Parallel Computation on Interval Graphs using PC Clusters: Algorithms and Experiments

A. Ferreira[1], I. Guérin Lassous[2], K. Marcus[3] and A. Rau-Chaplin[4]

[1] CNRS, INRIA, Projet SLOOP, BP 93, 06902 Sophia Antipolis, France. E-mail: ferreira@sophia.inria.fr.
[2] LIAFA – Université Paris 7, Case 7014, 2, place Jussieu, F-75251 Paris Cedex 05. E-mail: guerin@liafa.jussieu.fr.
[3] Eurecom, 2229, route des Cretes - BP 193, 06901 Sophia Antipolis cedex - France. E-mail: marcus@eurecom.fr.
[4] Faculty of Computer Science, Dalhousie University, P.O. Box 1000, Halifax, NS, Canada B3J 2X4. E-mail: arc@cs.dal.ca.

Abstract. The use of PC clusters interconnected by high performance local networks is one of the major current trends in parallel/distributed computing. These clusters can yield effective parallel systems for a fraction of the price of machines using special purpose hardware. Although significant effort has been undertaken on system-level and programming environment issues for such clusters, much less attention has been paid to some of the algorithmic issues. In this paper we show that theoretical (BSP-like) coarse-grained models are well adapted to solve important classes of problems on PC clusters. We give coarse-grained parallel algorithms to solve many problems arising in the context of interval graphs, namely connected components, maximum weighted clique, BFS and DFS trees, minimum interval covering, maximum independent set and minimum dominating set. All of the described p-processor parallel algorithms require only constant or $O(\log p)$ number of communication rounds and are efficient in practice as demonstrated by our experimental results obtained on a Fast Ethernet based cluster. In an interesting interplay between theory and practice we note that super-linear speedups occurred for large data because of swapping factors.

1 Introduction

The use of PC clusters interconnected by high performance local networks with raw throughput close to 1Gb/s and latency smaller than $10\mu s$ is one of the major current trends in parallel/distributed computing. The local networks are either realized with off-the-shelf hardware (e.g. Myrinet and Fast Ethernet), or application-driven devices, in which case additional functionalities are built-in, mainly at the memory access level. Such cluster-based machines (called henceforth PCC's) typically utilize some flavour of Unix and any number of widely available software packages that support multi-threading, collective communication, automatic load-balance, and others. Note that such packages typically simplify the programmers task by both providing new functionality and by promoting a view of the cluster as a single virtual machine. Clusters based on off-the-shelf hardware can yield effective parallel systems for a fraction of the price of machines using special purpose hardware. This kind of progress may thus be the key to a much wider acceptance of parallel computing, that has been postponed so far, perhaps primarily due to issues of cost and complexity.

Although a great deal of effort has been undertaken on system-level and programming environment issues as described above, little attention has been paid to methodologies for the design of algorithms for this kind of parallel systems. Despite the availability of a large number of built-in and/or highly optimized procedures, algorithms are still designed at the machine level and claims to portability lay only on the fact that they are implemented using communication libraries such as PVM or MPI.

In this paper we show that theoretical (BSP-like) coarse-grained models are well adapted to PCC's. In particular, algorithms designed for such models are portable and their theoretical and practical performance are closely related. Furthermore, they allow a reduction on the costs associated with software development since the main design paradigm is the use of existing sequential algorithms and communication sub-routines, usually provided with the systems.

Our approach will be to study a class of problems from the start of the algorithm design task until the implementation of the algorithms on a PCC. The class of problems will be those arising on a family of intervals on the real line which can model a number of applications in scheduling, circuit design, traffic control, genetics, and others [**?**, **?**, **?**].

Previous work

This class of problems provides a useful abstraction of many practical problems. Hence, it has been studied extensively in the parallel setting and many fine-grained PRAM algorithms have been described [**?**], as shown in Table **??**.

Problem	Time
Maximum weighted clique [**?**]	$O(\log(n))$
Maximum independent set [**?**]	$O(\log(n))$
Minimum clique cover [**?**, **?**]	$O(\log(n))$
Minimum dominating set [**?**, **?**]	$O(\log(n))$
Depth-first spanning tree [**?**, **?**]	$O(\log(n))$
Breadth-first spanning tree [**?**, **?**]	$O(\log(n))$
Connected components (PRAM CRCW) [**?**]	$O(\log(n))$
(using $\{[m+n]\alpha(m,n)]/\log n\}$ processors,	
where $\alpha(m,n)$ is the inverse of the Ackerman function)	

Table 1. Interval graph problems and the required time for PRAM algorithms with n processors, unless stated otherwise. The sequential complexity is $\Theta(n \log n)$ in all cases.

Whereas fine-grained PRAM algorithms are likely to be efficient on fine-grained shared memory architectures, it is common knowledge that they tend to be impractical on PCC's due to their failure to exploit locality. Therefore, there has been a recent growth of interest in coarse-grained computational models [?, ?, ?, ?] and the design of coarse-grained algorithms [?, ?, ?, ?, ?, ?, ?].

The BSP model, described by Valiant [?], uses slackness in the number of processors and memory mapping via hash functions to hide communication latency and provide for the efficient execution of fine grained PRAM algorithms on coarse-grained hardware. Culler et. al. introduced the LogP model which, using Valiant's BSP model as a starting point, focuses on the technological trend from fine grained parallel machines towards coarse-grained systems and advocates portable parallel algorithm design [?]. Other coarse grained models focus more on utilizing local computation and minimizing global operations. These include the Coarse-Grained Multicomputer (CGM) model used in this paper [?]. In this mixed sequential/parallel setting, there are three important measures of any coarse-grained algorithm, namely, the amount of *local computation* required, the number and type of *global communication phases* required and the *scalability* of the algorithm, that is, the range of values for the ratio $\frac{n}{p}$ for which the algorithm is efficient and applicable.

Recently, Cáceres et al. [?] showed that many problems in general graphs, such as list ranking, connected components and others, can be solved in $O(\log p)$ communication rounds in BSP and CGM. Note that while this work is of significant theoretical interest, these algorithms involve simulation of their corresponding PRAM algorithm for $\log n$ communication phases which is both complex to implement and computationally expensive in practice. So whereas in theory these results yield algorithms in interval graphs for coarse-grained machines, in practice a different approach is called for. Unlike general graphs, interval graphs can be more easily partitioned and treated in the distributed memory setting. Since each interval is given by its two extreme points, they can be sorted by left and/or right endpoints and distributed according to this ordering. This partitioning allows us to design less complex parallel algorithms; moreover, the derived algorithms are easier to implement and faster both in theory and in practice.

Our work

We describe constant communication round coarse-grained parallel algorithms to solve a set of the standard problems arising in the context of interval graphs [?], namely connected components [?], maximum weighted clique [?] and breadth-first-search (BFS) and depth-first-search (DFS) trees [?, ?]. We also propose $O(\log p)$ communication round algorithms for optimization problems as minimum interval covering, maximum independent set [?] and minimum dominating set [?].

In order to demonstrate the practicability of our approach, we implemented three of the above algorithms on a PCC interconnected by a Fast Ethernet backbone. Because of the paradigms used, the programs were easy to develop and are quite portable. The results presented in this paper show that high performance can be achieved with off-the-shelf PCC's along with the right model for algorithm design. Interestingly, super-linear speedups were observed in some cases due to memory swapping effects. Using multiple processors allows us to effectively utilize more RAM and therefore allows computation on data sets that are simply too large to be effectively processed on single processor machines.

In Section **??** we review the coarse-grained model adopted in this paper. In Section **??** the basic operations are described. Then, in Section **??**, chosen problems in interval family model are presented, and solutions are proposed using the basic operations from Section **??**. In Section **??**, we describe experiments on a Fast Ethernet based PCC. We close the paper with some conclusions and directions for further research.

2 The coarse-grained model

In a "bulk synchronous" processing model, an input of size n is distributed evenly across a p-processor parallel computer. In a single *computation round* or *superstep* each processor may send and receive h messages and then perform an internal computation on its internal memory cells using the messages it has just received. To avoid conflicts that might be caused by asynchronies in the network (whose topology is left undefined) the messages sent out in a round t by some processor cannot depend upon any messages that the processor receives in round t.

In this paper we use the Coarse-Grained Multicomputer model, or *CGM*(n, p) for short, introduced in [**?**]. The CGM(n, p) is a BSP model consisting of a set of p processors with $O(\frac{n}{p})$ local memory each. The term "coarse grained" refers to the fact that (as in practice) $O(\frac{n}{p})$ is defined to be "considerably larger" then $O(1)$. The definition of "considerably larger" is $\frac{n}{p} \geq p^{\epsilon}$, where ϵ depends on the proposed algorithms; in this paper $\epsilon = 1$.

The following result will be used in the remaining to achieve a constant number of communication rounds in the solution of many problems.

Theorem 1. [**?**] *Given a set S of n items stored $O(n/p)$ per processor on a CGM(n, p), $n/p \geq p$, sorting S takes a constant number of communication rounds.* □

The algorithms proposed for the CGM are independent of the communication network. Moreover, it was proved that the main collective communication operations can be implemented by a constant number of calls to global sort ([**?**]). Hence, by Theorem **??**, these operations take a constant number of communication rounds. However, in practice these operations will be implemented through built-in, optimized system-level routines. In the remainder, let $T_S(n, p)$ denote the time complexity of a global sort in the CGM.