Journal of Computing and Applied Informatics (JoCAI) Vol. 02, No. 2, 2018 | 101 - 109

DATA SCIENCE
Journal of Computing and Applied Informatics

Study of Scheduling in Programming Languages of Multi-Core Processor

Mina Hosseini-Rad[1*], Majid Abdolrazzagh-Nezhad[2], Seyyed-Mohammad Javadi-Moghaddam[2]

[1]*Department of Computer, Faculty of Engineering, Bozorgmehr University of Qaenat, Iran.*
[2]*Department of Computer, Faculty of Engineering, Islamic Azad University, Birjand Branch, Iran.*

Abstract. Over the recent decades, the nature of multi-core processors caused changing the serial programming model to parallel mode. There are several programming languages for the parallel multi-core processors and processors with different architectures that these languages have faced programmers to challenges to achieve higher performance. In addition, different scheduling methods in the programming languages for the multi-core processors have the significant impact on the efficiency of the programming languages. Therefore, this article addresses the investigation of the conventional scheduling techniques in the programming languages of multi-core processors which allows the researcher to choose more suitable programing languages by comparing efficiency than application. Several languages such as Cilk++، OpenMP، TBB and PThread were studied, and their scheduling efficiency has been investigated by running Quick-Sort and Merge-Sort algorithms as well.

Keyword: Multicore, Scheduling, Parallel Programming, Processor.

Abstrak. *Dalam beberapa tahun terakhir, sifat dasar prosessor multi-core menyebabkan perubahan pada model pemrograman seri menjadi model pemrograman parallel. Banyak bahasa pemrograman khusu untuk prosessor multi-core yang parallel dan prosessor yang berbeda arsitekturnya yang membuat programmer mendapatkan performa yang lebih tinggi. Sebagai tambahan, metode penjadwalan yang berbeda di dalam bahasa pemrograman memberikan dampak yang signifikan didalam efisiensi Bahasa program tersebut. Oleh karena itu, artikel ini mengalamatkan tentang investigasi teknik penjadwalan yang awal didalam bahasa pemrograman prosessor multi-core yang mengijinkan peneliti untuk memilih bahasa pemrograman yang sesuai dengan membandingkan koefisienannya di dalam aplikasi. Bahasa pemrograman tersebut adalah Cilk++, OpenMP,TBB dan PThred, yang mana kesemuanya diinvestigasi menggunakan Quick Sort dan Merge Sort algoritma dan dibandingkan juga.*

Kata Kunci: *Multi-core, Penjadwalan, Pemrograman Paralel, Prosessor.*

Received 13 April 2018 | Revised 25 May 2018 | Accepted 27 June 2018

*Corresponding author at: Bozorgmehr University of Qaenat, Abolmafakher St, Qaen, South Khorasan, Postal code: 9761986844, Iran
E-mail address: hosseinirad.edu@gmail.com, abdolrazzagh@buqaen.ac.ir, smjavadim@buqaen.ac.ir

ISSN: 2580-6769 | e-ISSN: 2580-829X | DOI: 10.32734/jocai.v2.i2-327
Journal Homepage: https://talenta.usu.ac.id/JoCAI

1. Introduction

Over the past decades, the multi-core processors have opened new view in the field of parallel programming. In the past, rich institutes and organizations possessed expensive Parallel architectures for scientific researches and military applications. However; recently they are accessible for use in multiple applications such as multimedia, trade, entertainment, etc. Different types of the multi-core processors architecture have encountered programmer to the new challenge including portability and program efficiency of these processors [1].

The multi-core processors are architecturally divided into three groups: General-purpose homogeneous, General-purpose heterogeneous and Graphic processors. The general-purpose homogeneous processors have a symmetric structure in which all the cores have equal processing throughput and are linked together using hierarchical of comprehensive cache. Processor cores of the general-purpose heterogeneous processors do not have equal processing throughput. In this structure, the cores are usually linked together by an internal network and special rules; for instance, IBM-Cell is one of them that possess eight worker symmetric cores and one master stronger core. The modern graphic processors usually maintain 400 to 1000 linear processor cores which can employ the SIMD structure for computation of matrix [2].

The parallel programming of shared memory is usually implemented in the multithread mode. Scheduling of threads execution at the running time is performed by an operating system or by the library of the programming language level, hence; due to access of all threads to the shared memory, we should consider preparations. The accessibility to the shared variables must be locked up, because, concurrent access of the threads to the variable may cause the production of invalid values. In addition, to preserve the proper order of execution in the threads at the particular points of program a barrier synchronization mechanism will have to be used [1]. However, the program efficiency can be dropped out due to several architectural complexities such as the size of memory and method of levels sharing [3]. These Problems include the following [1]:

- Load Imbalance: Inappropriate division of task between the processor's cores may cause early completion of some of the cores task and then being unemployed until the remaining cores complete their tasks.
- False Sharing: While two different variables have close distance on the main memory, it may place them on a line of cache. In such a case, if each of these variables being under control of separate core; thus, alteration in one of them can cause invalidation of the entire cache line, and on the other hand the other core is limited to access to its variable as long as the alterations of the cache line are spreading at the memory hierarchy.
- Nested parallelism overhead: Creating several thread children by one thread can lead to the reversibly repeated production of a large number of the threads which outnumber processors

cores. A large number of thread production due to the need to process in the direction of their creation and deletion can cause memory overhead.

This article addresses the impact of scheduling methods in programming languages for the hierarchical-structured multi-core processors. In the following, a short overview has been presented in the programming languages for the multi-core processors. In the third section, techniques used in the scheduling are investigated. Additionally, forth section shows a study on Quick-Sort and Merge-Sort algorithms to evaluate the efficiency of the various scheduling methods and finally; conclusion and suggestions have been presented.

2. Related Studies

Recently, many languages were presented for the multi-core programming which each of them in addition to the simplifying the method of parallel programming they have brought the efficiency as well(Aldinucci et al. 2009). In theory, they are programming languages but practically, are the library which can be added to common languages such as C++ and Java and can be an alternative for the low-level parallel programming. In the following, we will address some of these languages.

2.1. PThread

PThread is a low-level library which is added to C language to prepare the ability of creation and control on the threads of the operating system and then constructs the parallel program structure [1]. The threads scheduling in the operating system rotationally and exclusively acts, and several threads from each process are executed in the period. The most important standard for scheduling in the operating system is maximizing the system efficiency; consequently, improvement of the efficiency is the undefined target.

2.2. Skandium

Algorithmic skeletons are high-level model of distributed and parallel programming which using common programming patterns are able considerably to decrease the complexity of the parallel and distributed programs (Cole 1988). By the type of each skeleton, they have multiple muscles which in fact each muscle is predefined function along with specific outputs/inputs that have the certain role. In the Skandium skeleton, each muscle is run as a task [4]. All the tasks are kept inside of the global pool, and these processors cores will pick up a new task from the pool after completion of the previous task. Tasks children will be placed in the pool when they are ready. It should be noted that the unready tasks are not kept in the pool; accordingly, the order of tasks execution is unnecessary for the scheduling of the Skandium execution.

2.3. Open MP

In Open MP two guides such as Task and task wait to prepare creation and synchronization of the tasks [5]. In this language, the tasks will be implicitly created and deleted; however, the task guide can explicitly define the tasks, so then, this feature has an application in recursive algorithms as

well as pointer-based algorithms. The task in the OpenMP is defined as a piece of code that has its data space. To schedule the tasks, a private priority queue is considered for each core, to reduce the amount of concurrency and to increase the local access to the tasks instead. The Work stealing technique is used for improving the load balancing in the queue.

2.4. Cilk++

It is a library for C++ language which has been created to improve the recursive algorithms for the multi-cores(Leiserson 2010). Extensions of the Cilk++ include cilk_for ، cilk_spawn and cilk_sync parallel structure. Cilk_for allows iterations of the loop to be divided between several cores in parallel. Cilk_spawn runs a function as a new thread. In additional, cilk-syn causes blocking parent thread for ensuring that its children finish before it does. Similar to the OpenMP, Cilk++ uses the private queue for the task scheduling and utilizing the work stealing it establishes the load balancing as well.

2.5. TBB

TBB is a library template that enables programmers to develop their program for the multi-cores [6]. TBB schedules light weight tasks on the user threads. The scheduling is taken place when the tasks dependency graph continuously is being assessed as well as the tasks are kept in the hierarchical of pools. Each created task is stored in its parent pool when it is ready to use. While the task does not have children, it will not have the pool. Each pool has been formed from an array and additionally the children's pool is connected to its array via the linked list. The scheduling strategy of TBB is an execution in width and work-stealing of depth. In this manner, the available task is run in the deepest pool, and as long as there is not the available task in the deepest pool for running, it will be stolen from the shallowest pool and is brought to the current pool. This method avoids producing mass tasks which cause the memory loss as well as to extension; large grain size will be selected upon completion of the current tasks.

2.6. Multi-BSP

Multi-BSP model tries to make a bridge between the BSP programming and the multi-core programming for the parallel recursive algorithms [7]. In this model, the architectural details of hierarchical memory such as the size of the levels memory and the delay of memory release have been presented between surfaces and surface structure as well as it expresses the way of scheduling of created tasks caused by memory division in the recursive algorithm. Moreover, Similar to BSP, Multi-BSP postpones the tasks synchronization till the completion of all the same level tasks and during the maximum use of local accesses; it can decrease competition on the memory bus. The scheduling is as a hierarchical queue in memory surfaces which has been described in the part of 3-1-3.

3. Scheduling Techniques

In the parallel programming, the execution is as simultaneous tasks, and these tasks are kept in the queues, based on that, control on way of Scheduling of task exit from the queue as well as how to insert new tasks to the queue can set up the order of operation. Therefore, this part will address types of queues and relevant scheduling.

2.1. Types of Queue

Similar to all the queues, the applied queues in execution scheduling of the multi-cores have FCFS rule which it means the oldest element receives service earlier; however, they are different from each other regarding the combination of the queue in the scheduling system and exit priority from the queue. In the following, three groups of these queues are described.

A. Central Queue

In this case, all threads are kept inside of the central structure, and cores receive thread from the central structure and then mutually insert new threads into the central structure. In this structure cores had concurrency access to the queues and based on, insertion and deletion from the queue will be protected by mechanisms of mutual exclusion locking. In the operating system, this structure is implemented as Round Rabin queues or FCFS [8]. In languages such as Skandium and Fork/Join that use Java, the queue is installed for the tasks as a priority queue [4], [9].

B. Private Queue

In some of the programming languages, a private queue is considered to each of the cores. In fact, this queue is a stack in which the last input task is the first output of structure. Whenever a task creates children, it places them in its queue without stealing from other queues [10]. The reason for mentioning the queue in these languages is complexities added to the stack like the work stealing which has been addressed in 2-3 part. Because the private queues are accessed just by one of the processor cores at a time, they do not need mutual exclusion locking; hence, for locking, less overhead is imposed to access the queues in comparison with central queues. However, these queues appropriate much more space of the processor cache. Languages such as Cilk++ and OpenMP use this type of queue [5], [10].

C. Hierarchical Queue

In hierarchical queue, the tasks are more complex schedule. Recently, there are two implementations of the hierarchical queues which in one state, one queue is considered for each component of the memory. After the exit of a task from the queue, all of its children will have to be located in the subset queues that is related to the subset of parent queue [7]. In this case, selection of the queue level depends on the size of the memory which the tasks will use during its lifetime. The suggested method for the Multi-BSP model has not been yet implemented, and it is still based on increasing of local access to memory [7].

In the second state which is being used in TBB, each task is inside of a queue, and if it produces children, they will be located into the queue which that queue is linked to the location of its parent in an upstream queue by using the pointer [6]. As a result, there is only one queue which includes the primer tasks, and each of the tasks can have a queue within itself as well. The execution exclusively happens from one of the deepest queues of the highest element in the main queue and this manner; all cores pick up their tasks from one queue and run them. This method selects elements from the nearest queues of the highest levels when the queue elements were finished.

2.2. Work Stealing

In the work stealing, each core possesses a local queue of the tasks that are ready to run. To escape idle, if the task in the queue of a core is empty, that core tries to steal a task from the pool of other cores. This function is called the work stealing [10], [11]. In comparison with static scheduling which implements their tasks orderly or dynamic methods which use a central pool of the tasks, this method possesses advantages such as load balancing and conflict reduction at cache level for access to the queue.

This method requires mutual exclusion establishment due to access to another queue of the processor cores, to ensure the accuracy of the queues function. So long as the memory component-based hierarchical queues are being used, cost of the work stealing will be reduced due to the low probability of the concurrency access to queues. In additional, in the most implementations such as Cilk++, to choose the largest provided task, the work stealing will be taken place from the bottom of the queue of other core [10].

4. Experimental Results

To evaluate the discussed scheduling methods in part 3, two studies have been performed on quad-cores processors Intel Corei7 2670QM. The operating system of experiences was Ubuntu 12.04(Ubuntu Desktop 12.04 LTS), and all of them were compiled using GCC 4.5 compile [12]. Before compiling, TBB and Cilk++ programs must be prepared by their dedicated pre-compiler [6], [10].

4.1. Merge-Sort Study

The applied Merge-Sort algorithm in this study was chosen regarding Grama, A and et al. research(Petersen and Arbenz 2004), and it has been rewritten for TBB, Cilk++ and OpenMP. Its feature is balancing of the produced tasks which in each phase of division, the work division tree is symmetric at all phases. Therefore, tasks growth should be symmetrically done at the private and hierarchical queues. Fig 1 shows the results of the run of the Merge-Sort algorithm. The result shows that Cilk++ and OpenMP have the more comparative advantage than TBB and PThread. In pThread, the efficiency was less than others due to use of the central queue and that; the cores compete to access to the queue. As regards, TBB has the hierarchical queue, but the queue acts as a central queue for the cores and similar to the pThread; it suffered from competition to access

to the queue. TBB, showed the better efficiency than others, for the fact that user-level threads are lighter than the threads of operative system.

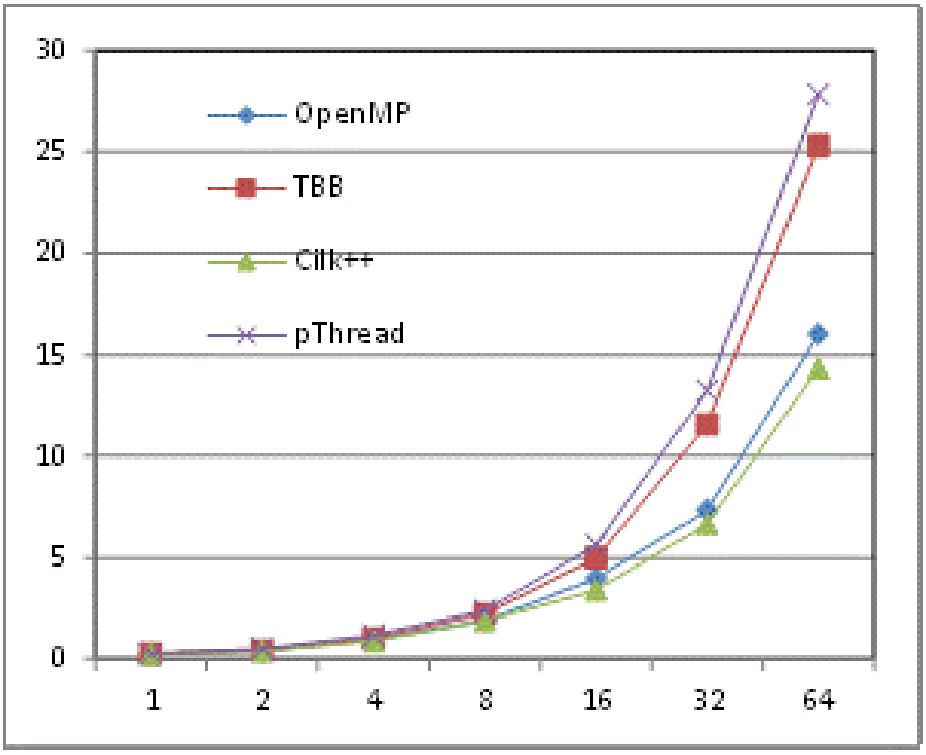

Figure 1 The horizontal axis is indicating input size/MB and the vertical axis indicating t/s.

4.2. Quick-Sort Study

The applied Quick-Sort algorithm was chosen regarding Leiserson and et al. research [10], and it has been rewritten for TBB, pThread and OpenMP. It can lead to the imbalance in the task dependency tree Because of being unpredictable of the split axle at every phase of Quick-Sort, as in some of them the tracks from root to leaves are shorten than other tracks. Therefore, we will observe imbalance in the private and hierarchical queues which it is predicted that the imbalance is compensated by the work stealing technique. Fig 2, shows the results of a run of the Quick-Sort algorithm. The results indicate the relative proximity of Cilk++, OpenMP, and pThread functions, while, TBB shows lower efficiency. The produced imbalance in the private queues caused lack of optimal function, and they do not show more benefits than the central queue. This may happen due to the overhead of the work stealing. On the other hand, TBB has faced to more overhead because of more complexes of the work stealing and having the central queue-like structure as well.

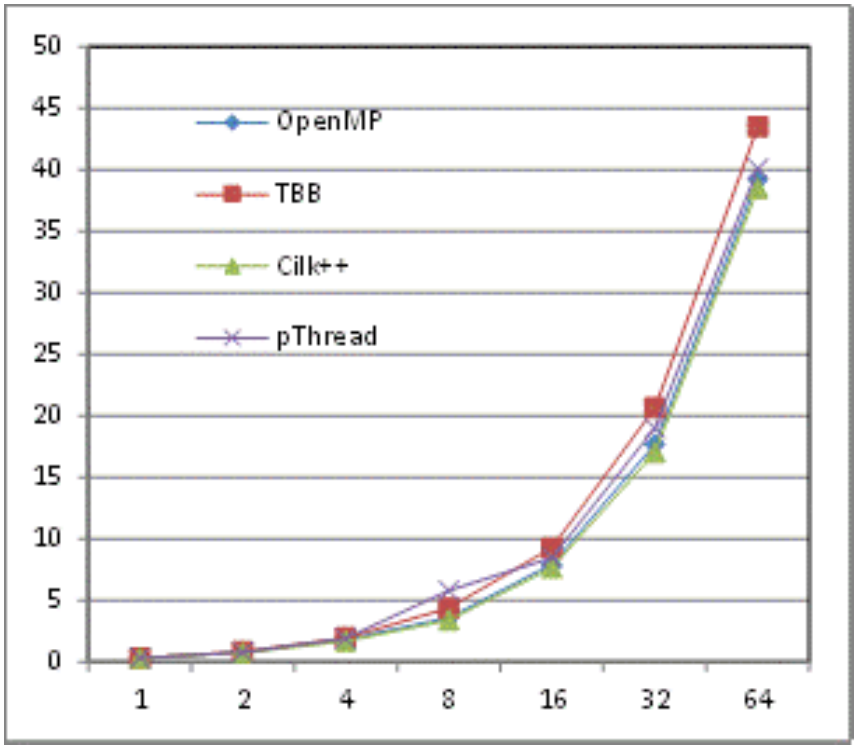

Figure 2 The horizontal axis is indicating input size/MB and the vertical axis indicating t/s.

5. Conclusion and Suggestions

By the results of part 4, we observed that the scheduling structure such as Cilk++ and OpenMP which are using the private queues have better efficiency than pThread that uses the central queue. The hierarchical queues have more complexes in scheduling algorithm and more overhead in memory which is unnecessary for the current hierarchical processors; hence, based on that, TBB which uses the hierarchical queues showed less efficient in comparison with the private queues. In the future, we will be able to evaluate the efficiency of the hierarchical queues by using either more complex processors in cache hierarchy and simulator for simulation of the complex structure.

REFERENCES

[1] M. Aldinucci, M. Danelutto, and P. Kilpatrick, "Skeletons for multi/many-core systems," in *Advances in Parallel Computing*, Lyon, 2010, vol. 19, pp. 265–272.

[2] S. Akhter and J. Roberts, *Multi-Core Programming: Increasing Performance through Software Multi-threading*, 1st ed. Intel Press, 2006.

[3] J. Kwiatkowski, M. Pawlik, and D. Konieczny, "Parallel Program Execution Anomalies," in *Proceedings of the International Multiconference on Computer Science and Information Technology*, Poland, 2006, pp. 355–362.

[4] M. Leyton and J. M. Piquer, "Skandium: Multi-core Programming with Algorithmic Skeletons," presented at the 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, Italy, 2010, pp. 289–296.

[5] C. Addison, "OpenMP 3.0 Tasking Implementation in OpenUH," Jul. 2018.

[6] A. Kukanov and M. J. Voss, "The Foundations for Scalable Multicore Software in Intel Threading Building Blocks," *Intel Technology Journal*, vol. 11, no. 04, pp. 309–322, Nov. 2007.

[7] L. G. Valiant, "A bridging model for multi-core computing," *Journal of Computer and System Sciences*, vol. 77, no. 1, pp. 154–166, Jan. 2011.

[8] A. S. Tanenbaum, *Modern operating systems*, Fourth edition. Boston: Pearson, 2015.

[9] D. Lea, "A Java fork/join framework," 2000, pp. 36–43.

[10] C. E. Leiserson, "The Cilk++ concurrency platform," 2009

[11] F. Gaud *et al.*, "Efficient Workstealing for Multicore Event-Driven Systems," in *Proceedings of IEEE 30th International Conference on Distributed Computing Systems*, Italy, 2010, pp. 516–525.

[12] GCC Team, "GCC, the GNU Compiler Collection - GNU Project - Free Software Foundation (FSF)," *GCC, the GNU Compiler Collection*. [Online]. Available: http://gcc.gnu.org/. [Accessed: 28-Jul-2018].