

Оптимизации для кодогенератора

Планирование инструкций

Синявин Анатолий

Планирование инструкций базового блока

Планирование инструкций базового блока

Оптимальное решение этой задачи NP-полное

Вместо точного решения часто используется
простой алгоритм list scheduling

Планирование инструкций базового блока

При планировании инструкций следует учитывать два типа ограничений:

- ограничения по данным
- ограничения по ресурсам процессора

Планирование инструкций базового блока

Граф зависимости по данным $G = \langle N, E \rangle$

N – множество инструкций

E – множество дуг-зависимостей

Особенности:

- Каждая инструкция n содержит таблицу резервирования RT_n
- Каждая дуга e помечена задержкой de

Планирование инструкций базового блока

Инструкция n

Таблица резервирования RT_n , K – количество видов ресурсов

Номер такта	Ресурс 1	Ресурс 2	Ресурс 3	Ресурс 4	...	Ресурс K
1	1		2			
2		1				
3						3

**int [,] RT_n =
new int [3, K];**

Кол-во ресурсов каждого вида R_{cnt}

Ресурс 1	Ресурс 2	Ресурс 3	Ресурс 4	...	Ресурс K
r1	r2	r3	r4	...	rK

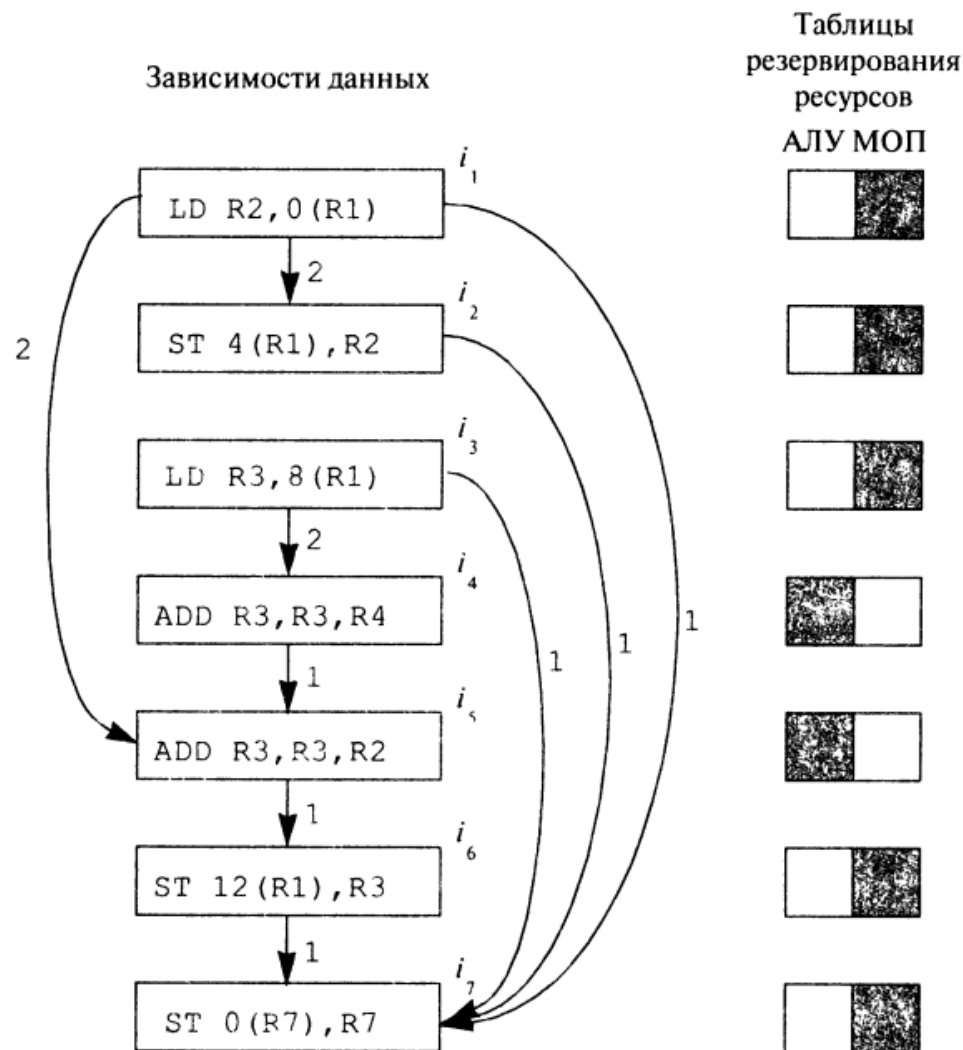
int [] R_{cnt} = new int [K];

Планирование инструкций базового блока

Инструкции $n1$ и $n2$, дуга $e: n1 \rightarrow n2$ с задержкой de

$n2$ не может выполняться раньше чем через de тактов
после $n1$

Планирование инструкций базового блока



Планирование инструкций базового блока

Вход:

- базовый блок v
- готовый граф зависимостей
- таблицы резервирования для каждой инструкции

Выход: план S

Метод:

```
map<stmt, int> S;
```

```
int[, ] RT = new int[N,K]; /* K – количество видов ресурсов, инициализируем RT нулями */
```

```
foreach ( stmt n in “приоритетном топологическом порядке” )
```

```
{
```

```
    /* самое раннее время запуска инструкции n с учётом графа зависимостей */
```

```
    int s = " $\max_{(e=p \rightarrow n) \in E} (S(p) + de)$ "; // если предшествующих инструкций нет, то s=0
```

```
    /* учёт таблицы резервирования ресурсов для инструкции n */
```

```
    while( " $\exists i, k \mid RT[s+i,k] + RTn[i,k] > Rcnt[k]$ " ) s++;
```

```
    S[n] = s;
```

```
    for ( “всех i,k” )
```

```
        RT[s+i,k]=RT[s+i] + RTn[i,k];
```

```
}
```

```
return S;
```

Планирование инструкций базового блока

Приоритетный топологический порядок.

Рекомендации по выбору #1:

- функция приоритета для инструкции n

$F(n) = \{ \text{длина самого длинного пути до узла } n \text{ в графе зависимостей} \}$

Узлы с большим приоритетом планировать раньше.

Т.о., длина плана приближается к критическому пути.

Планирование инструкций базового блока

Приоритетный топологический порядок.

Рекомендация по выбору #2:

- Степень использования

u_k / r_k , где

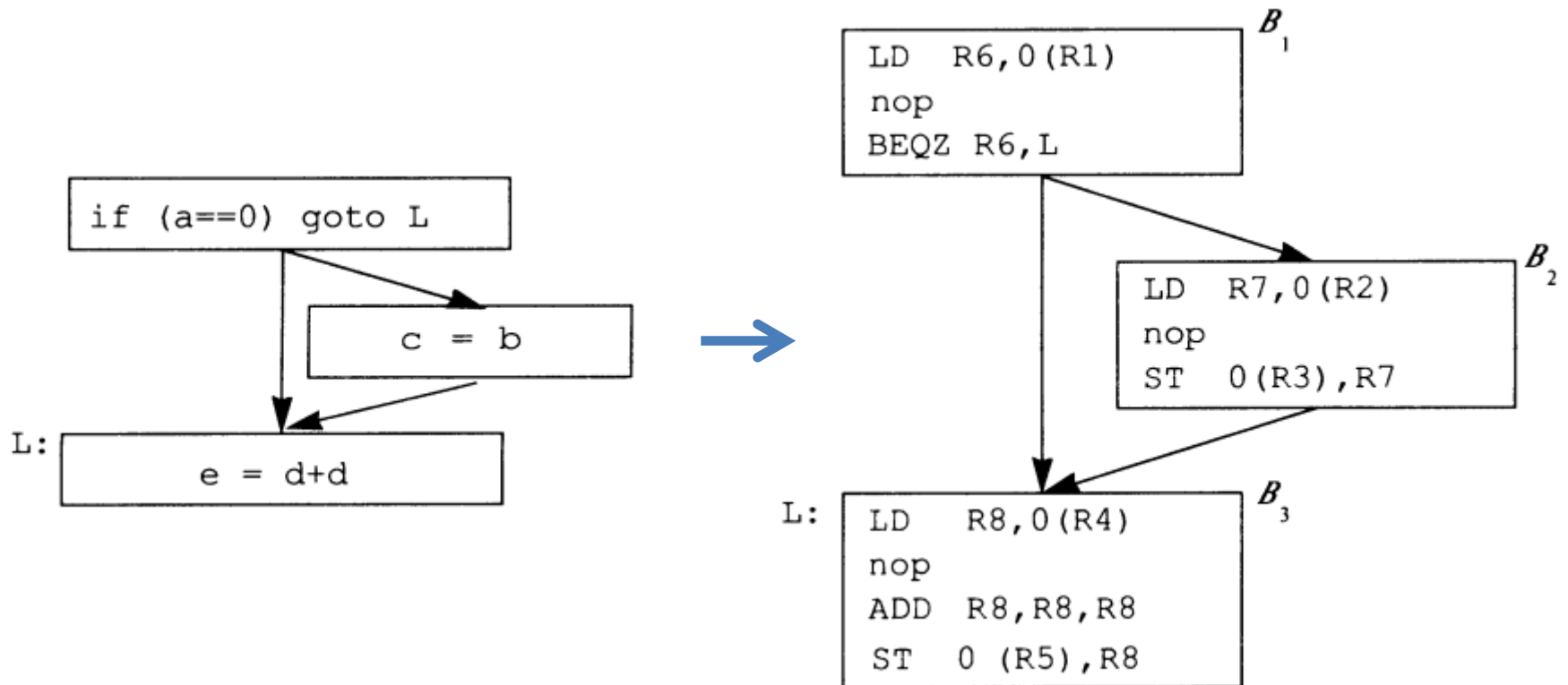
u_k – кол-во использования ресурса k

r_k – кол-во единиц ресурса k

Узлы, требующие ресурсы с большей степенью использования, планировать раньше

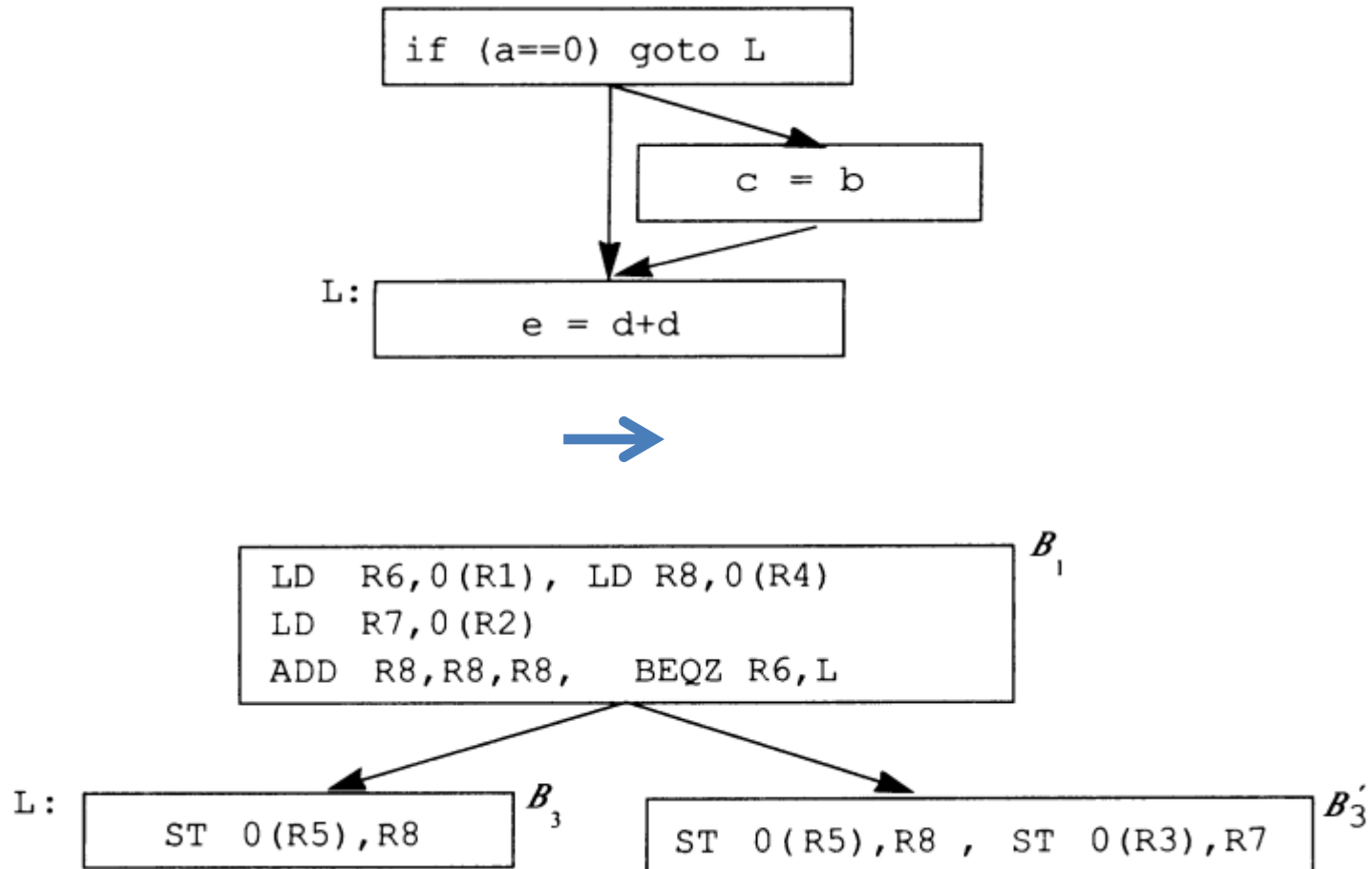
Глобальное планирование инструкций

Глобальное планирование инструкций



Локально спланированный код

Глобальное планирование инструкций



Глобально спланированный код

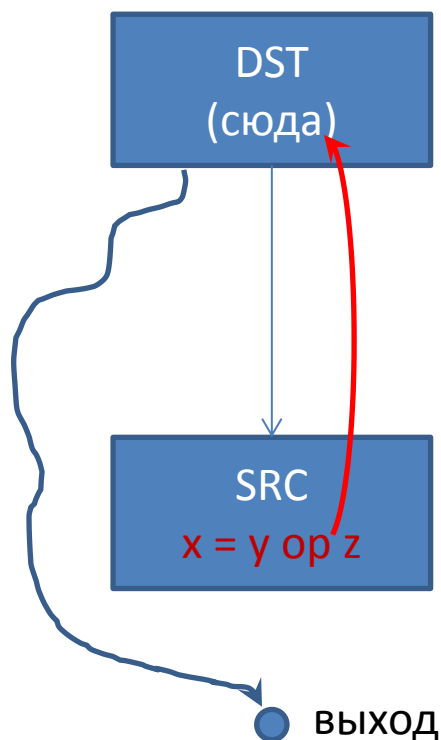
Глобальное планирование инструкций

Требования:

- должны выполняться все команды исходной программы
- команды, которые выполнены избыточно, не должны иметь нежелательных побочных действий

Восходящее перемещение кода

● ВХОД

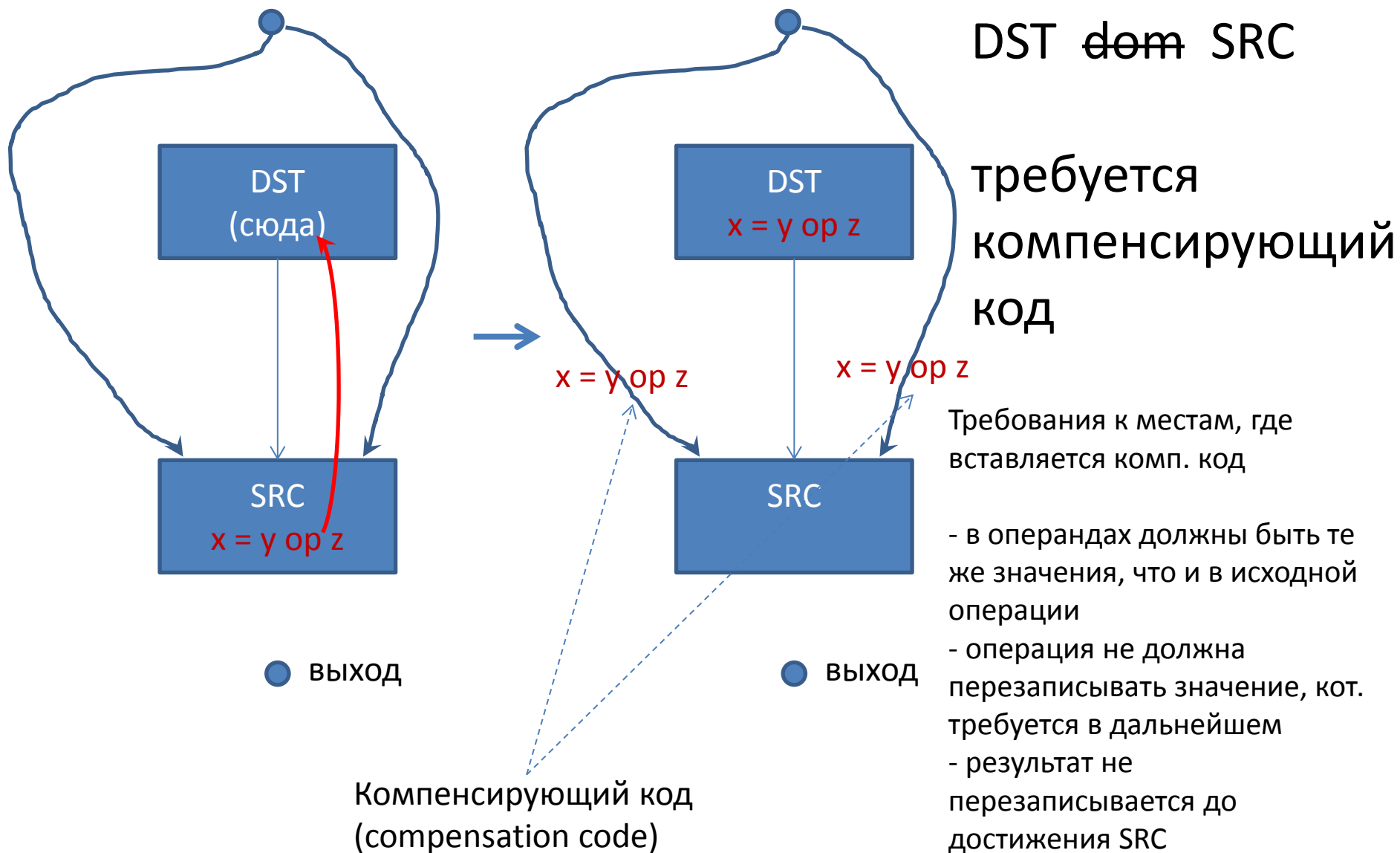


SRC ~~from~~ DST

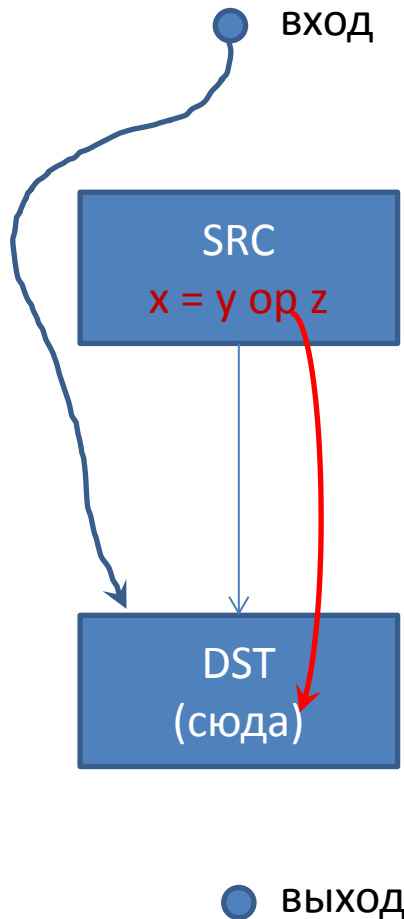
появляются пути, на которых
будет выполнена “ $x = y \text{ op } z$ ”

Корректно, если лишняя
операция “ $x = y \text{ op } z$ ” не
приводит к нежелательным
последствиям

Восходящее перемещение кода



Нисходящее перемещение кода



SRC ~~dom~~ DST

появляются пути, на которых будет выполнена “ $x = y \text{ or } z$ ”

Корректно, если лишняя операция “ $x = y \text{ or } z$ ” не приводит к нежелательным последствиям

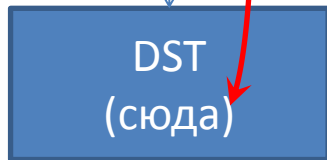
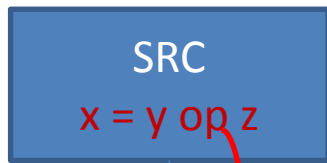
Нисходящее перемещение кода

● ВХОД

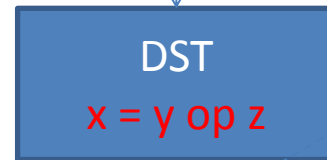
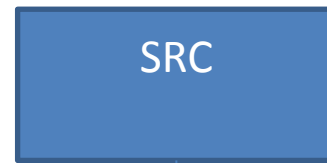
● ВХОД

DST ~~from~~ SRC

требуется
компенсирующий
код



● ВЫХОД

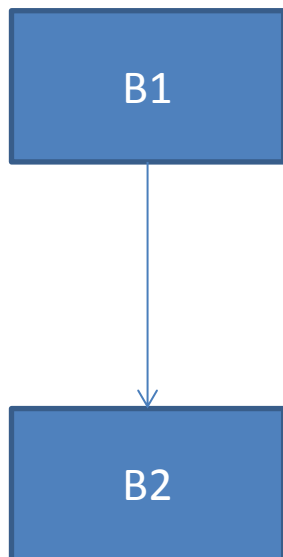


● ВЫХОД

Компенсирующий код
(compensation code)

Эквивалентность с точки зрения управления

$B1$ и $B2$ – эквивалентны с точки зрения управления, если



- $B1 \text{ dom } B2$ и
- $B2 \text{ pdom } B1$

$B1$ выполняется $\Leftrightarrow B2$ выполняется

Множество эквивалентных ББ будем обозначать $\text{ControlEquiv}(B)$

Идея алгоритма

Перемещение кода может привести к повышению эффективности на одних путях и к снижению эффективности на других

Т.о., цель – повысить эффективность наиболее часто выполнимых путей.

Программная конвейеризация (факультатив)

Пример #1

```
for(i = 0; i < n; i++)  
    D[i] = A[i] * B[i] + c;
```

Модель машины:

- За один такт машина может выполнить: одну загрузку, одно сохранение, одну арифметическую операцию или одну операцию ветвления.
- Машина имеет операцию цикла вида

BL R, L

Эта операция уменьшает значение регистра R и, если оно не равно 0, осуществляет переход к L.

- Операции с памятью могут выполняться в автоинкрементном режиме, на что указывают символы ++ после регистра. Значение регистра автоматически увеличивается с тем, чтобы после каждого обращения указывать на следующий адрес в памяти.
- Арифметические операции полностью конвейеризуемы. Они могут быть инициированы на любом такте, но их результаты становятся доступны два такта спустя. Задержка всех прочих команд — один такт.

Пример #1

Локально спланированный код

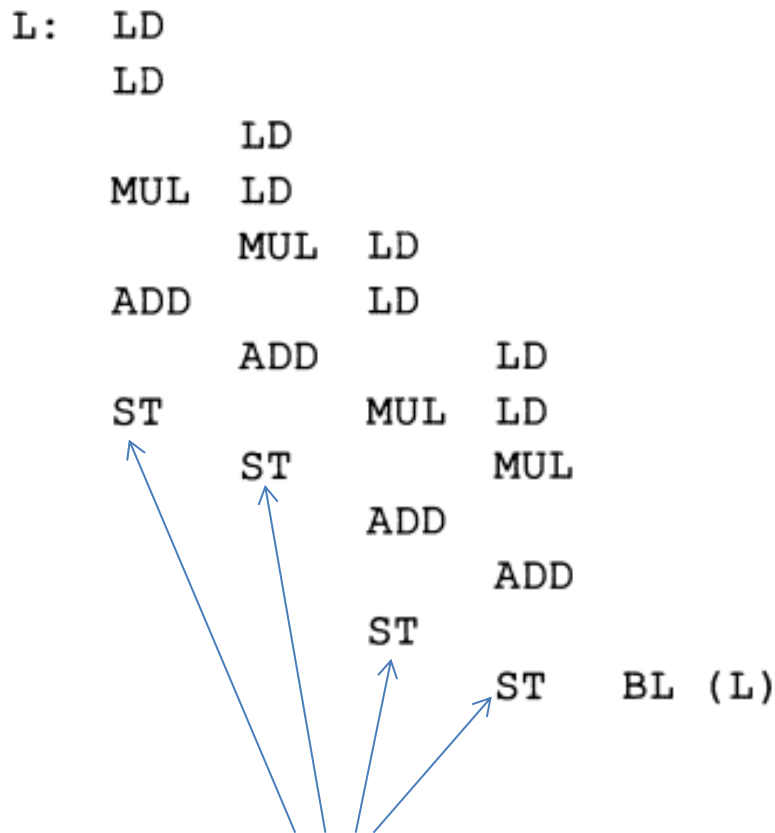
```
// R1, R2, R3 = &A, &B, &D  
// R4          = c  
// R10         = n-1
```

```
L: LD  R5, 0(R1++)  
   LD  R6, 0(R2++)  
   MUL R7, R5, R6  
   nop  
   ADD R8, R7, R4  
   nop  
   ST  0(R3++), R8      BL R10, L
```

7 тактов на одну
итерацию

Пример #1

Развернём на четыре
итерации
и спланируем код



Столбцы по-разному спланированы

4 итерации оригинального цикла
выполняются за 13 тактов

=>

1 итерация оригинального цикла
выполняется за 3.25 такта

цикла, развёрнутый k раз, требует $2*k + 5$

=>

1 итерация оригинального цикла
выполняется за $2 + 5/k$ такта

Мы получаем более оптимальное планирования
за счёт разрастания кода

Пример #1

Развернём на пять итераций
и спланируем код

Такт	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
1	LD				
2	LD				
3	MUL	LD			
4		LD			
5		MUL	LD		
6	ADD		LD		
7			MUL	LD	
8	ST	ADD		LD	
9				MUL	LD
10		ST	ADD		LD
11					MUL
12			ST	ADD	
13					
14				ST	ADD
15					
16					ST

Столбцы одинаково спланированы

На тактах 7, 8 выполняются те же операции, что и на тактах 9, 10

На тактах 7, 8 выполняются операции для итераций 1-4

На тактах 9, 10 выполняются операции для итераций 2-5

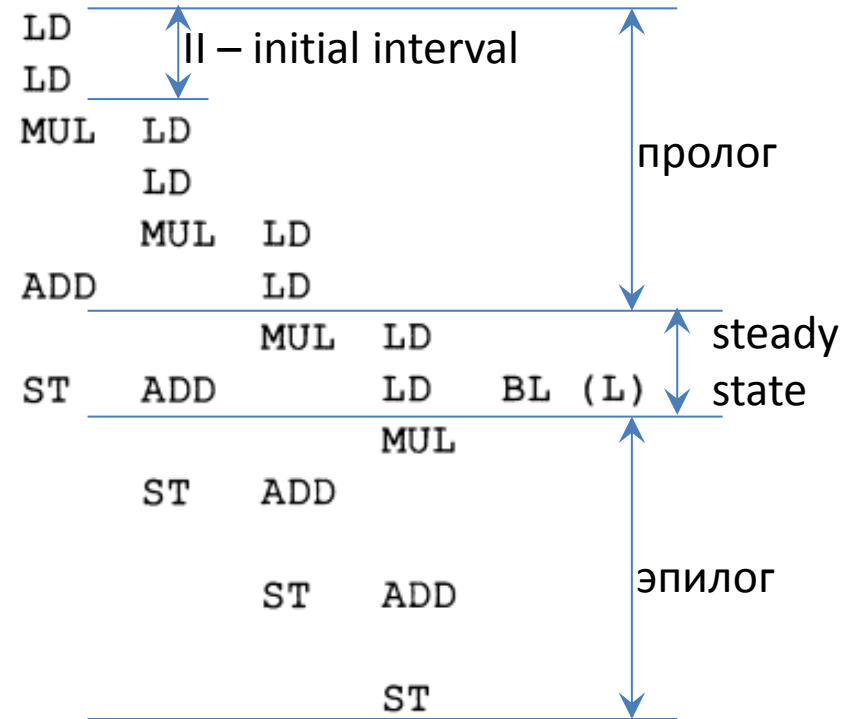
Пример #1

Более компактная запись кода

Это и есть программная конвейеризация (SWP)

Такт	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
1	LD				
2	LD				
3	MUL	LD			
4		LD			
5		MUL	LD		
6	ADD		LD		
7			MUL	LD	
8	ST	ADD		LD	
9				MUL	LD
10		ST	ADD		LD
11				MUL	
12			ST	ADD	
13					
14				ST	ADD
15					
16				ST	

=>
SWP L:



Цикл выполняется $2 \cdot n + 6$ тактов, n – число итераций

Одна итерация длится $2 + 6/n$

Пример #1

Сравним локально спланированный код и SWP-код:

в SWP-коде добавлена ещё одна задержка после mul

	Такт	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
// R1, R2, R3 = &A, &B, &D	1	LD				
// R4 = c	2	LD				
// R10 = n-1	3	MUL	LD			
	4		LD			
L: LD R5, 0(R1++)	5		MUL	LD		
LD R6, 0(R2++)	6	ADD		LD		
MUL R7, R5, R6	7			MUL	LD	
nop	8	ST	ADD		LD	
ADD R8, R7, R4	9				MUL	LD
nop	10		ST	ADD		LD
ST 0(R3++), R8	11					MUL
BL R10, L	12			ST	ADD	
	13					
	14				ST	ADD
	15					
	16					ST

Локально спланированный код не всегда подходит для SWP-планирования

Планирование инструкций при SWP

Определяется двумя факторами:

- интервалом между запусками итераций T (initial interval = II)
- относительным планом S , который для каждой операции указывает её время выполнения относительно начала итерации, которой принадлежит эта инструкция

Т.о., операция n в i -ой итерации выполняется в момент

$$i * T + S(n)$$

Выбор II (учёт ресурсов)

$R = [r_1, r_2, r_3 \dots]$, где $r[i]$ – количество единиц i -ого ресурса

Пусть итерация цикла требует $n[i]$ единиц i -ого ресурса, тогда II должен быть как минимум

$$\text{ResourceII} = \max(n[i] / r[i] \text{ по всем } i)$$

Выбор II (учёт DDG)

```
for(i = 2; i < n; i++)
    A[i] = B[i] + A[i-2];
```

...= A[i-2]

...= B[i]

ld R4, 0(R3++)

ld R2, 0(R1++)

<0, 1>

<0, 1>

add R5, R4, R2

<0, 2>

st 0(R6++), R5

A[i] = ...

<2, 1>

< δ , d >

Dependence
distance

задержка

ld;
add
nop
st

ld

ld;
add
nop
st

II = 2 такта

ld;
add
nop
st

ld

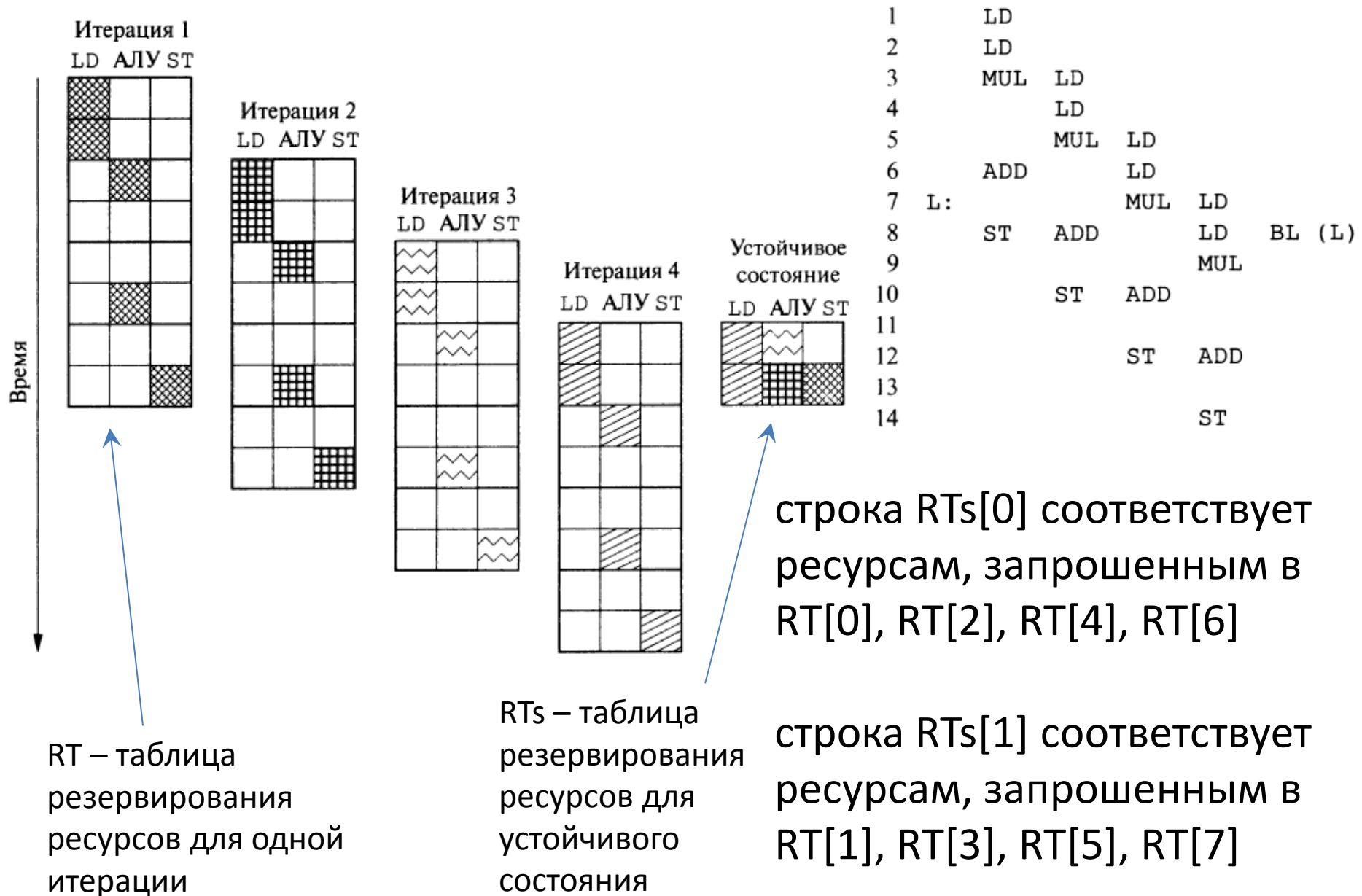
Выбор II (учёт DDG)

II должен быть как минимум

DependencyII =

$$\max_{c\text{—цикл в } DDG} \left[\frac{\sum_{e \in c} d_e}{\sum_{e \in c} \delta_e} \right]$$

Относительный план S



Относительный план S

Т.о., таблица ресурсов для устойчивого состояния строится как

$$RT_S [i] = \sum_{\{t | (t \bmod 2) = i\}} RT [t]$$

SWP алгоритм (ациклический DDG)

Вход:

- цикл на CFG, тело цикла один ББ
- $R = [r1, r2, \dots]$ – ресурсы
- для каждой инструкции n есть RT_n
- ациклический DDG, каждое ребро $e = n_1 \rightarrow n_2$ которого $\langle \delta_e, d_e \rangle$ имеет метку

Выход:

- II
- SWP план S

Метод:

след. слайд

SWP алгоритм (ациклический DDG)

```

main () {
     $T_0 = \max_j \left\lceil \frac{\sum_{n,i} RT_n(i,j)}{r_j} \right\rceil$ ;
    for ( $T = T_0, T_0 + 1, \dots$ , пока не будут спланированы все узлы из  $N$ ) {
         $RT$  = пустая таблица резервирования ресурсов с  $T$  строками;
        for (каждый узел  $n \in N$  в приоритетном топологическом порядке) {
             $s_0 = \max_{e=p \rightarrow n \text{ из } E} (S(p) + d_e)$ ;
            for ( $s = s_0, s_0 + 1, \dots, s_0 + T - 1$ )
                if ( $NodeScheduled(RT, T, n, s)$ ) break;
            if ( $n$  не может быть спланировано в  $RT$ ) break;
        }
    }
}

```

оценка
снизу II
по ресурсам

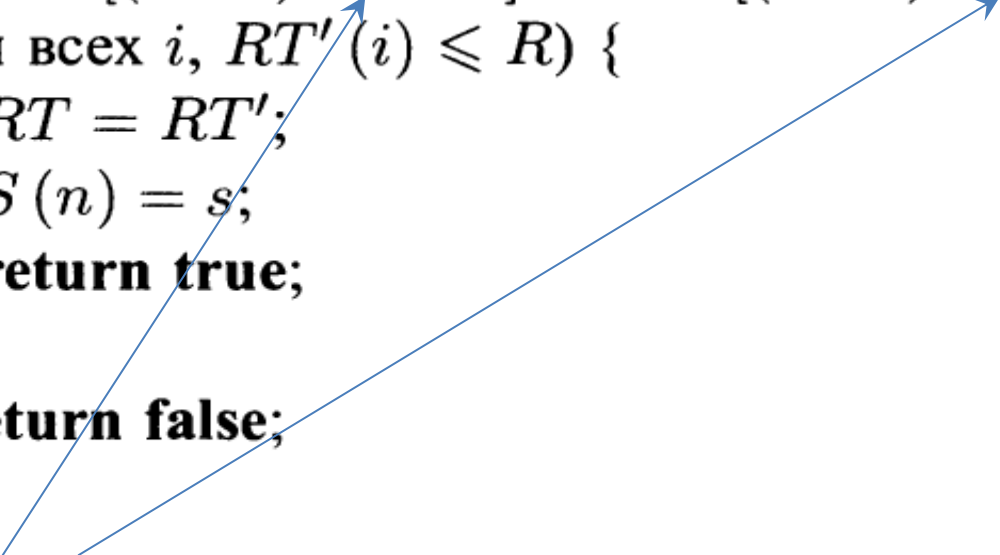
перебираем
возможные
значения II и
пытаемся
построить план

RT —
таблица для
устойчивого
состояния

пытаемся
спланировать
инструкцию n
при $II = T$ на
момент
времени s

SWP алгоритм (ациклический DDG)

```
NodeScheduled ( $RT, T, n, s$ ) {  
     $RT' = RT$ ;  
    for (каждая строка  $i$  из  $RT_n$ )  
         $RT' [(s + i) \bmod T] = RT' [(s + i) \bmod T] + RT_n [i]$ ;  
    if (для всех  $i$ ,  $RT'(i) \leq R$ ) {  
         $RT = RT'$ ;  
         $S(n) = s$ ;  
        return true;  
    }  
    else return false;  
}
```



Сравните с
алгоритмом
планирования ББ

SWP алгоритм (DDG общего вида)

Для двух инструкций $n1$ и $n2$ ($n2$ выполняется позже), которые связаны дугой DDG, мы можем сказать

$$\delta_1 \times T + S(n2) - S(n1) \geq d_1 \Rightarrow \\ S(n2) - S(n1) \geq d_1 - \delta_1 \times T$$

или транзитивно для некоторого пути p

$$S(n2) - S(n1) \geq \sum_{e \in p} d_e - \delta_e \times T$$

SWP алгоритм (DDG общего вида)

для некоторого пути p

$$S(n2) - S(n1) \geq \sum_{e \in p} d_e - \delta_e \times T$$

с другой стороны T должно быть как минимум

$$\max_{c\text{-цикл в } DDG} \left[\frac{\sum_{e \in c} d_e}{\sum_{e \in c} \delta_e} \right]$$

\Rightarrow

если p – цикл, то $\sum_{e \in p} d_e - \delta_e \times T \leq 0$

Т.о., более строгие ограничения будут получаться для ациклических путей на графе

SWP алгоритм (DDG общего вида)

Вход:

- цикл на CFG, тело цикла один ББ
- $R = [r1, r2, \dots]$ – ресурсы
- для каждой инструкции n есть RT_n
- DDG (возможно с циклами), каждое ребро которого имеет метку

Выход:

- II
- SWP план S

Метод:

след. слайд

SWP алгоритм (DDG общего вида)

```
main () {  
     $E' = \{e \mid e \in E, \delta_e = 0\};$   
     $T_0 = \max \left( \max_j \left\lceil \frac{\sum_{n,i} RT_n(i,j)}{r_j} \right\rceil, \max_{c-\text{цикл в } G} \left\lceil \frac{\sum_{e \in c} d_e}{\sum_{e \in c} \delta_e} \right\rceil \right);$   
    for ( $T = T_0, T_0 + 1, \dots$  или пока все сильно связанные компоненты  
        в  $G$  не будут спланированы) {  
         $RT$  = пустая таблица резервирования с  $T$  строками;  
         $E^* = \text{AllPairsLongestPath}(G, T);$   
        for (каждый сильно связанный компонент  $C$  из  $G$   
            в приоритетном топологическом порядке) {  
            for (все  $n$  из  $C$ )  
                 $s_0 = \max_{e=p \rightarrow n \text{ из } E^*, \text{ узел } p \text{ спланирован}} (S(p) + d_e);$   
             $first$  = некоторое  $n$ , такое, что  $s_0(n)$  является минимумом;  
             $s_0 = s_0(first);$   
            for ( $s = s_0; s < s_0 + T; s = s + 1$ )  
                if ( $\text{SccScheduled}(RT, T, C, first, s)$ ) break;  
            if ( $C$  не может быть спланирован в  $RT$ ) break;  
        }  
    }  
}
```

SWP алгоритм (DDG общего вида)

```
SccScheduled (RT, T, C, first, s) {  
    RT' = RT;  
    if (not NodeScheduled (RT', T, first, s)) return false;  
    for (каждый остающийся n из c в приоритетном  
        топологическом порядке ребер из E') {  
         $s_l = \max_{e=n' \rightarrow n \text{ из } E^*, n' \in c, \text{узел } n' \text{ спланирован}} (S(n') + d_e - (\delta_c \times T));$   
         $s_u = \min_{e=n' \rightarrow n \text{ из } E^*, n' \in c, \text{узел } n' \text{ спланирован}} (S(n') - d_e + (\delta_e \times T));$   
        for ( $s = s_l$ ;  $s \leq \min(s_u, s_l + T - 1)$ ;  $s = s + 1$ )  
            if (NodeScheduled (RT', T, n, s)) break;  
        if (n не может быть спланирован в RT') return false;  
    }  
    RT = RT';  
    return true;  
}
```

Список литературы

- Компиляторы. Принципы, технологии и инструментарий. 2-е изд. А. Ахо