**Static Scheduling of Graph Tasks on Homogeneous Multiprocessor Systems Using Task Duplication**

**By-**

**Chetan There.**

**Devendra Umbrajkar.**

**Index**

|  |  |
| --- | --- |
| **Topic** | **Page Number** |
| 1. **Abstract** | **2** |
| 1. **Introduction** | **2** |
| 1. **Problem Statement** | **4** |
| 1. **Overview of Techniques**   **-Exploiting Task Duplication in parallel Program Scheduling (CPFD : Critical Path Fast Duplication)**  **-** **Task Duplication Using Partial Schedules**  **-** **Economical Duplication Based Task Scheduling for Heterogeneous and Homogeneous Computing Systems**  **-** **Task scheduling algorithm using minimized duplications in Homogeneous systems**  **-** **An Optimal Scheduling Algorithm based on Task Duplication.** | **5**  **5**  **9**  **14**  **18**  **21** |
| 1. **Conclusion** | **25** |
| 1. **References** | **26** |

**Static Scheduling of Graph Tasks on Homogeneous Multiprocessor Systems Using Task Duplication**

Abstract:

Scheduling of Graph Tasks on multiprocessor systems is well known problem in computer science. It’s been proved to be NP complete except for few restricted cases based on assumptions. Various techniques and algorithms have been proposed to solve this problem. Different static approaches used to solve this problem mainly are list based, cluster based and task duplication. Task duplication approach try to reduce communication time among tasks by duplicating tasks on the same processor and thus try to reduce overall make span. We studied different task duplication based algorithms and presented quantitative comparison in this report.

We implemented CPFD and RD task duplication techniques, experimentally evaluated simulation results and compared results with respect to different parameters.

Introduction:

* Background

In parallel processing environment we can run multiple tasks at a time. These tasks share resources and have dependencies among them. Thus effective scheduling and mapping of these tasks to proper processor is required to reduce total completion time and number of processor required to complete execution of given tasks.

Various techniques were adopted to solve this problem. Mainly these techniques can be divided as static and dynamic. In static scheduling we assume that computation and communication time among processes are known at compile time while in dynamic approach these are calculated at run time. Static approaches are preferred for most of the cases since for may applications like DSP(digital signal processing ) parameters are well known at compile time and it also reduces run time overhead. In case of static scheduling various approaches used are list based, clusters based and task duplications. In list based approach tasks are store in descending order of their priority or schedule in some sophisticated order to reduce make span. Cluster based techniques try to put all tasks that have more dependencies on same cluster thus try to reduce communication time among tasks. Task duplication is relatively new approach which try to duplicate some of the tasks such that communication time get reduce and hence try to reduce overall make span. Though some of the tasks can run more than once and will consume resources but this is better approach since we can reduce overall completion time which is essential for many applications. It’s best suited for fine grained application where communication time is much more than computation time of individual tasks.

Many algorithms are published on task duplication strategy. They varies in terms of approach used for assigning priorities to tasks, task duplication strategies and mapping of tasks to available processors. Some algorithms considers unlimited number of processors while some restricted to limited number of processors. Overall aim of these approaches is to

1) Reduce make span

2) Minimize task duplication

3) Minimize number of processors required.

* Motivation

Well known algorithms proposed for task duplications are as below

1. DSH : This algorithm assign priorities based on S level(largest computation cost from node to exit node).It considers nodes in descending order of their priorities and try to find out free time slot on processor and try to duplicate parent which can be accommodate in available time slot and if it’s is successful then updates start time.
2. BTDH : It is similar to DSH but it continue to duplicate parents even if start time of given task increases assuming start time will reduce eventually.
3. LCTD : This is cluster based approach which forms cluster and then map each cluster on different processor. Main idea to put all more dependant tasks in common cluster.
4. CPFD : This algorithm first computes critical path in a given DAG i.e. largest path from entry node to exit node considering both computation and communication time. It divide nodes of DAG as CPN, IBN and OBN with respect to critical path. CPN are nodes on critical path, IBN are parents of CPN which determined start time of CPN and OBN are the nodes which are neither CPN nor IBN. Main idea here CPN are most important node which determines make span so should be scheduled first followed by IBN which are of second importance and at last OBN. Every node is assigned with priority based on it’s type, b level and t level values. These tasks then scheduled in descending order of their priorities and mapped to processor which can provide earliest start time for that task considering duplication. Duplicated tasks can be immediate parent or at any next level tasks which provides better start time.

Most of the times CPFD provides optimal scheduling and outperforms other algorithms. Our study report is focussing on new algorithms proposed and their quantitative comparison with CPFD.

We implemented CPFD and RD task duplication techniques and experimentally evaluated and compared results with respect to different parameters. Different parameters considered are as below.

* Input Parameters

1. Size of DAG
2. CCR

* Output Parameters

1. Make Span
2. Number of Processors Require
3. Number of Duplicates
4. Processor Consumption

We considered DAG with different sizes (different number of node) and CCR(Communication to Computation Ratio) and calculated values of output parameters mentioned above. Results of CPFD and RD have been calculated, compared and graphs have been plotted.

* Problem Statement

With advancement in parallel technologies using clusters/distributed nodes, lot of research are going on scheduling of tasks in parallel environment. The parallel application is generally represented by Directed Acyclic Graph (DAG). DAG is weighted directed graph given as G = {N,E,w,c}, consist of nodes from {n1,n2….nv} and edges(e1,e2…em). The value on each node represents computation cost of that task which is given by w (ni). Directed edge (ninj) represents task dependencies of node nj on node ni and weight on edge represents communication time between tasks which is given by C (ninj).

Fig 1 represents typical DAG with 9 nodes.

Some of the important terminologies in DAG are given below.

* Source node of the edge is known as the parent node and the node to which the edge points is known as the child node.
* A node without a parent is known as the entry node and a node without the child is known as the exit node.
* The Communication to Computation Ratio (CCR) of a program is defined as the ratio of total communication cost to total computation cost on a given system.
* S-level is the computation cost along the longest directed path from the concerned node to the exit node.
* B-level is the sum of computation cost and communication cost along the directed path from the concerned node to the exit node.
* T-level is the sum of computation cost and the communication cost along the directed path from the entry node to the concerned node excluding the latter’s computational cost.
* Earliest Start Time(EST) – A node ni can start execution only after it receives data from its parents. The EST is defined as the maximum value of DAT and the ready time of processor Pj.

EST (ni,Pj) = max {DAT (ni,,Pj), PjR} where PjR is the ready time of the processor Pj.

* Earliest Completion Time (ECT) – The ECT means the earliest execution finish time of node ni on processor Pj. ECT is represented as the sum of EST and computation time.
* Make span – It is maximum schedule length.

Some basic assumption of all the algorithms described in this report are general to assumptions in DAG. They are as below.

* A node cannot start execution till it has gathered all the messages required from its parents.
* If node scheduled on same processor then communication cost is zero.
* Contention free network and all task can do computation and communication together
* No Preemption of tasks

In general, the scheduling algorithm tries to schedule DAG tasks on given processors to achieve minimum make span with minimum use of processors.

* Overview of Studied Techniques:
* Exploiting Task Duplication in parallel Program Scheduling

(CPFD : Critical Path Fast Duplication)

Priority Assignment Technique

This algorithm first find our CP (critical path) in a given DAG which is largest path from start to exit node considering both computation and path communication time. Nodes along CP called CPN(Critical Path Node) and they determines overall span time so should be considered first while scheduling. These CPN can have dependencies on other non CPN nodes and cannot start before completion of those nodes. These parent nodes of CPN called IBN(In Branch Node) and OBN(Out Branch Node) these are neither CPN nor IBN. The main idea here is that CPN are most important node which determines make span so should be scheduled first followed by IBN which are of second importance and at last OBN. Every node is assigned with priority based on it’s type, b level and t level values. The algorithm is given as below[Ref 5]

Task Duplication Technique

Tasks are considered in order of their priorities and mapped to processor which provides earliest start time for the given task.It considers data arrival time and finish time of existing node on that processor. It also considers whether there is chance of improving start time by duplicating it’s parent on the same processor. Here it consider most important parent which is parent node that provides maximum data arrival time value to given node. Now MIP can be duplicated if there is proper time slot is available which is given as below[Ref 5]

ST(npi+1,P) – MAX{FT(npi ,P) , DAT(ny , P)} >= w(ny)

Where i=0,1……k ; ST(npk+1,P) =∞ and FT(n0p , P) = 0

The earliest start time of ny on P is then given by:

MAX{FT(njp,P) ,DAT(ny,P) }

After MIP this algorithm considers parents of MIP to check whether they can fit and reduce start time of node Ny. This process continued till it does not improve est of Ny on the given processor. Earliest start time of given task calculated on each processor and finally it maps to processor which gives smallest value among all processors.

This is given as in below algorithm [Ref 1]

Minimize\_start\_time(node,P):

1. Determine the ealrliest start time (EST) of node on processor P using rule 2.
2. If EST is undefined then quit because node cannot be scheduled on P.
3. Find out the VIP of node using Rule 1.
4. Minimise the start time of this VIP by recursively calling Minimize\_start\_time(VIP,P).
5. Compute the new EST of node. If the new EST does not improve then undo all the duplication just performed in step 4 schedule node to P and quit; otherwise find out the new VIP of node and repeat from step 4.

The CPFD algorithm can be given as below [Ref 1]

The Critical Path Fast Duplication Algorithm:

1. Determine a CP. Ties are broken by selecting the one with larger sum of computation costs. Construct the CPN-Dominant sequence.

Repeat

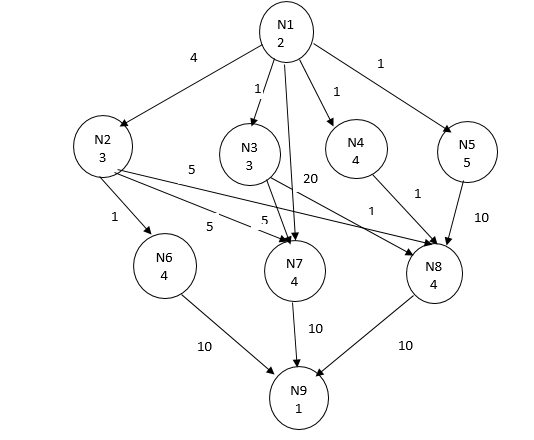
1. Let candidate be the first unschedules CPN in the CPN-Dominant sequence.
2. Let P\_SET be a set of processors, including all the processors holding candidates parent nodes and an empty processor.
3. For each processor P in P\_SET call Minimize\_start\_time(candidate,P) and record the EST of candidate on P.
4. Schedule candidate to the processor P that gives the smallest value of EST.

Until all the CPNs in the CPN-Dominant sequence are scheduled.

1. Repeat the process from step 2 to step 5 for each OBN in the CPN-Dominant sequence.

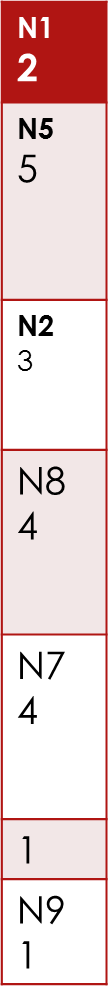
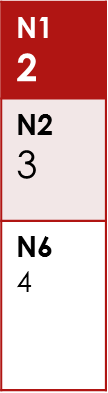
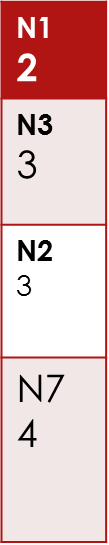
The scheduling of IBN is implicitly handled in the task duplication method.

[Figure Ref 1]



|  |  |  |
| --- | --- | --- |
| Node | S level | CPN-DS |
| N1 | 12 | 1 |
| N2 | 8 | 2 |
| N3 | 8 | 3 |
| N4 | 9 | 6 |
| N5 | 10 | 5 |
| N6 | 5 | 8 |
| N7 | 5 | 4 |
| N8 | 5 | 7 |
| N9 | 1 | 9 |

Gantt Chart can be Given as



* Task Duplication Using Partial Schedules

This is replication based two phase scheduling algorithm, In first phase it tries to reduce make span using partial schedules and in the second phase it tries to reduce number of processor required by eliminating and merging partial schedules.

The first phase called minSL aims to minimise schedule length by minimizing earliest finish time of each task on new processor. The given task on that processor is called partial schedule of that task. In the second phase called minNP it tries to remove and merge duplicated partial schedule such that there is no increase in the make span. This algorithm called DUPS claimed that it outperform CPFD in terms of make span and number of processors required.

minSL Phase : It assume unlimited number of processors are available and initially schedule each task on new processor and then parent (of all levels ) duplicated till there is reduction in make span. First all entry nodes scheduled on different processors. At next level any arbitrary node is considered as target node nt and try to generate it’s partial schedule. If nt is not a join node then it’s best partial schedule can be calculated by duplicating all it’s parents on new processor.it can start on its est(earlier start time).if node is join node then it will be first schedule on new processor say Pt at it’s est and then partial schedule of nt is try to improve by duplicating it’s parent on Pt. First we consider critical parent of nt, critical parent is the immediate parent of nt which has maximum communication cost among all immediate parents.

Duplication can be effective only when sum of computation time of tasks at Pt is smaller than any other partial schedule generated so far. If it is effective duplication then nc is scheduled on Pt at first idle slot after est(nc,Pt) or at st(ni,Pt) whichever is earliest. After duplication of nc on Pt all existing node on Pt will be send to rList and rescheduled. If nc is critical parent of other nodes other than ni then those also can start earlier. Also if nc have been copied as critical parent of some other node earlier on Pt and has start time higher than currently copied nc then in this case nc will be duplicated and such duplication can be removed. Tasks in rList are rescheduled in such a way that if processor is available at est of node then it start executing at its est otherwise start once currently allocated task on Pt gets complete. This algorithm believe that duplication of parents of non-critical child on Pt does not or slightly improve make span and thus neglecting such conditions.

Algorithm 1 MinSL [ref 3]

1. Schedule each entry task ns on a new processor Ps  with st(ns,Ps) 🡨 0
2. While there are unscheduled tasks do
3. nt 🡨 a randomly chosen task whose parents partial schedule have been determined
4. If nt  is a non-join task then
5. Copy its parents (np) partial schedule to Pt and set st(nt, Pt) 🡨 ft(np, Pt)
6. else
7. st(nt, Pt) 🡨 est(np, Pt)
8. ni 🡨 nt and nc 🡨 critical parent of ni
9. while nc exists and (SLt,current – idle timepn Pt) + wc < SLt,best do
10. compute est(nc, Pt) and schedule nc on Pt
11. put tasks scheduled after nc  into rlist
12. sort tasks in the rList and schedule on Pt.
13. if SLt,current < SLt,best then
14. Pt,best 🡨 Pt,current
15. Find the critical child ni and critical task nc

[ figure ref 3]

5

5

7

2

2

5

15

7

* minNP : Minimizing number of processors Used :

the minNP algorithm tries to minimize number of processor required without affecting schedule length. There are two steps, In first step redundant partial schedules are discarded and then remaining ones are merged as much as possible. All processors are considered one by one in decreasing order of their respective schedule length. Here it consider task which is completing before lft (latest finish time) and then it is marked as fix on that processor. Seperate array is maintained to keep track of such tasks on particular processors. Each task is fixed to at most one processor and there is no such restrictions on other duplicates of same task. If target task nt is an exit task on Pt then it is fixed to that processor. For non exit nt first we will find if there are dependent children are scheduled on that processor, if not then we can remove nt. If there are dependent child then lft(nt) is calculated and its duplicate on other processor Pl is checked. If at other processor nt is started before lft on current processor then nt mapped to Pl. Ties are broken by choosing less idle time on processor after execution of nt. nt on Pl will shift such that it can provide maximum lft.

Algorithm 2 elimProcs [ref 3]

1. for all Pt with non-increasing SLt order do
2. if target task nt on Pt is an exit task then
3. lft(nt) 🡨 SL; fixed (nt) 🡨 Pt
4. else
5. Dt = { Pl | (nt, nd ) belongs to Ɛ and ft (nt , Pl) > st(nd , Pl) }
6. Tk 🡨 start time of task after nt on Pk
7. If Dt = ɸ then
8. P 🡨 P – {Pt}
9. else
10. lf t(nt)← minPl ∈ Dt { st(nd , Pl )} – ct,d
11. if lft(nt) ≥ ft(nt, Pl) for Pl ∈ P such that SLl > SLt
12. and minPk∈ (P−{Pt}) {tk−ft(nt, Pk)} = tl − ft(nt, Pl) then
13. Pc ← Pl ; P ←P −{Pt}
14. else
15. Pc ← Pt
16. fixed(nt) ← Pc
17. ft(nt, Pk) ← tk for all Pk ∈ P −{Pc}
18. ft(nt, Pc) ← min{tc, lft(nt)}

Now further reduction in number of processor is possible by merging these partial schedules. It considers two processor at a time in descending order of the SL and tries to merge tasks keeping dependencies intact. Merging will be placed on temporary processor. dst of task been calculated for merging. Timer is maintained and it will be set to SL initially. Then next non considered task will be picked up from either of processor which is having dst greater than SL and merged with task on Pt. if Pt becomes non feasible then merging is cancelled and next pair is considered.

[ Figure ref 3]

2

1

3

1

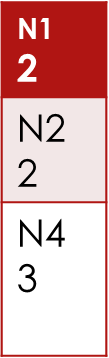
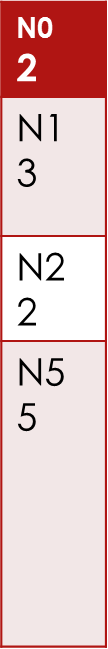
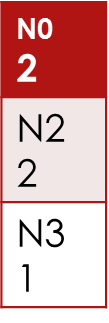
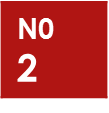
1

2

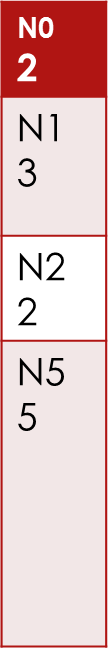
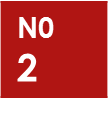
3

Better than CPFD in terms of number of tasks, number of parents and CCR.

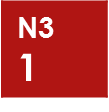
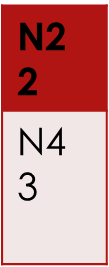
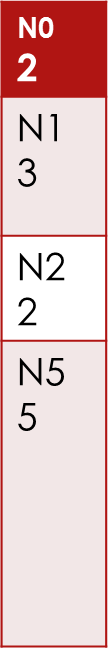
Gantt chart is as follows-



Algorithm 2



Algorithm 3



* Economical Duplication Based Task Scheduling for Heterogeneous and

Homogeneous Computing Systems

Two task duplication based algorithm have been proposed i.e. Reduced Duplication (RD) for homogeneous system and HED (Heterogeneous Economical Duplication) for heterogeneous system. We are considering study of only RD. Static task schedules generated based on insertion based task duplication and try to remove some duplicates whose removal does not affect much on make span. Earlier schedule of task can be useless if it’s been duplicated at later point and thus can be removed. Wholes generated by these removal can be used for scheduling of other tasks. It is claimed that RD generates better schedules with less number of duplications and using less number of processors as compared to SD and CPFD.

Reduced Duplication (RD):

In this algorithm priority is assigned same as in CPFD considering critical path. The predecessors of CPN are considered in decreasing order of b level and conflicts are removed by less t level value. After generating scheduling sequence of nodes tasks are mapped to processor which provides earliest start time as below.

Now DAT can be calculated as below

DAT(ni ,Pk) =maxnj Ɛ pred(nt) {min{ Fjk , Fjk + cji} }

And earliest start time can be calculated as

Sik  =max {DAT ( Mi ,Pk ) , min { Pkr , Grs }}

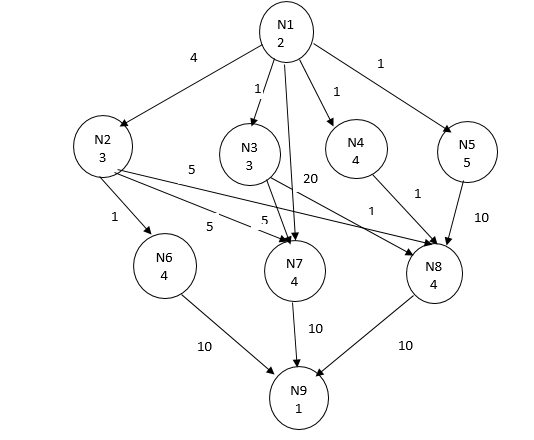
Where Grs  is the start time of first suitable scheduling hole Gr to accommodate task ni on the processor Pk , if exist .The finish time (Flk) is calculated as:

RD follows same method of scheduling and mapping as of CPFD. Further it maintains list A of all duplicated tasks and list B list of duplicated tasks in descending order of their earliest start time .If removal of task from list B does not affect make span much then it can be removed.

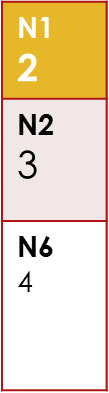
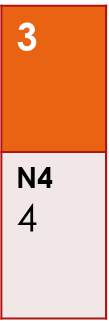
If we consider DAG and it’s CPFD Gant chart then we can remove duplicates as below.

1. It’s obvious that n7 on P1 is not useful and can be removed. Also we can remove N2 on P1 which has been duplicated for N7.
2. N1 on P4 can be removed without affecting make span.
3. Also N1 and N2on P3 can be removed without affecting make span.

[Figure ref 1]



Gantt chart is as follows



[ref 7]

Begin

1. Construct a priority based task sequence.
2. do {
3. select the first unscheduled task ni in the task sequence .
4. for (all Pk in all processor list P)
5. compute finish time Fik of ni on Pk and sort the list of immediate parents of ni in non-increasing order of data arrival time.
6. For all immediate parents, select the first immediate parent nj from the list at step 5 {
7. If duplication of nj can reduce the finish time Fik of task ni on Pk
8. Duplicate nj ;
9. }
10. Compute the earliest finish time Fik of ni on Pk
11. }
12. Find the minimum earliest finish time of task ni;
13. Assign task ni on processor Pk with minimum Fik in schedule S
14. } while (there are unscheduled tasks in the task sequence)
15. Maintain a list A of tasks which have been duplicated later and list B of duplicated tasks in non-increasing order of their earliest start time
16. for (each duplicated task ni in list B){
17. if (no change in makespan of schedule S after removing duplicated task ni)
18. remove this duplicated task ni from the schedule S and update list A;
19. }
20. for (each task ni in list A) {
21. if task ni is unproductive task in schedule S due to its duplication; remove this task ni from schedule S
22. }

End

Conclusion:

RD follows same method of scheduling and mapping as of CPFD. Afterwards it considers gant chart and try to remove duplicated nodes without affecting make span. It is improved CPFD since it offers less duplicates which means less use of resources. Also wholes generated after removing duplicates can be used to schedule next job.

* Task scheduling algorithm using minimized duplications in Homogeneous systems

In the paper, they have proposed an algorithm designed to allocate join nodes without redundant duplications, if the ancestor nodes of a join node are duplicated when scheduling the join node, the original allocations of these ancestor nodes are removed using a very efficient method.

The main function of the static scheduling algorithm is to minimize the completion time of a parallel program by mapping tasks to processors and ordering their executions. The main obstacles is the communication overhead that is there when the tasks executed on different processors exchange data.

Duplication based scheduling can help reduce the communication overhead by allocating some of the tasks to more than one processor. The objective is to prevent any redundant duplication from occurring when the join node is allocated to a processor, without causing any increase in the schedule length and time complexity. A set of k nodes {n1, n2. . . nk} that have already been scheduled on a processor Pj, we use FT(n0, Pj) = 0 to denote the finish time of the fictitious node, n0, and ST(nk+1, Pj) = 1 to denote the start time of another fictitious node, nk+1. A free time slot is suitable for node nd on processor Pj if the following condition is satisfied for some i, 0 < i < k.

ST (ni+1, Pj) − max {DAT (nd, Pj), FT (ni, Pj)} + w (nd).

Then, the start time of the earliest free slot, EFS (nd, pj), can be defined as follows.

EFS (nd, Pj) = max {DAT (nd, Pj), FT (nj, Pj)}

Where node ni is the first node that satisfies condition in the ordered set of nodes that have already been scheduled on processor Pj.

The algorithm is designed in 2 step, in the first step priorities are assigned to the nodes and the node with the highest priority is chosen for scheduling. In the second step the processor that allows the earliest start time is used to accommodate this node.

Classify each node according to the priority level in a bottom up fashion. The exit node has the lowest priority and the parents of the exit node have the higher priority. By applying this process recursively the entry node gets the highest priority. When there is a conflict between two nodes the nodes on critical path have higher priority ,if multiple critical path nodes with same priority level the ties are broken on the basis of higher computation costs and secondly on the basis of a higher number of immediate predecessors of CPNs. After considering the CPNs the remaining nodes with same priority are ordered on the basis of higher B-level and secondly on the basis of a lower T-level.

If a node, ni, has only one child and this child has multiple parents, in other words, the only child of node ni is a join node, the allocation of node ni is susceptible to be changed to a redundant duplication when allocating the join node at a later stage. If a join node and a set of its parent nodes satisfy the following two necessary conditions, redundant duplications can be generated.

1. A join node is the only child of at least one of its parents. That is, the join node has more than one parent and it is the only child of at least one of these parents.

2. The computation cost of a parent node is smaller than the communication cost between any one of the parents and the join node.

The above condition is referred to below as the redundant duplication condition (RDC). Note that a join node should have more than one parent for redundant duplications to be generated.

In the algorithm they first find out the set of tuples which satisfy the RDC. Denote the tuple as (J[i],Set\_RDC[i]) where J[i] is ith node in a given DAG and set\_RDC[i] is a set of its parents node. Then select the first unscheduled task from the scheduling list, that is , the candidate task nc, and allocate it to a processor, where duplications of ancestor nodes can be performed provided the duplications improve the EST of nc.

Scheduling Algorithm with Minimized Duplications [ref 5]

Assume that a set of available processors, Set\_AP is given

Begin

Construct Scheduling List based on Priority scheme.

Find the set of all tuples that satisfy RDC(J[i],Set\_RDC[i])

For(each candidate task nc belongs to LIST)

Call Duplication(nc,Pj) for all Pj belongs to Set\_AP.

Determine favourite processor Pj belongs to Set\_AP for nc

Undo duplications performed on the other processors except Pj

If nc belongs to J[i] for some i and nc does not belong to Set\_RDC[j] for any j

For each nk belongs to Set\_RDC[i]

Call removal nk, Pf.

Schedule nc on Pf at EST(nc,, Pf)

End

Duplication (nc,Pf)

Begin

If any new MIP(nc)

If condition ST(ni+1, pj) − max{DAT(nd, pj), FT(ni, pj)} >= w(nd) is satisfied for MIP(nc) on Pf

DetermineEFS(MIP(nc), Pj)

If EST(nc,Pj) is improved by duplication MIP(nc) on Pj{

Duplicate MIP(nc) on Pj and mark the duplicated MIP(nc) is for nc

Call Duplication MIP(nc,Pj)

Else exit.

Calculate EST(nc,Pj) and return it.

END

Removal (nd,Pf)

Begin

If nd is duplicated in Pf

Remove the original allocations of nd

Repeatedly remove any marked duplications related to the removed redundant duplications

Else nd belongs to j[i] for any i

For each nk  belongs to Set\_RDC[i]

Call removal(nk,pf)

End

[figure ref 5]

4

5

3

2

3

1

2

|  |  |  |
| --- | --- | --- |
| Node | EST | Priority |
| 1 | 0 | 6 |
| 2 | 1 | 4 |
| 3 | 4 | 3 |
| 4 | 6 | 2 |
| 5 | 1 | 5 |
| 6 | 8 | 1 |

In the above DAG we construct a scheduling list of the above nodes using the rules mentioned in priority level. The critical path for the above graph is n1-n2-n4-n6. We find all the tuples that satisfy the RDC properties. Those which satisfy the RDC properties are added to the Set\_RDC. Then we call duplication function for the processor from the set\_AP. Find the favourite processor to be duplicated, undo all the previous duplications. If the candidate node is does not belong to Set\_RDC and the parent belongs to Set\_RDC then remove the parent node and schedule the child node on processor at the EST.

* An Optimal Scheduling Algorithm based on Task Duplication.

Under the condition where the communication time is relatively shorter than the computation time for every task the task duplication based scheduling algorithm generates an optimal schedule.

Cluster refers to a set of tasks where the start and completion time of each task in the cluster are fixed that is, task ni in cluster C has stc(ni) and ctc(ni) as the start and completion time, respectively. We assume that the tasks are non preemptable. The interval (stc(ni),ctc(ni)) of task ni in cluster C should not be overlapped by any other task. The communication among tasks in the same cluster is assumed to be negligible because a cluster is mapped to one processor in task scheduling. The task scheduling problem is to allocate tasks to clusters in order to minimize the completion time of the exit task where tasks can be duplicated over multiple clusters. In this paper, they propose an optimal scheduling algorithm with a less restricted optimality condition than the TDS algorithm.

The main step of our scheduling process is the procedure Build\_C\_and\_est() which constructs cluster C(ni) using the clusters of parent tasks of ni, which we call parent clusters. If all the parent clusters of the tasks are available it is called a ready task. The procedure Build\_C\_and\_est() is applied to ready tasks only. ReadtTaskSet is set of Ready Tasks. When a task ni is merged into cluster C it forms a new cluster C’’ by C U{ni}. After the sequence of merge operations Build\_C\_and\_est() determines which parents clusters are merged with na. Build-C-and-est() generates C(na,) with the smallest est(na,) among all possible combinations of merging parent clusters. In function Merge() if two tasks withsame est values existed in the same cluster then the function preserves orders in their previous clusters.

Algorithm [ref 4]

Algorithm schedule(V, E, 7, c, r)

Begin

//input: DAG (V, E,T,c)

// output: the set of clusters r

C(na,) <--{na};

est(na,) <--0; // entry task n,

ReadyTaskSet 🡨 succ(na);

While (ReadyTaskSet > 0)do

Na 🡨 select a task in ReadyTaskSet;

Build\_C\_and\_est(na,C(na), est(na));

ReadyTaskSet 🡨 {nT l For V ni belongs to pred(nr),3 C(ni)};

end while

T 🡨 {C(ni) l ni belongs to V};

End

Procedure Build\_C\_and\_est(na, C(na), est(na))

Begin

// input: na a ready task

// output: C(na) the cluster of na

// output: est(na)

variable: Ml, clusters that merge parent clusters of na,

//side effect: stc(na) of tasks in C(na) are determined

//rdy(C(n1),na) >= rdy(C(n2),na) >= ….. >= rdy(C(nk),na)

// rdy(C(nk+1),na) = 0 and C(nk+1) = 0

M1 🡨 C(n1);

I 🡨 1;

do

I 🡨 l + l ;

Ml,🡨 Ml-1, U C(nl);

while (l<= k AND stMl-1 (na) >= stMl (na));

// 1 = k + I or STML-1, (na) < stMl,(na)

C(na) 🡨 Ml-1 U {na};

EST(na) 🡨 stMl-1(na);

End

Function merge(M, C)

Begin

//Input: M = {nm1 ,nm2, . . .}, where est(nmi) <= est(nmi+1)

//Input: C = {nc1, nc2 ,.. . }. where est (nci) <= est (nci+1 )

//Output: M = {nm1, nm2, . . .}

i 🡨 1,j 🡨 l , k 🡨 1;

while (there is any task in M or C) do

if (nmi = ncj ) then

nmk 🡨 nmi;

increase i and j ;

else if(EST(nmi) <=EST(ncj) then ;

nmk 🡨 ncj

increase i;

else

nmk 🡨 ncj ;

increase j ;

end if

if (nmk does not exist among nm1 , . . . , nmk-1 ) then

stM (nmk ) 🡨 max[ctM (nmk-1), EST(nmk )];

ctM( nmk ) 🡨 stm (nmk) + T mk ;

increase k;

end if

end while

return( M’);

End

[figure ref 1]

1

1

12

6

12

1

1

1

1

4

2

5

4

3

3

4

4

4

10

10

10

1

10

|  |  |  |
| --- | --- | --- |
| Task | Cluster | EST |
| 1 | {1} | 0 |
| 2 | {1,2} | 2 |
| 3 | {1,3} | 2 |
| 4 | {1,4} | 2 |
| 5 | {1,5} | 2 |
| 6 | {1,2,6} | 5 |
| 7 | {1,2,3,7} | 8 |
| 8 | {1,5,2,8} | 10 |
| 9 | {1,2,3,5,7,8,9} | 21 |

Set EST(0) =0. We now calculate EST of 2,3,4,5,6. These values are easy to calculate because all of these tasks have only one parent. The EST of task 2,3,4,5 is 2. The EST of task 6 is 5. Task 7 has 2 parents’ task 2 and task 3.We get the EST of task 2 and EST of task 3. If task 7 merges with task 2 then EST of task 7 becomes 17 as C(rdy(3),7) =17. But if we merge it with tasks 2 U task 3 then it is less as there is no task to be extracted from task 2 U task 3. So if we merge task 7 with task 2 and task 3 then we get an optimal EST for task 7 that is 8 and cluster of task 7 = {1,2,3,7}. For task 8 we get its all parents task that is task 2,3,4,5. We calculate the Ready set of all the parents. C(rdy(2),8)= 11 C (rdy(3),8)= 6 C (rdy(4),8)=6 C(rdy(5),8) =17 . If we merge task 2 and task 5 we get an optimal EST for task 8 that is 10 and cluster of task 8 ={1,5,2,8}. For task 9 the clusters are ordered as C(7) , C(8) , C(6) with ready times. When task 9 merges in C(7) EST(9) becomes 24 because rdy(C(8),9) =24. Task 9 when merges with C(7) U C(8), EST(9) becomes 21 because of ctc(9)(8) =21. The merge with C(7) U C(8) U C(6) is not tried because rdy(C(6),9)= 19 < 21 = ctc(9)(8). Thus the optimal EST for Task 9 is 21 and cluster of task 9 = {1,2,3,5,7,8,9}.

* Experimental Environment:

Experiments were carried out in below environment.

Platform: Microsoft Windows 10, 64 bits

IDE: Microsoft Visual Studio 2015

RAM: 3 GB

Processor: Core i3

* Simulation Results:

Simulation results for random task graphs are plotted below.

Task graphs with different sizes varying from 4 to 20 nodes and CCR varying from <1 (less than 1) to >2 (greater than two) have been considered. Results for CPFD and RD have been calculated and plotted. RD produces same make span as that of CPFD and takes equal number of processors. Performance improvement was seen in case of number of duplicates and processor consumption.

* Future Scope:

Task duplication technique have been adopted in heterogeneous systems too but there is scope of improvements. In grids task duplication is best suited due to large communication latency and delay.

Another area could be dynamic scheduling of graph tasks with duplication considering heterogeneous systems and contentions in networks.

New search algorithms like genetic algorithms can be adopted in task duplication strategy.

* Proposed Work

We implemented CPFD and RD task duplication techniques and experimentally evaluated and compared results with respect to different input and output parameters. We considered DAG with different sizes (different number of node) and CCR (Communication to Computation Ratio) and calculated values of output parameters mentioned above. Results of CPFD and RD have been calculated, compared and graphs have been plotted.

* Conclusion:

Static scheduling of graph tasks on multiprocessor systems is a NP complete problem and lot of heuristics techniques have been proposed to solve this problem. Task duplication is relatively new approach which aims to reduce communication costs among tasks by duplication of parent tasks. It is best suited for fine grained applications.

We have studied different task duplication algorithms and compared their performance with existing well known algorithms CPFD. Aim of those algorithms basically to make improvements in the result of CPFD in terms of make span, number of duplicates and number of processor required to execute given task. We have compared results of CPFD and RD algorithms with respect to different parameters. We can conclude that RD produces same make span as that of CPFD and takes equal number of processors. Performance improvement was obtained with respect to number of duplicated tasks and processor consumption values since RD exhibits an extra step to find out duplicates and tries to remove them without affecting make span.

* References:

1) I. Ahmad and Y.-K. Kwok. “On exploiting task duplication in parallel program scheduling”. *IEEE Transactions on Parallel and Distributed Systems,* 9(9):872–892, September 1998.

2) I. Ahmad and Y.-K. Kwok. “On parallelizing the multiprocessor scheduling problem”. *IEEE Transactions on Parallel and Distributed Systems*, 10(4):414–432, April 1999.

3) Doruk Bozdag, Eylem Ekici, Fu¨sun O¨ zgu¨ner, Umit Catalyurek, “A Task Duplication Based Scheduling Algorithm Using Partial Schedules”. *International Conference on Parallel Processing, June* 2005

4) Chan-Ik Park ,Tae-Young Choe, “An Optimal Scheduling Algorithm based on Task Duplication” June 2001.

5) Jin Ha Jung ,Kwang Sik Shin, Myong Jin Cha, Mun Suck Jang, , Wan Oh Yoon, Sang Bang Choi, “Task scheduling algorithm using minimized duplications in homogeneous systems” *Parallel and Distributed Computing,* 68 (2008) :1146–1156, April2008.

6) Doruk Bozdag, Fu¨sun O¨zgu¨ner, “Compaction of Schedules and a Two-Stage Approach for Duplication-Based DAG Scheduling”. *Transactions on Parallel and Distributed systems , 20(6)* June 2009.

7) Amit Agarwal and Padam Kumar, “Economical Duplication Based Task Scheduling for Heterogeneous and Homogeneous Computing Systems”. *International. Advance Computing Conference ,* March 2009.

8) J. Janecek and T. Hagras , “A High Performance, Complexity Algorithm for Compile-Time Job Scheduling in Homogeneous Computing Heterogeneous Distributed Computing Systems,Environments”, *Proc. Int'l Conf, on Parallel Processing Workshops*, pp 149-155, October 2003.

9) Kuldip Singh , Padam Kumar, Savina Bansal “An Improved Duplication Strategy for Scheduling Precedence Constrained Graphs in Multiprocessor Systems”,*IEEE Trans. on Parallel and DistributedSystems*, 14 (6): pp. 533-544, 2003.

10) H.J. Park, B.K. Kim, “Optimal task scheduling algorithm for cyclic synchronous tasks in general multiprocessor networks “, *Journal of Parallel and Distributed Computing* 65 (3) (2005) 261–274.