Performance and Productivity in Parallel Computing: A Case Study from the Past to the Future

Dr. Jangala. Sasi Kiran
Professor in CSE & Dean-Academics,
Farah Institute of Technology, Chevella, R.R. Dt – Telengana, India – 501503
sasikiranjangala@gmail.com

M. **Kavya**
Asst Professor in CSE
Farah Institute of Technology, Chevella, R.R. Dt – Telengana, India – 501503
kavyamunga@gmail.com

Dr.G.Charles Babu
Professor in CSE
Farah Institute of Technology, Chevella, R.R. Dt – Telengana, India – 501503
charlesbabu26@gmail.com

Md. Karimmudin
Professor in CSE
Farah Institute of Technology, Chevella, R.R. Dt – Telengana, India – 501503
mohammad.k1777@gmail.com

Abstract:

Reproduction of a computer is a decisive technology that plays a major role in several areas in science and engineering. The companies which are working on HPC (High Performance Computing) are simply shifting their work from the logical computing market toward the big business high-performance computer market where the furthermost demand is for money-making of midrange performance computers. In developing High Performance Computing software, time to solution is an important metric. This metric is comprised of two main components: The human effort required developing the software, plus the amount of machine time required to execute it. To date, little experiential work has been made to study the first component: the human endeavor required and the effects of approaches and practices that may be used to reduce it. In this paper, we describe a sequence of studies that address this problem and a survey is presented since beginning of parallel computing up to the use of present state-of-art multi-core processors.

Key Terms: *Parallelism, High Performance Computing, Modularity & Multi core processor.*

I. INTRODUCTION

Applying more than on processors to a single computational problem gave rise to parallel and Distributed Computing which opened up the doors for many computing possibilities for researchers and High Performance Computing (HPC) community. In the journey of Parallel and Distributed Computing, several milestones have been achieved and deployed using contemporary technologies like Dedicated Parallel Machines, SMP clusters, Super Computing, Network of Workstations, Commodity Clusters etc. to today's Grid and Cloud Computing. Parallel programming is more difficult than sequential programming because of the additional issues of determinism, synchronization, communication costs, load imbalances and performance portability that must be addressed by the programmer. As a result, productivity of parallel programming efforts tends to be low. Recognizing the importance of high productivity, in the early days of parallel computing researchers aimed at automatic parallelizing compilers. However, after decades of very stimulating research [1, 2, 3, 4, 5], it has become clear that although some of the tools produced can indeed extract almost all the parallelism from the given code, a from-scratch parallel reformulation is often required to attain higher performance.

Moore's Law [6] states that the number of transistors that can be placed on a microchip at a reasonable price will double approximately every two years. Importantly, as the number of transistors grows, the latency of transistor switching has not improved at nearly the same rate. In order to continue to improve the total throughput of computational machines, one solution class is to increase the parallelism of that computation. Intel's founder Andrew Grove thinks this is the inflection point [7] "the time in the life of a business when its fundamentals are about to change". 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 [8, 9]. The announcement of DOE (Department Of Energy) decision on the ASCI program has made a turning curve of high-

performance computing areas. Although ASCI main vision is to advance the DOE (Department Of Energy) defense 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. The purpose of this study was to measure programmer productivity, thus defined, over several years starting in 2002, the beginning of the HPCS (High Performance Computing Systems) initiative. The study was primarily focused on two approaches to parallel programming: the SPMD (single program multiple data) model as exemplified by C/MPI (message-passing interface), and the APGAS (asynchronous partitioned global address space) model supported by new languages such as X10 (http://x10-lang.org), although differences in environment and tooling were also studied.

The Rest of the paper is organized as follows. In Section 2, we studied and analyzed several challenges faced by current and future generation large scale systems in parallel computing. In Section 3, we discuss parallel programming languages and tools for shared and distributed memory. In Section 4, we outline the levels of parallelism and in Section 5, we outline the modularity and the composition in parallel computing. Finally, Section 6 contains our conclusion with the super scope.

II. RELATED WORK

Two main components make up the time to solution metric. The first component is the human effort/calendar time required to develop and tune the software. The second component is the amount of machine time required to execute the software to produce the desired result. Metrics and even predictive models have already been developed for measuring the code performance part of that equation, under various constraints (e.g. [10, 11]). However, little empirical work has been done to date to study the human effort required to implement those solutions. Only a handful of empirical studies have been run to examine factors influencing variables such as development time [12] or the difficulties encountered during HPC development [13]. Some authors have commented that "little work has been done to evaluate HPC systems' usability or to develop criteria for such evaluations"[13]. As a result, many of the practical decisions about development language and approach are currently made based on anecdote, "rules of thumb," or personal preference. Several prior studies[14, 15] have used lines of code as the principal metric to determine effort required to compare parallel programming models with the assumption that fewer lines of code implies less effort.

The challenges faced by current and future-generation large-scale systems include: *1) Frequency wall:* inability to follow past frequency scaling trends due to power and thermal limitations, *2) Memory wall:* inability to support a coherent uniform-memory access model with reasonable performance thereby leading to severe no uniformities in latency and bandwidth for accessing data in different parts of the system, and *3) Scalability wall:* inability to utilize all levels of available parallelism in the system, e.g., clusters, SMPs, multiple cores on a chip, co-processors, SMT, and SIMD levels. It is now common wisdom that the ongoing increase in complexity of large-scale parallel systems to address these challenges has been accompanied by a decrease in software productivity for developing, debugging, and maintaining applications for such machines [16]. This is a serious problem because current trends for next generation systems, including SMP on-a-chip and tightly coupled "blade" servers, indicate that these complexities will be faced not just by programmers for large-scale parallel systems, but also by mainstream application developers.

In the area of scientific computing, the programming languages community responded to these challenges with the design of several programming languages, including Sisal, Fortran 90, High Performance Fortran, Kali, ZPL, UPC,Co-Array Fortran, and Titanium. The ultimate challenge facing this community is *supporting high-productivity, high-performance programming:* that is, designing a programming model that is simple and widely and yet efficiently implementable on current and proposed architectures without requiring "heroic" compilation efforts. During the same period, significant experience has also been gained with the design and use of commercial object oriented languages, such as Java and C#. These languages, along with their accompanying libraries, frameworks and tools, have enjoyed much success in improving *productivity* for commercial applications. While the productivity benefits of commercial object oriented languages are well established for single-threaded applications, the results for concurrent applications have been decidedly mixed. Despite much effort [17], attempts to precisely define the semantics of the memory model for Java that is, the concurrent interactions between multiple threads and a single shared global heap continue to be very complicated technically and arguably beyond the reach of most practicing programmers. With some notable exceptions (e.g.JSR

166 [18]), research on concurrency in such languages has not addressed the issue of delivering the scalable performance that is required by large-scale parallel systems.[19, 20] describe an experiment in which undergraduates were trained in either C/MPI, UPC (Unified Parallel C), or (a very early version of) X10 and then attempted to parallelize the string-matching algorithm of SSCA 1 (Scalable Synthetic Compact Application 1), as described in [21] Average time to completion was approximately 510 minutes in C/MPI, 550 minutes in UPC, and 290 minutes in X10. When the differences in code-execution time during testing were removed, these times were approximately 360 minutes for C/MPI, 390 minutes for UPC, and 180 minutes for X10. Interpretation of this roughly two fold productivity gain was somewhat complicated, however, by the absence of clear success criteria for code completion. In a subsequent study, [22] added the Eclipse programming environment to the tools available to X10 programmers. Participants in this study were more experienced, having had some parallel programming training in earlier course work. A productivity gain for X10 and Eclipse over C/MPI was found here as well, but computing the size of this gain was complicated by the fact that only one of the seven C/MPI participants completed the task (in 407 minutes) vs. five of the seven X10 participants in an average of 321 minutes. X10 is an experimental new language currently under development at IBM in collaboration with academic partners. The X10 effort is part of the IBM PERCS project (Productive Easy-to-use Reliable Computer Systems) whose goalisto design adaptable scalable systems for the 2010 timeframe, with a technical agenda focused on hardware software co-design that combines advances in chip technology, computer architecture, operating systems, compilers, programming environments and programming language design. The main role of X10 is to simplify the programming model so as to increase the programming productivity for future systems like PERCS, without degrading performance. Combined with the PERCS Programming Tools agenda [23], the ultimate goal is to use a new programming model and a newest of tools to deliver a 10× improvement in development productivity for large-scale parallel applications by 2010.To increase programmer productivity, X10 starts with a state-of-the-art object-oriented programming model, and then raises the level of abstraction for constructs that are expected to be amenable to automatic static and dynamic optimizations by 2010 specifically.

III PARALLEL PROGRAMMING LANGUAGES AND TOOLS

Parallel programming languages and tools largely depend on the underlying architecture being used to run parallel application, viz., shared memory architecture and distributed memory architecture.

3.1 Languages and Tools for Shared Memory Architectures:

OpenMP (Multi Processing) is a popular programming option on shared memory systems. OpenMP programming standards consist of compiler directives that define and identify parallel region of the code that can run as threads. Some programs use proprietary compiler directives to form parallelism through threads whereas OpenMP (Multi Processing) provides a higher level of abstraction to programmers and create parallelism in a fork-and-join programming model. In this model program begins sequential execution as a single process or thread. When the directive for parallel region is found, a single thread becomes master thread and creates several other slave threads to execute parallel tasks. At the end of parallel region, all threads are synchronized & joined to produce clubbed results. In OpenMP (Multi Processing) programming paradigm, all threads use shared memory which creates possibility of memory contention among threads. This issue is resolved by implementing memory coherence protocol for data consistency. The other popular programming options for shared memory systems are Portable Operating System Interface (POSIX) Threads and Compute Unified Device Architecture (CUDA).

3.2 Languages and Tools for Distributed Memory Architectures:

The communication among two or more processors in distributed memory systems are carried out through Message Passing. MPI is a popular programming option for distributed memory systems. MPI (stands for Message Passing Interface) is a specification of message passing libraries for the researchers, developers and users. The goal of the Message Passing Interface is to provide portable, efficient and flexible standard for a wide use of writing message passing programs. The first version of MPI (later called MPI-1) came in existence in 1994. The second version MPI-2, included some more programming issues, was released in 1996 [24, 25].

MPICH is a portable implementation of the full MPI-1 specification for a wide variety of parallel and distributed computing environments. MPICH contains, along with the MPI library itself, a programming environment for working with MPI

programs. The programming environment includes a startup mechanism and a profiling library for studying the performance of MPI programs. Interface specifications have been defined for C/C++ and FORTRAN programs. MPICH2 is a portable implementation of the full MPI-2specification. Both portable implementations also include:

- Tracing and log file tools based on the MPI profiling interface, including a scalable log file format (SLOG).
- Parallel performance visualization tools (Jumpshot).
- Extensive correctness and performance tests.

The other popular programming options for distributed memory systems are Unified Parallel C (UPC) and Fortress. Both the programming models, viz., OpenMP and MPI, can be used within same program as hybrid MPI+ OpenMP (Multi Processing) paradigm which is suitable for architectures consisting of both shared and distributed memory such as cluster of multi-core processors [26, 27]. MPI can be used to provide process level parallelism across nodes while OpenMP can be used to implement loop level parallelism within anode by using compiler directives [28].

IV APPROACHES AND LEVELS OF PARALLELISM

A sequential program is one which runs on a single processor and has a single line of control. To make many processors collectively work on a single program, the program must be divided into smaller independent chunks so that each processor can work on separate chunks of the problem. The program decomposed in this way is a parallel program.

A wide variety of parallel programming approaches are available. The most prominent among them supported on PARAM are the following:

- Data Parallelism
- Process Parallelism
- Farmer and Worker Model

All these three models are suitable for task level parallelism. In case of data parallelism, divide and conquer technique is used to split data into multiple sets and each data set are processed on different PEs by using the same instruction. This approach is highly suitable for processing on machines based on SIMD model. In case of process parallelism, a given operation has multiple (but distinct) activities, which can be processed on multiple processors. In case of farmer and worker model, job distribution approach is used; one processor is configured as master and all other remaining PEs are designated as slaves; master assigns job to slave PEs and they on completion informs the master which in turn collects results. The above approaches can be utilized in different levels of parallelism.

4.1 Levels of Parallelism

Levels of parallelism decided based on the lumps of code (grain size) that can be a potential candidate for parallelism. Table 1 lists categories of code granularity for parallelism. Parallelism in an application can be detected at several levels. They are

Large-grain (or task-level)

-
- Medium-grain (or control-level)
- Fine-grain (data-level)
- Very-fine grain (multiple instruction issue)

The different levels of parallelism are depicted in Figure 1.

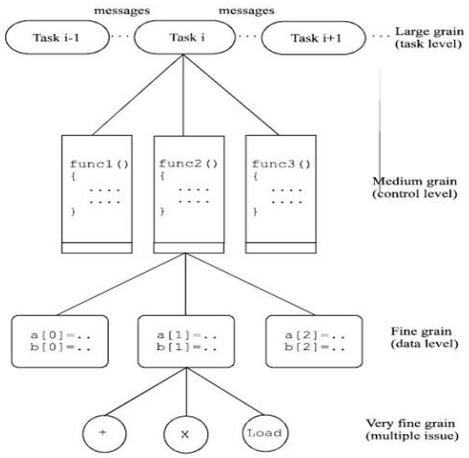

Figure 1: Levels Of Parallelism

Grain Size	Code item	Comments/ parallelized by
Large	Program-Separate Heavy weight process	Programmer
Medium	Standard One Page Function	Programmer
Fine	Loop/Instruction block	Parallelizing Compiler
Very fine	Instruction	Processor

Table 1: Categories of Code Granularity

Among the four levels of parallelism, the PARAM supports medium and large grain parallelism explicitly. However instruction level of parallelism is supported by the processor used in building compute engine of the PARAM. For instance, the compute engine in PARAM 8600 is

based on i860 processor having capability to execute multiple instructions concurrently.

Thread Level Programming

Threads are an emerging model for expressing concurrency on multiprocessor and multicomputer systems. In multiprocessors, threads are primarily used to simultaneously utilize all the available processors, whereas in uniprocessor or multicomputer system, threads are used to utilize system resources effectively by exploiting the asynchronous behavior (opportunity for computation and communication overlap) of threads.

Task Level Programming

PARAM as a MIMD distributed memory machine offers a wide variety of interfaces for task level parallelism. They include CORE (Concurrent Runtime Environment), MPI (Message Passing Interface), PVM (Parallel Virtual Machine). CORE is a custom built interface whereas MPI is the standard interface, which is available on most of the modern parallel supercomputers. The various primitives offered by them include task creation, deletion, control, and communication. They offer both the synchronous and asynchronous mode of communication.

V MODULARITY VIA CONCURRENT COMPOSITION

For high productivity in parallel programming, one should be able to modularize the program. In particular, it should be possible to compose independently developed parallel modules into a single parallel application (or into higher level modules, composed hierarchically). Further, the modules being composed should be allowed to overlap their execution in time, and over processors. Without this flexibility, one risks the danger of fragmenting the set of processors (especially when a large number of modules are being composed) and certainly loses the ability to exploit adaptive overlap of communication and computation across modules. This is illustrated withal schematic and application example below.

Consider the situation in Figure 2. A, B and C are each parallel modules spread across all processors. A must call B and C, but there is no dependence between Band C. In traditional MPI style programming, one must choose one of the modules (say B) to call first, on all the processors. The module may contain sends, receives, and barriers. Only when B returns can A call C on each processor. Thus idle time (which arises for a variety of reasons, including load imbalance and critical paths) in each module cannot be overlapped with useful computation from the other, even though there is no dependence between the 2 modules.

In contrast, with processor virtualization (and the message-driven execution induced by it), A invokes B on each processor, which computes, sends initial messages, and returns to A. A then starts off module C in similar manner. Now B and C interleave their execution based on availability of data (messages) they are waiting for. This automatically overlaps idle time in one module with computation in the other, as shown in the Figure. One can attempt to achieve such overlap in MPI, but at the cost of breaking the modularity between A, B andC. With processor virtualization, the code in B does not have to know about the code in A or C, and vice versa.

This phenomenon is illustrated in NAMD (Figure 2 (b)). The computation partitions atoms into a set of cubic cells called "patches". Interactions between atoms in adjacent cells are computed by separate virtual processors called the "pair wise compute objects" in the Figure. The PME (Particle-Mesh Ewald) module involves two 3D-FFTs (each with a communication intensive transpose operation) over a relatively small grid (192x144x144 in one case). By concurrently composing the PME and force-calculation modules, it becomes possible to use the considerable latency of the transposes in the PME algorithm with pair wise-force computations adaptively. Neither partitioning of processors among the two modules, nor sequencing their execution one after the other will yield the same efficiency of concurrent composition employed by NAMD. Moreover, this efficiency is attained without any coding by the programmer to juggle execution between the two modules.

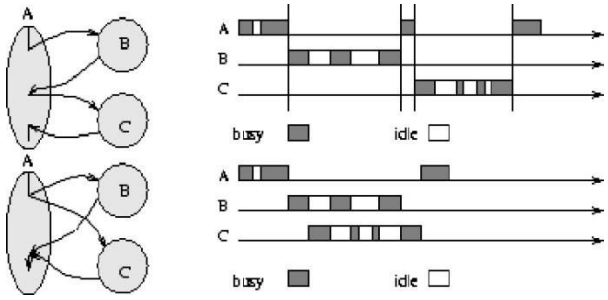

Figure 2 (a): Modularity and Adaptive Overlapping: Schematic

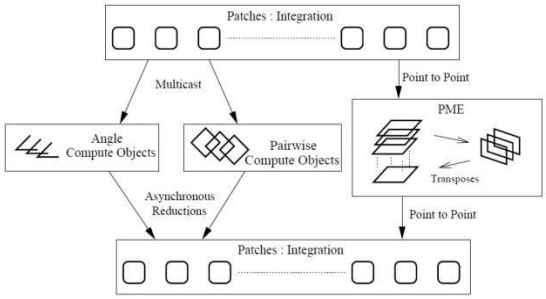

Figure 2 (b): Concurrent Composition of PME and Force Computations in NAMD
Figure 2. Concurrent Composition

VI CONCLUSION AND FUTURE WORK

We presented a research agenda, and our progress along it, which has been explicitly aimed at improving programmer productivity and computer performance on complex parallel applications. Many large computational problems cannot be solved within desired time constraint with sequential computing (using a uniprocessor computer). To achieve speedup in computing, more than one processor (computer) is used to solve a large problem, in form of Parallel and Distributed Computing. The processors of Parallel Systems are generally tightly coupled, i.e., use shared memory, whereas the processors of Distributed Systems are loosely coupled, i.e., use distributed memory and may be scattered in wide geographical area. Distributed systems such as Network of Workstations, SMP Cluster etc. provide cost effective solution to large computational problem then dedicated parallel systems. The multi-core architecture has put an opportunity and challenge to exploit all available cores present in the processor. Cluster of recent multi-core processors is as powerful as earlier day's supercomputers. Various parallel programming languages and tools are available for different computing environments. Due to improved computing performance of parallel applications, the demand of parallel programmers is expected to increase in future. A performance model that distinguishes between computation and communication must be made explicit and transparent. At the same time we believe that the interaction between the concurrency constructs and the place-based type system (including first-class support for type parameters) will enable much of the burden of generating distribution-specific code and coordination of activities to be moved from the programmer to the underlying implementation.

A future orchestration language, which will allow the interactions among components to be defined in a scripting language, will also improve the reuse of parallel components and make the logic of parallel applications more explicit.

VII ACKNOWLEDGEMENTS

I would like to express my cordial thanks to Sri. CA. BashaMohiuddin, Chairman, Smt. Rizwana Begum-Secretary and Sri. Touseef Ahmed-Vice Chairman, Dr.M.Anwarullah, Principal - Farah Group of Institutions, Hyderabad for providing moral support, encouragement and advanced research facilities. Authors would like to thank the anonymous reviewers for their valuable comments. And they would like to thank Dr.V. Vijaya Kumar, Anurag Group of Institutions for his invaluable suggestions and constant encouragement that led to improvise the presentation quality of this paper.

VIII REFERENCES

[1]W. Blume, R. Eigenmann, et all:Improving the effectiveness of parallelizing compilers. In Proceedings of 7th International Workshop on Languages and Compilers for Parallel Computing, number 892 in Lecture Notes in Computer Science, pages 141–154, Ithaca, NY, USA, and August 1994. Springer-Verlag.

[2] D. Padua and M. Wolfe. Advanced Compiler Optimizations for Super computers. Communications of the ACM, 29(12):829-842, Dec 1986.

[3] S. Hiranandani, K. Kennedy, and C. Tseng. Compiler support for machine independent parallel programming in fortran-d. In J. Salz and P. Mehrotra, editors, Compilers and Runtime Software for Scalable Multiprocessors. Elsevier Science Publishers B.V., 1992.

[4] S. Hiranandani, K. Kennedy, and C.-W. Tseng. Compiler optimizations for fortran-d on mimd distributed memory machines. In Proceedings of Supercomputing 1991, Nov. 1991.

[5] R. Cytron, D.J. Kuck, and A. V. Veidenbaum. The effect of restructuring compilers on program performance for high-speed computers. Computer Physics Communications,37(1–3):39–48, 1985.

[6] Moore's Law: Electronics Magazine 19 April 1965

[7] Andrew Grove, Only the Paranoid Survive 1999

[8]Parallel Computing: Technology and Practice: PCAT-94 Jonathan P. Gray, Fazel Naghdy.

[9] Dresden, Germany, Gerhard Joubert, Wolfgang Nagel, Frans Peters, Wolfgang Walter. Parallel Computing: Software Technology, Algorithms,

Architectures & Applications: Proceedings of the International Conference ParCo2003,
[10] A. Hoisie, O. Lubeck, H. Wasserman, F. Petrini, and H.Alme, "A General Predictive Performance Model forWavefront Algorithms on Clusters of SMPs," Proc.ICPP. 2000, pp. 219-229.
[11] A. Snavely, L. Carrington, N. Wolter, J. Labarta, R.Badia, and A. Purkayastha, "A Framework for ApplicationPerformance Modeling and Prediction," Proceedingsof SC2002. Nov. 2002, IEEE.
[12] J. C. Browne, T. Lee, and J. Werth, "ExperimentalEvaluation of a Reusability-Oriented Parallel ProgrammingEnviron-ment," IEEE Transactions on Software Engineering,16(2), 1990, pp. 111-120.
[13] D. Szafron and J. Schaeffer, "An Experiment to Measurethe Usability of Parallel Programming Systems," Concurrency:Practice and Experience, 8(2), 1996, pp. 147-166.
[14] F. Cantonnet, Y. Yao, M. Zahran, and T. El-Ghazawi,"Productivity Analysis of the UPC Language," IPDPS2004 PMEO workshop. April 2004, Santa FE, NM.
[15] B. L. Chamberlain, S. J. Dietz, and L. Snyder, "A comparativestudy of the NAS MG benchmark across parallellanguages and architectures," SC'2000. Nov. 2000.
[16] HPL Workshop on High Productivity ProgrammingModels and Languages May 2004.http://hplws.jpl.nasa.gov/?
[17] W. Pugh. Java Memory Model and ThreadSpecification Revision, 2004. JSR 133,http://www.jcp.org/en/jsr/detail?id=133?
[18] D. Lea. The Concurreny Utilities, 2001 JSR 166,http://www.jcp.org/en/jsr/detail?id=166?
[19] Ebcioglu, K., Sarkar, V., El-Ghazawi, T., Urbanic, J. 2006. An experiment in measuring the productivity of three parallel programming languages. In Proceedings of the Third Workshop on Productivity and Performance in High-End Computing: 30-36.
[20]Danis, C., Halverson, C. 2006. The value derived from the observational component in an integrated methodology for the study of HPC programmer productivity. In Proceedings of the Third Workshop on Productivity and Performance in High-End Computing: 11-21.
[21]Bader, D. A., Madduri, K., Gilbert, J. R., Shah, V., Kepner, J., Meuse, T., Krishnamurthy, A. 2006.Designing scalable synthetic compact applications for benchmarking high productivity computing systems.CTWatch Quarterly 2(4B); http://www.cse.psu.edu/~madduri/papers/SSCA-CTWatch06.pdf.
[22] Halverson, C. A., Swart, C., Brezin, J., Richards, J., Danis, C. 2008. Towards an ecologically valid study of programmer behavior for scientific computing.In Proceedings of theFirst Workshop on Software Engineering for Computational Science and Engineering.
[23] V. Sarkar, C. Williams, and K. Ebcio˘glu. Applicationdevelopment productivity challenges for high-endcomputing. In Proceedings of Workshop onProductivity and Performance in High-End Computing(P-PHEC), February 2004.http://www.research.ibm.com/arl/pphec/pphec2004-proceedings.pdf.
[24] Guo Liang Chen, Yun Quan Zhang, "Survey on Parallel Computing," Journal of Computer Science & Technology, Sept. 2012, Vol.21, No. 5, pp. 665-673.
[25] Intel Multi-core Series Processors, Available online at: http://www.intel.com/ products/processor/core2quad/.
[26] Rajkumar Sharma and PriyeshKanungo, "Performance Evaluation of MPI and Hybrid MPI+OpenMP Programming Paradigms onMulti-Core Processors Cluster," IEEE International Conference on Recent Trends in Information Systems, Jadavpur University,Kolkata, December 2011, pp. 137-140.
[27] R. Rabenseifner, "Hybrid Parallel Programming: Performance Problems and Chances," 45th CUG Conference, Columbus, May 2003,Available online at: http://www.cug.org.
[28] J. Roberts and S. Akhtar, "Multi-Core Programming : Increasing Performance through Software Multithreading," Technical Report,Available online at:http://www.intel.com/intelpress.

AUTHOR'S PROFILE

Dr. J. Sasi Kiran Graduated in B.Tech [EIE] from JNTU Hyd. He received Masters Degree in M.Tech [CSE] from JNT University, Hyderabad. He received Ph.D degree in Computer Science from University of Mysore, Mysore.

At present he is working as Professor in CSE and Dean – Academics in Farah Institute of Technology, Chevella, R.R. Dist Telangana State, India. His research interests include Image Processing, Data Mining and Network Security. He has published 45 research papers till now in various National, International Conferences, Proceedings and Journals. He has received best Teacher award twice from Farah Group, Significant Contribution award from Computer Society of India and Passionate Researcher

Trophy from Sri. Ramanujan Research Forum, GIET, Rajuhmundry, A.P, India.

Dr. G.Charles Babu received Ph.D in computer science & engineering from ANU in 2015, Masters in Software Engineering from JNTU in 1999 and B.Tech from KLCE in 1997. Presently working as a Professor CSE department in Malla Reddy College of Engineering (Autonomous). His research interest includes Data Mining, Information Retrieval, Cloud Computing and Image Processing. He has published more than 25 papers in Various international Journals

Ms. M. Kavya Graduated in B.Tech [CSE] from JNTU Hyd. She received Masters Degree in M.Tech from CBIT, HYD. Her Interested areas are Digital Image Processing and Artificial Intelligence. Currently, she is working as an Assistant Professor in Vidya Vikas Institute of Technology. She has published research papers in various National, International Conferences, Proceedings and Journals.

Mr.Md. Karimuddin graduated in B.Sc and Received MCA & M.Tech from JNTUH, Hyd. Presently working as Director-Academics in Farah Institute of Technology, Chevella from March 2015 to till date. His research interests include Data Mining, Information Retrieval and Cloud Computing. He has published research papers in various National, International Conferences, Proceedings and Journals.