International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 04 Issue: 07 | July -2017 www.irjet.net p-ISSN: 2395-0072

An Adjacent Analysis of the Parallel Programming Model Perspective: A Survey

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

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

***Abstract*— The growing nature of the demand on the increasing use of parallel computing and parallel programming by the application development industry is forcing the researcher community to bring the modern and novel frameworks and models in order to increase the performance. A numerous number of research attempts are been carried out by renounced researchers in the past. Nevertheless, this domain is been the point of focus for over a decade due to the endless possibilities of the scope. The existing research outcomes are always been criticized due to the lack of practically and understand ability. Specifically, the ease of use to improve the programming paradigms based on the proposed methods. Also, the scope for improvement is always been ignored by the researchers. Thus, this work significantly contributes towards the practical importance and implications of the parallel programming models from a newer perspective. The major outcome of this work is to compare the benefits and drawbacks of the existing models and furthermore propose the improvements in each model. This final outcome of the work is to develop new directions and guidelines for the upcoming research attempts, thus providing significant roadmap for the researches.**

***Keywords*— Weighted Comparative Model, PRAM, UMP, BSP, ASM, DPM, TPM**

I. INTRODUCTION

The hardware level threads provided by the computer hardware processor helps the developers to realize multiple program level threads. The expectations from the high performance computer systems are never ending and soon will come to an end. As the capabilities of providing hardware level threads are restricted to the processor clock speed and increasing further may cause high power consumption or heating up problem or both. Henceforward, the further deployment of the power of instruction level multithreading will reach the pick of the performance. In the other hand, the performance improvement demand is never supressed due to the demand of the consumer services. Thus, it is natural to understand that the development community of the application development industry soon will face the bottleneck in the high performance computing domains. Hence, it is the demand of the modern research to improve the situation. The implication of this situation can be only improved by improving the parallel programming models and deal with the multi-processing from the programming perspective. As implied by the demand of the research, it is necessary to study and find the practical feasibilities of implementation and limitations of the available parallel programming models. Further, identify the scope for the further research and improvements inspired by such studies.

Every parallel computational techniques or models are built upon two major components such as deployed parallel programming framework or model and the associated cost model. The first component, as noted as the parallel programming model is the associated parallel execution rules defined as various rules based on mathematical operations, task segregation, read and write operations from and on the working memory and finally the message passing technique. Also the rule sets contains the restrictions of the system as the conditional applicability of the rules while processing the source code. Few of the shared memory depended models are explicit about the memory access rules as well. The memory access rules define the strategies of memory access during the source code processing as how the memory locations will be accessible by the model. The other non-shared memory models define the memory access strategies implicitly but with a strong significance. The second component, as noted as the associated cost model for the framework, defines the cost of the operations and demonstrates the method for calculating the cumulative cost of the entire operation.

Another major component from the implementation outlook for the parallel computational frameworks is the associated programming language libraries. The programming language libraries are generally the collection of application programming interfaces which significantly improves the parallelism in the source code or inversely the parallelism can be obtained using these predefined methods.

The fundamental demand behind the evaluation of the parallel programming models or in a bigger viewpoint the parallel computational models are the rapid evolution and improvements of the parallel algorithms. The observations made on the parallel algorithm implantations are some of the algorithms have demonstrated better performance when implemented using specific parallel programming models. The identified but argued reason as stated by many researchers are specific parallel programming models are implemented focusing on specific hardware hence proven to be very success on those hardware architectures. This is also a notable downside of the existing models and demands significant improvements in hardware independence including other factors.

The rest of the work is constructed such as in Section – II the review of the recent improvements in this research domain is carried out, in Section – III the parallel programming models are been discussed with the light of implantation benchmarks, in Section – IV the comparative discussion are furnished considering the recompenses and hindrances, in Section – V the results from various models are discussed and compared and finally in the Section – VI the work delivers the conclusion.

II. Review of Literature

In this section, this work analyses and reviews the previous outcomes by other research attempts. Firstly, the work by Bal et al [1] have demonstrated significant study on parallel programming techniques on distributed systems. Also the works by Giloi et al. [2] have contributed in understanding the independence factors of parallel programming on parallel architecture. The notable recent outcome by Skillicorn et al. [3] in deriving the understanding of practicality of parallel programming models also contributed largely on this research domain. Nonetheless, the contribution of the survey by the B. M. Maggs et al. [4] is also appreciated by the research community. A decent amount of work is been analysed and surveyed by Domenico Talia et al. [5] [6] is also helped in understanding programming language associations with the parallel models. Another significant but different directive work by Susanne E. Hambrusch et al. [7] have demonstrated the trade-off points behind the instruction level parallelism. The works by Christian Lengauer et al. [8] and Claudia Leopold et al. [9] have demonstrated the continuous progressing nature of the parallel programming models.

Inspired by the existing studies, the work takes the surveying task presumptuously in analysing the other relevant models in this work.

Firstly, The parallel random access machine model is one of the significant research outcomes in this domain. The parallel random access machine developed and demonstrated by Fortune et al [10] is designed for analysing the sequential algorithms. The model demonstrates the use of multiple processing units and connected a shared memory unit. The model exhibits the use of a globally associated clock for defining the clock cycles for the shared memory and every processor in the model. The significant feature of the parallel random access machine is to define the synchronous execution of the complete system. The clock cycles controlling every processor in the model is the key factor [11] [Fig – 1].

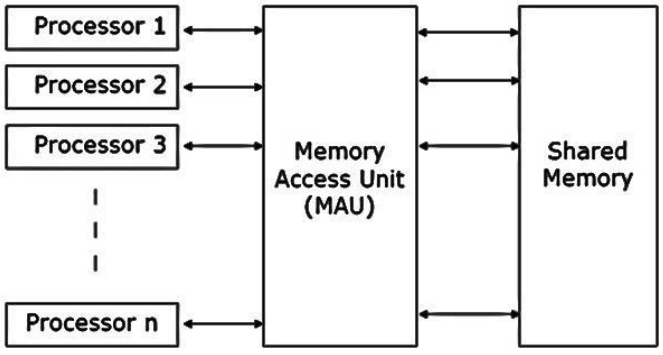

Fig. 1Parallel Random Access Machine Model

Secondly, the distributed memory architecture called the unrestricted message passing is generally a set of RAM configured in parallel and can run asynchronously. The architecture communicates over the message passing method over the communication channel connected with every RAM. The complete configuration is connected with the processor core. The functional factor for this model depends on the message routing of the connected network [Fig – 2].

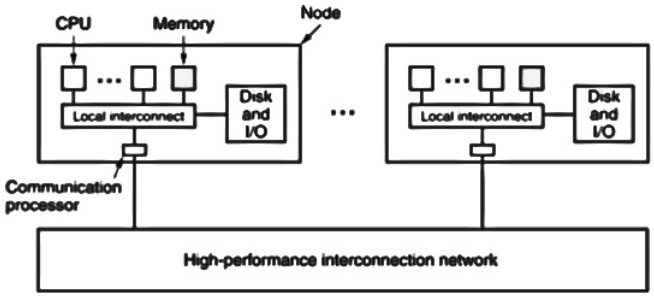

Fig. 2Message Passing Multicomputer

The associated cost model for unrestricted message passing model is classified into two parts. Firstly, the local operations on the same core are considered as performed on the same RAM and secondly, the point to point communications are modelled by LogP method model [12].

Thirdly, the Bulk Synchronous Model was first introduced by the Valiant et al. in the year of 1990 [13] and the revised model was presented by McColl by the year of 1993 [14]. This model works on the principle of structured dynamic message passing computations. The structured model deploys a sequence of supper steps. The sequence of super steps involves the computational operations on the locally configured data members. The associated cost model is denoted as following:

$$Cost = W\_Load + C\_Vol * N\_Bandwidth + Over\_Head \quad \text{(Eq. 1)}$$

Where,
W_Load denotes the number of loads on the local processor
C_Vol denotes the communication volume on each processors
N_Bandwidth denotes the network band width
Over_Head denote the total barrier overhead of the process

Looking at the popularity of this model, an extension by Skillicorn et al. was proposed by the year of 1996 and wise widely accepted [15].

Fourthly, another remarkable contribution is the Asynchronous Shared Memory Model and Partitioned Global Address Space [Fig – 3]. The shared memory model works with multiple threads of execution asynchronously accessing the shared memory. The model is partially identical to the parallel random access machines. Nevertheless, the models where been differentiated significantly based on simplified architecture of the Shared Memory Model by introducing multiple consistency models to ensure correct executions [11].

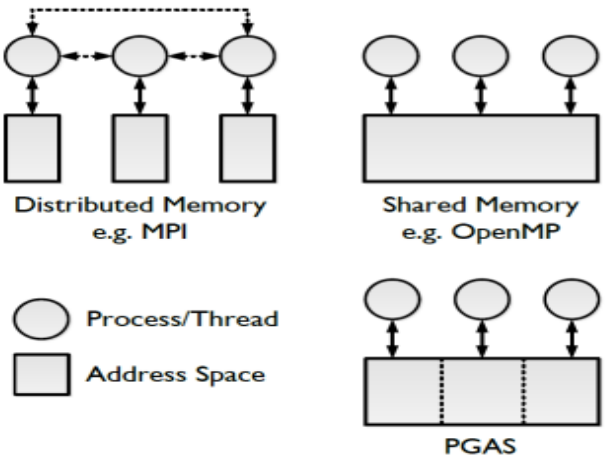

Fig. 3Asynchronous Shared Memory Model

Another recent advancement in the same direction is a transactional memory deploying the principal of primitive parallel computing [16] [17].

Fifth in the count is the Data Parallel Model. The Data Parallel model [Fig – 4] includes the principle of single instruction for multiple data. The model involves the workability of the parallel processing of the data, systolic computational process and stream processing of the data. The notable proposal by J. Rose et al [18] and the enhancement of the same model proposed by Johannes Jendrsczok et al. [19] is adopted by a wide number of researchers and developers.

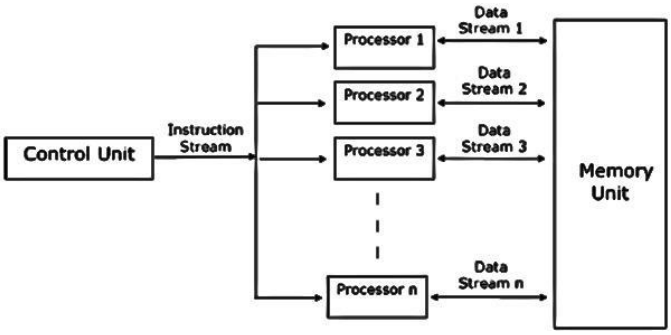

Fig. 4 Parallel Data Model

Finally in the list the Task Parallel Model [Fig – 5] is listed. Most of the applications are considered to be a collective set of tasks and need to be executed independently in case of a parallel programming approach. Frequently the distinguished tasks need to communicate between each other during the execution phase. The group of tasks can be represented by Task Graph. Based on the task graph, the tasks are scheduled for ordering the tasks and allocating the tasks to the processing units. The task graphs are used majorly in grid computing [11]. Here each node may already representan executable with a runtime of hours or days.

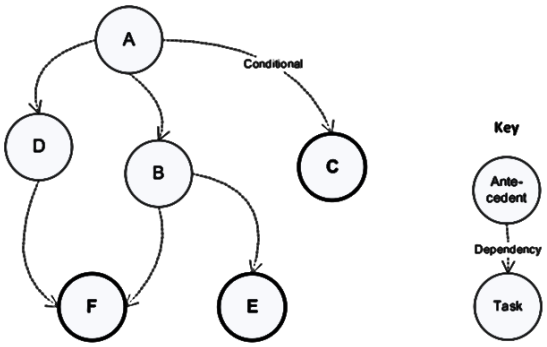

Fig. 5 Task Parallel Model

Henceforth, with the decent understand of the literature focusing on the various parallel computing architectural models, this work analyses the parallel programming models in the next section.

III. PARALLEL PROGRAMMING APPROACHES

In this section, the survey work analyses most popular and widely accepted methodologies for designing and developing the parallel programmed application.

A. PCAM Approach

The notable contribution by Foster et al. [20] [21] [22] proposed the solution of making any application parallel as considering the entry point as a sequential application and partition the tasks inside the algorithm. Further converting the independent partitions into tasks and identify the dependencies of the tasks. The dependencies are usually result into the communication between the tasks and can be managed in parallel.

B. Incremental Parallelization

The distinguished demonstration by Ken Kennedy et al.[23] have shown that, most of the scientific application are time complex and the high time complexity is due to the small code parts with the recursive execution. Most of the programming languages provide support to start the execution in the sequential manner and then convert each independent component of the source code into parallel execution. Those programming languages allow the recursive components such as syntactic loops into parallel execution thus reduce the time complexity.

C. Automation of Parallelization

The use of automation for converting application into parallel approaches is one of the most significant and highly demanded methods in the application development industry. The conversion of the parallelism can also be achieved manually. Nevertheless the complexity of manageability is tremendously high. Hence, the research by Beniamino di Martino et al. [24] is one of the notable guideline for automation of parallelization. The work demonstrates the conversion method by separating the source code of the application into two parts as convertible with the hardware acceleration and convertible by programming logic. The first category of the source code types can be manipulated by taking the advantages of hardware parallelization [25] and the significantly the use of code automation tools over the second category can achieve the desired automation.

D. Skeleton Based Parallelization

The skeletonized programming languages provide a numerous ways to convert the application source code into parallel programming approaches. The skeletons are generally the independent components of the application and can easily be converted into parallel execution. Various skeleton based programming languages as P3L introduced by Bruno Bacci et al. [26] and Susanna Pelagatti et al. [27], SCL introduced by J. Darlington et al. [28] [29], eSkel introduced by Murray Cole et al. [30] [31], MuesLi introduced by Herbert Kuchen et al. [32] and finally QUAFF introduced by Joel Falcou et al. [33] provides multiple solutions of converting the skeleton into parallel execution.

After understand various parallel programming approaches, this survey work understands the need for further research in order to enhance the existing methods. The highly targeted area as identified by this work is the automation of parallelization and identify as the area for further concentrations.

IV. COMPARATIVE DISCUSSIONS

This section of the survey work is dedicated for comparative study of the existing practically of the existing models with their relevance to the modern research outcomes.

A. PRAM Approach

The benefit of the PRAM model is to obtain support for the parallel computations and also the availability of the programming modules are apparently highly friendly to the application developers. The wide acceptance for the PRAM model forces the researchers to build and test multiple algorithms [34] for the model and also used for various parallel programming tutorials [35]. Nonetheless, the associated cost model is been criticised for being far from the reality. However, the enhancements demonstrated by various research attempts are been accepted with notable improvements. The cost effectiveness is realized focusing and taking the advantages of hardware acceleration such as NYU Ultra-computer [36], SBPRAM introduced by Wolfgang et al. [37, 38, 39], XMT introduced by Vishkin et al. [40] and Eclipse introduced by Forsell et al. [41].

B. UMP Approach

The unrestricted message passing or the UMP approach is notable for the benefit of using for concurrent and distributed application development. The popularity of this model increased with the introduction of the first parallel-distributed memory machine architecture in the year of 1980. This model is also not above the criticism by the researchers. The major two setbacks for this model is the message passing approach is not unique for this model and can be achieved in other systems as well and secondly, the scheduling of the tasks demands higher understand ability of the system and highly complex to maintain.

C. BSP Approach

The bulk synchronization parallelization approach or the BSP approach is introduced in the year of 1990 with the ease of higher parallelization on the bulk set of the instruction in the applications. The major outcome of the BSP model is to correctly predict the execution time for the algorithms thus helps the developers to enhance the algorithm to take the advantages of parallelization and reduce the trade off between the time complexity and architectural challenges. The BSP model is also used for educating the researchers and practitioners to build the knowledge on parallel algorithm design [42].

D. ASM Approach

The asynchronous shared memory approach or the ASM approach is widely used for the small scale programming methods and been proven to improve the performance. The major challenge of this model is maintain the memory with sufficient details and the complete memory information has to be exposed to the developers.

E. DPM Approach

The data parallel model approach or the DPM approach was introduced in the year of 1970 and till the year of 1980 was been the major focus point for the researchers. Nevertheless, the consistent focus and enhancements on this model made DPM one of the essential parts of the advanced outcome from the recent researches. Nonetheless, the data parallel access model is also been criticized for being restricted to highly configured hardware and not being available for small businesses.

F. TPM Approach

While data flow computing in itself has become a mainstream in programming, it has seriously influenced parallel computing and its techniques have found their way into many products. Hardware software co design has gained some interest by the integration of recognizable hardware with microprocessors on single chips. Grid computing has gained considerable attraction in the last years, mainly driven by the enormous computing power needed to solve grand challenge problems in natural and life sciences.

The comparative analysis is been carried out in this work and presented here [Table – 1].

TABLE I: COMPARATIVE ANALYSIS OF THE APPROACHES

Model	Introduced In	Benefits	Challenges	Minor Outcomes
PRAM	1992	• In Build Programming Modules are highly developer friendly • Can use the hard ware acceleration	• The associated cost model cannot be accepted for all purposes for various practical deficiencies	• Model for realizing learning approaches for the beginners
UMP	1980	• Highly efficient for distributed applications • Great hardware control for the parallel execution	• The message passing can also be achieved by other available models as well • Designing and maintaining the parallel tasks can be very complex	• Task level parallelization can be achieved
BSP	1990	• Nearly accurate prediction of the execution time	• The accurate prediction of the execution time is mostly hardware independent, hence the outcome may vary based on the configuration of the systems	• Can be used to enhance the application performances step by step

ASM	1989	• Highly effective for small scale applications and businesses	• Criticized for not been applicable for the highly time complex algorithms • During execution, the memory configuration need to be complexity exposed to the application	• Testing high scale application before migration can be beneficial
DPM	1970 & revised on 1980	• Exceedingly efficient for the recent digital media accessing applications	• Greatly dependent on the hardware and demands consistently high configuration for the hardware	• Digital media storage accessing can be appreciated looking at the performances
TPM	2005	• Influences the parallel programming theory • Applicable on Grid based computing	• Criticized for not been applicable for the highly time complex algorithms	• Greatly successful for grid based computing models

With the detailed understanding of the available approaches, in the results section this work analyses the programming language supports and grade the applications based on the weighted scores.

V. Results and Discussions

This work proposes an enhanced and highly novel model for comparison of the available models. The programming language library support and available data structure supports are the two components for the comparative analysis. The programming languages and the data structures are been ranked with relevant weights.

Firstly, the weighted programming language ranks are been framed [Table – 2] [43].

TABLE II: Programming Languages and Weights

Model	Weights
Fortran	1
C	2
C++	3
Vector – C	4
PGAS	5
FPGA	6

Secondly, the weighted data structures are been listed [Table – 3].

TABLE III: Data Structures and Weights

Model	Weights
Array	1
Matrix	2
Vector	3
Tree	4

Further, this model proposes the novel comparative weighted ranks for the approaches. The following formulation defines the calculation process.

$$P_{Rank} = \sum_{i=1}^{n} \begin{matrix} If\ \ Rank\ Value > 1, P_{Rank} + Weight_i * P_{iRank} \\ Else\ P_{Rank} \end{matrix} \quad \text{(Eq. 2)}$$

Where,
P_{RANK} denotes the rank of the model or the approach based on the programming language support
$Weight_i$ denotes the weight for the programming language

$$D_{Rank} = \sum_{i=1}^{n} \begin{matrix} If\ \ Rank\ Value > 1, D_{Rank} + Weight_i * D_{iRank} \\ Else\ D_{Rank} \end{matrix} \quad \text{(Eq. 3)}$$

Where,
D_{RANK} denotes the rank of the model or the approach based on the Data Structure support
$Weight_i$ denotes the weight for the data structure

$$Rank = P_{Rank} + D_{Rank} \quad \text{(Eq. 4)}$$

Henceforth the analysis of compatibility for programming languages [Table – 4] and data structure is been presented here [Table – 5].

TABLE IV: COMPATIBILITY WITH PROGRAMMING LANGUAGES

Model	C	C++	Fortran	PGAS	FPGA	Vector – C
PRAM	1	0	0	0	0	0
UMP	1	1	1	0	0	0
BSP	1	0	0	0	0	0
ASM	1	0	1	1	0	0
DPM	1	0	0	0	1	0
TPM	1	0	0	0	0	1

TABLE V: COMPATIBILITY WITH DATA STRUCTURES

Model	Matrix	Vector	Tree	Arrays
PRAM	0	0	0	1
UMP	1	1	0	1
BSP	0	0	0	1
ASM	1	0	0	1
DPM	0	1	0	1
TPM	1	1	1	0

The final score of the approaches are been presented here [Table – 6] with the light of Eq. 2 to Eq. 4.

TABLE VI: SCORE MODEL

Model	Programming Language Score	Data Structure Score	Total Score
PRAM	2	1	3
UMP	6	6	12
BSP	2	1	3
ASM	8	3	11
DPM	7	3	10
TPM	5	9	11

Thus the ranking of the approaches are considered here [Table – 7].

TABLE VII: RANKING MODEL

Model	Total Score	Ranking
UMP	12	1
ASM	11	2
TPM	11	3
DPM	10	4
PRAM	3	5
BSP	3	6

The results are been analysed graphically here [Fig – 6] [Fig – 7][Fig – 8].

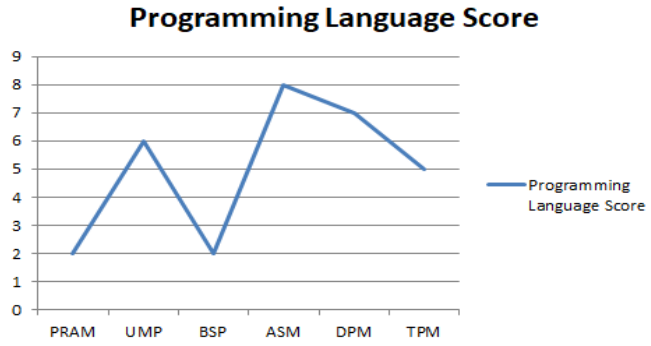

Fig.6Programming Language Score Analysis

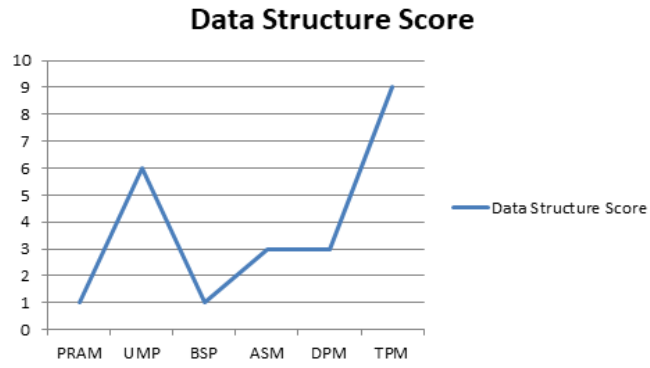

Fig.7Data Structure Score Analysis

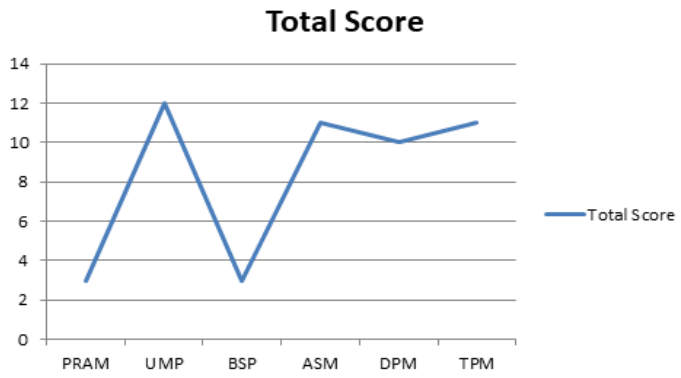

Fig.8Weighted Score Analysis

VI. CONCUSSION

The never degrading nature of the demand for parallelization in application development, the support for the programming paradigms and parallel computing models are strongly encouraged by the researchers. This survey work analyses the computation models and the programming approaches as existing with a critical view towards finding the enhanced research directions towards improvements of the performance and ease of use to the application developers. This work analyses PRAM, UMP, BSP, ASM, DPM andTPMcomputational methods. In the course of study, the major outcome is the novel ranking method based on the programming language support and available data structure support. Based on the obtained weights, the approaches are been ranked. The work finally concludes to elaborate the work on automation of parallelization and proposes to work further on this direction in order to save costly man houses and provide less time complex applications, which can be used for various purposes like Life Saving Analysis, Financial Analysis or Complex Storage structures.

REFERENCES

[1] Henri E. Bal, Jennifer G. Steiner, and Andrew S. Tanenbaum. Programming Languages for Distributed Com- puting Systems. ACM Computing Surveys, 21(3):261322, September 1989.
[2] W. K. Giloi. Parallel Programming Models and Their Interdependence with Parallel Architectures. In Proc. 1st Int. Conf. Massively Parallel Programming Models. IEEE Computer Society Press, 1993.
[3] D. B. Skillicorn. Models for Practical Parallel Computation. Int. J. Parallel Programming, 20(2):133158, 1991.
[4] B. M. Maggs, L. R. Matheson, and R. E. Tarjan. Models of Parallel Computation: a Survey and Synthesis. In
[5] David B. Skillicorn and Domenico Talia, editors. Programming Languages for Parallel Processing. IEEE Com- puter Society Press, 1995.
[6] David B. Skillicorn and Domenico Talia. Models and Languages for Parallel Computation. ACM Computing Surveys, June 1998.
[7] Susanne E. Hambrusch. Models for Parallel Computation. In Proc. Int. Conf. Parallel Processing, Workshop on Challenges for Parallel Processing, 1996.
[8] Christian Lengauer. A personal, historical perspective of parallel programming for high performance. In Günter Hommel, editor, Communication-Based Systems (CBS 2000), pages 111118. Kluwer, 2000.
[9] Claudia Leopold. Parallel and Distributed Computing. A survey of models, paradigms and approaches. Wiley, New York, 2000.
[10] S. Fortune and J. Wyllie. Parallelism in random access machines. In Proc. 10th Annual ACM Symp. Theory of Computing, pages 114118, 1978.
[11] Sarita V. Adve and KouroshGharachorloo. Shared Memory Consistency Models: a Tutorial. IEEE Comput., 29(12):6676, 1996.
[12] David E. Culler, Richard M. Karp, David A. Patterson, AbhijitSahay, Klaus E. Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. LogP: Towards a realistic model of parallel computation. In Principles & Practice of Parallel Programming, pages 112, 1993.
[13] Leslie G. Valiant. A Bridging Model for Parallel Computation. Comm. ACM, 33(8):103111, August 1990.
[14] W. F. McColl. General Purpose Parallel Computing. In A. M. Gibbons and P. Spirakis, editors, Lectures on Parallel Computation. Proc. 1991 ALCOM Spring School on Parallel Computation, pages 337 - 391. Cambridge University Press, 1993.
[15] D. B. Skillicorn. miniBSP: a BSP Language and Transformation System. Technical report, Dept. of Computing and Information Sciences, Queens's University, Kingston, Canada, Oct. 22 1996. http://www.qucis.queensu.ca/home/skill/mini.ps.
[16] Ali-Reza Adl-Tabatabai, Christos Kozyrakis, and BratinSaha. Unlocking concurrency: multicore programming with transactional memory. ACM Queue, (Dec. 2006/Jan. 2007), 2006.
[17] Maurice Herlihy and J. Eliot B. Moss. Transactional memory: Architectural support for lock-free data structures. In Proc. Int. Symp. Computer Architecture, 1993.
[18] J. Rose and G. Steele. C*: an Extended C Language for Data Parallel Programming. Technical Report PL87-5, Thinking Machines Inc., Cambridge, MA, 1987.
[19] Johannes Jendrsczok, Rolf Homann, Patrick Ediger, and Jörg Keller. Implementing APL-like data parallel functions on a GCA machine. In Proc. 21st Workshop Parallel Algorithms and Computing Systems (PARS), 2007.
[20] Ian Foster. Designing and Building Parallel Programs. Addison Wesley, 1995.
[21] Ian Foster. Globus toolkit version 4: Software for service-oriented systems. In Proc. IFIPInt.l Conf. Network and Parallel Computing, LNCS 3779, pages 213. Springer, 2006.
[22] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations.

[23] Ken Kennedy, Charles Koelbel, and Hans Zima. The rise and fall of High Performance Fortran: an historical object lesson. In Proc. Int. Sympos
[24] Beniamino di Martino and Christoph W. Keÿler. Two program comprehension tools for automatic parallelization. IEEE Concurr., 8(1), Spring 2000.
[25] Fredrik Warg. Techniques to reduce thread-level speculation overhead. PhD thesis, Dept. of Computer Science and Engineering, Chalmers university of technology, Gothenburg (Sweden), 2006.
[26] Bruno Bacci, Marco Danelutto, Salvatore Orlando, Susanna Pelagatti, and Marco Vanneschi. P3L: A structured high level programming language and its structured support. Concurrency Pract. Exp., 7(3):225255, 1995.
[27] Susanna Pelagatti. Structured Development of Parallel Programs. Taylor&Francis, 1998.
[28] J. Darlington, A. J. Field, P. G. Harrison, P. H. B. Kelly, D. W. N. Sharp, and Q. Wu. Parallel Programming Using Skeleton Functions. In Proc. Conf. Parallel Architectures and Languages Europe, pages 146160. Springer LNCS 694, 1993.
[29] J. Darlington, Y. Guo, H. W. To, and J. Yang. Parallel skeletons for structured composition. In Proc. 5th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming. ACM Press, July 1995. SIGPLAN Notices 30(8), pp. 1928.
[30] Murray Cole. Bringing skeletons out of the closet: A pragmatic manifesto for skeletal parallel programming. Parallel Computing, 30(3):389406, 2004.
[31] Murray I. Cole. Algorithmic Skeletons: Structured Management of Parallel Computation. Pitman and MIT Press, 1989.
[32] Herbert Kuchen. A skeleton library. In Proc. Euro-Par'02, pages 620629, 2002.
[33] Joel Falcou and Jocelyn Serot. Formal semantics applied to the implementation of a skeleton-based parallel programming library. In Proc. ParCo-2007. IOS press, 2008.
[34] Joseph JáJá. An Introduction to Parallel Algorithms. Addison-Wesley, 1992.
[35] Christoph W. Kessler. A practical access to the theory of parallel algorithms. In Proc. ACM SIGCSE'04 Symposium on Computer Science Education, March 2004.
[36] Allan Gottlieb. An overview of the NYU ultracomputer project. In J.J. Dongarra, editor, Experimental Parallel Computing Architectures, pages 2595. Elsevier Science Publishers, 1987.
[37] FerriAbolhassan, ReinhardDrefenstedt, Jörg Keller, Wolfgang J. Paul, and Dieter Scheerer. On the physical design of PRAMs. Computer J., 36(8):756762, December 1993.
[38] Jörg Keller, Christoph Kessler, and JesperTrä. Practical PRAM Programming. Wiley, New York, 2001.
[39] Wolfgang J. Paul, Peter Bach, Michael Bosch, Jörg Fischer, CédricLichtenau, and JochenRöhrig. Real PRAM programming. In Proc. Int. Euro-Par Conf.'02, August 2002.
[40] Xingzhi Wen and Uzi Vishkin. Pram-on-chip: rst commitment to silicon. In SPAA '07: Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures, pages 301302, New York, NY, USA, 2007. ACM.
[41] MarttiForsell. A scalable high-performance computing solution for networks on chips. IEEE Micro, pages 4655, September 2002.
[42] R. Bisseling. Parallel Scientic Computation A Structured Approach using BSP and MPI. Oxford University Press, 2004.
[43] Classification of the principal programming paradigms. https://www.info.ucl.ac.be/~pvr/paradigmsDIAGRAMeng108.pdf

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.Presently he is working as Associate Professor in JB Institute of Engineering Technology,Hyderabad.

.

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 .

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.