МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра прикладной математики**

**КУРСОВАЯ РАБОТА**

**ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ РЕШЕНИЯ ЭКОНОМИЧЕСКИХ ЗАДАЧ**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Г.С.Дегтярев

(подпись)

Направление подготовки 01.03.02 Прикладная математика и информатика

Направленность (профиль) Системный анализ, исследование операций и управление (Математическое и информационное обеспечение экономической деятельности)

Научный руководитель

канд. физ.-мат. наук, доц. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Письменский

(подпись)

Нормоконтролер

канд. физ.-мат. наук, доц. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Г.В. Калайдина

(подпись)

Краснодар

2018

**РЕФЕРАТ**

Курсовая работа содержит 29 страниц, 14 рисунков, 8 источников.

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ, ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ, ЭВМ, ЗАДАЧИ

Цель работы: рассмотреть параллельные алгоритмы решения экономических задач, которые позволяют получить какое-либо ускорение за счет равномерного распределения ресурсов.

В данной работе к рассмотрению представлены задачи оптимизации с хорошим потенциалом распараллеливания и задача поиска нахождения оптимального метода решения за счет параллельной проверки методов.

**СОДЕРЖАНИЕ**

[Введение 4](#_Toc533758649)

[1 Параллельные вычисления 5](#_Toc533758650)

[1.1 Общие понятия 5](#_Toc533758651)

[1.2 Области применения параллельных вычислений 11](#_Toc533758652)

[2 Экономические задачи с параллельными вычислениями 14](#_Toc533758653)

[2.1 Поиск метода оптимизации параллельными вычислениями 14](#_Toc533758654)

[2.2 Поиск оптимального решения транспортной задачи 18](#_Toc533758655)

[2.3 Модель региональной экономики на основе социальной стратификации 20](#_Toc533758656)

[Заключение 24](#_Toc533758657)

[Список использованных источников 25](#_Toc533758658)

[Приложение А 26](#_Toc533758659)

## **ВВЕДЕНИЕ**

С появлением в 1960-х годах векторных суперкомпьютеров, зародилось и связанное с ними параллельное программирование. Так называемые контроллеры устройств позволяли выполнять операции ввода-вывода параллельно с инструкциями главного процессора, которые контролировались с помощью прерываний – специальных сигналов, указывающих программе что ей необходимо делать.

Благодаря появлению параллельного программирования довольно просто стало возможным решение многих задач именно благодаря его структуре. Экономические задачи, расчет сложных хаотических систем (погоды, к примеру), криптография, data mining и т.д.

Основной задачей экономики является максимальная прибыль при минимальных затратах. Многие из методов оптимизации являются трудоемкими и, кроме того, имеется возможность частично или полностью распараллелить вычислительный процесс, т.е. выполнять отдельные его блоки (шаги) параллельно и независимо друг от друга с помощью разных процессорных элементов. Благодаря этому можно получить ускорение по сравнению с последовательным алгоритмом решения. Параллельные вычисления помогают решать экономические задачи с оптимальным распределением ресурсов.

## **1 Параллельные вычисления**

Параллельные вычисления – такой способ организации компьютерных вычислений, при котором программы разрабатываются как набор взаимодействующих вычислительных процессоров, работающих параллельно (одновременно). Термин охватывает совокупность вопросов параллелизма в программировании, а так же создает эффективно действующих аппаратных реализаций. Теория параллельных вычислений составляет раздел прикладной теории алгоритмов [1].

### **1.1 Общие понятия**

Существует две модели программирования: последовательная и параллельная. Традиционная ЭВМ была последовательной, т.е. в определенный момент времени только над одни операндом выполнялась только одна операция.

Основная же концепция параллельных вычислений, как раз, реализуется распараллеливанием процесса обработки данных. Программа использующая параллельные вычисления характеризуется как набор взаимодействующих вычислительных процессов.

Существует две парадигмы параллельного вычисления : параллелизм данных и параллелизм задач (data parallel и message passing). В основе этих подходов лежит идея распределения вычислительной работы по процессорам доступным пользователю. Параллелизм данных характеризуется:

* одна программа управляет обработкой всех данных;
* пространство имен является глобальным (детали скрыты от программиста);
* на параллельных процессах происходит слабая оптимизация вычислений;
* на всех доступных данной программе процессорах параллельные операции выполняются одновременно.

Т.е. для реализации такого способа от программиста не требуется больших усилий, достаточно подключить определенные библиотеки рассчитанные на архитектуру ЭВМ.

Для параллелизма задач требуется разбить искомую на несколько подзадач для загрузки процессора собственной. Программист может контролировать процесс обмена данными и распределение нагрузки на процессоры.

Параллелизм задач характеризуется:

* минимизация обмена данными между задачами;
* повышение трудоемкости программы и ее отладки;
* повышенная опасность возникновения тупиковых ситуаций;
* ответственность за разгрузку процессоров лежит на программисте.

Перед тем как перейти к классам ЭВМ давайте уточним способы решения задач поставленных перед ЭВМ методами параллельных вычислений. Можно выделить два основных метода: конвейер и параллелизм.

Конвейер можно представить как программу состоящую из отдельных блоков. После прохождения объекта первого блока он переходит на второй, при на этом уже на первый блок поступает новый объект. Таким образом можно заметить что конвейер это программа разбитая на блоки через которые последовательно одновременно выполняются определенные блоки и проходят данные. Для эффективной реализации конвейера должны выполнятся следующие условия:

* система выполняет повторяющуюся операцию;
* эта операция может быть разделена на независимые части;
* трудоемкость подоперации примерно одинакова.



Рисунок 1 – ЗУ – Запоминающее устройство; Пр – процессор.

Глубина конвейера–количество подопераций в конвейере. Обязательным условия должно являться независимость данных подаваемых на конвейер. Если же данное условия не выполняются и существуют операнды, зависящие от предыдущих, то в конвейере образовываются так называемые “пузыри”. Это еще одна проблема в работе конвейерных систем. Увеличение быстродействия которое может дать конвейер рассчитывается формулой:

$\frac{n\*d}{n+d}$ (1)

Где n-количество операндов, загружаемых в конвейер, а d-глубина конвейера. Допустим у нас есть 400 операндов, которые необходимо обработать за 5 операций, тогда ускорение составит (400\*5)/(400+5)=4.9 и только в случае идеальной ситуации, которая исключает пузыри и т.д., практически нереализуема в реальной жизни.

Параллельное решение задачи предусматривает распределение выполняемых команд и операндов на несколько процессоров.



Рисунок 2 – ЗУ – Запоминающее устройство; Пр –процессор.

Главной проблемой такой структуры являются информационная зависимость, операнды и команды выполняемые на процессорах должны либо не зависеть друг от друга, либо нужно организовать синхронную работу процессоров для получения корректных данных на выходе [2].

Далее рассмотрим классификацию Флинна. Рассмотрим архитектуру вычислительного оборудования по потокам данных и команд.

Существуют четыре основных класса:

1. SISD;
2. MISD;
3. SIMD;
4. MIMD.

В первом случае (SISD – single instruction single data) представлена последовательная вычислительная машина. Одна команда выполняется один раз над одним элементом данных. Сюда относятся все машины с одним процессором и одним комплектом памяти.

Далее идет MISD (multiple instruction single data) – множественный поток данных выполняется над одним элементом данных. Архитектура с единообразной памятью. Сюда относятся многоядерные машины, машины с конвейерными архитектурами процессоров, симметричные многопроцессорные, многопроцессорные графические адаптеры.

SIMD (single date multiple instruction) один поток команд выполняется над множеством элементов данных. Этот тип характерен для векторных машин и для машин с параллельными данными. В структуре можно выделить множество вычисляющих устройств. Сопоставление команд происходит векторным компилятором. Данный класс приспособлен к однообразным вычислениям.

Множественный поток команд и множественные операции над элементами (MIMD – multiple data multiple instruction). Возможно выполнение множественных команд множественным потокам данных. Выполнение задачи обеспечивается синхронизацией и обменом данными между вычислительными узлами. Сюда относятся кластеры и грид-сети в режиме обработки [3].

Далее перейдем к такому понятию как кластер, в общем случае кластер – набор компьютеров, объединенных некоторой коммуникационной сетью. Каждый вычислительный узел (компьютер) работает под управление своей операционной системы и имеет свою оперативную память. Самым популярным является использование однородных кластеров, в этом случае все узлы (компьютеры) абсолютно одинаковы по своей архитектуре и производительности.



Рисунок 3 – Кластер

В каждом кластере имеется выделенный компьютер – головная машина (front – end). На ней имеется программное обеспечение, которое управляет запуском программ. На каждый вычислительный процесс запущенный пользователем приходится не более одного процессора (происходит распределение). Запускать на главной машине кластера вычислительные процессоры нельзя.

Т.к. входить на узлы кластера для пользователей нет необходимости, им предоставляется терминальный доступ на головную машину кластера. Запуск же программ осуществляется в "пакетном" режиме – пользователь не имеет интерактивного взаимодействия с программой, программа не ожидает ввода данных с клавиатуры и не выводит непосредственно на экран. Программа может работать даже когда пользователь не подключен к кластеру.

Вычислительный кластер обычно работает под управлением многопользовательской многозадачной сетевой ОС (ОС Unix). Unix отличается от Windows, которая обычно работает на ПК, в частности это касается работы с процессами файловой системы и интерфейса с пользователем. На файл-сервере находится общая файловая система, к которой имеют доступ все узлы кластера.

Вызывать в своих программах параллельные библиотеки является следующим способом задействования вычислительного кластера. Существуют библиотеки которые помогают решать широкий круг стандартных подзадач (линейная алгебра и т.д.). Если обращение к таким подзадачам является основной частью программы, то фактически, это позволяет получить свою параллельную программу без написания собственного кода. Создание же собственных параллельных программ является самым трудоемким, но и наиболее универсальным способом. Существует два основных варианта:

1. В имеющиеся параллельные программы вставлять параллельные конструкции;
2. создать свою параллельную программу "с нуля" [4].

В отличие от кластеров технология SMP (Симметричные мульти процессоры) имеет другую архитектуру. Здесь система состоит из массива общей памяти (несколько независимых блоков) и нескольких однородных процессоров. Все процессоры имеют доступ к общей памяти с одинаковой скоростью. Подключение процессоров к памяти происходит либо с помощью общей шины, либо с помощью crossbar-коммутатора. Аппаратно поддерживается когерентность кэшей. Благодаря наличию общей памяти упрощается взаимодействие процессоров, но при этом накладывается ограничение на их число. Для построения масштабируемых систем на базе SMP используются кластерные или NUMA-технологии. Система работает под управлением единой операционной системы, которая автоматически (в процессе работы) распределяет по процессорам процессы/нити, при этом иногда возможна явная привязка. Для SMP-систем существуют сравнительно эффективные средства автоматического распараллеливания [5].

### **1.2 Области применения параллельных вычислений**

Существуют пять основных стилей параллельного программирования которые будут рассмотрены в данной работе:

1. рекурсивный параллелизм;
2. итеративный параллелизм;
3. клиенты и серверы;
4. производители и потребители;
5. взаимодействующие равные.

Рекурсия используется в программе с одной или несколькими рекурсивными процедурами. Применяется рекурсия в императивных языках программирования. Рекурсивный параллелизм часто используется для решения комбинаторных проблем: сортировка, планирование (задача коммивояжера) и игры (шахматы и др.).

Итеративный параллелизм основывается на распределении нагрузки при решении задачи на все процессоры. Каждый процесс может содержать один или несколько циклов, т.е. каждый процесс можно назвать итеративной программой. Синхронная работа в данном стиле осуществляется за счет пересылки данных или разделения переменных. Реализуется чаще всего на нескольких процессорах для научных вычислений.

Клиенты и серверы – наиболее распространенная система в распределенных системах, от локальных сетей до WWW (World Wide Web). Сервер ожидает запрос от клиента, в то время как он делает запрос к серверу, затем сервер продолжает действовать в соответствии с командами клиента. Сервер можно реализовать как многопоточную программу (при необходимости параллельной обработки запросов) или как одиночный процесс, который не может обрабатывать несколько клиентских запросов.

Такой стиль программирования представляет собой параллельное программное обобщение процедур и их вызовов: клиенты вызывают процедуру роль которой выполняет сервер. Обычный вызов процедуры нельзя использовать если код сервера и код клиента расположены на разных машинах. В таком случае необходимо использовать удаленный вызов процедуры или рандеву.



Рисунок 4 – Клиенты и серверы

Производители и потребители. Взаимодействующие процессоры которые часто организуются как конвейер через который проходит информация. Каждый процесс можно рассматривать как фильтр, который принимает информацию от предшественника и передает ее последующему процессу. Каждый процесс является так называемым фильтром. Такие фильтры на уровне приложений (оболочек) встречаются в операционных системах (Unix), внутри самих операционных систем, внутри прикладных программ, если один процесс читает входные данные, которые производит другой.

Рисунок 5 – Производители и потребители

Тип взаимодействующих равных встречается в распределенных программах, в которых несколько процессоров выполняют один и тот же код и обмениваются сообщениями.



Рисунок 6 – Взаимодействующие равные

Взаимодействующие равные используются при реализации распределенных параллельных программ, особенно при децентрализованном принятии решений и итеративном параллелизме [6].

## **2 Экономические задачи с параллельными вычислениями**

### **2.1 Поиск метода оптимизации параллельными вычислениями**

Для формализации задачи параллельных вычислений будем применять теория графов. Исходный алгоритм содержит n элементарных операций $x\_{1}$… $x\_{n}$ и m переходами между ними $h\_{1}$…$h\_{m}$. Представим такой алгоритм в виде графа Z=(X,U), он является ориентированным и циклическим. Где H – множество дуг, X – множество вершин графа. Представим граф Z с помощью матрицы смежности A, которая содержит элементы $a\_{ij}$=1, если дуга принадлежит графу и 0, если дуга не принадлежит. Данный алгоритм будет выполнятся в системе состоящей из q процессоров P=($p\_{1}$…$p\_{q}$). Допустим в момент времени k задается $f\_{k}$=($x\_{i}$,$ p\_{i}$), в котором номер процессора $p\_{i}$ ставится в соответствие операция $x\_{i}$. К $f\_{k}$ предъявим следующие требования:

1. Параллельно выполняются только операции без переходов ($a\_{ij}$=0);
2. Для операции $x\_{i}$ к моменту ее выполнения k все необходимы данные должны быть известны;
3. Процессор $P\_{i}$ в момент времени k может выполнять только одну операцию.

t($x\_{i})$-полное время обработки операции. Понятно что время выполнения параллельного алгоритма T, т.к. разные операция требуют разного времени обработки:

T=$\max\_{i}t(x\_{i})$ (2)

При выборе лучшей схемы распараллеливания вычислений можно добиться уменьшения времени выполнения. При v заданных схемах $T\_{1}$…$T\_{v}$ оптимальное значение будет

$T\_{0}$=$\min\_{r}T\_{r}$ (3)

Ускорение полученное на q процессорах при использовании схемы параллельных вычислений показывается величиной

W=$\sum\_{i}^{}\frac{t(x\_{i})}{T\_{0}}$ (4)

В идеальном случае при полной загрузке процессоров W=q. Если же структура алгоритма не допускает параллельных вычислений, то загрузка системы будет неполной, тогда максимальная производительность не будет достигнута. Коммуникационная система так же может помешать эффективной реализации процесса, т.е. когда q достаточно велико, эта сеть не может обеспечить быстрый обмен информацией между процессорами. При выборе оптимального способа решения задачи с помощью параллельных вычислений используется оценка их стоимости, равная произведению времени параллельного решения и числа процессоров.

Для эффективного решения можно выделить типовую схему разработки параллельного алгоритма:

* выполнить анализ схемы необходимого алгоритма и осуществить декомпозицию на части, которые могут выполняться независимо друг от друга;
* выделить информационные взаимодействия для сформированного набора подзадач, которые должны осуществляться в ходе решения задачи;
* выбрать доступную вычислительную систему и выполнить распределение задач между процессорами системы.

Основной принцип многометодных технологий состоит в применении в процессе решения задачи последовательности различных методов оптимизации, тем самым повышая эффективность процесса вычислений.

Рассмотрим базовую задачу оптимизации в следующей постановке

$$\begin{matrix}extr F(X)\\X\in D\_{x}\end{matrix}$$

 (5)

$D\_{x}$=$\left\{φ\left(X\right)>0,θ\left(X\right)=0\right\}$

F(X) - целевая функция; $D\_{x}$ – допустимая область значений в пространстве параметров;X – вектор управляемых параметров; $φ\left(X\right)$ и $θ\left(X\right)$ – функции-ограничители. Каждую оптимизационную задачу можно охарактеризовать признаками: тип функции, ограничений, параметров. Рациональный выбор метода основывается на оценке всех признаков.

Для лучшей адаптации к особенностям разнообразных задач оптимизации в многометодных стратегиях возможны следующие стратегии комплексирования:

* многомерный алгоритм состоит из отдельных схем решения, выход одного блока является началом следующего (рис 7);
* общее решение задачи формируется из экземпляров одного и того же алгоритма в различных подпространствах поиска (рис 8);
* выбор наилучшего решения по заданным критериям с помощью различных алгоритмов (рис 9).



Рисунок 7 – Последовательная схема вычислений



Рисунок 8 – Параллельная схема запусков методов различных оптимизаций



Рисунок 9 – Параллельный запуск различных экземпляров одного метода

### **2.2 Поиск оптимального решения транспортной задачи**

Данная задача достаточно актуальна и находит широкое применение в экономической области. Будем использовать параллельный алгоритм нахождения опорного плана, это позволит более эффективно решать данную задачу в многопроцессорной или кластерной среде. Итак, перейдем к математической постановке транспортной задачи:

$$min \sum\_{j=1}^{m}\sum\_{i=1}^{n}c\_{ij}X\_{ij}$$

$$\sum\_{j=1}^{m}X\_{ij}=b\_{j}, j=1…n;$$

(6)

$$\sum\_{i=1}^{n}X\_{ij}=a\_{i}, i=1…m;$$

$x\_{ij}\geq 0, t=1…r, i=1…q$.

Здесь x – это объем, с – тариф поставки продукции от поставщика к потребителю, b – потребности потребителей, a – запасы поставщиков.

Можно увидеть, что данная задача (7) является задачей линейного программирования. Мы имеем m \* n неизвестных x и m+n уравнений. Решение данной задачи выполним с помощью оптимального плана перевозок. Рассмотрим метод распределения ресурсов. Сначала построим опорный план, а затем найдем оптимальный.

Для получения лучшего решения можно использовать различные алгоритмы, но наиболее удобным для распараллеливания выглядит метод потенциалов. Используя некоторый опорный план он позволяет построить решение транспортной задачи за конечное число итераций.

Обозначим:

* c – план поставок;
* u – симплексный множитель для строк;
* v – симплексный множитель для столбцов;
* k – коэффициент для каждой ячейки;
* minc – переменная в которой будем хранить минимальное значение с (для всех ячеек, входящих в цикл перерасчета);

k рассчитывается по формуле:

$k\_{ij}=u\_{i}+v\_{j}-c\_{ij}$ (7)

Теперь рассмотри последовательный алгоритм оптимального решения поставленной задачи методом потенциалов:

1. $v\_{n}=0;$
2. найдем $u\_{i}$(i = 1…m), $v\_{j}$ ( j = 1..n–1);
3. для каждой клетки находим k (по формуле 7);
4. в случае $k\_{ij}\leq 0$, план оптимален, метод завершается;
5. выбираем ячейку (imax, jmax) с наибольшим $k\_{ij}$ = kmax и строим цикл, попутно находя наименьшее minc из значений $c\_{ij}$ в ячейках, имеющих в цикле четный номер;
6. $с\_{ij}^{,}$=$c\_{ij}$+ minc (l нечетное),$ с\_{ij}^{,}$=$c\_{ij}$ - minc (l четное). Порядковый номер ячейки в цикле;
7. переход к 3-му шагу.

Теперь модифицируем метод потенциалов для работы в параллельной среде.

1. $v\_{n}=0;$
2. найдем $u\_{i}$(i = 1…m), $v\_{j}$ ( j = 1..n–1);
3. разделим множество клеток на части пропорционально количеству процессоров N и передадим вычислительным узлам информацию;
4. для каждой клетки находим k (по формуле 7);
5. по полученным из вычислительных узлов данных выберем наибольший maxk из max $k\_{ij}$ по группам;
6. в случае maxk $\leq 0$, план оптимален, метод завершается;
7. находим minc из значений с в ячейках имеющих четный номер в цикле, параллельно строим цикл из ячейки (maxi, maxj);
8. разобьем цикл на пропорциональное количество узлов N, передаем информацию в вычислительные узлы (координаты и положение ячеек);
9. $с\_{ij}^{,}$=$c\_{ij}$+ minc (l нечетное),$ с\_{ij}^{,}$=$c\_{ij}$ - minc (l четное). Порядковый номер ячейки в цикле;
10. переходим к 3-му шагу.

На рисунках (А.3) и (А.4) в приложении можно увидеть схему алгоритма данного метода.

### **2.3 Модель региональной экономики на основе социальной стратификации**

Рассматриваемая задача находит применения в экономике, при планировании транспортного комплекса. Параллельный алгоритм реализованный в данной задаче, оптимизированный для работы в многопроцессорной среде, позволяет наиболее оптимально использовать вычислительные мощности кластера. Математическая постановка задачи имеет вид:

$$min \sum\_{j=1}^{m}\sum\_{i=1}^{n}c\_{ij}X\_{ij}$$

$$\sum\_{j=1}^{m}X\_{ij}=b\_{j}, j=1…n;$$

(8)

$$\sum\_{i=1}^{n}X\_{ij}=a\_{i}, i=1…m;$$

$$x\_{ij}\geq 0, j=1…n, i=1…m$$

Здесь $X\_{ij}$ – объем, $c\_{ij}$ – тариф поставки продукции от i-го поставщика к j-му потребителю, $b\_{j}, $ – потребности потребителей в продукции, $a\_{i}$ – запасы продукции у поставщиков. Решением транспортной задачи будет оптимальный план перевозок.

Два этапа метода распределения ресурсов:

1. построение опорного плана;
2. поиск оптимального плана.

Последовательный алгоритм решения этой задачи методом Фогеля для получения опорного плана выглядит следующим образом:

1) штрафы для строки или столбца представляют положительную разницу между минимальным элементом строки (столбца) и следующим за ним по величине минимальным элементом строки или столбца;

2) расчет штрафов для всех строк и столбцов матрицы;

3) формирование опорного плана начинается с минимального элемента строки или столбца с максимальным штрафом;

4) выбор элементов плана Х и коррекция векторов потребления и постановок осуществляется, как рассмотрено выше (шаги 1, 2);

5) строка или столбец матрицы С, которым соответствует нулевое. значение потребности или поставки, в дальнейшем не участвуют в формировании штрафов. Процесс продолжается до тех пор, пока не обнулятся все вектора.

Последовательное нахождение опорного плана методом графов можно увидеть в приложении (рис. А.1 и рис. А.2).

Данная задача является актуальной для различного масштаба торговых и производственный предприятий. Попробуем модифицировать приведенный выше алгоритм Фогеля для выполнения на вычислительном кластере. В качестве библиотеки коммуникаций используем MPI (Message Passing Interface).

Введем некоторые обозначения: u – вектор потребностей, х – матрица решения (поставок), v – вектор мощности поставщиков, z – матрица стоимости перевозок. Тогда наш алгоритм будет выглядеть следующим образом:

1. определяем разницу между наименьшим значением стоимости и ближайшему к нему для каждого столбца и строки матрицы стоимости. Поиск штрафа оформляется в виде отдельной задаче и направляется к не занятому процессору;
2. находим ячейки с наименьшим минимальным значением стоимости. Распределяем выполнение между свободными процессорами;
3. минимизируем поставку в ячейку x[i,j], выбрав минимальное значение из мощности поставщика и потребности потребителя, v[i] и u[i] соответственно;
4. если потребность полностью удовлетворена или мощность поставщика полностью реализована, вычеркиваем эту строку и столбец;
5. повторяем, если не все потребности или мощности задействованы.



Рисунок 10 – Параллельный алгоритм нахождения опорного плана методом Фогеля

## **ЗАКЛЮЧЕНИЕ**

Параллельные вычисления довольно успешно применяются в области экономики, особенно при масштабных задачах с трудоемкими вычислениями. Они помогают повысить и оптимизировать эффективность какого-либо предприятия за минимальное время. Особенно интересно смотрятся задачи оптимизации, ведь именно они помогают распределить ресурсы и экономить время, что является важными качествами для экономики, благодаря которым можно минимизировать расходы и повысить доходность. На следующем этапе планируется исследование и написание программы задачи оптимизации.

## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1 Словарь по кибернетике / под редакцией академика В.С. Михалевича – 2-е. – Киев: Главная редакция Украинской Советской Энциклопедии имени М.П.Б. ажана, 1989. – 751 с. – (С48).)

2 Программирование для высокопроизводительных ЭВМ / А.В.Комолкин, С.А.Немнюгин. URL : <http://www2.sscc.ru/links/litera/hpc/g1.1.html> [21 декабря 2018]

3 Some computer organizations and their effectiveness / M.J. Flynn // IEEE Trans. Comp. – 1972. – V. C‑21, № 9. – P. 948–960.

4 Кластеры URL: [https://parallel.ru/cluster/beginner\_guide.html [27](https://parallel.ru/cluster/beginner_guide.html%20%5B27) декабря 2018]

5 SMP системы URL: <https://parallel.ru/computers/classes.html#smp> [27 декабря 2018]

6 Руководство по работе на вычислительном кластере URL: <http://window.edu.ru/resource/070/24070/files/shpakovski_cluster.pdf> [22 декабря 2018]

7 Разработка параллельного алгоритма нахождения оптимального решения транспортной задачи на вычислительном кластере URL: https://cyberleninka.ru/article/n/razrabotka-parallelnogo-algoritma-nahozhdeniya-optimalnogo-resheniya-transportnoy-zadachi-na-klastere [22 декабря 2018]

8 Применения параллельных вычислений при решении зада оптимизации URL: https://cyberleninka.ru/article/v/primenenie-parallelnyh-vychisleniy-pri-reshenii-zadach-optimizatsii [23 декабря 2018]

9 Разработка параллельного алгоритма построения опорного плана транспортной задачи URL: https://cyberleninka.ru/article/v/razrabotka-parallelnogo-algoritma-postroeniya-opornogo-plana-transportnoy-zadachi [23 декабря 2018]

## **ПРИЛОЖЕНИЕ А**

**Схемы алгоритмов**



Рисунок А.1 – Последовательное нахождение опорного плана методом штрафов (начало)



Рисунок А.2 – Последовательное нахождение опорного плана методом штрафов (конец)



Рисунок А.3 – Параллельный алгоритм поиска оптимального решения транспортной задачи методом потенциалов (начало)



Рисунок А.4 – Параллельный алгоритм поиска оптимального решения транспортной задачи методом потенциалов (конец)