© INTERNATIONAL JOURNAL FOR RESEARCH PUBLICATION & SEMINAR
ISSN: 2278-6848 | Volume: 07 Issue: 06 | July – September 2016
Paper is available at www.jrps.in | Email : info@jrps.in

A MULTITHREADING BASED ENHANCED PROCESS SCHEDULING TECHNIQUE FOR HETEROGENEOUS DISTIBUTED ENIVORNMENT

Sujata, Research Scholar, Department of CSA, CDLU Sirsa, sujata.sanauswap@gmail.com
Vikram singh, professor, Department of CSA, CDLU Sirsa, sujata.sanauswap@gmail.com

ABSTRACT: In computer architecture, 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.

[1]INTRODUCTION

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.

[2] 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 scheddhg algorithms (LSAs). There are two ways to use BTDH for LSAs. BTDHcan be used with a LSA to form a new scheduling algorithm (LSA/BTDH). It could be usedas apure optimization algorithm for a LSA (LSA-BTDH)..

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. 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.

Joël Goossens1 and Pascal Richard wrote on "Optimal Scheduling of Periodic Gang Tasks".

The 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. They propose two DPFair (deadline partitioning) algorithms that schedule all jobs in every interval of time delimited by two subsequent deadlines. These algorithms define a static schedule pattern that is stretched at run-time in every interval of the DP-Fair schedule. The first algorithm is based on linear programming and is the first one to be proved optimal for the considered gang scheduling problem. Furthermore, it runs in polynomial time for a fixed number m of processors and an efficient implementation is fully detailed. The 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.

[3] RESEARCH METHODOLOGY

In computing, scheduling is method by which work specified by some means is assigned to resources that complete work. The work may be virtual computation elements such as threads, processes or data flows, that are within turn scheduled onto hardware resources such as processors, network links or expansion cards.

© INTERNATIONAL JOURNAL FOR RESEARCH PUBLICATION & SEMINAR
ISSN: 2278-6848 | Volume: 07 Issue: 06 | July – September 2016
Paper is available at www.jrps.in | Email : info@jrps.in

A scheduler is what carries out scheduling activity. Schedulers are often implemented so they keep all computer resources busy (as within load balancing), allow multiple users to share system resources effectively, or to achieve a target quality of service. Scheduling is fundamental to computation itself, &an intrinsic part of execution model of a computer system concept of scheduling makes it possible to have computer multitasking with a single central processing unit (CPU). Researchers have made assumptions and then implemented the algorithm to schedule the processes. Multithreading and gang scheduling technique are used separately, we will combine these two approaches and study the result. First we will ahead with simple processes scheduling using multithreading and then specifies it with one considering one scheduling algorithm.

[4] CHALLENGES WITHIN RESEARCH

Multiple threads could interfere with each other when sharing hardware resources such as caches or translation lookaside 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 l00% speed improvement when run within parallel. Another hand, hand-tuned assembly language programs using MMX or Altive 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.

Gang scheduling can employ either fixed or dynamic partitioning for CPUs. Fixed partitioning divides CPUs into disjoint subsets, and jobs are dispatched to be scheduled within the subset. This can introduce fragmentation if gang sizes are not equal. Dynamic partitioning allows partitions to change sizes (move CPUs from one partition to another) but this requires complex synchronization of context switching across the entire parallel system. It is advised to perform repartitioning only when new job arrives into the system (not with every context switch).

While designing DHC system it was assumed that moving gangs across different partitions introduces overhead that cannot be justified by the expected performance gain. Gang migration policy proposed in results in decreased response time. Decrease ranges from l4% to 67% depending on the system load. Another issue with gang scheduling is memory consumption. In order for jobs to run efficiently, paging in the system should be kept at minimum because it drastically affects thread synchronization. Admission control can prevent jobs to be gang scheduled if the system is running to low on memory.

© INTERNATIONAL JOURNAL FOR RESEARCH PUBLICATION & SEMINAR
ISSN: 2278-6848 | Volume: 07 Issue: 06 | July – September 2016
Paper is available at www.jrps.in | Email : info@jrps.in

These jobs are queued up and FCFS with backfilling can be used for queue control. Gang scheduling prevents long resource consuming jobs from starving shorter jobs, thus resulting in decreased slowdown. Scheduled interactive jobs appear to have near real time performance due to time sharing nature of the algorithm. Time complexity of algorithm has been improved by multithreading but it also impacts the schedule length of Gang Heuristic.

[5] PROPOSED IMPLEMENTATION

D-P FAIR SCHEDULE

GANG HEURISTIC ALOGRITHM

We study the scheduling of preemptive periodic implicit-deadline rigid gang real-time

63 tasks. We address the performance by modifying DP fair scheduling polynomial time algorithm .We will run the algorithm in parallel environment and control the number of threads used by each cycle in algorithm. Modified version of algorithm implemented in matlab and compared with original.

Algorithm

input :

n jobs *Jj*(*uj* , *vj*), 1 _ *j* _ *n* ;

m: number of processors;

output :

Slice lengths *S*(*s*), *s* = 1, 2, . . .;

Scheduled jobs *j* in slice *s*: Sched(*s*,*j*) 2 {0, 1}, 1 _ *j* _ *n* ;

List=Sort(*J*1, . . . , *Jn*) ; /* Jobs are sorted in non increasing order of *vj* */

s = 0 ; /* Number of slices */

rj := *uj* 8*j* = 1 · · · *n* ; /* Job remaining execution times *rj* */

while 9*j*, *rj* > 0, 1 _ *j* _ *n* do

/* create a new slice */

s = *s* + 1 ; /* Number of slices */

K = *m* ; /* Remaining processors */

` = 1 ; /* Slice length upper bound */

Sched(*s*,*j*)=0 1 _ *j* _ *n* ;

Create virtual parallel multiprocessor environment

foreach *j* 2 List do

/* For each job *j* in priority List */

Foreach i=1to 4 /*4 parallel processors are considered*/

Control the number of threads

if *vj* _ *K* then

/* the job *j* is schedulable in current slice */

Sched(*s*,*j*)=1;

` = min(`, *rj*); /* update slice length */

K = *K* − *vj* ; /* remaining processors */

end

end

end

S(s)=l; /* Slice length */

for *j* = 1 . . . *n* do

/* Update remaining execution times */

if *Sched*(*s*, *j*) then

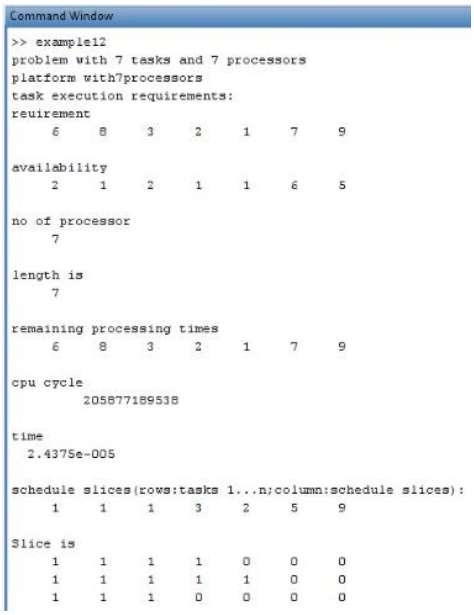
rj = *rj* − `;

end

end

end

© INTERNATIONAL JOURNAL FOR RESEARCH PUBLICATION & SEMINAR
ISSN: 2278-6848 | Volume: 07 Issue: 06 | July – September 2016
Paper is available at www.jrps.in | Email : info@jrps.in



```
Command Window
>> example12
problem with 7 tasks and 7 processors
platform with7processors
task execution requirements:
reuirement
     6     8     3     2     1     7     9

availability
     2     1     2     1     1     6     5

no of processor
     7

length is
     7

remaining processing times
     6     8     3     2     1     7     9

cpu cycle
         205877189538

time
  2.4375e-005

schedule slices(rows:tasks 1...n;column:schedule slices):
     1     1     1     3     2     5     9

Slice is
     1     1     1     1     0     0     0
     1     1     1     1     1     0     0
     1     1     1     0     0     0     0
```

Fig 1 Original Gang Heuristic Algorithm Output

Fig 2 Modified Gang Heuristic algorithm output

```
We are very interested in your feedback regarding these capabilities.
Please send it to parallel_feedback@mathworks.com.

Submitted parallel job to the scheduler, waiting for it to start.
Connected to a matlabpool session with 4 labs.
Sending a stop signal to all the labs...
Waiting for parallel job to finish...
Performing parallel job cleanup...
Done.
cpu cycle
         206477613428

time
    0.2661

schedule slices(rows:tasks 1...n;column:schedule slices):
     6     2     1     7     9

Slice is
     1     0     0     0     0
     1     1     0     0     0
     0     1     1     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1

schedule length:6  2  1  7  9
```

[6] 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.

By using Matlab as tool we can virtually take the four processors that work simultaneously.

This algorithm can also be implemented in real time heterogeneous environment.

[7] CONCLUSION

In this research, we have considered the preemptive scheduling of implicit-deadline periodic gang task systems upon identical multiprocessors. We have modified the existing gang heuristic algorithm which defines static patterns that are stretched at runtime in a DP-Fair way with aim to reduce the time complexity of algorithm. This has been implemented by multithreading and multiprocessing. It also briefed about and factors in parallel system scheduler design and gives a summary of existing scheduling algorithms, multithreading. When time complexity reduced it impacts the space complexity and other factors.

Workload (tasks) is classified with respect to parallelism, interactivity and migration capabilities. Types of input workload are clearly stated in the comparisons in cases in which standard benchmarks where not used. Current vendor offering is analyzed with respect to the scheduling algorithm used. There are many open questions in parallel job scheduling.

Some are of philosophic nature such as definition of effective system, while others consider utilization, scalability, job migration support, etc.

[8] REFERENCE

l. Abraham Silberschatz, Peter Baer Galvin & Greg Gagne (20l3). *Operating System Concepts 9*. John Wiley & Sons,Inc. ISBN 978-l-ll8-06333-0.

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*, l063-953Y92 $3.00 *0* l992 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) l035 – l046

4. Ahmad and Yu-Kwong Kwok, *On Parallelizing the Multiprocessor Scheduling Problem*,l998

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

6. Joël Goossens and Pascal Richard, *Optimal Scheduling of Periodic Gang Tasks*