1 Introduction

Partitioning a set of objects into groups according to certain rules is a fundamental process in mathematics and computer science. Often, the rules that determine whether or not a pair of objects can belong to the same group are conceptually simple. *Graph coloring* is one mathematical abstraction for such partitioning rules.

A graph represents binary relationships. It can be visualized as a diagram consisting of points called vertices, and lines called edges. A vertex represents some physical entity or abstract concept. An edge, which joins exactly two vertices, represents the relationship between them.

In its standard form, the graph coloring problem deals with assigning colors to the vertices of a graph such that adjacent vertices receive different colors and the number of colors used is minimized. This problem and its variants model a wide range of real-world problems in scientific, engineering and computing applications. For example, timetabling and scheduling, frequency assignment to mobile telephones, register allocation in compilers, printed circuit testing, computation of sparse Jacobian and Hessian matrices, partitioning of tasks in parallel computation, are just a few of the many areas in which some variant of the graph coloring problem is used as an effective model.

Among the examples mentioned above, computation of sparse derivative matrices is an application area considered in this study. The computation of Jacobian and Hessian matrices is a phase required in many numerical optimization algorithms. Automatic differentiation and finite differencing are two widely used techniques for computing, or estimating, the entries of these matrices. The computation involved in using the techniques can be made efficient by exploiting matrix structures such as sparsity and symmetry. This leads to several types of partitioning problems that can naturally be modeled using variants of the graph coloring problem.

This thesis is mainly about developing *algorithms* for solving different graph coloring problems. The coloring problems of our concern are all believed to be intractable. Therefore, we focus on finding polynomial-time algorithms that yield sub-optimal solutions. In particular, we emphasize greedy algorithms that give usable solutions quickly.

Like many other industrial problems, the graph coloring instances in our cases can be of very large size. One way of handling large-scale problems effectively is by using *parallel computers*. A parallel computer consists of several processors that work on different parts of a common problem. Today, parallel computers consisting of hundreds of processors are commercially available. The basic idea behind parallel computation is to carry out several tasks simultaneously and thereby reduce execution time. A successful application of this idea requires, among other things, developing parallel algorithms—algorithms that handle the task-division, coordination, and communication among the processors. Most of the coloring algorithms developed in this thesis are parallel.

In general, the design and analysis of algorithms, sequential as well as parallel, presumes the existence of an underlying *computation model*. Unlike the sequential case, there is currently no single unifying standard model for parallel computation. The literature contains an abundant number of models. Among them, the Bulk Synchronous Parallel (BSP) model stands out for being proposed as a standard bridging model between hardware and software in parallel computation. The introduction of BSP stimulated several studies that lead to various adaptations of the model along different directions. The Coarse Grained Multicomputer (CGM) model is one such adaptation. Inspired by the BSP and CGM models, in this study, we propose a new model called the Parallel Resource-Optimal (PRO) model. The PRO model is proposed in an attempt to further narrow the gap between theory and practice in developing efficient and scalable parallel algorithms.

In summary, this thesis deals primarily with the design, analysis, and implementation of practical parallel algorithms for a variety of graph coloring problems. Besides the standard coloring problem, we consider several specialized variants that arise in the computation of sparse Jacobian and Hessian matrices. Secondarily, it is concerned with the development of a simple and effective parallel computation model suitable for theoretical as well as practical purposes. The thesis consists of the following five papers:

I. A.H. Gebremedhin and F. Manne. *Scalable Parallel Graph Coloring Algorithms*. Concurrency: Practice and Experience. 2000; 12:1131-1146.

II. A.H. Gebremedhin, F. Manne and A. Pothen. *Graph Coloring in Optimization Revisited*. January 2003. To be submitted.

III. A.H. Gebremedhin, F. Manne and A. Pothen. *Parallel Distance-k Coloring Algorithms for Numerical Optimization*. In B. Monien and R. Feldmann (Eds.): Proceedings of the 8th International Euro-Par Conference, Paderborn, Germany, August 2002, volume 2400 of Lecture Notes in Computer Science, pages 912-921. Springer-Verlag 2002.

IV. A.H. Gebremedhin, I.Guérin Lassous, J. Gustedt and J.A. Telle. *Graph Coloring on Coarse Grained Multicomputers.* Discrete Applied Mathematics, to appear.

A preliminary version appeared In U. Brandes and D. Wagner (Eds): Proceedings of WG 2000, 26th International Workshop on Graph-Theoretic Concepts in Computer Science, Konstanz, Germany, June 2000, volume 1928 of Lecture Notes in Computer Science, pages 184-195. Springer-Verlag 2000.

V. A.H. Gebremedhin, I.Guérin Lassous, J. Gustedt and J.A. Telle. *PRO: a Model for Parallel Resource-Optimal Computation.* In Proceedings of HPCS'02, 16th Annual International Symposium on High Performance Computing Systems and Applications, Moncton, Canada, June 2002, pages 106-113. IEEE Computer Society Press 2002.

The versions of Papers I, IV and V included in this thesis are content-wise identical to the published versions. Paper III is slightly extended from its published form: it includes more experimental results and proofs that were omitted for space considerations. Paper II is planned to be submitted for publication; unlike the other papers, it is written with a broader audience in mind.

The topics addressed by Papers I–V can be classified into three groups:

- parallel algorithms for the standard graph coloring problem (Paper I),
- application of graph coloring in sparse Jacobian and Hessian computation (Papers II and III), and
- parallel computation models (Papers IV and V).

The purpose of this common introduction is to highlight the main results in each paper and to put the results in context. It is organized as follows. In Section 2 we discuss some parallel processing issues relevant for the subsequent presentation of the papers. In Sections 3–5, we consider the three topics listed above in their respective order. In particular, in each section, a concise summary of the corresponding paper(s) is given. Paper IV has aspects that make it relevant to both the first and the last topic. It is, however, presented under the last topic in Section 5. We conclude the common introduction in Section 6 with some remarks. The body of the thesis, Papers I–V, follows thereafter.

2 Multiprocessor parallel computing

The ever increasing need for faster solutions and for solving large scale problems is one of the major driving forces behind parallel computing. The primary objective here is to achieve increased computational speed by employing a number of processors concurrently.

Concurrent computation can be done at different levels. Our focus is on using a *multiprocessor parallel computer* where a number of processors communicate and cooperate to solve a *common* computational problem. Multiprocessor parallel computation involves three key ingredients: hardware, software (programming languages, operating systems and compilers), and parallel algorithms [18]. One can use these ingredients as basis for classifying the different kinds of *models* used in the context of parallel processing. Specifically, we identify three distinct, and yet closely related, categories of models called *parallel architecture*, *parallel programming*, and *parallel computation (algorithmic)* model. These categories are introduced merely to frame the discussion in the rest of the current section. The discussion focuses on issues relevant to the presentation of the thesis papers.

2.1 Parallel architecture models

Parallel computers are often divided into different *architectural* classes depending on such issues as *control mechanism* (SIMD and MIMD computers), *address space organization* (shared address space and message-passing architectures) and *interconnection network* (static and dynamic networks). More information on each of these classes can be found in books such as [9, 19].

2.2 Parallel programming models

From a *programming* point of view, shared address space[1] programming (SASP) and message-passing programming (MPP) are perhaps the two most widely used models in contemporary parallel computation.

In the SASP model, programs are viewed as a collection of processes accessing a central pool of shared variables. Data exchange among the processors is achieved by reading from, and writing to, the shared variables. This programming style is naturally suited to the shared-address-space architecture. In the MPP model, programs are viewed as a collection of processes with private local variables and the ability to exchange data via explicit mes-

[1]The phrases 'shared address space' and 'shared memory' are often used interchangeably, both in the literature and here in this thesis. There is, however, a difference in meaning: the former does not require memory to be physically shared.

sage passing. In this model, no variables are shared among processors. Each processor uses its local variables for computation, and whenever necessary, it sends data to, or receives data from, other processors. This programming model fits naturally to distributed-memory computers.

In using the SASP and MPP models, there is usually a trade-off in the choice to be made. MPP requires considerably more programming effort than SASP. This is mainly because MPP requires the data structures in a program to be explicitly partitioned whereas SASP does not. Moreover, SASP supports incremental parallelization of an existing sequential application whereas MPP does not. On the other hand, applications developed using MPP are likely to be more scalable than SASP based applications.

Currently, OpenMP [23] and MPI [21] are generally considered to be the standards for writing portable parallel programs using the SASP and MPP models, respectively. OpenMP, in which the parallel algorithms presented in Papers I and III are implemented, is a directive-based fork-join model for SASP. The fact that OpenMP can be used to write parallel programs relatively easily makes its usage appealing[2].

The experimental work reported in Papers I and III is performed on a Cray Origin 2000 [24], a machine that supports both SASP and MPP. In this machine, memory is physically distributed among the processors. However, for operational purposes, it behaves as a shared memory computer with the operating system taking care of maintaining memory consistency. This enables any processor to use the entire memory of the system.

2.3 Parallel computation models

A computation model is required to serve two major purposes [1]. First, it is used to *describe* a real entity, namely, a computer. As such, a computation model attempts to capture the essential features of a machine while ignoring less important details of its implementation. Second, it is used as a tool for analyzing problems and expressing *algorithms*. In this sense, the model is not necessarily linked to any real computer but rather to an understanding of *computation*.

In the realm of sequential computation, the Random Access Machine (RAM) is a standard model that has succeeded in achieving both of these purposes. It serves as an effective model for hardware designers, algorithm developers and programmers alike. When it comes to parallel computation, there is no such unifying standard model. This is mainly due to the complex

[2]OpenMP dedicated conference series such as EWOMP and WOMPAT could be cited as indicators for the increasing interest in using OpenMP.

set of issues inherent in parallel computation.

The natural parallel analogue of RAM, the PRAM, is an overly simplistic model for parallel computation. It is mainly of theoretical interest as it fails to capture the features of existing parallel computers. The Bulk Synchronous Parallel (BSP) model [26] is a relatively recent model, proposed to serve as a standard 'bridging model' between hardware (machine architecture) and software (algorithm design and programming). As opposed to the PRAM, parallel algorithms in the BSP model are organized in distinct *computation* and *communication* phases. Moreover, unlike the PRAM, the BSP model made parallel computation to be *coarse grained*, a concept we will make more precise shortly. Later, a variant of the BSP called the Coarse Grained Multicomputer (CGM) model was proposed [4, 11, 12]. The CGM model added a strictly coarse grained *communication* requirement to the BSP model. The CGM model uses fewer number of parameters than the BSP model, a feature that simplifies analysis to a certain extent.

In Paper V we propose a new parallel computation model called the Parallel Resource-Optimal (PRO) model. The PRO model inherits the advantages offered by the BSP and CGM models. It compromises between theoretical and practical considerations in the design of optimal and scalable parallel algorithms. Paper V is introduced in Section 5. However, since the BSP and CGM models are mentioned rather briefly in the paper, we discuss the key attributes of the two models in Sections 2.3.2 and 2.3.3 below. The two models are presented here as defined in [4].

We proceed in this section by first recapitulating the main features of the well known PRAM model, which is the model used in Papers I and III for theoretical runtime analysis. In the same section we include a discussion of a related complexity class. A good introduction to the PRAM model and parallel algorithm design using the model can be found in books such as [1, 16].

2.3.1 PRAM

The PRAM is an idealized parallel computation model. In its standard form, it consists of an arbitrarily large number of processors and a shared memory of unbounded size that is uniformly accessible to all processors. The processors share a common clock and operate in lockstep, but they may execute different instructions in each cycle. The PRAM is therefore a model for a synchronous shared memory MIMD computer. The PRAM can be classified into different subclasses depending on how simultaneous memory accesses are handled. For example, a subclass that allows simultaneous read access, but not write access, to a single memory location is called concurrent read exclusive write (CREW) PRAM.

The PRAM is a model for *fine-grain* computation as it supposes that the number of processors can be arbitrarily large. Usually, it is assumed that the number of processors is a polynomial in the input size. However, practical parallel computation is typically *coarse-grain*. In particular, on most existing parallel machines, the number of processors is several orders of magnitude less than the input size.

Despite its serious deviation from the nature of real parallel computers, the PRAM model is sometimes used for theoretical runtime analysis. A PRAM-analysis gives an idea on the 'computational parallelism' rendered by an algorithm, leaving communication costs aside. Such an analysis should be supported by empirical results in order to make reasonable conclusions. In Papers I and III we use the PRAM model for theoretical analyses and support the claims with experimental results. It should, however, be noted that the algorithms in Paper I and III are not actually PRAM-algorithms. First, they do not assume lockstep synchronization. Second, they are developed in a coarse-grain setting.

The class NC On the PRAM model, a parallel algorithm is considered 'efficient' if its running time is polylogarithmic, i.e., $O(\log^k n)$ for some fixed constant k, while the number of processors it uses is polynomial in the input size n. The class of problems that can be solved by such efficient parallel algorithms is called NC [16]. By simulation, an NC-algorithm can be converted into a polynomial time sequential algorithm. Thus, the class NC is included in the class P, i.e., NC $\subseteq$ P. On the other hand, whether or not P $\subseteq$ NC is an open problem in complexity theory. The general belief is that P $\nsubseteq$ NC, and hence that there are problems in P that do not have NC-algorithms. The class of P-complete problems consists of the most likely candidates for such problems. Informally, a problem is said to be P-complete if an NC-algorithm for it implies that all problems in P have NC-algorithms.

The NC versus P-complete dichotomy on PRAM suffers from several drawbacks. Most importantly, the dichotomy excludes practical parallelizability in a coarse-grain setting and secondly it does not take work optimality into account. A more detailed discussion of these drawbacks is given in Paper V. Moreover, the experimental results reported in Papers I and III demonstrate the relevance of developing coarse-grain algorithms.

2.3.2 The BSP model

A BSP computer is a collection of processor/memory modules connected by a router that can deliver messages in a point to point fashion between the processors. A BSP-algorithm is divided into a sequence of *supersteps* separated by barrier synchronizations. A superstep has distinct *computation* and *communication* phases. In a computation phase the processors perform computation on data that exists locally at the beginning of the superstep. In a communication phase, data is exchanged among the processors via the router. The BSP model uses the four parameters, n, p, L, and g. Parameter n is the problem size, p is the number of processors, L is the minimum time between synchronization steps (measured in basic computation units), and g is the ratio of overall system computational capacity (number of computation operations) per unit time divided by the overall system communication capacity (number of messages of unit size that can be delivered by the router) per unit time.

In a superstep, a processor may send (and receive) at most h messages. Such a communication pattern is called an *h-relation* and the basic task of the router is to realize arbitrary h-relations.

2.3.3 The CGM model

The CGM model is a specialization of the BSP model in which the number of parameters involved is reduced to just two—p and n—making theoretical analysis relatively simpler. The CGM model is a collection of p processors, each with $O(n/p)$ local memory, interconnected by a router that can deliver messages in a point to point fashion. A CGM algorithm consists of an alternating sequence of *computation rounds* and *communication rounds* separated by barrier synchronizations. A computation round is equivalent to the computation phase of a superstep in the BSP model. A communication round consists of a single h-relation with $h \leq n/p$. In other words, all the information sent from one processor to another in one communication round is packed into one long message, thereby minimizing communication overhead.

The CGM model is used for designing parallel algorithms for coarse grain systems, in particular, in the case where $n \gg p$. Usually, a lower bound on n/p, e.g. $n/p \geq p^{\delta}$, for some $\delta > 0$, is required [4]. The value of δ depends on the nature of the problem for which a parallel algorithm is sought. The goodness of a CGM parallel algorithm is often measured by the number of communication rounds it uses [4].

3 Parallel algorithms for standard graph coloring - Paper I

In the simplest case, the graph coloring problem asks for an assignment of positive integers (called colors) to the vertices of a simple, connected, undirected graph such that no two adjacent vertices are assigned the same color and the total number of colors used is minimized.

The graph coloring problem is NP-hard. The current best known approximation ratio for the problem is $O(n\frac{(\log\log n)^2}{(\log n)^3})$, where n is the number of vertices in the graph. Moreover, it is known to be not approximable within $n^{1/7-\epsilon}$ for any $\epsilon > 0$ [3].

Despite these rather pessimistic theoretical results, *greedy* coloring heuristics are often found to be effective in practice. In some applications, these heuristics find colorings that are within small additive constants of the optimal coloring [6, 17]. The number of colors used by a greedy heuristic depends on the order in which the vertices are visited and the choice of color made at each step. If the smallest possible color is chosen at each step, the number of colors used by any such sequential greedy heuristic is at most $\Delta + 1$, where Δ is the maximum degree in the graph. For some applications, this bound can be quite acceptable, especially if the coloring is obtained quickly.

Such greedy heuristics are inherently sequential and consequently difficult to parallelize. Specifically, the problem of coloring the vertices of a graph in a *prespecified* order and using the smallest possible color at each step is known to be P-complete [13].

Paper I deals with parallelizing greedy heuristics using the shared-address-space programming model in a coarse-grain setting. Similar previous efforts focused on using the message-passing programming model on distributed-memory parallel computers. These efforts yielded no speedup [2, 17]. The algorithms presented in Paper I overcome this shortcoming.

In Paper I we present a new parallel algorithm that colors a graph $G = (V, E)$ using at most $\Delta + 1$ colors and has an expected parallel runtime $O(|E|/p)$ on a CREW PRAM for any number of processors p such that $p \leq |V|/\sqrt{2|E|}$.

The algorithm consists of three phases. The first two phases are parallel while the last one is sequential. In the first phase, the vertex set is simply equally partitioned among the p available processors. Each processor colors its block of vertices paying attention to already colored (local and non-local) vertices. In this scenario, a pair of adjacent vertices residing on different processors may be colored concurrently, and possibly get the same color, thereby leading to an inconsistency. In the second phase, inconsistencies are detected in parallel. In the final sequential phase, the detected inconsistencies are rectified.

In the same paper we present a second algorithm that extends this idea to reduce the number of colors used. The reduction in the number of colors is obtained by dividing the first phase of the algorithm just described into two coloring steps. In the improved algorithm, the coloring in the first step is used to determine an ordering for a coloring in the second step.

Both algorithms are well suited for the shared address space programming model and are implemented using OpenMP. In agreement with the theoretical analyses, timing results obtained from experiments conducted on the Origin 2000 using a modest number of processors and graphs from finite element methods and eigenvalue computation show that the algorithms yield speedup.

4 Graph coloring and efficient computation of sparse derivative matrices

Computing a derivative matrix, i.e. a Jacobian or a Hessian, is a phase required in many numerical optimization algorithms. In large scale problems, this phase often constitutes an expensive part of the entire computation. Hence, efficient methods that exploit matrix structure such as sparsity and symmetry are often required. Finite difference and automatic differentiation (AD) are two widely used methods in the efficient determination of the elements of derivative matrices. In using these methods, the main goal is to minimize the number of function evaluations or AD passes required. The pursuit of this goal gives rise to several *matrix partitioning problems.*

Since the early 70's a considerable amount of research work has been done in the field of sparse derivative matrix computation [5, 6, 7, 8, 10, 14, 15, 20, 22, 25]. Early studies in this field, such as [10, 22, 25], used matrix oriented approaches. Later, following the pioneering work of Coleman and Moré [6], *graph theoretic* approaches proved to be advantageous in understanding, analyzing and solving the matrix problems. In particular, graph coloring—in its several variations—was found to be an effective model. However, existing algorithms and software for such coloring problems are developed for uniprocessor (sequential) computers. Currently, there is a need for parallel coloring algorithms that can be used to aid the solution of large-scale PDE-constrained optimization problems on parallel computers. Paper II is an expository work for, and Paper III is a first step towards, addressing this need in a flexible manner.

Matrix	Partition	Scheme	Entries	Prob.	Formulation	
Nonsym.	1d	direct	all	P1	D2 coloring	(II)
Sym.	1d	direct	all	P2	D$\frac{3}{2}$ coloring	[7]
Nonsym.	2d	direct	all	P3	D$\frac{3}{2}$ bicoloring	[8]
Sym.	1d	subst.	all	P4	acyclic coloring	[5]
Nonsym.	2d	subst.	all	P5	acyclic bicoloring	[8]
Nonsym.	1d	direct	some	P6	restricted D2 coloring	(II)
Sym.	1d	direct	some	P7	restricted D$\frac{3}{2}$ coloring	(II)
Nonsym.	2d	direct	some	P8	restricted D$\frac{3}{2}$ bicoloring	(II)
Sym.	1d	subst.	some	P9	not treated	
Nonsym.	2d	subst.	some	P10	not treated	

Table 1: Overview of partition/coloring problems considered in Paper II.

4.1 Unifying Framework – Paper II

In the literature, one finds several types of coloring problems characterizing different types of matrix partitioning problems. Understanding the similarities and differences among these problems greatly simplifies the development of algorithms—sequential or parallel—for solving them. Paper II aims at contributing to this end.

The nature of a partitioning problem in the context of efficient computation of sparse derivative matrices depends on the type of the underlying matrix and the scenario under which the matrix computation is carried out. Specifically, the particular problem depends on,

1. whether the matrix to be computed is *symmetric* or *nonsymmetric*,
2. whether a *one-dimensional* partition (involving only columns or rows) or a *two-dimensional* partition (involving both columns and rows) is used,
3. whether the evaluation scheme employed is *direct* (solves a diagonal system) or *substitution* based (solves a triangular system), and
4. whether *all* of the nonzero entries of the matrix or only *some* of them need to be determined.

The factors outlined above are mutually orthogonal to each other—potentially resulting in 2^4 different cases. Within our framework, ten of these correspond to practically meaningful partitioning problems whereas the remaining six do not. In Paper II, we use a unified graph theoretic framework to study eight of these problems; the remaining two are not considered due to their sophistication. The upper part of Table 1 gives an overview of the problems considered in the paper and the respective coloring formulations used. The lower part shows the cases not treated in the paper.

Paper II is a review as well as research paper that integrates known and new formulations. In the paper we give the necessary background to motivate and formally introduce the matrix partitioning problems, and then develop their equivalent graph coloring formulations. Problems P1–P5 have been studied previously whereas P6–P8 are investigated for the first time. The motivation for introducing the new problems and the corresponding restricted coloring formulations is the fact that the computation of a specified subset of the nonzero entries of a matrix—for instance, for preconditioning purposes—can be carried out even more efficiently than the computation of all of the nonzero entries.

In formulating problems P1–P8 as coloring problems, we use a bipartite graph to represent a nonsymmetric matrix and an adjacency graph to represent a symmetric matrix. In the case of nonsymmetric matrices, bipartite graph based formulations are shown to be attractive in terms of flexibility and storage space requirement in comparison with known formulations based on *intersection* graphs.

The major contributions made in Paper II can be summarized as follows.

- We propose distance-2 coloring as an alternative, more flexible, formulation for problem P1 as compared to the distance-1 coloring formulation of Coleman and Moré [6] for the same problem.

- We expose the interrelationship among the coloring formulations of problems P1 through P5 and identify distance-2 coloring as as a *generic* model.

- For problems P3 and P5, based on the previously known relationship to graph bicoloring [8], we observe a connection to finding a vertex cover in a graph.

- Using the insight gained from the unified graph theoretic treatment, we develop several simple, sequential, heuristic algorithms for the coloring formulations of problems P1 through P5.

- We formulate the partitioning problems arising in the computation of a subset of the entries of a matrix (i.e. problems P6–P8) as restricted coloring problems.

4.2 Parallel Algorithms – Paper III

In Paper III, we develop several shared address space parallel algorithms for the coloring formulations of problems P1 and P2. The algorithms are straightforward extensions of the algorithm for the standard (distance-1) graph coloring problem presented in Paper I. However, in the cases where the graph to be colored is relatively dense, *randomization* is used as a supplementary tool to improve scalability. Our PRAM-analyses show that the algorithms give almost linear speedup for sparse graphs that are large relative to the number of processors. The algorithms are implemented using OpenMP and results from experiments conducted on the Origin 2000 using various large graphs show that the algorithms indeed yield reasonable speedup for a modest number of processors.

5 A new parallel computation model

5.1 Graph coloring on the CGM model – Paper IV

The aim of Paper IV is to adapt the shared address space algorithm for the standard graph coloring problem given in Paper I to the CGM model. The CGM model used is basically the same as the CGM model described in Section 2.3.3 except for the fact that the quality of an algorithm is measured not using communication rounds but rather using an explicit analysis of the overall computation and communication cost involved. Particularly, a CGM algorithm is considered efficient if the parallel runtime of the algorithm, taking both communication and computation costs into account, yields a speedup (relative to the complexity of the best known sequential algorithm) that is a linear function in the number of processors used.

In Paper IV we present such an efficient CGM algorithm that colors a connected graph $G = (V, E)$ using at most $\Delta + 1$ colors, where Δ is the maximum degree in G. The algorithm is given in two variants: *randomized* and *deterministic*. It is shown that on a p-processor CGM model, where $|E|/p > p^2$, the proposed algorithms have a parallel running time $O(|E|/p)$ and an overall (computation and communication) cost $O(|E|)$. These bounds correspond to the *average* and *worst* case for the randomized and deterministic versions, respectively.

In comparison with the original algorithm in Paper I, the CGM coloring algorithm in Paper IV is more complicated. Some of the factors that contribute to the complexity are distributed-memory related issues such as data distribution, communication overhead, and load balancing. In terms of the underlying algorithmic idea, a key difference between the algorithms of Paper I and IV is that the latter uses *recursion*.

5.2 The PRO model – Paper V

The notion of 'efficiency' of a CGM algorithm as used in Paper IV motivated us to refine the definition of the model itself. One of the goals was to keep the number of parameters in the refined model as low as possible and at the same time make the algorithm design objective more precise. Another goal was to find a model that can serve as a design scheme for the algorithm developer, i.e., a model that identifies the features of a parallel algorithm that contribute to its practical scalability on different architectures. The PRO model is partly a result of this effort.

The novel idea behind the PRO model is the focus on relative resource optimality and the use of granularity as a quality measure. In this model, the design and analysis of a parallel algorithm is done relative to the complexity of a specific underlying sequential algorithm. The parallel algorithm is required to be time and space optimal with respect to the complexity of the reference sequential algorithm. As a consequence of its optimality, a PRO-algorithm yields linear speedup. The quality of a PRO-algorithm is measured by an attribute of the model called granularity function. This function, which is expressed in terms of the input size, shows the number of processors that can be employed without sacrificing optimality.

In Paper V, the PRO model is defined formally and it is systematically compared with a selection of other models, including BSP and CGM. The paper also presents PRO-algorithms for matrix multiplication and one-to-all broadcast to illustrate how the model is used in algorithm design and analysis.

6 Conclusion

The first algorithm introduced in Paper I, and later applied in Paper III, can be seen as a parallelization technique of general interest. The technique consists of (i) breaking up a given problem into p, not necessarily independent, subproblems of almost equal sizes, (ii) solving the p subproblems concurrently using p processors, and (iii) resolving any inconsistencies that may result due to the interdependence among the p subproblems sequentially. The technique assumes that resolving an inconsistency can be done locally. The success of the technique depends on the overall time spent on detecting and resolving inconsistencies. Papers I and III demonstrate the effectiveness of the technique in developing shared-address-space parallel algorithms for various coloring problems. In these papers, we also showed different ways, including randomization, by which the technique can be enhanced in the context of coloring problems. In Paper IV we extended the basic idea behind the technique to develop an efficient CGM-algorithm for standard graph coloring. This algorithm suggests a way for extending the parallelization technique mentioned earlier. Specifically, if the inconsistencies that could result from (ii) can be determined a priori, then (iii) can be replaced by a recursive application of the technique.

In Paper II we revisited the role of graph coloring in modeling matrix partitioning problems that arise in numerical estimation of sparse Jacobian and Hessian matrices. We used a unified graph theoretic framework for dealing with the matrix problems in a flexible manner. As a result, we developed several simple and effective sequential heuristic algorithms. In Paper III, some of these algorithms were parallelized using the shared address space programming model. The work in Papers II and III can serve as basis for further design and implementation of parallel algorithms for the various coloring problems.

In Paper V we proposed PRO as a candidate model for the design and analysis of efficient, scalable and portable parallel algorithms. Verifying its utility needs further exploration, both theoretical and experimental.

For more specific directions for further work on each of the topics addressed in this thesis, the reader is referred to the concluding remarks given at the end of each paper.

References

[1] S. G. Akl. *Parallel Computation.* Prentice Hall, New Jersey, USA, 1997.

[2] J. R. Allwright, R. Bordawekar, P. D. Coddington, K. Dincer, and C. L. Martin. A comparison of parallel graph coloring algorithms. Technical Report Tech. Rep. SCCS-666, Northeast Parallel Architecture Center, Syracuse University, 1995.

[3] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation.* Springer, Berlin, Germany, 1999.

[4] E. Caceres, F. Dehne, A. Ferreira, P. Locchini, I. Rieping, A. Roncato, N. Santoro, and S. W. Song. Efficient parallel graph algorithms for coarse grained multicomputers and BSP. In *The 24th International*

Colloquium on Automata Languages and Programming, volume 1256 of *LNCS*, pages 390–400. Springer Verlag, 1997.

[5] T. F. Coleman and J. Cai. The cyclic coloring problem and estimation of sparse Hessian matrices. *SIAM J. Alg. Disc. Meth.*, 7(2):221–235, April 1986.

[6] T. F. Coleman and J. J. Moré. Estimation of sparse Jacobian matrices and graph coloring problems. *SIAM J. Numer. Anal.*, 20(1):187–209, February 1983.

[7] T. F. Coleman and J. J. Moré. Estimation of sparse Hessian matrices and graph coloring problems. *Math. Program.*, 28:243–270, 1984.

[8] T. F. Coleman and A. Verma. The efficient computation of sparse Jacobian matrices using automatic differentiation. *SIAM J. Sci. Comput.*, 19(4):1210–1233, July 1998.

[9] D. E. Culler, J. P. Singh, and A. Gupta. *Parallel Computer Architecture*. Morgan Kaufmann Publishers, San Francisco, California, 1999.

[10] A. R. Curtis, M. J. D. Powell, and J. K. Reid. On the estimation of sparse Jacobian matrices. *J. Inst. Math. Appl.*, 13:117–119, 1974.

[11] F. Dehne. Coarse grained parallel algorithms. *Algorithmica Special Issue on "Coarse grained parallel algorithms"*, 24(3/4):173–176, 1999.

[12] F. Dehne, A. Fabri, and A. Rau-Chaplin. Scalable parallel computational geometry for coarse grained multicomputers. *International Journal on Computational Geometry*, 6(3):379–400, 1996.

[13] R. Greenlaw, H.J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press, New York, USA, 1995.

[14] S. Hossain and T. Steihaug. Computing a sparse Jacobian matrix by rows and columns. *Optimization Methods and Software*, 10:33–48, 1998.

[15] S. Hossain and T. Steihaug. Reducing the number of AD passes for computing a sparse Jacobian matrix. In G. Corliss, C. Faure, A. Griewank, L. Hascoët, and U. Naumann, editors, *Authomatic Differentiation of Algorithms: From Simulation to Optimization*, pages 263 – 270. Springer, 2002.

[16] J. Jájá. *An Introduction to Parallel Algorithms*. Addison-Wesley, 1992.

[17] M. T. Jones and P. E. Plassmann. A parallel graph coloring heuristic. *SIAM J. Sci. Comput.*, 14(3):654–669, May 1993.

[18] H. F. Jordan and G. Alaghband. *Fundamentals of Parallel Processing.* Prentice Hall, New Jersey, USA, 2003.

[19] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing.* The Benjamin/Cummings Publishing Company, Inc., California, 1994.

[20] S. T. McCormick. Optimal approximation of sparse Hessians and its equivalence to a graph coloring problem. *Math. Program.*, 26:153–171, 1983.

[21] MPI. Message-passing interface standard. *http://www.mpi-forum.org/.*

[22] G. N. Newsam and J. D. Ramsdell. Estimation of sparse Jacobian matrices. *SIAM J. Alg. Disc. Meth.*, 4:404–418, 1983.

[23] OpenMP. A proposed industry standard API for shared memory programming. *http://www.openmp.org/.*

[24] Parallab. High Performance Computing Laboratory. *http://www.parallab.uib.no/.*

[25] M. J. D. Powell and PH. L. Toint. On the estimation of sparse Hessian matrices. *SIAM J. Numer. Anal.*, 16(6):1060–1074, December 1979.

[26] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.