A Scheduling Algorithm for Fork-Join DAG in Bus-based and Heterogeneous Environment

Lisheng Wang, Liguo Chen,
Department of Computer Science
College of Electronic and Information Engineering
Tongji University
Shanghai, China
lishwang@tongji.edu.cn

Kete Wang
Department of Computer Science
College of Electronic and Information Engineering
Tongji University
Shanghai, China
chenliguo1989@163.com

***Abstract*—The Fork-Join task graph is one of the basic modeling structures for parallel processing. However, many previous scheduling algorithms ignore to economize processors and minimize the total completion time. What's more, many algorithms don't consider the competition caused by bus-based clusters and the heterogeneous of processors in real applications. This paper presents a new algorithm for Fork-Join task graph, considering economy of processors and minimization of the total completion time, the non-parallel communication, and heterogeneous environment as well. We propose a task scheduling algorithm based on task duplication which randomly generated a number of Fork-Join task graphs by producing the task execution time and communication time. Simulation results show that the proposed algorithm has less total completion time and less number of processors than other compared algorithms for more practical applications.**

Keywords-Task scheduling, Task duplication, Heterogeneous, Bus-based, Fork-join task graph

I. INTRODUCTION

The goal of task scheduling algorithm is to allocate the tasks of a parallel program to processors in order to minimize the completion time of the program and economize processors. Since it has been shown the multiprocessor scheduling problem is NP-complete, many researchers have proposed scheduling algorithms based on heuristics [1-14]. It is well known that the complexity and quality of scheduling algorithm largely depend on the task graph structure [3, 4]. The fork-join structure is one of the basic modeling structures for parallel processing.

Since the communication time between tasks assigned to the same processor is considered to be negligible, task duplication is one way of reducing overhead of the interprocessor communication. By using this approach, some of more critical tasks of a parallel program are duplicated on more than one processor. This can potentially reduce the start time of the waiting tasks and eventually improve the overall completion time of the entire program. It has been shown that the scheduling algorithms based on task duplication always perform better than any other scheduling algorithms without task duplication.

There are several task duplication based scheduling schemes. When certain conditions are met for the given task set with parallel homogeneous processors, TDS[1] algorithm proposed is able to find an optimal schedule. However, it ignores to economize the processors so that the speedup of parallel programs is decreased. TDS algorithm assumes that algorithms work in interconnected network environment and the communication between the processors can be executed in parallel. However, each processor in the sending and receiving multiple messages are serial in a not interconnected network, such as the bus-based clusters. We should not only consider the start time and end time of the task, but also consider the start and the end time of communication between processors due to the serial communication. Task Scheduling Algorithm (TSA_FJ) [7] only tries to utilize the idle time slot of the homogeneous processors, but the workload of other processors are not considered. This algorithm assumes that the communication between processors must be the serial. What's more, many algorithms assume that each processor has the same execution speed and the same communication time to fixed length messages. Experimental results show that this assumption would not result in good results in the practical application. Task scheduling algorithm should be based on a heterogeneous environment and take into account the serial communication between processors for better adapting to the actual systems.

In this paper, we propose a new algorithm based on fork-join task graphs that can be used in heterogeneous systems according to the algorithms used in homogeneous environment.

The algorithm based on bus-based clusters uses task duplication method to improve the scheduling performance. Section 2 describes the systems model and problem definition. Section 3 presents the analysis of our approach, followed by a description of the proposed algorithm and an example to demonstrate the operation of the algorithm. Section 4 includes the performance results and comparisons with other algorithms. We provide concluding remarks in the last section.

II. SYSTEMS MODEL AND PROBLEM STATEMENT

We assume that a parallel program can be expressed as a directed acyclic graph (DAG), which is also called a task graph. A DAG consists of a tuple (V, E, w, c), where V, E, w, c are the set of task nodes, the set of communication edges, the set of computation costs associated with the task nodes,

Published by Atlantis Press, Paris, France.

and the communication costs associated with the edges respectively. w(ni) is the computation cost for task ni and c(ni,nj) is the communication cost for edge e(ni,nj) that connect task ni and task nj. If task ni and task nj are assigned to the same processor, then c(ni,nj) is zero. The edge e(ni,nj) represents the precedence constraint between task ni and task nj. In other words, task nj can start only after the completion of task ni.

The fork-join structure is one of the basic modeling structures for parallel processing. A fork-join task graph is a hybrid of an in-tree task graph and an out-tree task graph, which is shown in Figure1. It has a fork node that spawns a number of children nodes. The output edges of the children nodes are connected to a join node. We assume the set of task nodes V={nx} ∪ {nz} ∪ {n1,n2,…nm}, the fork node nx , the join node nz, the set of intermediate nodes S={n1,n2,…nm} and assume bus-based and heterogeneous clusters P={ p0,p1, p2,…, pm}. We think execution speed of the processors p0 and the communication speed between the main processor and other processor to fixed-length messages have the same order, may wish to the processor set p0, p1,…, pm have been sorted in execution speed of descending order , their speed of execution order is denoted by v(0) ≥ v(1) ≥ v(2) ≥ …v(m), communication speed of a unit traffic between the main processor and other processor is denoted by u(0) ≥u(1) ≥u(2) ≥…u(m). Different processors of the systems perform the same task using the initial data of different copy and different copies of the same task generate consistent data.

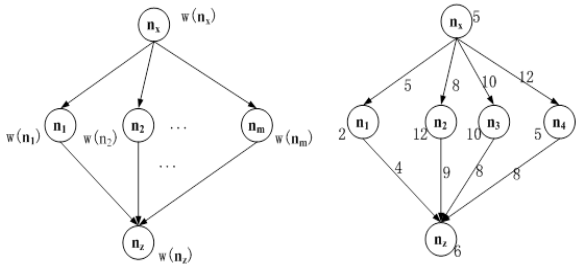

Figure 1. Fork-join Task Graph and An Example

A few terms and mathematical expressions are defined in the following for a more clear presentation.

st_ori(ni):the start time of task ni ;

ct_ori(ni):the completion time of task ni , ct_ori(ni)= st_ori(ni)+w(ni)/v;

cst_ori(ni):the start time of communication c(ni,nz), cst_ori(ni) ≥ ct_ori(ni);

cct_ori(ni):the completion time of communication c(ni,nz), cct_ori(ni)= cst_ori(ni)+ c(ni,nz)/u;

Pnum: the number of processors used; SL: the total completion time.

The task scheduling algorithm based on task duplication for fork-join task graphs replicate fork node nx in each processor node and executive task node nx in the only main processor(such as p0). Therefore, the goal of the proposed algorithm is to minimize the the total completion time SL and processors number Pnum in bus-based and heterogeneous clusters systems. How to compute SL is shown in below:

SL=max$ni \in S${cct_ori(ni)}+w(nz)/p0

If task node ni is assigned on processor p0, then cct_ori(ni)= cst_ori(ni)= ct_ori(ni), that is c(ni,nz)=0.

III. THE PROPOSED ALGORITHM FOR FORK-JOIN TASK GRAPHS

A. *analysis of the algorithm*

Since each task is independent in set S, tasks in set S can be divided into two disjoint parts, namely S= S1 ∪ S2, S1 ∩ S2=Φ, the tasks in set S1 are assigned to p0. At first, S1、nx、nz are assigned to processor p0, so the completion time of set S1 is $(\sum_{n_i \in S_1} w(n_i) + w(n_x)) / p_0$ which can be recorded Len(S1). As S2、nx are assigned to processors p1, p2,…, the communication time between tasks of set S2 and task node nz affects the final the total completion time. What's more, different scheduling methods to the tasks of set S2 lead to different completion time for the tasks of set S2. Record the set S2 earliest completion time for the Len(S2)= $\min\{\max_{n_i \in S_2}\{cct\_ori(n_i)\}\}$. The total completion time after determining set S1 and set S2 is SL=max{ Len(S1), Len(S2)}+w(nz)/p0. It's easy to calculate Len(S1), so we first consider how to calculate Len(S2).

The tasks of set S2 are assigned to processor p1, p2,…. The most direct way to copy task node nx and assign each task in set S2to different processors at the same time in order to minimize the maximum communication completion time.

At first, quick sort tasks from set S2 in non-ascending order by the sum of w(nik)+ c(nik,nz). Now we assume the results satisfy w(ni1)+ c(ni1,nz) ≥ w(ni2)+ c(ni2,nz)≥ …w(nij)+ c(nij,nz). Code of Len(S2) is shown in Figure2:

```
int Len(S2)
{
   cct_ori(ni1)=w(nx)/v(1)+ w(n1)/v(1)+ c(ni1,nz)/u(1);
     for k=2 to j
         cct_ori(nik)=max(cct_ori(ni(k-1),w(nx)/v(k)+
         w(nk)/v(k))+ c(nik,nz)/u(k);
     return cct_ori(nij);
}
```

Figure 2. Computation of Len(S2)

Len(S1)and Len(S2) in the calculation are only relevent to how to divide set S1 and S2 . Set S1 and S2 by the different the total completion time correspond to different SL. Therefore, the goal of scheduling algorithm is how to determine set S1 and set S2 for corresponding to a minimum of the SL .

B. *The Proposed Algorithm*

The new algorithm includes two steps:

Step 1: Determine set S1 and set S2 as it is shown is Figure2.

Step 2: Schedule reasonable tasks in set S2in order to save processors.

In step 1, the initial state of the sets is S1=Φ and S2=S. If S2is not empty, recursively select task node from set S2 which makes Q=max{Len(S∪ni), Len(S2-ni)} minimial. If Q ≤ max{Len(S1),Len(S2)}, that is, task node nz can be implemented in advance, then let S2= S2-ni,S1=S1∪ni;if Q >max{Len(S1),Len(S2)}, then let set S1 and set S2 remain.

```
void GetLen(G)
{
 quick sort tasks from set S in
 non-ascending order by the sum
 of w(nik)+ c(nik,nz) and put them
 in set S2;
 assign nx to p0;
  SLen1= w(nx)/v(0);
  SL=SLen2= Len(S2);
  for j=1 to m
  {
    k=-1;
    for each ni∈S2
    {
      SLen2= Len(S2-ni);
      Q=max{ SLen2, SLen1+ w
 (ni)/v(0);
    }
 if(Q≤SL)
 {
    k=i;
    SL= SLen2
 }
}
if(k!=-1)
 {
  assign ni to p0;
  S2= S2-ni;
  SLen1=SLen1+ w(nk)/v(0);
 }
else
  break;
}
SL=SL+ w(nz)/v(0);
}//GetLen(G)
```

Figure 3. Description of the Algorithm in Step 1

After determining the set S1, S2 and the total completion time SL under without increasing SL or the start time of the task nz condition, the tasks are assigned to processors used as much as possible in step 2.

After considering step 1 and step 2, the algorithm in bus-based and heterogeneous clusters environment is shown in Figure4.

```
void Schedule(V,E,w,c)
{
  GetLen(G);
  for(i=1;i≤| S1|;i++)
  assign tasks in set S1 to p0;
  quick sort tasks from set S2 in
  non-ascending order by the sum
  of computation and
  communication cost and record
  them n1,n2…;
  k=0;
  for(i=1;i≤| S2|;i++)
  {
    Insertion=false;
    if(task can be inserted into
     the processor used pj (j≤k))
      Insertion=true;
    if(Insertion)
     assign ni to pj (j≤k);
    else
    {
      k=k+1;
      assign nx and nz to pk;
    }
  }
  Assign nz to p0;
}//Schedule(V,E,w,c)
```

Figure 4. Tasks Assignment of the Set S2

C. *An example*

In this section, we use an example of fork-join task graph in Figure1 to illustrate the procedure of the proposed algorithm. In this exaple, we assume that the set of processors P={p0,p1, p2,…, p4}, their speed of execution order is denoted by 1 ≥ 1 ≥ 1/2 ≥ 1/3 ≥ 1/4, communication speed of a unit traffic between the main processor p0 and other processor is denoted by1 ≥ 1/2 ≥ 1/3 ≥ 1/4.

Step 1: Identify the set S1, S2 and the total completion time SL, the results are in Table I.:

Step 2: Schedule the tasks of the set S2. Considering orderly the following each task of set S2, the algorithm searches the scheduled processors to select which processor suits for inserting the task.

If the inserting condition is satisfied, the algorithm would modify the tasks in set S1, S2 and assign the tasks to the suited processor, such as task n1. Otherwise, it assign both entry task nx and ni to a new processor, such as task n4.

TABLE I. PROCESSOR OF DETERMINING TO THE SET S1, S2 AND SL

S_1	S_2	Len(S_1)	Len(S_2)	cst_ori(n_z)
Φ	{n_2, n_3, n_4, n_1}	5	86	86
{n_2}	{n_3, n_4, n_1}	17	51	51
{n_2, n_3}	{n_4, n_1}	27	26	27
SL= cst_ori(n_z)+ n_z=27+6=33				

Step 3: When all tasks in set S2 are assigned to processors, the algorithm schedules the task nz to the main processor p0. In the end, the processor allocation and the schedule time obtained by the algorithm are shown in Figure6. (shadow denotes the overlapp time between the main processor itself computing and communicating with other processors)

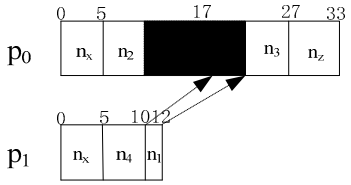

Figure 5. Processor Allocation and the Schedule Time Obtained by the Algorithm

IV. PERFORMANCE AND COMPARISON

The propoesed algorithm can be applied to any fork-join task graph and gerenate a better sheduling. To systemsatically evaluate and compare the algorithm with other algorithms, we consider the randomly generated application graphs as the workload for testing the algorithms. As shown in Figure 5, for the randomly generated fork-join task graphs, the schedule results of the algorithm are compared with those of other algorithm for the generated task graphs. TDS algorith assigns parents tasks of a join task to different porcessors. TSA_FJ algorithm only considers the workload of the first processor.

TABLE II. COMPARISON OF SCHEDULE RESULTS WITH OTHER ALGORITHMS

m	parameter	The Algorithm	TSA_FJ	TDS
3	SL	23	25	25
	Pnum	2	2	3
4	SL	33	34	36
	Pnum	2	2	4
5	SL	36	36	38
	Pnum	2	3	5
6	SL	41	42	43
	Pnum	2	3	6
7	SL	51	51	7
	Pnum	2	3	7

From the comparison in Table II, we can see that the algorithm based on fork-join task graphs that can be used in bus-based and heterogeneous systems has less the total completion time and much less processors. The algorithm outperforms the other algorithms in terms of the the total completion time and the number of used processors. Thus,

the speedup and efficiency of the algorithm are better than other algorithms.

V. CONCLUSION

This paper presents a better algorithm to shedule tasks of fork-join graph onto processors, which is based on fork-join task graphs that can be used in bus-based and heterogeneous environment. The performance of the algorithm has been compared with other algorithms in terms of the total completion time and the number of used processors. Simulation results showed that the algorithm has better speedup and efficiency than other compared algorithms. Therefore, the algorithm achieves considerable performance for practical applications where can be used in bus-based and heterogeneous systems.

REFERENCES

[1] S. Darbha and D.P. Agrawal, optimal Scheduling Algorithm for Distributed-Memory Machines, IEEE Trans. Parallel and Distributed Systems [J], Vo1.9,No.l, pp.87-94, January,l998

[2] Yang T, Gerasoulis A, DSC: Scheduling parallel tasks on an unbounded number of processors, IEEE Trans. On Parallel and Distributed Systems [J],1994,5(9):951-967.

[3] Kwok YK, Ahmad I, Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors, IEEE Trans. on Parallel and Distributed Systems [J], 1996,7(5):506−521.

[4] Palis MA, Jing-Chiou Liou, Wei DSL, Task clustering and scheduling for distributed memory parallel architecture, IEEE Trans. on Parallel and Distributed Systems [J], 1996,7(1):46-55.

[5] Ahmad I, Kwok YK, On exploiting task duplication in parallel program scheduling, IEEE Trans. on Parallel and Distributed Systems [J], 1998,9(9):872-891.

[6] Chan-Ik Park, Tae-Young Choe, An optimal scheduling algorithm based on task duplication, IEEE Trans. on Computers [J], 2002, 51(4):444-448.

[7] Liu ZY, Fang BX, Jiang Y, Zhang Y, Zhao H, A new algorithm for scheduling fork-join task graph, Chinese Journal of Software [J], 2002,13(4):693−696 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/13/693.pdf

[8] Liu ZY, Fang BX, Zhang Y, TSA-OT: An algorithm scheduling an out-tree DAG, Chinese Journal of Computers [J], 2001,24(4): 390-394 (in Chinese with English abstract).

[9] Andrei Radulescu, Arjan JC, van Gemund,Low-Cost task scheduling for distributed-memory machines, IEEE Trans. on Parallel and Distributed Systems [J], 2002,13(6):648-658.

[10] Li Qinghua, Ruan youlin, Liu Gan et al. An Optimal Scheduling Algorithm for Fork-Join Task Graphs, Chinese Journal of Software [J],May.2005,Vol.16,No.5

[11] Wang Lisheng , Wang Kete, Li Xixi, A false-sharing-eliminable parallel tasks scheduling algorithm based on DAG, IEEE Computer Society [J], 445 Hoes Lane - P.O.Box 1331, Piscataway, NJ 08855-1331, United States, Oct. 2010,Vol.9.

[12] Sang Cheol Kim, Sunggu Lee and Jaegyoon Hahm, Push-Pull Deterministic Search-Based DAG Scheduling for Heterogeneous Cluster, IEEE Trans. on Parallel and Distributed Systems [J], 2007 18(11):1489-1502.

[13] T.NTakPeandF.Suter, Critical Path and area based scheduling of Parallel task graphs on heterogeneous platforms, Proceedings of the 12th International on Parallel and Distributed Systems [J], (ICPADSZO06), 2006:3-10.

[14] A.Gerasoulis and T.Yang, A Comparison of Clustering Heuristics for Scheduling DAGs on Multiprocessor, Journal of Parallel and Distributed Computing [J], 1992, 16(4):276-91.

Published by Atlantis Press, Paris, France.