая модель гасителя пульсаций

Для поиска оптимальных характеристик шайб или перфорированных решеток, обеспечивающих наиболее эффективное снижение акустического шума клапана, ГПД была использована следующая модель глушителя.

Предполагалось, что глушитель представляет собой цилиндрическую трубу как на рисунке 19, в которой через одинаковые промежутки расположены клапан и шайбы (перфорированные решетки).

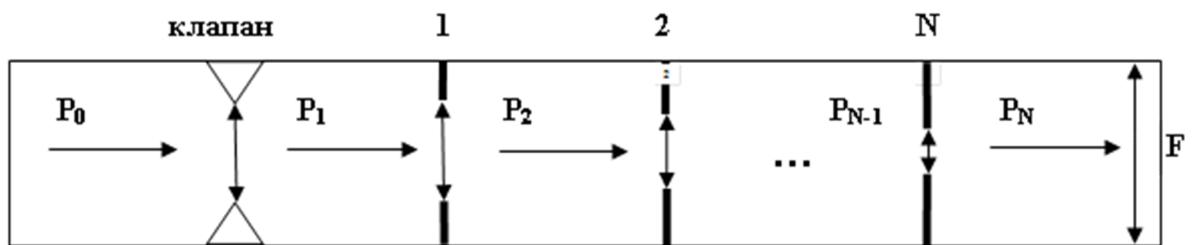


Рисунок 19 – Модель ГПД

При этом предполагалось, что в области за клапаном или шайбой установилось стационарное значение давления, рисунок 20, так что расстояние между шайбами  $L_{i,i-1} > L$ .

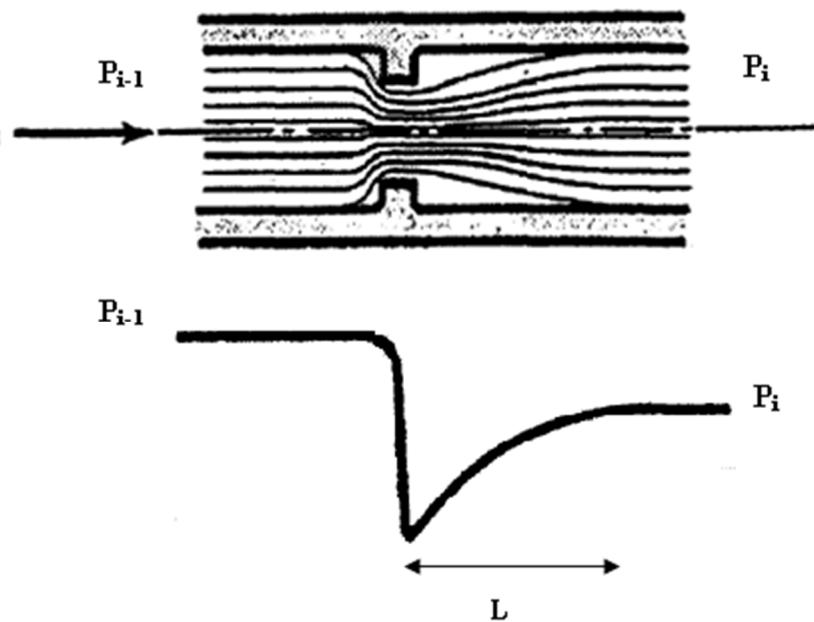


Рисунок 20 – Схематическое распределение давления в шайбе по данным [43]

#### 4.8.1 Акустическая мощность, излучаемая клапаном и N шайбами

Полная акустическая мощность, генерируемая гасителем пульсаций давления, равна сумме мощности клапана и мощностей каждой шайбы ГПД:

$$W = W_1 + \sum_{i=2}^N W_i,$$

то есть полагался выполненным принцип суперпозиции.

Уровень акустической мощности рассчитывается в соответствии со стандартным соотношением:

$$L_w = 10 \lg \frac{W}{10^{-12}} \text{ ДБ}$$

#### 4.8.2 Мощность шума клапана

Для расчета шума клапана использовалась модель, приведенная в [44]. В соответствии с этой моделью, мощность шума клапана (в Вт) вычисляется в соответствии с соотношением

$$W_1 = 7.7 \cdot 10^{-11} c_{vc}^3 \eta_1 c_{v1} F_{l1} P_0,$$

здесь  $c_{vc}$  - скорость звука в vena contracta,  $\eta_1$  - акустический к.п.д. клапана,  $c_{vc}$  – потоковый коэффициент клапана,  $F_{l1} = 0,9$  – коэффициент восстановления давления,  $P_0$  – статическое давление перед клапаном.

В переменных  $x_i = \frac{P_i}{P_{i-1}}$ ,  $m_i = 1 - \frac{F_{li}^2}{2}$ ,  $c_{vc} = \left( \frac{kRT_{vc}}{\mu} \right)^{\frac{l}{2}}$ ,  $T_{vc} = T_0 \left( \frac{P_{vc}}{P_0} \right)^{\frac{k-1}{k}}$ ,  $P_{vc} = P_0 \left( 1 - \frac{1-x_1}{F_{l1}^2} \right)$ , где  $k$  – показатель адиабаты,  $P_1$  - статическое давление после клапана, рисунок 2,  $T_0$  – температура газа перед клапаном,  $T_{vc}$  – температура газа в vena contracta,  $R$  – универсальная газовая постоянная,  $\mu$  - молярная масса газа, потоковый коэффициент клапана рассчитывается в соответствии с соотношением:

$$c_{v1} = \begin{cases} 1.95 \cdot 10^7 \frac{G}{F_{l1} P_0}, & x_1 \leq m_1 \\ 2.14 \cdot 10^7 \frac{G}{F_{l1} \sqrt{(1-x_1^2)}}, & x_1 > m_1 \end{cases}.$$

Акустический к.п.д. клапана  $\eta_1$  равен

$$\eta_1 = \begin{cases} 1.32 \cdot 10^{-3} F_{Li}^2 \left( \frac{1.89 m_1}{k^{k-1} x_1} \right), & n \leq x_1 < \frac{1.89 m_1}{3.2 x_1} \\ 10^{-4} F_{Li}^2 \left( \frac{m_1}{x_1} \right)^{3.7}, & \frac{1.89 m_1}{3.2 x_1} < x_1 \leq m_1, \\ 10^{-4} F_{Li}^2 \left( \frac{\frac{P_0}{P_1}}{\frac{P_0}{P_1} (F_L^2 - 1) + 1} \right)^{2.6} & m_1 \leq x_1 < 1 \end{cases} \quad (6)$$

где  $n$  – отношение конечного давления к начальному (здесь  $n = \frac{P_1}{P_0} = x_1$ ),

$G$  – массовый расход газа.

Первая область в соотношении (6) соответствует дозвуковым течениям, вторая – числам Маха от 1 до 1.4, а третья –  $M > 1.4$ .

#### 4.8.3 Мощность шума $i$ -й шайбы, $i = 2..N-1$

По аналогичным формулам можно рассчитать акустическую мощность, генерируемую каждой «внутренней» шайбой:

$$W_i = 7.7 \cdot 10^{-11} c_i^3 \eta_i c_{Vi} F_{Li} P_{i-1}, \quad (7)$$

где

$$c_{Vi} = \begin{cases} 2.14 \cdot 10^7 \frac{G}{P_0 \sqrt{(x_{i-1}^2 - x_i^2)}}, & x_i \leq x_{i-1} < \frac{x_i}{m_i}, \\ 1.95 \cdot 10^7 \frac{G}{F_{Li} P_0 x_{i-1}}, & x_{i-1} > \frac{x_i}{m_i} \end{cases} \quad (8)$$

$$\eta_i = \begin{cases} 10^{-4} F_{Li}^2 \left( \frac{x_{i-1} - x_i}{x_{i-1} F_{Li}^2 - x_{i-1} + x_i} \right)^{2.6}, & x_i \leq x_{i-1} < \frac{x_i}{m_i} \\ 10^{-4} F_{Li}^2 \left( \frac{m_i x_{i-1}}{x_i} \right)^{3.7}, & \frac{x_i}{m_i} < x_{i-1} \leq \frac{3.2 \cdot x_i}{1.89 \cdot m_i} \\ 1.32 \cdot 10^{-3} F_{Li}^2 \left( \frac{1.89}{k^{k-1}} \frac{m_i x_{i-1}}{x_i} \right), & \frac{3.2 \cdot x_i}{1.89 \cdot m_i} \leq x_{i-1} < \frac{22 \cdot x_i}{1.89 \cdot m_i} \\ 1.32 \cdot 10^{-3} F_{Li}^2 \frac{22 \cdot x_{i-1}}{1.89 \cdot m_i}, & x_{i-1} \geq \frac{22 \cdot x_i}{1.89 \cdot m_i} \end{cases} \quad (9)$$

#### 4.8.4 Мощность шума $N$ -й шайбы ( $x_N = n$ )

Для вычисления мощности шума последней ( $N$ -й) шайбы использовались те же соотношения, что и ранее, но с учетом известного значения выходного давления:

$$W_N = 7.7 \cdot 10^{-11} c_N^3 \eta_N c_{VN} F_{LN} P_{N-1}, \quad (10)$$

где

$$c_{VN} = \begin{cases} 2.14 \cdot 10^7 \frac{G}{P_0 \sqrt{(x_{N-1}^2 - n^2)}}, & n \leq x_{N-1} < \frac{n}{m_N}, \\ 1.95 \cdot 10^7 \frac{G}{F_{Li} P_0 x_{N-1}}, & x_{N-1} > \frac{n}{m_N} \end{cases}, \quad (11)$$

$$\eta_N = \begin{cases} 10^{-4} F_{Li}^2 \left( \frac{x_{N-1} - n}{x_{N-1} F_{LN}^2 - x_{N-1} + x_N} \right)^{2.6}, & n \leq x_{N-1} < \frac{n}{m_N} \\ 10^{-4} F_{Li}^2 \left( \frac{m_N x_{N-1}}{n} \right)^{3.7}, & \frac{n}{m_N} < x_{N-1} \leq \frac{3.2}{1.89} \frac{n}{m_N} \\ 1.32 \cdot 10^{-3} F_{LN}^2 \left( \frac{1.89}{k^{k-1}} \frac{m_N x_{N-1}}{n} \right) & \frac{3.2}{1.89} \frac{n}{m_N} \leq x_{N-1} < \frac{22}{1.89} \frac{n}{m_N} \end{cases}, \quad (12)$$

В соотношениях (7) – (12) скорость звука рассчитывалась по формуле  $c_i = \left( \frac{kRT_i}{\mu} \right)^{\frac{1}{2}}$ , коэффициент восстановления давления считался равным единице [43]  $F_{Li} = 1, i = 2..N$ , процесс расширения газа на каждой шайбе считался адиабатическим  $T_i = T_0 x_i^{\frac{k}{k-1}}$ .

#### 4.8.5 Вычисление сечений шайб

Для расчета живых сечений клапана и шайб необходимо воспользоваться выражением для скачка давления, уравнением непрерывности уравнением процесса на каждом элементе.

Для заданных  $x_i$ , определяющих распределение давлений по сегментам трубы можно организовать итерационно, используя следующие соотношения:

$$\left\{ \begin{array}{l} 1 - x_1 = \zeta_1 \Xi, \\ x_1 - x_2 = \zeta_2 \frac{\Xi}{x_1^{1/\gamma}}, \\ \dots \\ x_{i-1} - x_i = \zeta_i \frac{\Xi}{x_{i-1}^{1/\gamma}}, \\ \dots \\ x_{N-2} - x_{N-1} = \zeta_{N-1} \frac{\Xi}{x_{N-2}^{1/\gamma}}, \\ x_{N-1} - n = \zeta_N \frac{\Xi}{x_{N-1}^{1/\gamma}} \end{array} \right. \quad (13)$$

Здесь  $\Xi \approx 0,004$ ;  $F = \frac{\pi D^2}{4}$  - сечение трубы. Выражения в (11) связывают распределение давлений по элементам, коэффициенты гидродинамических сопротивлений  $\zeta_i$  и живые сечения клапана и шайб  $f_i = \frac{F_i}{F}$ . Процессы считаются адиабатными, поэтому для диафрагмы с острыми краями или решетки с острыми краями отверстий коэффициент перепада зависит только от относительного живого сечения шайбы  $f_i$  и равен:

$$\zeta_i = \left[ 1 - f_i + \frac{(1 - f_i)^{3/8}}{\sqrt{2}} \right]^2 \left( \frac{1}{f_i} \right)^2, \quad (14)$$

здесь  $F_i$  - живое сечение шайбы,  $F = \frac{\pi D^2}{4}$  - сечение трубы.

Для определения сечения каждой шайбы необходимо решить нелинейное относительно  $f_i$  уравнение (13) (с учетом зависимости (14)).

Произвольное  $i$ -е уравнение системы (14) запишем в неявной форме:

$$\Phi(f_i) = \zeta_i \frac{\Xi}{x_{i-1}^{1/\gamma}} - x_{i-1} + x_i = \frac{[\frac{(\Xi(\sqrt{2}(1-f_i)^{3/8})}{2} - f_i + 1]^2}{f_i^2 x_{i-1}^{1/\gamma}} - x_{i-1} + x_i = 0. \quad (15)$$

Вычислив производную функции  $\Phi(f_i)$  решение уравнения (15) найдем с помощью метода Ньютона по итерационной формуле:

$$f_i^{(n+1)} = f_i^{(n)} - \frac{\Phi(f_i^{(n)})}{\Phi'(f_i^{(n)})}.$$

Следует отметить, что функция  $\Phi(f_i)$  имеет протяженные пологие участки параллельные оси Ох, поэтому в качестве начального приближения целесообразно использовать небольшие значения  $f_i$  ( $f_i \approx 0,001$ ).

#### 4.9 Постановка задачи глобальной оптимизации для гасителя пульсаций давлений

В общем случае задача выбора рациональных параметров ГПД представляется смешанной задачей параметрической и структурной оптимизации.

Задача структурной оптимизации сводится к определению количества лайб, необходимых для обеспечения требуемого уровня акустической мощности ГПД, и реализуется простым перебором различных вариантов компоновок ГПД по числу шайб. Возможны и более тонкие исследования предполагающие использование шайб разного типа.

Задачу параметрической оптимизации можно поставить как задачу условной оптимизации:

$$W(x_1, x_2, \dots, x_N) \rightarrow \min_{x_1, \dots, x_N} \quad (16)$$

$$\begin{cases} x_i \in [0; 1], \quad i = 1, \dots, N \\ x_1 > x_2, \\ \dots \\ x_{N-1} > x_N. \end{cases} \quad (17)$$

Ограничения (17) возникают из физических соображений и определяют достаточно сложную допустимую область задачи оптимизации. Кроме того, задача нелинейного программирования (16)-(17) потребует для своего решения использования довольно сложной техники методов штрафных функций, множителей Лагранжа или метода Куна-Таккера. Однако, с помощью замены переменных задачу условной оптимизации (16)-(17) можно свести к задаче

безусловной оптимизации на стандартном (для многих методов ГО) допустимом множестве  $D_0 = \bigotimes_{i=1}^n [0, 1]$ .

Введем новые переменные, таким образом, что:

$$\begin{cases} y_i \in [0; 1], & i = 1, \dots, N \\ x_1 = y_1, \\ x_2 = y_1 \cdot y_2, \\ \dots \\ x_N = y_1 \cdot y_2 \cdot \dots \cdot y_N. \end{cases}, \quad (18)$$

а также обратное преобразование

$$\begin{cases} y_1 = x_1, \\ y_2 = x_2 / x_1, \\ \dots \\ y_N = x_N / x_2 \cdot \dots \cdot x_{N-1}. \end{cases} \quad (19)$$

В этом случае задачу ГО можно поставить как задачу безусловной оптимизации на единичном гиперкубе  $D_0 = \bigotimes_{i=1}^n [0, 1]$ :

$$W(y_1, y_2, \dots, y_N) \rightarrow \min_{y \in D_0} . \quad (20)$$

В процессе поиска рациональных параметров ГПД использовался предложенный двухфазный алгоритм глобальной оптимизации. Алгоритм позволил уверенно найти оптимальные сочетания оптимизируемых переменных и, соответственно, проходные сечения шайб.

#### 4.10 Результаты вычислительных экспериментов

Исходные данные вычислительных экспериментов приведены в таблице 8.

Предварительно было произведено исследование свойств оптимизируемой функции. Важно было определить, какой характер имеет функция  $W(x_1, x_2, \dots, x_N)$  – многоэкстремальный или унимодальный? Для

нимодальных функций можно обойтись достаточно быстродействующими локальными методами оптимизации.

Таблица 8 – Исходные данные численного расчета ГПД

Массовый расход газа $G, \text{кг/с}$	0,043
Диаметр трубы $D, \text{м}$	0,040
Плотность газа в сети потребителя $\rho_{out}, \text{кг\cdot м/с}$	1,29
Давление в сети потребителя $P_{out}, \text{Па.}$	$2 \cdot 10^5$
Показатель адиабаты $k$	1,4
Молярная масса $\mu, \text{кг/моль}$	0,029
Отношение конечного давления к начальному, $n$	1/4

На рисунке 21, в координатах  $x_1, x_2$  показан общий вид оптимизируемой функции  $W(x_1, x_2)$ . Из рисунка видно, что функция имеет не один локальный минимум.

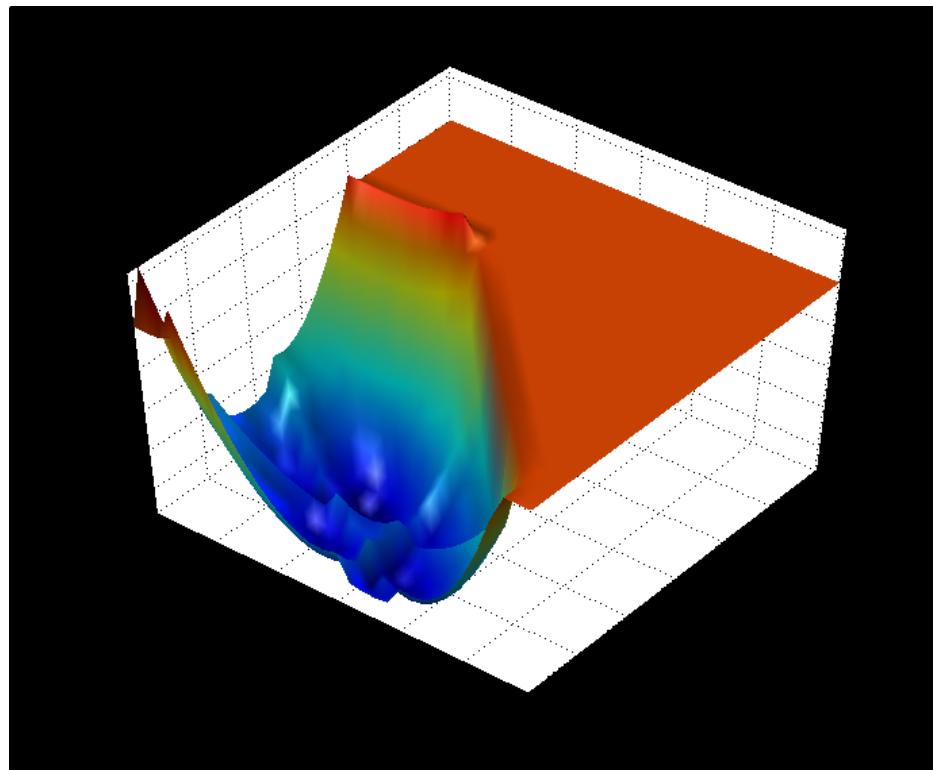


Рисунок 21 – Вид целевой функции для 2-х шайб

Эксперименты проводились на суперкомпьютерном кластере СГАУ «Сергей Королев».

Расчеты проводились с числом шайб от двух до семи. В таблице 9 представлены результаты работы алгоритма ГО. На рисунке 22 приведена гистограмма с распределением количества выполненных обращений к функции на каждом процессоре, из которой видно, что загрузка процессоров неравномерна. Это обусловлено особой формой области допустимых значений  $X$ , где функция  $W$  имеет физический смысл и, опять же, прореживанием бесперспективных областей, что снижает общее ускорение от распараллеливания. Более того, в ходе экспериментов подтвердилось, что топология функции  $W(x_1, x_2, \dots, x_N)$  является многоэкстремальной и зависит от количества шайб: для двух шайб имеется 2 минимума, для пяти – восемь, для семи – четыре.

Таблица 9 – Результаты работы алгоритма глобальной оптимизации

Число шайб	Кол-во выч. функции	Время работы алг., сек	Кол-во лок. минимумов	Кол-во исп. проц.	Ускорение
2	156	0,095	2	16	6,23
3	404	0,278	4	32	8,92
4	1088	0,500	2	64	26,69
5	12769	1,410	8	128	34,93
6	44684	8,153	4	256	67,13
7	357191	208,560	4	512	128,61

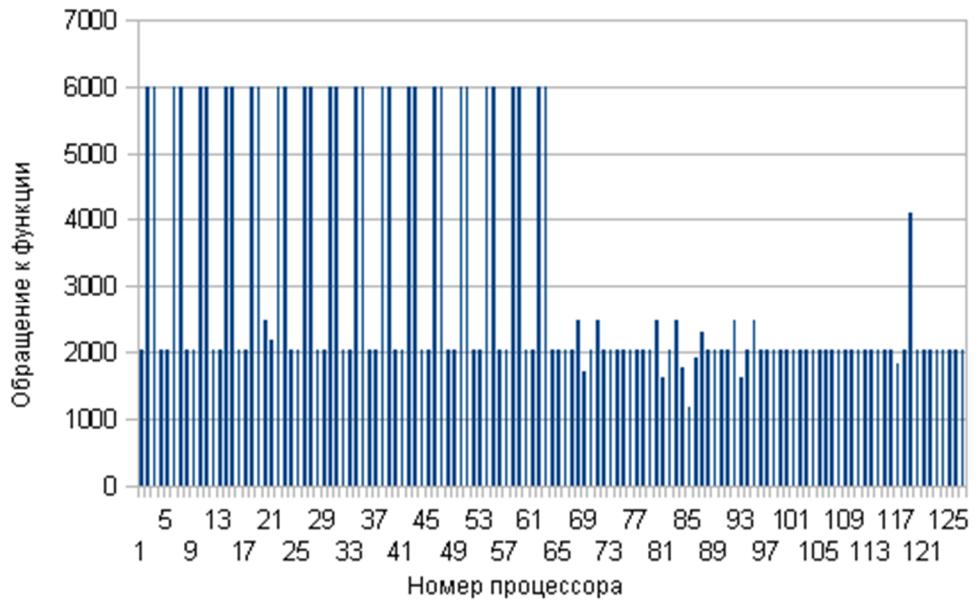


Рисунок 22 – Распределение загрузки по процессорам

В таблице 10 представлены результаты ГО ГПД с числом шайб от двух до семи.

Таблица 10 – Оптимальные параметры ГПД

x1	x2	x3	x4	x5	x6	x7	W, Вт	Lw, Дб
0,7089	0,4218						0,3358	115,2
0,7177	0,5125	0,3607					0,2124	113,2
0,7693	0,5898	0,4478	0,3364				0,1402	111,4
0,8053	0,6469	0,5160	0,4085	0,3209			0,1028	110,1
0,8318	0,6905	0,5702	0,4683	0,3823	0,3101		0,0803	109,0
0,852	0,7248	0,6140	0,5179	0,4349	0,3634	0,3022	0,0654	108,1

Оптимальные значения уровней акустической мощности (Вт) в ГПД в зависимости от числа шайб изменяются так, как это показано на рисунке 23. Из рисунка видно, что график монотонно убывает, и при количестве шайб больше семи наблюдается явно выраженный пологий участок. В таблице 11 приведены соответствующие оптимальные значения сечений клапана и шайб.

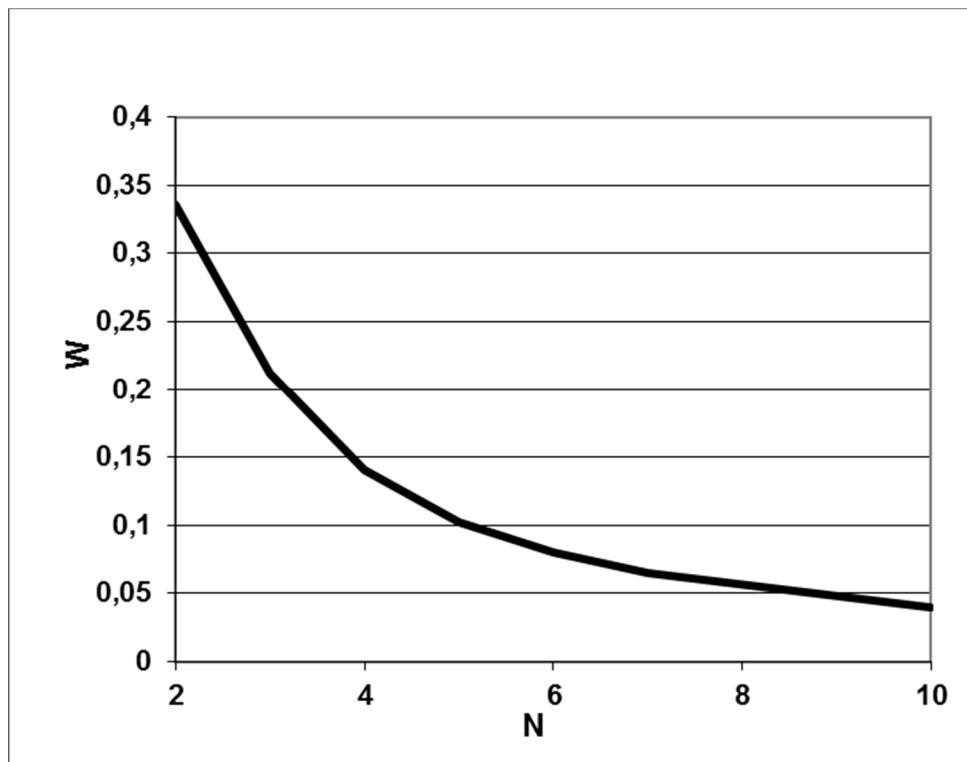


Рисунок 23 – Зависимость оптимального значения W от количества шайб

Таблица 11 – Относительные сечения клапана и шайб в процентах от сечения трубы

Fkl	F1	F2	F3	F4	F5	F6	F7
11,311	18,802	27,015					
11,248	21,573	26,855	33,293				
10,896	22,337	26,498	31,379	37,025			
10,670	23,200	26,662	30,614	35,084	40,083		
10,513	24,080	27,062	30,398	34,107	38,198	42,667	
10,398	24,945	27,576	30,474	33,652	37,117	40,870	44,901

Данные таблицы 11 в графической форме представлены на рисунке 24.

Из рисунка видно, что, не считая переходного участка от клапана к первой шайбе, оптимальные походные сечения шайб линейно увеличиваются от сечения к сечению. Причем угол наклона линейного участка уменьшается с увеличением числа шайб.

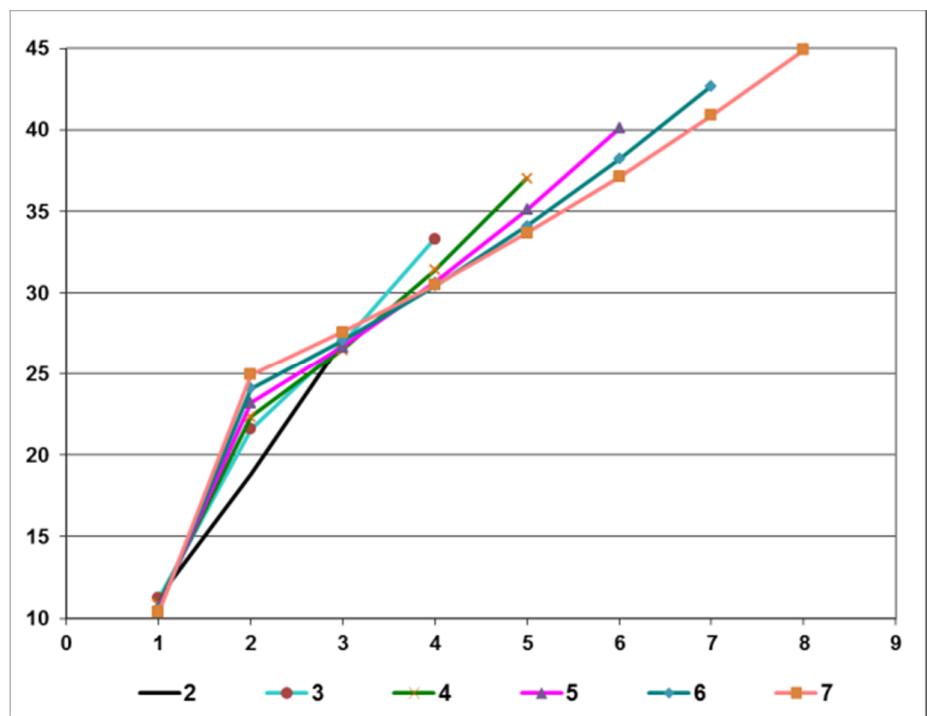


Рисунок 24 – Относительные сечения клапана и шайб в процентах от сечения трубы

## ЗАКЛЮЧЕНИЕ

В данной работе проведено общие обзор задачи глобальной оптимизации многоэкстремальных функций многих переменных. Показана актуальность данной задачи. Приведен краткий обзор графических способов описания параллельных вычислений.

Разработан двухфазный параллельный алгоритм глобальной оптимизации модифицированным методом половинных делений. Идея алгоритма заключается в сочетании техник локальной и глобальной оптимизации. Данный алгоритм относится к точным методам глобальной оптимизации – это означает, что можно с некоторой вероятностью гарантировать нахождение глобального оптимума. В фазе глобальной оптимизации формируется неравномерное покрытие многомерными параллелепипедами допустимого множества и список начальных приближений локальных минимумов. На каждом этапе деления осуществляется отсев бесперспективных областей по критерию Липшица с уточнением константы Липшица и рекордного значений. Этап глобальной оптимизации завершается при достижении заданного минимального размера параллелепипеда. На фазе локальной оптимизации из каждой точки списка начальных приближений на отдельном процессоре запускается метод деформированных многогранников, осуществляющий локальный спуск минимуму.

Реализован программный комплекс, позволяющий создавать с помощью визуальных средств модели параллельных алгоритмов и на их основе в автоматическом режиме генерировать коды программ на C++ с учетом стандарта MPI. С помощью данного комплекса реализован двухфазный параллельный алгоритм глобальной оптимизации модифицированным методом половинного деления. Предложено несколько модификаций алгоритма и показана высокая значимость средств визуального программирования в разработке новых параллельных алгоритмов. Данное средство разработки параллельных программ включено в учебный процесс и применяется при

проведении лабораторных работ по курсу «Методы и средства визуального программирования».

Проведено исследование алгоритма глобальной оптимизации на тестовой функции GKLS. Данная тестовая функция обладает большим набором параметров и позволяет генерировать различные варианты липшицевых функций в широком диапазоне. Эксперименты показали, что предложенный алгоритм успешно справляется с задачей поиска глобального минимума даже на наиболее трудном классе недифференцируемых функций. Алгоритм позволяет найти глобальный оптимум для задачи с размерностью 15, по крайней мере, для тестовой функции GKLS. Используемый в фазе глобальной оптимизации метод прореживания бесперспективных областей, основанный на информационно-статистический подход Р.Г. Стронгина, позволяет сократить число вычислений функции на порядки, по сравнению со случайным поиском или полным перебором.

С помощью параллельного алгоритма глобальной оптимизации модифицированным методом половинных делений была решена задача выбора оптимальных параметров гасителя пульсаций давлений. Таким образом, было показано применимость данного алгоритма для решения реальных технических задач оптимизации.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Немировский А. С., Юдин Б. Д. Сложность задач и эффективность методов оптимизации. — М. : Наука, 1979. — 384 с.
2. Коварцев А.Н., Попова-Коварцева Д.А. К вопросу об эффективности параллельных алгоритмов глобальной оптимизации функций многих переменных // Компьютерная оптика. — 2011. — Т. 35, №2. — С. 256-262.
3. Floudas C. A., Gounaris C. E. A review of recent advances in global optimization // Journal of Global Optimization. — Springer Netherlands, 2008. — Vol. 45, №1. — P. 3-38.
4. Horst R., Pardalos P. M. Handbook of Global Optimization. — Kluwer : Dordrecht, 1995. — 569 p.
5. Жиглявский А. А., Жилинскас А. Г. Методы поиска глобального экстремума. — М. : Наука, 1991. — 248 с.
6. Amdahl G. The Validity of Single Processor Approach to Achieving Large Scale Computing Capabilities // AFIPS Conf. — New York : AFIPS Press, 1967. — Vol. 30. — P. 483-485.
7. Евтушенко Ю. Г. Численный метод поиска глобального экстремума функций (перебор на неравномерной сетке) // ЖВМ и МФ. — 1971. — Т. 11, №6. — С. 1390-1403.
8. Ali M., Törn A., Viitanen S. Stochastic Global Optimization: Problem, Classes and Solution Techniques // Journal of Global Optimization. — 1999. — Vol. 14, №4. — P. 437-447.
9. Evtushenko Yu. G., Potapov M. A., Korotkikh V. V. Recent Advances in Global Optimization // Princeton University Press. — 1992. — №1. — P. 274-297.
10. Huyer W., Neumaier A. Global Optimization by Multilevel Coordinate Search // Journal of Global Optimization. — Springer, 1999. — Vol. 14, №1995. — P. 331–355.
11. Pintér J.D. Global Optimization in Action. — London : Kluwer, 1996. — 512 p.

12. Törn A., Zilinskas A. Global Optimization. — Berlin : Springer Verlag, 1989. — 255 p.
13. Васильев Ф.П. Численные методы решения экстремальных задач. — М. : Наука, 1980. — 552 с.
14. Завриев С.К. Перунова Ю.Н. Параллельные версии модифицированных методов покоординатного и градиентного спуска и их применение для решения некоторого класса задач глобальной оптимизации // Прикладная математика и информатика. — М. : Диалог- МГУ, 2002. — №10.
15. Поляк Б.Т. Введение в оптимизацию. — М. : Наука, 1983. — 384 с.
16. Сухарев А.Г. Глобальный экстремум и методы его отыскания // Математические методы в исследовании операций. — М. : МГУ, 1981.
17. Rinnooy Kan A.H.G. Timmer G.T. Stochastic global optimization methods part I: Clustering methods // Mathematical Programming. — Springer. — Vol. 39, №1. — P. 27–56.
18. Törn A. Global Optimization as a Combination of Global and Local Search. — Handelshögskolan vid Åbo akadem, 1974. — 65 p.
19. Hansen E.R. Global Optimization Using Interval Analysis. — New York : Marcel Dekker, 1992. — 515 p.
20. Pruul E.A., Nemhauser G. L., R.A. Rushmeier Branch-and-bound and parallel computation: A historical note // Operations Research Letters. — Elsevier, 1988. — Vol. 7, №2. — P. 65–69.
21. Michalewicz Z. Genetic Algorithms + Data Structures = Evolution Programs. — Springer-Verlag, 1996. — 387 p.
22. Гергель А. В. Многомерная многоэкстремальная оптимизация на основе адаптивной многошаговой редукции размерности // Вестник ННГУ. Математическое моделирование и оптимальное управление. — Нижний Новгород : Нижегородского гос. ун-та, 2010. — №1. — С. 163–170.
23. Стронгин Р.Г. Поиск глобального оптимума. — М. : Знание, 1990. — 240 с.

24. Коварцев А.Н. Автоматизация разработки и тестирования программных средств. — Самара : Самар. гос. аэрокосм. ун-т., 1999. — 150 с.
25. Котов В.Е. Сети Петри. — М. : Наука, 1984. — 160 с.
26. Шалыто А.А. SWITCH-технология. Алгоритмизация и программирование задач логического управления. — Спб. : Наука, 1998. — 628 с.
27. Harel D. Statecharts: A Visual Formalism for Complex Systems // Science of Computer Programming. — Elsevier, 1987. — №8. — P. 231–274.
28. Browne J.C., Azam M., Sobek S. CODE: a unified approach to parallel programming // IEEE Software. — IEEE, 1989. — Vol. 6, №4. — P. 10-18.
29. Babaoglu O. Paralex: An Environment for Parallel Programming in Distributed Systems // Proceedings of the 6th international conference on Supercomputing. — 1992. — Vol. October. — P. 178–187.
30. Beguelin A., Dongarra J. J., Geist G. A., Manchek R., Sunderam V. S. Graphical development tools for network-based concurrent supercomputing // Proceedings of Supercomputing 91. — Albuquerque, 1991. — №1. — P. 435-444.
31. Иванов Б.Н. Дискретная математика. Алгоритмы и программ. — М. : Лаборатория базовых знаний, 2003. — 288 с.
32. Т-Платформы Система пакетной обработки заданий torque. Руководство пользователя. — 2008.
33. Аткинсон Л. MySQL. Библиотека профессионала. — М. : Вильямс, 2002. — 624 с.
34. Евтушенко Ю.Г., Посыпкин М.А. Параллельные методы решения задач глобальной оптимизации // Труды четвертой международной конференции «Параллельные вычисления и задачи управления». — М., 2008. — С. 18-39.
35. Евтушенко Ю. Г. Численный метод поиска глобального экстремума функций (перебор на неравномерной сетке) // ЖВМ и МФ. — 1971. — Т. 11, №6. — С. 1390-1403.

36. Евтушенко Ю.Г., Ратькин В.А. Метод половинного деления для глобальной оптимизации функции многих переменных // Техническая кибернетика. — 1987. — №1. — С. 119-127.
37. Диго Г.Б., Диго Н.Б. Анализ эффективности поиска глобального экстремума алгоритмически заданной функции на основе методов половинных делений и перебора на неравномерной сетке // Труды VII Международной конференции «Идентификация систем и задачи управления» SICPRO. — М., 2008. — С. 512-525.
38. Кvasov D.E., Сергеев Я.Д. Многомерный алгоритм глобальной оптимизации на основе адаптивных диагональных кривых // ЖВМ и МФ. — 2003. — Т. 43, №1. — С. 42-59.
39. Баркалов К.А., Рябов В.В., Сидоров С.В. О некоторых способах балансировки локального и глобального поиска в параллельных алгоритмах глобальной оптимизации // Вычислительные методы и программирование. — 2010. — Т. 11, №2. — С. 189-194.
40. Орлянская И.В. Современные подходы к построению методов глобальной оптимизации // Электронный журнал «Исследовано в России». — 2007. — №1. — С. 189-192.
41. Nelder J.A., Mead R. A simplex method for function minimization // Computer Journal. — 1965. — Vol. 7. — P. 308—313.
42. Gaviano M., Kvasov D. E., Lera D., Sergeyev Ya. D. Softwere for generation of classes of test of functions with known local and global minima for global optimization // ACM TOMS. — 2009. — T. 29, №4. — C. 469-480.
43. Singh G. M., Rodarte E., Miller N. R., S. Hrnjak P. Modification of a Standard Aero-acoustic Valve Noise Model to Account for Friction and Two-Phase Flow // ACRC TR-162. — University of Illinois, 2000. — P. 13.
44. Ver I.L., Beranek L.L. Noise and Vibration Control Engineering: Principles and Applications. — Wiley, 2005. — 976 p.