International Journal of Latest Trends in Engineering and Technology
Vol.(8)Issue(4-1), pp.215-222
DOI: http://dx.doi.org/10.21172/1.841.37
e-ISSN:2278-621X

A FRAMEWORK FOR CONVERSION OF SERIAL PROGRAMMING PARADIGM INTO PARALLEL USING ARCHITECTURE INDEPENDENT FACTORS

Chennupalli Srinivasulu[1], Dr. Niraj Upadhyaya[2] & Dr A.Govardhan[3]

Abstract- Growing demand for the increase of performance of the programming paradigms to match with the increased demand for accommodating higher number of clients into smaller burst time to the processor is the significance and motivation for the research. The performance demand of the programming and the performance availability of the processing capabilities are the major trade-off in the recent advancements. The numerous research capabilities have demonstrated that the conversion of any serial programming model into parallel can achieve significant improvement in the scale of execution time, thus reducing the waiting time for the other processes running on the same processor and further improve the overall throughput of the system. The researchers are divided into two ideologies in order to achieve parallel models from serial programming models. The first demonstrated approach allows the developers and researchers to build the theoretical framework for converting the serial code models to parallel and in the other hand use of automated interactive tools allow the developers and researchers to build the parallel model from the serial code models automatically. Nevertheless, both demonstrated models can be argued in the light of advantages and shortcomings. The conversion of code into parallel under the aegis of theoretical framework is a highly time consuming task and demands higher knowledge on the parallel programming models. Nevertheless, the automated frameworks can convert the code without much control over the parallel models generated from the framework and argued for various uncontrolled performance variations. Thus the demand from the modern research is to build a framework with stable rules for converting the serial models into the parallel code models and the same algorithm can be implemented for interactive - controlled automation. Henceforth this work contributes in addressing the first demand to build a set of rules for converting the serial codes into parallel models. Furthermore, this work also proposes an algorithm for code conversion under the lights of novel defined rules. The yet another outcome of this work is to compare the proposed theoretical model to build the thoughts in order to demonstrate significant improvements in performances.
Keywords— Parallel Programming, Loop Optimization, Output Dependency Reduction, Operator Reduction, CUDA, HPHPU

1. INTRODUCTION

The variation of the performance for a same code model during different hardware architecture is majorly caused by the hardware support such as preloading and probabilistic execution of the hardware. For a same input data set, the execution time varies in different runs [1]. Nevertheless these variations are random and cannot be predicted, thus resulting into variable performance benchmarks. Thus a proper model of converting the serial programming models into a parallel code model is highly desired to ensure constant performance and improvements in the performance. In the last few years, a significant number of researches are been carried out. The outcomes of those researchers are argued based on the viability of the performance. The works are depended on the types of the architecture and bound by the architectural advantages. Other research dimensional limitations observed in the parallel research outcomes are measures of the performance improvements. Mostly the outcomes have demonstrated arithmetic average of the performances and demonstrated the improvements, thus resulting into a demand for higher order and accurate analysis.

The higher order of analysis is demonstrated by a few researchers considering the normalized architectural effects on the performance. The significant work by A. R. Alameldeenet. al. [1] and T. Kaliberaet. al. [2] has demonstrated the techniques to reduce the effects of hardware acceleration and other factors to identity the normalized performance measures of the programming models. Another notable work by T. Chenet. al. [3] demonstrate the use of non-parametric testing for a similar code model on various architectures and demonstrated the threshold to be considered for reducing the hardware effects on the analysis. Yet another leading research outcome by A. Georgeset. al. [4] considers the visual analysis of the performances which can be a newer direction in measuring the performances. The analysis of the hardware performance and influence of the hardware on the parallel programming models are hard to detect and the designers of the computer architectural models are sometimes clueless in terms of the factors to be improved to improve the performance of the parallel models. The work by D. J. Liljaet. al. [5] made a significant metric for dependent performance analysis of the hardware to detect the factors influencing the computational performances. Also, the work by S. Krishnamurthiet. al. [6] considerably explains the fact that programming models dependency on the hardware is strong. Nevertheless, with the understanding of that the existence of the factors influencing the performances of the programming model, the

[1] Research Scholar, JNTU Kakinada, AP, India.
[2] Dean and Professor of CSE, JBIT, Hyderabad, TS, India.
[3] Principal, JNTU Hyderabad, TS, India

independent factors of the programming model can be improved while converting a serial programming model into a parallel computational model. This work enhances the performance of the serial programming models by converting into parallel model with the emphasis on the architectural interdepend factors.
The rest of the paper is organized such as the second section focuses on the outcomes from the parallel researchers, in the third section the independent factors of the programming model is elaborated and the thoughts of improvement is established, in the fourth section of this paper the code conversion strategy is explained, in section – 5 the obtained results are been discussed and this paper presents the conclusion in the final section as section – 6.

2. OUTCOME FROM THE PARALLEL RESEARCHES
The recent advancements in the field of parallel programming is out bounded and motivated by the recent enhancements in the computer architecture. The biggest advancements in the recent years is the General Purpose Graphical Processing Units and inclusion of the GPGPUs in the modern architecture. The GPGPUs made the parallel programming and execution affordable in terms of the processing cost thus enhances the total performance of the system and makes the business more profitable. To support the parallelism, various SDKs, runtimes and APIs are been introduced by the companies inventing the GPGPUs. The use of these tools are extending the performance benefits and are been accepted widely. The demonstration of the CUDA from nVIDIA [7] and the Open CL by the Stoneet. al. [8] has significantly simplified the process of developing parallel programs for GPGPUs. The research outcomes by E. Alerstamet. al. [9] and Larsen E. S.et. al. [10] have motivated the researchers and developers to use parallel programming models for scientific applications. The image processing applications in the other hand demands even higher performance and parallelization for better processing and in time results. Some of the real-time implementation such as automation of traffic signal detection by Vladimir Glavtchevet a. [11], depth estimation from the real-time video sources by Woetzel J. at. Al [12] and higher order segmentation processes by Rumpf M.et. al. [13] has also demonstrated the benefits of using the parallel programming models. Also the animation and simulation operations focusing on processing higher order of the matrix input values demands parallel programming. Two major demonstrations of these benefits are by Purcell T.et. al. [14] for traceable inputs for graphical hardware acceleration and by Knott D.et. al. [15] for collision detection simulation.Also in the domain of security focusing on AES cryptography demands the parallel programming and been adopted by various researchers like Svetlin A. Manavski [16]. Nevertheless, these techniques are beneficial for developing the new applications; however the legacy applications also demand the improvements in the performance. The majority of the legacy programs and applications are built using the C programming languages. Thus conversion of these existing legacy applications and programs are the demand of the industry and research. The research outcome by T. D. Hanet. al. [17] has demonstrated the novel technique where the each functionality can be converted into separate modules and further can be allocated to individual GPGPUs for parallel execution using CUDA.
The code conversion methods for serial models to the parallel have seen a wide range of applications and frameworks. The automatic conversion of the source code into parallel execution is one of the approaches and widely accepted for the simplicity of the conversion process. The reduced complexity and reduced need for the knowledge to convert the code is the reason for this wide acceptance. The benefits are demonstrated by various researchers in the recent past. The work by David B. Lovemanet. al [18] related to parallelization of Fortran code and demonstrated notable outcome as High Performance Fortran or HPF. HPF focuses on the distributed memory based applications. Nevertheless, a number of applications demand the shared memory architecture for performance improvements. The work by Leonardo Dagum et. al. [19] has demonstrated the framework for such applications and widely accepted as OpenMP. Also various researchers have exhibited the interactive tools for code conversion as vfAnalyst by VectorFabrics [20], SUIF Explorer by Stanford University research group [21], and Polaris compiler by W. Blume et. al. [22] and ParaWise tool by Johnsonet. al.[23]. Nonetheless, the method for automatic conversion of the code or the process for manual conversion of the codes can be argued under the circumstance of developer’s knowledge on parallelization or the complexity of the process for manual and automatic process respectively. Thus the demand for simplification of the process for manual code conversion and the higher control for the automatic conversion needs to be improved. In the next section, this work attempts to reduce the complexity of manual code conversion by introducing simple rules and formulates the proofs of performance improvements by those rules.

3. THE PROPOSED THEORETICAL FRAMEWORK
The theoretical framework relies on few fundamental rules. These rules are subjected to proof and validation for further consideration and building the automatic framework algorithm. In this section, this work elaborates the rules and demonstrates the benefits in mathematical terms. The proofs are constituted in terms of Lemmas and will further be used to build the algorithm. Here this work formulates the lemmas consecutively.

Rule-1: The translation of the serial loops in the program necessarilyto be rehabilitatedtomandate the first level parallelization of the instructions. The use of pre-fetched results must be incorporated in the further instruction processing.

Rule-1 Elaboration: The program source codes are constituted with several programming blocks, where the iterative segments are often the larger parts. Thus converting the loops or the iterative segments will justify a significant amount of source code to parallel code. The improvement of the performance is also notable after unfolding the iterative segments. During the conversion process, the iterative statements needs to be readjusted in order to take the benefits from the pre-fetch and previous calculation values stored in the memory. Also the independent instructions can be converted into independent loops for parallel execution. Any sequential loop may have single or multiple statements to be iterated. These statements may be dependent on the previous statement or can be independent. The independent instructions can be executed in parallel [Table – 1].

Table I: Loop Parallelization – Independent Instructions

Original Code	**Parallel Code**
Loop: Do A[i] = A[i] + 1; B[i] = B[i] + 1; Done	Loop: Do A[i] = A[i] + 1; Done Loop: Do B[i] = B[i] + 1; Done

Another scenario may be demonstrated as the iterative instructions can be executed in parallel as the single instruction refers to different memory location for each instance [Table – 2].

Table II: Loop Parallelization – Independent Memory Access

Original Code	**Parallel Code**

Loop:
Do
A[i] = B[i] + C[i];
Done
	Loop:
	Do
	start_Parallel();
	A[i] = B[i] + C[i];
	Strop_Parallel();
	Done

Nevertheless, the performance benefits are subjected to demonstration of mathematical model.

*Lemma – 1***:** The conversion of theiteration parallelization reduces the execution time significantly.
Here,
T_1 denotes the amount of GPGPU time required to execute an instruction
T_2 denotes the time for each memory read operations
N denotes the number of iterations
M_1 denotes the number of instructions for GPGPU
M_2 denotes the number of memory read operations

Proof: Firstly the calculations for the serial code in terms of time taken is to be done

$$T(Ins) = \sum_{i=1}^{n} T_1 * N \qquad \text{(Eq. 1)}$$

And

$$T(Mem) = \sum_{i=1}^{n} T_2 * N \qquad \text{(Eq. 2)}$$

Where, T(Ins) and T(Mem) denote the total time taken for executing CPU based instructions and memory operations respectively.
Further, in case of a parallel execution the CPU instructions and the memory operations are designed to be performed in parallel. Thus,

$$T(P) = T(Ins) \cup T(Mem) \qquad \text{(Eq. 3)}$$

$$T(P) = \sum_{i=1}^{n} T_1 * N \cup \sum_{i=1}^{n} T_2 * N \qquad \text{(Eq. 4)}$$

Where, T(P) denotes the time taken to execute the instructions and memory operation in parallel design. The time taken here will be the maximum time for executing the instruction or the memory operations whichever is higher.

Thus can be formulated as,

$$T(P) = \left| \begin{array}{l} T(Ins),\ iff\ T(Ins) > T(Mem) \\ T(Mem),\ iff\ T(\text{Mem}) > T(Ins) \end{array} \right. \qquad \text{(Eq. 5)}$$

Hence, it is natural to say that the time required to execute the iteration will be less than the time required for executing the CPU instructions and memory operations combined.

$$T(P) < T(Ins) + T(Mem) \quad \text{(Eq. 6)}$$

Hence it is to be considered that, the conversion of the loops will demonstrate improvements in the performance.

Rule-2: The output dependency for the instructions must be altered to rename the variables.

Rule-2 Elaboration: The serial programming models defines the series of instructions which need to be executed in the specified order. In order to make the programming model execute in parallel, it is possible that the order of the sequence of the instruction execution is changed. This reordering of the instructions is highly efficient in order to achieve the parallelization. Nevertheless, it is also possible that due to the reordering the final output of the code may change. Thus it is proposed to change the order with newer variable in case the dependencies on the output are detected. The dependencies on the output are often observed while translating the serial code into parallel. Renaming or assigning a new variable for the code segment leaves the output undisturbed [Table – 3].

Table III: Variable Renaming – Output Dependency

Original Code	**Parallel Code**
X: = 3 A: = X + 3 Show A X: = 7 Show X	Parallel Execution – 1 X1: = 3 A: = X1 + 3 Show A Parallel Execution – 2 X: = 7 Show X

Nevertheless, the performance benefits are subjected to demonstration of mathematical model.

Lemma – 2: Avoiding Write After Write will reduce the chance of data hazards in case of concurrent execution.
Here,
T1 denotes the first instruction
T2denotes the second instruction

Proof: The serial instructions can be viewed as following:

$$\begin{aligned}&T1:\\&\mathrm{A} \leftarrow B + C\\&SHOW\,A\end{aligned} \quad \text{(Eq. 7)}$$

$$\begin{aligned}&T2:\\&Z \leftarrow A + X\\&SHOW\ Z\end{aligned} \quad \text{(Eq. 8)}$$

It is natural to understand that the serial execution policy applied on the T1 and T2 forces the execution order to be T1 and then followed by T2. Further, the instruction sets can be re-written as

$$\begin{aligned}&T1:[\text{Unchanged}]\\&\mathrm{A} \leftarrow B + C\\&SHOW\,A\end{aligned} \quad \text{(Eq. 9)}$$

And

$$\begin{aligned}&T2:[\text{Changed} - \text{Renamed}]\\&Z \leftarrow K + X\\&SHOW\ Z\end{aligned} \quad \text{(Eq. 10)}$$

Thus the effects of the Write and Write can be avoided during the translation into parallel codes.

Rule-3: The operators without deliberating the precedence must be reduced for any serial instruction.

Rule-3 Elaboration: The serial programming models executes the operators in a specified series in case of the similar priority. The instructions execute the operators with the results achieved in the previous execution. Nevertheless, the operators are not interdependent and executing them in parallel will not defer the final result. The trivial examples as performing the sum operation of all the elements in the array can be performed in parallel in order to improve the performance [Table – 4].

Table IV: Operator Reduction

Original Code	Parallel Code

Loop
Do
Sum = Sum + A[i]
Done
	Do_Parallel
	Temp = Temp + A [i] + A [i + 1]
	End_Parallel
	Sum = Sum + Temp

Nevertheless, the performance benefits are subjected to demonstration of mathematical model.

Lemma – 3: Reducing the number of operators can significantly improve the performance of the serial code and can be converted into parallel.
Here,
N denotes the number of operators in the instruction
T denotes the time to execute each instruction

Proof: In the serial execution model the operators will be executed sequentially and the total time for computation of N operators can be calculated as,

$$T(x) = \sum_{i=1}^{N} T_N \qquad \text{(Eq. 11)}$$

Further converting the number of operators to be executed in parallel demands reduction in the operators,

$$T'(x) = \sum_{i=1}^{N} \frac{T_N}{2} \qquad \text{(Eq. 12)}$$

Where $T'(x)$ denotes the reduction in the first level.
Further,

$$T''(x) = \frac{T'(x)}{2} = \sum_{i=1}^{N} \frac{T_N}{4} \qquad \text{(Eq. 13)}$$

Hence, it is natural to understand that this geometric series will result into a finite series.

$$T(N) = \frac{1}{2^n} \qquad \text{(Eq. 14)}$$

Henceforth, the comparison between the serial execution time and parallel execution time needs to be compared and the improvement is significant.

$$T(N) < \mathrm{T(x)} \qquad \text{(Eq. 15)}$$

Furthermore, these lemmas and the rules are to be used in the proposed algorithm demonstrated in the next section.

4. THE NOVEL CODE CONVERSION ALGORITHM

In this section of the work, the code conversion algorithm is designed and demonstrated.

Step-1. Partitioning the process of dividing the computation and the data into pieces.
Step-2. Communication The process of determining how tasks will communicate with each other, distinguishing between local communication and global communication.
Step-3. Agglomeration The process of grouping tasks into larger tasks to improve performance or simplify programming.
Step-4. Mapping The process of assigning tasks to physical processors.

The benefits and the detailed explanation of the steps are constituted in the section – 3.

5. RESULTS AND DISCUSSIONS

The theoretical model is been evaluated on legacy C codes and the results are been observed. Firstly, the Matrix multiplication code is been translated and the performance improvements are been observed [Table – 5].

Table V: Matrix Multiplication Code - Performance Improvements

Number of Elements	Time in Serial Execution (ns)	Time in Parallel Execution (ns)	Improvements (ns)
128	0.16	0.3	-0.14
256	1.2	0.8	0.4
512	10	4	6
1024	170	30	140

The results are been observed graphically [Fig – 1].

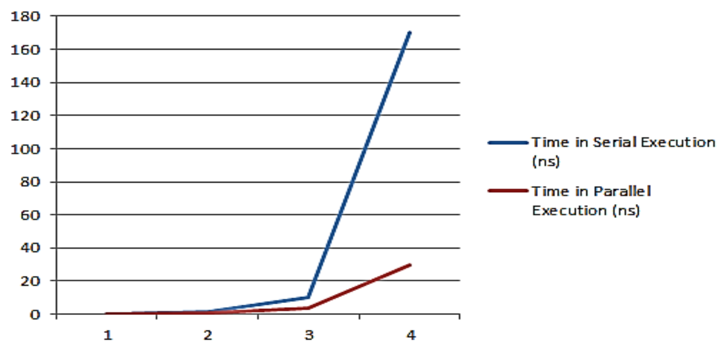

Fig. 1 Matrix Multiplication Code - Performance Improvements

Secondly, the Array power code is been translated and the performance improvements are been observed [Table – 6].

Table VI: Array Power Code - Performance Improvements

Number of Elements	Time in Serial Execution (ns)	Time in Parallel Execution (ns)	Improvements (ns)
1024	0.04	0.282	-0.242
4096	0.574	0.344	0.23
8192	2.284	0.486	1.798
16384	9.104	1.168	7.936
32768	36.402	3.884	32.518
65536	145.6	14.694	130.906

The results are been observed graphically [Fig – 2].

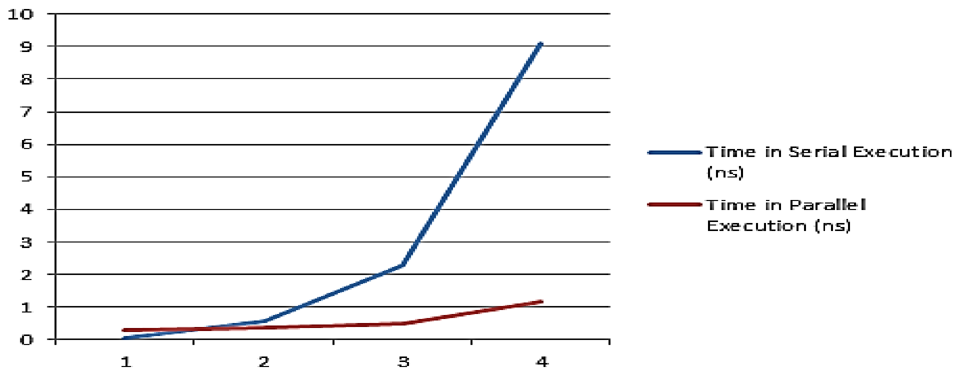

Fig. 2 Array Power Code - Performance Improvements

Thirdly, the Prime divisor codeis been translated and the performance improvements are been observed [Table – 7].

Table VII: Prime Divisor Code - Performance Improvements

Number of Elements	Time in Serial Execution (ns)	Time in Parallel Execution (ns)	Improvements (ns)
1024	0.004	0.202	-0.198
4096	0.012	0.256	-0.244
8192	0.556	0.424	0.132
16384	2.112	1.108	1.004
32768	8.324	3.63	4.694
65536	33.1	13.64	19.46

The results are been observed graphically [Fig – 3].

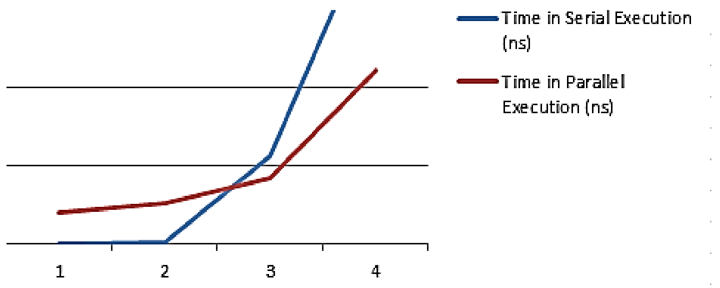

Fig. 3 Prime Divisor Code - Performance Improvements

Thus with the understanding of the performance improvements, the work presents the conclusion in the next section.

6. CONCLUSION

The outbound growth in the space of Computer Architecture is encouraging the parallel programming into practice. The recent advancements in processor architecture with the invention of GPGPUs made the parallel programming less costly. Thus in the recent research various researchers have demonstrated various models for the parallel programming. Nevertheless, the legacy systems that are build on serial programming models are also to be enhanced in order to take the advantages from the architectural improvements. Majority of the Legacy systems are built using C programming language, thus the frameworks for converting serial programs into parallel model like CUDA became highly popular and widely accepted. Nevertheless, many researchers have also demonstrated the use of automatic code conversion frameworks. The manual conversion of code demands a higher knowledge on parallel programming model and nonetheless a crucial task for understand or maintaining the code during the conversion process. Nevertheless, the automatic conversion process does not demand such higher level of understanding, but provides much lesser control of the code during the conversion process. Henceforth, this work makes an attempt to formulate the manual conversion process and proposes a framework, which can be automated further. The major outcome of this work is to standardize the steps for converting the serial program into a parallel model using simple to follow and understandable steps. This work also establishes the factors to realize the improvements using mathematic model and presents the results demonstrating nearly 70% reduction of execution time, thus promises to provide a better programming world to serve the mission critical application needs.

7. REFERENCES

[1] A. R. Alameldeen and D. A. Wood, “Variability in architectural simulators of multi-threaded workloads,” in Proc. 9th Int. Symp. High-Perform. Comput. Archit., 2003, pp. 7–18.

[2] T. Kalibera and R. Jones, “Rigorous benchmarking in reasonable time,” ACM SIGPLAN Notices, vol. 48, no. 11, pp. 63–74, 2013.

[3] T. Chen, Y. Chen, Q. Guo, O. Temam, Y. Wu, and W. Hu, “Statistical performance comparisons of computers,” in Proc. Int. Symp. High Perform. Comput. Archit., 2012, pp. 1–12.

[4] A. Georges, D. Buytaert, and L. Eeckhout, “Statistically rigorous java performance evaluation,” ACM SIGPLAN Notices, vol. 42, no. 10, pp. 57–76, 2007.

[5] D. J. Lilja, Measuring Computer Performance, A Practitioner’s Guide. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[6] S. Krishnamurthi and J. Vitek, “The real software crisis: Repeatability as a core value,” Commun. ACM, vol. 58, no. 3, pp. 34–36, 2015.

[7] NVIDIA, NVIDIA CUDA Compute Unified Device Architecture-Programming Guide, Version 3, 2010.

[8] Stone, J.E., Gohara, D., Guochun Shi, ―OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems‖, Computing in Science and Engineering, Vol. 12, Issue 3, pp. 66-73, May 2010

[9] E. Alerstam, T. Svensson and S. Andersson-Engels, "Parallel computing with graphics processing units for high speed Monte Carlo simulation of photon migration" , J. Biomedical Optics 13, 060504 (2008).

[10] Larsen E. S., Mcallister D., ―Fast matrix multiplies using graphics hardware‖, Proceedings of the 2001 ACM/IEEE Conference on Supercomputing, Nov. 2001, pp. 55.

[11] Vladimir Glavtchev, Pinar Muyan-Ozcelik, Jeffrey M. Ota, John D. Owens, "Feature-Based Speed Limit Sign Detection Using a Graphics Processing Unit", IEEE Intelligent Vehicles, 2011.

[12] Woetzel J., Koch R., ―Multi-camera realtime depth estimation with discontinuity handling on PC graphics hardware‖, Proceedings of the 17th International Conference on Pattern Recognition (Aug. 2004), pp. 741–744.

[13] Rumpf M., Strzodka R., ―Level set segmentation in graphics hardware‖, Proceedings of the IEEE International Conference on Image Processing (ICIP ’01), Oct. 2001, vol. 3, pp. 1103–1106.

[14] Purcell T. J., Buck I., Mark W. R., Hanrahan P., ―Ray tracing on programmable graphics hardware‖, ACM Transactions on Graphics 21, 3 (July 2002), pp 703–712.

[15] Knott D., Pai D. K., ―CInDeR: Collision and interference detection in real-time using graphics hardware‖, Proceedings of the 2003 Conference on Graphics Interface, June 2003, pp. 73–80.

[16] Svetlin A. Manavski, "Cuda compatible GPU as an efficient hardware accelerator for AES cryptography" Proc. IEEE International Conference on Signal Processing and Communication, ICSPC 2007, (Dubai, United Arab Emirates), November 2007, pp.65-68.

[17] T. D. Han and T. S. Abdelrahman, "hiCUDA: High-Level GPGPU Programming", IEEE Transactions on Parallel and Distributed Systems, Jan. 2011, vol. 22, no. 1, pp. 78-90.

[18] David B. Loveman, ―High Performance Fortran‖, IEEE Parallel & Distributed Technology: Systems & Technology, February 1993, v.1 n.1, pp 25-42.

[19] Leonardo Dagum and Ramesh Menon, ―OpenMP: An industry-standard API for shared-memory programming‖, IEEE Computational Science and Engineering, 5(1):46–55, January–March 1998.

[20] VectorFabrics. vfAnalyst: Analyze your sequential C code to create an optimized parallel implementation. http://www.vectorfabrics.com/.

[21] M. Hall, J. Anderson, S. Amarasinghe, B. Murphy, S.-W. Liao, E. Bugnion, and M. Lam, ―Maximizing multiprocessor performance with the SUIF compiler‖, IEEE Comput. 29, 12, Dec. 1996, pp 84–89.

[22] W. Blume, R. Doallo, R. Eigenmann, J. Grout, J. Hoeflinger, T. Lawrence, J. Lee, D. Padua, Y. Paek, B. Pottenger, L. Rauchwerger, and P. Tu. ―Advanced Program Restructuring for High-Performance Computers with Polaris‖, IEEE Computer, December 1996, Vol. 29, No. 12, pages 78- 82.

[23] Johnson, S.P., Evans, E., Jin, H., Ierotheou, C.S., ―The ParaWise Expert Assistant—Widening accessibility to efficient and scalable tool generated OpenMP code‖, WOMPAT, pp. 67–82 (2004).

ABOUT THE AUTHORS

Mr.Ch.Srinivasulu has obtained his B.Tech Degree from SV University,and M.Tech (CSE) from JNT University, Hyderabad. He is having nearly 20 years experience in Industry as well as a faculty of Computer Science and Information Technology departments. He is pursuing his PhD from JNTU kakinada. His area of research includes Computer Architecture, Parallel Computing, Software Engineering.

Dr. A.Govardhan is presently a Professor of Computer Science & Engineering at JNTUH Jawaharlal Nehru Technological University Hyderabad (JNTUH), India. He did his B.E.(CSE) from Osmania University College of Engineering, Hyderabad in 1992, M.Tech from Jawaharlal Nehru University(JNU), New Delhi in 1994 and Ph.D from Jawaharlal Nehru Technological University, Hyderabad in 2003. His area of interest Databases, Data Warehousing & Mining, Information Retrieval,Computer Networks,ImageProcessing and Object Oriented Technologies.

Dr. Niraj Upadhyaya, Ph.D, is an eminent scholar, professor in the CSE Department and Dean of Academics at J.B. Institute of Engineering & Technology, Hyderabad. He has his name established in the field of "Parallel Computing" and has many articles, papers and journals to his name. He had his Ph.D. from the University of West of England, Bristol, UK with the topic of "Memory Management of Cluster HPCs". He has more than 20 years of experience.