Applied Mathematical Modelling 36 (2012) 6243–6263

Contents lists available at SciVerse ScienceDirect

Applied Mathematical Modelling

journal homepage: www.elsevier.com/locate/apm

# Benchmarking the clustering algorithms for multiprocessor environments using dynamic priority of modules

Pramod Kumar Mishra [a,*], Abhishek Mishra [b], Kamal Sheel Mishra [c], Anil Kumar Tripathi [b]

[a] Department of Computer Science & DST Centre for Interdisciplinary Mathematical Sciences, Banaras Hindu University, Varanasi 221 005, India
[b] Department of Computer Engineering, Institute of Technology, Banaras Hindu University, Varanasi 221 005, India
[c] Department of Computer Science, School of Management Sciences, Varanasi 221 011, India

ARTICLE INFO

*Article history:*
Received 17 July 2010
Received in revised form 15 January 2012
Accepted 3 February 2012
Available online 15 February 2012

*Keywords:*
Benchmarking
Clustering
Distributed computing
Homogeneous systems
Scheduling
Task allocation

ABSTRACT

In this paper we give some extensive benchmark results for some dynamic priority clustering algorithms for homogeneous multiprocessor environments. By dynamic priority we mean a priority function that can change with every step of the algorithm. Using dynamic priority can give us more flexibility as compared to static priority algorithms. Our objective in this paper is to compare the dynamic priority algorithms with some well known algorithms from the literature and discuss their strengths and weaknesses. For our study we have selected two recently proposed dynamic priority algorithms: CPPS (*Cluster Pair Priority Scheduling algorithm*) having complexity $O(|V||E|(|V|+|E|))$ and DCCL (*Dynamic Computation Communication Load scheduling algorithm*) having complexity $O(|V|^2(|V|+|E|)log(|V|+|E|))$ where $|V|$ is the number of nodes in the task graph, and $|E|$ is the number of edges in the task graph. We have selected a recently proposed randomized algorithm with static priority (RCCL: *Randomized Computation Communication Load scheduling algorithm*) and converted it into a dynamic priority algorithm: RDCC (*Randomized Dynamic Computation Communication load scheduling algorithm*) having complexity $O(ab|V|(|V|+|E|)log(|V|+|E|))$ where $a$ is the number of randomization steps, and $b$ is a limit on the number of clusters formed. We have also selected three well known algorithms from literature: DSC (*Dominant Sequence Clustering algorithm*) having complexity $O((|V|+|E|)log(|V|))$, EZ (*Edge Zeroing algorithm*) having complexity $O(|E|(|V|+|E|))$, and LC (*Linear Clustering algorithm*) having complexity $O(|V|(|V|+|E|))$. We have compared these algorithms using various comparison parameters including some statistical parameters, and also using various types of task graphs including some synthetic and real task graphs. Our results show that the dynamic priority algorithms give best results for the case of random task graphs, and for the case when the number of available processors are small.



## 1. Introduction

A parallel system is designed so that it can execute the applications faster than a sequential system. For this we need to parallelize the program. The three steps involved in the parallelization of a program [1] are: task decomposition, dependence analysis, and scheduling. By scheduling we mean both the temporal allocation (assigning a start time to the task) and the

* Corresponding author.
*E-mail addresses:* mishra@bhu.ac.in (P.K. Mishra), abhishek.rs.cse@itbhu.ac.in (A. Mishra), ksmishra@smsvaranasi.com (K.S. Mishra), aktripathi.cse@itbhu.ac.in (A.K. Tripathi).

0307-904X/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.apm.2012.02.011

spatial allocation (assigning a processor to the task). The tasks are allocated on different processors to exploit the parallelism so that the parallel execution time of the tasks can be reduced.

A dependence relation among the tasks is represented as a directed acyclic graph, also known as the *task graph*. Nodes in the task graph represent the tasks and have a weight associated with them that represents the running time of the task. Edges in the task graph represent the dependence relation among the tasks and have a weight associated with them that represents the communication time among the tasks.

The problem of finding a schedule for a given task graph on a given set of processors that takes minimum time is NP-Complete [2, 3].

A fundamental scheduling heuristic is called the *list scheduling* heuristic. In list scheduling, first we assign a priority scheme to the tasks. Then we sort the tasks according to the priority scheme, while respecting the precedence constraints of the tasks. Finally each task is successively scheduled on a processor chosen for it. Some examples of list scheduling algorithms are: Adam et al. [4], Coffman and Graham [5], Graham [6], Hu [7], Kasahara and Nartia [8], Lee et al. [9], Liu [10], Wu and Gajski [11], Yang and Gerasoulis [12].

Another fundamental scheduling heuristic is called *clustering*. Basically it is a scheduling technique for an unlimited number of processors. It is often proposed as an initial step in scheduling for a limited number of processors. A cluster is a set of tasks that are scheduled on the same processor. Clustering based scheduling algorithms generally consist of three steps. The first step finds a clustering of the task graph. The second step finds an allocation of clusters to the processors. The last step finds a scheduling of the tasks. Some examples of clustering based scheduling algorithms are Mishra et al. [13], Yang and Gerasoulis [14], Kim and Browne [15], Kadamuddi and Tsai [16], Sarkar [2], Hanen and Munier [17].

Several benchmarking for task scheduling algorithms are proposed in literature [18, 19, 20, 21]. In this paper we give some extensive benchmark results for some dynamic priority clustering algorithms for homogeneous multiprocessor environments. By dynamic priority we mean a priority function that can change with every step of the algorithm. Using dynamic priority can give us more flexibility as compared to static priority algorithms. Our objective in this paper is to compare the dynamic priority algorithms with some well known algorithms from the literature and discuss their strengths and weaknesses. For our study we have selected two recently proposed dynamic priority algorithms: CPPS (*Cluster Pair Priority Scheduling algorithm*) [22], having complexity $O(|V||E|(|V|+|E|))$ and DCCL (*Dynamic Computation Communication Load scheduling algorithm*) [23], having complexity $O(|V|^2(|V|+|E|)log(|V|+|E|))$ where $|V|$ is the number of nodes in the task graph, and $|E|$ is the number of edges in the task graph. We have selected a recently proposed randomized algorithm with static priority RCCL (*Randomized Computation Communication Load scheduling algorithm*) [24], and converted it into a dynamic priority algorithm: RDCC (*Randomized Dynamic Computation Communication load scheduling algorithm*) having complexity $O(ab|V|(|V|+|E|)log(|V|+|E|))$ where $a$ is the number of randomization steps, and $b$ is a limit on the number of clusters formed. We have also selected three well known algorithms from literature: DSC (*Dominant Sequence Clustering algorithm*) [14], having complexity $O((|V|+|E|)log(|V|))$, EZ (*Edge Zeroing algorithm*) [2], having complexity $O(|E|(|V|+|E|))$, and LC (*Linear Clustering algorithm*) [15], having complexity $O(|V|(|V|+|E|))$. We have compared these algorithms using various comparison parameters including some statistical parameters, and also using various types of task graphs including some synthetic and real task graphs. Our results show that the dynamic priority algorithms give best results for the case of random task graphs, and for the case when the number of available processors are small.

The rest of the paper is organized in the following manner: Section 2 presents a description of algorithms used in the benchmarking, Section 3 gives a description of performance evaluation parameters used, Section 4 gives a description of task graphs used, Section 5 gives the performance results for peer set task graphs, Section 6 gives the performance results for random task graphs, Section 7 gives the performance results for systolic array task graphs, Section 8 gives the performance results for Gaussian elimination task graphs, Section 9 gives the performance results for divide and conquer task graphs, Section 10 gives the performance results for fast Fourier transform task graphs, Section 11 gives the performance results for small random task graphs with optimal solutions, and finally we conclude in Section 12.

## 2. A description of algorithms used in the benchmarking

### 2.1. The CPPS algorithm

Mishra and Tripathi [22] consider the EZ algorithm [2] for scheduling precedence constrained task graphs on parallel systems as a priority based algorithm in which the priority is assigned to edges. In this case, the priority can be taken as the edge weight. This can be viewed as a task dependent priority function that is defined for pairs of tasks. In the CPPS algorithm [22] this idea is extended in which the priority is a cluster dependent function of pairs of clusters (of tasks):

$$P_c(C_i, C_j) = comm(C_i, C_j) + comm(C_j, C_i) - comp(C_i) - comp(C_j), \tag{1}$$

where $P_c(C_i, C_j)$ is the priority function that is defined for a pair of clusters, $comm(C_i, C_j)$ is the total communication cost from the cluster $C_i$ to the cluster $C_j$, $comm(C_j, C_i)$ is the total communication cost from the cluster $C_j$ to the cluster $C_i$, $comp(C_i)$ is the total computation cost of the cluster $C_i$, and $comp(C_j)$ is the total computation cost of the cluster $C_j$. This can be viewed as a dynamic priority algorithm that depends on the current allocation in each step of the algorithm. The CPPS algorithm has complexity $O(|V||E|(|V|+|E|))$.