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

Enhancing Capability of Gang scheduling by integration of Multi Core Processors

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

Article Info
Volume 8, Issue 3
Page Number : 408-411

Publication Issue :
May-June-2022

Article History
Accepted: 03 June 2022
Published: 15 June 2022

ABSTRACT

In this paper, a new algorithm for gang scheduling is proposed. This method aims to reduce the average response time of gangs by increasing the serviceability of gangs in the shortest execution time possible. The performance of the proposed algorithm is examined and compared to the basic gang scheduling algorithm within the simulation.

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: Multiprocessor, Multi Core Processor, Gang Scheduling, MATLAB

I. INTRODUCTION

Gang scheduling is a scheduling algorithm for parallel systems that schedules related processes or threads to run simultaneously on different processors. These processes or threads of each job are packed into a single row of the matrix. Gang scheduling is a combination of time-sharing and space-sharing approaches. Gang scheduling appeared as the solution to the problems of job scheduling policies in those systems where the processor scheduling was a simple dispatch. In this kind of systems, the main problem seems to be the fragmentation, then reasons to use gang scheduling were presented as responsiveness and

efficient use of resources All processes belonging to a job run at the same time (the term gang denotes all processors within a job). Each process runs alone on each processor. but there is rapid coordinated context switching. It is possible to suspend/preempt jobs arbitrarily. Gang Scheduling Working :The different gangs are grouped in time slots following some re-packing algorithm. The total number of processors requested by gangs in a time slot must be less or equal than the total number of processors of the machine. Periodically, at each quantum expiration, the scheduler selects a new time slot to execute all of its gangs. If the workload has changed during the execution of the last quantum, the re-packing algorithm will be re-applied. In any case,the new slot selected is scheduled.

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 optimization 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

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

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 both tasks & 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.

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

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.

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.

III.Research Methodology

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.

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. Implementation : 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.

V. 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.

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

[6]. Remzi H. Arpaci-Dusseau; Andrea C. Arpaci-Dusseau (January 4, 2015). "Chapter 7: Scheduling: Introduction, Section 7.6: A New Metric: Response Time". Operating Systems: Three Easy Pieces (PDF). p. 6. Retrieved February 2, 2015.

Cite this article as :

Krishan Kumar, Suman, "Enhancing Capability of Gang scheduling by integration of Multi Core Processors", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 8 Issue 3, pp.408-411 , May-June 2022.

Journal URL : https://ijsrcseit.com/CSEIT2283106