Параллельные вычисления (часть 1)

Якобовский Михаил Владимирович

Простые параллельные алгоритмы

Статическая балансировка загрузки процессоров

- метод сдваивания
- геометрический параллелизм
- конвейерный параллелизм

Динамическая балансировка загрузки процессоров

- 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

- доля операций выполняемых последовательно

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

α - доля последовательно выполняемых и дублированных операций

t_d - дополнительные расходы времени

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

Накладные расходы

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

Последовательное выполнение и дублирование операций

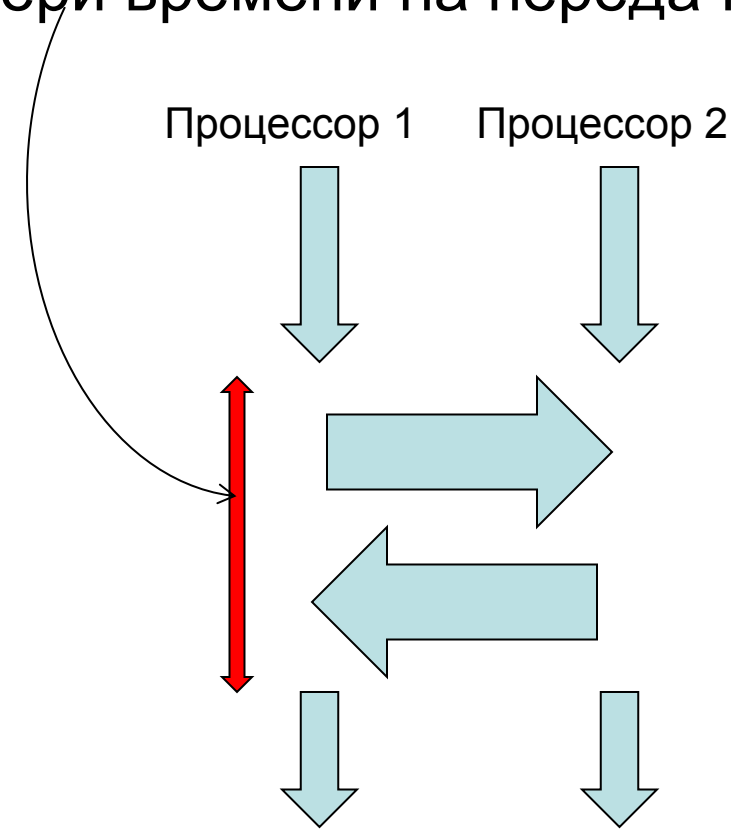
```
S=0;
For(i=0;i<n1;i++)
    S+=a[i];
Send(S)
```

```
S=0;
For(i=n1;i<n;i++)
    S+=a[i];
Send(S)
```

```
Recv(S1)
Recv(S2)
S=S1+S2
```

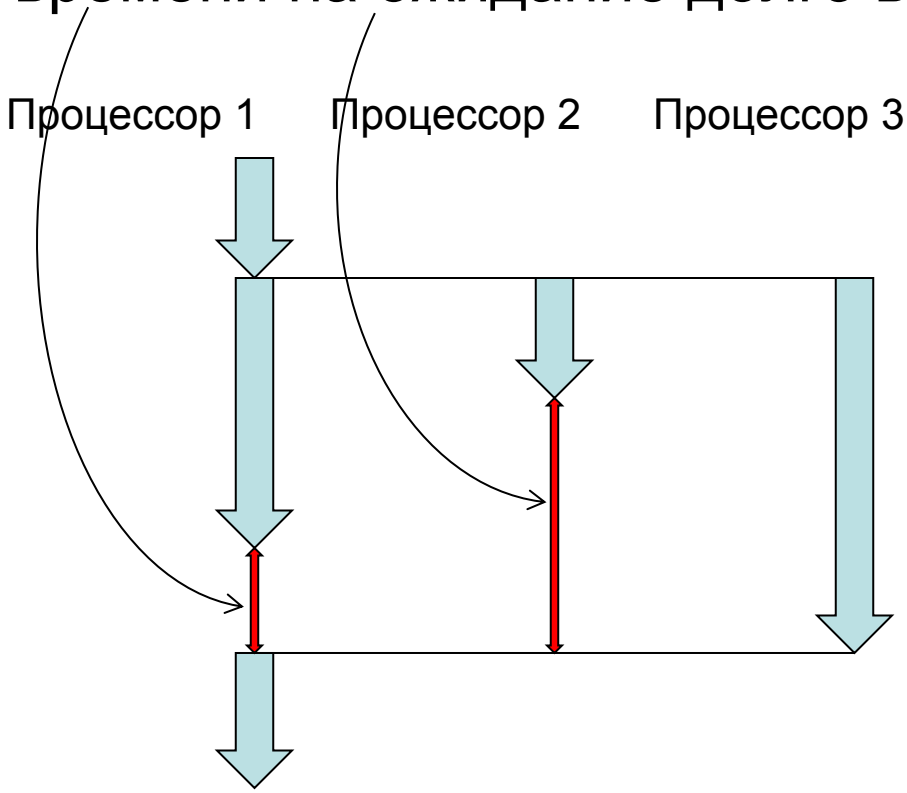
Обмен данными

Потери времени на передачу данных между процессами



Синхронизация

Потери времени на ожидание долго выполняющихся процессов



Новые операции

Вычисление всех факториалов до 4! включительно

Шаг	*Процессор 1*			
1	*2!=1·2*			
2	*3!= 2!·3*			
3	*4!=3!·4*			

Новые операции

Вычисление всех факториалов до 4! включительно

Шаг	*Процессор 1*			
1	*2!=1·2*			
2	*3!= 12·3*			
3	*4!=123·4*			

Новые операции

Вычисление всех факториалов до 4! включительно

Шаг	*Процессор 1*	*Процессор 2*	*Процессор 3*	*Процессор 4*
1	$2!=1\cdot2$			
2	$3!=1\cdot2\cdot3$	$4!=1\cdot2\cdot3\cdot4$		
3				

Новые операции

Вычисление всех факториалов до 4! включительно

Шаг	*Процессор 1*	*Процессор 2*	*Процессор 3*	*Процессор 4*
1	$2!=1\cdot2$			
2	$3!=(2!)\cdot(3)$	$4!=(1\cdot2)\cdot(3\cdot4)$		
3				

Шаг	*Процессор 1*	*Процессор 2*	*Процессор 3*	*Процессор 4*
1	$2!=1\cdot2$			
2	$3!=12\cdot3$	$4!=12\cdot34$		
3				

Новые операции

Вычисление всех факториалов до 4! включительно

Шаг	*Процессор 1*	*Процессор 2*	*Процессор 3*	*Процессор 4*
1	*1·2*	*3·4*		
2	*12·3*	*12·34*		
3				

Почему эффективность <1, если нет простаивающих процессоров?

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

```
F=1;
for(i=2;i <= n;i++)
        F*=i;
```

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

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

$$S = \frac{n-1}{\log_2 n}\Big|_{\substack{n=8\\p=4}} = \frac{7}{3} = 2.33(3) < 4 = p$$

$$E_{p=4}(n=8) = \frac{7}{12}$$

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*

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

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!*

Дополнительные расходы времени t_d

Операции, отсутствующие в наилучшем последовательном алгоритме:

- Синхронизация
- Обмен данными
- Новые операции

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

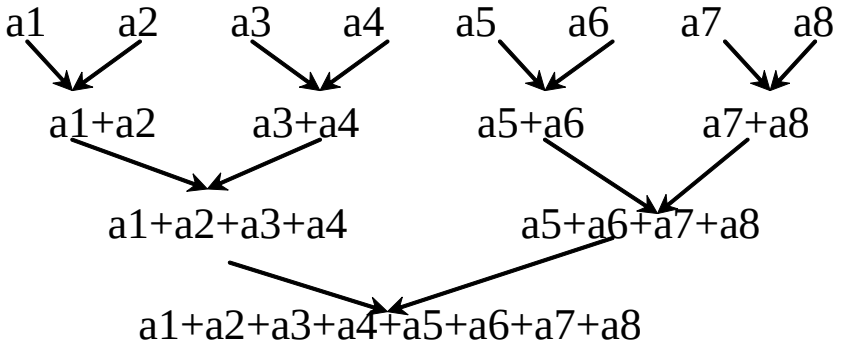
Принципы организации параллельных вычислений

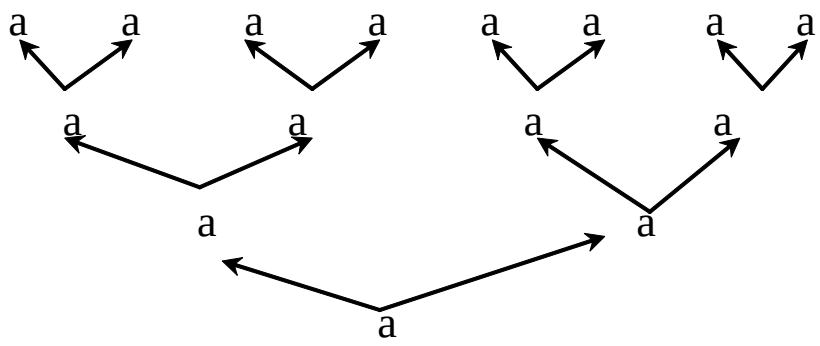
- Статическая балансировка загрузки
  - метод сдваивания
  - геометрический параллелизм
- Динамическая балансировка загрузки
  - Метод коллективного решения

Метод сдваивания

Выполнение редукционных и им подобных операций

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



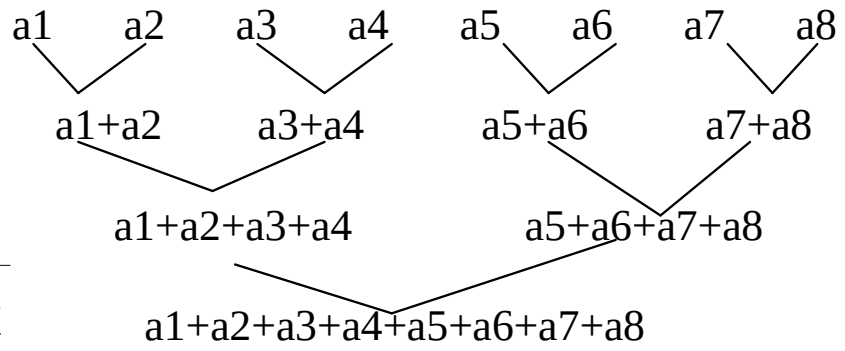


Метод сдваивания

- Каскадная схема

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

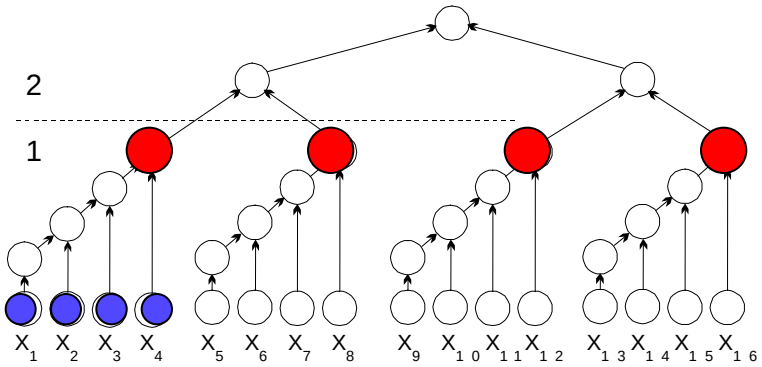
$$S_{p=n/2}(n) = \frac{(n-1)}{\log_2 n} \qquad E_{p=n/2}(n) \approx \frac{1}{\log_2 n}$$



- 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 сдваивания является методом:

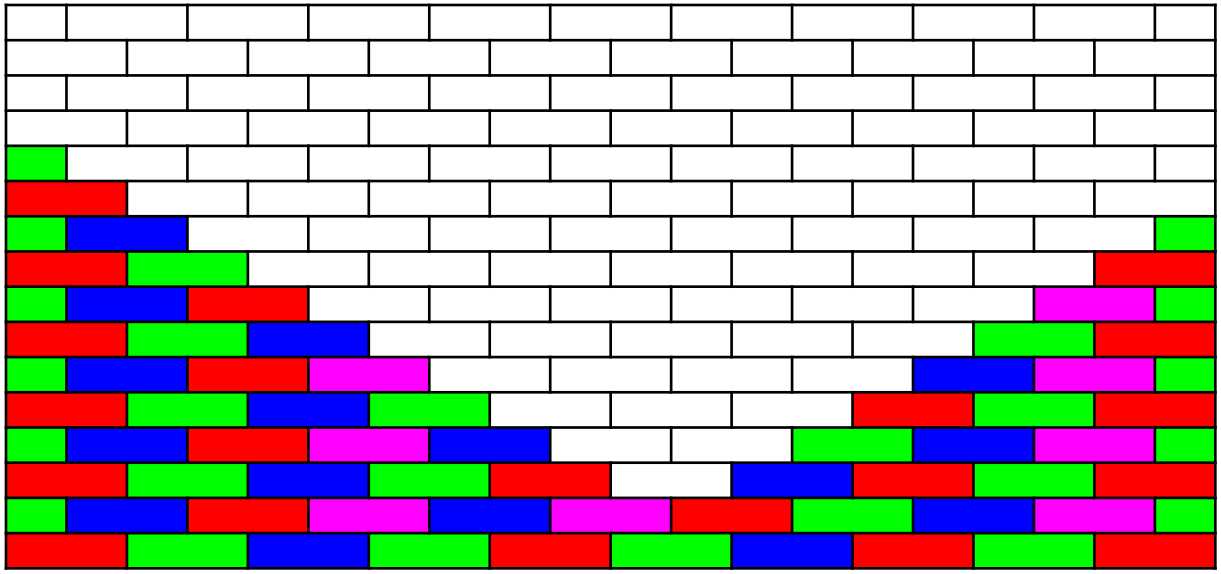
- Статической балансировки загрузки
- Динамической балансировки загрузки

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, …)
- …

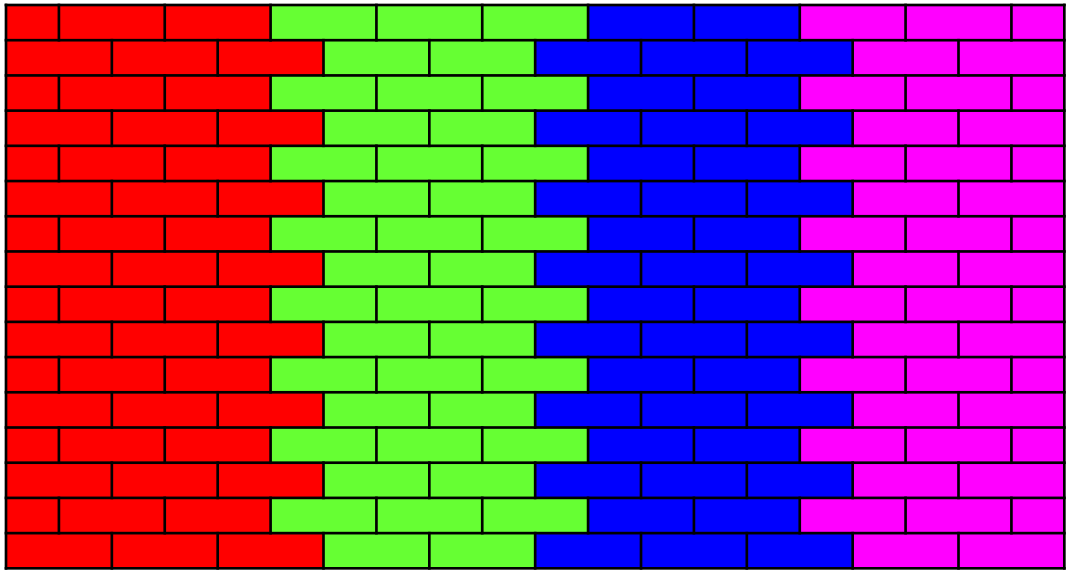
Пример модельной задачи: Стена Фокса



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
	⇐	⇐	⇐	⇐	⇐
⇒	⇒	⇒	⇒	⇒	
	>=	>=	>=	>=	>=
=<	=<	=<	=<	=<	

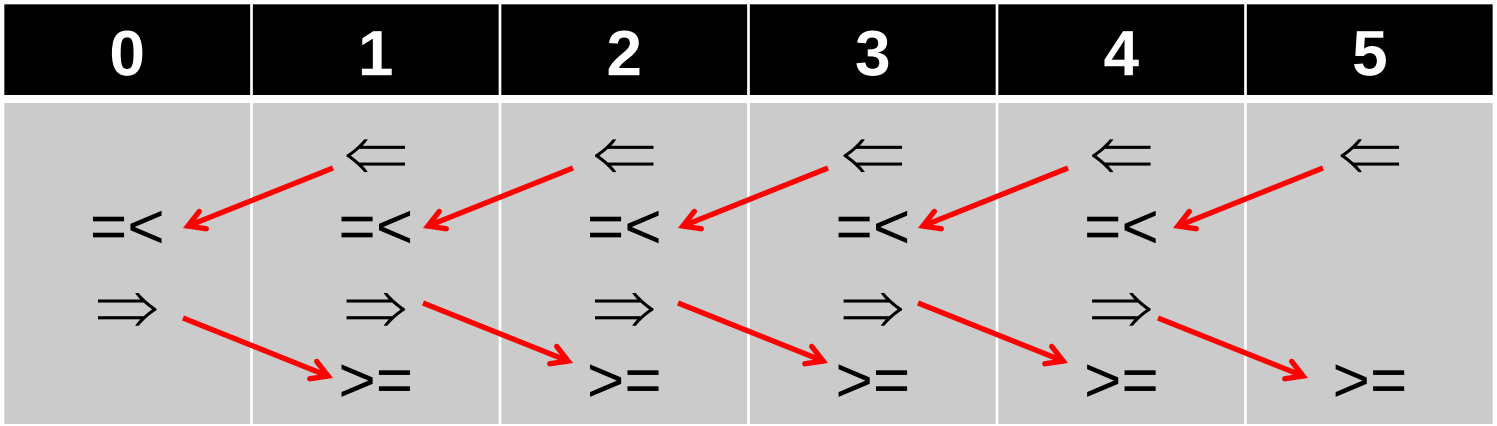
$$T_p(kn) = \tau_c \frac{kn}{p} + 4k\tau_s$$

```
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
=<	⇐	⇐	⇐	⇐	⇐
⇒	=<	=<	=<	=<	>=
	⇒	⇒	⇒	⇒	
	>=	>=	>=	>=	

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

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
=<	⇐	⇐	⇐	⇐	⇐
	=<	=<	=<	=<	>=
				⇒	
⇒	⇒	⇒	⇒	>=	
	>=	>=	>=		

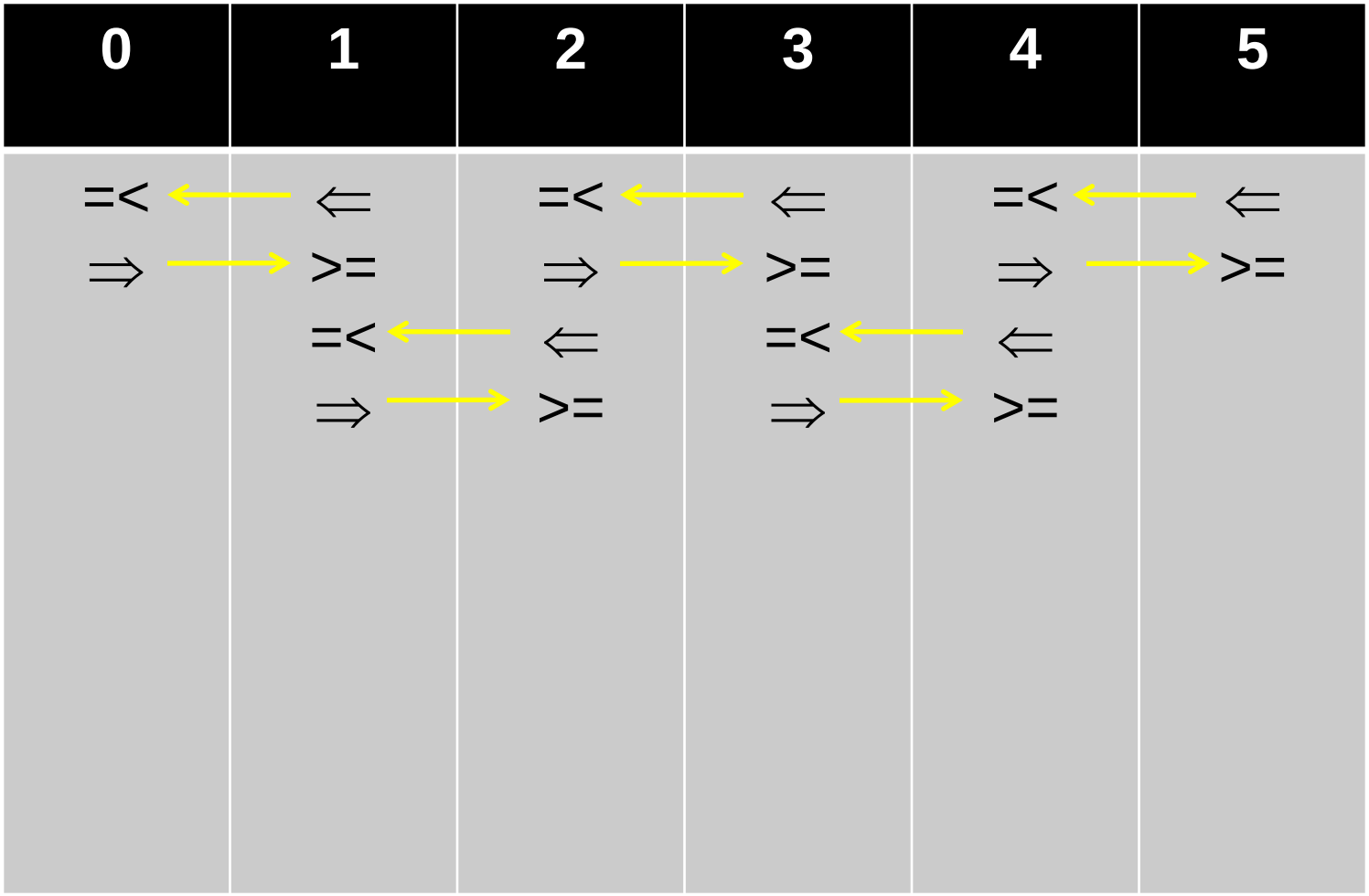
$$T_p(kn) = \tau_c \frac{kn}{p} + (p+4)k\tau_s$$

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 геометрического параллелизма является методом:

- Статической балансировки загрузки
- Динамической балансировки загрузки

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