Министерство образования и науки Российской федерации
Севастопольский государственный университет

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

По выполнению лабораторных работ
по дисциплине
" Математические методы параллельных и распределенных вычислений "
для студентов дневной и очно-заочной форм обучения
направления 09.04.02 "Информационные системы и технологии"

Севастополь
2015

УДК 681.06 + 658.5
Методические указания к лабораторным работам по дисциплине **«Математические методы параллельных и распределенных вычислений»** для студентов дневной и очно-заочной форм обучения направления 09.04.02 "Информационные системы и технологии " /Сост. К. В. Кротов. – Севастополь: Изд-во СевГУ, 2015. – 82 с.

Методические указания предназначены для проведения лабораторных занятий по дисциплине **«Математические методы параллельных и распределенных вычислений»**. Целью настоящих методических указаний является изучение и исследование методов распараллеливания последовательных ациклических программ.

Методические указания составлены в соответствии с требованиями программы дисциплины **«Математические методы параллельных и распределенных вычислений»** для студентов направления обучения 09.04.02 "Информационные системы и технологии" и утверждены на заседании кафедры Информационных систем, протокол №1 от 28 августа 2015г.

Допущено учебно-методическим центром СевГУ в качестве методических указаний.

Рецензент: Шевченко В.И. канд. техн. наук, доц. каф. Информационных технологий и вычислительной техники

СОДЕРЖАНИЕ

ОБЩИЕ ТРЕБОВАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

1. ЦЕЛЬ И ЗАДАЧИ ЛАБОРАТОРНЫХ РАБОТ

Цель настоящих лабораторных работ состоит в исследовании методов формирования параллельных алгоритмов для программ, представленных в последовательном виде. Задачами выполнения лабораторных работ являются:
- углубленное изучение основных теоретических положений дисциплины «Математические методы параллельных и распределенных вычислений»;
- получение практических навыков по распараллеливанию последовательных программ, построению параллельных алгоритмов, написанию программ, реализующих сформированные алгоритмы.

2. ОПИСАНИЕ ЛАБОРАТОРНОЙ УСТАНОВКИ

Объектом исследований в лабораторных работах являются методы формирования параллельных алгоритмов на основе последовательной формы программ, а также языковые средства их реализации.

Лабораторная установка состоит из ПЭВМ, снабженной системой программирования Visual studio и средствами реализации параллельных программ ─ библиотекой MPI.

3. СОДЕРЖАНИЕ ОТЧЕТА

Отчеты по лабораторной работе оформляются каждым студентом индивидуально. Отчет должен включать: название и номер лабораторной работы; цель работы; краткие теоретические сведения; постановку задачи; основные математические выкладки, на основании которых формируются параллельные алгоритмы, последовательное изложение реализации алгоритма распараллеливания программы, текст программы, реализующей задание; распечатку результатов выполнения программы, выводы по результатам исследований.

4.ЗАДАНИЕ НА РАБОТУ

Задание выбирается в соответствии с вариантом, назначаемым преподавателем. Вариант задания предполагает определение некоторого математического выражения, для которого должен быть сформирован параллельный алгоритм, реализуемый в дальнейшем программно.

ЛАБОРАТОРНАЯ РАБОТА №1
ИССЛЕДОВАНИЕ ПОНЯТИЙ НЕЗАВИСИМОСТИ ОПЕРАЦИЙ (БЛОКОВ) В ПОСЛЕДОВАТЕЛЬНЫХ ПРОГРАММАХ. ИССЛЕДОВАНИЯ МЕТОДА ПРИВЕДЕНИЯ ПРОГРАММЫ К ПОЛНОЙ ПАРАЛЛЕЛЬНОЙ ФОРМЕ ПРИ РАСПАРАЛЛЕЛИВАНИИ ПОСЛЕДОВАТЕЛЬНЫХ ПРОГРАММ

1.Цель работы: изучить понятие независимости операций (блоков) в программе, исследовать метод распараллеливания программ путем их приведения к полной параллельной форме.

2. Теоретическое введение

2.1. Понятие зависимостей и независимостей между операторами программы

Два оператора S_i и S_j являются информационно-зависимыми, если S_j выполняется над результатами S_i (если S_i и S_j связаны отношением частичного порядка). Операторы S_k и S_j являются зависимыми от S_i по управлению, если результат выполнения S_i определяет активацию либо S_k, либо S_j (S_i предшествует S_k и S_j, а S_i является управляющим оператором).

Введем следующие обозначения: $in(S_i)$ — множество переменных, являющихся входными для оператора S_i (множество переменных — исходных данных для оператора), $out(S_i)$ — множество выходных переменных оператора S_i (множество результатов выполнения оператора S_i). Тогда наличие конкуренционной зависимости между операторами S_i и S_j может быть определено следующим образом:

а) конкуренция за значение переменной: $in(S_i) \cap out(S_j) \neq \varnothing$ при $i < j$, то есть переменные, являющиеся входными для оператора S_i, являются выходными для оператора S_j, и оператор S_i является предшествующим оператору S_j в тексте программы; при этом информационная зависимость оператора S_j от оператора S_i отсутствует (т.е. возможно параллельное выполнение этих операторов с точки зрения отсутствия информационной зависимости); однако в случае их параллельного выполнения значения

переменных, являющихся исходными данными для оператора S_i, могут быть переопределены оператором S_j до их использования в операторе S_i;

б) конкуренция за переменную (область памяти): $out(S_i) \cap out(S_j) \neq \varnothing$ при $i < j$, то есть операторы S_i и S_j обращаются к одной и той же области памяти (переменной) при сохранении результатов своего выполнения (т.е. наряду с конфликтом из-за ресурса ―общей памяти,― возможно затирание результатов одного из операторов результатами другого).

Таким образом, операторы S_i и S_j являются конкуренционно зависимыми, если для них не выполняется следующее соотношение:

$$in(S_i) \cap out(S_j) = out(S_i) \cap in(S_j) = out(S_i) \cap out(S_j) = \varnothing. \qquad (1)$$

Отношение (1) - это условие отсутствия конкуренционной зависимости между операторами, учитывающее как конкуренцию за значение так и конкуренцию за переменную.

Программа является линейной, если представляет собой последовательность операторов, не содержащую операторов цикла и условного перехода (циклы и совокупности операций с передачей управления должны рассматриваться как единые вычислительные блоки программы).

Формализация понятий зависимости и независимости между операторами программы

Множеством $C(P)$ **изменяемых переменных программы** называются переменные, которые в ходе выполнения программы P меняют свое значение хотя бы один раз. Тогда через $C(S_i)$ (где S_i – некоторый оператор (либо программный блок), входящий в программу) может быть обозначено множество изменяемых переменных некоторого оператора. Множеством $R(P)$ упоминаемых (используемых) переменных программы P, называются переменные участвующие при выполнении некоторых операций в программе (при вычислении результатов некоторых операторов). Следовательно через $R(S_i)$ может быть обозначено множество упоминаемых переменных некоторого оператора S_i.

Таким образом, упоминаемые переменные программы $R(P)$ ― это те переменные, которые используются при вычислении переменных множества $C(P)$ (при получении результатов в операторах программы P) . Через m_i ― может быть обозначено количество упоминаемых переменных в операторе (программном блоке) S_i, через n ― число операций в программе, в которых выполняются вычисления.

Если через S_i обозначить некоторый вычислительный блок,

множества $R(P)$ и $C(P)$ могут быть представлены виде: $R(P)=\{x \mid x \in \bigcup_{i=1}^{n} R(S_i)\}$, где $R(S_i)=\{x_1^i, x_2^i, \ldots, x_{m_i}^i\}$ – множество упоминаемых (используемых в процессе вычисления) переменных оператора (вычислительного блока) S_i. То есть это те переменные, которые входят в множества $R(S_i)$ операторов программы (в объединение множеств исходных данных операторов S_i программы). Аналогично, $C(P)=\{y \mid y \in \bigcup_{i=1}^{n} C(S_i)\}$, где $C(S_i)=\{y_1^i, y_2^i, \ldots, y_{m_i}^i\}$ —множество переменных— результатов выполнения оператора S_i . Таким образом, изменяемые переменные программы — это те переменные, которые являются результатами выполнения какого-либо из операторов программы (входят в объединение множеств переменных — результатов выполнения операторов S_i).

Через $C(P) \cup R(P) = V(P)$ может быть обозначено множество всех переменных программы. Тогда

$$V(P) = C(P) \cup R(P) \rightarrow V(S_i) = C(S_i) \cup R(S_i).$$

Входные переменные программы образуют множество $I(P)$, формализуемое в виде:

$$I(P) = \{\bigcup_{i=1}^{n} x \mid x \in R(S_i) \,\&\, x \notin C(S_1, S_2, \ldots, S_{i-1})\},$$

где n – количество операторов в программе, i – текущая операция. Таким образом, входными переменными программы являются те переменные, значения которых используются при вычислении некоторых операторов S_i, и при этом эти значения не получены ни в одном из предшествующих S_i операторов.

Множество выходных переменных программы:

$$O(P) = \{\bigcup_{i=1}^{n} y \mid y \in C(S_i) \,\&\, y \notin C(S_{i+1}, S_{i+2}, \ldots, S_n)\}$$

Таким образом, выходными переменными программы являются те переменные, значения которых получены в некотором операторе S_i и при этом эти значения не изменяются во всех последующих операторах (до конца программы). Операции (операторы) в программе могут быть связаны различными отношениями зависимости и независимости.

Непосредственная (сильная) зависимость между операторами в программе

Оператор S_j называется непосредственно (сильно) зависимым от оператора S_i (обозначается $S_i \rightarrow S_j, i < j$) если:

1) изменяемые оператором S_i переменные (результаты оператора S_i) являются упоминаемыми (используемыми в качестве исходных данных) в операторе S_j. Данное утверждение может быть формализовано следующим образом:

$$C(S_i) \cap R(S_j) \neq \varnothing,$$

то есть некоторые переменные — результаты вычисления оператора S_i (множество $\tilde{N}(S_i)$) входят в множество исходных данных оператора S_j (множество $R(S_j)$) так как пересечение указанных множеств не есть пустое множество;

2) в том случае, когда оператор S_j не следует непосредственно за оператором S_i и между этими операторами имеется некоторая совокупность операторов $S_{k_1}, S_{k_2}, S_{k_3}, ..., S_{k_l}$, тогда для S_i и S_j должно выполняться условие предусматривающее, что изменяемые переменные исходной операции (оператора) S_i не могут быть изменены в промежуточных операторах S_{k_r} до их использования в конечной операции S_j. Данное условие может быть формализовано следующим образом:

$$C(S_i) \cap R(S_j) \not\subset \bigcup_{r=1}^{l} C(S_{k_r}), \qquad (2)$$

где ($i<k_1<k_2<...<k_l<j$) и $r = \overline{1,l}$. Условие (2) определяет, что переменные, входящие в оба множества $C(S_i)$ и $R(S_j)$ (т.е. $C(S_i) \cap R(S_j) \neq \varnothing$) не могут входить ни в одно из множеств $C(S_{k_r})$ — множеств результатов операторов $S_{k_1}, S_{k_2}, S_{k_3}, ..., S_{k_l}$ (т.е. не могут входить в объединение множеств $C(S_{k_r})$ переменных— результатов операторов S_j).

Примером реализации условий сильной зависимости между операторами может являться приведенная ниже последовательность операторов:

$$S_i : x = q\text{^}2 + (54 \quad w);$$

$$S_{k_1} : y = w \quad 2 * h;$$

$$S_{k_3:} : k = j + (q\text{^}3 * h);$$

$$S_j : z = x + 12;$$

Из примера видно, что результат оператора S_i (переменная x, входящая в множество $C(S_i)$) выступает в качестве исходных данных для оператора S_j (принадлежность переменной x множеству $R(S_j)$ оператора S_j) и при этом значение переменной x не переопределяется ни в одном из промежуточных операторов S_{k_1}, S_{k_2} (т.е. не является результатом ни одного из промежуточных операторов S_{k_1}, S_{k_2}).

Слабая зависимость между операторами в программе

Слабая зависимость представляет собой реализацию цепочки сильных зависимостей. Оператор S_j называется слабо зависимым от оператора S_i ($S_i \overset{*}{\rightarrow} S_j$) если:

1) переменные – результаты оператора S_i (входящие в множество $C(S_i)$) могут быть использованы при вычислении результатов промежуточных операторов $S_{k_1}, S_{k_2}, S_{k_3}, ..., S_{k_l}$, а результаты промежуточных операторов могут быть использованы при вычислении оператора S_j. Данное утверждение может быть представлено в виде цепочки сильных зависимостей следующим образом:

$$\begin{aligned} C(S_i) \cap R(S_{k_1}) &\neq \varnothing; \\ C(S_{k_1}) \cap R(S_{k_2}) &\neq \varnothing; \\ C(S_{k_2}) \cap R(S_j) &\neq \varnothing. \end{aligned} \qquad (3)$$

В общем виде рассмотренная цепочка сильных зависимостей может быть представлена в виде следующего выражения:

$$C(S_p) \cap R(S_q) \neq \varnothing, \qquad (4)$$

где $p \in \{i, k_1, k_2, k_3, ... k_l\}$, $q \in \{k_1, k_2, k_3, ... k_l, j\}$; p, q могут принимать любые значения из представленных (обязательно $p = i, q = j$). Т.о. переменные, полученные в качестве результирующих в операторе S_i (переменные – результаты оператора S_i, входящие в множество $C(S_i)$) используются в операциях S_{k_r} в качестве исходных данных для них (входят в множества $R(S_{k_r})$) до операции S_j, а результаты операций S_{k_r} (переменные–

результаты оператора S_{k_r}, входящие в множество $C(S_{k_r})$) используются в операции S_{j_r} в качестве исходных данных (входят в множество $R(S_j)$);

б) по аналогии с сильной зависимостью переменные – результаты оператора S_i (входящие в множество $C(S_i)$) не могут быть изменены в промежуточных операторах S_q (не могут входить в множества $\tilde{N}(S_q)$ при $i<q<k_r$) до их использования в качестве исходных данных в операторе S_{k_r} (множество $R(S_{k_r})$), а переменные – результаты промежуточного оператора S_{k_r} (входящие в множество $C(S_{k_r})$) не могут быть изменены до их использования в операторе S_{j_r} (множество $R(S_j)$).

Первое из сформулированных условий пункта б) может быть формализовано в следующем виде:

$$C(S_i) \cap R(S_{k_r}) \not\subset \bigcup_q C(S_q), \qquad (5)$$

которое определяет, что переменные входящие как в $C(S_i)$ так и в $R(S_{k_r})$ ($C(S_i) \cap R(S_{k_r}) \neq \varnothing$) не могут входить ни в одно из множеств $C(S_q)$ промежуточных (между S_i и S_{k_r}, $i<q<k_r$) операторов S_q (не могут входить в объединение множеств $\tilde{N}(S_q)$ промежуточных операторов S_q, $i<q<k_r$).

Второе из сформулированных условий пункта б) может быть формализовано в следующем виде:

$$C(S_{k_r}) \cap R(S_j) \not\subset \bigcup_h C(S_h), \qquad (6)$$

которое по аналогии определяет, что переменные, входящие как в $C(S_{k_r})$ так и в $R(S_j)$ ($C(S_{k_r}) \cap R(S_j) \neq \varnothing$) не могут входить в объединение множеств переменных- результатов промежуточных операторов S_{h_r} (ни в одно из множеств $C(S_h)$).

Выполненное по аналогии с (3) и (4) обобщение формул (5) и (6) позволяет получить для формализации условия б) выражение в следующем виде:

$$C(S_p) \cap R(S_q) \not\subset \bigcup_{r=1}^{l} C(S_r), \qquad (7)$$

где ($p < r < q$).

В качестве примера слабой зависимости может быть рассмотрена следующая последовательность операторов:

S_i : x = q^2 + (54 w);

$$S_{k_1}: y = w \quad 2 * h;$$
$$S_{k_{3:}}: k = j + (q\text{^}3 * x);$$
$$S_{k_{3:}}: q = j + (y \quad h\text{^}2);$$
$$S_j: z = k + 12;$$

Здесь переменные – результаты оператора S_{i_r} используются при вычислении оператора $S_{k_{2_r}}$, при этом в промежуточных операторах (в операторе $S_{k_{1_r}}$) результаты оператора S_{i_r} не переопределяются. В свою очередь переменные – результаты оператора $S_{k_{2_r}}$ используются при вычислении оператора S_{j_r} и при этом эти переменные не изменяют своего значения в промежуточных операторах (в операторе $S_{k_{3_r}}$).

Слабая независимость между операциями

Операции S_i и S_j слабо независимы($S_i \rightarrow S_j$, при $i<j$), если:

1) между операторами S_i и S_j отсутствует информационная зависимость, т.е.

$$C(S_i) \cap R(S_j) = \varnothing, \tag{8}$$

что указывает на отсутствие переменных, входящих в $C(S_i)$ и в $R(S_j)$ (пересечение множеств равно $\varnothing$). В данном случае переменные-результаты оператора S_i (входящие в множество $C(S_i)$) не используются при вычислениях в операторе S_j (не входят в качестве элементов в множество;

2) между операторами S_i и S_j отсутствует конкуренционная зависимость (отсутствует конкуренция за значения переменных). Данное утверждение при условии выполнении условия (8) и условия $i<j$ формализуется следующим образом:

$$C(S_j) \cap R(S_i) = \varnothing. \tag{9}$$

Наличие конкуренции за значение между S_i и S_j при выполнении условия (8) может быть определено следующим образом:

$$C(S_j) \cap R(S_i) \neq \varnothing. \tag{10}$$

Условие (10) может быть проинтерпретировано следующим образом: при выполнении условия (8) информационная зависимость между операторами

S_i и S_j отсутствует; в этом случае может быть реализовано параллельное выполнение этих операторов; однако при параллельном выполнении операторов S_i и S_j возможно «затирание» значений переменных, используемых в операторе S_i (входящих в множество $R(S_i)$) значениями, полученными в операторе S_j, до их использования в операторе S_i (значениями переменных входящих в $C(S_j)$), т.е. реализуется конкуренция за значения переменных, входящих в множество $R(S_i)$. Таким образом, выполнение условия (9) гарантирует отсутствие конкуренции за значение между операторами S_i и S_j, что позволяет вести речь о возможном их параллельном выполнении;

3) между операторами S_i и S_j нет таких операторов $S_{k_1}, S_{k_2}, S_{k_3}, \ldots, S_{k_l}$ ($i<k_1<k_2<\ldots<k_l<j$), что имело бы место условие слабой зависимости между операторами S_i и S_j в виде:

$$C(S_p) \cap R(S_q) \neq \varnothing, \tag{11}$$

где $p \in \{i, k_1, k_2, k_3, \ldots k_l\}, q \in \{k_1, k_2, k_3, \ldots k_l, j\}$. То есть для операторов S_i и S_j и находящихся между ними операторов $S_{k_1}, S_{k_2}, S_{k_3}, \ldots, S_{k_l}$ условие слабой зависимости S_j от S_i в виде (11) выполняться не должно. Это означает, что результаты оператора S_i не являются используемыми при вычислении какого-либо из промежуточных операторов $S_{k_1}, S_{k_2}, S_{k_3}, \ldots, S_{k_l}$, а результаты какого-либо из промежуточных операторов $S_{k_1}, S_{k_2}, S_{k_3}, \ldots, S_{k_l}$ (полученные с использованием результатов оператора S_i) не являются используемыми при выполнении вычислений в соответствии с оператором S_j. Сформулированное выше условие может быть формализовано в следующем виде:

$$C(S_p) \cap R(S_q) = \varnothing, \tag{12}$$

где $p \in \{i, k_1, k_2, k_3, \ldots k_l\}, q \in \{k_1, k_2, k_3, \ldots k_l, j\}$.

Таким образом, между операторами S_i и S_j не имеется цепочка сильных зависимостей, для которой предполагается, что результаты промежуточных операторов получены на основе результатов S_i, а результаты этих промежуточных операторов не используются при выполнении вычислений в операторе S_j.

В качестве примера выполнения условий слабой независимости (для операторов S_i и S_j) может быть рассмотрена приведенная ниже

последовательность операторов:

$$S_i : x = q\text{^}2 + (54 \quad w);$$

$$S_{k_1} : y = w - 2 * x;$$

$$S_{k_{3:}} : k = j + (q\text{^}3 * h);$$

$$S_j : x = k + 12;$$

Для рассмотренной последовательности операторов может быть сделан вывод о слабой независимости между операторами S_i и S_j, так как:

1) оператор S_j не использует при вычислениях результаты оператора S_i (отсутствие информационной зависимости);

2) между операторами S_i и S_j отсутствует конкуренция за значение, так как оператор S_j в качестве переменной- результата использует переменную *x*, которая не входит в множество исходных данных оператора S_i;

3) операторы S_i и S_j не связаны отношением слабой зависимости, так как результаты оператора S_i не используются при вычислении результатов промежуточного оператора S_{k_2}, на основе результатов которого выполняются вычисления в операторе S_j.

Сильная независимость между операторами

Операторы S_i и S_j сильно независимы ($S_i \rightarrow\!\!\!\!\!\!\sim\ S_j$ при $i<j$), если:

1) между операторами S_i и S_j отсутствует информационная зависимость и конкуренция за переменную. Отсутствие информационной зависимости определяется выражением вида:

$$C(S_i) \cap R(S_j) = \varnothing, \qquad (13)$$

которое указывает на отсутствие переменных, входящих в множество $C(S_i)$ (множество переменных – результатов оператора S_i) и входящих в множество $R(S_j)$ (множество исходных данных оператора S_j). То есть переменные-результаты S_i не принадлежат множеству переменных –исходных данных S_j. Отсутствие конкуренции за переменную между операторами S_i и S_j предполагает, что значениями результатов вычислений в соответствии с этими операторами инициализируются разноименные (различные) переменные. Отсутствие между указанными операторами S_i и

S_j конкуренции за переменную формализуется следующим условием:

$$C(S_i) \cap C(S_j) = \varnothing, \qquad (14)$$

которое определяет отсутствие переменных, входящих как в множество $C(S_i)$ так и в множество $C(S_j)$. Таким образом, значения результатов вычислений в операторах S_i и S_j присваиваются абсолютно различным переменным (записываются в различные области памяти), в силу чего при выполнении (14) отсутствует конфликт при обращении к общему ресурсу с целью записи данных (к переменной или области памяти). Выражения (13), (14) могут быть обобщены в следующем виде

$$(C(S_i) \cap R(S_j)) \cup (C(S_i) \cap C(S_j)) = \varnothing, \qquad (15)$$

либо по другому (в обобщенном виде):

$$C(S_i) \cap V(S_j) = \varnothing.$$

2) в силу того, что отношение сильной независимости рефлексивно (т.е. если S_j не зависит от S_i, то и S_i не зависит от S_j), то на основе выражений (13) и (14) могут быть записаны выражения, определяющие независимость S_i от S_j. При этом формируемые выражения учитывают как отсутствие конкуренции за переменную, так и отсутствие конкуренции за значение. По аналогии с (9) условие отсутствия конкуренции за значение между операторами S_j и S_i может быть формализовано следующим образом:

$$C(S_j) \cap R(S_i) = \varnothing. \qquad (16)$$

Выражение (16) (также как и выражение (9)) определяет, что переменные — результаты оператора S_j (переменные, входящие в множество $C(S_j)$) не являются элементами множества $R(S_i)$ (множества переменных — исходных данных оператора S_i), — пересечение множеств есть . Таким образом, при инициализации переменных — результатов оператора $\overset{=}{\alpha}$ (значениями, полученными в этом операторе) значения переменных — исходных данных оператора S_i (значения, используемые при вычислениях в этом операторе) изменяться не будут. Отсутствие конкуренции за переменную между операторами S_j и S_i могут быть введены в рассмотрение по аналогии с выражением (14) следующим образом:

$$C(S_j) \cap C(S_i) = \varnothing. \qquad (17)$$

Выражения (16) и (17) могут быть обобщены следующим образом:

$$(R(S_i) \cap C(S_j)) \cup (C(S_i) \cap C(S_j)) = \varnothing, \qquad (18)$$

либо с учетом введенного выше множества $V(S_i)$ в следующем обобщенном виде:

$$C(S_j) \cap V(S_i) = \varnothing.$$

3) между операторами S_i и S_j нет таких операторов $S_{k_1}, S_{k_2}, S_{k_3}, \ldots, S_{k_l}$ ($i<k_1<k_2<\ldots<k_l<j$), что имело бы место условие слабой зависимости между операторами S_i и S_j в виде:

$$C(S_p) \cap R(S_q) \neq \varnothing, \qquad (19)$$

где $p \in \{i, k_1, k_2, k_3, \ldots k_l\}, q \in \{k_1, k_2, k_3, \ldots k_l, j\}$. То есть для операторов S_i и S_j и находящихся между ними операторов $S_{k_1}, S_{k_2}, S_{k_3}, \ldots, S_{k_l}$ условие слабой зависимости S_j от S_i в виде (11) выполняться не должно. Это означает, что результаты оператора S_i не являются используемыми при вычислении какого-либо из промежуточных операторов $S_{k_1}, S_{k_2}, S_{k_3}, \ldots, S_{k_l}$, а результаты какого-либо из промежуточных операторов $S_{k_1}, S_{k_2}, S_{k_3}, \ldots, S_{k_l}$ (полученные с использованием результатов оператора S_i) не являются используемыми при выполнении вычислений в соответствии с оператором S_j. Сформулированное выше условие может быть формализовано в следующем виде:

$$C(S_p) \cap R(S_q) = \varnothing,$$

где $p \in \{i, k_1, k_2, k_3, \ldots k_l\}, q \in \{k_1, k_2, k_3, \ldots k_l, j\}$.

Таким образом, между операторами S_i и S_j отсутствует цепочка сильных зависимостей, для которой предполагается, что результаты промежуточных операторов получены на основе результатов S_i, а результаты этих промежуточных операторов не используются при выполнении вычислений в операторе S_j.

В качестве примера выполнения условий слабой независимости (для операторов S_i и S_j) может быть рассмотрена приведенная ниже последовательность операторов:

$$S_i : x = q\text{^}2 + (54 \quad w);$$

$$S_{k_1} : y = w - 2 * x;$$

$$S_{k_3:} : k = j + (q\text{^}3 * h);$$

$$S_j : z = k + 12;$$

Для рассмотренной последовательности операторов может быть сделан вывод о сильной независимости операторов S_i и S_j, так как:

1) оператор S_j не использует при вычислениях результаты оператора S_i (отсутствие информационной зависимости);

2) между операторами S_i и S_j отсутствует конкуренция за значение, так как оператор S_j в качестве переменной – результата использует переменную *x*, которая не входит в множество исходных данных оператора S_i;

3) между операторами S_i и S_j отсутствует конкуренция за переменную, так как оператор S_j в качестве переменной – результата использует переменную *x*, которая не входит в множество переменных – результатов оператора S_i;

4) операторы S_i и S_j не связаны отношением слабой зависимости, так как результаты оператора S_i не используются при вычислении результатов промежуточного оператора S_{k_2}, на основе результатов которого выполняются вычисления в операторе S_j.

Свойства отношений зависимости и независимости между операциями

Отношение сильной зависимости ($S_i \rightarrow S_j$): не рефлексивно: из $S_i \rightarrow S_j \not\Rightarrow S_j \rightarrow S_i$; не симметрично: $S_i \rightarrow S_i$ - не имеет смысла; не транзитивно: из $S_i \rightarrow S_j \,\&\, S_j \rightarrow S_k \not\Rightarrow S_i \rightarrow S_k$.

Отношение слабой зависимости ($S_i \overset{*}{\rightarrow} S_j$): не рефлексивно: $S_i \overset{*}{\rightarrow} S_j \not\Rightarrow S_j \overset{*}{\rightarrow} S_i$; не симметрично: $S_i \overset{*}{\rightarrow} S_i$ - не имеет смысла; но транзитивно: $S_i \overset{*}{\rightarrow} S_j$ и $S_j \overset{*}{\rightarrow} S_k \Rightarrow S_i \overset{*}{\rightarrow} S_k$.

Отношение слабой независимости ($S_i \overset{*}{\not\rightarrow} S_j$): рефлексивно: из $S_i \overset{*}{\not\rightarrow} S_j \Rightarrow S_j \overset{*}{\not\rightarrow} S_i$; симметрично: $S_i \overset{*}{\not\rightarrow} S_i$; не транзитивно.

Отношения сильной независимости ($S_i \not\rightarrow S_j$) : рефлексивно, симметрично, не транзитивно.

2.2. Алгоритм распараллеливания последовательных программ путем приведения их к полной параллельной форме (ППФ)

Программа последовательна, если отсутствуют операции ветвления и циклы. Участок программы называется параллельным, если любая пара его операторов является независимыми операторами (могут выполняться параллельно). Т.е. любой оператор участка выполняется независимо и

параллельно с каждым другим оператором участка. Два участка программы называются взаимно параллельными, если любой оператор одного участка выполняется независимо от любого оператора другого участка. Последовательность операторов α в виде:

$$\alpha = S_1, S_2, S_3, ..., S_l$$

,

либо в виде:

$$\alpha = S_i, S_{i+1}, ..., S_{i+p}$$

,

называется линейной последовательной цепочкой, если выполняется условие:

$$S_i \to S_{i+1}, S_{i+1} \to S_{i+2}, ..., S_{i+(p-1)} \to S_{i+p} \Rightarrow S_i \in \alpha, S_{i+1} \in \alpha, S_{i+p} \in \alpha$$

,

где через $\to$ обозначено отношение сильной зависимости между операторами. При этом операторы $S_i, S_{i+1}, S_{i+2}, ..., S_{i+(p-1)}, S_p$ входят в последовательную цепочку α.

Последовательная цепочка называется максимальной, если она не содержится в другой (большей) последовательности. Алгоритм приведения программы к ППФ предполагает выделение максимальных последовательных цепочек. Для двух или более максимальных последовательных цепочек выполняется условие: $\alpha_1 \cap \alpha_2 = \varnothing$.

То есть операторы, зависящие сильно от операторов одной и другой цепочек не могут входить ни в одну из них. Таким образом, два взаимно параллельных участка могут иметь вид максимально последовательных цепочек.

Пример максимальных последовательных цепочек, являющихся параллельными ветвями алгоритма представлен на Рисунке 2.

Рисунок 2 – Пример последовательных цепочек, входящих в параллельный оператор

Параллельная цепочка $\overline{\alpha} = \overline{S_i, S_{i+1}, ..., S_{i+p}}$ называется параллельным групповым оператором, если представляет собой совокупность независимо выполняющихся операций. Цепочка является максимально параллельной, если она не входит в состав других максимально параллельных цепочек. Для максимально параллельных цепочек выполняется условие: $\overline{\alpha}_1 \cap \overline{\alpha}_2 = 0$.

Программа P', полученная из последовательной программы P путем выделения в ней всех максимальных последовательных и параллельных цепочек, называется ППФ (полной параллельной формой) линейной программы P.

Программа P' – это параллельная форма программы P, если в программе P могут быть выделены последовательные, параллельные, взаимно параллельные участки и в результате получена ППФ. Программа P называется приводимой к ППФ, если путем замены максимально последовательных и параллельных цепочек соответствующими групповыми операторами может быть получена ППФ программы.

Условие приводимости к ППФ

Для того, чтобы линейная программа была приводимой к ППФ,

необходимо и достаточно, чтобы:

1) существовали взаимно независимые операторы S_i и S_j;

2) для каждой пары взаимно независимых операторов S_i и S_j между ними существовал некоторый оператор S_k такой, что $S_k \in N_2(S_i, S_j)$ (где S_k лежит между S_i и S_j);

3) при этом имело место условие:

$$N_1(S_i, S_j) \subseteq \xi^*(S_k) \qquad N_1(S_i, S_j) = \xi^*(S_i) \cap \xi^*(S_j),$$

$$N_2(S_i, S_j) = \xi^*(S_i) \cup \xi^*(S_j) \setminus N_1(S_i, S_j),$$

$$\xi^*(S_i) = \{S \mid S_i \xrightarrow{*} S\} \qquad (20)$$

Т.о. множество $\xi^*(S_i)$ состоит из элементов, сильно и слабо зависящих от S_i.

Из анализа теоремы видно, что любое распараллеливание последовательных программ начинается с построения множества входных и выходных переменных для каждой операции (оператора, блока) и выявления независимых операций в программе, для которых проверяется выполнение условия (20).

В том случае, если для линейной программы условие (20) выполняется, для нее существует ППФ и при том единственная.

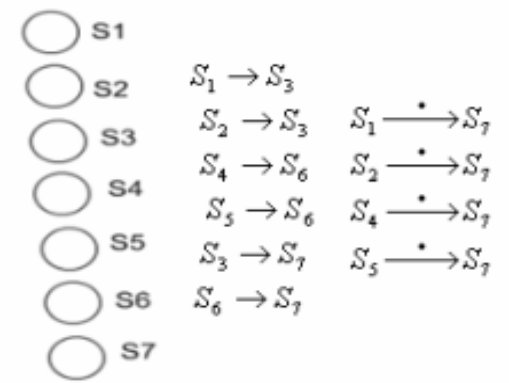
Пояснения к условию: множество $\xi^*(S_i)$ – это множество всех операторов, зависящих от S_i слабо либо сильно, $N_1(S_i, S_j)$ - это множество операторов, зависящих как от S_i, так и от S_j. Если S_i и S_j - независимы, то могут выполняться параллельно, то есть находиться в параллельных ветвях. Следовательно, $N_1(S_i, S_j)$ – это те элементы, которые являются точками слияния параллельных ветвей, $N_2(S_i, S_j)$ – это операторы, зависящие либо от S_i либо от S_j, тогда S_k, входящий в $N_2(S_i, S_j)$ – это оператор, принадлежащий одной из параллельных ветвей (если S_k принадлежит некоторой ветви, то оператор в $N_1(S_i, S_j)$ автоматически зависит от S_k).

Правило: в том случае, если для данной последовательности операторов условие приводимости не выполняется, в исходной линейной программе должен быть изменен порядок следования операторов.

Правило изменения порядка: если программа к ППФ не приводится, то операторы, использующие результаты вышестоящих, должны следовать непосредственно за ними.

Пример проверки условия приводимости программы к ППФ

представлен ниже.



Проверка приводимости к ППФ для операторов программы S_1 и S_4:

1. S_1 и S_4 - независимы;

2.между S_1 и S_4 имеется промежуточный оператор S_3;

3. $N_1(S_1,S_4)=\{S_7\}$, $N_2(S_1,S_4)=\{S_3,S_6\}$, $S_3 \in N_2(S_1,S_4)$, $\xi(S_3)=\{S_7\} \Rightarrow N_1(S_1,S_4) \subseteq \xi(S_3)$.

Таким образом, для операторов S_1 и S_4 программа является приводимой к ППФ.

Таким образом, указанные выше пункты 1-3 позволяют идентифицировать приводимость к ППФ установленной последовательности операторов. Если условия 1-3 не выполняются, то данная последовательность операторов не приводится к ППФ и последовательность операторов должна быть изменена. Изменение порядка предполагает, что операторы, использующие результаты вышестоящих операторов должны следовать непосредственно за ними.

Алгоритм преобразования программы к ППФ

1. Заменить каждую максимальную параллельную цепочку $\overline{\alpha}$ параллельным групповым оператором $\overline{Y}$ (оператор является либо параллельным участком или взаимно параллельными участками):

$$\{S_i, S_{i+1}, ..., S_{i+n}\} \in \overline{\overline{\alpha}}$$
$$\overline{\overline{\alpha}} \sim \overline{\overline{Y}}.$$

2. Заменить каждую максимально последовательную цепочку α последовательным групповым оператором Y:

$$\{S_i \to S_{i+1}, S_{i+1} \to S_{i+2}, ...S_{i+(k\ 1)} \to S_{i+k}\},$$

$$\{S_i, S_{i+1}, ..., S_{i+k}\} \in \overline{\alpha} \Rightarrow \overline{\alpha} \sim \overline{Y}.$$

3. Повторять шаги 1 и 2 до тех пор, пока не останется ни одной максимально параллельной либо максимально последовательной цепочек. В результате программа имеет вид одного участка, называемого ППФ.

Пример последовательного выделения параллельных и последовательных участков для приведения программы к ППФ.

1. Исходные отношения зависимостей и независимостей (Рис. 3).

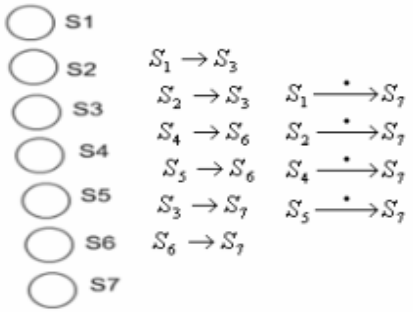


Рисунок 3 – Отношения зависимостей и независимостей в последовательной программе

2. Последовательность шагов по выделению параллельных и последовательных групповых операторов при приведении программы к ППФ (Рис.4).

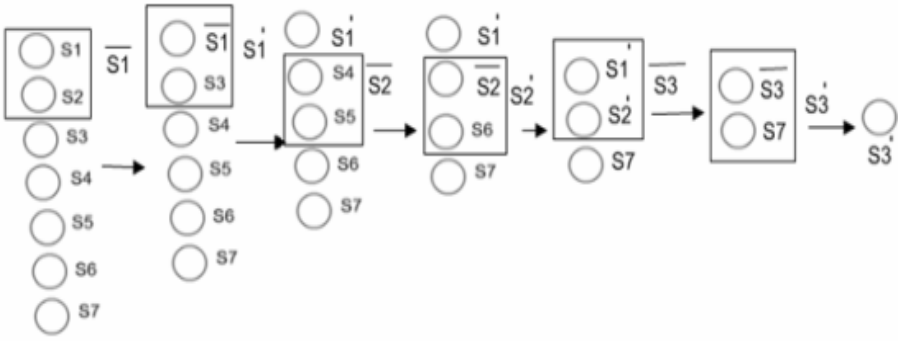


Рисунок 4 – Последовательность шагов алгоритма по приведению программы к ППФ

Пример приведения программы, представленной в виде графа управления к ППФ (Рис.5).

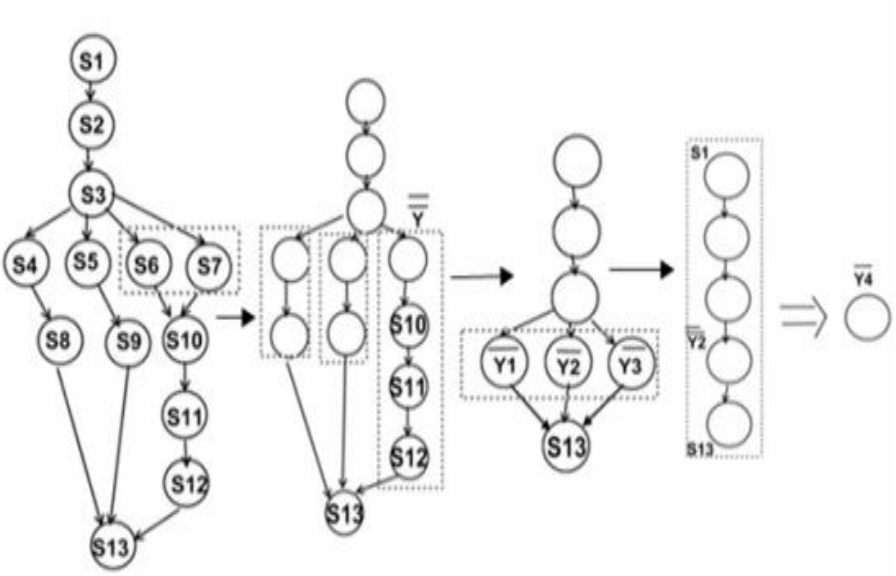


Рисунок 5 – Приведение программы, представленной в виде графа управления к ППФ

Метод формирования параллельных групповых операторов

Для реализации алгоритма приведения программы к ППФ должны быть введены дополнительные множества:

1 . U – множество всех операторов S_i программы.

2. U_i – множество операторов программы (элементарных и групповых), рассмотренных на i-ом шаге алгоритма распараллеливания;

3. β_i - множество всех параллельных групповых операторов программы, полученных до i -го шага и на i -м шаге включительно.

4. $\beta_i^{'}$ - множество параллельных операторов, формирующих групповой оператор на i -ом шаге $\beta_{i\ 1} \cup \beta_i^{'} = \beta_i$.

5 . K_i – множество последовательных операций, не входящих в параллельные участки, то есть те операторы, для которых будет применяться алгоритм распараллеливания на последующих шагах. Таким образом, метод формирования групповых операторов направлен на выделение из множества U_i всех операторов программы, множеств операторов являющихся параллельными и операторов, являющихся последовательными (не входящими в $\beta_i^{'}$).

Алгоритм выделения параллельных групповых операторов предполагает выполнение указанной ниже последовательности шагов:

1. Руководствуюсь понятиями зависимости и независимости операций сформировать матрицу независимости C_{pq}^{i} для одиночных и групповых операторов, рассматриваемых на *i*-м шаге. Строки *p* и столбцы *q* идентифицируются одинаковым образом операторами, рассматриваемые на *i* -м шаге.

2. Если операторы S_p и S_q являются независимыми, то элемент $C[p,q]=1$.

3. Для любого S_p обозначим $C[p,q]=1$ при *p=q*.

4. Обозначим

$$p=\{k,k+1,k+2,...\}$$

$$q=\{...,m-2,m-1,m\}$$

.

5.В последовательной программе должно быть установлено наличие такой совокупности операторов $S_k, S_{k+1}, ..., S_m$ таких, что для любых *p* и *q* ($1 \le k, k \le p, q \le m, m \le n$) имелись бы такие элементы матрицы независимости, что $C[p,q]=1$ и эти элементы образуют квадратную матрицу порядка n_1 ($2 \le n_1 \le n$).

Если таких матриц несколько, то выбирается та, у которой номера операторов являются наименьшими.

Таким образом, операторы $S_k, S_{k+1}, ...$ представляют собой вышестоящие операторы программы (им соответствуют индексы строк), $, ...S_{m-1}, S_m$ – нижестоящие операторы программы (им соответствуют индексы столбцов), а сама процедура определяет порядок просмотра матрицы независимости C_{pq}^{i} и выделения в ней квадратной матрицы (просмотр матрицы построчно с выделением соответствующих элементов в столбцах). Сама последовательность $S_k, S_{k+1}, ...S_{m-1}, S_m$ определяет независимые нижестоящие операторы от вышестоящих и наоборот.

Пример матрицы независимости:

	S1	S2	S3	S4	S5	S6
S1	1	① 1	0	1	1	1
S2	1	1	0	1	1	② 1
S3	0	0	1	1	1	1
S4	1	1	1	1	1	0
S5	1	③ 1	1	1	1	0
S6	1	1	1	0	0	1

Матрица (2) не может быть интерпретирована, так как номера операторов в ней больше, чем в (1). Матрица (3) не может быть интерпретирована, так как по ней не может быть сформирована последовательность операторов $S_k, S_{k+1}, \ldots S_{m-1}, S_m$, где S_k – вышестоящие операторы программы.

6. Отнести сформированную последовательность $S_k, S_{k+1}, \ldots S_{m-1}, S_m$ к множеству β_i' (параллельных операторов, выделяемых на i-ом шаге), остальные операторы к множеству K_i.

7. На основе множества β_i' сформировать параллельный групповой оператор, рассматриваемый в дальнейшем как элемент множества U_{i+1}.

Таким образом, на основе анализа множеств β_i' может быть осуществлено разворачивание ППФ к графу параллельного выполнения программы.

Пример разворачивания ППФ программы в граф ее параллельного выполнения представлен на Рис.6,7.

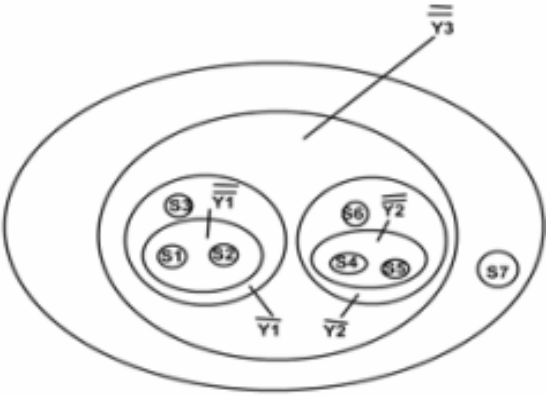


Рисунок 6 – Пример представления ППФ

$$U_n = \{S_7, \overline{Y_3}\}$$

$$U_{n-1} = \{\overline{Y_1}, \overline{Y_2}, S_7\}$$

$$\beta_{n-1} = \{\overline{Y_1}, \overline{Y_2}\}$$

$$U_{n-2} = \{\overline{Y_1}, \overline{\overline{Y_2}}, S_6, S_7\}$$

$$U_{n-3} = \{\overline{\overline{Y_1}}, S_3, \overline{\overline{Y_2}}, S_6, S_7\}$$

$$U_{n-4} = \{\overline{\overline{Y_1}}, S_3, S_4, S_5, S_6, S_7\}$$

$$\beta_{n-4} = \{S_4, S_5\}$$
$$U_{n-5} = \{S_1, S_2, S_{3,} S_4, S_5, S_6, S_7\}$$
$$\beta_{n-4}{}' = \{S_1, S_2\}$$

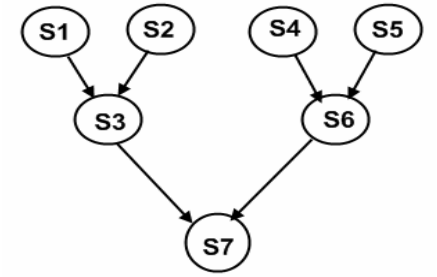


Рисунок 7 - Пример графа параллельного выполнения программы

Прямая процедура предполагает формирование множеств U_i, $\beta_i{}'$, при разворачивании ППФ множество интерпретируется в обратном порядке шагов.

3. Программа выполнения работы

В соответствии с вариантом задания необходимо выполнить максимальное распараллеливание, как для элементарных операций, так и для групповых операторов (линейных и параллельных цепочек). Для достижения поставленной задачи необходимо:

- построить схему программы, подлежащей распараллеливанию;
- определить для каждой операции множество упоминаемых (используемых), изменяемых входных и выходных переменных;
- на основании анализа сформированных множеств переменных определить зависимые и независимые операции (операторы) в программе;
- для всех независимых операций программы проверить условие преобразования этой программы к полной параллельной форме (условие приводимости к ППФ);
- используя алгоритм преобразования выполнить трансформацию последовательной программы к ППФ на *i*-м шаге распараллеливания;
- представить схему распараллеливания программы для выделенных на *i*-м шаге параллельных операторов, трансформировать параллельные операторы в один групповой оператор, который будет рассматриваться на следующем шаге распараллеливания;

- повторить шаги 2-6 столько раз, сколько это возможно (для максимального распараллеливания программы – параллельного выполнения как элементарных операций, так и групповых операторов);
- построить обобщенную схему распараллеливания программы, в которой отобразить как параллелизм элементарных операций, так и параллелизм групповых операторов;
- реализовать полученную схему распараллеливания программы, объединив требуемые операции в параллельно выполняемые процессы, синхронизируемые между собой и обменивающиеся необходимыми данными. Отладить программу и представить распечатку результатов ее работы;
- на каждом шаге распараллеливания представлять множества упоминаемых, изменяемых входных и выходных переменных как для элементарных операций, так и для групповых операторов;
- на каждом шаге распараллеливания представлять множества зависимых и независимых операций (одиночных и групповых);
- на каждом шаге распараллеливания представлять состав множества U_i (он будет изменяться при появлении групповых операций). Представлять вид матрицы независимости и полученной из нее матрицы независимых операций на i-м шаге распараллеливания. Представить вид множеств β_i и K_i на i-м шаге;
- на каждом i-м шаге представлять трансформированную схему программы к ППФ;
- представить обобщенную схему распараллеливания программы, приведенной к ППФ.

4. Варианты заданий

Осуществить распараллеливание последовательности программ в соответствии с вариантом:

1 вариант:
Найти определитель матрицы размером 4*4.
2 вариант:
Найти матрицу, обратную матрице размером 3*3.
3 вариант:
Найти произведение квадратных матриц размером 4*4.

5.КОНТРОЛЬНЫЕ ВОПРОСЫ

1. В чем заключается понятие последовательной ациклической программы?
2. В чем заключается понятие последовательной линейной цепочки и параллельной цепочки?
3. Что такое изменяемые, упоминаемые (используемые) переменные в программе?

4. В чем состоит понятие непосредственной зависимости и слабой зависимости операций в программе?
5. В чем состоит понятие сильной и слабой независимости операций?
6. Какими свойствами обладают отношения сильной и слабой зависимости, сильной и слабой независимости?
7. В чем заключается понятие полной параллельной формы программы?
8. Какой вид имеют условия приведения программы к полной параллельной форме?
9. Какие шаги содержит алгоритм преобразования программы к ППФ, множества, используемые при распараллеливании?
10. В чем заключается понятие групповой операции (группового оператора)?
11. В чем состоит алгоритм формирования параллельных групповых операторов?
12. Какой вид имеют условия формирования последовательных групповых операторов?

ЛАБОРАТОРНАЯ РАБОТА №2

ИССЛЕДОВАНИЕ МЕТОДА РАСПАРАЛЛЕЛИВАНИЯ ЛИНЕЙНЫХ ПОСЛЕДОВАТЕЛЬНЫХ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ ПОНЯТИЙ МОДЕЛИ ВЫЧИСЛИТЕЛЬНОГО ПРОЦЕССА

1. Цель работы: выполнить исследование двумерной модели вычислительного процесса, особенностей применения понятий модели к распараллеливанию программ.

2. Теоретическое введение

2.1. Понятие двумерной модели вычислительного процесса

Двумерная модель вычислительного процесса вводит понятия множеств операций, используемых при выполнении программы. На основе анализа этих множеств определяются процессы, выполняющиеся параллельно, моменты времени начала их выполнения и окончания.

Указанная модель задается в форме системы уравнений в следующем виде:

$$q_t = \psi(M_0, P_{\|t-1}) ;$$
$$^{\circ}A_t = {}^{\circ}A_{t-1} \setminus {}^{*}A_{t-1} ;$$
$${}^{*}A_t = F_1({}^{\circ}A_t, q_t) ;$$
$${}^{+}A_t = F_2({}^{*}A_t, J);$$
$${}^{p}A_t = F_3({}^{+}A_t, q_t);$$
$${}^{-}A_t = F_4({}^{p}A_t, q_t);$$

где q_t – текущее состояние памяти (значение информационных и управляющих переменных); $P\|_{t-1}$ – предыстория процесса в момент времени t (вычислительный процесс, выполняющийся до $(t-1)$шага включительно, вызывающий изменения состояния памяти); M_0 – начальное состояние памяти (начальные значения переменных); ${}^{0}A_t$ – множество операций, находящихся к моменту времени t в выключенном состоянии (то есть операции, которые не готовы к выполнению, так как для них не подготовлены исходные данные и соответствующие управляющие сигналы); ${}^{*}A_t$ – множество операций, для которых в текущий момент времени t дано разрешение на включение (для них подготовлены исходные данные,

восполнена синхронизация, т.е. готовы быть запущенными); ${}^{+}A_t$ – множество операций, включающихся в момент времени *t*; ${}^{P}A_t$ – множество операций, работающих в момент времени *t*; ${}^{-}A_t$ – множество операций, выключающихся в момент времени *t*, формируется из множества работающих операций с учетом требуемого для их (операций) выключения состояния памяти, то есть работающая операция переводит значение переменных в некоторое требуемое состояние и может быть выключена.

Таким образом, операции, входящие в множество ${}^{*}A_t$, выбираются из ${}^{0}A_t$ такие, что состояние памяти q_t определяет готовые для их выполнения информационно-управляющие переменные.

Множество ${}^{*}A_t$ формируется из множества не включившихся до (*t-1)*-шага операций, исключаются операции, которым на (*t-1)*- шаге дано разрешение на включение. Операции, принадлежащие одному множеству, переводятся в другое множество в момент времени (*t-1).*

Множество ${}^{+}A_t$ определяется на основе операций, готовых к включению, с учетом структурно-временных характеристик ВС и свободных ресурсов ВС. Таким образом, параметр *J* в модели определяет наличие свободной памяти и свободных процессоров, необходимых для активизации задач.

Множество ${}^{P}A_t$ формируется с учетом включившихся в момент *t* операций на основе текущего состояния памяти. Состояние памяти q_t влияет на работающие операции, так как для них могут быть не подготовлены некоторые синхронизирующие сигналы либо промежуточные значения.

Уточнение множеств двумерной модели вычислительного процесса

Множество ${}^{*}A_t$ может быть определено в виде:

$$ {}^{*}A_t = {}^{*}A_t^{1} \cup {}^{*}A_t^{2} .$$

Множество ${}^{*}A_t^{1}$ – это множество операций, для которых до момента t было дано разрешение на включение, но они не были включены в силу отсутствия свободных ресурсов. Множество ${}^{*}A_t^{2}$ – множество операций, которым разрешение на включение дано в момент времени *t.*

Множество операций ${}^{P}A_t$ определяется в следующем виде:

$$ {}^{P}A_t = ({}^{P}A_{t-1} \cup {}^{+}A_t) \setminus {}^{-}A_{t-1} .$$

Таким образом, множество операций, работающих в момент времени t,

образуется из множества работавших операций на (*t-1)*-шаге и множества включившихся операций на (*t-1)*- шаге, при исключении выключившихся на (*t-1)*- шаге операций.

Множество операций $\bar{A}_t$ формируется с учетом текущего времени t и времени обработки в следующем виде:

$$\bar{A}_t = \left\{ \bigcup a_j \in^P A_t \mid t_j^u + t_j^o \le t \right\},$$

где t_j^i - момент времени начала выполнения операции a_j, t_j^o - время обработки операции a_j, t- текущий момент времени. Таким образом, множество $\bar{A}_t$ - это те операции, для которых сумма значений времени начала работы и интервала времени обработки меньше, чем текущее значение (т.е. к текущему моменту времени операция закончила свою обработку).

2.2 Распараллеливание программ с использованием понятий двумерной модели вычислительного процесса

Понятие операции и выражения

Операция – некоторое элементарное неделимое вычислительное действие. Выражение – это совокупность элементарных действий, образующих последовательный участок программы. Последовательное выражение характеризуется наличием отношения строгого порядка между операторами в нем. Метод распараллеливания предполагает формирование ярусно-параллельной формы программы (ЯПФ). ЯПФ – это ориентированный граф, узлами которого являются некоторые операции. ЯПФ определяет наличие уровней, операторы на которых могут выполняться независимо. Последовательность ярусов определяет порядок последовательного выполнения задач.

Анализируемая последовательная программа в результате применения метода может быть представлена в виде управляющего графа $G(U,V)$, где U – множество вершин графа (операций), V- множество связей передачи управления.

Аналогичным образом, некоторый участок ξ (выполняющий вычисление некоторого выражения) может быть представлен в виде подграфа $G_\xi(U_\xi,V_\xi)$ графа $G(U,V)$. При этом $a_j \in U$ либо $a_{j_\xi} \in U_\xi$.

Через $U \times U$ и $U_\xi \times U_\xi$ может быть обозначено декартово произведение вершин графа и подграфа, тогда множество связей в графе определяется на основе $V \subseteq U \times U$. Множество связей участка ξ определяется:

$$C_\xi = (inU_\xi \cup outU_\xi) \setminus (inU_\xi \cap outU_\xi).$$

Множество связей между вершинами:

$$C_{\xi}^{'} = (ina_{j\xi} \cap outa_{i\xi}), \; i < j .$$

Выражение для $C_{\xi}^{'}$ определяет дуги между вершинами a_i и a_j на участке ξ. Дуги характеризуют передачу управления от одной вершины к другой. Таким образом задача распараллеливания состоит в формировании некоторого подграфа $G_{\xi}(U_{\xi},V_{\xi})$ для параллельного вычисления выражения и графа $G(U,V)$ параллельного выполнения программы.

Постановка задачи распараллеливания

Любая операция a_j, входящая в состав участка ξ (выполняющего вычисление некоторого выражения) может быть представлена в виде:

$$a_{j\xi} = \{j, \theta_{j\xi}, ina_{j\xi}, outa_{j\xi}, w_{j\xi}, | ina_{j\xi} |, | w_{j\xi} |\} \quad (1)$$

где $|\cdot|$ - мощность соответствующего множества (количество элементов, входящих в множество), $\theta_{j_{\xi}}$ - тип операции, определяет как выполнимое вычислительное действие, так и количество входных в операцию переменных, $in\, a_{j_{\xi}}$ - множество переменных, являющихся входными в данный оператор $a_{j_{\xi}}$, $out\, a_{j_{\xi}}$ - множество результатов оператора $a_{j_{\xi}}$, $W_{j_{\xi}}$ – множество номеров операций, использующих результаты данной j-ой операции (множество $W_{j_{\xi}}$ позволяет определить те операторы, для инициализации которых необходимы результаты операции $a_{j_{\xi}}$.

Если обозначить через J_{ξ} множество номеров операций ξ-го участка, то множество $W_{j_{\xi}}$ может быть определено следующим образом:

$$W_{j\xi} = \{j_1, j_2, j_3, ..., j_k\},$$

$$j \in J_{\xi}, j_1 \in J_{\xi}, j_2 \in J_{\xi}, ..., j_n \in J_{\xi}.$$

Метод распараллеливания предполагает формирование множеств операций ${}^{0}A_t$, ${}^{*}A_t$, ${}^{+}A_t$, ${}^{D}A_t$, ${}^{-}A_t$ для каждого дискретного момента времени, сопоставляемого с некоторым ярусом ЯПФ, таким образом, цель метода распараллеливания состоит в формировании указанных множеств двумерной модели вычислительного процесса. Реализация метода выполняется на основе Таблицы №1, формируемой в соответствии с выражением (1).

Таблица №1

№	Тип операции	Входные операнды				Результаты a_{k_ξ}	W_j	$\lvert ina_j \rvert$	$\lvert w_j \rvert$
		a_{j_ξ}	a_{i_ξ}	a_{p_ξ}	a_{q_ξ}				

Правила формирования таблицы операций некоторой программы (участка) предполагают выполнение указанной ниже последовательности шагов:

1) первоначально в таблицу заносятся те операции, которые определяют действия присваивания внутренней переменной данной программы (участка) значений, являющихся входными в нее (в рассматриваемую программу, унарные операции присваивания);

2) нумерация операций выполняется в порядке их следования в программе. Каждая строка в таблице формируется в соответствии с выражением (1);

3) вычисления внутри каждого участка ξ должно заканчиваться выполнением операции присваивания его результата некоторой переменной.

Для реализации оптимального распараллеливания должно выполняться уникальное именование результатов каждой из операций. Данное правило является обязательным.

Условиями формирования множества ${}^*A_t$ (оператора F_1 двумерной модели вычислительного процесса (способа формирования множества ${}^*A_t$)) являются условие бесконфликтности и готовности операций к выполнению ($a_j \in {}^*A_t$). Условие готовности оператора к выполнению имеет вид:

$$
{}^*A_k = \begin{cases} \bigcup\limits_j a_j \mid ina_j \subseteq C_\xi, \ \textit{при } k = 0, & (2) \\ \bigcup a_j \in {}^0A_k \mid ina_j \subseteq \bigcup\limits_{l=0}^{k-1} [\bigcup\limits_j out^{\,p} a_j]_l . & (3) \end{cases}
$$

выражение (2) определяет операцию $a_j \in {}^*A_k$, которая выполняет присваивание значений внешних по отношению к участку ξ (программы P) переменных внутренним в этом участке именам (локальным в программе переменным); выражение (3) определяет множество операторов (вычислительных операций), для которых исходные данные подготовлены операциями, находящимися на вышестоящих ярусах ЯПФ (здесь выражение $[\bigcup\limits_j out^{\,p} a_j]_l$ определяет переменные-результаты выполнения операций ${}^\partial a_j$

на некотором *l*-ом ярусе ЯПФ, а выражение $\bigcup_{l=0}^{k-1} [\bigcup_{j} out^{p} a_j]_l$ определяет переменные- результаты выполнения операций ${}^{\partial}a_j$ на всех ярусах, предшествующих формируемому *k*-ому ярусу ЯПФ).

Условие бесконфликтности

$$(\forall k)(\forall a_j, a_i \in {}^{*}A_k)(out a_j \cap out a_i = \varnothing) \& (in a_j \cap out a_i = \varnothing) \\ \& (out a_j \cap in a_i = \varnothing) \quad (4)$$

при $i \neq j$.

Для любого *k* -го яруса ЯПФ и находящихся на нем операций a_j и a_i не должно быть общих переменных результатов (отсутствие конкуренции за переменную) и входные переменные какой-либо операции на данном ярусе не могут являться выходными переменными другой операции на данном ярусе (отсутствие конкуренции за значение). Условие бесконфликтности выполняется автоматически в случае реализации правила уникального именования результатов каждой операции.

Алгоритм построения ЯПФ программы

1. На нулевой ярус ЯПФ (*k=0)* выносятся те операции a_j , входные операнды которых являются входными по отношению к данному участку либо программе (a_j в этом случае - операции присваивания). В результате сформировано множество ${}^{*}A_k$ операций, которым дано разрешение на включение.

2. На последующий *k* -ый ярус выносятся те операции, которые до этого входили в множество ${}^{0}A_k$, при этом значения входных переменных данных операций подготовлены операциями с вышестоящих ярусов.

3. Функция модели (оператор, как способ формирования множества) F_2 определяет ограничение на ресурсы или условия запуска $a_j \in {}^{*}A_k$ на выполнение.

4. Функция F_2 может быть проинтерпретирована в рассматриваемом случае таким образом, что ограничения на ресурсы отсутствуют, при этом ${}^{*}A_k = {}^{+}A_k$.

Оператор F_1 как способ построения множества ${}^{*}A_k$ с учетом условия готовности (2), (3) и с использованием формы (1) представления операции a_j

(множества W_j) может быть построен рассмотренным ниже способом. Для реализации оператора F_1 определяются условия вынесения операции на *k* -ый ярус ЯПФ. Эти условия формулируются для операции a_j в соответствии с количеством входных для нее операндов. В соответствии с этим операции a_j определяют как: унарные, бинарные, тернарные и т.д. N -арные.

Условия вынесения операции a_j (с номером j) на *k*-ый ярус ЯПФ:

1. Для унарной операции условием вынесения на *k*-ый ярус является принадлежность номера этой операции множеству $(W_i)_{k-1}$ операции a_i предшествующего *(k-1)*-го яруса. В результате номер операции j, выносимой на *(k)*-ый ярус, должен принадлежать объединению множеств $(W_i)_{k-1}$ операций a_i с предшествующего *(k-1)*-го яруса. В данном случае (для унарной операции) условие вынесения на *(k)*-ый ярус ЯПФ имеет вид:

$$j \in [\bigcup_{i=1}^{m} W_i]_{k-1},$$

где m –число операций на *(k-1)*-ом ярусе ЯПФ. Введенное выражение комментирует Рис.1

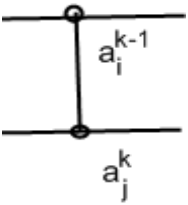


Рисунок 1- Условие вынесения унарной операции на *(k)*-ый ярус ЯПФ

2. Для бинарной операции номер операции j принадлежит любому из попарных пересечений множеств $(W_i)_{k-1}$ предшествующего яруса, либо один из операндов операции a_j формируется на одном из предшествующих ярусов от 0 до *(k-2)*-го включительно, а второй операнд формируется какой-либо операцией a_i на *(k-1)*-ом ярусе. В этом случае для бинарной операции a_j условия ее вынесения на *(k)*-ый ярус ЯПФ имеют вид:

$$j \in \begin{cases} \bigcup\limits_{l,i} (W_i \cap W_l)_{k-1}, \text{ где } l = \overline{i+1, m} \text{ при } i = \overline{1, m-1}; \\ \bigcup\limits_{p=0}^{k-2} [(\bigcup\limits_{i=1}^{s} W_i)_p \cap (\bigcup\limits_{l=1}^{m} W_l)_{k-1}], \text{ где } i = \overline{1, s} \text{ и } l = \overline{1, m}, \end{cases}$$

здесь *m*- число операций на *(k-1)*-ом ярусе, *s*- число операций на любом *p*-ом ярусе , начиная с 0-го и заканчивая *(k-2)*-ым. Введенное условие комментирует Рис.2.

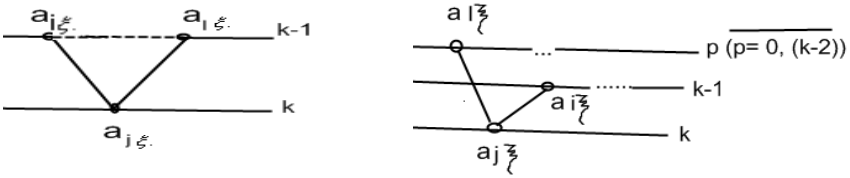


Рисунок 2- Условия вынесения бинарной операции на *k* -ый ярус

Таким образом, номер операции *j*, выносимой на *k* -ый ярус принадлежит объединению множеств, являющихся результатами попарных пересечений множеств $(W_i)_{k-1}$ на *(k-1)*-ом ярусе. Номер операции *j* принадлежит любому из пересечений множеств W_i на ярусах от 0-го до *(k-2)*-го и на *(k-1)*-ом ярусе (объединение пересечений множества W_i с любого яруса от 0-го до *(k-2)*-го с множеством W_i с *(k-1)*-ого яруса).

3. Для операции a_j с тремя входами данные могут быть подготовлены:

а) двумя операциями на любом из ярусов от *0*-го до *(k-2)*-го и одной из операций на *(k-1)*-ом ярусе;
б) двумя операциями на *(k-1)*- ярусе и одной операцией на любом из ярусов от *0*-го до *(k-2)*-го;
в) тремя операциями на трех различных ярусах от *0*-го до *(k-1)*-го;
г) тремя операциями на *(k-1)*-ом ярусе.

В этом случае для тернарной операции a_j условия ее вынесения на *(k)*-ый ярус ЯПФ имеют вид:

$$j \in \begin{cases} \bigcup\limits_{p=0}^{k-2} \{ [\bigcup\limits_{i,l} [W_i \cap W_l)_p] \cap (\bigcup\limits_{q-0}^{s} W_q)_{k-1} \}, \; l = \overline{i+1,m}, i = \overline{1,m-1}; & (5) \\ \bigcup\limits_{p=0}^{k-2} \{ [\bigcup\limits_{i,l} [W_i \cap W_l)_{k-1}] \cap (\bigcup\limits_{q-0}^{m} W_q)_p \}, \; l = \overline{i+1,s} \;\; i = \overline{1,s-1}; & (6) \\ \bigcup\limits_{h,p} [(\bigcup\limits_{i=0}^{m} W_i)_h \cap (\bigcup\limits_{l=0}^{r} W_l)_p \cap (\bigcup\limits_{q-0}^{m} W_q)_{k-1}], h = \overline{p+1,k-2} \;\; p = \overline{0,k-3}; & (7) \\ \bigcup\limits_{i,l,q} [W_i \cap W_l \cap W_q]_{k-1}, \; i = \overline{1,s-2}, \; l = \overline{i+1,s-1}, \; q = \overline{l+1,m} & (8) \end{cases}$$

Здесь *i,l,q*-индексы операций на некотором ярусе (*p*-ом либо *(k-1)*-ом), *s*- количество операций на *(k-1)*-ом ярусе, *m*- количество операций на некотором *p*-ом ярусе ($p = \overline{0,k-2}$), *h* и *p* – индексы ярусов ЯПФ.

Введенные условия комментирует Рис.3.

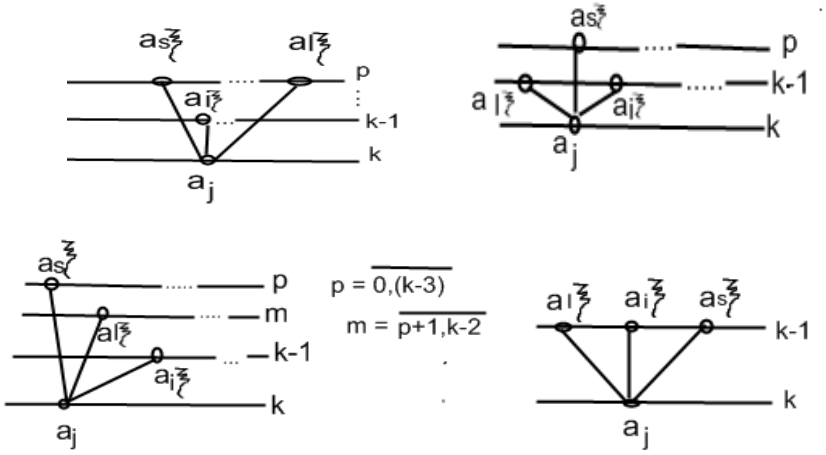


Рисунок 3- Условия вынесения тернарной операции a_j на *k*-ый ярус ЯПФ

Выражение (5) определяет принадлежность номера j операции a_j объединению (по ярусам) попарных пересечений множеств W_i на любом из ярусов начиная с *0*-го и заканчивая *(k-2)*-ым (на любом *p*-ом ярусе) и любому из множеств W_i на *(k-1)*-ом ярусе.

Выражение (6) определяет принадлежность номера j операции a_j любому из множеств W_i на любом *p*-ом ярусе (начиная с *0*-го и заканчивая *(k-2)*-ым) и любому попарному пересечению множеств W_i на *(k-1)*-ом ярусе.

Выражение (7) определяет принадлежность номера j операции a_j трем различным множествам W_i, находящихся на различных ярусах ЯПФ от *0*-го до *(k-2)*-ого (на *h*-ом и *p*-ом ярусах) и на *(k-1)*-ом ярусе.

Выражение (8) определяет принадлежность номера j операции a_j любому из трехкратных пересечений множеств W_i на *(k-1)*-ом ярусе.

Для операций с произвольным количеством входов условия их вынесения на *k*-ый ярусе могут быть сформулированы аналогично.

3. Программа выполнения работы

В соответствии с заданием должно быть реализовано построение ярусно-параллельной формы программы, выполняющей назначенные задания. Для этого необходимо:

- построить блок-схему последовательной программы;
- для каждой операции программы необходимо осуществить ее представление в форме (1). При этом предложить способ обозначения типов операций. Представление операций в виде (1) оформить в виде отдельной таблицы. Для исключения возникновения возможных ограничений при распараллеливании выполнять уникальное именование результатов выполнения каждой операции;
- определить для модели процесса *M0* – начальное состояние памяти.
- сформировать по таблице множество ${}^{*}A_0$ верхнего уровня ЯПФ;
- для каждой операции на этом уровне сформировать множества $(W_i)_0$ номеров операций, имеющих преемников на ниже лежащих уровнях;
- Используя понятие *n*-арности операции и условия нахождения операций на ниже стоящем уровне, связанных с множеством $(W_i)_0$, определить множество операций ${}^{*}A_1$ находящихся на первом уровне иерархии;
- рассуждая аналогичным образом, сформировать множества ${}^{*}A_k$ для остальных уровней ЯПФ;
- представить графический вид ЯПФ. Т.к. пути, проходящие на графе ЯПФ имеют одинаковую длину, то все операции будут иметь одинаковый приоритет, т.е. при возможности будут выполняться одновременно (принцип неограниченности ресурсов);
- для каждого уровня ЯПФ определить множества ${}^{0}A_k$, ${}^{*}A_k$, ${}^{+}A_k$, ${}^{p}A_k$, ${}^{-}A_k$ (учитывать все операции программы). Сформированные множества свести в специальную таблицу;

- на основании ЯПФ реализовать параллельную программу. Каждую операцию на уровнях ЯПФ реализовать в виде отдельно процесса, выполняющегося параллельно с другими. Процессы более высокого уровня иерархии синхронизируют процессы следующего за ним уровня, подготовив для них требуемые данные. Условие синхронизации процессов на ниже следующем уровне в случае готовности для них исходных данных интерпретируются формулой (3). В состав программы ввести программные средства, представляющие в ходе выполнения действительно параллельный характер реализации задачи. Распечатку параллельной реализации процесса счета представить в отчете (с учетом синхронизации ниже находящихся в графе процессов выше стоящими). Распечатку результатов работы программы также необходимо представить в отчете.
- выполнить формирование отчета следующего содержания:
  - полная формулировка задания на работу;
  - схема последовательной программы;
  - представления операций a_j программы в форме (1), обобщенные в виде таблицы;
  - состав множеств $(W_i)_k$ (k- номер яруса, i - номер операции) для каждой операции каждого ярусе ЯПФ. Результаты выполнения условия нахождения n-арной операции a_j на k-ом шаге ЯПФ в множестве ${}^{*}A_k$ (результаты формирования множеств ${}^{*}A_k$ для каждого уровня ЯПФ), состав множеств ${}^{0}A_k$, ${}^{*}A_k$, ${}^{+}A_k$, ${}^{p}A_k$, ${}^{-}A_k$ для каждого k-го уровня иерархии;
  - представить графический вид ЯПФ;
  - распечатка текста программы реализующей полученную ЯПФ;
  - распечатка результатов выполнения программы;
  - распечатка хода выполнения программы и синхронизации одними процессами других, поясняющая действительный параллельный ход выполнения программы.

4. Задание на работу

Осуществить распараллеливание последовательных программ в соответствии с вариантами:

1 вариант.

Вычислить полином $Y=a_0+a_1x_1+\ldots+a_5x_5$, где

$a_0=(ab+cd)/(g-h)$; $a_1=(ac-h)/(f+ae)$; $a_2=df-b$; $a_3=fc-ab/g$;

$a_4=a-h+ce/(be-h)$; $a_5=fc-ag/(fh-b)$.

Выполнить максимальное распараллеливание задачи.

2 вариант.

Вычислить полином $Y=a0\varphi0+a1\varphi1+\ldots+a5\varphi5$, где

A_i – коэффициент разложения (определяются как вводимые переменные),

φ_i – параболы *i*-го порядка.

$\varphi_0 = 1;\ \varphi_i = x-(n+1)/2;\ \varphi_{k+1} = \varphi_1\ \varphi_k - \varphi_{k-1}k^2(n^2-k^2)/(4(4k^2-1))$

n – число точек, в которых задано значение функции Y (*n*=5)

3 вариант.

$Y=A_0+A_1\varphi_1(x)+A_2\varphi_2(x)+A_3\varphi_3(x)+A_4\varphi_4(x)+A_5\varphi_5(x)$

$\varphi_k = \sqrt{2}\ sin(knx).$

5. Контрольные вопросы

1. Какие множества операторов используются при формировании двумерной модели вычислительного процесса?
2. Что представляет собой обобщенная форма представления операций в программе? Ее компоненты?
3. В чем заключаются условия построения множеств двумерной модели вычислительного процесса?
4. В чем состоит смысл операции переименования при построении ЯПФ?
5. Вид двумерной модели вычислительного процесса. Интерпретация множеств операций модели (компоненты двумерной модели вычислительного процесса)?
6. Какой вид имеют условия отсутствия конкуренции и условие готовности операции к выполнению?
7. Какие шаги входят в алгоритм формирования (интерпретации) функций *F1* и *F2* двумерной модели вычислительного процесса?
8. Какой вид имеют условия вынесения операции a_j на *k*-ом уровне ЯПФ в множестве $^*A_k$ (с использованием понятия готовности данных и множеств $(W_i)_k$)- для унарных, бинарных и тернарных операций?

ЛАБОРАТОРНАЯ РАБОТА №3

ИССЛЕДОВАНИЕ МЕТОДА СИНТЕЗА ПАРАЛЛЕЛЬНЫХ ПРОГРАММ НА ОСНОВЕ ГРАФА УПРАВЛЕНИЯ С ИСПОЛЬЗОВАНИЕМ ПОНЯТИЙ ОБРАЗА И ПРООБРАЗА ВЕРШИНЫ

1. Цель работы: исследование сущности и особенностей методов распараллеливания последовательных ациклических программ с использованием моделей вычислений.

2. Теоретическое введение

2.1. Модели последовательных программ

Свойства программ определяют их информационную структуру, которая характеризует ее элементы и связь их друг с другом. Возможны следующие подходы к изучению информационной структуры программы: а) денотационный – исследование состояния памяти, (соответствующих состояний переменных) и процесса изменения состояния памяти в ходе выполнения программы; б) операционный – ход выполнения программы представляется в виде набора исполняемых операций, связанных между собой. Операционный подход порождает графовые модели программ. Графовые модели характеризуются множеством вершин, соответствующих действиям в программе, множеством дуг, определяющих отношения разного вида (по управлению, по данным и т.д.) между операциями в программе.

Операционный подход определяет два типа блоков в моделях программ: а) преобразователи– выполняют действия с данными; б) распознаватели – определяют порядок срабатывания преобразователей.

При обозначении через S – множество преобразователей, R – множество распознавателей, тогда $S \cap R = \varnothing$. Определение видов графовых моделей выполним применительно к рассмотренному тексту программы:

1. y=b_1/ a_1;
2. x=y;
3. for i=1 to n do
4. x=(b_i- c_ix)/a_ix;
5. if (x<=y) goto 7;
6. y=x;
7. end for.

Для программы должны быть определены данные:

$$b_1, b_2, \ldots, b_n$$

$$a_1, a_2, \ldots, a_n$$

$$c_1, c_2, \ldots, c_n$$

Существуют два типа отношений между действиями в программе:
1) характеризуется выполнением одного действия непосредственно за другим, запуск последующего действия выполняется после окончания реализации предшествующего; такой тип отношения называется связью по управлению, т.е. связь по управлению определяет последовательность запуска операторов программы;
2) характеризуется использованием в качестве аргументов для нижестоящих операторов результатов выполнения вышестоящих операторов, тип связи – информационная (отношения связи по данным).

Оба типа отношений упорядочивают множество операторов в программе.

2.2. Граф управления последовательной программы

Вершина графа является оператором (преобразователь либо распознаватель). Между вершинами – отношение передачи управления, определяется взаимодействие между операторами, соответствующее передаче управления (активизация оператора после выполнения оператора, предшествующего ему по управлению). Вид графа управления программы на Рис.1.

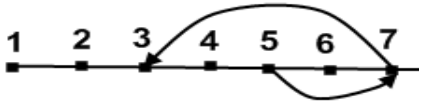


Рисунок 1 – Вид графа управления программы

2.3 Информационный граф программы

Основа графа – вершины–преобразователи. В результате граф определяет поток информации между операторами в программе. Дуги определяют отношения информационной зависимости между операторами – вершинами. Таким образом, вершины могут быть связаны дугами, если между ними существует информационная зависимость. Вид информационного графа программы представлен на Рис.2.

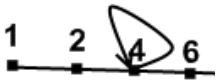


Рисунок 2 – Вид информационного графа программы

Т.о. свойства программы характеризуются либо множеством операторов, либо множеством переменных, характеризующих состояние памяти. Система управляющих связей в программе (связей по управлению), определяющих последовательность или порядок запуска операторов на выполнение, задается в виде графа управления (графа потока управления).

Дуга графа определяет передачу управления между операторами. При разделении программы на операторы отсутствует информация о внутренней структуре оператора и структуре памяти.

2.4. Формализация представления управляющего графа программы

Задан алфавит (совокупность) символов операторов A, состоящий из символов, обозначающих преобразователи и распознаватели (для которых $k>1$, где k– количество управляющих выходов (дуг) распознавателя). Функция (оператор, способ, алгоритм) $L(p)$ определяет сопоставление некоторой вершине p графа соответствующего ей оператора программы: $A=\{a,b,c,...\}$, $a \in A$, $L(p)=a$ (т.о. через $L(p)$ может быть обозначен оператор программы, соответствующий вершине p). Тогда через L может быть обозначена разметка графа G (сопоставление всем возможным вершинам графа G соответствующих им операторов). Тогда управляющий граф программы может быть представлен в виде (G,U,L), где G – множество вершин графа, U– множество его дуг (связей по управлению между операторами программы), L – разметка графа, сопоставляющая вершинам соответствующие им операторы. Пример управляющего графа программы на Рис.3.

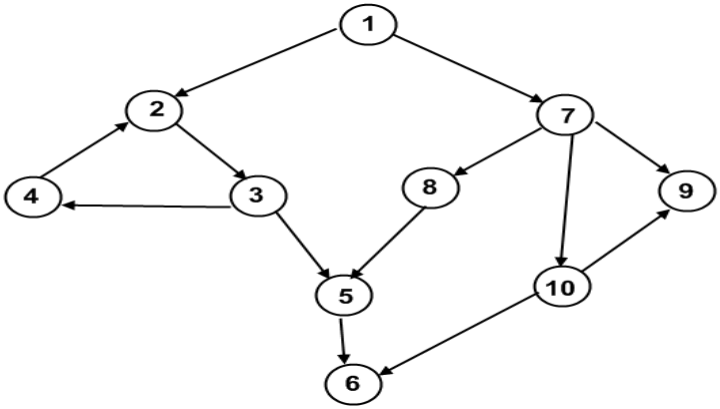


Рисунок 3 – Управляющий граф программы

Управляющий граф программы является правильным, если каждая его вершина принадлежит одному из путей из начальной вершины в конечную.

Транзитивные замыкания – метод потокового анализа программ (транзитивное замыкание отношений следования между операторами программы (вершинами управляющего графа программы). Если два оператора связаны отношением следования (непосредственное следование), тогда на

графе управления последовательной программы между соответствующими им вершинами присутствует дуга. Транзитивные замыкания отношений следования между операторами программы (между вершинами управляющего графа) предполагает наличие пути между соответствующими (рассматриваемыми) вершинами. Т.о. анализ потока управления для операторов программы предполагает определение пути на графе управления между соответствующими вершинами. При анализе потоков управления определяются отношения обязательного предшествования для операторов программы (соответственно, вершин графа), то есть какие операторы должны предшествовать рассматриваемому. Аналогично определяются отношения обязательной преемственности (транзитивное замыкание тех вершин графа, которые являются последующими за данной). Т.о. все вершины, лежащие на пути из исходной вершины в конечную через рассматриваемую вершину (до рассматриваемой вершины), связаны с ней отношением обязательного предшествования. Все вершины, лежащие на пути из исходной вершины в конечную через рассматриваемую вершину (после рассматриваемой вершины), связаны с ней отношением обязательного преемственности.

2.5 Понятие графа зависимости между операторами

Информационные связи между операторами могут быть определены на графе зависимости. Отношение информационной связи заменяется отношением зависимости. Две операции являются зависимыми, если при выполнении они обращаются к одной и той же переменной (ячейке памяти). Т.о. если информационная связь реализуется путем выполнением операции чтения – записи над одной и той же переменной программы, тогда информационная связь является частным случаем зависимости. Зависимость между операторами программы возникает и в том случае, если между ними есть конкуренция за переменную либо за значение. Пример графа зависимости представлен на Рис.4. Переменные *x, y* являются изменяемые в ходе выполнения программы. Значения *x, y* определяются в операторах 1,2,4,6.

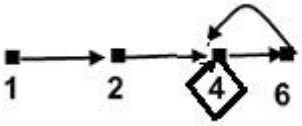


Рисунок 4 – Вид графа зависимостей

Таким образом, информационный граф программы является подграфом графа зависимости. Из графа зависимости вытекает понятие графа влияния программы.

2.6. Понятие графа влияния

Транзитивное замыкание непосредственных зависимостей позволяет получить граф влияния. Операция (оператор) программы влияет на другую (транзитивная зависимость), если изменение значения переменных, которые вычисляются в первой операции, влияет на значение переменной, вычисляемой в другой операции. Таким образом, граф влияния предполагает соединение дугой каждой пары вершин, в которых одна операция влияет на другую (направление дуги графа совпадает с направлением влияния между операторами).Отношения управления, информационной связи и зависимости не являются транзитивными, отношение влияния транзитивно. Граф влияния – это представление транзитивного замыкания графа зависимости. Пример графа влияния– на Рис 5 (изменение значений в операторах 1 и 2 влияет на значения переменных, вычисляемых в операторах 4 и 6).

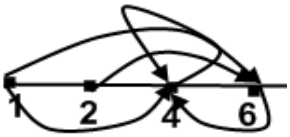


Рисунок 4 – Вид графа влияния

2.7.Теоретико-графовый подход к распараллеливанию последовательных алгоритмов

Задача представляется в виде последовательности операторов $A=\{a_1,a_2,a_3,\ldots a_n\}$ (программа представляется в виде последовательности операторов заданного вида). Т.о. A – упорядоченное множество операторов программы. Задача распараллеливания состоит в выделении из множества A операторов программы подмножеств Ω_i взаимно независимых операторов, где i определяет некоторый ярус ЯПФ. Тогда множество Ω_i – это операторы программы, которые могут быть выполнены совместно (параллельно) на i-ом ярусе ярусно-параллельной формы. Через Ω_0 обозначено множество операторов программы, зависящих от ее входных данных. Если результаты выполнения операций одного яруса являются исходными данными для операторов другого яруса, тогда для совокупности подмножеств Ω_i выполняются следующие условия: $\Omega_i \neq \varnothing$, $\Omega_i \cap \Omega_j = \varnothing$, $\bigcup_i \Omega_i = A$.

Задача распараллеливания может быть сформулирована в соответствии с теорией графов следующим образом. Программе, соответствующей множеству операторов A, может быть поставлен в соответствие граф

$G(A,U)$. Дуга $(a_j, a_i) \in U$ соединяет две вершины в случае если результат j-го оператора является аргументом для i-го. Для рассмотрения задачи распараллеливания последовательной программы в теоретико-графовой постановке в рассмотрение введены следующие обозначения.

Обозначим через $\Gamma\Omega_j$ множество вершин, последующих за j -м ярусов, в которые ведут дуги из вершин данного j -го яруса, тогда $\Omega_i \subseteq \Gamma\Omega_j$ (множество вершин Ω_i i-го яруса, в которые ведут дуги из вершин множества Ω_j j-го яруса). Через Γa_j обозначим множество вершин последующих ярусов, в которые ведут дуги из вершины a_j, тогда $a_i \in \Gamma a_j$ (вершина a_i – вершина, в которую ведет дуга из вершины a_j).

Таким образом, $\Gamma\Omega_j$ и Γa_j соответствуют множествам вершин, соединенных дугами с вершинами множества Ω_j либо с вершиной a_j. Тогда распараллеливание – это задача разбиения вершин графа $G(A,U)$ на подмножества Ω_j так, чтобы удовлетворить введенные условия и множество $\Omega_i \subseteq \Gamma\Omega_j$ (при $i>j$). Для каждой операции $a_i \in \Omega_s$ предполагается наличие другой операции $a_j \in \Omega_{s-1}$ такой, что $a_i \in \Gamma a_j$.

2.8. Транзитивные замыкания зависимостей между операторами

Если Γ – отношение, которое определяет непосредственную связь между вершинами (операторами) a и a' в графе управления, тогда через $Tr(\Gamma)$ обозначим бинарное отношение, которое выполняется для пары операторов (вершин) a и a' в том случае, если между ними существует последовательность операторов (вершин) вида $a, a_i, a_{i+1}, \ldots a_{i+k}, a'$, а каждая пара соседних элементов в этой последовательности связана отношением Γ: $a_i \in \Gamma a$, $a_{i+1} \in \Gamma a_i$,..., $a_{i+k} \in \Gamma a_{i+(k-1)}$, $a' \in \Gamma a_{i+k}$ (т.о. $aTr(\Gamma)a'$, все вершин последовательности (пути на графе) являются попарно связанными с использованием дуг).

Под транзитивным замыканием для вершины a_i (на множестве A) подразумевается выполнение отношения $Tr(\Gamma)$ для вершин a_i и некоторой вершины a_j (совокупности вершин a_j). Таким образом, наличие пути из a_i в a_j свидетельствует о выполнении бинарного отношения $Tr(\Gamma)$ для вершин a_i и a_j (то есть наличие пути характеризует транзитивное

замыкание $Tr(\Gamma)$ для вершины a_i и связанных с нею вершин a_j).

Транзитивный образ вершины a_i – это множество вершин графа $G(A,U)$, связанных отношением $Tr(\Gamma)$ с этой вершиной, то есть образ – это все вершины a_j, в которые ведут пути из данной вершины a_i при условии, что выполняется условие транзитивного замыкания в виде: $a_1 \in \Gamma a_i$, $a_2 \in \Gamma a_1$,..., $a_j \in \Gamma a_{i+k}$. При $k \geq 0$ предполагается, что вершины либо непосредственно следуют за рассматриваемой a_i, либо соединены с данной вершиной некоторым путем.

2.9. Формирование транзитивного замыкания

Множество $Tr(a_i)=\{a_1,a_2,...,a_j\}$ – это вершины, для которых выполняется отношение $Tr(\Gamma)$ транзитивного замыкания с вершиной a_i. Тогда образ вершины a_i - это множество вершин $Tr(a_i)=\{a_1,a_2,...,a_j\}$, в которые ведут пути из рассматриваемой вершины a_i. Таким образом, формирование транзитивного замыкания для вершины a_i – это определение множества $Tr(a_i)$ вершин графа, связанных с рассматриваемой вершиной отношением $Tr(\Gamma)$. Тогда образ вершины a_i (транзитивный образ) – это множество $Tr(a_i)$ вершин, связанных отношением $Tr(\Gamma)$ с рассматриваемой вершиной a_i (вершины, в которые ведут пути из рассматриваемой вершины a_i, т.е. зависящих от результатов оператора a_i).

На Рис.5 приведен пример управляющего графа программы, используемого для пояснения понятия образа вершины графа. В обозначении вершины $a_{k_{sj}}$ индекс s соответствует номеру яруса ярусно-параллельной формы, индекс j – номер операции на s-ом ярусе. Тогда $a_{k_{sj}}$ – обозначение вершины, в которую ведут пути из исходной рассматриваемой вершины a_i. Т.е. вершина $a_{k_{sj}}$ связана отношением $Tr(\Gamma)$ с исходной вершиной a_i ($a_i Tr(\Gamma) a_{k_{sj}}$). В итоге транзитивный образ вершины a_i имеет следующий вид: $Tr(a_i)=\{a_{k_{11}},a_{k_{12}},,...,a_{k_{33}},a_{k_{34}}\}$, где вершины $a_{k_{sj}}$ связаны с вершиной a_i отношением $Tr(\Gamma)$. Тогда определение множества $Tr(a_i)$ реализует замыкание для вершины a_i и вершин $a_{k_{sj}}$ бинарного отношения $Tr(\Gamma)$.

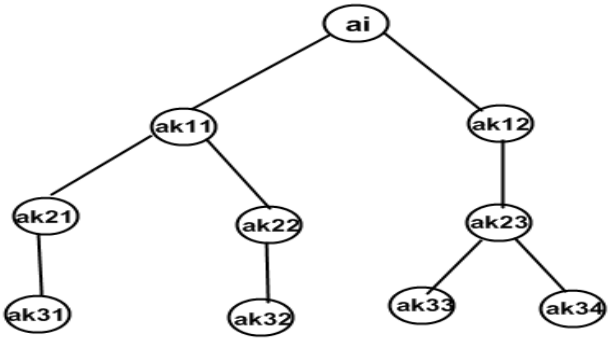


Рисунок 5 – Управляющий граф программы, используемый для пояснения понятия образа вершины

2.10. Формализация транзитивного замыкания (способа построения транзитивного образа вершины a_i)

Для формализации способа определения транзитивного образа вершины a_i дополнительно введены в рассмотрение множества $S^{(k)}$ и $T^{(k)}$, где $S^{(k)}$ – вершины, рассматриваемые на *k*-ом шаге алгоритма, $T^{(k)}$ – образ вершины a_i , сформированный на предыдущих и текущем k-ом шаге алгоритма. При этом перед началом определения транзитивного образа вершины a_i предполагается, $S^{(0)} = \{ a_i \}$, т.е. множество $S^{(0)}$ содержит исходную вершину, и $T^{(0)} = \varnothing$. Также введено в рассмотрение обозначение $\Gamma(S^{(k)})$ – множество вершин графа, в которые ведут дуги из вершин множества $S^{(k)}$. Предполагается, что $S^{(k)}$ – это вершины графа, связанные с исходной вершиной a_i отношением $Tr(\Gamma)$, рассматриваемые на текущем шаге алгоритма.

Тогда для определения транзитивного образа вершин графа реализуется рекурсивная процедура:

$$T^{(k+1)} = T^{(k)} \cup \Gamma(S^{(k)}); \; S^{(k+1)} = T^{(k+1)} \setminus T^{(k)},$$

где $\Gamma(S^{(k)})$ – множество вершин графа, в которые ведут дуги из вершин множества $S^{(k)}$, *k*– номер шага процедуры, $T^{(k)}$ – дополняемое множество (дополняемое на *(k+1)*-ом шаге алгоритма) всех вершин графа, связанных

путями с рассматриваемой вершиной a_i.

Пример реализации рекурсивной процедуры определения транзитивного образа вершины a_i для графа на Рис.5:

$$S^{(0)} = \{ a_i \};\ T^{(0)} = \varnothing;$$

$$T^{(1)} = T^{(0)} \cup \Gamma(S^{(0)}) = \{ a_{k11}, a_{k12} \};$$

$$S^{(1)} = T^{(1)} \setminus T^{(0)} = \{ a_{k11}, a_{k12} \};$$

$$T^{(2)} = T^{(1)} \cup \Gamma(S^{(1)}) = \{ a_{k11}, a_{k12} \} \cup \{ a_{k21}, a_{k22}, a_{k23} \} =$$

$$= \{ a_{k11}, a_{k12}, a_{k21}, a_{k22}, a_{k23} \};$$

$$S^{(2)} = T^{(2)} \setminus T^{(1)} = \{ a_{k21}, a_{k22}, a_{k23} \};$$

$$T^{(3)} = T^{(2)} \cup \Gamma(S^{(2)}) = \{ a_{k11}, a_{k12}, a_{k21}, a_{k22}, a_{k23} \} \cup$$

$$\cup \{ a_{k31}, a_{k32}, a_{k33} \} = \{ a_{k11}, a_{k12}, a_{k21}, a_{k22}, a_{k23}, a_{k31}, a_{k32}, a_{k33} \};$$

$$S^{(3)} = T^{(3)} \setminus T^{(2)} = \{ a_{k31}, a_{k32}, a_{k33} \}.$$

Для окончания процедуры формирования образа вершины a_i должно быть выполнено условие $S^{(k+1)} = T^{(k+1)} \setminus T^{(k)} = \varnothing$. Тогда

$$T^{(4)} = T^{(3)} \cup \Gamma(S^{(3)}) = T^{(3)} \cup \varnothing = T^{(3)};$$

$S^{(4)} = T^{(4)} \setminus T^{(3)} = \varnothing$. Тогда $Tr(a_i) = T^{(4)}$.

Транзитивный образ вершины определяет те вершины графа, которые должны находиться на нижестоящих ярусах ЯПФ. Соответствующий ярус ЯПФ формируется на основе множества $S^{(k+1)}$.

Обратное транзитивное замыкание предполагает выполнение бинарного отношения $Tr^{-1}(\Gamma)$, которое реализуется в том случае, если для пары вершин a_i, a_j существует дуга (a_i, a_j). Если прямое отношение $Tr(\Gamma)$ связывает вершины a_i, a_j в виде $a_i Tr(\Gamma) a_j$ (предшествующая вершина a_i – последующая вершина a_j, т.е. наличие дуги из вершины a_i в вершину a_j), то обратное отношение $Tr^{-1}(\Gamma)$ связывает вершины в виде $a_j Tr^{-1}(\Gamma) a_i$ (последующая вершина a_j – предшествующая вершина a_i, т.е. наличие дуги в вершину a_j из вершины a_i). Тогда транзитивный прообраз вершины a_j, обозначенный как $Tr^{-1}(a_j)$, представляет собой множество таких вершин a_i, для которых (совместно с вершиной a_j) выполняется отношение $Tr^{-1}(\Gamma)$. Таким образом, прообраз вершины a_j, обозначенный

как $Tr^{-1}(a_j)$ – это те вершины a_i графа, выполнение которых должно предшествовать выполнению вершины a_j.

2.11. Формализация процедуры определения транзитивного прообраза вершины a_j

Если $\Gamma(S^{(k)})$ – множество вершин графа, в которые ведут дуги из вершин множества $S^{(k)}$, тогда $\Gamma^{-1}(S^{(k)})$ – это множество вершин графа, из которых ведут дуги в вершины множества $S^{(k)}$. По аналогии с процедурой определения транзитивного образа вершины a_i рекурсивная процедура определения транзитивного прообраза вершины a_j имеет следующий вид:

$$S^{(0)} = \{a_i\},\ T^{(0)} = \varnothing;$$
$$T^{(k+1)} = T^{(k)} \cup \Gamma^{-1}(S^{(k)});$$
$$S^{(k+1)} = T^{(k+1)} \setminus T^{(k)}.$$

Использование процедуры определения прообраза вершины графа прокомментируем на примере управляющего графа программы на Рис.5, используя в качестве исходной вершины (вершины, для которой определяется прообраз) вершину a_{k31}.

$$S^{(0)} = \{a_{k31}\};\ T^{(0)} = \varnothing;$$
$$T^{(1)} = T^{(0)} \cup \Gamma^{-1}(S^{(0)}) = \{a_{k21}\};$$
$$S^{(1)} = T^{(1)} \setminus T^{(0)} = \{a_{k21}\};$$
$$T^{(2)} = T^{(1)} \cup \Gamma^{-1}(S^{(1)}) = \{a_{k21}\} \cup \{a_{k11}\} = = \{a_{k21}, a_{k11}\};$$
$$S^{(2)} = T^{(2)} \setminus T^{(1)} = \{a_{k11}\};$$
$$T^{(3)} = T^{(2)} \cup \Gamma^{-1}(S^{(2)}) = \{a_{k21}, a_{k11}, a_i\};$$
$$S^{(3)} = T^{(3)} \setminus T^{(2)} = \{a_i\};$$
$$T^{(4)} = T^{(3)} \cup \Gamma^{-1}(S^{(3)}) = T^{(3)} \cup \varnothing = \{a_{k21}, a_{k11}, a_i\};$$
$$S^{(4)} = T^{(4)} \setminus T^{(3)} = \varnothing.$$

Т.к. $S^{(4)} = \varnothing$, то процедура определения прообраза вершины a_{k31} должна быть завершена и $Tr^{-1}(a_{k31}) = \{a_{k21}, a_{k11}, a_i\}$. Таким образом, $T^{(k)}$ – это множество вершин (операторов), от выполнения которых

(непосредственно или опосредованно) зависит выполнение оператора a_j (т.е множество содержит вершины пути, ведущего из исходной вершины a_i в рассматриваемую вершину a_j, для которой формируется прообраз), а $S^{(k)}$ – множество вершин, которые должны выполняться на *k*-ом ярусе ЯПФ для того, чтобы впоследствии была активизирована вершина a_j.

Тогда задача распараллеливания алгоритма последовательной программы, представленной в виде управляющего графа, состоит в формировании множеств $S^{(k)}$ операторов, выполняемых на соответствующих *k*-ых ярусах.

2.12. Алгоритм формирования параллельной формы программы по графу управления на основе понятия транзитивного образа вершины

Через $\Gamma(a)$ обозначено множество вершин, в которые ведут дуги из данной (рассматриваемой) вершины *a*. Так как $S^{(k)}$ – множество вершин, выполняющихся на *k*-ом ярусе, тогда последовательность определения этих множеств при разных *k* может быть представлена в виде:

$$S^{(n)} = \{ a \mid a \in A; \Gamma a = \varnothing \}; \quad (1)$$

$$S^{(n-1)} = \{ a \mid a \in A; \Gamma a \subseteq S^{(n)} \}; \quad (2)$$

$$S^{(n-2)} = \{ a \mid a \in A; \Gamma a \subseteq [S^{(n)} \cup S^{(n-1)}] \}; \quad (3)$$

$$\ldots\ldots\ldots\ldots$$

$$S^{(0)} = \{ a \mid a \in A; \Gamma a \subseteq \bigcup_k S^{(k)} \}. \quad (4)$$

Выражение (1) определяет множество вершин (листьев дерева), из которых не ведут дуги ($\Gamma(a) = \varnothing$,т.е. множества $\Gamma(a)$ вершин, в которые ведут дуги из вершин множества $S^{(n)}$ является пустым). Выражение (2) определяет те вершины, которые соединены дугами с вершинами множества $S^{(n)}$(из вершины $a \in S^{(n-1)}$ идет дуга (дуги) в вершины множества $S^{(n)}$). По аналогии выражение (3) позволяет определить те вершины, из которых ведут дуги в вершины множеств $S^{(n-1)}$ и $S^{(n)}$, а выражение (4) позволяет определить вершины, из которых ведут дуги в вершины множеств $S^{(1)}$, $S^{(2)}$,..., $S^{(n-1)}$, $S^{(n)}$.

Пример графа управления программы и формирования ЯПФ. Формирование ЯПФ программы начинается с последнего яруса и заканчивается нулевым ярусом (Рис.6)

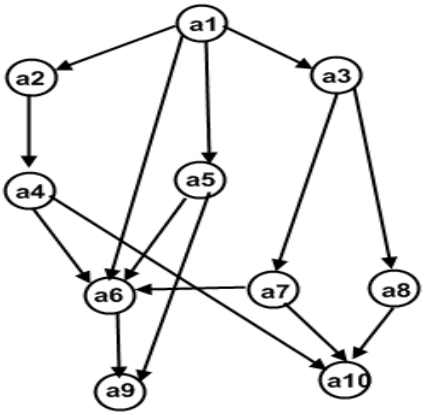


Рисунок 6 – Граф управления выполнение программы

Формализация построения множеств $S^{(k)}$ выполнена следующим образом:

$$S^{(n)} = \{ a_9, a_{10} \mid \Gamma a_9 = \varnothing, \Gamma a_{10} = \varnothing \};$$

$$S^{(n-1)} = \{ a_6, a_7, a_8 \mid \Gamma a_6 = \{ a_9 \} \subseteq S^{(n)}, \Gamma a_7 = \{ a_{10} \} \subseteq S^{(n)}, \Gamma a_8 = \{ a_{10} \} \subseteq S^{(n)} \};$$

$$S^{(n-2)} = \{ a_4, a_5 \mid \Gamma a_4 = \{ a_6, a_{10} \} \subseteq S^{(n)} \cup S^{(n-1)}, \Gamma a_5 = \{ a_6, a_9 \} \subseteq S^{(n)} \cup S^{(n-1)} \};$$

$$S^{(n-3)} = \{ a_2, a_3 \mid \Gamma a_2 = \{ a_4 \} \subseteq S^{(n-2)}, \Gamma a_3 = \{ a_5, a_7, a_8 \} \subseteq S^{(n-1)} \cup S^{(n-2)} \};$$

$$S^{(n-4)} = \{ a_1 \mid \Gamma a_1 = \{ a_2, a_3, a_5, a_6 \} \subseteq S^{(n-3)} \cup, S^{(n-2)} \cup S^{(n-1)} \}.$$

На основе сформированных множеств $S^{(k)}$ построена ярусно параллельная форма программы, которая имеет вид, представленный на Рис. 7.

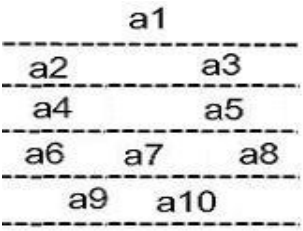


Рисунок 7– Ярусно-параллельная форма программы, сформированная с использованием понятия образа вершины графа

2.13 Алгоритм распараллеливания на основе понятия прообраза

Если $\Gamma^{-1}(a)$ – это множество вершин, из которых ведут дуги в данную

вершину a, тогда при $\Gamma^{-1}(a)=\{a_1,a_2,...a_l\}$ в вершину a ведут дуги из вершин a_1 , a_2 ,.., a_l . Обозначив через $S^{(k)}$ множество вершин, выполняющихся на k-ом ярусе ЯПФ, процесс формирования множеств, выполняющихся на k-ых ярусах может быть представлен в следующем виде:

$$S^{(0)}=\{a \mid a\in A;\Gamma^{-1}a=\varnothing\}; \quad (5)$$

$$S^{(1)}=\{a \mid a\in A;\Gamma^{-1}a\subseteq S^{(0))}\}; \quad (6)$$

$$S^{(2)}=\{a \mid a\in A;\Gamma^{-1}a\subseteq [S^{(0)}\cup S^{(1)}]\}; \quad (7)$$

$$\ldots\ldots\ldots\ldots$$

$$S^{(n)}=\{a \mid a\in A;\Gamma^{-1}a\subseteq \bigcup_{k=0}^{n-1} S^{(k)}\}. \quad (8)$$

Выражение (5) определяет множество вершин, в которые не ведут дуги ($\Gamma^{-1}(a)=\varnothing$, т.е. множества $\Gamma^{-1}(a)$ вершин, в которые ведут дуги являются пустыми). Выражение (6) определяет те вершины, в которые ведут дуги из вершин множества $S^{(0)}$, т.е. в соответствии с понятием отношения Tr^{-1} идентифицируются те вершины, для которых вершины множества $S^{(0)}$ являются предшествующими. По аналогии, $S^{(2)}$ – множество вершин, в которые ведут дуги из вершин множеств $S^{(0)}$ и $S^{(1)}$. Таким образом, формирование параллельной формы начинается с *0*-го яруса и заканчивается *n*-ым.

Пример формирования ярусно параллельной формы с использованием понятия прообраза вершины:

$$S^{(0)}=\{a_1 \mid \Gamma^{-1}a_1=\varnothing\};$$

$$S^{(1)}=\{a_2,a_3 \mid \Gamma^{-1}a_2=\{a_1\}\subseteq S^{(0)},\Gamma^{-1}a_3=\{a_1\}\subseteq S^{(0)}\};$$

$$S^{(2)}=\{a_4,a_5,a_7,a_8 \mid \Gamma^{-1}a_4=\{a_2\}\subseteq S^{(1)},\Gamma^{-1}a_5=\{a_1,a_3\}\subseteq S^{(0)}\cup S^{(1)},$$

$$\Gamma^{-1}a_7=\{a_3\}\subseteq S^{(1)},\Gamma^{-1}a_8=\{a_3\}\subseteq S^{(1)}\};$$

$$S^{(3)}=\{a_6,a_{10} \mid \Gamma^{-1}a_6=\{a_1,a_4,a_5,a_7\}\subseteq S^{(0)}\cup S^{(2)},\Gamma^{-1}a_{10}=\{a_4,a_7,a_8\}\subseteq S^{(2)}\};$$

$$S^{(4)}=\{a_9 \mid \Gamma^{-1}a_9=\{a_5,a_6\}\subseteq S^{(2)}\cup,S^{(3)}\}.$$

Итоговая ярусно параллельная форма представлена на Рис.8.

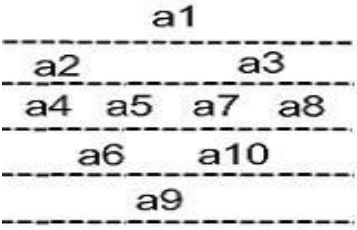


Рисунок 8– Ярусно-параллельная форма программы, сформированная с использованием понятия прообраза вершины графа

3. Программа выполнения работы

В соответствии с заданием должно быть реализовано построение ярусно-параллельной формы программы, выполняющей назначенные задания. Для выполнения задания лабораторной работы необходимо:

- сформировать граф управления для последовательной программы, выполняющей вычисление соответствующего арифметического выражения;
- в соответствии с приведенной в теоретическом введении процедурой для каждой вершины графа сформировать аналитически транзитивные замыкания образа и прообраза этой вершины; процесс формирования образа и прообраза представить в виде пошаговой реализации рекурсивной процедуры;
- в соответствии с приведенной в теоретическом введении процедурой аналитически сформировать ЯПФ программы с использованием понятия образа и прообраза вершины; процесс формирования ЯПФ представить в виде пошаговой реализации рекурсивной процедуры;
- в соответствии с вариантом задания разработать программу, реализующую пошаговое построение на основе графа управления образов либо прообразов вершин этого графа;
- в соответствии с вариантом задания разработать программу, реализующую пошаговое построение на основе графа управления ЯПФ с использованием понятий образов либо прообразов вершин этого графа;
- на основании ЯПФ реализовать параллельную программу, реализующую вычисление арифметического выражения, заданного в варианте;
- выполнить формирование отчета. следующего содержания:
  - полная формулировка задания на работу;
  - схема графа управления последовательной программы;
  - аналитический вид образов и прообразов вершин графа и пошаговая реализация рекурсивных процедур формирования образов и прообразов;
  - аналитический вид ЯПФ, соответствующей управляющему графу программы и пошаговая реализация рекурсивных процедур формирования ЯПФ;

- результаты реализации программы формирования образов и прообразов вершин графа и реализации программы формирования ЯПФ для управляющего графа;
- результаты функционирования параллельной программы вычисления арифметических выражений, реализующей сформированную ЯПФ;
- распечатка хода выполнения этой программы и синхронизации одними процессами других, поясняющая действительный параллельный ход выполнения программы.

4. Задание на работу

Осуществить формирование транзитивных замыканий образов и прообразов вершин, формирование ЯПФ и реализацию параллельной программы для управляющего графа, сформированного для последовательной программы, вычисляющей задаваемые в вариантах выражения.

Вариант 1. Z=B/(C+D)-K*(E+A), где

$$D=\begin{cases}\cos A, A<0;\\ \ln A, A>0;\\ 0, A=0.\end{cases} \qquad E=\begin{cases}B^2+D^2, B+D<0;\\ \sqrt{B+D}, B+D>0;\\ B^2-2D^2, B+D=0.\end{cases}$$

Вариант 2. L=(X+K)2/M-(J-K)/Z, где

$$K=\begin{cases}2X+Y, X>3;\\ 2Y+X, X<3;\\ X*Y, X=3.\end{cases} \qquad M=\begin{cases}\ln(K+5)+Z; K>-5;\\ |K+5|, K<-5;\\ Z-K, K=-5.\end{cases}$$

Вариант 3. G=(2D+3A)/(B+C)-E*(A+5B), где

$$D=\begin{cases}\sqrt{5B^2+1}, B>0;\\ |B+5|, B<0;\\ 0, B=0.\end{cases} \qquad E=\begin{cases}5D*C, D*C>0;\\ \cos DC, D*C<0;\\ D+5C, D*C=0.\end{cases}$$

Вариант 4. W=5(M+O2)-(3L+4K)*N/2, где

$$O=\begin{cases}L+M^2, L>M;\\ M^2-L, L<M;\\ 5L, L=M.\end{cases} \qquad K=\begin{cases}O+2N, O>5;\\ 2N-O, O<5;\\ (N+O)^2, O=5.\end{cases}$$

Вариант 5. N=(A+2B)/D-3E+A*(F+D), где

$$A=\begin{cases} E^2+2F, E+F>2; \\ \sqrt{(E+F)^2+1}, E+F<2; \\ 5E, E+F=2. \end{cases} \qquad B=\begin{cases} A+D^2, A>0; \\ |A|+\cos D, A<0; \\ 10, A=0. \end{cases}$$

Вариант 6. A=(N2+3F)/2-(G+N)*(W-2Z), где

$$N=\begin{cases} G+\cos W, W>0; \\ G-\sin W, W<0; \\ G, W=0. \end{cases} \qquad F=\begin{cases} \sin(G+Z), G+Z<0; \\ \ln(G+2Z), G+Z>0; \\ \cos(2Z-G), G+Z. \end{cases}$$

Вариант 7. *X*=cos(*I*+2*K*)*sin(*V*-4*L*)+*J*/(*L*+4*K*), где

$$V=\begin{cases} \sqrt{L^2+1}, L<0; \\ \sqrt{L+10}, L>0; \\ 5, L=0. \end{cases} \qquad J=\begin{cases} 10K, I=K; \\ 10I-K, I>K; \\ (I+K)^2, I<K. \end{cases}$$

Вариант 8. *I*=(2*J*+cos*C*)*5*D*-ln*M**(*N*+2*C*), где

$$C=\begin{cases} M*J, M>J; \\ M/J, M<J; \\ 0, M=J. \end{cases} \qquad D=\begin{cases} \ln(N-10), N>10; \\ |N^2-10|, N<10; \\ \cos N, N=10. \end{cases}$$

Вариант 9. *K*=(*X*+5*L*)/(*M*+*Z*)-2*J*+*L**M, где

$$L=\begin{cases} \ln(X^2+J^2), X>J; \\ J^2-X^2, X<J; \\ 100, X=J. \end{cases} \qquad M=\begin{cases} Z^2+10, Z<0; \\ 10-Z^2, Z>0; \\ 10, Z=0. \end{cases}$$

Вариант 10. $F=|2B-D|*E-(A2+E2)/C$, где

$$D=\begin{cases} A^2+\cos A, A>0; \\ \sin(A^2+10), A<0; \\ 10, A=0. \end{cases} \qquad E=\begin{cases} 2B/C, B>C; \\ \sqrt{|B+2C|}, B<C; \\ 0, B=C. \end{cases}$$

5. Контрольные вопросы

1. В чем заключаются подходы к изучению структуры программ?
2. Какие типы графовых моделей могут быть использованы при изучении структуры программ?
3. В чем заключается понятие транзитивного следования для операторов программ (понятие транзитивного замыкания отношения следования)?
4. В чем заключается понятие отношения предшествования и отношения преемственности для вершин управляющего графа программы?
5. Как в теоретико-графовой постановке интерпретируется наличие дуг, ведущих из вершин одного яруса ЯПФ в вершины другого яруса? Как в теоретико-графовой постановке интерпретируется наличие дуг, ведущих в вершины одного яруса ЯПФ из вершин другого яруса? Как интерпретируются аналогичные понятия для отдельной вершины?
6. Как интерпретируется понятие транзитивного замыкания зависимостей между операторами, каким образом интерпретируется понятие бинарного отношения транзитивного замыкания следования (зависимости) между вершинами?
7. В чем заключается понятие транзитивного образа вершины графа управления? Каким образом формализуется рекурсивная процедура построения транзитивного образа вершины управляющего графа? Какой вид имеет условие окончания процедуры?
8. В чем заключается понятие обратного транзитивного замыкания отношения зависимости?
9. В чем заключается понятие транзитивного прообраза вершины графа управления? Каким образом формализуется рекурсивная процедура построения транзитивного прообраза вершины управляющего графа? Какой вид имеет условие окончания процедуры?
10. Какой вид имеет процедура формирования ЯПФ с использованием понятия транзитивного образа вершины управляющего графа?
11. Какой вид имеет процедура формирования ЯПФ с использованием понятия транзитивного прообраза вершины управляющего графа?

ЛАБОРАТОРНАЯ РАБОТА №4
ИССЛЕДОВАНИЕ МЕТОДА ПОСТРОЕНИЯ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ НА ОСНОВЕ ГРАФ-СХЕМ, СОДЕРЖАЩИХ ОПЕРАТОРЫ УПРАВЛЕНИЯ

1. Цель работы: исследовать метод построения параллельных форм выполнения вычислительных алгоритмов, использующий граф-схемы, содержащие логические операторы.

2. Теоретическое введение

Граф-схема алгоритма отображает процесс взаимодействия между вершинами, соответствующими операторам программы. Если обозначить граф-схему алгоритма как $G(X,D)$, то $x_i \in X$ – вершины, соответствующие операторам программы, D - множество дуг двух типов – связи по управлению и по информации. Связь по управлению (логическая зависимость), может быть проинтерпретирована как связь по информации, т.к. она определяет изменение значения логической переменной.

В случае ограниченного количество ПЭ для реализации выполнения параллельных процессов необходимо составление расписания их реализации в ВС. Тогда с каждой вершиной сопоставляется значение $t_i^l (i = \overline{1,m})$ весов вершин, определяющих время выполнения каждого оператора. Если ВС является однородной, тогда значение t_i^l является одинаковым для всех ПЭ. Если ВС неоднородная (t_i^l являются различными для i-го оператора и каждого l-го ПЭ, $l = \overline{1,L}$), тогда каждому оператору ставится в соответствие вектор $\overline{t_i^l}$ следующего вида: $\overline{t_i^l} = (t_i^1, t_i^2, t_i^3, .., t_i^L)$.

Понятия свёртки и развёртки вершины графа

Свёртка k-й вершины граф-схемы – это наличие для k-й вершины n дуг графа (где $n \geq 1$ и n является конечным числом). Развёртка k-й вершины – это наличие для k-й вершины n исходящих дуг графа (где $n \geq 1$ и n – конечное число).

Пример свёртки и развёртки вершины графа представлен на Рис.1.

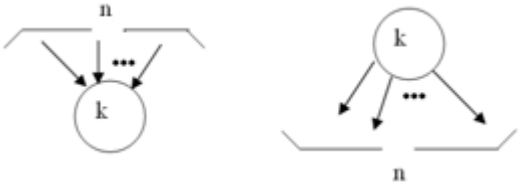


Рисунок 1– Пример свертки и развертки вершины графа

Элементарная свёртка и развёртка *k*-й вершины графа – это наличие одной входящей или одной выходящей дуг.

Виды дуг на графе

Может быть определено 2 множества дуг на графе (и, соответственно, два типа дуг): $D1 = D1.1 \cup D1.2$, где $D1.1$ - множество одиночных дуг графа, соответствующих элементарным свёрткам и развёрткам (соответствующих однократной передаче данных между вершинами графа); $D1.2$ - множество дуг реализующих функцию «И» граф-схемы (передача данных по всем дугам, являющихся исходящими из одной вершины, прием данных из всех дуг, являющихся входящими в данную вершину). Примеры дуг, входящих во множество $D1$.

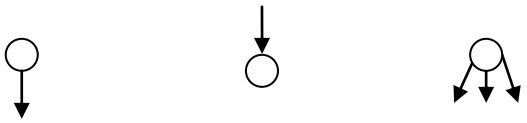
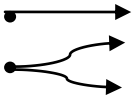


Рисунок 2 – Пример информационных дуг

Множество $D2$ – это множество дуг графа, реализующих функцию «исключающее ИЛИ» (передача управления по одной из дуг, входящих в развёртку вершины).

Вид дуги из множества $D2$.

Вид модифицированной дуги множества $D2$ (многохвостовая дуга):



Пример использования дуг множества $D2$ представлен на Рис.3.

Рисунок 3 – Пример управляющих дуг

Если граф содержит только дуги, входящие во множество $D1$, то это информационный граф программы (алгоритма) – информационная граф-схема алгоритма. Если граф содержит наряду с дугами множества $D1$ также дуги множества $D2$, то этот граф – информационно-логическая граф-схема алгоритма.

Пример информационный и информационно-логической граф-схем алгоритмов представлен на Рис.4.

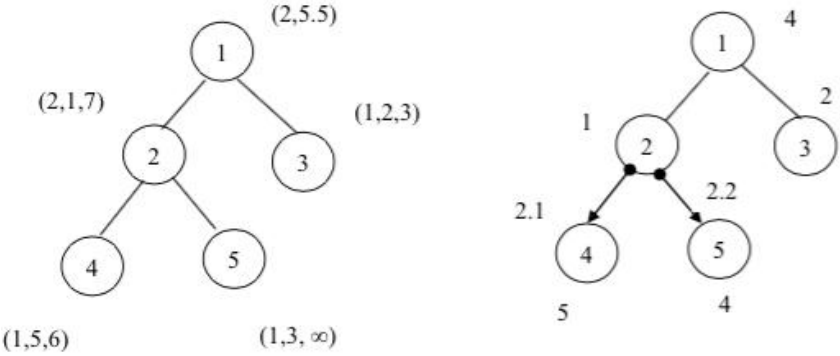


Рисунок 4 – Пример информационный и информационно-логической граф-схем алгоритмов: а) информационная граф-схема, используемая для решения задач на неоднородных системах; б) информационно-логическая граф-схема, используемая для решения задач на однородных системах

Формирование матрицы непосредственного следования для операторов программы

Если в алгоритме существует связь между операторами a_i и a_j (если в последовательной программе существует связь между операторами a_i и a_j), тогда на граф G должна быть указана дуга $d_S^1 \in D1$, исходящая из вершины a_i и входящая в вершину a_j. Тогда между a_i и a_j может быть определено отношение непосредственного следования: $a_i \to a_j$ (при этом a_i – блок преобразования данных). Если в алгоритме (в последовательной программе) существует связь между операторами a_i и a_j (при этом a_i - логический блок), то на графе G должна быть указана дуга $d_S^2 \in D2$ (где $d_S^2 = (a_i, a_j)$).

Между a_i и a_j определено отношение непосредственного следования по управлению (непосредственная логическая связь) $a_i \xrightarrow{a_i.n} a_j$, где n – номер исходящей логической дуги. Непосредственные связи $a_i \to a_j$ и $a_i \xrightarrow{a_i.n} a_j$ называются задающими связями.

На основе сформированных отношений информационного и логического следования ($\to$ и $\xrightarrow{a_i.n}$) формируется матрица

непосредственного следования для операторов программы S. Если $a_i \to a_j$, то $s_{ij} = 1$; если $a_i \xrightarrow{a_i.n} a_j$, то $s_{ij} = a_i.n$; остальные элементы матрицы равны 0. Если количество операторов равно *m*, количество ПЭ в неоднородной ВС равно *L*, тогда расширенная матрица следования S_R формируется путём добавления к исходной матрице S L вектор-столбцов по *m* элементов каждый (длительность реализации *i*-ых операторов $(i = \overline{1,m})$ на *l*-ых ПЭ $(l = \overline{1,L})$.

Виды графов информационно-логических связей и соответствующих им расширенных матриц следования представлены на Рис.5.

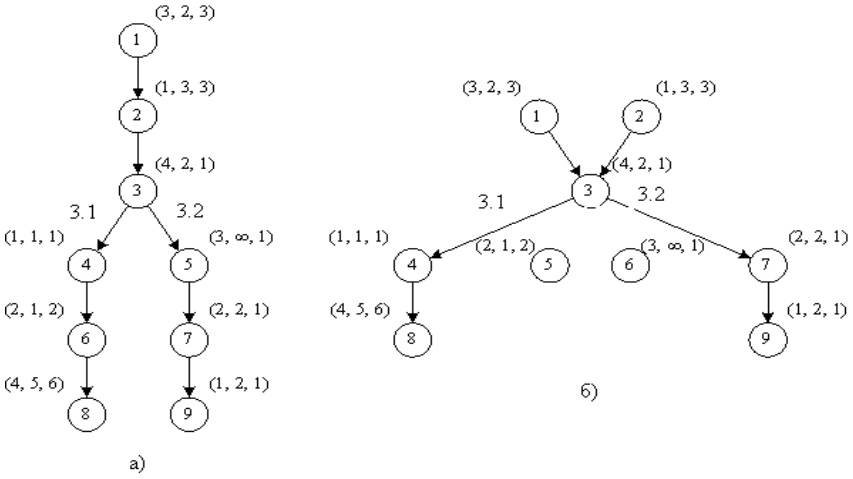


Рисунок 5– Виды графов информационно-логических связей и соответствующих им расширенных матриц следования представлены на

Матрицы (расширенные) S_R для информационно-логических схем алгоритмов (матрицы треугольного вида), представленных на Рис.5, приведены на Рис.6 (первая матрица для графа на Рис.5а), вторая матрица– для графа на Рис 5б)).

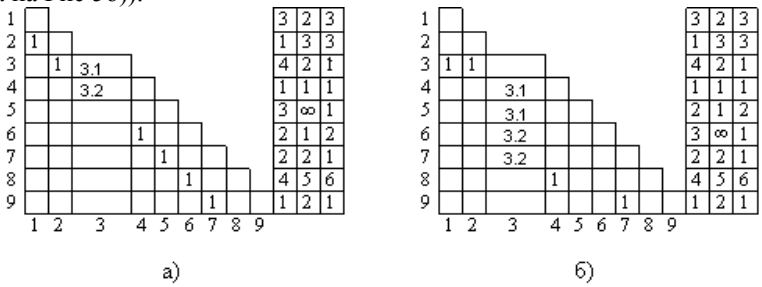


Рисунок 6– Виды расширенных матриц непосредственных связей между операторами

Т.к. главная диагональ содержит первые элементы, тогда на информационно-логической граф-схеме отсутствуют контуры.

Понятие транзитивных связей между операторами программы

Если $a_i \to a_j$ (при этом $s_{ji} = 1$) и $a_j \to a_k$ (соответственно, $s_{kj} = 1$) следовательно $a_i - \to a_k$ (где $- \to$ – обозначение отношения транзитивных связей между операторами a_i и a_k). Тогда должно быть сформировано множество транзитивных связей между операторами. Множество транзитивных связей полностью определяется множеством задающих связей.

Если $a_i \to a_j$, $A \to a_i$, где $A = \{ a_s \}$ – множество операторов, связанных с оператором a_i задающими связями, тогда все операторы $a_s \in A$ связаны транзитивно с оператором a_j . Таким образом, необходимо сформировать матрицу транзитивных связей, определяя тем самым все возможные зависимые элементы (все возможные связанные элементы).

Обозначим матрицу транзитивных связей как S^T и определим правила вычисления ее элементов. Для определения транзитивных зависимостей (формирования матрицы S^T) в рассмотрение введены операции $\otimes$ (операция транзитивной конъюнкции) и $\oplus$ (операция транзитивной дизъюнкции).

Если $a_i \to a_j$ ($s_{ji} = 1$) и $a_j \to a_k$ ($s_{kj} = 1$), тогда элемент матрицы S^T транзитивной связи $s^T_{ki} = 1$. Т.о. при отличных от нуля значениях элементов s_{ji} и s_{kj} должна быть сформирована транзитивная связь и элемент s^T_{ki} должен быть определён равным 1.

Таблица истинности операции транзитивной конъюнкции

Таблица истинности операции «⊗»		
s_{ji}	s_{kj}	$s_{ki} = s_{ji} \otimes s_{kj}$
0	0	0
0	1	0
0	L	0
1	1	1
1	L	L
L_1	L_2	$L_1 \_ L_2$

Здесь L_1, L_2, L обозначают некоторые кортежи из логических связей $s_{ki}^{T\prime}$, где $s_{ki}^{T\prime}$ - логическая связь из вершины a_i в вершину a_k, либо ранее определенная транзитивная связь. Из приведённой формулировки понятно, что матрица S^T должна быть сформирована (может быть сформирована) за несколько проходов алгоритма. Использование операции транзитивной дизъюнкции позволяет учесть связь либо транзитивную, либо непосредственную между операторами.

Пример учёта необходимости альтернативных (транзитивной и непосредственной) связей приведён на Рис.7.

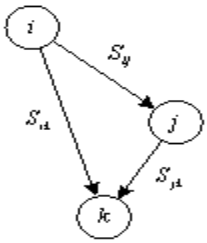


Рисунок 7 – Вид граф-схемы для определения транзитивных связей

Так как операция позволяет определять одну (какую-либо) из альтернативных связей (непосредственную либо транзитивную), важными параметрами для неё являются s_{ki} и s_{ki}^T. Таблица, приведенная ниже, соответствует операции транзитивной дизъюнкции, используемой при определении элементов s_{ki}^T матрицы S^T.

Таблица истинности операции транзитивной дизъюнкции

Таблица истинности операции «⊕»		
S_{ki}	S_{ki}^T	$S_{ki}^{T\prime} = S_{ki} \oplus S_{ki}^T$
0	0	0
0	1	1
0	L	L
1	L	L
1	1	1
L_1	L_2	$L_1 \_ L_2$

Тогда связь нового типа можно определить с использование введённых операций $\otimes$ и $\oplus$ по следующей формуле:

$$s_{ki}^{T'} = (s_{ji} \otimes s_{kj}) \oplus s_{ki},$$

или применительно к формируемой матрице транзитивности:

$$(k,i)^T = ((j,i) \otimes (k,j)) \oplus (k,i).$$

Если перед реализацией алгоритма формирования матрицы S^T положить, что $S^T = S$, тогда все «нисходящие» транзитивные связи вычисляются на основе формируемых «вышестоящих» связей.

Алгоритм формирования матрицы следования с транзитивными связями имеет следующий порядок шагов:

1) выполнить присваивание $S^T = S$;

2) в матрице S^T выполнить просмотр каждой *j*-ой строки ($j = \overline{1,m}$); если рассматривается *j*-я строка (определяется зависимость оператора a_j от оператора a_i), тогда просматриваются все *i*-ые столбцы ($i = \overline{1,m}$) этой строки;

3) если определен элемент $s_{ji}^T <> 0$, тогда в *j*-ом столбце матрицы S^T определяются *k*-е строки, такие, что $s_{kj}^T <> 0$; т.о. определяется индекс *k* элемента s_{ki}^T, значение которого будет вычисляться по приведённой формуле (определяется оператор a_k, который непосредственно зависит от оператора a_j и транзитивно зависит от оператора a_i);

4) для идентифицируемых таким образом элементов s_{ki}^T вычисляются значения по приведённой формуле;

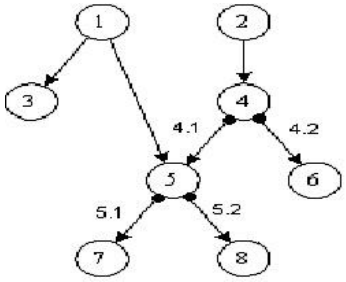
5) при изменении $i = \overline{1,m}$ осуществляются действия со всеми элементами *j*-ой строки;

6) выполняется изменение индекса строки *j:* $j=j+1$; если $j \le m$, тогда выполняется переход на шаг 2; если $j > m$, тогда переход на шаг 7;

7) останов алгоритма.

Реализация процедуры формирования матрицы транзитивного следования операторов

Вид информационно-логической граф-схемы и матрицы *S* непосредственного следования представлены на Рис.8 (Рис.8а и Рис. 8б соответственно).



а)

	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0
4	0	1	0	0	0	0	0	0
5	1	0	0	4.1	0	0	0	0
6	0	0	0	4.2	0	0	0	0
7	0	0	0	0	5.1	0	0	0
8	0	0	0	0	5.2	0	0	0

б)

Рисунок 8 – Вид граф-схемы алгоритма и соответствующей ей матрицы непосредственных связей

Порядок действий по построению матрицы транзитивного следования представлен ниже:

1) *j=3, i=1*, просмотр столбца 3 не позволил определить a_k, зависящего от a_3.

2) j=4, i=2, просмотр столбца *j=4*, идентифицируем $s^T_{5,4} <> 0$, следовательно *k=5*, вычисление значения $s^{T'}_{5,2}: s^{T'}_{5,2} = (s^T_{4,2} \otimes s^T_{5,4}) \oplus s^T_{5,2} = (1 \otimes 4.1) \oplus 0 = 4.1$; продолжаем просмотр столбца *j=4*, идентифицируем $s^T_{6,4} <> 0$, следовательно *k=6*, вычисление значения $s^{T'}_{6,2}: s^{T'}_{6,2} = (s^T_{4,2} \otimes s^T_{6,4}) \oplus s^T_{6,2} = (1 \otimes 4.2) \oplus 0 = 4.2$;

3) *j=5, i=1*, следовательно, просмотр столбца *j=5*, т.к. $s^T_{7,5} <> 0$, тогда *k=7* и вычисляется значение $s^{T'}_{7,1}$: $s^{T'}_{7,1} = (s^T_{5,1} \otimes s^T_{7,5}) \oplus s^T_{7,5} = [1 \otimes 5,1] \oplus 0 = 5.1$;

4) *j=5, i=4*, следовательно, просмотр столбца *j=5* и определение (при k=7) значения $s^{T'}_{7,4}$ $s^{T'}_{7,4} = (s^T_{5,4} \otimes s^T_{7,5}) \oplus s^T_{7,4} = (4.1 \otimes 5.1) \oplus 0 = 4.1\_5.1$;

5) по аналогии получаем: $s^{T'}_{8,1} = 5.2$; $s^{T'}_{8,4} = 4.1\_5.2$;

6) *j=6, i=4*, просмотр столбца *j=6*, т.к. он нулевой, тогда определить значение индекса *k* не представляется возможным; таким образом, после первого прохода алгоритма вид матрицы $S^{T'}$ следующий:

	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0
4	0	1	0	0	0	0	0	0
5	1	4.1	0	4.1	0	0	0	0
6	0	4.2	0	4.2	0	0	0	0
7	5.1	0	0	4.1 ,5.1	5.1	0	0	0
8	5.2	0	0	4.1 ,5.2	5.2	0	0	0

7) рассуждая по аналогии, могут быть сформированы значения элементов матрицы $s^{T'}_{7,2}$ и $s^{T'}_{8,2}$ (в результате реализации второго прохода); полученный вид матрицы $S^{T'}$ следующий:

1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0
4	0	1	0	0	0	0	0	0
5	1	4.1	0	4.1	0	0	0	0
6	0	4.2	0	4.2	0	0	0	0
7	5.1	4.1,5.1	0	4.1,5.1	5.1	0	0	0
8	5.2	4.1,5.2	0	4.1,5.2	5.2	0	0	0
	1	2	3	4	5	6	7	8

8) проходы алгоритма завершаются, если в результате их реализации ни один из элементов матрицы $S^{T'}$ не изменил своего значения.

Построение матрицы логической несовместимости операторов

Для того, чтобы в явном виде показать, какие операторы не могут выполняться параллельно, используются информационные связи на графе (информационные зависимости). Информационная связь прямо указывает на то, какие операторы не могут выполняться параллельно. Операторы, которые лежат в разных ветвях одного логического оператора не могут выполняться одновременно при однократной реализации алгоритма. Такие операторы являются логически несовместимыми операторами.

Информация о логической несовместимости операторов отражается в соответствующей матрицы логической несовместимости, которая обозначена через L. При этом различают матрицу непосредственной логической несовместимости L и матрицу транзитивных связей логической несовместимости операторов, обозначенную как L^T.

Формирование матриц L *и* L^T рассмотрим применительно к схеме графа алгоритма, представляющий собой модификацию предыдущей схемы, представленной на Рис.8. Граф-схема алгоритма изображена на Рис.9.

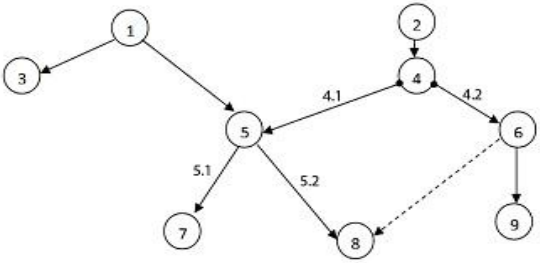


Рисунок 9 – Вид граф-схемы, используемой при формировании матрицы несовместности операторов

Вид матрицы транзитивных связей для рассматриваемой граф-схемы алгоритма (Рис.9) следующий:

$$S^T = \begin{array}{c|ccccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 1 & 4.1 & 0 & 4.1 & 0 & 0 & 0 & 0 & 0 \\ 6 & 0 & 4.2 & 0 & 4.2 & 0 & 0 & 0 & 0 & 0 \\ 7 & 5.1 & 4.1\_5.1 & 0 & 4.1\_5.1 & 5.1 & 0 & 0 & 0 & 0 \\ 8 & 5.2 & 4.1\_5.2 & 0 & 4.1\_5.2 & 5.2 & 0 & 0 & 0 & 0 \\ 9 & 0 & 4.2 & 0 & 4.2 & 0 & 1 & 0 & 0 & 0 \end{array}$$

Для построения матрицы *L* непосредственной логической несовместимости операторов в рассмотрение вводятся множества *Mi.n* – множество вершин операторов, входящих в *n-ю* ветвь *i-го* логического оператора. Соответственно, для ветви *k* вводится множество *Mi.k* . Сама вершина *i* ни в одно из этих множеств не входит.

Если вершины $p \in Mi.n$ и $q \in Mi.k$, а соответствующие им операторы могут выполняться либо один, либо другой при однократной реализации алгоритма, то операторы a_p и a_q являются логически несовместимыми. Тогда при планировании параллельных вычислений следует исключить планирование параллельного выполнения операторов, принадлежащих разным ветвям. Для формирования матрицы *L* непосредственной логической несовместимости введены и используются следующие обозначения: *S* – матрица непосредственных зависимостей; *RS* – размерность матрицы *S*; *MLO* – множество логических операторов*; RMLO* – мощность множества *MLO; Mi.n* – множество вершин, принадлежащих путям, включающим *n*-ю дугу *i-го* логического оператора; *RMi.n* – мощность множества вершин, принадлежащих *Mi.n.* Аналогичным образом интерпретирующих обозначения *Mi.k* и *RMi.k.*

При формировании матрицы *L* непосредственной несовместимости операторов используется матрица *S* непосредственных связей, то *RMi.n = RMi.k=1.* Обобщённая форма алгоритма построения матрицы *L,* предполагает реализацию следующего порядка шагов:

1) задание матрицы *S* и ее размерности;

2) определение по матрице *S* множества логических операторов *MLO* и его размерности (для этого используется процедура *PMLO,* рассматриваемая ниже);

3) реализация цикла по всем логическим операторам ($i = \overline{1,RMLO}$):

   а) определение множеств *Mi.n* и *Mi.k* для текущего *i*-го оператора;

   б) определение номеров вершин–операторов *p* и *q* таких, что $p \in Mi.n$,

$q \in Mi.k$; задание элементов матрицы *L* с индексами *[p,q]* и *[q,p]* равных 1: $L_{p,q} = 1$; $L_{q,p} = 1$.

При формировании множества логических операторов используется процедура *PMLO*, алгоритм которой приведён ниже. Признаком того, что *i*-ый оператор является логическим, является то, что в *j*-ом столбце ему соответствующей, должны быть определены значения элементов *i.n* или *i.k.*

Алгоритм формирования *MLO* имеет следующий порядок шагов:

1) *i=1* (*i-ый* столбец матрицы *S* соответствующий *i*-му оператору), *RMLO = 0*; $MLO = \varnothing$;

2) просмотр *i*-го столбца по строкам и определение элементов $s_{p,i}$ либо $s_{q,i}$ таких, что $s_{p,i} = i.n$; $s_{q,i} = i.k$;

3) если элемент $s_{p,i}$ либо $s_{q,i}$ такой, что $s_{p,i} = i.n$ либо $s_{q,i} = i.k$, найден, то $MLO = MLO \cup \{ i \}$; если столбец *i* не закончен, тогда переход на шаг 2;

4) модификация индекса оператора (столбца матрицы *i*): *i = i+1*, если $i \le RS$, тогда переход на шаг 2;

5) формирование значения *RMLO*.

Т.к. матрица *L* – матрица непосредственной логической несовместимости операторов, тогда множества *Mi.n* и *Mi.k* должны содержать по одному элементу (при этом используется матрица непосредственных зависимостей *S*). Для получения множеств *Mi.n* и *Mi.k* *i-го* логического оператора необходимо просматривать *i-ый* столбец и включать в множество *Mi.n* те номера операторов *p*, для которых в соответствующих *p-ых* строках *i*-го рассматриваемого столбца элементы равны *i.n*, в множество *Mi.k* включаются те номера операторов *q*, для которых в соответствующих *q*-ых строках рассматриваемого *i*-го столбца указано *i.k*. В соответствии с выполненными рассуждениями алгоритм формирования множеств *Mi.n* и *Mi.k* имеет следующий порядок шагов (выполняется формирование матрицы *L*):

1) задание индекса (номера) *s* рассматриваемого логического оператора из множества *MLO: s=1; RMLO = RMLO-1;*

2) определение элемента множества *MLO*, соответствующего индексу s*:* $i_s \in MLO$*;*

3) в i_s -ом столбце матрицы *S* просмотр *j*-ых строк, номер строки *j*, в которой $s_{j,i_s} = i_s.n$ либо $s_{j,i_s} = i_s.k$ буферизируется; при выполнении $s_{j,i_s} = i_s.n$ выполняется *p=j;* при выполнении $s_{j,i_s} = i_s.k$ выполняется *q=j* (инициализируются множества *Mi.n* и *Mi.k:* $Mi_s.n = Mi_s.n \cup \{ p \}$ *(при* $s_{j,i_s} = i_s.n$*);* $Mi_s.n = Mi_s.n \cup \{ q \}$ *(при* $s_{j,i_s} = i_s.k$*)*);

4) если просмотр i_s -го столбца не закончен, переход на шаг 3;

5) инициализация элементов матрицы *L:* $L_{p,q}=1$; $L_{q,p}=1$.

6) изменение номера элемента в *MLO: s=s+1; RMLO=RMLO-1;* если $RMLO \geq 0$, переход на шаг 2;

7) останов алгоритма.

В соответствии с рассмотренным алгоритмом на основе матрицы *S* матрица *L* получена в следующем виде:

$$L = \begin{array}{c|ccccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 8 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

На основе матрицы *L* должна быть сформирована матрица L^T транзитивной логической несовместимости операторов.

Для рассматриваемой схемы алгоритма имеем $4 \xrightarrow{4.1} 5$, $4 \xrightarrow{4.2} 6$, тогда операторы a_5 и a_6 являются логически несовместимыми. Т.к. $a_5 \rightarrow a_7$, $a_5 \rightarrow a_8$, то a_7 и a_8 также являются логически несовместимыми, однако $a_4 - \rightarrow a_8$, $a_4 - \rightarrow a_9$, тогда a_8 и a_9 также являются логически несовместимыми. Т.о. если оператор a_i несовместим с оператором a_j и существует связь $a_i \rightarrow a_k$, то оператор a_k несовместим с оператором a_j только в том случае, если отсутствует связь $a_j \rightarrow a_k$ (либо $a_j - \rightarrow a_k$).

Для каждого оператора a_k необходимо выделить множество $C_k = \{ a_i,...,a_r \}$ операторов, которые образуют связи $a_h - \rightarrow a_k$ ($h \in \{ i,..,r \}$) (входной оператор можно не рассматривать, т.к. у него нет логически несовместимых элементов). По матрице *L* (непосредственная логическая несовместимость) для каждого a_h формируется множество B_h логически несовместимых с ним операторов.

После того, как все множества B_h построены, определяется множество A_k операторов, несовместимых с оператором a_k следующим образом:

$$A_k = \begin{cases} B_i, если \quad r = 1; \\ B_i \cap B_{i'} \cap ... \cap B_r, если \quad r > 1. \end{cases}$$

Рассмотрим реализацию способа формирования матрицы L^T. Для оператора a_7 : $C_7 = \{a_2, a_4, a_5\}$; $B_2 = \varnothing$; $B_4 = \varnothing$; $B_5 = \{a_6\}$; $A_{a_7} = B_2 \cap B_4 \cap B_5 = \{a_6\}$, следовательно, $L^T_{7,6} = 1$ и тогда $L^T_{6,7} = 1$. Для оператора a_8 : $C_8 = \{a_2, a_4, a_5\}$; $B_2 = \varnothing$; $B_4 = \varnothing$; $B_5 = \{a_6\}$; $A_{a_8} = B_2 \cap B_4 \cap B_5 = \{a_6\}$, следовательно, $L^T_{8,6} = 1$ и тогда $L^T_{6,8} = 1$. Для оператора a_9 : $C_7 = \{a_2, a_4, a_6\}$; $B_2 = \varnothing$; $B_4 = \varnothing$; $B_6 = \{a_5\}$; $A_{a_9} = B_2 \cap B_4 \cap B_6 = \{a_5\}$, следовательно, $L^T_{9,5} = 1$ и тогда $L^T_{5,9} = 1$.

В промежуточном виде матрица L^T примет следующий вид:

$$L^T = \begin{array}{c|ccccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 6 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 7 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 8 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 9 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{array}$$

Повторная реализация приведенного алгоритма позволила на основе полученной матрицы L^T сформированы дополнительно следующие единичные значения ее элементов; $L^T_{7,9} = 1$ и $L^T_{9,7} = 1$; $L^T_{8,9} = 1$ и $L^T_{9,8} = 1$. Тогда итоговый вид матрицы транзитивной логической несовместимости операторов следующий:

$$L^T = \begin{array}{c|ccccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 6 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 7 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 8 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 9 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{array}.$$

Таким образом, проверка условия формирования транзитивной матрицы несовместимости операторов выполняется до тех пор, пока

полученный на следующем шаге вид этой матрицы не будет отличаться от вида этой матрицы с предыдущего шага.

После формирования матрицы несовместности операторов выполняется построение матрицы независимости операторов программы.

Формирование матрицы независимости *M* операторов программы

Для формирования матрицы M первоначально должны быть выполнены преобразования матрицы S^T:

1) выполнить транспонирование матрицы S^T (будет сформирована матрица $S^{T'}$);

2) выполнить сложение исходной матрицы S^T и её транспонированного варианта $S^{T'}$.

Замечание. Перед выполнением приведённых операций необходимо все элементы матрицы S^T, не равные 0 либо не равные 1, заменить на 1.

Полученная таким образом матрица S' совместно с матрицей L^T используются при определении матрицы независимости M следующим образом: $M = S' \cup L^T$, $\cup$ –операция дизъюнкции.

Для рассматриваемого вида граф-схемы алгоритма итоговая матрица независимости операторов является следующей:

$$M = \begin{array}{c|ccccccccc} 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 2 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 5 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 6 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 7 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 8 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 9 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \end{array}$$

После получения матрицы независимости для операторов программы формируется множества взаимно независимых операторов программы (множества ВНО). Сформулируем алгоритм построения множеств взаимно независимых операторов (ВНО) применительно в матрице независимости полученного вида (для рассматриваемого вида граф-схемы алгоритма). Последовательность действий по формированию множеств ВНО следующая:
1) записываем в стек (*i=1*)-ю строку матрицы M; выполняем проверку строки в стеке на наличие нулей в позициях *j>1* (*j=i+1*), определено наличие 0 в позиции *j=2*; это показывает, что операторы a_1 и a_2 независимы; выполняем логическое сложение строк 1 и 2, формируем строку s_1 ($s_1 = "1" \vee "2"$);

2) формируем новый вид стека, размещая в нем полученную строку s_1 (Рис.10а); строки 1 и s_1 содержат 0 только в позициях, соответствующих операторам a_1 и a_2 , поэтому полное множество ВНО имеет вид: $W_1 = \{a_1, a_2\}$;

3) удаляем стоку s_1 из стека, продолжаем просмотр строки 1 с позиции $j>2$ (предыдущая рассматриваемая позиция $j=2$), определяем наличие 0 в позиции $j=4$, формируем новую стоку в стеке, представляющую собой дизъюнкцию строк 1 и 4 ($s_2 = "1" \vee "4"$), полученный вид стека представлен на Рис.10б), рассуждая по аналогии, получаем $W_2 = \{a_1, a_4\}$;

4) удаляем строку s_2 из стека, продолжаем просмотр строки 1 с позиции $j>4$, определяем наличие 0 в позиции $j=6$, формируем строку $s_3 = "1" \vee "6"$, размещаем ее в стеке, вид стека представлен на Рис.10в); строка 1 и строка s_3 содержат 0 в позиции $j=6$, поэтому операторы a_1 и a_6 могут выполняться параллельно; тогда множество ВНО имеет вид: $W_3 = \{a_1, a_6\}$;

5) удаляем строку s_3 из стека, идентифицируем наличие 0 в строке 1 в позиции $j=9$, формируем строку $s_4 = "1" \vee "9"$; размещаем ее в стеке; получаем вид стека, представленный на Рис. 10г); т.к. в позициях $j=9$ строк 1 и s_4 фиксируется наличие 0, тогда операторы a_1 и a_9 могут выполняться параллельно, а множество ВНО имеет вид: $W_4 = \{a_1, a_9\}$;

s_1	0	0	1	1	1	1	1	1	1
1	0	0	1	0	1	0	1	1	0

а)

s_2	0	1	1	0	1	1	1	1	1
1	0	0	1	0	1	0	1	1	0

б)

s_3	0	1	1	1	1	0	1	1	1
1	0	0	1	0	1	0	1	1	0

в)

s_4	0	1	1	1	1	1	1	1	0
1	0	0	1	0	1	0	1	1	0

г)

s_5	1	0	0	1	1	1	1	1	0
2	0	0	0	1	1	1	1	1	1

д)

s_6	1	1	0	0	1	1	1	1	1
3	1	0	0	0	0	0	0	0	0

е)

s_7	1	1	0	1	0	1	1	1	1
3	1	0	0	0	0	0	0	0	0

ж)

Рисунок 10– Формирование множеств взаимно независимых операторов

6) т.к. строка 1 просмотрена полностью, тогда вместе со строкой s_4 из стека удаляется строка 1, в стеке размещается строка 2;

7) выполняем просмотр строки 2 с позиции *j>2*; идентифицируем наличие 0 в позиции *j=3*, формируем строку $s_5 = "2" \vee "3"$; размещаем ее в стеке; получаем вид стека, представленный на Рис. 10д); т.к. в позициях *j=3* строк 2 и s_5 фиксируется наличие 0, тогда операторы a_2 и a_3 могут выполняться параллельно, а множество ВНО имеет вид: $W_5 = \{a_2, a_3\}$;

8) в строке 2 отсутствуют 0 в позициях *j>3*, поэтому строки s_5 и 2 удаляются из стека, в котором размещается строка 3; т.к. в строке 3 в позиции *j=4* фиксируем 0, тогда формируем строку s_6 в виде $s_6 = "3" \vee "4"$; получаем вид стека, представленный на Рис.10е); на основе полученного вида стека формируем множество ВНО в виде: $W_6 = \{a_3, a_4\}$;

9) формируем новый вид стека путем удаления из него строки s_6 и добавления в него строки s_7 вида $s_7 = "3" \vee "5"$; получаем стек в виде, представленном на Рис. 10ж); в позициях *j=5* в строках 3 и s_7 фиксируется наличие 0, поэтому операторы a_3 и a_5 могут выполняться параллельно, а множество ВНО получено в следующем виде: $W_7 = \{a_3, a_5\}$;

10) выполняя аналогичные рассуждения, получим для оператора a_3 и операторов a_6 , a_7 , a_8 , a_9 множества ВНО в следующем виде: $W_8 = \{a_3, a_6\}$; $W_9 = \{a_3, a_7\}$; $W_{10} = \{a_3, a_8\}$; $W_{11} = \{a_3, a_9\}$; при этом видно, что оператор a_3 может выполняться параллельно либо с a_6 , либо с a_7 , либо с a_8 , либо с a_9 (понятно, что множество ВНО вида $W = \{a_6, a_9\}$ построено быть не может, тогда операторы a_6 и a_9 параллельно выполняться не могут);

11) удаляем строку 3 из стека, размещаем в нем строку 4; т.к. в позициях *j>4* нули не обнаружены, то параллельность оператора a_4 с последующими операторами не исследуется;

12) последовательное размещение в стеке строк 5, 6, 7, 8, 9 и их анализ позволил определить отсутствие множеств ВНО для операторов a_5, a_6, a_7, a_8, a_9.

Таким образом, множества ВНО получены в следующем итоговом виде: $W_1 = \{a_1, a_2\}$; $W_2 = \{a_1, a_4\}$; $W_3 = \{a_1, a_6\}$; $W_4 = \{a_1, a_9\}$; $W_5 = \{a_2, a_3\}$; $W_6 = \{a_3, a_4\}$; $W_7 = \{a_3, a_5\}$; $W_8 = \{a_3, a_6\}$; $W_9 = \{a_3, a_7\}$; $W_{10} = \{a_3, a_8\}$; $W_{11} = \{a_3, a_9\}$. На основе этих множеств требуется сформировать ярусно-параллельную форму программы.

Особенности функционирования приведенного алгоритма рассмотрим на примере граф-схемы алгоритма, изображенной на Рис. 11 и его матрицы независимости, представленной на Рис.12.

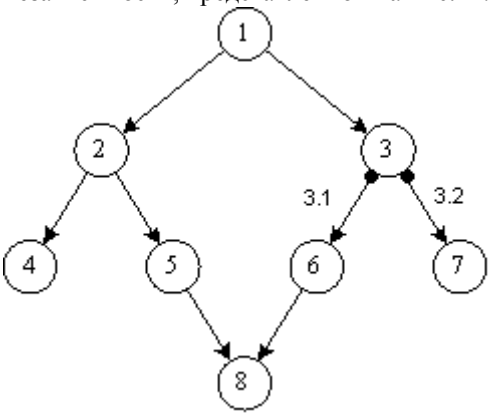


Рисунок 11 – Граф-схема G алгоритма с информационно- логическими связями

1		1	1	1	1	1	1
2	1			1	1			1
3	1					1	1	1
4	1	1						
5	1	1						1
6	1		1				1	1
7	1		1			1		
8	1	1	1		1	1		
	1	2	3	4	5	6	7	8

Рисунок 12 – Матрица независимости *M* для графа G

Записываем в стек первую строку *(i=1)* матрицы *M* (Рис. 12а). Начинаем просматривать эту строку на наличие нулей, начиная со второй позиции *(i +1).* Выясняем, что в первой строке нет нулей в старших позициях. Это означает, что первое найденное нами полное множество ВНО будет состоять из единственного оператора: *{1}.* Исключаем первую строку матрицы *M* из стека и заносим вторую строку (Рис. 12б). Осуществляем просмотр этой строки, начиная с третьей позиции. Находим первый нулевой элемент, который стоит в позиции 3. Этот нуль указывает, что операторы 2 и 3 взаимно независимы (могут выполняться параллельно). Складываем логически строки, соответствующие операторам 2 и 3.

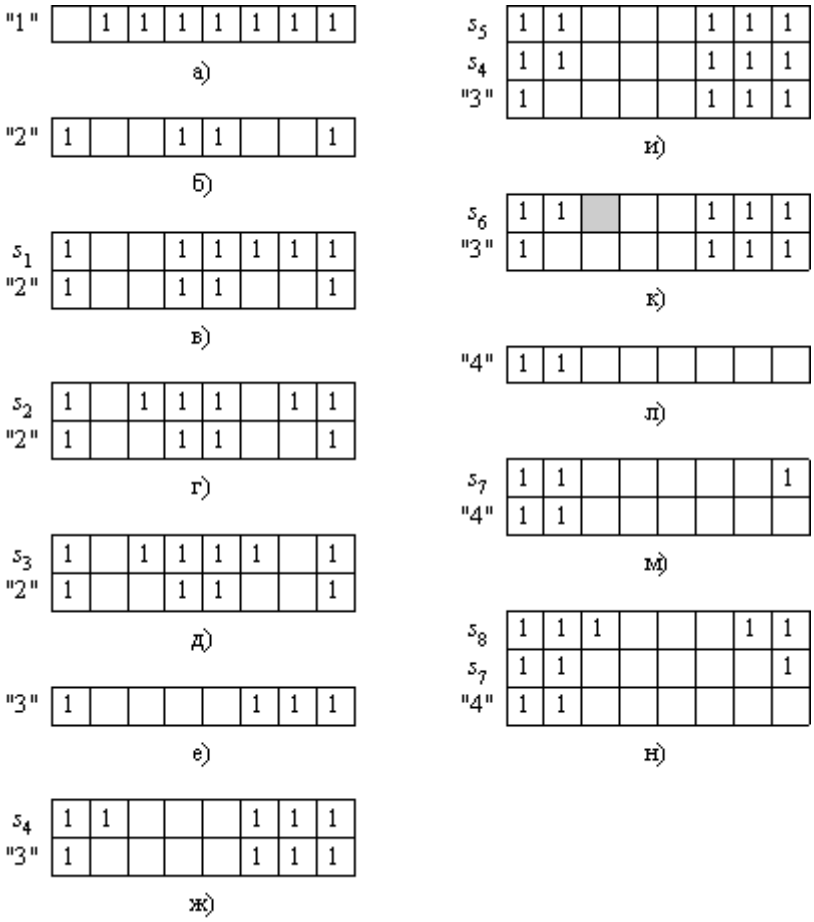


Рисунок 12 – Пример работы алгоритма нахождения полных множеств ВНО. Начало

Получаем новую строку $s_1 = "2" \vee "3"$ в вершине стека и весь стек в виде, представленном на Рис. 12в). Строка в вершине стека содержит нули только в тех позициях, которые соответствуют образующим её операторам 2 и 3. Это означает, что найдено второе полное множество ВНО *{2,3}*. Убираем из стека строку s_1 и начинаем просматривать строку 2 со следующего места после остановки (когда стали формировать строку s_1), т.е. с позиции 4. Находим следующий нуль, который оказывается в позиции 6. Формируем новую вершину стека, складывая логически строки 2 и 6 (Рис. 12г). Новая строка содержит нули только в позициях 2 и 6. Значит, найдено ещё одно

полное множество ВНО *{2, 6}*. Исключаем строку s_2 из стека и во второй строке находим следующий нуль, находящийся в позиции 7. Формируем новую вершину стека – строку $s_3 = "2" \vee "7"$ (Рис. 12д). В этой строке нули так же соответствуют только операторам, «участвующим» в её формировании. Значит, найдено четвёртое полное множество ВНО *{2,7}*. Исключаем строку s_3 из стека. Все нули строки 2 найдены, поэтому исключаем и её из стека. Помещаем в стек третью строку матрицы независимости *M* (Рис. 12е). Находим первый нуль в строке 3 правее третьей позиции. Этот нуль стоит в позиции 4. Складываем логически строки 3 и 4. Формируем новую строку в вершине стека и весь стек в виде, представленном на Рис. 12ж). Строка s_4 содержит нули не только в позициях 3 и 4. Первый такой нуль, правее позиции 4, находится в позиции 5. Формируем новую вершину стека $s_5 = s_4 \vee "5"$. Стек принимает вид, показанный на Рис. 12и). Строк s_5 содержит нули только в позициях 3, 4, 5. Это значит, что найдено очередное полное множество ВНО *{3, 4, 5}*. Исключаем строку s_5 из стека. Строка s_4 также исчерпана, поэтому исключаем и её. Просматриваем дальше строку 3. Следующий нуль будет найден в позиции 5. Формируем новую вершину стека, складывая логически строки 3 и 5 (Рис. 12к). Проверяем полученную строку на наличие нулей правее позиции 5. Таких нулей нет. Казалось бы, можно формировать очередное полное множество ВНО *{3, 5}*, однако проверим множество *{3, 5}* на полноту. Для этого достаточно проверить строку s_6 на наличие «дополнительных» нулей (т.е. нулей, позиции которых не совпадают с номерами из найденного множества ВНО), левее позиции 5. В строке s_6 есть такой «дополнительный» нуль – в позиции 4. Это означает, что множество ВНО *{3, 5}* не является полным и его необходимо отбросить.

Следует обратить внимание на то, что проверку на полноту необходимо проводить лишь после получения «полного» множества ВНО, т.е. когда все нули правее последнего оператора из множества ВНО найдены. Если такую проверку проводить не после получения множества ВНО, а во время, то будут полностью потеряны все возможные полные множества ВНО, которые могли бы получиться в результате дальнейших действий с данной строкой. Исключаем строку s_6 из стека. Дальнейший просмотр строки 3 не выявил новых нулей, поэтому исключаем и её из стека. Поскольку стек пуст, то помещаем туда следующую строку – строку 4 (Рис. 12л).

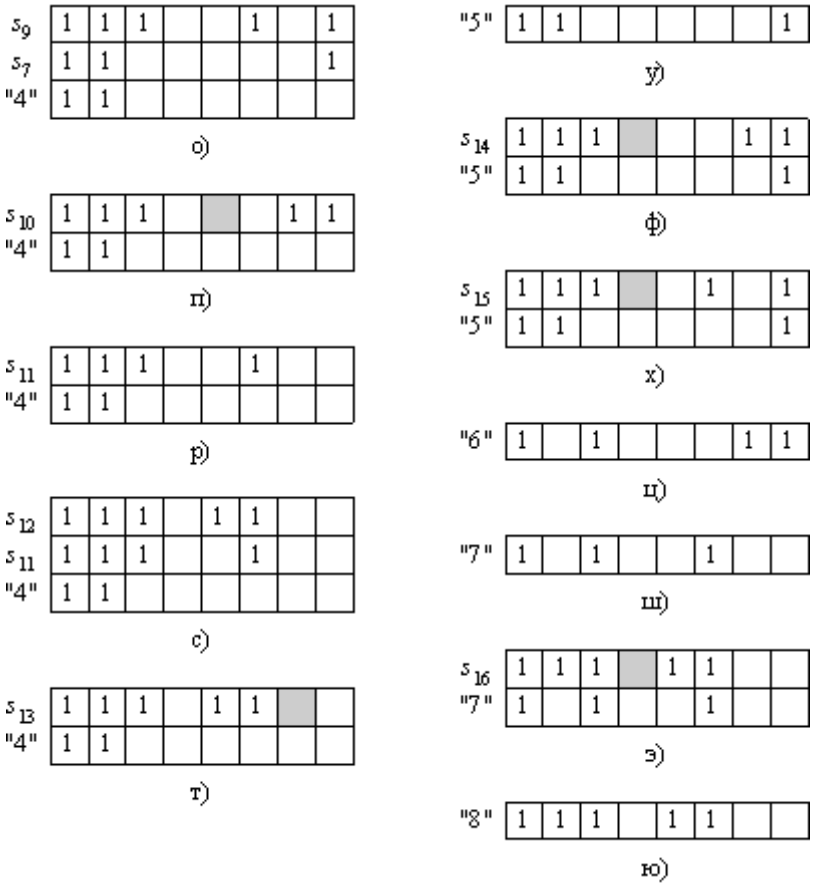


Рисунок 12– Пример работы алгоритма нахождения полных множеств ВНО. Окончание

Начинаем просмотр строки 4 с позиции 5. Первый нуль как раз находится в позиции 5. Складываем логически строки 4 и 5 (Рис. 12м). В получившейся строке s_7 имеются нули правее позиции 5. Найденный нуль занимает позицию 6. Формируем новую строку $s_8 = s_7 \vee "6"$. Тогда стек будет иметь вид, изображённый на Рис. 12н). Строка в вершине стека содержит нули только в позициях, соответствующих операторам, «участвующим» в её формировании. Значит, найдено ещё одно (шестое по счёту) полное множество ВНО *{4, 5, 6}*.

Убираем из стека строку s_8. Следующий найденный нуль в строке s_7 находится в позиции 7. Складываем соответствующие строки (s_7 и 7).

Получаем новую строку в вершине стека и весь стек в виде, показанном на Рис. 12о). Строка s_9 содержит нули только в позициях 4, 5 и 7. Это значит, что найдено очередное полное множество ВНО *{4, 5, 7}*. Выгружаем строку s_9 из стека. Строка s_7 так же исчерпана, поэтому исключаем и её. Следующий найденный нуль в строке 4 занимает позицию 6. Формируем новую вершину стека, складывая строки 4 и 6 (Рис. 12п). Получившаяся строка не содержит нулей правее позиции 6. Однако экзамен на полноту тоже не выдерживает: в позиции 5 появился «дополнительный нуль». Это значит, что найденное множество ВНО *{4, 6}* не является полным (т.е. комбинация операторов 4 и 6 уже встречалась с какими-то другими операторами, образующими полное множество ВНО) и его необходимо отбросить. Исключаем строку s_{10} из стека. Следующий найденный нуль в строке 4 находится в позиции 7. Формируем новую строку $s_{11} = "4" \vee "7"$ и помещаем её в вершину стека (Рис. 12р). Ищем нули во вновь образованной строке правее позиции 7. Такой нуль есть – он находится в позиции 8. Формируем строку $s_{12} = s_{11} \vee "8"$ и помещаем её в вершину стека. Стек будет выглядеть так, как показано на Рис. 12с. Строка s_{12} содержит нули только в позициях 4, 7 и 8. Это означает, что найдено восьмое полное множество ВНО *{4, 7, 8}*. Убираем строку s_{12} из стека. Строка s_{11} так же исчерпана, поэтому исключаем её из стека. В стеке остаётся единственная 4-ая строка. Последний найденный нуль находится в позиции 8. Формируем новую вершину стека, складывая строки 4 и 8 (Рис. 12т). В получившейся строке обнаруживаем «дополнительный» нуль левее 8-ой позиции, но не совпадающий с позицией 4. Это значит, что найденное множество ВНО *{4, 8}* не является полным и его необходимо исключить из рассмотрения. Исключаем строку s_{13} из стека. Строка 4 пройдена до конца, следовательно, её так же нужно исключить из стека. Заносим в стек следующую – 5-ую строку матрицы независимости *M* (Рис. 12у). Строка 5 содержит нуль в позиции 6. Формируем новую вершину стека $s_{11} = "4" \vee "7"$. Стек будет выглядеть так, как представлено на Рис. 12ф). В строке s_{14} нет нулевых элементов, правее позиции 6. Проверяем найденное множество ВНО на полноту. Выясняем, что в позиции 4 строки s_{14} есть «дополнительный» нуль – найденное множество ВНО не полное, поэтому отбрасываем его.

Выгружаем строку s_{14} из стека. Следующий нуль в строке 5 занимает позицию 7. Складываем логически строки 5 и 7 и формируем новую вершину стека – строку s_{15} (Рис. 12х). Анализируя строку s_{15}, замечаем, что эта строка

содержит «дополнительный» нуль левее позиции 7, который не совпадает с позицией 5. Делаем вывод, что множество ВНО *{5, 7}* не является полным и отбрасываем его. Исключаем строку s_{15} из стека. Строка 5 оказывается пройдена до конца, поэтому исключаем её. Помещаем в стек строку 6 (Рис. 12ц). В этой строке нет нулевых элементов правее позиции 6. Возникает ситуация, похожая на ту, что складывалась на шаге 1. Проверим, является ли множество ВНО *{6}* полным. В отличие от шага 1, данное множество не является полным т.к. в строке 6 существуют другие нули, помимо позиции 6. Отбрасываем это множество. Убираем строку 6 из стека и загружаем туда строку 7 (Рис. 12ш). Строка 7 содержит нуль в позиции 8, следовательно, формируем новую строку $s_{16} = "7" \vee "8"$ и помещаем её в вершину стека (Рис. 12э). Обнаруживаем «дополнительный» нуль в строке S_{15} в позиции 4. Это говорит о том, что множество ВНО *{7, 8}* не является полным и его необходимо отбросить. Выгружаем из стека строку s_{16}. Все комбинации операторов с участием оператора 7 исчерпаны. Исключаем строку 7 из стека. Загружаем последнюю 8-ю строку в стек (Рис. 12ю). В восьмой строке (как и в шестой) имеются нули левее позиции 8. Это значит, что множество ВНО *{8}* не является полным и его необходимо отбросить.

Таким образом, найдено 8 полных множеств ВНО: *{1}, {2,3}, {2,6}, {2,7}, {3,4,5}, {4,5,6}, {4,5,7}, {4,7,8}*, на основании которых выполняется формирование ярусно- параллельной формы программы. На основе множеств ВНО может быть сформирована ярусно-параллельная форма программы.

3. Программа выполнения работы

В соответствии с заданием должно быть реализовано построение ярусно-параллельной формы программы, выполняющей назначенные задания. Для выполнения задания лабораторной работы необходимо:

- на основе заданного выражения сформировать граф-схему алгоритма, на которой должны быть указаны информационные операторы и логические операторы с указанием исходящих из них управляющих дуг;
- по граф-схеме алгоритма сформировать матрицу непосредственных связей, в которой указать информационные зависимости, а также логические зависимости в виде *i.n* (где *i*- идентификатор оператора, *n*- идентификатор выходной управляющей ветви);
- в соответствии с приведенным в теоретическом введении алгоритмом аналитически сформировать матрицу транзитивных связей (информационных и логических) между операторами программы (на основе матрицы непосредственных связей сформировать матрицу транзитивных связей между операторами программы);
- в соответствии с приведенном в теоретическом введении алгоритмом аналитически сформировать матрицу непосредственной логической несовместимости операторов программы;

- в соответствии с алгоритмом аналитически сформировать транзитивную матрицу несовместимости операторов;
- в соответствии с алгоритмом аналитически сформировать множества взаимно независимых операторов и на их основе построить ярусно-параллельную форму программы;
- разработать процедуру, формирующую на основе матрицы непосредственных связей между вершинами графа (операторами программы) матрицу транзитивных информационно-логических связей программы;
- разработать процедуру, формирующую на основе матрицы непосредственных связей между вершинами графа матрицу несовместимости операторов (реализовать процедуры формирования множества MLO и матрицы непосредственной логической несовместимости операторов);
- разработать процедуру построения транзитивной матрицы логической несовместимости операторов программы;
- разработать процедуру формирования множеств ВНО для операторов программы, сформировать на основе множеств ВНО ярусно- параллельную форму программы;
- выполнить формирование отчета следующего содержания:
  - полная формулировка задания на работу;
  - схема информационно- логических связей между операторами программы;
  - полученные виды матриц непосредственных связей, транзитивных связей, непосредственной и транзитивной логической несовместимости операторов;
  - полученные виды множеств ВНО операторов программы;
  - полученный вид ЯПФ программы;
  - выводы по результатам выполнения работы.

4. Задание на работу

Осуществить формирование транзитивных замыканий образов и прообразов вершин, формирование ЯПФ и реализацию параллельной программы для управляющего графа, сформированного для последовательной программы, вычисляющей задаваемые в вариантах выражения.

Вариант 1. Z=B/(C+D)-K*(E+A), где

$$D=\begin{cases}\cos A, A<0;\\ \ln A, A>0;\\ 0, A=0.\end{cases} \qquad E=\begin{cases}B^2+D^2, B+D<0;\\ \sqrt{B+D}, B+D>0;\\ B^2-2D^2, B+D=0.\end{cases}$$

Вариант 2. L=(X+K)2/M-(J-K)/Z, где

$$K=\begin{cases}2X+Y, X>3;\\ 2Y+X, X<3;\\ X*Y, X=3.\end{cases} \qquad M=\begin{cases}\ln(K+5)+Z; K>-5;\\ |K+5|, K<-5;\\ Z-K, K=-5.\end{cases}$$

Вариант 3. G=(2D+3A)/(B+C)-E*(A+5B), где

$$D=\begin{cases}\sqrt{5B^2+1}, B>0;\\ |B+5|, B<0;\\ 0, B=0.\end{cases} \qquad E=\begin{cases}5D*C, D*C>0;\\ \cos DC, D*C<0;\\ D+5C, D*C=0.\end{cases}$$

Вариант 4. W=5(M+O2)-(3L+4K)*N/2, где

$$O=\begin{cases}L+M^2, L>M;\\ M^2-L, L<M;\\ 5L, L=M.\end{cases} \qquad K=\begin{cases}O+2N, O>5;\\ 2N-O, O<5;\\ (N+O)^2, O=5.\end{cases}$$

Вариант 5. N=(A+2B)/D-3E+A*(F+D), где

$$A=\begin{cases}E^2+2F, E+F>2;\\ \sqrt{(E+F)^2+1}, E+F<2;\\ 5E, E+F=2.\end{cases} \qquad B=\begin{cases}A+D^2, A>0;\\ |A|+\cos D, A<0;\\ 10, A=0.\end{cases}$$

Вариант 6. A=(N2+3F)/2-(G+N)*(W-2Z), где

$$N=\begin{cases}G+\cos W, W>0;\\ G-\sin W, W<0;\\ G, W=0.\end{cases} \qquad F=\begin{cases}\sin(G+Z), G+Z<0;\\ \ln(G+2Z), G+Z>0;\\ \cos(2Z-G), G+Z.\end{cases}$$

Вариант 7. X=cos(I+2K)*sin(V-4L)+J/(L+4K), где

$$V=\begin{cases}\sqrt{L^2+1}, L<0;\\ \sqrt{L+10}, L>0;\\ 5, L=0.\end{cases} \qquad J=\begin{cases}10K, I=K;\\ 10I-K, I>K;\\ (I+K)^2, I<K.\end{cases}$$

Вариант 8. I=(2J+cos$C$)*5D-ln$M$*(N+2C), где

$$C=\begin{cases} M*J, M>J; \\ M/J, M<J; \\ 0, M=J. \end{cases} \qquad D=\begin{cases} \ln(N-10), N>10; \\ \left|N^2-10\right|, N<10; \\ \cos N, N=10. \end{cases}$$

Вариант 9. *K*=(*X*+5*L*)/(*M*+*Z*)-2*J*+*L***M*, где

$$L=\begin{cases} \ln(X^2+J^2), X>J; \\ J^2-X^2, X<J; \\ 100, X=J. \end{cases} \qquad M=\begin{cases} Z^2+10, Z<0; \\ 10-Z^2, Z>0; \\ 10, Z=0. \end{cases}$$

Вариант 10. *F*=|2*B*-*D*|**E*-(*A*2+*E*2)/*C*, где

$$D=\begin{cases} A^2+\cos A, A>0; \\ \sin(A^2+10), A<0; \\ 10, A=0. \end{cases} \qquad E=\begin{cases} 2B/C, B>C; \\ \sqrt{\left|B+2C\right|}, B<C; \\ 0, B=C. \end{cases}$$

5. Контрольные вопросы

1. Какие виды вершин указываются на граф-схеме алгоритма?
2. Какие виды дуг указываются на граф-схеме алгоритма?
3. Каким образом реализуется формирование матрицы непосредственных связей для операторов программы?
4. Какой вид имеет выражение, используемое при формировании матрицы транзитивных связей между операторами программы?
5. Каким образом используется выражение для определения значений элементов матрицы транзитивных связей операторов программы? Прокомментировать алгоритм использования выражения.
6. Каким образом формализуется алгоритм определения множества логических операторов программы?
7. Каким образом формулируется алгоритм формирования матрицы непосредственной логической несовместимости операторов программы?
8. Каким образом формализуется способ построения транзитивной матрицы логической несовместимости операторов программы?
9. В чем состоит алгоритм формирования множеств взаимно независимых операторов программы?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Немнюгин С.А. Параллельное программирование для многопроцессорных вычислительных систем./ С.А. Немнюгин, О.А. Стесик – СПБ.: Изд-во ” BHV”,2002. –400 с.
2. Эндрюс Г.Р. Основы многопоточного параллельного и распределенного программирования. / Г.Р. Эндрюс – М.: Издательство “Вильямс”, 2003. – 505 с.
3. Воеводин В.В. Параллельные вычисления. / В.В. Воеводин.– СПБ.: Изд-во ” BHV”,2002.– 599 с.
4. Демьянович Ю.К. Технология программирования для распределенных параллельных систем./ Ю.К. Демьянович, О.Н. Иванцова. – СПБ.: Изд-во СПГУ, 2005.– 90 с.
5. Шпаковский Г.И. Программирование для многопроцессорных систем в стандарте MPI./ Г.И. Шпаковский, Н.В.Серикова. – Минск: Изд-во БГУ, 2002.– 323 с.

Заказ № ___от ____ ____2015 Тираж ____экз.
Изд-во СевГУ