Distributed Computing

Aliabbas Merchant
Atharva Veer
Divy Patel
Vaibhav Bhujade

Duration of Project: 17/5/2018 - 20/7/2018

Distributed Computing

Abstract

This project aims at reducing the runtime of compute intensive programs working on large data sets, by using multiple computational devices instead of using just one. As of now we are achieving this state by creating instances of this compute intensive program (provided by user) on all available devices. This project is using server – client paradigm with the most compute capable device as server. The server then transfers working data using TCP-IP abstracted by sockets. Suitable data to work on is supplied by the server to the clients. Results are collected and stored into a local file which can be viewed by the user.

Completion status

The program is working and has been tested for the setup of 1 server and 1 client.

Software used

Visual Studio
Dev c++
Gnu c++ compiler
Gedit
Ubuntu 18 and Windows 10 operating systems

Software and Code

Github Link: https://github.com/AliabbasMerchant/distributedComputing

Our program’s basic purpose is to use multiple computers’ resources to compute results faster rather than by using only one computer.

The program has been divided into 2 modules :-

distribute_master: The program at the server-side. This program is made up of 6 subparts:

1. Accepting the user’s program, compiling it, extracting the required data from the program, making changes to the program and generating a new executable file, to be sent to the various clients.
2. Accepting connection requests from all the clients.
3. Sending the generated executable file to all the clients.
4. Sending suitable data to each client, for it to execute the program and compute the result for that data.
5. Receiving the results back from the clients.
6. Storing the results in a local file.

distribute_slave: The program at the slave(client)-side. This program is made up of 5 subparts:

1. Making connections to the server.
2. Accepting the executable file from the server.
3. Accepting the data from the server, for execution.
4. Calculating the results using the given data.
5. Sending the results back to the server.

1. PARSING of user provided program. The program provided by the user is taken as input and a standard template of the program is created. The template is now passed to networking module, so that all the computational devices can receive the template and further compute it.
2. NETWORKING:
   This module is responsible for allowing the clients to connect to the server, send template of user program to all available computational devices, send commands to the clients and receive results back from the clients. at suitable time.

Use and Demo

(This project only will work on Linux platform as the networking module that we created uses Linux operating system’s sockets.)

User just have to code the program and just provide information about where exactly the compute intensive functions and their place of implementation are.

This can be done as shown:-

For compute intensive functions:-

// __CIF__ just before definition of compute intensive function.

For their place of implementation like the loops:-

// __LOOP__ just before the loop where this function is implemented.

Sample structure of the user’s program:

```
// include statements
//__CIF__
// declaration and definition of the compute intensive function
int main() {
        //declaration of variables
        // __LOOP__
        //loop statement
        //loop body calling the CIF
        return 0;
}
```

There are a few restrictions on the user’s program:

1. The program must not accept any command-line arguments
2. There CIF must return only 1 value, of a standard C++ data-type.
3. There should be only 1 CIF.

Sample user program:

```
#include<iostream>

int func(int a)
{
   int b = a * 2;
   return b;
}
int main()
{
   int a[] = {55,77,66,88};
   int b[(sizeof(a)/sizeof(a[0]))];
   for(int i=0; i<(sizeof(a)/sizeof(a[0])); i++)
```

```
    {
        a[i] = func(a[i]);
    }
    return 0;
}
```

Suitable data is now extracted from the user’s program. The program is then modified to generate a template.

Sample template:

```
#include<fstream>
#include<stdlib.h>
#include"helper_funcs.h"
#include<iostream>

int func(int a)
{
    int b = a * 2;
    return b;
}
int main( int argc, char * argv[2]){
    int a[] = {55,77,66,88};
    int b[(sizeof(a)/sizeof(a[0]))];
{
        int i = atoi(argv[1]);
        a[i] = result_store(func(a[i]));
}
        return 0;
}
```

Usage:

On the server(master) side:

```
$ chmod +rwx dmtce
$ ./dmtce <user’s program.cpp> <self IP address> <number of clients>
```

On the client(slave) side:

```
$ chmod +rwx ds
$ ./ds <server IP address> <self IP address>
```

Inheritance ‘18

```
student@vjtiadmin-ThinkCentre-M62z:~/Desktop/speedup$ hostname -I
172.16.100.79
student@vjtiadmin-ThinkCentre-M62z:~/Desktop/speedup$ chmod +rwx dmtce
student@vjtiadmin-ThinkCentre-M62z:~/Desktop/speedup$ ./dmtce user.cpp 172.16.100.79 1
```

```
ies  Terminal                                   Fri 15:26
student@vjtiadmin-ThinkCentre-M62z: ~/Desktop/speedup
File  Edit  View  Search  Terminal  Help
student@vjtiadmin-ThinkCentre-M62z:~/Desktop/speedup$ ./dmtce user.cpp 172.16.100.79 1
172.16.100.79
no_of_clients1
1
on client_connect thread
The program has been successfully compiled!
dc/user_number.cpp
dc/user_template.cpp
The program has been successfully compiled!
The program dc/dist_number has been executed!
The program has been successfully compiled!
Initital work done!
done_listening_for_clients = true


Client has connected!!!!!!!!!!!!!!!!!
Index= 1
Status= 1
IP= 172.16.100.143
Sent_args= 0
NO of PCs = 2
172.16.100.79
172.16.100.143
PORT = 10010
PORT1 = 10010
PORT2 = 10011
number_of_iterations = 4
Args noted
:0000012576:
File sent
Args = 0
assigned_a_pc = true 0
Args = 1
assigned_a_pc = true 1
solving:  iteration_no=0 pc_no=0
command = Args = 2
./dc/dist_template 0
solving:  iteration_no=1 pc_no=1
test: 110
solved = true;
received_ans = 110
------------pc_no = 0, Arg = 0, Answer = 110, iteration_no = 0
Sent to pc. PORT=11000
ANSWER = 154
```

```
------------pc_no = 1, Arg = 1, Answer = 154, iteration_no = 1
assigned_a_pc = true 1
Args = 3
solving:  iteration_no=2 pc_no=1
assigned_a_pc = true 0
solving:  iteration_no=3 pc_no=0
command = ./dc/dist_template 3
test: 176
solved = true;
received_ans = 176
------------pc_no = 0, Arg = 3, Answer = 176, iteration_no = 3
Sent to pc. PORT=11010
ANSWER = 132
------------pc_no = 1, Arg = 2, Answer = 132, iteration_no = 2
```

```
student@vjtiadmin-ThinkCentre-M62z:~/Desktop/speedup$ hostname -I
172.16.100.204
student@vjtiadmin-ThinkCentre-M62z:~/Desktop/speedup$ chmod +rwx ds
student@vjtiadmin-ThinkCentre-M62z:~/Desktop/speedup$ ./ds 172.16.100.79 172.16.100.204
```

```
student@vjtiadmin-ThinkCentre-M62z:~/Desktop/speedup$ ./ds 172.16.100.79 172.16.100.143
PORT = 10001
1
12576
Object file has been received
RECEIVED: 1
./dc/obj 1
test: 154
Sending answer. PORT=11001 ans=154
RECEIVED: 2
./dc/obj 2
test: 132
Sending answer. PORT=11011 ans=132
RECEIVED DONE!!!!
```

Future Work

- We can optimize the code, so as to make the program faster and more efficient.
- We could also edit the code to allow the user’s program to be a bit more flexible, so as to reduce the restrictions imposed on the user’s program.

Bug Report and Challenges

- It was quite difficult to make and maintain socket connections between the server and client.
- We had to use some innovative ways to send the executable file from the server to the clients.
- Parsing of the user’s program and generation of templates did pose some issues.

Bibliography

- https://www.tutorialspoint.com/cplusplus/cpp_multithreading.htm
- C++ sockets: Linux Socket Programming by Example (Book) by Que Publishing
- *Object Oriented Programming with C++ by Balagurusamy*