Multi-Core Concurrency: Guarantees and Confronts

Rakhee Chhibber[1, #], Chandikaditya Kumawat[1]
[1]*Research Scholar, Mewar University, Gangrar Chittorgarh,*
Assistant Professor, Asian School of Business, Noida, U.P.
[2]*Mewar University, Gangrar Chittorgarh,*
[#]Corresponding Author, Email: rakheechhibber1971@gmail.com

Abstract—Multi-core processors signify a transformative change in traditional computing and setting the new trend for high- performance computing (HPC) where are parallelism is an ancient concept. This research is focused on the parallelism technique (data and functional parallelism) and the different algorithms used in the multi core processors with a large number of cores. The chip-level architecture offers better performance and performance characteristics but there is a movement towards the chip-level multiprocessing architectures with a large number of cores continues to provide significantly increased performance and power characteristics. However, this step also offers considerable confronts which arises due to parallelism or in other words due to concurrency. In multi-core computing most of the confronts arises due to parallelism and the fact that different things can be simultaneous. There are many algorithms for the concurrency but in this research, paper is focused for the guarantees and confronts of multi-core concurrency using the FIFO algorithm.

Keywords: **Multi-core Processors, Parallelism, Concurrency, Pipelining, Synchronization**

1. INTRODUCTION

The guarantees and confronts that results due to the arrival of a new technology of core in a computer system (multi-core technology) [16] are based on the concept of concurrency which means the execution of more than one process at the same time. However, when things happen at the same time, it is not easy to follow the coordination between a complex piece of software for interference so firs to fall, it is essential to differentiate between the parallelization of application as well as concurrency and a unique algorithm or process provide these facilities. An actual implementation is chosen for the available possibilities and goes further with that. Since each core has its own cache, so the only one operating system which has sufficient resources and provides a noticeable improvement to multitasking can handle intensive tasks in parallel. Synchronization between the arbitrary numbers of cores in their access to the shared memory is necessary [17-21].

Let discuss an example of pizza-making as if there is two-person one is applying cheese on the pizza base, and another one apply the toppings on it simultaneously which results in a messy pizza but if there are more assembly lines which can work simultaneously then it could make pizza more quickly with the infinite supply of materials but if we check it in reality, there would be a limited number of persons to perform the work and possibly there could be 2 lines while there could be more and similarly in the multi-core systems, the number of processors and other resources decides for the capacity of parallelism which can be implemented. A picked execution depends on the facilities provided by the algorithm itself; parallelism will not help in this calculation to the algorithm which has minimal characteristic simultaneousness. So the program section which runs independently marked by symbols which provide a place where these codes need to exchange data and this even is called as synchronization.

To understand this principle, if we consider the example of the process of the pizza preparation which we had already discussed as there are two persons on can implement cheese on the pizza base and another spread the toppings which are two different processes but if out of these two processes one is faster and another is slower than the faster process have to wait until the second process cannot

finish so now there is a need of synchronization between the speed of these processes.

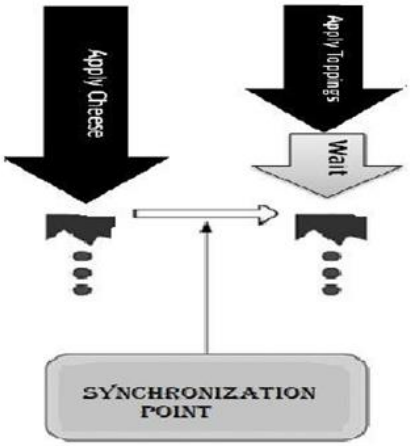

Fig. 1 Synchronization Point where two independent processes interact.

Therefore, it is a feature that two independent processes can work at different unexpected speed as different employees can work differently at different shifts. A program, which produces several independent processes with unique check-in numbers, is shown in Figure 2. The autonomous portions of an application either are called as threads or processes depending upon the implementation as shown in Figure 3. There are two ways to run more than one process simultaneously either run many processors at the same time (Data Parallelism) or if different processors are doing various tasks at the same time (Functional Parallelism), then you can use the work segmentation.[18].

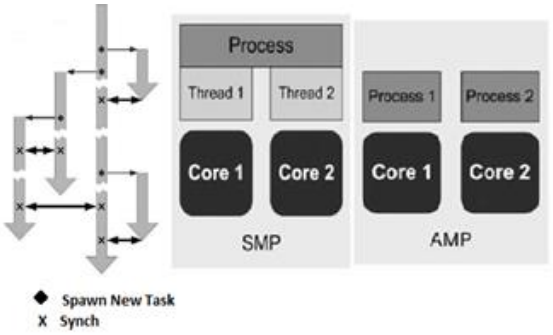

Fig. 2 A number of tasks which are running mutually asynchronously with random synchronization points, Different threads within a process executing tasks and independent tasks on different processes.

2. PARALLELISM

The shift towards multicore processing has led to a much wider population of developers being faced with the challenge of exploiting parallel cores to improve software performance. Debugging and optimizing parallel programs is a complex and demanding task. Tools which support development of parallel programs should provide salient information to allow programmers of multicore systems to diagnose and distinguish performance problems.

2.1 Data Parallelism

This concept is very easy to explain with the help of an example of a 4-bit vector. Let's consider a requirement to increase the value of each access in the array or vector with the help of a loop. The use of parallel processor is easiest to explain let take an example of a 4-bit vector on which we want to operate and assume that there is a requirement to increase the value of each access in the vector as in the below code which is an example of loop:

For I = 1 to 4
Begin
Increment the ith value of the processor
End

This problem can be easily parallelize which will belong to a collective group of the issues called as "embarrassingly parallel" [1] as shown in Figure 3 where each array or vector entry is autonomous and so able to increase independently and with the help of 4-core processor, it is easy that each core perform job on one entry and process the entire process in one-fourth of the time as taken by a single processor or it may be even less than 1/4 because there is no requirement of an iterator for the above pseudo-code is shown in Figure 3 and is known as data parallelism where no more than one instance of data can be operated simultaneously. The built-in concurrency allows acceleration off our times, although there can be as election if at least four processors are available in the given implementation [11]. There are two properties of this looping problem which make

parallelization easy as first the operation on the input does not depend on any other input, and second is the number of inputs is already fixed and known in advance.[1]

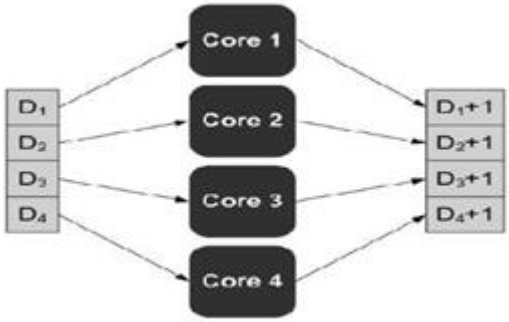

Fig. 3 Data Parallelism

This is a case where the operating system already knows about the parallelism in advance because there is a constant number of iterations which are known at compile time but if there is a 'while' loop or 'loop' where the number of iterations are calculated at runtime means it does not know how many time parallelization has been taken place so cannot be so neatly parallelized.

2.2 Functional parallelism

The other way of splitting task based on their functionality as different processors perform different task for example a program has many text files "to count the number of characters" in each one as in the following pseudo- code:

For more than one file
Begin
Open a file.
Count the number of characters.
Finally Close that file.
End

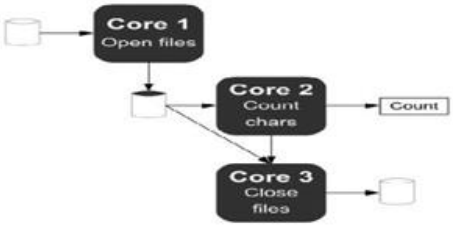

Fig 4. Functional Parallelism

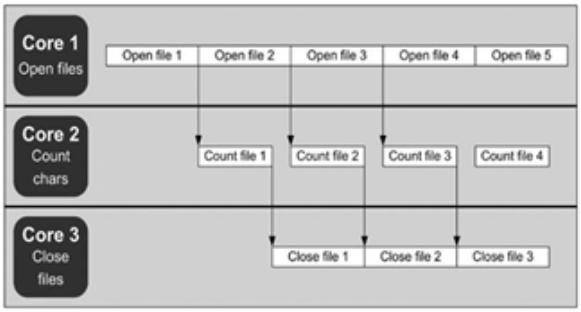

Fig. 5 Pipelining

Functional parallelism is shown in Figure 4 where an example of three processors has been taken and each core is performing a different task as core 1 opens files; core 2 is employed to count the characters; and the core 3 is responsible for closing the files. There is a fundamental difference between functional and data parallelism as in the vector example the parallelism solve problem by eliminating the loop but the another example where the three tasks which are serial in nature there would be no savings at all because it works only when a function has repeated iterations. It is clearly visible in Figure 5 showing pipelining, in which when the file is opened in 1stcore at that time 2nd and 3rd cores sit idle because when a file is opened then it can be read by the second processor and count the characters and till this time the 3rd core will still idle which will start working only when core 2 will finish its job, which explains that this method is not beneficial in the case of only one file. But when real-world programs and algorithms are considered then it uses both "data and functional concurrency" and in several situations both can be used.

One of the pipeline challenges is called to balancing of it because the execution of a program can only as quick as the speed of the slowest phase as in Figure 5, it has been shown that the opening files takes longer time than the counting of characters so performing rapid counting will not increase performance but it increase inactive time among files so the perfect situation for every pipeline is the distribution of task to every stage so it takes the same time which is not only difficult but more or less

impossible to balance the pipeline perfectly[20]. The above examples are very simple examples which have independent operations but when a calculation of one operation depends on the results of another operation then things become complicated and there are several ways in which these dependencies arise so some basic issues are the challenges in parallelizing software for multi-core like it is not possible to analyse all the dependencies manually in a program but some tools are required.

3. PRODUCERS AND CONSUMERS IN PARALLELISM

Some program uses the producers and consumers data as shown in (Figure 6), then it became easier to understand dependency, one part of the program makes some calculation which can be used by any other part, especially in object-oriented approach, it results at "fine-grained instruction levels" where objects are also "producers as well as consumers of the data".

Example of Data Parallelism is

X= A+B (Producer of X)
Y=X+3 (consumer of X)

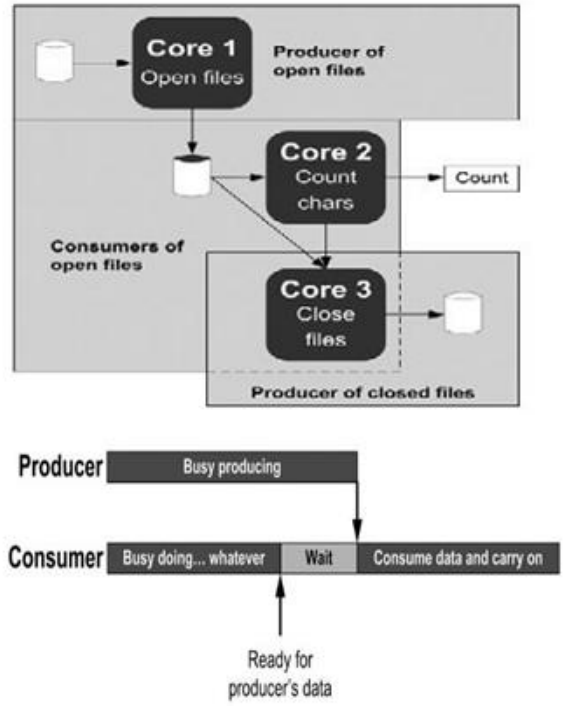

Fig. 6 Producer and consumer in Functional Parallelism and their processing.

Figure 6 is representing the basic dependency where a consumer of data wait sit until the producer has produced that data but this mainly depend on the language as well as approach. There are many compilers had been designed which can implement a low-level concurrency at the instruction level by instruction reordering for efficient execution. It is complicated with languages like C having pointers and compilers does not understand the way how various pointers can relate for optimization because of two reasons associated with pointers as "pointer aliasing and pointer arithmetic".

The "aliasing of a pointer" is a very regular phenomenon in a "C program" for example let's take an example of a function (void foo (*image imagePtr)) which accept an image as pointer parameter (imagePtr) and a program which call this function for 2 different images leftImage and rightImage. So when this function is called with the left image as a parameter refer the same data as when it is called with the right image as shown in (Figure 7).

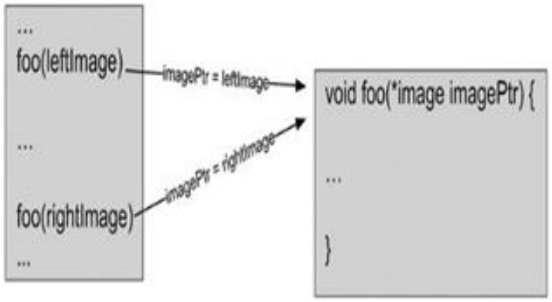

Fig. 7 A function foo which accept an image pointer and different pointers pointing to the same data at different times with different image.

Pointer arithmetic is another noticeable problem because, if the program knows the start location for a pointer but when a manipulation has been taken place in the pointers it may possible that it may point some other address which is not required to point.

randomPtr= knownPtr + sizeof(someObject)

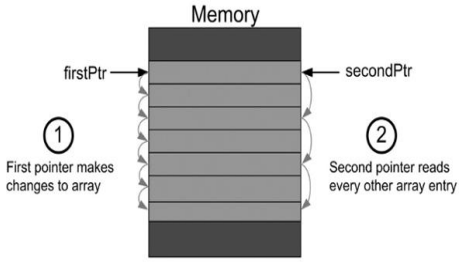

Fig. 8 2 pointers firstPtr and secondPtr working on the same matrix create a dependency (a dynamic inspection).

For example, when we want to access a matrix with the help of a pointer and make changes, it can be difficult to recognize later operations, in which a different pointer accesses the same matrix (with a "different pointer arithmetic") called "read that data (Figure 8)[20]". If the another check uses data which is going to perform the first scan, then the program will work incorrectly by giving those similarities as independent. In several cases, this reliance cannot be recognized by a fixed scrutiny so the only mode to find out is to watch in the execution time that the signs are directed at one place. The discussed "dependencies" has focused on a consumer who should stay until the manufacturer has the data: "Write earlier to reading" and in contrast "If a producer is about to rewrite a memory location", then he must ensure that all the users of old data are ready earlier than overwriting of previous data over latest data (Figure 9)[14]. "This is called anti-dependency so everything which has been discussed about dependency is also valid for anti-dependencies, apart from that until all the readings are done; we have to wait to write: Read before writing".

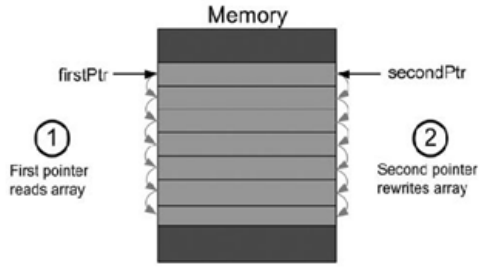

Fig. 9 firstPtr reads array and secondPtr rewrites is an example of anti-dependency.

3.1 More complex situations of dependencies as in Loops

The situation of dependencies become more complicated if a program consists of a loop and targeted for parallelization. For example: consider code as an example.

For I = 1 to 4
Begin
Add the (i-1) th value to the ith value
End

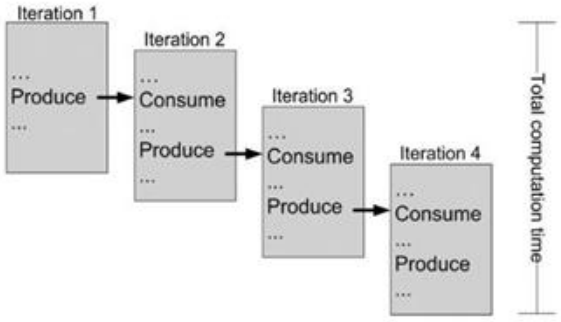

Fig 10. Iterations are in parallel, each one should wait until the necessary data is produced by the previous recurrence, which causes displacement, which increases the time of total calculation required for independent iterations.

These changes slightly as a result of each loop repetition, resulting in the next loop being processed in repetition. Thus, the second loop repetition cannot be started till the first repetition has generated its data so the loop iterations can not one in parallel because they are offset by its antecedent (Figure 10).

As the largely calculation time is less than the time it takes to cycle through the processor, "it is not as fast as there was no dependency between iterations of a cycle." Such dependence is called "loop- carrier" (or "loop-carry") dependence. It becomes still further complex if used for nested loops that iterate over much iteration. Another example is a two-dimensional matrix using i variable for rows and j for columns as shown in (Figure 11.a.)[14]. Now assume that the value of a cell based upon the fresh value of the cell exist in xthe previous row (Figure 11.b.).

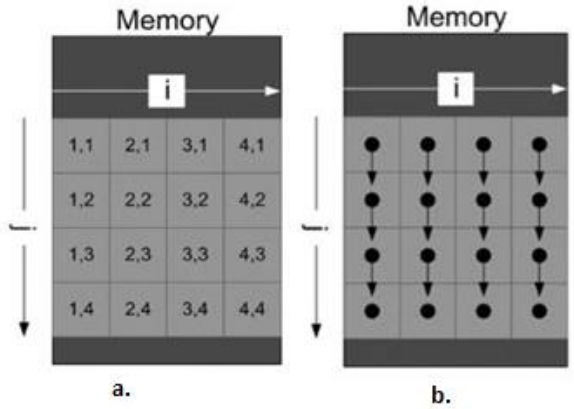

Fig. 11(a). 4 X 4 matrix with accessing variable I for rows and j for columns. (b). if the value of one cell depends on the new value

For j=1 to 4(outer loop)
Begin
for i=1 to 4 (inner loop)
Begin
add the (i, j-1)th value to the (i,j)th value
end
end

There are many ways to parallel this code but depend upon how many core we have for example if we go for maximum number of cores, we must require 16 cores as there are 16 cells in 4 X 4 matrix. If 4 cores are available, one row can be assign to one core but in this case, the second line could not start until the first cell of the first line was read as in Figure 12.a. But if it could be completely parallelized then first entries of all the rows can start simultaneously, second entries have to wait until the processing has not completed on their respective first-entries as in Figure 12.b[14]. So having many cores doesn't guarantee for the speed as only 4 cores can execute this code in efficient manner as shown in Figure 13 in which each column has been assigned to each core therefore "the cycle can be processed faster because one core does not have to wait for the other core". The nested loops raise a concept of 'loop distance' which is acquired by each iterator as shown in figure 12.a where the dependency has shown by arrows and represent the load distance as it is 0 for i is therefore no dependency and 1 for j because the data used in a cell depends on its previous cell, this means the j's previous iteration.

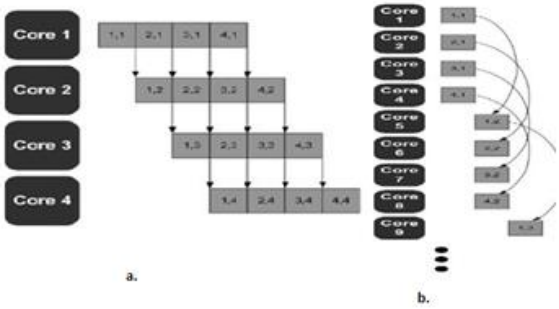

Fig. 12 (a). separate row of a matrix to separate core and each row wait until the first cell gets read. (b). First cell of each row to its own core but all other cores are waiting until the first entry of each row has been processed.

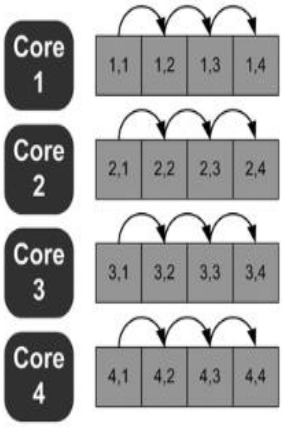

Fig. 13. 4 Cores each with the independent row.

So "the loop distance for i and j is [0,1]" but if there is slightly change in a code as shown below than dependency becomes 2 on j instead of one and "the loop distance for j is also 2" as Figure 14[14] which shows that 2nd line does not have to wait for the 1st row because it does not depend on it.

for j =1 to 4
Begin
for i=1 to 4
Begin
add the (i, j-2)th value to the (i, j)th value
End
End

According to the above code the third row should wait for the first line, so we can continue to parallel with the more core then work will require half time for the previous example. Looping is an important measure of data synchronization because it is a core productive data and the second is not

immediately consuming. Before consumption of data, the person who is going to consume might have to stay for a series of iterations. It depends on how things are parallel to the things. During the waiting, the producer may continue its repetition and so writes more data. For example, such data can be written in "first-in first- out (FIFO)" and the distance of the loop finds the length of FIFO. Let's apply the previous example on 4-core at the place of eight cores (Figure 15 a and b).

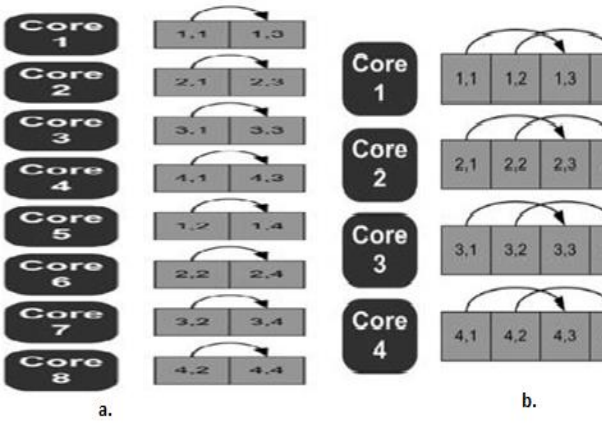

Fig. 15 (a). loop distance is 2 when execution with 8 cores. (b). loop distance 2 when execution with 4 cores.

Although, the third row should wait for the first line (Figure 15) but we can continue to parallel with the more core. If we want, then work in the required half time for the previous example. Although it may look dark, looping is an important measure of data synchronization. This is a core productive data and the second is not immediately consuming. Before consumption of data, the consuming person may have to wait for a series of iterations. It depends on how things are parallel to the things. During the waiting, the producer may continue its repetition and so writes more data. For example, such data can be written in "first-in first- out(FIFO) memory, and the distance" of the loop finds the length of FIFO. Let's apply the previous example on 4-core at the place of eight cores as in Figure 15.b shows that as soon as cell[1,1] accessed it should with the "cell [1,2], but cell[1,3] requires the result of cell[1,1] which is a case of an anti-dependency"[14].

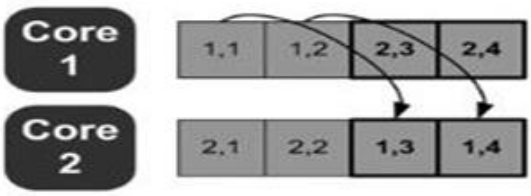

Fig. 16 two cores out of 4 have different assignment of cells.

As shown above, we can apply it only on an array where each core, keep the entire outcome separately. Though, in multi-core, the determination of which "thread get the thread" done by the operating system so if each cell is formed as a thread then the codes can be allocated in different manner. "For example, the first two cores can replace the last two cells (Figure 16)[7]". Now Core1 will have to give its results to Core 2 (and so on). Its solution is that the core1 [1, 1] puts the result of a safe place until it is ready for [1, 3]. Then [1, 2] can continue and when it is done then Core 2 can get the necessary result. However, [1, 2] results are ready even before preparing for core1, 2 [outcome]. [1, 2] The result cannot be kept in the same place as the only [1,1] result, otherwise it will be overwritten.

"One type of FIFO structure is used between core 1 and core 2 solutions (Fig. 18) [7]. Since the loop space for j is 2, the FIFO must be at least 2 deep to avoid burning. The solution is also effective for any thread assignment capable of creating an operating system, rather than strictly encoding an array implementation, using a FIFO. "

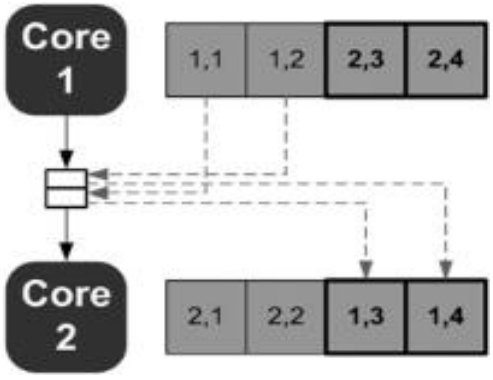

Fig 17. FIFO mode of communication between cores

FIFO can be expensive sometimes but it depends on its implementation. Here it is for

understanding the loop clearly because "manual determination" of loop deletion can be quite puzzling. A loop can have many variables in the body, with each separate loop distance. Branches make things more complicated [14].

4. SHARING OF RESOURCES

Another major challenge with the concurrency is that it may possible that unlike jobs may require the similar resources simultaneously the same time and so challenges for a multithread program on the single-core system are same in all threads for this reason use of some important segments and locks and their use is implemented but this solution used in multi-threaded programs may not work effectively for multi-core systems. Let's take a simple example of the method to stop another thread from breaking an important section of code is to suspend the block ages in this important section. It can work for the same core, but it will not work if another core reaches a shared space (or loss). In multi-core there is a new concept of having its private cache by each core and the repeated universal data in the cache which may not be sometimes in synchronization with the latest version of data which results in the complications because cash consistency approaches can be difficult itself, as distinct platforms have dissimilar schemas so a developer can overlook the cache on the single-core system but not in case of multi-core systems. Synchronization between the arbitrary numbers of cores in their access to the shared memory is necessary. The task allocation problem in multi-core processor based massively parallel systems [23].

5. CONCLUSION AND FUTURE WORK

Most of the confronts in multi-core processing happen due to parallelism and the fact that dissimilar things can be simultaneous. When we are used for events occurring in a given order, there may be a slight mental gymnastics to consider that two operations can be arbitrarily in relation to each other in two different parallel threads. There are many more algorithms other than FIFO to organize data in multiple core processors, so further research can be conducted on the basis of other algorithms and compare.

REFERENCES

[1] Creeger, M. "Multicore CPUs for the masses", Queue, 3(7), 64,2005

[2] Darlington, J., Ghanem, M., Guo, Y., & To, H. W. Guided resource organization in heterogeneous parallel computing. Journal of High-Performance Computing, 4(10), 13-23,1997.

[3] Geer, D. "Chip makers turn to multi-core processors". Computer, 38(5), 11-13, 2005

[4] Geer, D. "For programmers, multicore chips mean multiple challenges". Computer, 40(9), 17-19, 2007

[5] Gorder, P. F. (2007). Multicore processors for science and engineering. Computing in science & engineering, 9(2), 3-7, 2007

[6] Kahle, J.," The cell processor architecture". In 38th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'05), pp. 3. IEEE. 2005

[7] Knight, W. "Two heads are better than one [dual-core processors]". IEE Review, 51(9), 32-35,2005

[8] Merritt, R, "Multicore Goals Mesh at Hot Chips". EETimes Online, 2007

[9] Merritt, R. "Multicore puts screws to parallel-programming models". EETimes Online, 2008.

[10] Merritt, R "X86 war cuts to the cores". Electronic Engineering Times-Manhasset-, 1494, 1, 2007

[11] Moyer, B "The Promise and Challenges of Concurrency", Real World Multicore Embedded Systems ,11-31, Newnes,2013

[12] Muthana, P. Swaminathan, P., Tummala, R., Sundaram, V., Wan, L., Bhattacharya, S. K., & Raj, P. M. "Packaging of multi-core microprocessors: Tradeoffs and potential solutions". In Proceedings Electronic Components and Technology, 2005. ECTC'05, IEEE. 1895-1903,2005

[13] Ni, Jun. "Multi-core Programming for Medical Imaging", 2013

[14] Oshana, R. "Software Engineering of Embedded and Real-Time Systems. Software Engineering for Embedded Systems", 1–32,2013, doi:10.1016/b978-0-12-415917-4.00001-3

[15] R. Merritt, "CPU Designers Debate Multi-core Future", EETimes Online, 2008.

[16] Rouse, M. "Definition: multi-core processor". TechTarget.,2013

[17] Schauer, B., "Multicore processors–a necessity". ProQuest discovery guides, 1-14,2008

[18] Shrout, R., Intel Shows 48-core x86 Processor as Single-chip Cloud Computer. URL: http://www. pcper. com/reviews/Processors/Intel-Shows-48-

corex86-Processor-Single-chip-Cloud-Computer ,2012.

[19] Sodan, A. C., Machina, J., Deshmeh, A., Macnaughton, K., & Esbaugh, B. "Parallelism via multithreaded and multicore CPUs". Computer, 43(3), 24-32.2010

[20] Suleman, A., "What makes parallel programming hard, Future Chips",2013.

[21] Roy, A., Xu, J., & Chowdhury, M. H, "Multi-core processors: A new way forward and challenges". In 2008 International Conference on Microelectronics ,454-457, IEEE,2008

[22] Stoif, C., Schoeberl, M., Liccardi, B., & Haase, J," Hardware synchronization for embedded multi-core processors", 2011 IEEE International Symposium of Circuits and Systems,2011, doi:10.1109/iscas.2011.5938126

[23] Liu, Y., Zhang, X., Li, H., & Qian, D." Allocating Tasks in Multi-core Processor based Parallel System", 2007 IFIP International Conference on Network and Parallel Computing Workshops,2007