**МИНОБРНАУКИ РФ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САМАРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра "Вычислительная техника"

# Методы параллельных вычислений

*Методические указания к лабораторным работам*

Направление 09.03.01 - Информатика и вычислительная техника
09.03.04 – Программная инженерия

Составитель: профессор Орлов С.П.

Самара
2014

## *Оглавление*

## Лабораторная работа №1
## Модель конвейерных вычислений в высокопроизводительной системе

### Краткая теория

**Цель работы.** Изучение принципов конвейерной обработки с помощью простейшей имитационной модели.

Максимальная производительность современных процессоров и ЭВМ определяется, в первую очередь, уровнем развития соответствующей технологии, которая ограничивает время распространения сигнала по электронным схемам. Дальнейшее увеличение производительности возможно за счёт архитектурных решений. Одним из основных способов построения высокопроизводительных систем является параллелизм. Параллельная обработка реализуется двумя способами:

- во времени и
- в пространстве.

Первый способ называют конвейерным, а второй - матричным. В обоих случаях необходимо использовать несколько обрабатывающих устройств, количество которых определяет максимальное число параллельных процессов.

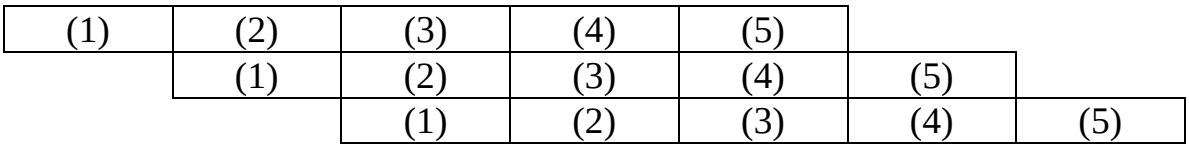
Известно, что типичную арифметическую команду можно разделить на следующие микрооперации:

1) выборка команд из памяти (по адресу в счётчике команд);
2) декодирование кода операции;
3) выборка операндов из регистров;
4) выполнение операции в АЛУ;
5) запоминание результата в регистре.

Если система содержит 5 обрабатывающих устройств, каждое из которых обеспечивает выполнение одной из перечисленных микроопераций, то имеется возможность реализовать конвейерную обработку. Устройства должны быть специализированными. Их необходимо расположить в порядке следования микроопераций в типовой команде, что позволит совместить выполнение отдельных микроопераций разных команд и сократить общее время обработки данных.

Если предположить, что каждая микрооперация занимает один такт машинного времени, то реализация последовательности из нескольких (например, трёх) команд с совмещением может быть представлена в виде схемы рис.1. Из рисунка видно, что результат первой команды будет получен после 5-го такта, второй - после 6-го, а третьей - после 7-го. Таким образом, среднее время выполнения последовательности из трех команд будет равно 2.33 такта. С увеличением длины последовательности это время уменьшается и в пределе стремится к 1 такту. Такой эффект наблюдается только при одинаковых длительностях микроопераций. Если эти длительности отличаются, то некоторые устройства будут простаивать, и среднее время выполне-

ния команды увеличится. Для согласования работы устройств в таких условиях применяют буферизацию.




Р и с . 1 . Возможное совмещение микроопераций
при выполнении трех последовательных команд
(в скобках указаны номера микроопераций)

Эффективность работы произвольного конвейера определяется величиной разности между средним временем выполнения команды в нем и предельным временем. Целью предлагаемой лабораторной работы является исследование влияния длины последовательности команд и соотношения длительностей отдельных микроопераций на эффективность работы конвейера.

Занятие выполняется на ПЭВМ типа IBM, работающей в среде Windows XP/NT. Программа имитационного моделирования конвейера команд составлена на языке Delphi 3.0 и позволяет выполнить следующее:

Варьировать количество команд в последовательности от 3 до 10 (по умолчанию их 3).

Изменять количество тактов любой микрооперации в диапазоне от 1 до 30. По умолчанию эти значения равны единице.

Установить один из двух режимов моделирования: непрерывно или по тактам. По умолчанию моделирование осуществляется в непрерывном режиме.

В имитационной модели последовательность микроопераций в команде (занятость соответствующих устройств) для наглядности изображена разными цветами, как показано на рис.1.2, а простой устройства (ожидание прихода очередной команды) представлен черным цветом. Изменение параметров команд и микроопераций сопровождается соответствующим изменением схемы их выполнения, которая выводится на экран. При варьировании параметров микрооперации изменяется ширина ее поля на схеме.

Исполняемый модуль программы моделирования имеет имя **Labcv** и находится в папке **Lab 1**. Его запуск осуществляется традиционным способом. После запуска на экране появляется основное окно, которое позволяет задать параметры модели и режим моделирования. Все перечисленные значения и режимы устанавливаются с помощью соответствующих кнопок, как это принято в Windows 95 и выше. Если запуск модели был осуществлён в тактовом режиме, то каждый раз для продолжения работы необходимо нажимать на кнопку «Следующий такт».

| Выборка команд (1) | Декодирование КОП (2) | Выборка операндов (3) | Выполнение операции (4) | Запись результата (5) |
|---|---|---|---|---|
| желтый | голубой | синий | красный | светло-зеленый |

Рис. 2. Представление последовательности микроопераций при моделировании типовой команды

Во время работы программы в обоих режимах на экране цветом представляется реализация команд, выводится количество завершенных на данный момент тактов, среднее время выполнения команды и суммарный простой устройств в тактах. Работа модели прекращается, если выполнены все команды заданной последовательности. При этом программа автоматически завершает работу и осуществляет подсчёт и вывод всех характеристик:

- среднего времени выполнения одной команды,
- суммарного времени ожидания устройств в системе,
- графиков зависимости среднего времени выполнения команды от количества команд и от длительностей каждой из пяти микроопераций в команде,
- пяти графиков зависимости среднего времени ожидания каждого обрабатывающего устройства от длительностей любой из пяти микроопераций в команде.

Перечисленные графики выводятся после нажатия кнопки «Графики» в соответствующие окна, которые можно расположить каскадом и просматривать в любой последовательности, нажав соответствующую кнопку.

## Порядок выполнения лабораторной работы

### Подготовка к работе

1. Знакомство со всеми разделами руководства.
2. Получение у преподавателя задания на исследование конвейера с различными параметрами потока команд.
3. Исследование заданного конвейера.
4. Оформление отчета.

### Последовательность выполнения лабораторной работы

Необходимо исследовать следующие режимы работы конвейера команд:

a) **С одинаковым количеством команд и одинаковыми длительностями микроопераций**. Последние изменяются во всех командах от 1 до 5 (30) тактов - по заданию преподавателя. Количество команд в последовательности также задается преподавателем.

b) **С переменным количеством команд и одинаковыми длительностями микроопераций**. Количество команд в последовательности изменяется от 1 до 10

(по заданию преподавателя). Длительности микроопераций задаются преподавателем.

c) **С изменением длительности последней микрокоманды.** Пределы длительности микрокоманды задаются преподавателем. Длительности остальных микрокоманд остаются постоянными.

d) **С изменением длительности первой микрокоманды.** Пределы длительности микрокоманды задаются преподавателем. Длительности остальных микрокоманд остаются постоянными.

e) **С изменением длительности одной из средних микрокоманд.** Пределы длительности и тип микрокоманды задаются преподавателем. Длительности остальных микрокоманд остаются постоянными.

### *Содержание отчета о выполненной работе*

Отчет должен содержать следующее:

1. Название и цель работы.
2. Исходные данные.
3. Графики зависимостей, полученных на ЭВМ (по указанию преподавателя ).
4. Выводы об эффективных режимах работы конвейера.

## Лабораторная работа №2
## Модель конвейерных вычислений с конфликтами по данным

### *Краткая теория*

**Цель лабораторной работы.** Изучение влияния различных конфликтов на эффективность конвейерной обработки.

Одним из основных способов построения высокопроизводительных систем, как известно, является распараллеливание процессов, которое используется в конвейерной обработке. Типичную арифметическую команду, как предлагалось в лабораторной работе 1, можно разделить на следующие микрооперации:

1) выборка команд из памяти (по адресу в счётчике команд);
2) декодирование кода операции;
3) выборка операндов из регистров;
4) выполнение операции в АЛУ;
5) запоминание результата в регистре.

Если система содержит 5 обрабатывающих устройств, каждое из которых обеспечивает выполнение одной из перечисленных микроопераций, то имеется возможность реализовать конвейерную обработку. Известно, что наиболее эффективным является режим работы конвейера, при котором

- все микрокоманды имеют одинаковую длительность;
- обрабатывающие устройства располагаются в порядке, соответствующем порядку следования микрокоманд.

Если одно из указанных условий нарушается, то некоторые устройства будут простаивать, а эффективность снизится. Ситуации, которые препятствуют очередной команде из потока выполняться в предназначенном для нее такте называются **конфликтами** в конвейере. Очевидно, что конфликты снижают производительность конвейера. Существует три типа конфликтов:

1) Структурные, которые возникают из-за занятости отдельных ресурсов (когда аппаратные средства не могут поддерживать выполнение команд в режиме с совмещением);

2) Конфликты по данным, появляющиеся, если выполнение одной команды зависит от результата предыдущей;

3) Конфликты по управлению, которые возникают при наличии команд переходов и других команд, изменяющих значение счетчика команд.

Все конфликты приводят к приостановке выполнения команды, в которой они возникли (pipeline stall), а также всех следующих за ней до конца конвейера. Эта ситуация называется "конвейерным пузырем" (pipeline bubble). Пузырь проходит по конвейеру, не выполняя никакой работы.

Примером структурных конфликтов являются коллизии, возникающие при промахе в кэш-памяти и при необходимости выполнения записей в два или более регистров регистрового файла в течение одного машинного такта. Аналогичная ситуация возникает при разных длительностях тактов отдельных микроопераций.

Конфликты по данным связаны с последовательным выполнением команд в программах. Например, во фрагменте:

```
Add R1,R2
Sub R3,R1
And R4,R1
Or  R5,R1
Xor R6,R1
```

все команды, следующие за Add, используют ее результат (который помещен в регистр R1) и должны ожидать завершения ее последней микрооперации. Конфликты по данным возможны везде, где имеет место зависимость между командами, и они расположены достаточно близко друг к другу. Причем эти ситуации могут возникать не только при работе с регистрами АЛУ, но и при обращении к одной и той же ячейке памяти.

Конфликты по управлению могут быть самыми тяжелыми с точки зрения потерь производительности конвейера. При выполнении условного перехода вся цепочка команд, находящихся в начале конвейера, выполнялась напрасно. Работа конвейера приостанавливается до тех пор, пока в его начало не будет загружена команда с новым адресом (адресом перехода). Однако эффект от конфликта можно

уменьшить, если учесть, что значение счетчика команд изменяется только в одном из двух случаев (когда переход происходит).

В реальных системах реализованы различные меры борьбы с перечисленными конфликтами. Например, структурные конфликты могут быть ликвидированы за счет создания так называемых «полностью конвейерных устройств», в которых длительности тактов выполнения всех микроопераций одинаковы, используется раздельная память команд и данных и другие средства. Конфликты по данным устраняются с помощью оптимизирующих компиляторов или аппаратных средств микропроцессоров, которые изменяют порядок выполнения команд в программе так, чтобы команды, связанные по данным, располагались в ней на расстоянии, превышающем длину конвейера. Другим путем ликвидации этих конфликтов является расширение регистровой и кэш памяти, в частности, создание регистровых файлов, у которых запись данных производится в разные ячейки. Конфликты по управлению устраняются за счет дублирования ветвей, а также использования так называемого отсроченного или прогнозируемого перехода.

Занятие выполняется на ПЭВМ типа IBM, работающей в среде Windows XP/NT. В ней используется та же программа имитационного моделирования, что и в работе 1. Она позволяет выполнить следующее:

Варьировать количество команд в последовательности от 3 до 10 (по умолчанию их 3).

Изменять количество тактов любой микрооперации в диапазоне от 1 до 30. По умолчанию эти значения равны единице.

Задать координату «пузыря» в конвейере. Она соответствует номеру команды, в течение которой конвейер простаивает.

Установить один из двух режимов моделирования: непрерывно или по тактам. По умолчанию моделирование осуществляется в непрерывном режиме.

В имитационной модели, как и в первой работе, последовательность микроопераций в команде для наглядности изображена разными цветами, как показано на рис. 1.2, а «пузырь», который приводит к задержке в работе конвейера, представляется черным цветом (цветом фона).

Исполняемый модуль программы моделирования имеет имя **Labcv** и находится в папке **Lab 2**. Его запуск осуществляется традиционным способом. После запуска на экране появляется основное окно, которое позволяет задать параметры модели и режим моделирования. Все перечисленные значения и режимы устанавливаются с помощью соответствующих кнопок. Если запуск модели был осуществлён в тактовом режиме, то каждый раз для продолжения работы необходимо нажимать на кнопку «Следующий такт».

Во время работы программы в обоих режимах на экране цветом представляется реализация команд, выводится количество завершенных на данный момент тактов, среднее время выполнения команды и суммарный простой устройств в тактах. Ра-

бота модели прекращается, если завершены все команды заданной последовательности. При этом программа автоматически прекращает работу и осуществляет подсчёт и вывод следующих характеристик:

- среднего времени выполнения одной команды,
- суммарного времени ожидания устройств в системе,
- графиков зависимости среднего времени выполнения команды от количества команд и от длительностей каждой из пяти микроопераций в команде,
- пяти графиков зависимости среднего времени ожидания каждого обрабатывающего устройства от длительностей любой из пяти микроопераций в команде.

Перечисленные графики, как и в работе 1, выводятся после нажатия кнопки «Графики» в соответствующие окна, которые можно расположить каскадом и просматривать в любой последовательности, выбрав необходимую кнопку.

## Подготовка к работе

1. Знакомство со всеми разделами руководства.
2. Получение у преподавателя задания на исследование конвейера с различными параметрами потока команд и координатой «пузыря» в нем.
3. Исследование заданного конвейера.
4. Оформление отчета.

## Последовательность выполнения лабораторной работы

В лабораторной работе необходимо исследовать следующие режимы функционирования конвейера команд:

a) **С одинаковым количеством команд, одинаковыми длительностями микроопераций и изменяющимся положением начала «пузыря» в конвейере.**

Начало «пузыря» изменяется в пределах, задаваемых преподавателем. Количество команд в последовательности и длительности тактов микроопераций также задаются преподавателем;

b) **С одинаковым количеством команд, изменяющейся длительностью первой микрооперации и постоянным положением начала «пузыря» в конвейере.** Пределы изменения длительности первой микрооперации и все остальные параметры задаются преподавателем;

c) **С одинаковым количеством команд, изменяющейся длительностью последней микрооперации и постоянным положением начала «пузыря» в конвейере.** Пределы изменения длительности последней микрооперации и все остальные параметры задаются преподавателем.

d) **С одинаковым количеством команд, изменяющейся длительностью одной из средних микроопераций и постоянным положением начала «пузыря» в конвейере.** Номер и пределы изменения длительности средней микрооперации, а также все остальные параметры задаются преподавателем.

### Содержание отчета о выполненной работе

Отчет должен содержать следующее:

1. Название и цель работы.
2. Исходные данные.
3. Графики зависимостей, полученных на ЭВМ (по указанию преподавателя).
4. Выводы об эффективных режимах работы конвейера и влиянии положения и длительности «пузыря» на производительность конвейера.

## Лабораторная работа №3 Построение модели вычислений в виде сети Петри

### Краткая теория

**Цель работы.** Изучение принципов имитационного моделирования вычислительного процесса с помощью сети Петри.

Сети Петри состоят из двух компонентов: *базовой сети* и *начальной маркировки*. Базовая сеть — это граф с конечным числом вершин двух типов — *местами* и *переходами* — и направленными дугами. При этом ни одна дуга графа не связывает вершины одинаковых типов. Кроме того, две вершины сети могут быть связаны более, чем одной дугой. Вершина сети может иметь *входные* вершины, т.е. такие, из которых дуги ведут в данную вершину, и *выходные* вершины, т.е. такие, в которые ведут дуги из данной вершины. В графическом представлении места изображаются кружками, а переходы — барьерами. Места могут хранить *фишки*, изображаемые жирными точками. Распределение фишек по местам сети называется *маркировкой* и соответствует состоянию моделируемой системы. Говорят, что переход сети *готов к срабатыванию* при некоторой маркировке, если каждое его входное место содержит достаточное (т.е. большее или равное числу дуг между данными местом и переходом) количество фишек. *Срабатывание* готового перехода приводит к смене маркировок: из каждого входного места перехода удаляются и к каждому выходному месту перехода добавляются фишки в количестве, соответствующем числу дуг, связывающих данный переход с его входными и выходными местами. На рис. 3 показан пример сети Петри до и после смены маркировок при срабатывании готового перехода.

Рис.3. Элементарная сеть Петри

Используя такое правило срабатывания переходов, сети Петри могут моделировать различные динамические системы. Рассмотрим в качестве примера сеть Петри, моделирующую систему "производитель/потребитель" с неограниченным буфером.

При маркировке, которую будет называть *начальной*, производитель может начать свою работу, т.е. производить данные, которые впоследствии помещаются в буфер. С этого момента может начать свою работу потребитель, который берет данные из буфера, а затем потребляет их. Независимо от потребителя производитель после того, как поместит данные в буфер, может продолжать свою работу.

Для моделирования работы нескольких производителей и потребителей достаточно разместить соответствующее число фишек в места сети Петри.

В случае, когда производитель помещает данные в буфер быстрее, чем потребитель их забирает, в буфере может накапливаться неограниченное число фишек, что не должно происходить в реальных системах, так как число ячеек буфера всегда ограничено. Видим, что в сети появилось дополнительное место, содержащее столько фишек, сколько свободных ячеек в буфере (исходно n фишек). Теперь потребитель помещает данные в буфер, пока в нем есть свободные ячейки, так как у перехода "Положить в буфер" появилось дополнительное входное место, а у перехода "Взять из буфера" — дополнительное выходное.

## Порядок выполнения лабораторной работы

### Подготовка к работе

1. Знакомство со всеми разделами руководства.
2. Получение у преподавателя задания на моделирование вычислительного процесса параллельного алгоритма заданного типа.
3. Разработка структуры сети Петри, определение множества переходов и позиций.
4. Выполнение заданного процесса на сети Петри.
5. Оформление отчета.

### Последовательность выполнения лабораторной работы

1. Построить сеть Петри.

2. Задать множество начальных разметок позиций сети.
3. Для каждой начальной разметки провести имитацию выполнения процесса вычислений.
4. Определить множество достижимых разметок в сети.
5. Определить множество недостижимых разметок в сети Петри.
6. Выявить причины конфликтных ситуаций, соответствующих недостижимым разметкам сети Петри.

### Содержание отчета о выполненной работе

Отчет должен содержать следующее:

1. Название и цель работы.
2. Исходные данные.
3. Общую структуру сети Петри.
4. Множество начальных и терминальных разметок.
5. Результат имитации процесса на сети Петри.
6. Выводы об условиях бесконфликтного выполнения вычислительного процесса.

## Лабораторная работа №4 Построение и изучение модели алгоритма в виде графа «операнд – операции»

### *Краткая теория*

**Цель работы.** Овладение методикой моделирования параллельных алгоритмов с помощью графового представления вычислительного процесса.

Для описания информационных зависимостей алгоритмов решения задач широко используют модель в виде ациклического ориентированного графа, называемую *графом алгоритма*. В этой модели множество операций алгоритма и существующие между операциями информационные зависимости описываются двойкой:

$$G = (V, E), \qquad (1.1)$$

где $V = \{1,...,|V|\}$ – множество вершин графа, представляющих операции алгоритма, а $E$ – множество дуг графа, устанавливающих *частичный* порядок операций. Дуга $E_{i,j} = (i, j)$ принадлежит графу только в том случае, если операция $j$ использует результат выполнения операции $i$. Свойство *ацикличности* графа алгоритма состоит в том, что никакая величина не может определяться через саму себя.

Описанная выше модель является *направленным графом*. Если дугам графа приписать веса $c_{ij},\left(i, j = \overline{1, N}\right)$, отражающие интенсивность информационного обмена между $i$-й и $j$- ветвями программы, такой граф называется *взвешенным направленным графом*. В общем случае граф алгоритма есть мультиграф, т.е. две вершины могут быть связаны несколькими дугами]. При этом в качестве разных аргументов

одной операции используется одна и та же величина. Количество вершин графа (не считая вершин ввода) далее будем обозначать $|\bar{V}|$. Путь максимальной длины в графе называют критическим.

Для ориентированного ациклического графа с $n$ вершинами существует число $s<n$ , для которого все вершины графа можно так пометить одним из индексов 1,2,…,$s$, что если дуга из вершины с индексом $i$ идет в вершину с индексом $j$, то $i<j$.

Нетрудно заметить, что, например, для графов, показанных на рис. 1.4 и 1.5, в обоих случаях $n$=7, а число $s$ при этом принимает значения 4 и 3 соответственно. Из схемы разметки, в частности, следует:

- никакие две вершины с одинаковым индексом не связаны дугой;
- минимально число индексов на единицу больше длины критического пути;
- для любого целого $s$ , не превосходящего числа вершин, но большего длины критического пути, существует разметка, при которой используются все $s$ индексов.

Граф, размеченный в соответствии с описанной схемой, называют *строгой параллельной формой графа*.

Существует строгая параллельная форма, в которой максимальная из длин путей, оканчивающихся в вершине с индексом $k$, равна $k$ -1, и все входные вершины находятся в одной группе с индексом 1. Она называется *канонической*. Для заданного графа каноническая форма *единственна*.

Группа вершин с одинаковыми индексами называется *ярусом*, число вершин в группе – *шириной* яруса, а число ярусов – *высотой* параллельной формы. Параллельная форма минимальной высоты называется *максимальной*, т.к. в каждом ярусе такой формы максимальное число вершин.

Предположим теперь, что все операции алгоритма выполняются за одинаковое время, равное 1, каждая операция может начаться в момент готовности ее аргументов, а все операции, не имеющие предшествующих, могут выполняться одновременно (параллельно). Обозначим момент начала реализации алгоритма нулем, а каждой операции будем присваивать индекс, равный моменту окончания ее выполнения. Если эти индексы перенести на вершины графа алгоритма, то мы получим каноническую форму.

Ограничивая число операций, которые могут выполняться параллельно, можно получить отличающиеся строгие параллельные формы. В предельном случае, когда на каждом шаге вычислительного процесса может выполняться только одна операция, т.е. все ярусы имеют ширину, равную 1, будет получена так называемая *линейная* форма, т.е. граф упорядочивается *линейно*.

## Порядок выполнения лабораторной работы
## Подготовка к работе и последовательность выполнения практического занятия

1. Знакомство со всеми разделами руководства.
2. Получение у преподавателя задания на моделирование вычислительного процесса параллельного алгоритма заданного типа.
3. Разработка графовой модели типа «Операнд – операции».
4. Определение критического пути на графе.
5. Определение кратчайшего пути на графе.
6. Оценка времени выполнения алгоритма высокопроизводительных вычислений.
7. Оформление отчета.

## Содержание отчета о выполненной работе

Отчет должен содержать следующее:

1. Название и цель работы.
2. Исходные данные.
3. Общую структуру графа «операнд - операции».
4. Расчет критических путей и кратчаейшего пути.
5. Выводы о скорости выполнения параллельного вычислительного алгоритма.

## Лабораторная работа №5 Сравнительное изучение разновидностей операторов циклов и ветвлений в последовательных и параллельных программах

### Краткая теория

**Цель работы.** Изучение особенностей операторов циклов и ветвлений, работающих в последовательных и параллельных программах.

Рассмотрим типы операторов:

- **Структура вхождений DO-циклов.** Фиксировались все формы вложенности операторов DO (DO-циклов). Сложные конструкции дополнительно разбивались на составные части по их вложенности, которые также включались в общую статистику. Выяснилось, что тесно вложенных гнезд циклов всего около 7%, причем такие гнезда с вложенностью 3 и более не встречались ни разу. Учитывая популярность тесно вложенных гнезд в исследовательских работах, была дополнительно определена доля программ, в которых хотя бы просто содержались подобные гнезда. Результат оказался еще хуже. Это говорит о том, что методы, ориентированные только на тесно вложенные циклы, не будут иметь большой перспективы. Интересным оказалось то, что около 40% всех конструкций представляют собой циклы, в тело которых линейно входит большое число других циклических конструкций. В среднем около 5, а максимальная последовательность состояла из 19 конструкций. Примерно в 20% случаев отдельные срабатывания тел одиночных циклов оказались независимыми друг от друга только за счет несовпадения идентификаторов входов и выходов операторов.
- **Мощность и внешняя вложенность DO-циклов.** Рассматривались все операторы цикла DO и для каждого определялись две характеристики, позволяющие хотя бы приблизительно оценить вклад фрагмента в общее время счета. Первая — внешняя вложенность. Она определяется глуби ной погружения в другие циклы и равна числу циклов, охватывающих данный оператор. Вторая — мощность. Она равна максимальной глубине вложенности входящих в цикл операторов относительно рассматриваемого оператора ОО. Мощность любого цикла не меньше единицы, в то время как внешняя вложенность может быть равной нулю. Проведенное исследование говорит о том, что реальные циклические конструкции не являются даже правильными гнездами и содержат по несколько других операторов цикла на каждом уровне. Например, в анализируемом материале имелось 119 циклов внешней вложенности 0 и мощности 2, но 280 циклов внешней вложенности 1 и мощности 1. Это значит, что для проведения эффектив-

ного анализа необходимо разрабатывать и использовать методы, умеющие исследовать совместное функционирование линейно расположенных циклических конструкций.

- **Количество максимально вложенных циклов.** Была сделана попытка определить число наиболее критичных по времени счета мест в программе. Для этого сначала определялась мощность программы как наибольшая мощность ее циклических конструкций. Затем по каждой мощности определялась доля программ, имеющих 1, 2, 3 и более максимально вложенных циклов. Обнаружен довольно интересный факт — доля программ, содержащих более одного критического фрагмента, достаточно велика. В частности, среди программ мощности 3 свыше 20% имеют 6 и более циклических конструкций максимальной вложенности.
- **Структура тел внутренних циклов.** Исследовалось, какие типы операторов входят в самые внутренние циклы, т. е. циклы мощности 1. Статистика собиралась раздельно по циклам с различной внешней вложенностью, чтобы выявить структуру наиболее значимых по времени счета фрагментов. Обнаружено, что с ростом глубины вложенности заметно растет доля циклов, тела которых содержат только операторы присваивания. Например, для циклов глубины вложенности 1 эта доля составляет 73%, для циклов глубины вложенности 3 — 93%. Следовательно, циклическим конструкциям с телами циклов, состоящими только из операторов присваивания, необходимо уделять особое внимание.
- **Структура вхождения альтернативных операторов.** Альтернативные операторы осуществляют выбор дальнейшего действия в зависимости от выполнения некоторого условия. В статистике учитывались условные операторы, вычисляемый переход и переход по предписанию. Проверялось, задают ли альтернативные операторы ветвления в рамках тела данного цикла или же возможен побочный выход из цикла и соответственно прекращение его выполнения. Одновременно определялось, насколько часто в программах встречались безусловные фрагменты. Важным выводом является то, что относительно немного циклов имеют побочный выход. Это дает уверенность в их хорошей исследуемости.

Выбирая класс анализируемых программ, приходится решать две противоречивые задачи. С одной стороны, хотелось бы его взять как можно более широким, с тем чтобы он покрывал значительную часть, а в идеале даже все практически встречающиеся программы или их наиболее значимые фрагменты. С другой стороны, чем шире класс программ, тем меньше надежды на то, что для любой его программы удастся провести одинаково глубокий анализ. Умение, граничащее с искусством и удачей, состоит в нахождении "золотой середины". Проведение только что описанного статистического анализа как раз и было направлено на ее поиск. После

некоторых как удачных, так и неудачных попыток было решено класс анализируемых программ сделать двухуровневым. Первый или базовый уровень образует строго описанный так называемый линейный класс. Для любой его программы проводится самый тщательный анализ информационной структуры. Второй уровень открытый. В него входят все программы, которые каким-либо способом сводятся к программам базового уровня.

### Порядок выполнения лабораторной работы

1. Знакомство со всеми разделами руководства.
2. Получение у преподавателя задания на исследование программы с циклическими и ветвящимися участками.
3. Построение последовательной программы.
4. Построение параллельной программы.
5. Оформление отчета.

### Содержание отчета о выполненной работе

Отчет должен содержать следующее:

1. Название и цель работы.
2. Исходные данные.
3. Схему алгоритма заданной программы.
4. Последовательную и параллельную программы.
5. Циклические участки программы.
6. Участки программы с ветвлениями.
7. Анализ эффективности работы операторов различного типа.
8. Выводы об эффективных режимах работы циклических операторов и операторов ветвления для различных математических алгоритмов.

## Лабораторная работа №6 Изучение потоковой модели высокопроизводительных вычислений

### Краткая теория

**Цель работы.** Изучение способов представления вычислений в виде потока .

На рис. 4 и 5приведены примеры ориентированных графов, описывающих алгоритмы нахождения суммы последовательности числовых значений

$$S = \sum_{i=1}^{n} x_i \quad , \tag{1.2}$$

где $n$ – количество суммируемых значений. В частности, на рис. 1.4 показан ориентированный граф

$$G_S = (V_S, R_S) \tag{1.3}$$

алгоритма последовательного суммирования элементов числового набора. Здесь $V_S = \{v_{01},...,v_{0,n},v_{11},...,v_{1,n}\}$ – множество операций (ввода – $v_{0,i}$ и суммирования – $v_{1i}$, $1 \le i \le n-1$), а $R_S = \{(v_{0,i}, v_{1,i}), (v_{1,i}, v_{1,i+1}), \quad 1 \le i \le n-1\}$ – множество дуг, определяющих информационные зависимости операций. В данном случае операции ввода обозначены цифрами 1-4, а операции суммирования – цифрами 5-7. Нетрудно заметить, что этот граф является линейной формой и не допускает параллельную реализацию на многопроцессорной системе.

Параллельная реализация алгоритма суммирования возможна, например, в случае, когда алгоритм строится в виде каскадной схемы:

- на первой итерации каскадной схемы все исходные данные разбиваются на пары, и для каждой пары вычисляется сумма их значений;
- полученные суммы также разбиваются на пары, и снова выполняется суммирование значений пар и т.д.

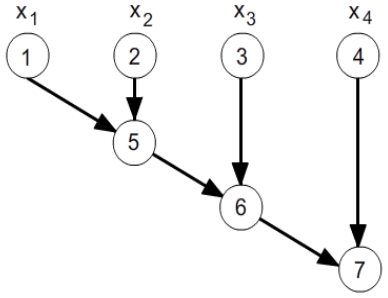



Рис. 4 Граф алгоритма последовательного суммирования

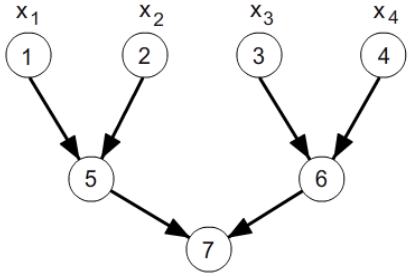



Рис. 5 Граф алгоритма каскадного суммирования

Соответствующий ориентированный граф каскадной схемы приведен на рис. 5. Он является одной из возможных строгих параллельных форм. Цифровые обозначения операций здесь те же, что и на рис. 4.

## Порядок выполнения лабораторной работы

1. Знакомство со всеми разделами руководства.
2. Получение у преподавателя задания на исследование программы с циклическими и ветвящимися участками.
3. Построение последовательной программы.
4. Построение потоковой модели вычислений.
5. Построение параллельной программы.
6. Оформление отчета.

### Содержание отчета о выполненной работе

Отчет должен содержать следующее:

1. Название и цель работы.
2. Исходные данные.
3. Схему алгоритма заданной программы.
4. Циклические участки программы.
5. Участки программы с ветвлениями.
6. Потоковую модель вычислений.
7. Последовательную и параллельную программы.
8. Анализ эффективности работы операторов различного типа с помощью потоковой модели..
9. Выводы об эффективных режимах работы циклических операторов и операторов ветвления для различных математических алгоритмов.

## Лабораторная работа № 7 Построение параллельной программы матричных вычислений

### Краткая теория

**Цель работы.** Изучение способов представления вычислений в виде потока .

Пусть вектор $y$ вычисляется как произведение квадратной матрицы $A$ и вектора $x$ порядка $n$. Если $y_i$, $a_{ij}$, $x_j$,- суть элементы соответствующих векторов и матрицы, то для всех i имеем:

$$y_i = \sum_j^n a_{ij} x_j.$$

Предположим, что вычислительная система имеет $n^2$ процессоров. Тогда на первом шаге можно параллельно вычислить все $n^2$ произведений $a_{ij}$ $x_j$, а затем, используя схему сдваивания для сложения, за $\log_2 n$ шагов вычислить параллельно все

*n* сумм, определяющих координаты вектора *y*. Следовательно, можно построить алгоритм вычисления произведения квадратной матрицы и вектора порядка *n* высоты порядка $\log_2 n$. При этом ширина алгоритма равна $n^2$. Процессоры используются неравномерно. Только на первом шаге задействованы все процессоры. Затем число работающих процессоров на каждом шаге уменьшается вдвое. Нельзя согласно утверждению 2.1 построить алгоритм, имеющий меньшую высоту. Но существуют алгоритмы, имеющие при логарифмической высоте несколько меньшую ширину.

Задачу вычисления произведения двух матриц порядка *n* можно рассматривать как задачу вычисления *n* произведений одной матрицы и *n* независимых векторов порядка *n,* считая векторами столбцы второй матрицы. Если все эти произведения вычислять параллельно по описанному алгоритму, то полученный алгоритм будет иметь высоту порядка $\log_2 n$ и ширину $n^3$. Снова процессоры используются неравномерно, и опять нельзя построить алгоритм, имеющий меньшую высоту, но существуют алгоритмы, имеющие при логарифмической высоте несколько меньшую ширину.

Обратим внимание на следующее обстоятельство. Если пытаться реализовывать описанные алгоритмы по ярусам канонической параллельной формы, то возникает необходимость одновременной рассылки одних и тех же данных по многим процессорам. Такую операцию нельзя осуществить очень быстро. Поэтому в реальных условиях временные затраты на рассылки приводят к значительному замедлению процессов вычислений.

Построение параллельных алгоритмов наименьшей высоты для матрично-векторных сумм и произведений не вызвало никаких затруднений. Это может создать впечатление, что такие алгоритмы легко строятся и для других матрично-векторных задач. Подобное впечатление, конечно, не верно. Рассмотренные алгоритмы скорее являются исключением, чем правилом.

Рассмотрим процесс вычисления векторов $x_i$ ,$1 \le i \le s$ , при заданных векторах $x_0$,

$x_{-1}, \ldots, x_{-r+1}$ с помощью линейных рекуррентных соотношений следующего вида:

$$x_i = A_{i1} x_{i-1} + \ldots + A_{ir} x_{i-r} + b_i .$$

Здесь $A_{ij}$, $b_i$ — заданные матрицы и векторы порядка $n$, при $n = 1$ они становятся числами. Предположим, что мы хотим построить при фиксированном s параллельный алгоритм вычисления векторов $x_1$ $x_s$ по возможности меньшей высоты. Очевидно, что, используя порядка $n^2 r$ процессоров, правую часть соотношений можно вычислить примерно за $\log_2 n$ шагов. Это дает параллельный алгоритм высоты порядка $s\log_2 nr$.

Обозначим матрицу через $Q_i$, вектор слева через $y_i$. Тогда будем иметь

$$y_i = Q_i y_{i-1} = .. = (Q_i Q_{i-1} .. Q1) y0.$$

Матрицы $Q_i$ и векторы $y_i$ имеют порядок $nr + 1$. Согласно алгоритму сдваивания, все произведения $(Q_i Q_1)$ $y_0$ можно вычислить за $\log_2(s+ 1)$ макрошагов, используя при этом $nr + 1$ макропроцессоров, выполняющих в качестве макрооперации умножение двух матриц порядка $nr + 1$.

## Порядок выполнения лабораторной работы

1. Знакомство со всеми разделами руководства.
2. Получение у преподавателя задания на умножение двух матриц заданного вида.
3. Выбор модели вычислений.
4. Построение последовательной программы.
5. Построение параллельной программы.
6. Решение на компьютере последовательной и параллельной программ.
7. Анализ ускорения вычислений в параллельной программе.
8. Исследование факторов и параметров, влияющих на ускорение перемножения двух матриц.
9. Оформление отчета.

## Содержание отчета о выполненной работе

Отчет должен содержать следующее:

1. Название и цель работы.
2. Исходные данные.
3. Схему алгоритма перемножения матриц.
4. Последовательную программу перемножения.
5. Параллельную программу перемножения.
6. Таблицы времен вычисления результата перемножения матриц при различных размерностях и наполненности матриц.

7. Выводы об эффективных способах перемножения матриц в зависимости от их вида.

## Лабораторная работа № 8 Практическая работа в компьютерных сетях при проведении облачных вычислений

### Краткая теория

**Цель работы.** Изучение методики работы в облачных системах.

Облачные вычисления –Cloud Computing – это технология предоставления информационных услуг в виде высоко масштабируемых сервисов.

Ресурсы облака:

Аппаратная часть в дата-центрах:

- Серверы
- Системы хранения данных
- Сетевое оборудование

Программное обеспечение:

- Нативные ОС и гостевые (эмулируемые) ОС
- Программные средства управления серверами и сетевым оборудованием
- Компиляторы, трансляторы, интерпретаторы
- Среды кодирования, отладки и размещения приложений в сети
- Прикладные программы общего назначения (офисное ПО, СУБД и др.)

Возможности облаков:

- Все хранится, обновляется и контролируется в облаке
- Автоматическая антивирусная защита облачных ресурсов
- Практически безразмерное хранилище данных для приложений

- Автоматическое перераспределение вычислительных ресурсов
- Взаимная совместимость различных операционных систем
- Коллективный доступ пользователей к документам и другим данным
- Повышенная надежность хранения данных и программ
- Виртуализация аппаратных и программных средств
- Доступность ресурсов облака с любого устройства, подключенного к Internet.

Виды облаков.

1. Частное облако (Private cloud).

Облако обслуживает одну организацию. Его инфраструктура может управляться самой организацией или третьей стороной и может существовать как на стороне потребителя, так и у внешнего провайдера.

2. Публичное облако (Public cloud).

Облако доступно для всех или для большой группы потребителей, принадлежащих к одной области деятельности. Такое облако принадлежит организации, реализующей соответствующие облачные услуги и предоставляющей облачные сервисы.

3. Гибридное облако (Hybrid cloud).

Облако является композицией частных и общих облаков, остающихся уникальными сущностями, но объединенными вместе возможностью портирования данных и приложений между собой.

Виды облачных сервисов.

1. Инфраструктура как сервис – Infrastructure as a Service (IaaS).

Это предоставление аппаратных средств, ОС и системного ПО на время их фактического использования, за которое производится оплата. IaaS основана на технологии виртуализации серверов и ПО, а также автоматическом масштабирова-

нии и автобалансировке (Amazon Web Service и др.).

2. Платформа как сервис – Platform as a Service (PaaS).

Предоставление для разработчиков приложений программной платформы, средств создания, отладки, тестирования и публикации web-приложений в Internet (Google App Engine, Widows Azure). Используя такую платформу, программисты потребителя сами разрабатывают в этой среде собственные программы и размещают их в облаке. Такой сервис нужен для разработки и развертывания собственного облачного программного обеспечения. Сервис Google App Engine (GAE) позволяет в его среде создавать и размещать в нем собственные приложения пользователя на языках Java, Python и Go.

3. Программное обеспечение как сервис – Software as a Service (SaaS).

Предоставление конечному пользователю готовых прикладных программ с оплатой по факту их использования (Google Apps, Microsoft Office 365 и др.).

Потребителю предоставляются программные средства - приложения провайдера, выполняемые на облачной инфраструктуре. Приложения доступны с различных клиентских устройств через интерфейс тонкого клиента, такой как браузер (например, электронная почта с web-интерфейсом). Облачными ресурсами могут пользоваться не только настольные компьютеры или ноутбуки, но и любые мобильные устройства, подключаемые к интернету. Это могут быть планшетники, таблетки, айфоны, айпады и другие устройства.

Потребитель не управляет и не контролирует саму облачную инфраструктуру, на которой выполняется приложение, будь то сети, серверы, операционные системы, системы хранения или даже некоторые специфичные для приложений возможности. В ряде случаев, потребителю может быть предоставлена возможность доступа к некоторым пользовательским конфигурационным настройкам.

Наибольшие возможности и удобства предоставляет сервис Google Apps, в который входит множество приложений, часть из которых бесплатна, например:

- Gmail (mail.google.com) - электронная почта,

- Документы (docs.google.com) - набор приложений, где можно создавать и работать с текстовыми, табличными и графическими документами, создавать свои HTML-формы, создавать каталожную систему в виде коллекций и размещать созданные документы;

- Сайты (sites.google.com) - где можно создавать и размещать в облаке свои собственные сайты, доступные в Интернет.

## Порядок выполнения лабораторной работы

1. Знакомство со всеми разделами руководства.
2. Получение доступа к облачной системе вычислений.
3. Получение у преподавателя задания на решение вычислительной задачи в облачной системе.
4. Выбор модели вычислений.
5. Построение последовательной программы.
6. Построение параллельной программы.
7. Запуск на решение в облако и получение результатов вычислений.
8. Анализ ускорения вычислений в параллельной программе.
9. Исследование факторов и параметров, влияющих на ускорение при решении заданной математической задачи.
10. Оформление отчета.

## Содержание отчета о выполненной работе

Отчет должен содержать следующее:

1. Название и цель работы.
2. Исходные данные.
3. Схему алгоритма решения вычислительной задачи.
4. Последовательную программу для задачи.
5. Параллельную программу для задачи.
6. Таблицы времен вычисления при различных размерностях и других параметрах.
7. Выводы об эффективности вычислений в облачной системе.

# Список литературы

## Основная литература


1. Гергель, В.П. Теория и практика параллельных вычислений [Текст] : учеб. пособие / В. П. Гергель. - М. : Интернет-Ун-т Информ.Технологий: БИНОМ. Лаб.знаний, 2007. - 423 с. - ISBN 978-5-94774-6 45-7.
2. Орлов, С.П. Организация компьютерных систем [Текст]: учеб. пособие / С.П.Орлов, Н.В.Ефимушкина. – Самара: Самарский гос. техн. ун-т, 2011. – 188 с. – ISBN 978-5-7964-1451-4.


## Дополнительная литература


1. Миллер, Р. Последовательные и параллельные алгоритмы [Текст] : общ.подход:Пер.с англ. / Р.Миллер,Л.Боксер. - М. : БИНОМ.Лаб.знаний, 2006. - 406 с. - ISBN 5-94774-325-6.
2. Барский, А.Б. Параллельные информационные технологии [Текст] : учеб.пособие / А. Б. Барский. - М. : Интернет-Ун-т Информ.Технологий:БИНОМ.Лаб.знаний, 2007. - 502 с. - ISBN 978-5-9556-00 71-0.
3. Гергель, В.П.. Лекции по параллельным вычислениям [Текст]: учеб. пособие /В.П.Гергель, В.А.Фурсов – Самара: Изд-во Самар. гос. аэрокосм. ун-та, 2009. – 164 с.
4. Ефимушкина, Н.В. Вычислительные системы и комплексы [Текст] : учеб. пособие / Н.В.Ефимушкина, С.П.Орлов. - М. : Машиностроение-1, 2006. - 268 с. – ISBN 5-94275-281-8.