Assignment No.: 01

Title: Implement Binary Search Tree using C/C++/Java.

Aim: Using Divide and Conquer Strategies design a cluster/Grid of BBB or Rasberi pi or Computers in network to run a function for Binary Search Tree using C /C++/ Java/Python/ Scala

Objectives:

- To understand working of Cluster of BBB (BeagleBone Black)
- To understand Divide and Conquer strategy
- To implement Binary Search

Theory:

STEPS FOR CREATING CLUSTER OF BEAGLEBONE

Configuring the BeagleBone : Once the hardware is set up and a machine is connected to the network, Putty or any other SSH client can be used to connect to the machines. The default hostname to connect to using the above image is ubuntu-armhf. My first task was to change the hostname. I chose to name mine beaglebone1, beaglebone2 and beaglebone3. First I used the hostname command:

sudo hostname beaglebone1

Next I edited /etc/hostname and placed the new hostname in the file. The next step was to hard code the IP address for so I could probably map it in the hosts file. I did this by editing /etc/network/interfaces to tell it to use static IPs. In my case I have a local network with a router at 192.168.1.1. I decided to start the IP addresses at 192.168.1.51 so the file on the first node looked like this:

```
iface eth0 inet static
address 192.168.1.51
netmask 255.255.255.0
network 192.168.1.0
broadcast 192.168.1.255
gateway 192.168.1.1
```

It is usually a good idea to pick something outside the range of IPs that your router might assign if you are going to have a lot of devices. Usually you can configure this range on your router. With this done, the final step to perform was to edit /etc/hosts and list the name and IP address of each node that would be in the cluster. My file ended up looking like this on each of them:

```
127.0.0.1    localhost
192.168.1.51  beaglebone1
192.168.1.52  beaglebone2
192.168.1.53  beaglebone3
```

Creating a Compute Cluster With MPI

After setting up all 3 BeagleBones, I was ready to tackle my first compute project. I figured a good starting point for this was to set up MPI. MPI is a standardized system for passing messages between machines on a network. It is powerful in that it distributes programs across nodes so each instance has access to the local memory of its machine and is supported by several languages such as C, Python and Java. There are many versions of MPI available so I chose MPICH which I was already familiar with. Installation was simple, consisting of the following three steps:

sudo apt-get update

sudo apt-get install gcc

sudo apt-get install libcr-dev mpich2 mpich2-doc

MPI works by using SSH to communicate between nodes and using a shared folder to share data. The first step to allowing this was to install NFS. I picked beaglebone1 to act as the master node in the MPI cluster and installed NFS server on it:

sudo apt-get install nfs-client

With this done, I installed the client version on the other two nodes:

sudo apt-get install nfs-server

Next I created a user and folder on each node that would be used by MPI. I decided to call mine hpcuser and started with its folder:

sudomkdir /hpcuser

Once it is created on all the nodes, I synced up the folders by issuing this on the master node:

echo "/hpcuser *(rw,sync)" | sudo tee -a /etc/exports

Then I mounted the master's node on each slave so they can see any files that are added to the master node:

sudo mount beaglebone1:/hpcuser /hpcuser

To make sure this is mounted on reboots I have edited /etc/fstab and added the following:

beaglebone1:/hpcuser /hpcusernfs

Finally I created the hpcuser and assigned it the shared folder:

sudouseradd -d /hpcuserhpcuser

With network sharing set up across the machines, I installed SSH on all of them so that MPI could communicate with each:

sudo apt-get install openssh-server

The next step was to generate a key to use for the SSH communication. First I switched to the hpcuser and then used ssh-keygen to create the key.

su - hpcuser

sshkeygen-t rsa

When performing this step, for simplicity you can keep the passphrase blank. When asked for a location, you can keep the default. If you want to use a passphrase, you will need to take extra steps to prevent SSH from prompting you to enter the phrase. You can use ssh-agent to store the key and prevent this. Once the key is generated, you simply store it in our authorized keys collection:

cd .ssh

cat id_rsa.pub >>authorized_keys

I then verified that the connections worked using ssh:

ssh hpcuser@beaglebone2

Testing MPI

Once the machines were able to successfully connect to each other, I wrote a simple program on the master node to try out. While logged in as hpcuser, I created a simple program in its root directory /hpcuser called mpi1.c. MPI needs the program to exist in the shared folder so it can run on each machine. The program below simply displays the index number of the current process, the total number of processes running and the name of the host of the current process. Finally, the main node receives a sum of all the process indexes from the other nodes and displays it:

```
#include <mpi.h>
#include <stdio.h>
int main(intargc, char* argv[])
{
int rank, size, total;
char hostname[1024];
gethostname(hostname, 1023);
MPI_Init(&argc, &argv);
MPI_Comm_rank (MPI_COMM_WORLD, &rank);
MPI_Comm_size (MPI_COMM_WORLD, &size);
MPI_Reduce(&rank, &total, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
printf("Testing MPI index %d of %d on hostname %s\n", rank, size, hostname);
if (rank==0)
   {
printf("Process sum is %d\n", total);
   }
MPI_Finalize();
return 0;
}
```

Next I created a file called machines.txt in the same directory and placed the names of the nodes in the cluster inside, one per line. This file tells MPI where it should run:

```
beaglebone1
beaglebone2
beaglebone3
```

With both files created, I finally compiled the program using mpicc and ran the test:

mpicc mpi1.c -o mpiprogram

mpiexec -n 8 -f machines.txt. /mpiprogram

This resulted in the following output demonstrating it ran on all 3 nodes:

```
Testing MPI index 4 of 8 on hostname beaglebone2
Testing MPI index 7 of 8 on hostname beaglebone2
Testing MPI index 5 of 8 on hostname beaglebone3
Testing MPI index 6 of 8 on hostname beaglebone1
Testing MPI index 1 of 8 on hostname beaglebone2
Testing MPI index 3 of 8 on hostname beaglebone1
Testing MPI index 2 of 8 on hostname beaglebone3
Testing MPI index 0 of 8 on hostname beaglebone1
Process sum is 28
```

Divide and Conquer strategy: The most well-known algorithm design strategy, given a function to compute on n inputs, the divide and conquer strategy consists of: 1. Divide the problem into two or smaller sub-problems. That is splitting the inputs into k distinct subsets, $1 \leq k \leq n$, yielding k sub-problems. 2. Conquer the sub-problems by solving them recursively. 3. Combine the solutions to the sub-problems into the solutions for the original problem. 4. If the sub-problems are relatively large, then divide Conquer is applied again. 5. If the sub-problems are small, then sub-problems are solved without splitting.

Algorithm for Binary Search:

1. Algorithm Bin search(a,n,x)

2. //Given an array a[1:n] of elements in non-decreasing

3. //order, n>=0,determine whether „x" is present and

4. // if so, return „j" such that x=a[j]; else return 0.

5.{ low:=1;high:=n;

6. while(low<=high) do

7. {mid:=[(low+high)/2];

8. if(x<a[mid])then high;

9. else if(x>a[mid]) then low=mid+1;

10. else return mid; }

```
11. return 0;}
```

Mathematical Model:

Binary Search is used for searching of elements from given data.

Following parameters are used for Binary search tree:

M= {s, e, i, o, n, F, Success, Failure}

s = Start state = inserting numbers to array to be sorted in BST manner.

e = End state = key element found & displayed.

i is the set of input elements and key element.

o is the required output i.e. position of the key element.

n = Number of processes = {hostname of processors in a file}

F is the set of functions required for BST search = {f1, f2}

f1 = {Sort the given input in the form of Binary Search Tree}

f2 = {Searching of the key element}

i= {a1, a2, a3,……., an}. = Array of elements to be sorted in BST.

o={Postion of the key element}

Success – Sorted list of elements by BST and keyt element found..

Failure-ϕ

Conclusion: We have studied the concept of divide and conquer strategy and binary search tree algorithm is implemented on Grid of BBB.

Code

```
#include<stdio.h>
#include<mpi.h>
#include<stdlib.h>

struct BSTNode
{
   int data;
   struct BSTNode *left;
   struct BSTNode *right;
};

//Inserting element in BST
struct BSTNode *Insert(struct BSTNode *root, int data)
{
   if(root == NULL)
   {
       root = (struct BSTNode *) malloc (sizeof(struct BSTNode));

       if(root == NULL)
       {
           printf("Memory Error");
           return;
       }
       else
       {
           root -> data = data;
           root -> left = root -> right = NULL;
       }
   }
   else
   {
       if(data < root -> data)
           root -> left = Insert(root -> left, data);
       else if(data > root -> data)
           root -> right = Insert(root -> right, data);
   }
   return root;
}
//int p=0
//Inorder
void inorder(struct BSTNode *root,int *arr){
    if(root){
         inorder(root -> left,arr);
        printf("%d\t", root -> data);
 //        arr[p] = root -> data;
   //      p++;
         inorder(root -> right,arr);
    }
}

int main(int argc,char *argv[])
{

   int a[10] = {1,6,8,3,5,2,4,9,7,0};
   int i,rank,size,b[10],search;

   printf("\n Insert the key element to be searched : ");
   scanf("%d",&search);
   printf("\n You entered : %d\n",search);

   MPI_Request request;
   MPI_Status status;
```

```
/*  int flag;
  int flag1=0;
  int flag2=0;
  */
  MPI_Init(&argc,&argv);
       MPI_Comm_rank(MPI_COMM_WORLD,&rank);
       MPI_Comm_size(MPI_COMM_WORLD,&size);

       MPI_Scatter(&a,5,MPI_INT,&b,5,MPI_INT,0,MPI_COMM_WORLD);

       if(rank == 0)
       {

          struct BSTNode *root_1 = NULL;
          for(i=0;i<5;i++)
          {
               root_1=Insert(root_1, b[i]);
          }
          if (root_1 != NULL)
          {
               printf("\nInorder at rank-%d processor:\t",rank);
           inorder(root_1,b);
       }
       int flag=0;
       int flag1=0;
       MPI_Irecv(&flag,1,MPI_INT,1,3,MPI_COMM_WORLD,&request);
       while(root_1)
       {
           if(flag ==1)
           {
               break;
           }
           if(search == root_1 -> data)
           {
               printf("\nkey %d found at rank-%d processor\n",search,rank);
               flag1=1;
               MPI_Send(&flag1,1,MPI_INT,1,2,MPI_COMM_WORLD);
               break;
           }
           else if(search > root_1 -> data)
               root_1 = root_1 -> right;
           else
               root_1 = root_1 -> left;
       }
       MPI_Send(&flag1,1,MPI_INT,1,2,MPI_COMM_WORLD);
       MPI_Wait(&request,&status);
       if(flag ==0 && flag1 ==0)
       {
           printf("\nKey %d not found\n",search);
       }
       }

       if(rank == 1)
       {
          struct BSTNode *root_2 = NULL;
          for(i=0;i<5;i++)
          {
              root_2=Insert(root_2, b[i]);
          }
          if (root_2 != NULL)
          {
              printf("\nInorder at rank-%d processor:\t",rank);
            inorder(root_2,b);
       }
```

```
        int flag=0;
        int flag1=0;
        MPI_Irecv(&flag,1,MPI_INT,0,2,MPI_COMM_WORLD,&request);
        while(root_2)
        {
            if(flag ==1)
            {
                break;
            }
            if(search == root_2 -> data)
            {
                printf("\nkey %d found at rank-%d processor\n",search,rank);
                flag1=1;
                MPI_Send(&flag1,1,MPI_INT,0,3,MPI_COMM_WORLD);
                break;
            }
            else if(search > root_2 -> data)
                root_2 = root_2 -> right;
            else
                root_2 = root_2 -> left;
        }
        MPI_Send(&flag1,1,MPI_INT,0,3,MPI_COMM_WORLD);
        }

        MPI_Finalize();
        return 0;
}
```

Machinefile

```
ub0:20

ub1:20
```

Output

```
mpiu@DB21:/mirror/mpiu/CL4 prog/A-1$ mpicc bst_mpi.c



mpiu@DB21:/mirror/mpiu/CL4 prog/A-1$ mpiexec -f machinefile -n 4 ./a.out


 Insert the key element to be searched :

 You entered : 0


 Insert the key element to be searched :

 You entered : 0


 Insert the key element to be searched :

 You entered : 0
```

```
6


 Insert the key element to be searched :
 You entered : 6


Inorder at rank-0 processor:
Inorder at rank-1 processor: 0      2      4      7      9
key 0 found at rank-1 processor
1      3      5      6      8
key 6 found at rank-0 processor
mpiu@DB21:/mirror/mpiu/CL4 prog/A-1$
```

Assignment No.: 02

Title: Implement Concurrent Quick Sort using C++.

Aim: Write a program to implement Concurrent Quick Sort in C++ using Divide and Conquer Strategy.

Objectives:

- To learn the concept of Divide and Conquer Strategy.
- To study the design and implementation of Quick Sort algorithm.

Theory:

Divide and Conquer strategy:

A divide and conquer algorithm works by recursively breaking down a problem into two or more sub- problems of the same (or related) type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.

Quick sort:

The sorting algorithm in vented by Ho is, usually known as "quick sort "is also based on the idea of divide-and-conquer. To make it concurrent/parallel, we will use OpenMP API. OpenMP is available along with g++ compiler. In Quick sort, input is any sequence of numbers and output is sorted array (here we will consider ascending order).We start with one number, mostly the first number, and finds its position in sorted array. This number is called pivot element. Then we divide the array into two parts considering this pivot elements position in sorted array. In each part, separately, we find the pivot elements. This process continues until all numbers are processed and we get the sorted array. In concurrent Quick sort, each part is processed by independent thread i.e. different threads will find out pivot element in each part recursively. Check in following diagram. Here PE stands for Pivot Element.

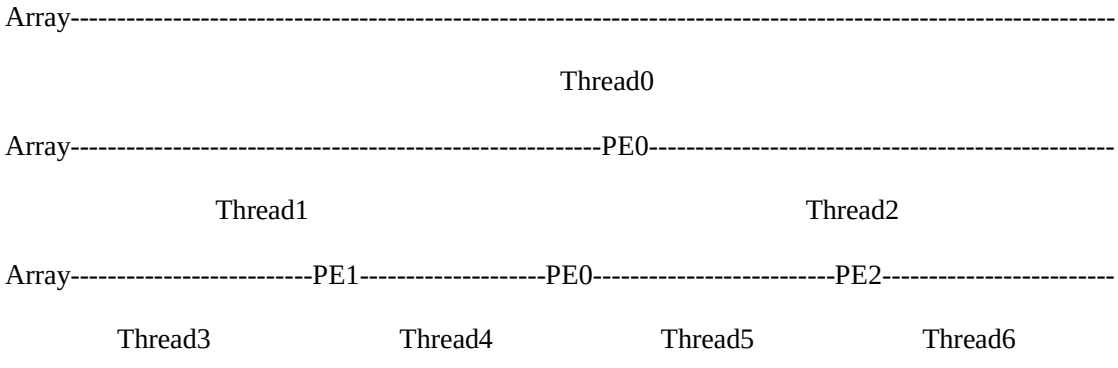


Fig 1 – Concurrently executing Threads

```
void quicksort(int* A,intq,int r)
{
        Int x, s, i;
        if(q<r)
        {       x=A[q];
                s=q;
        for ( i=q+1 ; i<r ; i++)
        {       if (A[i]<=x)
                {       s=s+1;
                        Swap (A[s], A[i]);
                }
        }
}
voidh_quicksort (int* A, int q, int r)
{       int x, s, i ;
        if(q<r)
        {       x=A[q];
                s=q;
                for (i=q+1; i<r; i++)
                {
                        if (A[i]<=x)
                        {
                                s=s+1; swap(A[s],A[i]);
                        }
                }
        swap (A[q], A[s]);
        #pragma ompparallelsections
        {
        #pragma ompsection
        h_quicksort (A, q, s);
        #pragma ompsection
        h_quicksort (A, s+1, r);
        }
}
```

Sections are performed in parallel.

Without omp_set_nested(1) no threds will be created recursively

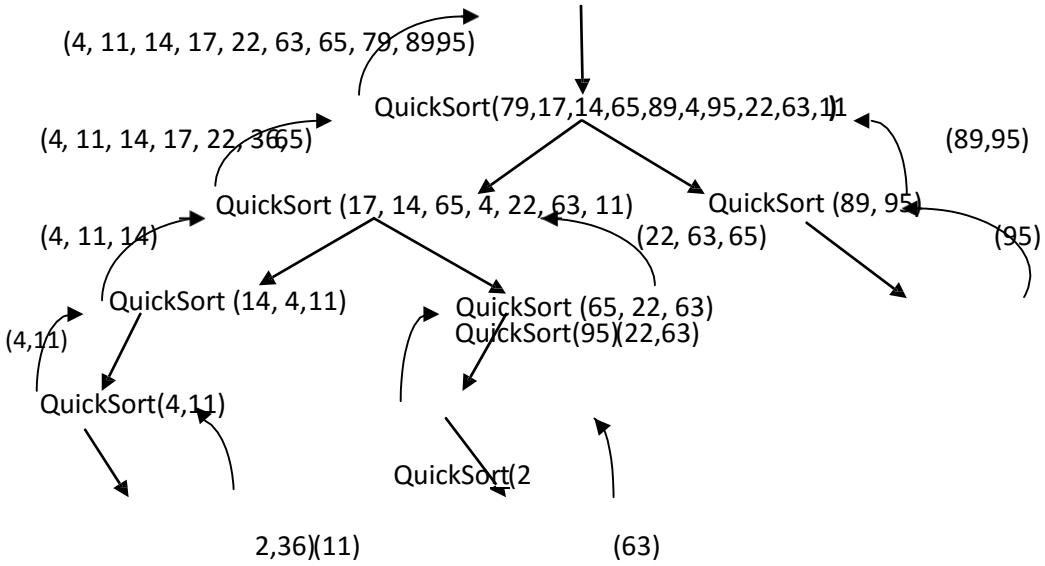
Example:

Given a list of numbers :{ 79, 17, 14, 65, 89, 4, 95, 22, 63,11}
The first number 79 is chosen as pivot

- Low list contains {17, 14, 65, 4, 22, 63, 11}
- High List contains {89,95}

- For sub list {17, 14, 65, 4, 22, 63, 11} choose 17 as pivot
- Low list contains {14, 4,11}
- High List contains {64, 22,63}

{4, 11, 14, 17, 22, 63, 65} is the sorted result of sublist
{17, 14, 65, 4, 22, 63,11}
For sub list {89, 95} choose 89 as pivot
Low list is empty (no need of further recursions)
High list contains {95} (no need of further recursions)
{89, 95} is the sorted
Final sorted list {4, 11, 14, 17, 22, 63, 65, 79, 89, 95}



Mathematical Model :

- Quick Sort is used for sorting of given data.
- Following parameters are used for QuickSort:
- M= {s, e, i, o, F, Success, Failure}
- s = Start state = inserting numbers to array to be sorted.
- e = End state = Sorted list displayed.
- i is the set of input element.
- o is the set of required output.
- F is the set of functions required for QuickSort. = {f1, f2, f3}

- f1 = {Select pivote elemant}
- f2 = {Spliting of array}
- f3 = {Merging of arrays}
- i= {a1, a2, a3,……., an}. = Array of elements to be sorted.
- o={Sorted list of elements by Quick sort}
- Success – Sorted list of elements by Quich sort.
- Failure-ϕ

Output:

Sorted array (in ascending order).

Conclusion:

The concept of divide and conquer strategy is studied and Concurrent Quick Sort algorithm is implemented using C++.

Code

```
#include<iostream>
#include<omp.h>
using namespace std;
int k=0;
class sort
{
        int a[20];
        int n;
public:
        void getdata();
        void Quicksort();
        void Quicksort(int low, int high);
        int partition(int low, int high);
        void putdata();
};
void sort::getdata()
{
        cout<<"Enter the no. of elements in array\t";
        cin>>n;
        cout<<"Enter the elements of array:"<<endl;
        for(int i=0;i<n;i++)
        {
        cin>>a[i];
        }
}
void sort::Quicksort()
{
Quicksort(0,n-1);
}

void sort::Quicksort(int low, int high)
{
if(low<high)
{
        int partn;
        partn=partition(low,high);

cout<<"\n\nThread Number: "<<k<<"  pivot element selected : "<<a[partn];
        #pragma omp parallel sections
        {
        #pragma omp section
                {
                k=k+1;
                Quicksort(low, partn-1);
                }
        #pragma omp section
                {
                k=k+1;
                Quicksort(partn+1, high);
                }
        }//pragma_omp Parallel_end
}

}
int sort::partition(int low ,int high)
{
        int pvt;
        pvt=a[high];
        int i;
        i=low-1;
        int j;
        for(j=low;j<high;j++)
        {
                if(a[j]<=pvt)
```

```
                {
                        int tem=0;
                        tem=a[j];
                        a[j]=a[i+1];
                        a[i+1]=tem;
                        i=i+1;
                }
        }
        int te;
        te=a[high];
        a[high]=a[i+1];
        a[i+1]=te;
        return i+1;
}
void sort::putdata()
{
        cout<<endl<<"\nThe Array is:"<<endl;
        for(int i=0;i<n;i++)
        cout<<" "<<a[i];
}
int main()
{
        int n;
        sort s1;
        int ch;
do
{
                s1.getdata();
                s1.putdata();

                cout<<"\nUsing Quick Sort";
   double start = omp_get_wtime();
                s1.Quicksort();
   double end = omp_get_wtime();
                cout<<"\nThe Sorted  ";
                s1.putdata();
   cout<<"\nExcecution time : "<<end - start<<" seconds ";

cout<<"Would you like to continue? (1/0 y/n)"<<endl;
cin>>ch;
}while(ch==1);
}
```

Machinefile

```
ub0:20
ub1:20
```

Output

```
mpiu@DB21:/mirror/mpiu/CL4 prog/A-2$ g++ qsort.cpp -fopenmp
mpiu@DB21:/mirror/mpiu/CL4 prog/A-2$ ./a.out
Enter the no. of elements in array  5
Enter the elements of array:
7 9 0 -3 2


The Array is:
 7 9 0 -3 2
Using Quick Sort

Thread Number: 0  pivot element selected : 2

Thread Number: 1  pivot element selected : -3

Thread Number: 2  pivot element selected : 7
The Sorted

The Array is:
 -3 0 2 7 9
Excecution time : 0.00439173 seconds Would you like to continue? (1/0 y/n)
0
```

Assignment No.: 03

Title: An MPI program for calculating a quantity called coverage from data files

Aim: Write a MPI program for calculating a quantity called coverage from data files

Objectives:

- To understand clustering
- To implement the Message Passing Interface

Theory:

MPI:

- MPI is a specification for the developers and users of message passing libraries. By itself, it is NOT a library - but rather the specification of what such a library should be.

- MPI primarily addresses the message-passing parallel programming model: data is moved from the address space of one process to that of another process through cooperative operations on each process.

- Simply stated, the goal of the Message Passing Interface is to provide a widely used standard for writing message passing programs. The interface attempts to be:

    - Practical
    - Portable
    - Efficient
    - Flexible

Reasons for using MPI:

- **Standardization -** MPI is the only message passing library which can be considered a standard. It is supported on virtually all HPC platforms. Practically, it has replaced all previous message passing libraries.

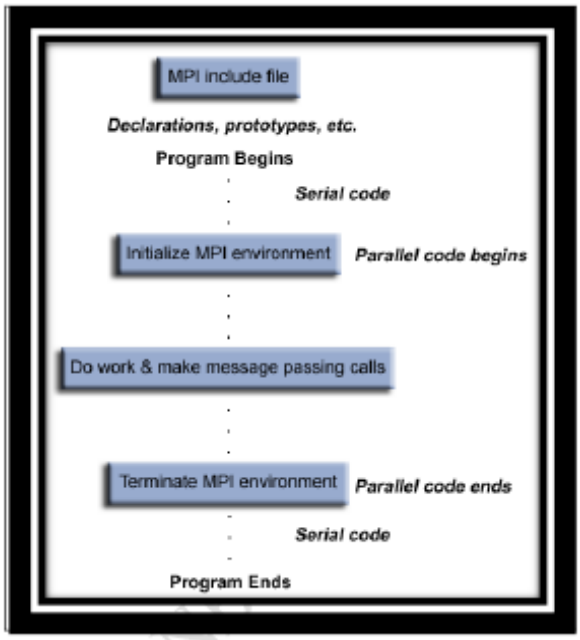
- **Portability**- There is little or no need to modify your source code when you port your application to a different platform that supports (and is compliant with) the MPI standard.

- **Performance Opportunities -**Vendor implementations should be able to exploit native hardware features to optimize performance. Any implementation is free to develop optimized algorithms.

- **Functionality**-There are over 430 routines defined in MPI-3, which includes the majority of those in MPI-2 and MPI-1.

- **Availability**-A variety of implementations are available, both vendor and public domain.

General MPI program structure:



Environment management routines:

This group of routines is used for interrogating and setting the MPI execution environment, and covers an assortment of purposes, such as initializing and terminating the MPI environment, querying a rank's identity, querying the MPI library's version, etc. Most of the commonly used ones are described below.

1) **MPI_Init :**

Initializes the MPI execution environment .This function must be called in every MPI program, must be called before any other MPI functions and must be called only once in an MPI program. For C programs, MPI_Init may be used to pass the command line arguments to all processes, although this is not required by the standard and is implementation dependent.

MPI_Init (&argc,&argv)
MPI_INIT (ierr)

2) **MPI_Comm_size :**

Returns the total number of MPI processes in the specified communicator, such as MPI_COMM_WORLD. If the communicator is MPI_COMM_WORLD, then it represents the number of MPI tasks available to your application.

MPI_Comm_size(comm,&size)
MPI_COMM_SIZE (comm, size, ierr)

3) **MPI_Comm_rank :**

Returns the rank of the calling MPI process within the specified communicator. Initially, each process will be assigned a unique integer rank between 0 and numbers of tasks - 1 within the communicator MPI_COMM_WORLD. This rank is often referred to as a task ID. If a process

becomes associated with other communicators, it will have a unique rank within each of these as well.

MPI_Comm_rank(comm,&rank)
MPI_COMM_RANK (comm,rank,ierr)

4) MPI_Abort :

Terminates all MPI processes associated with the communicator. In most MPI implementations it terminates ALL processes regardless of the communicator specified.

MPI_Abort(comm,errorcode)
MPI_ABORT (comm,errorcode,ierr)

5) MPI_Get_processor_name :

Returns the processor name. Also returns the length of the name. The buffer for "name" must be at least MPI_MAX_PROCESSOR_NAME characters in size. What is returned into "name" is implementation dependent - may not be the same as the output of the "hostname" or "host" shell commands.

MPI_Get_processor_name(&name,&resultlength)
MPI_GET_PROCESSOR_NAME (name,resultlength,ierr)

6) MPI_Get_version:

Returns the version and subversion of the MPI standard that's implemented by the library.

MPI_Get_version(&version,&subversion)
MPI_GET_VERSION (version,subversion,ierr)

7) MPI_Initialized :

Indicates whether MPI_Init has been called - returns flag as either logical true (1) or false (0). MPI requires that MPI_Init be called once and only once by each process. This may pose a problem for modules that want to use MPI and are prepared to call MPI_Init if necessary. MPI_Initialized solves this problem.

MPI_Initialized(&flag)
MPI_INITIALIZED (flag, ierr)

8) MPI_Wtime :

Returns an elapsed wall clock time in seconds (double precision)on the calling processor.

9) **MPI_Finalize :**

Terminates the MPI execution environment.This function should be the last MPI routine called in every MPI program - no other MPI routines may be called after it.

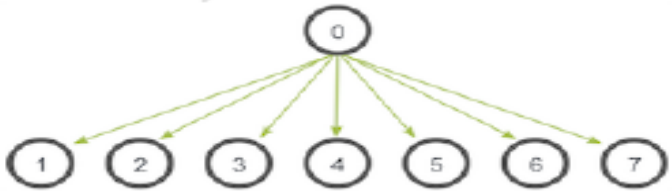
MPI_Finalize()

MPI_FINALIZE (ierr)

10) MPI_Bcast :

A broadcast is one of the standard collective communication techniques. During a broadcast, one process sends the same data to all processes in a communicator. One of the main uses of broadcasting is to send out user input to a parallel program, or send out configuration parameters to all processes.

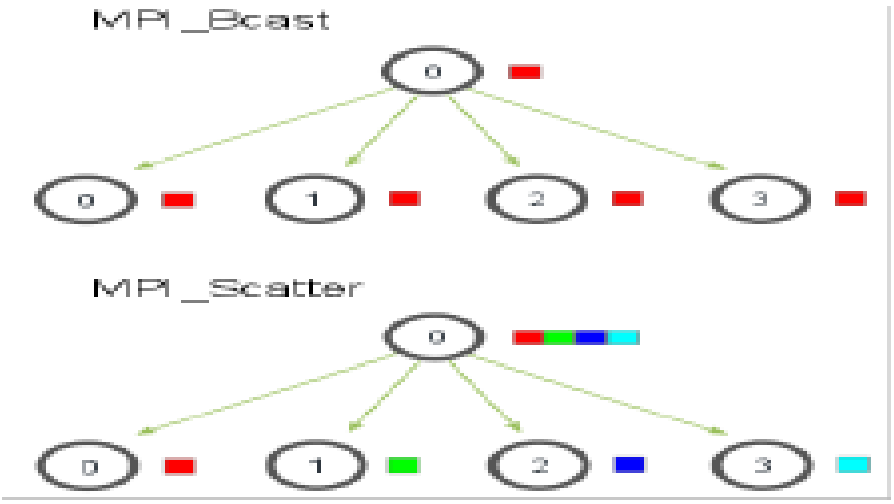
The communication pattern of a broadcast looks like this:



In this example, process zero is the root process, and it has the initial copy of data. All of the other processes receive the copy of data.

11) MPI_Scatter:

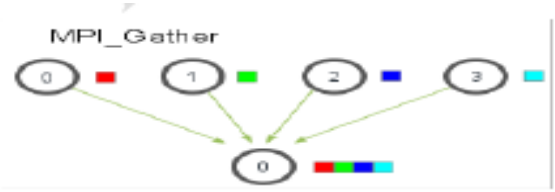
MPI_Scatter is a collective routine that is very similar to MPI_Bcast. MPI_Scatter involves a designated root process sending data to all processes in a communicator. The primary difference between MPI_Bcast and MPI_Scatteris small but important. MPI_Bcast sends the same piece of data to all processes while MPI_Scatter sends chunks of an array to different processes.



In the illustration, MPI_Bcast takes a single data element at the root process (the red box) and copies it to all other processes. MPI_Scatter takes an array of elements and distributes the elements in the order of process rank. The first element (in red) goes to process zero, the second element (in green) goes to process one, and so on. Although the root process (process zero) contains the entire array of data, MPI_Scatter will copy the appropriate element into the receiving buffer of the process.

12) MPI_Gather:

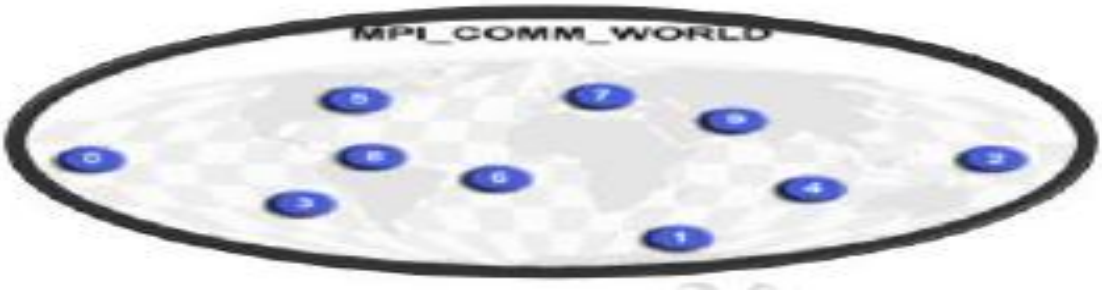
MPI_Gather is he inverse of MPI_Scatter. Instead of spreading elements from one process to many processes, MPI_Gather takes elements from many processes and gathers them to one single process. This routine is highly useful to many parallel algorithms, such as parallel sorting and searching. Below is a simple illustration of this algorithm.



Similar to MPI_Scatter, MPI_Gather takes elements from each process and gathers them to the root process. The elements are ordered by the rank of the process from which they were received.

RANK:

- Within a communicator, every process has its own unique, integer identifier assigned by the system when the process initializes. A rank is sometimes also called a "task ID". Ranks are contiguous and begin at zero.
- Used by the programmer to specify the source and destination of messages. Often used conditionally by the application to control program execution (if rank=0 do this / if rank=1 do that).
- MPI uses objects called communicators and groups to define which collection of processes may communicate with each other.
- Most MPI routines require you to specify a communicator as an argument.
- Communicators and groups will be covered in more detail later. For now, simply use MPI_COMM_WORLD
- whenever a communicator is required - it is the predefined communicator that includes all of your MPI processes.



Steps for creating cluster of computers:

1. At least Two Computers with a Linux Distribution installed in it (I'll use Ubuntu 14.04 here). Make sure that your system has GCC installed in it.

2. A network connection between them. If you have just two computers, you can connect them using an Ethernet wire. Make sure that IP addresses are assigned to them. If you don‟t have a router to assign IP, you can statically assign them IP addresses.

3. Rest of the document will assume that we are having two computers having host names node0 and node1. Let node0 be the master node.

4. The following steps are to be done for every node:

5. Add the nodes to the /etc/hosts file. Open this file using your favorite‟s text editor and add your node's IP address followed by its host name. Give one node information per line. For example,

   node0 10.1.1.1
   node1 10.1.1.2

6. Create a new user in both the nodes. Let us call this new user as mpiuser. You can create a new user through GUI by going to System->Administration->Users and Groups and click "Add User". Create a new user called mpiuser and give it a password. Give administrative privileges to that user. Make sure that you create the same user on all nodes. Although same password on all the nodes is not necessary, it is recommended that you do so because it'll eliminate the need to remember passwords for every node.

7. Now download and install ssh-server in every node. Execute the command sudo apt-get install open ssh-server in every machine

8. Now logout from your session and log in as mpiuser.

9. Open terminal and type the following ssh-keygen -t dsa. This command will generate a new ssh key. On executing this command, it'll ask for a paraphrase. Leave it blank as we want to create a passwordlessssh (Assuming that you've a trusted LAN with no security issues)

10. A folder called .ssh will be created in your home directory. It‟s a hidden folder. This folder will contain a file id_dsa.pub that contains your public key. Now copy this key to another file called authorized_keys in the same directory. Execute the command in the terminal cd /home/mpiuser/.ssh; cat id_dsa.pub >>authorized_keys;.

11. Now download MPICH from the following website (MPICH1). Please download the MPICH 1.xx version from the website. Do not download MPICH 2 version. I was unable to get MPICH 2 to work in the cluster.

12. Untar the archive and navigate into the directory in the terminal. Execute the following commands:

    mkdir/home/mpiuser/mpich1
    ./configure--prefix=/home/mpiuser/mpich1
    make
    make install

13. Open the file .bashrc in your home directory. If file does not exist, create one. Copy the following code into thatfile

    exportPATH=/home/mpiuser/mpich1/bin:$PATH
    exportPATH
    LD_LIBRARY_PATH="/home/mpiuser/mpich1/lib:$LD_LIBRARY_PATH"
    export LD_LIBRARY_PATH

14. Now we'll define the path to MPICH for SSH. Run the following command: sudo echo /home/mpiuser/mpich1/bin >> /etc/environment

15. Now logout and login back into the user mpiuser.

16. In the folder mpich1, within the sub-directory share or util/machines/ a file called machines. LINUX will be found. Open that file and add the hostnames of all nodes except the home node ie. If you're editing the machines. LINUX file of node0, then that file will contain host names of all nodes except node0. By default MPICH executes a copy of the program in the home node. LINUX file for the machine node0 is as follows
    node1: 2
    The number after: indicates number of cores available in each of the nodes.

Mathematical Model:

Let M be the system.

M = {I, P, O, S, F}

Let I be the input.
I = {1,2,3,…..100}
k no of nodes= n1,n2,n3,n4.

Let P be the process.
P= {n,q,r}
n=divide the I into 4 nodes.
q=addition
r= collect the result from each node.

Let O be the Output.
O = {c}
c =add collected result {r}.

Let S be the case of Success.
S = {l}
l = Satisfied all conditions.

Let F be the case of Failure.
F = {f}

f = Satisfied result not generated.

Conclusion:

Hence we studied MPI program for calculating a quantity called coverage from data files

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#define v 1 /* verbose flag, output if 1, no output if 0 */
int main ( int argc, char *argv[] )
{
        int myid,j,*data,tosum[25],sums[5],sums2[5];
        MPI_Init(&argc,&argv);
        MPI_Comm_rank(MPI_COMM_WORLD,&myid);
        if(myid==0) /* manager allocates and initializes the data */
        {

                data = (int*)calloc(100,sizeof(int));
                for (j=0; j<100; j++) data[j] = j+1;
                if(v>0)
                {
                        printf("The data to sum : ");
                        for (j=0; j<100; j++)
                        printf(" %d",data[j]);
                        printf("\n");
                }
        }
        MPI_Scatter(data,25,MPI_INT,tosum,25,MPI_INT,0,MPI_COMM_WORLD);
        if(v>0) /* after the scatter, every node has 25 numbers to sum*/
        {
                printf("Node %d has numbers to sum :",myid);
                for(j=0; j<25; j++) printf(" %d", tosum[j]);
                printf("\n");
        }
        sums[myid] = 0;
        for(j=0; j<25; j++) sums[myid] += tosum[j];
        if(v>0) printf("Node %d computes the sum %d\n",myid,sums[
        myid]);

        MPI_Gather(&sums[myid],1,MPI_INT,&sums2[myid],1,MPI_INT,0,MPI_COMM_WORLD);

        if(myid==0) /* after the gather, sums contains the four sums*/
        {
                printf("The four sums : ");
                printf("%d",sums2[0]);
                for(j=1; j<4; j++) printf(" + %d", sums2[j]);
                for(j=1; j<4; j++) sums2[0] += sums2[j];
                printf(" = %d, which should be 5050.\n",sums2[0]);
        }

        MPI_Finalize();
        return 0;
}
```

machinefile

```
ub0:20

ub1:20
```

Output

```
sagar@sagar-Lenovo-G580:~/CL4 prog/A-3$ mpicc A3_MPI.c

sagar@sagar-Lenovo-G580:~/CL4 prog/A-3$ mpiexec -n 4 -f machinefile ./a.out
```

```
The data to sum :  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87
88 89 90 91 92 93 94 95 96 97 98 99 100

Node 0 has numbers to sum : 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
25

Node 0 computes the sum 325

Node 1 has numbers to sum : 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
47 48 49 50

Node 1 computes the sum 950

Node 2 has numbers to sum : 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
72 73 74 75

Node 2 computes the sum 1575

Node 3 has numbers to sum : 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96
97 98 99 100

Node 3 computes the sum 2200

The four sums : 325 + 950 + 1575 + 2200 = 5050, which should be 5050.
```

Assignment No.: 04

Title: Implement a graph of unloaded cluster for several number of nodes

Aim: Write a program on an unloaded cluster for several different numbers of nodes and record the time taken in each case. Draw a graph of execution time against the number of nodes.

Objectives:

- To understand clustering
- To implement Message Passing Interface

Theory:

Cluster:

A computer cluster consists of a set of loosely or tightly connected computers that work together so that, in many respects, they can be viewed as a single system. Unlike grid computers, computer clusters have each node set to perform the same task, controlled and scheduled by software.

The components of a cluster are usually connected to each other through fast local area networks ("LAN"), with each node (computer used as a server) running its own instance of an operating system. In most circumstances, all of the nodes use the same hardware and the same operating system, although in some setups (i.e. using Open Source Cluster Application Resources (OSCAR)), different operating systems can be used on each computer, and/or different hardware.

They are usually deployed to improve performance and availability over that of a single computer, while typically being much more cost-effective than single computers of comparable speed or availability.

Computer clusters emerged as a result of convergence of a number of computing trends including the availability of low-cost microprocessors, high speed networks, and software for high-performance distributed computing. They have a wide range of applicability and deployment, ranging from small business clusters with a handful of nodes to some of the fastest supercomputers in the world such as IBM's Sequoia.

Loading & Unloading Clusters/ Onlining & Offlining:

After a cluster resource is enabled, the load and unload scripts take care of starting and stopping the services or mounting and dismounting the volumes that are configured in the resource. You start services and mount devices by onlining the resource. You stop services and unmount devices by offlining the resource.

Onlining a resource runs the load script, which loads the resource on its primary preferred node, or on an alternate preferred node if possible, according to the order in its Preferred Nodes list. No action is taken if the preferred nodes are not active in the cluster, or if there are Resource Mutual Exclusion conflicts on its active preferred nodes.

Offlining a resource runs the unload script and unloads the resource from the server. The resource cannot be loaded on any other servers in the cluster and remains unloaded until you load it again.

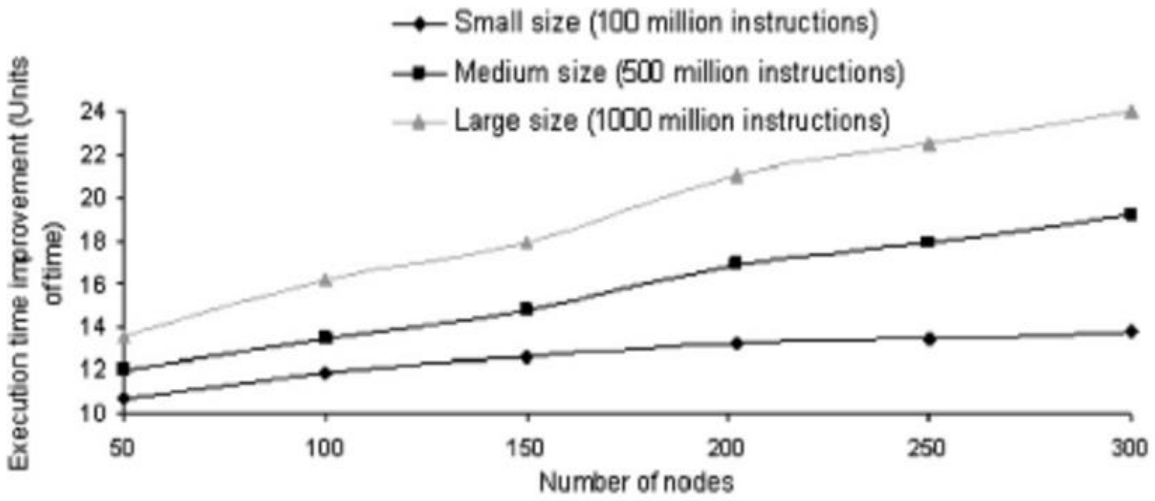


Figure 1 illustrates the improvement of the exe- cution time according to the nodes number.

n: number of nodes in a cluster (its cardinality load : load of a node i = (2) n

μ : average load of a cluster.

Load Balancing Threshold value: The threshold is a value that is used to indicate whether a node is heavily or lightly loaded. It is important to de- termine an appropriate threshold for a good load balancing algorithm.If the threshold is set too low, excessive load balancing will occur, thus causing degradation in performance (as this will result in high communication costs) .

However, if the threshold is set too high, the load balanc- ing mechanism will not be very effective.

Communication cost: This is the total number of load balancing-related messages sent by a node. It gives a measure of the overhead created for balancing loads, which is also an indication of our algorithm cost in terms of energy. After simulations, the obtained results are ana- lyzed by studying the impact of the considered parameters on our load balancing algorithm per- formance. To evaluate the performance of our algorithm in term of execution time improvement, we varied the number of nodes and registered the execu- tion time for different sizes of works (small, average, large) .

Benefits of clustering

1. Clustering solves this as the load of increasing users can be evened out on all the nodes of the cluster.

2. Combined resources of all the nodes of a cluster would greatly boost the performance, as well as allow flexible scalability.

3. Clustering can also reduce the time complexity of the program, another node can take on the responsibilities of a node that expires without letting the affect of the expired node to be felt by the users.

4. Caching provides better data management as well as fast data access. Efficient recovery from any sort of malfunction is possible.

Mathematical Model:

Let M be the system.

M = {I, P, O, S, F}

Let I be the input. I = {k} k = any number.

Let P be the process. P = {n, q, r} n=squaring the number. q=calculating the run time

r= plotting the graph

Let O be the Output. O = {c} c =square of the number and run time.

Let S be the case of Success. S = {l} l = Satisfied all conditions.

Let F be the case of Failure.

F = {f} f = Satisfied result not generated.

Conclusion: We have implemented a graph of unloaded cluster for several number of nodes.

Code

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

#define v 1 /* verbose flag, output if 1, no output if 0 */
#define tag 100 /* tag for sending a number */

int main ( int argc, char *argv[] )
{
        int p,myid,i,f,*x;
        double start, end;
        MPI_Status status;

        MPI_Init(&argc,&argv);
        MPI_Comm_size(MPI_COMM_WORLD,&p);
        MPI_Comm_rank(MPI_COMM_WORLD,&myid);

        if(myid == 0) /* the manager allocates and initializes x */
        {
                x = (int*)calloc(p,sizeof(int));
                x[0] = 50;
                for (i=1; i<p; i++) x[i] = 2*x[i-1];

                if(v>0)
                {
                        printf("The data to square : ");
                        for (i=0; i<p; i++)
                        printf(" %d",x[i]);
                        printf("\n");
                }
        }

        if(myid == 0) /* the manager copies x[0] to f */
        {               /* and sends the i-th element to the i-th processor */
                f = x[0];
                for(i=1; i<p; i++)
                MPI_Send(&x[i],1,MPI_INT,i,tag,MPI_COMM_WORLD);
        }




        else /* every worker receives its f from root */
        {
                MPI_Recv(&f,1,MPI_INT,0,tag,MPI_COMM_WORLD,&status);

                if(v>0)
                        printf("Node %d will square %d\n",myid,f);
        }

        start = MPI_Wtime();

        f *= f;         /* every node does the squaring */

        if(myid == 0) /* the manager receives f in x[i] from processor i */
                for(i=1; i<p; i++)
                        MPI_Recv(&x[i],1,MPI_INT,i,tag,MPI_COMM_WORLD,&status);

        else     /* every worker sends f to the manager */
```

```
                MPI_Send(&f,1,MPI_INT,0,tag,MPI_COMM_WORLD);

        if(myid == 0) /* the manager prints results */
        {
                x[0] = f;
                printf("The squared numbers : ");
                for(i=0; i<p; i++)
                printf(" %d",x[i]);
                printf("\n");

                end = MPI_Wtime();
                printf("Runtime = %f\n", end-start);
        }

        MPI_Finalize();
        return 0;
}
```

Code

```
mpiu@DB21:/mirror/mpiu/CL4 prog/A-4$ mpicc a4.cpp



mpiu@DB21:/mirror/mpiu/CL4 prog/A-4$ mpiexec -n 4 -f machinefile ./a.out

The data to square :  50 100 200 400

Node 1 will square 100

Node 2 will square 200

Node 3 will square 400

The squared numbers :  2500 10000 40000 160000

Runtime = 0.000386

mpiu@DB21:/mirror/mpiu/CL4 prog/A-4$
```

Assignment No.: 05

Title: Implement Booth Multiplication algorithm.

Aim: Build a small compute cluster using Raspberry Pi/BBB modules to implement booth Multiplication algorithm.

Objectives:

To understand working of Cluster of BBB (BeagleBone Black)

Implement booths multiplication algorithm.

Theory:

Booth's algorithm can be implemented by repeatedly adding (with ordinary unsigned binary addition) one of two predetermined values A and S to a product P, then performing a rightward arithmetic shift on P.

Let m and r be the multiplicand and multiplier, respectively; and let x and y represent the number of bits in m and r.

1. Determine the values of A and S, and the initial value of P. All of these numbers should have a length equal to (x + y + 1).

    1. A: Fill the most significant (leftmost) bits with the value of m. Fill the remaining (y + 1)bits with zeros.

    2. S: Fill the most significant bits with the value of (−m) in two's complement notation. Fill the remaining (y + 1) bits with zeros.

    3. P: Fill the most significant x bits with zeros. To the right of this, append the value of r. Fill the least significant (rightmost) bit with a zero.

2. Determine the two least significant (rightmost) bits of P.

    1. If they are 01, find the value of P + A. Ignore any overflow.

    2. If they are 10, find the value of P + S. Ignore any overflow.

    3. If they are 00, do nothing. Use P directly in the next step.

    4. If they are 11, do nothing. Use P directly in the next step.

3. Arithmetically shift the value obtained in the 2nd step by a single place to the right. Let P now equal this new value.

4. Repeat steps 2 and 3 until they have been done y times. Drop the least significant (rightmost) bit from P. This is the product of m and r.

We will explain Booth's Multiplication with the help of following **example**

Multiply 14 times -5 using 5-bit numbers (10-bit result).

14 in binary: 01110

-14 in binary: 10010 (so we can add when we need to subtract the multiplicand)

-5 in binary: 11011

Expected result: -70 in binary: 11101 11010

The Multiplication Process is explained below:

Step	Multiplicand	Action	Multiplier upper 5-bits 0, lower 5-bits multiplier, 1 "Booth bit" initially 0
0	01110	Initialization	00000 11011 0
1	01110	10: Subtract Multiplicand	00000+10010=10010 10010 11011 0
		Shift Right Arithmetic	11001 01101 1
2	01110	11: No-op	11001 01101 1
		Shift Right Arithmetic	11100 10110 1
3	01110	01: Add Multiplicand	11100+01110=01010 (Carry ignored because adding a positive and negative number cannot overflow.) 01010 10110 1
		Shift Right Arithmetic	00101 01011 0
4	01110	10: Subtract Multiplicand	00101+10010=10111 10111 01011 0
		Shift Right Arithmetic	11011 10101 1
5	01110	11: No-op	11011 10101 1
		Shift Right Arithmetic	11101 11010 1

Booth's Multiplication algorithm is an elegant approach to multiplying signed numbers. Using the standard multiplication algorithm, a run of 1s in the multiplier in means that we have to add as many successively shifted multiplicand values as the number of 1s in the run.

0010two

x 0111two

+ 0010 multiplicand shifted by 0 bits left

+ 0010 multiplicand shifted by 1 bit left

+ 0010 multiplicand shifted by 2 bits left

+ 0000

00001110two

We can rewrite 2i - 2i-j as:

2i - 2i-j = 2i-j x (2j - 1)

= 2i-j x (2j-1 + 2j-2 + ... + 20)

= 2i-1 + 2i-2 + ... + 2i-j

For example 0111two = 23ten - 20ten. So 0010two x 0111two can be written as:

0010two x (1000two - 0001two)

Or to make it look like the multiplication before:

0010two

x 0111two

- 0010 = 0010two x - 0001two

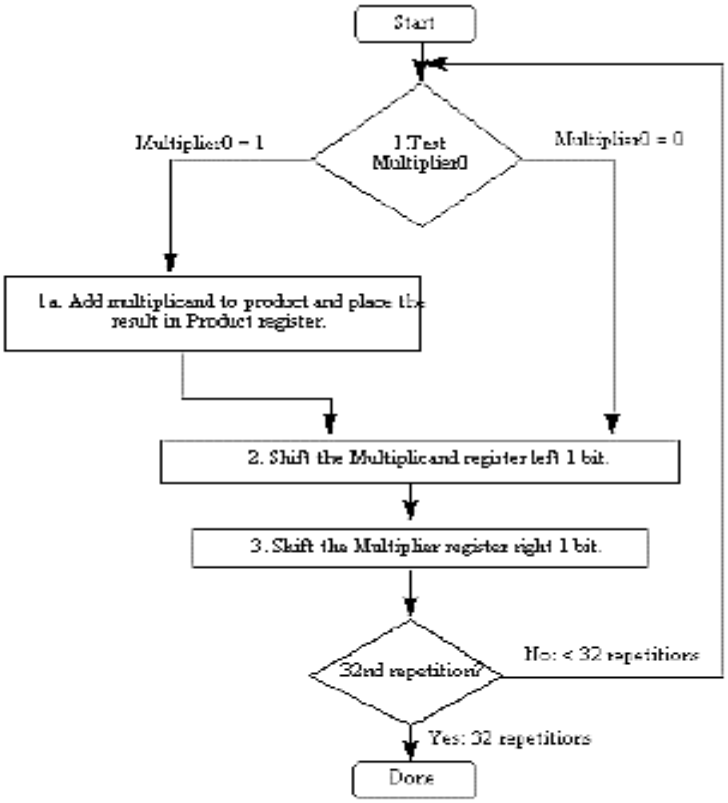
+ 0000

+ 0000

+ 0010 = 0010two x + 1000two

00001110two

Flowchart of booth's multiplication Algorithm :



Multiply Algorithm

Mathematical Model:

Let, S be the System Such that,

A={S, E, I, O, F, DD, NDD, F_min, F_fri, CPU_Core, Mem_Shared, success, failure} Where,

S= Start state,

E= End State,

I= Set of Input

O= Set of Out put

F =Set of Function

DD=Deterministic Data

NDD=Non Deterministic Data

F_Min=Main Function

F_Fri= Friend Function

CPU_Core= No of CPU Core.

Mem_Shared=Shared Memory.

Function:

1) Splitting Function = this function is used for splitting unsorted list.

2) Sorting Function = this function is used for sorting list.

3) Binary Search = this function apply binary search on sorted list.

Success Case: It is the case when all the inputs are given by system are entered correctly. Failure Case: It is

the case when the input does not match the validation Criteria.

Conclusion: Thus we have implemented booths algorithm using small cluster of BBB.

Steps to be followed for successful exection of program

Prerequisites:

First disable the firewall on the machine used for running server code.

Check the IP addres and port number in socket's connect() and bind() functions in all the files.

Make sure all the code have the IP and port number of the machine running the server code.

Steps:

Run the server code first and enter the multiplicand an multiplier in integer base 10 format.(The usual numbers like -5, 3 ,4...)

Run the multiplicand code next on another machine.

Run the multiplier code on yet another machine.

1. The multiplicand code will display the values of A and S. It will also display the packed string which is sent back to the server.

2. The multiplier code will display the value of P

3. The server code will display the connection received from both the clients. Finally, it also outputs the answer in binary format.

Code:

server_booth.py

```
# take values A,S from clients and run the main loop for calculating P

import socket

def addition(op1, op2):
        # length of P and A and S is same.
        result=""
        carry="0"
        for i in range(len(op1)-1, -1, -1):         #run reverse loop
                if op1[i]=="1" and op2[i]=="1":
                        if carry=="1":
                                result="1"+result
                                carry="1"
                        else:                                   #carry = 0
                                result="0"+result
                                carry="1"

                elif op1[i]=="0" and op2[i]=="0":
                        if carry=="1":
                                result="1"+result
                                carry="0"
                        else:                                   #carry = 0
                                result="0"+result
                                carry="0"

                elif op1[i]=="0" and op2[i]=="1":
                        if carry=="1":
                                result="0"+result
                                carry="1"
                        else:                                   #carry = 0
                                result="1"+result
                                carry="0"

                else:                                           # 1, 0
                        if carry=="1":
                                result="0"+result
                                carry="1"
                        else:                                   #carry = 0
                                result="1"+result
                                carry="0"
        return result

s = socket.socket()         # Create a socket object
s.bind(("192.168.6.80", 9001))        # Bind to the port


M=int(input("Enter a multiplicant:"))
R=int(input("Enter a  multiplier:"))
M, R=bin(M), bin(R)
print "Binary representation: ", M, R

s.listen(2)                     # Now wait for client connection.
client, addr = s.accept()     # Establish connection with client.
print 'Got connection from', addr

client2, addr2 = s.accept()
print 'Got connection from', addr2
```

```
'''
Send the value of both. Client will return A, S. It will also return length_R as first param.
<Length_R>A<A>S<S>
Send the value of length of R and value of R. Client will return P.      P<P>
'''
client.send(M+" "+R)

AS=client.recv(1024)  # recv A, S
index_A=AS.index("A")
index_S=AS.index("S")
A=AS[index_A+1:index_S]
S=AS[index_S+1:]

length_R=int(AS[:index_A])
client2.send(str(length_R)+" "+R)
P=client2.recv(1024)  # recv P
index_P=P.index("P")
P=P[index_P+1:]
P_length=len(P)

#we've got A,S,P in strings
for i in range(length_R):

        last_two_digits=P[P_length-2:P_length]

        if last_two_digits == "01":
                #add A in P and store that in P and ignore overflows
                P=addition(A, P)
        elif last_two_digits == "10":
                #add S in P aND store the result in P and IGNORE OVerflows
                P=addition(S, P)

        #print "After addn", P
        #arithmetic right shift (copy the sign bit as well). Start looping from the right
most digits
        P=P[0]+P[0:P_length-1]

P=P[:P_length-1]
print P


client _multiplier.py
# calculate value P and send it back to server.

import socket

def twos_comp(binM):
        S = [int(x) for x in binM]
        flag = 0
        for i in range(len(S)-1, -1, -1):
                if flag==1:
                        #invert
                        if S[i]==1:
                                S[i]=0
                        else:
                                S[i]=1
                        continue

                if S[i]==1:
                        flag=1
        return S
```

```
s = socket.socket()         # Create a socket object
s.connect(("192.168.6.80", 9001))
temp=s.recv(1024)
temp=temp.split()
length_R, R= int(temp[0]), temp[1]

if R[0]=="-":
        R=R[3:]
        #origR=R
        R=twos_comp(R)
        R=[str(x) for x in R]
        R=''.join(R)
        for i in range (length_R-len(R)):
                R="1"+R
else:
        R=R[2:]
        for i in range (length_R-len(R)):
                R="0"+R
        #flag_R=2

P = []
for i in range(2*length_R + 1):
        P.append(0)

print "check length of P: ", P

for i in range(len(R)):
        P[length_R+i]=int(R[i])

P=[str (x) for x in P]
P="".join(P)
print P
s.send("P"+P)
```

Client_multiplicand.py

```
# calculate values A, S and send it back to server.

import socket

def twos_comp(binM):
        S = [int(x) for x in binM]
        flag = 0
        for i in range(len(S)-1, -1, -1):
                if flag==1:
                        #invert
                        if S[i]==1:
                                S[i]=0
                        else:
                                S[i]=1
                        continue

                if S[i]==1:
                        flag=1
        return S

s = socket.socket()         # Create a socket object
s.connect(("192.168.6.80", 9001))
temp=s.recv(1024)
temp=temp.split()
M, R=temp[0], temp[1]
```

```
origM, origR="", ""
Max_length=0

flag,flag_R=0,0                     # flag=1: -M, flag=2: M. flag_R=1: -R, flag_R=2: R
if M[0]=="-":
        M=M[3:]
        origM=M
        M=twos_comp(M)
        M=[str(x) for x in M]
        M=''.join(M)
        flag=1
else:
        M=M[2:]
        flag=2

if R[0]=="-":
        R=R[3:]
        origR=R
        R=twos_comp(R)
        R=[str(x) for x in R]
        R=''.join(R)
        flag_R=1
else:
        R=R[2:]
        flag_R=2

if len(M)>= len(R):
        padding=len(M)-len(R)+1 #+1 for sign bit
        if flag==1:
                M="1"+M
        else:
                M="0"+M
        for i in range (padding):
                if flag_R==1:
                        R="1"+R
                else:
                        R="0"+R
        Max_length=len(M)
else:
        padding=len(R)-len(M)+1
        if flag_R==1:
                R="1"+R
        else:
                R="0"+R
        for i in range (padding):
                if flag==1:
                        M="1"+M
                else:
                        M="0"+M
        Max_length=len(R)

print M, R

#now calc A, S using the length of M and R and 1 (lenM+lenR+1)
A = []
for i in range(len(M)+len(R)+1):
        A.append(0)

for i in range(len(M)):
        A[i]=int(M[i])
A=[str(x) for x in A]
print "A: ", A
#A is ready at this point
```

```
if flag==1:           # orignal M was -ve. So we need origM with the minus sign eliinated
        for i in range(Max_length-len(origM)):
                origM="0"+origM
        S=[str(x) for x in origM]

else:
        S=twos_comp(M)

for i in range(len(M)+len(R)+1-len(S)):
        S.append(0)

S=[str(x) for x in S]

#S is ready at this point
print "S: ", S

#pack the A ans S in a buffer string
Send_AS= str(len(R))+"A"+''.join(A) #secret- length of both operands is same. So u can re-
place R with M
Send_AS += "S"+''.join(S)
print Send_AS

#send the A and S to server and the job here is done
s.send(Send_AS)
```

OUTPUT

```
Server side :

root@Student-301:/home/student/Documents/A-5 Booth's Algorithm# python server_booth.py

Enter a multiplicant:5

Enter a  multiplier:2

Binary representation:  0b101 0b10

Got connection from ('192.168.6.67', 36176)

Got connection from ('192.168.6.79', 36224)

00001010

root@Student-301:/home/student/Documents/A-5 Booth's Algorithm#


--------------------------------------------------------------------------------


Client1 :

root@student-302:/home/student/Documents# python client_multiplier.py check length of P:  [0,
0, 0, 0, 0, 0, 0, 0, 0]

000000100

root@student-OptiPlex-3010:/home/student/Documents#


--------------------------------------------------------------------------------


Client2 :

root@student-303:/home/student/Documents# python client_multiplicand.py

0101 0010

A:  ['0', '1', '0', '1', '0', '0', '0', '0', '0']

S:  ['1', '0', '1', '1', '0', '0', '0', '0', '0']

4A010100000S101100000

root@student-303:/home/student/Documents#
```

Assignment No.: 06

Title: Use Business intelligence and analytics tools to recommend the combination of share purchases and sales for maximizing the profit.

Aim: Use Business intelligence and analytics tools to recommend the combination of share purchases and sales for maximizing the profit.Use Business intelligence and analytics tools to recommend the combination of share purchases and sales for maximizing the profit.

Objectives:

- To study the Business intelligence.
- To study the analytics tools.

Theory:

Go to the Fact Orders sheet of the Northwind Traders data workbook. Individually select all values in the columns OrderDate, RequiredDate and ShippedDateand paste them in one single column in a new sheet. The below 2

snapshots will assist you in doing the same.

	OrderID	CustomerSK	EmployeeSK	OrderDate	RequiredDate	ShippedDate	ShipperSK	Freight	ShipName	ShipAddress
1940	10999	42	6	4/3/1998 0:00	5/1/1998 0:00	4/10/1998 0:00	2	96.35	Ottilies Käseladen	Mehrheimerstr. 369
1941	10999	42	6	4/3/1998 0:00	5/1/1998 0:00	4/10/1998 0:00	2	96.35	Ottilies Käseladen	Mehrheimerstr. 369
1942	11000	79	2	4/6/1998 0:00	5/4/1998 0:00	4/14/1998 0:00	3	55.12	Rattlesnake Canyon Grocery	2817 Milton Dr.
1943	11000	79	2	4/6/1998 0:00	5/4/1998 0:00	4/14/1998 0:00	3	55.12	Rattlesnake Canyon Grocery	2817 Milton Dr.
1944	11000	79	2	4/6/1998 0:00	5/4/1998 0:00	4/14/1998 0:00	3	55.12	Rattlesnake Canyon Grocery	2817 Milton Dr.
1945	11001	38	2	4/6/1998 0:00	5/4/1998 0:00	4/14/1998 0:00	2	197.3	Folk och fä HB	Åkergatan 24
1946	11001	38	2	4/6/1998 0:00	5/4/1998 0:00	4/14/1998 0:00	2	197.3	Folk och fä HB	Åkergatan 24
1947	11001	38	2	4/6/1998 0:00	5/4/1998 0:00	4/14/1998 0:00	2	197.3	Folk och fä HB	Åkergatan 24
1948	11001	38	2	4/6/1998 0:00	5/4/1998 0:00	4/14/1998 0:00	2	197.3	Folk och fä HB	Åkergatan 24
1949	11002	83	4	4/6/1998 0:00	5/4/1998 0:00	4/16/1998 0:00	1	141.16	Save-a-lot Markets	187 Suffolk Ln.
1950	11002	83	4	4/6/1998 0:00	5/4/1998 0:00	4/16/1998 0:00	1	141.16	Save-a-lot Markets	187 Suffolk Ln.
1951	11002	83	4	4/6/1998 0:00	5/4/1998 0:00	4/16/1998 0:00	1	141.16	Save-a-lot Markets	187 Suffolk Ln.
1952	11002	83	4	4/6/1998 0:00	5/4/1998 0:00	4/16/1998 0:00	1	141.16	Save-a-lot Markets	187 Suffolk Ln.
1953	11003	29	3	4/6/1998 0:00	5/4/1998 0:00	4/8/1998 0:00	3	14.91	The Cracker Box	55 Grizzly Peak Rd.
1954	11003	29	3	4/6/1998 0:00	5/4/1998 0:00	4/8/1998 0:00	3	14.91	The Cracker Box	55 Grizzly Peak Rd.
1955	11003	29	3	4/6/1998 0:00	5/4/1998 0:00	4/8/1998 0:00	3	14.91	The Cracker Box	55 Grizzly Peak Rd.
1956	11004	71	3	4/7/1998 0:00	5/5/1998 0:00	4/20/1998 0:00	1	44.84	Maison Dewey	Rue Joseph-Bens 532
1957	11004	71	3	4/7/1998 0:00	5/5/1998 0:00	4/20/1998 0:00	1	44.84	Maison Dewey	Rue Joseph-Bens 532
1958	11005	31	2	4/7/1998 0:00	5/5/1998 0:00	4/10/1998 0:00	1	0.75	Wilman Kala	Keskuskatu 45
1959	11005	31	2	4/7/1998 0:00	5/5/1998 0:00	4/10/1998 0:00	1	0.75	Wilman Kala	Keskuskatu 45
1960	11006	19	3	4/7/1998 0:00	5/5/1998 0:00	4/15/1998 0:00	2	25.19	Great Lakes Food Market	2732 Baker Blvd.
1961	11006	19	3	4/7/1998 0:00	5/5/1998 0:00	4/15/1998 0:00	2	25.19	Great Lakes Food Market	2732 Baker Blvd.
1962	11007	77	8	4/8/1998 0:00	5/6/1998 0:00	4/13/1998 0:00	2	202.24	Princesa Isabel Vinhos	Estrada da saúde n. 58
1963	11007	77	8	4/8/1998 0:00	5/6/1998 0:00	4/13/1998 0:00	2	202.24	Princesa Isabel Vinhos	Estrada da saúde n. 58
1964	11007	77	8	4/8/1998 0:00	5/6/1998 0:00	4/13/1998 0:00	2	202.24	Princesa Isabel Vinhos	Estrada da saúde n. 58

Orders | Customer | Employee | Product | Shipper | Time | Fact Orders | Sheet1

Figure 1: Output

Once all the dates are copied, select the data tab, select the Remove Duplicates option and click OK twice. Use the Month and Year Excel functions to fetch the corresponding month numbers and years from the dates. Make sure all the dates have the respective month numbers and years in the adjacent columns as shown below in the 2 snapshots.

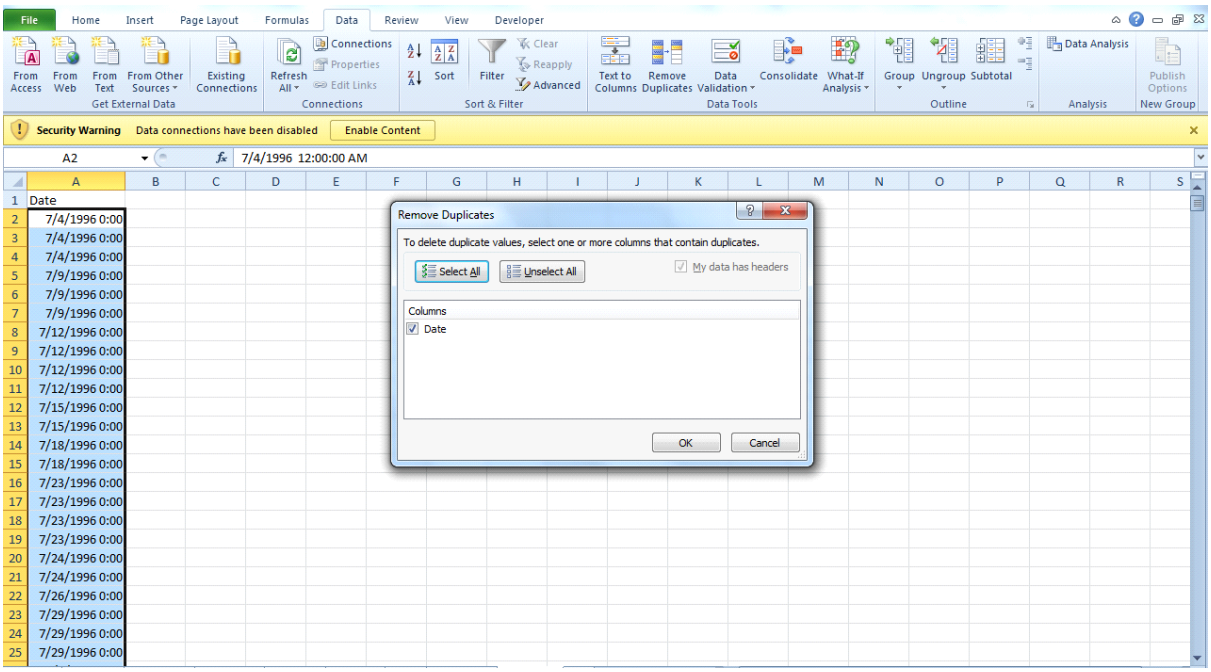
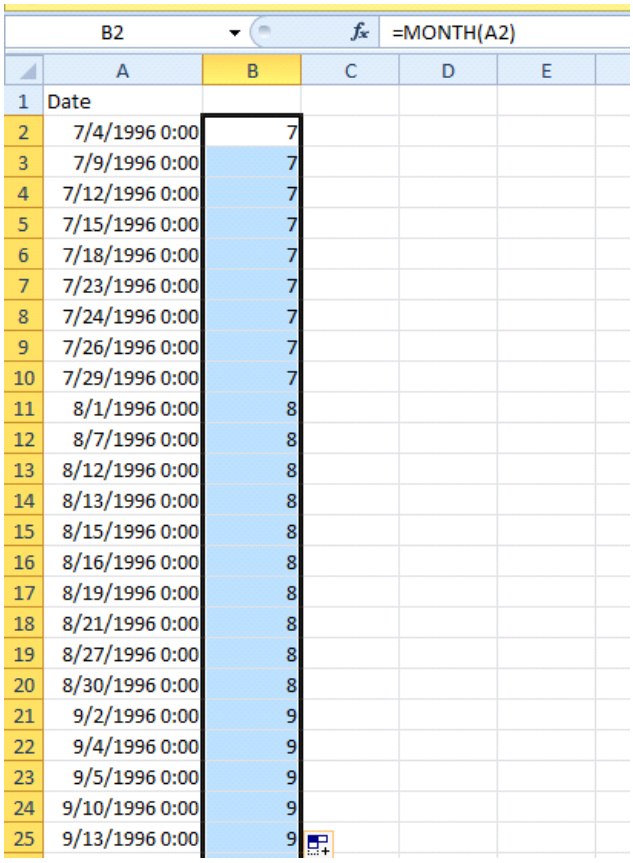


Figure 2:Output



	A	B	C	D	E
1	Date				
2	7/4/1996 0:00	7			
3	7/9/1996 0:00	7			
4	7/12/1996 0:00	7			
5	7/15/1996 0:00	7			
6	7/18/1996 0:00	7			
7	7/23/1996 0:00	7			
8	7/24/1996 0:00	7			
9	7/26/1996 0:00	7			
10	7/29/1996 0:00	7			
11	8/1/1996 0:00	8			
12	8/7/1996 0:00	8			
13	8/12/1996 0:00	8			
14	8/13/1996 0:00	8			
15	8/15/1996 0:00	8			
16	8/16/1996 0:00	8			
17	8/19/1996 0:00	8			
18	8/21/1996 0:00	8			
19	8/27/1996 0:00	8			
20	8/30/1996 0:00	8			
21	9/2/1996 0:00	9			
22	9/4/1996 0:00	9			
23	9/5/1996 0:00	9			
24	9/10/1996 0:00	9			
25	9/13/1996 0:00	9			

Figure 3: Output

C2 =YEAR(A2)

	A	B	C	D
1	Date	Month Number	Year	
2	7/4/1996 0:00	7	1996	
3	7/9/1996 0:00	7	1996	
4	7/12/1996 0:00	7	1996	
5	7/15/1996 0:00	7	1996	
6	7/18/1996 0:00	7	1996	
7	7/23/1996 0:00	7	1996	
8	7/24/1996 0:00	7	1996	
9	7/26/1996 0:00	7	1996	
10	7/29/1996 0:00	7	1996	
11	8/1/1996 0:00	8	1996	
12	8/7/1996 0:00	8	1996	
13	8/12/1996 0:00	8	1996	
14	8/13/1996 0:00	8	1996	
15	8/15/1996 0:00	8	1996	
16	8/16/1996 0:00	8	1996	
17	8/19/1996 0:00	8	1996	
18	8/21/1996 0:00	8	1996	
19	8/27/1996 0:00	8	1996	
20	8/30/1996 0:00	8	1996	
21	9/2/1996 0:00	9	1996	
22	9/4/1996 0:00	9	1996	
23	9/5/1996 0:00	9	1996	
24	9/10/1996 0:00	9	1996	
25	9/13/1996 0:00	9	1996	

Figure 4: Output

Create a dataset in a new excel sheet as shown below:

	Month Number of year	Month Name	Calendar Quarter
1			
2	1	January	Quarter 1
3	2	February	Quarter 1
4	3	March	Quarter 1
5	4	April	Quarter 2
6	5	May	Quarter 2
7	6	June	Quarter 2
8	7	July	Quarter 3
9	8	August	Quarter 3
10	9	September	Quarter 3
11	10	October	Quarter 4
12	11	November	Quarter 4
13	12	December	Quarter 4
14			

Figure 5: Output

Select the entire dataset and name it as?Time in the Name Box as shownbelow. Do not forget to hit Enter once name is typed in the name box.

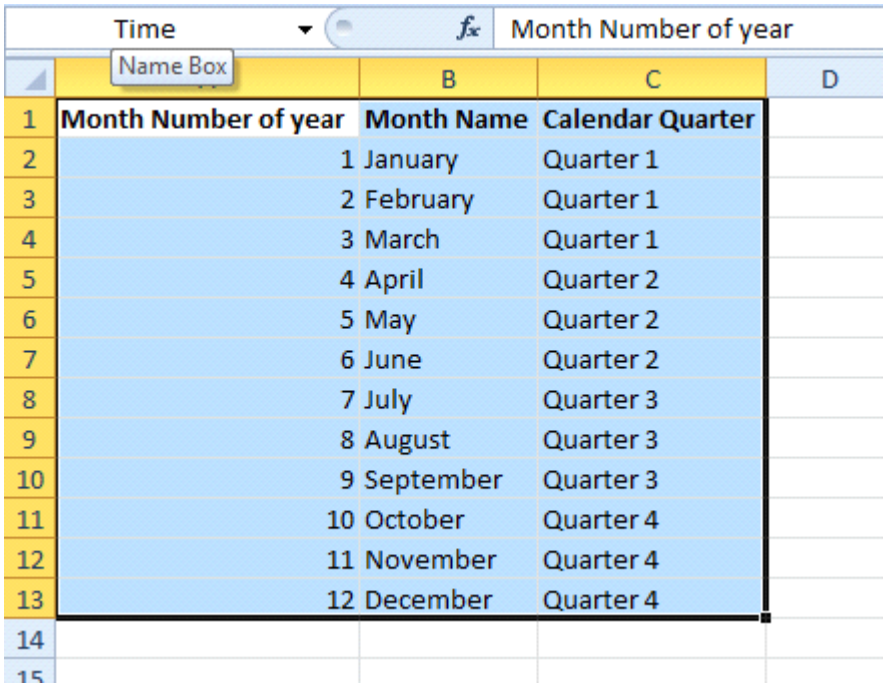


Figure 6: Output

In the sheet where we have the Date, Month Number and Year, add a newcolumn ? ?Month Name? and type the vlookup formula as shown below;

	Date	Month Number	Year	Month Name	Calendar Quarter
2	7/4/1996 0:00	7	1996	=VLOOKUP(	
3	7/9/1996 0:00	7	1996	J(VLOOKUP(**lookup_value**, table_array, col_index_num, [range_lookup])	
4	7/12/1996 0:00	7	1996	July	Quarter 3
5	7/15/1996 0:00	7	1996	July	Quarter 3

Figure 7: Output

Follow the same procedure for Calendar Quarter and use the formula=vlookup(B2,Time,3,False).

Organizing and Analyzing the Data When we have a large amount of data,we want to organize it, analyze it, get summary information and then graph it.We need analytical tools for this, and the PivotTable and PivotChart tools in Microsoft Excel are some of the most popular tools. The data does not need tobe in one workbook. We may analyze data from multiple workbooks withouttoo much trouble.

Basics of PivotTables Pivot Tables allow us to consolidate huge amounts of data with similar fields and analyze the consolidated data or just make asummary of the consolidated data. The PivotTable in Excel gives us a simpleway to create a PivotTable for our data. Please note that the data should haveat least 1 field in common, or else the consolidation will not work and any spelling error in the data will produce incorrect PivotTables. PivotTables inExcel are synonymous to Cubes in other analytical tools.Basics of Charts are used to graphically represent data. The PivotChart tool in Excel provides a simple way to create a PivotTable and anaccompanying chart. Remember, a chart is only as good as the data or thesummary table (PivotTable). If we try to cram too many fields into a chart,we will end up with a non-informative chart. We must always try and keep itsimple and informative.Let us see some scenarios which explore the different ways of analyzing theNorthwind data.

Scenario 1: Graph the percentage sales over time to see the trends.

Here is the answer:

Sum of Quantity	Column Labels								
Row Labels	Beverages	Condiments	Confections	Dairy Products	Grains/Cereals	Meat/Poultry	Produce	Seafood	Grand Total
Calendar 1996	20.85%	11.69%	15.00%	18.78%	6.30%	8.88%	5.02%	13.49%	100.00%
Calendar 1997	15.80%	11.02%	15.90%	18.23%	10.18%	8.68%	5.76%	14.43%	100.00%
Calendar 1998	21.15%	9.05%	14.96%	17.17%	7.76%	7.45%	5.76%	16.70%	100.00%
Grand Total	**18.38%**	**10.45%**	**15.45%**	**17.95%**	**8.78%**	**8.29%**	**5.65%**	**15.07%**	**100.00%**

Figure 8: Output

Creating a PivotTable and PivotChart

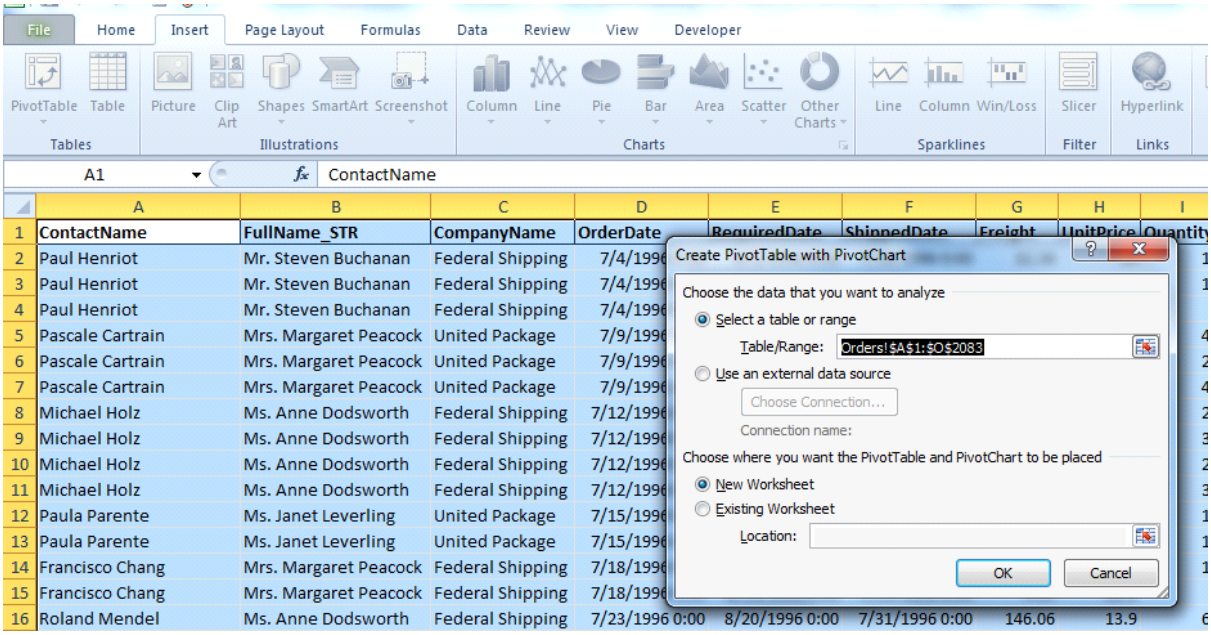
1. Open the Northwind Traders data.xlsx workbook.

2. Go to the Insert tab on the ribbon.

3. Look for the Tables group, and select PivotTable. Choose PivotChart

from the drop-down options.

4. You should now see the Create PivotTable with PivotChart dialog box.



	A	B	C	D	E	F	G	H	I
1	ContactName	FullName_STR	CompanyName	OrderDate	RequiredDate	ShippedDate	Freight	UnitPrice	Quantity
2	Paul Henriot	Mr. Steven Buchanan	Federal Shipping	7/4/1996					
3	Paul Henriot	Mr. Steven Buchanan	Federal Shipping	7/4/1996					
4	Paul Henriot	Mr. Steven Buchanan	Federal Shipping	7/4/1996					
5	Pascale Cartrain	Mrs. Margaret Peacock	United Package	7/9/1996					
6	Pascale Cartrain	Mrs. Margaret Peacock	United Package	7/9/1996					
7	Pascale Cartrain	Mrs. Margaret Peacock	United Package	7/9/1996					
8	Michael Holz	Ms. Anne Dodsworth	Federal Shipping	7/12/1996					
9	Michael Holz	Ms. Anne Dodsworth	Federal Shipping	7/12/1996					
10	Michael Holz	Ms. Anne Dodsworth	Federal Shipping	7/12/1996					
11	Michael Holz	Ms. Anne Dodsworth	Federal Shipping	7/12/1996					
12	Paula Parente	Ms. Janet Leverling	United Package	7/15/1996					
13	Paula Parente	Ms. Janet Leverling	United Package	7/15/1996					
14	Francisco Chang	Mrs. Margaret Peacock	Federal Shipping	7/18/1996					
15	Francisco Chang	Mrs. Margaret Peacock	Federal Shipping	7/18/1996					
16	Roland Mendel	Ms. Anne Dodsworth	Federal Shipping	7/23/1996 0:00	8/20/1996 0:00	7/31/1996 0:00	146.06	13.9	6

Figure 9: Output

5. There are a couple of things you?ll need to de_ne in order to create yourPivotTable. First, you need to tell Excel where to get the data from. Inthe Choose the data you want to analyze section of the dialog box, youwill define where Excel is going to get the data for the PivotTable from.

Since the data for this example is within the workbook, we will chooseSelect a table or range.

(a) Under Table/Range, Excel might be smart enough to automaticallyselect the proper range of data from the worksheet.

(b) If it does not, click on the little button on the right of the range box.Select the range of data. In the example, we have data in cells A1 toO2083. This means that we have selected fifteen columns (A throughO) and 2083 rows.

6. In the Choose where you want the PivotTable and PivotChart to be placedsection of the dialog box, you can choose to either create the PivotTableand the accompanying PivotChart on a new worksheet in the Excel work-book, or you can place both on the current worksheet. For this exercise, wewill choose to create a new worksheet for the PivotTable and PivotChart.To do this, select New Worksheet, and click OK. You should now be takento the new worksheet.

1. While on the new worksheet, you should see four new tabs on the ribbon:

(a) Design, Layout, Format, and Analyze.

(b) You should also see the PivotTable Field List and the PivotChartFilter Pane (if you dont see the panes, click on the Analyze tab inthe ribbon, and, in the Show/Hide group, make sure that at leastField List is selected).

2. You now need to set up the PivotTable so that you can get the data summary you desire. The new tabs, as well as the two panes, will be used tohelp format your PivotTable and PivotChart to do the necessary analysis.For this exercise, let"s see the total number of products sold by category ineach year. In the PivotTable Field List, select Quantity, Year and CategoryName to add to the PivotTable report. Place the CategoryName in the Legends field, Year in the Axis Field and Quantity in the Value field. Youshould see the PivotTable and PivotChart update appropriately. Selectthe chart type as Stacked List to get the chart as shown below

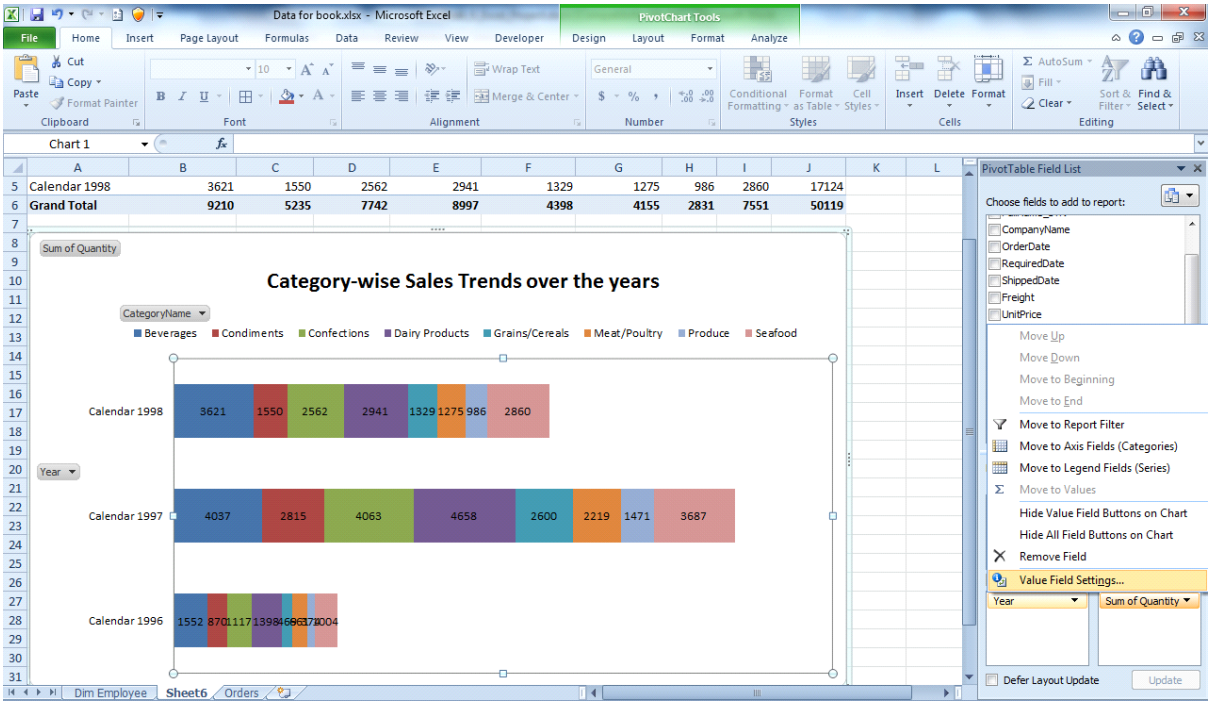
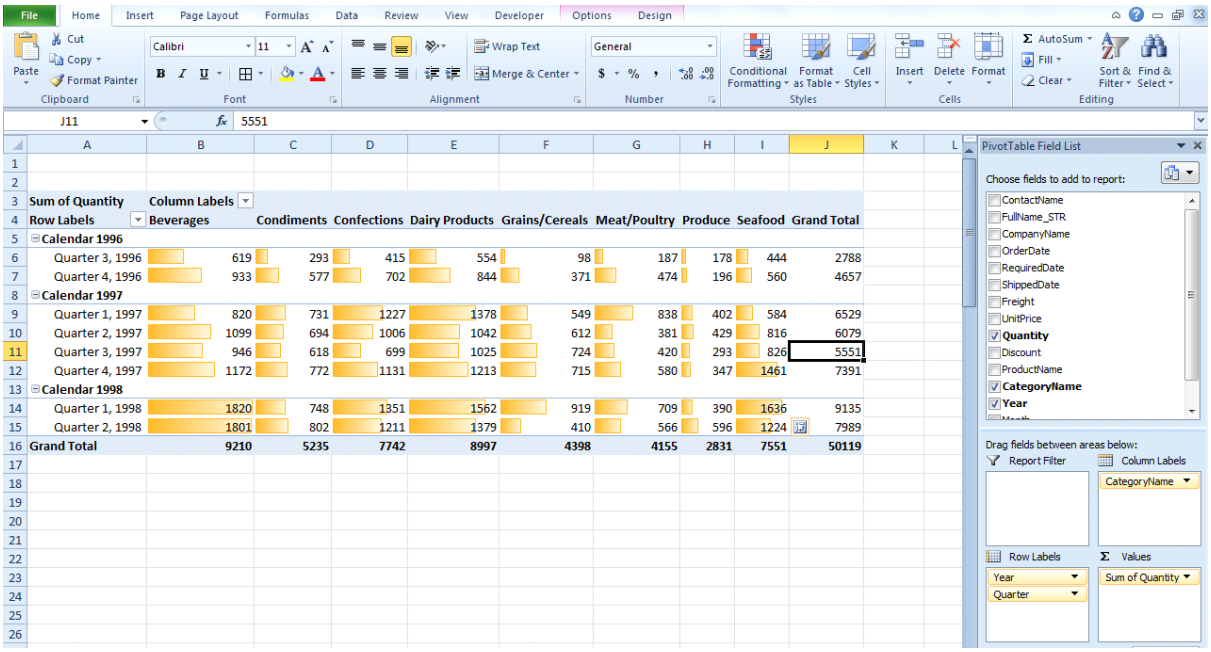


Figure 10: Output

You can use di_erent functions on the _eld placed under Values. For example, if you wanted to see average quantity for each category, you would click on Sum of Quantity and choose Value Field Settings. In the Value Field Settings dialog box, choose the ?Show Values As? tab and select the % of Row Total function to apply to Sum of Quantity (see the screenshot below). A number ofothe functions are available here to use in your PivotTable.

Scenario 2: Pivot the data to see total sales by quarter and category. Arethere any highs? Are there any lows that need to be addressed? Here is the solution:



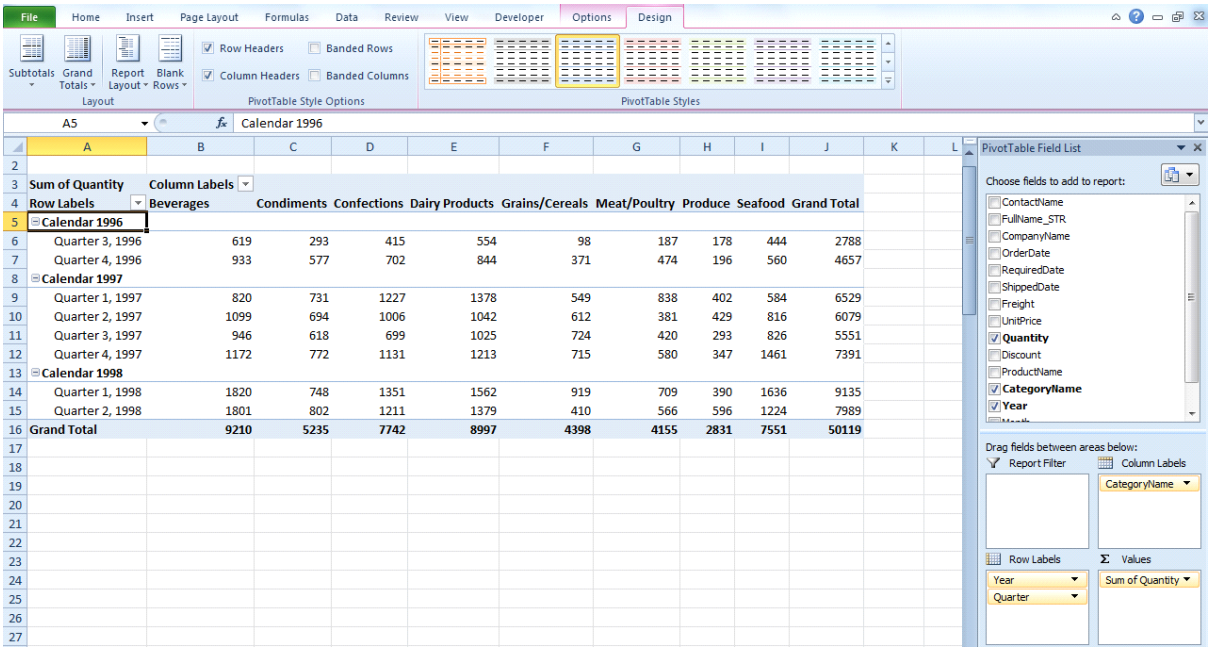
Sum of Quantity	Column Labels								
Row Labels	Beverages	Condiments	Confections	Dairy Products	Grains/Cereals	Meat/Poultry	Produce	Seafood	Grand Total
Calendar 1996									
Quarter 3, 1996	619	293	415	554	98	187	178	444	2788
Quarter 4, 1996	933	577	702	844	371	474	196	560	4657
Calendar 1997									
Quarter 1, 1997	820	731	1227	1378	549	838	402	584	6529
Quarter 2, 1997	1099	694	1006	1042	612	381	429	816	6079
Quarter 3, 1997	946	618	699	1025	724	420	293	826	5551
Quarter 4, 1997	1172	772	1131	1213	715	580	347	1461	7391
Calendar 1998									
Quarter 1, 1998	1820	748	1351	1562	919	709	390	1636	9135
Quarter 2, 1998	1801	802	1211	1379	410	566	596	1224	7989
Grand Total	9210	5235	7742	8997	4398	4155	2831	7551	50119

Figure 11: Output

Here is how we do it:

1. Select the the entire table in the Orders worksheet.

2. Go to the Insert menu, select PivotTable.

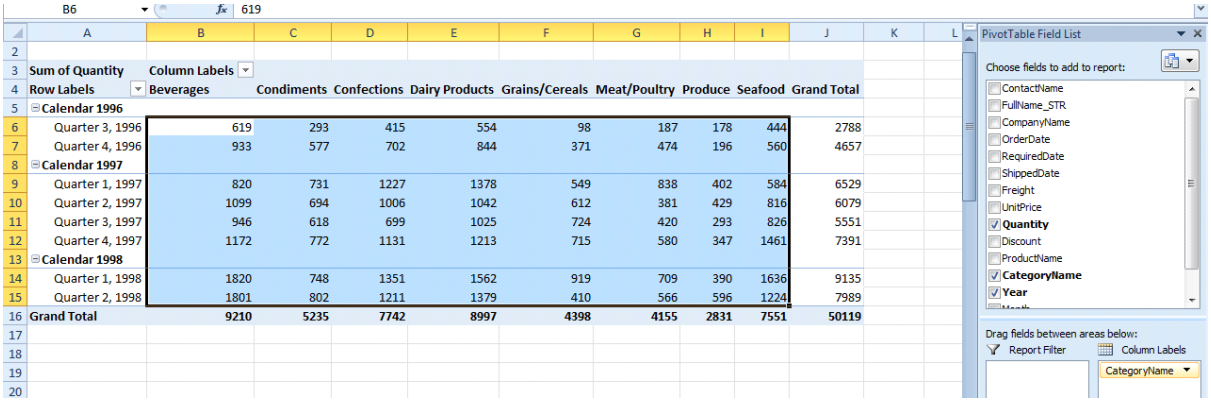


Sum of Quantity	Column Labels								
Row Labels	Beverages	Condiments	Confections	Dairy Products	Grains/Cereals	Meat/Poultry	Produce	Seafood	Grand Total
Calendar 1996									
Quarter 3, 1996	619	293	415	554	98	187	178	444	2788
Quarter 4, 1996	933	577	702	844	371	474	196	560	4657
Calendar 1997									
Quarter 1, 1997	820	731	1227	1378	549	838	402	584	6529
Quarter 2, 1997	1099	694	1006	1042	612	381	429	816	6079
Quarter 3, 1997	946	618	699	1025	724	420	293	826	5551
Quarter 4, 1997	1172	772	1131	1213	715	580	347	1461	7391
Calendar 1998									
Quarter 1, 1998	1820	748	1351	1562	919	709	390	1636	9135
Quarter 2, 1998	1801	802	1211	1379	410	566	596	1224	7989
Grand Total	9210	5235	7742	8997	4398	4155	2831	7551	50119

Figure 12: Output

3. Select the columns as highlighted in the circle below (make sure Quantityis represented as Sum of Quantity. Else use the previously used ValueField Settings option to sum it).

4. Select the data as shown in the screenshot below



Sum of Quantity	Column Labels								
Row Labels	Beverages	Condiments	Confections	Dairy Products	Grains/Cereals	Meat/Poultry	Produce	Seafood	Grand Total
Calendar 1996									
Quarter 3, 1996	619	293	415	554	98	187	178	444	2788
Quarter 4, 1996	933	577	702	844	371	474	196	560	4657
Calendar 1997									
Quarter 1, 1997	820	731	1227	1378	549	838	402	584	6529
Quarter 2, 1997	1099	694	1006	1042	612	381	429	816	6079
Quarter 3, 1997	946	618	699	1025	724	420	293	826	5551
Quarter 4, 1997	1172	772	1131	1213	715	580	347	1461	7391
Calendar 1998									
Quarter 1, 1998	1820	748	1351	1562	919	709	390	1636	9135
Quarter 2, 1998	1801	802	1211	1379	410	566	596	1224	7989
Grand Total	9210	5235	7742	8997	4398	4155	2831	7551	50119

Figure 13: Output

5. Apply the conditional formatting as shown below. (Make sure you are inthe Home menu to view the Conditional Formatting tab)

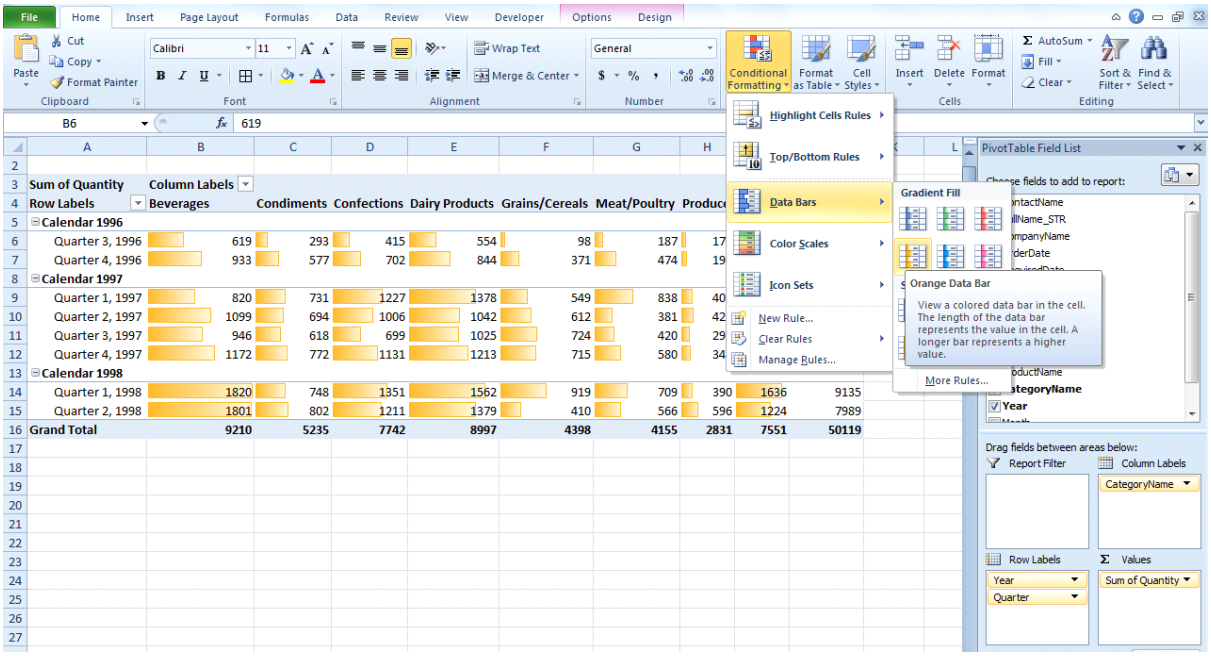
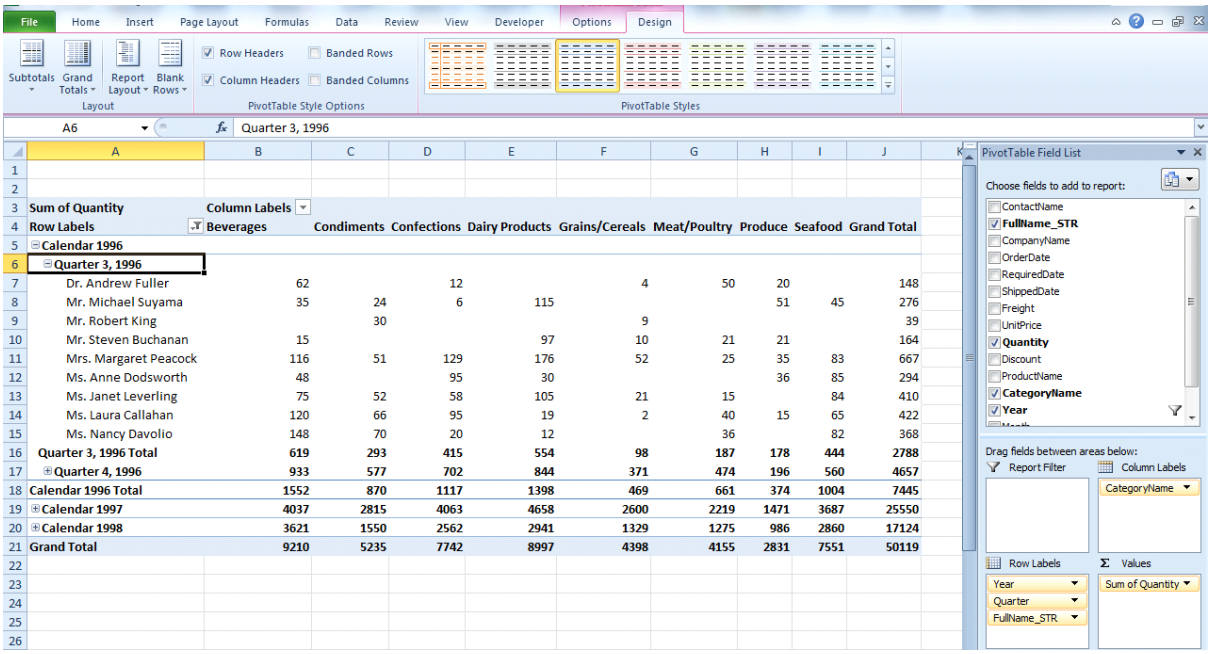


Figure 14: Output

By applying conditional formatting we are able to achieve better visualisationof data. It can be observed that the 2nd quarter of 1998 had the highest totalsales and also the beverages category had the highest total sales for all years,closely followed by dairy products.

Scenario 3: How are quarterly sales totals by salesperson? Subtotal thedata. Here is the answer:



Sum of Quantity	Column Labels								
Row Labels	Beverages	Condiments	Confections	Dairy Products	Grains/Cereals	Meat/Poultry	Produce	Seafood	Grand Total
Calendar 1996									
Quarter 3, 1996									
Dr. Andrew Fuller	62		12		4	50	20		148
Mr. Michael Suyama	35	24	6	115			51	45	276
Mr. Robert King		30			9				39
Mr. Steven Buchanan	15			97	10	21	21		164
Mrs. Margaret Peacock	116	51	129	176	52	25	35	83	667
Ms. Anne Dodsworth	48		95	30			36	85	294
Ms. Janet Leverling	75	52	58	105	21	15		84	410
Ms. Laura Callahan	120	66	95	19	2	40	15	65	422
Ms. Nancy Davolio	148	70	20	12		36		82	368
Quarter 3, 1996 Total	619	293	415	554	98	187	178	444	2788
Quarter 4, 1996	933	577	702	844	371	474	196	560	4657
Calendar 1996 Total	1552	870	1117	1398	469	661	374	1004	7445
Calendar 1997	4037	2815	4063	4658	2600	2219	1471	3687	25550
Calendar 1998	3621	1550	2562	2941	1329	1275	986	2860	17124
Grand Total	9210	5235	7742	8997	4398	4155	2831	7551	50119

Figure 15: Output

Here is how it is done:

Just drag and drop FullName STR (Name of the salesperson) to the Row Label.Then, in the PivotTable tools, select the Design menu, in the Design menu selectthe Subtotals tab and then select the option-Show all Subtotals at Bottom of Group as shown below.

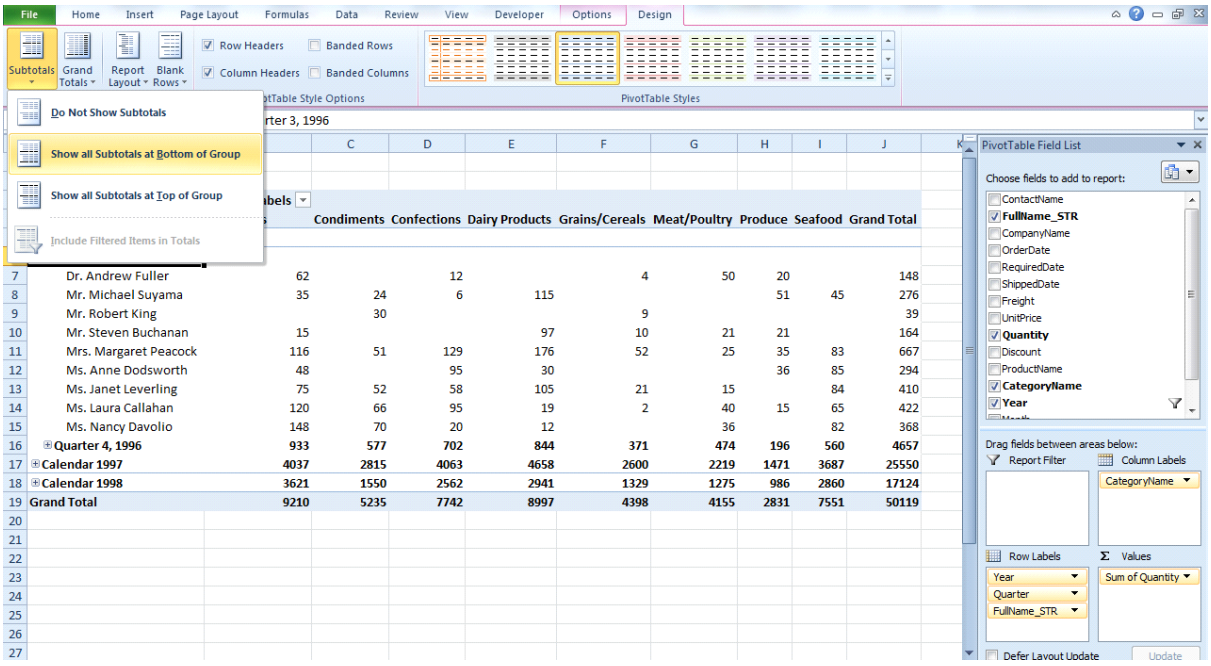
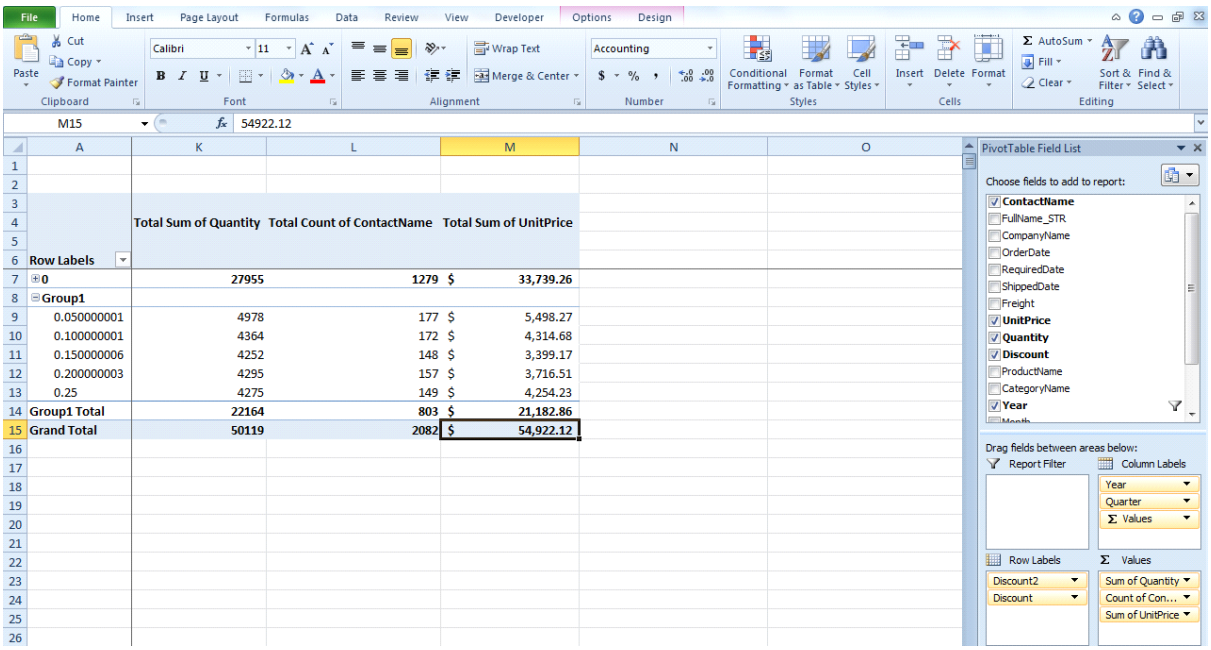


Figure 16: Output

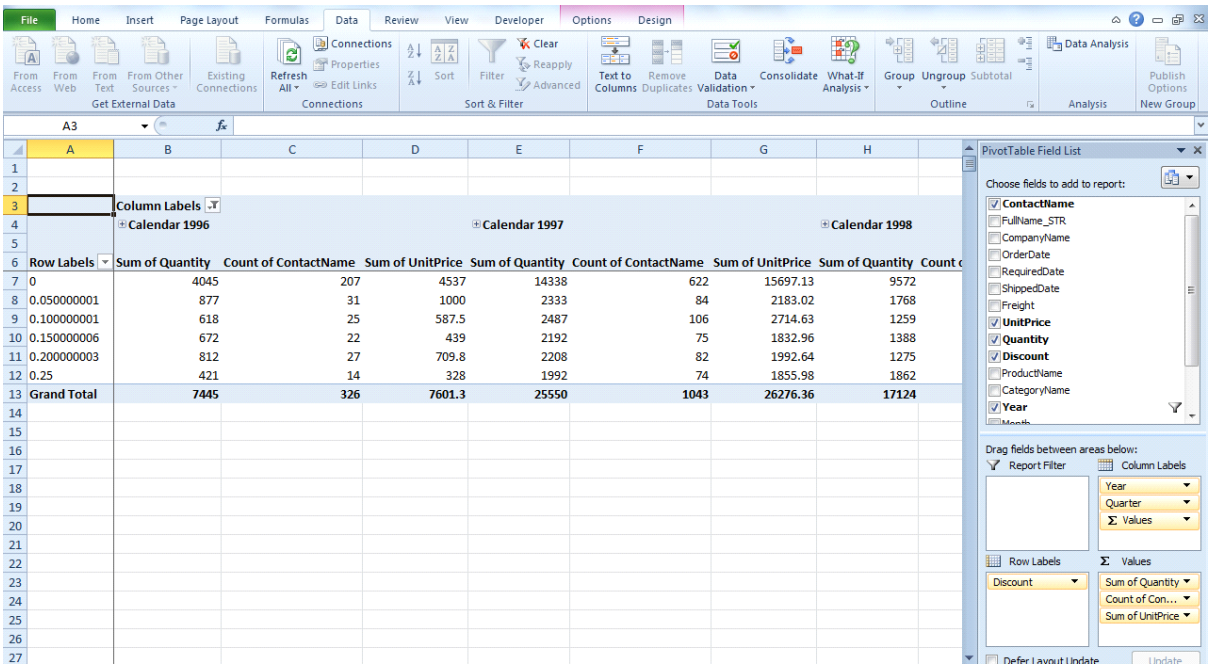
Scenario 4: Is there any increase in sales when the products are sold ata discounted rate? Here is the answer: In the Pivot Table, drag the Year and



Row Labels	Total Sum of Quantity	Total Count of ContactName	Total Sum of UnitPrice
0	27955	1279	$ 33,739.26
Group1			
0.050000001	4978	177	$ 5,498.27
0.100000001	4364	172	$ 4,314.68
0.150000006	4252	148	$ 3,399.17
0.200000003	4295	157	$ 3,716.51
0.25	4275	149	$ 4,254.23
Group1 Total	22164	803	$ 21,182.86
Grand Total	50119	2082	$ 54,922.12

Figure 17: Output

the Quarter as Column Labels, Discount as the Row Labels and Quantity andUnit Price as Values. To see how many customers bought something at differentdiscount rates drag the ContactName to Values. You will see the ContactNameautomatically changing to Count of ContactName. The screenshot is shownbelow for reference.



	Column Labels	Calendar 1996			Calendar 1997			Calendar 1998	
Row Labels	Sum of Quantity	Count of ContactName	Sum of UnitPrice	Sum of Quantity	Count of ContactName	Sum of UnitPrice	Sum of Quantity	Count
0	4045	207	4537	14338	622	15697.13	9572	
0.050000001	877	31	1000	2333	84	2183.02	1768	
0.100000001	618	25	587.5	2487	106	2714.63	1259	
0.150000006	672	22	439	2192	75	1832.96	1388	
0.200000003	812	27	709.8	2208	82	1992.64	1275	
0.25	421	14	328	1992	74	1855.98	1862	
Grand Total	7445	326	7601.3	25550	1043	26276.36	17124	

Figure 18: Output

Except for 0, ctrl select the row label and right click. You will see an optioncalled Group in the menu that appears. Select that option as shown below.Once it is selected, you will be able to see the performance of products with and without discounts.

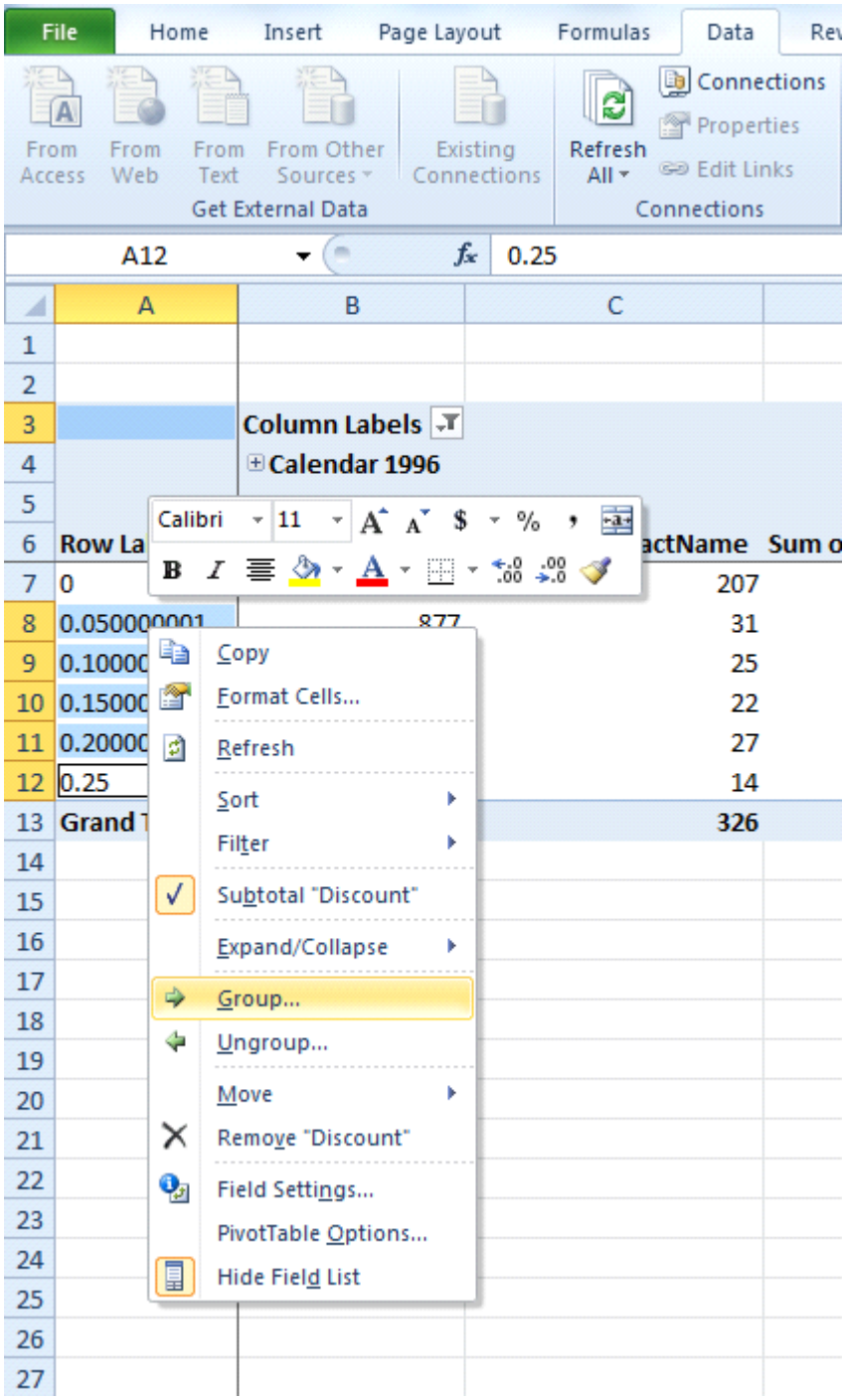
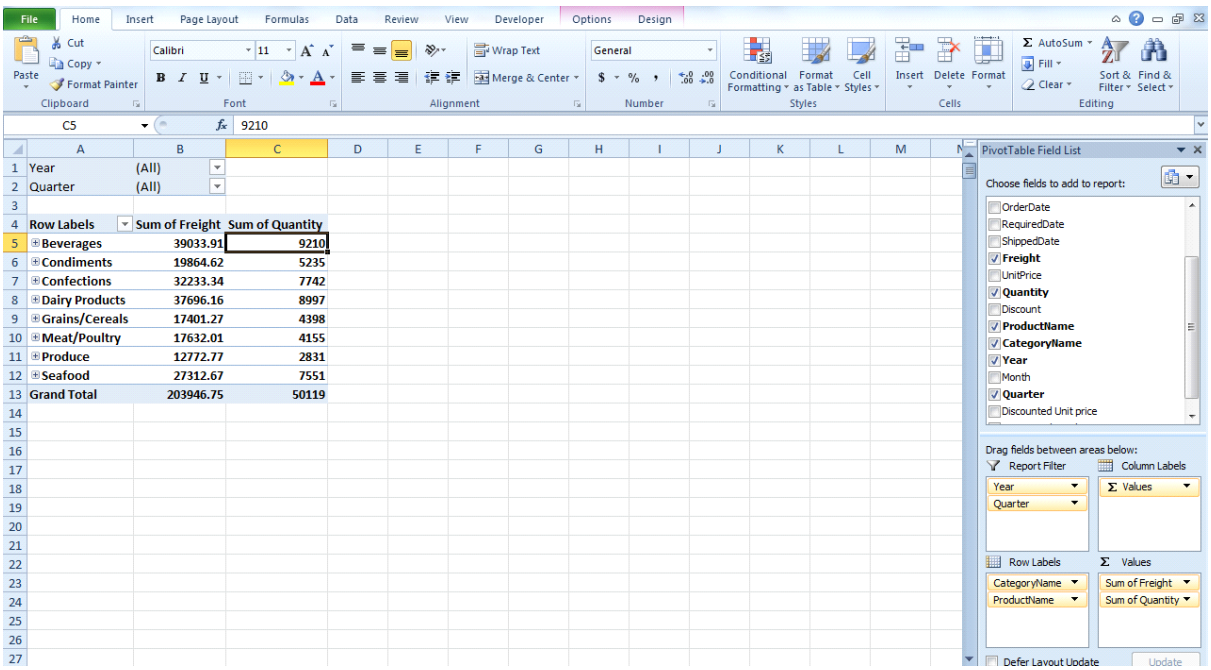


Figure 19: Output

Scenario 5: Report the sales by category and the corresponding freightcharges. Filtering should be enabled in the Year and Quarter columns, and theselected Year and Quarter need to be visible.We will take a slightly different approach to

solve this scenario. Please take alook at the below screenshot. To see the individual Years and Quarters, theconcept of slicers is used.



	A	B	C
1	Year	(All)	
2	Quarter	(All)	
3			
4	Row Labels	Sum of Freight	Sum of Quantity
5	Beverages	39033.91	9210
6	Condiments	19864.62	5235
7	Confections	32233.34	7742
8	Dairy Products	37696.16	8997
9	Grains/Cereals	17401.27	4398
10	Meat/Poultry	17632.01	4155
11	Produce	12772.77	2831
12	Seafood	27312.67	7551
13	Grand Total	203946.75	50119

Figure 20: Output

Let‟s see how it is done: Click on the Options menuwhich gets highlighted under PivotTable Options. Click on the Insert Slicer.You will get to see the Insert Slicers window as shown. Select the Year andQuarter and click OK.

Scenario 6: Sort the Sales data in terms of Year, Quarter and Month.

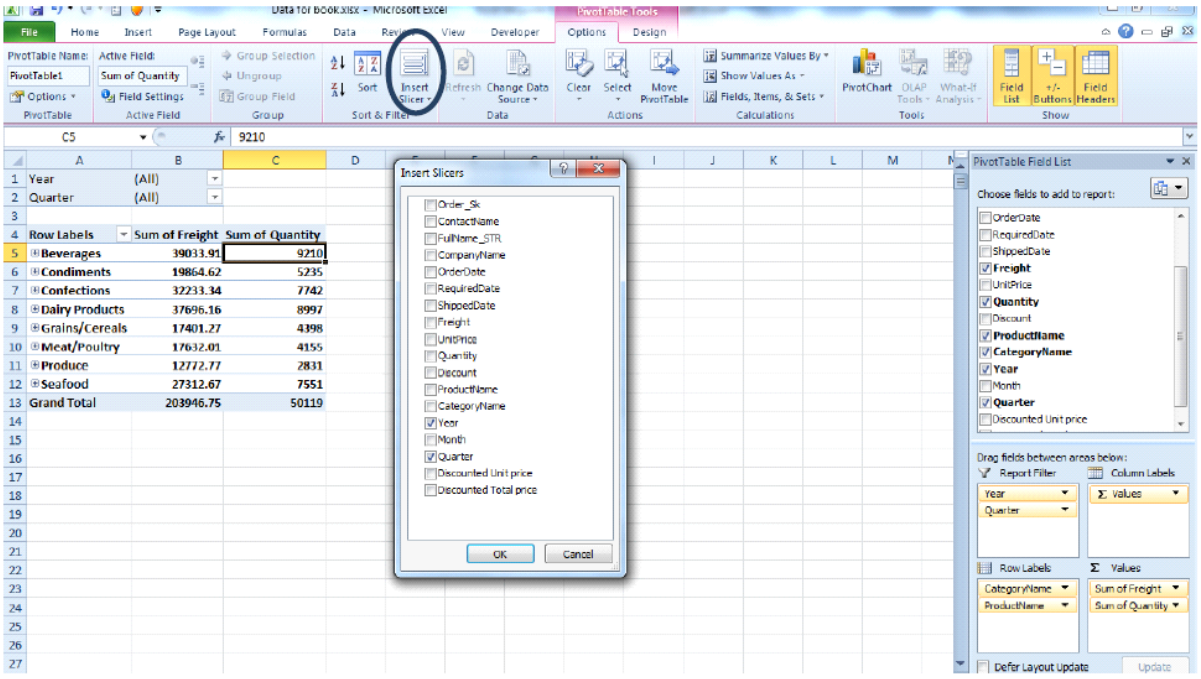


Figure 21: Output

While splitting the components of a cell make sure that the Quarter columnis cut and pasted in the neighboring column because the two outputs of thesplit will occupy the adjacent columns and if the Quarter column is not moved as described, it will be overwritten by the data as shown.

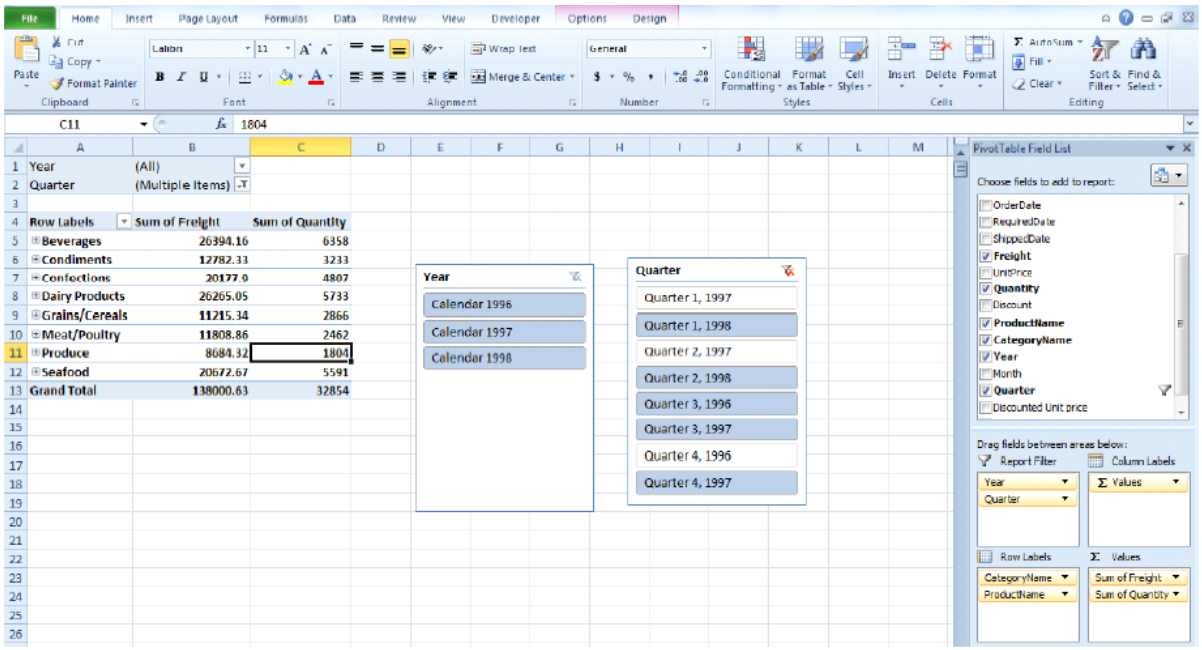


Figure 22: Output

Once the above steps are achieved, sort the Month column by Month first as shown below.

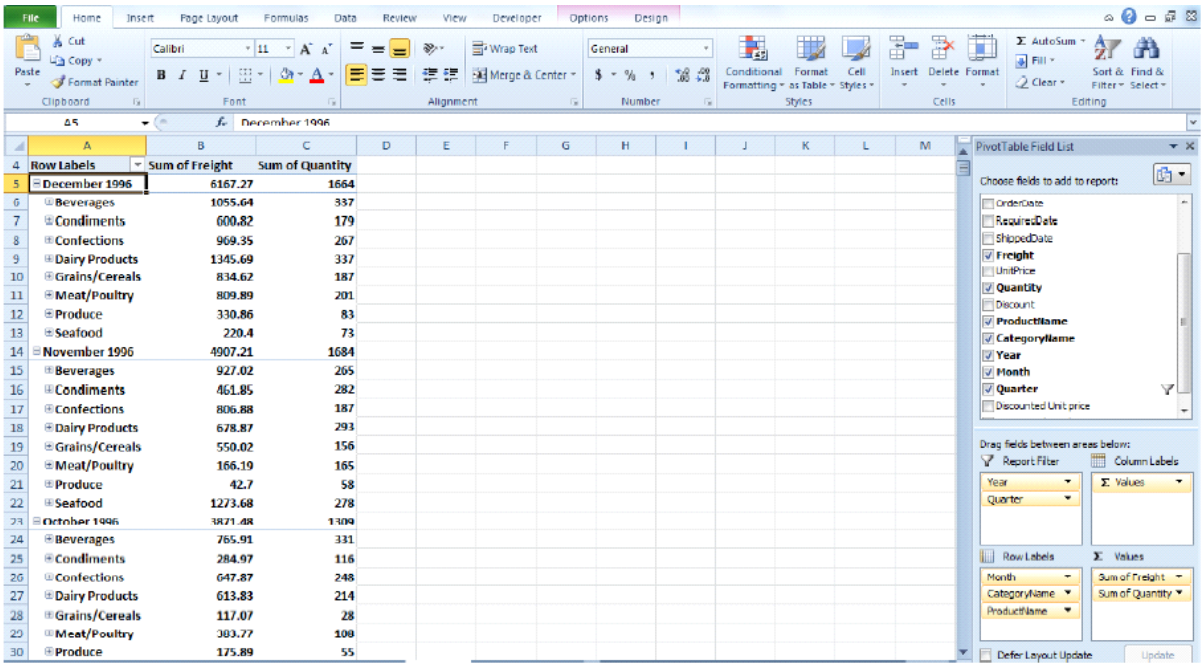
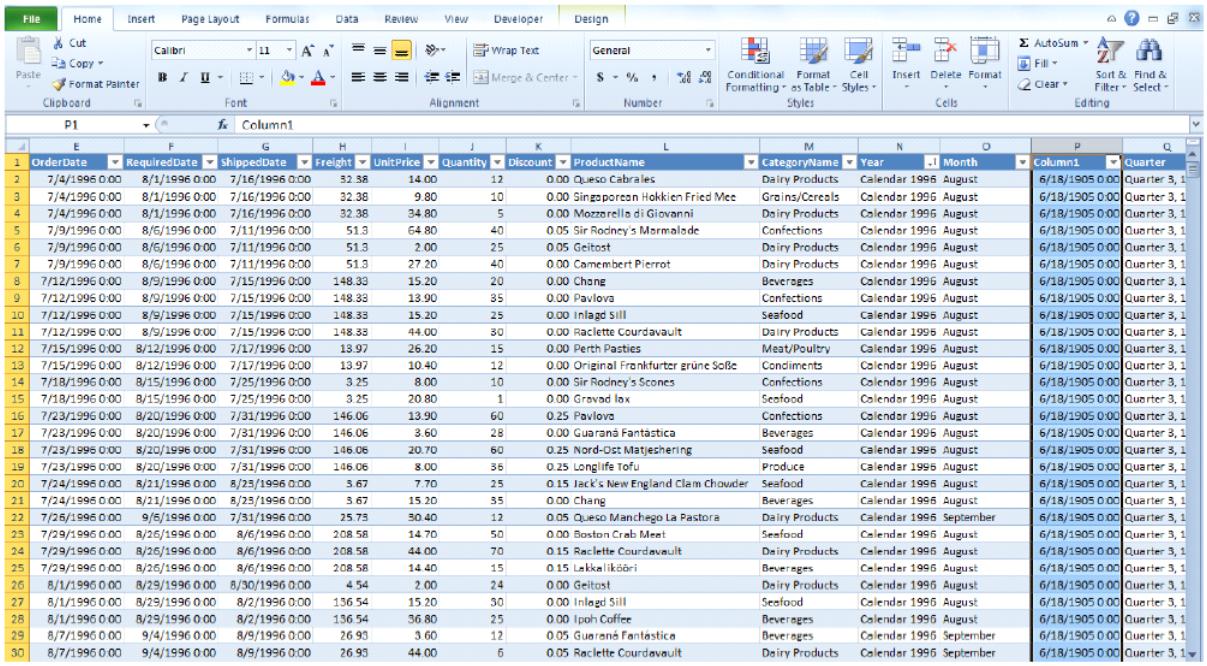


Figure 23: Output



OrderDate	RequiredDate	ShippedDate	Freight	UnitPrice	Quantity	Discount	ProductName	CategoryName	Year	Month	Column1	Quarter
7/4/1996 0:00	8/1/1996 0:00	7/16/1996 0:00	32.38	14.00	12	0.00	Queso Cabrales	Dairy Products	Calendar 1996	August	6/18/1905 0:00	Quarter 3, 1
7/4/1996 0:00	8/1/1996 0:00	7/16/1996 0:00	32.38	9.80	10	0.00	Singaporean Hokkien Fried Mee	Grains/Cereals	Calendar 1996	August	6/18/1905 0:00	Quarter 3, 1
7/4/1996 0:00	8/1/1996 0:00	7/16/1996 0:00	32.38	34.80	5	0.00	Mozzarella di Giovanni	Dairy Products	Calendar 1996	August	6/18/1905 0:00	Quarter 3, 1
7/9/1996 0:00	8/6/1996 0:00	7/11/1996 0:00	51.3	64.80	40	0.05	Sir Rodney's Marmalade	Confections	Calendar 1996	August	6/18/1905 0:00	Quarter 3, 1
7/9/1996 0:00	8/6/1996 0:00	7/11/1996 0:00	51.3	2.00	25	0.05	Geitost	Dairy Products	Calendar 1996	August	6/18/1905 0:00	Quarter 3, 1
7/9/1996 0:00	8/6/1996 0:00	7/11/1996 0:00	51.3	27.20	40	0.00	Camembert Pierrot	Dairy Products	Calendar 1996	August	6/18/1905 0:00	Quarter 3, 1
7/12/1996 0:00	8/9/1996 0:00	7/15/1996 0:00	148.33	15.20	20	0.00	Chang	Beverages	Calendar 1996	August	6/18/1905 0:00	Quarter 3, 1
7/12/1996 0:00	8/9/1996 0:00	7/15/1996 0:00	148.33	13.90	35	0.00	Pavlova	Confections	Calendar 1996	August	6/18/1905 0:00	Quarter 3, 1
7/12/1996 0:00	8/9/1996 0:00	7/15/1996 0:00	148.33	15.20	25	0.00	Inlagd Sill	Seafood	Calendar 1996	August	6/18/1905 0:00	Quarter 3, 1
7/12/1996 0:00	8/9/1996 0:00	7/15/1996 0:00	148.33	44.00	30	0.00	Raclette Courdavault	Dairy Products	Calendar 1996	August	6/18/1905 0:00	Quarter 3, 1
7/15/1996 0:00	8/12/1996 0:00	7/17/1996 0:00	13.97	26.20	15	0.00	Perth Pasties	Meat/Poultry	Calendar 1996	August	6/18/1905 0:00	Quarter 3, 1
7/15/1996 0:00	8/12/1996 0:00	7/17/1996 0:00	13.97	10.40	12	0.00	Original Frankfurter grüne Soße	Condiments	Calendar 1996	August	6/18/1905 0:00	Quarter 3, 1
7/18/1996 0:00	8/15/1996 0:00	7/25/1996 0:00	3.25	8.00	10	0.00	Sir Rodney's Scones	Confections	Calendar 1996	August	6/18/1905 0:00	Quarter 3, 1
7/18/1996 0:00	8/15/1996 0:00	7/25/1996 0:00	3.25	20.80	1	0.00	Gravad lax	Seafood	Calendar 1996	August	6/18/1905 0:00	Quarter 3, 1
7/23/1996 0:00	8/20/1996 0:00	7/31/1996 0:00	146.06	13.90	60	0.25	Pavlova	Confections	Calendar 1996	August	6/18/1905 0:00	Quarter 3, 1
7/23/1996 0:00	8/20/1996 0:00	7/31/1996 0:00	146.06	3.60	28	0.00	Guaraná Fantástica	Beverages	Calendar 1996	August	6/18/1905 0:00	Quarter 3, 1
7/23/1996 0:00	8/20/1996 0:00	7/31/1996 0:00	146.06	20.70	60	0.25	Nord-Ost Matjeshering	Seafood	Calendar 1996	August	6/18/1905 0:00	Quarter 3, 1
7/23/1996 0:00	8/20/1996 0:00	7/31/1996 0:00	146.06	8.00	36	0.25	Longlife Tofu	Produce	Calendar 1996	August	6/18/1905 0:00	Quarter 3, 1
7/24/1996 0:00	8/21/1996 0:00	8/23/1996 0:00	3.67	7.70	25	0.15	Jack's New England Clam Chowder	Seafood	Calendar 1996	August	6/18/1905 0:00	Quarter 3, 1
7/24/1996 0:00	8/21/1996 0:00	8/23/1996 0:00	3.67	15.20	35	0.00	Chang	Beverages	Calendar 1996	August	6/18/1905 0:00	Quarter 3, 1
7/26/1996 0:00	9/6/1996 0:00	7/31/1996 0:00	25.73	30.40	12	0.05	Queso Manchego La Pastora	Dairy Products	Calendar 1996	September	6/18/1905 0:00	Quarter 3, 1
7/29/1996 0:00	8/26/1996 0:00	8/6/1996 0:00	208.58	14.70	50	0.00	Boston Crab Meat	Seafood	Calendar 1996	August	6/18/1905 0:00	Quarter 3, 1
7/29/1996 0:00	8/26/1996 0:00	8/6/1996 0:00	208.58	44.00	70	0.15	Raclette Courdavault	Dairy Products	Calendar 1996	August	6/18/1905 0:00	Quarter 3, 1
7/29/1996 0:00	8/26/1996 0:00	8/6/1996 0:00	208.58	14.40	15	0.15	Lakkalikööri	Beverages	Calendar 1996	August	6/18/1905 0:00	Quarter 3, 1
8/1/1996 0:00	8/29/1996 0:00	8/30/1996 0:00	4.54	2.00	24	0.00	Geitost	Dairy Products	Calendar 1996	August	6/18/1905 0:00	Quarter 3, 1
8/1/1996 0:00	8/29/1996 0:00	8/2/1996 0:00	136.54	15.20	30	0.00	Inlagd Sill	Seafood	Calendar 1996	August	6/18/1905 0:00	Quarter 3, 1
8/1/1996 0:00	8/29/1996 0:00	8/2/1996 0:00	136.54	36.80	25	0.00	Ipoh Coffee	Beverages	Calendar 1996	August	6/18/1905 0:00	Quarter 3, 1
8/7/1996 0:00	9/4/1996 0:00	8/9/1996 0:00	26.93	3.60	12	0.05	Guaraná Fantástica	Beverages	Calendar 1996	September	6/18/1905 0:00	Quarter 3, 1
8/7/1996 0:00	9/4/1996 0:00	8/9/1996 0:00	26.93	44.00	6	0.05	Raclette Courdavault	Dairy Products	Calendar 1996	September	6/18/1905 0:00	Quarter 3, 1

Figure 24: Output

If analysis based on the discounts is required, add 2 columns as explained below and create a new Pivot Table on this sheet. First go to the Orderssheet and create 2 new columns as shown:

Column 1: Discounted Unit Price:This column will contain the unit prices after discount. So the formula wouldbe (1-Discount)(Unit Price). In terms of excel columns the formula would be (1-J2)*(H2).

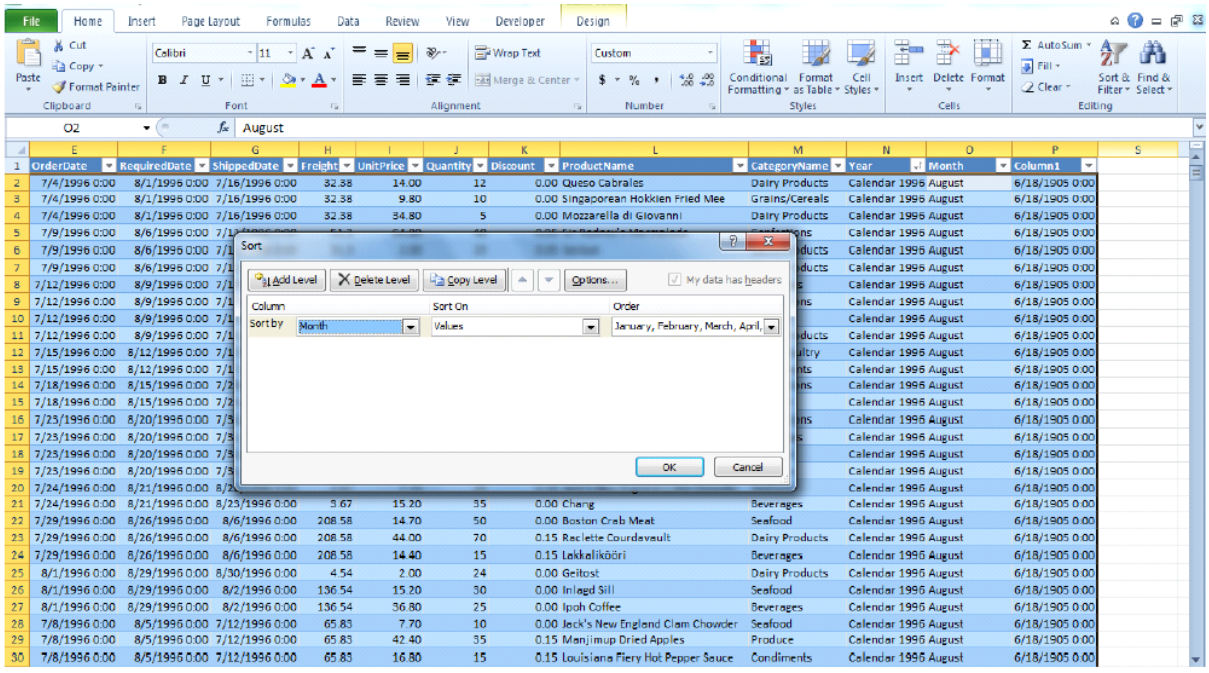
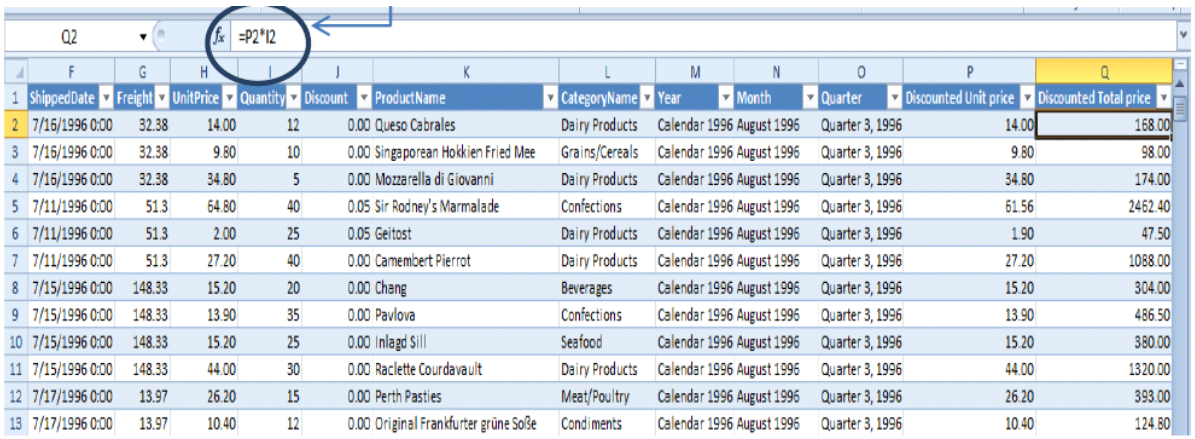


Figure 25: Output

Column 2: Discounted Total Price: This column would containdiscounted unit price multiplied by quantity. In terms of excel columns, it wouldbe P2*I2.



Q2 =P2*I2

	F	G	H	I	J	K	L	M	N	O	P	Q
1	ShippedDate	Freight	UnitPrice	Quantity	Discount	ProductName	CategoryName	Year	Month	Quarter	Discounted Unit price	Discounted Total price
2	7/16/1996 0:00	32.38	14.00	12	0.00	Queso Cabrales	Dairy Products	Calendar 1996	August 1996	Quarter 3, 1996	14.00	168.00
3	7/16/1996 0:00	32.38	9.80	10	0.00	Singaporean Hokkien Fried Mee	Grains/Cereals	Calendar 1996	August 1996	Quarter 3, 1996	9.80	98.00
4	7/16/1996 0:00	32.38	34.80	5	0.00	Mozzarella di Giovanni	Dairy Products	Calendar 1996	August 1996	Quarter 3, 1996	34.80	174.00
5	7/11/1996 0:00	51.3	64.80	40	0.05	Sir Rodney's Marmalade	Confections	Calendar 1996	August 1996	Quarter 3, 1996	61.56	2462.40
6	7/11/1996 0:00	51.3	2.00	25	0.05	Geitost	Dairy Products	Calendar 1996	August 1996	Quarter 3, 1996	1.90	47.50
7	7/11/1996 0:00	51.3	27.20	40	0.00	Camembert Pierrot	Dairy Products	Calendar 1996	August 1996	Quarter 3, 1996	27.20	1088.00
8	7/15/1996 0:00	148.33	15.20	20	0.00	Chang	Beverages	Calendar 1996	August 1996	Quarter 3, 1996	15.20	304.00
9	7/15/1996 0:00	148.33	13.90	35	0.00	Pavlova	Confections	Calendar 1996	August 1996	Quarter 3, 1996	13.90	486.50
10	7/15/1996 0:00	148.33	15.20	25	0.00	Inlagd Sill	Seafood	Calendar 1996	August 1996	Quarter 3, 1996	15.20	380.00
11	7/15/1996 0:00	148.33	44.00	30	0.00	Raclette Courdavault	Dairy Products	Calendar 1996	August 1996	Quarter 3, 1996	44.00	1320.00
12	7/17/1996 0:00	13.97	26.20	15	0.00	Perth Pasties	Meat/Poultry	Calendar 1996	August 1996	Quarter 3, 1996	26.20	393.00
13	7/17/1996 0:00	13.97	10.40	12	0.00	Original Frankfurter grüne Soße	Condiments	Calendar 1996	August 1996	Quarter 3, 1996	10.40	124.80

Figure 26: Output

Conclusion:

We have successfully studied the tool used for business intelligence and analytics tools to recommend the combination of share purchases and sales for maximizing the profit

Assignment No: B2

Title: Implementation of Strassen's matrix multiplication.

Aim: Concurrent implementation of Strassen's Multiplication using BBB HPC or equivalent infrastructure. Use Java/ Python/ Scala/ C++ as programming language.

Objectives:

- To understand Strasson‟s matrix multiplication.
- To understand clustering

Theory:

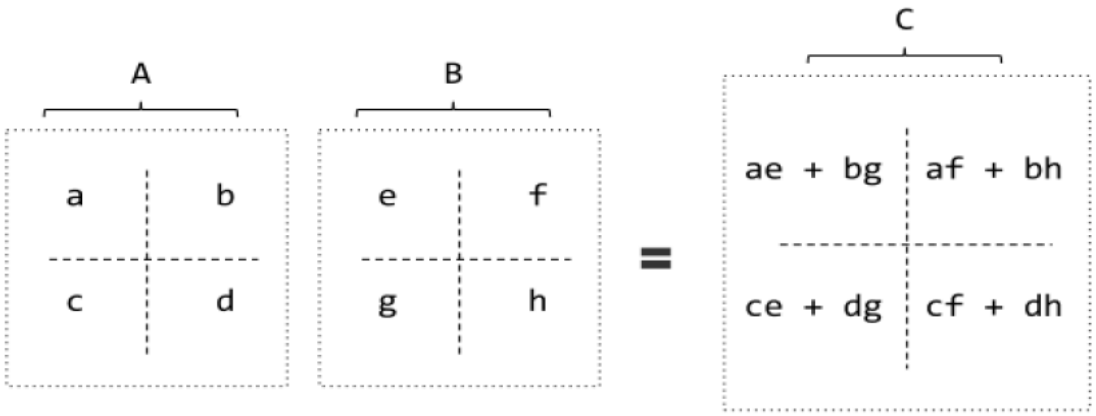
The Strassen's method of matrix multiplication is a typical divide and conquer algorithm. We've seen so far some divide and conquer algorithms like merge sort and the Karatsuba's fast multiplication of large numbers. However let's get again on what's behind the divide and conquer approach.

Unlike the dynamic programming where we "expand" the solutions of sub-problems in order to get the final solution, here we are talking more on joining sub-solutions together. These solutions of some sub-problems of the general problem are equal and their merge is somehow well defined.

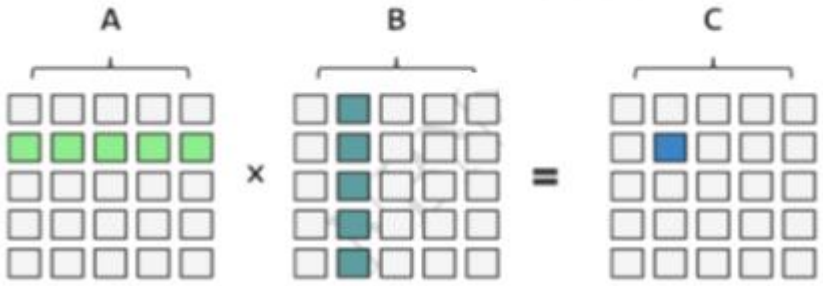
A typical example is the merge sort algorithm. In merge sort we have two sorted arrays and all we want is to get the array representing their union again sorted. Of course, the tricky part in merge sort is the merging itself. That's because we've to pass through the two arrays, A and B, and we've to compare each "pair" of items representing an item from A and from B. A bit off topic, but this is the weak point of merge sort and although its worst-case time complexity is O(n.log(n)), quicksort is often preferred in practice because there's no "merge". Quicksort just concatenates the two sub-arrays. Note that in quicksort the sub-arrays aren't with an equal length in general and although its worst-case time complexity is O(n^2) it often outperforms merge sort.

This simple example from the paragraph above shows us how sometimes merging the solutions of two sub-problems actually isn't a trivial task to do. Thus we must be careful when applying any divide and conquer approach.

DIVIDE AND CONQUER



MATRIX MULTIPLICATION




```
C[i][j] = sum(A[i][k] * B[k][j]) for k = 0 ... n
```

```
In our case:
C[1][1] =>
A[1][0]*B[0][1] + A[1][1]*B[1][1] + A[1][2]*B[2][1] + A[1][3]*B[3][1] + A[1][4]*B[4][1]
```

Algorithm :

$$\begin{vmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{vmatrix} = \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} \begin{vmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{vmatrix}$$

P1 = (A11+ A22)(B11+B22)
P2 = (A21 + A22) * B11
P3 = A11 * (B12 - B22)
P4 = A22 * (B21 - B11)
P5 = (A11 + A12) * B22
P6 = (A21 - A11) * (B11 + B12)
P7 = (A12 - A22) * (B21 + B22)

C11 = P1 + P4 - P5 + P7

C12 = P3 + P5
C21 = P2 + P4
C22 = P1 + P3 - P2 + P6

Mathematical Model:

- M= {s, e, i, o, n, F, Success, Failure}
- s = Start state = input the two matrices for multiplication.
- e = End state = multiplication displayed bt strassons multiplication.
- o is the required output i.e. answer matrix.
- n = Number of processes = {hostname of processors in a file}
- F is the set of functions required for Strassons matrix multiplication = {f1, f2}
- f1 = {send data to processor for computation}
- f2 = {receive data to parent processor for result}
- i= input the two matrices for multiplication.
- o={answer matrix}
- Success – multiplication displayed bt strassons multiplication.
- Failure-ϕ

Conclusion: We have successfully studied and implemented Strassen's matrix multiplication.

Code:

```
#include<stdio.h>
#include<mpi.h>

int main(int argc, char* argv[])
{
   int i,j;
   int m1,m2,m3,m4,m5,m6,m7;
   int rank,size;

   MPI_Request request;
   MPI_Status status;

   MPI_Init(&argc,&argv);
       MPI_Comm_rank(MPI_COMM_WORLD,&rank);
       MPI_Comm_size(MPI_COMM_WORLD,&size);
   if(rank == 0)
   {
         int a[2][2],b[2][2];
         printf("Enter the 4 elements of first matrix: ");
         for(i=0;i<2;i++)
             for(j=0;j<2;j++)
                  scanf("%d",&a[i][j]);

         printf("Enter the 4 elements of second matrix: ");
         for(i=0;i<2;i++)
             for(j=0;j<2;j++)
                  scanf("%d",&b[i][j]);

         printf("\nThe first matrix is\n");
         for(i=0;i<2;i++)
         {
             printf("\n");
             for(j=0;j<2;j++)
             {
                  printf("%d\t",a[i][j]);
             }
         }

         printf("\nThe second matrix is\n");
         for(i=0;i<2;i++)
         {
             printf("\n");
             for(j=0;j<2;j++)
             {
                  printf("%d\t",b[i][j]);
             }
         }

         m1= (a[0][0] + a[1][1])*(b[0][0]+b[1][1]);
         MPI_Isend(&m1,1,MPI_INT,1,2,MPI_COMM_WORLD,&request);

         m2= (a[1][0]+a[1][1])*b[0][0];
         MPI_Isend(&m2,1,MPI_INT,1,3,MPI_COMM_WORLD,&request);

         m3= a[0][0]*(b[0][1]-b[1][1]);
         MPI_Isend(&m3,1,MPI_INT,1,4,MPI_COMM_WORLD,&request);

         m4= a[1][1]*(b[1][0]-b[0][0]);
         MPI_Isend(&m4,1,MPI_INT,1,5,MPI_COMM_WORLD,&request);

         m5= (a[0][0]+a[0][1])*b[1][1];
         MPI_Isend(&m5,1,MPI_INT,1,6,MPI_COMM_WORLD,&request);
```

```
        m6= (a[1][0]-a[0][0])*(b[0][0]+b[0][1]);
        MPI_Isend(&m6,1,MPI_INT,1,7,MPI_COMM_WORLD,&request);

        m7= (a[0][1]-a[1][1])*(b[1][0]+b[1][1]);
        MPI_Isend(&m7,1,MPI_INT,1,8,MPI_COMM_WORLD,&request);

    }

    if(rank == 1)
    {
        int c[2][2];

        MPI_Irecv(&m1,1,MPI_INT,0,2,MPI_COMM_WORLD,&request);
        MPI_Irecv(&m2,1,MPI_INT,0,3,MPI_COMM_WORLD,&request);
        MPI_Irecv(&m4,1,MPI_INT,0,5,MPI_COMM_WORLD,&request);

        MPI_Wait(&request,&status);
        c[1][0]=m2+m4;

        MPI_Irecv(&m3,1,MPI_INT,0,4,MPI_COMM_WORLD,&request);
        MPI_Irecv(&m5,1,MPI_INT,0,6,MPI_COMM_WORLD,&request);
        MPI_Wait(&request,&status);
        c[0][1]=m3+m5;


        MPI_Irecv(&m6,1,MPI_INT,0,7,MPI_COMM_WORLD,&request);

        MPI_Wait(&request,&status);
        c[1][1]=m1-m2+m3+m6;

        MPI_Irecv(&m7,1,MPI_INT,0,8,MPI_COMM_WORLD,&request);

        MPI_Wait(&request,&status);
        c[0][0]=m1+m4-m5+m7;

        printf("\nAfter multiplication using \n");
        for(i=0;i<2;i++)
        {
            printf("\n");
            for(j=0;j<2;j++)
            {
                    printf("%d\t",c[i][j]);
            }
        }
      printf("\n");
    }

    MPI_Finalize();
    return 0;
}
```

```
Output

mpiu@DB21:/mirror/mpiu/CL4 prog/B-2$ mpicc stasson_mpi.c

mpiu@DB21:/mirror/mpiu/CL4 prog/B-2$ mpiexec -n 2 -f machinefile ./a.out

Enter the 4 elements of first matrix: 1 2 3 4

Enter the 4 elements of second matrix: 5 6 7 8


The first matrix is


1       2

3       4

The second matrix is


5       6

7       8

After multiplication using


19      22

43      50

mpiu@DB21:/mirror/mpiu/CL4 prog/B-2$ mpicc stasson_mpi.c

mpiu@DB21:/mirror/mpiu/CL4 prog/B-2$ mpiexec -n 2 -f machinefile ./a.out

Enter the 4 elements of first matrix: 1 2 2 1

Enter the 4 elements of second matrix: 1 2 2 1


The first matrix is


1       2

2       1

The second matrix is


1       2

2       1

After multiplication using
```

```
5       4

4       5

mpiu@DB21:/mirror/mpiu/CL4 prog/B-2$
```

Assignment No. B3

Title: Implement a stack sampling using threads using VTune Amplifier.

Aim: Write a program to develop a stack sampling using threads using VTune Amplifier

Objectives:

- To understand sampling and VTune Amplifier.

Theory:

Intel VTune Amplifier is a commercial application for software performance analysis. Although basic features work on both Intel and AMD hardware, advanced hardware-based sampling requires an Intel-manufactured CPU. It is available as part of Intel Parallel Studio or as a stand-alone product.

Code optimization

VTune Amplifier assists in various kinds of code profiling including stack sampling, thread profiling and hardware event sampling. The profiler result consists of details such as time spent in each sub routine which can be drilled down to the instruction level. The times taken by the instructions are indicative of any stalls in the pipeline during instruction execution. The tool can be also used to analyze thread performance.

Features

1. **Software sampling**
   Works on x86 compatible processors and gives both the locations where time is spent and the call stack used.

2. **JIT profiling support**
   Profiles dynamically generated code.

3. **Locks and waits analysis**
   Finds long synchronization waits that occur when cores are underutilized.

4. **Threading timeline**
   Shows thread relationships to identify load balancing and synchronization issues. It can also be used to select a region of time and filter the results. This can remove the clutter of data gathered during uninteresting times like application start-up.

5. **Source view**
   Sampling results are displayed line by line on the source / assembly code.

6. **Hardware event sampling**
   This uses the on chip performance monitoring unit and requires an Intel processor. It can find specific tuning opportunities like cache misses and branch mispredictions.

7. Performance tuning utility (PTU)

PTU Was a separate download that gave VTuneAmplifer XE users access to experimental tuning technology. This includes things like Data Access Analysis that identifies memory hotspots and relates them to code hotspots. PTU now is fully integrated into VTuneAmplifer XE.

Performance Profiling with Intel VTune Amplifier XE

System Requirements

Processor requirements

The list of supported processors is constantly being extended. Here is a partial list of processors where the EBS analysis is enabled:

Mobile Processors

- Intel® Atom™ Processor
- Intel® Core™ i7 Mobile Processor Extreme Edition (including 2nd, 3rd, 4th and 5th Generation Intel® Core™ processors)
- Intel® Core™ i7, i5, i3 Mobile Processors (including 2nd, 3rd, 4th, 5th and 6th Generation Intel® Core™ processors)
- Intel® Core™2 Extreme Mobile Processor
- Intel® Core™2 Quad Mobile Processor
- Intel® Core™2 Duo Mobile Processor
- Intel® Pentium® Mobile Processor

Desktop Processors

- Intel® Atom™ Processor
- Intel® Core™ i7 Desktop Processor Extreme Edition (including 2nd, 3rd, 4th and 5th Generation Intel® Core™ processors)
- Intel® Core™ i7, i5, i3 Desktop Processors (including 2nd, 3rd, 4th, 5th and 6th Generation Intel® Core™ processors)
- Intel® Core™2 Quad Desktop Processor
- Intel® Core™2 Extreme Desktop Processor
- Intel® Core™2 Duo Desktop Processor

Server and Workstation Processors

- Intel® Xeon® processors E7 family (including v2 and v3)
- Intel® Xeon® processor E5 family (including v2 and v3)
- Intel® Xeon® processors E3 family (including v2, v3 and v4)
- Intel® Xeon® Processor D family
- Intel® Xeon® Processor 7000 Sequence
- Intel® Xeon® Processor 6000 Sequence

- Intel® Xeon® Processor 5000 Sequence
- Intel® Xeon® Processor 3000 Sequence
- Intel® Xeon Phi™ Coprocessors
- Quad-Core Intel® Xeon® processors 7xxx, 5xxx, and 3xxx series
- Dual-Core Intel® Xeon® processors 7xxx, 5xxx, and 3xxx series

System Memory Requirements

- At least 2 GB of RAM

Disk Space Requirements

- 900 MB free disk space required for all product features and all architectures

Software Requirements

1. **Supported Linux distributions:**

- Red Hat* Enterprise Linux 5, 6 and 7 [1]
- CentOS* versions equivalent to Red Hat* Enterprise Linux* versions listed above
- SUSE* Linux* Enterprise Server (SLES) 11 and 12
- Fedora* 221 and 232 □ Ubuntu* 12.04, 14.04 and 15.1004
- Debian* 7.0 and 8.0

2. **Supported compilers:**

- Intel® C/C++ Compiler 11 and higher
- Intel® Fortran Compiler 11 and higher
- GNU C/C++ Compiler 3.4.6 and higher

3. **Supported programming languages:**

- Fortran
- C
- C++
- Java*
- OpenCL*

Installation steps

1. Unzip the tar file
2. After Unzipping, enter the unzipped folder and copy the path of that folder using RIGHT CLICK + PROPERTIES.
3. In the terminal, enter the following command: cd <path> (path: the path you copied in the previous step)
4. Enter the next command : ./install_GUI.sh
5. Installation window will open; continue the installation. (SIMPLE) 6. Installation done!

Steps to use VTune Amplifier.

1. Go to Computer in Files; Enter opt folder, enter vtune_apmlifier<version> folder.
2. Copy the path of that folder using RIGHT CLICK + PROPERTIES.
3. Open a new terminal; Enter the following command: source <path>/amplxe-vars.sh (path: Paste the path copied in the previous step)
4. Enter the next command: source <path>/amplxe-vars.sh&
5. You have now entered into the Vtune Amplifier Interpreter.
6. Enter 'amplxe-gui'
7. Vtune Amplifier Application Opens!

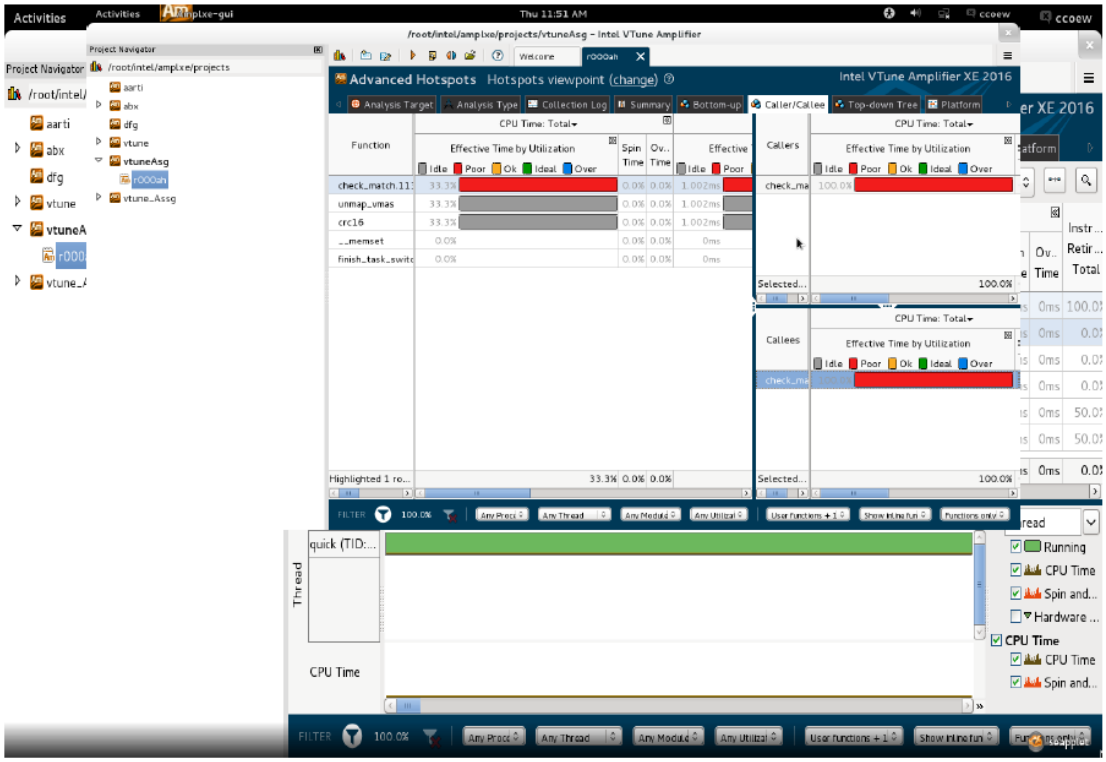
Conclusion: Hence we have studied stack sampling using threads and using VTune.

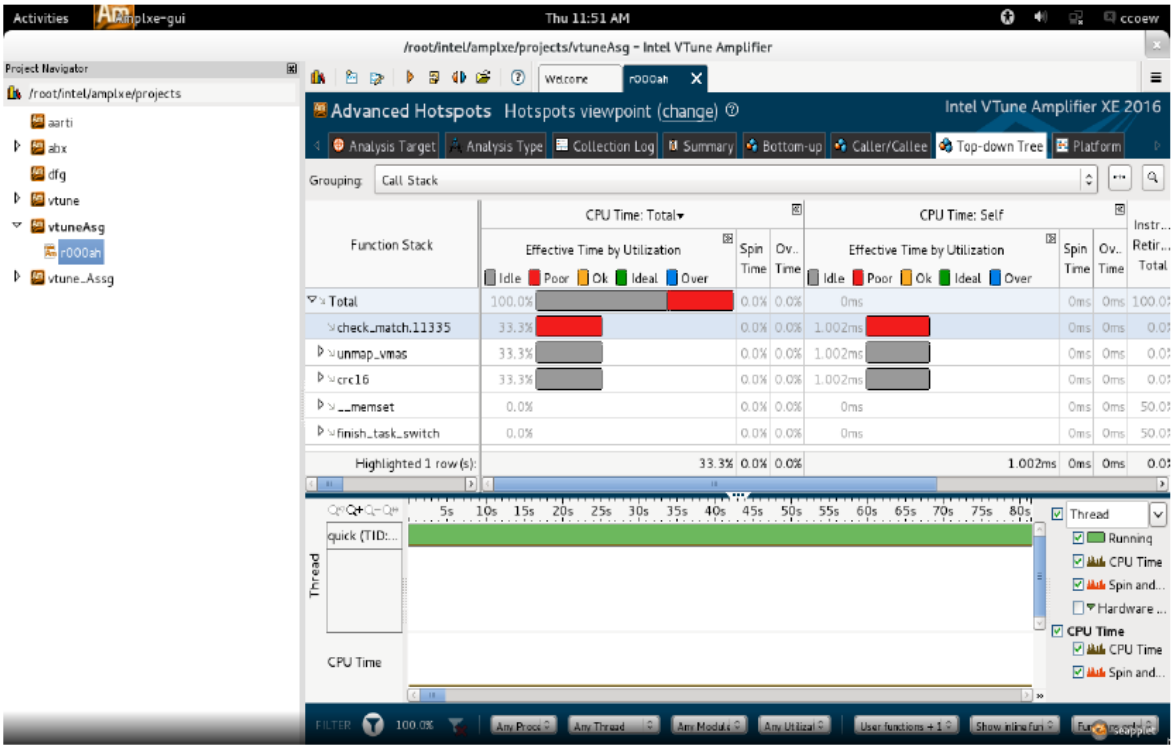
Code:

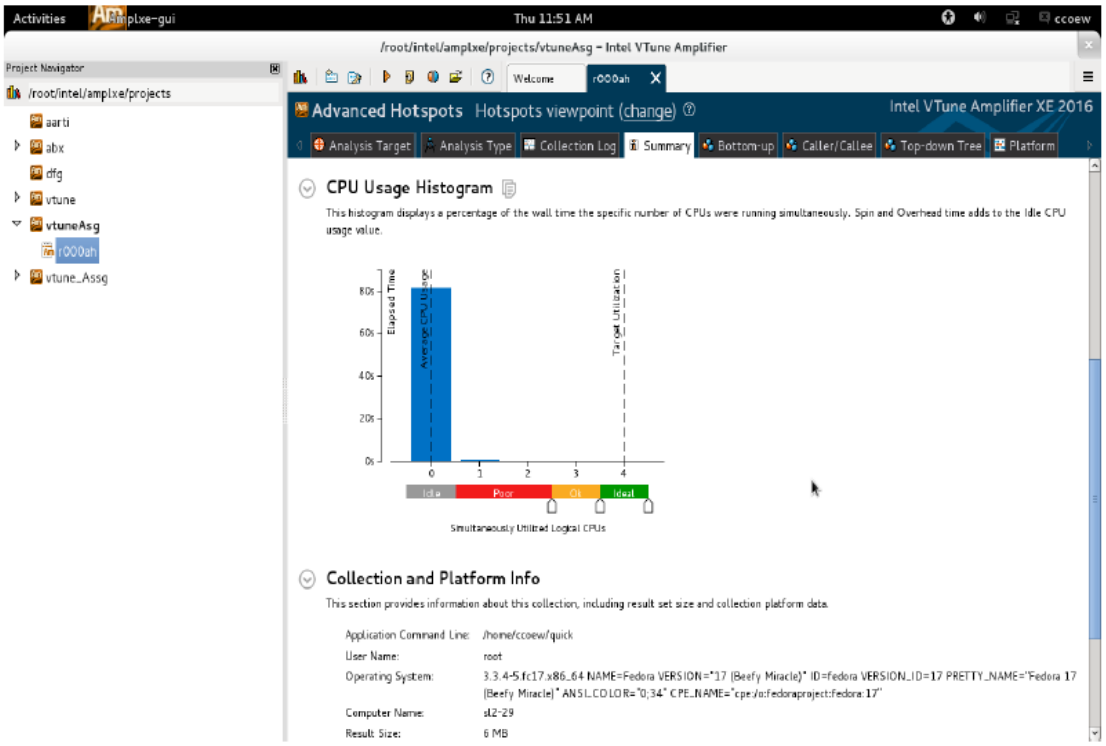
```
#include<iostream>
#include<omp.h>
using namespace std;
int k=0;
class sort
{
        int a[20];
        int n;
public:
        void getdata();
        void Quicksort();
        void Quicksort(int low, int high);
        int partition(int low, int high);
        void putdata();
};
void sort::getdata()
{
        cout<<"Enter the no. of elements in array\t";
        cin>>n;
        cout<<"Enter the elements of array:"<<endl;
        for(int i=0;i<n;i++)
        {
        cin>>a[i];
        }
}
void sort::Quicksort()
{
Quicksort(0,n-1);
}

void sort::Quicksort(int low, int high)
{
if(low<high)
{
        int partn;
        partn=partition(low,high);

cout<<"\n\nThread Number: "<<k<<"  pivot element selected : "<<a[partn];
        #pragma omp parallel sections
        {
        #pragma omp section
                {
                k=k+1;
                Quicksort(low, partn-1);
                }
        #pragma omp section
                {
                k=k+1;
                Quicksort(partn+1, high);
                }
        }//pragma_omp Parallel_end
}

}
int sort::partition(int low ,int high)
{
        int pvt;
        pvt=a[high];
        int i;
        i=low-1;
        int j;
        for(j=low;j<high;j++)
```

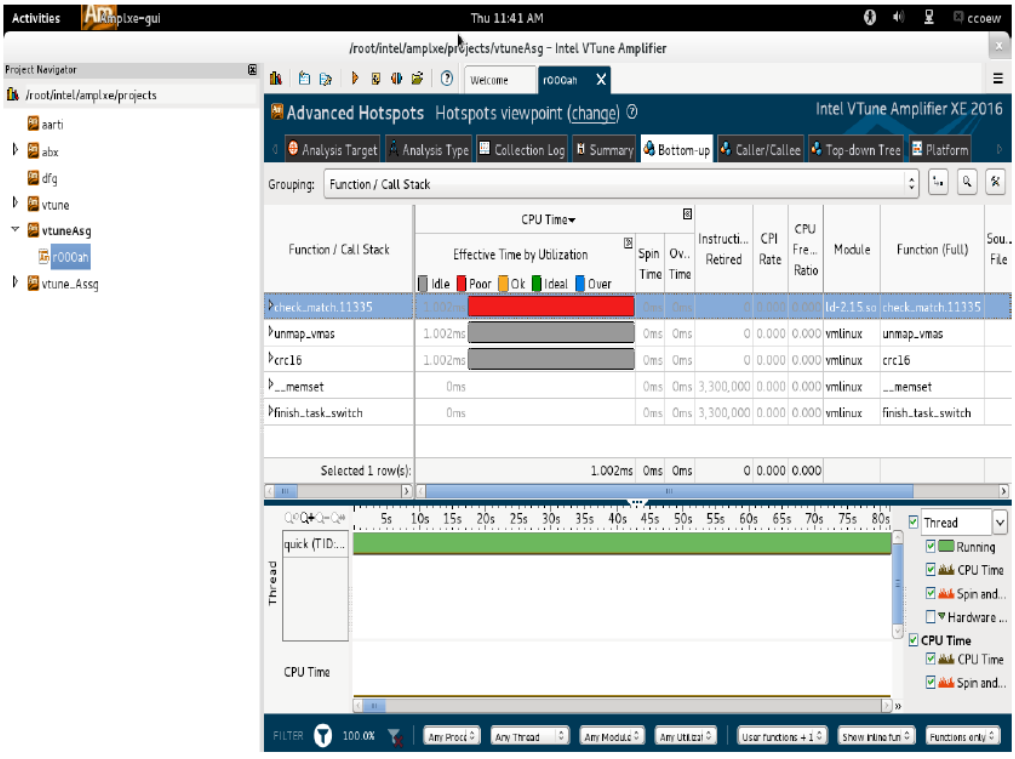
```
        {
                if(a[j]<=pvt)
                {
                        int tem=0;
                        tem=a[j];
                        a[j]=a[i+1];
                        a[i+1]=tem;
                        i=i+1;
                }
        }
        int te;
        te=a[high];
        a[high]=a[i+1];
        a[i+1]=te;
        return i+1;
}
void sort::putdata()
{
        cout<<endl<<"\nThe Array is:"<<endl;
        for(int i=0;i<n;i++)
        cout<<" "<<a[i];
}
int main()
{
        int n;
        sort s1;
        int ch;
do
{
                s1.getdata();
                s1.putdata();

                cout<<"\nUsing Quick Sort";
   double start = omp_get_wtime();
                s1.Quicksort();
   double end = omp_get_wtime();
                cout<<"\nThe Sorted  ";
                s1.putdata();
   cout<<"\nExcecution time : "<<end - start<<" seconds ";

cout<<"Would you like to continue? (1/0 y/n)"<<endl;
cin>>ch;
}while(ch==1);
}
```

```
mpiu@DB21:/mirror/mpiu/CL4 prog/B-3$ g++ quicksort_sample.cpp -fopenmp
mpiu@DB21:/mirror/mpiu/CL4 prog/B-3$ ./a.out
Enter the no. of elements in array  7
Enter the elements of array:
4 5 6 7 8 0 1


The Array is:
4 5 6 7 8 0 1
Using Quick Sort

Thread Number: 0  pivot element selected : 1

Thread Number: 2  pivot element selected : 5

Thread Number: 4  pivot element selected : 7
The Sorted

The Array is:
0 1 4 5 6 7 8
Excecution time : 0.00218175 seconds Would you like to continue? (1/0 y/n)
0
mpiu@DB21:/mirror/mpiu/CL4 prog/B-3$
```




Activities
Thu 11:51 AM
/root/intel/amplxe/projects/vtuneAsg – Intel VTune Amplifier
Project Navigator
/root/intel/amplxe/projects
aarti
abx
dfg
vtune
vtuneAsg
r000ah
vtune_Assg
Advanced Hotspots
Hotspots viewpoint (change)
Intel VTune Amplifier XE 2016
Analysis Target
Analysis Type
Collection Log
Summary
Bottom-up
Caller/Callee
Top-down Tree
Platform
Grouping:
Call Stack
Function Stack
CPU Time: Total
CPU Time: Self
Effective Time by Utilization
Idle
Poor
Ok
Ideal
Over
Spin Time
Ov.. Time
Instr... Retir... Total
Total
check_match.11335
unmap_vmas
crc16
__memset
finish_task_switch
Highlighted 1 row(s):
Thread
quick (TID:...
CPU Time
Running
Spin and...
Hardware ...
FILTER
100.0%
Any Proc
Any Thread
Any Module
Any Utilizat
User functions + 1
Show inline fun


Activities
amplxe-gui
Thu 11:51 AM
ccoew
/root/intel/amplxe/projects/vtuneAsg – Intel VTune Amplifier
Project Navigator
/root/intel/amplxe/projects
aarti
abx
dfg
vtune
vtuneAsg
r000ah
vtune_Assg
Welcome
r000ah
Advanced Hotspots
Hotspots viewpoint (change)
Intel VTune Amplifier XE 2016
Analysis Target
Analysis Type
Collection Log
Summary
Bottom-up
Caller/Callee
Top-down Tree
Platform
CPU Usage Histogram
This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU usage value.
Elapsed Time
Average CPU Usage
Target Utilization
80s
60s
40s
20s
0s
0
1
2
3
4
Idle
Poor
Ok
Ideal
Simultaneously Utilized Logical CPUs
Collection and Platform Info
This section provides information about this collection, including result set size and collection platform data.
Application Command Line: /home/ccoew/quick
User Name: root
Operating System: 3.3.4-5.fc17.x86_64 NAME=Fedora VERSION="17 (Beefy Miracle)" ID=fedora VERSION_ID=17 PRETTY_NAME="Fedora 17 (Beefy Miracle)" ANSI_COLOR="0;34" CPE_NAME="cpe:/o:fedoraproject:fedora:17"
Computer Name: sl2-29
Result Size: 6 MB


Activities
amplxe-gui
Thu 11:41 AM
ccoew
/root/intel/amplxe/projects/vtuneAsg – Intel VTune Amplifier
Project Navigator
/root/intel/amplxe/projects
aarti
abx
dfg
vtune
vtuneAsg
r000ah
vtune_Assg
Welcome
r000ah
Advanced Hotspots
Hotspots viewpoint (change)
Intel VTune Amplifier XE 2016
Analysis Target
Analysis Type
Collection Log
Summary
Bottom-up
Caller/Callee
Top-down Tree
Platform
Grouping: Function / Call Stack
Function / Call Stack
CPU Time
Effective Time by Utilization
Idle Poor Ok Ideal Over
Spin Time
Ov.. Time
Instructi... Retired
CPI Rate
CPU Fre... Ratio
Module
Function (Full)
Sou.. File
check_match.11335
unmap_vmas
crc16
__memset
finish_task_switch
1.002ms
0ms
3,300,000
0.000
vmlinux
ld-2.15.so
Selected 1 row(s):
Thread
Running
CPU Time
Spin and...
Hardware ...
CPU Time
quick (TID:...
FILTER
100.0%
Any Proce
Any Thread
Any Module
Any Utilizat
User functions + 1
Show inline fun
Functions only

Assignment No: B4

Title: Implement a program to check task distribution using Gprof.

Aim: Write a program to check task distribution using Gprof.

Objectives:

- To understand the use of Gprof for task distribution.

Theory:

Gprof

Profiling allows you to learn where your program spent its time and which functions called which other functions while it was executing. This information can show you which pieces of your program are slower than you expected, and might be candidates for rewriting to make your program execute faster. It can also tell you which functions are being called more or less often than you expected. This may help you spot bugs that had otherwise been unnoticed.

Since the profiler uses information collected during the actual execution of your program, it can be used on programs that are too large or too complex to analyze by reading the source. However, how your program is run will affect the information that shows up in the profile data. If you don't use some feature of your program while it is being profiled, no profile information will be generated for that feature.

Profiling has several steps:

- You must compile and link your program with profiling enabled.
- You must execute your program to generate a profile data file.
- You must run gprof to analyze the profile data.

Performance is one of the biggest challenges programmers face while developing software. That is the reason why code profiling is one of the most important aspects of software development, as it lets you identify bottlenecks, dead code, and even bugs. If you are a programmer who develops software applications for Linux, the GNU profiler “gprof” is the tool to look out for."gprof" produces an execution profile of C, Pascal, or Fortran77 programs. The effect of called routines is incorporated in the profile of each caller. The profile data is taken from the call graph profile file (gmon.out default) which is created by programs that are compiled with the -pg option of "cc", "pc", and "f77". The -pg option also links in versions of the library routines that are compiled for profiling. "Gprof" reads the given object file (the default is "a.out") and establishes the relation between its symbol table and the call graph profile from gmon.out. If more than one profile file is specified, the "gprof" output shows the sum of the profile information in the given profile files.

"Gprof" calculates the amount of time spent in each routine. Next, these times are propagated along the edges of the call graph. Cycles are discovered, and calls into a cycle are made to share the time of the cycle. Several forms of output are available from the analysis.

The flat profile shows how much time your program spent in each function, and how many times that function was called. If you simply want to know which functions burn most of the cycles, it is stated

concisely here. The call graph shows, for each function, which functions called it, which other functions it called, and how many times. There is also an estimate of how much time was spent in the subroutines of each function. This can suggest places where you might try to eliminate function calls that use a lot of time. The annotated source listing is a copy of the program's source code, labeled with the number of times each line of the program was executed.

Download and Install

Gprof comes pre-installed with most of the Linux distributions, but if that‟s not the case with your Linux distro, you can download and install it through a command line package manager like apt-get or yum. For example, run the following command to download and install gprof on Debian-based systems:

sudoapt-get install binutils

How to use gprof

Using the gprof tool is not at all complex. You just need to do the following on a high-level:

- Have profiling enabled while compiling the code
- Execute the program code to produce the profiling data
- Run the gprof tool on the profiling data file (generated in the step above).

The last step above produces an analysis file which is in human readable form. This file contains a couple of tables (flat profile and call graph) in addition to some other information. While flat profile gives an overview of the timing information of the functions like time consumption for the execution of a particular function, how many times it was called etc. On the other hand, call graph focuses on each function like the functions through which a particular function was called, what all functions were called from within this particular function etc So this way one can get idea of the execution time spent in the sub-routines too.

Using gprof Task Distribution Performance Analysis:

Consider the following C program as an example:

```
#include <stdio.h>
void func2()
        {
                 int count = 0;
                for(count=0; count < 0XFFFFF; count++);
                return;
        }
 void func1(void)
        {
                 int count = 0;
                for(count=0; count < 0XFF; count++)
                func2();
                return;
        }
 int main(void)
        {
                 printf("\n Hello World! \n");
                func1();
                func2();
```

```
        return 0;
    }
```

Compile the code with the -pg option:

gcc -Wall -pgtest.c -o test

-pg : Generate extra code to write profile information suitable for the analysis program gprof.

You must use this option when compiling the source files you want data about, and you must also use it when linking.

Once compiled, run the program:

./test

After successful execution, the program will produce a file named "gmon.out" that contains the profiling information, but in a raw form, which means that you cannot open the file and directly read the information. To generate a human readable file, run the following command:

gproftestgmon.out>prof_output

This command writes all the profiling information in human readable format to "prof_output" file. Note that you can change the output file name as per your convenience.

Flat profile and Call graph

If you open the file containing profiling data, you'll see that the information is divided into two parts: Flat profile and Call graph. While the former contains details like function call counts, total execution time spent in a function, and more, the latter describes the call tree of the program, providing details about the parent and child functions of a particular function.

For example, the following is the Flat profile in our case:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
100.00	0.94	0.94	256	3.67	3.67	func2
0.00	0.94	0.00	1	0.00	936.33	func1

The below is the Call graph:

Index	%time	self	children	called	name
		0.00	0.00	1/256	main [2]
		0.94	0.00	255/256	func1 [3]
[1]	100.0	0.94	0.00	256	func2 [1]
---					
[2]	100.0	0.00	0.94		main [2]
		0.00	0.94	1/1	func1 [3]
		0.00	0.00	1/256	func2 [1]

		0.00	0.94	1/1	main [2]
[3]	99.6	0.00	0.94	1	func1 [3]
		0.94	0.00	255/256	func2 [1]

Here is the explanation (taken from the file output) of what each column means:

- **Index -** A unique number given to each element of the table. Index numbers are sorted numerically. The index number is printed next to every function name so it is easier to look up where the function is in the table.
- **% time -** This is the percentage of the `total' time that was spent in this function and its children. Note that due to different viewpoints, functions excluded by options, etc., these numbers will NOT add up to 100%.
- **Self -** This is the total amount of time spent in this function. For function's parents, this is the amount of time that was propagated directly from the function into this parent. While, for function's children, this is the amount of time that was propagated directly from the child into the function.
- **Children -** This is the total amount of time propagated into this function by its children. For the function's parents, this is the amount of time that was propagated from the function's children into this parent. While, for the function's children, this is the amount of time that was propagated from the child's children to the function.
- **Called -** This is the number of times the function was called. If the function called itself recursively, the number only includes non-recursive calls, and is followed by a `+' and the number of recursive calls. For the function's parents, this is the number of times this parent called the function `/' the total number of times the function was called. Recursive calls to the function are not included in the number after the `/'. While, for the function's children, this is the number of times the function called this child `/' the total number of times the child was called. Recursive calls by the child are not listed in the number after the `/'.
- **Name -** The name of the current function. The index number is printed after it. If the function is a member of a cycle, the cycle number is printed between the function's name and the index number. For function's parents, this is the name of the parent. The parent's index number is printed after it. If the parent is a member of a cycle, the cycle number is printed between the name and the index number. For function's children, this is the name of the child. The child's index number is printed after it. If the child is a member of a cycle, the cycle number is printed between the name and the index number.

Conclusion: Hence we successfully checked task distribution using Gprof..

CODE

```
#include <stdio.h>
void func2()
{
        int count = 0;
        for(count=0; count < 0XFFFFF; count++);
        return;
}
void func1(void)
{
        int count = 0;
        for(count=0; count < 0XFF; count++)
                func2();
        return;
}
int main(void)
{
        printf("\n Hello World! \n");
        func1();
        func2();
        return 0;
}
```

OUTPUT

```
sagar@sagar-Lenovo-G580:~/new$ gcc test.c -pg
sagar@sagar-Lenovo-G580:~/new$ ./a.out

 Hello World!
sagar@sagar-Lenovo-G580:~/new$ gprof ./a.out gmon.out


Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
100.00      0.52     0.52      256     2.03     2.03  func2
  0.00      0.52     0.00        1     0.00   517.97  func1

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.

 self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
          else blank.

 total     the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
          function is profiled, else blank.

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
```

```
          the function in the gprof listing. If the index is
          in parenthesis it shows where it would appear in
          the gprof listing if it were to be printed.
Copyright (C) 2012 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.

                     Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) for 1.92% of 0.52 seconds

index % time    self  children    called     name
                0.00    0.00       1/256         main [2]
                0.52    0.00     255/256         func1 [3]
[1]    100.0    0.52    0.00     256         func2 [1]
-----------------------------------------------
                                                <spontaneous>
[2]    100.0    0.00    0.52                 main [2]
                0.00    0.52       1/1           func1 [3]
                0.00    0.00       1/256         func2 [1]
-----------------------------------------------
                0.00    0.52       1/1           main [2]
[3]     99.6    0.00    0.52       1         func1 [3]
                0.52    0.00     255/256         func2 [1]
-----------------------------------------------

 This table describes the call tree of the program, and was sorted by
 the total amount of time spent in each function and its children.

 Each entry in this table consists of several lines.  The line with the
 index number at the left hand margin lists the current function.
 The lines above it list the functions that called this function,
 and the lines below it list the functions this one called.
 This line lists:
     index     A unique number given to each element of the table.
               Index numbers are sorted numerically.
               The index number is printed next to every function name so
               it is easier to look up where the function is in the table.

     % time    This is the percentage of the `total' time that was spent
               in this function and its children.  Note that due to
               different viewpoints, functions excluded by options, etc,
               these numbers will NOT add up to 100%.

     self      This is the total amount of time spent in this function.

     children  This is the total amount of time propagated into this
               function by its children.

     called    This is the number of times the function was called.
               If the function called itself recursively, the number
               only includes non-recursive calls, and is followed by
               a `+' and the number of recursive calls.

     name      The name of the current function.  The index number is
               printed after it.  If the function is a member of a
               cycle, the cycle number is printed between the
               function's name and the index number.


 For the function's parents, the fields have the following meanings:

     self      This is the amount of time that was propagated directly
```

```
               from the function into this parent.

     children  This is the amount of time that was propagated from
               the function's children into this parent.

     called    This is the number of times this parent called the
               function `/' the total number of times the function
               was called.  Recursive calls to the function are not
               included in the number after the `/'.

     name      This is the name of the parent.  The parent's index
               number is printed after it.  If the parent is a
               member of a cycle, the cycle number is printed between
               the name and the index number.

 If the parents of the function cannot be determined, the word
 `<spontaneous>' is printed in the `name' field, and all the other
 fields are blank.

 For the function's children, the fields have the following meanings:

     self      This is the amount of time that was propagated directly
               from the child into the function.

     children  This is the amount of time that was propagated from the
               child's children to the function.

     called    This is the number of times the function called
               this child `/' the total number of times the child
               was called.  Recursive calls by the child are not
               listed in the number after the `/'.

     name      This is the name of the child.  The child's index
               number is printed after it.  If the child is a
               member of a cycle, the cycle number is printed
               between the name and the index number.

 If there are any cycles (circles) in the call graph, there is an
 entry for the cycle-as-a-whole.  This entry shows who called the
 cycle (as parents) and the members of the cycle (as children.)
 The `+' recursive calls entry shows the number of function calls that
 were internal to the cycle, and the calls entry for each member shows,
 for that member, how many times it was called from other members of
 the cycle.

Copyright (C) 2012 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.

Index by function name

   [3] func1                   [1] func2
```

Assignment No. B7

Title:
Implement concurrent ODD-Even Merge sort algorithm.

Aim:
Perform concurrent ODD-Even Merge sort using HPC infrastructure (preferably BBB) using Python/ Scala/ Java/ C++.

Objectives:

- To understand working of Cluster of BBB (BeagleBone Black)
- Implement concurrent ODD-Even Merge sort algorithm.

Prerequisites:
- 64-bit Ubuntu or equivalent OS with 64-bit Intel-i5/i7
- Cluster of BBB, that is two or more BBB

Theory:

Building a Compute Cluster with the BeagleBone Black

Configuring the BeagleBones

Once the hardware is set up and a machine is connected to the network, Putty or any other SSH client can be used to connect to the machines. The default hostname to connect to using the above image is ubuntu-armhf. My first task was to change the hostname. I chose to name mine beaglebone1, beaglebone2 and beaglebone3. First I used the hostname command:

sudo hostname beaglebone1

Next I edited /etc/hostname and placed the new hostname in the file. The next step was to hard code the IP address for so I could probably map it in the hosts file. I did this by editing /etc/network/interfaces to tell it to use static IPs. In my case I have a local network with a router at 192.168.1.1. I decided to start the IP addresses at 192.168.1.51 so the file on the first node looked like this:

```
iface eth0 inet static
address 192.168.1.51
netmask 255.255.255.0
network 192.168.1.0
broadcast 192.168.1.255
gateway 192.168.1.1
```

It is usually a good idea to pick something outside the range of IPs that your router might assign if you are going to have a lot of devices. Usually you can configure this range on your router. With this done, the final step to perform was to edit /etc/hosts and list the name and IP address of each node that would be in the cluster. My file ended up looking like this on each of them:

127.0.0.1 localhost
192.168.1.51 beaglebone1
192.168.1.52 beaglebone2
192.168.1.53 beaglebone3

Creating a Compute Cluster With MPI
After setting up all 3 BeagleBones, I was ready to tackle my first compute project. I figured a good starting point for this was to set up MPI. MPI is a standardized system for passing messages between machines on a network. It is powerful in that it distributes programs across nodes so each instance has access to the local memory of its machine and is supported by several languages such as C, Python and Java. There are many versions of MPI available so I chose MPICH which I was already familiar with. Installation was simple, consisting of the following three steps:

sudo apt-get update
sudo apt-get install gcc
sudo apt-get install libcr-dev mpich2 mpich2-doc

MPI works by using SSH to communicate between nodes and using a shared folder to share data. The first step to allowing this was to install NFS. I picked beaglebone1 to act as the master node in the MPI cluster and installed NFS server on it:

sudo apt-get install nfs-client

With this done, I installed the client version on the other two nodes:

sudo apt-get install nfs-server

Next I created a user and folder on each node that would be used by MPI. I decided to call mine hpcuser and started with its folder:

sudomkdir /hpcuser

Once it was created on all the nodes, I synced up the folders by issuing this on the master node:

echo "/hpcuser *(rw,sync)" | sudo tee -a /etc/exports

Then I mounted the master's node on each slave so they can see any files that are added to the master node:

sudo mount beaglebone1:/hpcuser /hpcuser

To make sure this is mounted on reboots I edited /etc/fstab and added the following:

beaglebone1:/hpcuser /hpcusernfs

Finally I created the hpcuser and assigned it the shared folder:

sudouseradd -d /hpcuserhpcuser

With network sharing set up across the machines, I installed SSH on all of them so that MPI could communicate with each:

sudo apt-get install openssh-server

The next step was to generate a key to use for the SSH communication. First I switched to the hpcuser and then used ssh-keygen to create the key.

su - hpcuser
sshkeygen-t rsa

When performing this step, for simplicity you can keep the passphrase blank. When asked for a location, you can keep the default. If you want to use a passphrase, you will need to take extra steps to prevent SSH from prompting you to enter the phrase. You can use ssh-agent to store the key and prevent this. Once the key is generated, you simply store it in our authorized keys collection:

cd .ssh
cat id_rsa.pub >>authorized_keys

I then verified that the connections worked using ssh:

ssh hpcuser@beaglebone2

Testing MPI
Once the machines were able to successfully connect to each other, I wrote a simple program on the master node to try out. While logged in as hpcuser, I created a simple program in its root directory /hpcuser called mpi1.c. MPI needs the program to exist in the shared folder so it can run on each machine. The program below simply displays the index number of the current process, the total number of processes running and the name of the host of the current process. Finally, the main node receives a sum of all the process indexes from the other nodes and displays it:

```
#include <mpi.h>
#include <stdio.h>
int main(intargc, char* argv[])
{
int rank, size, total;
char hostname[1024];
gethostname(hostname, 1023);
MPI_Init(&argc, &argv);
MPI_Comm_rank (MPI_COMM_WORLD, &rank);
MPI_Comm_size (MPI_COMM_WORLD, &size);
MPI_Reduce(&rank, &total, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
printf("Testing MPI index %d of %d on hostname %s\n", rank, size,
hostname);
if (rank==0)
{
printf("Process sum is %d\n", total);
}
MPI_Finalize();
return 0;
}
```

Next I created a file called machines.txt in the same directory and placed the names of the nodes in the cluster inside, one per line. This file tells MPI where it should run:

```
beaglebone1
beaglebone2
beaglebone3
```

With both files created, I finally compiled the program using mpicc and ran the test:

mpicc mpi1.c -o mpiprogram
mpiexec -n 8 -f machines.txt. /mpiprogram

This resulted in the following output demonstrating it ran on all 3 nodes:

```
Testing MPI index 4 of 8 on hostname beaglebone2
Testing MPI index 7 of 8 on hostname beaglebone2
Testing MPI index 5 of 8 on hostname beaglebone3
Testing MPI index 6 of 8 on hostname beaglebone1
Testing MPI index 1 of 8 on hostname beaglebone2
Testing MPI index 3 of 8 on hostname beaglebone1
Testing MPI index 2 of 8 on hostname beaglebone3
Testing MPI index 0 of 8 on hostname beaglebone1
Process sum is 28
```

ODD-Even Merge sort algorithm:

In computing, an odd–even sort or odd–even transposition sort is a relatively simple sorting algorithm, developed originally for use on parallel processors with local interconnections. It is a comparison sort related to bubble sort, with which it shares many characteristics. It functions by comparing all odd/even indexed pairs of adjacent elements in the list and, if a pair is in the wrong order (the first is larger than the second) the elements are switched. The next step repeats this for even/odd indexed pairs (of adjacent elements). Then it alternates between odd/even and even/odd steps until the list is sorted.

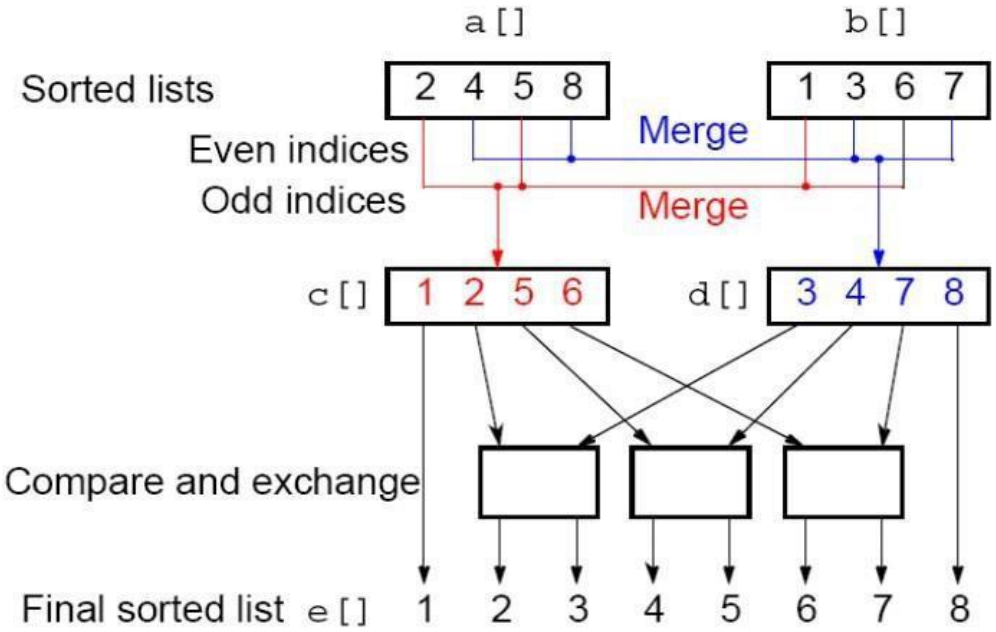
Sorting on processor arrays

On parallel processors, with one value per processor and only local left–right neighbor connections, the processors all concurrently do a compare–exchange operation with their neighbors, alternating between odd–even and even–odd pairings. This algorithm was originally presented, and shown to be efficient on such processors, by Habermann in 1972.

The algorithm extends efficiently to the case of multiple items per processor. In the Baudet–Stevenson odd–even merge-splitting algorithm, each processor sorts its own sublist at each step, using any efficient sort algorithm, and then performs a merge splitting, or transposition–merge, operation with its neighbor, with neighbor pairing alternating between odd–even and even–odd on each step.

1. It starts by distributing n/p sub-lists (p is the number of processors and n the size of the array to sort) to all the processors.
2. Each processor then sequentially sorts its sub-list.
3. The algorithm then operates by alternating between an odd and an even phase :
    1. In the even phase, even numbered processors (processor i) communicate with the next odd numbered processors (processor i+1). In this communication process, the two sub-lists for each 2 communicating processes are merged together. The upper half of the list is then kept in the higher number processor and the lower half is put in the lower number processor.

2. In the odd phase, odd number processors (processor i) communicate with the previous even number processors (i-1) in exactly the same way as in the even phase.



Implementation:
The first step is to distribute a sub list to each process, the master process send to all process a sub-array, to do that we use the MPI_Scatter() function :

```
int *subArray = malloc(N/hostCount * sizeof(int));
if(rank == 0) { /* The master send data to everyone */
MPI_Scatter(arrayToSort,N/hostCount,MPI_INT,subArray,N/hostCount,MPI_INT,0
,
      MPI_COMM_WORLD);
}
```

After that each process has to receive data, we use MPI_Scatterv(), we need 2 particular array : displs that specifies the displacement of sub-array relative to arrayToSort and sendcnts that specifies the number of elements to send to each host.

```
int *displs = malloc(hostCount * sizeof(int));
int i;
for (i=0;i<hostCount;i++) {
      displs[i] = i*(N/hostCount);
}
int *sendcnts = malloc(hostCount * sizeof(int));
for (i=0;i<hostCount;i++) {
      sendcnts[i]=N/hostCount;
}
/* reieve data */
MPI_Scatterv(arrayToSort,sendcnts,displs,MPI_INT,subArray,N/hostCount,MPI_
INT,0,MPI_COMM_WORLD);
free(displs);
free(sendcnts);
```

The next step is to write a sequential sort algorithm. I choose the odd-even sort algorithm. Here is my C function:

```
void sequentialSort(int *arrayToSort, int size) {
int sorted = 0;
while( sorted == 0) {
      sorted= 1;
      int i;
      for(i=1;i<size-1; i += 2) {
            if(arrayToSort[i] >arrayToSort[i+1])
            {
                  int temp = arrayToSort[i+1];
                  arrayToSort[i+1] = arrayToSort[i];
                  arrayToSort[i] = temp;
                  sorted = 0;
            }
      }
      for(i=0;i<size-1;i+=2) {
            if(arrayToSort[i] >arrayToSort[i+1])
            {
                  int temp = arrayToSort[i+1];
                  arrayToSort[i+1] = arrayToSort[i];
                  arrayToSort[i] = temp;
                  sorted = 0;
            }
      }
      }
}
```

The third step is the odd-even phases. For this step we need 2 functions. One function to send data to the next host and keep the lower part of the to arrays. And another one to send data to the previous host and keep the higher part.

```
/* Parameter :
* subArray : an integer array
* size : the size of the integer
* rank : the rank of the host
* Send to the next host an array, recieve array from the next host
* keep the lower part of the 2 array
*/
void exchangeWithNext(int *subArray, int size, int rank)
{
      MPI_Send(subArray,size,MPI_INT,rank+1,0,MPI_COMM_WORLD);
      /* recieve data from the next odd numbered host */
      int *nextArray = malloc(size*sizeof(int));
      MPI_Status stat;
      MPI_Recv(nextArray,size,MPI_INT,rank+1,0,MPI_COMM_WORLD,&stat);
      /* Keep the lower half of subArray and nextArray */
      lower(subArray,nextArray,size);
      free(nextArray);
}
/* Parameter :
* subArray : an integer array
* size : the size of the integer
```

```
* rank : the rank of the host
* Send to the previous host an array, recieve array from the previous host
* keep the higher part of the 2 array
*/
void exchangeWithPrevious(int *subArray, int size, int rank)
{
	/* send our sub-array to the previous host*/
	MPI_Send(subArray,size,MPI_INT,rank-1,0,MPI_COMM_WORLD);
	/* recieve data from the previous host */
	int *previousArray = malloc(size*sizeof(int));
	MPI_Status stat;
	MPI_Recv(previousArray,size,MPI_INT,rank-1,0,MPI_COMM_WORLD,&stat);
	/* Keep the higher half of subArray and previousArray */
	higher(subArray,previousArray,size);
	free(previousArray);
}
```

Then we can write easily the odd-even phase :

```
i =0;
for(i=0;i<hostCount;i++) {
	/* even phase */
	if (i%2==0) {
	/* even numbered host */
		if(rank%2==0) {
		/* even numbered host communicate with the next odd numbered host */
		/* make sure the next odd exits */
			if(rank<hostCount-1) {
			/* send our sub-array to the next odd numbered host
			* receive data from the next odd numbered host
			* Keep the lower half of our array and of the next host
array's
			*/
			exchangeWithNext(subArray,N/hostCount,rank);
			}
	} else {
		/* odd numbered host communicate with the previous even numbered
host */
		/* make sure the previous even exits */
			if (rank-1 >=0 ) {
				/* send our sub-array to the previous even numbered
host
				* receive data from the previous even numbered host
				* Keep the higher half of our array and of the previous
host array's
				*/
				exchangeWithPrevious(subArray,N/hostCount,rank);
			}
		}
	}
	/* odd phase */
	else {
		/* odd host */
		if(rank%2!=0) {
			/* In the odd phase odd numbered host communicate with the
next
```

```
                    * even numbered host make sure the next even exits */
                    if (rank<hostCount-1) {
                    /* send our sub-array to the next even numbered host
                    * receive data from the next even numbered host
                    * Keep the lower half of our array and the next host array's
                    */
                    exchangeWithNext(subArray,N/hostCount,rank);
                    }
             }
             /* even host */
             else {
                    /* In the odd phase even numbered host communicate with the
previous
                    * odd numbered host make sure the previous host exits */
                    if (rank-1 >=0 ) {
                    /* send our sub-array to the previous odd numbered host
                    * receive data from the previous odd numbered host
                    * Keep the higher half of our array and of the previous host
array's
                    */
                    exchangeWithPrevious(subArray,N/hostCount,rank);
                    }
             }
      }
}
```

Now our array is sorted, the lower element own to the host with the rank 0 and the higher to p-1. We just need to gather data, we use MPI_Gather()

```
MPI_Gather(subArray,
N/hostCount,MPI_INT,arrayToSort,N/hostCount,MPI_INT,0,MPI_COMM_WORLD);
```

Performances

The question is: what is the execution time of the parallel algorithm compare to the sequential one. Well it's pretty good, on average in a cluster of 8 machines and for an array of 128 000 elements the parallel algorithm run in 1.50 s, while the sequential one run in 81.71 s on a single machine. We are in a case of super-linearity because sort a sub-array 8 times smaller than the original array run 64 times faster.

Mathematical Model:

Odd-Even Merge sort is used for sorting of given data.

Following parameters are used for Odd-Even Merge sort:

M= {s, e, i, o, n, F, Success, Failure}

s = Start state = inserting numbers to array to be sorted.

e = End state = Sorted list displayed.

i is the set of input element.

is the set of required output.

n = Number of processors = {host names of processors in a file}

F is the set of functions required for Odd-Even Merge sort. = {f1, f2}

f1 = {Odd-Even Cycle}

f2 = {Even-Odd Cycle}

i= {a1, a2, a3,…….., an}. = Array of elements to be sorted.

o={Sorted list of elements by Odd-Even Merge sort}
Success – Sorted list of elements by Odd-Even Merge sort.
Failure-ϕ

Conclusion:

Thus we have implemented concurrent ODD-Even Merge sort algorithm.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int merge(double *ina, int lena, double *inb, int lenb, double *out) {
   int i,j;
   int outcount=0;

   for (i=0,j=0; i<lena; i++) {
       while ((inb[j] < ina[i]) && j < lenb) {
           out[outcount++] = inb[j++];
       }
       out[outcount++] = ina[i];
   }
   while (j<lenb)
       out[outcount++] = inb[j++];

   return 0;
}

int domerge_sort(double *a, int start, int end, double *b) {
   if ((end - start) <= 1) return 0;

   int mid = (end+start)/2;
   domerge_sort(a, start, mid, b);
   domerge_sort(a, mid,   end, b);
   merge(&(a[start]), mid-start, &(a[mid]), end-mid, &(b[start]));
int i;
   for (i=start; i<end; i++)
       a[i] = b[i];

   return 0;
}

int merge_sort(int n, double *a) {
   double b[n];
   domerge_sort(a, 0, n, b);
```

```
   return 0;
}

void printstat(int rank, int iter, char *txt, double *la, int n) {
   printf("[%d] %s iter %d: <", rank, txt, iter);
int i,j;
   for (j=0; j<n-1; j++)
       printf("%6.3lf,",la[j]);
   printf("%6.3lf>\n", la[n-1]);
}

void MPI_Pairwise_Exchange(int localn, double *locala, int sendrank, int recvrank,
                           MPI_Comm comm) {

   /*
    * the sending rank just sends the data and waits for the results;
    * the receiving rank receives it, sorts the combined data, and returns
    * the correct half of the data.
    */
   int rank;
   double remote[localn];
   double all[2*localn];
   const int mergetag = 1;
   const int sortedtag = 2;

   MPI_Comm_rank(comm, &rank);
   if (rank == sendrank) {
       MPI_Send(locala, localn, MPI_DOUBLE, recvrank, mergetag, MPI_COMM_WORLD);
       MPI_Recv(locala, localn, MPI_DOUBLE, recvrank, sortedtag, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
   } else {
       MPI_Recv(remote, localn, MPI_DOUBLE, sendrank, mergetag, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
       merge(locala, localn, remote, localn, all);

       int theirstart = 0, mystart = localn;
       if (sendrank > rank) {
           theirstart = localn;
           mystart = 0;
       }
       MPI_Send(&(all[theirstart]), localn, MPI_DOUBLE, sendrank, sortedtag,
MPI_COMM_WORLD);
int i;
       for (i=mystart; i<mystart+localn; i++)
           locala[i-mystart] = all[i];
   }
}

int MPI_OddEven_Sort(int n, double *a, int root, MPI_Comm comm)
{
   int rank, size, i;
   double *local_a;

// get rank and size of comm
   MPI_Comm_rank(comm, &rank); //&rank = address of rank
   MPI_Comm_size(comm, &size);

   local_a = (double *) calloc(n / size, sizeof(double));


// scatter the array a to local_a
   MPI_Scatter(a, n / size, MPI_DOUBLE, local_a, n / size, MPI_DOUBLE,
       root, comm);
```

```
// sort local_a
   merge_sort(n / size, local_a);

//odd-even part
   for (i = 1; i <= size; i++) {

       printstat(rank, i, "before", local_a, n/size);

       if ((i + rank) % 2 == 0) {  // means i and rank have same nature
           if (rank < size - 1) {
               MPI_Pairwise_Exchange(n / size, local_a, rank, rank + 1, comm);
           }
       } else if (rank > 0) {
           MPI_Pairwise_Exchange(n / size, local_a, rank - 1, rank, comm);
       }

   }

   printstat(rank, i-1, "after", local_a, n/size);

// gather local_a to a
   MPI_Gather(local_a, n / size, MPI_DOUBLE, a, n / size, MPI_DOUBLE,
          root, comm);

   if (rank == root)
       printstat(rank, i, " all done ", a, n);

   return MPI_SUCCESS;
}

int main(int argc, char **argv) {

   MPI_Init(&argc, &argv);

   int n = argc-1;
   double a[n];
int i;
   for (i=0; i<n; i++)
       a[i] = atof(argv[i+1]);

   MPI_OddEven_Sort(n, a, 0, MPI_COMM_WORLD);

MPI_Finalize();

   return 0;
}
```

```
mpiu@DB21:/mirror/mpiu/CL4 prog/B-7$ mpicc odd_even.c
mpiu@DB21:/mirror/mpiu/CL4 prog/B-7$ mpiexec -n 4 -f machinefile ./a.out
[0] before iter 1: < 0.000>
[0] before iter 2: < 0.000>
[0] before iter 3: < 0.000>
[0] before iter 4: < 0.000>
[0] after iter 4: < 0.000>
[1] before iter 1: < 0.000>
[1] before iter 2: < 0.000>
[1] before iter 3: < 0.000>
[1] before iter 4: < 0.000>
[1] after iter 4: < 0.000>
[2] before iter 1: < 0.000>
```

```
[2] before iter 2: < 0.000>
[2] before iter 3: < 0.000>
[2] before iter 4: < 0.000>
[2] after iter 4: < 0.000>
[3] before iter 1: < 0.000>
[3] before iter 2: < 0.000>
[3] before iter 3: < 0.000>
[3] before iter 4: < 0.000>
[3] after iter 4: < 0.000>
[0]  all done  iter 5: < 0.000>
mpiu@DB21:/mirror/mpiu/CL4 prog/B-7$
```

Assignment No. B9

Title : Build a required Database to develop BAI tool for considering one aspect of growth to the business such as an organization of products based on expiry dates using Hadoop Ecosystem for unstructured data analytics.

Aim: A Mall has number of items for sale. Build a required Database to develop BAI tool for considering one aspect of growth to the business such as an organization of products based on expiry dates using Hadoop Ecosystem for unstructured data analytics.

Objectives:

- To study the Business intelligence.
- To explore Hadoop and Mapreduce concepts.

Theory :

Hadoop:

Hadoop is an open-source framework that allows to store and process big data in a distributed environment across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage.
Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models. A Hadoop frame-worked application works in an environment that provides distributed storage and computation across clusters of computers. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.

Hadoop Architecture

Hadoop framework includes following four modules:

1. Hadoop Common: These are Java libraries and utilities required by other Hadoop modules. These libraries provides file system and OS level abstractions and contains the necessary Java _les and scripts required to start Hadoop.
2. Hadoop YARN: This is a framework for job scheduling and cluster resource management.
3. Hadoop Distributed File System (HDFS): A distributed _le system that provides high-throughput access to application data.
4. Hadoop MapReduce: This is YARN-based system for parallel processing of large data sets.

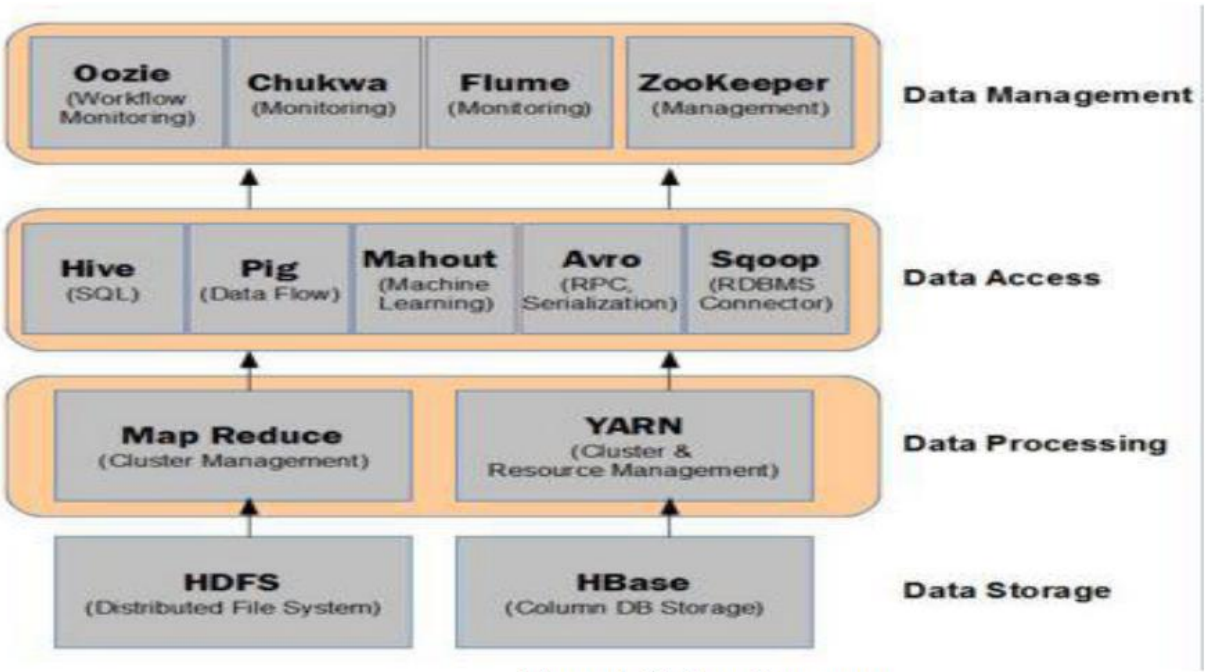


Figure 1: Hadoop Ecosystem

HDFS

Hadoop File System was developed using distributed file system design. It is run on commodity hardware. Unlike other distributed systems, HDFS is highly fault tolerant and designed using low-cost hardware.

Features of HDFS:-

1. It is suitable for the distributed storage and processing.
2. Hadoop provides a command interface to interact with HDFS.
3. The built-in servers of namenode and datanode help users to easily check thestatus of cluster.
4. Streaming access to _le system data.
5. HDFS provides _le permissions and authentication.

HDFS Architecture

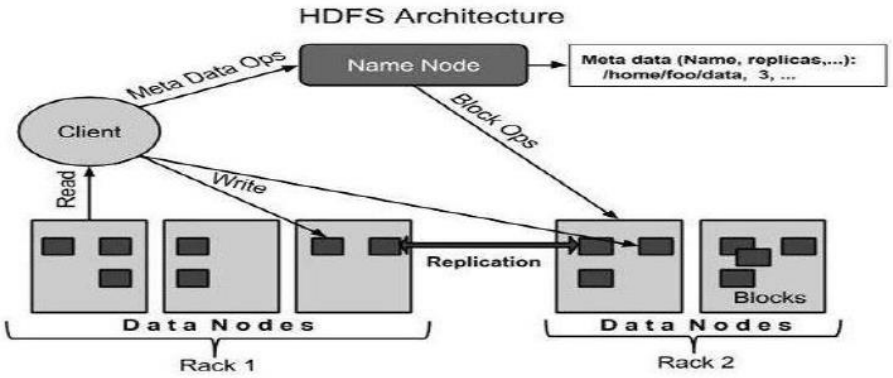


Figure 2: Hadoop Architecture

HDFS follows the master-slave architecture and it has the following elements.

Namenode :

The namenode is the commodity hardware that contains the GNU/Linux operating system and the namenode software. It is software that can be run on commodity hardware.
The system having the namenode acts as the master server and it does the following tasks:

1. Manages the file system namespace.
2. Regulates client's access to files.
3. It also executes file system operations such as renaming, closing, and opening files and directories.

Datanode :

The datanode is a commodity hardware having the GNU/Linux operating system and datanode software. For every node (Commodity hardware/System) in a cluster, there will be a datanode. These nodes manage the data storage of their system.

Datanodes perform read-write operations on the file systems, as per client request. They also perform operations such as block creation, deletion, and replication according to the instructions of the namenode.

Block :

Generally the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These file segments are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block. The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration.

Goals of HDFS :

1. 1.Fault detection and recovery : Since HDFS includes a large number of commodity hardware, failure of components is frequent. Therefore HDFS should have mechanisms for quick and automatic fault detection and recovery.
2. 2.Huge datasets : HDFS should have hundreds of nodes per cluster to manage the applications having huge datasets.
3. 3.Hardware at data : A requested task can be done efficiently, when the computation takes place near the data. Especially where huge datasets are involved, it reduces the network traffic and increases the throughput.

MapReduce :

MapReduce is a framework using which we can write applications to process huge amounts of data, in parallel, on large clusters of commodity hardware in a reliable manner. MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

Mathematical Model

Let S be the solution
S={s, e, i, o, f, DD, NDD, success, failure}
s = initial state that is constructor of the class.
e = be the end state or destructor of the class.
i = input of the system.
o = output of the system.
DD-deterministic data it helps identifying the load store functions or assignment functions.
NDD- Non deterministic data of the system S to be solved. Success-desired outcome generated.
Failure-Desired outcome not generated or forced exit due to system error.
Success= {Desired outcome generated}
Failure= {Desired outcome not generated or forced exit due to system error}

Conclusion:
We have successfully implemented Hadoop and MapReduce.

Code:

PriceTypeDriver.java

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import com.pricetype.mapper.PriceTypeMapper;
import com.pricetype.reducer.PriceTypeReducer;

public class PriceTypeDriver
{
	public static void main (String[] args) throws IOException,
	ClassNotFoundException, InterruptedException
	{
		Configuration conf = new Configuration();
		Job j=new Job(conf);
		j.setJarByClass(PriceTypeDriver.class);
		j.setMapperClass(PriceTypeMapper.class);
		j.setReducerClass(PriceTypeReducer.class);
		j.setMapOutputKeyClass(Text.class);

		j.setMapOutputValueClass(LongWritable.class);
		j.setOutputKeyClass(Text.class);
		j.setOutputValueClass(LongWritable.class);
		j.setInputFormatClass(TextInputFormat.class);
		j.setOutputFormatClass(TextOutputFormat.class);
		Path input = new Path(args[0]);
		Path output = new Path(args[1]);
		FileInputFormat.addInputPath(j, input);
		FileOutputFormat.setOutputPath(j, output);
		boolean isJobRunning = j.waitForCompletion(true);
		System.exit(isJobRunning ? 0 : 1);
	}
}
```

PriceTypeMapper.java

```
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class PriceTypeMapper extends Mapper<LongWritable,Text,Text,LongWritable>
{
	@Override
	protected void map(LongWritable key, Text value, Context context)throws
	IOException, InterruptedException
	{
		LongWritable one = new LongWritable(1);
		String line = value.toString();
		String [] words = line.split("\\|");
		if(words.length>=3)
		{
			if(words[1].equals("R"))
			{
```

```
                                context.write(value,one);
                        }
                }
        }
}
```

PriceTypeReducer.java

```
import java.io.IOException;
import java.util.Date;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class PriceTypeReducer extends Reducer<Text,LongWritable,Text,Text>
{
        @SuppressWarnings("deprecation")
        @Override
        protected void reduce(Text key, Iterable<LongWritable> values,Context context)
        throws IOException, InterruptedException
        {
                String line = key.toString();
                String [] words = line.split("\\|");
                Date startDate = new Date(words[3]);
                long differ = (System.currentTimeMillis() - startDate.getTime());
                long diff_h=differ/(60*60*1000);
                long days=diff_h/24;
                if(days > 365)
                {
                        context.write(new Text(line), new Text(" "));
                }
        }
}
```

Commands

To create input file:
1. vim in.txt
2. To add records: type i (i-insert mode)
3. To save and exit from vim, press escape key- : - wq and hit enter
4. hadoop fs - copyFromLocal MyProject.jar
5. hadoop jar MyProject.jar myproject.com.driver.PriceTypeDriver in.txt output125
6. Output is generated in directory output125
7. hadoop fs -ls output125
8. hadoop fs -cat output125/part-r-00000

Input

```
1|R|10|01/01/2015

2|R|20|01/02/2016

3|R|30|01/02/2000

4|R|40|01/03/2004
```

Output

```
[cloudera@quickstart ~]$ vim in.txt

[cloudera@quickstart ~]$ hadoop jar MyProject.jar
myproject.com.driver.PriceTypeDriver
in.txt output125
16/03/16 01:04:03
```

```
INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
16/03/16 01:04:04
WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not
performed.
Implement the Tool interface and execute your application with ToolRunner to
remedy this.
16/03/16 01:04:05
INFO input.FileInputFormat: Total input paths to process : 1
16/03/16 01:04:05
INFO mapreduce.JobSubmitter: number of splits:1
16/03/16 01:04:05
INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1457534540297_0028
16/03/16 01:04:06

INFO impl.YarnClientImpl: Submitted application application_1457534540297_0028
16/03/16 01:04:06
INFO mapreduce.Job: The url to track the job:
http://quickstart.cloudera:8088/proxy/
application_1457534540297_0028
16/03/16 01:04:06
INFO mapreduce.Job: Running job: job_1457534540297_0028
16/03/16 01:04:17
INFO mapreduce.Job: Job job_1457534540297_0028 running in uber mode : false
16/03/16 01:04:17
INFO mapreduce.Job: map 0% reduce 0%
16/03/16 01:04:35
INFO mapreduce.Job: map 100% reduce 0%
16/03/16 01:04:55
INFO mapreduce.Job: map 100% reduce 100%
16/03/16 01:04:56
INFO mapreduce.Job: Job job_1457534540297_0028 completed successfully
16/03/16 01:04:56
INFO mapreduce.Job: Counters: 49
File System Counters
FILE: Number of bytes read=118
FILE: Number of bytes written=224531
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=189
HDFS: Number of bytes written=60
HDFS: Number of read operations=6
HDFS: Number of large read operations=0
HDFS: Number of write operations=2
Job Counters
Launched map tasks=1
Launched reduce tasks=1

Data-local map tasks=1
Total time spent by all maps in occupied slots (ms)=16314
Total time spent by all reduces in occupied slots (ms)=16099
Total time spent by all map tasks (ms)=16314
Total time spent by all reduce tasks (ms)=16099
Total vcore-seconds taken by all map tasks=16314
Total vcore-seconds taken by all reduce tasks=16099
Total megabyte-seconds taken by all map tasks=16705536
Total megabyte-seconds taken by all reduce tasks=16485376
Map-Reduce Framework
Map input records=4
```

```
Map output records=4
Map output bytes=104
Map output materialized bytes=118
Input split bytes=117
Combine input records=0
Combine output records=0
Reduce input groups=4
Reduce shuffle bytes=118
Reduce input records=4
Reduce output records=3
Spilled Records=8
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=185
CPU time spent (ms)=3600
Physical memory (bytes) snapshot=335437824
Virtual memory (bytes) snapshot=3008466944
Total committed heap usage (bytes)=226562048
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=72
File Output Format Counters
Bytes Written=60
[cloudera@quickstart ~]$ hadoop fs -ls output125
Found 2 items
-rw-r--r-- 1
cloudera cloudera 0 2016-03-16 01:04 output125/_SUCCESS
-rw-r--r-- 1 cloudera cloudera 60 2016-03-16 01:04

output125/part-r-00000
[cloudera@quickstart ~]$ hadoop fs -cat output125/part-r-00000
1|R|10|01/01/2015
3|R|30|01/02/2000
4|R|40|01/03/2004
```

Assignment No.: 12

Title : Frame the suitable assignment to perform computing using BIA tools effectively

Aim: Frame the suitable assignment to perform computing using BIA tools effectively

Objectives:

- To study the Business intelligence.
- To perform the computation using BIA tool like Kettle

Theory :

Kettle:-

The Kettle repository is a workspace that the data integrator works on. This workspace is a physical region of the hard-drive that is designated exclusively for Kettle. In the repository, all information about transformations, jobs, schedules, etc. is stored. The repository concept promotes re-usability, which in turn saves time and effort. We can create a new repository in two ways:
1. Kettle database repository: This repository uses a central relational database to store the ETL (Extraction Transformation Load) metadata.
2. Kettle _le repository: This is a repository stored in a _le in a certain folder.Creating a repository through Kettle database repository:-

Step 1: In the Repository Connection dialog box click on the + button. The Select the repository type dialog box will appear.

Step 2: Select Kettle database repository and click on OK.

Step 3: Now in the Repository information dialog box click on the New button.

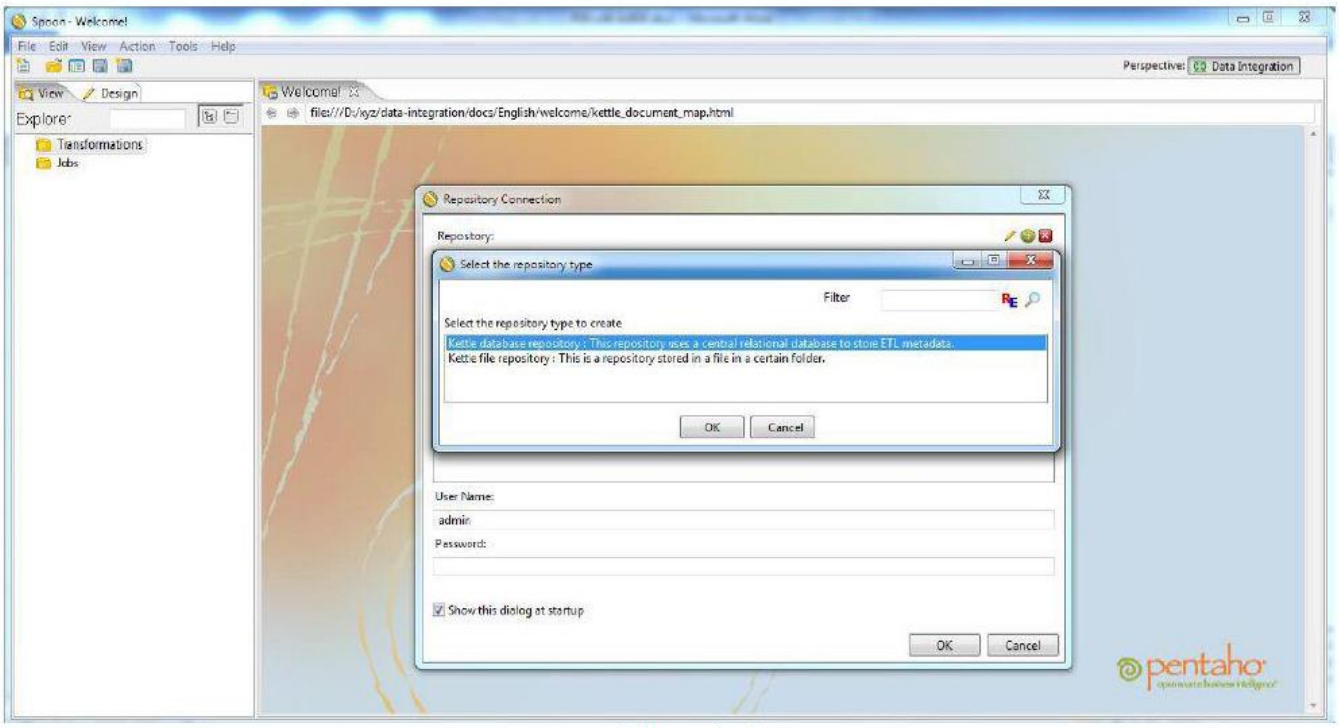


Figure:1

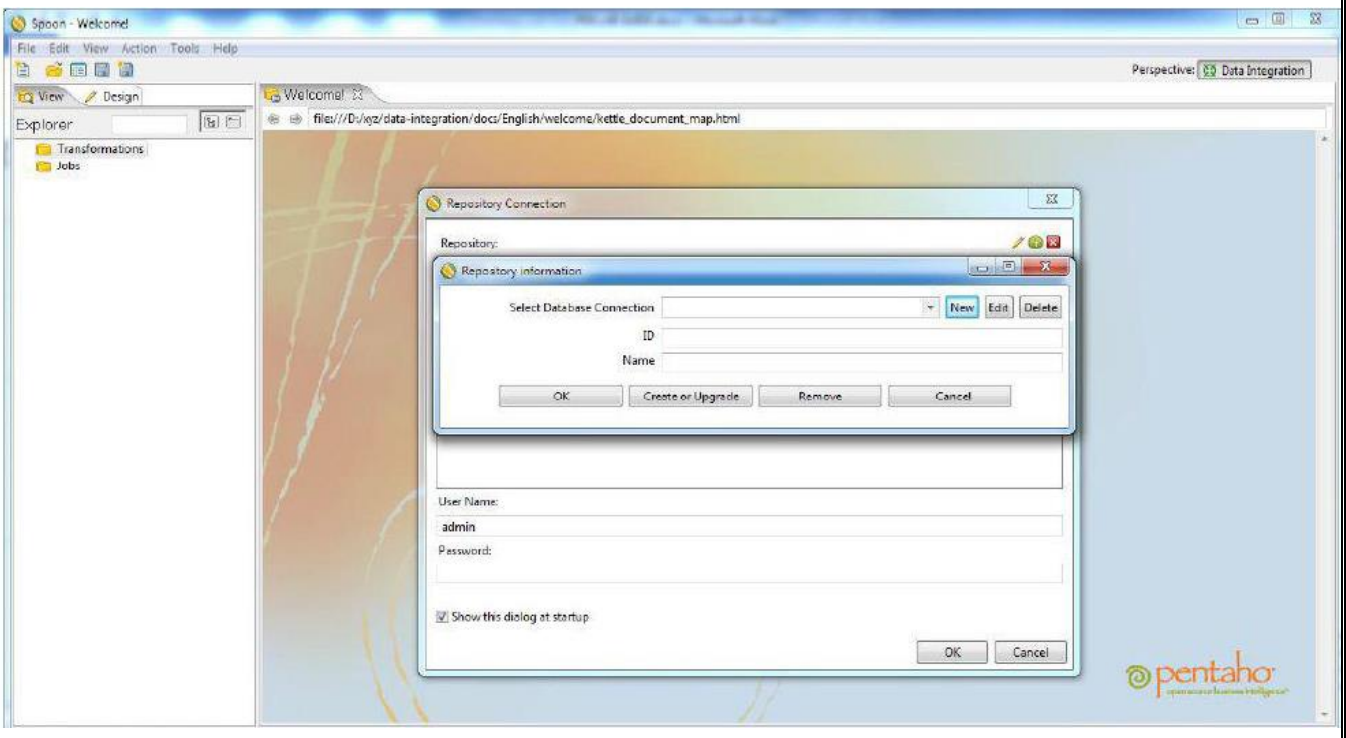


Figure:2

Step 4: In the Database Connection dialog box give a name in the Connection Name textbox, select MS SQL Server in the Connection Type list, and select Native (JDBC) in the Access list. In the Settings level, provide Host Name (Example: localhost/machine name/servername), Database Name, Instance Name (optional), User Name and Password.

Step 5: Now click on the Test button to see the connection status.

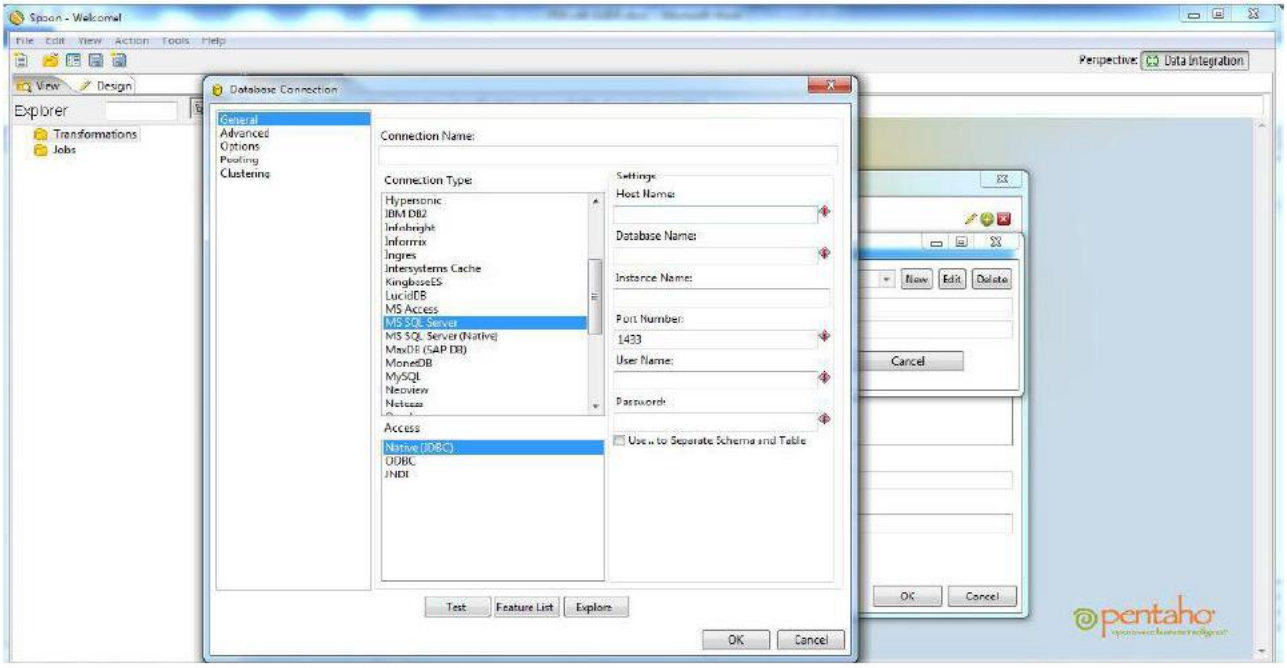


Figure:3

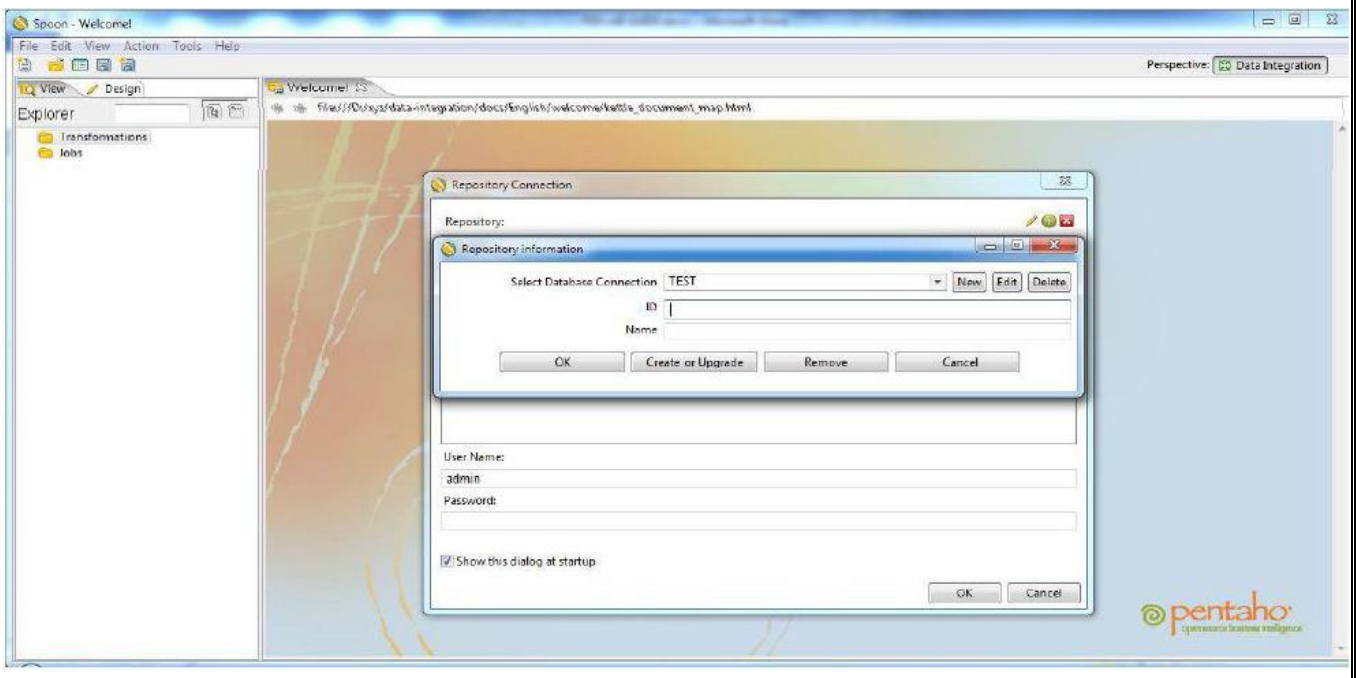


Figure:4

Step 6: In the Repository information dialog box, select the database connection from the drop down box, provide ID (to be displayed in spoon title bar) and Name (the actual name of the repository) and click on the Create and Upgrade button.

Step 7: In the OK dialog box, in response to the question, "Are you sure you want to create the repository on the speci_ed database connection?," if you click on the No button, everything will be created in the database and if you click on the Yes button, it will ask for a dry run.

Step 8: In the Dry run dialog box click on Yes, if you want to see the SQL statements as well as make some modi_cation. Else click on the No button.

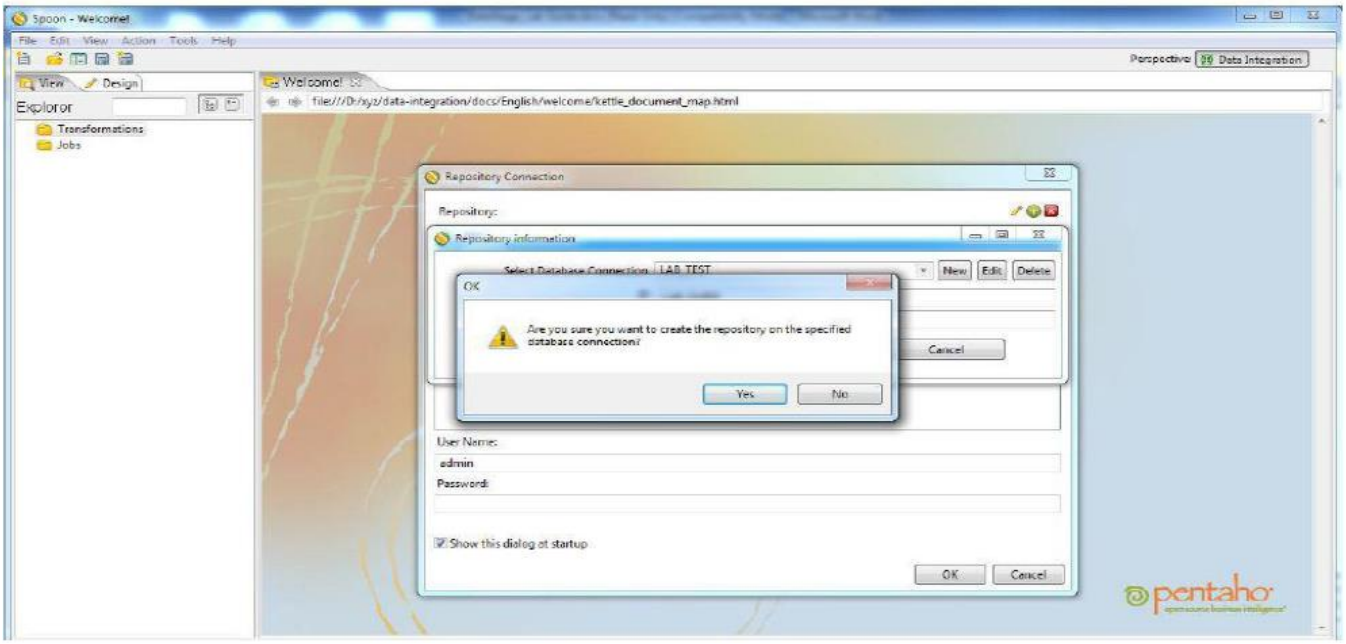


Figure:5

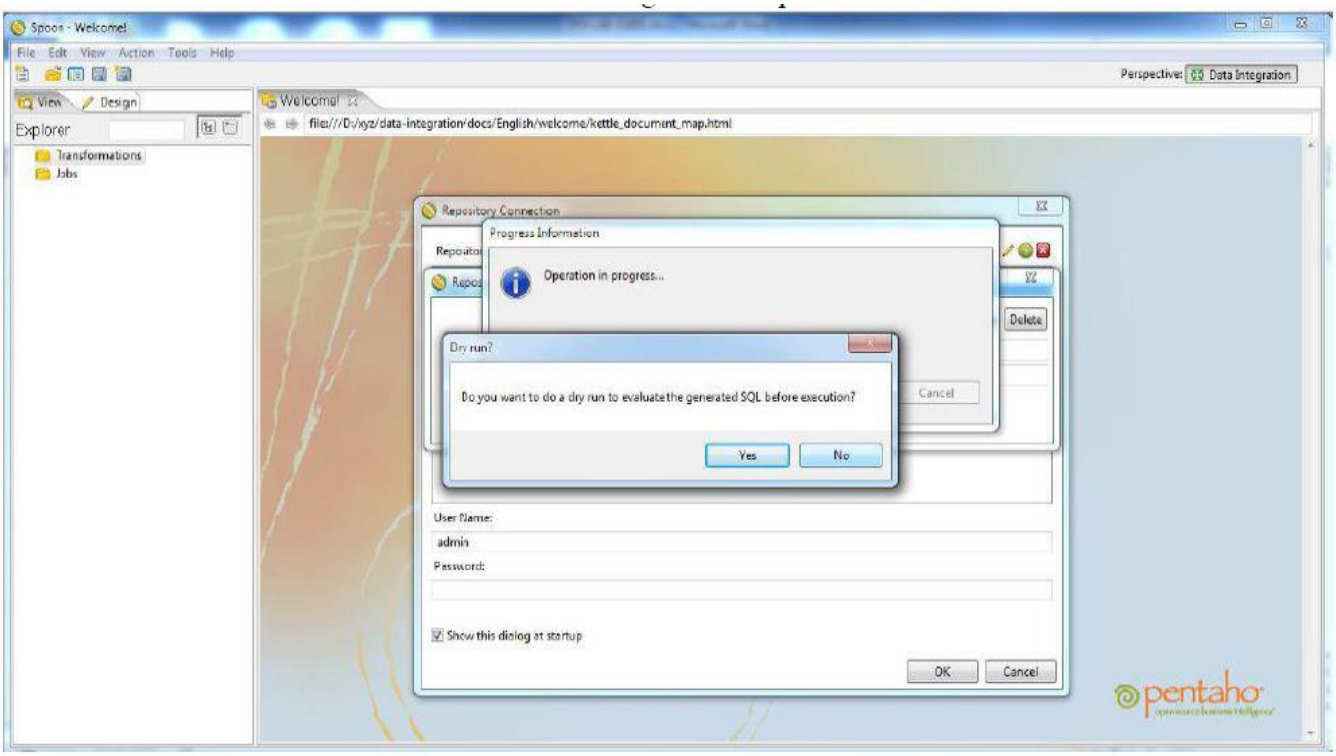


Figure:6

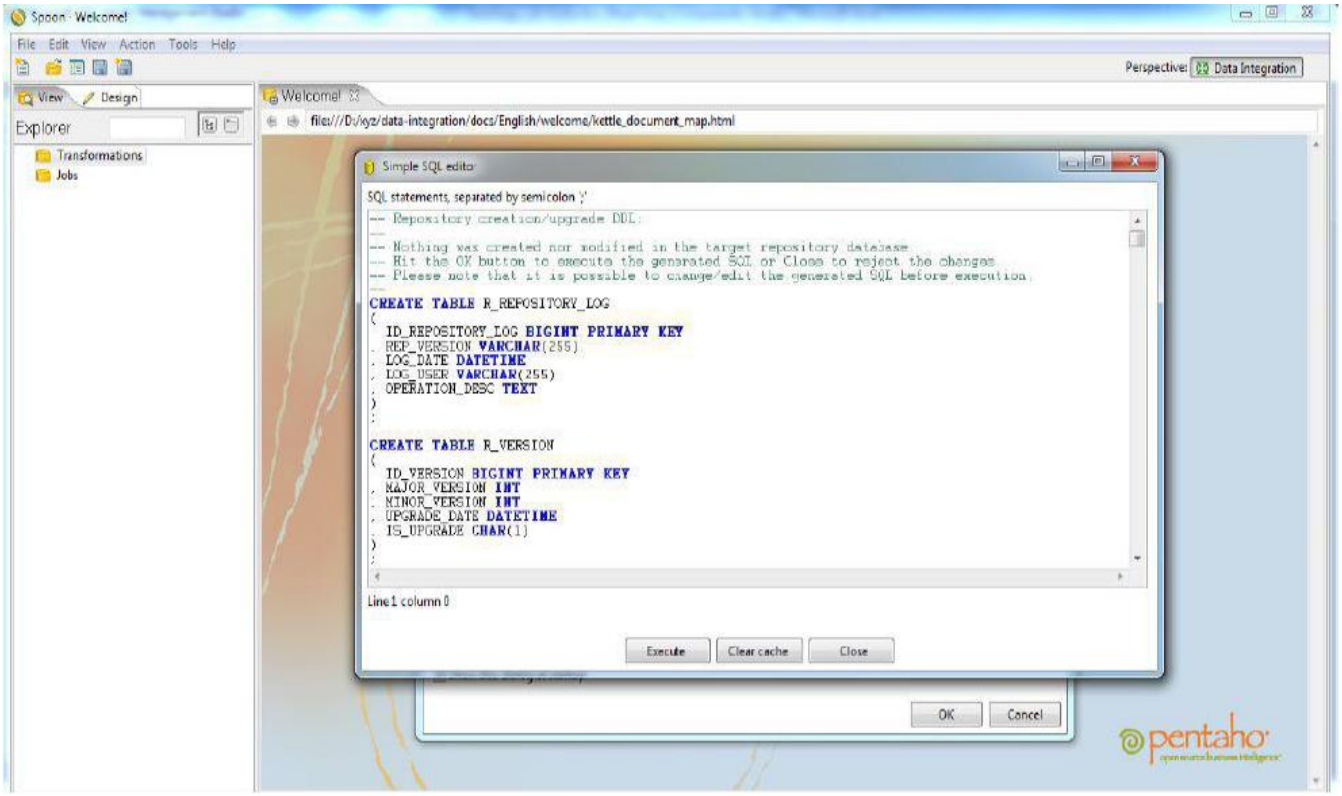


Figure:7

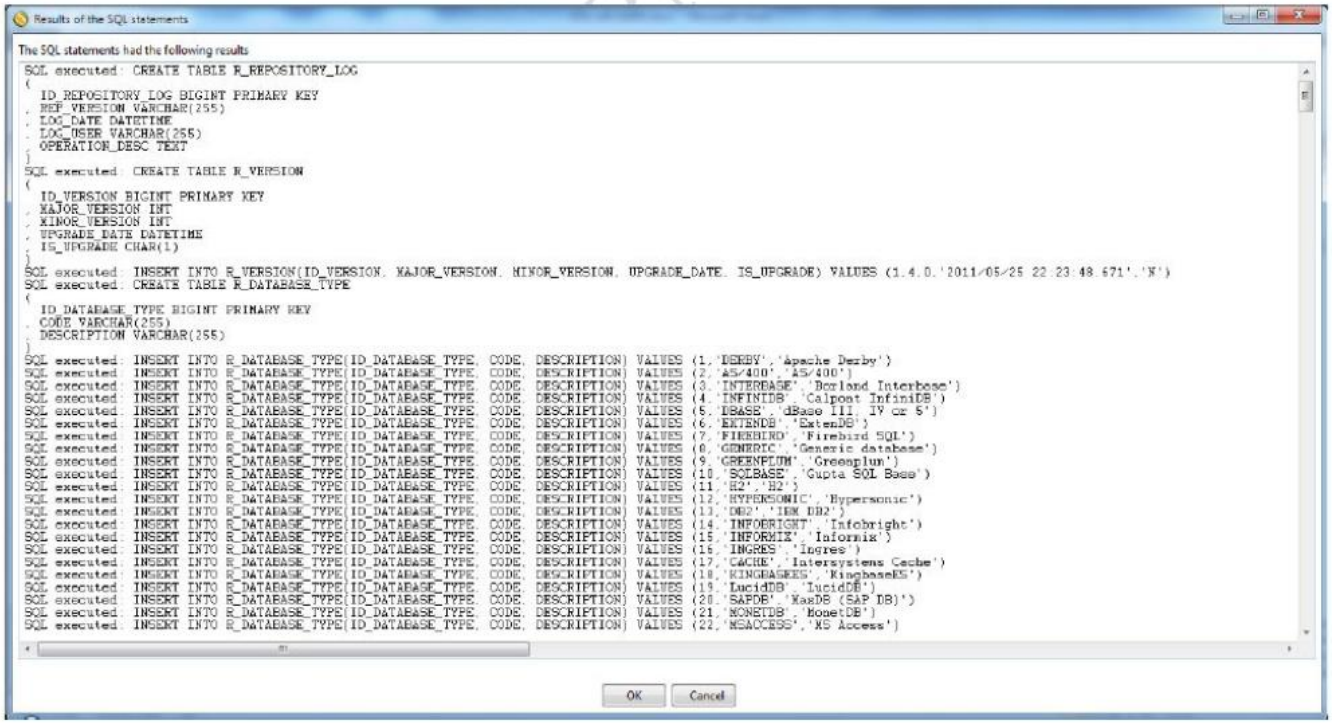


Figure:8

Creating a repository using the Kettle _le repository:-

Step 1: In the Repository Connection dialog box click on the + button. The Select the repository type dialog box will appear.

Step 2: Select Kettle _le repository and click on OK.

Step 3: Now in the File repository settings dialog box click on the Browse button, select a folder, fill the ID and Name, and click on the OK button.

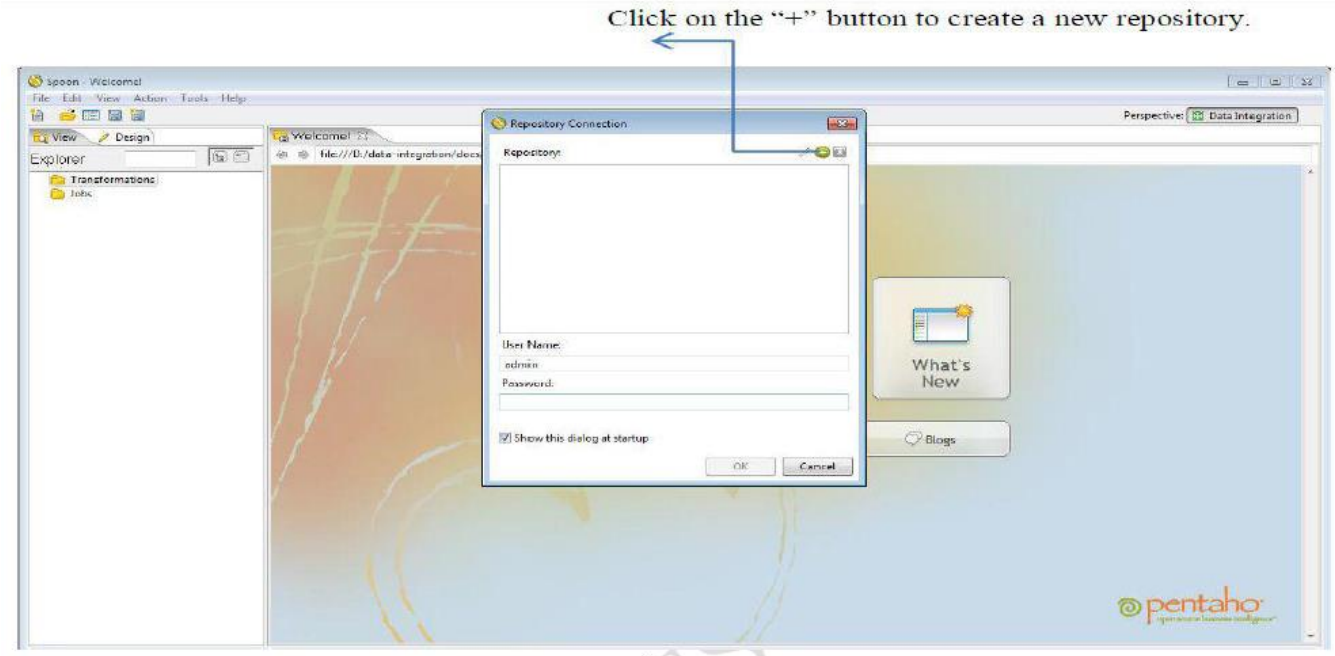


Figure:9

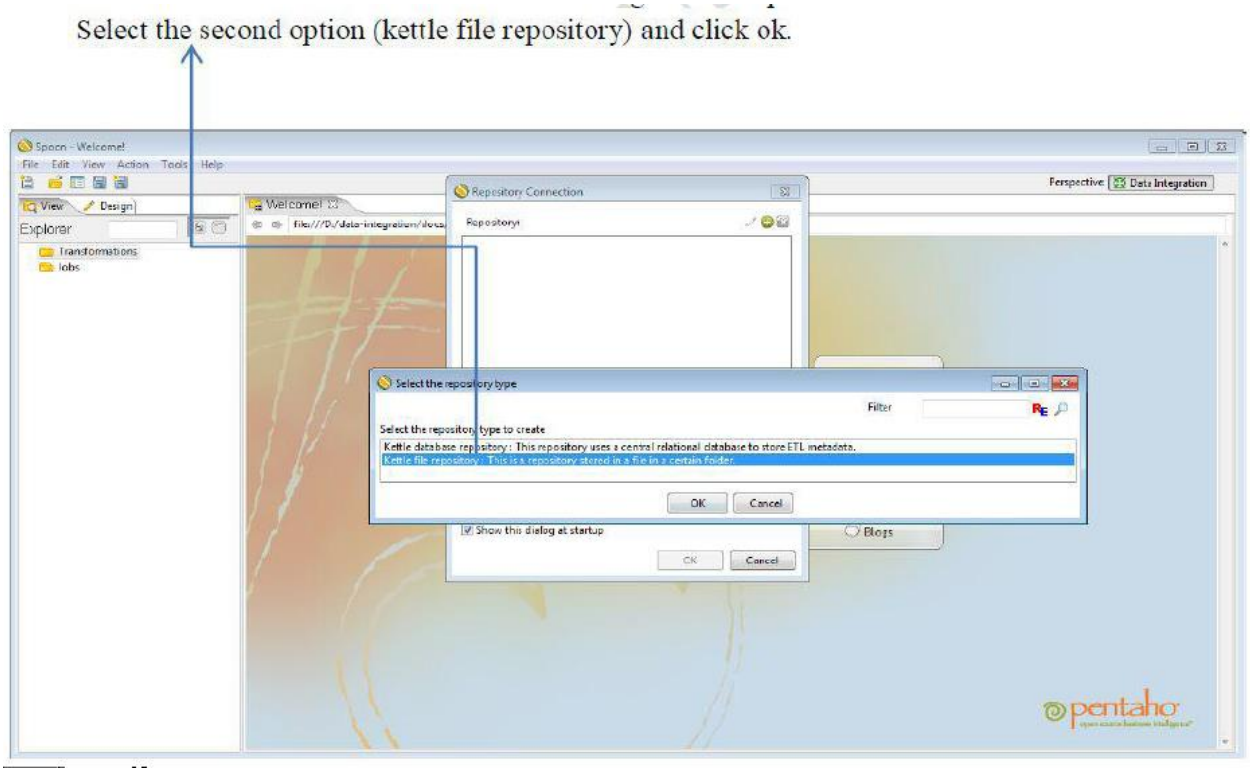


Figure:10

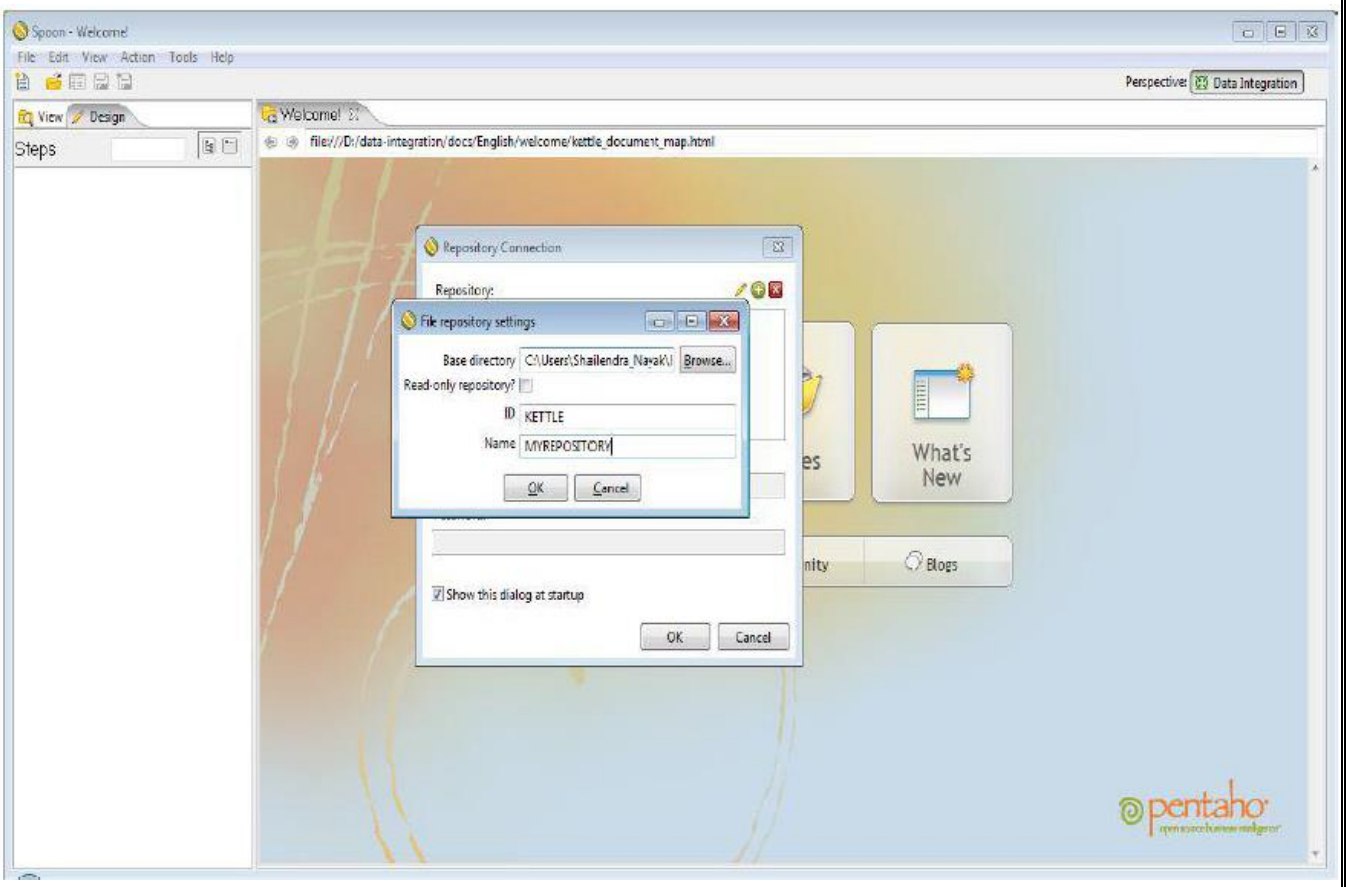


Figure:11

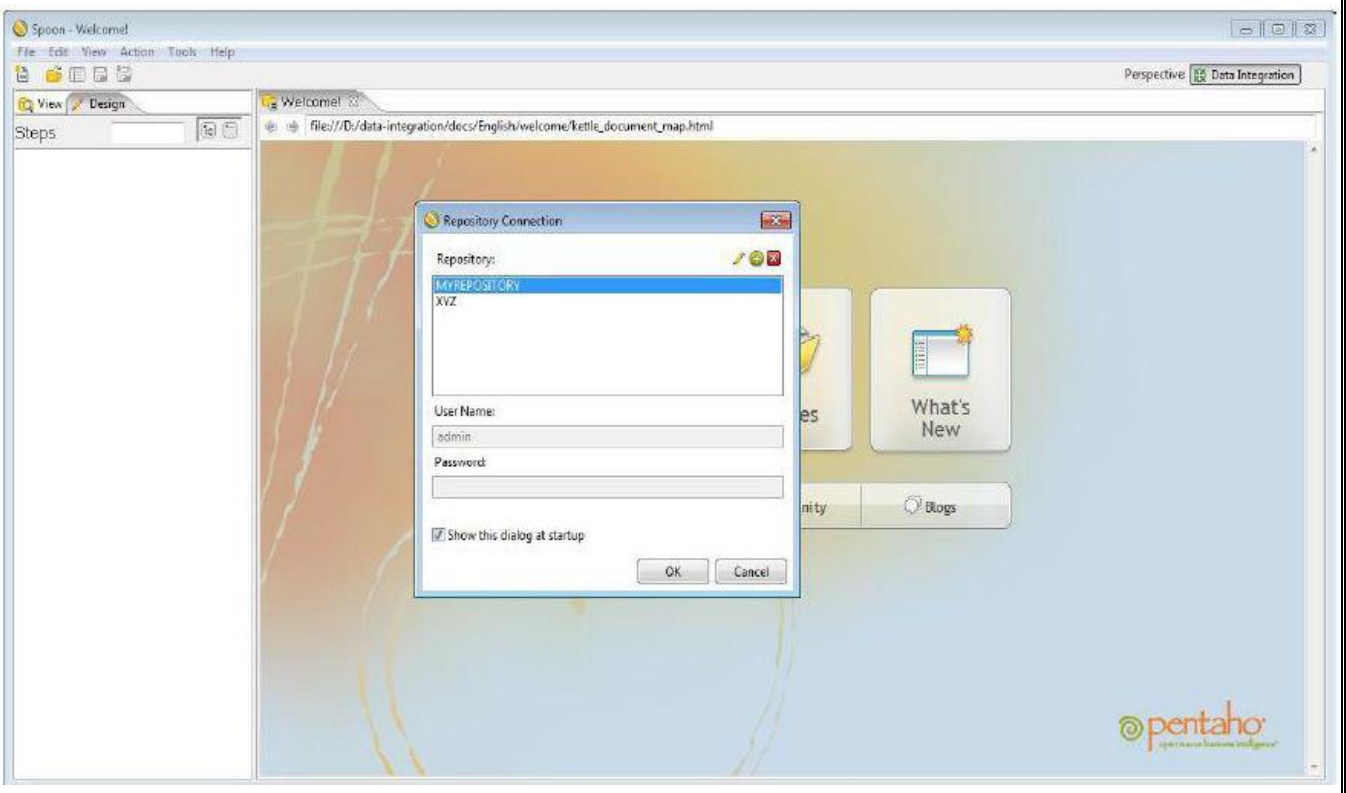


Figure:12

Repository Connection:-

Step 1: Normally the Repository Connection dialog box automatically opens with the opening of the Spoon window. If the Repository Connection dialog box does not open with the Spoon window or suppose you want to open it manually or you want to change your Repository, then: In the Spoon window from menu bar, click on Tools and select the Repository option and click on Connect to open the Repository Connection dialog box.

Step 2: To connect with your desired repository, select the corresponding repository name from the Repository Connection dialog box and click on OK. Note: In case of a database repository connection, you have to give User Name and Password (both are by default admin) and in case of a _le repository connection, User Name and Password are not required.

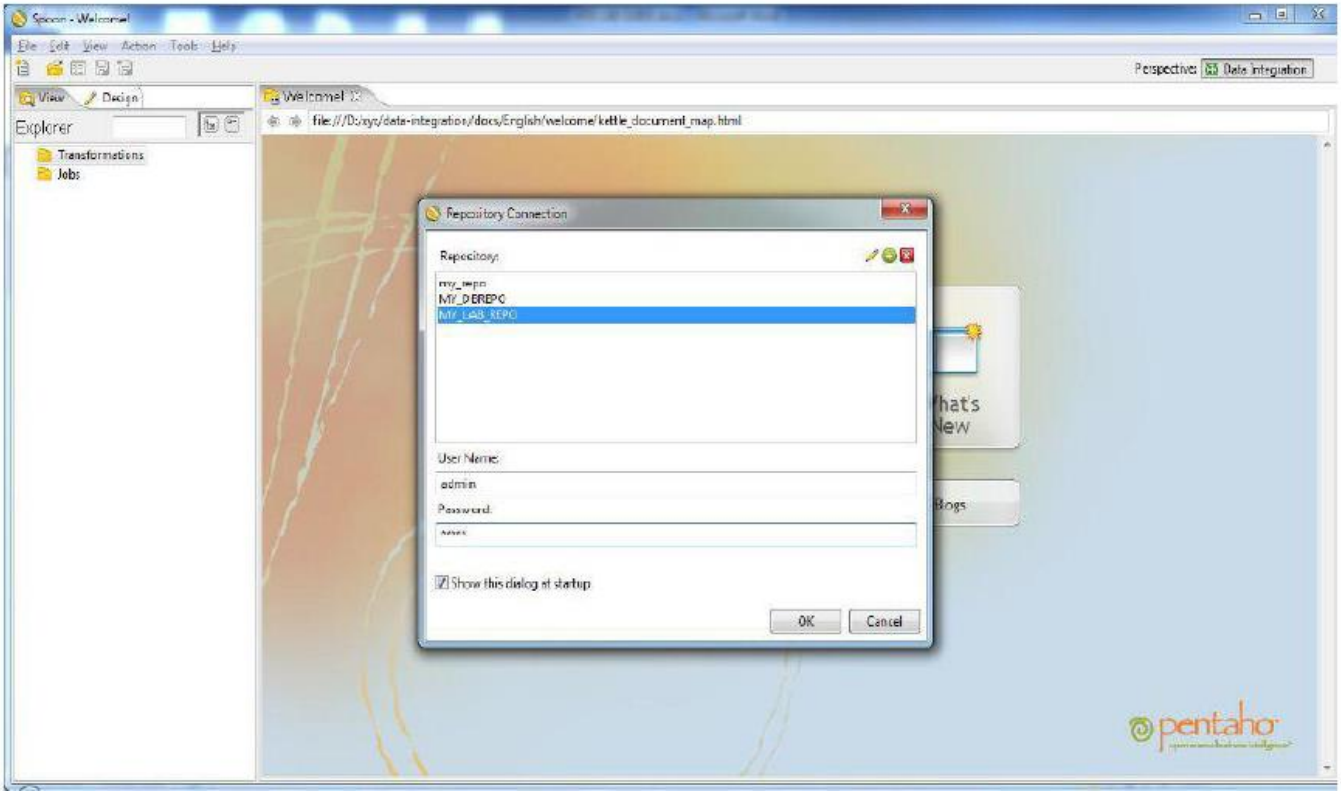


Figure:13

Conclusion: We have successfully performed the computation using Kettle tool.

Assignment No: C2

Title : To study different business and analytics tools.

Aim: Write Select an Industrial sector and write a BIA tool for maximizing the profit

Objectives:

- To study the Business intelligence.
- Select an Industrial sector and write a BIA tool for maximizing the profit.

Theory

BIA:

Business intelligence (BI) can be described as "a set of techniques and tools for the acquisition and transformation of raw data into meaningful and useful information for business analysis purposes". The term "data surfacing" is also more often associated with BI functionality. BI technologies are capable of handling large amounts of unstructured data to help identify, develop and otherwise create new strategic business opportunities. The goal of BI is to allow for the easy interpretation of these large volumes of data. Identifying new opportunities and implementing an effective strategy based on insights can provide businesses with a competitive market advantage and long-term stability.

BI technologies provide historical, current and predictive views of business operations. Common functions of business intelligence technologies are reporting, online analytical processing, analytics, data mining, process mining, complex event processing, business performance management, benchmarking, text mining, predictive analytics and prescriptive analytics.

BI can be used to support a wide range of business decisions ranging from operational to strategic. Basic operating decisions include product positioning or pricing. Strategic business decisions include priorities, goals and directions at the broadest level. In all cases, BI is most effective when it combines data derived from the market in which a company operates (external data) with data from company sources internal to the business such as financial and operations data (internal data). When combined, external and internal data can provide a more complete picture which, in e_ect, creates an "intelligence" that cannot be derived by any singular set of data.

Business intelligence is made up of an increasing number of components including: Multidimensional aggregation and allocation Denormalization, tagging and standardization Real-time reporting with analytical alert A method of interfacing with unstructured data sources Group consolidation, budgeting and rolling forecasts Statistical inference and probabilistic simulation Key performance in dicators optimization Version control and process management Open item management.

BIA tool

Business intelligence tools are a type of application software designed to retrieve, analyze, transform and report data for business intelligence. The tools generally read data that have been previously stored, often, though not necessarily, in a data warehouse or data mart.

Types of business intelligence tools:

The key general categories of business intelligence tools are:

- Spreadsheets
- Reporting and querying software: tools that extract, sort, summarize, and
- present selected data
- OLAP: Online analytical processing
- Digital dashboards
- Data mining
- Process Visualization
- Data warehousing
- Local information systems
- Except for spreadsheets, these tools are provided as standalone tools, suites of tools, components of ERP systems, or as
- Components of software targeted to a specific industry. The tools are sometimes packaged into data warehouse appliances.

Simple cumulative value calculator:

In this approach, the scores of each individual parameter is added, and a cumulative value is obtained. For the maximum salary, the corresponding parameters are displayed. When parameters are given as input, the cumulative value of those parameters is compared with the available dataset. If a nearby match is found, the corresponding company name and salary is displayed.

KNN:

The K Nearest Neighbor (k-NN) is a very intuitive method that classifies unlabeled examples based on their similarity with examples in the training set.k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms.

k-NN advantages

1. The cost of the learning process is zero
2. No assumptions about the characteristics of the concepts to learn have to be done
3. Complex concepts can be learned by local approximation using simple procedures
4. Robust to noisy training data

MATHEMATICAL MODEL

Let S be the solution
S={s, e, i, o, f, DD, NDD, success, failure}
s = initial state that is constructor of the class.
e = be the end state or destructor of the class.
i = input of the system.
o = output of the system.
DD-deterministic data it helps identifying the load store functions or assignment functions.
NDD- Non deterministic data of the system S to be solved.
Success-desired outcome generated.

Failure-Desired outcome not generated or forced exit due to system error.
Success= {Desired outcome generated}
Failure= {Desired outcome not generated or forced exit due to system error}

Algorithm

1. Start
2. Determine parameter K = number of nearest neighbours
3. Calculate the distance between the query instance and all the training samples
4. Sort the distance and determine nearest neighbours based on the K-th minimum distance
5. Gather the category Y of the nearest neighbours
6. Use simple majority of the category of nearest neighbours as the prediction value of the query instance

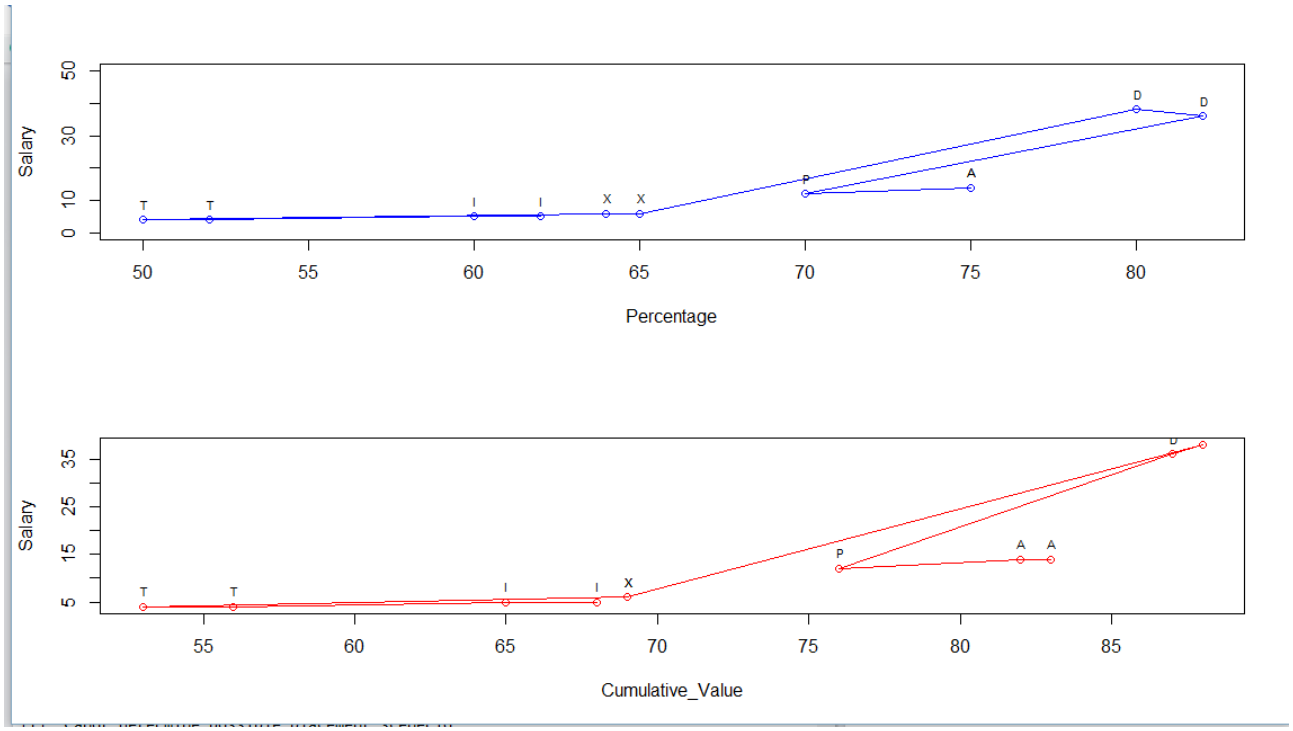
Conclusion:
Thus we have implemented Knn in java.

Code:

```
import java.io.*;
import java.util.*;
class Knn
{
	ArrayList<String> transactions = new ArrayList<String>();
	ArrayList<Double> transCount = new ArrayList<Double>();
	String testData;
	int count,k;
	void print()
	{
		for(int i=0;i<k;i++)
		{
			int index=transCount.indexOf(Collections.min(transCount));
			System.out.println(transactions.get(i));
			transactions.remove(index);
			transCount.remove(index);
		}
	}
	void operate()
	{
		String tData[]=testData.split(" ");
		for(String input:transactions)
		{
			double cost=0;
			String train[]=input.split(" ");
			for(int i=0;i<tData.length;i++)
			{
				try
				{
					cost+=Math.abs(Integer.parseInt(train[i])-Integer.par-
seInt(tData[i]));
				}
				catch(NumberFormatException e)
				{
					if(train[i].equals(tData[i]))
					cost+=1;
				}
			}
			cost/=tData.length;
			transCount.add(cost);
		}
	}
	void input()throws Exception
	{
		BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
		//Input of training data can also be taken from file in case of large input
		/*
		BufferefReader fr = new Bufferedreader(new FileReader(filename));
		String input;
		while((input=fr.readLine())!=null){
		transactions.add(input);
		count++;
	}
	And replace lines below, directly take testing data.
	*/
		System.out.println("Enter No of transactions");
		count=Integer.parseInt(br.readLine());
		System.out.println("Enter the training data in (FirstYearMarks-SecondYearMarks-
ThirdYearMarks-CompanyAssigned)");
		for(int i=0;i<count;i++)
		{
			transactions.add(br.readLine());
		}
		System.out.println("Enter the testing data in (FirstYearMarks-SecondYearMarks-
ThirdYearMarks-enter)");
```

```
                testData=br.readLine();
                System.out.println("Enter the value of k");
                k=Integer.parseInt(br.readLine());
        }
        public static void main(String []args)throws Exception
        {
                Knn knn=new Knn();
                knn.input();
                knn.operate();
                knn.print();
        }
}

/*
Enter No of transactions
5
Enter the training data in (FirstYearMarks-SecondYearMarks-ThirdYearMarks-CompanyAssigned)
890 980 850 Accenture
1040 1030 990 Accenture
1033 1034 1017 TechMahindra
890 870 1015 Infosys
950 1000 990 Congnizant
Enter the testing data in (FirstYearMarks-SecondYearMarks-ThirdYearMarks-enter)
980 990 1080
Enter the value of k
3
890 980 850 Accenture
1040 1030 990 Accenture
890 870 1015 Infosys
*/
```

```
cat ("Reading placement data from records .csv file . Data is : " )
cat ("\n")
heads<-read.csv("record.csv")
heads
Percentage=heads$Percentage
Projects=heads$Projects
Internships=heads$Internships
Papers=heads$Papers
Company=heads$Company
Salary=heads$Salary
#curve ( x^2+x , from=0, to=100)
calculate <-function(Percentage , Projects , Internships , Papers ){
  return ( Percentage+Projects+Internships+Papers )
}
Cumulative_Value=calculate ( Percentage , Projects , Internships , Papers )
par (mfrow = c ( 2 , 1 ) )
plot ( Percentage , Salary , type="o" , col="blue " , ylim = c ( 0 , 50 ) )
text ( Percentage , Salary , labels=Company , cex= 0.7 , pos=3)
plot ( Cumulative_Value , Salary , type="o" , col="red ")
text ( Cumulative_Value , Salary , labels=Company , cex= 0.7 , pos=3)
max_value=max( Salary )
index=match(max_value , Salary)
cat ("The maximum profit ( highest salary ) is in the case of : ")
cat ("\n")
cat ( paste (" Percentage : " , Percentage [ index ] ) )
cat ("\n")
cat ( paste (" Projects : " , Projects [ index ] ) )
cat ("\n")
cat ( paste (" Internships : " , Internships [ index ] ) )
cat ("\n")
cat ( paste (" Papers : " , Papers [ index ] ) )
cat ("\n")
cat ( paste ("With the company being : " ,Company [ index ] ) )
cat ("\n")
cat ( paste (" Cumulative Value being : " , Cumulative_Value [ index ] ) )
cat ("\n")
cat ("*** Refer Rplots.pdf for overview ***")
cat ("\n")
cat (" Enter data to check possible placement company and salary : " )
cat ("\n")
cat (" Enter percentage : ")
percent1 <-80
cat (" Enter number of projects : ")
projects1 <-6
cat (" Enter number of internships : ")
internships1 <-1
cat (" Enter number of papers : ")
papers1 <-2

check_value=percent1+projects1+internships1+papers1
temp<-c( ( check_value-1) : (check_value+1))
storeindex=-1
for(i in 1 : length ( temp ) ) {
  for( j in 1 : length ( Cumulative_Value ) ) {
    if ( temp [ i ]==Cumulative_Value [ j ] ) {
      storeindex=j
      break
    }}}

if(storeindex!=-1){
  print(paste ("Recommended company is : " ,Company [ storeindex] , " with salary : " , Salary))
}
else{
  print("Canot determine possible placement scenerio ")
}
```



Record.csv

	Percentage	Projects	Internships	Papers	Company	Salary
1	75	5	2	1	A	14
2	75	4	0	3	A	14
3	70	4	1	1	P	12
4	82	5	0	0	D	36
5	80	6	1	1	D	38
6	65	4	0	0	X	6
7	64	4	1	0	X	6
8	50	3	0	0	T	4
9	52	3	1	0	T	4
10	60	4	1	0	I	5
11	62	5	0	1	I	5