Федеральное агентство по образованию

Государственное образовательное учреждение
высшего профессионального образования

Владимирский государственный университет

Кафедра вычислительной техники

ВЫЧИСЛИТЕЛЬНЫЕ СРЕДСТВА РАСПРЕДЕЛЕННЫХ АВТОМАТИЗИРОВАННЫХ СИСТЕМ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ

Составитель
В. И. БЫКОВ

Владимир 2010

УДК 004.382(075.8)
ББК 32.973.2-02я 7
В92

Рецензент
Доктор технических наук, профессор,
зав. кафедрой информатики и защиты информации
Владимирского государственного университета
М. Ю. Монахов

Печатается по решению редакционного совета
Владимирского государственного университета

Вычислительные средства распределенных автоматизированных систем : метод. указания к лаб. работам / Владим. гос. ун-т; сост. В.И. Быков. − Владимир: Изд-во Владим. гос. ун-та, 2010. – 28 с.

В92

Содержат четыре лабораторные работы по дисциплине «Вычислительные средства распределенных автоматизированных систем», требования к оформлению отчета и вопросы для контроля знаний. Тематика работ охватывает все разделы дисциплины. Основное внимание уделено оценке производительности ВС и средствам ее повышения.

Предназначены для магистрантов направления 230100 – информатика и вычислительная техника (магистратура) и могут быть использованы студентами любых форм обучения

Ил. 7. Табл. 3. Библиогр.: 4.

УДК 004.382(075.8)
ББК 32.973.2-02я 7

ВВЕДЕНИЕ

Дисциплина «Вычислительные средства распределенных автоматизированных систем» является важной в подготовке магистрантов по направлению «Информатика и вычислительная техника». Она знакомит их с основами параллельных вычислений и проблемами, возникающими при этом.

Методические указания содержат четыре лабораторные работы, в которых магистранты изучают оценку производительности компьютеров, распараллеливание задачи на этапе программирования, конфликты, возникающие при конвейерной обработке и способы их разрешения, как программные, так и аппаратные.

ЛАБОРАТОРНАЯ РАБОТА № 1

ОЦЕНКА ПРОИЗВОДИТЕЛЬНОСТИ ЭВМ И ЕЕ УСТРОЙСТВ И УЗЛОВ

Цель работы: Изучение способов и методик оценки производительности компьютера и его устройств и узлов.

Выполнение работы

1. Изучить способы и методики оценки производительности компьютера и его устройств и узлов, используя литературу и материалы лекций.

2. Разработать методику оценки производительности компьютера или его устройства или узла по заданию преподавателя.

3. Разработать алгоритм и программу для оценки производительности компьютера или его устройства или узла.

4. Выполнить программу с несколькими наборами исходных данных, определяя каждый раз время выполнения. Количество повторений заданных операций должно быть большим и изменяться от 100 или 1000 до 1 млрд. с кратностью 10, т. е. в логарифмическом масштабе. Количество экспериментов, т. е. точек

на графике – не менее 5. Для измерения времени использовать процедуры или функции, имеющиеся в языках программирования. Например в Паскале, процедура

GetTime(h,m,s,d)

дает значение текущего времени, где *h* – часы; *m* – минуты; *s* – секунды; *d* – сотые доли секунд (все параметры – типа *word*).

Время выразить в сотых или тысячных долях секунды.

5. Повторить эксперименты на другом компьютере, имеющем другую конфигурацию и параметры производительности.

6. Вычислить (или получить на компьютере) некоторые оценки производительности компьютера или его устройства или узла.

7. Составить таблицу и построить графики полученных зависимостей. При построении графиков на формате А4 по горизонтальной оси абсцисс должно быть количество повторений в логарифмическом масштабе (не менее пяти точек), по вертикальной оси ординат на всю высоту листа – время в сотых или тысячных долях секунды. При малом разрешении по вертикали можно построить два графика – для малых и больших значений времени.

Примечания:

1. При выполнении теста компьютер не должен быть загружен другими задачами.

2. При программировании необходимо использовать язык более низкого уровня (Ассемблер, Паскаль, Си, а не Delphi, Java).

3. При выполнении операций результаты должны иметь тот же тип, что и операнды, чтобы не было затрат на преобразование типов.

4. При выполнении операций результаты не должны вызывать переполнение.

Содержание отчета

1. Цель работы.
2. Методика оценки.
3. Алгоритм.

4. Программа.

5. Условия экспериментов (детальное описание параметров и конфигурации ВС).

6. Таблицы и графики полученных зависимостей.

7. Выводы.

Темы индивидуальных заданий

1. Производительность дисковых операций (чтение и запись).

2. Сравнительная производительность при выполнении операций с действительными (2 вида) и целыми числами.

3. Сравнительная оценка производительности двух различных ВС (машин) при выполнении различных операций, а именно:

а) короткие операции с действительными (3 вида) числами;

б) длинные операции с действительными (3 вида) числами;

в) короткие операции с целыми (3 вида) числами;

г) длинные операции с целыми (3 вида) числами.

4. Сравнительная оценка накладных расходов на организацию циклов различного типа (*for, while, repeat*) в различных языках программирования: Ассемблер (а), Паскаль (б), Си (в).

5. Сравнительная производительность при выполнении операций преобразования типов:

а) различные целые типы – в различные действительные;

б) различные действительные типы – в другие действительные.

6. Производительность матричных операций с числами различных типов.

7. Производительность векторных операций с числами различных типов.

Вопросы для контроля

1. Оценить производительность ЭВМ (проблемы, требования).

2. Составляющие производительности ЭВМ.

3. Опишите методы оценки производительности ЭВМ.

ЛАБОРАТОРНАЯ РАБОТА № 2

РАСПАРАЛЛЕЛИВАНИЕ ЗАДАЧИ НА ОСНОВЕ РАСЩЕПЛЕНИЯ ЦИКЛА

Цель работы: Изучение методик и способов распараллеливания вычислений с использованием метода расщепления цикла.

Теоретические сведения

Основной характеристикой любого компьютера является производительность, т. е. количество операций, выполняемых в единицу времени. Существуют два направления повышения производительности: повышение частоты процессора на основе достижений технологии и распараллеливание действий (операций, команд, микрокоманд) за счет увеличения аппаратных ресурсов. Повышение частоты процессора становится все более сложным исходя из физических принципов. Принцип распараллеливания применяется в процессорах уже давно на основе конвейеризации. Однако дальнейшее увеличение ступеней конвейера затруднительно в связи с многочисленными конфликтами [1, 2]. Поэтому в настоящее время переходят к мультипроцессорным компьютерам. Практически все новые процессоры являются многоядерными.

Мультипроцессорные компьютеры повышают производительность лишь в том случае, когда программа позволяет обеспечить распараллеливание выполнения операций или их групп. В противном случае производительность мультипроцессорного компьютера повышается незначительно [3]. Распараллеливание может быть выполнено на различных уровнях.

Уровни распараллеливания

1. Уровень заданий – наиболее легко осуществлять распараллеливание. Недостатки уровня распараллеливания: небольшая надежность; ситуации, когда один процессор решает длинную задачу, а остальные процессоры простаивают.

2. Распараллеливание на уровне процедур. Требуется анализ возможностей распараллеливания.

3. Распараллеливание на уровне операций или команд. Пример: распараллеливание циклических операций.

4. Распараллеливание на арифметическом уровне:

а) на уровне арифметических операций за счет уменьшения высоты дерева вычислений;

б) параллельная обработка ряда битов (секционированные микропроцессоры, например К1804).

В реальных задачах наиболее трудоемкими являются задачи обработки векторов и матриц. Эти задачи сводятся в программе к циклическим операциям. Распараллеливание циклических операций возможно на основе расщепления цикла, т. е. выполнения операций тела цикла с различными индексами на разных процессорах. В простейшем случае, когда количество итераций цикла N равно количеству процессоров K, на каждом процессоре тело цикла будет выполняться один раз. В случае $N<K$ $K - N$ процессоров будут свободны; при $N>K$ на каждом процессоре будет выполняться $L=N/K$ повторов тела цикла (дробное значение L необходимо округлить до ближайшего большего целого). Расщепление цикла возможно на основе двух подходов к распределению индексов между процессорами – последовательном и поблочном. Для случая N=10 и K=3 распределение индексов представлено в табл. 1.

Табл. 1. Распределение индексов между процессорами

№ процессора	Последовательное распределение				Поблочное распределение			
	№ повтора				№ повтора			
	1	2	3	4	1	2	3	4
	№ индекса				№ индекса			
1	1	4	7	10	1	2	3	4
2	2	5	8		5	6	7	
3	3	6	9		9	10		

Выбор подхода к распределению зависит от алгоритма и возможностей дальнейшей загрузки процессоров. Пример распараллеливания на основе расщепления цикла приведен после вопросов для контроля.

Оценка параллельных мультипроцессоров (метрика параллельных вычислений)

1. Степень параллелизма *D(t)* – количество процессоров, участвующих в вычислениях в каждый момент времени.

2. В любой программе не может быть такой ситуации, что все вычисления можно сделать параллельно, *f* – процент обработки последовательной части.

3. Δ – производительность.

4. *n* – общее число процессоров (процессоры одинаковые).

Одним из показателей возможности распараллеливания программы является профиль параллелизма программы, определяющий в каждый момент времени количество процессоров, на которых программа может выполняться параллельно.

От профиля зависит общий объем вычислений за интервал времени:

$$W = \Delta \int_{t_n}^{t_k} D(t)dt = \Delta \sum_{i=1}^{m} i \cdot t_i$$

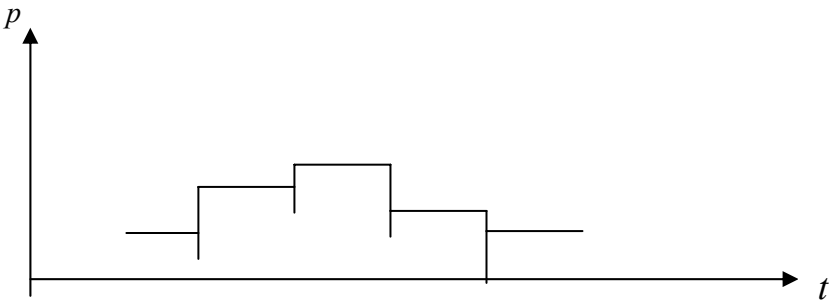


Рис. 1. Пример параллелизма профиля программы

Максимальный параллелизм профиля – *m*.

5. *O(n)* – общее количество операций, выполняемых на многопроцессорных ВС.

6. *T(n)* – время, затраченное на выполнение общего количества операций в виде числа квантов времени.

В однопроцессорных ВС $T(1)=O(1)$, если учесть что квант времени равен длительности одной операции. При $n>=2$ $T(n)<O(n)$.

7. Ускорение – это отношение времени выполнения на однопроцессорной машине к времени выполнении на многопроцессорной $S(n)=\frac{T(1)}{T(n)}$ $S(n)<n$.

8. Эффективность ускорения на однопроцессорной машине $E(n)=\frac{S(n)}{n}<1=\frac{T(1)}{n\cdot T(n)}$ $\frac{1}{n}\le E(n)\le n$.

9. Избыточность $R(n)=\frac{O(n)}{O(1)}=\frac{1}{E(n)}-1$, где $1\le R(n)\le n$.

10. Коэффициент полезного использования или утилизации $U(n)=R(n)\cdot E(n)$.

Издержки при использовании параллельных компьютеров по отношению к однопроцессорным:

1. программные – дополнительные индексные вычисления при декомпозиции данных;

2. дисбаланс загрузки процессора – один процессор ждет завершения операции другими процессами;

3. коммуникационные издержки (время коммуникаций между процессорами).

Для уменьшении издержек желательно, чтобы вычислительные гранулы были по возможности больше.

Оценка времени ожидания

Если существует время установки канала T_s, позиции информации P, количества каналов b, то $T_{ож}=T_S+\frac{P}{b}$ – при симплексной связи; $T_{ож}=T_S+2\cdot\frac{P}{b}$ – при дуплексной связи.

При прохождении дополнительных коммутаторов: $T_{ож}=T_a+n\cdot(\frac{P}{b}+T_d)+\frac{P}{b}$, где n – количество коммутаторов; T_d – время прохождения коммутатора; T_a – время установления $T_a<T_S$.

Качество параллельных вычислений**:**

$$Q(n) = \frac{S(n) \cdot E(n)}{R(n)} = \frac{T^3}{n \cdot T^2(n) \cdot O(n)}$$, где $Q(n)<=S(n)$

Закон Амдала: $T_p=f \cdot T_s+((1-f) \cdot T_S)/n)$ (справедлив для малых f);
где T_s – время выполнения задачи на последовательном компьютере;
f – доля последовательных вычислений.

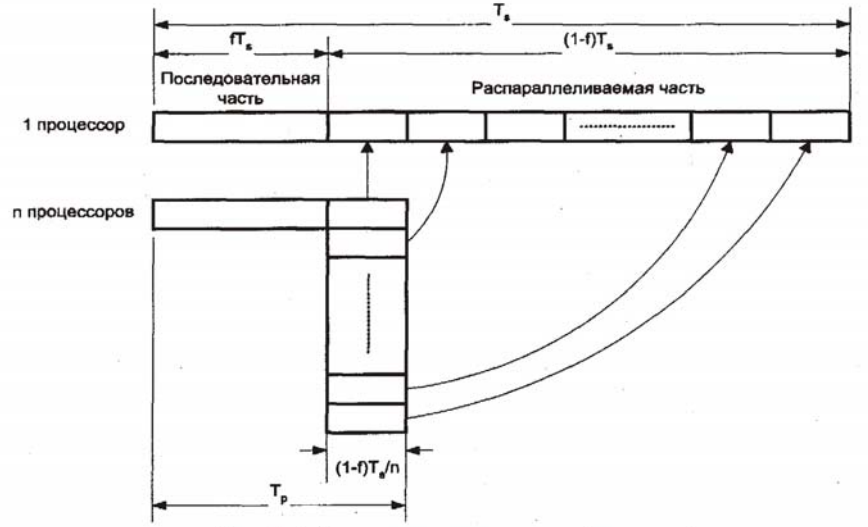


Рис. 2. Диаграмма распределения времени
в соответствии с законом Амдала

Выполнение работы

1. Изучить способы и методики распараллеливания вычислений с использованием метода расщепления цикла, используя литературу и материалы лекций.

2. Разработать алгоритмы и программы для оценки затрат времени при последовательном и псевдопараллельном решении задачи (по вариантам).

3. При псевдопараллельном выполнении задачи необходимо разбить операции на группы так, чтобы одна группа выполня-

лась как бы на одном процессоре, другая – на втором и т. д. В мультипроцессорной ВС эти группы операций будут выполняться параллельно, т. е. одновременно на разных процессорах. При пседопараллельном выполнении они должны выполняться последовательно на одном процессоре. При этом необходимо определять время выполнения каждой группы и принять максимальное из них как время выполнения при параллельном режиме.

4. Выполнить программу с несколькими значениями размера матрицы (вектора) N и с различным количеством процессоров K, определяя каждый раз время выполнения. Размеры матрицы N должны быть большими и изменяться от 10 или 100 до 1 млн. с кратностью 10, т. е. в логарифмическом масштабе; размеры вектора N должны изменяться от 1000 или 10000 до 1млрд. с кратностью 10. Количество процессоров K должно изменяться от 2 до 1024 с кратностью 2, т. е. должно быть 10 значений.

Для измерения времени использовать процедуры или функции, имеющиеся в языках программирования. Например в Паскале, процедура

GetTime(h,m,s,d)

дает значение текущего времени, где h – часы; m – минуты; s – секунды; d – сотые доли секунд (все параметры – типа *word*).

Время выразить в сотых или тысячных долях секунды.

5. Повторить эксперименты на другом компьютере, имеющем другую конфигурацию и параметры производительности.

6. Вычислить (или получить на компьютере) оценки затрат времени.

7. Составить таблицу и построить семейства графиков полученных зависимостей в двух системах координат:

1) $t = f(N)$ при различных K;

2) $t = f(K)$ при различных N; где N – размер матрицы (вектора); K – количество процессоров.

При построении графиков на формате А4 по горизонтальной оси абсцисс должно быть количество повторений в логариф-

мическом масштабе (не менее 5 точек), по вертикальной оси ординат на всю высоту листа – время в сотых или тысячных долях секунды. При малом разрешении по вертикали можно построить два графика – для малых и больших значений времени.

Примечания:

1. При выполнении теста компьютер не должен быть загружен другими задачами.

2. При программировании необходимо использовать язык, позволяющий организовать псевдопараллельную работу, например, создавать различные потоки (Си++, *Delphi, Java*).

Содержание отчета

1. Цель работы.
2. Методика оценки.
3. Алгоритм.
4. Программа.
5. Условия экспериментов (детальное описание параметров и конфигурации ВС).
6. Таблицы и графики полученных зависимостей.
7. Выводы.

Темы индивидуальных заданий

1. Умножение матриц.
2. Сложение матриц.
3. Расчет обратной матрицы.
4. Транспонирование матрицы.
5. Почленное сложение векторов.
6. Почленное умножение векторов.
7. Решение СЛАУ.
8. Решение СНАУ.
9. Решение СОДУ (заданным методом).
10. Расчет кратного интеграла методом Монте-Карло.
11. Расчет определителя матрицы.
12. Расчет скалярного произведения векторов.

Вопросы для контроля

1. Компьютеры параллельного действия – уровни параллелизма.
2. Оценки компьютеров параллельного действия.
3. Издержки компьютеров параллельного действия.
4. Способы распараллеливания программ.
5. Два подхода к расщеплению цикла.

Пример распараллеливания на основе расщепления цикла

Условие задачи: сложение векторов a и b.

For *i*:=1 to *M* do
r[*i*]:= *a*[*i*]+*b*[*i*];

Распараллеливание
K – количество процессоров;
L – количество повторов на 1 процессоре;
L=*M*/*K* +1 (если *M*/ *K* не целое число);
L:=*M* div *K*;
If (*M* mod *K*) <> 0 then *L*:=*L*+1.

1-й вариант – последовательное исполнение:

```
for mc:=1 to K do
begin
        {исполняется на каждом прц}
        j:=(mc − 1)·K+npc;
        while j <= M do
        begin
                r[j]:= a[j] + K[j];
                j:=j+K;
        end;
end;
```

2-й вариант – поблочное исполнение с длиной блока L:

```
for kc:=1 to K do
begin
        {исполняется на каждом прц}
        jn:=1+(kc − 1)·L;
        jk:=jn+L − 1;
        if jk > M then jk:=M;
        for j:=jn to jk do
                r[j]:= a[j] + b[j];
  end;
```

ЛАБОРАТОРНАЯ РАБОТА № 3

КОНФЛИКТЫ ПРИ КОНВЕЙЕРНОЙ ОБРАБОТКЕ ДАННЫХ В ПРОЦЕССОРЕ

Цель работы: Ознакомление с принципами конвейерной обработки данных в процессоре. Изучение конфликтов при конвейерной обработке методов и способов их предотвращения и разрешения.

Теоретические сведения

Конфликты при работе конвейеров

1. Структурные конфликты (риски), возникают из-за конфликтов по ресурсам при попытке нескольких команд одновременно обратиться к одному и тому же ресурсу вычислительной машины (например к памяти).

2. Конфликты по данным, возникают в случае, когда выполнение одной команды зависит от результата выполнения предыдущей команды.

3. Конфликты по управлению, возникают при конвейеризации команд переходов и других команд передачи управления (вызов процедуры, прерывание и т. п.).

Разрешение конфликтов

1. Структурные конфликты возникают не часто в связи с тем, что память кэшируется. В некоторых случаях возникает необходимость дублирования некоторых ресурсов.

2. Конфликты по данным возникают при логической зависимости между командами. Для решения зависимости необходимо выбирать команды специальным образом.

Виды конфликтов по данным

1. Чтение после записи (ЧПЗ – *RAW*). Существует две команды и, если одна команда пытается прочитать операнд используемых данных прежде, чем другая команда туда запишет, возникает конфликт.

2. Запись после чтения (ЗПЗ – *WAR*). Одна команда пытается записать результат в приемник раньше, чем другая команда прочитывает эти данные.

3. Запись после записи (ЗПЗ – *WAW*). Одна команда пытается записать операнд прежде, чем будет записан результат другой команды, т. е. записи заканчиваются в неверном порядке.

Признак конфликта

Нарушение хотя бы одного условия Бернстайна, служащее критерием возможности возникновения конфликта по данным.

1. ЧПЗ

$O(i) \cap I(j)=0,$

где $O(i)$ – множество ячеек, при команде i; $I(j)$ – множество ячеек, при команде j, следующие за командой i.

2. ЗПЧ

$I(i) \cap O(j)=0,$

где *I-input; O-output.*

3. ЗПЗ

$O(i) \cap O(j)=0.$

Методы устранения конфликтов по данным

1. Программные:

а) на этапе трансляции создается объектный код, в котором между командами, склонными к конфликту по данным, вставляются пустые операции;

б) на этапе трансляции создается объектный код, в котором команды i и j, склонные к конфликту по данным, переставляются

так, чтобы между ними были команды, не связанные по данным с этими командами; при этом команда *i* может быть перенесена вперед – что более затруднительно, или команда *j* переставлена назад, что легче выполнимо.

2. Аппаратные средства:

а) разрешают конфликт во время выполнения, т. е. команда *j* задерживается на несколько тактов, чтобы *i* успела завершиться и миновать ступени конвейера, вызвавшие конфликт;

б) ускоренное продвижение требуемой информации *forwarding data* (табл. 2).

Табл. 2. Пример ускоренного продвижения информации

Add R1, R2, R3	IF	ID	EX	MEM	WB				
SUB R4, R1 R5		IF	ID	EX	MEM	WB			
AND R6, R1, R7			IF	ID	EX	MEM	WB		
OR R8, R1, R9				IF	ID	EX	MEM	WB	
XOR R10, R1,R11					IF	ID	EX	MEM	WB

В регистр информация записывается упреждающим способом. Для этого существует специальные схемы АЛУ с цепями обхода и ускоренной пересылкой.

3. Оптимизация с помощью компилятора*: $a = в + c$; $d = e + f$* (табл. 3).

Табл. 3. Пример оптимизации

Неоптимальное	Оптимальное
LW R_b, *b*	*LW* R_b, *b*
LW R_c, *C*	*LW* R_c, *C*
ADD R_a, R_b, R_c	*LW* R_e, *e*
SW a, R_a	*ADD* R_a, R_b, R_c
LW R_e, *e*	*LW* R_f, *f*
LW R_f, *f*	*SW a*, R_a
SUB R_d, R_e, R_f	*SUB* R_d, R_e, R_f
SW d, R_d	*SW d*, R_d

В простейшем алгоритме компиляции компилятор планирует распределение команд в одном и том же базовом блоке (это линейный участок кода, в котором нет переходов).

4. Использование временных регистровых результатов. Существуют логические регистры и физические регистры. Временные значения записываются в файловые регистры вместе с постоянными. Временные значения становятся новым постоянным только после фиксации результата (пятая ступень конвейера). Завершение выполнения команды происходит тогда, когда все предыдущие команды завершились успешно в заданном программой порядке. Такой подход называется методом переименования регистров.

Конфликты по управлению

Это неоднозначность при выборке следующей команды в случае перехода.

Методы борьбы:

1) буферы предвыборки;
2) множественные потоки;
3) задержанный переход;
4) предсказание перехода.

Наиболее часто используется предсказание перехода, для которого характерны следующие стратегии.

1. Статическое определение направления перехода:

- переход происходит всегда (ПВ), т. е. переход выполняемый, применяется при организации цикла;
- переход не происходит (ПНВ), т. е. переход невыполняемый;
- предсказание определяется по результатам профилирования (это тестовые прогоны программы);
- предсказание определяется кодом операции команды перехода;
- предсказание зависит от направления перехода (вперед или назад);
- при первом выполнении переход выполняется всегда.

2. Динамическое определение направления перехода выполняется на основе прогноза.

Прогнозы различаются по глубине:

- однобитовый – т. е. такой, какой был на предыдущем шаге. Осуществляется на основе автомата.

- двухбитовый – т. е. на основе двух предыдущих выполнившихся переходов.

Выполнение работы

1. Изучить виды конфликтов при конвейерной обработке данных в процессоре и методы и способы их предотвращения и разрешения, используя литературу и материалы лекций.

2. Разработать алгоритмы и программы, позволяющие обнаружить, и либо предотвратить конфликт, либо выйти из него.

3. Выполнить программу с несколькими (2–4) наборами исходных данных.

Примечания:

1. При программировании можно использовать любой язык.

2. Фрагмент программы, в котором возможен конфликт, должен быть написан на языке Ассемблера для RISC-процессора, т. е. с использованием только регистровых операндов и специальных операций работы с памятью.

Содержание отчета

1. Цель работы.

2. Теоретические сведения о возникновении и разрешении конфликта.

3. Алгоритм обнаружения и предотвращения конфликта.

4. Программа обнаружения и предотвращения конфликта.

5. Условия экспериментов (детальное описание параметров и конфигурации ВС).

6. Фрагменты исходных программ и программ, полученных в результате преобразования, в которых конфликты устранены.

7. Выводы.

Темы индивидуальных заданий

Вид конфликта, который надо разрешить, и способ разрешения:

1. Конфликт по данным типа ЧПЗ.
2. Конфликт по данным типа ЗПЧ.
3. Конфликт по данным типа ЗПЗ.
4. Конфликт по управлению – статическое предсказание выполняемого перехода.
5. Конфликт по управлению – статическое предсказание невыполняемого перехода.
6. Конфликт по управлению – статическое предсказание выполняемого перехода.
7. Конфликт по управлению – динамическое предсказание на основе однобитового прогноза перехода.
8. Конфликт по управлению – динамическое предсказание на основе двухбитового прогноза перехода.

Вопросы для контроля

1. Конвейерная организация. Типы конфликтов в конвейере.
2. Конвейерная организация. Структурные конфликты и способы минимизации потерь при них.
3. Конвейерная организация. Конфликты по данным и способы их разрешения.
4. Конвейерная организация. Конфликты по данным – классификация.
5. Конвейерная организация. Конфликты по данным – методика планирования компилятора для их устранения. Средства динамической оптимизации.
6. Конвейерная организация. Методы снижения потерь на выполнение команд перехода.
7. Конвейерная и суперскалярная обработка. Аппаратное прогнозирование направления переходов и снижение потерь на организацию переходов.

8. Виды конфликтов по данным при работе конвейера и способы их устранения.

9. Виды конфликтов по управлению при работе конвейера и способы их устранения.

10. Автомат двухбитового предсказания.

ЛАБОРАТОРНАЯ РАБОТА № 4

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА ТОМАСУЛО

Цель работы: Исследовать алгоритм Томасуло и реализовать его программно.

Теоретические сведения

Обобщенное словесное описание алгоритма

Программа представляет собой программную модель архитектуры арифметико-логического устройства с плавающей точкой (ПТ), разработанную Томасуло.

Устройство состоит:

- из очереди операций ПТ;
- буфера загрузки из памяти;
- буфера записи в память;
- регистров ПТ;
- станций резервирования сложения;
- станций резервирования умножения;
- устройства сложения;
- устройства умножения;
- ОШД (общей шины данных).

Шаг прохода алгоритма Томасуло

На каждом шаге происходит просмотр состояния каждого из устройств.

1. Просмотр очереди команд.

Из очереди забирается наиболее возможное число команд при условии наличия свободных буферов загрузки, записи, станций резервирования сложения, умножения.

2. Просмотр буфера загрузки.

Если есть активный буфер загрузки, то он продолжает читать память еще один такт, если он закончил чтение, то результат выдается на ОШД, и буфер освобождается. Активного буфера загрузки теперь нет.

Если нет активного буфера загрузки, то он ищется. Если находится буфер, готовый к выполнению (с приоритетом 0), то он начинает выполняться и становится активным.

3. Просмотр устройства сложения.

Так как у нас буферное устройство сложения, то оно параллельно может выполнять несколько команд сложения/вычитания. Поэтому каждая операция, выполняемая в нем, продолжает выполняться еще один такт. Если какая-то операция выполнена, то результат выдается на ОШД, если она свободна, а если занята – то результат сохраняется в устройстве сложения. Освобождаются станции резервирования сложения, где хранились операции, завершившиеся в этом такте.

4. Просмотр устройства умножения.

Так как у нас буферное устройство умножения, то оно параллельно может выполнять несколько команд умножения/деления. Поэтому каждая операция, выполняемая в нем, продолжает выполняться еще один такт. Если какая-то операция выполнена, то результат выдается на ОШД, если она свободна, а если занята – то результат сохраняется в устройстве умножения. Освобождаются станции резервирования умножения, где хранились операции, завершившиеся в этом такте.

5. Просмотр ОШД.

Если на ОШД есть результат выполнения какой-либо операции, то все устройства, ждущие операнда, забирают значение с ОШД, если это ожидаемый операнд.

6. Просмотр станции резервирования сложения.

Если после просмотра ОШД какие-либо операции получили ожидаемый операнд и готовы к выполнению, то они отправляются на выполнение в устройство сложения.

7. Просмотр станции резервирования умножения.

Если после просмотра ОШД какие-либо операции получили ожидаемый операнд и готовы к выполнению, то они отправляются на выполнение в устройство умножения.

8. Просмотр буферов записи.

Если есть активный буфер записи, то он продолжает записывать память еще один такт, если он закончил запись, то буфер освобождается. Активного буфера записи теперь нет.

Если нет активного буфера записи, то он ищется. Если находится буфер, готовый к выполнению (имеющий операнд, с приоритетом 0), то он начинает выполняться и становится активным.

Примечание: Блок-схемы работы программы и текст программы приведены в электронном варианте методических указаний.

Интерфейс программы (рис. 3 – 6)

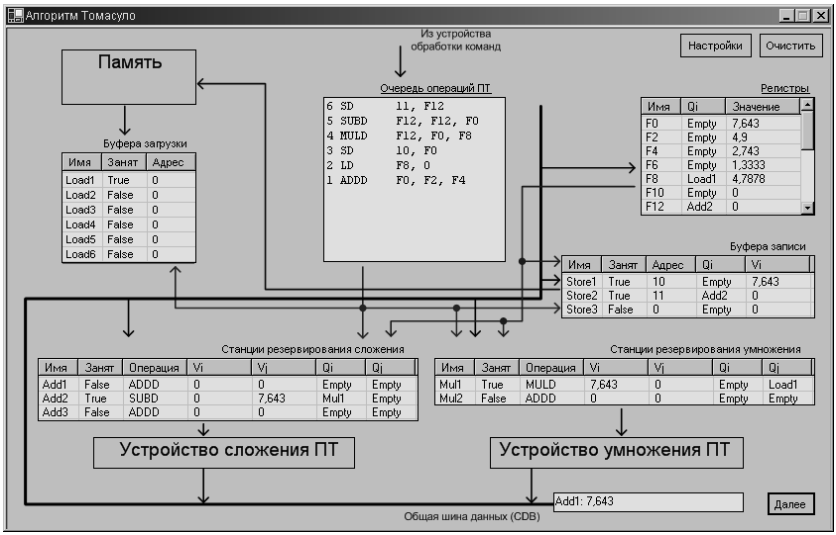


Рис. 3. Окно «Алгоритм Томасуло»

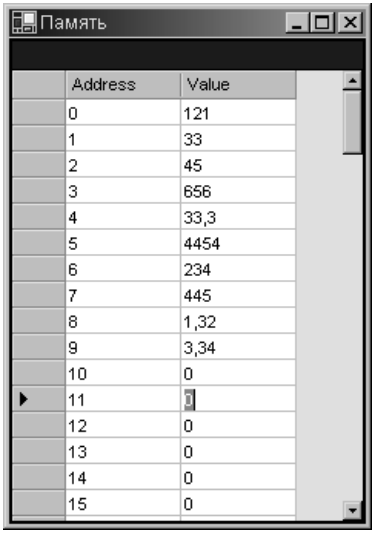


Рис. 4. Окно «Память»

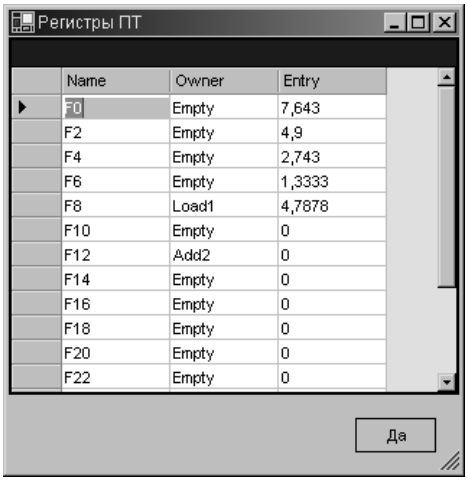


Рис. 5. Окно «Регистры»

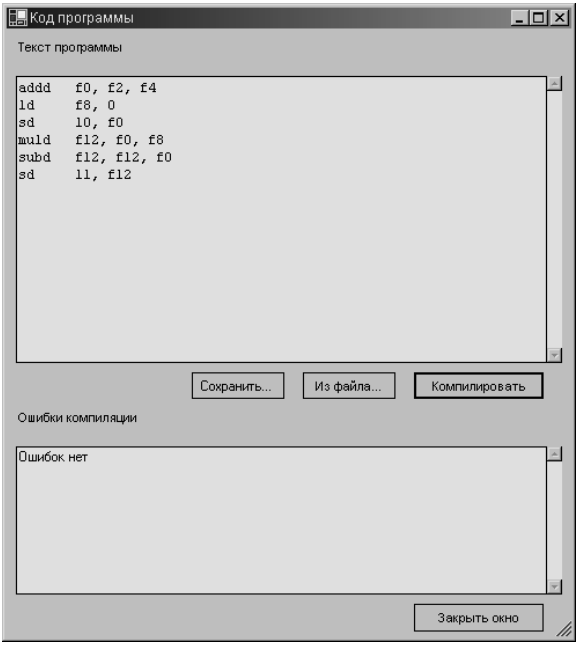


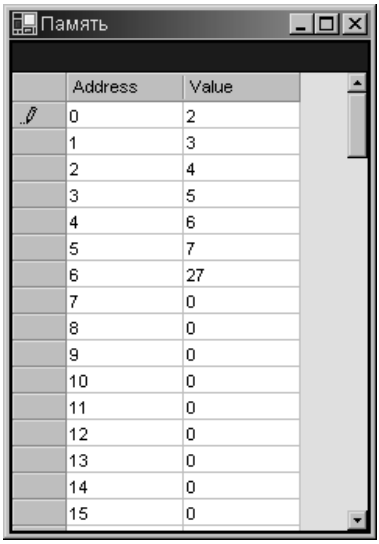
Рис. 6. Окно «Код программы»

Тесты

Код программы:

```
; проверка конфликта типа RAW, WAW (регистры);
; задача: найти сумму первых 6-ти ячеек памяти;
; и записать результат в 7-ю;
; исходные данные: ненулевые значения
; 0-5 ячеек памяти,
; f2 = 0
ld f0, 0
addd f2, f2, f0
ld f0, 1
addd f2, f2, f0
ld f0, 2
addd f2, f2, f0
ld f0, 3
addd f2, f2, f0
ld f0, 4
addd f2, f2, f0
ld f0, 5
addd f2, f2, f0
sd 6, f2
```

После проведения теста окно памяти будет выглядеть как показано на рис. 7.



	Address	Value
	0	2
	1	3
	2	4
	3	5
	4	6
	5	7
	6	27
	7	0
	8	0
	9	0
	10	0
	11	0
	12	0
	13	0
	14	0
	15	0

Рис. 7. Окно памяти после выполнения

Выполнение работы

1. Изучить алгоритм Томасуло и его программную реализацию.

2. Запустить тестовый пример и убедиться в правильной работе программы.

3. Исследовать алгоритм Томасуло и его программную реализацию при обработке программы, содержащей конфликт заданного типа в соответствии с вариантом. Для этого составить программу

на языке Ассемблер, содержащую конфликт заданного типа, и проверить правильность его разрешения.

Варианты заданий – тип конфликта:

1) RAW;
2) WAW;
3) WAR.

Вопросы для контроля

1. Конвейерная организация. Конфликты по данным и способы их разрешения.

2. Конвейерная организация. АЛУ с цепями обхода и ускоренной пересылки.

3. Конвейерная организация. Конфликты по данным – классификация.

4. Конвейерная и суперскалярная обработка. Идеи динамической оптимизации и ее реализация с централизованной схемой обнаружения конфликтов.

5. Алгоритм Томасуло.

Библиографический список

1. Танненбаум Э. Архитектура компьютера / Э. Танненбаум. − 4-е изд. – СПб.: Питер, 2006. – 698 с. – ISBN 5-318-00298-6.

2. Цилькер Б.Я. Организация ЭВМ и систем: учебник для вузов. / Б. Я. Цилькер, С. А.Орлов. – СПб.: Питер, 2004. – 668 с. – ISBN 5-94723-759-8.

3. Организация ЭВМ / К. Хамахер, З. Вранешич, С. Заки. – 5-е изд. – СПб.: Питер; Киев: Издательская группа BHV, 2003. – 848 с. – ISBN 5-8046-0162-8.

4. Распределенные системы. Принципы и парадигмы / Э. Танненбаум, М. ван Стеен. – СПб.: Питер, 2003. – 877 с. – ISBN 5-272-00053-6.

Оглавление

ВЫЧИСЛИТЕЛЬНЫЕ СРЕДСТВА
РАСПРЕДЕЛЕННЫХ АВТОМАТИЗИРОВАННЫХ СИСТЕМ

Методические указания к лабораторным работам

Составитель
БЫКОВ Валерий Ильич

Ответственный за выпуск – зав. кафедрой профессор В.Н. Ланцов

Подписано в печать 15.03.10.
Формат 60x84/16. Усл. печ. л. 1,63. Тираж 100 экз.
Заказ
Издательство
Владимирского государственного университета.
600000, Владимир, ул. Горького, 87.