**Лабораторная работа №2**

**(выполнил Лучко Александр Юрьевич, студент 2 курса МатМех СПБГУ)**

**Цели работы:**1) исследовать алгоритм нахождения среднего арифметического использованием не единичного числа потоков.

2)построить модель оператор операнд.

3)написать программный код для реализации данного алгоритма.

**Инструменты:**

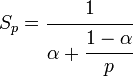
1)VS 2013 Desktop, c#, NET 4.5

**Часть 1**

**Основные определения и законы**

Вспомним основной принцип распараллеливания программ:

закон Амдала



где a-это часть работы которая не поддаётся распараллеливанию(от 0 до 1) p-количество потоков( очевидно, неотрицательное натуральное число). Формула понятна и интуитивно очевидна.

**Модель вычислений в виде графа "операции–операнды"**

Для описания существующих информационных зависимостей в выбираемых алгоритмах решения задач может быть использована модель в виде графа "операции–операнды" Для уменьшения сложности излагаемого материала при построении модели будет предполагаться, что время выполнения любых вычислительных операций является одинаковым и равняется 1 (в тех или иных единицах измерения). Кроме того, принимается, что передача данных между вычислительными устройствами выполняется мгновенно без каких-либо затрат времени (что может быть справедливо, например, при наличии общей разделяемой памяти в параллельной вычислительной системе).

В нашем случае всё так и будет, это простой пример построения подобной модели, все операции будут выполняться в идеале одинаково по времени.

Представим множество операций, выполняемых в исследуемом алгоритме решения вычислительной задачи, и существующие между операциями информационные зависимости в виде ациклического ориентированного графа

G=\left(V,R\right)

где V=\lbrace 1,\ldots, \left|V\right|\rbrace есть множество вершин графа, представляющих выполняемые операции алгоритма, а R есть множество дуг графа (при этом дуга r=\left(i,j\right) принадлежит графу только в том случае, если операция j использует результат выполнения операции i). Как будет показано далее, разные схемы вычислений обладают различными возможностями для распараллеливания и, тем самым, при построении модели вычислений может быть поставлена задача выбора наиболее подходящей для параллельного исполнения вычислительной схемы алгоритма.

В рассматриваемой вычислительной модели алгоритма вершины без входных дуг могут использоваться для задания операций ввода, а вершины без выходных дуг – для операций вывода. Обозначим через \overline V множество вершин графа без вершин ввода, а через d\left(G\right) диаметр (длину максимального пути) графа.

**Описание схемы параллельного выполнения алгоритма**

Пусть p есть количество процессоров, используемых для выполнения алгоритма. Тогда для параллельного выполнения вычислений необходимо задать множество (расписание)

H_p=\lbrace\left(i,P_i,t_i\right):i\in V\rbrace

в котором для каждой операции i\in V указывается номер используемого для выполнения операции процессора P_i и время начала выполнения операции t_i. Для того, чтобы расписание было реализуемым, необходимо выполнение следующих требований при задании множества H_p

1. \forall i,j\in V : t_i=t_j\Longrightarrow P_i\neq P_j, т. е. один и тот же процессор не должен назначаться разным операциям в один и тот же момент времени,
2. \forall \left(i,j\right)\in R\Longrightarrow t_j\geq t_i+1, т. е. к назначаемому моменту выполнения операции все необходимые данные уже должны быть вычислены.

(очевидные правила, вспомним тот же сетевой граф планирования, как кто-то может делать сразу несколько задач и как кто-то может сделать что-то, не сделав подзадачу этого, в этом и есть во многом смысл этой модели).

**Определение времени выполнения параллельного алгоритма**

Вычислительная схема алгоритма G совместно с расписанием H_p может рассматриваться как модель параллельного алгоритма A_p\left(G,H_p\right), исполняемого с использованием p процессоров. Время выполнения параллельного алгоритма определяется максимальным значением времени, используемым в расписании

T_p\left(G,H_p\right)=\max\limits_{i\in V}\left(t_i+1\right)

Для выбранной схемы вычислений желательно использование расписания, обеспечивающего минимальное время исполнения алгоритма

T_p\left(G\right)=\min\limits_{H_p}T_p\left(G,H_p\right)

Уменьшение времени выполнения может быть обеспечено и путем подбора наилучшей вычислительной схемы

T_p=\min\limits_{G}T_p\left(G\right)

Оценки T_p\left(G,H_p\right), T_p\left(G\right) иT_p могут быть использованы в качестве показателей времени выполнения параллельного алгоритма. Кроме того, для анализа максимально возможного параллелизма можно определить оценку наиболее быстрого исполнения алгоритма

T_\infty=\min\limits_{p\geq 1}T_p

Оценку T_\infty можно рассматривать как минимально возможное время выполнения параллельного алгоритма при использовании неограниченного количества процессоров (концепция вычислительной системы с бесконечным количеством процессоров, обычно называемой паракомпьютером, широко используется при теоретическом анализе параллельных вычислений).

Оценка T_1 определяет время выполнения алгоритма при использовании одного процессора и представляет, тем самым, время выполнения последовательного варианта алгоритма решения задачи. Построение подобной оценки является важной задачей при анализе параллельных алгоритмов, поскольку она необходима для определения эффекта использования параллелизма (ускорения времени решения задачи). Очевидно, что

T_1\left(G\right)=\left|\overline{V}\right|

где \left|\overline{V}\right|, напомним, есть количество вершин вычислительной схемы G без вершин ввода. Важно отметить, что если при определении оценки T_1 ограничиться рассмотрением только одного выбранного алгоритма решения задачи и использовать величину

T_1=\min\limits_{G}T_1\left(G\right)

то получаемые при использовании такой оценки показатели ускорения будут характеризовать эффективность распараллеливания выбранного алгоритма. Для оценки эффективности параллельного решения исследуемой задачи вычислительной математики время последовательного решения следует определять с учетом различных последовательных алгоритмов, т. е. использовать величину

T_{1}^*=\min T_1

где операция минимума берется по множеству всех возможных последовательных алгоритмов решения данной задачи.

Приведем без доказательства теоретические положения, характеризующие свойства оценок времени выполнения параллельного алгоритма.

**Теорема 1**. Минимально возможное время выполнения параллельного алгоритма определяется длиной максимального пути вычислительной схемы алгоритма, т. е.

T_\infty\left(G\right)=d\left(G\right)

**Теорема 2**. Пусть для некоторой вершины вывода в вычислительной схеме алгоритма существует путь из каждой вершины ввода. Кроме того, пусть входная степень вершин схемы (количество входящих дуг) не превышает 2. Тогда минимально возможное время выполнения параллельного алгоритма ограничено снизу значением

T_\infty\left(G\right)=\log_2n

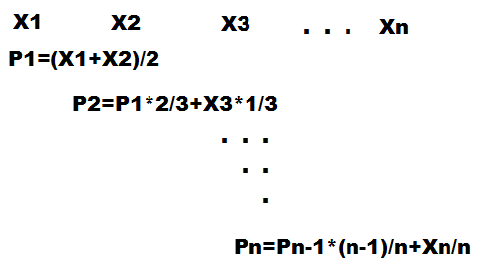
где n есть количество вершин ввода в схеме алгоритма.

**Теорема 3**. При уменьшении числа используемых процессоров время выполнения алгоритма увеличивается пропорционально величине уменьшения количества процессоров, т. е.

\forall q=cp, 0<c<1\Longrightarrow T_p\leq cT_q

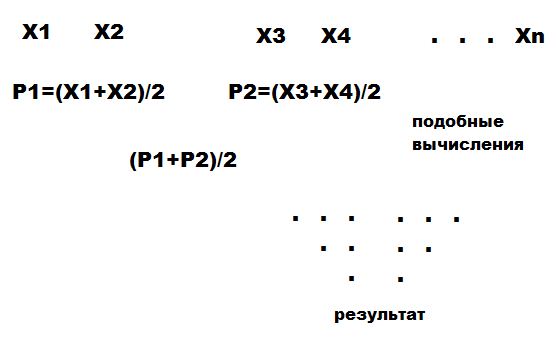
**Построим нашу модель (количество элементов степень 2)**

**1 процессор**

****

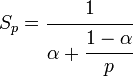
**T1(G)=n-1**

**теперь для p процессоров**

****

**Tp(G,Hp)=LOG2n**

**Sp(n)=(n-1)/ LOG2n**

Подставим в закон Амдала 

Уровни дерева очевидно и есть участки которые можно параллелить нашей программе их **LOG2n**

**Sp=1/( LOG2n+(1- LOG2n)/p)**

В папке с этим документом находится код, который позволяет наглядно увидеть принцип работы подобных моделей.

В данной статье я пользовался материалами из  
1) http://www.intuit.ru/studies/courses/4447/983/lecture/14931?page=12  
2) https://ru.wikipedia.org