5th SASTech
Iran |Mashhad
May 12 - 17 | 2011
5th Symposium on Advances in Science & Technology

A new task Pre-scheduling algorithm by Reduction of Critical Path Length in Grid Computing

Elnaz Rashid Hossein Zadeh

Department of Computer Engineering, Islamic Azad University - Tabriz Branch
Tabriz, Iran
El.rashidi@yahoo.com
Paper Reference Number: **0802-1048**
Name of the Presenter: Elnaz rashid hossein zadeh

Abstract

In this paper, we propose a new algorithm for restructuring task graphs for suitable scheduling in grid computing. This algorithm obtains the critical path length in task graph and then start to reduce the length of this path. For optimize the length of critical path To do this**.** this algorithm reduces communication costs by merging tasks from task graph who are pertained to this path and their communication costs exceed their execution time. Task duplication techniques are applied when the task merging operation on critical path change at least the length of one of other paths and its length is greater than the updated critical path length .Afterward, these operations apply to the new critical path, if it exists .this algorithm changing critical path to optimized path example is shown to improve performance.
Key words: Task graphs, Critical path, Merging, Duplication, Scheduling

1. Introduction

Grid is a large geographical distribution resources system that is collection of heterogeneous resources. Grid scheduling is explained as the process of making scheduling decisions involving resources over multiple managerial domains. The purpose of this paper is task graphs restructuring such that when applying any task scheduling algorithm to the restructured task graph, the minimum possible completion is achieved. Task merging and clustering are techniques to restructure task graphs for benefit scheduling [1-5].

When merging a task node in its parent nodes, other children of parent in the task graph, could be delayed. To find a solution the difficulty, it is proposed to duplicate the parent

nodes before the merge, under the condition that their execution times is less than the maximum time needed to communicate with their successors [2]. On the other hand, above condition does not always necessitate the merge. In the approach presented in [5], a node is merged in a subset of its parents on the condition that the merge operation reduces its earliest time to start. If the merge retards the execution of the other children of the parent node, the parent node is duplicated.

In this paper we propose the approach that applies the merge and duplication techniques by considering length of critical path and other paths from start task to end task. In this approach, task duplication techniques are applied when sum of delays caused by task merging cause the length of other path be longer than critical path length. If after the duplication, the number of independent tasks gets above the number of available processors, the duplication is considered as a contrary factor.

The rest of this paper organized as follows: In Section 2, our proposed algorithm is presented. In Section 2.1, a relation for computing the earliest start time of tasks within a task graph is presented. In Section 2.2 a relation to compute the benefits of merging a task node with any subset of its parents is offered. Section 3 explores the evaluation results. Final section contains the conclusion and future works.

2. Our Algorithm

In this section our proposed algorithm: a new task scheduling algorithm by reduction of critical path length in grid computing (RCPL) is presented. When combining a node *v* which is on the critical path with one of its parents, *p*, the start time of the siblings of *v* that is in other path may be increased. Sometimes it is may be that sum of these delays cause increasing the length of it path so that be greater than updated critical path length. To resolve the difficulty, the parent node, *p*, could be duplicated before the merge. However, if there are not enough processors to execute the duplicated tasks in parallel, the merge may not be beneficial. Thus, in the task merging quality function, *Q*, is shown in section 2.2, the number of available processors, the total execution time of the tasks to be duplicated and the amount of reduction in earliest start time are considered as main factors in deciding whether to merge a task *v* with a subset, P_k, of its parents. Our new task merging algorithm is presented in Fig. 1.

2. 1. The RCPL Algorithm

Our purposed algorithm, RCPL, is shown in Fig. 1. The algorithm tries to increase parallelism in the execution of a definite task graph by reducing the earliest start time of tasks in the task graph. As shown in relation (1) [5], to compute the earliest start time, EST_v, of a task, *v*, the earliest start time and the execution time of its parent nodes and the time that takes to receive the results from its parent, $\tau(pi)$, the size of the data to be received by the task nodes, *c(pi ,v)*, the latency, L, and the bandwidth, B, of the communication lines are needed.

$$EST_v = \begin{cases} 0 & Parent(v) = \phi \\ MAX_{p_i \in Parents(v)} (EST(p_i) + \tau(p_i) + L + \frac{c(p_i, v)}{B}) & Parent(v) \neq \phi \end{cases} \quad (1)$$

A node *v* is merged in the subset of its parents, which reduce its earliest start time generally , EST_v ,. To obtain this, the time, $R_{pi,v}$, at which the outputs of each parent node, p_i, can be aggregated by the child, *v*, is computed .As shown in relation (2) [5]:

$$R_{p_i,v} = EST(p_i) + \tau(p_i) + L + \frac{c(p_i, v)}{B}. \qquad (2)$$

After $R_{pi,v}$ are computed, the parents are sorted in decreasing order of $R_{pi,v}$. Starting with the node with the highest value of $R_{pi,v}$, the benefit of merging *v* with each of the parents, p_i, on the earliest start time of *v* is computed. The node *v* is then merged with the subset of its parents, which reduce its earliest start time, and their number does not exceed the number of available processors.

```
Step1
 1.  Algorithm RCPL
 2.  Input: a task graph G (V, E, τ, c) where:
 3.        V: Set of tasks, E: set of task inter-connection lines
 4.        τ: A function to compute the execution cost of each task
 5.        c: A function to compute the communication costs
 6.  Output: G' (V', E', τ', c') = the modified task graph
 7.  Method:
 8.    For each v in V do apply relation (1) to compute EST (v); End For;
Step2
 9.    For each pi in parents (ET) do // ET: End Task.
10.      Apply relation (2), below, to compute the earliest time, Rpi,ET,
11.         to collect data from parent, pi, on ET;
12.      Sort the PLs on the value of Rpi,ET, descendingly,
13.       Giving the sequence PL = (PL1, PL2, …, PLk); //PL: Path Length from Start Task to End Task.
14.    End For;
15.    i : = 1;
Step3
16. L1: For each v (start from ST to ET) in V which is pertain to PLi do // ST: Start Task.
17.      For each pi in parents (v) do
18.        Apply relation (2), below, to compute the earliest time, Rpi,v, to collect data
19.            From parent, pi, on v;
20.        Sort the parents, pi, on the value of Rpi,v, descendingly,
21.            Giving the sequence P = (P1, P2, …, Pk);
22.      End For;
23.      Apply relation (3) to compute the benefit Qv,k, of merging v with its parents
24.                  p1 to pk in P for 1 ≤ k ≤ n;
25.      If there are no Qv,k > 0 then it is not possible to merge v with any subset of its parents;
26.      go to L1; End if;
27.      Find the subsequence Pk = {(p1, p2, …, pk) in P, 1 ≤ k ≤ n} such that Qv,k is maximum;
28.    End For;
Step4
29.    Let Ψv,Pk be the set of siblings of v whose earliest start time increases
30.    Let Dv,Pk be the set of parent nodes pi in Pk with at least one child in Ψv,Pk
31.    For j from 1 to n do
32.      If PLj is increased to its previous value and is greater than PLi then
33.        For each node p in Dv,pk which pertain to PLj do
34.          Let p′ be a copy of p
35.          Remove any edges from p to nodes in Ψv,Pk ;
36.          Remove any edges from p′ to nodes that not in Ψv,Pk ;
37.        End For;
38.      End If;
39.    End For;
Step5
40.        Sort the PLs on the value of Rpi,ET, descendingly,
41.          Giving the sequence PL = (PLj, PLj+1, …, PLj+n);
42.    If j is not equal i then i is equal to the j; go to L1; End If;
43.  End;
```

Fig1: The RCPL Algorithm

Step1 presents the input and output parameters. First, this algorithm computes the earliest start time of each node *v* in *V*, EST_v, .The result of step2 is sorting the paths

discerningly on the value of their lengths and discovering the critical path. The critical path is a path that value of its length from start task to end task is more than length of other paths which exist between start task and end task. In step3, proposed algorithm starts the merging of each node *v* in *V* which is pertain to critical path with its parents in a top to down manner, from start task to end task. Step4 duplicates the nodes of critical path that merging of them with their child caused that length of other path is increased to its previous value and is greater than critical path. Step5 discovers the new critical path if exists and repeat these operations for it.

2.2.Task Merging Benefits

We used checking utility of merging node from[5] that used follow relating:

$$Q = \alpha \times \Delta T - (1 - \alpha) \times \Delta E \quad (3)$$

3. Evaluation of Proposed Algorithm

In this section, we evaluate the RCPL algorithm in comparison to RTM algorithm proposed in [5]. We have applied two known algorithm, to generated five task graphs, FFT1, FFT2, FF3, FFT4 and FFT5. The number of nodes in task graphs specified in table1. Applying RCPL and a known task merging algorithm RTM[5], to restructure these benchmark task graphs,the restructured graphs where scheduled by a simple genetic algorithm. The parallel execution times of the resultant scheduling were compared with the execution times resulted by applying 2 known scheduling algorithms: RTM[5], Genetic. Below, in table2,table3 and table 4, the comparison results are presented.

Task graphs	FFT1	FFT2	FFT3	FFT4	FFT5
nods	10	15	20	25	30

Table1. Number of nodes in the task graphs

	FFT1	**FFT2**	**FFT3**	**FFT4**	**FFT5**
RCPL+Genetic	402	753	1443	2218	3428
RTM +Genetic	436	884	1717	2387	3455
Genetic	733	1372	2355	3058	4414

Table2. Assuming that the execution environment has just two processors.

	FFT1	**FFT2**	**FFT3**	**FFT4**	**FFT5**
RCPL+Genetic	247	553	1419	1888	3251

RTM +Genetic	324	843	1571	2115	3411
Genetic	713	1273	2234	2872	4321

Table3. Assuming that the execution environment has just five processors.

	FFT1	**FFT2**	**FFT3**	**FFT4**	**FFT5**
RCPL+Genetic	222	536	1019	1785	2862
RTM +Genetic	317	827	1506	1920	3206
Genetic	657	1236	1893	2756	4113

Table4. Assuming that the execution environment has just eight processors.

Below, Performance evaluation RCPL and RTM algorithms on task graphs in Fig2, Fig3, Fig4, Fig5 and Fig6 are presented.

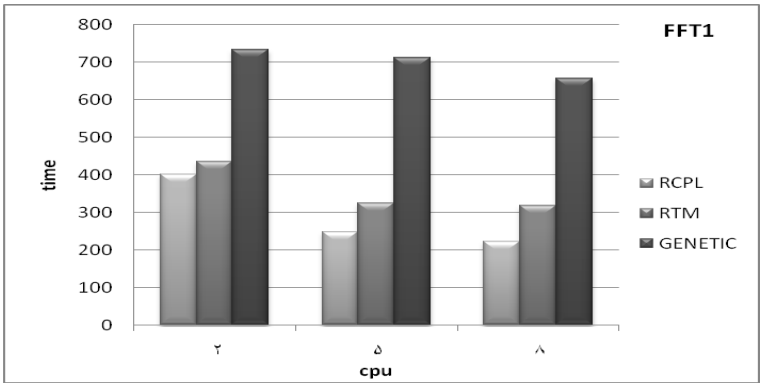

Fig2: Performance evaluation RCPL and RTM algorithms on FFT1 task graph

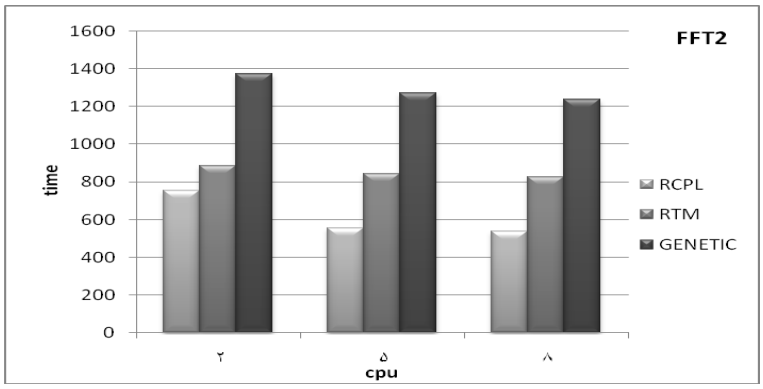

Fig3: Performance evaluation RCPL and RTM algorithms on FFT2 task graph

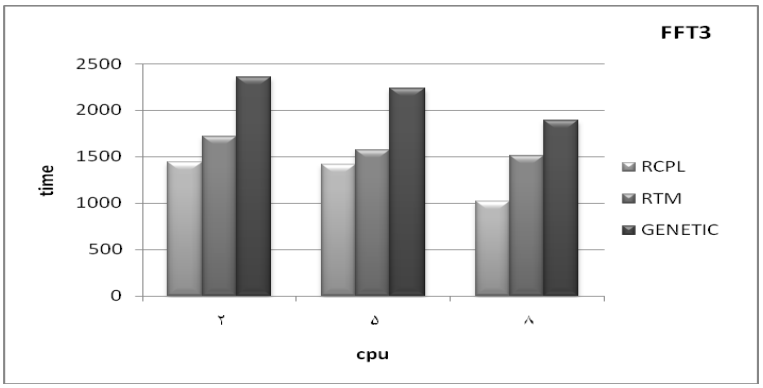

Fig4: Performance evaluation RCPL and RTM algorithms on FFT3 task graph

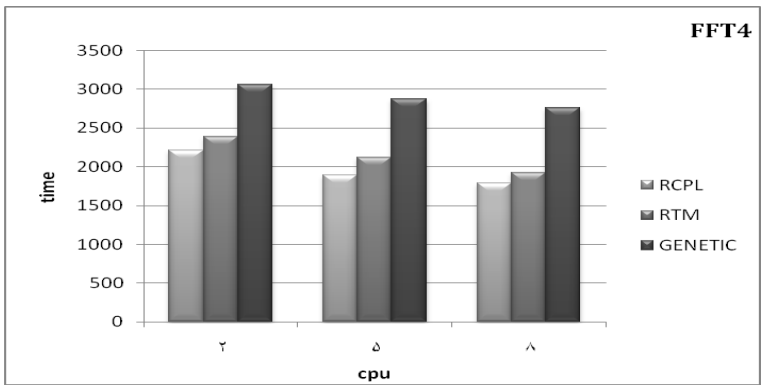

Fig5: Performance evaluation RCPL and RTM algorithms on FFT4 task graph

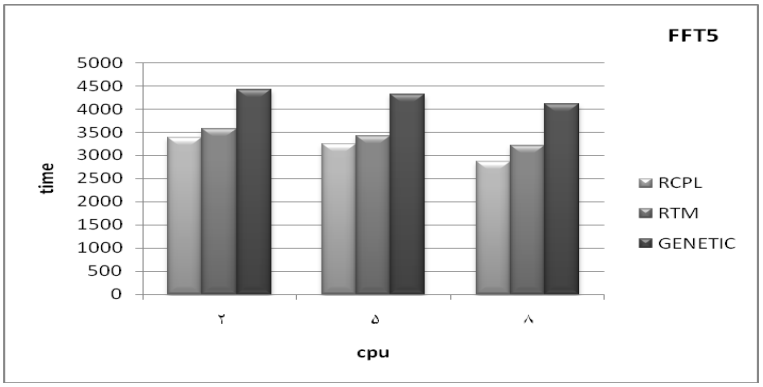

Fig6: Performance evaluation RCPL and RTM algorithms on FFT5 task graph

4 Conclusion and Future Work

This work proposes a pre-scheduling algorithm that can restructure the task graph before applying any scheduling algorithm to it. The tasks could be merged with their parents by applying the RCPL algorithm in order to take advantage of parallelism inherent in the execution of tasks in a task graph. The algorithm attempts to exploit the inherent parallelism by minimizing the earliest start time of each task which is pertained to critical path within the task graph. Task duplication techniques are applied when the task merging operation on critical path change at least the length of one of other paths and its length is greater than the updated critical path length.

We are going to uses the standard task graphs and famous scheduling algorithms for simulation and evaluation of RCPL and other pre-scheduling algorithms in future work.

References

1. Aronsson, P., Fritzson, P. (2005): A Task Merging Technique for Parallelization of Modelica Models. In: 4th International Modelica Conference, Hamburg
2. Aronsson, P., Fritzson, P. (2003): Task Merging and Replication using Graph Rewriting. In: 2nd International Modelica Conference, Germany 3. Ayed, M., Gaudiot, J. (2000): An efficient heuristic for code partitioning. Parallel Computing 26(4), 399–426
4. Kwok, Y., Ahmad, I. (1999): Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Computing Surveys (CSUR) 31(4), 406–471
5. Parsa, S., Soltani, N., Shariati, S. (2010): Task Merging for Better Scheduling. Lecture Notes in Computer Science, Springer, 311-316