МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ имени академика С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»

Моделирование параллельных вычислений при глобальной оптимизации на основе языка визуального программирования PGRAPH

Электронные методические указания

С А М А Р А

2011

УДК СГАУ: 621.9.02

Составители: **Коварцев Александр Николаевич,**
Жидченко Виктор Викторович

Рецензент: д-р техн. наук, проф. В.С. Кузьмичев

Моделирование параллельных вычислений при глобальной оптимизации на основе языка визуального программирования PGRAPH [Электронный ресурс] : электрон. метод. указания / Минобрнауки России, Самар. гос. аэрокосм. ун-т. С.П. Королева (нац. исслед. ун-т); сост. А.Н. Коварцев, В.В. Жидченко. - Электрон. текстовые и граф. дан. (1,56 Мбайт). - Самара, 2011. - 1 эл. опт. диск (CD-ROM). - Систем. требования: ПК Pentium; Windows 98 или выше.

В методических указаниях к курсовой работе представлены: краткое изложение теоретических сведений, необходимых для понимания проблем глобальной оптимизации функций многих переменных; технология моделирования параллельных вычислений в системе PGRAPH; методика оценки сложности задачи глобальной оптимизации; методы анализа эффективности параллельных алгоритмов, связанных с оптимизацией сложных технических систем; учебный пример моделирования параллельных вычислений при решении задачи глобальной оптимизации сложной системы.

Методические указания ориентированы на студентов второго факультета, дневного отделения, занимающихся проблемой оптимизации проектных параметров газотурбинных двигателей. Указания могут быть полезны студентам шестого факультета при изучении методов глобальной оптимизации.

Содержание

Введение

Научно-техническая революция, которая в полной мере оформилась в 50 годах прошлого века в связи с появлением вычислительной техники, породила такое научное направление как системный анализ. Академик Н. Н. Моисеев определяет системный анализ как совокупность методов, основанных на использовании ЭВМ и ориентированных на исследование сложных систем – технических, экономических, экологических, социальных и т.д. Результатом системных исследований является выбор вполне определенной альтернативы: плана развития региона, параметров конструкции и т.д.

Методологическую основу системного анализа составляют: построение адекватных математических моделей объекта исследования и применение *методов оптимизации*. В 70 годах принципы системного анализа были настолько популярны в научном сообществе, что казалось: достижение основной цели – гармонического развития общества - «не за горами».

Действительно, в те времена были разработаны революционные методы линейного программирования, сформулирован принцип максимума Понтрягина, развита теория локальных экстремумов. Более того, была доказана полиномиальная сложность алгоритмов линейного программирования и метода Ньютона для некоторого класса задач оптимизации. В тоже время, каждый раз развитие методов системного анализа «спотыкалось» на несовершенстве методов оптимизации. Например, было установлено, что многие практические задачи имеют многокритериальную постановку в условиях неопределенности исходных данных. Целевая функция может иметь многоэкстремальный характер и т.д.

Развитие вычислительной техники и ее философской основы - теории алгоритмов, установило, что в, общем случае, методы оптимизации относятся к классу труднорешаемых задач, имеющих экспоненциальную

сложность (NP-полнота задачи). 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, подобные 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.

Методы разработки параллельных алгоритмов коренным образом отличаются от традиционных методов создания последовательных кодов программ. Учитывая широкую распространенность вычислительных систем с распределенной памятью, для организации информационного взаимодействия между процессорами часто используется интерфейс передачи сообщений MPI. Необходимость равнозначного владения одновременно технологиями программирования на языках высокого уровня (C++, C#) и средством организации параллельных вычислений MPI еще дальше отдаляет «конечных» пользователей от возможности участия в разработке параллельных программных приложений. Существует множество графических моделей описания алгоритмов

параллельных вычислений. В области представления вычислительных алгоритмов наиболее удобной является форма, близкая описанию блок-схемами, которая реализована в параллельной версии технологии графосимволического программирования (ГСП) в виде системы PGRAPH, описание которой в краткой форме приводятся в данных методических указаниях. Технология графосимволического программирования GRAPH, созданная на кафедре ПС СГАУ, является одним из способов наглядного представления алгоритмов программ в виде графа управления.

Применение системы PGRAPH рассматривается на примере решения задачи глобальной оптимизации целевых функций большой размерности, часто возникающей при оптимизации проточной части ГТД.

В указаниях представлены: краткое описание современных методов глобальной оптимизации; технология моделирования параллельных вычислений; учебный пример моделирования параллельных вычислений при решении задачи глобальной оптимизации сложной системы, имеющей большое количество варьируемых параметров; методика оценки сложности задачи глобальной оптимизации; методы анализа эффективности параллельных алгоритмов, связанных с оптимизацией сложных технических систем.

Курсовая работа ориентирована на самостоятельное изучение студентами методов глобальной оптимизации функций многих переменных и технологий разработки моделей параллельных алгоритмов, связанных с решением сложных задач.

По результатам работы оформляется пояснительная записка, которая сдается преподавателю на проверку.

Пояснительная записка составляется в соответствии со стандартами оформления научно-технической документации.

Записка должна содержать:

1. Титульный лист с реквизитами учреждения, наименования предмета, номера группы, ФИО исполнителя и т.д.
2. Полное описание постановки задачи.
3. Описание выбранного метода решения глобальной задачи оптимизации.
4. Исследование эффективности параллельного алгоритма глобальной оптимизации.
5. Выводы по результатам проделанной работы.

1. Методы глобальной оптимизации

1.1. Проблема глобальной оптимизации

Большое количество постановок технических или научных проблем можно сформулировать как задачу глобальной оптимизации. В связи с чем, алгоритмы глобальной оптимизации находят широкое применение в разнообразных областях науки и техники, везде, где необходимо получить наилучший результат целевой функции, оценивающей качество принимаемого решения. Не является исключением, и задача проектирования проточной части ГТД, при решении которой, начиная с 70 годов XX века, применяется системный подход, использующий, в том числе и методы оптимизации целевой функции.

В этой области в начале XXI века сложились, с одной стороны благоприятные условия, связанные с появлением высокопроизводительных вычислительных систем и соответствующего программного обеспечения, специализирующегося на решении разнообразных задач проектирования ГТД. С другой стороны, по мере детализации модели ГТД или его узлов [5] растут размерности задач оптимизации, и усложняется топология целевых функций. Так, например, для проточной части ГТД число оптимизируемых параметров может составить сотни переменных, а целевая функция, в общем случае, может иметь множество экстремумов. Таким образом, и в этой области возникает задача глобальной оптимизации функции многих переменных.

К настоящему времени разработано большое количество алгоритмов и методов решения задачи многоэкстремальной оптимизации. Сегодня разработка методов глобальной оптимизации стимулируется развитием электронно-вычислительных средств и во многом связано с доступностью параллельных компьютерных систем высокой производительности. Появилось большое количество разнообразных подходов к распараллеливанию алгоритмов глобальной оптимизации.

Специфика задачи глобальной оптимизации заключается в многоэкстремальности функции цели и её неразрешимости в общем случае.

Для одномерных функций существуют эффективные стратегии многоэкстремальной оптимизации. Лидером методов одномерной глобальной оптимизации является информационный алгоритм Р. Г. Стронгина [8] и его модификации, основанный на использовании приближенного апостериорного распределения вероятностей расположения глобального экстремума, формируемого в процессе испытаний функции. Но в области многомерных функций он не имеет аналогов. Разработка эффективных методов многомерной глобальной оптимизации ведется в следующих направлениях (хороший обзор методов глобальной оптимизации можно найти в работе [7]):

1. Редукция многомерной задачи оптимизации.
2. Использование локальной техники.
3. Аппроксимативные методы.
4. Эвристические алгоритмы.

1.2. Формальная постановка задачи глобальной оптимизации

В данной работе ограничимся простой задачей безусловной глобальной оптимизации. Без ограничения общности, задачу глобальной оптимизации рассмотрим в следующей постановке:

$$f(x) \to \min_{x \in D_0}, \tag{1.1}$$

где $D_0 = \bigotimes_{i=1}^{n} [0, 1]$ - допустимое множество векторов оптимизируемых переменных задачи оптимизации (многомерный единичный гиперкуб), $f(\cdot)$ - многоэкстремальная функция цели. Обычно $f(x)$ является гладкой невыпуклой функцией, обладающей определенными свойствами. Следуя традициям, положим, что оптимизируемая функция

удовлетворяет условию Липшица с константой L, т.е. выполняется неравенство

$$\left| f(x) - f(z) \right| \le L \left\| x - z \right\|_\infty . \qquad (1.2)$$

Здесь и далее $\left\| x \right\|_\infty = \max\limits_{1 \le i \le n} \left| x_i \right|$. Условие Липшица (1.2) фактически ограничивает рост функции, и без этого условия невозможно разрабатывать сколь-нибудь эффективные стратегии оптимизации.

Основной проблемой задачи глобальной оптимизации (1.1) является многоэкстремальность целевой функции. На рисунке 1.1 приведены примеры одномерной и двухмерной многоэкстремальных функций.

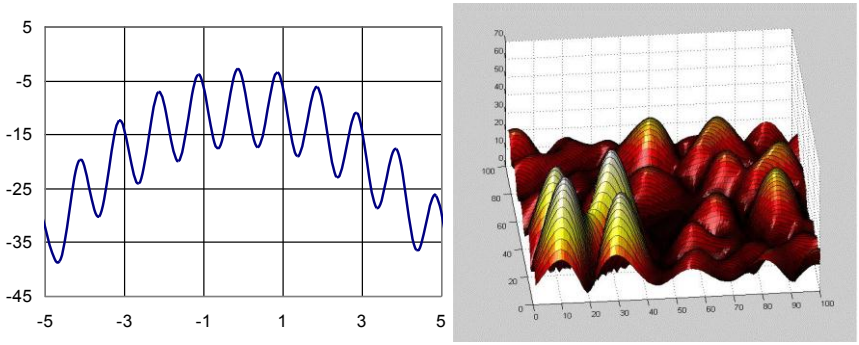

Рис. 1.1. Примеры многоэкстремальных функций

Очевидно, что в этих условиях непросто среди множества локальных экстремумов найти единственный глобальный экстремум.

Второй проблемой глобальной оптимизации является число оптимизируемых переменных. Чем больше переменных, тем сложнее найти глобальный экстремум.

1.3. Сложность задач глобальной оптимизации

В области решения многоэкстремальной задачи оптимизации существенные результаты получены главным образом для функций одной переменной усилиями отечественной научной школы Р. Г. Стронгина [8]. Что же касается многомерных функций, то результаты выглядят более скромно. Причина этого явления понятна, поскольку, как известно, задача глобальной оптимизации функций многих переменных, в общем случае, является неразрешимой, т.е. относится к классу NP-полных задач [6].

Еще в 70 годы прошлого века для многоэкстремальной задачи условной оптимизации k раз непрерывно дифференцируемой функции n переменных была получена нижняя оценка стохастической сложности [2]:

$$\tilde{N}(\varepsilon) \geq c(n,k)\left(\frac{1}{\varepsilon}\right)^{n/k}, \qquad (1.3)$$

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

Полученная нижняя информационная оценка (1.3) показывает, что при заданной точности увеличение размерности задачи оптимизации приводит к катастрофическому росту ее сложности. Реальная вычислительная сложность задачи глобальной оптимизации обычно значительно выше. Конечно, на практике используют методы, не слишком задумываясь о том, какие гарантии они способны доставить, и удовлетворяются полученными результатами, которые кажутся приемлемыми, во всяком случае, значительно лучше, чем приближенные решения. Отсюда Интернет предоставляет большое количество сайтов декларирующих «легкое» решение задач глобальной оптимизации для функций многих переменных. Подобная, ничем не обоснованная, подмена теоретически обоснованных методов эмпирическими

подходами таит определенную опасность, поскольку, если не рассматривать специфические классы оптимизационных задач, то эмпирические методы, либо не находят глобальный экстремум, либо делают это очень грубо.

В работе [4] для задачи безусловной глобальной оптимизации гладкой липшицевой функции была получена более оптимистическая оценка сложности:

$$N(n) = 2 \cdot \left(\frac{1}{4\varepsilon} \right)^n - 1 . \qquad (1.4)$$

Тем не менее, для решения задачи глобальной оптимизации функции 100 переменных для гладких липшицевых функций потребуется приблизительно $N(100) \approx 7 \cdot 10^{150}$ обращений к оптимизируемой функции. Из выражения (1.3 и 1.4) ясно, что при заданной точности увеличение размерности задачи оптимизации приводит к катастрофическому росту ее сложности. В связи с чем, непонятно - за счет каких скрытых ресурсов, используя кластерные системы, авторы рассчитывают преодолеть экспоненциальный рост сложности задачи глобальной оптимизации, когда количество процессоров в кластере можно увеличить в лучшем случае линейно. Специфика задачи глобальной оптимизации заключается в многоэкстремальности функции цели и её неразрешимости в общем случае.

Проблему сложности алгоритмов оптимизации проинтерпретируем на простых примерах.

Рассмотрим проблему численного решения задач *математического программирования*:

$$f_0(X) \to \min_x \quad \Big| \; X \in G, \quad f_j(X) \le 0, \quad j = 1,..., m, \qquad (1.5)$$

где $X = (x_1, x_2, ..., x_n)'$ - оптимизируемые переменные, G – замкнутое подмножество евклидова пространства $G \subset R^n$, $f_0(X)$ - оптимизируемая функция, $f_i(X)$, $i = 1, ..., m$ - система ограничений.

На рисунке 1.2 образно представлена задача математического (в данном случае выпуклого) программирования. Тонкими линиями представлены изолинии (линии равного уровня) оптимизируемой функции $f_0(X)$ в пространстве двух переменных $X = (x_1, x_2)'$. Четыре ограничения представлены жирными линиями. Задача оптимизации ставится как задача поиска наименьшего (минимального) значения оптимизируемой функции, при условии выполнения всех заданных ограничений (*задача условной оптимизации*).

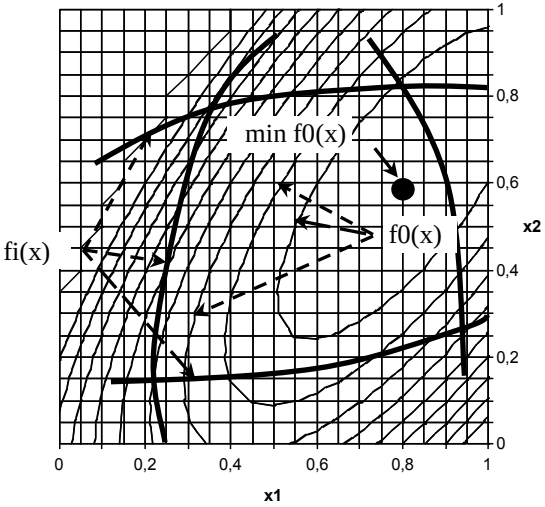

Рис. 1.2. Задача математического программирования

Эффективность алгоритмов оптимизации зависит от множества факторов. В первую очередь от свойств оптимизируемой функции и ограничений. Топологии пространства поиска, размерности вектора оптимизируемых параметров и т.д.

Для дискретной функции $y_k = f(X^k)$, заданной набором значений функции $Y = (y_1, y_2, ..., y_N)$ задача оптимизации решается достаточно просто. Для этого можно использовать любой из алгоритмов переборного поиска. При этом если множество $Y = (y_1, y_2, ..., y_N)$ упорядоченно, то решение находится за 1 шаг (эффективность алгоритма равна $O(1)$) и не зависит от размерности вектора независимых переменных. Иначе минимальное значение находится в худшем случае за N шагов (эффективность $O(N)$).

Значительно хуже обстоят дела для непрерывных функций. В первую очередь, в этом случае необходимо ввести понятие точности решения задачи оптимизации, поскольку *численными методами* (на ЭВМ) как правило, невозможно найти точное решение.

Обычно вводят некоторую меру оценки погрешности решения поставленной задачи. Например, если X^* - точное решение задачи математического программирования, а $\tilde{X}$ - приближенное решение, полученное тем или иным алгоритмом, то оценку точности решения можно вычислить по формулам

$$\varepsilon = \sum_{i=1}^{n} \left| x_i^* - \tilde{x}_i \right| \text{ - для оценки абсолютной погрешности,}$$

$$\varepsilon = \sum_{i=1}^{n} \left| x_i^* - \tilde{x}_i \right| / \left| x_i^* \right| \text{ - для оценки относительной погрешности.} \quad (1.6)$$

Численные методы (алгоритмы) оптимизации формируют некоторую последовательность (траекторию) $X_1, X_2, ..., X_N$ точек области G такаю,

что последняя точка $\tilde{X} = X_N$ лежит в окрестностях решения задачи математического программирования X^*.

Определение 1.1. *Траекторией алгоритма оптимизации F называется всякая последовательность*

$$X^{\infty} = \{X_1, X_2, \ldots\}, \qquad X_i \in G .$$

Формально детерминированные алгоритмы оптимизации можно задать с помощью рекуррентного правила

$$X^{k+1} = F(X^k), \tag{1.7}$$

где $F(\bullet)$ - итерационная функция.

Определение 1.2. *Трудоемкостью траектории (трудоемкостью алгоритма оптимизации) будем называть число* $l(X^{\infty})$ - *членов последовательности обеспечивающих попадание последней точки траектории в* ε - *окрестность решения задачи математического программирования.*

В методах оптимизации эффективность алгоритма традиционно оценивается по числу обращений к оптимизируемой функции. К особенности рассматриваемого класса задач можно отнести *приближенный характер формируемого решения*.

Часто невозможно найти точное решение поставленной задачи. Одним из общих подходов к решению NP-трудных задач, бурно развивающийся в настоящее время, является разработка приближенных алгоритмов с гарантированными оценками качества получаемого решения.

Точность приближенного алгоритма характеризует мультипликативная ошибка, которая показывает, в какое максимально

возможное число, раз полученное решение отличается от оптимального (по значению заданной целевой функции).

Определение 1.3. *Алгоритм называется C-приближенным, если при любых исходных данных он находит допустимое решение со значением целевой функции, отличающимся от оптимума не более чем в C раз.*

Заметим, что иногда, также говорят об C-приближенных алгоритмах, причем смысл отклонения (больше или меньше единицы) обычно ясен из контекста и направления оптимизации (максимизации или минимизации). Мультипликативная ошибка может быть константой или зависеть от параметров исходной задачи. Наиболее удачные приближенные алгоритмы позволяют задавать точность своей работы.

Таким образом, эффективность алгоритмов оптимизации зависит:

- от свойств оптимизируемой функции;
- размерности и топологии пространства независимых переменных;
- точности решения задачи оптимизации.

1.3.1. Унимодальные, непрерывные одномерные функции

Пусть при n=1 и $x \in [0;1]$ оптимизируемая функция является линейной функцией $y = ax + b$. В этом случае минимум или максимум функции находится на одном из концов отрезка [0; 1]. Для нахождения минимума функции достаточно вычислить два значения $f(0)$, $f(1)$ и выбрать минимальное. Сложность алгоритма минимальна $O(1)$. Решение находится точно.

Пусть для одномерной задачи оптимизации ($x \in [0;1]$) задана унимодальная функция (например, функция имеющая минимум см. рис. 1.3).

Известно множество алгоритмов оптимизации унимодальных функций. Если задана непрерывная функция, то можно применить самый простой алгоритм двоичного деления *отрезка неопределенности*. Под

отрезком неопределенности будем понимать отрезок области поиска, к которому, безусловно, принадлежит минимум функции.

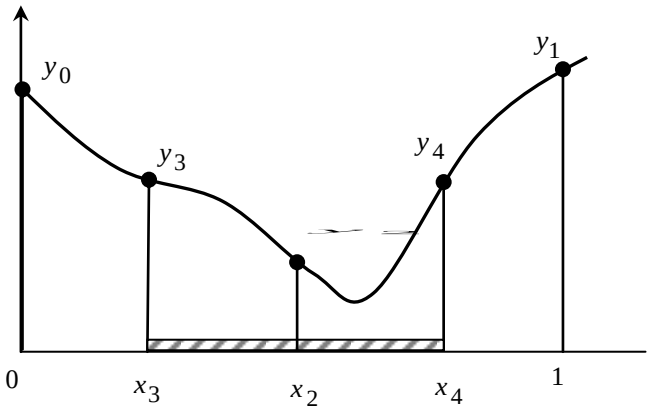

Рис. 1.3. Оптимизация унимодальной функции

Первоначально отрезок неопределенности равен исходному отрезку $X_{\min} = [0; 1]$. Разделим исходный отрезок пополам и вычислим еще два промежуточных значения функции в точках $x_3 = 1/4$ и $x_4 = 3/4$ [*].

Новый отрезок неопределенности будет частью исходного отрезка, для которого выполняется условие $y_{i2} \le y_{i1} \le y_{i3}$ (i1, i2, i3 – смежные точки отрезка неопределенности). В нашем случае новый отрезок неопределенности равен $[x_3, x_4] = [0.25; 0.75]$ (заштрихованная линия).

За 5 обращений к оптимизируемой функции мы в 2 раза уменьшили отрезок неопределенности. Дале, вычислив 2 раза исходную функцию (справа и слева от средней точки), мы уменьшим отрезок неопределенности еще в 2 раза.

---

[*] Для деления отрезка [a; b] в заданных пропорциях $t \in [0; 1]$ можно использовать формулу $x = (1 - t)a + tb$. В нашем случае t=0,5.

Число итераций, необходимых для вычисления минимума функции с заданной точностью ε, можно определить из выражения

$$\varepsilon \approx (1/2)^{[\frac{N-3}{2}]} \text{ или}$$

$$N \approx 3 - 2\log_2 \varepsilon = 3 + 2\log_2 1/\varepsilon. \quad (1.8)$$

Из формулы (1.8) следует, что предложенный алгоритм имеет сложность равную $O(\log_2 1/\varepsilon)$, которая для непрерывных унимодальных функций зависит от точности поиска минимума функции. Для более эффективного алгоритма, например, метода золотого сечения, можно получить существенно меньшую сложность $O(-\frac{\ln 1/\varepsilon}{\ln 0.382})$. На рисунке 1.4 показаны графики трудоемкости алгоритмов двоичного деления и золотого сечения в зависимости от точности поиска минимума функции.

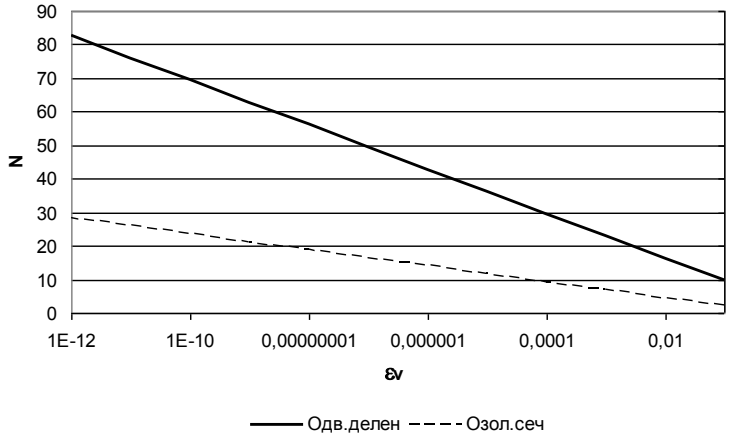

Рис. 1.4. Сложности алгоритмов оптимирзации

1.3.2. Оптимизации многоэкстремальных функций

Для многоэкстремальных функций предложенная выше стратегия совершенно не годится. Функция содержит заранее неизвестное

количество локальных минимумов и максимумов. В общем случае без дополнительных предположения, накладываемых на функцию, данная задача, по-видимому, относится к классу *неразрешимых задач.* Не вдаваясь в детали проблемы оптимизации многоэкстремальных задач, рассмотрим аспекты формирования сложности алгоритма на простом примере.

Пусть оптимизируемой функцией является функция $y = a\sin(\omega x)$. Здесь a – амплитуда, ω - частота. Функция $y = a\sin(\omega x)$ является непрерывной, имеющей бесконечное число непрерывных производных высокого порядка, что, безусловно, хорошо само по себе. На отрезке $[0;2\pi]$ данная функция имеет приблизительно $[\omega]$ минимумов. Если на функцию наложить *условие Липшица*, ограничивающую скорость роста непрерывной функции, то для алгоритма оптимизации появляются некоторые «ориентиры», позволяющие строить достаточно эффективные алгоритмы.

Условие Липшица задается неравенством:

$$\left| f(x_1) - f(x_2) \right| \le L \left| x_1 - x_2 \right|, \qquad (1.9)$$

для любых $x_1, x_2 \in [0;1]$.

Упрощенно можно положить $\left| f'(x) \right| \le L$. Для примера $\left| y' \right| = \left| a\omega\cos(\omega x) \right| < \left| a\omega \right| < L$ или $L \approx \left| a\omega \right|$, следовательно, исходный участок $[0;\ 2\pi]$ неопределенности будет содержать приблизительно $\left|\omega\right|$ минимумов функции (участков унимодальности функции) равномерно распределенных на исходном отрезке. Для исходной функции участки унимодальности приблизительно равны $d_1 = d_2 = \ldots = d_{[\omega]} = d = 2\pi/[\omega]$. На каждом участке унимодальности функции можно применить алгоритм золотого сечения со сложностью для заданной длины отрезка неопределенности d: $N \approx \dfrac{\ln(\varepsilon / d)}{\ln 0.382}$. Тогда интегральная трудоемкость

алгоритма равна $N \approx [\omega] \frac{\ln(\varepsilon\omega)}{\ln 0.382}$ и, следовательно, сложность алгоритма многоэкстремальной оптимизации функции $y = a\sin(\omega x)$ можно оценить величиной $O([\omega] \frac{\ln(\omega/\varepsilon)}{-\ln 0.382})$, которая не противоречит оценки сложности (1.3) для общего случая.

К*атастрофический рост* $N(\varepsilon)$ *при* $\varepsilon \to 0$ *и* $n \to \infty$ *показывает, что бессмысленно ставить вопрос о построении универсальных методов решения «всех вообще» гладких задач оптимизации сколько-нибудь заметной размерности*.

Последнее высказывание относится как детерминированным методам, так и стохастическим, в том числе и генетическим алгоритмам, которым в последнее время незаслуженно приписывают высокие оценки их эффективности. Естественно, что речь идет о методах дающих гарантированные результаты, на отдельных задачах может повезти любому методу.

1.3.3. Сложность выпуклых экстремальных задач

Если на оптимизируемые функции задачи (1.5) наложить еще более жесткие ограничения, например, положить что $f_i(X)$, $i = 0,..., m$ являются выпуклыми непрерывными функциями, G – выпуклое множество, то сложность алгоритмов оптимизации можно оценить величиной $O(n\ln(1/\varepsilon))$, что существенно лучше предыдущего случая.

Интересно, что для самого простого случая, когда все $f_i(X)$, $i = 0,..., m$ функции являются линейными (*задача линейного программирования*), и можно было бы ожидать алгоритма точного решения, наилучший из известных методов - симплекс метод дает только

полиномиальную оценку сложности. Конечно это верхняя оценка сложности, для реальных задач он работает значительно эффективнее.

1.4. Информационно статистический алгоритм глобальной оптимизации функции одной переменной

Среди методов глобальной оптимизации одномерных функций наиболее эффективным является информационно-статистический, однопараметрический метод оптимизации Р.Г. Стронгина [8].

В теоретическом плане метод основывается на определенном способе построении испытаний функции

$$\omega_k = \{(x^1, y^1), (x^2, y^2),..., (x^k, y^k)\}, \qquad (1.10)$$

где (x^j, y^j) - значения независимой переменной и функции в j-ой точке.

Относительно функции предполагается, что она является липшицевой функцией с константой L, т.е. для нее выполняется условие $\left| f(x') - f(x'') \right| \le L \left| x' - x'' \right|$.

При общих предположениях об априорном распределении вероятностей минимизируемой функции в процессе испытаний строятся приближенные апостериорные распределения вероятностей $\xi(\alpha / \omega_k)$ расположения глобального экстремума α, с учетом имеющейся в (1.10) экспериментальной информации.

На каждом шаге абсолютный экстремум выбирается из соображений обеспечения максимума вероятности обнаружить абсолютный экстремум минимизируемой функции

$$\xi(\alpha^*) = \max_{\alpha} \xi(\alpha / \omega_k),$$

что, как показано в работе [8], реализуется за счет максимизации достаточно простой функции

$$R(s) = m(x^s - x^{s-1}) + \frac{(y^s - y^{s-1})^2}{m(x^s - x^{s-1})} - 2(y^s + y^{s-1}),$$

где $m = \begin{cases} 1 & M = 0, \\ rM & M > 0, \end{cases}$ $M = \max\limits_s \dfrac{\left| y^s - y^{s-1} \right|}{(x^s - x^{s-1})}$, r – параметр.

В работе [8] описан алгоритм информационно-статистического метода поиска глобального экстремума. Сам алгоритм представлен в схеме характеристической представимости.

Схему характеристической представимости численного метода оптимизации можно описать следующим образом.

Пусть имеется некоторая последовательность испытаний функции:

$$a = x_1 < x_2 < \ldots < x_k = b\ . \qquad (1.11)$$

Каждому из интервалов (x_{i-1}, x_i) ставится в соответствие *характеристика* интервала $R(i)$.

Работа характеристической схемы заключается в:

1. поиске интервала с максимальной характеристикой

$$R(j) = \max_{1<i\le k} R(i)\ .$$

2. вычислении координаты нового испытания функции и соответствующего значения функции

$$x_{k+1} = S(x_{j-1}, x_j), \quad y_{k+1} = f(x_{k+1}).$$

3. проверке условия остановки алгоритма

$$(x_m - x_{k+1}) < \varepsilon,\ x_m > x_{k+1},\ (x_m - x_{k+1}) = \min_{1<i\le k} (x_i - x_{k+1})\ .$$

4. упорядочивании последовательности точек x_i с учетом нового испытания.

5. перерасчете характеристической функции.

Таким образом, каждое новое испытание увеличивает информацию о характере поведения оптимизируемой функции и позволяет более точно

определять наиболее «перспективные» для исследования участки функции.

Для метода Стронгина:

$$R(i) = m(x_i - x_{i-1}) + \frac{(y_i - y_{i-1})^2}{m(x_i - x_{i-1})} - 2(y_i + y_{i-1}), \qquad (1.12)$$

где $\boldsymbol{m}$ - оценка константы Липшица определяется по результатам проведенных испытаний

$$m = \begin{cases} 1, M = 0, \\ rM, M > 0, \end{cases} \qquad M = \max_{1<i\le k} \frac{|y_i - y_{i-1}|}{x_i - x_{i-1}}. \qquad (1.13)$$

Функция $S(x_{j-1}, x_j)$ вычисляется по формуле:

$$S(x_{j-1}, x_j) = 0.5(x_{i-1} + x_i) - 0.5(y_i - y_{i-1}).$$

Информационно-статистический алгоритм «стремиться» размещать точки испытаний, либо в окрестностях локальных минимумов оптимизируемой функции, тяготея при этом к глобальному минимуму, либо в интервалах неопределенности, размеры которых велики по сравнению с другими участками функции.

1.5. Методы глобальной оптимизации функции многих переменных

Методы глобальной оптимизации функций многих переменных многочисленны и разнообразны. Причиной тому является быстрый рост сложности задачи оптимизации по мере увеличения числа оптимизируемых переменных.

Методические особенности алгоритмов глобальной оптимизации функций многих переменных рассмотрим на примере метода половинных делений. Идея метода половинных делений [3] заключается в организации непропорционального деление исходной области поиска (обычно – единичного гиперкуба) на гиперпараллелепипеды, меньшей размерности. При этом (для двухмерного пространства) единичный куб

делится на две прямоугольные области, которые затем делятся на квадраты четвертной размерности т.д. (см. рис. 1.5). В результате в пространстве поиска формируется нерегулярная сетка испытаний исследуемой функции.

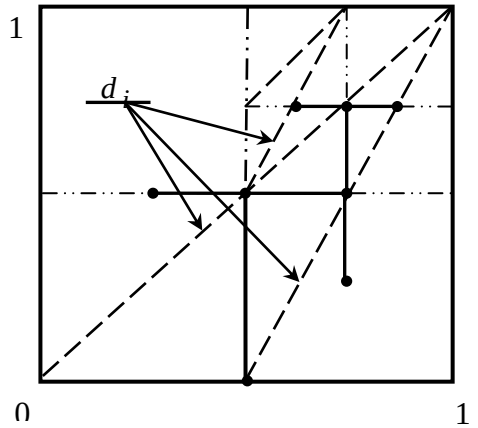

Рис. 1.5. Деление допустимого множества

Пусть D_0 делится на прямоугольные параллелепипеды D_i с центрами c_i и главными диагоналями d_i. Метод половинных делений [3] сводится к построению последовательности $B_k = \{D_1, D_2, ..., D_k\}$ параллелепипедов $D_i \subset D_0$, причем значения функции вычисляются в соответствующей последовательности центров параллелепипедов $Z_k = \{c_1, c_2, ..., c_k\}$.

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

$$\varphi_s = \min_{1 \le i \le k} f(c_i) - (L/2)\left\| d_i \right\|_\infty . \qquad (1.14)$$

Именно он подвергается дальнейшему делению на две части.

Используя условие Липшица, параллелепипеды, для которых $\varphi_j \geq R^{(k)}$, исключаются из дальнейшего рассмотрения. Здесь $R^{(k)} = \min\limits_{c_i \in Z_k} f(c_i)$ - текущий рекорд при поиске минимума функции.

Для остановки алгоритма можно использовать условие

$$\left\| d_i \right\|_\infty \leq \varepsilon \,. \qquad (1.15)$$

1.5.1. Информационно-статистический вариант метода половинных делений

Сохранив предложенную в [3] схему двоичного деления параллелепипедов, изменим процедуру выбора «критического» параллелепипеда. Ориентировка на «минимальный» параллелепипед может привести к чрезмерно детальному рассмотрению области одного из локальных минимумов и «досрочной» остановке алгоритма при уменьшении размеров параллелепипеда в соответствии с условием остановки (1.15).

В работе [8] 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. Данная стратегия не позволяет алгоритму оптимизации «зацикливаться» на локальных экстремумах и реализует более сбалансированную стратегию просмотра параллелепипедов при поиске глобального экстремума функции (см. п. 1.4).

С каждым параллелепипедом D_i свяжем набор параметров $S_i = (K_i, c_i, f_i, h_i, d_i)$, где h_i - расстояние между текущим параллелепипедом и его *p*-м «родителем». K_i - характеристика i-го

параллелепипеда, которую в соответствии с [8] можно вычислить по формуле:

$$K_i = mh_i + \frac{(f_i - f_p)^2}{mh_i} - 2(f_i + f_p).$$

Здесь m - оценка константы Липшица, которая определяется по результатам проведенных испытаний:

$$m = \begin{cases} 1, & M = 0, \\ rM, & M > 0, \end{cases} \qquad M = \max_{1 < i \le k} \frac{|f_i - f_p|}{h_i},$$

где $r > 1$ – «регулировочный» коэффициент.

Из наборов S_i сформируем, упорядоченный в направлении убывания характеристики K_i, характеристический список $S = \{S_{i_1}, S_{i_2}, ..., S_{i_k}\}$, где $K_{i_1} \ge K_{i_2} \ge ... \ge K_{i_k}$.

Модификация метода половинных делений сводится к тому, что при двоичном делении всегда выбирается параллелепипед α, для которого $K_\alpha = \max_{1 \le i \le k} K_i$, т.е. первый параллелепипед списка *S*.

Следует отметить, что в списке S на любом этапе работы модифицированного алгоритма половинных делений содержатся параллелепипеды ещё не подвергнутые двоичному делению. Причем «разделяемые» параллелепипеды удаляются из списка.

1.5.2. Модификация метода половинных делений на основе локальной техники

Не смотря на существенное повышение эффективности модифицированного алгоритма метода половинных делений, для него сохраняется экспоненциальный характер роста сложности в зависимости от размерности задачи оптимизации. Дальнейшее совершенствование алгоритмов глобальной оптимизации возможно на пути использования локальной техники, когда стратегия глобального поиска удачно

сочетается с методами локальной оптимизации, для которых рост сложности имеет полиномиальный характер.

В локальной технике этап глобальной оптимизации можно производить достаточно грубо, поскольку основной целью этого этапа является определение областей притяжения локальных экстремумов функции, из которых запускаются алгоритмы локальной оптимизации. При ограниченном количестве экстремумов области притяжения имеют значительные размеры и их можно быстро идентифицировать.

На рисунке 1.6 для многоэкстремальной функции двух переменных показаны зоны притяжения локальных минимумов.

Предлагается следующий адаптивный алгоритм формирования списка областей притяжения оптимизируемой функции.

Пусть r – планируемое количество зон притяжения экстремума; ρ_r - предполагаемый радиус области притяжения экстремума. Сформируем список областей притяжения локальных экстремумов $V = \{V_1, V_2, ..., V_r\}$, 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 области оценку $\tilde{f}_i$. Вектора $\tilde{c}_i$ условно считаются центрами областей притяжения.

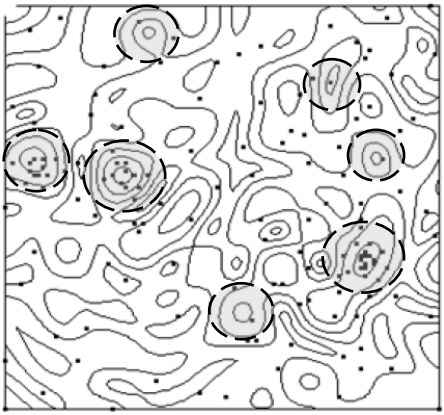

Рис. 1.6. Области притяжения экстремумов функции

Первоначально список V – пуст. По мере испытаний он наполняется элементами, но в конце этапа глобального поиска не может содержать больше чем r элементов. Элементы списка V упорядочены таким образом, что $\tilde{f}_1 > \tilde{f}_1 > \ldots > \tilde{f}_r$.

Пусть в процессе работы алгоритма глобальной оптимизации произведено очередное испытание $f_k = f(c_k)$. Эволюция содержимого списка V (размера m ($m<=r$)) происходит по следующим простым правилам:

1. Проверяется принадлежность c_k окрестностям одной из имеющихся областей притяжения $V_i \quad i = 1,\ldots, m$.

   1.1. Если

   $$c_k \in \{x \mid \| (x - \tilde{c}_i) \| \le \rho_r \}, \qquad (1.16)$$

   то при $f_k < \tilde{f}_i$ содержимое элемента V_i обновляется ($V_i = (c_k, f_k)$).

   Иначе полученная информация игнорируется.

   Проверяется правило 2.

   1.2. Если условие (1.16) не выполняется, то проверяем правило 2.

2. Определяется возможность включения нового элемента в список V.

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

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

   2.3. Если $(f_k < \tilde{f}_m)$ & $(m < r)$, то элемент V_k записывается в конец списка V.

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

Общая идея использования локальной техники в методе половинных делений заключается в организации вычислений оптимума функции в два этапа.

На первом этапе организуется глобальный поиск областей притяжения локальных экстремумов так, как это описано в разделе 1.5.1. При существенно сниженных требованиях к точности глобальной оптимизации.

На втором этапе из точек, принадлежащих областям притяжения локальных экстремумов, организуется поиск минимумов функции локальными алгоритмами оптимизации. В нашем случае использовался метод деформированных многогранников. Как показали вычислительные эксперименты, локальная техника позволяет существенно повысить эффективность алгоритма глобальной оптимизации.

1.5.3. Параллельные версии модифицированного метода половинных делений

В качестве первого варианта параллельной версии рассмотрим тривиальную схему организации параллельных вычислений для модифицированного метода половинных делений.

Исходную область D_0 разобьем, например, на 2^n равных частей. В каждой из областей на отдельном процессоре организуем глобальный поиск модифицированным методом половинных делений (см. рис. 1.7, здесь область D_0 делится на 4 части). В данном случае используются только первая модификация метода половинных делений. При этом выигрывают все запущенные процессы, поскольку быстрее оценивается константа Липшица, что приводит к большему проценту прореживания «бесперспективных» прямоугольников. Кроме того, поиск зон

притяжения в областях меньшей размерности завершается несколько раньше.

В финале, в режиме параллельных вычислений, для каждой из точек, принадлежащих областям притяжения, организуется локальная оптимизация.

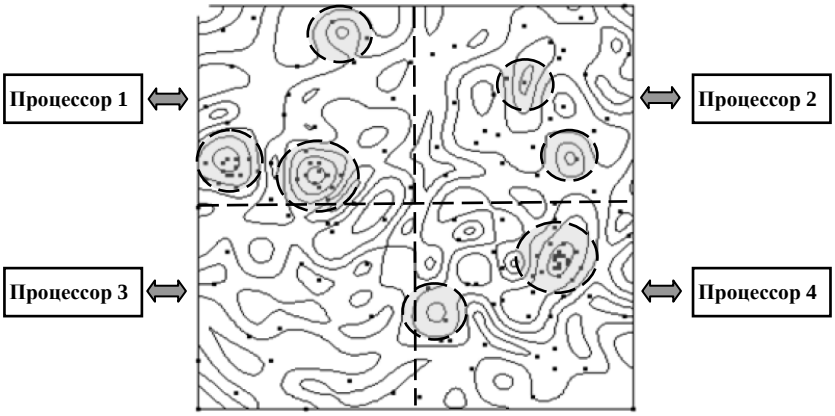

Рис. 1.7. Деление области поиска экстремума на локальные подобласти

Вторая версия организации распараллеливания вычислений несколько изящнее. В основу алгоритма параллельных вычислений положена идея распределения по параллельным процессам элементов списка *S*. На рисунке 1.8 представлена схема алгоритма.

Первоначально с помощью последовательного варианта алгоритма двоичного деления, описанного в разделе 1.5.1, формируется первоначальный характеристический список *S* с числом элементов не менее *p*. Далее, старшие элементы списка *S* «разбираются» по *p* параллельным процессорам, на которых, с помощью идентичных процедур метода двоичного деления организуется, глобальный поиск областей притяжения.

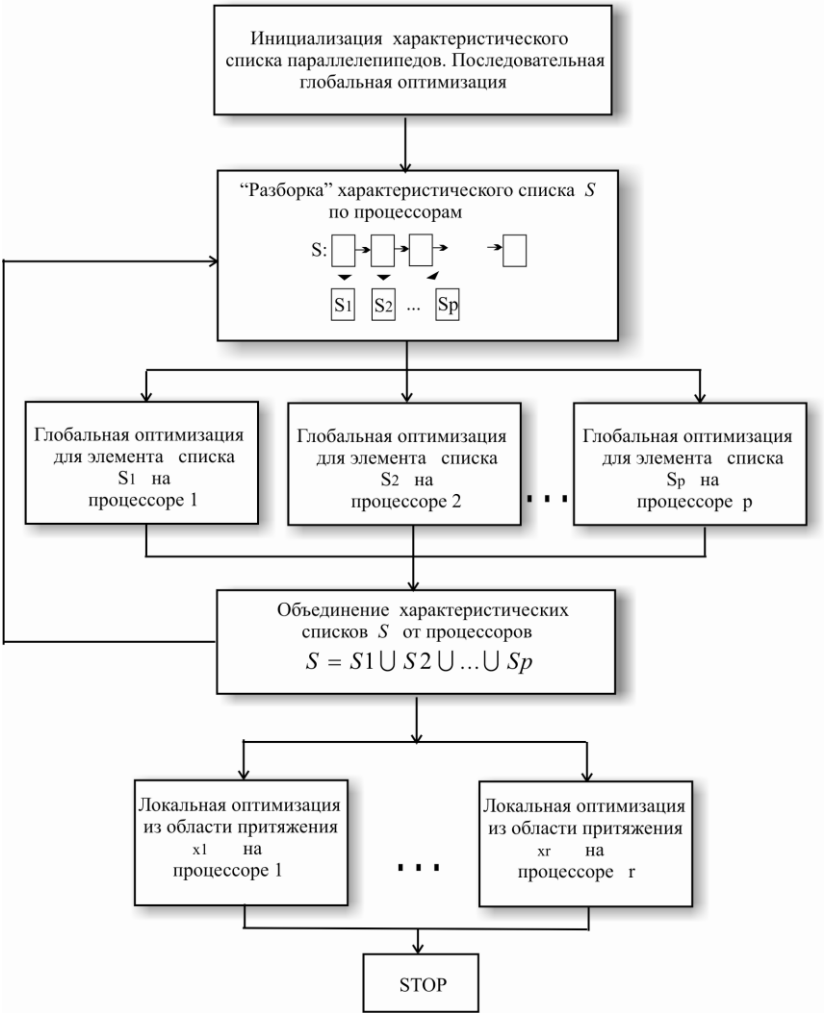

Рис. 1.8. Параллельная версия алгоритма двоичного деления

Для выравнивания загруженности процессоров, на каждом из них реализуется одинаковое количество делений исходных

параллелепипедов. С целью повышения эффективности вычислений, каждому из процессоров доступна информация о текущей достигнутой оценке константы Липшица. В результате на каждом процессоре формируется собственные характеристические списки S_i, которые в совокупности, по завершению работы параллельных веток, объединяются в один общий список $S = S_1 \cup S_2 \cup ... \cup S_p$.

Операция объединения списков выгодна, поскольку в общем списке *S* элементы списков S_i по критерию K_j упорядочиваются относительно друг друга, что на следующем шаге параллельных вычислений позволяет организовать поиск из более «перспективных» параллелепипедов. В результате - алгоритм раньше входит в зону глобального экстремума и быстрее происходит оценка верхней грани константы Липшица.

Этап параллельного поиска областей притяжения экстремумов заканчивается либо при достижении заданного уровня точности, либо при вырождении списка *S*. На завершающем этапе работы алгоритма из найденных точек притяжения x_j запускается параллельный поиск локальных экстремумов, например, с помощью метода деформированных многогранников

1.6. Показатели эффективности параллельных программ

Для оценки эффективности организации параллельных вычислений существует ряд критериев [2]. Одним из основных показателей является *ускорение.*

Определение 1.4. ***Ускорение****, получаемое при использовании параллельного алгоритма для p процессоров, по сравнению с последовательным вариантом выполнения вычислений, определяется величиной*

$$S_p = T_1 / T_p , \qquad (1.17)$$

где T_1 *- время решения задачи на одном процессоре,* T_p *- время решения задачи на* p *процессорах*.

Ускорение по-существу определяется как отношение времени решения задач на скалярной ЭВМ к времени выполнения параллельного алгоритма. Для алгоритмов оптимизации ускорение можно подсчитывать по формуле

$$S_p = N_1 / \max_p N_p , \qquad (1.18)$$

где N_1 - число обращений к оптимизируемой функции на одном процессоре, N_p - число обращений к функции на каждом из p процессоров.

Другим критерием оценки качества параллельного алгоритма может служить *эффективность.*

Определение 1.5. ***Эффективность*** *использования параллельным алгоритмом процессоров при решении задачи определяется соотношением*

$$E_p = T_1 / pT_p = S_p / p . \qquad (1.19)$$

Величина эффективности определяет среднюю долю времени выполнения алгоритма, в течение которой процессоры реально используются для решения задачи.

Из приведенных соотношений можно показать, что в наилучшем случае: $E_p = 1$ *и* $S_p = p$. На практике для оценки эффективности параллельных вычислений следует учитывать следующих два важных момента.

При определенных обстоятельствах ускорение может оказаться больше числа используемых процессоров - в этом случае говорят о существовании *сверхлинейного* ускорения. Несмотря на парадоксальность таких ситуаций (ускорение превышает число

процессоров), на практике сверхлинейное ускорение имеет место. Одной из причин *сверхлинейного* ускорения являются особенности выполнения последовательной и параллельной программ. Например, при решении задачи на одном процессоре оказывается недостаточно оперативной памяти для хранения всех обрабатываемых данных. В результате используется более медленная внешняя память. Для нескольких процессоров оперативной памяти может оказаться достаточно за счет разделения данных между процессорами.

Еще одной причиной сверхлинейного ускорения может быть нелинейный характер зависимости сложности решения задачи в зависимости от объема обрабатываемых данных. Так, например, известный алгоритм пузырьковой сортировки имеет квадратичную сходимость в зависимости от числа упорядочиваемых данных. В результате при распределении сортируемого массива между процессорами может быть получено ускорение, превышающее число процессоров. Причина сверхлинейного ускорения может быть скрыта в различиях реализации вычислительных схем последовательного и параллельного методов.

При внимательном рассмотрении можно обратить внимание, что попытки повышения качества параллельных вычислений по одному из показателей (ускорению или эффективности) может привести к ухудшению ситуации по другому показателю, ибо показатели качества параллельных вычислений являются противоречивыми. Так, например, повышение ускорения обычно может быть обеспечено за счет увеличения числа процессоров, что приводит, как правило, к падению эффективности. Верно и обратное, повышение эффективности достигается во многих случаях при уменьшении числа процессоров (в предельном случае идеальная эффективность $E_p = 1$ легко обеспечивается при использовании одного процессора). В результате разработка методов параллельных вычислений часто предполагает поиск

компромиссного варианта решения с учетом желаемых показателей ускорения и эффективности.

При выборе надлежащего параллельного способа решения задачи может оказаться полезной оценка *стоимости* вычислений, определяемой как произведение времени параллельного решения задачи и числа используемых процессоров.

$$C_p = pT_p$$

Определение 1.6. *Определим понятие* ***стоимостно-оптимального*** *параллельного алгоритма как метода, стоимость которого пропорциональна времени выполнения наилучшего последовательного алгоритма.*

1.7. Контрольные вопросы

1. Опишите формальную постановку задачи глобальной оптимизации.
2. Какую роль играет условие Липшица в задачах глобальной оптимизации?
3. Основные проблемы задачи глобальной оптимизации?
4. Чему равна нижняя оценка стохастической сложности задачи глобальной оптимизации?
5. К чему приводит повышение точности поиска глобального экстремума и увеличение числа оптимизируемых параметров?
6. Как оценить трудоемкость алгоритма оптимизации?
7. От чего зависит эффективность алгоритмов оптимизации?
8. Опишите алгоритм двоичного деления оптимизации одномерной функции.
9. Опишите информационно-статистический алгоритм глобальной оптимизации функции одной переменной.
10. Как вычисляется характеристическая функция?
11. В чем заключается идея метода половинных делений?

12. В чем основная идея модификации метода половинных делений?
13. Почему модифицированный метод половинных делений не «зацикливается» в локальных экстремумах?
14. Основное содержание локальной техники?
15. Как определяется области притяжения локальных экстремумов?
16. Опишите параллельные версии модифицированного метода половинных делений.
17. Как определяются понятия ускорения и эффективности?
18. В чем состоит противоречие показателей ускорения и эффективности?
19. В чем состоит понятие стоимостно-оптимального параллельного алгоритма?

2. Разработки моделей параллельных алгоритмов в системе PGRAPH

2.1. Знакомство с системой

Система PGRAPH – автоматизированная система проектирования и разработки программ с использованием технологии графосимволического программирования. Система предоставляет визуальную среду для создания графического образа агрегатов и спецификации других объектов технологии ГСП, средства автоматизированного анализа проектируемых программ с целью повышения их надежности, а также средства автоматического синтеза исходных текстов программ на основе объектов технологии ГСП. С помощью внешнего компилятора для языка программирования, на котором написаны базовые модули, система позволяет компилировать и запускать на выполнение созданные в ней программы. В текущей версии системы поддерживается язык программирования C++.

Главное окно системы изображено на рисунке 2.1.

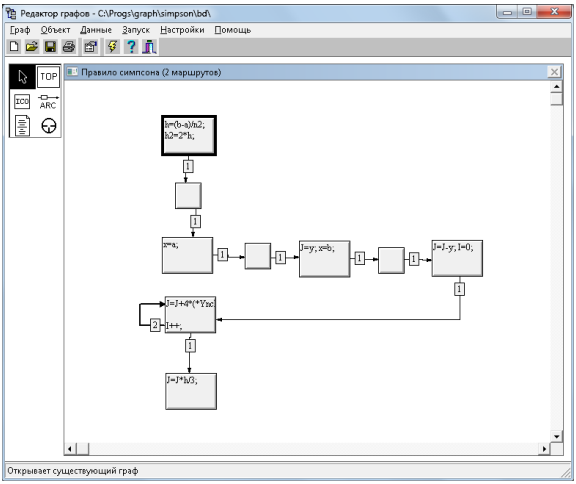

Рис. 2.1 – Главное окно системы ГСП GRAPH

2.2. Создание словаря данных ПОП

В соответствии с методологией ГСП, разработка программы начинается с определения используемых типов данных и словаря данных ПОП. Эти действия выполняются в диалоговых окнах «Список типов» и «Словарь данных», вызываемых с помощью одноименных пунктов меню «Данные».

Для вновь создаваемого типа вводится его имя и определение в соответствии с синтаксисом языка C++. Базовые типы языка C++ уже содержатся в информационном фонде системы.

При создании данного (переменной) предметной области пользователь системы определяет:

- имя данного;
- тип данного (выбирается из списка существующих в предметной области типов);
- начальное значение;
- комментарий, описывающий назначение данного.

2.3. Создание базовых модулей

Базовые модули служат основой для построения объектов технологии ГСП (акторов и предикатов).

Исходный текст базового модуля создается во внешнем текстовом редакторе и сохраняется в виде файла с расширением ".C" в рабочий каталог системы PGRAPH.

Для использования базового модуля его необходимо зарегистрировать в системе. Регистрация производится путем выбора меню "Объект" -> "Зарегистрировать модуль" (рисунок 2.2).

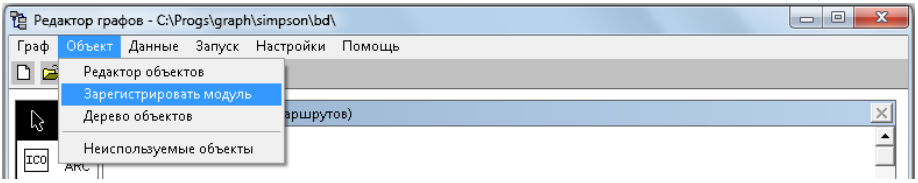

Рис. 2.2 – Меню регистрации базового модуля

Диалоговое окно регистрации базового модуля изображено на рисунке 2.3. Процесс регистрации состоит из двух шагов. На первом шаге нужно выбрать файл с исходным текстом базового модуля. В поле «Комментарий» вводится назначение и краткое описание создаваемого базового модуля.

На втором шаге для каждого параметра базового модуля необходимо определить его класс (Исходный, Вычисляемый или Модифицируемый).

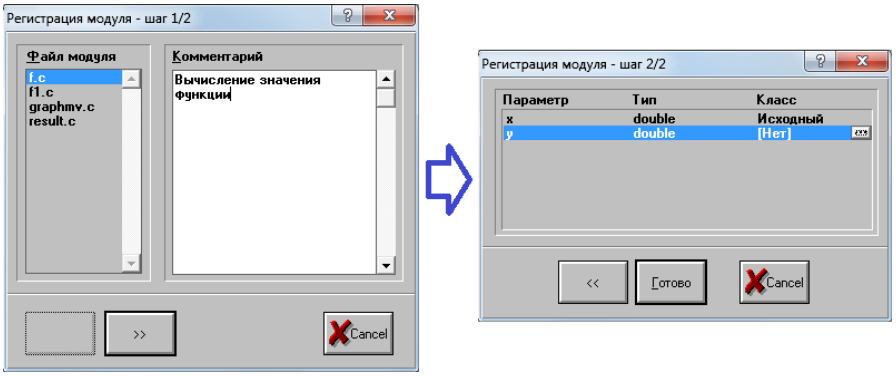

Рис. 2.3 - Диалоговое окно регистрации базового модуля

После нажатия на кнопку «Готово» базовый модуль регистрируется в системе.

2.4. Создание объектов технологии ГСП

Граф-программа строится из объектов технологии ГСП: акторов и предикатов.

Для создания объектов ГСП служит «Редактор объектов», вызываемый выбором одноименного пункта в меню «Объект» (рисунок 2.4).

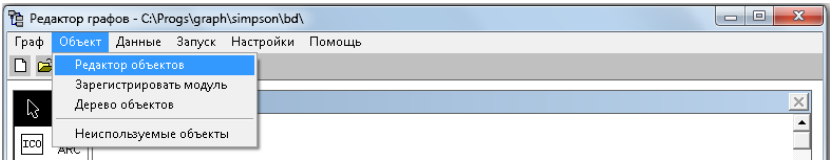

Рис. 2.4 – Вызов редактора объектов

Окно редактора объектов приведено на рисунке 2.5.

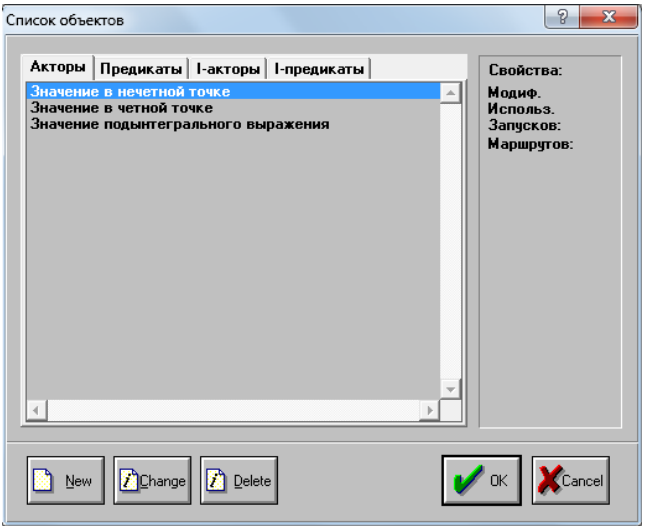

Рис. 2.5 – Окно редактора объектов

Объекты ГСП условно делятся на две группы: Inline-объекты (I-акторы, I-предикаты) и обычные объекты (Акторы и Предикаты).

Обычные объекты создаются на основе базовых модулей. Inline-объекты отличаются от обычных тем, что их базовые модули создаются непосредственно при создании самого объекта. Как правило, inline-объекты выполняют простые действия. Например, I-актор может выполнять инкремент счетчика (J = J + 1;).

В этом случае для него нецелесообразно создавать отдельный файл с базовым модулем. Вместо этого текст базового модуля вводится непосредственно при создании I-актора. Аналогично, несложные предикаты, осуществляющие например, сравнение двух переменных, оформляются в виде I-предикатов.

Для создания актора, предиката, I-актора или I-предиката необходимо выбрать соответствующую вкладку в окне Редактора объектов и нажать кнопку «Создать».

2.4.1. Создание inline-акторов

Создание I-актора состоит из двух шагов, приведенных на рисунках 2.6 и 2.7.

На первом шаге (рисунок 2.6) вводится исходный текст I-актора и выбирается картинка (иконка), которая может отображаться на вершинах с этим I-актором вместо текста.

Переход ко второму шагу происходит по нажатию на кнопку " >> ".

На втором шаге для каждого данного ПОП, используемого в I-акторе, назначается его класс (исходное, вычисляемое или модифицируемое) и вводится краткий комментарий с описанием этого данного (рисунок 2.7).

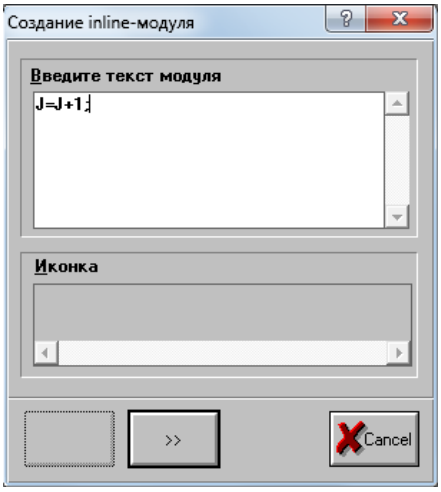

Рис. 2.6 – Окно редактирования исходного текста inline-актора

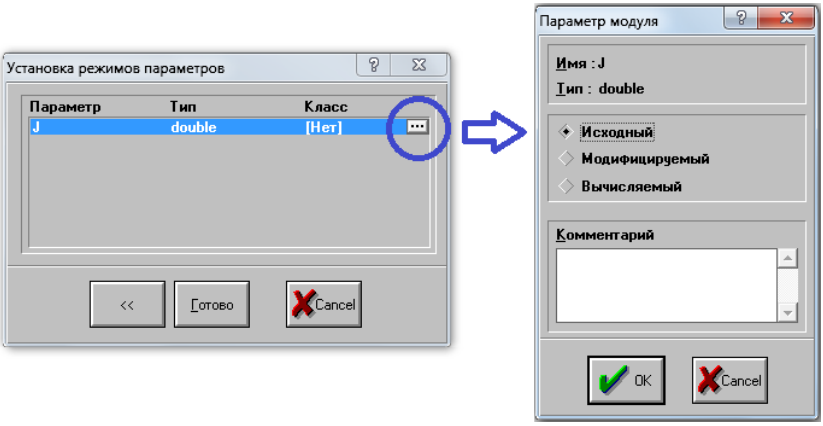

Рис. 2.7 – Назначение класса использования для данных, используемых актором

2.4.2. Создание inline-предикатов

Создать I-предикат очень просто. Достаточно ввести его исходный текст (т.е. логическое условие над данными ПОП) и нажать кнопку «OK» (рисунок 2.8).

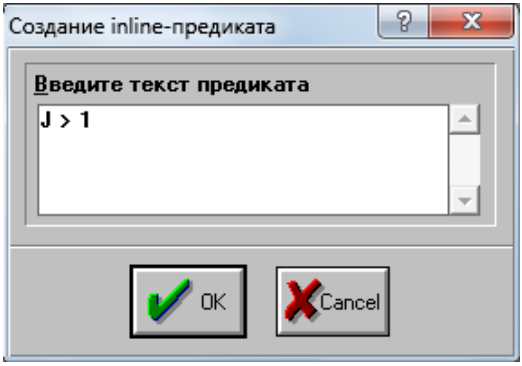

Рис. 2.8 – Создание inline-предиката

2.4.3. Создание акторов

Акторы, в отличие от I-акторов, создаются на основе базовых модулей. Процесс создания актора состоит из трех шагов.

На первом шаге вводится имя для вновь создаваемого актора и выбирается картинка (иконка) для отображения на вершинах в этим актором (рисунок 2.9).

На втором шаге выбирается базовый модуль, на основе которого строится актор (рисунок 2.10).

На третьем шаге производится паспортизация базового модуля: каждому его параметру ставится в соответствие данное ПОП такого же типа (рисунок 2.11).

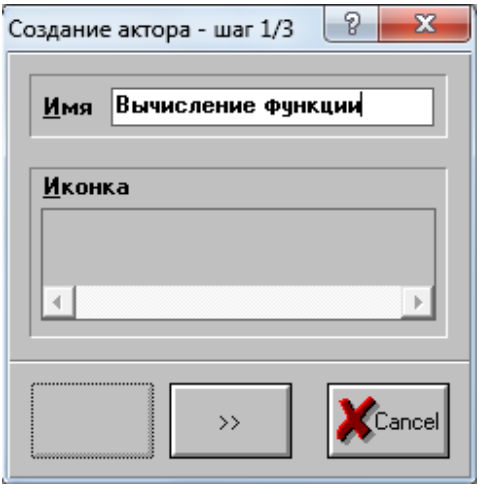

Рис. 2.9 – Первый шаг создания актора

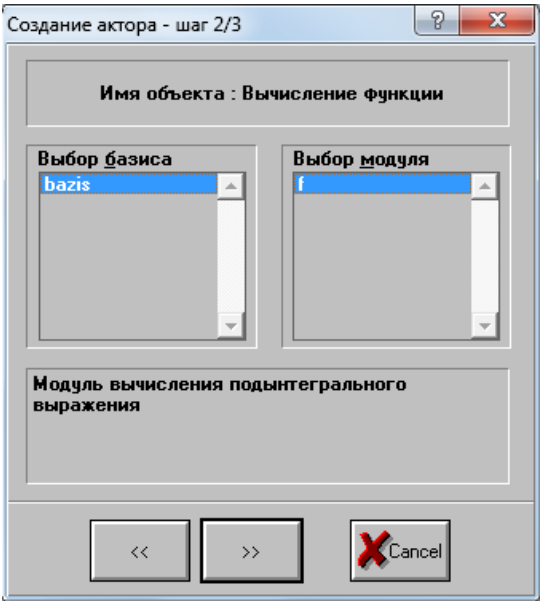

Рис. 2.10 – Второй шаг создания актора

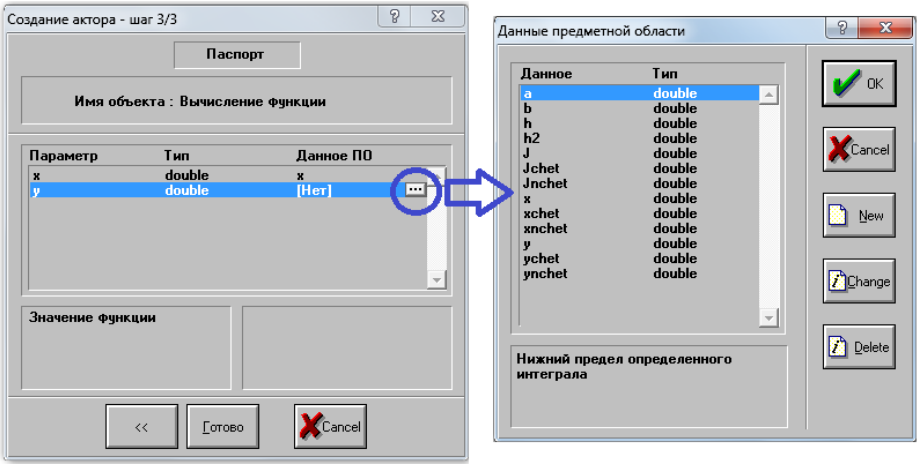

Рис. 2.11 – Третий шаг создания актора (паспортизация)

2.4.4. Создание предикатов

Процесс создания предиката аналогичен процессу создания актора. Он также состоит из трех шагов: ввод имени предиката, выбор базового модуля, паспортизация.

2.5. Разработка граф-модели программы

Программа в технологии ГСП представляется в виде граф-модели – ориентированного помеченного графа (агрегата), описывающего развитие вычислительного процесса. Агрегат состоит из вершин и дуг.

Для рисования агрегата используются готовые визуальные элементы, находящиеся на панели редактирования. Внешний вид панель редактирования и назначение ее элементов приведены на рисунке 2.12.

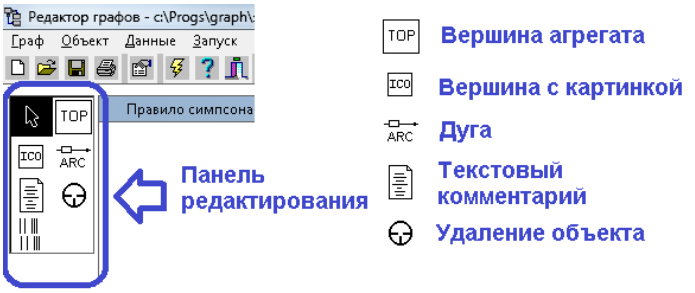

Рис. 2.12 – Панель редактирования агрегата

Щелчок на изображении объекта в панели редактирования переводит в режим добавления к агрегату соответствующих объектов (вершин и дуг). Добавление осуществляется щелчком в поле редактирования агрегата.

Рисование агрегата начинается с добавления вершин. Затем вершины соединяются дугами. Пример созданного в системе агрегата приведен на рисунке 2.13.

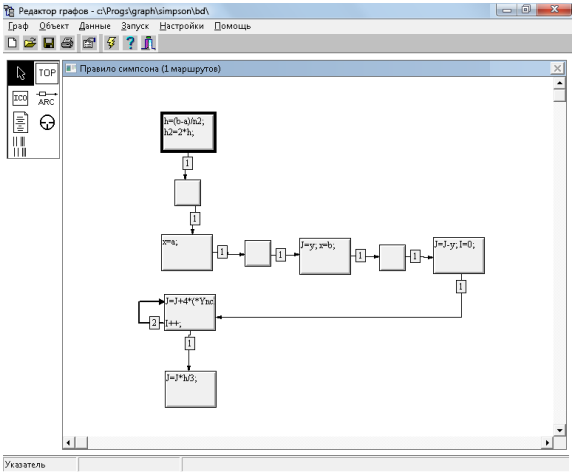

Рис. 2.13 – Пример агрегата

Редактирование агрегата завершается назначением актора каждой вершине и назначением предиката каждой дуге. Для назначения актора необходимо дважды щелкнуть на соответствующей вершине и выбрать актор в открывшемся окне «Свойства вершины» (рисунок 2.14).

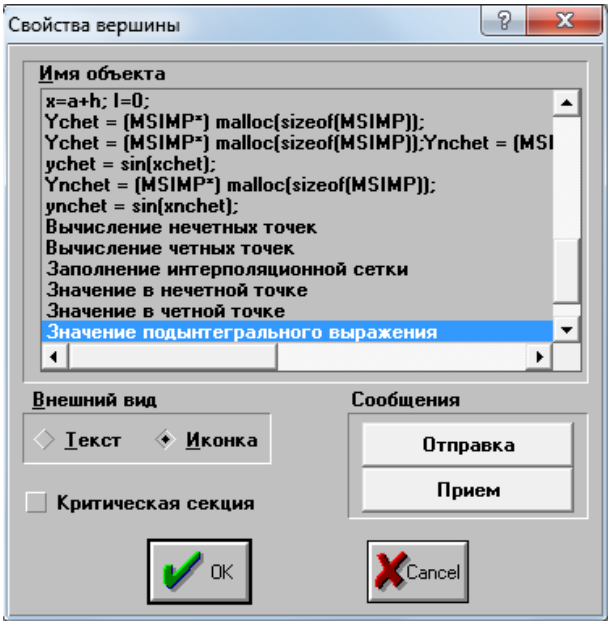

Рис. 2.14 – Назначение актора для вершины агрегата

Для назначения предиката некоторой дуге агрегата необходимо дважды щелкнуть на прямоугольнике в начале дуги. После этого откроется окно «Свойства дуги», в котором можно выбрать предикат, а также изменить приоритет дуги (рисунок 2.15).

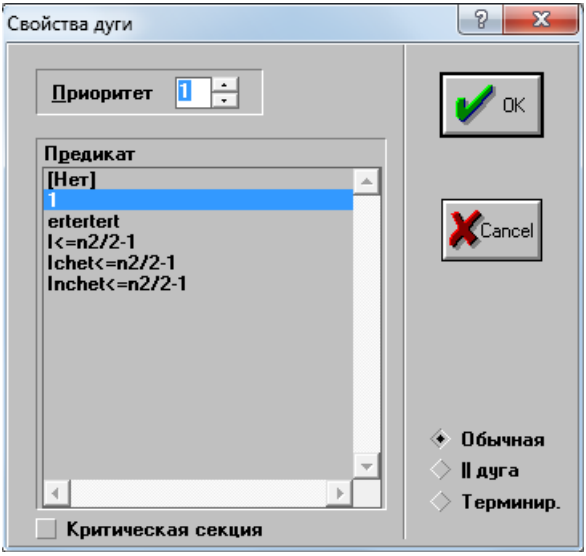

Рис. 2.15 – Выбор предиката для дуги агрегата

Отличие модели параллельного агрегата от последовательного заключается в наличии в нем фрагментов, которые могут исполняться одновременно, т.е. в использовании параллельных ветвей. Любая параллельная ветвь начинается с дуги параллельного типа и заканчивается дугой терминирующего типа.

В системе PGRAPH любая дуга изначально создается как последовательная. После создания дуги можно изменить ее тип в окне «Свойства дуги» (рисунок 2.15).

При изменении типа дуги изменяется также ее внешний вид. Для параллельной дуги прямоугольник в начале дуги заменяется на кружок, а для терминирующей дуги – на наклонную линию.

2.6. Компиляция и запуск программы

Генерация исполняемого файла на основе созданной граф-программы осуществляется выбором пункта «Построение и запуск» в меню «Запуск».

При выборе этого пункта система сначала проверяет корректность параллельного агрегата, то есть его соответствие правилам построения параллельных граф-моделей в технологии ГСП. Затем синтезируется исходный текст на целевом языке программирования и вызывается внешний компилятор.

Для создания параллельных программ текущая версия системы PGRAPH использует библиотеку и среду исполнения MPI. По умолчанию созданная в системе программа запускается на локальном компьютере, поэтому на нем должна быть установлена одна из реализаций MPI (например, MPICH).

Для проведения вычислительных экспериментов на кластере необходимо отметить галочкой пункт «Запуск на кластере» меню «Настройки». В этом случае система сгенерирует отдельную версию исходных текстов параллельной программы, предназначенную для компиляции на кластере. Каталог, в который система сохраняет сгенерированные файлы, указывается в диалоговом окне «Размещение», в поле «Выходные тексты на С». Указанное окно открывается при выборе пункта «Размещение» меню «Настройки».

Сгенерированные системой исходные тексты необходимо скопировать на кластер, откомпилировать там и запустить полученную программу в соответствии с инструкциями по работе на конкретном кластере.

Инструкции по работе на кластерах суперкомпьютерного центра СГАУ можно найти на сайте hpc.ssau.ru в разделе «Инструкции».

3. Предметная область разработки моделей параллельных алгоритмов глобальной оптимизации

Описание данных предметной области (ПО) и её программных модулей можно найти в соответствующих разделах ПО системы PGRAPH. Более подробно опишем основные компоненты ПОП.

3.1. Описание основных структур данных предметной области «Глобальная оптимизация»

В первую очередь опишем структуру характеристического списка, используемого в одном из параллельных алгоритмов глобальной оптимизации.

Характеристический список типа **QVAD2S** имеет структуру представленную ниже.

```
typedef struct LQV{
        int Nsimp;              // Номер элемента списка
        double d2z;             // Индекс направления деления
        double Str;             // Длина диагонали частичного параллелепипеда
        double Ksum;            // Характеристическая оценка параллелепипеда
        XI  x0;                 // Координаты центра параллелепипеда
        double z;               // Значение функции в центре параллелепипеда
        double hx;              // Размер параллелепипеда
        FUNC DelS;              // Значение функции параллелепипеда-предка
        int Loran;              // Флаг управления
        int Hbd;                // Глубина поиска
        struct LQV *next;       // Указатель на следующий элемент списка
                        } QVAD2;
```

В алгоритмах глобальной оптимизации используется список представителей локальных зон притяжения глобальных максимумов. Список имеет следующую структуру.

```
typedef struct LM{
        XI  xf0;                // Координаты точки притяжения
        FUNC fmax;              // Значение функции в точке притяжения
        int Npri;               // Рабочий параметр
        XI  xLm;                // Координаты локального максимума (ЛМ)
        FUNC fLmax;             // Значение функции в точке ЛМ
        struct LM *next;        // Указатель на следующий элемент списка
        } LMAX;
```

Остальные данные предметной области представлены в таблице 3.1.

Таблица 3.1. Данные ПОП «Глобальная оптимизация»

Имя данного	Тип	Нач. значение	Комментарий
1	2	3	4
hx	double	0.5	Шаг по x
z2	FUNC	0	Значение функции
Str	double	0	Площадь симплекса
Nsimp	int	1	Номер симплекса
Ksum	double	0	Основной критерий оптимизации
MaxF	long	0	Максимальный номер текущего значения массива функций
j1	int	0	Индекс вершины
j6	int	0	Индекс вершины
zp1	FUNC	-1.0e77	Дополнительная функция
zp2	FUNC	1.0e77	Дополнительная функция
x0	double	0	x0
lx	double	0.0078	Величина «зерна» просмотра

Продолжение таблицы 3.1

1	2	3	4
Lor	int	0	Критерий разрывности функции
Nmax	int	0	Число сохранения максимума функции. (Число локальных экстремумов симплекс метода.)
Sumstep	long	0	Суммарное число шагов в тернарном дереве
Nstep	long	0	Число шагов текущего обращения к тернарному дереву
Ntern	int	0	Число обращений к тернарному дереву
NFUNC	int	21	Номер тестируемой функции
Hqva	QVAD2S	NULL	Голова списка квадратов
wold	FUNC	0	Старое значение функции в симплексе
mi	double	1	Константа Лифшица
mmax	double	0	Оценка констатнты Лифшица
Esimp	double	1.0e-4	Зерно просмотра метода деформированных многогранников
Estron	double	0.016	Критерий остановки метода Стронгина
HXMAX	LMAXS	NULL	Список приближений локальных максимумов и их точных значений
Eloc	double	0.1	Радиус зоны локального максимума
KlocMax	int	10	Количество начальных приближений при поиске локальных максимумов
Nmax2	int	0	Число локальных областей поиска экстремума функции
Htmax	LMAXS	NULL	Текущий указатель на список локальных максимумов

Продолжение таблицы 3.1

1	2	3	4
Nlok	int	2	Число локальных шагов в параллельных ветках оптимизации
Nloc0	int	10	Количество шагов при начальном формировании симплексов
Krep	int	5	Количество подтверждений точек локализации экстремумов
GKLtip	int	1	Тип тестовой функции (1- D; 2 - D2; 3 - ND)
N_fun	int	63	Номер тесмтовой функции
Имя данного	Тип	Нач. значение	Комментарий
d2z	double	0	Абсолютная кривизна
j2	int	0	Индекс вершины
xp1	double	0	Дополнительный аргумент
xp2	double	0	Дополнительный аргумент
pstream	PFILE	NULL	
Akfun	MPAR		Коэффициенты тестовых функций
X	XI		Вектор аргументов функции
w0	double	0	Zi
Hq2	QVAD2S		Указатель на текущий квадратов
Hbd	int	0	Номер текущего уровня двоичного дерева на ветке в процессе поиска
Diag	double	0	Диагональ частичного гиперпрямоугольника

Продолжение таблицы 3.1

1	2	3	4
PRLp	SEGM		Описание параллельных процессов: указатели на списки, число испытаний в процессах
NprlTec	int	0	Число текущих процессов
MaxPRL	int	0	Максимальное число обращений к функции в !! процессах

3.2. Описание основных программных модулей предметной области «Глобальная оптимизация»)

Глобальный поиск максимума функции реализуется с помощью базового модуля **Bkvadr2**, который имеет следующие параметры:

HXMAX – список зон притяжения;

Xn1 – первая начальная точка алгоритма;

Xn2 – вторая начальная точка алгоритма;

hx – шаг алгоритма оптимизации;

Estron - Критерий остановки метода Стронгина;

lx – размер «зерна» (размера частичного прямоугольника) поиска;KlocMax – установленное число регистрируемых локальных зон притяжения;

NFUNC – номер оптимизируемой функции;

Akfun – структура, предназначенная для хранения накопленной информации;

Nstep – число шагов алгоритма;

iNX – индекс (направление) деления параллелепипеда;

Zmax – достигнутое максимальное значение оптимизируемой функции.

Модуль **Bkvadr2** в соответствии с информационно-статистическим методом половинных делений (см. п. 1.5.1) реализует поиск глобального

максимума в выделенной (параметрами Xn1, Xn2, hx, iNX) локальной области.

Локальный поиск максимума функции реализуется с помощью базового модуля **deform,** который имеет следующие параметры:

X – координаты вектора начального приближения локального максимума и результат поиска максимума;

Akfun - структура, предназначенная для хранения накопленной информации;

NFUNC – номер оптимизируемой функции;

Esimp – относительная точность поиска максимума функции.

Модуль **deform** реализует поиск локального максимума с помощью метода деформированных многогранников.

3.3. Описание генератора тестовых многоэкстремальных функций многих переменных (GKLS)

В процессе разработки методов глобальной оптимизации большое значение приобретают тестовые функции, с помощью которых можно исследовать свойства разрабатываемых программных приложений, сравниваться по эффективности с другими алгоритмами, конкретизировать проблемы исследуемого алгоритма и находить способы их преодоления.

В 1998 году появился свободно распространяемый пакет генерации тестовых функций GKLS [1] с заданными свойствами, облегчающий разработчикам алгоритмов глобальной оптимизации проводить свои исследования. В силу возрастающей популярности в настоящее время GKLS играет роль своеобразного стандарта, по которому настраивают алгоритмы глобальной оптимизации. Генератор GKLS распространяется бесплатно. Его можно получить, послав сообщение на почтовый ящик <yaro@si.deis.unical.it>.

Пакет GKLS позволяет генерировать функции с фактически неограниченным числом переменных трех классов: класс непрерывно дифференцируемых функций (функции D-типа); дважды непрерывно дифференцируемые функции (функции D2-типа) и класс не дифференцируемых функций (функции ND-типа).

В генераторе, пользователь задает только несколько параметров, определяющих для выбранного типа функции размерность задачи, количество локальных минимумов, значение глобального минимума, радиус области притяжения глобального минимума, расстояние от глобального минимума локальных минимумов, в то время как остальные параметры выбираются автоматически случайным образом.

Для каждого класса функций генератор формирует 100 основных функций произвольного размерности с произвольным числом локальных минимумов. На рисунке 3.1 представлен общий вид тестовых функций.

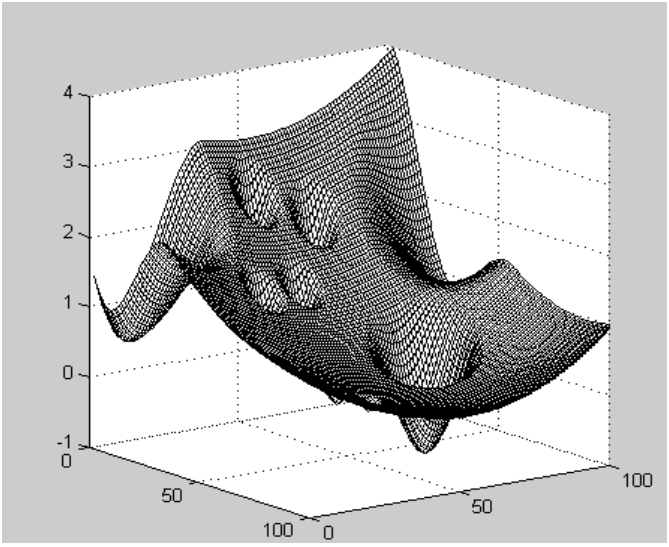

Рис. 3.1. Тестовая функция класса D

Генератор GKLS предоставляет пользователю, в специальном файле, полное описание всех сгенерированных тестовых функций.

4. Задания для курсовой работы

1. Для тестовой функции № 14 двух переменных класса D2, используя модифицированный метод половинных делений и организацию параллельного алгоритма методом деления области поиска на подобласти, найти глобальный максимум функции. Оценить сложность последовательного алгоритма. Разработать параллельную модификацию алгоритма. Оценить эффективность параллельной версии алгоритма оптимизации при $p = 2, 4, 8$.

2. Для тестовой функции № 4 двух переменных класса ND, используя модифицированный метод половинных делений и организацию параллельного алгоритма методом деления области поиска на подобласти, найти глобальный максимум функции. Оценить сложность последовательного алгоритма. Разработать параллельную модификацию алгоритма. Оценить эффективность параллельной версии алгоритма оптимизации при $p = 2, 4$.

3. Для тестовой функции № 63 двух переменных класса ND, используя модифицированный метод половинных делений и организацию параллельного алгоритма методом деления характеристического списка, найти глобальный максимум функции. Оценить сложность последовательного алгоритма. Разработать параллельную модификацию алгоритма. Оценить эффективность параллельной версии алгоритма оптимизации при $p = 2, 3, 4$.

4. Для тестовой функции № 75 двух переменных класса D2, используя модифицированный метод половинных делений и организацию параллельного алгоритма методом деления характеристического списка, найти глобальный максимум функции. Оценить сложность последовательного алгоритма. Разработать параллельную модификацию алгоритма. Оценить эффективность параллельной версии алгоритма оптимизации при $p = 2, 3, 4, 5$.

5. Для тестовой функции № 57 трех переменных класса D2, используя модифицированный метод половинных делений и организацию параллельного алгоритма методом деления характеристического списка, найти глобальный максимум функции. Оценить сложность последовательного алгоритма. Разработать параллельную модификацию алгоритма. Оценить эффективность параллельной версии алгоритма оптимизации при $p = 2, 3, 4, 5$.

6. Для тестовой функции № 63 трех переменных класса ND, используя модифицированный метод половинных делений и организацию параллельного алгоритма методом деления характеристического списка, найти глобальный максимум функции. Оценить сложность последовательного алгоритма. Разработать параллельную модификацию алгоритма. Оценить эффективность параллельной версии алгоритма оптимизации при $p = 2, 3, 4$.

7. Для тестовой функции № 14 четырех переменных класса D2, используя модифицированный метод половинных делений и организацию параллельного алгоритма методом деления области поиска на подобласти, найти глобальный максимум функции. Оценить сложность последовательного алгоритма. Разработать параллельную модификацию алгоритма. Оценить эффективность параллельной версии алгоритма оптимизации при $p = 2, 4, 8$.

8. Для тестовой функции № 14 одной переменной класса D2, используя характеристический метод глобальной оптимизации Стронгина и организацию параллельного алгоритма методом деления характеристического списка, найти глобальный максимум функции. Оценить сложность последовательного алгоритма. Разработать параллельную модификацию алгоритма. Оценить эффективность параллельной версии алгоритма оптимизации при $p = 2, 4, 8$.

5. Пример выполнения курсовой работы

5.1. Постановка задачи

Для тестовой функции № 4 двух переменных класса ND, используя модифицированный метод половинных делений и организацию параллельного алгоритма методом деления области поиска на подобласти, найти глобальный максимум функции. Оценить сложность последовательного алгоритма. Разработать параллельную модификацию алгоритма. Оценить эффективность параллельной версии алгоритма оптимизации при $p = 2, 4$.

5.2. Последовательный алгоритм

В области поиска максимума функции $D_0 = \bigotimes_{i=1}^{2} [0, 1]$, рассмотрим задачу нахождения глобального максимума тестовой функции № 4:

$$f(x) \to \max_{x \in D_0} .$$

Для оценки эффективности параллельных алгоритмов необходимо знать характеристики последовательного алгоритма. С помощью алгоритма модифицированного метода половинных делений и локальной техникой, выбирая из предметной области соответствующие программные модули (объекты), несложно сформировать модель последовательного алгоритма глобальной оптимизации. На рисунке 5.1 представлен последовательный алгоритм глобальной оптимизации.

Модуль 1 настраивает характеристики тестируемой функции. В модуле 2 задаются значения начальных точек, необходимых для запуска модифицированного алгоритма половинных делений. Модуль 3 реализует поиск областей притяжения локальных максимумов. В модуле 4 в текстовый файл выводятся статистические данные по результатам поиска областей притяжения. Модуль 5 организует локальный поиск

максимумов тестируемой функции методом деформированных многогранников. Модуль 6 производит очистку памяти и закрывает рабочие файлы.

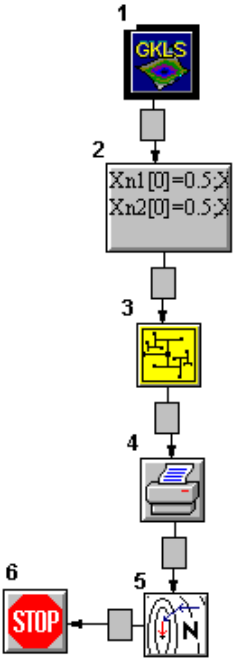

Рис. 5.1. Последовательный алгоритм глобальной оптимизации методом половинных делений

В таблице 5.1 приведено описание модулей алгоритма 5.1.

Уточним правила задания начальных точек для модифицированного метода половинных делений.

Начальные точки, необходимые для «раскрутки» алгоритма глобальной оптимизации, располагаются одна – на границе выделенной подобласти, другая – в ее центре (см. рис. 5.2). При этом если

размещение начальных точек вертикальное, то переменной iNX присваивается значение 2, иначе – 1.

Таблица 5.1. **Объекты алгоритма 5.1**

№	Иконка	Имя актора
1.		Настройка теста GKLS.
2.		Xn1[0]=0.5;Xn1[1]=0; Xn2[0]=0.5;Xn2[1]=0.5; hx=0.25; iNX=2;.
3.		Поиск глобального максимума в частичном квадрате методом половинных делений
4.		fprintf(pstream, "%f; %e; %d;\n",Akfun.A[0][4],Zmax,Nstep); Akfun.A[0][4]=0;.
5.		Локальный поиск максимумов методом деформированных многогранников
6.		Очистка памяти и закрытие файлов.

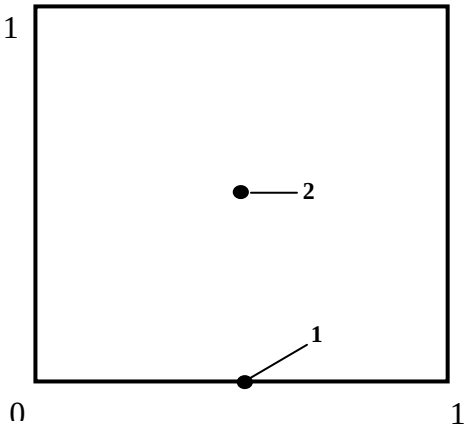

Рис. 5.2. Размещение начальных точек

Компиляция визуального образа программы автоматически установит межмодульные информационные связи. Совокупность данных программного приложения представлены в таблицах 5.2 и 5.3. Обратите внимание на параметры lx и Estron, определяющие глубину просмотра области поиска (рекомендуемые значения 0.032). Идея алгоритма 5.1 состоит в двух этапной организации поиска глобального максимума.

Таблица 5.2. Исходные данные алгоритма 5.1

Имя данного	Нач. значение	Комментарий
hx	0.5	Шаг по x
z2	0	Значение функции
lx	0.032	Величина "зерна" просмотра
NFUNC	24	Номер тестируемой функции
X		Вектор аргументов функции
Hqva	NULL	Голова списка квадратов
Esimp	1.0e-4	Зерно просмотра метода деформированных многогранников
Estron	0.032	Критерий остановки метода Стронгина
HXMAX	NULL	Список приближений локальных максимумов и их точных значений
KlocMax	6	Количество начальных приближений при поиске локальных максимумов
Htmax	NULL	Текущий указатель на список локальных максимумов
GKLtip	3	Тип тестовой функции (1- D; 2 - D2; 3 - ND)
N_fun	4	Номер тесмтовой функции

Первый этап (метод половинных делений) имеет экспоненциальный характер роста сложности, но на нем реализуется грубый поиск зон

притяжения. Второй этап организует, с помощью полиномиального по сложности алгоритма деформированных многогранников, поиск локальных экстремумов.

Таблица 5.3. **Вычисляемые данные алгоритма 5.1**

Имя данного	Нач. значение	Комментарий
Zmax	-1.0e77	Максимум функции
Nstep	0	Число шагов текущего обращения к тернарному дереву
Akfun		Коэффициенты тестовых функций
iNX	NX_-1	Номер координаты направляющего вектора
Xn1		Первое начальное значение вектора для модуля BKvadr2
Xn2		Второе начальное значение вектора для модуля BKvad

Если мы назначим высокую точность решения задачи оптимизации для первого этапа, то потребуется большое количество обращений к функции на этом этапе. Однако точки в областях притяжения окажутся близкими к локальным максимумам, и метод деформированных многогранников найдет их быстро. И наоборот, грубый поиск на первом этапе сократит количество обращений к тестируемой функции и увеличит трудоемкость второго этапа. Поэтому между этапами оптимизации необходимо отыскивать разумный компромисс.

5.3. Параллельный алгоритм делением области на две части

Рассмотрим вариант параллельной модели алгоритма, когда исходная область делится на две части (см. рис. 5.3).

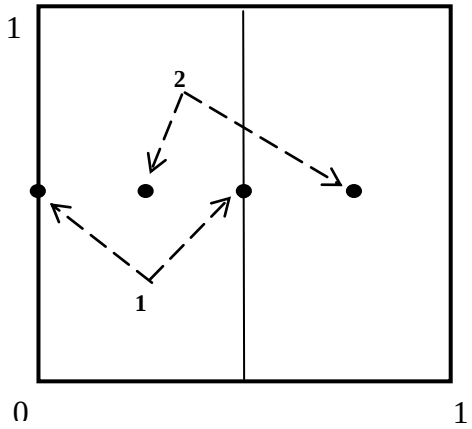

Рис. 5.3. Деление области поиска на две части. Размещение начальных точек

Комбинируя модулями поиска зон притяжения локальных максимумов и поиска локальных максимумов, несложно построить параллельный алгоритм глобальной оптимизации, когда все вычислительные процессы развиваются параллельно в каждой из выделенных зон. На рисунке 5.4 представлен параллельный алгоритм глобальной оптимизации.

В таблице 5.4 приведено описание используемых модулей. В модулях 2 и 3 задаются начальные приближения для алгоритма поиска зон притяжения модифицированным методом половинных делений. Модули 6 и 7 выводят статистические данные результатов поиска. Модуль 8 «разбирает» список представителей локальных зон притяжения, подготавливая, таким образом, параллельные процессы поиска

локальных максимумов с помощью объектов: 9, 10, 11, 12, 13. Модуль 14 производит закрытие файлов и очистку памяти.

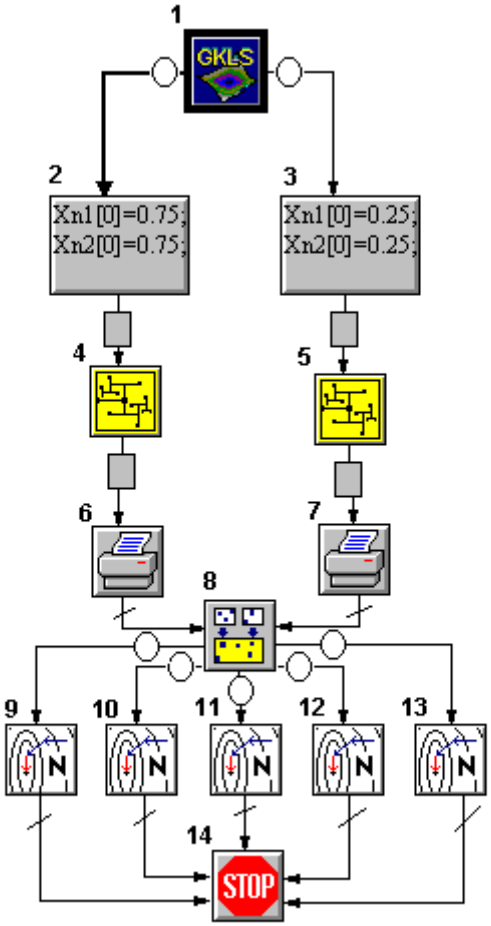

Рис. 5.4 Параллельный алгоритм глобальной оптимизации. Деление области на две части

Таблица 5.4. **Объекты алгоритма 5.4**

№	Иконка	Имя актора
1.		Настройка теста GKLS.
2.		Xn1[0]=0;Xn1[1]=0.5; Xn2[0]=0.25;Xn2[1]=0.5; hx=0.25; iNX=1;.
3.		Xn1[0]=0.5;Xn1[1]=0.5; Xn2[0]=0.5;Xn2[1]=0.75; hx=0.5; iNX=1;.
4, 5.		Поиск глобального максимума в частичном квадрате методом половинных делений
6, 7.		fprintf(pstream, "%f; %e; %d;\n",Akfun.A[0][4],Zmax,Nstep); Akfun.A[0][4]=0;.
8.		Разборка представителей зон притяжения для организации локального поиска максимума функции.
9, 10, 11, 12, 13.		Локальный поиск максимумов методом деформированных многогранников
14.		Очистка памяти и закрытие файлов.

5.4. Параллельный алгоритм делением области на четыре части

Рассмотрим вариант параллельной модели алгоритма, когда исходная область делится на четыре части. Теоретически данный вариант параллельного алгоритма мало чем отличается от алгоритма описанного выше, поэтому ограничимся представлением его визуального образа (см. рис. 5.5).

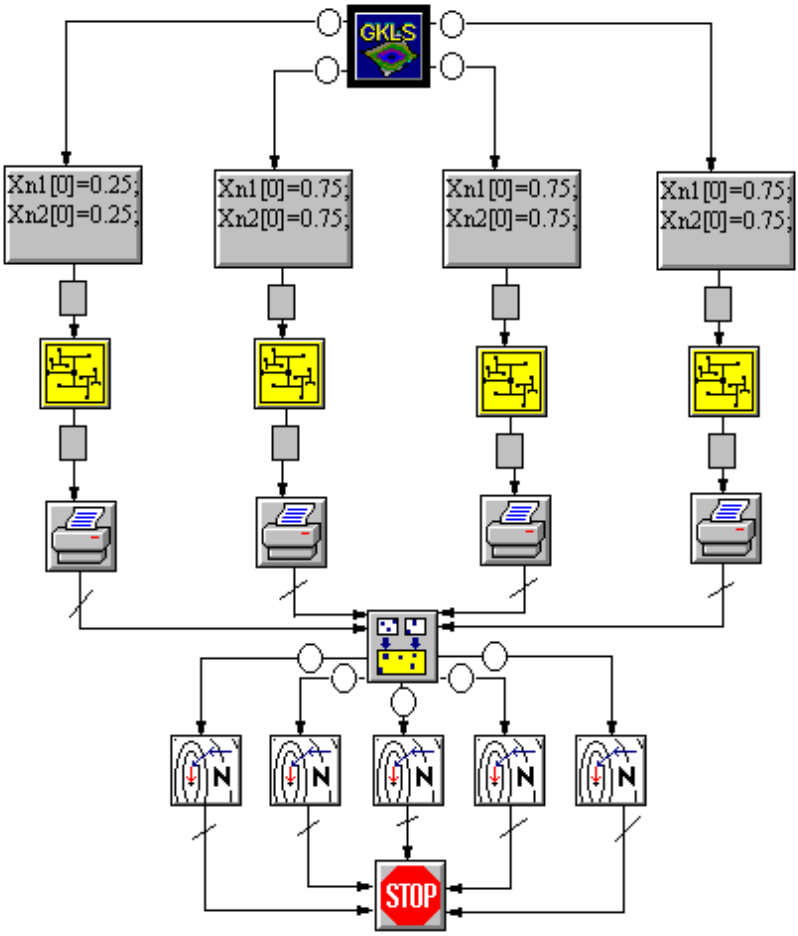

Рис. 5.5 Параллельный алгоритм глобальной оптимизации. Деление области на четыре части

5.5. Вычислительные эксперименты

Результаты вычислительных экспериментов с тестовой функцией № 4 класса ND представлены в таблицах 5.5 - 5.7. Исходная тестовая функция двух переменных имеет 10 локальных максимумов и один глобальный максимум (*f*(0.120569; 0,65789) =1.000). Радиус области притяжения равен 0.3333.

Таблица 5.5. Результаты расчета для последовательного алгоритма

Поиск зон притяжения модифицированным методом половинных делений				
№	**N**	**fmax**		
1	**36**	3,06E-01		
Поиск локальных максимумов из зон притяжения				
№	**Ni**	**fmax**	**Координаты**	
1	64	-1,10E+00	0,731921	0,40732
2	95	3,08E-01	0,446296	0,373863
3	**124**	**1,00E+00**	**0,120569**	**0,657891**
4	33	3,08E-01	0,44625	0,373906
Σ	**316**			

Таблица 5.6. Результаты расчета для параллельного алгоритма (2 области)

Поиск зон притяжения модифицированным методом половинных делений				
№	**Nj**	**fmax**		
1	70	9,55E-01		
2	72	-3,08E-01		
max Nj	**72**			
Поиск локальных максимумов из зон притяжения				
№	**Ni**	**fmax**	**Координаты**	
1	85	-1,06E-07	0,066871	0,329191
2	42	-1,47E-07	0,066777	0,329043
3	42	1,98E-01	0,44625	0,373906
4	33	3,08E-01	0,44625	0,373906
5	**74**	**1,00E+00**	**0,120593**	**0,65805**
max Ni	**85**			

Таблица 5.7. Результаты расчета для параллельного алгоритма (4 области)

Поиск зон притяжения модифицированным методом половинных делений				
№	**Nj**	**fmax**		
1	42	3,06E-01		
2	34	-3,26E-01		
3	30	9,55E-01		
4	52	-1,08E+00		
max Nj	**52**			
Поиск локальных максимумов из зон притяжения				
№	**Ni**	**fmax**	**Координаты**	
1	**124**	**1,00E+00**	**0,120569**	**0,657891**
2	42	-1,47E-07	0,066777	0,329043
3	42	1,98E-01	0,066699	0,197559
4	33	3,08E-01	0,44625	0,373906
5	**74**	**1,00E+00**	**0,120593**	**0,65805**
max Ni	**124**			

Первые, вслед за глобальным, 3 локальных максимумов принимают следующие значения:

f[2] = f(0.44625; 0.373906) = 0.308;

f[3] = f(0.06679; 0.197559) = 0.198;

f[4] = f(0.06677; 0.329043) = 0.000.

Из таблиц 5.6, 5.7 видно, что параллельные алгоритмы глобальной оптимизации находят, кроме глобального, ещё следующие 3 локальных экстремумов.

Результаты оценки эффективности параллельных алгоритмов, вычисляемые по формулам (1.18), (1.19), сведены в таблицу 5.8.

Таблица 5.8. Эффективность параллельных алгоритмов

p	***N1***	***N2***	***NΣ***	***S1***	***E1***	***S2***	***E2***	***SΣ***	***EΣ***
1	36	316	352	1	1	1	1	1	1
2 (5)	72	85	157	0,5	0,25	3,72	0,743	2,24	0,320
4 (5)	52	124	176	0,692	0,173	2,55	0,509	2	0,222

Здесь p – число процессоров; $S1$, $S2$, $S\Sigma$ - ускорения соответственно первого, второго этапов и всего алгоритма. $E1$, $E2$, ES – соответствующие эффективности.

5.7. Анализ результатов вычислительных экспериментов

Из таблицы 5.8 видно, что ускорения вычислений на первом этапе алгоритма не наблюдается. Более того, для первого этапа присутствует существенное замедление (0.5 и 0.69) работы алгоритма модифицированного метода половинных делений и соответственно невысокие значения для показателя эффективности. В качестве причины данного явления можно указать небольшое количество оптимизируемых параметров. Для двухмерной функции еще толком не обозначилась проблема размерности. Если бы мы рассматривали функцию одной переменной, то результат был бы ещё хуже. Данное яывление в какой-то мере подтверждает гипотезу некоторых ученых о том, что в реальности не существует хороших параллельных алгоритмов, а имеются плохие последовательные алгоритмы.

В тоже время, для второго этапа для параллельного алгоритма деления на 2 части и 4 части получилось приличное ускорение. Суммарное ускорение для всего алгоритма показывает, что в параллельном исполнении параллельный алгоритм выигрывает в среднем в два раза, по сравнению с последовательным вариантом.

В итоге, для алгоритмов глобальной оптимизации функций небольших размерностей можно порекомендовать первый этап поиска организовать в последовательном варианте исполнения.

Список литературы

1. 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 // ASM TOMS/ 2003/ 29, № 4. 469-480
2. Гергель В.П. Теория и практика параллельных вычислений. – М.: БИНОМ, 2007.- 423 с
3. Евтушенко Ю.Г., Ратькин В.А. Метод половинного деления для глобальной оптимизации функции многих переменных. // Техническая кибернетика, №1, 1987. с 119-127
4. 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
5. Маслов В.Г., Кузьмичев В.С., Коварцев А.Н., Григорьев В.А. Теория и методы начальных этапов проектирования авиационных ГТД: Учебн. пособие. // Самара, Самар. гос. аэрокосм.ун-т, 1996. - 147 с.
6. Немировский А.С., Юдин Д.Б. Сложность задач и эффективность методов оптимизации. – М.: Наука, 1979. 384 с.
7. Орлянская И.В. Современные подходы к построению методов глобальной оптимизации [Электронный журнал «Исследовано в России», 2007, с.189-192]: [web-сайт] <http://zhurnal.ape.relarn.ru/articles/2002/189.pdf >
8. Стронгин Р.Г. Численные методы в многоэкстремальных задачах (информационно-статистические алгоритмы) - М.: Наука, 1978. – 240 с.