ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(Н И У «Б е л Г У»)

ИНСТИТУТ ИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК

КАФЕДРА МАТЕМАТИЧЕСКОГО И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

ПРИМЕНЕНИЕ ТЕОРИИ ГРАФОВ ПРИ ОПТИМИЗАЦИИ ГРАНИЦ ОТКРЫТОЙ РАЗРАБОТКИ В СИСТЕМЕ НЕДРОПОЛЬЗОВАНИЯ

Магистерская диссертация
обучающейся по направлению подготовки 02.04.01 Математика и компьютерные науки очной формы обучения, группы 07001531
Прудникова Владислава Олеговича

Научный руководитель
к.т.н., доцент
Чашин Ю.Г.

Рецензент
к.т.н., доцент
Заливин А.Н.

БЕЛГОРОД 2017

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На данный момент компьютерные системы предоставляют большие вычислительные мощности. Для этих систем не составляет труда за короткий срок выполнить такой объём вычислений, который был бы не под силу одному человеку. В то время как возрастают вычислительные мощности, растут и сложности задач, которые необходимо решать. К одной из таких задач можно отнести нахождение предельных границ открытых карьеров по добыче полезных ископаемых.

Задача поиска предельных границ открытых карьеров по добыче полезных ископаемых представляет собой важный этап в процессе разработки открытых месторождений полезных ископаемых. С её помощью появляется возможность оценить такие важные вопросы, как экономический потенциал, процесс поиска оптимальной сети транспортных путей, а также расположение отвалов и фабрик по переработке сырья.

Благодаря вычислительным системам, способным обрабатывать большие объёмы информации за короткий промежуток времени, предоставляется возможность ускорить процесс горно-геометрических расчётов. Для достижения данной цели необходимо грамотно подойти к использованию вычислительных возможностей систем, предоставляющих возможность параллельного выполнения задачи. Так как системы, используемые для вычислений, могут отличаться не только мощностями, но и комплектующими, усложняется процесс реализации программы, которая будет работать эффективно для всех систем.

Для правильной организации параллельных вычислений необходима реструктуризация привычного последовательного кода в код с областями, содержащими инструкции для их распараллеливания. То есть, параллельные вычисления проектируются, как системы параллельных взаимодействующих между собой процессов. Стандарты, применяемые для параллельных

вычислений, а также системное обеспечение должны предоставлять возможность:

- создавать параллельные программы;
- обеспечивать синхронизацию между выполняемыми процессами;
- исключать асинхронные вычисления;
- подстраивать вычислительные процессы под различные вычислительные системы.

Таким образом, **объектом исследования является** процесс оптимизации границ открытой разработки запасов в системе недропользования.

В качестве **предмета исследования** выступают геоинформационные модели объектов, разрабатываемых с применением метода открытого карьера и алгоритмы, используемые для осуществления горно-геометрических расчётов, с целью оптимизации границ карьеров по добыче полезных ископаемых в системе недропользования.

Цель работы - реализация программы оптимизации границ открытой разработки запасов в системе недропользования на основе теории графов с применением технологий параллельного вычисления. Исходя из цели, можно выделить задачи, которые необходимо решить:

- исследовать применение теории графов при оптимизации границ открытой разработки запасов в системе недропользования;
- рассмотреть алгоритмы, использующие теорию графов для оптимизации открытой разработки запасов в системе недропользования;
- рассмотреть методы реализации программ, использующих параллельные вычисления.

В первой главе «Обзор и анализ предметной области» рассматривается задача поиска предельных границ карьера открытых месторождений полезных ископаемых, приводится обзор задач и проблем, связанных с добычей рудных тел, а также рассматриваются методы решения поставленной задачи.

Во второй главе «Оптимизация алгоритма Лерча-Гроссмана для работы на

графических процессорах» описываются подходы для разработки параллельных программ, математическая постановка задачи, математическое представление алгоритма; рассматриваются теоретические особенности и программная реализации алгоритма Лерча-Гроссмана на графическом ускорителе с применением технологии параллельного программирования CUDA.

В третьей главе «Программная реализация алгоритма Лерча-Гроссмана для gpu и тестирование» освещаются основные этапы программной реализации рассматриваемого алгоритма, в том числе и параллельные части программы. Также рассматриваются и анализируются результаты тестирования и эффективность реализованного алгоритма на графическом ускорителе.

Данная магистерская диссертация выполнена на 74 страницах, содержит 28 рисунков, 1 приложение, 1 таблицу, 4 листинга, выполнена с использованием 40 источников литературы.

1. ОБЗОР И АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Задача поиска предельных границ открытых месторождений полезных ископаемых

Для добычи твёрдых полезных ископаемых используются открытый, подземный или комбинированный открыто-подземный способы.

Способы добычи полезных ископаемых с использованием метода открытого карьера являются наиболее выгодными, по сравнению с методом подземной добычи, по ряду причин:

- низкая стоимость добычи, по сравнению со способом подземной добычи полезных ископаемых;
- более высокий уровень безопасности;
- скорость добычи полезных ископаемых;
- гибкость добычи.

Наиболее трудоёмким из процессов проектирования горнодобывающего карьера является процесс горно-геометрических расчётов. Данный тип расчётов необходим для оценки и подсчёта всех видов запасов, находящихся в недрах карьера, а также для оптимизации границ карьера, порядка отработки, горно-геометрического анализа карьерного поля и составления календарного плана. В процессе проектирования приблизительно 40% затраченного времени приходится на горно-геометрические расчёты. Однако, на практике, в зависимости от общего числа факторов, обусловленных географическим расположением месторождения, время, затрачиваемое на горно-геометрические расчёты, может достигать 70% от общего времени, затраченного на проектирование карьера.

Современные условия горнодобывающей промышленности вносят свои требования в процесс разработки карьеров: сокращение времени, затрачиваемого на горно-геометрические расчёты, при неизменном высоком уровне

достоверности результатов.

Существует множество методов и алгоритмов для налаживания процесса добычи полезных ископаемых с использованием открытых карьеров. Все они направлены на максимизацию и эффективность добычи полезных ископаемых.

Процесс проектирования открытого карьера по добыче сырья состоит из нескольких шагов:

- геологический анализ;
- исследование механики горных пород;
- моделирование залежей полезных ископаемых;
- формирование дизайна окончательного карьера;
- планирование большого радиуса добычи;
- планирование небольшого радиуса добычи;
- контроль качества добываемого сырья.

На основе полученных знаний формируется блочная модель, состоящая из блоков фиксированных размеров, содержащих в себе рудные тела, а также блоки с пустой породой. Блоки, содержащие в себе полезные породы, имеют положительную ценность, тогда как блоки с пустой породой имеют отрицательную ценность. Сложность также состоит в том, что на каждом этапе, в процессе углубления, блоки перекрывают друг друга, поэтому должны быть строго определены списки блоков, которые добываются первоначально, чтобы карьер приобрёл максимально допустимый контур своих границ, и при этом не произошло обрушений его склонов.

Далее появляется ещё один ряд задач, которые должны быть выполнены при разработке блочной модели:

- инициализировать модель в память ЭВМ;
- указать поверхностную топографию;
- указать геологическую информацию;
- для каждого блока, представленного в модели карьера, назначить его оценку, в зависимости от того, какие ископаемые он в себе содержит.

Благодаря компьютерному моделированию и вычислительным мощностям ЭВМ, существенно сокращается время, необходимое для проведения расчётов и прогнозирования, а также сам процесс построения конечной модели контура открытого карьера.

Теория графов представляет собой отдельный раздел дискретной математики, в котором изучаются свойства графов. Развитие теории графов дал Леонард Эйлер в начале 1763 года. Графы используются при описании связей между объектами. На основе теории графов появились алгоритмы, позволяющие провести анализ блочных моделей предельного контура карьеров. Для обозначения связей в таких алгоритмах используется понятие «узла». Как и в теории графов, узлы служат для установления связей между блоками.

В качестве главной основы планирования всей добычи рудных тел выступает план предельного контура карьера. План предельного контура карьера выступает в качестве помощи в оценке экономического потенциала и начального технико-экономического обоснования месторождения полезных ископаемых. Также он важен для оценки потенциала горно-перерабатывающих предприятий. В процессе расчёта оптимального контура карьера учитываются такие факторы как:

- оценка месторождения;
- рыночные цены;
- пространственное расположение месторождения, то есть оценка распределения рудных тел;
- скорость добычи рудных тел;
- последовательность добычи рудных тел;
- угол наклона вершин внутри карьера;
- исследование механики горных пород;
- моделирование залежей полезных ископаемых;
- стоимость добычи.

Данные факторы тесно взаимосвязаны между собой. Например, суммарное

влияние расходов на добычу руды таково, что более высокие затраты на добычу приведут к меньшему углублению. Общий запас руды, в свою очередь, влияет на скорость добычи, инвестирование капитала и порядок добычи, которые, в свою очередь, могут изменить затраты на добычу, что приводит к круговому процессу оценки добычи (рис.1.1).

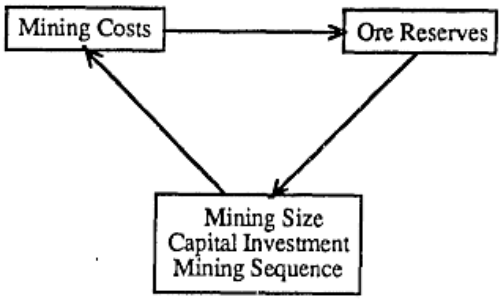

Рис. 1.1 Естественная циркуляция процесса добычи рудных тел

На рис.1.2 изображено сечение или вертикальный слой простой блочной модели:

2	-1	-1	-1	-1	-1	-1	-1
-1	2	2	-1	-1	1	-1	-1
-1	-1	2	3	2	1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1

Рис.1.2. Пример блочной модели

Из рис. 1.2 можно увидеть, что для обозначения ценных блоков используется разноцветная палитра. Блоки, не представляющие ценности, окрашены в голубой цвет.

Если допустить, что угол наклона бортов карьера составляет 45 градусов, то в результате будет получен следующий контур выемки (рис. 1.3).

-1							-1
-1	-1	2		2		-1	-1
-1	-1	-1	-1	-1	-1	-1	-1

Рис. 1.3. Пример контура карьера с углами 45 градусов

Исходя из рисунка, данный контур является правдоподобным, так как он удовлетворяет геотехническим требованиям. Но, при этом, не факт, что данный контур является оптимальным.

Для оптимизации можно произвести усечение блоков, в результате которого наклон бортов карьера будет равен 45 градусам.

В качестве пример карьера с неподходящими бортами можно представить карьер с углом наклона бортов равным 90 градусов (рис. 1.4).

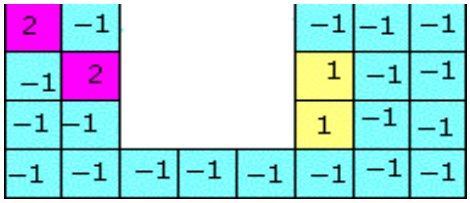

Рис. 1.4. Пример контура карьера с углами 90 градусов

Данный карьер не удовлетворяет техническим требованиям и нарушает выбранный критерий в 45 градусов.

На рисунке 1.5 показано, как должен выглядеть карьер с правильным наклоном и расположением бортов.

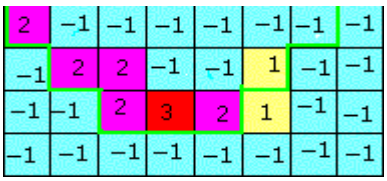

Рис. 1.5. Пример Оптимального контура карьера с углами 45 градусов

На рисунке видна зелёная линия, которая служит для обозначения контура сечения оптимального карьера. Данный контур карьера является оптимальным, и, кроме него, не может быть получен никакой другой контур карьера, который являлся бы оптимальным.

Склоны в карьере формируются постепенно, по мере углубления, в результате добычи нового сырья. Для угла наклона склонов карьера существуют

специальные ограничения, чтобы не произошёл обвал в результате смещения пород под своим весом.

1.2. Обзор существующих методов

Алгоритм сетевого потока для оптимизации контура открытого карьера получил своё развитие в 1968 году. Идеей данного алгоритма является перенос карьера в 3D модель, состоящую из двойственной сети, где каждый блок представляет собой один узел. Блоки с полезными ресурсами помещаются в одну сторону, а фиктивные блоки в другую.

Каждый положительный блок соединён со своим блоком отходов, он обязательно будет переработан перед добычей блока с рудами, так как этого требуют правила построения карьерных откосов. Каждый положительный блок соединён с фиктивным источником, и каждый отрицательный блок соединён с фиктивной раковиной. Направление дуги определяет направление потока. Направление дуг в сети представляется следующим образом:

- все дуги соединяют фиктивные ресурсы и рудные блоки, служат указателями к другим рудным блокам;
- все дуги соединяют блоки отходов и фиктивную точку погружения от блоков отходов к раковине.

Мощность дуги определяется следующим образом. Фиктивный источник имеет неограниченный запас для отправки. Фиктивная раковина имеет неограниченные возможности для приёма. Дуги, соединяющие источник и рудные блоки, имеют такую пропускную способность, соответствующую значениям рудного блока, что потоки с большим значением, чем рудный блок не допускаются. Дуги, соединяющие блоки отходов и рудные блоки, имеют пропускную способность из соответствующих значений блочных отходов, таких, что нет больше потоков, чем блоки отходов могут позволить. Наконец, дуги,

соединяющие рудные блоки и блоки отходов, имеют неограниченную пропускную способность, потому что, если блок отходов представляет собой помеху для добычи рудного блока, он должен быть добыт любым способом, чтобы осуществить возможность добычи рудного блока.

Алгоритм состоит из следующих шагов:

1. Преобразование блочной 3D модели добычи в сетевой поток, как показано на рис. 1.6;

2. Решение проблемы сетевого потока для максимального потока с помощью алгоритма маркировки;

3. Удалить все узлы отходов, дуги которых не соответствуют стандартам сети, вместе со всеми узлами, которые связаны с этими узлами отходов;

4. Узлы в текущей сети образуют блоки в пределах оптимального контура карьера. Сумма остальных значений дуг, соединяющих рудные блоки и источник, являются прибылью от разработки карьера.

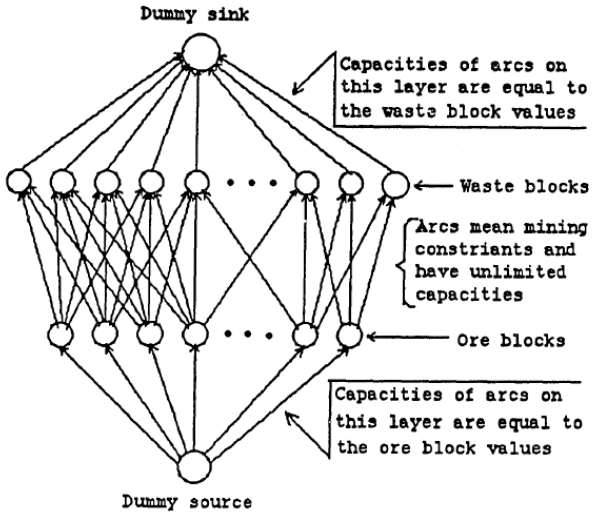

Рис.1.6. Дерево ориентированных графов предельного контура карьера

К проблемам такого метода относятся:

− необходимость в компьютерах с большим запасом памяти и вычислительной мощности;

− трудность перераспределения потока от одного блока к другому.

Типичная проблема проектирования карьера с ограниченным контуром заключается в том, что может понадобиться один миллион узлов для представления одного миллиона блоков. В модели карьера может быть до сотни или тысячи возможных преемников и предшественников, для некоторых узлов сети, которые должны быть представлены направленными дугами. Раковины и потоки должны быть записаны для каждой дуги в сети. Перераспределение чрезвычайно трудно для сети с таким количеством узлов и дуг. Все эти трудности создают большие препятствия для применения алгоритма сетевого потока в проектировании карьера.

Алгоритмы семейства плавающего конуса относятся к эвристическим, то есть способствующим достижению цели в условиях недостаточного количества исходной информации при отсутствии чётких действий.

Существует множество вариаций данного метода, в основном, все они пытаются усовершенствовать оригинальный метод движущегося конуса. Главным преимуществом этих модифицированных методов остаётся симуляция удаления материала в шаге, равном конусу, в алгоритме движущегося конуса. Поскольку не существует каких-либо четких определений предшественников и преемников для каждого блока в модели карьера, такие методы имеют возможность только частично справиться с проблемой совместной поддержки и не имеют возможности решить проблему перераспределения. Каждый из модифицированных алгоритмов по-своему улучшил работу оригинального алгоритма движущегося конуса, однако ни один из них не является оптимальным для проектирования оптимальных границ открытого карьера.

Обязательные шаги в оригинальном алгоритме движущегося конуса:

1. Определить объём добываемой породы, которая будет рассматриваться для потенциальной добычи, накладывая усеченный перевёрнутый конус в модели рудного тела, как показано на рис. 1.7;

2. Суммировать значение всех блоков, центры которых лежат внутри конуса;

3. Удалить все блоки, если их общее количество позитивное;

4. Перемещать основание конуса из рудного блока верхнего выступа к нижнему выступу, до тех пор, пока не найдётся приращение с положительным значением.

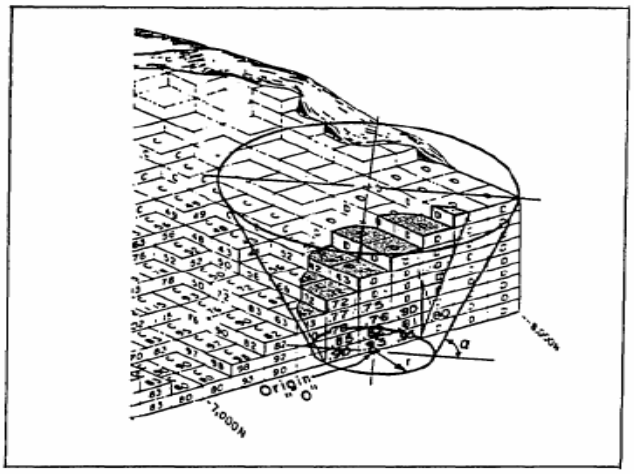

Рис. 1.7. Усеченный перевёрнутый конус в модели рудного тела

За счёт использования концепции конуса, любая форма может быть аппроксимирована с помощью перекрытия конусов, как показано на рис. 1.8 и 1.9.

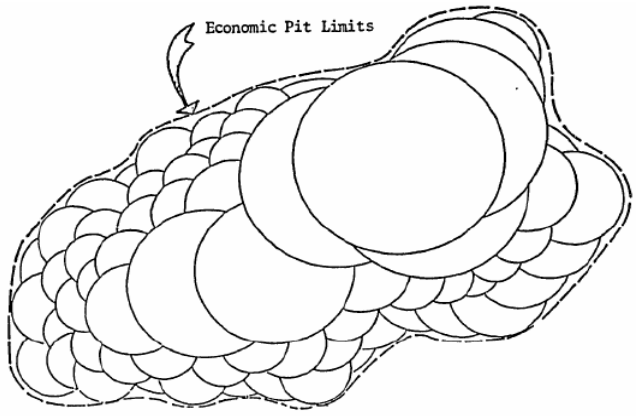

Рис. 1.8. Контур карьера согласно алгоритму движущегося конуса

Из рисунков можно увидеть, что модифицированный способ перемещения конуса имитирует актуальный процесс добычи. Основным недостатком алгоритма движущегося конуса является то, что он не создаёт истинное

оптимальное решение, поскольку он не имеет возможности для обработки совместной поддержки и перераспределения нагрузок.

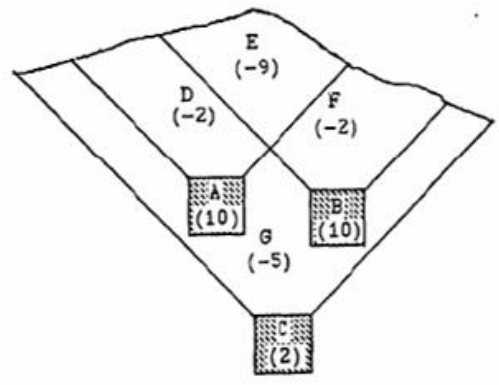

Рис. 1.9. Представление изменений контура карьера на различных этапах обработки алгоритмом движущегося конуса

Рис. 1.9 показывает пример, где метод движущегося конуса не определяет общий вклад и окончание добычи одновременно. Отклонение от нормы может быть довольно большим для рудных тел неправильной или необычной формы.

Из недостатков, мешающих методу получить оптимальные решения, можно составить следующие пункты:

− данный метод оценивает один конус на время, поэтому он не может обнаружить общие залежи;

− чрезмерная добыча полезных ископаемых: данный метод будет добывать все блоки внутри конуса, если конус считается выгодным. То есть, процесс добычи будет затрагивать блоки с пустой породой.

Алгоритм Лерча-Гроссмана использует два типа графов:

1. Граф откосов:

в данном представлении рудные блоки в модели являются вершинами графов. Рёбра служат указателями на другие вершины, которые должны быть добыты раньше, если сохраняются допустимые углы наклона бортов карьера. Граф не претерпевает никаких изменений в процессе работы алгоритма. Цель работы данного алгоритма заключается в нахождении максимального замыкания

данного графа.

2. Граф дерева:

граф дерева претерпевает изменения над собой в процессе работы алгоритма. В данном графе всегда присутствует корневой узел или фиктивный блок. Рёбра данного графа всегда являются рёбрами из графа бортов.

Работа Лерча и Гроссмана 1965 года была первым математическим подходом к проблеме определения оптимального максимального контура карьера. Алгоритм выражается в виде блочной модели внутри специального графа, представляющей из себя ориентированное дерево. Вершины в дереве связываются с блоками, и наложенные направленные дуги формируют из себя ограниченную яму наклона. Эти направленные дуги указывают на связь между блоками отходов, которые должны быть удалены для того, чтобы можно было приступить к добыче конкретного блока руды. Так как любой возможный контур карьера содержится в замкнутом графе, проблема проектирования оптимальных границ открытого карьера сводится к определению закрытого графа с максимальной массой.

Дерево графа может содержать Плюс-ветви (положительные ветви) и Минус-ветви (отрицательные ветви). Плюс-ветви представляют из себя одну связанную дугу, идущую от корня дерева графа, тогда как минус-ветви являются одной связывающей дугой, указывающей на корень дерева графа. Плюс-ветвь является сильной, если она поддерживает общее положительное значение. В противном же случае она слабая. Минус-ветвь является сильной, если она поддерживает нулевое или отрицательное значение. В противном же случае она является слабой.

Нормализация играет ключевую роль в решении проблемы перераспределения в алгоритме. Основными целями нормализации являются:

1) избежать поддержки другими блоками той ветви, общее значение которой положительное;

2) предотвратить поддержку блоков ветвью с отрицательным общим

значением.

На рис. 1.10 видно, что не нужно поддерживать ветвь 1 с помощью дуг arc(10,5), так как общая масса ветви 1 сама по себе уже положительная (+2). Таким образом, нормализация осуществляется путём удаления дуг arc(10,5) и соединение ветви 1 к фиктивному корню X0 дуги arc(x0,5) (рис. 1.11).

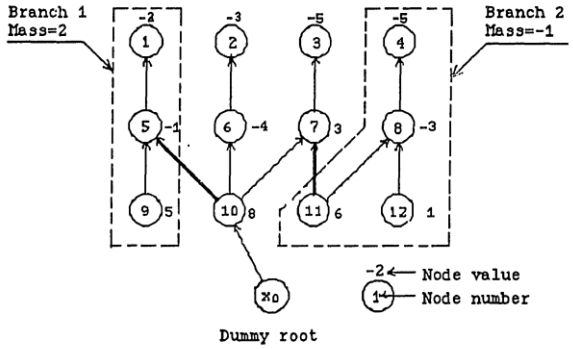

Рис. 1.10. Ненормализованное дерево в графах

Кроме того, она не имеет смысла для поддержки блока 7 ветвью 2 через дугу arc(11,7), так как ветвь 2 сама по себе отрицательная (-1). Таким образом, нормализация должна быть выполнена путём удаления дуги arc(11,7) и соединение ветви 2 с фиктивным корнем X0 дуги arc(x0,11). На рис. 1.11 изображено нормализованное дерево, полученное из графа, представленного на рис. 1.10. В нормализованном дереве каждая сильная ветвь подсоединена к фиктивному корню.

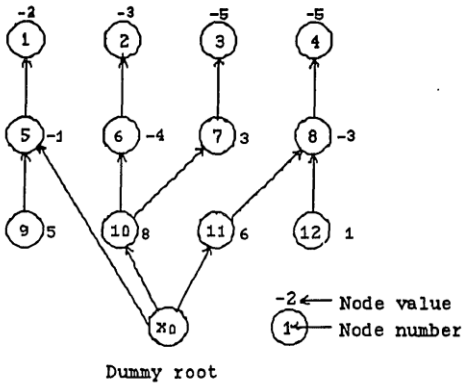

Рис. 1.11. Нормализованное дерево

Алгоритм использует итерационный процесс, который преобразует

нормализованное дерево в новое нормированное дерево. Алгоритм заканчивает свою работу, когда множество сильных ветвей в нормированном дереве становятся замыкающими ориентированного графа.

Алгоритм:

Шаг 1. Инициализация: в дерево добавляется фиктивный узел X0 и будет использован в качестве опорной вершины. Весь граф с самого начала нормализован с помощью подключения фиктивного узла X0 к каждой вершине в модели добычи.

Шаг 2. Поиск: Для вершины X, принадлежащей к сильной ветви, если существует покрывающая вершина Y, принадлежащая к слабой ветви, которая должна быть добыта для раскрытия вершины X (рис. 1.12), определяется корневая вершина Xr сильной ветви, а затем осуществляется переход к шагу 3. В противном случае нужно перейти к шагу 5.

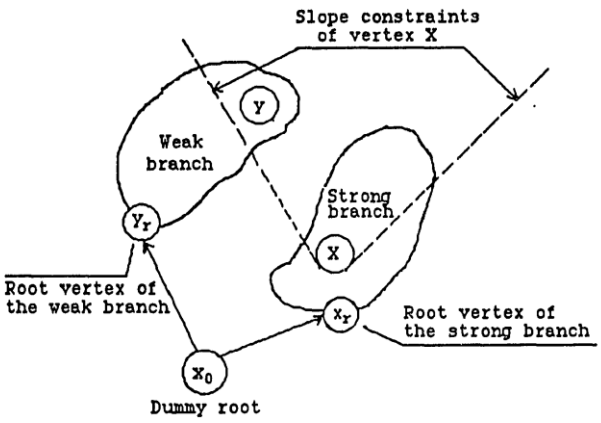

Рис. 1.12. Покрытие вершиной Y со слабой ветвью вершины X с сильной ветвью

Шаг 3. Соединение и преобразование: две ветви, найденные в шаге 2, объединяются в одну с помощью удаления дуги между Xr и фиктивным корнем X0, и присоединением вершины X к вершине Y (рис. 1.13). В дополнение, все дуги в цепи (X,…,Xr) изменили свой статус. То есть плюс-рёбра стали минус-рёбрами и наоборот.

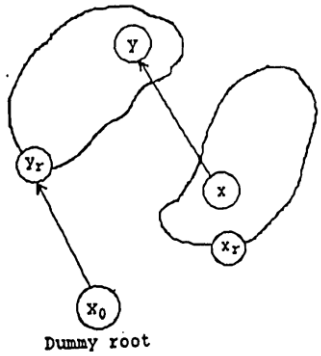

Рис. 1.13. Объединение слабой ветви и сильной ветви

Шаг 4. Нормализация: комбинированная ветвь должна быть повторно оценена для каждого узла в ней. Затем нормализация выполняется для результирующей ветви путём удаления любой сильной дуги из ветви и соединением с фиктивным корнем. Далее происходит возврат к шагу 2.

Шаг 5. Конец итерационного процесса. Максимальное замыкание состоит из всех вершин сильных ветвей конечного нормализованного дерева.

Алгоритм Лерча-Гроссмана – это систематический метод с достижением максимально оптимального контура карьера. Оптимальность и конвергенция были доказаны Лерчем и Гроссманом в 1965 году. К недостаткам данного метода относится сложность в понимании и сложность его программируемости. Процесс нормализации, который разделяется на блоки, которые ранее представляли собой единое целое, требует времени и тяжело реализуем в компьютерной программе. Однако данный метод позволяет максимизировать добычу полезных рудных тел, тем самым повысить экономическую эффективность карьера за счёт использования блочной модели карьера.

1.3. Постановка задачи

На основе обзора и анализа предметной области можно сделать вывод, что проблема оптимизации границ открытой разработки запасов в системе

недропользования является актуальной в настоящее время. Существующие методы возможно улучшить с помощью параллельных вычислений.

По результатам анализа изученной теоретической части был сделан вывод, что в качестве алгоритма, над которым будет производиться распараллеливание, наиболее правильным будет взять алгоритм Лерча-Гроссмана, так как он является самым распространённым стандартом и, основываясь на нём, производятся горно-геометрические расчёты для открытых горнодобывающих карьеров; и разработать эффективную схему его реализации для параллельных вычислений. В качестве технологии параллельной реализации программы была выбрана CUDA от компании NVIDIA. Технология CUDA предоставляет набор расширений для языков C++, C и Fortran, предоставляющих возможность выражать параллелизм данных и параллелизм на уровнях мелких и крупных структурных единиц.

В качестве среды разработки была выбрана Embarcadero Rad Studio, так как она предоставляет широкий набор инструментов и возможность реализовать параллельные вычисления с применение различных методов и технологий.

2. ОПТИМИЗАЦИЯ АЛГОРИТМА ЛЕРЧА-ГРОССМАНА ДЛЯ РАБОТЫ НА ГРАФИЧЕСКИХ ПРОЦЕССОРАХ

2.1. Подходы в реализации параллельных программ

Для разработки параллельных программ применяются несколько технологических подходов, предназначенных для параллельных систем:

- программирование на основе широко распространённых стандартных языков программирования, с применением высокоуровневых библиотек и интерфейсов, для осуществления взаимодействия между частями программы, выполняемыми в различных процессах;
- добавление специальных конструкций, предназначенных для распараллеливания, в код, написанный на стандартном языке программирования;
- применение к последовательным программам средств автоматизированного распараллеливания;
- использование специальных библиотек с конструктивными элементами заранее распараллеленных процедур;
- применение набора инструментальных решений, предназначенных для облегчения процесса создания и проектирования параллельных программ;
- использование специализированных прикладных пакетов;
- создание специализированных языков или расширений, предназначенных для разработки параллельных программ, на основе стандартных и широко распространённых языков, с помощью специализированных конструкций, предназначенных для распараллеливания программ.

Message Passing Interface (MPI, интерфейс передачи сообщений) — программный интерфейс (API), предназначенный для обмена информацией между процессами, направленными на выполнение одной задачи.

MPI - наиболее распространённый из стандартов интерфейса обмена

данными внутри параллельных программ, за счёт существования его реализации для различных компьютерных платформ. MPI используется для реализации параллельных программ для кластеров и суперкомпьютеров. Процессы в MPI взаимодействуют между собой за счёт передачи сообщений между друг другом.

Стандартизацией MPI занимается MPI Forum. В стандарте MPI описан интерфейс передачи сообщений, которого необходимо придерживаться как на платформе, так и в приложениях. Существуют реализации для языков Фортран 77/90, Java, Си и Си++.

Технология MPI ориентирована на системы с распределенной памятью, когда затраты на передачу данных велики, а технология OpenMP ориентирована на системы с общей памятью (многоядерные с общим кешем). Для достижения наиболее оптимального использования кластеров MPI и OpenMP используются совместно.

Приём и передача сообщений является основным способом связи между процессами в MPI. В сообщении содержатся передаваемые данные и информация, на основании которой принимающая сторона производит выборочный приём данных:

- отправитель — ранг (номер в группе) отправителя сообщения;
- получатель — ранг получателя;
- признак — может использоваться для разделения различных видов сообщений;
- коммуникатор — код группы процессов.

Существует два типа операций приёма и передачи данных: блокирующиеся и неблокирующиеся. Для операций, относящихся к неблокирующимся, существуют функции проверки готовности и ожидания выполнения операции.

Ещё один способ связи - удалённый доступ к памяти (RMA), предоставляющий возможность для чтения и изменения области памяти удалённого процесса. Локальный процесс имеет возможность осуществлять

перенос области памяти удалённого процесса (внутри указанного процессами окна) в свою память и обратно, а также комбинировать данные, передаваемые в удалённый процесс с имеющимися в его памяти данными (например, путём суммирования). Операции, осуществляющие удалённый доступ к памяти, не являются блокирующимися, но, до и после их выполнения, возникает необходимость вызывать блокирующиеся функции с целью синхронизации процессов.

Целесообразность распараллеливания программы зависит от сложности поставленной задачи. Существуют риски столкнуться с тем, что на практике время, за которое произойдут вычисления внутри одного процесса, намного меньше, чем время, которое понадобится на инициализацию и передачу данных между процессами. Однако, технология MPI даёт возможность воспользоваться многоядерностью процессоров, а в этом случае скорость передачи данных будет иметь совершенно другие показатели и другой порядок передачи данных. В таких случаях всегда нужно иметь представление об архитектуре и топологии системы.

OpenMP (Open Multi-Processing) – открытый стандарт, предназначенный для распараллеливания программ на языках C, C++ и Fortran. Стандарт несёт в себе описание совокупностей директив компилятора, переменных окружения и библиотечных процедур, которые предназначены для программирования многопоточных программ на многопроцессорных системах с общей памятью.

OpenMP осуществляет параллельные вычисления за счёт многопоточности, в которой главным потоком создаётся набор подчинённых потоков, после чего процесс вычисления распределяется между ними. Выполнение потоков происходит параллельно на машине с несколькими процессорами (количество процессоров не привязано к количеству потоков).

Задачи, которые выполняются потоками параллельно, как и данные, необходимые для их выполнения, описываются с использованием специальных директив препроцессора соответствующего языка – прагм.

Количество потоков, используемых программой, может контролироваться программой самостоятельно за счёт библиотечных процедур, так и извне, с помощью переменных окружения.

Преимущества, которые даёт OpenMP разработчику:

− придерживаясь идеи «инкрементального распараллеливания», OpenMP является прекрасным решением для желающих быстро распараллелить программу с большими параллельными циклами. Программистам не приходится создавать новую программу, вместо этого они добавляют в код программы, согласно инструкциям, OpenMP-директивы.

− OpenMP является гибким механизмом, предоставляющим разработчику большие возможности по контролю поведения параллельных приложений.

− Также, программа, написанная с использование OpenMP, на однопроцессорной платформе может быть использована как последовательная программа, то есть необходимость в поддержании последовательности и параллельности исчезает. Директивы OpenMP игнорируются компилятором, а для вызова процедур OpenMP используются «заглушки» (stubs).

− Также достоинством OpenMP считается поддержка «orphan» (оторванных) директив, то есть директивы, необходимые для синхронизации и распределения выполняемой программы между процессами, могут не входить непосредственно в лексический контекст параллельной области.

Для параллельных вычислений в Embarcadero Rad Studio содержится специальная библиотека PPL (Parallel Programming Library) – библиотека, предоставляющая приложениям возможность параллельного выполнения задач, используя преимущества работы с многопроцессорными устройствами и компьютерами. PPL включает в себя ряд дополнительных функций для запущенных задач, ожидающих задач, группы ожидающих задач и так далее, для обработки программного кода. Для всего этого существует пул потоков, который настраивается автоматически, в зависимости от нагрузки на процессор. Поэтому

нет необходимости в настройке и создании инструкций для потоков. Данная библиотека мультиплатформенная и поддерживающая такие операционные системы, как Windows, MacOSX, Android и iOS. Для распараллеливания программного кода происходит внедрение в код специальных указателей и конструкций, дающих указания библиотеке о необходимости распараллеливания указанной области.

CUDA — представляет собой программно-аппаратную вычислительную архитектуру Nvidia, основанную на расширении языка C (Си), дающую возможность организовать доступ к набору инструкций графического ускорителя и управлять его памятью во время выполнения параллельных вычислительных работ. Использовать технологию CUDA для реализации вычислительных алгоритмов можно на графических ускорителях восьмого поколения и более старших моделях, а также на специально разработанных графических ускорителях семейства Quadro и Tesla.

Сложность реализации программ для вычислений на графических ускорителях с помощью технологии CUDA является довольно высокой, однако, она ниже, по сравнению с ранними реализациями неспециализированных вычислений на графических ускорителях (GPGPU - General-purpose computing for graphics processing units). Для реализации таких программ необходимо производить разбиение приложения между мультипроцессорами, по аналогии с MPI программированием, но при этом не происходит разделения данных, хранящихся в общей видеопамяти. Для использования технологии CUDA для каждого из мультипроцессоров, необходимо хорошее понимание организации памяти. Конечно же, сложность разработки программ на технологии CUDA также зависит и от сложности самой идеи будущей программы.

Для технологии CUDA существует большое разнообразие примеров кода и качественная документация. В основу технологии CUDA входит расширенная версия языка C, для трансляции с которого используется входящий в состав CUDA SDK компилятор командной строки nvcc, который создан на основе

компилятора Open64.

К основным характеристикам языка CUDA относятся:

- является унифицированным программно-аппаратным решением для параллельных вычислений на картах с видеочипами Nvidia;
- содержит в себе большой набор с поддержкой решений, начиная с мобильных до мультипоточных;
- в качестве основы используется стандартный язык программирования C;
- содержит в себе стандартные библиотеки для численного анализа: FFT, для быстрого преобразования Фурье и BLAS для линейной алгебры;
- оптимизированный обмен данными между CPU и GPU;
- возможность взаимодействовать с графическими API OpenGL и DirectX;
- поддержка 32 и 64 битных операционных систем;
- возможность разрабатывать приложения на низком уровне.

Стоит отметить, что среда разработки CUDA (CUDA Toolkit) включает в себя:

- компилятор nvcc;
- библиотеки FFT и BLAS;
- профилировщик;
- отладчик gdb для GPU;
- CUDA runtime драйвер в комплекте стандартных драйверов Nvidia
- руководство по программированию;
- CUDA Developer SDK (исходный код, утилиты и документация).

Модель памяти в CUDA позволяет осуществлять побайтовую адресацию с gather и scatter. Для каждого потокового процессора доступно большое количество регистров, до 1024. Хранить в них можно 32 битные целые числа или числа с плавающей точкой, а доступ к ним очень быстрый.

Каждый из потоков имеет доступ к типам памяти указанным на рис. 2.1.

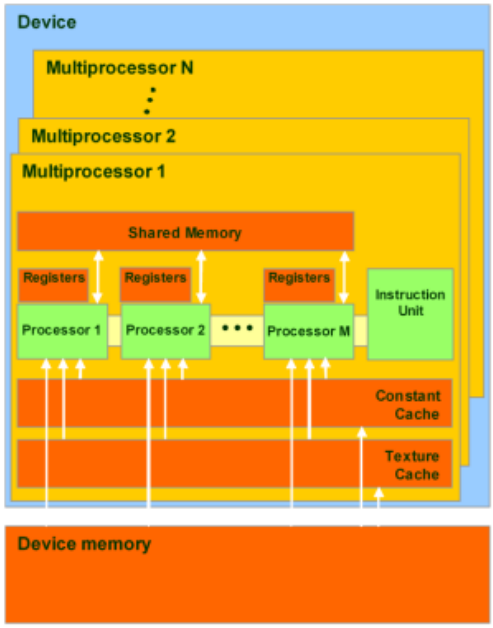

Рис.2.1. Типы памяти доступные потокам в CUDA

Глобальная память является самым большим объёмом памяти, доступным для всех мультипроцессоров на графическом ускорителе, её размер может составлять от 256 мегабайт до 12 гигабайт. Данный тип памяти имеет высокую пропускную способность на топовых решениях от Nvidia, но при этом обладает большими задержками, составляющими несколько сот тактов. Так же этот тип памяти не кэшируется, обладает поддержкой обобщённой инструкции load и store и имеет обычные указатели на память.

Локальная память является небольшим объёмом памяти, доступ к которому имеет лишь один потоковый процессор. Она относительно медленная — такая же, как и глобальная.

Разделяемая память является блоком памяти, размером в 16 килобайт, с общим доступом, доступным всем процессорам в мультипроцессоре. Данная память является весьма быстрой, так же, как и регистры. С помощью неё обеспечивается взаимодействие между потоками, управление доступно разработчику напрямую и обладает низкими задержками. К преимуществам разделяемой памяти относится возможность использовать её как управляемый

программистом кэш первого уровня, снижая задержки при доступе исполнительных блоков (ALU) к данным, тем самым сократить количество обращений к глобальной памяти.

Память констант — область памяти, имеющая объём 64 килобайта, доступ для чтения из которой имеют все мультипроцессоры. Данная область кэшируется по 8 килобайт для каждого мультипроцессора и является довольно медленной – задержка достигает несколько сот тактов, когда в кэше отсутствуют нужные данные.

Текстурная память выступает в виде блока памяти, доступного для чтения всеми мультипроцессорами. С помощью текстурных блоков видеочипа осуществляется выборка данных, в связи с чем предоставляется возможность линейной интерполяции данных с отсутствием дополнительных затрат. Данный блок памяти также, как и константная память, подвергается кэшированию по 8 килобайт для каждого мультипроцессора. Является медленной, как и глобальная память — присутствуют сотни тактов задержек при отсутствующих данных в кэше.

Глобальная, текстурная, локальная и константная память физически являются одной и той же памятью, именуемой как локальная видеопамять видеокарты. Всё отличие заключается в использовании различных алгоритмов кэширования и различных моделей доступа. Центральный процессор имеет возможность обновлять и запрашивать данные только во внешней памяти: глобальной, константной и текстурной.

Технология CUDA, разработанная компанией Nvidia для расчётов на видеоускорителях, отлично подходит для задач с высоким параллелизмом. CUDA доступна для большого количества видеочипов Nvidia, и, несомненно, привносит улучшения в модель программирования на GPU, упрощая её, а также добавляя большое количество возможностей, к которым можно отнести разделяемую память, возможность синхронизации потоков, вычисления с двойной точностью и целочисленные операции.

CUDA — представляет из себя доступную каждому разработчику ПО технологию, которую может использовать любой из программистов, обладающий знаниями языка С. Всего лишь необходимо привыкнуть к несколько иной парадигме программирования, которая присуща параллельным вычислениям. Но, если алгоритм подвергается хорошему распараллеливанию, то затраты на изучение и потраченное время окупятся в многократном размере.

2.2. Математическая постановка задачи

В аналитическом плане задачу по оптимизации границ открытой разработки запасов в системе недропользования можно описать следующим образом:

В качестве примера, за функцию плотности берутся m, v, c, определение которых происходит по всему трёхмерному пространству месторождения.

v (x, y, z) – берётся в качестве себестоимости руды;

c (x, y, z) – означает денежную стоимость разработки пород и руды;

m (x, y, z) = v (x, y, z) – c (x, y, z) – является прибылью от разработки данного объёма.

α (x, y, z) – служит определителем угла в каждой из точек пространства;

S – представляет собой семейство поверхностей, наклон которых ни в одной из точек, по отношению к постоянной горизонтальной плоскости, не должен превосходить α.

V – является семейством объёмов, которые соответствуют семейству поверхности S. Из всего многообразия формирований объёмов необходимо найти тот, который максимизирует интеграл (2.2.1).

$$\int_V m\,(x, y, z)\,dxdydz \qquad (2.2.1)$$

Говоря другими словами, необходимо найти такие границы открытого карьера по добыче полезных ископаемых, при достижении которых будет получена максимальная прибыль.

Чтобы решить задачу по оптимизации границ открытого карьера по добыче полезных ископаемых с применением ЭВМ, используется блочная математическая модель карьера с полезными ископаемыми. Каждый блок данной модели имеет свой вес, который обозначает чистую прибыль, получаемую в результате добычи блока, с учётом прибыльного содержания полезных ископаемых, себестоимости выработки блока и рыночной стоимости полезных ископаемых.

На рис. 2.2 показан пример используемых блочных моделей, красная линия обозначает оптимальную форму карьера.

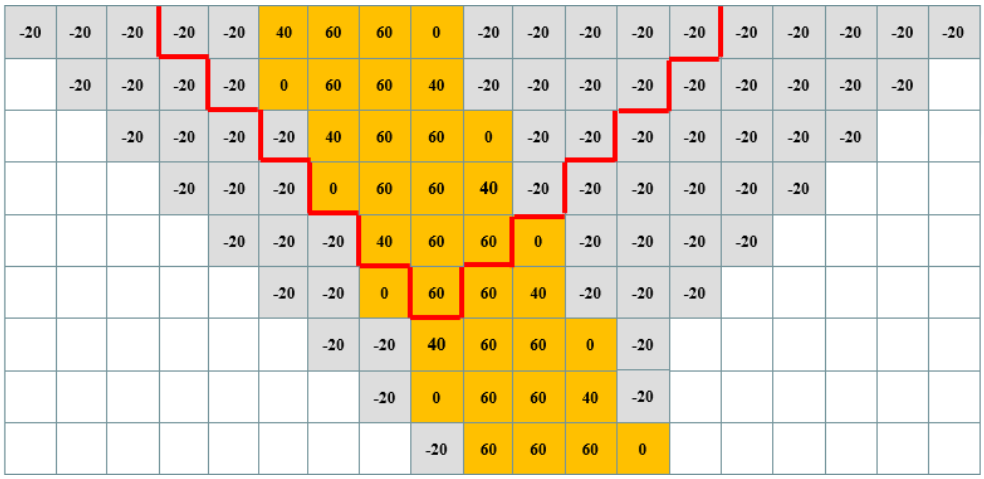

Рис. 2.2 Блочная модель месторождения в двумерном представлении

На рис. 2.2 жёлтыми блоками с положительными значениями веса представлены блоки, содержащие полезные ископаемые, добыча которых экономически выгодна. Серые блоки, имеющие отрицательный вес, представляют собой пустую породу, добыча которой экономически не выгодна предприятию, так как средства на добычу данных блоков тратятся, а экономическая выгода отсутствует.

В данном случае, задача оптимизации границ открытой разработки запасов в системе недропользования сводится к нахождению конечного набора соседних блоков, вес которых в сумме будет давать максимальный результат. При этом на большинство блоков будут накладываться определённые ограничения.

Допустим, что P_{IxJxK} является множеством блоков для исходной стоимостной модели месторождения, тогда как V_{IxJ} является такой матрицей высот, что каждый её элемент $V_{\mathrm{i,j}}$, $\mathrm{i} \in [0, \mathrm{i}]$, $\mathrm{j} \in [0, \mathrm{j}]$ служит показателем глубины карьера в каждой из точек его верхней горизонтальной плоскости.

В таком случае, для того, чтобы учесть ограничения для максимального угла наклона бортов карьера в каждой из точек его поверхности, необходимо ввести дополнительную матрицу $\mathrm{A_{IxJxK}}$, каждый элемент которой: $\mathrm{a_{i,j,k}}$, , $\mathrm{I} \in [0, \mathrm{i}]$, $\mathrm{j} \in [0, \mathrm{j}]$, $\mathrm{k} \in [0, \mathrm{k}]$, демонстрирует максимальную разницу высот для элемента $\mathrm{V_{i,j}}$ и соседствующими с ним элементами: $\mathrm{V_{i+1,j}}$, $\mathrm{V_{i-1,j}}$, $\mathrm{V_{i,j+1}}$, $\mathrm{V_{i+1,j+1}}$, $\mathrm{V_{i+1,j-1}}$, $\mathrm{V_{i-1,j+1}}$, $\mathrm{V_{i-1,j-1}}$.

Как правило, ограничения в исходной модели месторождения на углы наклона бортов задаются не в каждой из точек месторождения, а лишь в нескольких опорных точках. Также указываются координаты точки, значение и направление максимального угла наклона. При этом матрица $\mathrm{A_{IxJxK}}$ получается за счёт аппроксимации исходных данных на каждую из точек блочной модели месторождения полезных ископаемых.

2.3. Математическое представление алгоритма Лерча-Гроссмана

Согласно алгоритму Лерча-Гроссмана, оптимальный контур карьера представляет собой такую блочную 3D модель, которая при добыче руды будет обеспечивать максимальную прибыль, согласно заданным ограничениям крутизны бортов. Вся руда, которую возможно добыть в заданный промежуток времени и получить при этом прибыль, извлекается. При этом из оптимального

карьера нельзя что-то изъять или же, наоборот – добавить, что позволило бы увеличить его ценность без нарушения ограничений на крутизну склонов.

Математический метод поиска работает с двумя типами источников информации:

− экономическая, или ценностная, блочная модель: множество блоков, полностью заполняющих пространство горного отвода;

− список связей между блоками: рёбра и дуги, помечающие связи между блоками в блочной модели.

Если о количестве и качестве руды известно в каждой из точек, то задача поиска сводится к тому, чтобы определить контур карьера и решить, за какое количество стадий возможно его достижение.

Всё пространство карьера проходит через процесс разбиения на совокупность объёмных элементов. Такое разбиение может быть совершено произвольно, к примеру, возможно, будут выбраны единичные объёмы, которые определяются некоторой трёхмерной решёткой. Для каждого объёма v_i присваивается масса m_i, которую возможно определить, используя различные критерии, которые желательно было бы выбрать для оптимизации.

В качестве массы m_i можно указать, например, величину прибыли добычи i – го блока: $m_i = \bar{c}_i - C_i$, где $\bar{c}_i$ является стоимостью добычи на единицу объёма, а C_i – это себестоимость добычи одной единицы объёма.

Процесс разбиения карьера на элементарные объемы происходит в зависимости от структуры карьера и от угла наклона в каждой из точек функции $\alpha(x, y, z)$. В свою очередь, совокупность всех элементарных объёмов рассматривается как совокупность вершин графа G.

Если вершины v_i и v_j соседние, то вершина x_i, которая соответствует элементарному объёму v_i, объединяется с помощью дуги (x_i, x_j) с вершиной (x_j, v_j), другими словами – если для v_i и v_j есть хотя бы одна общая точка, извлечение v_i не представляется возможным до того момента, пока не будет извлечён объём v_i (рис.2.3).

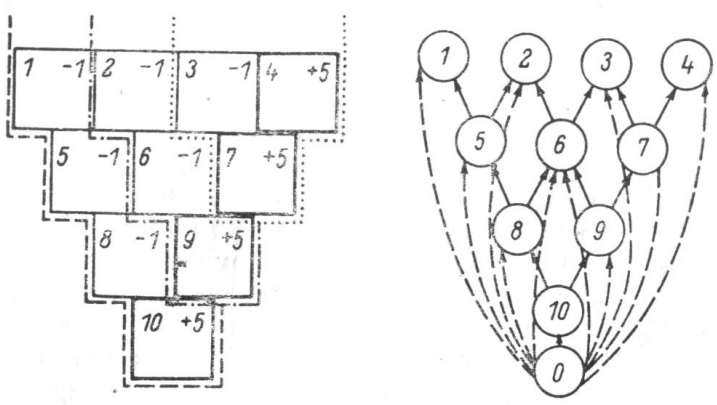

Рис.2.3. Разбиение карьера на элементарные объёмы V_i и соответствующий граф $G(X, A)$.

Все ориентированные дуги, соединяющие один блок (А) с другим блоком (В), указывают на то, что если необходимо добыть блок А, то перед этим необходимо извлечь блок В (рис.2.4).

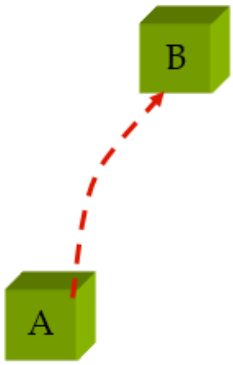

Рис.2.4. Дуга от блока А к блоку В

Обратное утверждение не может являться верным, так как блок В можно извлечь, не прибегая к извлечению блока А. Дуги также характеризуют наклон, который допустим для бортов карьера.

В трёхмерной блочной модели карьера каждый из блоков имеет связь с четырьмя, пятью или девятью блоками. При переносе блочной модели в математическую модель, происходит замена блоков на граф узлов. При этом добавляется корневой узел, служащий в качестве опорной точки (рис.2.5).

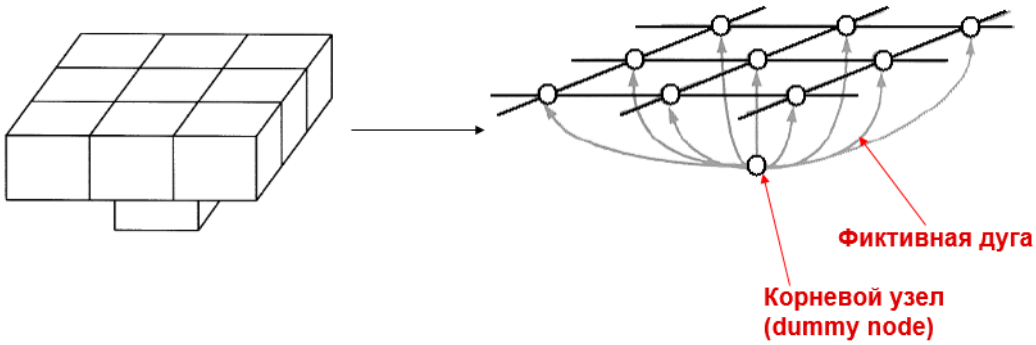

Рис.2.5. Замена блоков на граф узлов

Ориентированный граф $G(X, A)$ задаётся с помощью множества узлов X и множества дуг A. Алгоритм Лерча-Гроссмана находит максимальное замыкание в графе $G(X, A)$ (рис.2.6). Узлом графа G называют $X = \{x_1, x_2, \dots, x_n\}$. A – является множеством дуг графа G. Замыкание графа G определяется как множество таких узлов, где $Y \subset X$. Если $x_i \in Y$ и $(x_i, x_j) \in A$, тогда $x_j \in Y$.

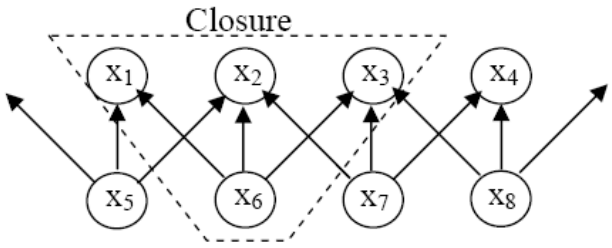

Рис.2.6. Граф модели рудного тела и его замыкание

Корневой узел не имеет предков. Дуга, которая выходит из корневого узла, обладает типом $p(+)$, а дуга, которая идёт к корневому узлу, обладает типом $m(-)$. Дуги, относящиеся к типу p, являются сильными, если они содержат положительную массу $(p - s)$, в обратном же случае, данная дуга является слабой $(p - w)$. Дуги, относящиеся к типу m, являются слабыми, если имеют

положительную массу $(m - w)$, в противном случае они считаются сильными $(m - s)$.

Цепь представляет собой последовательность дуг (рёбер), в которой каждое из рёбер обладает одним узлом, общим со следующим ребром. Цикл – это цепь, в которой совпадают начальные и конечные узлы.

В таком случае, задача определения конечного контура карьера имеет следующую формулировку: имеется направленный граф $G = (X, A)$ и каждая из вершин x_i имеет заданную числовую величину $m_i \geq 0$ или $m_i \leq 0$, которая является массой вершины x_i. Поэтому следующее, что необходимо сделать, это определить подмножество $y \subset G$ с максимальной массой, то есть с максимально возможной прибылью. Подмножество, обладающее максимальной массой, – это максимальное подмножество.

Для рассматриваемой задачи имеется следующая физическая аналогия: на центр каждого блока (рис.2.7) оказывают влияние две силы, одна имеет направление вверх – стоимость руды, содержащейся в блоке; другая – вниз – себестоимость извлечения блока. В виде стрелки для каждого блока изображается результирующая сила.

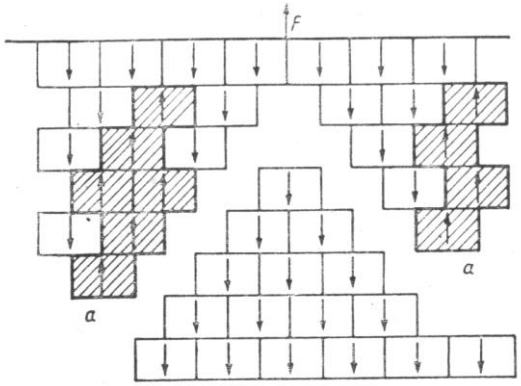

Рис.2.7. Выделение предельного контура карьера при наличии двух рудных тел

Если для системы блоков представляется возможность свободного движения вдоль вертикальной оси, то какую-то из частей блоков можно извлечь. Полная работа равна $F \times 1 = F$, если система совершает полное перемещение на

единицу длины. F является результирующей силой для всех блоков, которые перемещаются. Движение любой механической свободной системы происходит таким образом, чтобы произвести оптимизацию проделанной работы. Исходя из этого, F представляет из себя максимально возможную равнодействующую среди всех соответствующих системам блоков, которые могли бы свободно двигаться вверх (для которых равнодействующая направлена вверх).

Если вернуться к рассматриваемой модели карьера, можно сказать, что будет происходить деление системы блоков на две части вдоль оптимального контура карьера.

Процедура поиска подмножества с максимальной массой в графе G представляет собой итерационный процесс, который состоит в построении последовательности деревьев S^t, которые удовлетворяют каким-то заданным требованиям нормализации.

На первом шаге алгоритма производится процесс соединения всех узлов с корневым фиктивным узлом и обозначение дуг (рис. 2.8).

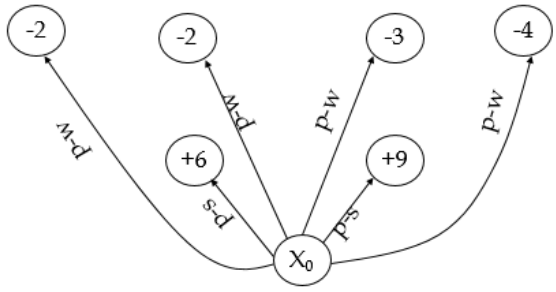

Рис.2.8. Объединение узлов с фиктивным узлом и обозначение дуги

На втором шаге сильные дуги $(p - s)$ связывают положительный узел с остальными узлами, которые необходимо удалить при извлечении сильного узла. После объединения исключается сильный узел, который выходит из узла.

На втором шаге происходит смена меток (рис.2.9).

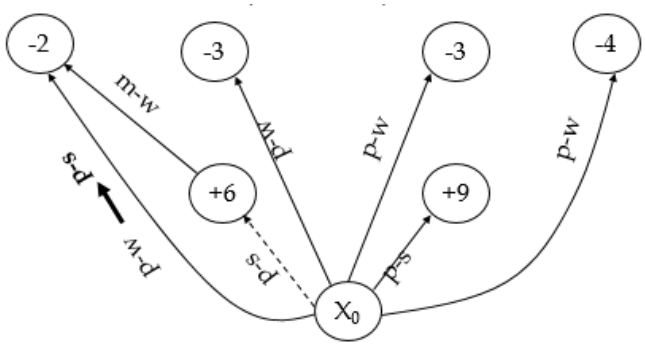

Рис.2.9. Смена меток

На третьем шаге выполняется нормализация дерева (рис.2.10): корень (x_0) - фиктивный корень, который должен быть общим для каждой сильной дуги, иначе есть два варианта работы алгоритма на данном этапе:

– происходит замена сильного ребра $p(x_m - x_n)$ фиктивной дугой $(x_0 - x_n)$, с целью избегания нежелательной поддержки;

– замена сильного ребра $m(x_q - x_r)$ на фиктивную дугу $(x_0 - x_q)$, во избежание поддержки положительной массы такими узлами, которые имеют отрицательную массу.

То есть, таким образом предотвращается процесс расширения карьера.

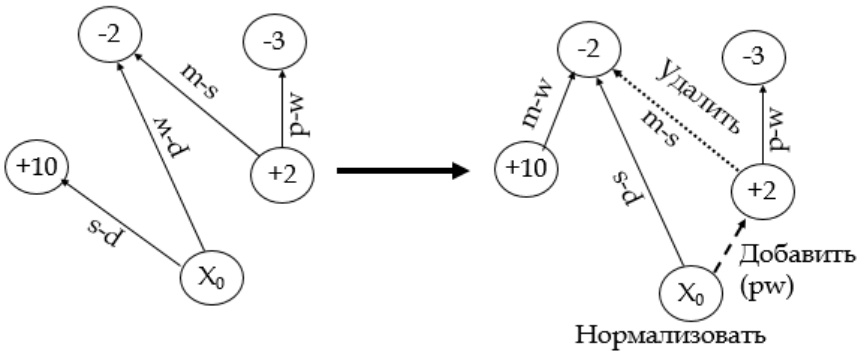

Рис.2.10. Процесс нормализации дерева

В случае, если имеется какой-то из узлов, где вышележащий над ним узел связан с корневым узлом сильной дугой, происходит возврат к шагу два. В противном же случае работа алгоритма прекращается.

Задача определения промежуточных контуров карьера для различных этапов разработки месторождения является более сложной задачей. Для решения данной задачи используется такой подход, в котором к модели карьера применяются дополнительные ограничения. Например, когда нужно максимизировать прибыль в первый год разработки, и имеющееся в распоряжении техническое оборудование позволяется вскрыть объём, не превышающий V. Для этого необходимо рассмотреть функцию $P = M - \lambda V$, где M – будет массой карьера, а V – объём карьера, а $\lambda \geq 0$. Вместо того, чтобы оптимизировать M, будет оптимизирована P. В результате, в алгоритме масса блока m_i заменяется на $m_i - \lambda(m_i' = m_i - \lambda)$. При возрастании λ P будет убывать, однако для достаточно малого приращения λ оптимальный контур и V будут оставаться постоянными.

Если происходит приращение достаточно большой λ, то P будет стремиться к минимальному объёму. Функция λ стремится к ∞, а контур стремится к нулю, функция $V = V(\lambda)$ является ступенчатой функцией. До тех пор, пока V постоянна, P является линейной функцией с угловым коэффициентом – V (рис. 2.11).

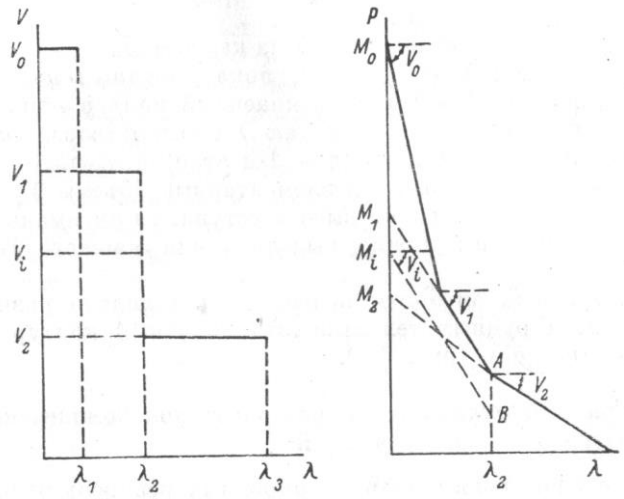

Рис.2.11. Построение кусочно-линейной функции $P = M - \lambda V$

Согласно условию $M = P + \lambda V$, отсюда следует, что значение M, которое

соответствует объёму V, получается за счёт пересечения линией, переходящей через отрезок ломаной P(λ), имеющей наклон V с осью P.

Для каждого из отрезков ломаной P(λ) получается определённое значение M(λ), которое равно максимальной массе, то есть прибыли, которую возможно получить при изменении объёма V. Кривая $M = M(V)$ не может быть полностью построена при таком подходе, тем не менее поведение кривой между полученными точками можно исследовать (рис.2.12).

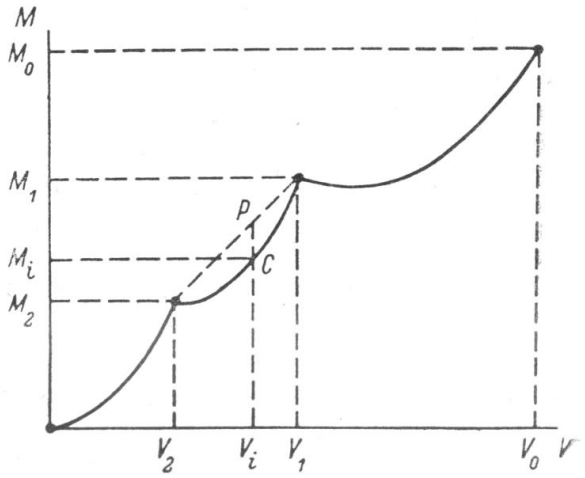

Рис.2.12. Кривая $M = M(V)$

Кривая M(V) будет выпуклой, если находится между парой характеристических точек $(M_1, V_1), (M_2, V_2)$. Можно увидеть, что в действительности значение λ_2 соответствует значению V_i , где $V_2 \leq V_i \leq V_1$. Каждой из точек кривой соответствует промежуточный контур карьера, являющийся оптимальным, если извлекаемый блок точно равен V. Отсюда можно сделать вывод, что в случае, когда не накладываются какие-либо другие ограничения, то рудное тело разрабатывается в соответствии с кривой M(V). И тогда для каждой из точек кривой будет получен оптимальный промежуточный карьер.

Оптимальный карьер, согласно математической модели алгоритма, имеет следующие особенности и свойства:

− алгоритм Лерча-Гроссмана производит отметку каждого блока, помечая находится ли данный блок внутри или вне блочной модели оптимального карьера.

− в трёхмерном виде алгоритма оптимальный карьер обладает наибольшей ценностью. Вычисление прибыли осуществляется по массе руды, качеству и цене продукции.

− зачастую цена представляет собой самую неопределённую величину, но для проектирования карьера необходимо задать какую-то цену.

Изменение углов наклона бортов:

− возможно задать переменные углы наклона бортов для различных частей карьера (для разных уровней или для разных координат X, Y), что приводит к неровным бортам карьера (производится удаление только полных блоков);

− выступы внутри карьера приводят к геомеханическим нарушениям;

− контур оптимального карьера нуждается в корректировке до конечного этапа проектирования карьера.

2.4. Разработка интерфейса ввода-вывода данных

Данные, представляющие из себя блочную модель, хранятся в специальном текстовом формате и имеют свою структуру. К примеру, для блочной модели карьера размером в десять тысяч блоков в файле будут храниться десять тысяч строк. Каждая строка файла, хранящего в себе информацию о блочной модели карьера, содержит в себе данные об одном блоке. В данную информацию входит вес (стоимость) блока и его идентификационный номер (рис.2.13).

X	Y	Z	VALUE
11	0	18	-20
12	0	18	10
41	15	13	5
53	29	5	-11
12	24	3	85

Рис.2.13. Фрагмент исходных данных блочной модели

Программа использует данные из уже готовых блочных моделей, производя построчное считывание из файла и записывая данные в специальную структуру типа: id, x, y, z, data. Где x – это номер блока по оси x, y – номер блока по оси y, z – номер блока по оси z, a data – информация об индивидуальной стоимости блока карьера.

Программа производит загрузку данных из уже имеющихся файлов блочной модели, используя встроенную функцию Initialize. Код, считывающий информацию о положении блока в пространстве и информацию о его стоимости, приведён в листинге 2.1.

Листинг 2.1 Функция Initialize

```
while (pch != NULL)
{
   int tmpNumber =stoi(pch);
   int tmpX = 0;
   int tmpY = 0;
   int tmpZ = 0;
   float tmpData = 0;
   switch(it){
     case 0:
     if(tmpNumber > nxmax){
       nxmax = tmpNumber;
     }
      tmpX = tmpNumber;
     break;
     case 1:
```

```
            if(tmpNumber > nymax){
              nymax = tmpNumber;
            }
             tmpY = tmpNumber;
            break;
            case 2:
            if(tmpNumber > nzmax){
              nzmax = tmpNumber;
            }
             tmpZ = tmpNumber;
            break;
            case 3:
            tmpData = tmpNumberp;
            break;
           }
           val[tmpX][tmpY][tmpZ] = tmpData;
           it++;
          pch = strtok (NULL, " ");
       }
     }
```

Конец листинга 2.1

Для считывания данных используется цикл, записывающий их в трёхмерный массив, так как данные являются трёхмерным представлением карьера. В случае, если файл не найден, будет выведена ошибка с предупреждением об отсутствии файла.

Был также реализован метод, позволяющий сохранять результат работы программы в текстовый файл (листинг 2.2). Данные, хранящиеся в этом файле, представляют собой аналогичную структуру, как и файлы, подающиеся для анализа, с той разницей, что в них хранится структура оптимизированного карьера. Функция сохраняет данные в файл под указанными именем, но с заранее заданным форматом.

Листинг 2.2 Функция сохранения оптимизированного карьера

```
void  write(string FName){
  int br = 0;
  ofstream fout(FName+".block");
  for(int it = 0; it < nxmax; it++){
    for(int ik = o; ik < nymax; ik++){
```

```
      for(int ij = 0; ij < nzmax; ij++){
        fout << val[it][ik][ij]<<" ";
        br++;
        if(br==3){
          fout << "\n";
          br = 0;
        }
      }
    }
  }
  fout.close();
}
```

Конец листинга 2.2

Таким образом, программа осуществляет работу со строго заданной структурой данных, содержащих в себе всю необходимую информацию для оптимизации границ открытой разработки запасов в системе недропользования.

2.5. Разработка алгоритма Лерча-Гроссмана для GPU

Реализация алгоритма Лерча-Гроссмана начинается с того, что данные, которые поступили из текстового файла, переносятся в трёхмерный массив специальной структуры, где индексы массива – это координаты x, y, z, а значения, которые хранятся в под данными индексами – это данные о стоимость блока. Благодаря индексации появляется возможность установить связи между блоками, за счёт чего строится начальное, не оптимизированное дерево графов, после чего над ним будет производиться работа по оптимизации.

Первым этапом, начиная с верхнего уровня блоков, удаляются все блоки с положительным значением (рис.2.14). Эти блоки принадлежат к оптимальному открытому карьеру. Значения этих блоков суммируются в переменную s, которая

является суммой положительных блоков оптимального карьера. Также эти блоки записываются в массив, хранящий в себе модель карьера, через переменную log[i][j][1] и добавляются в контурный массив iplan[i][j].

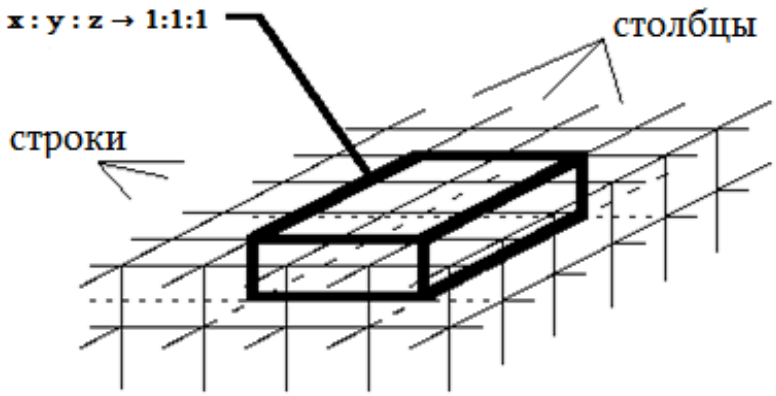

Рис.2.14. Удаление блоков с положительным значением

Далее происходит подключение блоков k-го уровня к фиктивному корню, генерируя таким образом массив itree, который выступает в качестве дерева графов карьера.

Далее происходит процесс создания двух структур:

− структура вершин, содержащая в себе значения экономической ценности для каждого блока в модели;

− структура дуг, содержащая в себе обозначение типа дуги, то есть положительная она или отрицательная; её вес, начало дуги, идущее от корневой вершины, и окончание дуги, находящееся в текущей вершине.

Следующим шагом происходит расчёт общего количества дуг, содержащихся в модели карьера. Для вершины, над которой производится процесс разбора данных, в текущий момент происходит инициализация переменной. Далее происходит проверка выхода за пределы общего количества вершин, которые содержит в себе блочная модель карьера.

Далее осуществляется проверка на принадлежность дуги рассматриваемой вершины к сильной дуге. В случае, если дуга является сильной, ищется слабая

вершина, которая извлекается вместе с текущей сильной вершиной. Началу дуги найденной слабой вершины присваивается адрес рассматриваемой в данный момент сильной вершины, при этом осуществляется перерасчёт веса для данной дуги. Выполняется процесс нормализации ветви, которая располагается между рассматриваемой вершиной и слабой вершиной, которая была найдена. По окончании процесса нормализации осуществляется переход к следующей вершине.

Работа параллельного участка кода начинается с того, что происходит процесс инициализации переменной, которая указывает на рассматриваемую вершину. Рассматриваемая вершина проходит проверку на принадлежность к сильным вершинам, в случае, если вершина сильная, то она помечается, как вершина, пригодная для извлечения. Если же вершина не проходит проверку на принадлежность к сильным вершинам, то происходит процесс удаления структуры вершины, а также структуры дуги. Все отмеченные вершины образуют предельный контур карьера. Схема, отображающая работу алгоритма Лерча-Гроссмана на графическом ускорителе, представлена на рис. 2.15.

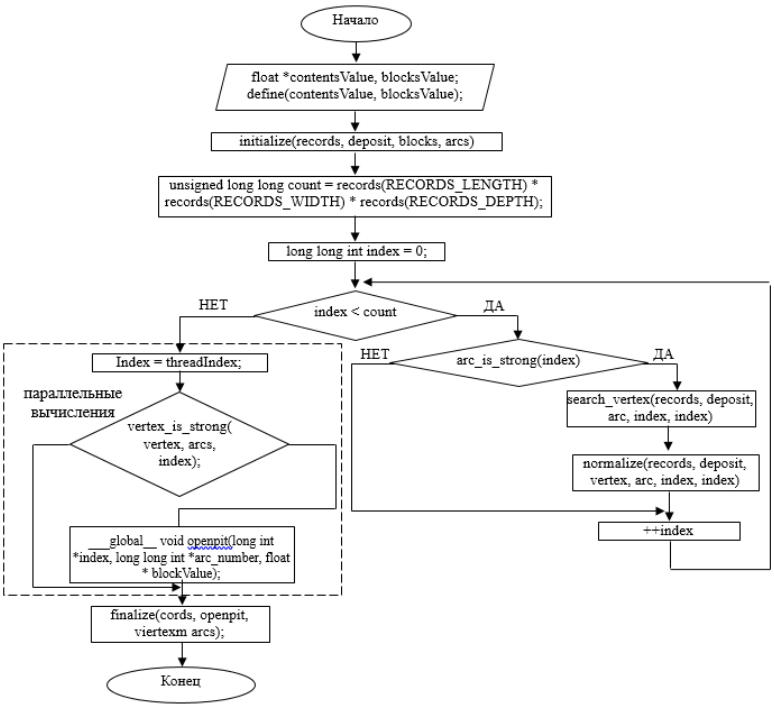

Рис.2.15. Блок-схема алгоритма Лерча-Гроссмана на GPU

Операция поиска оптимальных границ открытой разработки запасов в системе недропользования для открытого карьера возможна с использованием технологии CUDA. Но такой метод накладывает определённые трудности, оказывающие влияние на сложность реализации задачи. Операция поиска оптимальных границ открытого карьера подразделяется на три этапа:

- инициализация и создание дерева графов, на основе подающейся на вход блочной модели карьера;
- поиск и оптимизация связей между вершинами;
- определение типа вершин: сильная или слабая, с последующим удалением или отметкой на извлечение.

3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА ЛЕРЧА-ГРОССМАНА ДЛЯ GPU И ТЕСТИРОВАНИЕ

3.1. Выбор среды разработки программного обеспечения

Embarcadero Rad Studio является средой разработки приложений от компании Embarcadero Technologies. Embarcadero Rad Studio распространяется по платной лицензии и работает в операционной системе Windows от компании Microsoft.

RAD – это концепция, согласно которой ставится цель - создать средства разработки программных продуктов, позволяющих достигнуть наиболее высокой скорости разработки, при этом оставаясь удобными для программирования. Таким образом, программист имеет в своих руках среду разработки, позволяющую добиться наиболее быстрых результатов в разработке программного обеспечения.

В состав Embarcadero Rad Studio входит ряд интегрированных сред разработки:

− C++ Builder – система, предоставляющая программистам возможность разработки программного обеспечения на языке C++. C++ Builder объединяет в себе комплекс объектных библиотек (STL, VCL, CLX, MFC и др.), компилятор, отладчик, редактор кода и многие другие компоненты;

− Delphi – интегрированная среда разработки ПО для Microsoft Windows, Mac OS, iOS и Android на языке Delphi (ранее носившем название Object Pascal), созданная первоначально фирмой Borland и на данный момент принадлежащая и разрабатываемая Embarcadero Technologies.

Microsoft Visual Studio представляет собой линейку продукции фирмы Microsoft, которые включают в себя среду для разработки программного обеспечения и набор специальных инструментальных средств. Данный дистрибутив предоставляет разработчику возможность создавать:

- консольные приложения;
- приложения с графическим интерфейсом;
- веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, .NET Framework и Silverlight.

В состав Visual Studio входит редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего изменения структуры кода. Встроенный отладчик предоставляет возможность работы с исходным кодом и режим отладчика машинного уровня. Другие наборы инструментов упрощают процесс работы с графическим дизайном программы, с классами и дизайном схем баз данных. Visual Studio даёт возможность создания и подключения сторонних плагинов для расширения функциональных возможностей дистрибутива. Например, возможность подключения системы контроля версий или расширение инструментальных средств разработчика.

Visual Studio даёт возможность без дополнительной настройки дистрибутива работать с программами, написанными с использованием программного интерфейса OpenMP. При необходимости работы с MPI, необходимо скачать официальный пакет расширения MPI с сайта Microsoft и настроить дистрибутив Visual Studio для работы с ним.

3.2. Реализация параллельной версии алгоритма Лерча-Гроссмана

Главным фактором при распараллеливании программного кода является его анализ, чтобы понять какие блоки кода необходимо выполнять параллельно, а какие нет. Прежде всего, это зависит от того, над каким объёмом информации будет работать программный код. Для реализации некоторых задач лучше использовать центральный процессор, так как с относительно небольшими объёмами данных он справится намного быстрее.

Ядра центрального процессора задействованы также и в тех случаях, когда необходимо выполнить один поток, содержащий в себе последовательные инструкции для максимальной производительности, а графический ускоритель используется для параллельного выполнения большого числа потоков инструкций. Процессоры графических ускорителей оптимизированы для достижения наиболее высокой производительности одного потока команд, который производит обработку целочисленных значений и значений с плавающей точкой.

Ядра в графическом ускорителе представляют собой одиночный поток команд с множеством потоков данных, выполняющих одинаковые инструкции одновременно. Данный подход программирования является обыденным для алгоритмов, выполняемых на графических ускорителях, однако он требует специфических знаний программирования.

Ядро, использующее возможности параллельного вычисления видеоускорителя, выполняет операции по проверке принадлежности рассматриваемой вершины к сильным или к слабым, а также дальнейшую работу с этой вершиной (листинг 3.1): отсеивание вершины, в случае, если она слабая, или же она помечается, как вершина для извлечения, после чего формируется массив из отмеченных вершин, который представляет собой оптимальный контур карьера. Данная функция использует спецификатор __global__, который обозначает, что функция является началом вычислительного ядра. Выполнение функции осуществляется на видеоускорителе, но её инициализация происходит на хосте – центральном процессоре.

Для синхронизации выполняемых функций используется флаг cudaThreadSynchronize(), блокирующий устройство до тех пор, пока не будут выполнены все предыдущие запрошенные задачи. Если для этого устройства был установлен флаг cudaDeviceBlockingSync, поток хоста будет блокироваться до тех пор, пока устройство не завершит свою работу.

Вызов ядра с использованием __global__ должен соответствовать спецификации. Спецификация служит для определения размерности сети и блоков, которые будут задействованы для выполнения функции на видеоускорителе.

Листинг 3.1 Ядро поиска сильных вершин
__global__ void openpit(long int *index, long int *xnumber, long int *ynumber, long int *znumber, long long int *arc_number, float * blockValue);
Конец листинга 3.1.

Для запуска ядра на графическом ускорителе используется конструкция, представленная в листинге 3.2.

Листинг 3.2 Вызов ядра
openpit <<<index, threads_per_block >> >(modelMatrixSummDevice, model->getPenalty(), points_cfg_array_device, results_device_array, conuses.size());
Конец листинга 3.2.

Параллельная часть алгоритма использует для вычислений большое число отдельных нитей, зачастую каждому из вычисляемых элементов соответствует одна нить. Ядро объединяет в себе все нити, которые задействованы в работе над ним. Также есть массив блоков, который представляет собой одномерный, двумерный или трёхмерный массив нитей. Каждый из блоков является полностью независимым набором нитей, которые взаимодействуют между собой. Если нити находятся в разных блоках, то они не могут осуществлять взаимодействие между собой.

Говоря более простым языком, каждый блок отвечает за независимо решаемую подзадачу. К примеру, если необходимо произвести сортировку какого-то массива, то с использованием определённых методов сортировки возможно разбить один массив на несколько и производить операции сортировки

по отдельности друг от друга. Сортировка одной из частей массива – это задача одного отдельного блока.

Таким образом, производится разбиение исходной задачи на набор подзадач, которые решаются независимо друг от друга. Решение задачи происходит с помощью взаимодействующего между собой набора нитей. Нити выполняют одну и ту же задачу, но со своими данными.

Скорость работы программы зависит от того, какой тип памяти с какой скоростью работает. Для видеоускорителя существует несколько типов памяти:

- регистровая память – это самый быстрый тип памяти из всех имеющихся на видеоускорителе. CUDA не предоставляет возможности явно использовать регистровую память, за размещение данных в ней отвечает сам компилятор;
- локальная память является более медленной, по сравнению с регистровой памятью. Данным типом памяти может воспользоваться компилятор, в тех случаях, когда количество локальных переменных в какой-то функции достаточно большое;
- глобальная память – самая медленная память из доступных на графическом ускорителе. Глобальные переменные выделяются с помощью объявления спецификатора __global__ или с помощью функции cudaMalloc. В основном, глобальная память используется для хранения больших объёмов информации, поступивших с центрального процессора на графический ускоритель.
- разделяемая память – это быстрый тип памяти, в которой удобно хранить локальные переменные функций. Чтобы разместить данные в разделяемой памяти применяется спецификатор __shared__;
- константная память относится к быстрой памяти графического ускорителя. В этот тип памяти возможно записывать данные из центрального процессора и хранить их в виде констант, так как на графическом ускорителе будет возможно только считывать из неё информацию, но не изменять;

− текстурная память – основное предназначение этой памяти в хранении и работе с текстурами – обладает своими специфическими особенностями в адресации.

Функция cudaMalloc используется для выделения памяти на видеоускорителе «cudaMalloc(void**)&openpit, blockValue))».

Копирование данных из центрального процессора на графический ускоритель происходит с помощью функции cudaMemcpy. Копирование данных из центрального процессора происходит в глобальную память.

3.3. Тестирование работы алгоритма

Тестирование осуществлялось на компьютере с 4-ядерным процессором Intel Core i7-4770 3.9 ГГц, 8-ю гигабайтами оперативной памяти, графическим ускорителем Nvidia GTX 760 4 ГБ. Технические характеристики графического ускорителя:

− количество ядер CUDA – 1152;
− частота работы ядер CUDA – 980 МГц;
− быстродействие памяти – 6 Гб/с
− общий объём памяти – 4 ГБ GDDR5.

Для тестирования работы алгоритма использовались готовые модели блочных карьеров minelib, хранящиеся в виде текстовых файлов специального формата. Minelib представляет собой библиотеку общедоступных экземпляров данных для тестирования классических типов проблем по открытой добыче ископаемых: предельный контур карьера и два варианта планирования производства открытым способом. Minelib включает в себя блочные модели и файл приоритетов для добычи. Блочные файлы представляют собой текстовые файлы с информацией о блочной модели, в которых построчно хранится различная информация для каждого из блоков в отдельности. Файлы

приоритетов хранят в себе информацию о блоках, представляющих наиболее важную экономическую ценность для добычи.

Результаты тестирования программы на блочных моделях различных размеров представлены в таблице 3.1.

Таблица 3.1

Результаты тестирования программы

Количество блоков в модели	Затраченное время CPU на одном потоке, мин.	Затраченное время GPU, мин.
14 000	0,38	0,13
50 000	1,37	0,47
100 000	2,75	0,92
600 000	16,5	7,3
1 000 000	27,48	16,87

Из данных, представленных в таблице 3.1, видно, что скорость выполнения параллельной версии алгоритма Лерча-Гроссмана на видеоускорителе, с применение технологии Nvidia CUDA имеет преимущество в скорости работы по сравнению с версией алгоритма для центрального процессора.

ЗАКЛЮЧЕНИЕ

По результатам выполненной магистерской диссертации были решены следующие задачи.

Исследовано применение теории графов при оптимизации границ открытой разработки запасов в системе недропользования. Наиболее трудоёмким процессом является процесс горно-геометрических расчётов, по результатам вычисления которых строится план максимального контура карьера, прокладываются пути грузовых автомобилей и раковины под пустую породу. Весь карьер в памяти ЭВМ представляется как дерево графов со связанными вершинами. За счёт такого представления возможно осуществить поиск связных вершин, которые должны быть добыты одна за другой. В результате вычислений происходит отсев ненужных вершин, и получается конечный контур карьера.

Рассмотрены алгоритмы, использующие теорию графов для оптимизации открытой разработки запасов в системе недропользования. Были рассмотрены алгоритмы движущегося конуса, максимального сетевого потока, Лерча-Гроссмана. Все алгоритмы построены на теории графов и работают с блочными моделями карьеров. Большая часть алгоритмов является усовершенствованными версиями более ранних алгоритмов. От того, какая задача стоит перед добывающей компанией, зависит и выбор алгоритма.

Рассмотрены методы реализации программ, использующих параллельные вычисления. Существует несколько стандартов для реализации параллельных вычислений на ЭВМ: MPI (Message Passing Interface), OpenMP (Open Multi-Processing), Cuda и PPL. Каждый из них предоставляет свой уникальных подход для реализации параллельных программ. Каждый из методов актуален для определённых вычислительных систем, так как требования к системам у стандартов отличаются.

Реализован алгоритм Лерча-Гроссмана для видеоускорителей поддерживающих технологию Nvidia CUDA. Алгоритм работает с готовыми моделями открытых месторождений по добыче полезных ископаемых.

Были проведены вычислительные эксперименты, которые показали, что результат работы программы на центральном процессоре и видеоускорителе совпадает, что говорит о том, что параллельная реализация алгоритма на видеоускорителе имеет такую же эффективность работы, что и на центральном процессоре, однако алгоритм, выполняемый на видеоускрителе, имеет преимущества по скорости.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Zhao, Yixian, Ph.D. Algorithms for optimum design and planning of open pit mines [Текст] / Zhao, Yixian, Ph.D. – The University of Arizona, 1992. – 226 с.

2. Бобровский А.Н. Практическое программирование на C++ [Текст] / А.Н. Бобровский – СПб.: БХВ-Петербург, – 2012. – 496 с.

3. Боресков, А.В., Харламов, А.В. Основы работы с технологией CUDA [Текст] : учебное пособие / А.В. Боресков, А.В. Харламов. – М.: ДМКПресс, 2010. – 232 с.

4. Гергель В.П. Современные языки и технологии параллельного прогаммирования [Текст] / В.П. Гернель - Издательство Московского Университета, – 2012. -408 с.

5. Казённов, А.М. Основы технологии CUDA И OpenCL [Текст] : учебное пособие / А.М. Казённов. – Москва. – 2013. – 67 с.

6. Петин, А.Н., Васильев, П.В. Геоинформатика в рациональном недропользовании [Текст] : монография / А.Н. Петин, П.В. Васильев. – Издательство НИУ БелГУ , 2011. – 268 с.

7. Рейзлин В.И. Численные методы оптимизации [Текст] / В.И. Рейзлин – Томск: Изд-во Томского политехнического университета, – 2011. – 105 с.

8. Скиена, С. Алгоритмы [Текст] : Руководство по разработке / С. Скиена. – 2-е изд.: Пер. с англ. — СПб.: БХВ-Петербург, 2011. – 720 с.

9. Lerchs, H. and Grossmann, I. F. Optimum design of Open-Pit Mines [Текст] / H. Lerchs, I. F. Grossmann // The Canad. Mining and metallurg. Bull. – 1965. – Vol. 58. – № 633. – С. 47–54.

10. Васильев, П.В., Михелев, В.М., Петров, Д.В. Оценка вычислительной сложности алгоритмов оптимизации границ карьеров в системе недропользования [Текст] / П.В. Васильев, В.М. Михелев, Д.В. Петров // Научные ведомости БелГУ. Серия Экономика. Информатика. – 2015. - № 19 (216). – С. 110 – 119.

11. Ерёмин, Д.И., Ягфарова, Н.И. Абишев, Д.А. Алгоритм построения открытого карьера. Алгоритм Лерча-Гроссмана [Текст] / Д.И. Ерёмин, Н.И. Ягфарова, Д.А. Абишев // Достижения и перспективы технических наук. – 2014. – С. 29-33.

12. Ерёмин Д. И., Ягфарова Н. И., Абишев Д. А. Алгоритм Лерча-Гроссмана и его реализация на центральном и графическом процессорах [Текст] / Д. И. Ерёмин, Н. И. Ягфарова, Д. А. Абишев // Современные тенденции технических наук: материалы III Междунар. науч. конф. (г. Казань, октябрь 2014 г.). – Казань: Бук, 2014. — С. 8-13.

13. Капутин Ю.Е. Горные компьютерные технологии и геостатистика [Текст] : учебное пособие / Ю.Е. Капутин – СПб., 2002. – 324 с.

14. Лесонен, М.В., Сень, М.С. Параметры кондиций, предлагаемые к применению при подсчете запасов месторождений твердых полезных ископаемых с использованием блочной модели [Текст] / М.В. Лесонен, М.С. Сень // Проблемы методологического и правового обеспечения экспертизы недропользования. – 2010. – № 3. – С. 84–86.

15. Макаров, И.В., Пронский, В.А. Опыт использования горно-геологической системы Micromine при оценке экономической эффективности отработки Горевского свинцово-цинкового месторождения [Текст] / И.В. Макаров, В.А. Пронский // Журнал сибирского федерального университета. Серия: Техника и технологии. – 2013 – Том 6 – № 4. – С. 374-386.

16. Мигер, К., Димитракопулос, Р., Эйвис, Д. Оптимизация метода проектирования карьера, размера выемочных блоков и проблема межблочного интервала [Текст] / К. Мигер, Р. Димитракопулос, Д. Эйвис // Физико-технические проблемы разработки полезных ископаемых. – 2014. – № 3. – С. 96-117.

17. Васильев П.В. Интеграция параллельных вычислений в системе моделирования и оптимизации рудных карьеров [Текст] / П.В. Васильев //

Информатика: проблемы, методология, технологии материалы XV международной научно-методической конференции. – 2015. – С. 161–164.

18. Васильев П.В., Михелев В.М., Петров Д.В. Применение параллельного алгоритма плавающего конуса для решения задачи поиска предельных границ карьеров [Текст] / Васильев П.В., Михелев В.М., Петров Д.В. // Научные ведомости Белгородского государственного университета. Серия: Экономика. Информатика. – 2016. том. 37. – № 2 (223). С. 101-107.

19. Васильев П.В., Михелев В.М., Петров Д.В. Параллельные алгоритмы оптимизации границ карьеров по методу псевдопотока на модели данных со структурой октодерева [Текст] / П.В. Васильев, В.М. Михелев, Д.В. Петров // Научные ведомости Белгородского государственного университета. Серия: Экономика. Информатика. – 2016. том. 38. – № 9 (230). С. 123-128.

20. Петров, Д.В., Дроник, В.И., Михелев, В.М. Реализация алгоритма Лерча-Гроссмана для поиска предельных границ карьеров рудных месторождений [Текст] / Д.В. Петров, В.И. Дроник, В.М. Михелев // Информатика: проблемы, методология, технологиим сборник материалов XVII международной научно-методической конференции: в 5 т. Секция 6. – 2017. – С. 68-72.

21. Петров Д.В., Михелев В.М. Решение задачи оптимизации блочных моделей при проектировании открытых горных работ с использованием гибридных вычислительных систем [Текст] / Д.В. Петров, В.М. Михелев // Научные ведомости Белгородского государственного университета. Серия: Экономика. Информатика. том. 35. – № 13-1 (210). –2015. – С. 93-98.

22. Петров Д.В., Букреев П.Э., Михелев В.М. Решение задачи поиска предельных границ открытых карьеров на основе параллельного алгоритма плавающего конуса [Текст] / Д.В. Петров, П.Э. Букреев, В.М. Михелев // В сборнике: Параллельные вычислительные технологии (ПаВТ'2016) труды международной научной конференции. – 2016. С. 655-662.

23. Петров Д.В., Михелев В.М. Геоинформационные технологии анализа границ карьеров рудных месторождений [Текст] / Д.В. Петров, В.М. Михелев // В сборнике: Научный сервис в сети Интернет труды XVII Всероссийской научной конференции. ИПМ им. М.В.Келдыша. – 2015. С. 279-284.

24. Петров Д.В., Михелев В.М. Высокопроизводительные алгоритмы решения задачи поиска предельных границ открытых карьеров [Текст] / Д.В. Петров, В.М. Михелев // В сборнике: Актуальные проблемы вычислительной и прикладной математики труды Международной конференции, посвященной 90-летию со дня рождения академика Г. И. Марчука. – 2015. С. 580-584.

25. Петров Д.В., Михелев В.М. Моделирование карьеров рудных месторождений на высокопроизводительных гибридных вычислительных системах [Текст] / Д.В. Петров, В.М. Михелев // В сборнике: Параллельные вычислительные технологии (ПАВТ'2014) Труды международной научной конференции. Ответственные за выпуск: Л.Б. Соколинский, К.С. Пан. – 2014. С. 299-302.

26. Rossi F., McQuay C., So P. Gpu based tlm algorithms in cuda and opencl [Текст] / F. Rossi, C. McQuay, P. So // Applied Computational Electromagnetics Society Journal. Том. 25. – № 4. – 2010. – С. 348-354.

27. Сандерс Дж., Кэндрот Э. Технология CUDA в примерах: введение в програмирование графических процессоров [Текст] / Дж. Сандерс, Э. Кэндрот – М.: ДМК Пресс, – 2011. – 232 с.

28. Стагурова, О.В. Алгоритм Лерча-Гроссмана в задаче определения границ карьера в его предельном положении [Текст] / О.В. Стагурова // Недропользование XXI век. – 2010. – № 6. – С. 38-42.

29. Сухарев А.Г., Тимохов А.В., Федоров В.В. Курс методов оптимизации [Текст] : учебное пособие – 2-е издание / А.Г. Сухарев, А.В. Тимохов, В.В. Федоров – М.:ФИЗМАЛИТ. – 2005. – 368 с.

30. Ческидов, В.И., Саканцев, Г.Г., Саканцев, М.Г. Комплексное обоснование границ карьеров и способов вскрытия глубоких горизонтов при разработке крутопадающих пластообразных залежей [Текст] / В.И. Ческидов, Г.Г. Саканцев, М.Г. Саканцев // Известия высших учебных заведений. Горный журнал. – 2015. – № 2. – С. 17-23.

31. Шариф, Д.А. Регулярная проверка конечного контура рабочей зоны карьера на основе алгоритма скользящего конуса [Текст] / Шариф Д.А. // Горный информационно-аналитический бюллетень (научно-технический журнал). – 2007. – № 6. – С. 352-356.

32. Элкингтон, Т., Дурхэм, Р. Объединение задач определения размера приконтурных блоков и оптимизации производственной мощности карьера [Текст] / Т. Элкингтон, Р. Дурхэм // Физико-технические проблемы разработки полезных ископаемых. – 2011. – № 2. – С. 41-56.

33. Solving ultimate pit limit problem through graph closure (L-G Algorithm) and the fundamental tree algorithm [Электронный ресурс] / SpotiDoc. – Режим доступа: http://www.spotidoc.com/doc/725786.

34. Embarcadero RAD Studio [Электронный ресурс] / Режим доступа: https://ru.wikipedia.org/wiki/Embarcadero_RAD_Studio

35. Microsoft Visual Studio / [Электронный ресурс] / Режим доступа: https://ru.wikipedia.org/wiki/Microsoft_Visual_Studio

36. Валуев, А.М. О моделях и методах оптимизации в задачах проектирования разработки месторождений открытым способом [Электронный ресурс] / Горный информационно-аналитический бюллетень. – Режим доступа: http://www.giab-online.ru/files/Data/2015/02/29_197-206_Valuev.pdf.

37. Введение в OpenMP: параллельное программирование на C++ [Электронный ресурс] / Режим доступа: https://software.intel.com/ru-ru/blogs/2011/11/21/openmp-c

38. Параллельные вычисления CUDA [Электронный ресурс] / Режим доступа: http://www.nvidia.ru/object/cuda-parallel-computing-ru.html

39. Технология Nvidia CUDA — неграфические вычисления на графических процессорах [Электронный ресурс] / Режим доступа: http://www.ixbt.com/video3/cuda-1.shtml

40. Черепанов, Е.В., Макаров, И.В. Фисенко, А.И. Оценка экономической эффективности вовлечения в разработку площадей с прогнозными ресурсами категории Р1, с применением горно-геологических информационных систем [Электронный ресурс] / Сибирский федеральный университет. – Режим доступа: conf.sfu-kras.ru/sites/mn2010/pdf/4/80.pdf.

Приложения

Приложение 1

```
//
// This unit is part of the Geoblock Project, http://sourceforge.net/projects/geoblock
//
// unit uOptimizeLG;

// interface

/*uses
  System.Math,
  Vcl.ComCtrls,
  uGlobals,
  uCommon,
  uInterpol;
  */

#include <stdio.h>
#include <stdlib.h>
#include <cstdlib>
#include <math.h>
#include <iostream>

using namespace std;

/*
 Source code LG23D.F was adopted from paper of P.A.Dowd (1994)
 Lerchs-Grossman method for determining open pit limits

 The deposit is divided into two- and three-dimensional array
 of rectangular blocks and Economic Block Values (EBV) is assigned to each.
 These values are stored in a two- and three-dimensional matrix value
 Val(i, j, k) with dimensions:
   numx - number of rows
   numy - number of columns
   numz – number of levels
 The maximum values of these dimensions are set in a parameter statement
 to nx, ny and nz respectively.
 Log(i, j, k) is a matrix with the same dimensions as val and which is used to indicate
 whether block (i, j, k) is inside (=1) or outside (=O) the pit
 iplan(i, j) has dimensions IK and JK respectively and defines contours of the optimum
 pit by storing the pit level at horizontal location (i, j)

 Other working matrices have dimensions:
 iroot(lkm, 2), itree(nem), ipath(ipkm, 3), nd(nem, 2), d(nem), norm(knrn):
 a sufficient value for each of them is numx * numy * numz /20
  - input file containing Economic Block Values
  - input file containing dimensions of the block model
  - output file of results
*/

const lkmax = 2000;
const nemax = 2000;
const ipkmax = 2000;
const knmax = 2000;
const nxmax = 50;
const nymax = 50;
const nzmax = 13;
```

```
//  arrays with dimensions
 float val[nxmax][nymax][nzmax];
 float log[nxmax][nymax][nzmax];
 float d[nemax]; // contains value of tree
 float iplan[nxmax][nymax];
 int iroot[nemax][2];
 int itree[nemax]; // contains tree number
 int nd[nemax][2];
 int ipath[ipkmax][2];
 int norm[knmax];

function OptimizeLG(var InBlocks: TCoordinateArray;
  var OutBlocks: TCoordinateArray; Params: TInvDistPars;
  ProgressBar: TProgressBar): Boolean;

//=============================================================================
//implementation
//=============================================================================


 int i,j,k;
 int numx,numy,numz; //number of blocks in the x, y and z directions
 int ixdim,iydim,izdim; //dimensions of the blocks for x, y and z axes
 int ires; // element of itree;

/*****************************************************************************)
//Read total values of blocks included in pit
(*****************************************************************************/
void  Initialize(){

  for (i=1; i < numx; i++){
    for (j=1;  j < numy; j++){
      for (k=1; k < numz; k++){
  ifstream file(FName);
    string line;
    while(!file.eof()) {   //объяснение ниже
       getline(file, line);
       char str[256] = {'\0'};
       strcpy(str, line.c_str());

       char * pch = strtok (str," "); // во втором параметре указан разделитель (пробел)
       int it=0;

       while (pch != NULL)                    // пока есть лексемы
       {
          int tmpNumber =stoi(pch);
          int tmpX = 0;
          int tmpY = 0;
          int tmpZ = 0;
          float tmpData = 0;
          switch(it){
            case 0:
            if(tmpNumber > nxmax){
              nxmax = tmpNumber;
            }
             tmpX = tmpNumber;
            break;
            case 1:
            if(tmpNumber > nymax){
              nymax = tmpNumber;
            }
```

```
            tmpY = tmpNumber;
            break;
            case 2:
            if(tmpNumber > nzmax){
              nzmax = tmpNumber;
            }
            tmpZ = tmpNumber;
            break;
            case 3:
            tmpData = tmpNumberp;
            break;
           }

           val[tmpX][tmpY][tmpZ] = tmpData;
           it++;
          pch = strtok (NULL, " ");
       }
      }
  }
}
}
  for (i = 1; i < numx; i++){
    for (j = 1; j < numy; j++){
      iplan[i][j] = 0;
    }
  }

  for (i = 1; i < numx; i++){  //Fortran: do 15 i=1,numx do 15 k=l,numz
    for (j = 1; j < numy; j++){
      for (k = 1; k <numz; k++){
        log[i][j][k] = 0;
        if (val[i][j][k] == 0){
        log[i][j][k] = 1;
        }
      }
    }
  }
}

//*************************************************************************
//  Begin with uppermost level of blocks and remove all positive valued
//  blocks. These blocks belong to optimal open pit: add their values
//  to s, record their inclusion in the pit via log(i j, 1) and add them
//  to the contour array iplan(i,j)
//*************************************************************************
void  LerchsGrossman(){

 float s; //sum of values
 float cpm;
 int i, j, k, l, m, n;
 int la, l1, l2, lg, lk, lkm, lir, lt, ltr, ltc, lar, lsw, lcon;
 int mbs, mbw, mes, mew, md, mdl, mc, mem, ml;
 int na, na1, na2, naf, nc, ne, nl, nel, ny, n1, n2, nf, nn, nod, nodl, ndc, nz;
 int nem, node, nit, nitk, nnd;
 int im, jm, ipkm, ipa, ipk, ipl, ip, iq;
 int knm : Integer;
 int kl, km, kn, ks, kt, kollu, kol, koll;
 int ntk, nts, nds, ndw, ntw;
 float sm;
 /* label 25, 30, 32, 35, 40, 41, 42, 45, 50, 55, 60, 65, 70, 75,
       80, 90, 95, 100, 105, 110, 120, 125, 130, 135, 140, 145, 150, 155,
```

```
      160, 165, 170, 175, 180, 185, 190, 195,
      200, 205, 210, 215, 220, 230, 235, 240, 245,
      250, 255, 260, 265, 270, 275, 280, 285, 290,
      300, 305, 310, 315, 320, 325, 330, 335, 340,
      345, 350, 355, 360, 365, 370, 380, 400, 420, 450;
 */

 /*{sub}*/ void coord(int n, int numx, int numy, int k, int j, int i){
 //  subroutine to determine the array index co-ordinates of node (block)
 //  n given that there are ik (x direction) * jk (y direction) nodes
 //  on each horizontal level
 //  Array index co-ordinates are returned as (i,j,k)

   int ik, kt, jt;
 // begin
   kt = round(n/(numx*numy));
   k = kt+1;
   if (n ==( kt*numx*numy)){
     k=k-1;
   }
   jt = round((n-numx*numy*(k-1))/numx);
   if ((n-numx*numy*(k-1)) == (ik*jt)){
     j = j - 1;
   }
   i = n-numx*numy*(k-1)-numx*(j-1);
 // end;

// begin
  for (i = 1; i < numx; i++){     //do 20
    for (j = 1; j < numy; j++){    // do 20
    begin
      if (val[i][j][1] <= 0){
       break;
       } // goto 20
      s = s + val[i][j][1];
      log[i][j][1] = 1;
      iplan[i][j] = 1;
    }
  }
}
// {20}end; // continue

// Increment the level counter k by 1 and add the blocks on the k-th level
     k = 1;
25:  if (k >= numz){
    goto 400 /*{ print results}*/
    }else{
     // Inc(k); //k++;
      k++;
    }
30:  if(lk >= 0){
  goto 35;
}
32:  ks = k;
      goto 380;
35:  ltr = l;

// Connect blocks on Kth level to the root and establish trees
// itree - contains tree number
// d - contains value of tree
```

```
40:  nts = iroot[ltr][1];
     nds = itree[nts];
     if(d[nds] > 0){
     goto 55;
   }
41:  if (ltr < lk){
     // begin
       ltr = ltr+1;
       goto 40;
     // end; //endif
   }
42:  ltc = l;
45:  if (ltc > lk){
     // begin
       ks = k;
       goto 380;
     // end; //endif
   }
     nts = iroot[ltc][1];
     nds = itree[nts];
     if (d[nds] <= 0){
      goto 50;
    }
     lar = ltc;
     lsw = 4;
     goto 340;

50:  ltc++; //Inc(ltc);          //ltc++;
     goto 45;

55:  lar = ltr;
60:  lsw = l;
     goto 340;
65:  coord(node, numx, numy, kl, j, i);
     if (kl == l){
      goto 350;
     }
70:  ny = (lg - 1)*numx*numy+(n-1)*numx + m;
     for (i = 1; i < lk; i++){
     // begin //do 80
       lir = l;
       ntw = iroot[l][1];
       ndw = itree[ntw];
       if (d[ndw] > 0){
        goto 80;
       }
       ntk = ntw + iroot[l][2] - l;
       for (lt = ntw; lt < ntk; lt++){
       // begin //do 75
          na = itree[lt];
          na1 = nd[na][1];
          na2 = nd[na][2];
          if ((ny == na1) || (ny == na2)){
            goto 95;
          }
75:    }// end;  //continue
80:  }//end;  //continue
     goto 350;
90:  ny = (lg-l)*numx*numy+(n-l)*numx + m;
     cpm = val[m][n][lg];
     log[m][n][lg] = 2;
```

```
    ne++; // Inc(ne);            //ne++;
    nd[ne][l] = 0;
    nd[ne][2] = ny;
    d[ne] = cpm;
    itree[ne] = ne;
    lk++; // Inc(lk);            // ik++;
    iroot[lk][1] = ne;
    iroot[lk][2] = l;
    if(lkm < lk){
     lkm = lk;
    }
    if (nem < ne){
     nem = ne;
    }
    lir = lk;
95:  nd[nds][l] = node;
    nd[nds][2] = ny;
    mbw = iroot[lir][1];
    mew = iroot[lir][2] + mbw - 1;
    mbs = iroot[lar][1];
    mes = mbs + iroot[lar][2] - 1;
    iroot[lir, 2] = iroot[lir][2] + iroot[lar][2];
    iroot[lar, 1] = 0;
    iroot[lar, 2] = 0;
    if (mew + l = mbs){
      goto 100;
    }else{
      goto 140; //? 120;
    }

100:  ires = itree[mbs];
     n1 = mew + l;
     n2 = mbs - l;

     for (n = n1; n < n2; n++){
     // begin //do 105
       nf = n2 - n + n1;
       itree[nf][1] = itree[nf];
     // end;  //continue
    }
105:  itree[mew + l] = ires;
     for (l = 1; l < lk; l++){
     // begin // do 110
       if (iroot[l][l] == 0){
         goto 110;
       }
       if ((iroot[l][l] !> mew) && (iroot[l][1] !< mbs || iroot[l][1] !== mbs)){
      /*{+}*/ goto 110;
    }
       iroot[l][1] = iroot[l][1] + 1;
     // end; //continue
    }
110:  if (mbs == mes){
 goto 140;
}
     mbs = mbs + 1;
     mew = mew+1;
     goto 100;

120:  for (m = mbs; m < mes; m++){
     // begin  //do 135
```

```
      ires = itree[mbs];
      n1 = mbs + 1;
      n2 = mew;
      for (n = n1; n <n2; n++){
      // begin//do 125
        itree[n - 1] = itree[n];
      // end; //continue
     }
125:    itree[mew] = ires;
      mbw = mbw - 1;
      for (l = 1; l < lk; l++){
      // begin  //do 130
        if (iroot[l][l] == 0){
          goto 130;
        }
        if ((iroot[l][l] !> mbs || iroot[l][l] !== mbs) && (iroot[l][l] !< mew || iroot[l][l] != mew)){
        /*{+}*/ goto 130;
      }
        iroot[l][1] = iroot[l][1] - 1;
130:    }//end; //continue
135: } //end; //continue

140:  lcon = l;
     goto 310;
145: //continue
     ipa = ip;
150:  n = ipath[ipa][0];
     if (n == nds){
       goto 155;
     }
     d[n] = d[nds] - d[n];
     ipa = ipath[ipa][2];
     if (ipa != 0){
       goto 150;
     }
155:  lar = lir;
     lsw = 3;
     goto 340;
160:  if (node != ny){
       goto 350;
     }
     ipa = ip;
165:  nn = ipath[ipa][1];
     d[nn] = d[nn] + d[nds];
     ipa = ipath[ipa][2];
     if (ipa != 0){
       goto 165;
     }
170:  kn = 1;
     norm[kn] = lir;
175:  for (kt = l; kt < kn; kt++){
     // begin //do 180
       if (norm[kt] == 0){
         break;
       } //goto 180
       lar = norm[kt];
       lsw = 2;
       goto 340;
     // end; //continue
   }
180:  goto 30;
```

```
185: //continue
     for (ip = l; ip < ipk; ip++){
     // begin //do 190
       if (ip == 1){
       break;
       }   // goto 190
       md = ipath[ip][1];
       nod = abs(ipath[ip][2]);
       if ((ipath[ip][2] < 0) && (d[md] <= 0)){
         goto 195;
       }
       if ((ipath[ip][2] > 0) && (d[md] > 0)){
         goto 195;
       }
     // end;  //continue
    }
190:  norm[kt] = 0;
     goto 175;
195:  nd[md][1] = 0;
     nd[md][2] = nod;
     nodl = nod;
     ipl = ip;
200:  iq = ipath[ip][2];
     mdl = ipath[iq][0];
     d[mdl] = d[mdl] - d[md];
     if (ipath[iq][2] == 0){
       goto 205;
     }
     ip = iq;
     goto 200;

205:  for (iq = ipl; iq < ipk; iq++){
     // begin //do 230
       mc = ipath[iq][1];
       ndc = abs(ipath[iq][2]);
       naf = ipath[iq][2];
       if (ndc == nodl){
         goto 215;
       }
       ip = naf;
210:    if (ip == ipl){
         goto 215;
       }
       ip = ipath[ip][2];
       if (ip < ipl){
         goto 230;
       }
       goto 210;
215:    for (n = nit; n < nitk; n++){
       // begin //do 225
         if (itree[n] != mc){
           break;
         } // goto 225;
         if (n == nitk){
          break;
          } //goto 225
         mem = itree[n];
         n1 = n + l;
         for (nz = n1; nz < nitk; nz++){
         // begin //do 220
           itree[nz - 1] = itree[nz];
```

```
        // end; // continue
         }
220:      itree[nitk] = mem;
         goto 230;
        // end; //continue
      }
    }
     // end; //continue
230:  for (n = nit; n < nitk; n++){
     // begin //do 235
       if (itree[n] == md){
        break;
       } //goto 240;
     // end; //continue
      }
240:  iroot[lar][2] = n - nit;
     // Inc(lk);          //lk++;
     lk++;
     iroot[lk][1] = n;
     iroot[lk][2] = nitk - n + l;
     // Inc(kn);          //kn++;
     kn++;
     norm[kn] = lk;
     if (knm < kn){
       knm = kn;
     }
     if (lkm < lk){
       lkm := lk;
     }
     goto 175;

245:  n = ipath[l][1];
     s = s + d[n];
     for (ip = l; ip < ipk; ip++){
     // begin //do 250
       n = ipath[ip][1];
       nd[n][1]:=0;
       nd[n][2]:=0;
       d[n] = 0;
       node = abs(ipath[ip][1]);
       coord (node, numx, numy, kl, j, i);
       log[i][j][kl] = 1;
       if (iplan[i][j] < kl){
         iplan[i][j] = kl;
       }
     // end; //continue
     }
250:  nel = ne;
     n = 0;
255:  n++; // Inc(n);       //n++;
260:  if (n == ne){
       goto 280;
     }
     if (nd[n][1] == 0){
       goto 265;
     }
     goto 255;
265:  n1 = n;
     n2 = ne - 1;
     for (na = n1;  na < n2; na++){
      // begin //do 270
```

```
        nd[na][1] := nd[na+1][1];
        nd[na][2] := nd[na+l][2];
        d[na] = d[na + 1];
      // end; //continue
       }
270:  ne--; //Dec(ne);      //ne--;
      ml = nel;
      for (m = 1; n < ml; m++){
       // begin //do 275
        if (itree[m] > n){
        itree[m] = itree[m] - 1;
      }
      // end; // continue
      }
275:  goto 260;
280:  if (nd[ne][2] == 0){
       // Dec(ne);     //ne--
        ne--;
      }
      for (n = nit; n < nitk; n++){
      // begin //do 285
        itree[n] = 0;
      // end; //continue
      }
285:  iroot[lar][1] = 0;
      iroot[lar][2] = 0;
      lcon = 2;
      goto 310;
290: //continue
      if (nitk == nel){
        goto 300;
      }
      nl = nitk + 1;
      for (n = nl; n < nel; n++){
      // begin //do 295
        itree[nit + n - nl] = itree[n];
      // end; //continue
      }
300:  for (l = 1; l < lk; l++){
        // begin //do 305
        if (iroot[l][1] < nit){
          break; //goto 305
        }
        iroot[l][1] = iroot[l][1] - mc;
      // end; //continue
      }
305:  goto 42;

310:  l = 0;
315:  l++; //Inc(l);   //l++;
320:  if(l == lk){
        goto 335;
      }

      if(iroot[l][1] == 0){
        goto 325;
      }
      goto 315;
325:  l1 = l;
      l2 = lk - 1;
      for (la = l1; la < l2; la++){
```

```
      // begin //do 330
        iroot[la][1] = iroot[la+l][1];
        iroot[la][2] = iroot[la+l][2];
      //end; //continue
      }
330:  if (lir > l1){
       lir = lir - 1;
      }
      //Dec(lk);     //lk--;
      lk--;

335:  if(iroot[lk][1] == 0){
        // Dec(lk);  //lk--;
          lk--;
      }
      if (lk == 0){
         goto 32;
      }else{
         goto 145; ///? 290, lcon;
      }
340:  nit = iroot[lar][1];
      nc = iroot[lar][2];
      nitk = nit + nc - 1;
      ipk = l;
      nnd = itree[nit];
      ipath[l][0] = nnd;
      ipath[l][1] = nd[nnd, 1];
      ipath[l][2] = 0;
      ip = l;
345:  node = abs(ipath[ip][1]);
      nn = ipath[ip][0];
///?      goto 65, 350, 160, 35O, lsw;
350:  for (n = nit; n < nitk; n++){
      // begin //do 360
        nnd = itree[n];
        if (nnd == nn){
          break; //goto 360
        }
        if (node != nd[nnd][1]){
          goto 355;
        }
        // Inc(ipk); //ipk++;
        ipk++;
        ipath[ipk][1] = nd[nnd][1];
        ipath[ipk][0] = nnd;
        ipath[ipk][2] = ip;
        if (ipkm < ipk){
           ipkm = ipk;
        }
        goto 360;
355:    if (node != nd[nnd][1]){
           break; //goto 360
        }
        // Inc(ipk);  //ipk := ipk + 1;
        ipk++;
        ipath[ipk][1] = -nd[nnd][0];
        ipath[ipk][0] = nnd;
        ipath[ipk][2] = ip;
        if (ipkm < ipk){
          ipkm = ipk;
        }
```

```
    // end; //continue
    }
360:  if (ip == ipk){
        goto 365;
    }else{
      goto 370;
    }
365:  ip++; //Inc(ip);          //ip++;
    goto 345;
370: ///? goto (41, 185, 450, 245), lsw;
380:  im = 0;
    jm = 0;
    sm = 0;
    for (i = 2; i < numx - 1; i++ ){ //do 385
      for (j = 2; j < numy - 1; j++){ //do 385
        if (log[i][j][ks] > 0){
           break; //goto 385
        }
        if (val[i][j][ks] <= 0){
           break; //goto 385
        }
        if (val[i][j][ks] <= sm){
           break; //goto 385
        }
        sm = val[i][j][ks];
        im = i;
        jm = j;
        km = ks;
      }
386: } //{385} //end; //continue
    if (sm == 0){
      goto 25;
    }
    log[im][jm][km] = 2;
    // Inc(ne);       //ne++;
    ne++;
    nd[ne][1] = 0;
    nd[ne][2] = (km - 1)*numx*numy + (jm - 1)*numx+im;
    d[ne] = sm;
    itree[ne] = ne;
    // Inc(lk); //lk := lk+1;
    lk++;
    iroot[lk][1] = ne;
    iroot[lk][2] = 1;
    if (nem < ne){
      nem = ne;
    }
    if (lkm < lk){
      lkm = lk;
    }
    nds = ne;
    ltr = lk;
    goto 55;

/*************************************************************************)
// print results
(*************************************************************************/
400: void  write(){

  int br = 0;
```

```
 ofstream fout("cppstudio.txt");
 for(int it = 0; it < nxmax; it++){
  for(int ik = o; ik < nymax; ik++){
    for(int ij = 0; ij < nzmax; ij++){
      fout << val[it][ik][ij]<<" ";
      br++;
      if(br==3){
        fout << "\n";
        br = 0;
      }
    }
  }
 }
 fout.close();

    kollu=0;
    kol=0;
    koll=0;
    for (k = 1; k < numz; k++){
    for (j = 1; j < numy; j++){
    for (i = 1; i < numx; i++){
    // begin
      if (log[i][j][k] == 1){
        iplan[i][j] = k;
      }
      if ((log[i][j][k] == 1) && (val[i][j][k] < 0)){
         kol = kol + 1;
      }
      if ((log[i][j][k] == 1) && (val[i][j][k] > 0)){
        koll = koll + 1;
      }
      if ((log[i][j][k] ==  1) && (val[i][j][k] != 0)){
        kollu = kollu + 1;
      }
    // end; //405, 410 415 continue
        }
      }
    }

    cout<<"total number of blocks in pitl"<< kollu<<endl;
    cout<<"number of positive blocks in pitl"<<koll<<endl;
    cout<<"number of negative blocks in pitl"<<kol<<endl;
    for i:= l to numx do
    begin
      //Output of result plan;
///    write(6,905) (iplan[i,j], j := 1, numy);
    end;
/*
420: //continue
///   stop;
450:  zz := 1;
///   write(6,900) zz,nem,lkm,node
    close(4)
    close(5)
    close(6)
    stop;
*/
}
} // of LG
```

```
//=================================================================

function OptimizeLG(var InBlocks: TCoordinateArray;
  var OutBlocks: TCoordinateArray; Params: TInvDistPars;
  ProgressBar: TProgressBar): Boolean;

// begin
  LerchsGrossman();
// end;


// end.
```