www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 6 Issue 11 November 2017, Page No. 23219-23223
Index Copernicus value (2015): 58.10 DOI: 10.18535/ijecs/v6i11.22

Designing Ideal Task Partitioning Scheduling Model In Distributed Computing Scenario

Maninderjit Singh Khanna, Dr. Jitendra Sheetlani

Dept. of Computer Application,
Chitkara University,
Punjab, India
Dept. of Computer Application,
Sri Satya Sai University of Technology & Medical Science, Sehore, Bhopal (MP) India

Abstract:
This paper represents optimal iterative task partitioning scheduling in distributed heterogeneous environment. The main goal of the algorithm is to improve the performance of the schedule in the form of iterations. This algorithm first uses b-level computation to calculate the initial schedule. Main characteristics of our method are optimal scheduling and strong link between partitioning, scheduling and communication. Some important models for task partitioning have also been discussed in this paper. The proposed algorithm improves inter process communication between tasks by using recourses of the system in an efficient manner.

Keywords: Task partitioning scheduling; distributed heterogeneous environment; inter process communication.

Introduction

Parallel computing systems compose task partitioning strategies in a true multiprocessing manner. Such systems share the algorithm and processing unit as computing resources to achieve higher inter process communications capabilities A large variety of experiments have been conducted on the proposed algorithm. Goal of computational models is to provide a realistic representation of the costs of programming.

Scheduling techniques might be used by an algorithm to optimize the code that comes out from parallelizing algorithms. A researcher is always keen to construct a parallel algorithm that runs in the shortest time. Another use of these techniques is the designing of high-performance computing systems. Threads can be used for task migration dynamically [1]. They are used to increase the efficiency of the algorithm. Parallel computing systems have been implemented upon heterogeneous platforms which comprise different kinds of units, such as CPUs, graphics co-processors, etc. An algorithm has been constructed to solve the problem according to the processing capabilities of the machines [10]. Communication factor is highly important feature to solve the problem of task partitioning in the distributed systems. A computer cluster is a group of computers working together closely in such a manner that it's treated as a single computer. The cluster is always used to improve the performance and availability over that of a single computer. A cluster is used to improve the scientific calculation capabilities of the distributed system [2]. Task partitioning has been achieved by linking the computers closely to each other as a single implicit computer.

Large tasks partitioned into sub tasks by the algorithms to improve the productivity and adaptability of the systems. Process division is a function that divides the process into the number of processes or threads. Thread distribution distributes threads proportionally among several machines in the cluster network [15]. A thread is a function which executes on the different nodes independently, so the communication cost problem is negligible [3]. Some important models [4] for task partitioning in a parallel computing system are: PRAM, BSP etc.

I. Parallel Random Access Machine (PRAM) model

It is a robust design paradigm provider. PRAM comprised of P processors, each with its own un- modifiable program. A single shared memory composed of a sequence of words, each of which capable of containing an arbitrary integer [5].

PRAM model is an extension of the familiar RAM model and it's used in algorithm analysis. It consists of a read-only input tape and a write-only output tape. Each instruction in the instruction stream has been carried out by all processors simultaneously. It requires unit time, reckless of the number of processors. Parallel Random Access Machine (PRAM) model of computation consists of a number of processors operating in lock step [13]. In this model each processor has a flag that controls whether it is active in the execution of an instruction or not. Inactive processors do not participate in the execution of instructions.

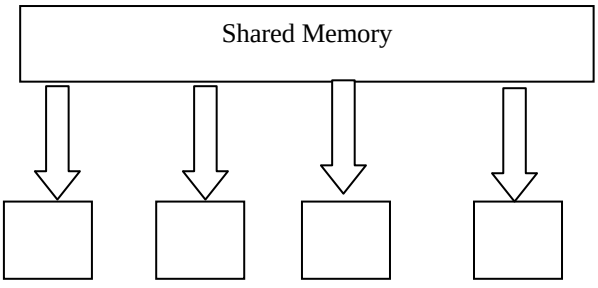

Figure 1: PRAM model for shared memory.

The processor ID can be used to distinguish processor behavior while executing the common program. Synchronous PRAM produces results by multiple processors to the same location in shared memory. Highest processing power of this model can be used by using the Concurrent Read Concurrent Write (CRCW) operation. It's a baseline model of concurrency and explicit model which specify operations at each step [11]. It allows both concurrent read and concurrent write instructions to shared memory locations. Many algorithms for other models (such as the network model) can be derived directly from PRAM algorithms [12].

Classification of the PRAM model is as follows:

1. In the Common CRCW PRAM, all the processors must write the same value.

2. In the Arbitrary CRCW PRAM, one of the processors arbitrarily succeeds in writing.

3. In the Priority CRCW PRAM, processors have priorities associated with them and the highest priority processor succeeds in writing.

III. PROPOSED MODEL FOR TASK PARTITIONING IN DISTRIBUTED ENVIRONMENT SCHEDULING:

Task partitioning strategy in parallel computing system is the key factor to decide the efficiency, speedup of the parallel computing systems. The process has been partitioned into the subtasks where the size of the task is determined by the run time performance of each server [9]. In this way assign a number of tasks will be proportional to the performance of the server participated in the distributed computing system. Inter-process communication cost amongst tasks is very important factor which is used to improve the performance of the system [6]. Inter processes communication cost estimation criteria is important for the enhancement of speed up and turnaround time [8]. Call Procedure (C. P.) is used to dispatch the task according to the capability of the machines. In this model server machine can be used for large computations. Every processing element executes one task at a time and all tasks can be assigned to any processing element. In the proposed model, subtasks communicate to each other by sharing of data. So execution time is reduced due to sharing of data. These subtasks assign to the server which dispatch the tasks to the different nodes.

The proposed scheduling algorithm is used to compute the execution cost and communication cost of the tasks. So the server is assumed by a system (P, [Pij], [Si], [Ti] [Gi] [Kij]) as follows:

a) P = {Pi... Pn}, Where Pi denotes the processing elements on cluster

b) [Pij], where NxN is processors topology

c) Si, 1 < i < N, specify the speed of processor Pi

d) Ti, 1 <i< N, specify the startup cost of initiating message on Pi

e) Gi, 1 < i < N, specify the startup cost of initiating process on Pi

f) Kij is the transmission rate over the link connecting to adjacent processors Pi and Pj

Total (t)	Total Task
DAG(H)	DAG Height
P	Number of Processor
MinCT	Minimum Computational Time
MaxCT	Maximum Computational Time
MinCR	Minimum Computational Rate
MaxCR	Maximum Computational Rate
ψ	Speed up
b-level	Bottom Level of DAG
E	Serial execution portion of algorithm

Table 1: Nomenclature for proposed task partitioning model

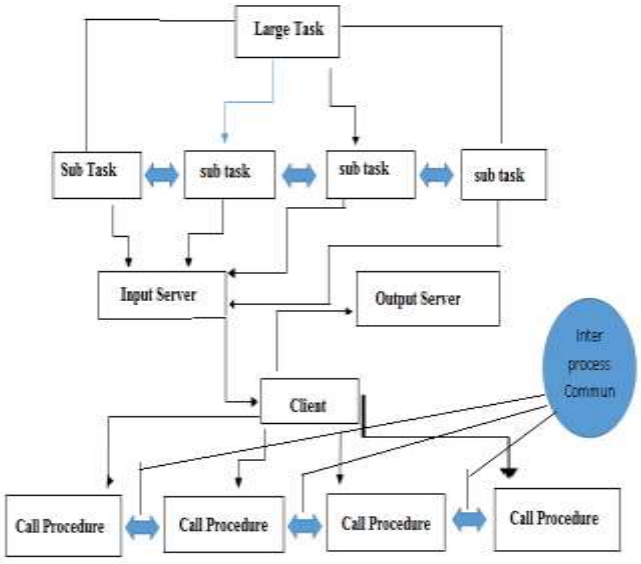

Figure 2: Proposed dynamic task partitioning model

This model splits both computation and data into small tasks [14].

The following basic requirements have been fulfilled by the proposed model:

1. There is at least one order of magnitude more primitive than processors upon the target machine to avoid later design constraints.
2. Redundant data structure storage and redundant computations are minimize which cause to achieve

large scalability for high performance computations.

3. Primitive tasks are roughly of the same size to maintain the balance work among the processors.
4. Number of tasks is increasing function of the problem size which avoids the constraints. It's impossible to see more processors to solve large problem instances.

The model comprises the existence of an I/O (Input/Output) element associated with each processor in the-system. Processing time can be calculated with the help of the Gantt chart. The connectivity of the processing element can be represented by using an undirected graph called the scheduler machine graph [7]. Program completion cost can be computed as:

Total Cost = Communication cost + Execution cost

where, Execution cost = Schedule length, Communication cost = the number of node pairs (w, μ) such that (w, μ) € A and proc (w) = proc (μ).

III. THE PROPOSED ALGORITHM FOR INTER-PROCESS COMMUNICATION AMONGST TASKS:

It's an optimal algorithm for scheduling interval ordered tasks on (m) processors. The algorithm generates a schedule *f* that maps each task v € V to a processor Pv with a starting time tv. Communication time between the processor Pi and Pj, may be defined as:

comm.(i,j) = {O for i =j, otherwise 1}

task-ready (μ, i, *f*): the time when all the messages from all task in N(v) have been received by processor Pi in schedule f .

start time (μ, i, *f*): the earliest time at which task v can start execution on processor Pi in schedule f .

proc (μ, *f*): the assigned processor to task μ in schedule *f* .

start (μ, *f*): the time in which task μ begins its actual execution in schedule *f* .

task (μ,T, *f*): the task schedule on processor P, at time T in schedule *f* . If there is no task schedule on processor Pi at time T in schedule *f* , then task(μ,T, *f*) returns the empty task ϕ. It's assumed that n2(ϕ) < n2(μ).

In this algorithm edge cut gain parameter is considered to calculate the communication cost amongst the tasks [9].

gain(i,j) = €.gain edge cut + (1 - €)

gain edge cut = edge cut actor / old edge cut

edge cut factor = old edge cut - new edge cut

Where € is used to set the percentage of gains from edge-cut and workload balance to the total gain. Higher value of € contribute total gain of the communication cost.

V. PSEUDO CODE FOR THE PROPOSED ALGORITHM:

1. start

2. task (i, τ, f)←Φ, for all positive integers i, where 1 < i <P and τ > 0

3. repeat

4. Let μ be the unmark task with highest priority

5. for i = 1 to P do

6. compute b-level for all tasks

7. schedule all tasks into non-increasing order of b-level

8. compute ALAP, constructs a list of tasks in the ascending order of the ALAP time

9. task_ready (μ, i, f ← max(start(μ, f) + comm(proc(μ ,f), i + 1) + gain(i,j) for each μ

10. start_time(μ, i, f)← min τ, where task(i,τ,f)← Φ and t > task_ready (μ,i,f)

11. endfor

12. f(μ) ← (start_time(μ, i, f) if

13.start_time(μ,i,f)<(start_time(μ,j,f), 1 < j < P, i ≠ j or

start_time(μ,i,f)=(start_time(μ,j,f) and

n2(task(i, (start_time(μ,i,f) - 1), f) ≤ n2(task(j, (start_time(μ,i,f) - 1), f)

1 ≤ j ≤P, i ≠ j.

14. mark task μ until all tasks marked

15.endif

VI. LOW COMMUNICATION OVERHEAD PHASE:

Optimality of the algorithm over the target machine can be achieved due to the following reasons:

Fact(l): comm(i, T1,j,T2) where 1 ≤ i,j ≤ P

Swapping of the task by the task schedule on processor node (nj) at time (T1) with the task schedule on node (nj) at time (T2). When the swapping of the task amongst the different processor then

Fact (2): total comm(i,j, T) where 1 ≤ i,j ≤P

The effect of the above operation is to swap all the task schedule on node (ni) at time T1 with the task schedule node (nj) at time T2, where T2 ≥ T1.

The following operation is equivalent to the more than one swap operations:

Fact (3): total comm(i,j,T) ~ comm(I, T1, j, T2) □ TI,T2 ≥ T

VII. PRIORITY ASSIGNMENT AND START TIME COMPUTING PHASE:

Computation of the b-level of DAG is used for the initial scheduling. The following instructions have been used to compute the initial scheduling cost of the task graph:

1. Construct a list of nodes in reverse order(Li)

2. for each node ai Ɛ LI do

3. max = 0

4. for each child ac of ai do

5. lf c{ai, ac) + b-level(ac) >M then

6. M = c{ai, ac) + b-level(ac)

7. endif

8. endfor

9. b-level(ai) = weight(ai) + M

10. endfor

In the scheduling process b-level is usually constant until the node has been scheduled. Procedure computes b-level and schedules a list in descending order. The quantitative behavior of the proposed strategy depends upon the topology used on the target system. This observation might lead to the conclusion that b-level perform best results for all experiments. The algorithm employs the attribute ALAP (As Late as Possible) start time which measure that how far the node's start time can be delayed without increasing schedule length.

VIII. PROCEDURE FOR COMPUTING THE ALAP IS AS FOLLOWS:

1. construct ready list in reverse topological order (Mi)

2. for each node aj Ɛ Mj do

3. min = k , where k is call procedure(C.P.) length

4. for each predecessor ac of ai do

5. if alap(ac) - c(ac, ai) < k then

6. k = alap(ac) - c(ac, a,)

7. endif

8. endfor

9. alap(ai) = k - wgt (a,)

10. endfor

According to priority of nodes, tasks allocated on the processors in distributed computing environment. The ALAP time is computed and then constructs a list of tasks in ascending order of ALAP time. Ties have been broken by considering ALAP time of predecessors of tasks.

The following results from the above facts prove the optimality of the proposed model:

1. The operation comm(i, τi, j, τ2) on the schedule f of the tasks preserves the feasibility of the schedule of any task (w)

   f(w) = (p, τi) where p ϵ {i,j} and τi = τ-1

2. Feasibility of the schedule f in the proposed model increased for any task schedule

   f(w) = (p, τi) where p ϵ {i,j] and □ τi

3. The operation conim(i, τi, j, τ2) and n2(total comm(i,j,τ)) > n2(task(i, j, τ)) shows optimality on the schedule of any task(w)

   f(w) = (p, τ3) where p≠ [i,j} and □ T3

4. The operation comm(i,,j,τ) preserves feasibility of the schedule of any task(w)

   f(w) = (p, τi) where p Ɛ [i,j] and τi ≤ τ -l

5. The operation conim(i,j,τ) also shows optimality of the schedule of any task(w)

   f(w) = (p, τi) where p≠ [i,j} and □τi

In this paper, we proposed a new model for the estimation of communication cost amongst various nodes at the time of the execution. The improvement ratio of iterations has also been discussed. Our contribution gives cut edge inter-process communication factor which is a highly important factor to assign the task to the heterogeneous systems according to the processing capabilities of the processors on the network. The model can also adapt the changing hardware constraints.

IX. REFERENCES:

1. N. Islam, A. Prodromidis and M. S. Squillante, "Dynamic Partitioning in Different Distributed Memory Environments," Proceedings of the 2nd Workshop on Job Scheduling Strategies for Parallel Processing, (1996), 155-170.
2. D. J. Lilja, "Experiments with a Task Partitioning Model for Heterogeneous Computing," University of Minnesota AHPCRC Preprint no. 92-142, Minneapolis, MN, (1992), 15-19.
3. L. G. Valiant, "A Bridging Model for Parallel Computation," Communications of the ACM, 33(8), (1990), 103-111.
4. B. H. Juurlink and H. A. G. Wijshoff, "Communication primitives for BSP Computers," Information Processing Letters, 58, (1996), 303-310.

5. H. EI-Rewini and H. Ali, "The Scheduling Problem with Communication," Technical Report, University Of Nebraska at Omaha, (1993), 78-89.

6. D. Menasce and V. Almeida, "Cost-Performance Analysis of Heterogeneity in Supercomputer Architectures," Proc. Supercomputing, (1990), 169-177.

7. T. L. Adam, K. M. Chandy and J. R. Dickson, "A Comparison of List Schedules for Parallel Processing Systems," Comm. ACM, 17, (1974), 685- 689.

8. L. G. Valiant, "A Bridging Model for Parallel Computation," Communications of the ACM, 33(8), (1990), 103-111.

9. El-Rewini, T. G. Lewis, H. Ali, "Task Scheduling in Parallel and Distributed Systems," Prentice Hall Series in Innovative Technology, (1994), 48-50.

10. M. D. Ercegovac, "Heterogeneity in Supercomputer Architectures," Parallel Computing, (7), (1987), 367-372.

11. P. B. Gibbons, "A More Practical PRAM Model," In Proceedings of the 1989 Symposium on Parallel Algorithms and Architectures, Santa Fe, NM,(1989), 158-168.

12. Y. Aumann and M. O. Rabin, "Clock Construction in Fully Asynchronous Parallel Systems and PRAM Simulation," In Proc. 33rd IEEE Symp. On Foundations of Computer Science, (1992), 147-156.

13. R. M. Karp and V. Ramachandran, "Parallel Algorithms for Shared-Memory Machines," In J. van Leeuwen, editor. Handbook of Theoretical Computer Science, (1990), 869-941.

14. H. Topcuoglu, S. Hariri and M. Y. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," IEEE Trans. Parallel and Distributed Systems, 13(3), (2002), 250-271.

15. J. Ali and R. Z. Khan, "Dynamic Task Partitioning Model in Parallel Computing Systems," Proceeding First International conference on Advanced Information Technology (ICAIT), Coimbatore, Tamil Nadu, (2012), 7-10.