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).

$A = \{O_1,...,O_N\}$ = последовательный алгоритм и его операции $\left|\begin{matrix} O_{11} \\ ... \\ O_{1N_1} \end{matrix}\right| ... \left|\begin{matrix} O_{k1} \\ ... \\ O_{kN_k} \end{matrix}\right| \Rightarrow \left|\begin{matrix} O_{01} \\ ... \\ O_{0N_0} \end{matrix}\right|$

параллельная форма алгоритма

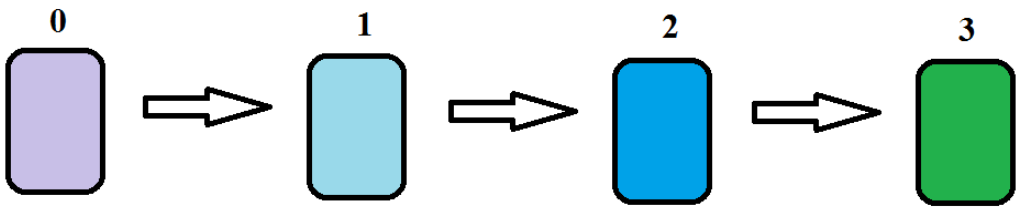
$Q_1,...,Q_k$ - выполняем независимо

Q_0 - выполнение зависит от $Q_1,...,Q_k$

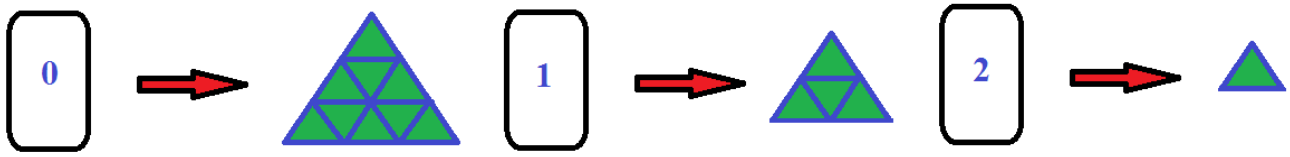
2) ***Геометрический (координатный)*** – разбиение последовательного алгоритма на группы однородных действий над однотипными данными, при котором связи между группами отсутствуют или минимальны. Последнее свойство называют свойством ***локальности*** алгоритма. Локальный алгоритм допускает разбиение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

СУММА(1:1000) = СУММА(1:500)+СУММА(501:1000)

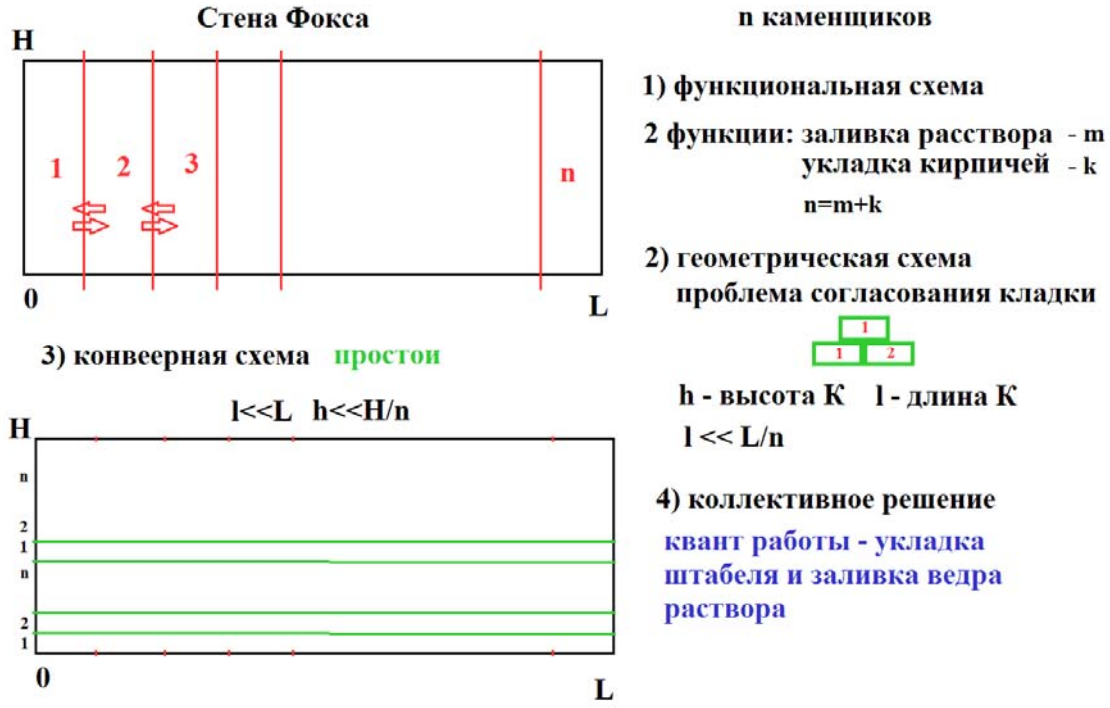
3) ***Конвеерный*** – объединение различных операций последовательного алгоритма в конвеер.



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.



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<<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 простои 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 => есть простои. Эффективен при большом объеме работ (число квантов работы >> 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) $c_i=a_i+b_i$ ($i=1,\ldots,N$): степень параллелизма $d = N$; 2) $c_0=0$, $c_i=c_{i-1}+a_i$ ($i=1,\ldots,N$): степень параллелизма $d = 1$. Степень параллелизма не зависит от числа вычислителей, это внутреннее свойство алгоритма. Ускорить алгоритм более чем в ***d*** раз нельзя => $p <= d$.

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

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

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

Уточнение понятия масштабируемости.

Масштабируемость алгоритма – зависимость времени решения задачи от количества используемых вычислителей.

Сильная масштабируемость – зависимость времени решения задачи от количества используемых вычислителей при фиксированном объеме операций, приходящемся на все вычислительные устройства.

Слабая масштабируемость – зависимость времени решения задачи от количества используемых вычислителей при фиксированном объеме операций, приходящемся на каждое вычислительное устройство.

Алгоритмов с бесконечной сильной масштабируемостью не существует – 2й класс.

Алгоритмы с бесконечной слабой масштабируемостью существуют, например, построенные на принципе геометрического параллелизма – 3й класс.

4. Примеры оценки ускорения и эффективности ПА.

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

A_1: $s=a_1$, $s=s+a_i$, $i=2,…,N$. – 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 $N=2^q$ => $q=\log_2 N$ 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 s, αt – время обмена. Посчитаем: $S_p = tN / [2t*\log_2 N+2\alpha t*(\log_2 N-1)] = p / [(1+\alpha)*\log_2 p+1]$, $E_p = 1 / [(1+\alpha)*\log_2 p+1]$.

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

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

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

Анализ: Можно достичь высокой эффективности $p<<N$. 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) $\beta>0, T_d > 0, T_d>>T_1 \Rightarrow S_p<1$ – при больших накладных расходах не имеет смысла параллелить.

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

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

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

3) ***Схема Горнера***: $e^x=1+x*(1+x/2*(1+x/3*(…(1+x/(N-1)*(1+x/N))…)))$ –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 геометр. 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, αt – 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: $|e^x - s(x)| <= eps$? ***Ответ:*** $\mathbf{A_1}$ – схема Горнера, $\mathbf{A_2}$ – Табл. 1, $\mathbf{A_3}$ – Табл. 2, $\mathbf{A_k}$ – 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***
	<-- k --; -- p -->			***<-- p --***	
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***
	<-- k --; -- p -->			***-- p -->***	
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***
	<-- k --; -- p -->			***<-- p --***	
…			***…***		
p=p*x/k		***if(k<N):s=s+p, k=k+1***	***p=p*x/k***		***else: s2=s2+p***
	<-- k --; -- p -->			***-- p -->***	
		else:s=s+p, stop	***else:s1=s1+p***		
				<-- s2 --	
			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***
	-- p -->			
	<-- p --		***-- p -->***	
p=p*x				***if(k<N):s=s+p, k=k+1***
	-- p -->		***<-- k --***	
		p=p/k		
	<-- p --		***-- p -->***	
p=p*x				***if(k<N):s=s+p, k=k+1***
…				
p=p*x				***if(k<N):s=s+p, k=k+1***
	-- p -->		***<-- k --***	
		p=p/k		
	<-- p --		***-- p -->***	
				else:s=s+p, stop

Решение для произвольного числа процессоров (модифицированная схема):

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

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

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

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

И т.д. до $(p-1)$.

Схема параллельного счета:

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

Этап 2. Передача C_m и вычисление A_m: $0 \text{ -- } C_0 \text{ --> } 1,\ A_1=C_0*x/k,\ C_1=C_1*A_1,\ 1 \text{ -- } C_1 \text{ --> } 2,\ A_2=C_1*x/(2k),\ C_2 = C_2*A_2,\ 2 \text{ -- } C_2 \text{ --> } 3, \ldots$

Этап 3. Вычисление S_m

Этап 4. Вычисление S методом сдваивания.