International Journal of Scientific Research in Computer Science, Engineering and Information Technology
ISSN : 2456-3307 (www.ijsrcseit.com)
doi : https://doi.org/10.32628/IJSRCSEIT

A Multithreading Based Enhanced Process Scheduling Technique for Heterogeneous Distributed Environment

Krishan Kumar[1], Renu[2]
[1]Assistant Professor, Department of CSE, JCDM College of Engineering, Sirsa
[2]M.Tech. Scholar, Department of CSE, JCDM College of Engineering, Sirsa

Article Info
Volume 7, Issue 5
Page Number: 125-129

Publication Issue :
September-October-2021

Article History
Accepted : 15 Oct 2021
Published : 30 Oct 2021

ABSTRACT

Multithreading is ability of a central processing unit (CPU) or a single core within a multi-core processor to execute multiple processes or threads concurrently, appropriately supported by operating system. This approach differs from multiprocessing, as with multithreading processes & threads have to share resources of a single or multiple cores: computing units, CPU caches, & translation lookaside buffer (TLB). Multiprocessing systems include multiple complete processing units, multithreading aims to increase utilization of a single core by using thread-level as well as instruction-level parallelism. Objective of research is increase efficiency of scheduling dependent task using enhanced multithreading. gang scheduling of parallel implicit-deadline periodic task systems upon identical multiprocessor platforms is considered. In this scheduling problem, parallel tasks use several processors simultaneously. first algorithm is based on linear programming & is first one to be proved optimal for considered gang scheduling problem. Furthermore, it runs in polynomial time for a fixed number m of processors & an efficient implementation is fully detailed. Second algorithm is an approximation algorithm based on a fixed-priority rule that is competitive under resource augmentation analysis in order to compute an optimal schedule pattern. Precisely, its speedup factor is bounded by $(2-1/m)$. Both algorithms are also evaluated through intensive numerical experiments. In our research we have enhanced capability of Gang Scheduling by integration of multi core processor & Cache & make simulation of performance in MATLAB.

Keywords : MATLAB, computing units, CPU caches, translation lookaside buffer

I. INTRODUCTION

MULTITHREADING

The multithreading paradigm has become more popular as efforts to further exploit instruction-level parallelism have stalled since late 1990s. This allowed concept of throughput computing to re-emerge from more specialized field of transaction processing; even though it is very difficult to further speed up a single thread or single program, most computer systems are

actually multitasking among multiple threads or programs. Thus, techniques that improve throughput of all tasks result within overall performance gains.

Types of multithreading:

Block multithreading

The simplest type of multithreading occurs when one thread runs until it is blocked by an event that normally would create a long-latency stall. Such a stall might be a cache miss that has to access off-chip memory, that might take hundreds of CPU cycles for data to return. Instead of waiting for stall to resolve, a threaded processor would switch execution to another thread that was ready to run. Only when data for previous thread had arrived, would previous thread be placed back on list of ready-to-run threads.

Interleaved multithreading

The purpose of interleaved multithreading is to remove all data dependency stalls from execution pipeline. Since one thread is relatively independent from other threads, there is less chance of one instruction within one pipelining stage needing an output from an older instruction within pipeline. Conceptually, it is similar to preemptive multitasking used within operating systems; an analogy would be that time slice given to each active thread is one CPU cycle.

Simultaneous multithreading

The most advanced type of multithreading applies to superscalar processors. Whereas a normal superscalar processor issues multiple instructions from a single thread every CPU cycle, within simultaneous multithreading (SMT) a superscalar processor could issue instructions from multiple threads every CPU cycle. Recognizing that any single thread has a limited amount of instruction-level parallelism, this type of multithreading tries to exploit parallelism available across multiple threads to decrease waste associated with unused issue slots.

Implementation specifics

A major area of research is thread scheduler that must quickly choose among list of ready-to-run threads to execute next as well as maintain ready-to-run & stalled thread lists. An important subtopic is different thread priority schemes that could be used by scheduler. The thread scheduler might be implemented totally within software, totally within hardware, or as a hardware/software combination.

Another area of research is what type of events should cause a thread switch: cache misses, inter-thread communication, DMA completion, etc.

II. LITERATURE REVIEW

Yeh-Ching Chung wrote on **"Applications & Performance Analysis of A Compile-Time Optimization Approach for List Scheduling Algorithms on Distributed Memory Multiprocessors"**

They have proposed a compile-time optimization approach, *bottom-up top-down duplication heuristic* (BTDH), for static scheduling of *directed+cyclic graphs* (DAGS) on *distributed memory multiprocessors* (DMMs). In this paper, they discuss applications of BTDH for *list scheduling algorithms* (LSAs). There are two ways to use BTDH for LSAs. BTDH can be used with LSAto form a new *scheduling* algorithm (LSA/BTDH). It could be used as a pure *opti*mization algorithm for a LSA (LSA-BTDH). We have applied BTDH with two well known LSAs,*highest level first with estimated time* (HLFET)&*earlier taskfirst*(ETF) heuristics. We have performed extensive simulation to study performance of BTDH for LSAs. Three parameters, *graph parallelism* (GP) of a DAG,ratio of average communication cost to average computation cost *(CCR)* of a DAG&number(PN) of a DMM, ares imulated. From simulation, they have following conclusions. Given a DAG,GP of DAG could accurately predict number of processors to be used such that a good scheduling length &a good resource utilization (or efficiency) could be achieved .

Maruf Ahmed, Sharif M. H. Chowdhury wrote on" List Heuristic Scheduling Algorithms for Distributed Memory Systems with Improved Time Complexity"

They present a compile time list heuristic scheduling algorithm called *Low Cost Critical Path algorithm (LCCP)* for distributed memory systems. LCCP has low scheduling cost for both homogeneous & heterogeneous systems. In some recent papers list heuristic scheduling algorithms keep their scheduling cost low by using a fixed size heap &a FIFO, where heap always keeps fixed number of tasks & excess tasks are inserted within FIFO. When heap has empty spaces, tasks are inserted within it from FIFO. Best known list scheduling algorithm based on this strategy requires two heap restoration operations, one after extraction& another after insertion. Our LCCP algorithm improves on this by using only one such operation for both.

Ishfaq Ahmad1&Yu-Kwong Kwok2 wrote on "On Parallelizing Multiprocessor Scheduling Problem"

Existing heuristics for scheduling a node& edge weighted directed task graph to multiple processors could produce satisfactory solutions but incur high time complexities that tend to exacerbate within more realistic environments with relaxed assumptions. Consequently, these heuristics do not scale well& cannot handle problems of moderate sizes. A natural approach to reducing complexity while aiming for a similar or potentially better solution is to parallelize scheduling algorithm. This could be done by partitioning task graphs& concurrently generating partial schedules for partitioned parts, that are then concatenated to obtain final schedule. The problem, however, is nontrivial as there exists dependencies among nodes of a task graph that must be preserved for generating a valid schedule. Moreover, time clock for scheduling is global for all processors (that are executing parallel scheduling algorithm), making inherent parallelism invisible. In this paper, they introduce a parallel algorithm that is guided by a systematic partitioning of task graph to perform scheduling using multiple processors. The algorithm schedules bothtasks & messages, &is suitable for graphs with arbitrary computation & communication costs &is applicable to systems with arbitrary network topologies using homogeneous or heterogeneous processors. They have implemented algorithm on Intel Paragon &compared it with three closely related algorithms. The experimental results indicate algorithm yields higher quality solutions while using an order of magnitude smaller scheduling times. The algorithm also exhibits an interesting trade-off between solution quality & speedup while scaling well with problem size.

Wayne F. Boyer wrote on "Non-evolutionary algorithm for scheduling dependent tasks within distributed heterogeneous computing environments"
The Problem of obtaining an optimal matching & scheduling of interdependent tasks within distributed heterogeneous computing (DHC) environments is well known to be an NP-hard problem. In a DHC system, task execution time is dependent on machine to which it is assigned &task precedence constraints are represented by a directed acyclic graph. Recent research within evolutionary techniques has shown that genetic algorithms usually obtain more efficient schedules that other known algorithms.

They propose a non-evolutionary random scheduling (RS) algorithm for efficient matching& scheduling of inter-dependent tasks within a DHC system. RS is a succession of randomized task orderings &a heuristic mapping from task order to schedule. Randomized task ordering is effectively a topological sort where outcome may be any possible task order for which task precedent constraints are maintained. A detailed comparison to existing evolutionary techniques (GA&PSGA) shows proposed algorithm is less complex than evolutionary techniques, computes schedules within less time, requires less memory &fewer tuning parameters. Simulation results show

that average schedules produced by RS are approximately as efficient as PSGA schedules for all cases studied &clearly more efficient than PSGA for certain cases. extraction &insertion, that within theory reduces scheduling cost without compromising scheduling performance. In our experiment they compare LCCP with other well known list scheduling algorithms &it shows that LCCP is fastest among all.

III. RESEARCH METHODOLOGY

3.1 OBJECTIVE:

1. To analyze previous design algorithms for task scheduling and find the limitations.
2. Researcher to design new algorithm for task scheduling.
3. Implementation and deployment of proposed algorithm for providing better performance.
4. comparison between previously developed algorithm and proposed one .

3.2 CHALLENGES WITHIN RESEARCH

Multiple threads could interfere with each other when sharing hardware resources such as caches or translation look aside buffers (TLBs). As a result, execution times of a single thread are not improved but could be degraded, even when only one thread is executing, due to lower frequencies or additional pipeline stages that are necessary to accommodate thread-switching hardware. Overall efficiency varies; Intel claims up to 30% improvement with its Hyper Threading technology,[1] while a synthetic program just performing a loop of non-optimized dependent floating-point operations actually gains a 100% speed improvement when run within parallel. On other hand, hand-tuned assembly language programs using MMX or Altivec extensions &performing data pre-fetches (as a good video encoder might) do not suffer from cache misses or idle computing resources. Such programs therefore do not benefit from hardware multithreading& could indeed see degraded performance due to contention for shared resources. From software standpoint, hardware support for multithreading is more visible to software, requiring more changes to both application programs &operating systems than multiprocessing. Hardware techniques used to support multithreading often parallel software techniques used for computer multitasking of computer programs. Thread scheduling is also a major problem within multithreading.

IV. IMPLIMENTATION

PROPOSED WORK

Choosing a scheduling algorithm

When designing an operating system, a programmer must consider which scheduling algorithm would perform best. There is no universal “best” scheduling algorithm,& several operating systems use extended or combinations of scheduling algorithms above.

Operating system process scheduler implementations

The algorithm used may be as simple as round-robin within which each process is given equal time (for instance 1 ms, usually between 1 ms&100 ms) within a cycling list. So, process A executes for 1 ms, then process B, then process C, then back to process A.

More advanced algorithms take into account process priority, or importance of process. This allows some processes to use more time than other processes. The kernel always uses whatever resources it needs to ensure proper functioning of system, &so could be said to have infinite priority. In SMP(symmetric multiprocessing) systems, processor affinity is considered to increase overall system performance, even if it may cause a process itself to run more slowly. This generally improves performance by reducing cache thrashing.

In computer science, thrashing occurs when a computer's virtual memory subsystem is within a constant state of paging, rapidly exchanging data within memory for data on disk, toexclusion of most application-level processing. This causes performance of computer to degrade or collapse. The situation may continue indefinitely underlying cause is addressed. The term is also used for various similar phenomena,

particularly movement between other levels of memory hierarchy, where a process progresses slowly because significant time is being spent acquiring resources.

Scope of research

If a thread gets a lot of cache misses, other threads could continue taking advantage of unused computing resources, that may lead to faster overall execution as these resources would have been idle if only a single thread were executed. Also, if a thread cannot use all computing resources of CPU (because instructions depend on each other's result), running another thread may prevent those resources from becoming idle. If several threads work on same set of data, they could actually share their cache, leading to better cache usage or synchronization on its values .

V. CONCLUSION

Scheduling of a task and deployment on processers in heterogeneous environment is very complex. We will make assumption and implement our proposed algorithm in abstract environment.

VI. REFERENCES

[1]. Abraham Silberschatz, Peter Baer Galvin & Greg Gagne (2013). Operating System Concepts 9. John Wiley & Sons,Inc. ISBN 978-1-118-06333-0.

[2]. Yeh-Ching Chung and Sanjay Ranka, Applications and Performance Analysis of A Compile-Time Optimization Approach for List Scheduling Algorithms on Distributed Memory Multiprocessors, 1063-953Y92 $3.00 0 1992 IEEE

[3]. Ishfaq 5. Wayne F. Boyer, Gurdeep S. Hurab, Non-evolutionary algorithm for scheduling dependent tasks in distributed heterogeneous computing environments, J. Parallel Distrib. Comput. 65 (2005) 1035 - 1046

[4]. Ahmad and Yu-Kwong Kwok, On Parallelizing the Multiprocessor Scheduling Problem,1998

[5]. Maruf Ahmed , Sharif M. H. Chowdhury and Masud Hasan, List Heuristic Scheduling Algorithms for Distributed Memory Systems with Improved Time Complexity.

Cite this article as :

Krishan Kumar, Renu, "A Multithreading Based Enhanced Process Scheduling Technique for Heterogeneous Distributed Environment", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 7 Issue 5, pp. 125-129, September-October 2021. Journal URL : https://ijsrcseit.com/CSEIT217543