International Journal of Technology Research and Management
ISSN (Online): 2348-9006
Vol 7 Issue 4 April 2020

A Tree Based Register Allocation Algorithm Using Graph Coloring Approach

Anju Dave[1], Dr. Deepika Pathak[2]
Asst. Prof., M.K.H.S. Gujarati Girls College, Maharani Road, Indore(M.P.) 452001, India[1]
Vice Chancellor Dr. A.P.J. Abdul Kalam University, Indore (M.P.) – 452016, India[2]
anjusumitbhatt@gmail.com[1], deepikapathak23@gmail.com[2]

Abstract: *Code generation is an important part of compiler design. This phase takes optimized Intermediate Representations (IR) code from IR optimizer. The main goal of this phase is to select appropriate machine instruction from all IR instructions and allocate the finite machine resources like register allocation, cache memory etc. In this paper a tree based register allocation technique is proposed. This proposed algorithm is inspired by graph coloring algorithm. Proposed algorithm gives good performance in high amount variables program.*

Keywords: *Register allocation, Graph coloring, Compiler, Algorithm, Code Optimization, Time Complexity, Register Interference Gragp(RIG).*

1. INTRODUCTION

Memory hierarchy is used in computer system to manage the memory. Every part of computer plays their roll to execute a program in optimized way. Programs are written by programmer to run in main memory (RAM) and stored in disk (Secondary storage). Programmers are responsible for disk and main memory data movement. Hardware is responsible for data movement between main memory and cache memory. Lastly compilers are responsible for data movement between main memory and registers. Registers are the very fast and limited memory. They are integrated with processor and used to store the values of program variables while execution of program. It is costly memory so that it is limited. It is not possible every time to allocate all variables to the registers. Compiler uses register allocation algorithms to allocate registers to the program variables [1] [2].

1.1 Register Allocation:

Most of the high level programming language is machine independent. Programmer need not to vary about hardware during programming. High level programming languages do not pay attention on memory hierarchy and memory management. There are certain regions behind doing this, like versatility of hardware configuration, speed of memory, size of memory, generalization of programs etc. All these details are taken by compiler which translates the high level code into machine level codes. Compilers are design and implemented in such a way that it take cares of all hardware and memory available. On the same, CPU registers allocation is also taken care by compiler. Most of the compilers are designed and implemented in 6 phases, i.e. Lexical analyzer, Syntax analyzer, Semantic analyzer, Intermediate code generation, Code optimizer, and Target code generator. These phases orientation is shown in figure 1.

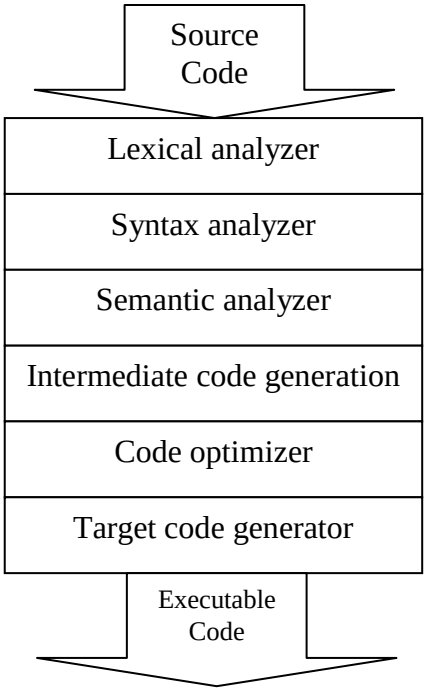

Figure 1: Compiler Structure and Phases

Broadly these phases are divided into three categories front end, optimizer and backend [3]. Front end is machine independent. It translates high level language code into intermediate code. Compile time error checking is done by front end. Front end is always language dependent. First four phases of compiler (Lexical analyzer, Syntax analyzer, Semantic analyzer and Intermediate code generation) comes in this category. Code optimizer phase of compiler is the second section (optimizer). It performs global optimization in different phases. Optimizer transfers the intermediate code to increase the performance of code. Optimizer is independent from language as well as machine. It includes removal of common sub-expression, identified and elimination of unreachable code, loop invariant code motion and strength reduction etc. Target code generator phase of compiler is known as backend section of compiler. Backend translate globally optimized code into machine dependent code. Back end perform many operations including instruction selection, scheduling of instruction and register allocation. Back end in totally language independent phase.

Register allocation algorithm works in target code generator phase on optimized code received from code optimizer. Code is optimized at program level by code optimizer phase but register allocation of target code generator phase optimizes the code at hardware level. This phase do this by assigning and utilizing proper hardware and memory to the program instructions. Register allocation must be in such a way that maximum variable is assigned to minimum available registers, so that program execution and memory operations get more faster.

Further, this article is structured as follows. In Section 2, some existing graph coloring approaches used for register allocation and other applications are reviewed. In Section 3, some basic terminology used in register allocation using graph coloring is explored. In Section 4, a small description of tree based graph coloring approach is included. In Section 5, all the major important steps and flow of proposed register allocation algorithm is explained. In Section 6, entire approach is explained through an example. Section 7 summarized this research work.

2. RELATED WORK

G.J. Chaitin et al. [5] (1981) first time introduced register allocation via graph coloring. They implemented and tested register allocation phase on PL/I compiler for the IBM system/370. 17 registers are maped to unlimited number of symbolic registers (variables) of intermediate code. They have successfully implemented this important optimization phase using graph coloring approach. There was problem in register allocation when number of colors equal or greater to the number of machine registers. This problem is solved using introducing spill and reloads register concepts in algorithm [4].

Keith D. Cooper and Anshuman Dasgupta [7] (2006) explore a global register allocation graph coloring for runtime compilation. Their technique is better than linear-scan [2] and Chaitin-Briggs [4][5] allocators for most of the runtime environment.

David Koes and Seth Copen Goldstein [8] (2006) proposed an integer linear program (ILP) based graph coloring algorithm to solve register allocation problem. In this algorithm author improved the traditional graph coloring algorithm by improving objective function to reduce spill cost. Cascading was modeled using ILP. Selection of coloring was also in a particular order to improve the selection phase and register assignment was also in a particular order to improve the register allocation.

Sevin Shamizi and Shahgiar Latfi [9] in (2011) proposed register allocation optimization technique based on graph coloring evolutionary algorithm. In this algorithm selection of spilling variable and graph coloring take place simultaneously.

By the time there was many other graph coloring algorithms (Sequential Greedy Algorithm [16], First Fit [17], Largest-Degree-First-Ordering [18], Incidence-Degree-Ordering [19], Smallest-Degree-Last–Ordering [20] and Saturation-Degree-Ordering [21]. There are many parallel graph coloring approaches such as: Parallel Maximal Independent set [22], Jones and Plassmann [23], the Largest-Degree-First and Smallest-Degree-Last [16], Graph Partitioning Approach, Block Partitioning Approach [24] etc) were proposed and used to solve the register allocation as well as other optimization problems like timetabling and scheduling, radio frequency assignment [14], and satellite range scheduling [15].

3. REGISTER ALLOCATION USING GRAPH COLORING

This section included some important terminologies related to register allocation algorithm using graph coloring (GC) and general procedure of register allocation using GC is also included in this section.

3.1 Basic Terminologies:

i. Temporary variables: - These variables are used in programs to perform any mathematical and logical operation in programs. Generally these variables get space in main memory. But to speed up the variable access, CPU registers are allocated to those variables.
ii. Variable Live Range: - Every variable is considered in live range at point in program where variable is lives.
iii. Variable Live Interval: - Compiler convert the source code into intermediate representation code (IR Code). Live range for a variable is smallest sub-range of the IR Code.
iv. Register Interference Graph (RIG):- It is a undirected graph in which all vertices represents the variables and connecting edges represents live status of variables at the same program point.

4. TREE BASED GRAPH COLORING ALGORITHM (GCA) [11]

This section exploring graph coloring algorithm used to implement proposed register allocation algorithm.

A tree data structure based GCA is used to implement register allocation algorithm. The entire process of graph coloring is divided into three basic steps as shown in figure 2.

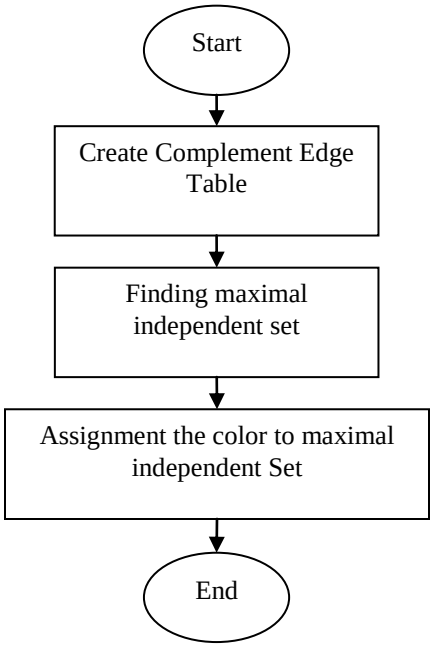

Figure 2: Basic Steps of Tree Based GCA

Solving the graph coloring problem is NP-hard [10] and it is hard to determine time complexity of algorithm. But by the experimental results and hypothesis the worst case complexity of selected algorithm is O(2n-1) [11]. Time complexity to find maximal independent set is O ($2^{\log n}$) [12]. Selected algorithm gives high performance for high degree graphs i.e. this algorithm is more suitable for the programs where number of variables are high and live intervals of variables in program at same time are also high.

5. PROPOSED REGISTER ALLOCATION ALGORITHM

This section completely devoted to proposed register allocation algorithm, its approach, implementation and performance.

Proposed register allocation technique is based on graph coloring approach. Integrated graph coloring algorithm is tree based graph coloring. Broadly entire process is divided into six steps.

Step 1: Identification of variables:

Register allocation is all about allocating registers to variable so according to the programming language, firstly

identification of all variables need to complete. In this step all the declared variables are identified and numbered.

Step 2: Evaluation of live interval of variables:

Live interval of variable is life time of variable, for that time duration variable is required memory to store and process the data. Using liveliness analysis [13], live interval of each identified variables need to calculate.

Step 3: Register Interference Graph generation:

An undirected graph has to be generated in which each vertex represents a single variable of program and all vertices are interconnected, if variables corresponding to the vertices has the common live interval.

Step 4: Applying tree based graph coloring algorithm on interference graph:

Input interference graph to the graph coloring algorithm. The internal process of graph coloring is divided in three steps as shown in figure 2.

Step 5: Sort independent set:

After applying graph coloring algorithm, we will get independent sets. Each set of variables can be assign to a single register. All the independent sets need to be arranged in sorted order (deciding order) according sum of variables live interval time in that set.

Step 6: Register assignment:

Now sorted list of independent sets can be assign to available registers. For example if processer has 4 register than first four independent sets are assigned to register (each set to single register).

Entire process flow is shown in figure 3.

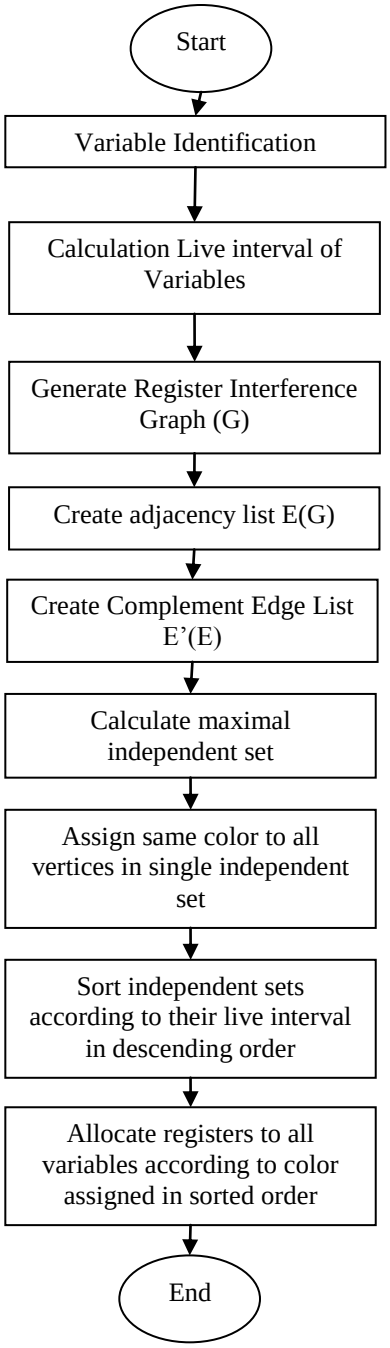

Figure 3: Process Follow Register Allocation Algorithm

6. ILLUSTRATION BY EXAMPLE

Let us explore the proposed approach using an example. Consider a code segment as shown in figure 4. This code segment consist seven different variables named a, b, c, d, e, f and g. This example shows how we can assign registers to those variables in such a way that maximum variables get registers and increase the performance program execution process.

```
    e = d + a
    f = b + c
    f = f + b
    IfZ e Goto _L0
    d = e + f
    Goto _L1;
_L0:
    d = e - f
_L1:
    g = d
```

Figure 4: Sample Code Segment

Figure 4: Energy Consumption Comparisons

6.1 Liveliness Analysis:

Figure 5 explored the variables (a, b, c, d, e, f and g) liveliness. Pare of braces before and after instructions containing variables shows liveliness status of variables. The green vertical bars in figure 6 shows the live interval of each variable in code segments.

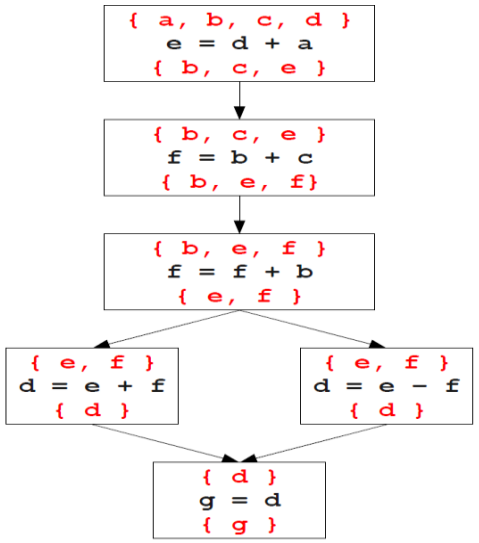

Figure 5: Liveliness Analysis

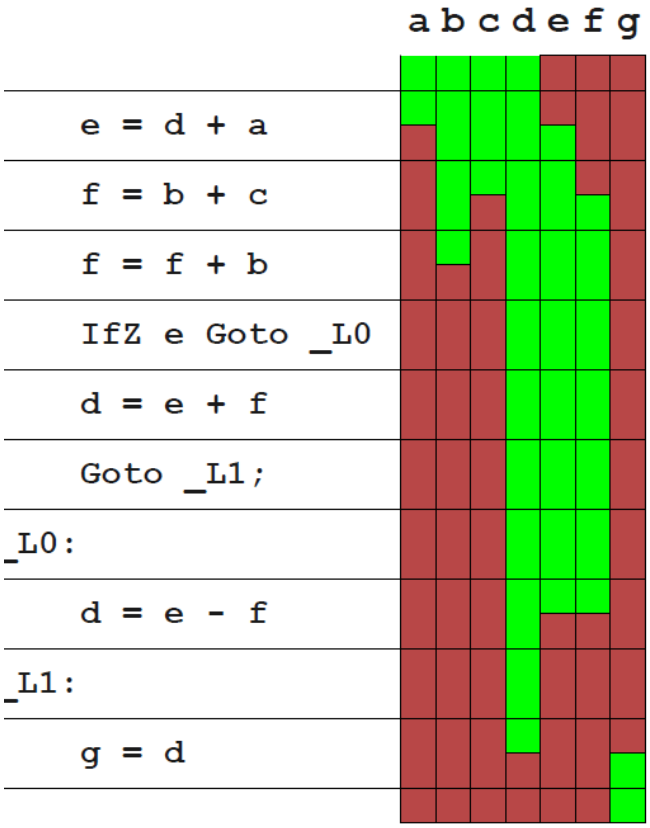

Figure 6: Live Interval of Variables

6.2 Register Interference Graph (RIG):

An interference graph G= (V, E) is generated using live interval of variables in such a way, that every edge (e ∈ E) connects vertices (v1, v2 ∈ V) has common live interval. Figure 7 shows the RIG consist of 7 vertices (a, b, c, d, e, f and g) and 11 connecting edges.

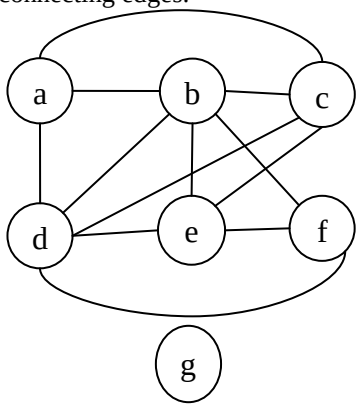

Figure 7: Register Interference Graph (RIG)

6.3 Adjacency List Creation:

Table 6.1 shows the adjacency list of Graph G=(V,E) shown in figure 7. Table contains the three columns (Edge Number, Vertex 1 and Vertex 2).

Table 6.1: Adjacency List

Edge Number	Vertex 1	Vertex 2
1	a	b
2	a	c
3	a	d
4	b	c
5	b	d
6	b	e
7	b	f
8	c	d
9	c	e
10	d	e
11	d	f
12	e	f

6.4 Complement Adjacency List:

The complement of graph G = (V, E) is a graph G'= (V, E'), where E' = {(i, j) | i, j ∈ V, i ≠ j and (i, j) ∉ E}. Table 6.2 shows the complement adjacency list of graph G= (V, E). Complement adjacency list contain all the pare of vertices where connecting edge is not present in graph.

Table 6.2: Complement Adjacency List

Edge Number	Vertex 1	Vertex 2
1	a	f
2	a	f
3	a	g
4	b	g
5	c	f
6	c	g
7	d	g
8	e	g
9	f	g

6.5 Tree Exploring:

It is multi step process. In this multiple trees are explored and each tree gives single independent set. Each set of vertices representing variables is single independent set can be assigned to single register. Following steps executed until all the vertices of graph G=(E,V) explored in tree form:

(1) Select any random vertex which is not included in any maximal independent set and make it root of tree.
(2) Explore root vertex as follows: Select all vertices from complement adjacency list which are adjacent to root and make all those vertices as a child node of that root vertex.
(3) Same step need to repeat for every child node of tree. Select only those vertices which are in against being explored node in tree and it must be the sibling of explored node in tree.
(4) Select the first longest path in tree as maximum independent set.
(5) Remove the all vertices from set of vertices and graph to update adjacency list and complement adjacency list.
(6) Repeat step 1 to 5 through updated complement adjacency list utile all the vertices are included in any independent set.

Using this set of rules trees are explored as shown in figure 8, 9, 10 and 11.

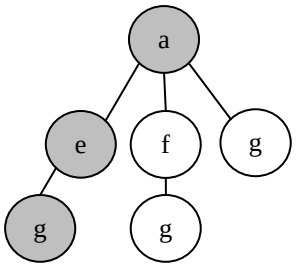

Figure 8: Explored Tree for First Independent Set {a, e, g}

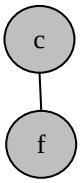

Figure 9: Explored Tree for Second Independent Set {c, f }

Figure 10: Explored Tree for third Independent Set {b}

Figure 11: Explored Tree for fourth Independent Set {d}

6.6 Sorting of MIS:

If number of registers is equal or more then number of MIS each MIS variables can be allocated to single register. But if number of registers is less then MIS in that case we have select register allocation which is most suitable and efficient. Large live interval time set show that it is used most frequently and for long time in the program. So all the independent sets are arranged in sorted order according to total live interval time of individual MIS all variables. Table 6.3 shows the MIS with their live interval time in sorted order.

Table 6.3: Sorted Live Interval Time of Individual MIS

S.No.	MIS	Live Interval Time (Clock Cycle)
1	{d}	11
2	{a,e,g}	9
3	{c,f}	8
4	{b}	3

Register should be allocated according table 6.3 i.e. if processor has 4 or more register than we can assign all MIS to the registers. But if processor has 3 or less registers for use than MISs are allotted according to sequence shown in table 6.3 (higher to low live interval time).

7. CONCLUSION

Proposed register allocation technique is design using tree based graph coloring approach. This approach can be used in case of un-sufficient register availability. It gives maximal utilization of CPU registers and improves the execution time by allotting registers to those variables which has long live interval time. Used graph coloring approach is good for coloring high degree graph, so it gives good performance in the program execution where number of variables is high and most of the variables are live at same time interval, Due to dynamic nature of algorithm this approach can be used to develop any type of register allocation phase of compiler.

REFERENCES

[1] Josef Eisl , Stefan Marr , Thomas Würthinger and Hanspeter Mössenböck, “Trace Register Allocation Policies”, In Proc. ManLang 2017, September 27–29, 2017, Prague, Czech Republic. https://doi.org/10.1145/3132190.3132209.

[2] Massimiliano Poletto and Vivek Sarkar, “Linear Scan Register Allocation”, ACM Transactions on Programming Languages and Systems, Vol. 21, No. 5, September 1999, Pages 895-913.

[3] Preston Briggs, “Register Allocation via Graph coloring”, PhD Thesis RICE University Houston Texas, April 1992.

[4] Gregory J. Chaitin, “Register allocation and spilling via graph coloring”, SIGPLAN notices, 17 (6): 98-105, June 1982. In Proc of the ACM SIGPLAN’ 82 Symposium of Compiler Construction.

[5] G.J.. Chaitin, M.A. Auslander, A.K. Chandra, J. Coeke, M.E. Hopkins and P.W. Markstein, "Register allocation via coloring", Computer Languages 6, 1981, pp. 47-57

[6] Burke E. K., McCollum B., Meisels A., Petrovic S. and Qu R., “A graph-based hyper heuristic for timetabling problems”. European Journal of Operational Research, 176, 2007, page no. 177–192.

[7] Keith D. Cooper and Anshuman Dasgupta, “Tailoring Graph-coloring Register Allocation For Runtime Compilation”, International

Symposium on Code Generation and Optimization (CGO'06), 26-29 March 2006, DOI: 10.1109/CGO.2006.35

[8] David Koes and Seth Copen Goldstein, "An Analysis of Graph Coloring Register Allocation", Carnegie Mellon University Technical Report No. CMU-CS-06-111, March 1990.

[9] Sevin Shamizi and Shahriar Lotfi, "Register Allocation via Graph Coloring Using an Evolutionary Algorithm", B.K. Panigrahi et al. (Eds.): SEMCCO 2011, Part II, LNCS 7077, 2011, Page no. 1–8.

[10] Garey M. R., and Johnson D. S. "Computers and intractability: A guide to the theory of NPcompleteness". San Francisco: W.H. Freeman and Company, 1979.

[11] Patidar H., Chakrabarti P., "A Tree-Based Graph Coloring Algorithm Using Independent Set". In: Panigrahi C., Pujari A., Misra S., Pati B., Li KC. (eds) Progress in Advanced Computing and Intelligent Engineering. Advances in Intelligent Systems and Computing, vol 714. Springer, Singapore, 2019.

[12] Prakash C. Sharma and Narendra S. Chaudhari," A Tree Based Novel Approach for Graph Coloring Problem Using Maximal Independent Set", Wireless Personal Communications, September 2019, https://doi.org/10.1007/s11277-019-06778-0.

[13] Frank Pfenning, Andr´e Platzer and Rob Simmons, "Liveness Analysis", Lecture Notes 15-411: Compiler Design Lecture 4, September 2014.

[14] Smith D. H., Hurley S., and Thiel S. U., Improving heuristics for the frequency assignment problem. European Journal of Operational Research, 107(1), 1998, page no. 76–86.

[15] Zufferey N., Amstutz P., and Giaccari P. "Graph colouring approaches for a satellite range scheduling problem", Journal of Scheduling, 11(4), 2008, page no. 263–277.

[16] Allwright JR, Bordawekar R, Coddington PD, Dincer K and Martin CL. "A comparison of parallel graph coloring algorithms" ,Technical Report SCCS-666, Northeast Parallel Architecture Center, Syracuse University, 1995.

[17] Dr. Hussein Al-Omari and Khair Eddin Sabri "New Graph Coloring Algorithms", American Journal of Mathematics and Statistics 2 (4), 2006, page no. 739-741.

[18] C. Avanthay, A. Hertz, N. Zufferey "A variable neighborhood search for graph coloring",European Journal of Operational Research 151 (2), 2003, page no. 379–388

[19] Burke EK, McCollum B, Meisels A, Petrovic S and Qu R. "A graph-based hyper heuristic for timetabling problems", European Journal of Operational Research 2007; 176, page no. 177-192.

[20] D. W. Matula and L. L. Beck, "Smallest-last ordering and clustering and graph coloring algorithms", JACM, 1983.

[21] E. Falkenauer, "A hybrid grouping genetic algorithm for bin packing", Journal of Heuristics 2 (1),1996, page no. 5–30.

[22] M. Luby, "A simple parallel algorithm for the maximal independent set problem", SIAM Journal on Computing 4 (1986) 1036.

[23] M. T. Jones and P. E. Plassmann, "A Parallel Graph Coloring Heuristic" SIAM Journal of Scientic Computing 14 (1993) 654.

[24] Assefaw Hadish Gebremedhin and Fredrik Manne "Scalable parallel graph coloring Algorithms", CONCURRENCY: PRACTICE AND EXPERIENCE Concurrency: Pract. Exper. 2000; 12:1131–1146