

## **Multi-Core Processor Based TCP/IP Client and Server Module Using OpenMP**

**Ch. Venkata siva prasad<sup>1</sup>, Dr. S.Ravi<sup>2</sup> and V. Karthikeyan<sup>3</sup>**

*<sup>1</sup>Research Scholar, ECE Department,  
Dr.M.G.R.Educational and Research Institute, Chennai  
Email:siva6677@gmail.com*

*<sup>2</sup>Professor & Head, ECE Department,  
Dr.M.G.R.Educational and Research Institute, Chennai  
Email:ravi\_mls@yahoo.com*

*<sup>3</sup>Asst.Professor, ECE Department, Dr.M.G.R.Educational and Research Institute  
Email:keyansethu@gmail.com*

### **Abstract**

Data streaming, processing and interaction across the different OS, display functions, etc. forms a key link to realize contention less data flow. In this context, timeliness, accuracy and efficiency of communication among different cores (latency effects) directly affect the performance of a system. Multi-threaded realization process both at the protocol level (TCP/IP) and at the operating system (OS) level (socket creation and binding and its semaphore based sharing) between client/server is discussed in this paper. The paper discusses how to manage data received between client and server in both asynchronous and synchronous mode. The studied metrics include, core affinity with respect to different process, collective communication performance and scalability under multi-core environment. The work is tested on a dual-core AMD Gizmo Processor Multi-core board (running in Linux OS) with the openMP programming model.

**Keywords:** TCP/IP, Multi-Core, OpenMP, Multi-thread, Client, Server

### **Introduction**

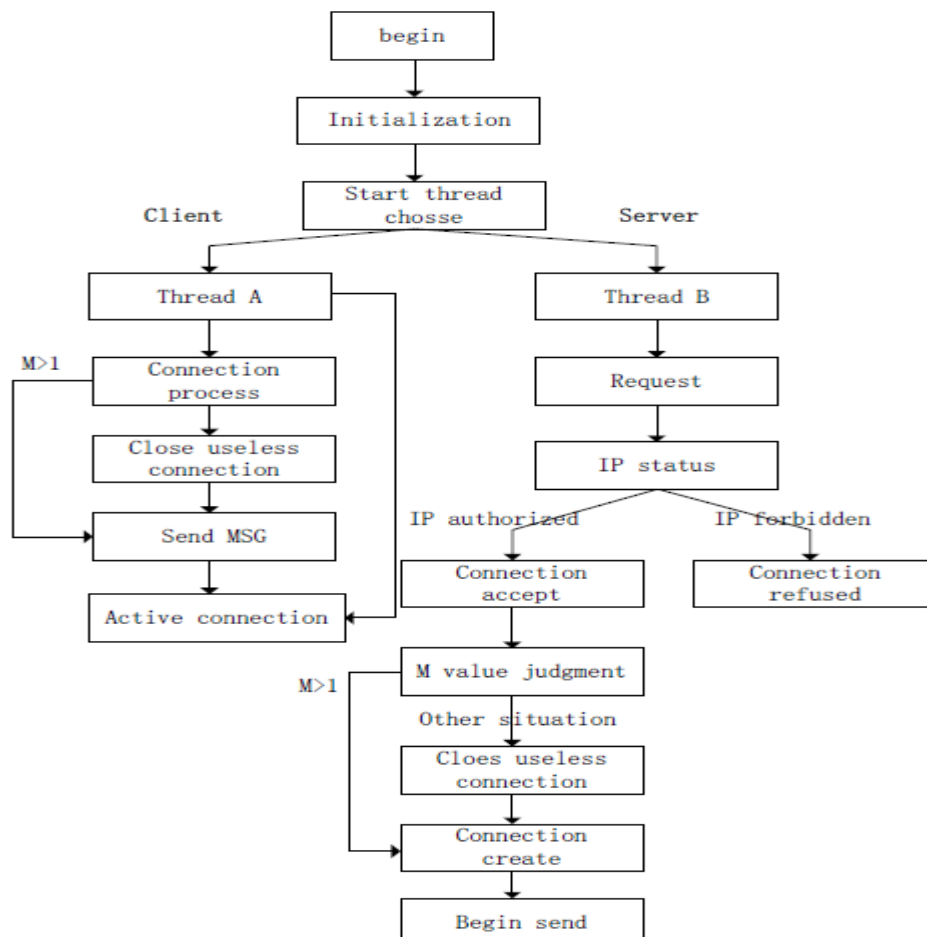
#### **TCP/IP (Transmission Control Protocol /Internet Protocol)**

The TCP is a core protocol of the Internet Protocol Suit.TCP provides reliable, ordered, and error-checked delivery of a stream of octets between applications running on hosts communicating over an IP network. The Two sets of TCP/IP nodes

exist: TCPIP Server nodes and TCPIP Client nodes. Both these nodes perform identical function in terms of accessing the data streams, but, one node uses client connections and the other node uses server connection sets. As a result, the nodes establish the connections in different ways but they use the streams in the same way when the connections have been established. All TCPIP Server nodes that use the same port must be in the same execution group because the port is tied to the running process. TCPIP Client nodes on the same port can be used in different execution groups, but client connections cannot be shared because the client connections are tied to a particular execution group. Within the two sets of nodes (TCPIP Client and TCPIP Server), are three types of node.

- TCPIP Server Input and TCPIP Client Input
- TCPIP Server Receive and TCPIP Client Receive
- TCPIP Server Output and TCPIP Client Output

The Connection process activity program between client and server is shown on the Figure 1.



**Figure 1:** Interaction mechanism in client and server module

### TCP Client/Server Interaction

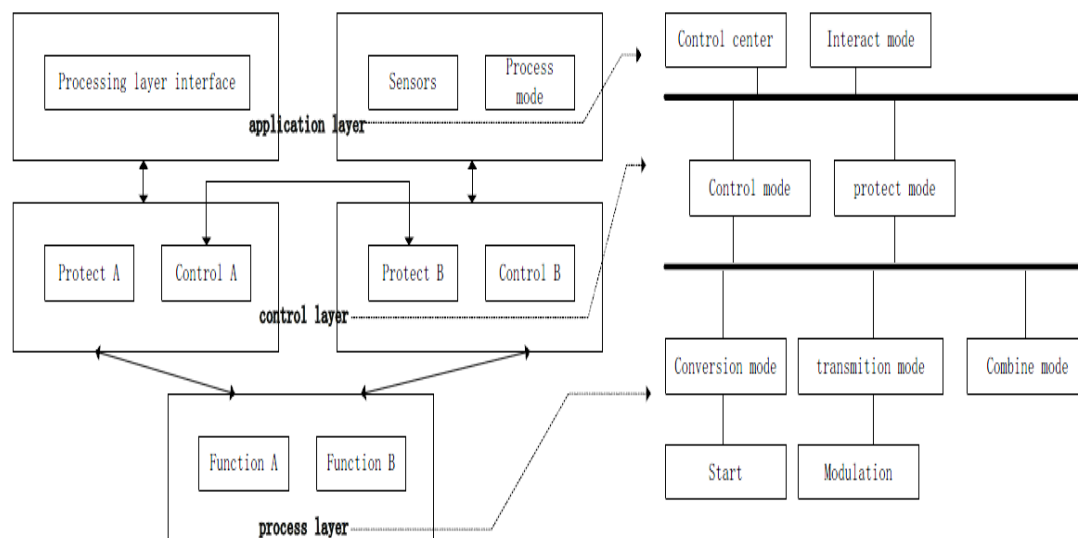
Server starts by getting ready to receive client connections. This is illustrated in Table 1.

**Table 1:** TCP Client/server interaction

Server	Client
1. Establish a listening socket and wait for connections from clients.	
	2. Create a client socket and attempt to connect to server.
3. Accept the client's connection attempt.	
4. Send and receive data.	4. Send and receive data.
5. Close the connection.	5. Close the connection.

### TCP/IP communication standard and bus mode design

In realtime applications, certain functions are divided into control layer, application layer and process layer according to the standard. Separate communication systems for controlling and measuring data will be merged together, and the standard is considered based on process bus and the application mode as shown in Figure 2.



**Figure 2:** Communication System Interface and Communication Mode

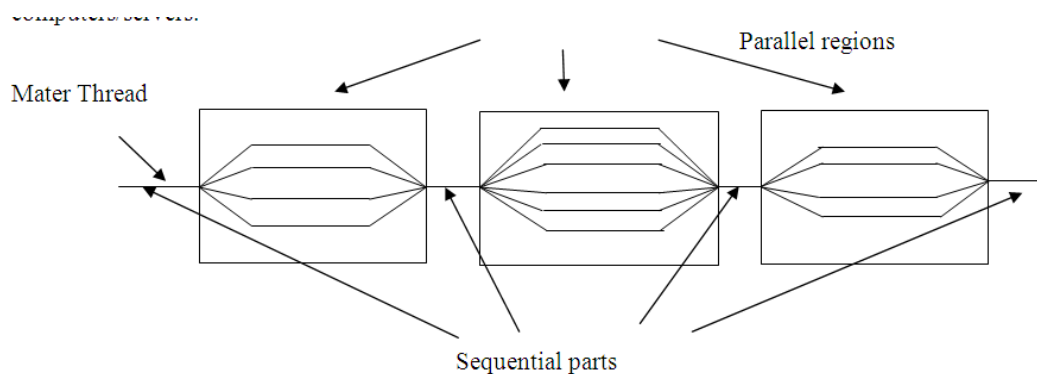
### OpenMP- Multi thread Programming

Open Multi-Processing is collection of API that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran, on most processor architectures and operating systems in present years. The OpenMP uses a portable,

scalable model that gives programmers a simple and flexible interface for developing parallel applications. However, if the software does not take advantage of these multiple cores, it may not run any faster. That is where OpenMP plays a key role by providing an easy method for threading applications without burdening the programmer with the complications of creating, synchronizing, load balancing, and destroying threads. These OpenMP consists of:

- (i) Set of compiler directives
- (ii) Library functions
- (iii) Environment variables

In multithreading programming the Open Multi-Processing (OpenMP) is an implementation of multithreading, a method of parallelization whereby the master "thread" (a series of instructions executed consecutively) "forks" a specified number of slave "threads" and a task is divided among them. The threads then run concurrently, with the runtime environment allocating threads to different processors. The core elements of OpenMP are the constructs for thread creation, workload distribution (work sharing), data-environment management, thread synchronization, user-level runtime routines and environment variables. Figure .3 illustrates the multithread programming concept. In a shared memory multiprocessor, threads in OpenMP are eventually bound to physical processors for efficient parallel execution. Thus, OpenMP parallel loop directives can be used to execute data receiving in parallel for different remote computers/servers.



**Figure 3: Illustration of Multithreading**

### Implementation& Performance analysis

The parameters for the performance analysis of testing model includes

- (i) The client/server of the stream buffers and
- (ii) The number of threads

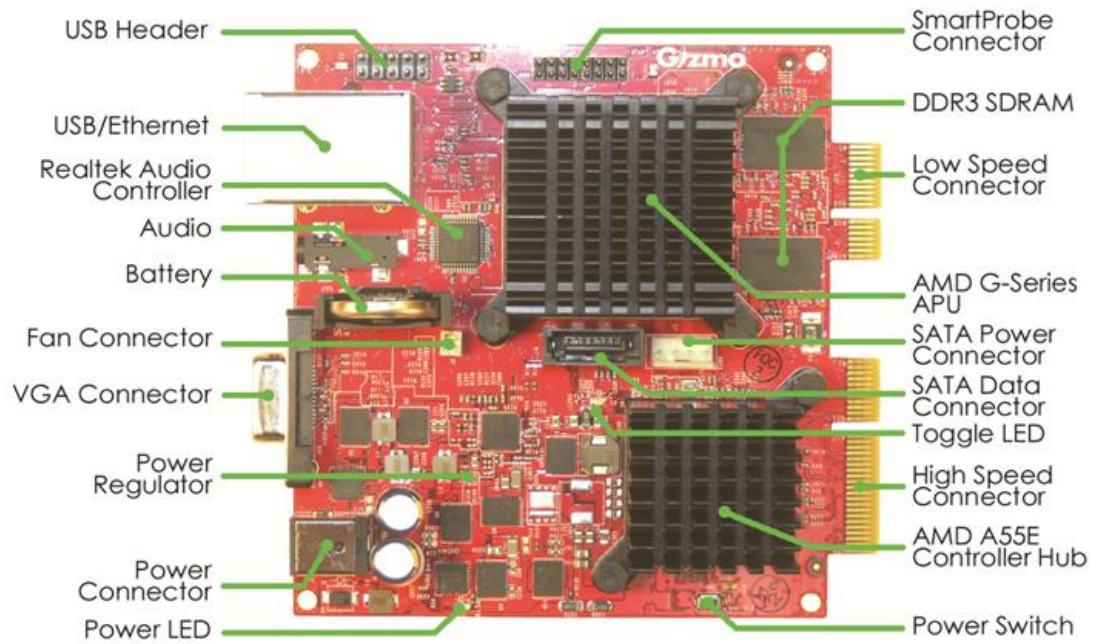
The hardware details are listed in Table 2

**Table 2:** Hardware Details

<i>Hardware Requirements</i>	<i>Software Requirements</i>
Processors: AMD Gizmo Multi-core Soc&Multiple operating system 52.8 GFLOPS at less than 10W • 64-bit processing	Operating System: Linux,windows Software: openMP 4.0 openMP GCC compiler

With the advent of the APU, the silicon-level integration of general purpose, programmable processor cores for high speed parallel processing establishes a new level of AMD gizmo processor of low-power x86 CPU as shown in Figure 4 with the parallel processing performance of a discrete-level general-purpose graphics processing unit (GPGPU) in a single device drives the high speed processing required to handle the intensive number crunching that characterizes high performance embedded systems.

The AMD Embedded G-Series APU at the heart of each Gizmo development board combines a low-power CPU and a discrete-level GPU on a single die with a high-speed bus architecture. Combining a GPU core on the same die as the CPU enables the system to offload computation-intensive data processing from the CPU to the GPU. Freed from this task, the CPU can serve I/O requests with much lower latency, significantly improving the real-time performance of the whole system. With AMD G-Series APUs, the general purpose processor engines within the embedded GPU – up to 80 shader cores running at 280MHz (AMD G-Series T40E APU) – far outstrip the computational capabilities of traditional multi-core CPU-GPU hybrid compute models. The Explorer Board: A companion board for Gizmo, the Explorer expansion I/O board allows for even greater experimentation and exploration opportunities. This two-layer board connects to Gizmo via the low-speed connector and provides an alpha-numeric keypad, a micro-display, and a sea of holes for prototyping and customization.



**Figure 4:** GIZMO Board

### Client

When the client starts running it creates a socket through which it connects to server and using this the actual communication happens. This is done by the function

```
socket( ), sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

This function creates socket and give descriptor for the created socket *sockfd*. After creating socket client adds up its address attributes and sends a connection request to server. This is done by function

```
connect( ), connect(sockfd, (structsockaddr *)&server_addr, SIZE);
```

Once the connection is established client can send and receive data through the created socket. This is done by function

```
write( ) and read( ).
```

### Algorithm for client

The algorithm for a master-slave mode of data communication between the client(s) concurrently among two cores is illustrated below:

- Start
- Define relevant macros (to optimize the code during compilation)
- Enter the ip address

- Read the ip address
- Create the TCP/IP socket
- Configure the settings
- Set the port number
- Call the default number of threads (2) here parallel region starts
- Initialize some private copy
- Divide the program into two sections
- One thread take care of the sending data and 2<sup>nd</sup> section (receiving data) is executed by other thread simultaneously
- Execute 1 and continue with thread 2 simultaneously
- Continue with thread 1
- Enter the data
- Send the data; enter data
- Continue with thread 2
- Continuously monitor receiving port
- If data comes print the data on the screen. Goto Continuously monitor receiving port
- Stop

Control flow graph for the client is shown in Figure 5.

### Server

The server responds to the call from clients (by running in a openMP parallel loop) and dispatch the received files to clients. In case of multiple files, a scheduler co-exists. When server is up and running it creates a socket through which it accepts client connection request. This is done by the function

```
socket( ), sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

This function creates new socket with socket descriptor *sockfd*. Next the server sets up its address attributes and binds to the socket. This is done by function *bind( )*,

```
bind(sockfd, (structsockaddr *)&server_addr, SIZE);
```

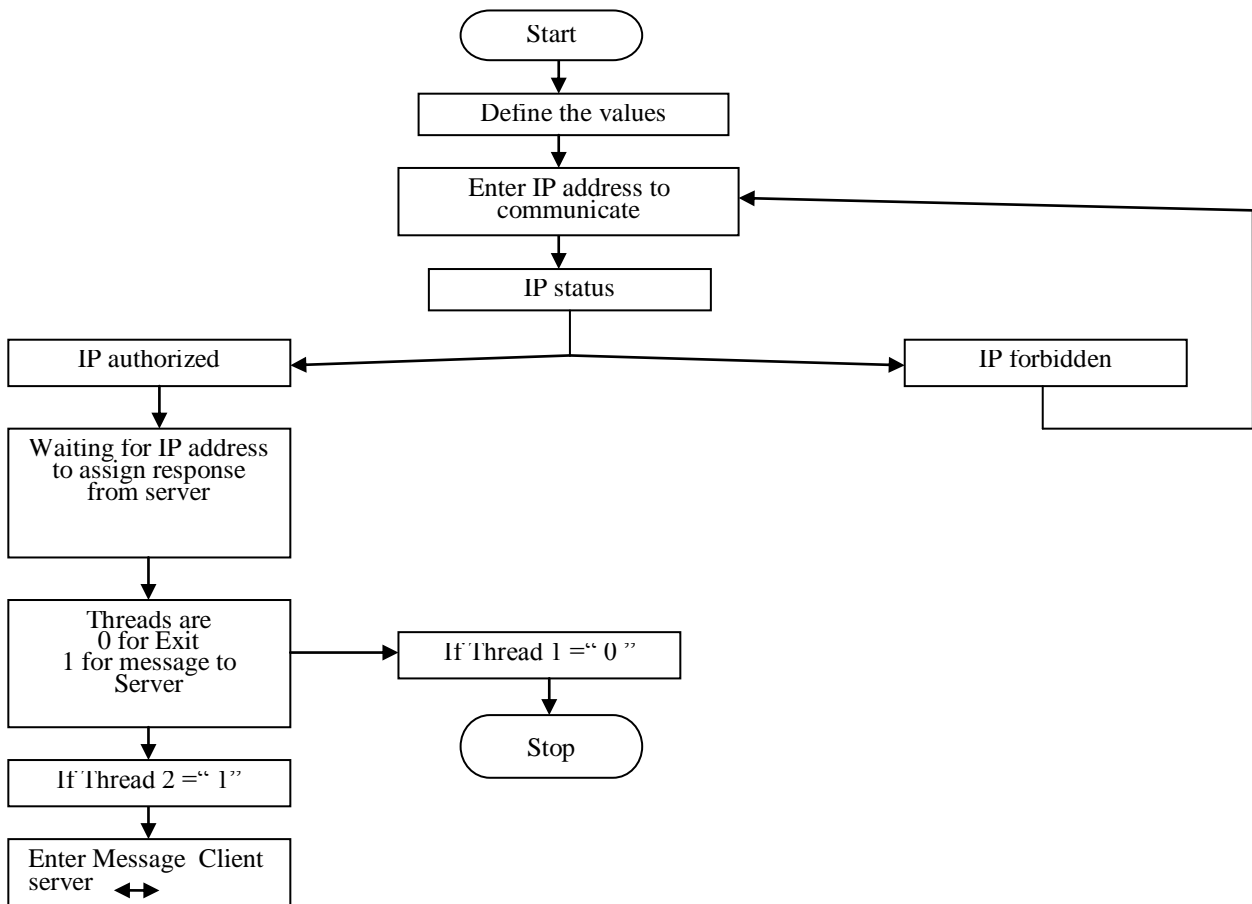
After binding to socket server listen to socket for any connection request from clients. This is done by function

```
listen( ), listen(sockfd, 5);
```

This function listens on the socket *sockfd* for connection requests. If any connection request comes from client, server accepts the request by function

```
accept( ).  
Newsockfd = accept(sockfd, (structsockaddr *)&client_addr, &client_addr_len);
```

This function accepts the client request for connection and provides each client with new unique socket descriptor.



**Figure 5:** Control flow graph for Client

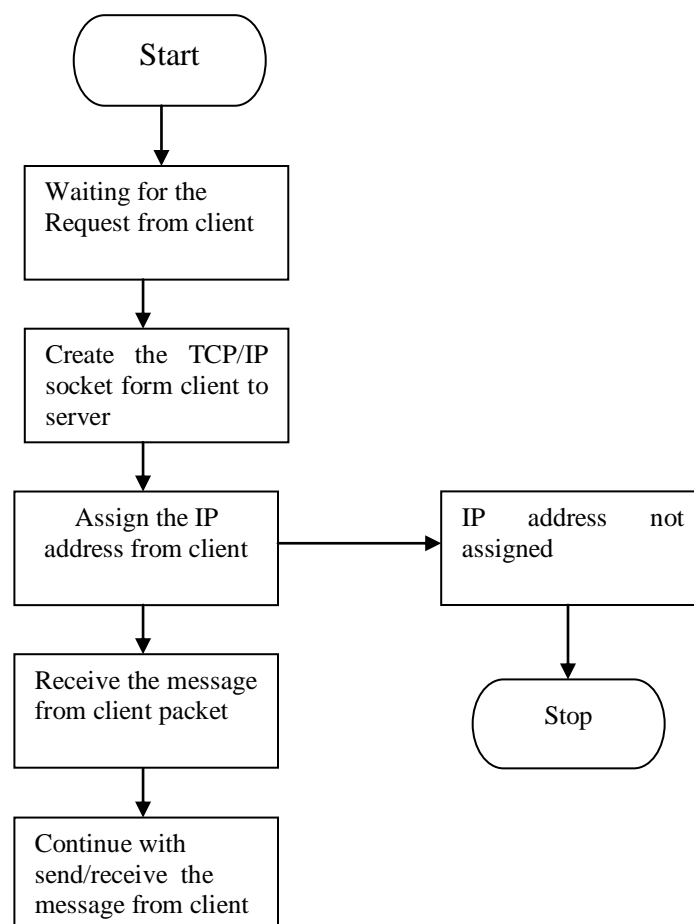
**Algorithm for server**

- Start
- Define some macros
- Initialize some variables
- Enter the ip address
- Read the ip address
- Create the TCP/IP socket
- Configure the settings
- Set the port number
- Bind the address to the socket
- Call the default number of threads (2) here parallel region starts
- Initialize some private copy
- Divide the program into two sections



- One thread take care of the sending data and 2<sup>nd</sup> section (receiving data) is executed by other thread simultaneously
- Execute Thread 1 and Thread 2 steps simultaneously
- Continue with thread 1
- Enter the data
- Send the data; goto enter data
- Continue with thread 2
- Continuously monitor receiving port
- If data comes print the data on the screen. Goto step receiving data
- Stop

The control flow graph for server is shown in Figure 6.



**Figure 6:** Control Flow Graph of a Server

### Configuring Input and Output nodes

The client IP address input node allows access to a connection's input stream. The node is triggered by the arrival of data in the stream and starts processing the message

flow. The TCP/IP nodes are not transactional in the way that they interact with TCP/IP, but other nodes in the same flow can be transactional. The input node does not create a thread for every connection being used, but waits for two requirements to be met: For example, 1,000 TCP/IP connections can be handled by one input node that has only one additional instance. This situation is possible because the node does not poll the connections, but is triggered when the specified conditions are met.

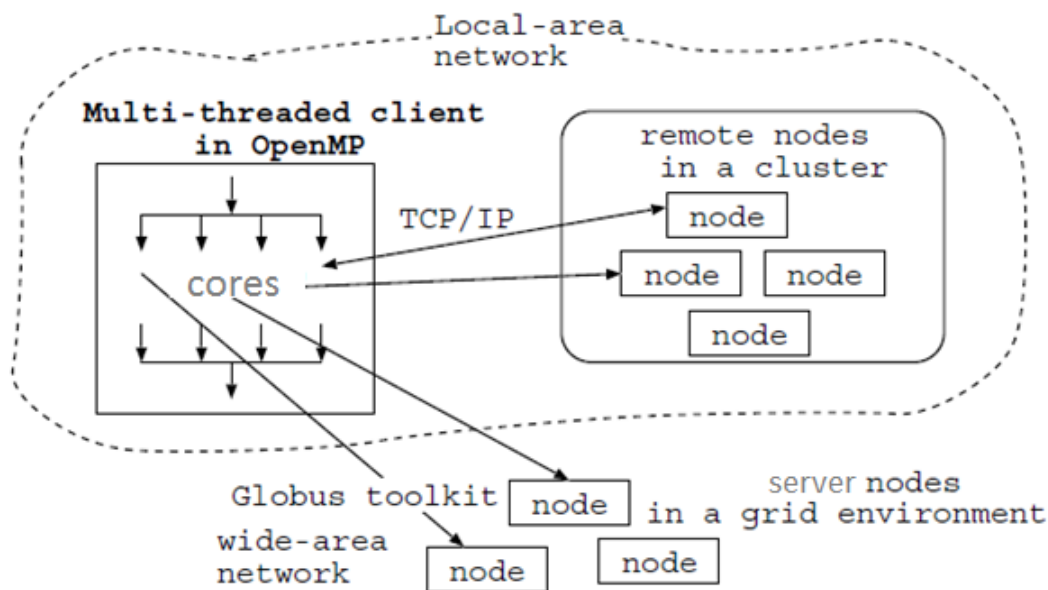
### **Output nodes**

The output node sends data to a server connection. It is triggered by a message arriving on its 'IN' terminal, and subsequently sends the data contained in the message to the stream.

### **Combining nodes**

The client and server nodes can be combined to provide more complex operations for reducing the time delay and latency between the nodes. For example, an output node followed by a receive node enables a synchronous request of data: If the message flows used are single threaded and only one connection ever exists, the sequence of nodes requires no further configuration. Two additional mechanisms are included to enable multithreading and multiple connection

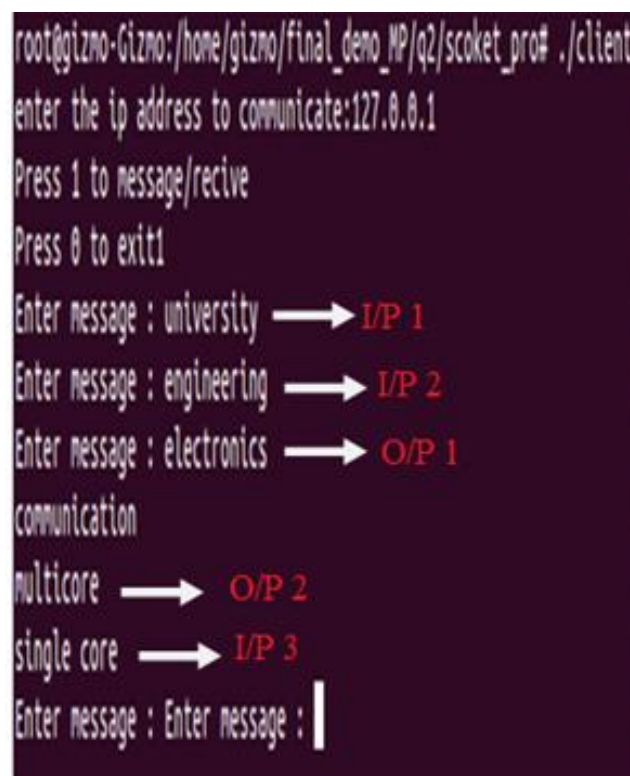
By applying OpenMP commands and threads to that particular program, we can derive a program which can be run in multi-core processors. We have implemented our program in OpenMP by studying several examples in OpenMP like Matrix Multiplication, Vector Multiplication, Hello World, Loop Sharing, Arithmetic Expressions etc. After running the program in multi-core processors, we found the increased performance in parallel execution of getting data from server to client and client to server. TCP/IP control nodes between Client and server is shown in Figure 7.



**Figure 7:** TCP/IP control nodes B/W Client and Server

### Results and Discussion

A new programming (portable and high performance) client and server module for TCP/IP clusters in client and server (both asynchronous and synchronous modes) is presented in both single core and multi core. It uses an OpenMP programming environment on top of a multi-threaded system and the OpenMP abstraction translator bridges the gap between the application specification (client) and the underlying runtime system (server). The code uses openMP directives of the single core is responsible for accepting incoming connections and then produce the new tasks to the server of output files. Multi-core (server) offers explicit support for executing multiple threads in parallel and thus reduces the idle time compare with the single core processor. We also solved socket connection between the two nodes (client & server) using OpenMP to improve performance by reducing execution time in multi core programming while running the threads in parallel. with the TCP/IP nodes sockets creating the communication in client and server that assign the IP address to communicate the client mode for that has server to client has openMP multithread programming has been running across with Linux OS with the new implementation of threads. The experimental results of data transferring and assigning from Client to Server and from Server to Client with assign the received packet from 127.0.0.1: XX openMP tools (Multithreading) of asynchronous (single core) and synchronous modes(multi core) are shown in the Figure 8 and Figure 9 respectively.



```
root@gizmo-Gizmo:/home/gizmo/final_demo_MP/q2/socket_pro# ./client
enter the ip address to communicate:127.0.0.1
Press 1 to message/recive
Press 0 to exit1
Enter message : university → I/P 1
Enter message : engineering → I/P 2
Enter message : electronics → O/P 1
communication
multicore → O/P 2
single core → I/P 3
Enter message : Enter message :
```

**Figure 8:** Multithread socket for control nodes B/W Client and Server

```

root@gizmo-Gizmo:/home/gizmo/final_demo_MP/q2/scoket_pro# ./server
Enter message : Waiting for data...Received packet from 127.0.0.1:48166
Data: universityv → O/P 1
Waiting for data...Received packet from 127.0.0.1:48166
Data: engineering → O/P 2
Waiting for data...electronics communication → I/P 1
Enter message : multicore → I/P 2
Enter message : Received packet from 127.0.0.1:48166
Data: singleering → O/P 3
Waiting for data...Received packet from 127.0.0.1:48166
Data: coreleering
Waiting for data...Received packet from 127.0.0.1:48166
Data: 0oreleering → 0 FOR EXIT
Waiting for data...Received packet from 127.0.0.1:48166
Data: 0oreleering
Waiting for data...Received packet from 127.0.0.1:48166
Data: exitleering → EXIT

```

**Figure 9:** Multithread socket for control nodes B/W Server and Client

### Conclusion and Future work

The work has successfully solved interaction between the both client and server using OpenMP on multi-core threading in single core and multi core processor. In this paper, the socket communication between the client and server with openMP multithreading concept has received/sending messages between the client to server vice versa with assigning the IP address from the TCP socket. The client module has the AMD Multi-core kit has used for execution of the messages between the client to server using a two cores parallel on the multi threading concept. With this module we will develop the large data base servers from getting the input across the OS change socket has developed in Linux threads. The method presented shows improved performance between the client and server with the multi-core processor performance and parallaziation is done. The main limitation of the research work is that its practical implementation requires client to server the number of multi-core units that the number of threads used in openMP compliers. The future enhancement of this work is highly laudable as parallelization using OpenMP is gaining popularity with multi core processor. This work will be carried out in the near future for the real time implementation over a large Scale. The experimental results client and server with two kernels and two real applications on a Linux cluster demonstrate the benefits of the proposed environment, easy and portable programming, and high performance execution, the openMP programming system shows the performance.

**References**

- [1] Leyuan Liu, Weiqiang Kong, and Akira Fukuda , 2014, “Implementation and Experiments of a Distributed SMT Solving Environment”, *International Journal on Computer Science and Engineering (IJCSE)*, Vol.6, No.3
- [2] W. Kong, T. Shiraishi, N. Katahira, M. Watanabe, T. Katayama, and A. Fukuda, 2011, “An smt-based approach to bounded model checking of designs in state transition matrix”, *IEICE Transactions*, vol. 94-D, no. 5, pp. 946–957
- [3] Sebastien Salva, Clement Delamare, and Cedric Bastoul, 2007, “Web Service Call Parallelization Using OpenMP”, *IWOMP 2007*, LNCS 4935, pp. 185–194
- [4] Zhang Xinyang, 2014, “Research of Network Communication Key Technology Based On Multi-core Multi-thread Digital Substation”, *International Conference on Mechatronics, Electronic, Industrial and Control Engineering ( MEIC 2014)*
- [5] Vijayalakshmi Saravanan, Mohan Radhakrishnan , A.S.Basavesh , and D.P. Kothari, 2012, “A Comparative Study on Performance Benefits of Multi-core” *IJCSI International Journal of Computer Science Issues*, Vol. 9, Issue 1, No 2
- [6] Shi Jung Kao, 2003, “Managing C++OpenMP code and its exception handling”, *International conference on openMP shared memory parallel programming*, pp- 227-243
- [7] Walter Goralski, 2009, “The Illustrated Network How TCP/IP works in a modern Network”, Morgan Kaufmann Publishers is an imprint of Elsevier
- [8] Yun(Helen) He, 2007, “ Intoduction to OpenMP”, Cray XE6 workshop
- [9] F. Liu and V. Chaudhary, 2003, “Extending OpenMP for heterogeneous chip multiprocessors Parallel Processing”, *Proceedings of International Conference on Parallel Processing*, pp. 161-168.
- [10] T. Wang, F. Blagojevic and D. S. Nikolopoulos, 2004, “Runtime Support for Integrating Pre-computation and Thread-Level Parallelism on Simultaneous Multithreaded Processors”, *the Seventh Workshop on Languages, Compilers, and Run-time Support for Scalable Systems*, Houston, TX.
- [11] Cristiano Pereira, Harish Patil and Brad Calder, 2008, “Reproducible simulation of multi-threaded workloads for architecture design exploration”, in *Proceedings of the IEEE International Symposium on Workload Characterization*, pp. 173-182.
- [12] Priya Mehta, Sarvesh Singh, Deepika Roy and M. Manju Sharma, 2014, “Comparative Study of Multi-Threading Libraries to Fully Utilize Multi Processor/Multi Core Systems”, *International Journal of Current Engineering and Technology*, Vol. 4, No. 4.
- [13] Sanjay Kumar Sharma and Kusum Gupta, 2012, “Performance Analysis of Parallel Algorithms on Multi-core System using OpenMP”, *International Journal of Computer Science, Engineering and Information Technology*, Vol. 2, No. 5.

- [14] V.Karthikeyan, Dr.S.Ravi and S. Flora Magdalene, 2014, “OpenMP based fast data searching with Multithreading”, International Journal of Applied Engineering Research, Vol.9, No.21,pp.9497-9508
- [15] <http://spcl.inf.ethz.ch/Teaching/2014-dphpc/assignments/openmp-tutorial.pdf>
- [16] <http://www.gizmosphere.org/wp-content/uploads/2013/07/Gizmo-Explorer-Kit-User-Guide-Rev-2.6.pdf>
- [17] <http://www.openmp.org/mp-documents/spec30.pdf>