Chapter 2
Parallel Environments

Bhabani Shankar Prasad Mishra and Santwana Sagnika

Abstract To cater to the increasing demand for high-speed and data-intensive computing in the current scenario, a widely followed approach has been the development of parallel computing techniques, which enables simultaneous processing of a huge amount of data. By implementing such mechanisms, the total execution time is greatly reduced, and the available resources are utilized in a most efficient manner. This chapter provides a bird's-eye view of the various parallel processing models available in literature and their distinctive features.

Keywords Parallel processing · Message passing · Object-oriented · Shared memory · Partitioned global address space · Algorithm skeleton · Bulk synchronous programming

2.1 Introduction

The increasing dependence on automated systems has widened the scope of computers to work in various fields and solve large complex problems. This involves dealing with large datasets. Even so, performing the necessary tasks in a minimum time-frame is also of paramount importance. Hence, a solution for providing high speed computation of huge amount of data comes in the form of parallel processing. The advent of parallel processing has expanded the application of computers to areas like weather forecast, image processing, military applications, aviation, etc.

To provide parallelism, a possible approach can be to increase the number and power of processors. This may not always be feasible, considering the high costs involved. Practical solutions for parallelism can be implemented by following certain models, as well as programming tools that match the underlying parallel models. Parallelism is achieved at the basic level by two mechanisms data-parallelism and

B.S.P. Mishra (✉) · S. Sagnika
School of Computer Engineering, KIIT University, Bhubaneswar, Odisha, India
e-mail: mishra.bsp@gmail.com

S. Sagnika
e-mail: santwana.sagnika@gmail.com

B.S.P. Mishra et al. (eds.), *Techniques and Environments for Big Data Analysis*,
Studies in Big Data 17, DOI 10.1007/978-3-319-27520-8_2

control parallelism. Data parallelism involves dividing the data into smaller sets and running the same set of instructions on all of them over different processors. On the other hand, in control parallelism the instructions are divided over various processors that execute concurrently. The parallel programming tools rely on either or both mechanisms [12, 13].

For parallel processing, the hardware requirements can be described as follows.

Multi-core systems—In such a system, a single chip contains a number of cores which can operate independently. The memory can be shared. The throughput of the system can be multiplied by the number of processors.

Multi-processor systems—In this case, multiple processors collaborate to perform a parallel task. This can be either (i) Symmetric Multi-Processor (SMP) and (ii) Non-Uniform Memory Access (NUMA). The memory can either be shared or distributed. Communication occurs through high-speed bus. Every processor has a control unit.

Cluster systems—A cluster represents a group of nodes (desktops, servers and workstations) connected over a network. Every node is a self-sufficient system and is able to parallelize computation as well as access to memory. A cluster provides good scalability, and hence, better bandwidth and storage capacity.

Parallel programming models comprise of a combination of mechanisms that implement parallelism as suited to the system that they run on. A parallel programming model and its analogous cost model together forms a parallel model of computation. The parallel programming model simulates a virtual parallel machine that performs basic operations and also includes a memory model that controls visibility of memory to different modules of the parallel system. This model encompasses the necessary frameworks or languages that implement the said model. Besides, the parallel cost model associates specific costs to the various basic operations and also provides mechanisms to estimate the overall costs for the complete task executing in parallel [4].

2.2 Commonly Used Models

The most popular parallel models are discussed here.

2.2.1 Message Passing Models

In this model, a set of tasks perform computation using their local memory. Multiple tasks reside on the same or across multiple machines. Some amount of programming is needed to use multiple nodes for a job to attain faster execution. The various processors executing a particular job can communicate over the network. For writing such programs, certain language-free protocols are used, which have their own implementations using various languages. These are added as routines to the programs. Figure 2.1 shows a sample message passing model architecture [2].

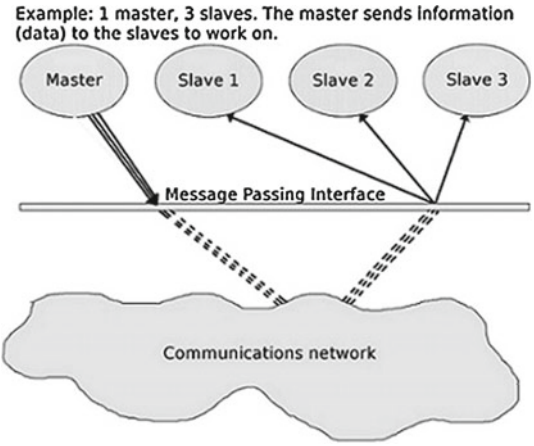

Fig. 2.1 Message passing model architecture

Various available message passing models are as described below [9, 14].

Model	Features
Message Passing Interface (MPI)	• Language-free communication protocol • Group of processes connected by a Communicator • Performs point-to-point communication using MPI_SEND and MPI_RECV • Runs in three modes—ready, standard and synchronous • Collective/group communication via MPI_Bcast • Uses pre-defined datatypes like MPI_CHAR, MPI_INT, etc • Doesn't define fault tolerance
Parallel Virtual Machine (PVM)	• Appropriate for coarse-grained parallelism • Library modules provide control, locks, message passing and broadcasting • Logical names assigned to executable files • Finds versions suitable for compilation on a specific machine
ClusterM	• Works on heterogeneous environments • Partitions programs and assigns them to parallel machines • Maintains links using trees • Forms clusters of correlated nodes • Concurrent executions of same level clusters is possible
MATLAB MPI	• MPI implemented through MATLAB scripts • Performs coarse-grained parallelism • Follows MPI standards • Realizes global semantics of data distribution & communication

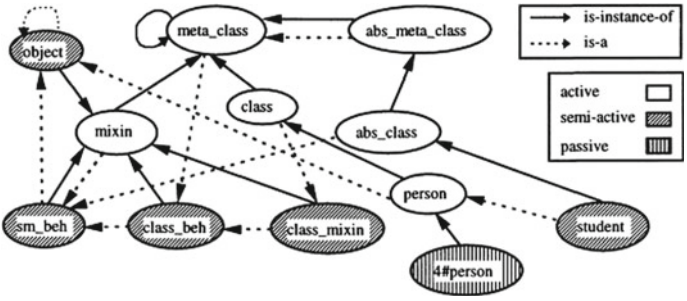

Fig. 2.2 Parallel object-oriented model architecture

2.2.2 Parallel Object-Oriented Models

This model provides software engineering-based methods and abstractions for designing applications. Objects are designed encapsulating their internal states. This model can be considered as a group of shared objects, just like a shared data model. It is considered as a potential technique for parallel models, even though the current inclination of programmers towards traditional programming languages can be an obstacle for its wide acceptance. Some compatible environments like Mentat and PC++ have been discussed in existing literature [17]. Figure 2.2 shows a parallel object-oriented model architecture.

Various parallel object-oriented models are described below [10, 13].

Model	Features
PC++	• Exploits medium-to-coarse grained parallelism • Messages are used for communication • Compiler provides synchronization
Jade	• Works on coarse-grained parallelism • Dynamic extraction from a serial program • Analysis of dependencies and scheduling done at runtime
p-CORBA	• Provides concurrency to standard CORBA architecture • Adds load-balancing feature
Mentat	• Supports messaging between objects • Auto-management by the compiler • Medium-to-coarse grained parallelism
ABCL/1	• Communication though message queues • Validity of messages defined by script construct • Based on the Actor model
Charm++	• Distributes parallelization task between programmer and system • Programmer indicates parallelism • System performs workload distribution • Supports both data and control parallelism • Implements medium-to-coarse grained parallelism

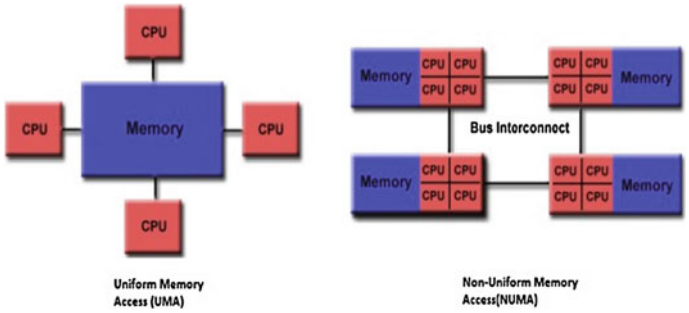

Fig. 2.3 Shared memory programming model architecture

2.2.3 Shared Memory Programming Models

In this model, the program is made to run on one or more number of processors, that share a common memory space that can be used to write and read data in asynchronous manner. Concurrent access control mechanisms are required to be implemented. It can utilize Unified Memory Access (UMA) and Non-Uniform Memory Access (NUMA). Data ownership norms are not strict. Program development can be made simple but locality management issues become difficult [1, 5]. Figure 2.3 shows the shared memory programming architecture.

Various shared memory models are discussed below [13, 16].

Model	Features
Threads	• Threads provide faster switching • POSIX threads (Pthreads) provide programming standards • Provide synchronization, control and scheduling • Communicate through the global memory
ADA 9X	• Stored shared code and data in partitions • Partitions interact through RPCs
Delirium	• Coordinating language that uses FORTRAN and C • Medium-to-coarse grained parallelism support • Access to local data faster than remote data
Global Array	• Asynchronous access to blocks of multi-dimensional arrays • Base for current Global Address Space (GAS) languages • Access to local data faster than remote data
Open MP	• Uses fork-join multi-threading • Compiler and library functions control the working • Task allocation through work-sharing constructs
High-performance FORTRAN	• Library functions handle parallelism • Medium-to-fine grained parallelism

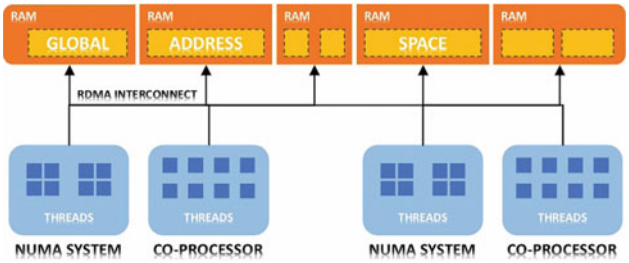

Fig. 2.4 PGAS model architecture

2.2.4 Partitioned Global Address Space Models

Partitioned Global Address Space (PGAS) assumes a logically partitioned memory space in which every partitioned is allocated as local to a processor. The sections of the shared memory space exploit locality of reference and have preference towards particular processes. This model tries to merge SPMD programming for distributed memory systems and the data referential semantics for shared memory systems. This helps to model the hardware-specific locality of data [6, 8]. Figure 2.4 shows the PGAS model architecture. Various PGAS models are discussed as follows [13, 17].

Model	Features
Unified Parallel C (UPC)	• Uses Single Program Multiple Data model • Parallelism amount fixed from startup • Runs on both shared and distributed memory hardware
X10	• Divides computations into places holding data and related activities • Supports user-defined struct types • Constrained structure for object-oriented programming
Chapel	• Multi-threaded model • High-level abstraction for parallelism • Generic programming features and support for code reuse • Portable model suitable for clusters

2.2.5 Algorithm Skeleton Models

Algorithmic skeletons represent a higher level programming approach that uses general programming patterns that abstract the complexity of parallel systems, by building complex patterns from more basic ones. The skeleton optimizes the structure to achieve good performance and better adaptation to external environments. After definition of the structure, the programmer implements the functional aspects by writing what is known as *muscle codes*. Figure 2.5 shows an algorithm skeleton architecture [3].

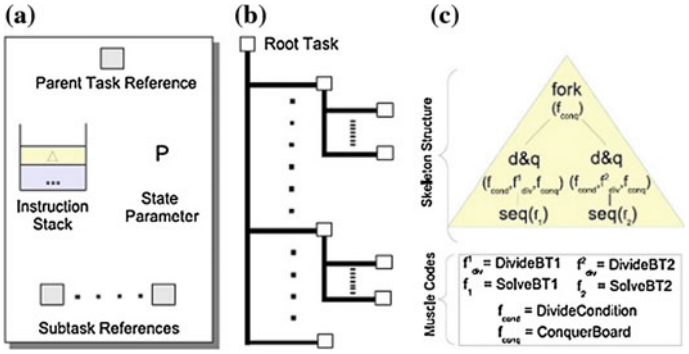

Fig. 2.5 Algorithm skeleton model architecture. **a** Task definition, **b** task tree, **c** N-Queens skeleton

Different algorithm skeleton models are described as follows.

Model	Features
JaSkel	• Framework based on Java • Implementable on grid and cluster structures • Provides concurrent, sequential and dynamic skeletons
Calcium	• Provides a type system for nesting of skeletons • Performance tuning model for debugging • Transparent file access model
ASSIST	• Sequential and parallel models • Performance portability • High-level programmable software • High reusability
Skeleton-BAsed Scientific COmponents (SBASCO)	• Suitable for numerical applications • Customized component language
Higher-Order Divide and Conquer (HDC)	• Polymorphic functions linkable to skeletons • Estimates ratio of tasks to processors

2.2.6 Bulk Synchronous Programming Models

A BSP model consists of a distributed memory architecture, an algorithmic framework, a cost calculation model and a barrier synchronization mechanism. In this model, a master co-ordinates a group of workers. The execution takes place in a lock-step manner. Each worker reads its allocated data from a queue and performs specific processing on it. Then it sends its own result to the communication channel. This process is repeated till there are no more active workers remaining [15]. Figure 2.6 shows a BSP model architecture. Some BSP models are discussed as follows [7, 11, 18].

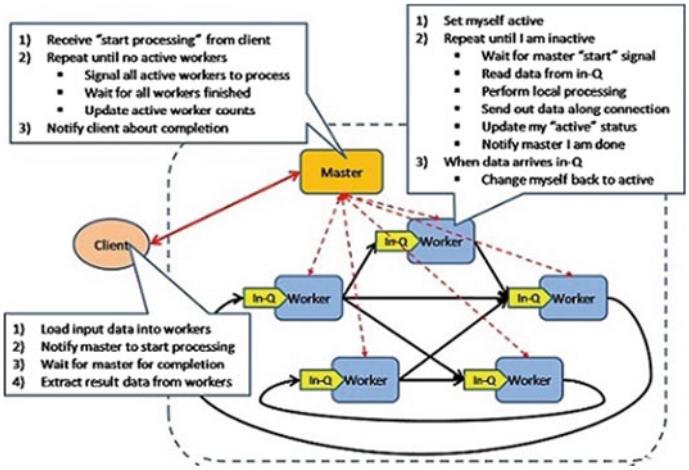

Fig. 2.6 BSP model architecture

Model	Features
Bulk Synchronous Parallel ML (BSML)	• Follows data-parallel mechanism • Collective communications • Avoids deadlocks • Based on primitives accessing physical parameters of the machine
BSPLib	• Small library with 19 operations • Provides data parallelism in Single Program Multiple Data manner • Follows Direct Remote Memory Access
Apache Hama	• Runs on very large datasets in Hadoop Distributed File System • Easier and flexible than traditional message passing models • Speeds up the iteration processes
Pregel	• Works on parallelization of graph algorithms • Vertex-centric approach with message interchange • Synchronous and deadlock-free
BSPonMPI	• Runs on MPI enabled platforms • Uses communication procedures of MPI • Uses request and delivery buffers for communication
Multicore BSP	• Designed for C programming on ANSI C99 standard • Extends BSPLib by adding high-performance primitives • Thread-based parallelization • Suited for shared memory structure

2.3 Conclusion

Parallel computation is an efficient mechanism to implement feasible solutions for complicated and large problems. Such problems are categorized according to the way they communicate data, i.e. either by accessing a shared memory or by passing messages. The selection of the mechanism depends upon the implementation architecture as well as the features of the problem. A suitable language needs to be chosen to specify the partitioning details. The design of the parallel algorithm needs to be done according the environment, i.e. hardware and software on which it needs to be implemented. Parallelism has gained popularity in recent times due to the increasing availability of appropriate machines. This chapter has discussed various characteristics of parallel algorithms and the tools available for their implementation.

References

1. A parallel programming model. http://www.mcs.anl.gov/~itf/dbpp/text/node9.html
2. Beginner guide—www.hpc2n.umu.se. https://www.hpc2n.umu.se/support/beginners_guide
3. Caromel, D., & Leyton, M. (2007). Fine tuning algorithmic skeletons. In A.-M. Kermarrec, L. Bougé, & T. Priol (Eds.), *Euro-Par 2007 Parallel Processing* (Vol. 4641, pp. 72–81)., LNCS Berlin: Springer.
4. Chandrasekaran, K. Analysis of different parallel programming models. Indiana University Bloomington.
5. Demmel, J., & Yelick, K. Shared memory programming: Threads and OpenMP. http://www.cs.berkeley.edu/~demmel/cs267_Spr11/
6. Diaz, J., Muoz-Caro, C., & Nio, A. (2012). A survey of parallel programming models and tools in the multi and many-core era. *IEEE Transactions on Parallel & Distributed Systems, 23*(8), 1369–1386.
7. Gesbert, L., Gava, F., Loulergue, F., & Dabrowski, F. (2007). *Bulk synchronous parallel ML with exceptions* (pp. 33–42). In P. Kacsuk, T. Fahringer, Z. & Németh (Eds.), *Distributed and Parallel Systems*. New York: Springer.
8. Hudak, D. E. Introduction to the Partitioned Global Address Space (PGAS) programming model. https://www.osc.edu/sites/osc.edu/files/staff_files/dhudak/pgas-tutorial.pdf
9. IBM. Big data at the speed of business, February 2015. http://www-01.ibm.com/software/data/bigdata/. Accessed Feb 19, 2015.
10. Kessler, C., & Keller, J. (2007). Models for parallel computing: Review and perspectives. *PARS Mitteilungen*, *24*, 13–29.
11. Malewicz, G., Austern, M. H., Bik, A. J. C., Dehnert, J. C., Horn, I., Leiser, N., & Czajkowski, G. (2010). Pregel: A system for large-scale graph processing. In *ACM SIGMOD10* (pp. 135–145).
12. Mishra, B. S. P., Dehuri, S., Mall, R., & Ghosh, A. (2011). Parallel single and multiple objectives genetic algorithms: A survey. *International Journal of Applied Evolutionary Computation*, *2*(2), 21–58.
13. Mishra, B. S. P., & Dehuri, S. (2011). Parallel computing environments: A review. *IETE Technical Review*, *28*(3), 155–162.
14. Mivule, K., Harvey, B., Cobb, C., & Sayed, H. E. (2014). A review of CUDA, mapreduce, and pthreads parallel computing models. *CoRR*, 1–10.
15. Pragmatic Programming Techniques—Atom. http://horicky.blogspot.in/2010/10/scalable-system-design-patterns.html

16. Sharma, M., & Soni, P. (2014). Comparative study of parallel programming models to compute complex algorithm. *International Journal of Computer Applications*, *96*(19), 9–12.
17. Silva, L. M., & Buyya, R. (1999). Parallel programming models and paradigms. *High Performance Cluster Computing: Architectures and Systems*, *2*, 4–27.
18. Yzelman, A. N., Bisseling, R. H., Roose, D., & Meerbergen, K. (2013). MulticoreBSP for C: A high-performance library for shared-memory parallel programming. Technical report TW 624, KU Leuven, pp. 1–15.

Springer

http://www.springer.com/978-3-319-27518-5

Techniques and Environments for Big Data Analysis
Parallel, Cloud, and Grid Computing
Mishra, B.S.P.; Dehuri, S.; Kim, E.; Wang, G.-N. (Eds.)
2016, XI, 191 p. 103 illus., 76 illus. in color., Hardcover
ISBN: 978-3-319-27518-5