2018 5th International Conference on Information Science and Control Engineering

Parallel Computing: Review and Perspective

Yuxiang Li
School of Information Engineering
Henan University of Science and Technology
Luoyang, China
liyuxiang@haust.edu.cn

Zhiyong Zhang
School of Information Engineering
Henan University of Science and Technology
Luoyang, China
xidianzzy@126.com

***Abstract*—As parallelism on different levels becomes ubiquitous in today′s computers, it seems worthwhile to provide a review of the wealth of every aspect of parallel computing that has evolved over the last decades. We refrain from a comprehensive survey and concentrate on parallel programming patterns, design for parallel program, parallel programming models, parallel programming languages, design of parallel algorithms, together with a perspective of parallel computing. Besides presenting the patterns, models, design frameworks, we also refer to languages, implementation, and tools.**

***Keywords*-parallel computing; pattern; model**

I. Introduction

With the rapid development of computational techniques and computational methods, almost all disciplines are now tending quantification and precision, generating new areas, i.e. computational physics, computational chemistry, computational materials, computational mechanics, computational biology, computational meteorology and computational science electronics. Computing Science and Engineering (CSE) is meanwhile generated. Computation enhances people′s ability to engage in scientific research, broadening the perspective of people′s insight into nature, accelerating the process of transforming science and technology into productive forces, and profoundly changes the ways of human beings. Computational science, theoretical science and experimental science have been three primary disciplines.

Parallel computing [6], [9], in short, is the computation on a parallel computer, and is often synonymous with high performance computing and supercomputing because any high-performance computing and supercomputing can not be separated from the use of parallel computing technology. The development of parallel computing technology is inseparable from the human endless pursuit in computer performance. A very high challenge for parallel computing is requested in the following aspects:construction and simulation of prediction model, engineering design and automation, can be used in exploration, medical, military and basic theoretical research.

The demand for parallel computing is extensive, and there are three main types of demand: 1. Compute-intensive applications, such as large-scale scientific engineering calculation and numerical simulation; 2. Data-intensive applications, such as digital libraries, data warehousing, data mining and computing visualization; 3. Network-intensive applications, such as system work, remote control and remote medical diagnosis.

This paper gives a simple summary of the parallel computing technology, and the main branches of parallel computing. Section 1 presents introduction; section 2 gives the descriptions of parallel programming patterns; section 3 shows the designs for parallel programs; section 4 presents the parallel programming models; section 5 shows the parallel programming languages; section 6 presents the parallel algorithm design; finally, a perspective is given.

II. Parallel Programming Pattern

In general, parallel programming pattern [2] is a form of parallel programming. Similar to the adopted functional or structural programming in sequential programming, it is the way that programmers parallelize every modules. Parallel programs can be inferred into some definitive programming patterns, every of which includes a class of algorithms possessing the same control structure. Parallel programming patterns can be classified into six classes: master/slave pattern, single program multiple data (SPMD) pattern, data pipelining pattern, divide and conquer pattern, speculative multithreading (SpMD) pattern, hybrid pattern.

A. Master/Slave Pattern

Most representative distributed computers use master/slave pattern [8], which uses Remote Procedure Call (RPC) to communicate among modules, which exist inherent parallelism. Master/slave pattern contains two parts: master and slaves. Master takes charge of task decomposition, collecting the solutions of every subtask, and aggregating all solutions into the final solution. Slaves perform the following actions recurrently: message receipt, subtask processing, returning results to master.

B. Divide and Conquer

Divide-and-conquer strategy [5] aims to divide a problem into two or more sub-problems, solve every sub-problem to obtain independent solutions, and then combine these results to get the final solution. The common operations in divide-and-conquer strategy are decomposition, calculation and

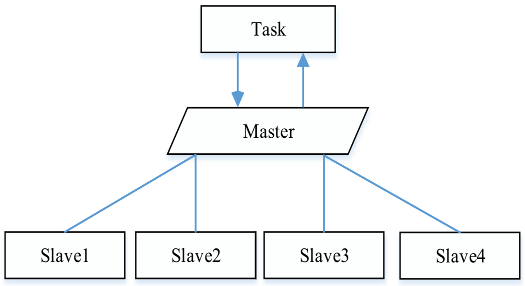

Figure 1: Simplified master/slave pattern

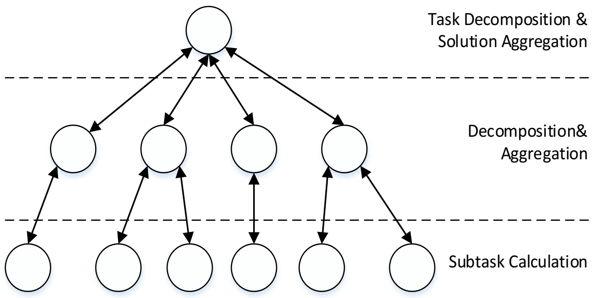
Task Decomposition & Solution Aggregation
Decomposition& Aggregation
Subtask Calculation

Figure 2: Virtual tree of divide-and-conquer

summary. The structure of the divide-and-conquer strategy is organized into a virtual tree. Some processes derive subtasks and the results of these subtasks are aggregated to compute an integrated solution. The calculation tasks are performed by the leaf nodes of the virtual tree, and the execution process is shown in Fig 2.

C. Pipelining

A pipeline includes a chain of processing elements (functions, coroutines, processes, threads, etc.), which are arranged to reach a goal that the output of one element is the input of another element. Usually, this pattern buffers some amount of results between consecutive elements. Pipeline pattern divides tasks into several sub-tasks, and begins to execute them in different stages. What flows through these pipelines are often a stream of records, bits or bytes, and the elements of a pipeline may be filters.

A pipeline is one-directional and linear, though more general flows apply the pipelining pattern. Take an one-directional pipeline as an example, it may have some communication in another direction, known as one return channel, or one pipeline may be fully bi-directional.

D. Speculative Multithreading

Speculative Multithreading [3] (SpMT), also called Thread-Level Speculation (TLS), is a promising technique which parallelizes sequential codes aggressively, without considering too much about the success of execution, which is guaranteed by hardware. Compared to manual parallelization, irregular sequential programs are accelerated by SpMT with lower cost and fewer user interactions, and has a wide range of applications [1].

SpMT was proposed as a perspective method for parallelizing sequential programs to achieve more parallelization running on Chip Multiprocessors (CMP). SpMT execution model [7] partitions a sequential program into multiple speculative threads, which respectively executes a different part of the sequential program. Among parallelly executed threads, only one special thread is called non-speculative thread, allowing to commit its results to memory, and other threads are all speculative.

When the non-speculative thread completes execution, it needs verify the execution results of its successor thread. If the results of successor thread proves correct, all the values generated by the non-speculative thread will be committed to memory and then its successor thread turns into non-speculative. Otherwise, all speculative child threads would be revoked by the non-speculative thread and its successor threads would be re-executed. Fig.3 shows four scenes, including sequential execution, speculative execution, speculative failure, Read-After-Write(RAW) violation. Fig.3(a) shows sequential execution, while Fig.3(b) illustrates the scene of successful speculation. Fig.3(c) presents the scene of failed speculative parallelization and sequentially executes the child thread again. Fig.3(d) illustrates the scene of RAW violation, in which child thread is re-executed.

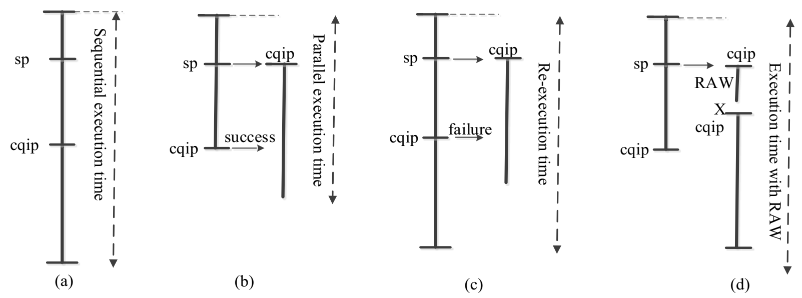

Figure 3: Thread-level speculative model: (a) sequential execution; (b) parallel execution; (c) failed parallel execution; (d) RAW.

III. Designs for parallel programs

During the design of parallel programming, two points need to be considered, namely correctness and efficiency. In the process of parallel programming design, we not only need consider the development environment, but also the difficulties and progresses. The difficulties for parallel program design primarily lie in four aspects:

- There are not too much good paradigms to parallelize algorithms.
- Computational models are not unify.
- Parallel programming languages are still not mature and sophisticated.
- Development environment and tools lack growing period.

While, the progresses can be summarized as follows:

- Many of algorithms have good paradigms.
- Programming type has been classified into three classes, including message passing, shared variables, data parallelization.

A. Design Process and Parallel Programming Environment

1) Design Process: The design process can be summarized into two steps:

- The specific algorithms are extracted and written in accordance with the specific phenomena.
- The generated algorithms are realized by programming on the foundation of parallel computation models, for example LogP, PRAM, BSP, PGAS, and *et al.*

2) Parallel Programming Environment: In the current parallel machines, the more popular parallel programming environment[10] can be divided into three categories: message passing, shared memory and data parallelism. Their characteristics: typical representatives, portability, parallel granularity, parallel operation mode, data memory mode, data distribution method, learning difficulties, scalability and other aspects are given in Table I.

IV. Parallel Program Model

Parallel program model is a collection of procedural abstractions that provide programmers with a transparent diagram of the computer hardware/software system. The model can be used by programmers to design parallel programs for multiprocessors, multiple computers, and clusters of workstation.

Classifications of parallel programming models can be divided broadly into two categories:implicit parallel program model and explicit parallel program model.

A. Implicit Parallel Program Model

The most famous method in the implicit parallel model is the automatic parallelization of serial programs. The compiler performs a correlation analysis of the serial source code program, and then uses a set of conversion techniques to convert the sequential codes into parallel codes. The key to parallelizing the serial code is the correlation analysis, which mainly includes analysis of data correlation and analysis of control correlation. If operation A depends on B, then A must be executed after B; both operations can execute in parallel if they are not correlated with each other. If the correlation does exist, it must be removed by use of conversion technique. Three most important conversion methods are the privatization, parallel reduction, and induction variables.

B. Explicit Parallel Program Model

The explicit parallel program model can be primarily categorized into three parts: data parallel model, shared-variable model, message passing model. The difference of them are shown in Table I.

1) Data Parallel Model: The Data Parallel Model can be implemented either on a SIMD computer or on an SPMD computer, depending on the granularity. SIMD procedures focus on the development of instruction-level fine-grained parallelism, while SPMD program focuses on the parallelism of granularity at subprogram level. Data parallel program emphasizes local computation and data routing, and is suitable for the application of regular networks, templates and multidimensional signal/image data sets to solve the problem of fine grained applications. Synchronization of data-parallel operations is done at compile-time rather than at run-time. Hardware synchronization is performed by the controller performing a SIMD program lock step operation. In the

Table I
Comparisons of three parallel programming environments

Characters	**Message Passing**	**Shared Memory**
Representative	MPI,PVM	OpenMP
Parallel granularity	process-level granularity	thread-level granularity
Operating mode	asynchronous	asynchronous
Memory mode	distributed memory	shared memory
Data distribution	explicit	implicit
Degree of difficulty	more difficult	easy
Scalability	good	bad
Characters	**Data Parallelism**	
Representative	HPF	
Parallel granularity	process-level fine granularity	
Operating mode	loose synchronization	
Memory mode	shared memory	
Data distribution	semi-implicit	
Degree of difficulty	easier	
Scalability	general	

synchronous SIMD programs, the communications between all PEs are controlled by hardware. In addition to the interlocking operation between all PEs, data communication between PEs is also carried out in a lock-step manner. The execution of these synchronization instructions and the data routing operation makes the SIMD computer very efficient in exploiting the spatial parallelism of large arrays, large grids, or grid data.

2) Shared Memory Model: The shared memory model is an abstraction of a general centralized multiprocessor, such as the parallel machines with SMP architectures. The bottom layer is a series of processors, and all processors can access the data of the shared memory. All data are accessible for every processor, and need not be transmitted among processors. As all processors can interact and synchronize through shared variables as they can access the same location in memory.

3) Message Passing Model: In the message passing model, processes residing on different processor nodes can communicate with each other by delivering messages over the network. The message can be an instruction, a data, a synchronization signal, or an interrupt signal. In message passing parallel programs, the user must explicitly allocate data and loads. The message passing parallel model is more suitable to develop large-grain parallelism, which is multithreaded and asynchronous, and requires explicit synchronization (such as roadblocks) to ensure correct execution sequence.

However, these processes have their separate address space. The messaging model is more flexible than the data parallel model, and the two widely used standard libraries, PVM and MPI, make messaging a much better portability. A messaging program can be executed not only on a multiprocessor with shared variables, but also on a multi-machine with distributed storage.

V. Perspectives

Recently, with the continuous development of new applications as well as hardware technology, and parallel computing also has several new developments, primarily including cloud computing, multi-core architecture, personal high-performance computers.

A. Data-Centric Cloud Computing

Cloud computing [4] is analogy to the distributed computing, parallel computing and grid computing, but different from them. Cloud computing is based on parallel computing.

Cloud computing means an increased demand for parallel computing on the server side, as tens of thousands of user applications are implemented over the Internet in the cloud. While cloud computing brings about fundamental changes in the users$'$ work ways and business models, large-scale parallel computing technologies put forward new requirements for parallel computing.

B. Multi-Core Architecture

In recent years, with the limit approximation of chip integration, and energy consumption and cost, the products with multi-core architectures are gradually becoming the market mainstream.

Multi-core architecture makes use of the resources within the chip effectively, and can effectively exploit the programs$'$ parallelism, improving the performance rapidly. In multi-core structure, the interconnection among processor cores is shortened and the data transmission bandwidth is increased, and resources are effectively shared, while the power consumption of chips is also reduced.

C. Personal Computer with High Performance

Personal computers are developing toward high-performance computers, which are characterized by high performance. They are faster than PC computers, ordinary workstation or network computers in terms of floating-point speed, ability to handle data set size, I/O performance,

performance of exchange and synchronization, *et al*. There are four major shifts in personal high-performance computers, including:

1) Personal computers with high performance can be used in offices, while they were only used in computer room.
2) Personal computers with high performance are a type of volume products, so the cost of the computers, reliability, production capacity, management, maintenance, service, and application areas all have a higher demand.
3) Personal computers with high performance are turning to user-centric using model.
4) Personal computers with high performance are adopting high productivity programming.

VI. Acknowledgement

We thank our 3C laboratory for their great support during our work. We give our best hope to all our colleagues of laboratory for their collaboration. We also thank reviewers for their careful comments and suggestions. This work is supported by Doctoral Fund of Ministry of Education of China under Grant No.2013021110012 && National Natural Science Foundation of China under Grants No.61173040.

References

[1] David Bader. *Analyzing Massive Social Networks Using Multicore and Multithreaded Architectures*. Springer-Verlag, 2010.

[2] Manuel I. Capel, Antonio J. Tomeu, and Alberto G. Salguero. A set of patterns for concurrent and parallel programming teaching. In *European Conference on Parallel Processing*, pages 203–215, 2017.

[3] Alvaro Estebanez, Diego R Llanos, and Arturo Gonzalez-Escribano. A survey on thread-level speculation techniques. *ACM Computing Surveys (CSUR)*, 49(2):22, 2016.

[4] Brian Hayes. Cloud computing. *Communications of the Acm*, 51(7):9–11, 2008.

[5] Yi Mei, Mohammad Nabi Omidvar, Xiaodong Li, and Xin Yao. A competitive divide-and-conquer algorithm for unconstrained large-scale black-box optimization. *Acm Transactions on Mathematical Software*, 42(2):13, 2016.

[6] Aaftab Munshi and Jeremy Sandmel. Data parallel computing on multiple processors, 2018.

[7] Christopher JF Pickett and Clark Verbrugge. Sablespmt: A software framework for analysing speculative multithreading in java. In *ACM SIGSOFT software engineering notes*, volume 31, pages 59–66. ACM, 2005.

[8] Aleksander Rydzewski and Pawe? Czarnul. A distributed system for conducting chess games in parallel. *Procedia Computer Science*, 119:22–29, 2017.

[9] Wenwu Tang, Wenpeng Feng, Jing Deng, Meijuan Jia, and Huifang Zuo. Parallel computing for geocomputational modeling. 2018.

[10] YongbingLi. Parallel programming environment and tools. *Journal of ChangZhi College*, 26(2):41–43, 2009.