# User Mannual of PMS Simulator

huarzail

(huarzail@foxmail.com)

2015-04-03

## 1 Introduction

The research of many-core is more and more popular and the mane-core simulation platform is needed. In this paper, I introduce to you a new simulator-PMS which is short for Large-scale parallel simulator for Million-core System. The PMS simulator is a modular platform for computer-system architecture research. It can be used to simulate many-core system which is up to million cores.

## 2 Key features

1. ***Multiple ISA support.*** PMS is composed of many CMP(Chip Multi-core Processor) and many popnet. CMP is based on alpha ISA 64 bits multi-issue simulator.

2. ***Configurable.*** PMS can be configured to form a million-cores system easily.

3. ***Distributed structure and parallel structure.***PMS adopts the shared memory programming mode inside CMP and adopts the distributed programming mode between CMPs.

4. ***Package passing mode.*** The communication mechanism between CMPs and CMP and memory is the massage passing mode, which is same to TCP/IP protocol. In this paper, the message is equivalent to packet.

## 3 Overview the PMS

Now, we can construct the PMS Simulator with CMPs and popnet. The CMP is joined by a popnet which can be configured with (0<*n*<*N*) nodes where the *N* is defined by users, each node connected with a CMP. The top level of PMS is shown in Fig.1.



Fig.1. Overview the PMS simulator

As is shown in Fig.1, there are 4 CMPs connected by a 2x2 mesh popnet, which can simulate 256 cores system. The router (0,1) in the left upper connected a CMP, which is marked as CMP0. The router (1,1) in the right upper connected a CMP, which is marked as CMP1 and so on.

The communication between CMPs is via packet passing though popnet. Inspired by TCP/IP protocol in internet, I implement the read and write through packet passing in the simulator.

In the mechanism of write, the App in CMP0 sends a message to a certain core firstly. Then the core inserts this message to its local router though the router and interface. There is a buffer implemented by FIFO used to restore the subsequent message if the current message has not received the ACK message. The router injects the message to local network and transmits it to router (0,0) which is used to communicate with other CMPs. When the router(0,0) received the message, it analysis the packet and fetch the corresponding data through accesses the memory based on the source address in message and appends them to the tail of message and writes to a file which can be read by outer popnet. The function of popnet is to transmit the message, it can read the file at intervals and sends the message to destination node and write to a file which can be read by the CMP1. A certain router in CMP1 reads the file periodically and writes the data to corresponding address. After that, The CMP1 will send back a ACK message to source CMP to information the source CMP that it had received the data. The ACK message is transmitted as the real data packet. When the CMP0 received the ACK message, the CMP0 fetched a packet from the head of buffer and sent it out. Thus, the write operation is over. The flow of write operation is shown in Fig.2.



Fig.2. The write operation flow

The read operation is complex compared to the write operation. In the mechanism of write, if the App in CMP0 want to read some data from CMP1. At firstly, the cmp0 sends a message to CMP1 and suspends the core(a thread) until it received the response message of CMP1. When the core receives the packet and analysis the packet and fetchs the data based on the address of read. then it can append the data to the tail of original pachet and send to CMP1, the process of message passing is same as write operation. The flow of write operation is shown in Fig.3.

 Fig.3. The read operation flow

After each CMPs receives the packet, it must to analyze the message to distinguish the operation is read or write.

**3.2** **How to guarantee the data is received**

When a core in one CMP reads data from another CMP, it needs to wait for the response received before it goes on. So I add some code to suspend the core at the beginning in read syscall in syscall() and add some other code to continue the core after the response has finished the receive operation in the read package in local network. Every read event contains an id of the core which created it. If the response package has received, the core goes on. The process is shown in Fig.4



Fig.4. Suspend the current and wait for the response

**4 Internal of PMS**

**4.1 Internal of CMP**

In the CMP, an App must be compiled into an executable file. The core will read the binary file and fetch the instruction. At the same time, When the App will communicate with other CMPs, the App will call the system functions to send a read or write packet, then the packet will be inserted into network via the function *popnetMessageInsert*( ) .The popnet via *inject packet*( )function to inject the packet into the nework and began to traversal. At last, the packet will arrival at router (0,0) via *accept\_flit* ( )function and write tofile *pt\_net\_in.txt*. Simultaneously, the router in CMP will read file *pt\_net\_x.txt* via function *readfile*( ) to get other CMP’s packet. Then the core must to analysis the packet to distinguish the packet operation is read or write. If the operation is write, then the CMP fetches the data from the packet and write to corresponding local address. If the operation is read, then the CMP reads the data from a certain local address through memory access according the destination address in packet and packages the data to original packet and exchange the cmp number and exchange the address. At last, the new packet is inserted to network and transfer to source CMP. The internal of CMP is shown in Fig.5.

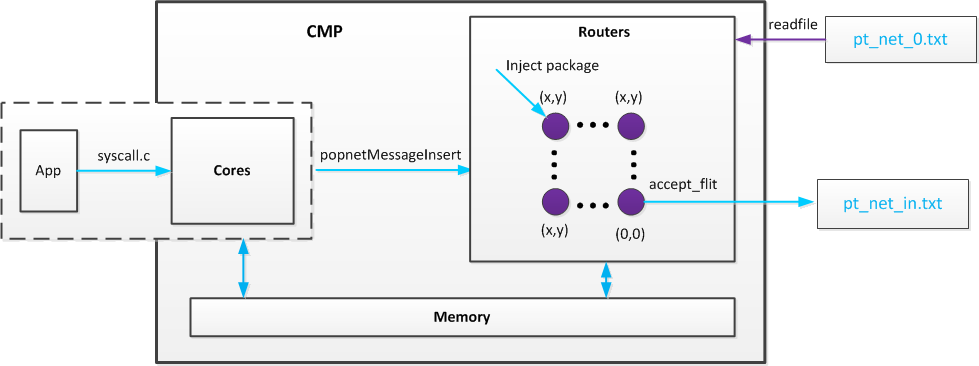


Fig.5. internal of CMP

**4.2 Internal of popnet**

The popnet reads the ***pt\_net\_in.txt*** via function *readpackege*( ) periodically and fetches a packet each time, and inserts it to network. The packet is transmitted in the network hop by hop. The routing algorithm is XY\_Routing. When the destination router received the packet, the popnet writes this packet to corresponding file ***pt\_net\_x.txt*** according to the destination CMP. The flow of this process is shown in Fig.6.

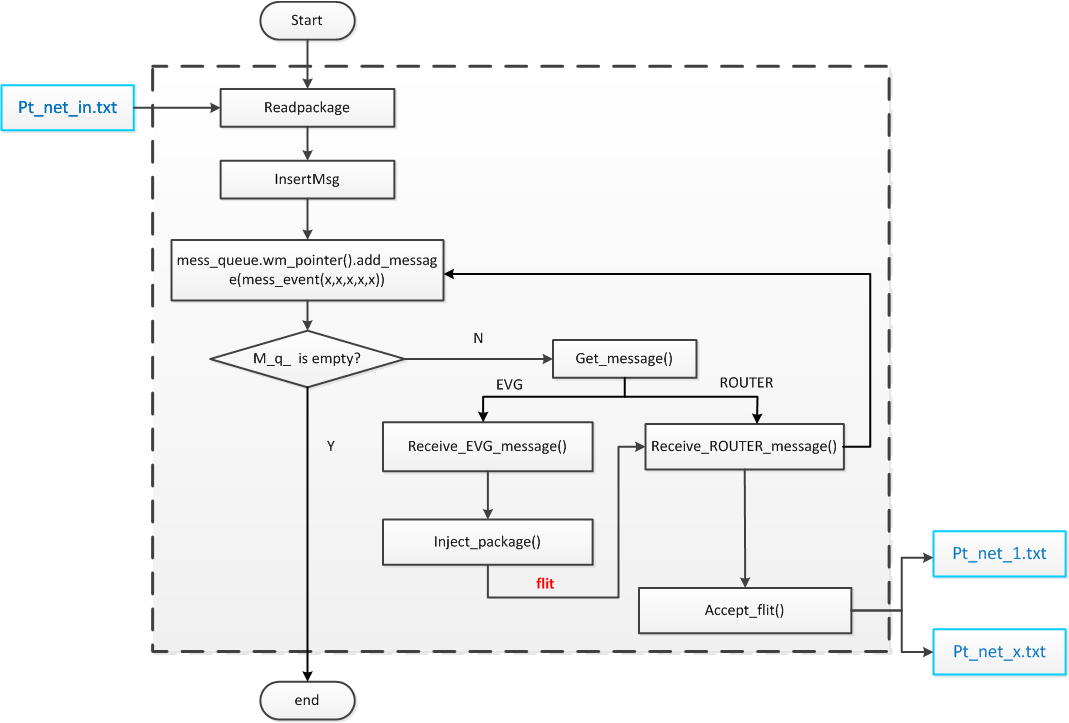


Fig.6. internal of popnet

## 5 The modification contents

**5.1. Modify of CMPs.**

When APPs want to read data from other CMPs or write data to other CMPs, they must send a request with MTA\_mailboxsend which is a system call and transmit this message to core of local CMP. The format of OSF\_SYS\_mailboxsend’s parameters is:

**[Source\_cmp] [Source\_address] [Destination\_cmp] [Destination\_address] [Operation] [Data\_ length]**

The meaning of this parameters is:

|  |  |  |
| --- | --- | --- |
| **Syscall Name** | **Function** | **Arguments** |
| **OSF\_SYS\_mailboxsend** | **Send the message (packet)** | regs->regs\_R[**R\_A0**]: source ID of cluster  regs->regs\_R[**R\_A1**]: local memory address  regs->regs\_R[**R\_A2**]: target ID of cluster  regs->regs\_R[**R\_A3**]: target memory address  regs->regs\_R[**R\_A4**]:operation(read/write)  regs->regs\_R[**R\_A5**]: number of integer to write |

For instance, OSF\_SYS\_mailboxsend(0,0xFF0000000 1,0xFF0000001,1,1000) show that a packet is generated at address 0xFF0000000 of CMP0 and write to address 0xFF0000001 of CMP1. The data length is 1.

1. Codes modified in ***syscall.c*** include:

Definition a syscall number (add line 768):

***#define OSF\_SYS\_mailboxsend 197; //Jianhua Li 20150324***

Definition the OSF\_SYS\_mailboxsend (add line 2550):

***case OSF\_SYS\_mailboxsend: {***

***struct DIRECTORY\_EVENT \*event;***

***extern md\_addr\_t src\_addr;***

***extern md\_addr\_t dst\_addr;***

***extern int src\_cmp;***

***extern int dst\_cmp;***

***extern int data\_length;***

***extern int operation;***

***src\_cmp = regs->regs\_R[R\_A0];***

***src\_addr = (md\_addr\_t)regs->regs\_R[R\_A1];***

***dst\_cmp = regs->regs\_R[R\_A2];***

***dst\_addr = (md\_addr\_t)regs->regs\_R[R\_A3];***

***data\_length = regs->regs\_R[R\_A4];***

***operation = regs->regs\_R[R\_A5];***

***popnetMsgNo++;***

***popnetMessageInsert(current->id/8+1, current->id%8, 0, 0, sim\_cycle, 5, popnetMsgNo, (md\_addr\_t) dst\_cmp, 1, 0,(md\_addr\_t)src\_addr, data\_length, (md\_addr\_t)src\_cmp, (md\_addr\_t) dst\_addr, operation);***

***}break;***

2. Codes modified in ***sim-outorder.c*** include.

(1). If the operation is **read**, the CMP will fetch the data from local address.

***void read\_memory(int mailbox[], long data\_length, md\_addr\_t src\_addr)***

***{***

***int \*p;***

***long size;***

***size = data\_length;***

***p = mailbox;***

***while(size--){***

***mem\_access\_drct (thecontexts[0]->mem, Read, src\_addr, p, sizeof(int), 0);***

***p++;***

***src\_addr += sizeof(int);***

***}***

***}***

(2). If the operation is **write**, the CMP will write the data to local arrdess:

***void write\_memory(int data\_length, md\_addr\_t dst\_addr, int mailbox[])***

***{***

***int \*p,\*q;***

***p = mailbox;***

***int a=0;***

***q =&a;***

***while(data\_length--){***

***mem\_access\_drct (thecontexts[0]->mem, Write, dst\_addr, p, sizeof(int), 0);***

***p++;***

***dst\_addr +=sizeof(int);***

***}***

***}***

**5.2. Modify inner popnet of CMP**

After the packet is generated, the packet is inserted into network and transmits to router (0,0). Because the router (0,0) is an interface which is responsible for communicating with other CMPs. When the router (0,0) received the packet, the real data is fetched from source address of packet and write to a file named *pt\_net\_in.txt*, which is read by of outer popnet. The format of packet is:

**[Cycle][Delay][Source\_cmp][Source\_address][Destination\_cmp][Destination\_address] [Operation][Data\_length][Message\_num][Sync\_info][Real\_data]**

The meaning of parameters is:

|  |  |
| --- | --- |
| **Cycle** | The time of this packet generated |
| **Delay** | The transmission delay of this packet so far |
| **Source\_cmp** | The source CMP number |
| **Source\_address** | The source address of this packet |
| **Destination\_cmp** | The destination CMP number |
| **Destination\_address** | The destination address of this packet |
| **Operation** | The read or write |
| **Data\_length** | The data length of packet |
| **Message\_num** | The number of this packet, it is unique |
| **Sync\_info** | The sync information |
| **Real\_data** | The data want to transport |

1. Codes modified in ***popnetForSimplescalar/index.h*** include:

Define a struct to store the information of a packet in header file ***index.h***(added in line 49):

***typedef struct message {***

***long src1;***

***long src2;***

***long dst1;***

***long dst2;***

***long long int sim\_cycle\_m;***

***int src\_cmp\_m;***

***unsigned long long src\_addr\_m;***

***int dst\_cmp\_m;***

***unsigned long long dst\_addr\_m;***

***int data\_length\_m;***

***int operation\_m;***

***long long int messageNo\_m;***

***int size\_m;***

***int vc\_m;***

***} mess\_struct\_type;***

2. Codes modified in ***popnetForSimplescalar/mainPopnet.cc*** include:

(1) Store the information of a packet using a structure array in function ***popnetMessageInsert*** (add line 218):

***if(d1==0&&d2==0){***

***MessageInfo[countNum].src1 = s1;***

***MessageInfo[countNum].src2 = s2;***

***MessageInfo[countNum].dst1 = d1;***

***MessageInfo[countNum].dst2 = d2;***

***MessageInfo[countNum].sim\_cycle\_m = sim\_cycle;***

***MessageInfo[countNum].src\_cmp\_m = src\_cmp;***

***MessageInfo[countNum].src\_addr\_m = src\_addr;***

***MessageInfo[countNum].dst\_cmp\_m = dst\_cmp;***

***MessageInfo[countNum].dst\_addr\_m = dst\_addr;***

***MessageInfo[countNum].data\_length\_m = data\_length;***

***MessageInfo[countNum].operation\_m = operation;***

***MessageInfo[countNum].messageNo\_m = msgNo;***

***MessageInfo[countNum].size\_m = size;***

***MessageInfo[countNum].vc\_m = vc;***

***countNum++;***

***}***

(2). Add codes to control the sending FIFO whether received the ACK signal in the function ***popnetMessageInsert*** (add line 236):

***int sendFlag = 0;//ljh***

***int firstFlag = 0;***

***long sendNum = 0;***

***if(d1==0&&d2==0){***

***……………***

***if(firstFlag == 0 )***

***{***

***network\_mess\_queue\_ptr->insertMsg(MessageInfo[sendNum].src1,***

***MessageInfo[sendNum].src2, MessageInfo[sendNum].dst1,***

***MessageInfo[sendNum].dst2, MessageInfo[sendNum].sim\_cycle\_m,***

***MessageInfo[sendNum].size\_m, MessageInfo[sendNum].messageNo\_m,***

***MessageInfo[sendNum].dst\_cmp\_m, (long)MessageInfo[sendNum].vc\_m);***

***firstFlag = 1;***

***sendNum++;***

***}***

***if(sendFlag == 0)***

***{***

***;***

***}***

***else if(sendFlag == 1)***

***{***

***network\_mess\_queue\_ptr->insertMsg(MessageInfo[sendNum].src1,***

***MessageInfo[sendNum].src2, MessageInfo[sendNum].dst1,***

***MessageInfo[sendNum].dst2, MessageInfo[sendNum].sim\_cycle\_m,***

***MessageInfo[sendNum].size\_m, MessageInfo[sendNum].messageNo\_m,***

***MessageInfo[sendNum].dst\_cmp\_m, (long)MessageInfo[sendNum].vc\_m);***

***sendNum++;***

***sendFlag = 0;***

***}***

***}***

***else{***

***network\_mess\_queue\_ptr->insertMsg(s1, s2, d1, d2, sim\_cycle, size, msgNo,***

***dst\_cmp, (long)vc);***

***}***

(3). Define the function ***readfile***() (Add line 421)

① Read ***pt\_net\_in.txt*** line by line and use a file lock.

***unsigned int fileLine = GetFileContent(readFile, filecontent);***

***if ((fp = fopen(filename, "r")) == NULL){***

***printf("file open error!\n");***

***exit(0);***

***}***

***if (flock(fp->\_fileno, LOCK\_EX) != 0)***

***fclose(fp);***

***ifstream infile (filename);***

***while(!infile) {***

***cerr<<"Can not open source file."<<endl;***

***assert(0);***

***}***

***infile.seekg(ios::beg);//file point to begin***

***while(!infile.eof()){***

***getline(infile,str);***

***if(infile.fail()) {***

***break;***

***}***

***filecontent.push\_back(str);***

***counterLine++;***

***}***

***infile.close();***

***infile.clear();***

***flock(fp->\_fileno, LOCK\_UN);***

***return counterLine; // return the number of file line***

***}***

② Division the string to character string.

***std::vector<std::string>***

***split(std::string str,std::string pattern)//divide the string***

***{***

***std::string::size\_type pos;***

***std::vector<std::string> result;***

***str+=pattern;***

***int size=str.size();***

***for(int i=0; i<size; i++)***

***{***

***pos=str.find(pattern,i);***

***if(pos<size)***

***{***

***std::string s=str.substr(i,pos-i);***

***result.push\_back(s);***

***i=pos+pattern.size()-1;***

***}***

***}***

***return result;***

***}***

③ Convert the character string to value.

***bool ConvertStringToNum(string lineSrc, long long int \*numDest)***

***{***

***bool bRet = false;***

***std::vector<std::string> stringResult = split(lineSrc," "); //divide the string to single value***

***if(stringResult.size()>0){***

***for(int i=0; i<stringResult.size(); i++)***

***{***

***char \*stopstring;***

***numDest[i] = strtoll(stringResult[i].c\_str(),***

***&stopstring, 10);***

***}***

***bRet = true;***

***}***

***return bRet;***

***}***

④ Judge the message type of after read the message from file ***pt\_net\_x.txt***.

1). If the operation is 1, it is stand for other CMP want to write data to this CMP. That is to say, after this CMP write data to local memory, this CMP must to send a ACK message back to source CMP to inform that I have received the data.

***if(operation\_t == 1){ //operation equal to 1 is stand for other clusters want to write this cluster.***

***write\_memory(data\_length\_t, dst\_addr\_t, mailbox\_receive);***

***operation\_t =2;***

***ofstream fout(writeFile,ios::app);***

***if(!fout){***

***cerr<<"Can not open source file."<<endl;***

***assert(0);***

***}***

***fout << event\_time\_t+1000 <<" " <<costCycle << " " << dst\_cmp <<" " << dst\_addr\_t <<" " << src\_cmp << " " << src\_addr\_t <<" " << "1" <<" " << operation\_t <<" " << messageNo\_t\*10+1 <<" "<<"111"<<endl;***

***fflush(stdout);***

***fout.close();***

***ofstream fout\_b(writeFileBackup,ios::app);***

***if(!fout){***

***cerr<<"Can not open source file."<<endl;***

***assert(0);***

***}***

***fout\_b << event\_time\_t+1000 <<" " <<costCycle << " " << dst\_cmp <<" " << dst\_addr\_t <<" " << src\_cmp << " " << src\_addr\_t <<" " << "1" <<" " << operation\_t <<" " << messageNo\_t\*10+1 <<" "<<"111"<<endl;***

***fflush(stdout);***

***fout\_b.close();***

***}***

2). If the operation is 0, it is stand for other CMP want to **read** data from this CMP. That is to say, after this CMP received the read request, this CMP must to fetch the data from a certain address in local memory and appended to original message and send back to source CMP .

***else if(operation\_t == 0){ //operation equal to 0 is stand for this cluster want to read other clusters.***

***operation\_t = 1;***

***read\_memory(mailbox\_send, data\_length\_t, dst\_addr\_t);***

***ofstream fout(writeFile, ios::app);***

***if(!fout){***

***cerr<<"Can not open source file."<<endl;***

***assert(0);***

***}***

***fout << event\_time\_t <<" " <<Time << " " << dst\_cmp <<" " << dst\_addr\_t <<" " << src\_cmp << " " << src\_addr\_t <<" " << data\_length\_t <<" " << operation\_t <<" " << messageNo\_t <<" ";***

***for(int i=0; i<data\_length\_t; i++){***

***fout << mailbox\_send[i] <<" ";***

***}***

***fout << endl;***

***fflush(stdout);***

***fout.close();***

***ofstream fout\_b(writeFileBackup, ios::app);***

***if(!fout\_b){***

***cerr<<"Can not open source file."<<endl;***

***assert(0);***

***}***

***fout\_b << event\_time\_t <<" " <<Time << " " << dst\_cmp <<" " << dst\_addr\_t <<" " << src\_cmp <<" " << src\_addr\_t <<" " << data\_length\_t <<" " << operation\_t <<" "<< messageNo\_t<< " ";***

***for(int i=0; i<data\_length\_t; i++){***

***fout\_b << mailbox\_send[i] <<" ";***

***}***

***fout\_b << endl;***

***fflush(stdout);***

***fout\_b.close();***

***}***

3). If the operation is 2, it is stand for other CMP send a ACK message to this CMP. That is to say, after this CMP received the ACK message, this CMP must to modify a flag which is used for control the message queue whether to send message go on.

***else if(operation\_t == 2)***

***{***

***sendFlag = 1;***

***printf("I received the ACK signal.\n");***

***}***

(4). Define the function ***accept\_flit*** () (Add line 405)

When the router (0,0) received the packet, it will fetch the data and write file ***pt\_net\_x.txt*** and use a file lock:

***if(y==0&&z==0){***

***for(j=0; j<countNum; j++)***

***if(MessageInfo[j].src1 == w && MessageInfo[j].src2 == x && MessageInfo[j].messageNo\_m == msgNo)***

***{***

***insertNetCycle = MessageInfo[j].sim\_cycle\_m;***

***src\_cmp = MessageInfo[j].src\_cmp\_m;***

***src\_addr = MessageInfo[j].src\_addr\_m;***

***dst\_cmp = MessageInfo[j].dst\_cmp\_m;***

***dst\_addr = MessageInfo[j].dst\_addr\_m;***

***data\_length = MessageInfo[j].data\_length\_m;***

***operation = MessageInfo[j].operation\_m;***

***FILE \*fp\_o;***

***fp\_o = fopen (writeFile, "a+");***

***while(fp\_o == NULL){***

***cout << "package for popnet is not ready!!" << endl;***

***assert(0);***

***}***

***simNetCycle = sim\_cycle - insertNetCycle;***

***if (flock(fp\_o->\_fileno, LOCK\_EX) != 0)***

***printf("file pt\_net\_in.txt lock by others\n");***

***if(operation == 0){***

***if(dst\_cmp == 1){***

***fprintf(fp\_o, "%llu %llu 0 0 %lld 0 1 %lld %d %d %d ",sim\_cycle,simNetCycle, src\_addr, dst\_addr, data\_length, operation, msgNo);***

***for(i=0; i<data\_length; i++)***

***{***

***fprintf(fp\_o,"%d ",read\_data);***

***}***

***fprintf(fp\_o,"\n");***

***fflush(stdout);***

***}***

***}***

***}***

***}***

**5.3. Modify outer popnet**

The popnet read the ***pt\_net\_in.txt*** periodically and fetches a packet each time, and inserts it to network. The packet is transmitted through the network hop by hop. The routing algorithm is XY\_Routing .When the destination router received the packet, the popnet writes this packet to corresponding file ***pt\_net\_x.txt*** according to the destination CMP. The process is same as popnet in CMPs.

## 6 Application example

When use this simulator, The Benchmark must be write by yourself. In this section, I will instruct to how to run a app in PMS.

**6.1 Benchmark**

Such as, we use a SSSP parallel algorithm as the benchmark. File list need in this example is shown in follow.

Figure 7. File list need in this SSSP example

***sim-outorder*** : PMS introduced here.

***SSSP, SSSP.BNC, inpit.txt***: The ***SSSP*** and the ***SSSP.BNC*** is the executable program and its arguments set file. ***inpit.txt*** is input data file. These files should be set based on the application you select to run in the simulator.

***SSSP.sh***: the shell script used to run the simulator, set output file paths, select configure file and set some other arguments.

***barrier.sh*** : the configuration files.

***config\_mesh\_ooo\_xy\_org*** :

The routing algorithm selection argument is set in this configurable file, and in ***config\_mesh\_ooo\_xy\_org*** we select dimension routing

***-mesh\_network:routing\_algr 0***

(1). Needs to define some macro and global address:

***#define global***

***#include "stdio.h"***

***#include "barrier.h"***

***#include "stdint.h"***

***#define NPROC 64***

***volatile int CountLock = 0;***

***#define cmp0\_0 0x1ff000000***

***#define cmp0\_1 0x1ff100000***

***#define cmp0\_2 0x1ff200000***

***#define cmp0\_3 0x1ff300000***

***#define cmp0\_4 0x1ff400000***

***#define cmp0\_5 0x1ff500000***

***#define cmp1\_0 0x1ff600000***

***#define cmp1\_1 0x1ff700000***

***#define cmp1\_2 0x1ff800000***

***#define cmp1\_3 0x1ff900000***

***#define cmp1\_4 0x1ffA00000***

***#define cmp1\_5 0x1ffB00000***

***#define cmp1\_6 0x1ffC00000***

***#define cmp1\_7 0x1ffD00000***

***#define INF 9999***

***#define NODENUM 512***

***#define LOCALNUM 256***

***#define INITIALNUM 336***

***#define MULTIPLE 4***

***volatile int Dist[NODENUM];***

***volatile int DistTmp[NODENUM];***

***volatile int x[NODENUM][NODENUM];***

(2). The main code is :

***for(k=0; k<NODENUM; k++){***

***if (my\_id == 3){***

***printf("\*\*\*\*\*\*\*\*\*\*This is the %dth loop\*\*\*\*\*\*\*\*\*\*\n",k);***

***fflush(stdout);***

***}***

***for(i=0;i<MULTIPLE;i++){***

***CulculateMin(MULTIPLE\*my\_id+i, Dist, x);//culculate the minimize***

***DistTmp[MULTIPLE\*my\_id+i] = \*(localAddress1+MULTIPLE\*my\_id+i);***

***}***

***MTA\_Bar\_Stats(1);***

***barrier(MTA\_getthreadID() , NPROC);***

***MTA\_Bar\_Stats(0);***

***if(my\_id == 0){***

***CopyArray(Dist,DistTmp);***

***printf("Data from cluster0 send to cluster1.\n"); MTA\_mailboxsend(0,localAddress2,1,otherAddress0,LOCALNUM,1);***

***printf("Cluster0 sending is over!\n");***

***Wait();//wait***

***ReceiveData(Dist);//receive the other Dist[]***

***printf("I received the data.\n");***

***fflush(stdout);***

***}***

***MTA\_Bar\_Stats(1);***

***barrier(MTA\_getthreadID() , NPROC);***

***MTA\_Bar\_Stats(0);***

***}***

(3). The kernel code is:

***void CulculateMin(int node, int dist[NODENUM], int c[NODENUM][NODENUM]){***

***int \*localAddress0 = (int \*)cmp0\_2;***

***int tmp[LOCALNUM];***

***int i;***

***for(i=0;i<LOCALNUM;i++)//initial the tmp[]***

***tmp[i] = INF;***

***for(i=0; i<NODENUM; i++){***

***if(c[node][i]+dist[i] < tmp[node]){***

***tmp[node]= c[node][i] + dist[i];***

***}***

***}***

***\*(localAddress0+node) = tmp[node];***

***}***

**6.2 Run the benchmark**

The shell script is as follow. We must to specify some path of file.

***rm ../trace/flow\_SSSP\_0.txt***

***rm ../trace/cpu\_SSSP\_0.txt***

***rm ../trace/L2\_SSSP\_0.txt***

***rm ../trace/router\_SSSP\_0.txt***

***~/huarzail/CMP\_POPNET/CMP/PMS\_trace/cmp0/sim-outorder \***

***-redir:sim ./result/test\_SSSP.txt \***

***-redir:dump ./result/test\_SSSP.txt.out \***

***-read\_file ../../../pt\_net\_0.txt \***

***-write\_file ../../../pt\_net\_in.txt \***

***-write\_file\_backup ../../../pt\_net\_in\_backup.txt \***

***-flow\_trace ../trace/flow\_SSSP\_0.txt \***

***-router\_trace ../trace/router\_SSSP\_0.txt \***

***-cpu:trace ../trace/cpu\_SSSP\_0.txt \***

***-L2:trace ../trace/L2\_SSSP\_0.txt \***

***-config ./config\_mesh\_ooo\_xy\_org \***

***-max:barrier 0 -max:inst 100000000000 hello.BNC***

Set correct file paths, just execute the shell script to run the simulator.

***$sh SSSP.sh***

## 7 Multimachine PMS

**7.1 Introduction**

More PMSs can run in different machine simultaneously via the internet to communicate with each other. The overview is as follows.

Fig.8. Overview of the communication flow

**7.2 Implement communication between the two host**

As is shown in the Fig.8, the files in Local directory in host0 is transmitted to Outer directory in host1. That is to say, the files in the two directory is synchronized constantly. Now we use the ftp protocol of Linux kernel to realize it. The code of lftp is as follows.

***#define MAX\_BUFFER\_SIZE 2048***

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

***int fd, wd;***

***int length;***

***char buffer[MAX\_BUFFER\_SIZE];***

***fd = inotify\_init() ;***

***if (-1 == fd) {***

***perror("inotify\_init failed");***

***return -1;***

***}***

***wd = inotify\_add\_watch(fd, argv[1], IN\_CREATE | IN\_MODIFY | IN\_DELETE);***

***if (-1 == wd) {***

***perror("inotify\_add\_watch failed");***

***return -1;***

***}***

***while ((length = read(fd, buffer, MAX\_BUFFER\_SIZE)) > 0) {***

***struct inotify\_event \*event = (struct inotify\_event \*)buffer;***

***for (; length > 0; ++event, length -= sizeof(struct inotify\_event)) {***

***if (0 == event->len) {***

***continue;***

***}***

***if ((event->mask & IN\_CREATE) || (event->mask & IN\_MODIFY)) {***

***std::cout << "file created or modified:" << event->name << std::endl;***

***// now we put it to the ftp server***

***std::string ftpCmd = "lftp -c \"open -u w,111111 192.4.244.130; put ";***

***ftpCmd = ftpCmd + argv[1] + "/" + event->name + " -o /home/w/huarzail/PMS\_network/outer/\"";***

***system(ftpCmd.c\_str());***

***}***

***}***

***}***

***close(fd);***

***return 0;***

***}***

The lftp format is : lftp -c \"open -u w,111111 192.4.244.130; put”+ argv[1] / + event->name + " -o /home/w/huarzail/PMS\_network/outer/\"";

***./lftp /home/w/huarzail/PMS\_network/local***

**7.3 Benchmark**

The code of benchmark is same as the benchmark in PMS. We only to place the files to right location(directory ). The shell script is as follows.

***#!/bin/sh***

***./clear.sh***

***cd local***

***./clear.sh***

***cd ..***

***cd outer***

***./clear.sh***

***cd ..***

***#echo cluster1***

***gnome-terminal -x bash -c "cd program/allShortPath/cluster0/; sh ./allShortPath\_trace.sh;exec bash;"***

***./200 /home/w/huarzail/PMS\_network/local***

***exit 0***

**7.4 Running**

Before run the benchmark, you must to be sure that the FTP service is started.

(1) Check whether this machine is installed the FTP.

[w@localhost ~]$**which vsftpd**

/usr/sbin/vsftpd

[w@localhost ~]$

(2) Check the status of FTP server.

[w@localhost ~]$**service vsftpd status**

vsftpd (pid 28778) is running...

[w@localhost ~]$

(3) Running FTP server if the FTP is not running.

[root@localhost **/etc/init.d]#/etc/init.d/vsftpd restart**

Shutting down vsftpd: [ OK ]

Starting vsftpd for vsftpd: [ OK ]

[root@localhost /etc/init.d]#

Note: This command is must use the root account and the content is must to be **/etc/init.d**

(4) Examine whether the FTP is running.

[w@localhost ~]$**netstat -an | grep 21**

**tcp 0 0 0.0.0.0:21 0.0.0.0:\* LISTEN**

[w@localhost ~]$