Graph Coloring on a Coarse Grained Multiprocessor
(Extended Abstract)

Assefaw Hadish Gebremedhin\textsuperscript{1}, Isabelle Guérin Lassous\textsuperscript{2},
Jens Gustedt\textsuperscript{3}, and Jan Arne Telle\textsuperscript{1}

\textsuperscript{1} University of Bergen, Norway. (assefaw,telle)@ii.uib.no
\textsuperscript{2} INRIA Rocquencourt, France. Isabelle.Guerin-Lassous@inria.fr
\textsuperscript{3} LORIA & INRIA Lorraine, France. Jens.Gustedt@loria.fr

Abstract. We present the first efficient algorithm for a coarse grained multiprocessor that colors a graph $G$ with a guarantee of at most $\Delta_C + 1$ colors.

1 Introduction and Overview

The problem of graph coloring is crucial both for the applications of graph algorithms to real world problems and for the domain of parallel graph algorithms itself. For the latter, graph colorings using a bounded number of colors are often used in a theoretical setting to ensure the independence of tasks that are to be accomplished on the vertices of a graph, i.e. since the color classes form independent sets that don’t interact each one of them can be treated in parallel. For a long time, no efficient parallel implementation of a graph coloring heuristic with good speedups was known [1]. However, in a recent result, Gebremedhin and Manne [4,5] present an algorithm and an implementation for a shared memory computer that proves to be theoretically and practically efficient with good speedups.

In this paper we make this successful approach feasible for a larger variety of architectures by extending it to the more general setting of coarse grained multiprocessors (CGM) [3]. This model of parallel computation makes an abstraction of the interconnection network between the processors of a parallel machine (or network) and tries to capture the efficiency of a parallel algorithm using only a few parameters. Several experiments show that the CGM model is of practical relevance: implementations of algorithms formulated in the CGM model in general turn out to be feasible, portable, predictable and efficient [7].

This paper is organized as follows. In the next section we review the coarse grained models of parallel computation and the basics of graph coloring heuristics. Then, we present our algorithm together with an average case analysis of its time and work complexity. Finally, we show how to handle high degree vertices and how to alter the algorithm to achieve the same good time and work complexity also in the worst-case.

© Springer-Verlag Berlin Heidelberg 2000
1.1 Coarse Grained Models of Parallel Computation

In recent years several efforts have been made to define models of parallel (or distributed) computation that are more realistic than the classical PRAM models. In contrast to the PRAM, these new models are coarse grained, i.e. they assume that the number of processors $p$ and the size of the input $N$ of an algorithm are orders of magnitudes apart, $p \ll N$. By that assumption these models map much better on existing architectures where in general the number of processors is at most some thousands and the size of the data that are to be handled goes into millions and billions.

This branch of research got its kick-off with Valiant [8] introducing the so-called bulk synchronous parallel machine, BSP, and was refined in different directions for example by Culler et al. [2] to LogP, and by Dehne et al. [3] to CGM.

We place ourselves in the context of CGM which seems to be well suited for a design of algorithms that are not too dependent on an individual architecture. We summarize the assumptions of this model as follows:

- All algorithms perform in so-called supersteps. A superstep consists of one phase of interprocessor communication and one phase of local computation.
- All processors have the same size $M = O(N/p)$ of memory.
- The communication network between the processors can be arbitrary.

The goal when designing an algorithm in this model is to keep the individual workload, time for communication and idle time of each processor within $T/s(p)$ where $T$ is the runtime of the best sequential algorithm on the same data and $s(p)$, the speedup, is a function that should be as close to $p$ as possible. To be able to do so, it is considered as good idea to keep the number of supersteps of such an algorithm as low as possible, preferably $o(M)$.

The rationale for that is that for the communication time there are at least two invariants of the architecture that come into play: the latency, i.e. the minimal time a communication needs to startup before any data reach the other end, and the bandwidth, i.e. the overall throughput per time unit of the communication network for large chunks of data. In any superstep there are at most $O(p)$ communications for each processor and so a number of supersteps of $o(M)$ ensures that the latency can be neglected for the performance analysis of such an algorithm. The bandwidth restriction of a specific platform must still be observed, and here the best strategy is simply to reduce the communication volume as much as possible. See Guérin Lassous et al. [7] for an introduction and overview on algorithms, code and experiments within the coarse grained setting.

As a legacy from the PRAM model it is usually assumed that the number of supersteps should be polylogarithmic in $p$, but there seems to be no real world rationale for that. In fact, no relationship of the coarseness models to the complexity classes $NC^k$ have been found, and algorithms that simply ensure a number of supersteps that are a function of $p$ (and not of $N$) perform quite well in practice [6].