International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 04 | Apr 2020 www.irjet.net p-ISSN: 2395-0072

PRODUCER-CONSUMER PROCESS SYNCHRONIZATION IN MULTICORE SYSTEM AND ENERGY PROFILE

Sahana P K[1], Soumya A[2]

[1]*Under Graduate Student, Department of Computer Science Engineering, RV College of Engineering, Karnataka, India*
[2]*Associate Professor, Department of Computer Science Engineering, RV College of Engineering, Karnataka, India*

Abstract : *Interaction between processes producer and consumer or parent and child process usually is time consuming. The consumer waits for the producer to produce the buffers and populate them so that it can use the populated buffer. This usually results in a convoy effect, wherein one process that keeps a crucial segment is forestalled, other processes on separate processors that wait for the buffer cannot continue. There is wastage of time as consumer process does not start until producer populates the buffers. Since the advent of the multicore era, coordination between producer and user is the most perfect fit. The consumer and producer classical adapt to these architectures and enables strong task and data parallelism to be accomplished. Hence this model should be improved further by bringing about non-blocking synchronous implementation and develop a dynamic algorithm for the multiple producer-consumer problem, in which consumers in a many core structures use learning mechanisms to predict creation rates of items and thereby reduce energy use. Hence producer consumer algorithm is useful in multiple scenario, bringing about synchronization between processes and reducing overall energy consumption and brings about efficient utilization of computer resources.*

Keywords: Producer/consumer synchronization, non-blocking, synchronization

1.0 INTRODUCTION

Heterogeneous Systems involve more than one core to efficiently process a task. There are lot of real time cases which has high computation needs that can be only met only with usage of the multiple cores available. This is a typical problem in parallel computing, where 2 procedures share a common buffer, the producer and the consumer. Because these processes operate concurrently, they will coordinate to avert deadlocks and race conditions [2].

The benefits of non-blocking synchronization can be seen in a variety of applications operating on top of modern multiprocessors by using them on a wide range of applications with various communication characteristics, meaning that applications not spending any time in synchronization are also used, as well as adjusted lock-based synchronization points of such applications where necessary. The main advantage of non-blocking synchronization is seen in sporadic applications. While the significance of these implementations is projected to grow in the future, it is also anticipated to increase the value of lock-free synchronization in high-performance parallel systems [9].

The Producers (parent process) are process that work on the input information and generate outputs that will then be submitted to the Customers that is child process [3]. Consumers(child process) is one who obtain the information from the parent, work on the information and then generate outcomes that will either retained in the main memory or transferred to the next customer in producer - consumer chain (and thus therefore play the function of the parent process)[3]. Thus Producer-Consumer model can be implemented in multicore processor.

Producer consumer program can be implemented in multiple ways [2][4]:

- Mutex uses variables to indicate when data is accessible to the producer and when for consumer.
- Semaphores implementation for synchronizing the fullness and emptiness us es 2 semaphores.
- Batch processing (BP) is like applying a semaphore, only that the process pauses until the shared memory region is complete and then executes all the objects in the common shared buffer in one batch.
- Periodic Batch Processing (PBP) is identical to BP execution in that the user executes the batch within a set stretch frame.

Each can bring about synchronization between processes in different way and have different energy usage.

2.0 SYNCHRONIZATION

Heterogeneous Systems usually involve more than one core to efficiently process a task. Each subsystem has its own process to carry out or run, there might exist dependencies between the subsystem. Some subsystem might need to wait for another subsystem to finish some tasks before it can resume working or begin working on that related task. This works typically like producer consumer setup; producer allocates buffer memory and populates the buffers until then consumer process won't begin, this idle waiting time can be removed if synchronization is brought about between processes. Critical section is a section of code that can be gained access by only one process at a time. It contains shared data which need to be synchronized to maintain consistency and validity of data. In this case the common buffer is the critical section. If both the processes access the same memory, there is a chance that the final value in the buffer is incorrect, all the race processes agree that their performance is false, and this phenomenon is known as race phenomenon. Many processes simultaneously view and process the manipulations over the same records, so the outcome depends on the order the view takes place in.

Mutual exclusion comprises of bringing together activities into critical sections that are not once interweaved during program execution, thus guaranteeing that other processes do not get view of the unpredictable states of a certain method]. Condition synchronization postpones process until the system state satisfies some stated condition [1]. Consider one case where communication is every so often realized through a shared buffer between a source process and the target process. The writer(producer) writes in the buffer; from the buffer the recipient reads. The previous is used to guarantee no interpretation of a partly written buffer. The latter guarantee that a shared data is not overwritten into, and that a shared data is not read over once [5].

Concurrent software execution leads to series of atomic acts per each operation. History is a specific execution of a program that is equal to the orders of atomic acts generated by the processes. Note that the number of likely histories in the number of atomic acts is exponential. An abstract way of characterizing the possible histories created by a program is by using a programming logic to construct a proof of correctness [1][5]. A convenient way of expressing such a proof is through a proof description consisting of the program text scattered with declarations.

Consider an atomic statement B, it is preceded and succeeded by a statement.

{A} B{C}

This means that if execution of B is started only when A finishes, and C starts after B finishes. A is considered the precondition of B and C is considered the postcondition of B [1]. Therefore, B is used as a predicate transformation as it converts the condition from one where A is valid to one where C is valid [5].

Semaphores are abstract data structures on which each illustration is handled by two functions defined below. These functions have condition that number of times eq1 is completed is never more than number of times eq2 is completed [1]. The sem, is semaphore whose value is determined by number of times P and V is executed.

P(sem): :(awaits>O+sem=sem-1) eq (1)
V (sem): :(sem =sem+1) eq (2)

A special type of semaphore is binary semaphore, the condition is number of times P is executed is 1 more than V at the most [1][5].

P(bins): (await bins>0 →bins: =bins -1),
V(bins): (await bins < 1 → bins: = bins + 1).

In the question of producers/consumers, producers submit messages which consumers receive. The processes interact by means of a shared mutual buffer which is controlled by two operations: deposit and fetch [1]. Deposit is called when producer has to insert message and fetch is called to obtain message by consumer. The deposit and fetch should alternate with first being the former, this condition is there so that the message isn't made inconsistent by overwriting [5].

The important part is the beginning and finishing of execution of deposit and fetch [1][5]. Therefore, enterD and leaveD are integer values which list amount of occasions that producers have begun and finished deposit execution similarly enterF and leaveF are integer values which list the amount of occasions that producers have begun and finished fetch execution

PC: enterD <= leaveF + 1 ^ enterF <= leaveD [5].

In terms, this implies deposit may be begin at most one more time than fetch has been finished, and fetch cannot start more times than deposit has been done [1].

variable buffer: X // for some type X
variable E, F: semaphore = 1 & 0 // Invariant 0 s E + F < 1
ProducerProcess [a: l.. A]: while true→ item c produced
deposit: P(E)
buffer: = c
V(F)

ConsumerProcess [b: l.. B]: while true→ fetch: P(F)
c: = buffer
V(E)
Item c consumed

Both full and empty are semaphores in the solution. In fact, they create a split binary semaphore together. In binary semaphore the initial value of one of the semaphores is one [5]. The part of program which had to me synchronized is in between P and V, hence mutual exclusion is brought about. When 1 process enters this region the value of semaphore will be 0 hence ensuring no other process can enter this region.

3.0 PROGRAMMING MODEL FOR MULTICORE ARCHITECHTURE

The two major paradigms for leveraging multi-core system parallelism are: parallel data paradigm, and parallel task paradigm. The data is divided into subsets of sequential computing set of instructions, each of which is then processed separately by a processing machine. Programs representing this approach gain a large degree of parallelism, often reaching super-linear speed-ups due to re-use of the cache rows. Nonetheless, system dependencies are popular due to the complexity of the current frameworks and it is not often feasible to split the data.

Within the parallel task model, each processor unit is committed to executing a particular task. Such tasks may either be subtasks within a program, or multiple instances within the same system. Nonetheless, according to the data parallel model, it is often challenging to define these activities, or it is not even feasible to perform separate instances of the same program due to a lack of data [6].

In P/C model program will be divided and categorized into producer or consumer role [3]. The Producers are tasks that perform on the input data and generate results that later submitted to the Consumers. Consumers are the functions that obtain the data from the producers, perform some job on the data and then generate the tests that can either be retained in the main memory or transferred to the subsequent user [3][7].

This concept is used in various scenarios. The Producer supervises the compilation of all data activities in a GUI structure while the Customer utilizes this occurrence to execute the necessary behavior. The Processor distributes the frames among a collection of Customers in an MPEG-4 video encoder, which encodes them.

4.0 PRODUCER -CONSUMER ENERGY PROFILE AND OPTIMIZATION

Energy consumption is a very important factor which should be considered, and as this producer consumer model is used in diverse ways, in depth analysis of energy usage by different producer consumer application should be done. In this paper energy consumption is measured using two approaches: PowerTop1 and RunningAveragePowerLimit (RAPL).

- PowerTop is a prominent software which makes use of counters for processor output to estimate the power usage of all processes operating on device [2]s.
- RAPL, an app built to track and regulate the power usage of different Intel CPUs.

Testing is performed using a M: M:1: B queue dependent virtual dataset. Meaning 1 customer is present for each producer and output and processing periods are exponential in nature, and items are buffered in a buffer of size B [4].

The 5 features measured in each experiment:

- The energy consumption/energy profile
- Number of Wake-ups
- CPU usage
- CPU idle percentage
- CPU average frequency percentage

Mutex and Semaphore deployment of wakeups are identical in amount as is their energy usage. Batch processing has a relatively large amount of wake-ups, which can be interpreted by customers waiting for the buffer to be full, ensuing in a longer idle time, an ability that the dynamic power management (DPM) utilizes to bring the process to sleep. PBP achieves a solution by implementing a regular process(consumer) restart, which decreases the amount of wake-ups by around 49 percent, because there are quicker idle cycles and therefore fewer ability to bring the machine to sleep [2]. This reduction in the amount of wake ups contributes to greater usage, as the CPU has little hope of sleeping so long.

When staying in lower frequencies, Mutex and Semaphore have small ratios, with a 3 GHz jump. Therefore, they use comparatively large energy likened to batch-based implementations. In comparison, BP invests much of the day at the lowest 800MHz CPU frequency (18.6 per cent). PBP invests far less energy on frequencies in range of 800 MHz PBP is better than BP because, at higher frequencies CPU consumes less energy relative to BP. Wake up is done only when buffer is filled in BP thereby the average frequency at which CPU runs is higher [2]. The nature(periodic), though, keeps that from occurring, because it always stimulates consumers to buy a limited amount of products, and the

consumers may soon be idle again. This makes it difficult for the CPU to ascend to higher frequencies. PBP's wake-up figures are fewer than BP's. PBP finishes processing the dataset marginally quicker, owing to the regular consuming of objects.

The effect is a correlation of 0.73 between the CPU frequency (weighted average) and both power and energy, and a correlation of 0:71 with the amount of wake-ups per second and energy / power [2].
Therefore, the major disparity in energy usage between batch implementation and all other implementations is attributed to the lower average frequency induced by the higher number of wake-ups. Response time latency is the major drawback of BP. Implementations for Mutex and Semaphore-based are often less latentious. Hence, plus points of both, less latency and energy efficiency can be combined [4].

In a multicore system, this producer consumer problem, each consumer is allied only with 1 producer, a innovative energy-efficient algorithm is suggested [4]. This method is focused on complex, periodic batch processing, since consumers are processing a series of products and enabling the CPU to move to idle mode, thereby saving resources. Consumers forecast the amount of incoming data products, then work together themselves. This contributes to two energy reducing effects:

- the total amount of wake-ups is pointedly condensed compared the algorithm in question, and
- High loads are mitigated and evenly spread stopping Dynamic Voltage and frequency scaling (DVFS) processes from supporting the CPU frequency, resulting in the Processor consuming more of the time at lower frequencies [4].

This algorithm has been observed and it can lessen energy consumption by 40 percent compared to the other 2 when running multiple (5-10) consumers. In datum, it offers up to 18 percent enhancement over a simple BP execution. It is a detected that this algorithm outshines when number of consumers are more, hence becoming more scalable and robust [2][4].

REFERENCES

[1] Andrews, G.R. "A method for solving synchronization problems", Science of Computer Programming, 1989
[2] R. Medhat, B. Bonakdarpour and S. Fischmeister, "Energy-Efficient Multiple Producer-Consumer," in IEEE Transactions on Parallel and Distributed Systems, vol. 30, no. 3, pp. 560-574, 1 March 2019.
[3] "Architecture of Computing Systems – ARCS 2013", Springer Science and Business Media LLC, 2013
[4] Ramy Medhat, Borzoo Bonakdarpour, Sebastian Fischmeister. "Energy-Efficient Multiple Producer-Consumer", IEEE Transactions on Parallel and Distributed Systems, 2019
[5] Gregory R. Andrews. "A method for solving synchronization problems", Science of Computer Programming, 1989
[6] Lecture Notes in Computer Science, 2013.
[7] Arnau Prat-Pérez, David Dominguez-Sal,Josep-Lluis Larriba-Pey, Pedro Trancoso."Chapter 10 Producer-Consumer: Programming Model for Future Many-Core The Processors", Springer Science and Business Media LLC, 2013 Programming Model for Future Many-Core
[8] Yi Zhang. "Evaluating the performance of nonblocking synchronization on shared-memory multiprocessors", Proceedings of the 2001 ACM SIGMETRICS international conference on Measurement and modeling of computer systems - SIGMETRICS 01 SIGMETRICS 01,2001
[9] "Languages and Compilers for Parallel Computing", Springer Science and Business Media LLC, 2013
[10] Fred B. Schneider. "On Concurrent Programming", Springer Science and Business Media LLC, 1997