Параллельные высокопроизводительные вычисления

Якобовский М.В., проф., д.ф.-м.н.
СКИ МГУ,
Институт прикладной математики
им. М.В.Келдыша РАН, Москва

© Якобовский М.В.

Ускорение и эффективность параллельных алгоритмов

ускорение параллельного алгоритма

$$S(p)=T(1)/T(p)$$

эффективность использования вычислительной мощности

$$E(p)=S(p)/p$$

Может ли ускорение превышать число процессоров?

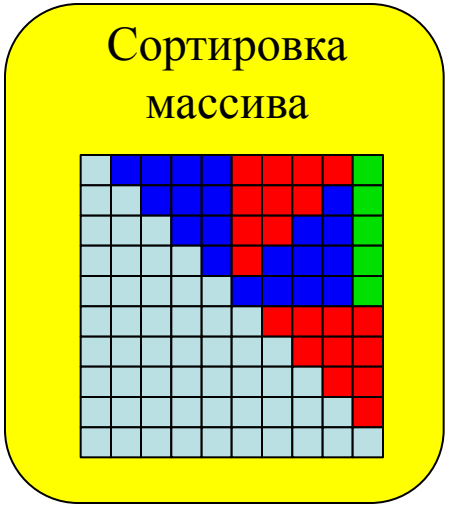
$$S(p)> p$$
$$E(p)>100\%$$

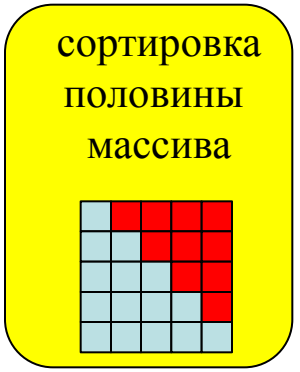
Да, если учитывать влияние аппаратных особенностей вычислительной системы

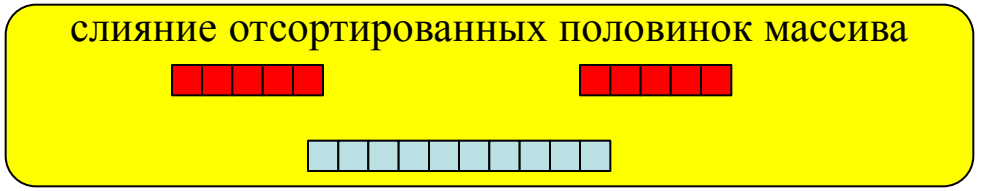
Да, если выбран неудачный последовательный алгоритм

Может ли быть S(p)>p ?

– В последовательном алгоритме выполняется больше операций, чем в параллельном







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

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

Ускорение и эффективность параллельных алгоритмов

ускорение параллельного алгоритма

$$S(p)=T(1)/T(p)$$

эффективность использования вычислительной мощности

$$E(p)=S(p)/p$$

Ускорение параллельного алгоритма относительно наилучшего последовательного

$$S^*(p)= T^*(1)/T(p)$$

$$E^*(p)= S^*(p)/p$$

Неубедительность тестирования

… если для нас представляют интерес реально работающие системы, то требуется убедиться, (и убедить всех сомневающихся) в корректности наших построений

… системе часто придется работать в невоспроизводимых обстоятельствах, и мы едва ли можем ожидать сколько-нибудь серьезной помощи от тестов

Dijkstra E.W.

1966

Простые алгоритмы и их свойства

- Время выполнения параллельного алгоритма
- Методы построения параллельных алгоритмов
  - Статическая балансировка
    - метод сдваивания
    - геометрический параллелизм
    - конвейерный параллелизм
  - Динамическая балансировка
    - коллективное решение
    - диффузная балансировка загрузки

Хороший параллельный алгоритм

большим

- Обладает запасом внутреннего параллелизма
  - Есть возможность одновременного выполнения множества операций
- Допускает возможность равномерного распределения вычислительных операций между процессорами

большим числом

- Обладает низким уровнем накладных расходов

К вопросу о накладных расходах

α - доля операций выполняемых последовательно

$$T_p = \alpha T_1 + (1-\alpha)\frac{T_1}{p} + t_d$$

t_d - дополнительные расходы времени

$$S_p = \frac{T_1}{T_p} = \frac{T_1}{\alpha T_1 + (1-\alpha)\frac{T_1}{p} + t_d}$$

Закон Амдаля

α - доля операций выполняемых последовательно

$$S_p \leq \frac{1}{\alpha + \frac{1-\alpha}{p}}$$

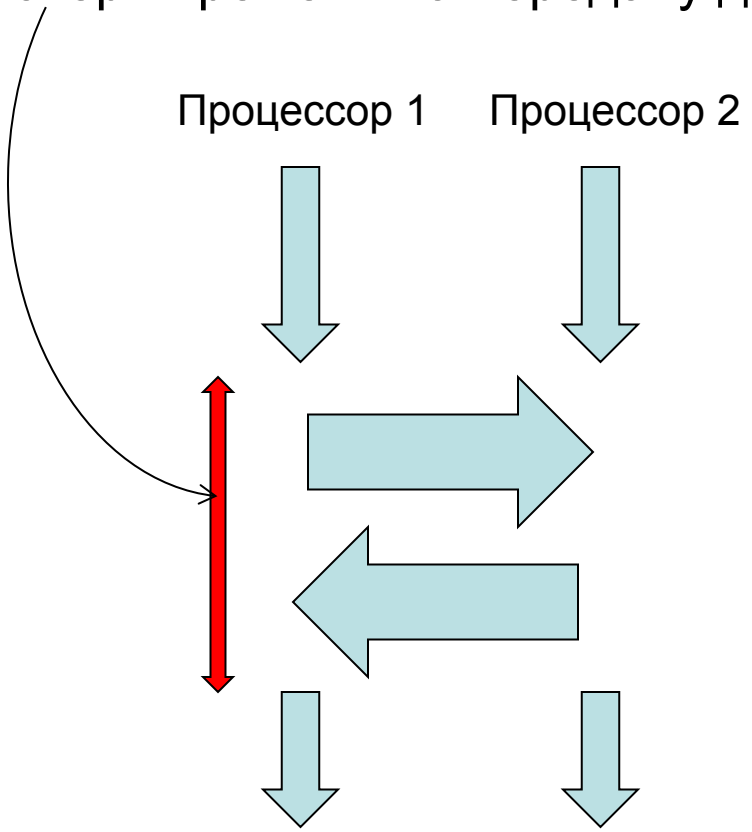
$$S_p \leq \frac{1}{\alpha}$$

Накладные расходы

- Операции, отсутствующие в наилучшем последовательном алгоритме:
  - Синхронизация
  - Обмен данными
  - Дублирование операций
  - Новые операции

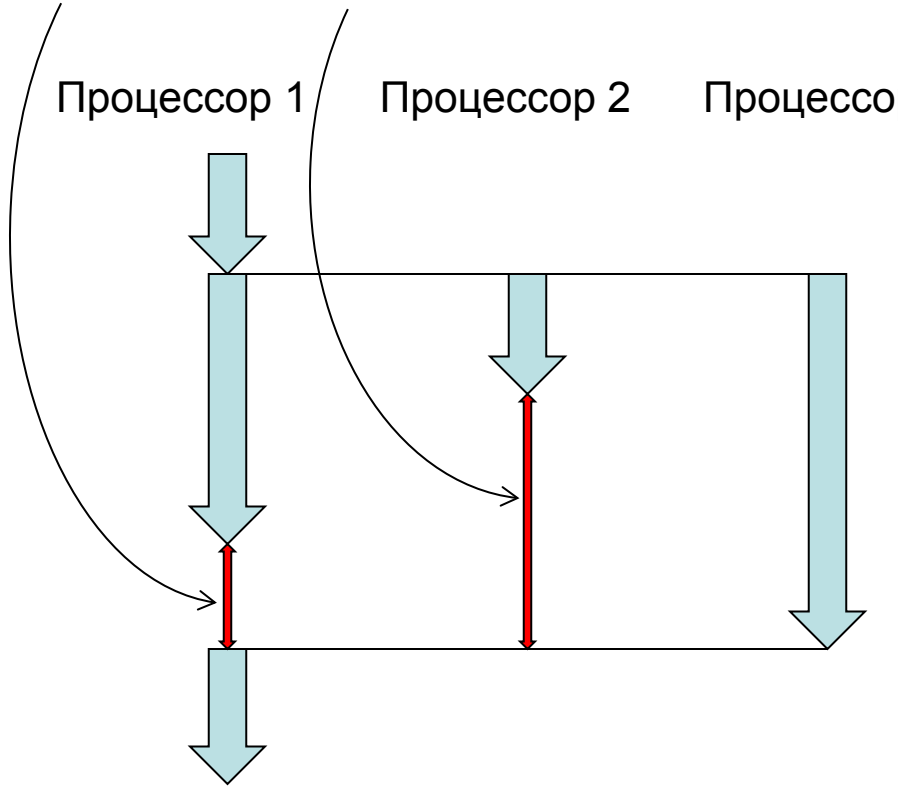
Обмен данными

- Потери времени на передачу данных между процессами

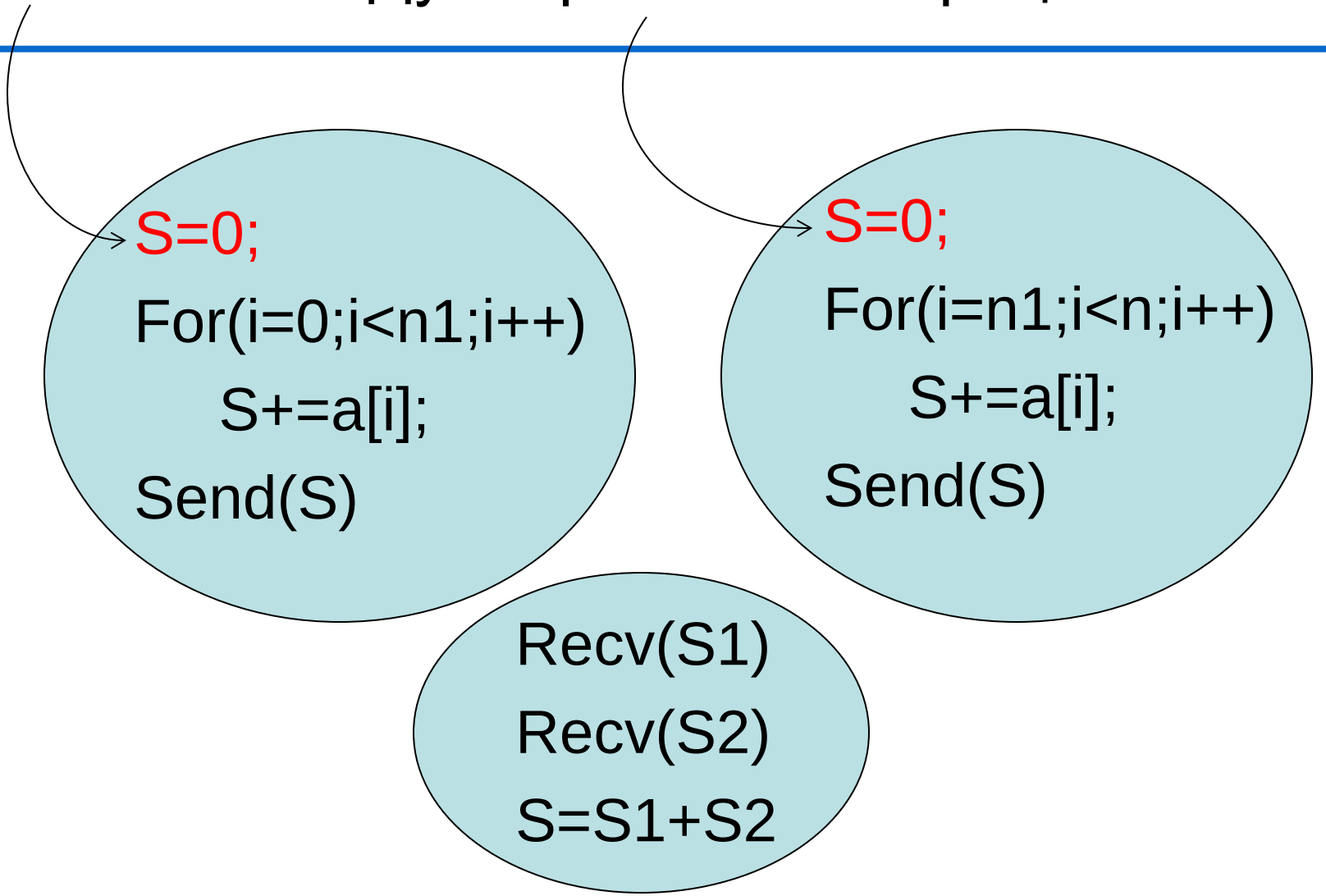


Синхронизация

- Потери времени на ожидание долго выполняющихся процессов



Дублирование операций



Новые операции

Шаг	Процессор 1	Процессор 2	Процессор 3	Процессор 4
1	1·2	3·4	5·6	7·8
2	12·3	12·34	56·7	56·78
3	1234·5	1234·56	1234·567	1234·5678

$$T_{p=n/2}(n) = \tau_c \log_2 n \qquad S = \frac{n-1}{\log_2 n}\Big|_{\substack{n=8\\p=4}} = \frac{7}{3} < 4 = p \qquad E_{p=4}(n=8) = \frac{7}{12}$$

Шаг	Процессор 1	Процессор 2	Процессор 3	Процессор 4
1	1 2!	8 3·4	9 5·6	10 7·8
2	2 3!	3 4!	11 56·7	12 56·78
3	4 5!	5 6!	6 7!	7 8!

t_d - дополнительные расходы времени

Операции, отсутствующие в наилучшем последовательном алгоритме:

- Синхронизация
- Обмен данными
- Дублирование операций
- Новые операции

$$T_p = \alpha T_1 + (1 - \alpha)\frac{T_1}{p} + t_d$$

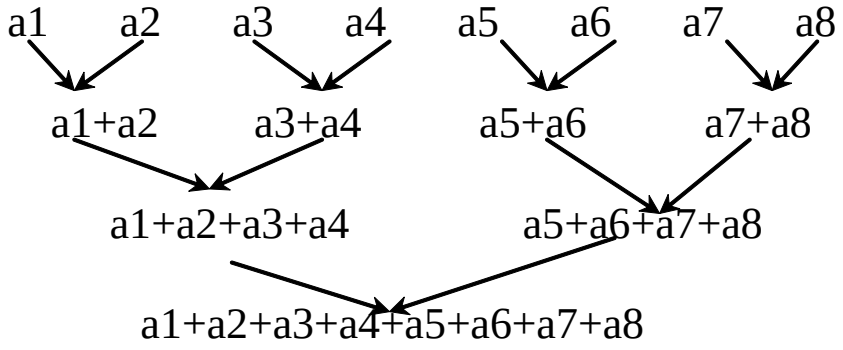
Принципы организации параллельных вычислений

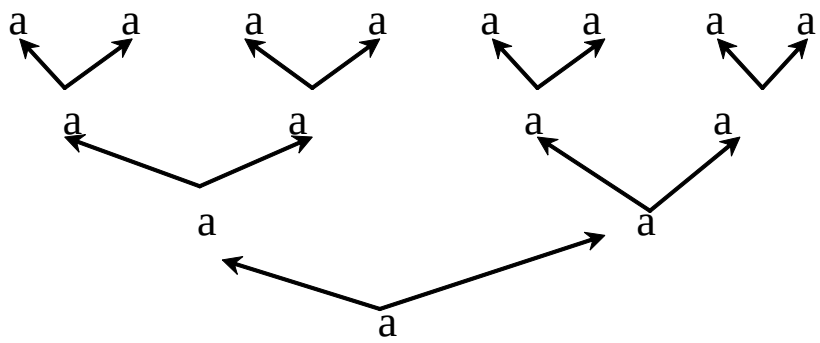
- Статическая балансировка загрузки
  - метод сдваивания
  - геометрический параллелизм
- Динамическая балансировка загрузки
  - Метод коллективного решения

Метод сдваивания

Выполнение редукционных и им подобных операций

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



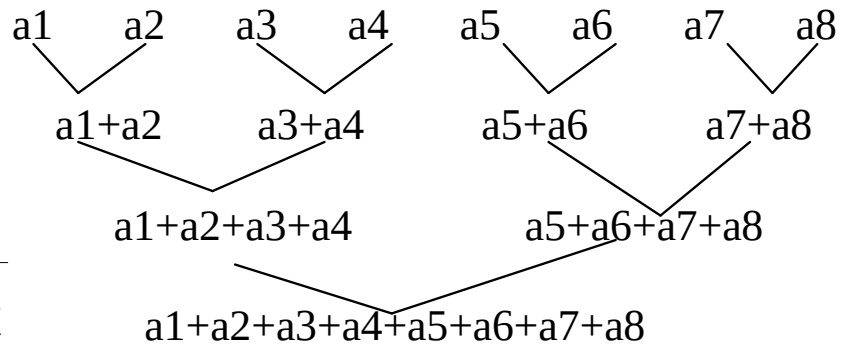


Метод сдваивания

- Каскадная схема

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

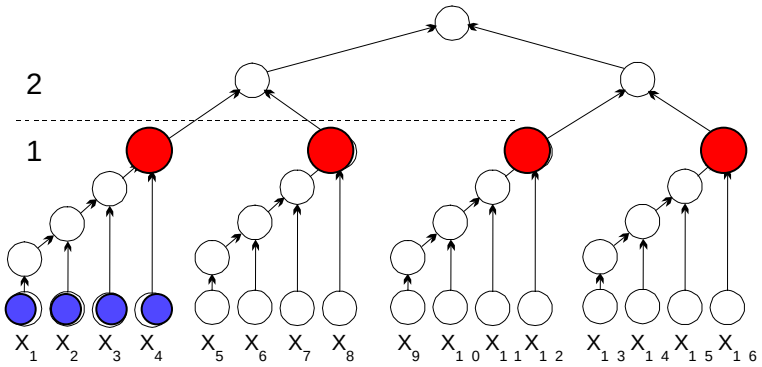
$$S_{p=n/2}(n) = \frac{(n-1)}{\log_2 n} \qquad E_{p=n/2}(n) \approx \frac{1}{\log_2 n}$$



- Модифицированная каскадная схема

$$T_{p=\frac{n}{\log_2 n}}(n) \approx 2\tau_c \log_2 n$$

$$S_{p=\frac{n}{\log_2 n}}(n) \approx \frac{(n-1)}{2\log_2 n} \qquad E_{p=\frac{n}{\log_2 n}}(n) \approx \frac{1}{2}$$



В чем причина низкой эффективности метода сдваивания?

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

$$E_{p=n/2}(n) \approx \frac{1}{\log_2 n}$$

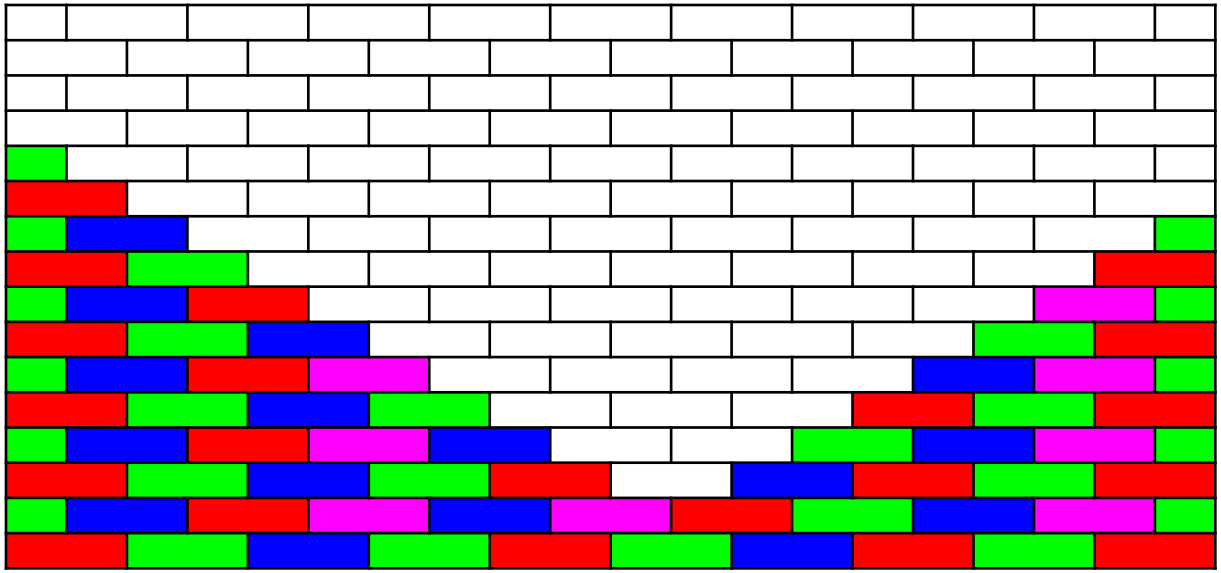
$$S_{p=n/2}(n) \approx \frac{n}{\log_2 n}$$

Метод геометрического параллелизма

Циклическая обработка локально связанных данных

- Обработка изображений
- Обработка данных, заданных на решетках или произвольных графах
- Моделирование физических процессов (течений жидкости и газов, теплопереноса, упругости, …)
- …

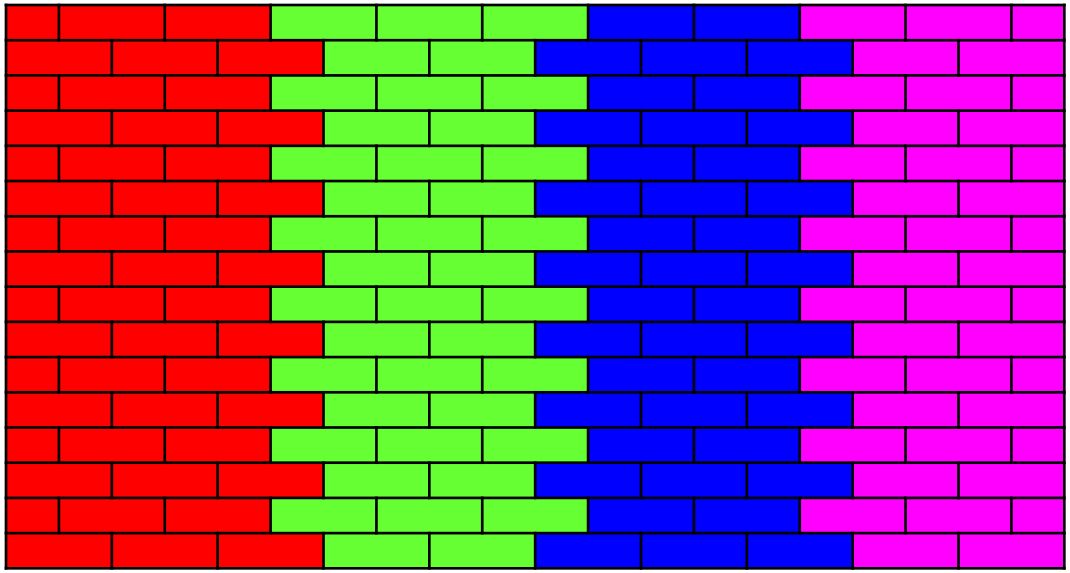
Пример модельной задачи: Стена Фокса



n – ширина стены

κ – высота стены

Метод геометрического параллелизма



$$T_1(kn) = \tau_c kn \qquad T_p(kn) = \tau_c \frac{kn}{p} + 4k\tau_s$$

$$S_p(kn) = p\frac{1}{1+4\frac{p}{n}\frac{\tau_s}{\tau_c}} \qquad E_p(kn) = \frac{1}{1+4\frac{p}{n}\frac{\tau_s}{\tau_c}}$$

А почему?

$$T_p(kn) = \tau_c \frac{kn}{p} + 4k\tau_s$$

```
for(шаг=0;шаг<k;шаг++)
{
   for(кирпич=rank*n/p;кирпич<(rank+1)*n/p;кирпич++)
     Уложить(кирпич)

   if(rank>0)    Send(rank-1, кирпич уложен! )
   if(rank<p-1)  Send(rank+1, кирпич уложен! )
   if(rank>0)    Recv(rank-1, место готово?  )
   if(rank<p-1)  Recv(rank+1, место готово?  )
}
```

$$T_p(kn) = \tau_c \frac{kn}{p} + 4k\tau_s$$

? Верно ли, что именно **4**?

```
for(шаг=0;шаг<k;шаг++)
{
    for(кирпич=rank*n/p;кирпич<(rank+1)*n/p;кирпич++)
        Уложить(кирпич)

    if(rank>0)    Send(rank-1, кирпич уложен! )
    if(rank<p-1)  Send(rank+1, кирпич уложен! )
    if(rank>0)    Recv(rank-1, место готово?  )
    if(rank<p-1)  Recv(rank+1, место готово?  )
}
```

0	1	2	3	4	5
	⇐	⇐	⇐	⇐	⇐
⇒	⇒	⇒	⇒	⇒	
	>=	>=	>=	>=	>=
=<	=<	=<	=<	=<	

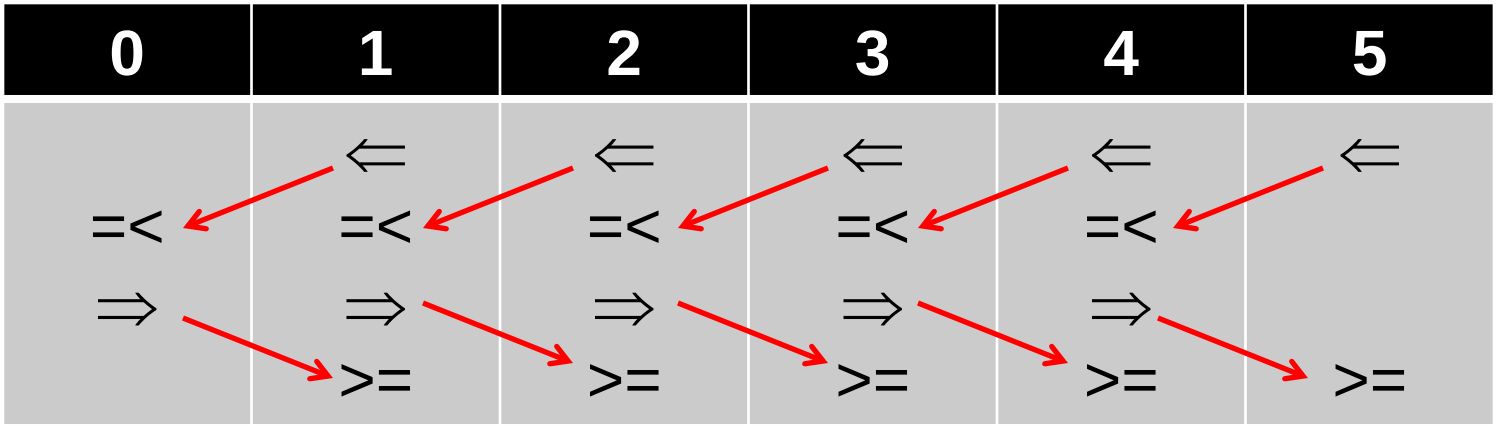
$$T_p(kn) = \tau_c \frac{kn}{p} + 4k\tau_s \leftarrow ?$$

```
for(шаг=0;шаг<k;шаг++)
{
    for(кирпич=rank*n/p;кирпич<(rank+1)*n/p;кирпич++)
        Уложить(кирпич)

    if(rank>0)    Send(rank-1, кирпич уложен! )
    if(rank<p-1)  Send(rank+1, кирпич уложен! )
    if(rank>0)    Recv(rank-1, место готово?  )
    if(rank<p-1)  Recv(rank+1, место готово?  )
}
```

0	1	2	3	4	5
	⇐	⇐	⇐	⇐	⇐
⇒	⇒	⇒	⇒	⇒	
	>=	>=	>=	>=	>=
=<	=<	=<	=<	=<	

$$T_p(kn) = \tau_c \frac{kn}{p} + 4k\tau_s$$?

```
for(шаг=0;шаг<k;шаг++)
{
    for(кирпич=rank*n/p;кирпич<(rank+1)*n/p;кирпич++)
        Уложить(кирпич)

    if(rank>0)    Send(rank-1, кирпич уложен! )
    if(rank<p-1)  Recv(rank+1, место готово?  )
    if(rank<p-1)  Send(rank+1, кирпич уложен! )
    if(rank>0)    Recv(rank-1, место готово?  )
}
```



$$T_p(kn) = \tau_c \frac{kn}{p} + 4k\tau_s$$?

0	1	2	3	4	5
=<	$\Leftarrow$	$\Leftarrow$	$\Leftarrow$	$\Leftarrow$	$\Leftarrow$
$\Rightarrow$	=<	=<	=<	=<	>=
	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	
	>=	>=	>=	>=	

$$T_p(kn) = \tau_c \frac{kn}{p} + 1k\tau_s$$

!

0	1	2	3	4	5
=<	⇐				
⇒	=<	⇐	⇐	⇐	⇐
	⇒	=<	=<	=<	>=
	>=	⇒	⇒	⇒	
		>=	>=	>=	

$$T_p(kn) = \tau_c \frac{kn}{p} + 2k\tau_s$$!

0	1	2	3	4	5
=<	⇐				
	=<	⇐			
⇒	⇒	=<	⇐	⇐	⇐
	>=	⇒	=<	=<	>=
		>=	⇒	⇒	
			>=	>=	

$$T_p(kn) = \tau_c \frac{kn}{p} + 3k\tau_s$$

!

0	1	2	3	4	5
=<	⇐				
	=<	⇐			
		=<	⇐		
⇒	⇒	⇒	=<	⇐	⇐
	>=	>=	⇒	=<	>=
			>=	⇒	
				>=	

$$T_p(kn) = \tau_c \frac{kn}{p} + 4k\tau_s$$!

0	1	2	3	4	5
=<	⇐				
	=<	⇐			
		=<	⇐	⇐	
			=<	=<	⇐
⇒	⇒	⇒	⇒	⇒	>=
	>=	>=	>=	>=	

$$T_p(kn) = \tau_c \frac{kn}{p} + pk\tau_s$$

!

0	1	2	3	4	5
=<	⇐	⇐	⇐	⇐	⇐
	=<	=<	=<	=<	>=
⇒	⇒	⇒	⇒	⇒	
	>=	>=	>=	>=	

$$T_p(kn) = \tau_c \frac{kn}{p} + (p+1)k\tau_s$$!

0	1	2	3	4	5
=<	⇐	⇐	⇐	⇐	⇐
	=<	=<	=<	=<	>=
⇒	⇒	⇒	⇒	⇒	
	>=	>=	>=	>=	

$$T_p(kn) = \tau_c \frac{kn}{p} + (p+2)k\tau_s$$!

0	1	2	3	4	5
=<	⇐	⇐	⇐	⇐	⇐
	=<	=<	=<	=<	>=
⇒	⇒	⇒	⇒	⇒	
	>=	>=	>=	>=	

$$T_p(kn) = \tau_c \frac{kn}{p} + (p+3)k\tau_s$$!

0	1	2	3	4	5
=<	⇐	⇐	⇐	⇐	⇐
	=<	=<	=<	=<	>=
				⇒	
⇒	⇒	⇒	⇒	>=	
	>=	>=	>=		

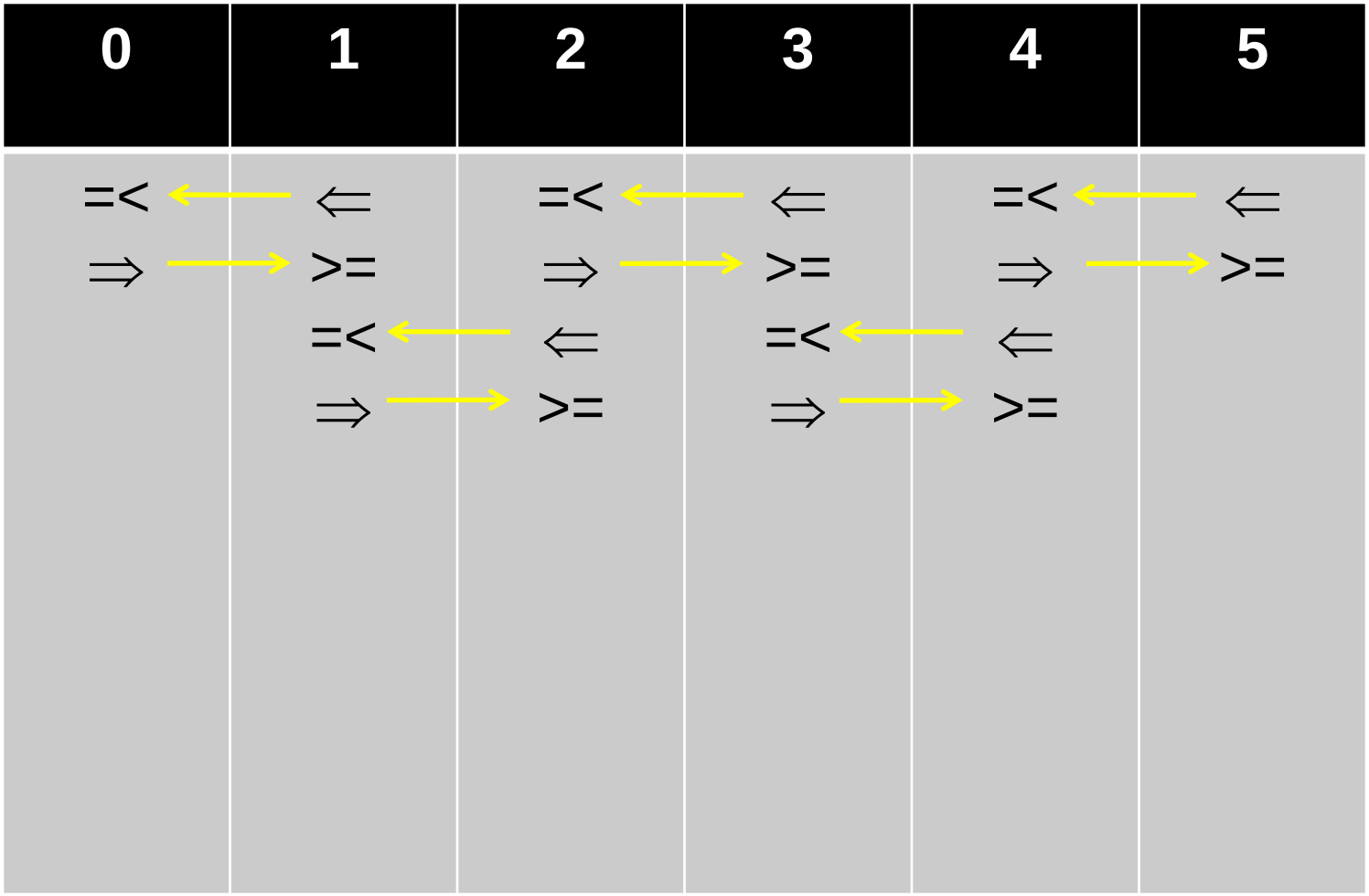
$$T_p(kn) = \tau_c \frac{kn}{p} + (p+4)k\tau_s$$!

0	1	2	3	4	5
=<	⇐	⇐	⇐	⇐	⇐
	=<	=<	=<	=<	>=
				⇒	
			⇒	>=	
		⇒	>=		
	⇒	>=			
⇒	>=				

$$T_p(kn) = \tau_c \frac{kn}{p} + 2pk\tau_s$$!

0	1	2	3	4	5
=<	⇐	⇐	⇐	⇐	⇐
	=<	=<	=<	=<	>=
				⇒	
			⇒	>=	
		⇒	>=		
	⇒	>=			
⇒	>=				

$$T_p(kn) = \tau_c \frac{kn}{p} + 4k\tau_s$$!



$$T_p(kn) = \tau_c \frac{kn}{p} + 4k\tau_s$$!

```
for(шаг=0;шаг<k;шаг++)
{
    for(кирпич=rank*n/p;кирпич<(rank+1)*n/p;кирпич++)
       Уложить(кирпич)

    if(rank%2)
    {
    if(rank>0)    Send(rank-1, кирпич уложен! )
    if(rank>0)    Recv(rank-1, место готово?  )
    if(rank<p-1)  Recv(rank+1, место готово?  )
    if(rank<p-1)  Send(rank+1, кирпич уложен! )
    }
    else
    {
    if(rank<p-1)  Recv(rank+1, место готово?  )
    if(rank<p-1)  Send(rank+1, кирпич уложен! )
    if(rank>0)    Send(rank-1, кирпич уложен! )
    if(rank>0)    Recv(rank-1, место готово?  )
    }
}
```

Метод коллективного решения

Решение множества независимых друг от друга задач:

- Табулирование функций
- Решение задач методами Монте-Карло
- Численное интегрирование многомерных функций
- …

Последовательность Фибоначчи

Для заданного $n > 1$ требуется определить значение элемента n последовательности Фибоначчи

$$F(n) = F(n-1) + F(n-2)$$
$$F(0) = 1 \text{ и } F(1) = 1$$

Последовательный алгоритм

```
F(n)
{
        a=1; b=1;  c=1;
        for(i=2;i<=n;i++)
                {
                c=a+b;
                b=a;
                }
        return c;
}
```

Потребуется операций

$$G_{A0}(n) = n - 1$$

A1: Рекурсия

```
F(n)
{
        if(n<2) return 1;
        return   F(n-2)+F(n-1);
}
```

Сколько потребуется операций?

A1: Рекурсия

```
F(n)
{
        if(n<2) return 1;
        return   F(n-2)+F(n-1);
}
```

Сколько потребуется операций?

$$G_{A1}(n) = F(n) - 1$$

Рекурсия

```
F(n)
{
        if(n<2) return 1;
        return(F(n-2)+F(n-1);
}
```

Потребуется операций:

$$G_{A1}(n) = F(n) - 1$$

Это много или мало?

Рекурсия: распараллелить можно, но …

```
F(n)
{
        if(n<2) return 1;
        return(F(n-2)+F(n-1);
}
```

Потребуется операций:

$$G_{A1}(n) = F(n) - 1$$

$$n \to \infty, \quad F_n \sim \frac{\varphi^n}{\sqrt{5}}, \varphi = \frac{1+\sqrt{5}}{2}$$

$$G_{A1}(n) \sim 0{,}45 \cdot 10^{n/4{,}78}$$

Параллельная рекурсия

```
F(n)
{
    if(n<2) return 1;
```

Процессор 1	Процессор 2
a= F(n-2)	*b= F(n-1)*

```
    return a+b;
}
```

Сколько времени потребуется на выполнение?

Быстрее, чем за *O(n)*

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \mathrm{A}^{\mathrm{n}}, \text{ где } \mathrm{A} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

- Сколько времени потребуется на выполнение?
- Сколько процессоров потребуется?

Быстрее, чем за *O(n)*

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \mathrm{A}^{\mathrm{n}}, \text{ где } \mathrm{A} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

$A^k = \left(A^{k/2}\right)^2$, при четном k

$A^k = \left(A^{(k-1)/2}\right)^2 A$, при нечетном k

A4: Последовательный матричный алгоритм

```
F(n) // n>1
{
  A[2][2]={ {1,1} , {1,0} };
```

$$k = 2^{\lfloor log_2 n \rfloor - 1}$$

```
 B=A;
 for( i=k ; i ; i/=2 )
     {
          B=B*B;
          if(i&n) // если в числе n двоичный разряд с номером log2 i равен 1
                    B=B*A;
     }

  return B[0][1];
}
```

$$G_{A4}(n) = O(log_2 n)$$

A5: Параллельный матричный алгоритм

Пусть $B = \begin{pmatrix} p & q \\ q & r \end{pmatrix}$, тогда: $B^2 A^\beta = \begin{pmatrix} p & q \\ q & r \end{pmatrix}^2 \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^\beta$.

$$B^2 A^\beta = \begin{cases} \begin{pmatrix} \bar{p} & \bar{q} \\ \bar{q} & \bar{r} \end{pmatrix}, \text{при } \beta = 0 \\ \begin{pmatrix} \bar{\bar{p}} & \bar{\bar{q}} \\ \bar{\bar{q}} & \bar{\bar{r}} \end{pmatrix}, \text{при } \beta = 1 \end{cases}, \quad \begin{matrix} \bar{p} = p^2 + q^2 \\ \bar{q} = q(p + r) \\ \bar{r} = q^2 + r^2 \end{matrix} \quad \begin{matrix} \bar{\bar{p}} = \bar{p} + \bar{q} \\ \bar{\bar{q}} = \bar{p} \\ \bar{\bar{r}} = \bar{q} \end{matrix}$$

Для вычисления $\bar{p}$, $\bar{q}$, $\bar{r}$ и $\bar{\bar{p}}$ потребуются три такта (при $\beta = 1$)

```
F(n)
{
    p=1;   q=1;  r=0; // B = (p q; q r) = (1 1; 1 0)
    for(i=1;i<n;i*=2)
    {
```

	Процессор 1	Процессор 2	Процессор 3	Процессор 4
Такт 1	$p_p = p^2$	$q_q = q^2$	$r_r = r^2$	$p_r = p + r$
Такт 2	$\bar{p} = p_p + q_q$	$\bar{r} = q_q + r_r$	$\bar{q} = q \cdot p_r$	
Такт 3	$\bar{\bar{p}} = \bar{p} + \bar{q}$			

```
        if(i&n) // если в числе n двоичный разряд с номером log2 i равен 1
        { p=p̄;   q=q̄;   r=r̄  }
        else { p=p̿;   q=p̄;   r=q̄ }
    }
    return q;
}
```

$$G_{A5}(n) = 3\log_2 n$$

Для вычисления $\bar{p}$, $\bar{q}$, $\bar{r}$ и $\bar{\bar{p}}$ потребуются три такта (при $\beta = 1$)

```
F(n)
{
    p=1;   q=1;  r=0; // B = (p q; q r) = (1 1; 1 0)
    for(i=1;i<n;i*=2)
    {
```

	Процессор 1	Процессор 2	Процессор 3	Процессор 4
Такт 1	$p_p = p^2$	$q_q = q^2$	$r_r = r^2$	$p_r = p + r$
Такт 2	$\bar{p} = p_p + q_q$	$\bar{r} = q_q + r_r$	$\bar{q} = q \cdot p_r$	
Такт 3	$\bar{\bar{p}} = \bar{p} + \bar{q}$			

```
        if(i&n) // если в числе n двоичный разряд с номером log2 i равен 1
        {  p=p̄;    q=q̄;    r=r̄   }
        else {     p=p̿;    q=p̄;    r=q̄  }
    }
    return q;
}
```

$$G_{A5}(n) = 3\log_2 n$$

$$G_{A4}(n) = 8\log_2 n$$

Заключение

- Отмечена важность использования логически простых алгоритмов
- Рассмотрены основные причины потерь времени при выполнении параллельных алгоритмов
- Рассмотрены методы сдваивания и геометрического параллелизма, относящиеся к классу методов статической балансировки загрузки процессоров

Якобовский М.В.

проф., д.ф.-м.н.,

зав. сектором «Программного обеспечения вычислительных систем и сетей»

Института прикладной математики им. М.В.Келдыша Российской академии наук

mail: lira@imamod.ru

web: http://lira.imamod.ru

© Якобовский М.В.