УДК 004

Система организации компьютеров в сети для совместной вычислительной деятельности

Размахнина Анна Николаевна
Приамурский государственный университет имени Шолом-Алейхема
Студент

Лучанинов Дмитрий Васильевич
Приамурский государственный университет имени Шолом-Алейхема
Старший преподаватель кафедры информационных систем, математики и методик обучения

Аннотация
В статье представлен вариант реализации организации распределенной вычислительной деятельности в клиент-серверной системе. Представлен пример вычисления, сравнение скорости выполнения алгоритма на локальном компьютере и с помощью параллельного вычисления.
Ключевые слова: совместная вычислительная деятельность, клиент, сервер, локальный параллелизм

System of network computers' organization for computer sharing computation activity

Razmakhnina Anna Nikolaevna
Sholom-Aleichem Priamursky State University
Student

Luchaninov Dmitry Vasilyevich
Sholom-Aleichem Priamursky State University
Senior Lecturer of department of information systems, mathematics and teaching methods

Abstract
The article presents organization distributed computing activity embodiment in client-server system. The example of calculation, comparing speed of the algorithm on the local computer and by using parallel computation is described.
Keywords: cooperative computing activity, client, server, local parallelism

С каждым днем возрастает спрос на компьютеры, работающие со сверхбыстрой скоростью. Наука и техника стремительно развиваются, ставя перед собой решение самых непростых задач. Так, например, оптимальным выходом для фармацевтов является разработка лекарственных средств, применяя компьютерные технологии, для чего бы не пришлось в ходе опытов

убивать легионы крыс. Астрономия, исследование виртуальной реальности, диаграммы химических соединений, гидродинамика, промышленность, прогнозы погоды и другие научные области нуждаются в ускорении работы своих программ.

Производители машин делают все для их усовершенствования, им удается добиться того, что быстрота работы процессоров возрастала, но скорость передачи данных при этом не меняется.

Одним из способов повышения скорости работы компьютера признан параллелизм (взаимодействие двух параллельных процессоров друг с другом, для решения прикладных задач).

На сегодняшний день сформировано множество компьютеров, архитектура которых использует параллельную обработку данных.

Выделяют две ведущие формы параллелизма: параллельная обработка на командном уровне и на уровне процессоров. Первый вид параллелизма предполагает запуск в одно время большого количества команд, второй реализует работу над поставленной задачей на нескольких процессорах одновременно. И один и второй подходы имеют свои плюсы и минусы. Далее более подробно будет рассмотрен второй вид параллелизма.

Данная тема весьма актуальна, так как прогресс не стоит на месте, создаются новые технологии, появляются сложнейшие задачи. В связи с этим, возникает необходимость выполнения программ, для которых важна скорость обработки информации. Одним из вариантов является организация совместной вычислительной деятельности компьютеров.

Цель исследования заключается в изучении принципов построения параллельной вычислительной сети, организация совместной вычислительной деятельности двух и более компьютеров.

1 Характеристика параллельных вычислений в сети

Параллельные вычислительные системы – это компьютерные системы, создающие единую вычислительную сеть, организующие одновременную обработку данных и выполнение машинных операций параллельно на нескольких процессорных модулях. Принцип параллельных вычислений заключается в том, что множество задач разделяют на меньшие и в целях ускорения выполнения процесса, решают их одновременно.

Параллельные вычисления начали использовать много лет назад, долгое время исключительно только в вычислениях с высокой производительностью, но на сегодня интерес к распараллеливанию процессов возрастает, потому как рост тактовой частоты процессоров имеет свои ограничения [1].

Так параллельные вычисления заняли доминирующее место среди компьютерной архитектуры, чаще всего в виде многоядерных процессоров. Важность параллельных алгоритмов заключается в том, что благодаря параллелизму появляется возможность ускорить производительность процесса вычислений.

Под параллелизмом понимают свойство систем, подразумевающее выполнение операции в нескольких копиях одновременно, с возможностью взаимодействия друг с другом.

Вычисления, можно производить, как на нескольких ядрах одного чипа, с разделением потоков на одном процессоре, так и на не связанных друг с другом процессорах.

Для параллельных вычислений, когда в один момент времени осуществляется несколько операций работы с данными, чаще возникает необходимость введения многопроцессорности.

В таком случае, получается, достичь улучшения быстроты выполнения процесса в решении поставленной задачи, для этого необходимо применить алгоритм по разбиению вычисления на части и произвести расчет каждой части задачи параллельно, но нескольких процессорах.

Данный способ вычисления позволяет сокращать время решения задачи, а возможность ускорения зависит только от числа имеющихся процессоров.

Организация параллельных вычислений на нескольких машинах может выполняться в нескольких режимах: многозадачный – режим с разделением времени, в таком случае решение происходит на одном процессоре; режим является псевдопараллельным, если главным, исполняемым выступает единственный процесс, а остальные ждут своей очереди для использования процессора; при использовании режима разделения времени удается достигнуть эффективного процесса вычислений, а так же в данном режиме частично осуществляются параллельные вычисления, в итоге такой режим применим на начальном этапе организации параллельных программ; при параллельном режиме, в одно и то же время совместно выполняется несколько команд по обработке данных, для совершения вычислений может использоваться, не только работа нескольких процессоров, но и машина с конвейерным и векторным обрабатывающим устройством; распределённые вычисления, содержат несколько устройств обработки данных, находящихся на определенном расстоянии друг от друга, передача осуществляется по линиям связи, что приводит к увеличению времени передачи и как следствие, эффективность обработки данных снижается, в таком случае, данный способ вычислений можно применять только для параллельных алгоритмов с небольшим количеством потоков данных; данные условия характерно использовать, при организации вычислительных процессов в многомашинных комплексах, созданных из нескольких компьютеров, объединенных в каналах связи локальных или глобальных сетей. В курсовой работе мы рассмотрим второй метод организации параллелизма, реализация которого происходит на многопроцессорных вычислительных системах [2].

Модель многопроцессорной системы представляет собой организацию параллельных процессов, взаимосвязанных друг с другом и выполняющих команду на отдельных процессорах.

Использующиеся языки программирования и программное обеспечение компьютера обеспечивают создание параллельных вычислений, реализуют синхронное выполнение процессов, и исключают асинхронное.

2 Средства передачи данных для параллельной работы компьютеров в сети с помощью сокетов

Сокеты являются мощным и универсальным механизмом совместной работы процессоров. Они могут быть использованы как средство взаимодействия программ на одной машине, на компьютерах, объединенных в локальную сеть или через интернет, что позволяет нам запускать, создавать, обрабатывать приложения различного типа сложности.

Сокеты работают со многими стандартными сетевыми протоколами и содержат определенный интерфейс для взаимодействия с ними. Чаще всего сокеты используются для передачи в IP-сетях. В таком случае, сокеты можно использовать для вычислительной деятельности приложений по протоколам – HTTP, Telnet, FTP и др. Например, можно создать собственный Web-сервер, который будет взаимодействовать одновременно с множеством клиентов. Достоинством обмена информацией с помощью сокетов считается его гибкость. Основной принцип работы с сокетами – передача другому компьютеру сообщения или целого файла.

Итак, можно убедиться, что сокеты многофункциональное и удобное средство сетевого программирования.

В зависимости от вида создаваемой сети выбирается архитектура сети.

Архитектура сети задает главные составляющие сети, определяет её общую структурно-логическую организацию, описывает способ кодирования, техническое и программное обеспечение. Кроме того, архитектура сети включает в себя принципы функционирования и интерфейс.

В данной курсовой работе мы рассмотрим архитектуру клиент-сервер.

Архитектура клиент-сервер – система информационно-вычислительной сети, основные ресурсы которой сосредоточены в серверах, для обслуживания клиентов (рис.1). Архитектура состоит из двух устройств: клиенты и серверы.

Сервер – это объект, который отвечает на запросы клиентов по сети. Сервер способен обрабатывать несколько запросов от нескольких клиентов одновременно. При этом каждое соединение сервера с клиентом является отдельным сокетом. Сервер выполняет задания от клиентов и следит за выполнением их заданий. После каждого решения сервер отправляет полученные результаты на клиентскую часть, пославшую этот запрос.

Для выполнения прикладных процессов выделяют системный комплекс прикладных программ, составляющих сервисную функцию. Сервис – процесс работы с клиентами.

Процесс, вызывающий, с помощью некоторых операций, сервисную функцию, называется клиентом. Клиентом выступает пользователь или программа.

Клиенты – это станции, которые используют услуги сервера, инициализируют соединение и устанавливают взаимодействие пользователя с сетью. Чаще всего, клиенты взаимодействуют с одним сервером единожды. Если есть необходимость общения с несколькими серверами, то создается несколько рабочих станций (клиентов).

Рис. 1. Архитектура клиент – сервер

3 Реализация соединения компьютеров в сети через протокол TCP/IP с помощью среды программирования Delphi

Для создания чата, или программ, обрабатывающих одну и ту же задачу одновременно, необходим обмен данными между программами на разных вычислительных машинах.

Для реализации обмена данными между компьютерами существует множество способов. В данной курсовой работе мы рассмотрим обмен данными по протоколу TCP/IP [3].

Взаимодействие с сервером TCP осуществляется при наличии главного процесса, работа которого заключается в отслеживании запросов с клиентской части. При соединении клиента и сервера, создается новый дочерний процесс, выполняющий заданную работу клиента. В свою очередь сервер передает клиента дочернему процессу и потом снова возвращается к обслуживанию запросов уже других клиентов.

Программа состоит из двух частей серверная и клиентская. Для создания сетевого клиент-серверного приложения используются компоненты ServerSocket и ClientSocket.

Рассмотрим создание серверной части. Как уже сказано, для создания сервера необходим компонент ServerSocket. На форму поместим компонент TServerSocket и зададим ему следующие параметры:

Active = false (отвечает за состояние сервера, активен он на данный момент или нет);

Name = srv (добавляем событие OnCreate, задаем ему srv.active:=true, для запуска сервера);

Port = 22500 (порт, через который происходит взаимодействие клиента и сервера);

ServerType = stNonBlocking (позволяет вести асинхронную работу с сервером посредством операторов чтения/записи);

Также разместим на форме компонент Memo (назовем его log), в котором будет отслеживаться подключение клиентов.

У srv добавим событие onClientConnect, где мы сможем фиксировать подключенных к серверу клиентов и их IP адрес. Код будет выглядеть:

```
Procedure TForm1.srvClientConnect (Sender: TObject; Socket:
TCustomWinSocket);
begin
log.Lines.Add (Подключился клиент с IP адресом ' +
Socket.RemoteAddress); End;
```

Аналогично добавим событие TServerSocket — onClientDisconnect, которое контролирует отключение клиентов от сервера. Для создания клиентской части используется компонент TClientSocket.

Зададим параметры TClientSocket:

Active = false

Port = 22500 (порт, через который происходит взаимодействие клиента и сервера);

ClientType = ctNonBlocking

Address = 127.0.0.1 (IP адрес компьютера, на котором запущено серверное приложение). Добавим компонент TMemo, присвоим ему имя log, чтобы фиксировать подключение к серверу. Добавим на форму компонент Edit, Label и Button. В Edit будем записывать адрес сервера, к которому хотим подключиться. Подключение к серверу будет осуществляться по нажатию кнопки. Запишем в событие OnClick компонента Button1:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
ClientSocket1.Host:=Edit1.Text;
ClientSocket1.Open; end;
```

Для принятия сообщения от сервера используем событие OnRead:

```
procedure TForm1.ClientSocket1Read(Sender: TObject; Socket:
TCustomWinSocket);
var i,s,b,e:integer;
str:string;
begin
s:=0;
str:=Socket.ReceiveText;
b:=StrToInt(Copy(str,1,Pos(' ',str)-1));
e:=StrToInt(Copy(str,Pos(' ',str)+1,length(str)-1));
t := GetTickCount;
for i:=b to e do if IsPrime(i) then Inc(s);
t := GetTickCount-t;
Socket.SendText (IntToStr (t) +' '+IntToStr(s)); end;
```

Теперь, с помощью протокола TCP/IP, мы создали приложение типа клиент-сервер и можем соединить клиентские и серверные части.

4 Реализация совместной вычислительной деятельности компьютеров с помощью метода локального параллелизма

Основной идеей локального параллелизма является выполнение одной операции над всеми элементами массива данных.

Части фрагментов такого массива обрабатываются на разных, связанных параллельно машинах.

Распределяет данные между машинами программа. Участие пользователя сводится к минимуму, он лишь задает настройки.

На основе, созданного соединения между серверной и клиентскими частями программы, организуем параллельную вычислительную деятельность между процессорами.

В качестве задачи выступает подсчет всех простых чисел до числа, заданного пользователем.

На серверной части добавим компоненту srv событие onClientRead, которое используется для приема и обработки информации, полученной от клиента. Код будет выглядеть:

```
procedure TForm1.srvClientRead(Sender: TObject; Socket:
TCustomWinSocket);
var
  str:string;
begin
  str:=Socket.ReceiveText;
  r:=r+1;
  t:=t+StrToInt(Copy(str,1,Pos(' ',str)-1));
  s:=s+StrToInt(Copy(str,Pos(' ',str)+1,length(str)-1));
  if r=srv.Socket.ActiveConnections then
    begin
      ShowMessage ('Количество найденных простых чисел: ' +
        IntToStr(s) +' Время выполнения: '+IntToStr (t)+' мс');
      s:=0;
      t:=0;
      r:=0;
    end;
end;
```

Кроме того, расположим на форме две кнопки, один Edit и Label. Button1 будет считать простые числа на сервере, а Button2 на клиентах. Соответственно код будет выглядеть так:

```
procedure TForm1.Button1Click(Sender: TObject);
  var i,s,t:integer;
begin
  s:=0;
  t := GetTickCount;
```

```
  for i:=0 to StrToInt(Edit1.Text) do if IsPrime(i) then Inc(s);
  t := GetTickCount-t;
  ShowMessage ('Количество найденных простых чисел: ' +
IntToStr(s)+' Время выполнения: '+IntToStr(t)+' мс');
end;
```

И для подсчета, простых чисел на клиентах:

```
procedure TForm1.Button2Click(Sender: TObject);
  var i,j,d,e,b:integer;
begin
  d:=StrToInt(Edit1.Text) div srv.Socket.ActiveConnections;
  b:=0;
  e:=d;
  for i := 0 to srv.Socket.ActiveConnections-1 do
   begin
    srv.Socket.Connections [i].SendText(IntToStr(b)+'
'+IntToStr(e));
    if (StrToInt(Edit1.Text)-e>=d*2) then
     begin
       b:=e+1;
       e:=e+d;
     end
    else
     begin
       b:=e+1;
       e:=StrToInt(Edit1.Text);
     end;
   end;
end;
```

Сама функция подсчета простых чисел будет выглядеть:

```
function IsPrime (n: Integer): boolean;
var
  test: integer;
begin
//Считаем, что число по умолчанию простое.
  IsPrime: = true;
//Пробуем делить число на другие числа.
  for test := 2 to n - 1 do
   if (n mod test) = 0 then
    begin
//Если число делится на другое число (кроме 1 или n) без
остатка,
//то число не является простым.
    IsPrime: = false;
//Выходим из цикла.
     break;
    end;
end;
```

В итоге получаем сетевое клиент-серверное приложение, для подсчета простых чисел.

Мы организовали совместную вычислительную деятельность компьютеров, и на практике убедились, что при параллельном вычислении, для решения задачи было затрачено 30 секунд (рис. 2). Для решения задачи на одной машине потрачено 40 секунд (рис. 3). Можно сделать вывод, что параллельная работа компьютеров оптимизирует скорость решения.

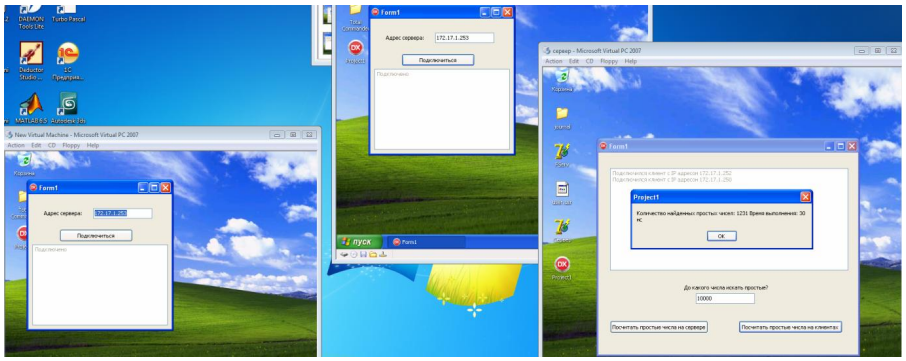

Рисунок 2. Клиент-серверное приложение. Подсчет простых чисел на сервере

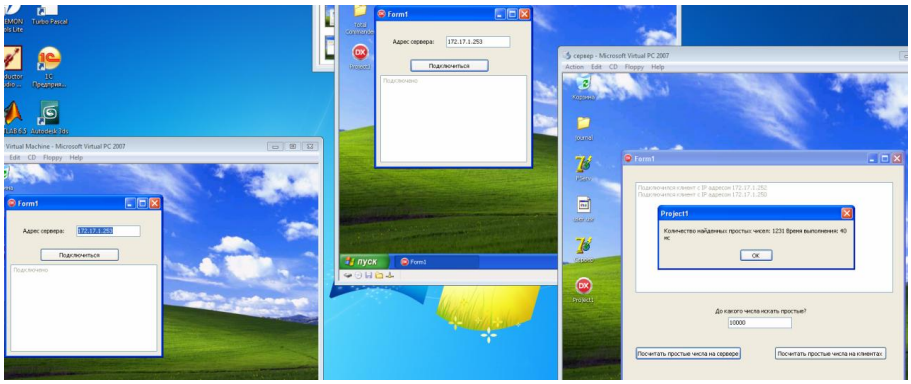

Рисунок 3. Клиент-серверное приложение. Подсчет простых чисел на клиенте

Заключение

На сегодняшний день для увеличения скорости решения прикладных задач, наиболее перспективным направлением является идея параллелизма.

Уже спроектировано и опробовано множество компьютеров, использующих в своей архитектуре параллельные вычисления. Суть параллелизма заключается в разделение задачи на набор меньших задач, которые решаются одновременно.

В данной курсовой работе мы организовали соединение компьютеров по локальной сети. Создали приложение для подсчета простых чисел в заданном диапазоне. Принцип работы программы заключался в параллельной обработке данных на клиентских частях.

Можно отметить, что подсчет на отдельных клиентских частях занимает меньше времени, чем подсчет без распараллеливания данных.

Следовательно, основная задача, которая ставится перед организацией параллельной обработки данных выполнена.

Библиографический список

1. Антонов А. С. Введение в параллельные вычисления. М.: Изд-во МГУ, 2002. 346 с.
2. Гергель В.П, Стронгин Р.Г. Основы параллельных вычислений для многопроцессорных вычислительных систем. 2 изд. Нижний Новгород: Издательство Нижегородского госуниверситета, 2003. 179 с.
3. Гайсина Л.Ф. Основы TCP/IP (часть 1): Методические указания к лабораторному практикуму. Оренбург: РИК ГОУ ОГУ, 2005. 53 с.
4. Головкин Б. А. Параллельные вычислительные системы. М.: Наука. 1980. 520 с.
5. Хокни Р., Джессхоуп К. Параллельные ЭВМ. Архитектура, программирование и алгоритмы. М.: Радио и связь. 1986. 392 с.