International Journal of Recent Advances in Science and Technology, 2014; 1(1): 12-18

Models for Parallel Computing Review and Perspectives

Nahar hansmukh prayank*, Bhavsar Dharmeshkumar Bhalchandra, Vishal Bhatnagar, Richa Tomer and S.K. Agarwal

Department of Computer Science, Shri Venkateshwara University, Gajraula Distt- Amroha (U.P), India

ABSTRACT

Today's confederacy of computers parallelism has now become insidious, so to endow an investigation of the models seems important for parallel computation. Parallel computing models are specifically subject of attentiveness with certain matter-of-fact appliance with a perspective of possible expectations applicability. This parallel computing model paper also describes feat in language of programming and means.

Keywords: Parallel Cost Model, Parallel Computational Model, Parallel Programming Language.

Introduction

Parallel computing is fundamentally an appearance of computation in which many calculations are carried out all unitedly. Multiple hardware threads (multithreading and multiple processor cores) implement performance processors. Multiple threads are the smaller unit of large problems which can often be divided and then solved concurrently. Limited credulous in applications of instruction level parallelism is the power consumptions (consequently heat generation) has become a concern any further due to power consumption and heat exertion and the processor clock frequency which cannot be augmented, , and there is no uncertainty that our requirements and prospect of machine presentation will increase further if performance improvement in processor is required. In the desktop and embedded both parallel programming will actually apprehend the widely held of application and system programmers in the predictable future.

Parallel computing programming model and a corresponding outlay model are the elements of parallel computation model.

**Correspondence*
Nahar Hashmukh Prayank
Department of Computer Science, Shri Venkateshwara University, Gajraula Distt- Amroha (U.P), India

A nonrepresentational parallel machine by its operations (arithmetic operations, spawning , reading from and writing to shared memory, or sending and receiving messages), the constraints of when and where these can be functional, their possessions on the state of the computation , and how they can be self-possessed is described by a parallel programming model.

A parallel programming model also contains, shared memory programming models, a memory model that describes how and when a single memory can be accesses by different parts of a parallel computer processors. A cost (describes resource occupation and parallel execution time) is associated with parallel cost model, and describes prediction of accumulated cost of composed operations up to entire parallel programs with each basic operation.

There are several computing parallel programming models in which there is a unpredictability to sequential programming, the Von Neumann model is the initial leading programming model (data flow and declarative programming). Details of the fundamental hardware programming models are nonfigurative to some scale, in which a wider range of parallel programs languages and systems portability of parallel algorithms increases.

In this paper a theoretical and practical view both are been offered by a brief survey of parallel programming models and observation on their merits and perspectives. Basic references are articles and books in the literature such as by Henri[7],

Skillicorn[44], Giloi[24], Maggs[38], Skillicorn and Talia[45,46], Lengauer[35], and Leopold[36].

Survey on Models

In the survey on models two elementary issues in parallel program execution are found that occur in implementations of several models.

Different Ways of Parallel Execution

There are many dissimilar parallel execution techniques, which describe different ways from any programmer's views for execution of parallel programs when they are created or terminated. There are two most well-known ways SIMD style and fork-join parallel execution.

In Fork join style of parallel execution performance with dynamism at certain fork points in the program, in which a job is divided into N servers, after service, sub-jobs have also been processed. Then sub-jobs have also been processed. From commencement and the closing stages of program execution, only one action is executing, but the number of parallel activities is able to show a discrepancy significantly throughout execution and in that way become accustomed to the presently on hand parallelism. The activity representing on physical processors requirements within the track is be finished by a thread package or by the language's run-time system at run time by the operating system. In contrast to fork-join style execution, programmer is responsible for the representing of parallel execution.

In SIMD (Single Instruction, Multiple Data) style execution at the beginning of program execution (ingress to main) creates a constant number q of parallel activities (physical or virtual processors), and no new parallel activities can be spawned, that is this number will be kept constant throughout program execution.

Consequently, when the dynamic scheduling is automatically provided in the fork-join style programmer has the conscientiousness for stack consideration.

Nested parallelism can also be achieved under SIMD way of execution, if a collection of n processors are divided into m subgroups of n_i processors each, where $\sum n_i \leq n$. Subtask in parallel are taken care by each subgroup. When all subgroups are completed by means of their subtask they are discarded and the parent group resumes execution. Group splitting can be nested, and the group hierarchy forms a tree structure, with the leaf groups being the currently active ones at any time during program execution.

Parallel Random Access Machine (PRAM)

The Parallel Random Access Machine (PRAM) model is a shared-memory abstract machine extension of the Random Access Machine (RAM) and was proposed by Fortune and Wyllie [20], this model is used in the design and analysis of sequential algorithms. The PRAM supposes shared memory to be connected to a particular set of processors. Both processors and memory feed by a global clock and execution of any instruction consumes exactly one unit of time, which is autonomous of the processor execution. There is no limitation on the number of processors for simultaneous access to shared memory.

The PRAM memory model is known for strict consistency [3], which says that a write process in clock cycle *c* becomes visible globally to all processors at the beginning of clock cycle *c+1*.

In the same clock cycle, the effect of multiple processors writing or reading the same memory location is determined by the PRAM model.

Realistic Application

It supports deterministic parallel computation is the unique property of PRAM model, and it is one of the most programmer friendly models available. There are abundant algorithm that have been developed for the PRAM model JaJa[29] (most basic model for parallel algorithms[32]) and focuses on neat parallelism only.

Implementations

Hardware techniques multithreading and smart combining networks are cost effective realization of PRAMs which is possible using, such as the NYU Ultracomputer [25], SBPRAM by Wolfgang Paul's group in Saarbrucken [1,30,40], XMT by Vishkin [49], and ECLIPSE by Forsell [19]. PRAM is is completely insensitive to data locality and focuses on parallelism.

The parallel algorithms hypothesis community have been proposed variants of PRAM model such as the hierarchical PRAM, asynchronous PRAM [13, 23], the distributed PRAM (DRAM) ,the block PRAM [4], and the queuing PRAM (Q PRAM),to name a few.

Unrestricted Message Passing

Message passing multicomputer are distributed memory machine which consists of a number of RAMs that run asynchronously and communicate via messages sent over a communication network. Generally message routing is performed by the network, so that a processor can send a message to any other processor without consideration of the particular network structure. Send and receive commands can be whichever blocking (processors get synchronized), or none blocking (the sending processor). The message passing subsystem forwards the message to the receiving processor and buffers it there until the receiving processor executes the receive command. Group of processors that involve more complex forms of communication, are called collective communication operations such as broadcast, multicast, or reduction operations.

Message passing multicomputer of cost model consists of two parts. The operations performed treated as in a RAM. Non blocking communications point to point are modelled by the *LogN* model [14]. The latency N specifies the time that a message requires one word to be transmitted from sender to receiver. The visual projection k specifies the time that the sending processor is occupied in executing the send command. The time that must pass between two successive send operations of a processor is given by gap *g*, and thus models the processor's bandwidth to the communication network. The processor count *P* gives the number of processors in the machine. The LogN model has been extended to the LogGN model [5], by introducing bandwidth for longer messages parameter G.

Realistic Application: CSP (Communicating Sequential Processes) is one of the message passing models that have been used in the hypothesis of concurrent and distributed systems from several years. By means of the explanation of hawker autonomous sagacious ephemeral libraries, message passing became the overriding programming style on huge parallel computers systems.

Implementations: In the early 1990s portable message passing libraries such as PVM and MPI were replaced by vendor specific libraries. Later MPI was extended in the MPI 2.0 standard (1997) by fork join style and one sided communication. FORTRAN, C and C++ have been defined by MPI interfaces.

Bulk Synchronous Parallelism

This model was proposed by Valiant in 1990 [48] and modified by McColl [39], a structure of message passing computations as a progression of obstruction unconnected super steps, where each super step consists of a computation stage operating on local variables only, followed by a global interprocessor communication phase. Only three parameters are involved in the cost model (number of processors *p*, point to point network bandwidth *g*, and message latency), only if the maximum local work for each processor and the maximum communication volume for each processor are identified. By summing up the costs of all executed super steps then cost for a program is then simply determined.

Realistic Application

Reasonable predictions of execution time to guide algorithmic design decisions and balance trade-offs are derived using BSP model.

Implementations

For an SIMD execution style the BSP model is mainly realized in the form of libraries such as BSPlib [27] or PUB [9].

Partitioned Global Address Space and Asynchronous Shared Memory

A number of threads for execution have access to a general memory in the shared memory model, the threads of execution run asynchronously.

A current improvement is transactional memory ([26], [2]), which is adopted by the concept of transaction known from database programming. A sequential code section enclosed in a statement which fail completely or performs completely to share memory as an infinitesimal operation is called a transaction.

Realistic Application

Programming for small scale parallel computers, shared memory programming has become the leading form, particularly SMP systems. SMP nodes, shared memory programming has been combined with message passing concepts to consist clusters of large scale parallel computers.

Implementations

A shared memory parallel language for algorithmic multithreading is Cilk [8].
With the arrival of multicore processors OpenMP is gaining popularity and may finally substitute Pthreads absolutely. Structured parallelism in a combination of SIMD and fork join styles are provided by OpenMP. Shared memory via the concept of tuple spaces, which is much more nonrepresentational than one dimensional addressing, partially resembles the access to a relational database provided by the Linda system [10].

Data Parallel Models

SIMD and vector computing are included by Data parallel models, data parallel computing, systolic computing cellular automata, stream data processing, and VLIW computing.
The component wise application of the same scalar computation to several elements of one or several operand vectors, creating a result vector are involved by data parallel computing. All element computations must be autonomous of each other, and may therefore be executed in a pipelined way, or any order in parallel.

Realistic Application
An early vector super computer in the 1990s and 1980s were based on the paradigm of Vector computing and is still a necessary ingredient of contemporary elevated performance computer architectures. It is a special case of the SIMD computing paradigm. For the most part contemporary elevated end processors include vector units extending their instruction set by SIMD/vector operations. In high performance processors for the digital signal processing (DSP) arena, VLIW in today also a popular concept.

Implementations
APL [28] is an early SIMD programming language. Some SIMD languages are Vector-C [37] and C* [43]. Vector computing and even a simple form of data parallelism are supported by Fortran 90. It became a full-fledged data parallel language with the HPF [31] extensions. ZPL [47] NESL Data parallel C and Modula-2* [42] are included by other data parallel languages.

Models of Parallel-Task and Graphs of Job

Every program or applications may be well thought-out as a collection of jobs where each job is solving part of the problem. Jobs may also communicate with each other during their execution or existence, or may recognize inputs only as per requirement to their start off, and transmit results to other jobs only while they will be terminated. Jobs may generate other jobs in a fork-join way, and this may be done even in a forceful and data reliant approach. Collections of such jobs possibly will be represented by a job graph, where nodes represent jobs and curved like parts represent data inter-dependencies.

Realistic Application
In the most recent years grid computing has gained substantial desirability, mostly motivated next to the vast computing control necessary to get to the bottom of impressive face up to inconvenience in ordinary and life sciences. The integration of reconfigurable hardware with microprocessors on single chips has gained some interest using hardware with software.

Implementations
The MIT Alewife mechanism with the ID functional programming language [3] is a well-known instance intended for parallel data flow computation. There are rather a lot of grid middleware's, most significantly Globus [22] and Unicore [17].

Methodologies of General Parallel Programming

In this section, a detailed property of extensively used approaches to the parallel software has been revised.

An existing sequential program is actually start for the same problem, which is additional constrained of exceedingly high consequence for software industry that has to port a host of inheritance code to parallel platforms in these existence.

PCAM Method of Foster

The design of a parallel program suggested by Foster [21] that have to begin from an active (sequential if possible) algorithmic elucidation to a computational trouble by dividing it into various small jobs and identifying inter-dependencies involving these, that possibly will outcome in communication and synchronization. Partitioning and communication are first two design phases for a model in which there is no restriction on the number of processors. To decrease inner communication and synchronization associations a comprehensive job to local memory accesses, the

jobs are agglomerated to comprehensive jobs. finally, the comprehensive jobs are programmed to substantial processors to balance load and additional communication.

Parallelization Increment

In many technical programs most of the execution time is spent in a fairly small part of the code. HPF and OpenMP are some instruction based parallel programming languages, FORTRAN and C are considered as a semantically expandable for sequential maintained language allow initialisation from sequential source code to parallelize incrementally. Frequently, the most concentrated interior loops are recognized and parallelized first by inserting commands.

Autonomous Parallelization

Autonomous parallelization has high importance to industry of sequential inheritance code but it is rather difficult. There are two forms: static parallelization and run time parallelization.

Library-Based Parallel Programming and Skeleton Based Programming

Skeleton programming [12, 41] is structured parallel programming which restricts the compositions of only a few, predefined patterns, which provide many ways of expressing parallelism, called skeletons. Nonspecific, transferable, and reusable essential program building blocks for which parallel implementations may be presented are skeletons [12, 15]. Skeletons are derived from advanced command functions as known from functional programming languages. P3L [6, 41], SCL [15, 16], eSkel [11], MuesLi [34], or QUAFF [18], are skeleton based parallel programming system that usually provides a moderately small, fixed set of skeletons. Each skeleton is an illustration of a exclusive way of using parallelism, in a particularly well thought-out type of computation such as parallel divide and conquer, data parallelism, job farming, or pipelining.

Conclusion

The re-rating of parallel programming models provides assumption about the future of parallel programming models and surfaces the existing trends. Physical and technical necessity will make the future of parallel computing. By combining hardware multi cores, multithreading, SIMD units, accelerators and on chip communication systems parallel computer architectures will be more advance, which increase the requirement for the programmer and the compiler to ask for parallelism, coordinate computations and handle data position in order to achieve efficient performance e.g. the Cell BE processor. Parallel computing is relatively simple, purely sequential languages will stay on for definite applications that are not performance decisive, applications which are not performance decisive such as word processors. Aspect oriented and view based programming and model driven developments are new software engineering techniques that may assist in complexity management.

Tools that allow to more or less automatically port sequential inheritance software are of very high significance. Useful models are deterministic and time predictable parallel. With the beginning of new parallel language, compilers and tools technology must maintain swiftness. If compilers are impulsive at beginning and generate poor code then the most advanced parallel programming language is destined to failure, as possibly was observed in the 1990s where HPC programmers instead switched to the lower level MPI as their main programming model for HPF in the high performance computing arena [31].

References

1. Ferri Abolhassan, Reinhard Drefenstedt, Jorg Keller, Wolfgang J. Paul, and Dieter Scheerer. On the physical design of PRAMs. Computer J, 1993;36(8):756-762.
2. Ali-Reza Adl-Tabatabai, Christos Kozyrakis, and Bratin Saha. Unlocking concurrency: multicore programming with transactional memory. ACMQueue, (Dec. 2006/jan. 2007), 2006.
3. Anant Agarwal, Ricardo Bianchini, The MIT Alewife machine: Architecture and performance. In Proc. 22nd Int. Symp. Computer Architecture, pages 2-13, 1995.
4. A Aggarwal. A. K.Chandra, and M.Snir.Communication complexity of PRAMs. Theoretical Computer Science, 71:3-28, 1990.
5. Albert Alexandrov, Mihai F. Ionescu, Klaus E. Schauser, and Chris Scheiman. LogGP: Incorporating long messages into the LogP model for parallel computation. Journal of Parallel and Distributed Computing, 44(1):71-79, 1997.

6. 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):225-225, 1995.

7. Henri E. Bal, Jennifer G. Steiner, and Andrew S. Tanenbaum. Programming Languages for Distributed Computing Systems. ACM Computing Surveys 21(3):261-322, September1989.

8. Robert D. Blumofe, Christopher F. Joerg, Bradley C. Kuszmaul. Cilk: an efficient multi-threaded run-time system. In Proc. 5th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming, pages 207-216, 1995.

9. Olaf Bonorden, Ben Juurlink, Ingo von Otte, and Ingo Rieping. The Paderborn University BSP (PUB) Library. Parallel Computing, 29:187-207, 2003.

10. Nicholas Carriero and David Gelernter. Linda in context. Commun. ACM, 32(4):444-458, 1989.

11. Murray Cole. Bringing skeletons out of the closet: A pragmatic manifesto for skeletal parallel programming. Parallel Computing, 30(3):389-406, 2004.

12. Murray I. Cole. Algorithmic Skeletons: Structured Management of Parallel Computation. Pitman and MIT Press, 1989.

13. Richard Cole and Ofer Zajicek. The APRAM: Incorporating Asynchrony into the PRAM model. In Proc. 1st Annual ACM Symp. Parallel Algorithms and Architectures, pages 169-178, 1989.

14. David E. Culler, Richard M. Karp, David A. Patterson, Abhijit Sahay, Klaus E. Schause. LogP: Towards a realistic model of parallel computation. In Principles & Practice of Parallel Programming pages 1-12, 1993.

15. J. Darlington, A. J. Field, P. G. Harrison. Parallel Programming Using Skeleton Functions. In Proc. Conf. Parallel Architectures and Languages Europe, pages 146-160. Springer LNCS 694, 1993.

16. 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 38(3), pp. 19-28.

17. Dietmar W. Erwin and David F. Snelling. Unicore: A grid computing environment. In Proc. 7th Int.l Conference on Parallel Processing (Euro Par), pages 825-834, London, UK, 2001. Springer-Verlag.

18. Joel Falcou and Jocelyn Serot. schematic semantics applied to the implementation of a skeleton based parallel programming library. In Proc. ParCo-2007. IOS press, 2008.

19. Martti Forsell. A scalable high performance computing solution for networks on chips. IEEE Micro, pages 46-55, September 2002.

20. S. Fortune and J. Wyllie Parallelism in random access machines. In Proc. 10th Annual ACM Symp. hypothesis of Computing, pages 114-118, 1978.

21. Ian Foster. Designing and Building Parallel Programs. Addison Wesley, 1995.

22. Ian Foster. Globus toolkit version 4: Software for service oriented systems. In Proc. IFIP Int.l Conf. Network and Parallel Computing, LNCS 3779, pages 2-13, Springer, 2006.

23. Phillip B. Gibbons, A. More Practical PRAM Model. In Proc. 1st Annual ACM Symp. Parallel Algorithms and Architectures, pages 158-168, 1989.

24. 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.

25. Allan Gottlieb. An overview of the NYU ultracomputer project. In J. J. Dongarra, editor, Experimental Parallel Computing Architectures, pages 25-95. Elsevier Science Publishers, 1987.

26. Maurice Herlihy and J. Eliot B. Moss. Transactional memory Architectural support for lock free data structures. In Proc. Int. Symp. Computer Architecture, 1993.

27. Jonathan M. D. Hill, Bill McColl. BSPlib: the BSP Programming Library. Parallel Computing, 24(14): 1947-1980, 1998.

28. Kenneth E. Iverson. A Programming Language. Wiley, New York, 1962.

29. Joseph JaJa. An Introduction to Parallel Algorithms. Addison Wesley, 1992.

30. Jorg Keller, Christoph Kessler, and Jesper Traff. Practical PRAM Programming. Wiley, New York, 2001.

31. Ken Kennedy, Charles Koelbel, and Hans Zima. The rise and fall of High Performance Fortran: an historical article lesson. In Proc. Int. Symposium on the History of Programming Languages (HOPL III) June, 2007.

32. Christoph W. Kessler. A practical access to the hypothesis of parallel algorithms. In Proc. ACM SIGCSE'04 Symposium on Computer Science Education, March 2004.

33. Christoph W. KeBler and Helmut Seidl. The Fork95 Parallel Programming Language: Design, Implementation, Application. Int. J. Parallel Programming, 24(1):17-50, February 1997.

34. Herbert Kuchen, A skeleton library. In. Proc. Euro Par'02, pages 620-629, 2002.

35. Christian Lengauer. A personal, historical perspective of parallel programming for high performance. In Gunter Hommel, editor, Communication Based Systems (CBS 2000), pages 111-118, Kluwer, 2000.

36. Claudia Leopold. Parallel and Distributed Computing. A survey of models, paradigms and approaches. Wiley, New York, 2000.

37. K. C. Li and H. Schwetman. Vector C: A Vector Processing Language. J. Parallel and Distrib. Comput., 2:132-169, 1985.

38. B. M. Maggs, L. R. Matheson, and R. E. Tarjan. Models of Parallel Computation: a Survey and Synthesis. In Proc. 28th Annual Hawaii Int. Conf. System Sciences, volume 2, pages 61-70, January 1995.

39. 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.

40. Wolfgang J. Paul, Peter Bach, Michael Bosch.Real PRAM programming. In Proc. Int. Euro Par Conf.'02, August 2002.

41. Susanna Pelagatti. Structured Development of Parallel Programs. Taylor Francis, 1998.

42. Michael Philippsen and Walter F. Tichy Modula and its Compilation. In Proc. 1st Int. Conf. of the Austrian Center for Parallel Computation, pages 169-183. Springer LNCS 591, 1991.

43. J. Rose and G. Steele. C*: an Extended C Language for Data Parallel Programming. Technical Report PL87-5, Thinking Machines Inc., Cambridge, MA, 1987.

44. D. B. Skillicorn. Models for Practical Parallel Computation. Int. J. Parallel Programming, 20(2):133-138, 1991.

45. David B. Skillicorn and Domenico Talia, editors. Programming Languages for Parallel Processing. IEEE Computer Society Press, 1995.

46. David B. Skillicorn and Domenico Talia. Models and Languages for Parallel Computation. ACM Computing Surveys, June 1998.

47. Lawrence Snyder. The design and development of ZP.L In Proc. ACM SIGPLAN Third symposium on history of programming languages (HOPL III). ACM Press, June 2007.

48. Leslie G. Valiant. A Bridging Model for Parallel Computation. Comm. ACM, 33(8):103-111, August 1990.

49. Xingzhi Wen and Uzi Vishkin. Pram-on-chip: first commitment to silicon. In SPAA '07: Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures, pages 301-302, New York, NY, USA, 2007.

Source of Support: NIL
Conflict of Interest: None