ISSN: 2321-7782 (Online)
Volume 1, Issue 4, September 2013
International Journal of Advance Research in Computer Science and Management Studies
Research Paper
Available online at: www.ijarcsms.com

# Advances in Parallel Computing from the Past to the Future

**D. Vijaya Krishna**[1]
Department of Computer Science
JNTUH College of Engineering
Jagtial – India

**Dr. P. Sammulal**[2]
Department of Computer Science
JNTUH College of Engineering
Jagtial - India

***Abstract: Simulation of a computer is a critical technology that plays a major role in many areas in science and engineering. The companies which are working on HPC (High performance computing) are merely shifting their work from the scientific computing market toward the business high-performance computer market where the greatest demand is for cost-effective of midrange performance computers. In this market, the performance improvisation is emphasized by the price-to-performance ratio rather than on the high performance as in scientific applications.***

***Keywords: Parallel computing, Computer architecture, High Performance Computing, SISD, SIMD, MISD, MIMD.***

## I. INTRODUCTION

The speed of serial computers has steadily increased to match the needs of emerging applications over time. With the limitation in speed of light, saturation in serial computer speed is not avoidable and using an ensemble of processors to solve grant challenge problems is a natural way. Moreover, a cost-performance comparison of serial computers over the last few decades shows the saturation of this curve beyond a certain point. With the results of VLSI technology, cost-performance curve of microprocessors became very attractive over serial computers [2]. The idea of connecting microprocessors to form parallel systems is considerably reasonable both in terms of cost-performance and speed.

The fastest solution is not the only advantage of parallel computers. Bigger and more complex problems which require too large amount of data from a single CPU memory is also an important factor. Parallel computing permits simulations to be run at finer resolution and physical phenomena can be modelled more realistically, in theory, the concept of parallel. Computing is not anything else but the applying of multiple CPUs to a single computational problem. In practice parallel computing requires a lot of learning efforts which include parallel computer architectures, parallel models, algorithms and programming, parallel run-time environment, etc.…

Different problems can be paralleled to different degrees. In terms of performance, some problems can be fully paralleled and others cannot. The nature of the problem is the most significant factor to ultimate success or failure in parallel programming.

For more than a decade, lots of efforts have spent for the development of parallel computing in both hardware and software. With the lack of heavy financial supports, research and commercial in HPC areas have slowly decreased in the past few years [2] [3]. The announcement of DOE decision on the ASCI program has made a turning curve of high-performance computing areas. Although ASCI main vision is to advance the DOE defence program computational capabilities to meet future needs of stockpile stewardship, it surely will accelerate the development of high-performance computing far beyond what may be achieved in the absence of this program.

## II. Matter of Contention in Parallel Computing

Parallel computing can only be used effectively if its technical issues are defined clearly and practically understood. Issues need to be concerned with parallel computing science include the design of parallel computers, parallel algorithms and languages, parallel programming tools and environment, performance evaluation, and parallel applications. Parallel computers should be designed so that they can be scaled up to a large number of processors and are capable of supporting fast communication and data sharing among processors [4]. The resource of a parallel computer will not be used effectively unless there are available parallel algorithms, languages, and methods to evaluate the performance. Parallel programming tools and environment are also very much important in helping parallel program development and shielding users from low-level machine characteristics [5]. Parallel applications are the final target of parallel computing, efficiency and portability of a parallel application are the most to date research topics of parallel computing society.

## III. Parallel Computer Architectonics

Basically computers can be classified into three exist categories: Single Instruction Single Data Stream (SISD), Single Instruction Multiple Data (SIMD), and Multiple Instruction Multiple Data (MIMD). The SISD or sequential computer model takes a single sequence of instructions and operate on a single sequence of data. A typical sequential computer evolution of this category is shown in Figure (a). The speed of SISD computer is limited by the execution rate of instructions and the speed at which information is exchanged between the memory and CPU [2]. The latter can be increased with the implementation of memory interleaving and cache memory and the execution rate can be improved by the execution pipelining technique as shown in Figure 2-1 (b). Although these techniques do improve the performance of the computer, limitations are still exist [3] [4]. Cache memory is limited due to hardware technology. Memory interleaving and pipelining are only useful if small set of operations is performed on large arrays of data.

In theory parallel computing are the techniques that cans peed up the performance without any limit. In practice performance of parallel computers is still limited both by the hardware and the software. The SIMD and MIMD computers are both belong to parallel computer category. As shown in Figure (a), a SIMD architecture system has a single control unit dispatches instructions for each processing element in the system [2]. A MIMD architecture system has each control unit in each processing element such that each processor is capable of executing a different program independent of the others in the system.


**Figure (a) Simple sequential computer**

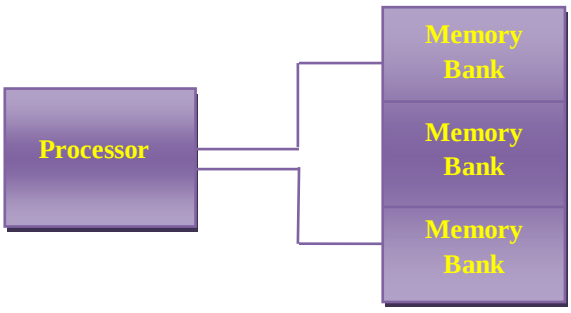



**Figure (b) With memory interleaving**

The SIMD architecture is capable of applying the exact same instruction stream to multiple streams of data simultaneously. For certain classes of problems called data-parallel problems, this architecture is perfectly suited to achieving very high processing rates, as the data can be split into many different independent pieces, and the multiple instruction units can all operate on them at the same time [2]. SIMD architecture is synchronous (lockstep), deterministic, and very well-suited to instruction and operation level parallelism. The architecture can be classified into two distinguish categories called processor array machines and vector pipeline machines.

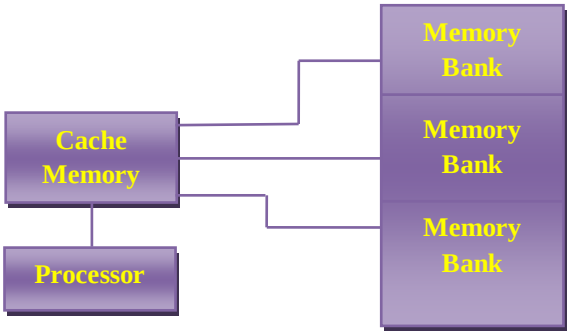



**Figure (c) With memory interleaving and cache**

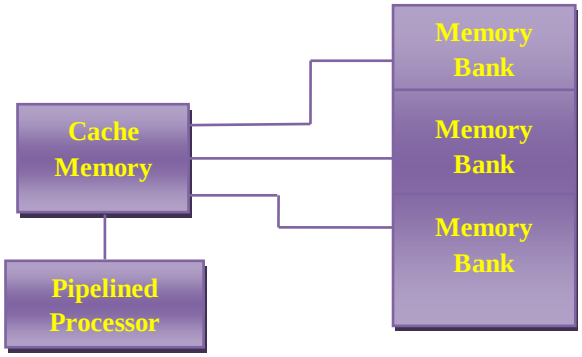
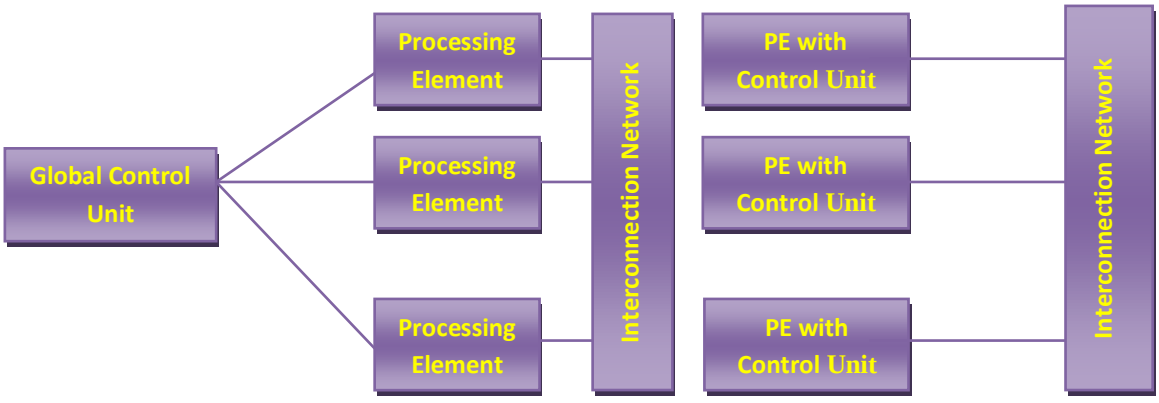



**Figure (d) with memory interleaving, cache and pipe lined processor**

The processor array machines have an instruction dispatcher, a very high-bandwidth internal network, and a very large array of very small-capacity instruction units, each typically capable of operating on only a single bit or nibble (4-bit) data element at a time. Characteristically, many of these are ganged together in order to be able to handle typically encountered data types. The vector pipeline machines have only a fairly small number, typically between 1 and 32, of very powerful execution units called vectors because they are specially designed to be able to effectively handle long strings ("vectors") of floating point numbers [2]. The main CPU handles dispatching jobs and associated data to the vector units, and takes care of coordinating whatever has to be done with the results from each, while the vectors they concentrate on applying the program they have been loaded with their own unique slice of the overall data.

The MIMD architecture is capable of running in true "multiple-instruction" mode, with every processor doing something different, or every processor can be given the same code. The latter case is called SPMD "Single Program Multiple Data" and is a generalization of SIMD-style parallelism with much less strict synchronization requirements. The MIMD architecture is capable of being programmed to operate as synchronous or asynchronous, deterministic or non-deterministic, well-suited to block, loop, or subroutine level parallelism, and multiple Instruction or single program. Some examples of MIMD machines are the IBM SP-series, the clusters of workstations/PCs using PVM/MPI, the IBM ES-9000, the KSR, the nCUBE, the Intel Paragon, and the NEC SX-series. Still another computer model which is based on both SIMD and MIMD architectures (such as the CM-5 and the Fujitsu VPP-series) in order to take advantage of the either when the algorithm benefits most from it. In the Fujitsu VPP-series system, each processing element is a vector pipeline computer and these processors are connected together by an internal communication network.




**PE-Processing Element**

**Figure (e): MIMD architecture**

There are many parallel computer models which are classified based on their control mechanism, address-space organization, interconnection network, and granularity of processors. Depend on the memory structure of the system, MIMD architecture is classified into shared memory and distributed memory architectures. In shared memory architecture, the same memory is accessible to multiple processors and synchronization is achieved by controlling tasks' reading from and writing to the shared memory. The advantage of this type of architecture is the ease of programming but there are many disadvantages to pay off for this. Since multiple processors can access to the same memory, the scalability of the system is limited by the number of access pathways to memory and synchronization has to be controlled manually.

To improve the performance, each processor can be provided with a local memory and the global memory is shared among all the processors in the system. Depend on the speed of memory accessing, shared memory computers can be classified into the Uniform Memory Access (UMA) or Non-Uniform Memory Access (NUMA) computers. Some examples of shared memory parallel computers are the IBM ES-9000, the SGI Power Challenger, and the convex systems.

In a distributed memory parallel computer each processor element has its own local memory and synchronization is achieved by the communication between processors via interconnection network [3] [4]. The major concern in distributed memory architecture is the data decomposition, that is, how to distribute data among processors (local memories) to minimize processor communication. Distributed memory systems can virtually view as "message-passing" and "virtual-shared or distributed-shared memory" models. In message-passing systems, programs communicate explicitly by sending and receiving data packages to and from each other. The process of sending/receiving data is specified and controlled by the application program via message-passing libraries (MPI, PVM, etc.) or languages (VPP-Fortran, Linda, etc.) Some examples of message-passing systems are the IBM SP-series, the Fujitsu VPP-Series, and the hypercube machines nCube. The virtual-shared memory systems provide programmers with a shared-memory programming model but the actual design of the hardware and the network is distributed. In this type of system, a very sophisticated address-mapping scheme insures that the entire distributed memory space can be uniquely represented as a single shared resource. The actual communication model is the lowest level and in fact it is the message-passing but it is not accessible to the programmer.

The other class of MIMD computers that based on both shared- and distributed-memory architectures is the Symmetric Multi-Processor (SMP) clusters. A typical SMP system in fact is a cluster of a large group of nodes as a distributed-memory computer (each node behaves as a processor). Each node in turn has small number of processors (typical 4 or 8 processors) with shared-memory. Programming on SMP clusters is very similar to programming in distributed-memory computers with each processor is a vector processor. Shared as well as distributed-memory systems are conceptualized by connecting processors and memory units using a variety of interconnection networks. In shared-memory systems, interconnection network is for the processors to communicate with the global memory. In distributed systems, interconnection network is for processors to communicate each other. Interconnection networks can be classified as static and dynamic interconnection networks. Static networks (direct networks) consist of point-to-point communication links among processors and are typically used in message-passing computers. In dynamic networks (indirect networks), communication links are connected to one another dynamically with the switching elements to establish communication paths [5] [6]. Dynamic networks are typically used in the shared-memory computers. Some examples of static networks are ring network, mesh network, tree network, and hypercube network. Some examples of dynamic networks include crossbar switching networks, bus-based networks, and multistage interconnection networks.

Parallel computers are also classified based on the granularity, i.e., the ratio of the time required for a basic communication operation for the time required for a basic computation. Parallel computers with small granularity are suitable for algorithms requiring frequent communication and the ones with large granularity are suitable for algorithms that do not require frequent communication [5] [6]. The granularity of a parallel computer is defined by the number of processors and the speed of each

individual processor in the system. Computers with large number of less powerful processors will have small granularity and are called fine-grain computers or Massively Parallel Processors (MPP). In contrast computers with a small number of very powerful processors have large granularity and are called coarse-grain computers or multicomputer.

## IV. Aspects Of Parallel Programming

Creating a parallel program very much depends on finding independent computations that can be executed simultaneously. Currently most of the parallel programmers produce a parallel program in three steps:

1) Developing and debugging a sequential program.

2) Transform a sequential program into a parallel program.

3) Optimizing the parallel program.

This approach is called implicit parallel programming in which mostly data are paralleled. This is because of the sequential nature of the human and also because of the long time training in sequential program developments [5] [6]. The right way for developing an effective parallel program is "everything has to be in parallel initially", that is, the programmers have to explicitly program the code in parallel and there is no sequential-to-parallel step. This requires the development of parallel mathematics, numerical methods, schemes, as well as parallel computational algorithms which are currently under developed. With the existing of too many well recognized computer codes, the term parallel programming currently includes the conversion of sequential programs into the parallel ones. For many years from the first parallel system was constructed, the parallelization of existing codes has been done largely by manual conversion. Many research and development projects are under funding to develop parallel tools for automatic parallelization. Many current compilers attempt to do some automatic parallelization, especially of do-loops, but the bulk of parallelization is still the realm of parallel programming. The parallelization of a serial code can involve several actions such as the removal of inhibitors for parallelization, the insertion of constructs or calls to library routines, and the restructure of serial algorithms to the parallel ones.

There are different aspects of developing a parallel program on a shared-memory and a distributed-memory system. In the shared-memory programming, programmers view their programs as a collection of processes accessing a central pool of shared variables. In the message-passing programming, programmers view their programs as a collection of processes with private local variables and the ability to send and receive data between processes by passing messages [5] [6]. Since most of the shared-memory computers are NUMA architectures, using message-passing programming paradigm on the shared-memory machines in fact improves the performance for many applications.

In some problems many data items are subjected to identical processing and such problems can be parallelized by assigning data elements to various processors, each of which performs identical computations on its data. This type of parallelism called data parallelism is naturally suited to SIMD computers. Data parallelism is not suited to MIMD computers due to the requirement of global synchronization after each instruction. This problem can be solved by the relaxing of synchronous execution of instructions and this leads to another programming model called Single Program Multiple Data (SPMD). In the SPMD programming model, each processor executes the same program asynchronously and synchronization takes place only when processors need to exchange data.

The other type of parallelism called control parallelism in which each processor executes different instruction streams simultaneously. This type of parallelism is suited to MIMD computers but not to SIMD computers since it requires multiple instruction streams, and so programming in this model is called Multiple Program Multiple Data (MPMD) model. Although the amount of data parallelism available in a problem increases with the size of the problem, the amount of control parallelism available in a problem is usually independent of the size of the problem and so control parallelism is limited to the problem size.

## V. Feature Research Topics in Parallel Computing

As in other sciences and engineering fields, the main issues in parallel computing are to build efficient parallel systems, that is, the large or small systems capable of exploiting greater or smaller degrees of parallelism, as desired, without wasting the resources. This is a very difficult task since different parallel architectures are suitable for different applications. Currently parallel computers on the average can only deliver about ten percent of their peak performance when applied to a wide range of applications. In order to build efficient parallel systems, parallel computing industries should first decide to concentrate on one of the two directions:

1) The fix in hardware and software resources.

2) The variable in parallel computer designs.

By fixing the computer hardware and software resources, parallel computing research than facing the problem of how to use the system efficiently for a wide range of applications, and this leads to the mapping problems that must be solved [5] [6]. By designing different parallel systems for different target applications, parallel computing research then faces the problems of computer design, both technically and economically.

Different parallel computer applications require different levels of parallel performance. In terms of hardware requirements, the current industry trends generally satisfy most of the current commercial applications except applications for national defence purposes. In terms of software requirements, the current industry trends generally do not satisfy most of scientific/engineering problems. Parallel software technologies are too much behind the hardware that hardware resources are used wastefully in most of the cases. Some critical future research in parallel computer hardware includes topics related to interconnection networks, interfaces, wide and local networks, and storage peripherals. In terms of the software critical future research include topics related to visualization, computational mathematics and algorithms, mesh generations, domain decomposition, scientific data management, programming models, compilers and libraries, development tools, parallel I/O systems, and distributed operating systems software.

The increase in computational power directly relates to the increase in the number of connected processors and so the interconnection networks. Achieving high performance becomes significantly more difficult with increases in communication time. For the year 2013, performance at PFLOPS levels seem to be the target. These high performance levels require thousands of interconnected CPUs and so intercommunication becomes one of the critical factors to achieve the high performance [5] [6]. The current inter-network and interface technologies can not satisfy the requirements of PFLOPS systems. Moreover, very often the requirements in computational power of an application directly relates to the requirements in the data size, high performance networks for data transfer and large storage capacity are also critical topics needed to be studied. The current technology in these areas only can satisfy applications in the order of TFLOPS.

In terms of visualization, parallel computer designers/developers will soon need advanced techniques to analyze the large data sets in high performance computations. These should include scalable parallel visualization methods; hierarchical methods are used for representation and visualization; data-mining techniques used for feature localization; integration with data archiving, retrieval, and analysis systems; and scalable visualization methods for performance monitoring and for debugging of large parallel applications.

In terms of computational mathematics and algorithms, scalable numerical methods and code frameworks are needed to enable scientific simulations. The current numerical methods, although adequate for most 2-D simulations will not scale to 3-D problems of future interests. Numerical methods topics of interest include preconditioned iterative methods for sparse linear systems of equations, methods for large systems of nonlinear equations, methods for time-dependent differential equations,

parallel adaptive mesh refinement techniques and support libraries, flexible code frameworks for building physics codes, and application-aware communication routines.

The interests in the analysis of problems of complex geometries also require sophisticated, intuitive, and easy to use graphical mesh generation tools. Mesh generation needs to be interactive, with the scientist able to specify some individual points numerically, and others through a set of interpreting mathematical-like expressions. The largest single source of scientist error is running the wrong problem, caused by incorrect specification of the mesh, or incorrect selection of boundary conditions, physics models and parameters [5] [6]. It would be very useful to provide an automated comparison of problem generators to ensure that deviations from accepted settings and parameters are intentional. Expert system software or on-line help to assist in problem generation and parameter selection are desirable. These systems should also provide assistance to the scientist to optimize the domain decomposition of the problem. Scientific data management (SDM) technology is also required to provide the tools and supporting infrastructure to organize, navigate and manage large data sets. An effective SDM framework should seamlessly integrate databases, mass storage systems, networking, visualization, and computing resources in a manner that hides the complexity of the underlying system from the scientist performing data analysis.

The current message passing programming models require application developers to explicitly manage data layout across memory hierarchies from on-chip caches to archival storage. There is a need for portable higher level abstractions that simplify the process of parallel programming by automating data layout and hiding low level architectural details such as memory hierarchies. Applications should be able to adapt quickly and reliably as new computational requirements arise and are saleable and easily portable to successive generations of platforms. High level programming models and abstractions are required that facilitate code reuse, reduce code complexity, and abstract away low level details necessary to achieve performance on a particular architecture.

The current models for message passing are too low level to achieve this objective and promising work is underway in object-oriented frameworks and HPF but much work remains. Four programming models are

Currently of interest are:

1) The message passing (MPI) everywhere (shared and/or distributed memory)

2) The standard language extensions or compiler directives for shared memory parallelism and the message passing for distributed.

3) The explicit threads for shared and the message passing for distributed.

4) The high level language and run-time (object) constructs to present a serial programming model and a global data space over shared and/or distributed memory.

Compilers are also required to generate highly optimized code that effectively utilizes multiple levels of memory. Run-time libraries that provide efficient and scalable methods for numerical analysis, adaptive mesh refinement, load levelling, mesh partitioning and parallel I/O are also needed. Performance measurement, debugging, quality control, verification, and validation of parallel codes become increasingly difficult as code size and complexity increases and so the process of code development for complex parallel architectures needs to be made simpler.

Tools to assist in this process need to have common capabilities and user interfaces across variable platforms. Networking infrastructure and the tool design must make tools usable remotely over a wide area network. Several areas in operating system research also need to be addressed to support the saleable distributed computing environment. To support the degree of parallelism required (e.g. Millions of threads), mechanisms for efficient thread creation, scheduling, and destruction will be

needed. High performance, scalable, portable local and distributed file systems and end-to-end parallel I/O mechanisms are needed.

There are many challenging resource management issues that need to be addressed to integrate the entire system, from the computing node and local disks to multiple nodes and archival storage to the integration of the sharing of data and computer resources in the distributed computing. Research is also needed on both operating and programming system environments that provide services and tools to support the transparent creation, use, and maintenance of distributed applications in a heterogeneous computing environment. In order for the system to distribute resources to users within a parallel distributed computing environment with heterogeneous machine types (e.g. Single processor, SMP, MPP), operating systems will have to control single machine usage and parallel usage spread over many machines. Currently no system exists that provides this coordination while delivering specific allocations to projects. Distributed resource management is needed capable of managing the closely interacting processing and storage resources at all levels.

## VI. Conclusion

Parallelism is to reduce the turnaround time but even increase the CPU time due to overhead and increase the required memory due to duplicate data and instructions. Writing and debugging a parallel program is much more complicated than a sequential program, at least an order-of-magnitude. Different architectures, different programming models are suitable for different applications and so the characteristics of the applications should make the decision for the selection of parallel hardware architecture and also the parallelization of the applications.

Performance at PFLOPS levels will be required for the year 2013 parallel computers. Since most of the current technologies, especially in the software areas are only work for the GFLOPS/TFLOPS systems, many technical issues in high performance computing need to be addressed as discussed in Section III. Besides the semiconductor technology, parallel computer designers and developers also have to aware of the rapid movements in other technologies such as analogy optical, superconductivity, biotechnology, and neural networks. While semiconductor technology will continue to be used, these new technologies will sooner or later be available to parallel processor designers and developers.

## References

1. Parallel Computing: Principles and Practice, T. J. Fountain.
2. Parallel Computing: Technology and Practice: PCAT-94 Jonathan P. Gray, Fazel Naghdy.
3. Parallel Computing: Software Technology, Algorithms, Architectures & Applications: Proceedings of the International Conference ParCo2003, Dresden, Germany, Gerhard Joubert, Wolfgang Nagel, Frans Peters, Wolfgang Walter.
4. Introduction to Parallel Processing, Cornell Theory Center Workshop, 1997.
5. O. Serlin, The Serlin Report On Parallel Processing, No.54, pp. 8-13, November 1991.
6. The Accelerated Strategic Computing Initiative Report, Lawrence Livermore National Laboratory, 1996.

## Author Profile

D. Vijaya Krishna, He is pursuing M.Tech (CSE) in JNTUH College of Engineering, Jagtial, AP, INDIA. He has received B.Tech degree in Computer Science and engineering. His main research interest includes Cluster Computing, Remote Sensing.

**Dr. P Sammulal,** He did his PhD (Computer Science and Engineering) from Osmania University, Hyderabad, India. He has received the B.E degree from Osmania University in 2000 and MTech degree from JNT University in Computer Science and Engg., in 2002. He is in teaching and research since 2002. He has published 30 papers in International conferences/journals. His main research interest includes Parallel, Distributed, Cluster, Grid Computing concepts. He is working as an Assistant Professor in JNTUH College of Engineering (JNTUH University), Jagtial, INDIA.