Лабораторная работа 1

Основным заданием данной лабораторной работы является разработка и исследование параллельной программы, осуществляющей поиск численного решения для уравнения переноса:

$$\frac{\partial u(x,t)}{\partial t} + \frac{\partial u(x,t)}{\partial x} = 5xt, 0 \le t \le T, 0 \le x \le X$$
$$u(0,x) = -\frac{5}{6}x^3, 0 \le x \le X$$
$$u(t,0) = 0, 0 \le t \le T$$

Для решения задачи используется равномерная сетка с шагами τ по времени и h по координате. Функция $u(x,t)$ рассматривается в точках
$t = k\tau, x = mh, 0 \le k \le K, 0 \le m \le H, T = K\tau, X = Mh$

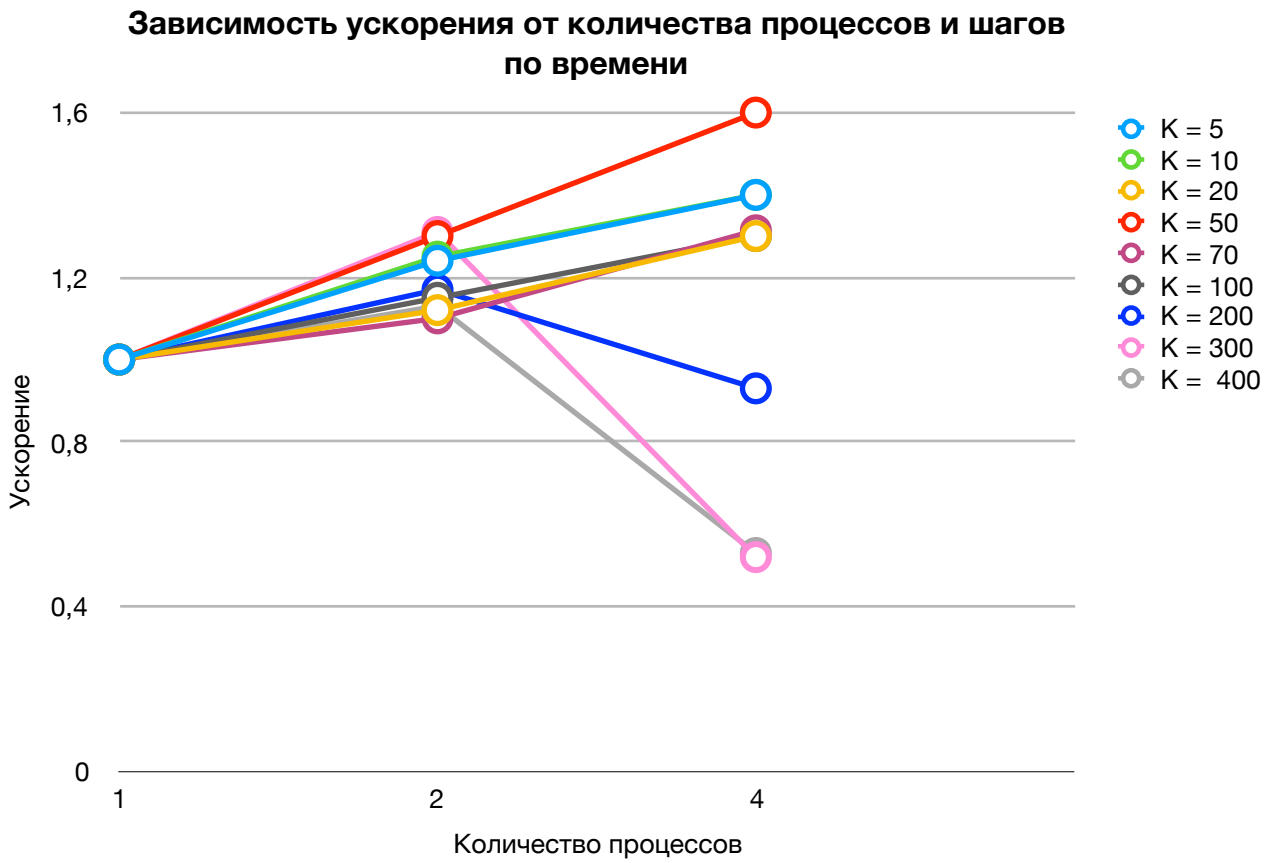
Для поиска численного решения используется явная центральная трехточеченая схема

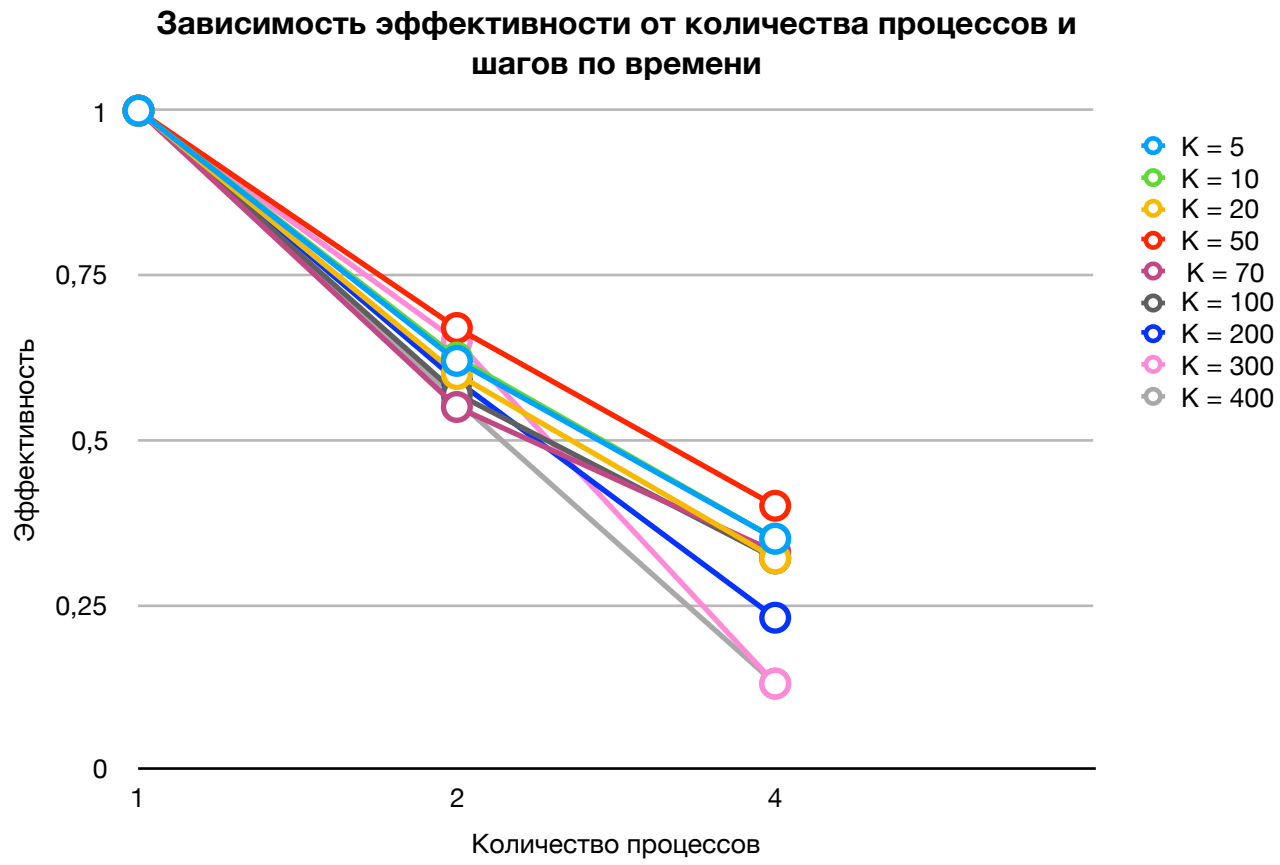
$$\frac{y_m^{n+1} - 0.5(y_{m+1}^n + y_{m-1}^n)}{\tau} + a\frac{y_{m+1}^n - y_{m-1}^n}{2h} = f_m^k$$

В нашей задаче мы использовали такие параметры: $T = 1, h = 0.025, X = 24000, M = 800000$

Измерим время выполнения программы для разного количества процессов и разного количества шагов по времени

Процессы\ τ	T_1	T_2	$S_2 = \frac{T_1}{T_2}$	$E_2 = \frac{S_2}{2}$	T_4	$S_4 = \frac{T_1}{T_4}$	$E_4 = \frac{S_4}{4}$
K = 10	0,07	0,056	1,25	0,625	0,0498	1,40562248995	0,35140562248
K = 100	0,7	0,612	1,14379084967	0,5718954248	0,549	1,27504553734	0,31876138433
K = 200	1,4	1,2	1,16666666666	0,5833333333	1,5	0,93333333333	0,23333333333
K = 5	0,031	0,025	1,24	0,62	0,022	1,40909090909	0,35227272727
K = 20	0,13	0,116	1,12068965517	0,5603448275	0,101	1,28712871287	0,32178217821
K = 50	0,4	0,3	1,33333333333	0,6666666666	0,25	1,6	0,4
K = 70	0,46	0,42	1,09523809523	0,5476190476	0,35	1,31428571428	0,32857142857
K = 300	2,38	1,83	1,30054644808	0,6502732240	4,6	0,51739130434	0,12934782608
K = 400	3,13	2,78	1,12589928057	0,5629496402	5,9	0,53050847457	0,13262711864


Зависимость ускорения от количества процессов и шагов по времени
1,6
1,2
0,8
0,4
0
Ускорение
1
2
4
Количество процессов
K = 5
K = 10
K = 20
K = 50
K = 70
K = 100
K = 200
K = 300
K = 400


Зависимость эффективности от количества процессов и шагов по времени
1
0,75
0,5
0,25
0
Эффективность
1
2
4
Количество процессов
K = 5
K = 10
K = 20
K = 50
K = 70
K = 100
K = 200
K = 300
K = 400

Контрольные вопросы:

1.Ускорение и эффективность параллельных алгоритмов.

Рассмотрим некоторый последовательный алгоритм решения какой-либо задачи. В нем есть как операции, которые не могут выполнятся параллельно (например, ввод/вывод), так и операции, которые можно выполнять на нескольких процессорах одновременно. Пусть доля последовательных операций в алгоритме равна α.

Время выполнения последовательного алгоритма обозначим T_1. Время выполнения параллельной версии алгоритма на p одинаковых процессорах можно записать следующим образом:

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

Ускорением параллельного алгоритма называют отношение времени выполнения лучшего последовательного алгоритмам к времени выполнения параллельного алгоритма:

$$S = \frac{T_1}{T_p}$$

Параллельный алгоритм может давать большое ускорение, но использовать для этого множество процессов неэффективно. Для оценки масштабируемости параллельного алгоритма используется понятие эффективности:

$$E = \frac{S}{p}$$

2. Закон Адамаля

Зако́н Амдаля — иллюстрирует ограничение роста производительности вычислительной системы с увеличением количества вычислителей.

«В случае, когда задача разделяется на несколько частей, суммарное время её выполнения на параллельной системе не может быть меньше времени выполнения самого медленного фрагмента».

Согласно этому закону, ускорение выполнения программы за счёт распараллеливания её инструкций на множестве вычислителей ограничено временем, необходимым для выполнения её последовательных инструкций.

Теоретическую оценку максимального ускорения, достижимого для параллельного алгоритма с долей последовательных операций равной α определяется законом Амдала:

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

Таким образом, если всего 10% операций алгоритма не может быть выполнена параллельно, то никакая параллельная реализация данного алгоритма не может дать больше ускорение более чем в 10 раз.

3. Свойства канала передачи данных. Латентность.

Латентность - свойство объектов или процессов находиться в скрытом состоянии, не проявляя себя явным образом

HTTP://MASTERS.DONNTU.ORG/2011/FKNT/LYAMINA/LIBRARY/MARSHRUTIZ.HTM

Время передачи данных между процессорами определяет коммуникационную составляющую (communication overhead) длительности выполнения параллельного алгоритма в многопроцессорной вычислительной системе. Основной набор параметров, описывающих время передачи данных, состоит из следующего ряда величин:

- **время начальной подготовки** ($t_н$) характеризует длительность подготовки сообщения для передачи, поиска маршрута в сети и т.п.;
- **время передачи служебных данных** ($t_с$) между двумя соседними процессорами (т.е. для процессоров, между которыми имеется физический канал передачи данных); к служебным данным может относиться заголовок сообщения, блок данных для обнаружения ошибок передачи и т.п.;
- **время передачи одного слова данных** по одному каналу передачи данных ($t_к$); длительность подобной передачи определяется полосой пропускания коммуникационных каналов в сети.

Под пропускной способностью *R* сети будем понимать количество информации, передаваемой между узлами сети в единицу времени (байт в секунду). Очевидно, что реальная пропускная способность снижается программным обеспечением за счет передачи разного рода служебной информации.

Латентностью (задержкой) называется время, затрачиваемое программным обеспечением и устройствами сети на подготовку к передаче информации по данному каналу. Полная латентность складывается из программной и аппаратной составляющих.

Значения пропускной способности будем выражать в мегабайтах в секунду (MB/sec), значения латентности – в микросекундах (msec = 10^{-6} sec).

Время *T(L)*, необходимое на передачу сообщения длины *L*, можно определить следующим образом:

$$T(L)=s+L/R,$$

где *s* - латентность, а *R* - пропускная способность.

4.Виды обменов «точка-точка»: синхронные, асинхронные. Буферизация данных.

В MPI существуют два типа буферов: буфер прикладной программы и системный буфер.

В прикладной программе буфером является та переменная (скалярная или массив), в которую программа записывает данные и адрес которой она указывает системе MPI в качестве источника для отсылки или в качестве приемника для размещения получаемых данных. Система MPI сама не накапливает данные для их совместной пересылки в едином сообщении. Каждый вызов подпрограммы передачи сообщения приводит к пересылке указанных подпрограмме данных, какого бы маленького или большого размера они не были.

Системный буфер служит для копирования данных при асинхронной передаче. Значения из программного буфера копируются в системный буфер, далее инициируется процедура межпроцессорной передачи данных и управление сразу возвращается пользовательской программе. Так как MPI реально будет посылать копию данных, то программа сразу же может использовать переменную-буфер для записи других данных.

Блокирующими и неблокирующими бывают как прием, так и передача.

Блокирующий обмен такой, при котором вызывающая программа приостанавливается до завершения той части обмена, после которой программа может свободно использовать свой программный буфер. В процессе-отправителе различаются два случая - блокирующая синхронная и блокирующая асинхронная передача. В первом случае процесс-отправитель приостанавливается до получения данных процессом-получателем, а во втором случае - до окончания копирования данных из программного буфера в

системный. При блокирующем приеме процесс-получатель приостанавливается до получения всех данных. После завершения блокирующей операции обмена программа может свободно использовать программный буфер.

Неблокирующий обмен только запускает операцию передачи или приема данных и сразу возвращает управление вызвавшей программе. При возврате управления из подпрограммы неблокирующего обмена программный буфер нельзя использовать некоторое время. В процессе-отправителе данные из буфера могут быть еще не переданы получателю или не сохранены в системном буфере, а в случае неблокирующего приема - получаемые данные еще не записаны в буфер. После инициализации операции неблокирующего обмена программа может выполнять работу, не связанную с пересылаемыми данными, и время от времени проверять текущее состояние операции неблокирующего обмена. Программа, инициировавшая операцию неблокирующего обмена, обязательно должна вызвать подпрограмму ожидания завершения обмена. До выполнения такого вызова операция будет считаться незавершенной и MPI не будет освобождать занятые системные ресурсы. Если подпрограмма ожидания вызывается уже после фактического завершения операции обмена (когда подпрограмма проверки состояния выдала соответствующий ответ), то процедура ожидания приводит только к освобождению системных ресурсов. Если процедура ожидания вызвана до фактического завершения операции, то она блокирует программу до окончания обмена.

Обмен между двумя процессами (будем называть его типа "точка-точка" от английского "point-to-point") представляет более простую, но, в тоже время, более гибкую разновидность передачи сообщений. В таком обмене участвуют только два процесса - передающий и принимающий. В оличие от коллективного обмена обмен точка-точка не предусматривает никакого изменения данных при пересылке и в общем случае даже не требует синхронизации участвующих в обмене процессов.

5. Балансировка нагрузки: статическая и динамическая

Балансировка нагрузки (Load Balancing) применяется для оптимизации выполнения распределённых (параллельных) вычислений с помощью распределённой (параллельной) ВС. Балансировка нагрузки предполагает равномерную нагрузку вычислительных узлов (процессора многопроцессорной ЭВМ или компьютера в сети). При появлении новых заданий программное обеспечение, реализующее балансировку, должно принять решение о том, где (на каком вычислительном узле) следует выполнять вычисления, связанные с этим новым заданием. Кроме того, балансировка предполагает перенос (migration – миграция) части вычислений с наиболее загруженных вычислительных узлов на менее загруженные узлы.

Проблема балансировки вычислительной нагрузки распределённого приложения возникает по той причине, что:

- структура распределенного приложения неоднородна, различные логические процессы требуют различных вычислительных мощностей;
- структура вычислительного комплекса (например, кластера), также неоднородна, т.е. разные вычислительные узлы обладают разной производительностью;
- структура межузлового взаимодействия неоднородна, т.к. линии связи, соединяющие узлы, могут иметь различные характеристики пропускной способности.

Статическая балансировка выполняется до начала выполнения распределенного приложения. Очень часто при распределении логических процессов по процессорам используется опыт предыдущих выполнений, применяются генетические алгоритмы. Однако предварительное размещение логических процессов по процессорам (компьютерам) не даёт эффекта.
Это объясняется тем, что:

- Может измениться вычислительная среда, в которой происходит выполнение приложения, какой либо вычислительный узел может выйти из строя.

- Вычислительный узел, на котором выполняется распределенное приложение, занят ещё и другими вычислениями, доля которых со временем может возрасти.

Так или иначе, выигрыш от распределения логических процессов по компьютерам с целью выполнения параллельной обработки становится неэффективным.

Динамическая балансировка предусматривает перераспределение вычислительной нагрузки на узлы во время выполнения приложения. Программное обеспечение, реализующее динамическую балансировку, определяет:

- загрузку вычислительных узлов;
- пропускную способность линий связи;
- частоту обменов сообщениями между логическими процессами распределенного приложения и др.

На основании собранных данных как о распределенном приложении, так и вычислительной среде) принимается решение о переносе логических процессов с одного узла на другой.

6. Геометрический параллелизм

Исходную задачу мы можем разбить на группу областей, независимых друг от друга на каждом расчетном шаге и пересекающихся только по границе разбиения. Т.е. мы рассчитываем (n+1)-м временной слой в каждой области, затем согласуем границы и переходим к расчету следующего слоя.
Однако, при таком подходе, когда мы делим расчетную область на непересекающиеся подобласти, у нас возникают проблемы с пересчетом значений на границах между данными областями, поэтому предлагается следующий достаточно логичный шаг, - делить исходную область на взаимно перекрывающиеся подобласти.
Появятся по две "фиктивные" точки слева для первой области и справа для последней области. Таким образом, мы получаем четыре независимых на каждом шаге по времени процесса. Для перехода к следующей итерации необходимо согласование границ, так как первая область должна передать второй ее левую границу для следующего шага по времени, в свою очередь, вторая область должна предать первой ее правую границу, и т.д.
Данная методика может быть обобщена на большинство численных методов, основанных на уравнениях для моделирования физических процессов.

7. Конвейерный параллелизм.

Конвейерная обработка инструкций – это метод реализации CPU, при котором множество операции над несколькими инструкциями перекрываются.