Параллельные алгоритмы
20250218_03

Основные понятия(2)

Якобовский Михаил Владимирович

Основные характеристики параллельной программы

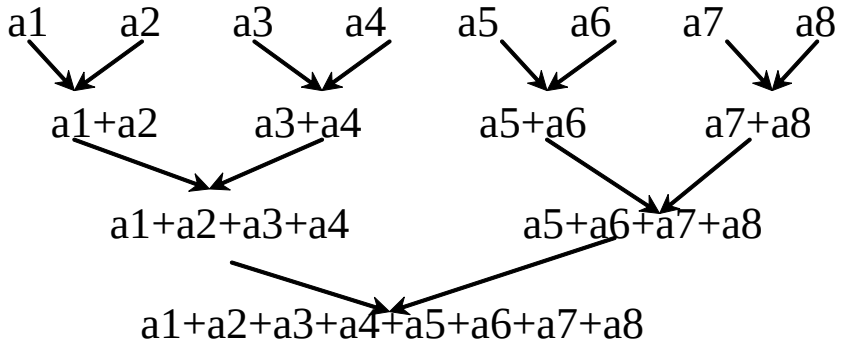
- Ускорение
- Эффективность
- Масштабируемость

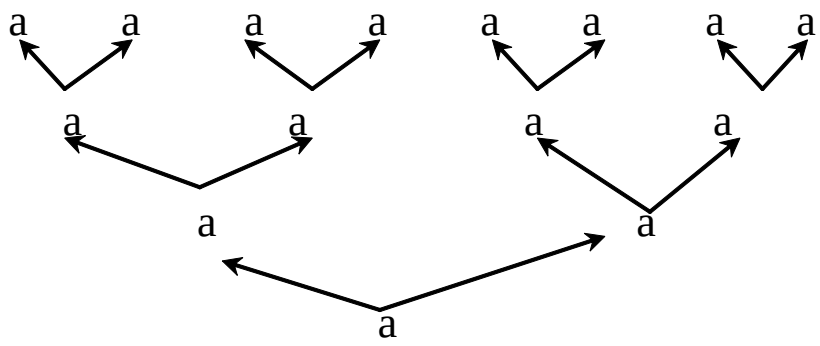
  – Число выполняемых операций
  – Время выполнения
  – Объём обрабатываемых данных

Метод сдваивания

Выполнение редукционных и им подобных операций

- Определение суммы элементов массива
- Определение минимального элемента массива
- Широковещательная рассылка данных
- …



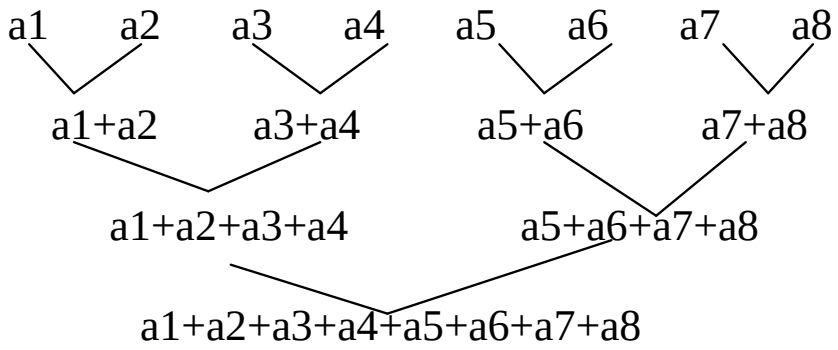


Основные характеристики параллельной программы

- ***Ускорение*** $S_p = \frac{T_1}{T_p}$
- ***Эффективность*** $E_p = \frac{S_p}{p}$
- ***Предел масштабируемости*** – минимальное число процессоров, при котором достигается максимальное ускорение
  - Число выполняемых операций
  - Время выполнения
  - Объём обрабатываемых данных

Метод сдваивания

Каскадная схема



$$T_1(n) = \tau_c(n-1)$$

$$T_{p=n/2}(n) = (\tau_c + \tau_s)\log_2 n$$

$$S_{p=n/2}(n) \approx \frac{n\tau_c}{(\tau_c+\tau_s)\log_2 n} = \frac{n}{\left(1+\frac{\tau_s}{\tau_c}\right)\log_2 n}$$

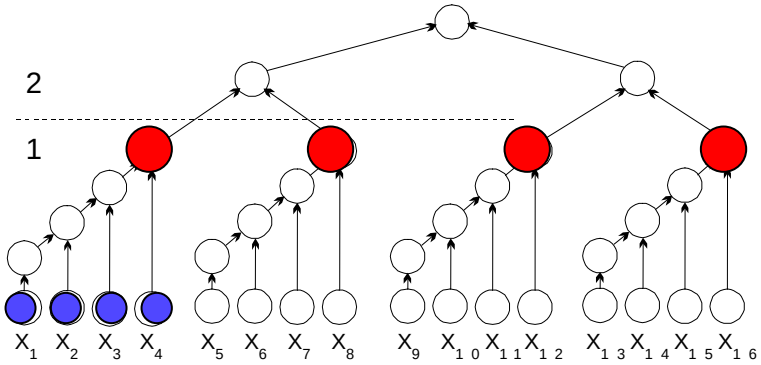
$$E_{p=n/2}(n) = \frac{s_p}{p} = \frac{2}{\left(1+\frac{\tau_s}{\tau_c}\right)\log_2 n}$$

Метод сдваивания

Модифицированная каскадная схема

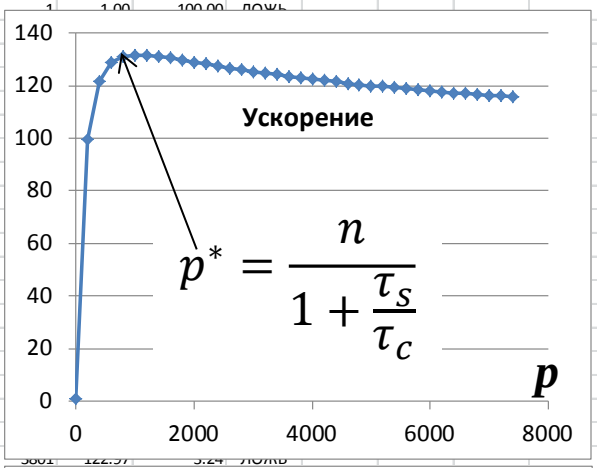
$$T_p = \frac{n}{p}\tau_c + (\tau_c + \tau_s)\log p$$

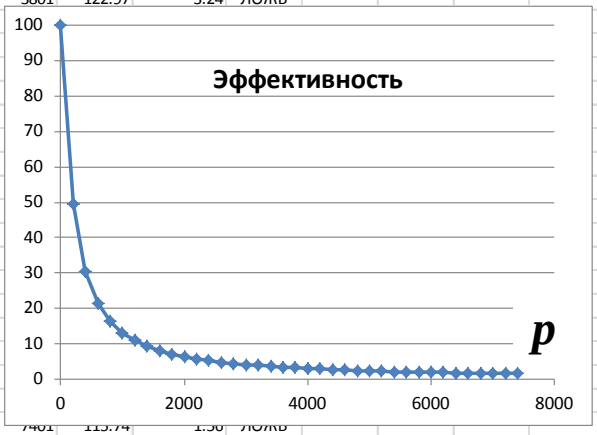
$$S_p = p\frac{1}{1 + \left(1 + \frac{\tau_s}{\tau_c}\right)\frac{p}{n}\log p}$$

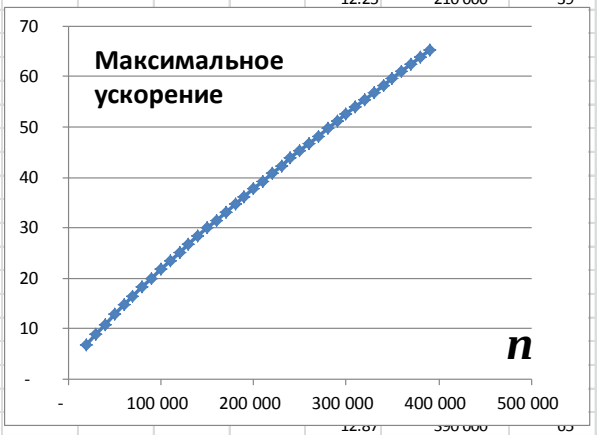


Масштабируемость

ts/tc	1 000															
n=2^c	1 048 576	20														
step	200													10 000		
p*	1 047.53															
log(p)	p	S	E										log(n)	n	S*	p*
0.00																20
5.30																30
5.99																40
6.40																50
6.69																60
6.91																70
7.09																80
7.24																90
7.38																100
7.50																110
7.60																120
7.70																130
7.78																140
7.86																150
7.94																160
8.01																170
8.07																180
8.13																190
8.19																200
8.24																210
8.29																220
8.34																230
8.39																240
8.43																250
8.48																260
8.52																270
8.56																280
8.59																290
8.63																300
8.67																310
8.70																320
8.73																330
8.76																340
8.79																350
8.82																360
8.85																370
8.88																380
8.91																390







Сверхлинейное ускорение

ускорение параллельного алгоритма

$$S_p = \frac{T_1}{T_p}$$

эффективность использования вычислительной мощности

$$E_p = \frac{S_p}{p}$$

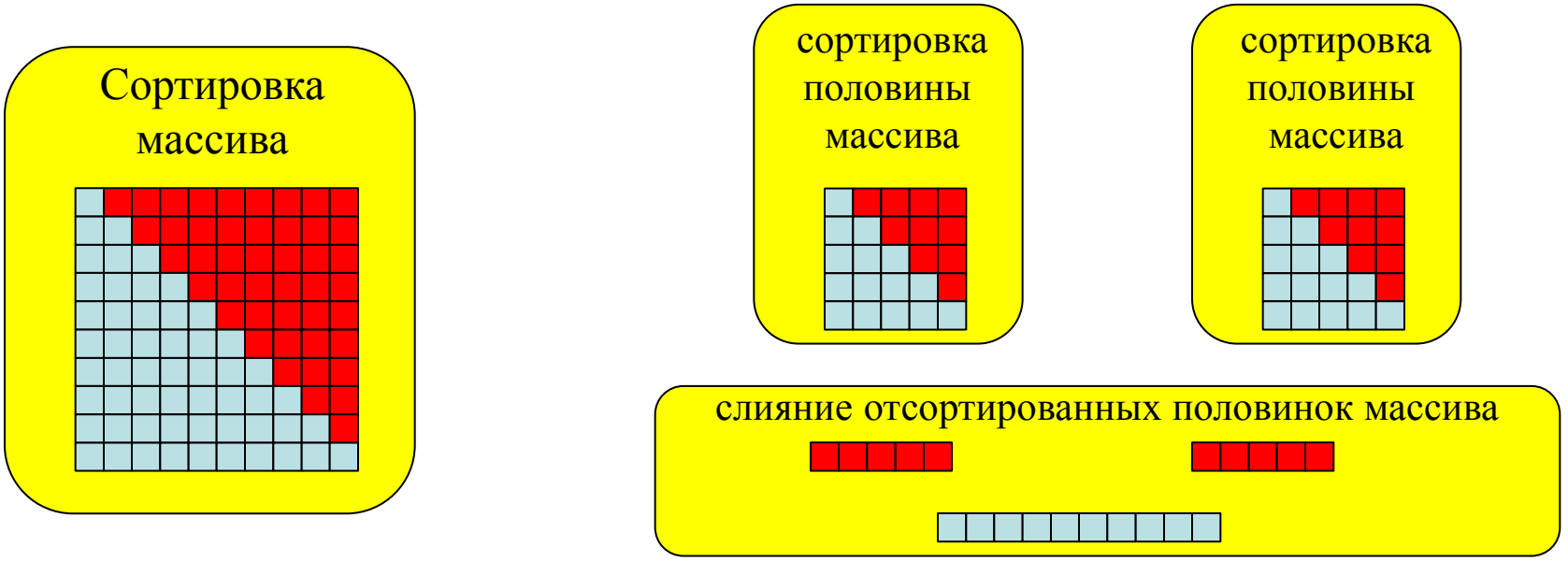
Может ли ускорение превышать число процессоров?

$$S_p = \frac{T_1}{T_p} > p \quad ?$$

$$E_p = \frac{S_p}{p} > 1$$

Может ли быть $S_p > p$?

– Пример неудачного последовательного алгоритма



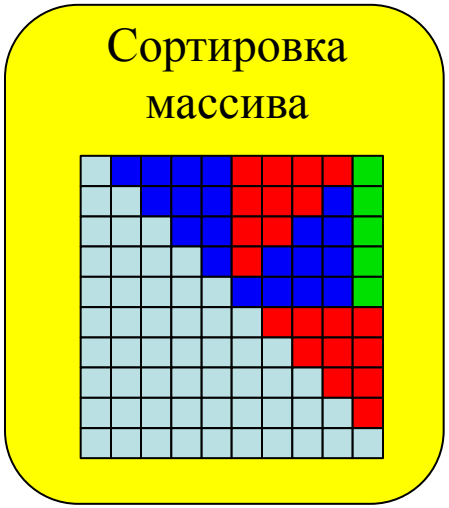
$$\frac{n(n-1)}{2} \sim \frac{n^2}{2}$$

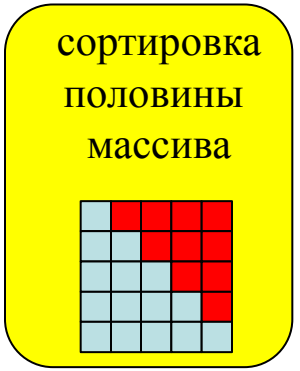
$$\frac{\left(\frac{n}{2}\right)^2}{2} + n \sim \frac{1}{4}\frac{n^2}{2}$$

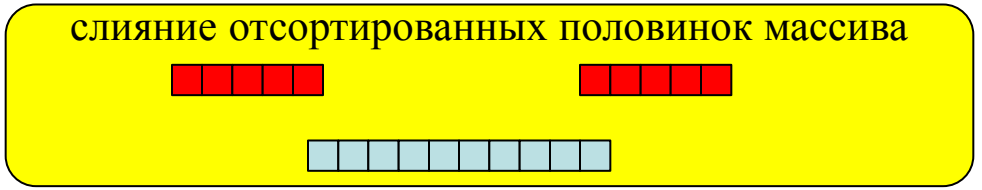
$$\boldsymbol{S_p = 4} \qquad \boldsymbol{E_p = \frac{4}{2} = 200\%}$$

Почему $S_p > p$?

– В последовательном алгоритме выполняется больше операций, чем в параллельном







$$\frac{n(n-1)}{2} \sim \frac{n^2}{2}$$

$$\frac{\left(\frac{n}{2}\right)^2}{2} + n \sim \frac{1}{4}\frac{n^2}{2}$$

$$\boldsymbol{S_p = 4} \qquad \boldsymbol{E_p = \frac{4}{2} = 200\%}$$

Сверхлинейное ускорение

ускорение параллельного алгоритма

$$S_p = \frac{T_1}{T_p}$$

эффективность использования вычислительной мощности

$$E_p = \frac{S_p}{p}$$

Ускорение параллельного алгоритма относительно наилучшего последовательного

$$S_p^* = \frac{T_1^*}{T_p} \qquad E_p^* = \frac{S_p^*}{p}$$

Сверхлинейное ускорение

ускорение параллельного алгоритма

$$S_p = \frac{T_1}{T_p}$$

эффективность использования вычислительной мощности

$$E_p = \frac{S_p}{p}$$

Может ли ускорение превышать число процессоров?

$$S_p = \frac{T_1}{T_p} > p$$

$$E_p = \frac{S_p}{p} > 1$$

?

Да, если выбран неудачный последовательный алгоритм

Сверхлинейное ускорение

ускорение параллельного алгоритма

$$S_p = \frac{T_1}{T_p}$$

эффективность использования вычислительной мощности

$$E_p = \frac{S_p}{p}$$

Может ли ускорение превышать число процессоров?

$$S_p = \frac{T_1}{T_p} > p$$

$$E_p = \frac{S_p}{p} > 1$$

?

Да, если на скорость выполнения влияют аппаратные особенности вычислительной системы

Может ли неэффективный алгоритм решить задачу быстрее эффективного?

- Да
  - Если первый алгоритм позволяет эффективно использовать больше процессоров, чем второй, если предел его масштабируемости выше

$$M_1 E_1 p_1^* > M_2 E_2 p_2^*, \quad \text{при } E_1 < E_2, \qquad \text{но } M_1 p_1^* > \frac{E_2}{E_1} M_2 p_2^*$$

- Самый эффективный алгоритм – алгоритм, использующий один **(1)** процессор.
  - Его эффективность равна 100%, но он не всегда самый быстрый.

Модель выполнения программы на общей памяти

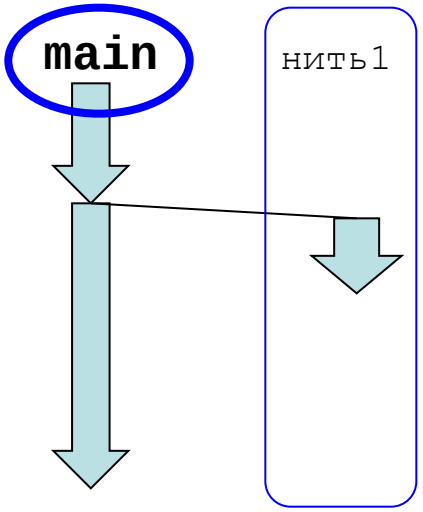
- Работа начинается с запуска одного экземпляра программы
- При необходимости программа порождает новые процессы, каждый из которых:
  - Обладает собственными локальными переменными
  - Имеет доступ к глобальным переменным

```
int a_global;
main()
{
int b1_local;
Запуск нити(fun())
}
fun()
{
int b2_local;
Запуск нити(…)
}
```

Модель выполнения программы на общей памяти

- Работа начинается с запуска одного экземпляра программы
- При необходимости программа порождает новые процессы, каждый из которых:
  - Обладает собственными локальными переменными
  - Имеет доступ к глобальным переменным

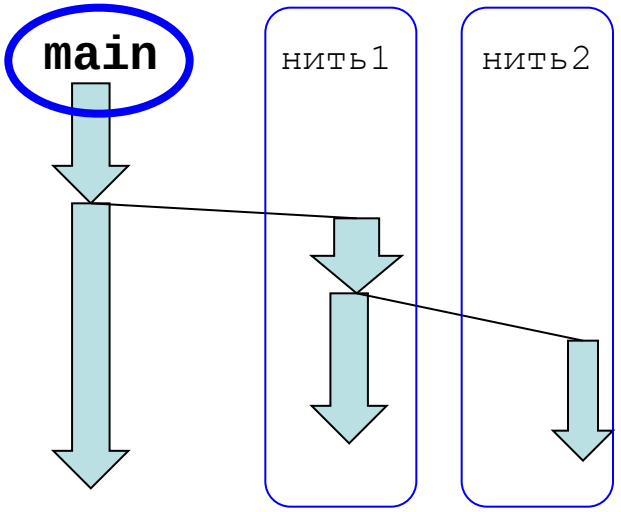
```
int a_global;
main()
{
int b1_local;
Запуск нити(fun())
}
fun()
{
int b2_local;
Запуск нити(…)
}
```



Модель выполнения программы на общей памяти

- Работа начинается с запуска одного экземпляра программы
- При необходимости программа порождает новые процессы, каждый из которых:
  - Обладает собственными локальными переменными
  - Имеет доступ к глобальным переменным

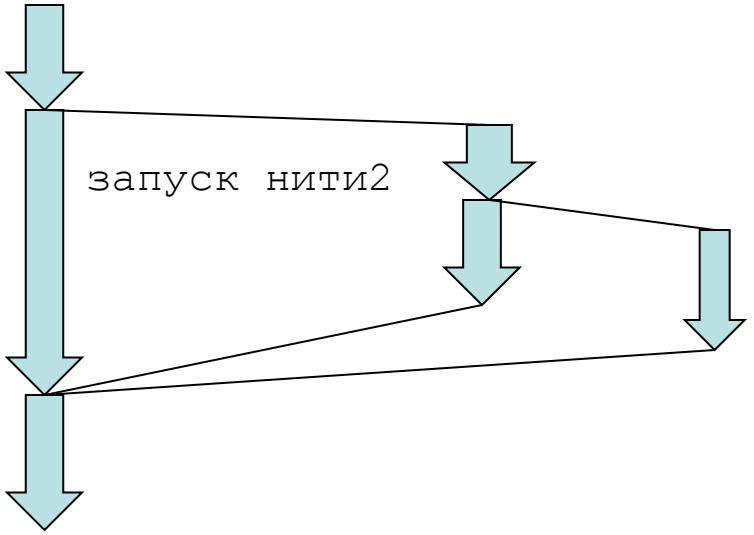
```
int a_global;
main()
{
int b1_local;
Запуск нити(fun())
}
fun()
{
int b2_local;
Запуск нити(…)
}
```



Модель программы на общей памяти

- Работа начинается с запуска одной программы
- При необходимости программа порождает новые процессы, эти процессы:
  - Обладают собственными локальными переменными
  - Имеют доступ к общим глобальным переменным

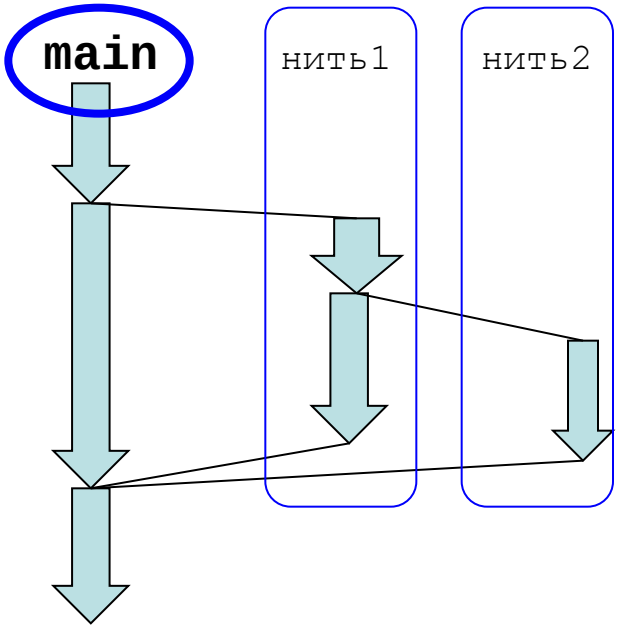
```
int a_global;
main()
{
Start_process(fun);
int b1_local;
}

fun()
{
int b2_local;
}
```



Модель выполнения программы на общей памяти

- Работа начинается с запуска одного экземпляра программы
- При необходимости программа порождает новые процессы, каждый из которых:
  - Обладает собственными локальными переменными
  - Имеет доступ к глобальным переменным

```
int a_global;
main()
{
int b1_local;
Запуск нити(fun())
}
fun()
{
int b2_local;
Запуск нити(…)
}
```



Что будет напечатано?

```
int a=1;
```

```
Нить1
{
  a=a+2
}
```

```
Нить2
{
  a=a+3
}
```

```
Нить3
{
  print(a)
}
```

a=?

Что будет напечатано?

```
int a=1;
```

```
Нить1
{
   a=a+2
}
```

```
Нить2
{
   a=a+3
}
```

```
Нить3
{
   print(a)
}
```

6

Что будет напечатано?

```
int a=1;
```

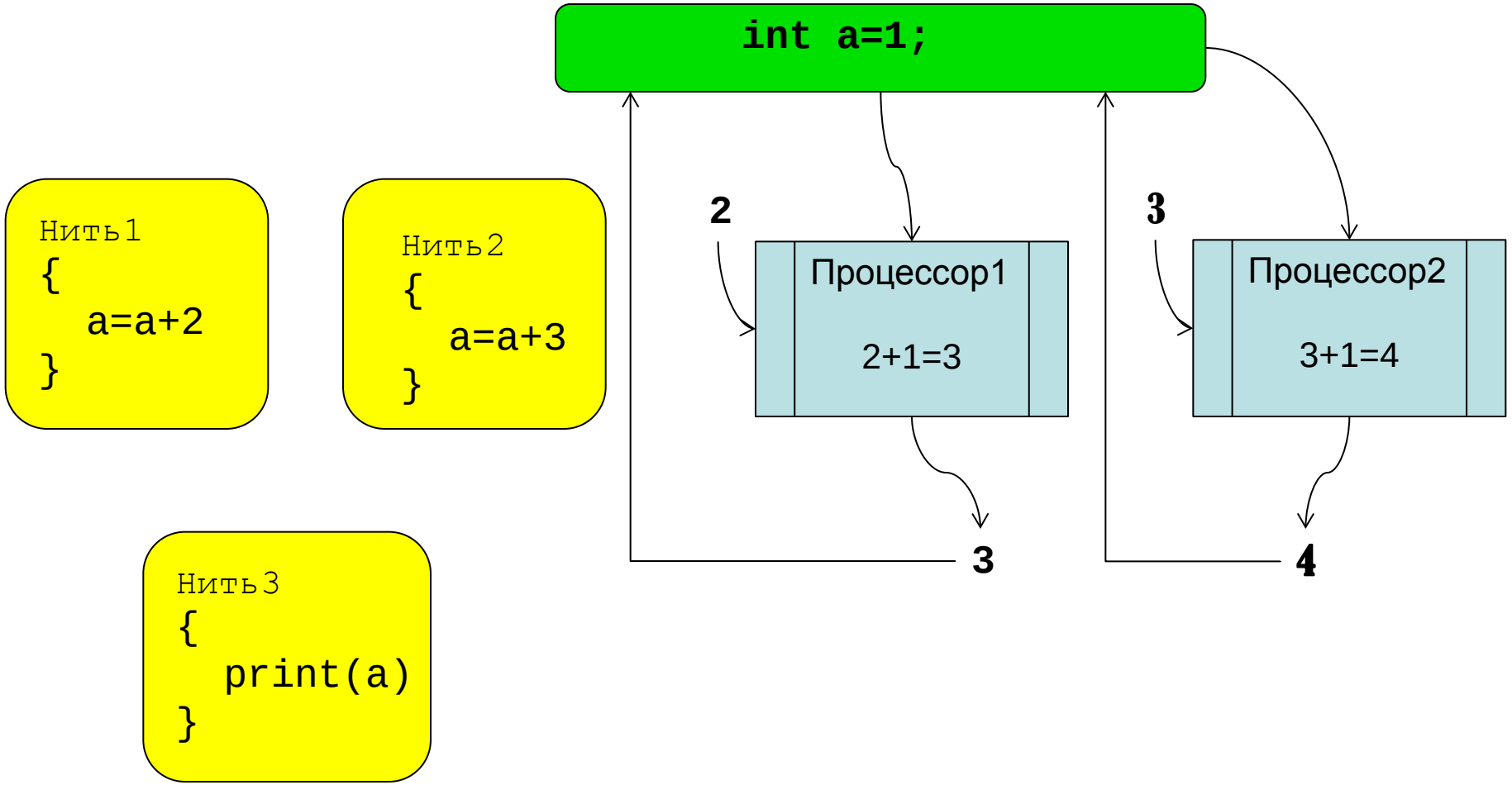
```
Нить1
{
  a=a+2
}
```

```
Нить3
{
  print(a)
}
```

```
Нить2
{
  a=a+3
}
```

3

Что будет напечатано?



6 ? 4 ? 3

a=?

```
int a=0
```

```
// Нить 1:

{  int i

   for(i=0;i<100;i++)
      a += 1
}
```

```
// Нить 2:

{  int i

   for(i=0;i<100;i++)
      a += 1
}
```

```
print a
```

a=?

```
a = 0;
```

```
// Нить 1:

{
  //  a += 1
    mov r1 , a
    inc r1
    mov a , r1
}
```

```
// Нить 2:

{
  //  a += 1
    mov r1 , a
    inc r1
    mov a , r1
}
```

a = 1 или a = 2 ?

Что будет напечатано?

```
int a=1;
```

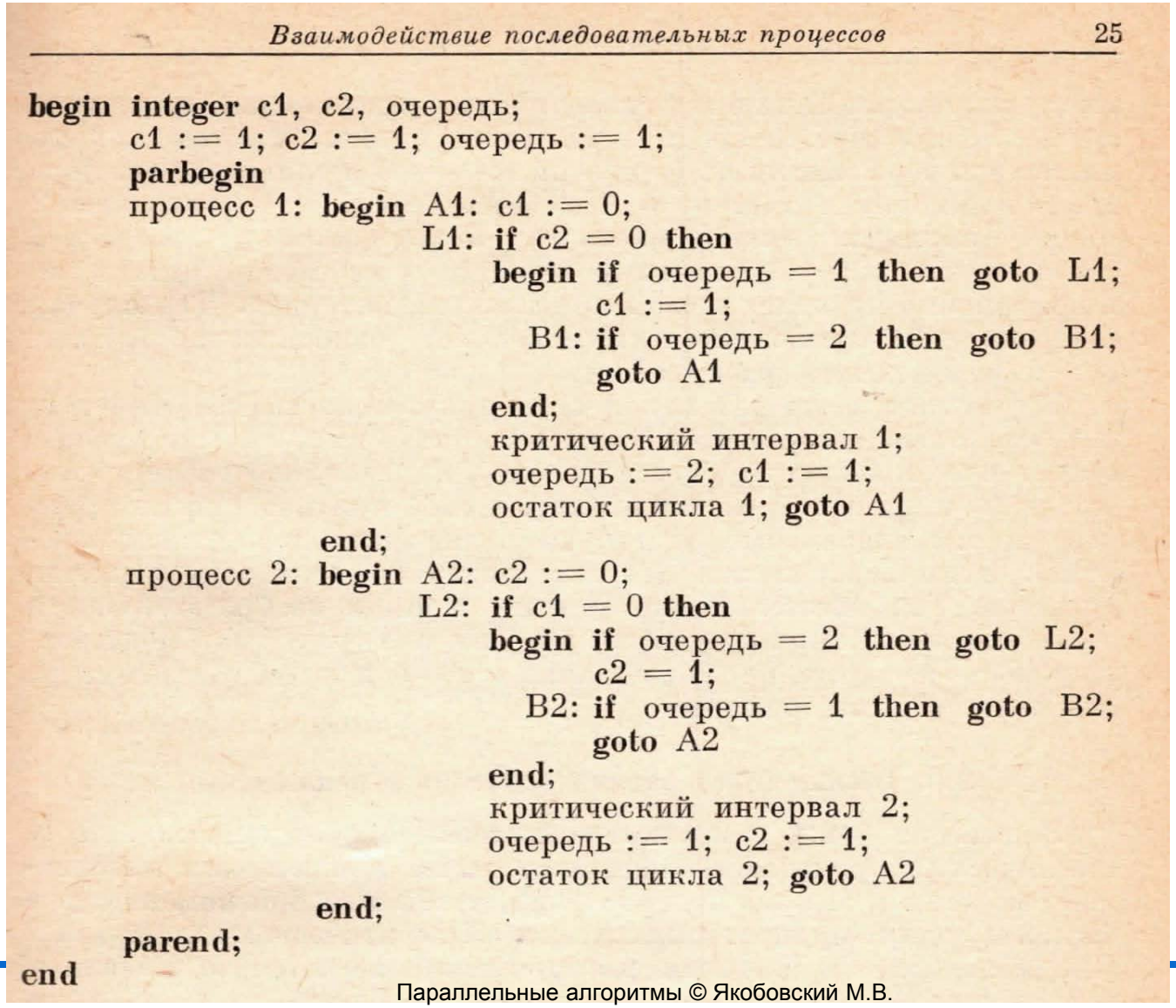
```
Нить1
{
  a=a+2
}
```

```
Нить2
{
  a=a+3
}
```

```
Нить3
{
  print(a)
}
```

6 ? 4 ? 3 ? ___

Алгоритм Деккера



Взаимодействие последовательных процессов 25

```
begin integer c1, c2, очередь;
      c1 := 1; c2 := 1; очередь := 1;
      parbegin
      процесс 1: begin A1: c1 := 0;
                     L1: if c2 = 0 then
                         begin if очередь = 1 then goto L1;
                               c1 := 1;
                           B1: if очередь = 2 then goto B1;
                               goto A1
                         end;
                         критический интервал 1;
                         очередь := 2; c1 := 1;
                         остаток цикла 1; goto A1
                 end;
      процесс 2: begin A2: c2 := 0;
                     L2: if c1 = 0 then
                         begin if очередь = 2 then goto L2;
                               c2 = 1;
                           B2: if очередь = 1 then goto B2;
                               goto A2
                         end;
                         критический интервал 2;
                         очередь := 1; c2 := 1;
                         остаток цикла 2; goto A2
                 end;
      parend;
end
```

Алгоритм Деккера

http://lectures.stargeo.ru/add_qst/semaph.htm Староверов В.М.

```
//указывает занятость ресурса, соответственно, первым и вторым процессами
int flag[2]={0,0};
int Num=0;
```

```
//Процесс 1:

void CriticalBegin1()
{
  flag[0]=1;//занять ресурс первым процессом
 while(flag[1]==1)//пока ресурс занят вторым процессом
       if(Num==1)//если надо уступить второму процессу
         {
          flag[0]=0;//освободим ресурс
          while(Num==1); //подождем завершения второго процесса
          flag[0]=1;//займем ресурс
         }
}

void CriticalEnd1();
{
 Num=1;
 flag[0]=0;//освободить ресурс первым процессом
}
```

```
//Процесс 2:

void CriticalBegin2()
{
  flag[1]=1;//занять ресурс вторым процессом
 while(flag[0]==1)//пока ресурс занят первым процессом
     if(Num==0)//если надо уступить первому процессу
       {
        flag[1]=0;//освободим ресурс
        while(Num==0); //подождем завершения первого процесса
        flag[1]=1;//займем ресурс
       }
}

void CriticalEnd2();
{
 Num=0;
 flag[1]=0;//освободить ресурс вторым процессом
}
```

Семафор

- Целочисленная неотрицательная переменная
- Инициализация и две **атомарные** операции

- Операция V(S) – атомарная !
  - Атомарно увеличивает значение S на 1

- Операция P(S) – атомарная !
  - Если S положительно, то уменьшает S на 1
  - Иначе ждет, пока S не станет больше 0

Языки программирования. Редактор Ф.Женюи. Перевод с англ. В.П.Кузнецова. Под ред. В.М.Курочкина. М:."Мир", 1972 Э. Дейкстра. Взаимодействие последовательных процессов.
http://khpi-iip.mipk.kharkiv.edu/library/extent/dijkstra/ewd123/index.html

Что будет напечатано?

```
int a=1;
Sem S=1, S1=0, S2=0;
```

```
Нить1
{
  P(S)
  a=a+2
  V(S)
  V(S1)
}
```

```
Нить2
{
  P(S)
  a=a+3
  V(S)
  V(S2)
}
```

```
Нить3
{
 P(S1)
 P(S2)
 print(a)
}
```

6

Контакты

Якобовский М.В.

чл.-корр. РАН, проф., д.ф.-м.н.,

заместитель директора по научной работе
Института прикладной математики
им. М.В. Келдыша Российской академии наук

mail: lira@imamod.ru

web: http://lira.imamod.ru