

Лекция 5. Принципы построения параллельных алгоритмов. Виды параллелизма.

1. Параллельные вычисления как взаимодействие последовательных процессов.

Вычисления - последовательный процесс выполнения арифметических или логических действий.

Параллельные вычисления – одновременное выполнение нескольких последовательных вычислений, результаты которых используются для получения общего результата. => **Параллельные вычисления есть взаимодействие последовательных процессов вычислений** (см. Хоар, Дейкстра).

В случае последовательного алгоритма необходимо доказательство его конечности и правильности (корректности результата), а также его корректная реализация на выбранном языке программирования. Даже при соблюдении этих условий имеется ограничение по входным и выходным данным, которое делает исполнение программы либо невозможным, либо приводит к неверным результатам.

В случае параллельного алгоритма не существует формального способа доказательства его правильности. Для конкретной его реализации в виде программы на некотором языке тем более. Не помогает никакая система тестов. Причина неадекватного поведения параллельных программ – в **неопределенности** в порядке выполнения действий, необходимых для получения результата. Каждый параллельный процесс "живет в своем мире по своим часам". Следствие неопределенности – образование **тупиков** (программа ничего не делает) и выдачи **неверных результатов**.

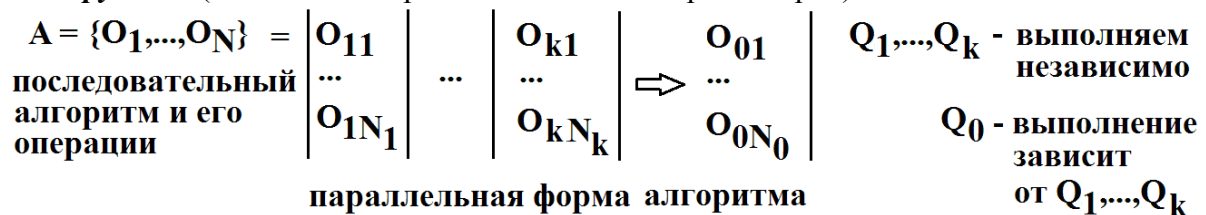
Тем не менее, существуют способы разработки параллельных алгоритмов и программ, позволяющие на практике решать поставленные задачи. Их суть – ограничение в свободе выбора структуры алгоритма.

2. Виды параллелизма.

Параллелизм – 1) **свойство последовательного алгоритма** совершать независимо все или некоторые действия; 2) **способ преобразования** последовательного алгоритма в параллельный.

Рассмотрим второе определение. А именно виды параллелизма, которые успешно применяются:

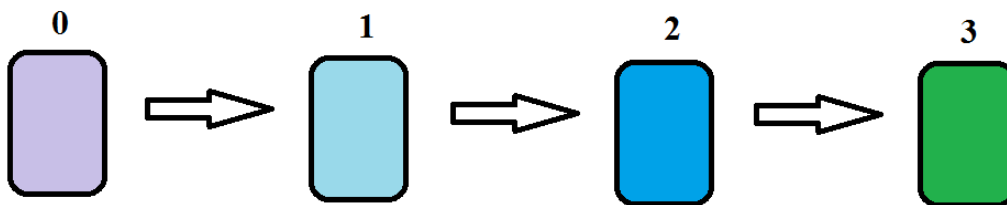
1) **Алгоритмический (внутренний, функциональный)** – разбиение последовательного алгоритма на отдельные независимые части. Здесь число параллельных ветвей параллельного алгоритма не превосходит числа независимых частей исходного алгоритма, т.е. в отличие от следующих этот способ **не масштабируемый** (нельзя взять произвольное число процессоров).



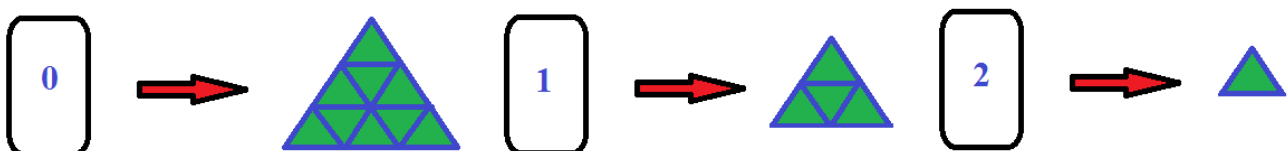
2) **Геометрический (координатный)** – разбиение последовательного алгоритма на группы однородных действий над однотипными данными, при котором связи между группами отсутствуют или минимальны. Последнее свойство называют свойством **локальности** алгоритма. Локальный алгоритм допускает разбиение данных на части соответственно числу процессоров, т.е. **масштабируем**.

$$\text{СУММА}(1:1000) = \text{СУММА}(1:500) + \text{СУММА}(501:1000)$$

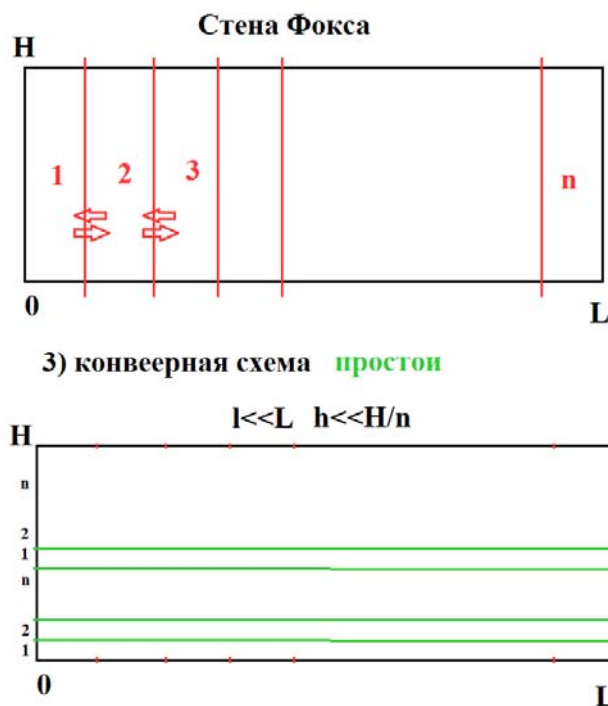
3) **Конвейерный** – объединение различных операций последовательного алгоритма в конвейер.



4) **Коллективное решение (процессорная ферма)** – разбиение последовательного алгоритма на элементарные фрагменты (кванты), которые выполняются не одинаковыми по производительности вычислителями.



Пример: **Стена Фокса** (длиной L и высотой H). Ее складывают n каменщиков. Работа – заливка раствора и укладка кирпичей.



n каменщиков

1) функциональная схема

2 функции: заливка раствора - m
укладка кирпичей - k
 $n = m + k$

2) геометрическая схема

проблема согласования кладки



h - высота К l - длина К

$l \ll L/n$

4) коллективное решение

квант работы - укладка
штабеля и заливка ведра
раствора

Алгоритмический параллелизм – два каменщика, один заливает раствор, другой кладет кирпичи (**разбиение по типу работ**). Способ не эффективный из-за ограничения на число работников. Если $n = m + k$ работников, m – заливщики, $k \ll m$ – каменщики – **неэффективное решение**.

Геометрический параллелизм – вся стена разбита на одинаковые участки по длине, каждый каменщик кладет свой участок и общается только с соседями на стыках слоев (**разбиение по x**). Способ эффективен, когда длина участков велика. При коротких участках возникает проблема ожидания друг друга на стыковке слоев.

Конвеерный параллелизм – распределение работы каменщиков по слоям: каждый делает один или несколько целых слоев стены (**разбиение по y**). Способ предполагает простой работников. Эффективен, если стена достаточно высокая и длинная, а все каменщики работают с одинаковой скоростью. Темп работы зависит от самого медленного из них.

Коллективное решение – каждому каменщику выдается некоторое количество кирпичей и раствора; каждый кладет стену в любом готовом к кладке и не занятом другими месте, пока не закончатся материалы; затем снова берет порцию материалов и продолжает процесс (**квантование работ**). Требуется синхронизация при получении материалов и выборе участков работ \Rightarrow есть простои. Эффективен при большом объеме работ (число квантов работы \gg числа работников) и быстрой раздаче материалов и свободных участков.

3. Теоретическое ускорение и эффективность параллельных алгоритмов.

При разработке паралл. алгоритмов (ПА) и оценке их эффективности примем следующие **соглашения**:

а) ВС состоит из p универсальных одинаковых по производительности вычислителей;

б) решаем только большие задачи, которые обеспечивают максимальную загруженность вычислителей.

При этих условиях решить задачу на всей ВС нельзя быстрее, чем в p раз, по сравнению с ее решением на одном вычислителе.

Задача программиста – добиться сокращения времени решения задачи. Однако это достигается часто неэффективным способом.

Причины неэффективности: 1) неэффективность алгоритма; 2) межпроцессорные обмены; 3) доп. действия, связанные с параллелизацией.

При разработке ПА необходимо **минимизировать** все три фактора.

Основная характеристика алгоритма – **степень параллелизма d** – число действий, которые могут выполняться одновременно, каждое на своем вычислителе.

Примеры: 1) $c_i = a_i + b_i$ ($i = 1, \dots, N$): степень параллелизма $d = N$; 2) $c_0 = 0$, $c_i = c_{i-1} + a_i$ ($i = 1, \dots, N$): степень параллелизма $d = 1$. Степень параллелизма не зависит от числа вычислителей, это внутреннее свойство алгоритма. Ускорить алгоритм более чем в d раз нельзя $\Rightarrow p \leq d$.

Параллельный алгоритм определим как последовательность алгоритмов $\{A_k\}$ ($k=1,2,\dots,p,\dots$).

Три класса ПА: 1) $\{A_1, 0, 0, \dots\}$ – вырожденный ПА; 2) $\{A_1, A_2, \dots, A_k, 0, 0, \dots\}$ – ограниченный параллелизм;

3) $\{A_1, A_2, \dots, A_k, \dots, A_{2k}, \dots\}$ – неограниченный параллелизм.

В соответствии с соглашениями: $S_k = T_1/T_k$ – ускорение, $E_k = S_k/k$ или $(S_k/k)*100\%$ – эффективность для p -го члена последовательности, T_1, T_k – оценки времен.

4. Примеры оценки ускорения и эффективности ПА.

Пример 1: вычисление суммы чисел $s = a_1 + \dots + a_N$;

$A_1: s = a_1, s = s + a_i, i = 2, \dots, N$. – последовательный алгоритм. Нужен параллельный вариант.

1) **ПА по методу сдваивания:** $s = ((a_1 + a_2) + (a_3 + a_4)) + ((a_5 + a_6) + (a_7 + a_8))$.

Если $N = 2^q \Rightarrow q = \log_2 N$ этапов, степень параллелизма $d_k = N/2^k$ на каждом этапе сложения. Пусть число процессоров $p = N/2$ (равно макс. степени параллелизма алгоритма сдваивания). Пусть t – время сложения и инициализации, at – время обмена. Посчитаем: $S_p = tN / [2t * \log_2 N + 2at * (\log_2 N - 1)] = p / [(1 + \alpha) * \log_2 p + 1]$, $E_p = 1 / [(1 + \alpha) * \log_2 p + 1]$.

Почему: на каждом этапе каждый вычислитель выполняет: инициализацию и сложение \Rightarrow один из процессоров выполнит $2tq = 2t \log_2 N$ действий. Передача двух чисел - $2at$, время всех обменов $2at * (\log_2 N - 1)$.

Анализ: если α велико, то никакой эффективности, если $\alpha = 0$, то эффективность не более 50% (достигается на 2 проц.). Почему? Плохой метод в основе ПА.

2) **ПА на основе геометрического параллелизма.** Разобьем массив на p частей. На каждом вычислителе выполним сложение $m = N/p$ чисел обычным образом, а затем сложим p чисел на одном из вычислителей: $s = (a_1 + a_2) + (a_3 + a_4) + (a_5 + a_6) + (a_7 + a_8)$. $\Rightarrow S_p = tN / [tN/p + tp + atp] = p / [1 + (1 + \alpha)p^2/N]$, $E_p = 1 / [1 + (1 + \alpha)p^2/N]$.

Анализ: Можно достичь высокой эффективности $p \ll N$. С ростом числа вычислителей эффективность падает, т.к. размер задачи фиксирован.

Вывод: последовательности ПА могут быть разные. Часто хороший последовательный алгоритм имеет малую степень параллелизма и приводит к плохому ПА. И наоборот, плохой последовательный алгоритм хорошо "параллелится". Поэтому вводят понятия: **ускорение и эффективность ПА по сравнению с наилучшим последовательным**. Можно говорить также о наилучшей последовательности параллельных алгоритмов и методах построения. Для оценки конкретной последовательности алгоритмов будем использовать 2-ой закон Амдала в сделанных предположениях:

$S_p = T_1/T_p = T_1 / [\beta T_1 + (1 - \beta) T_1/p + T_d] = 1 / [\beta + (1 - \beta)/p + T_d/T_1]$, β – доля непараллельных операций, $T_d = T_d(p)$ – время на обмены и дополнительные действия. Характерные ситуации:

1) $\beta = 0, T_d = 0 \Rightarrow S_p = p$ – идеальный случай;

2) $\beta > 0, T_d = 0 \Rightarrow S_p = p / [1 + \beta(p - 1)] < p$ ($\beta = 1/2 \Rightarrow S_p < 2$) – ограниченность ускорения;

3) $\beta > 0, T_d > 0, T_d \gg T_1 \Rightarrow S_p < 1$ – при больших накладных расходах не имеет смысла параллелить.

Пример 2: вычисление $e^x \sim s(x) = 1 + x/1! + x^2/2! + \dots + x^N/N!$, $N > 1$.

1) **Прямая схема** - N сложений, $N-1$ делений, $N*(N-2)+1$ умножений ($x^k = x * x * \dots * x \rightarrow k-1, k! = 1 * 2 * \dots * k \rightarrow k-2$);

main: $s = 1 + x$; for ($k=2; k \leq N; k++$) { $p = f(x, k); s = s + p$ }; $f(x, k): q = x/2$; for ($i=3; i \leq k; i++$) { $q = q * x/i$ }

2) **Модиф. схема** (накопление члена последовательности) - N сложений, $N-1$ делений, $N-1$ умножений;

main: $s = 1 + x$; $p = x$; for ($k=2; k \leq N; k++$) { $p = p * x/k$; $s = s + p$ }

3) **Схема Горнера:** $e^x = 1 + x * (1 + x/2 * (1 + x/3 * (\dots (1 + x/(N-1) * (1 + x/N))))$ – N сложений, $N-2$ делений, $N-1$ умножений.

main: $s = 1$; for ($k=N; k > 1; k--$) { $s = 1 + s * x/k$ }; $s = 1 + s * x$;

Анализ:

1) плохая точность, избыточная арифметическая сложность (АС) алгоритма, параллелится хорошо geometr. методом, но с некоторым дисбалансом загрузки процессоров;

2) средняя точность, оптимальная АС, параллелится плохо;

3) хорошая точность, оптимальная АС, не параллелится совсем.

Задача: предложить оптимальную параллельную последовательность. Посчитать ускорение и эффективность, учитывая, что t – время на сложение, at – на умножение, βt – на деление, γt – на пересылку одного числа; учесть операции на увеличение счетчика цикла и проверки на конец цикла (время сравнения = времени сложения). Что изменится, если считать сумму с проверкой условия на точность: $|e^x - s(x)| \leq \epsilon$? **Ответ:** A_1 – схема Горнера, A_2 – Табл. 1, A_3 – Табл. 2, A_k – см. ниже.

Таблица 1. Решение для двух процессоров:

Вариант 1			Вариант 2		
1-ый проц.	Обмен	2-ой проц.	1-ый проц.	Обмен	2-ой проц.
$p=x$		$s=1, k=2$	$s1=1, k=2$		$s2=0, p=x, k=1$
	$\leftarrow k \rightarrow; \rightarrow p \leftarrow$			$\leftarrow p \rightarrow$	
$p=p*x/k$		$if(k < N): s=s+p, k=k+1$	$p=p*x/k$		$if(k < N): s2=s2+p, k=k+2$
	$\leftarrow k \rightarrow; \rightarrow p \leftarrow$			$\rightarrow p \leftarrow$	
$p=p*x/k$		$if(k < N): s=s+p, k=k+1$	$if(k < N): s1=s1+p, k=k+2$		$p=p*x/k$
	$\leftarrow k \rightarrow; \rightarrow p \leftarrow$			$\leftarrow p \rightarrow$	
...			...		
$p=p*x/k$		$if(k < N): s=s+p, k=k+1$	$p=p*x/k$		$else: s2=s2+p$
	$\leftarrow k \rightarrow; \rightarrow p \leftarrow$			$\rightarrow p \leftarrow$	
		$else: s=s+p, stop$	$else: s1=s1+p$		
				$\leftarrow s2 \rightarrow$	
			$s=s1+s2, stop$		

Таблица 2. Решение для трех процессоров:

1-ый проц.	Обмен	2-ой проц.	Обмен	3-ий проц.
$p=x$				$s=1, k=1$
	$\rightarrow p \leftarrow$			
	$\leftarrow p \rightarrow$		$\rightarrow p \leftarrow$	
$p=p*x$				$if(k < N): s=s+p, k=k+1$
	$\rightarrow p \leftarrow$		$\leftarrow k \rightarrow$	
		$p=p/k$		
	$\leftarrow p \rightarrow$		$\rightarrow p \leftarrow$	
$p=p*x$				$if(k < N): s=s+p, k=k+1$
...				
$p=p*x$				$if(k < N): s=s+p, k=k+1$
	$\rightarrow p \leftarrow$		$\leftarrow k \rightarrow$	
		$p=p/k$		
	$\leftarrow p \rightarrow$		$\rightarrow p \leftarrow$	
				$else: s=s+p, stop$

Решение для произвольного числа процессоров (модифицированная схема):

Пусть N – точное число членов: $S_N(x) = 1 + x + x^2/2! + \dots + x^{N-1}/(N-1)!$; p – число процессоров, и $k = N/p$ (дел. нацело).

На 0-м проц.: $S_0 = 1 + x + x^2/2! + \dots + x^{k-1}/(k-1)! = 1 * [1 + x + x^2/2! + \dots + x^{k-1}/(k-1)!] = A_0 * B_0$

На 1-м проц.: $S_1 = x^k/k! + x^{k+1}/(k+1)! + \dots + x^{2k-1}/(2k-1)! = [x^k/k!] * [1 + x/(k+1) + \dots + x^{k-1}/((k+1) \dots (2k-1))] = A_1 * B_1$

На 2-м проц.: $S_2 = x^{2k}/(2k)! + x^{2k+1}/(2k+1)! + \dots + x^{3k-1}/(3k-1)! = [x^{2k}/(2k)!] * [1 + x/(2k+1) + \dots + x^{k-1}/((2k+1) \dots (3k-1))] = A_2 * B_2$

И т.д. до $(p-1)$.

Схема параллельного счета:

Этап 1. Вычисление B_m (m -номер процессора) и накопление величины $C_m = x^{k-1}/(mk+1) \dots ((m+1)k-1)$ к концу этапа.

Этап 2. Передача C_m и вычисление A_m : $0 \rightarrow C_0 \rightarrow 1, A_1 = C_0 * x/k, C_1 = C_1 * A_1, 1 \rightarrow C_1 \rightarrow 2, A_2 = C_1 * x/(2k), C_2 = C_2 * A_2, 2 \rightarrow C_2 \rightarrow 3, \dots$

Этап 3. Вычисление S_m

Этап 4. Вычисление S методом сдваивания.