# Multiple Instruction Multiple Data (MIMD) Implementation on Clusters of Terminals

**Sudhir Kumar Meesala[1], Dr. Pabitra Mohan Khilar[2], Dr. A. K. Shrivastava[3]**

[1]Ph.D. Scholar, Dept. of Computer Science & Engineering, Dr. C.V. Raman University, Kota, Bilaspur(CG), India

[2]Assistant Professor, Department of Computer Science and Engineering, NIT, Rourkela (Orrissa) , India

[3]Professor & Head, Department of Physics, Dr. C. V. Raman University, Kota, Bilaspur(CG), India

**Abstract:** *Today's life style is totally infatuated with computer and technical world and we all are also the part of the crowd . Many scientific and economic fields need a large computer power for their solution, but maximum solution are highly economic effective and expensive. The numeric simulation of complex systems like weather forecast, climate modeling, molecular biology and circuit design are some of such problem. There are two approaches to solve them. Either an expensive parallel supercomputer has to be used [First], or the computer power of workstations in a net can be bundle to computer the task distributed [Second]. The second approach has the advantage that we use the available hardware cost-effective. This paper describes the architecture of a heterogeneous, concurrent, and distributed system, which can be used for solving large computational problems. Here we present the basic solution by Multiple instruction stream and multiple data stream(MIMD) architecture for solving large complex problem. We present a concurrent tasks distributed application for solving complex computational tasks in parallel. The design process is parallel processing implementation on clusters of terminals using Java RMI.*

**Keywords:** Cluster of Terminals (COT), heterogeneous, concurrent, distributed system, weather forecast, climate modeling, molecular biology and circuit design etc.

## 1. Introduction

In computing, **MIMD** (multiple instruction, multiple data) is a technique employed to achieve parallelism. Machines using MIMD have a number of processors that function asynchronously and independently. At any time, different processors may be executing different instructions on different pieces of data. MIMD architectures may be used in a number of application areas such as computer-aided design/computer-aided manufacturing, simulation, modeling, and as communication switches. MIMD machines can be of either shared memory or distributed memory categories. These classifications are based on how MIMD processors access memory. Shared memory machines may be of the bus-based, extended, or hierarchical type. Distributed memory machines may have hypercube or mesh interconnection schemes.. Vector processing was especially popularized by Cray in the 1970s and 1980s. Vector-processing architectures are now considered separate from SIMD machines, based on the fact that vector machines processed the vectors one word at a time through pipelined processors (though still based on a single instruction), whereas modern SIMD machines process all elements of the vector simultaneously.[1]
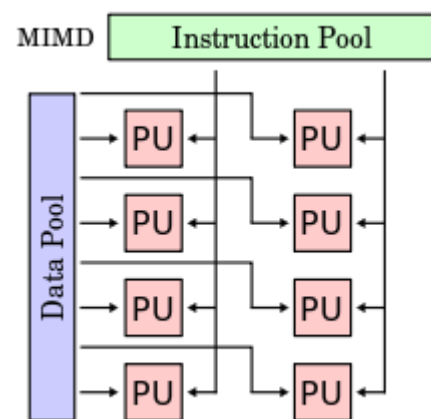


**Figure 1:** Mingle Instruction Stream Multiple Data Stream (MIMD) Architecture

### A. Shared Memory Model
The processors are all connected to a "globally available" memory, via either software or hardware means. The operating system usually maintains its memory coherence.[3]

From a programmer's point of view, this memory model is better understood than the distributed memory model. Another advantage is that memory coherence is managed by the operating system and not the written program. Two known disadvantages are: scalability beyond thirty-two processors is difficult, and the shared memory model is less flexible than the distributed memory model.

There are many examples of shared memory (multiprocessors): UMA (Uniform Memory Access), COMA

(Cache Only Memory Access) and NUMA (Non-Uniform Memory Access).[4]

### 1) Bus-based

MIMD machines with shared memory have processors which share a common, central memory. In the simplest form, all processors are attached to a bus which connects them to memory. This means that every machine with shared memory shares a specific CM, common bus system for all the clients.

For example, if we consider a bus with clients A, B, C connected on one side and P, Q, R connected on the opposite side, any one of the clients will communicate with the other by means of the bus interface between them.

### 2) Hierarchical

MIMD machines with hierarchical shared memory use a hierarchy of buses (as, for example, in a "Fat tree") to give processors access to each other's memory. Processors on different boards may communicate through inter-nodal buses. Buses support communication between boards. With this type of architecture, the machine may support over a thousand processors.
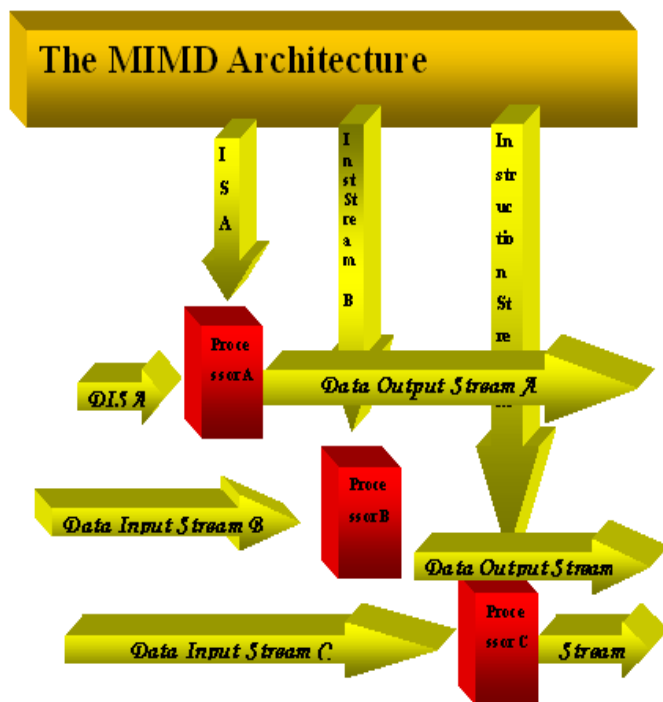


**Figure 2:** Multiple Instruction Stream Multiple Data Stream (MIMD) Architecture

## 2. Parallel Architectures

Distributed computing is method of computer processing in which different parts of a program run simultaneously on two or more computers that are communicating with each other over a network.

Distributed computing is a type of parallel computing.[7] But the latter term is most commonly used to refer to processing in which different parts of a program run simultaneously on two or more processor that are part of the same computer. While both types of processing require that a program be parallelized - divided into sections that can run simultaneously, distributed computing also requires that the division of the program take into account the different environments on which the different sections of the program will be running. For example, two computers are likely to have different file systems and different hardware components.[3]

Distributed computing is a natural result of the use of network to allow computers to efficiently communicate. But distributed computing is distinct from *networking*. The latter refers to two or more computers interacting with each other, but not, typically, sharing the processing of a single program. The World Wide Web is an example of a network, but not an example of distributed computing.[14]

There are numerous technologies and standards used to construct distributed computations, including some which are specially designed and optimize for that purpose, such as Remote Procedure Calls (RPC), Remote Method Invocation (RMI) or Net Remoting.[5]

Organizing the interaction between each computer is of prime importance. In order to be able to use the widest possible range and types of computers, the protocol or communication channel should not contain or use any information that may not be understood by certain machines. Special care must also be taken that messages are indeed delivered correctly and that invalid messages are rejected which would otherwise bring down the system and perhaps the rest of the network.

Various hardware and software architectures are used for distributed computing. At a lower level, it is necessary to interconnect multiple CPUs with some sort of network, regardless of whether that network is printed onto a circuit board of made up of loosely -coupled devices and cables. At a higher level, it is necessary to interconnect processes running on those CPUs with some sort of communication system.

Distributed programming typically falls into one of several basic architecture or categories:
• Client-Server
• 3-tier Architecture
• N-tier architecture
• Distributed objects
• Loose coupling or tight coupling.

[1] Client-Server-- Smart client code the server for data, then formats and displays it to the user. Input at the client is committed back to the server when it represents a permanent change.

[2] 3-tier architecture:- Three tier systems move the client intelligence to a middle tier so that stateless clients can be used. This simplifies application deployment. Most web applications are 3-Tier.

[3] N-Tier architecture:- N-tier refers typically to web application which further forward their request to other Enterprise services. This type of application is the one most responsible for the success of application servers.

Tightly coupled (clustered):- refers typically to a set of highly integrated machines that run the same process in parallel, subdividing the task in part that are made individually by each one, and then put back together to make the final result.

Peer-to-Peer:- architecture where there is no special machine of machines that provide a service or manage the network resources. Instead all responsibilities are uniformly divided among all machines, known as peers.

A multi computer system is a system made up of several independent computers interconnected by a telecommunication network. Multi computer system can be homogeneous or heterogeneous: A homogeneous distributed system is one where all CPUs are similar and are connected by a single type of network. They are often used for parallel computing.[11][12]

A heterogeneous distributed system is made up of different kind of computers, possibly with vastly differing memory sizes, processing power and even basic underlying architecture. They are in widespread use today, with many companies adopting this architecture owing to the speed with which hardware goes obsolete and the cost of upgrading a whole system simultaneous.

The types of distributed systems are based on Flynn's taxonomy of systems: -
[1] Single instruction single data (SISD)
[2] Single instruction multiple data (SIMD)
[3] Multiple instruction single data (MISD)
[4] Multiple instruction multiple data (MIMD)
[5] Single program multiple data (SPMD)

We are implementing Client-Server architecture and single program multiple date (SPMD) taxonomy.

## 3. Remote Method Invocation (RMI)

Remote Method Invocation (RMI) allows a Java object that executes on one machine to invoke a method of a Java object that executes on another machine. This is an important feature, because it allows you to build distributed application. While a complete discussion of RMI is outside the scope of this paper, the following paper describes the basic principles of Java RMI.[22]

The RMI implementation is essentially built from three abstraction layers

### A. The Stub/Skeleton Layer
This layer intercepts method calls made by the client to the interface reference and redirects these calls to a remote object. Stubs are specific to the client side, whereas skeletons are found on the server side. To achieve location transparency, RMI introduces two special kinds of objects known as stubs and skeletons that serve as an interface between an application and rest of the RMI system. This Layer's purpose is to transfer data to the Remote Reference Layer via marshalling and unmarshalling. Marshalling refers to the process of converting the data or object being transferred into a byte stream and unmarshalling is the reverse – converting the stream into an object or data. This conversion is achieved via object serialization.

The Stub/ Skeleton layer of the RMI lies just below the actual application and is based on the proxy design pattern. In the RMI use of the proxy pattern, the stub class plays the role of the proxy for the remote service implementation. The skeleton is a helper class that is generated by RMI to help the object communicate with the stub; it reads the parameters for the method call from the link, makes the call to the remote service implementation object, accepts the return value and then writes the return value back to the stub. In short, the proxy pattern forces method calls to occur through a proxy that acts as a surrogate, delegating all calls to the actual object in a manner transparent to the original caller.

*Stub*
The stub is a client-side object that represents (or acts as a proxy for) the remote object. The stub has the same interface, or list of methods, as the remote object. However when the client calls a stub method, the stub forwards the request via the RMI infrastructure to the remote object (via the skeleton), which actually executes it.

*Sequence of events performed by the stub:*
3 Initiates a connection with the remote VM containing the remote object.
4 Marshals (writes and transmits) the parameters to the remote.
5 VM Waits for the result of the method invocation.
6 Unmarshals (reads) the return value or exception returned.
7 Return the value to the caller.
In the remote VM, each remote object may have a corresponding skeleton[16].

*Skeleton*
On the server side, the skeleton object takes care of all the details of "remoteness" so that the actual remote object does not need to worry about them. In other words we can pretty much code a remote object the same way as if it were local; the skeleton insulates the remote object from the RMI infrastructure.

*Sequence of events performed by the skeleton*
• Unmarshals (reads) the parameters for the remote method (remember that these were marshaled by the stub on the client side)
• Invokes the method on the actual remote object implementation.
• Marshals (writes and transmits) the result (return value or exception) to the caller (which is then unmarshalled by the stub)

The diagram shows the RMI Architecture (Fig 3 & 4)
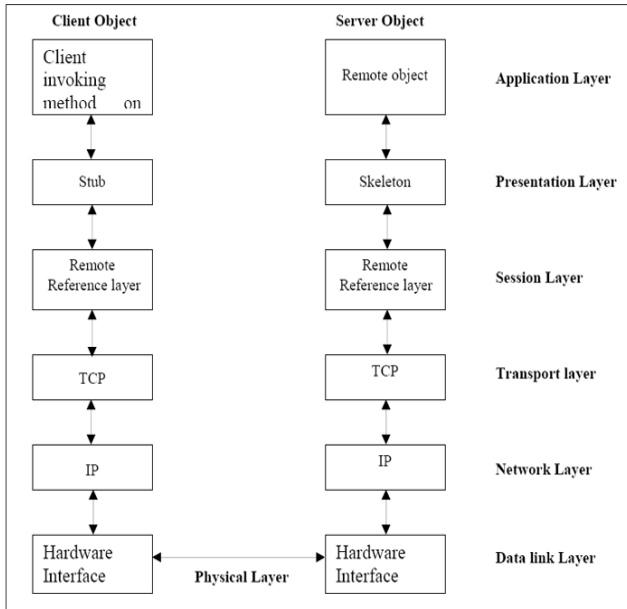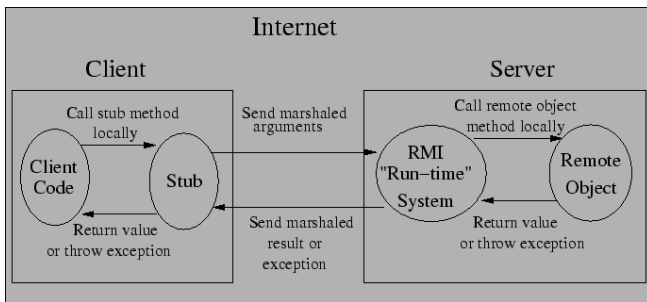
RMI Architecture
**Figure 3:** RMI Architecture



**Figure 4:** RMI Architecture

### B. The Remote Reference Layer
The remote reference layer defines and supports the invocation semantics of the RMI connection. This layer maintains the session during the method call.

### C. The Transport Layer
The Transport layer makes the stream-based network connections over TCP/IP between the JVMs, and responsible for setting and managing those connections. Even if two JVMs are running on the same physical computer, they connect through their host computers TCP/IP network protocol stack. RMI uses a protocol called JRMP (Java Remote Method Protocol) on top of TCP/IP (an analogy is HTTP over TCP/IP).

## 4. Single instruction Multiple Data (MIMD) Based Algorithm

We are implementing Remote Method Invocation from JAVA language as platform to apply parallel processing concept Single Instruction Stream Multiple Data Stream(SIMD) in Distributed Network; here we are using Client /Server architecture. Server is the class where the distribution process occurs. We are having a set of randomly generated numbers. Here as we have single client we retrieve three numbers from client and give them to server for Factorial calculation and Summation. Client has job of

Distribution of numbers. There can be many servers and they can have different methods, which can be applied concurrently, result will be returned to client for further operations.

As many server are present in this application. So, we have to implement Thread to bring access of server to one server at one time. This will not cause corruption of Data and thus the work produce satisfactorily results.

RMI is a simple method used for developing and deploying distributed object application in a java environment. Creating distributed object application using RMI is a simple as writing a stand-alone Java application.

RMI enables a programmer to create distributed Java application, in which the methods of Remote Java object can be called from other Java Virtual Machines running either on the same host or on different hosts scattered across a network.
A call to remote object using RMI is identical to a call made to a local object with the following exceptions:

1. An object passed as a parameter to a remote method or returned from the method must be serialization or be another remote object.
2. An object passed as a parameter to a remote method or returned from the method called is passed by value and not by reference.
3. A client always refers to a remote object through one of the Remote Interface those implements. A Remote object can be typecast to any of the interfaces that a client implements.

When a client application makes a remote call, the call passes to the stub and then on to the Remote Reference Layer, if then passes it via the Network Layer from the client to the server, where the remote reference layer, on the sever side, unpacks the arguments and passes them to the skeleton and then to the server. class file. The return value of the method call then takes the reverse trip back to the client side.

When a client makes a call to a remote method, that client receives a reference to the remote object, which implements the remote method. All interactions by the client are performed with the stub is responsible for data transfer between the local system and the remote system.

The stub object on the client does not interact direct directly with the remote object on the server. There exists a sever side proxy object called the skeleton, which is responsible for transferring data between a stub and the actual object being reference on the server.

In any distributed application, for the client side of the application to make the call to remote object, that client object would first be able to locate the remote object RMI provide the registry services n or the name services to make this possible.

We register any remote object that it is exporting with a name server called a registry. We can maintain a registry

server that is running on a well-known pre defined port number. An application can register with the registry if it is on same physical machine.

Steps For Creating RMI Applications: -
- Define an interface of the remote classes.
- Implement the interface in Server-side application.
- Bind objects to Registry Service.
- Create Stubs and Skeleton classes.
- Create and compile Client program to access the remote objects.
- Install files on client and server machines.
- Start the RMI registry

**B. Steps involved in running the RMI Application:**

Incase the server application and client application is run in the same machine: -
- Run the RMI registry at specified port, if not specified, it runs at the default port 1099.
- Run the server application in another DOS window.
- Run the client application from the same machine.

Incase the server application and client application is run on the separate machine: -
- Run the RMI registry at specified port, if not specified, it runs at the default port 1099.
- Run the server application in another DOS Window.
- Run the client application from a separate machine.

Following these steps RMI application can be implemented.

*Algorithm for Developing and Running the RMI Application for Distributed System.*

*Step 1*: Enter and Compile The Source Code
Enter the Source code for AddserverIntf.java, AddServerImpl.java, AddServer.java, AddClient.java then Compile all above java files.

*Step 2*: Generate Stubs and Skeletons
Compile the Remote Method Invocation (rmic) from AddServerImpl java file. The rmic AddServerImpl generates two new files: AddServerImpl_ Skel.class(Skeleton) and AddServerImpl_Stub.class (stub). When using rmic ,be sure that CLASSPATH is set to include the current directory.

*Step 3*: Install Files On The Client and Server Machines.
Copy AddClient.class, AddServerImpl_Stub, and AddServerIntf. Class to a directory on the Client Machine. Copy AddServerIntf.class, AddServerImpl.class, AddServerImpl_ Skel.class, AddServerImpl_Stub.class and AddServer.class to a directory on the Server Machines.

*Step 4*: Start The RMI Registry on the server Machine.
Start rmiregistry

*Step 5*: Start The Server
Java AddServer

*Step 6*: Start The Client

For Calculating Serially (run at each and individual Machine). The AddClient software requires four arguments: The name or IP address of the servermachine and the three numbers that are to be summed together of first two number and factorial of third number. You may invoke it from the command line by using one of the two formats shown here. (Ex: java AddClient 172.16.16.14 458 475 5 or java AddClient server1 485 475 5)

For Calculating Parallel (run all at same time) The AddClient software requires arguments: Three numbers that are to be summed together of first two number and factorial of third number. In this Process we never use the IP Address because we already use all IP address in AddClient.java. You may invoke it from the command line by using one of the two formats shown here ex: java AddClient 458 475 5).

## 5. Results & Conclusion

We successes in implementing Remote Method Invocation from JAVA language as a platform to apply Single Instruction Stream Multiple Data Stream(SIMD) on clusters of Terminal's (COT's). Here we are using Client/Server architecture. Server is the class where the distribution process occurs.

We are having a set of randomly generated numbers. Here as we have single client we retrieve nth tasks and give them to Nth server for various complex calculations. Client has job of Distribution of numbers. There can be many nodes as a servers and they can have different methods, which can be applied concurrently, result will be returned to client for further operations.

To estimate the performance of the distributed system the time for the computation of the task solved by different servers has to be measured in the sequential and parallel case. The result was examined only within the area 1 to 20 terminal's (see Table1)

**Table 1:** Serial and Parallel Time in Seconds on Number of Terminal

| No of Terminals | Serial Time(Sec.) | Parallel Time(Sec.) |
|---|---|---|
| 2 | 3.965 | 2.382 |
| 4 | 5.968 | 2.437 |
| 6 | 7.452 | 2.492 |
| 8 | 9.656 | 2.579 |
| 10 | 11.482 | 2.592 |
| 12 | 13.745 | 2.719 |
| 14 | 15.56 | 2.827 |
| 16 | 17.75 | 2.843 |
| 18 | 19.76 | 2.906 |

Following Charts are representing the graphically performance of Serial and Parallel distribution and compare between both process (see Fig 3,4,5 and 6)

## 6. Conclusion

An advantage of using parallel processing instead of serial processing is low cost, high efficiency resulting from use of multiprocessing technique. Using parallel processing with

Paper ID: NOV153204

distributed network provides additional advantage of flexibility and speed up in complex calculations.

Using RMI we can further enhance the application by performing file transfer remotely. We can also use the output given by the server to client for further calculations
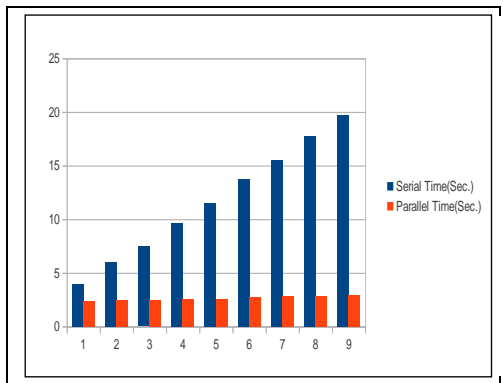
Figure 4: Parallel Time Chart



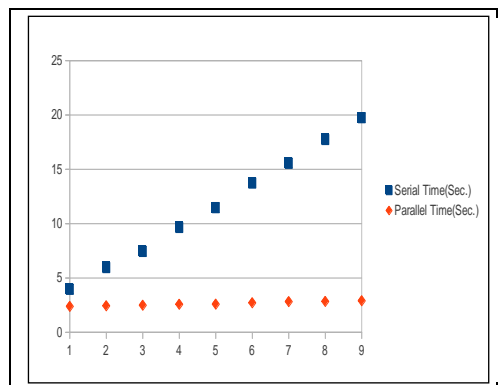**Figure 5:** Comparison Chart of parallel and Serial time using MIMD



**Figure 6:** Comparison Chart of parallel and Serial time using MIMD

## 7. Acknowledgment

My express thanks and gratitude to all the departments' personals and sponsors who give me a opportunity to present and express my paper on this level. I wish to place on my record my deep sense of gratitude to all reference papers authors for them valuable help through their papers, books, websites etc.

## References

[1] http://perilsofparallel.blogspot.gr/2008/09/larrabee-vs-nvidia-mimd-vs-simd.html

[2] Huber, W.; *"Paralleles Rechnen: Eine Einfuhrung"*. Oldenburg Verlag, Munchen, 1997..

[3] Hoffmann, P.; Entwicklung objektorientierter Konzeptezur Erstellung paralleler and verteilter Systemes rechnergestuzten Schaltungsentwurf. Dr. Kovac Varlag, Hamburg, 1997.

[4] Batcher, K. E., "*Sorting Networks and Their Applications*", Proc~ AFIPS 1968 SJCC, vol. 32, Montvale, NJ: AFIPS Press, pp. 307-314.

[5] Dina Bitton , David J. DeWitt , David K. Hsaio , Jaishankar Menon,"*A taxonomy of parallel sorting, ACM Computing Surveys (CSUR)*", v.16 n.3, p.287-318, Sept. 1984 [doi>10.1145/2514.2516]

[6] Dongarr; J. J.; and Eisenstat; "*Squeezing the most out of Algorithms in Cray Fortran*", Argone National Laboratory, May 1983.

[7] Dongarra, J. J.; and Hiromoo, Robert E.; "*A Collection of Parallel Linear Equation Routines for the Denelcor HEP*", Parallel Computing, vol. 1, no. 2, December 1984.

[8] "A resource estimation and call admission algorithm for wirelessmultimedia networks using the shadow … - all 7 versions "*»DA Levine, IF Akyildiz, M Naghshineh* - Networking, IEEE/ACM Transactions on, 1997-ieeexplore.ieee.org.

[9] Spinodal-type dynamics in fractal aggregation of colloidal clusters- all 4 versions *»M Carpineti, M Giglio* - Physical Review Letters, 1992.

[10] UImplementing global memory management in a workstation cluster - all 7 versions *»MJ Feeley, WE Morgan, EP Pighin, AR Karlin, HM …* - ACM SIGOPS Operating Systems Review, 1995 - cs.ubc.ca

[11] Supporting parallel applications on clusters of workstations: The Virtual Communication Machine- …-all 6 versions *»MC Rosu, K Schwan, R Fujimoto* - Cluster Computing, 1998

[12] Supporting parallel applications on clusters of workstations: Theintelligent network interface … - all 3 versions » *M Rosu, K Schwan, R Fujimoto* - High Performance Distributed Computing, 1997. Proceedings. …, 1997 - ieeexplore.ieee.org

[13] [PS] Iterative solution of general sparse linear systems on clusters of workstations - all 16 versions *»GC Lo, Y Saad* - Report umsi-96-117, Minnesota Supercomputer Institute, …, 1996 - cs.umn.edu

[14] JavaParty– transparent remote objects in Java - all 21 versions » *M Philippsen, M Zenger* - Concurrency Practice and Experience, 1997 - doi.wiley.com

[15] Gosling, J.; Joy, B.; Steele, G.; Bracha, G.: "*The Java Language Specification*", Second Edition. Addison-Wesley Publishing Company,1999.

[16] Lindholm, T.; Yellin, F.: "*The Java Virtual Machine Specification*", Second Edition. Addison-Wesley Publishing Company, 1999.

[17] Downing, T.: "*Java RMI: Remote Method Invocation*". IDG Books Worldwide, 1998.

[18] Herbert Schildt : "*Java 2: The Complete Reference*" Fifth Edition 2002 . Tata McGraw-Hill Publishing Company Limited New Delhi.

[19] Lea, D.: "*Concurrent Programming in Java-Design Principles and Patterns*". Addison-Wesley Publishing Company, 1998.

[20] Liang, S.: The Java Native Interface: Programmer's Guide and Specification. Addison-Wesley Publishing Company, 1999.

[21] "A Survey of Parallel Computer Architectures", Duncan, Ralph, IEEE Computer, Feb 1990, pp 5-16

[22] http://www.geekmantra.com/subsection.php?section= MI&subSection = RMI+Architecture.

## Author Profile

**Sudhir Kumar Meesala** has received his Master of Technology degree in Computer Technology from National Institute of Technology, Raipur (CG) the year 2007. At present he is pursuing Ph.D.. with the specialization of Computer Science and Engineering College. His area of interest parallel processing, distributed technology, compiler design, image processing, operating system, network programming and structured computer engineering ec.

**Dr. Pabitra Mohan Khilar** has received his Ph.D. in Computer Science and Engineering from IIT Kharagpur(WB) India in the year 2009. At present he is working as an Associate Professor at Department of Computer Science and Engineering, National Institute of Technology, Rourkela(Orissa). His areas of interests are Parallel and Distributed Computing, Cloud and Grid Computing, distributed wireless network, distributed embedded network etc.

**Dr. A. K. Shrivastava** has received his Ph.D. in Physics(Microwave Propagation) from B. R. Ambedkar Bihar University, Muzafferpur (Bihar) in the year 2003. At present he is working as an Professor and Head at Dr. C. V. Raman University, Kota, Bilaspur (CG) in the department of Physics. His areas of interests are Power electronics, Electrical Drives, Power Systems, Renewable Energy Sources and Custom Power Devices.