A Design for Node Coloring and 1-fair Alternator on de Bruijn Networks

Jyh-Wen Mao
Department of Applied Mathematics
National Sun Yat-sen University, Kaohsiung, Taiwan
maojw@math.nsysu.edu.tw

Chang-Biau Yang
Department of Computer Science and Engineering
National Sun Yat-sen University, Kaohsiung, Taiwan
cbyang@cse.nsysu.edu.tw

Abstract

In this paper, we study the coloring problem for the undirected binary de Bruijn interconnection network. The coloring scheme is simple and fast. We propose the coloring algorithm by using the pseudo shortest-path spanning tree rooted at $(0\cdots00)$. *Each processor can find its color number by its own identity. Then, based on our coloring algorithm, we propose a 1-fair alternator. Our design is optimal. In our design, each processor can execute the critical step once in every 3 steps.*

Key words: alternator; coloring; de Bruijn graph; phase synchronization

1 Introduction

A k-coloring of an undirected graph $G(V,E)$ is a function $C: V \rightarrow \{1,2,\cdots,k\}$ such that $C(u) \neq C(v)$ for every edge $(u,v) \in E$. In other words, each of $1,2,\cdots,k$ represents one of the k colors and adjacent nodes have different colors [3, 6]. The graph coloring problem is to determine the minimum number of colors needed to color a given graph.

The problem of coloring graphs has been studied intensively for many years. Unfortunately, the coloring problem for a general graph with the minimum number of colors is NP-complete [7]. For example, the problem for determining whether a graph is 3-colorable is NP-complete. It may be difficult to decide if a graph can be colored with a given number of colors. However, it is easy to check where a suggested coloring is valid. The computation to find an optimal solution for coloring problem is very expensive. Fortunately, we may try to look for additional information about the coloring problem at hand. We may find that the graph has some special properties. For examples, k colors are needed for a k-clique, and the bipartite graphs are 2-colorable.

The problem of coloring graphs can be applied to solve many problems. It is not hard to see that the traffic light design problem is one of coloring the graph of incompatible turns using as few colors as possible [1]. In a fault-tolerant processor system, node coloring is used to allocate additional bypass links [18]. In concurrent system, the critical step problem of alternator can be transformed into the coloring problem [12].

Recently, progress in computer technology enables the construction of massive parallel computers providing large number of processors interconnected via high throughput networks [4, 5, 14–16, 18]. The de Bruijn graph has received much of attention by researchers as a graph model for networks [4, 5, 10, 15, 16]. The node set of the undirected binary de Bruijn graph $dB(2,m)$ consists of all m-bit binary numbers, which has 2^m nodes. Node $(v_m\cdots v_2v_1)$ is connected with nodes $(v_{m-1}\cdots v_10)$, $(v_{m-1}\cdots v_11)$, $(0v_m\cdots v_2)$ and $(1v_m\cdots v_2)$. The graph $dB(2,m)$, a shift-and-replace graph, has maximum degree 4, minimum degree 2, diameter m and admits simple broadcasting and routing algorithms [4, 10, 15, 16].

In this paper, we shall propose a simple and quick algorithm to solve the node coloring problem on the undirected binary de Bruijn graphs. In our algorithm, the number of colors used is 3, and it is an optimal design. By adopting this algorithm, we also construct an optimal 1-fair alternator [8, 12] on binary de Bruijn networks.

The rest of this paper is organized as follows. In Section 2, we propose the node coloring algorithm for binary de Bruijn graphs. In Section 3, the design of 1-fair alternator is given. Finally, some conclusions will be given in Section 4.

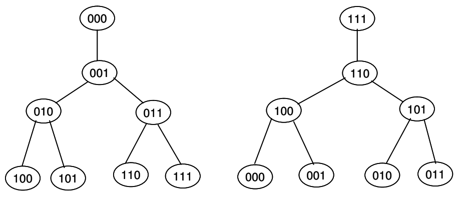

Figure 1: The shortest-path spanning tree rooted at (000) and (111) in $dB(2,3)$

2 The design of node coloring for binary de Bruijn graphs

It is obvious that every connected graph contains a spanning tree. The degree of the internal nodes of a binary tree is 3, and it can be colored by using 2 colors easily. The degree of most nodes of a binary de Bruijn graph $dB(2,m)$ is 4. Thus, our design uses the shortest-path spanning tree (or breadth first search tree). Only links $(v_m \cdots v_2 v_1)(v_{m-1} \cdots v_1 0)$ and $(v_m \cdots v_2 v_1)(v_{m-1} \cdots v_1 1)$ are considered to construct the spanning tree. We do not consider the loops. Figure 1 shows the two shortest-path spanning trees rooted at (000) and (111) in $dB(2,3)$, respectively.

Some properties of the shortest-path spanning tree in $dB(2,m)$ are useful in our design. The leaves of the left tree in Figure 1 are the root and internal nodes of the right spanning tree, vice versa. The internal nodes of the same depth are not adjacent to each other. Furthermore, each of these root and internal nodes is connected to a leaf.

Let $v = (v_m \cdots v_2 v_1)$ be a node of $dB(2,m)$ and C be the color assignment function such that $C(v)$ is the color assigned to node v. For simplifying description, we use P_0 and P_1 as the leftmost bit position which is 0 and 1 in $(v_{m-1} \cdots v_2 v_1)$, respectively. For example, if v_j is the leftmost bit position when its value is 1, then the value of P_1 is j. Initially, the values of both P_0 and P_1 are set to 0. Our node coloring algorithm is as follows.

Algorithm 1:

```
Step 1: If $m$ is even then $c4 = c1$ else $c4 = c2$

Step 2: For each node $v$,
        If $v_m = 0$ then find $P_1$ else find $P_0$

Step 3: For each node $v$,
        If $v_m = 0$ then if $P_1$ is even then $C(v) = c1$
                                          else $C(v) = c2$
             else if $(m - P_0)$ is odd then $C(v) = c3$
                                          else $C(v) = c4$
```

Theorem 1 *The binary de Bruijn graph is 3-colorable.*

Proof: Since the value of $c4$ is assigned to be either $c1$ or $c2$, it is sufficient to show the number of colors required to color a binary de Bruijn graph is 3 provided that Algorithm 1 is valid.

In $dB(2,m)$, node $(v_m \cdots v_2 v_1)$ is adjacent to nodes $(v_{m-1} \cdots v_1 0)$, $(v_{m-1} \cdots v_1 1)$, $(0 v_m \cdots v_1)$ and $(1 v_m \cdots v_1)$. Let N_1, N_2, N_3 and N_4 denote nodes $(v_{m-1} \cdots v_1 0)$, $(v_{m-1} \cdots v_1 1)$, $(0 v_m \cdots v_1)$ and $(1 v_m \cdots v_1)$, which are the neighbors of node $(v_m \cdots v_2 v_1)$, respectively. It is clear, by the bit presentation of nodes in $dB(2,m)$, the color assigned to a node is decided by one of the following 4 cases:

1. $v_m = 0$ and P_1 is even.
2. $v_m = 0$ and P_1 is odd.
3. $v_m = 1$ and $m - P_0$ is odd.
4. $v_m = 1$ and $m - P_0$ is even.

To avoid proliferation of the discussion for the exceptional nodes, that is $v = (1 \cdots 11)$ or $v = (0 \cdots 00)$. First of all, we will discuss the color assignment for nodes $(0 \cdots 00)$, $(1 \cdots 11)$ and their adjacent nodes. For node $(0 \cdots 00)$, its neighbors are nodes $(0 \cdots 01)$ and $(10 \cdots 0)$. It is obvious that $C(0 \cdots 00)$, $C(0 \cdots 01)$ and $C(10 \cdots 0)$ are $c1$, $c2$, and $c3$, respectively. Node $(0 \cdots 00)$ and its adjacent nodes have different colors. For node $(1 \cdots 11)$, its neighbors are nodes $(1 \cdots 10)$ and $(01 \cdots 1)$. If m is even, i.e. $c4 = c1$, it is obvious that $C(1 \cdots 11)$, $C(1 \cdots 10)$ and $C(01 \cdots 1)$ are $c4$, $c3$, and $c2$, respectively. If m is odd, i.e. $c4 = c2$, it is obvious that $C(1 \cdots 11)$, $C(1 \cdots 10)$ and $C(01 \cdots 1)$ are $c3$, $c4$, and $c1$, respectively. Thus, node $(1 \cdots 11)$ and its adjacent nodes have different colors.

In the following proof, we will discuss the above four cases except nodes $(1 \cdots 11)$ and $(0 \cdots 00)$. In case 1, $C(v) = c1$. The colors assigned to the neighbors of v can be stated as follows:

1. If $v_{m-1} = 0$, then P_1 of N_1 and P_1 of N_2 are odd. Hence, both $C(N_1)$ and $C(N_2)$ are $c2$.
2. If $v_{m-1} = 1$, it is clear that m is odd and $c4 = c2$. Thus, both $C(N_1)$ and $C(N_2)$ are either $c3$ or $c4$ except $v = (v_m \cdots v_2 v_1) = (01 \cdots 1)$. If $v = (01 \cdots 1)$, then $N_1 = (1 \cdots 10)$ and $N_2 = (1 \cdots 11)$. Thus, $C(N_1)$ and $C(N_2)$ are $c4$ and $c3$, respectively.
3. It is obvious that $C(N_3)$ is $c2$.
4. Since $v_m = 0$, P_0 of N_4 is $m - 1$. Thus, $C(N_4)$ is $c3$ because of $m - P_0 = 1$ which is an odd number.

In case 2, $c2$ is assigned to node v.

1. If $v_{m-1} = 0$, then P_1 of N_1 and P_1 of N_2 are even. It is clear that both $C(N_1)$ and $C(N_2)$ are $c1$.

2. If $v_{m-1} = 1$, it is clear that m is even and $c4 = c1$. Thus, both $C(N_1)$ and $C(N_2)$ are either $c3$ or $c4$ except $v = (v_m \cdots v_2 v_1) = (01 \cdots 1)$. If $v = (01 \cdots 1)$, then $N_1 = (1 \cdots 10)$ and $N_2 = (1 \cdots 11)$. Thus, $C(N_1)$ and $C(N_2)$ are $c3$ and $c4$, respectively.

3. It is clear that $C(N_3)$ is $c1$.

4. It is similar to (4) in Case 1. $C(N_4)$ is $c3$.

In case 3, $c3$ is assigned to node v.

1. If $v_{m-1} = 0$, it is clear that both $C(N_1)$ and $C(N_2)$ are either $c1$ or $c2$ except $v = (v_m \cdots v_2 v_1) = (10 \cdots 0)$. If $v = (10 \cdots 0)$, then $N_1 = (0 \cdots 00)$ and $N_2 = (0 \cdots 01)$. Thus, $C(N_1)$ and $C(N_2)$ are $c1$ and $c2$, respectively.

2. If $v_{m-1} = 1$, it implies $m - P_0$ of N_1 and N_2 are even. Thus, both $C(N_1)$ and $C(N_2)$ are $c4$.

3. It is clear that $C(N_3)$ is either $c1$ or $c2$.

4. It is clear that $m - P_0$ of N_4 is even. $c4$ is assigned to node N_4.

In case 4, $C(v) = c4$.

1. Since $m - P_0$ of v is even, it implies $v_{m-1} = 1$. It is clear that $m - P_0$ of both N_1 and N_2 are odd. Thus, both $C(N_1)$ and $C(N_2)$ are c3.

2. It is clear that $C(N_3)$ is $c1$ if $c4 = c2$ or $C(N_3)$ is $c2$ if $c4 = c1$.

3. It is clear that $m - P_0$ of N_4 is odd. $C(N_4)$ is $c3$.

As a result, we see that node v and its neighbors have different colors. It is clear that, based upon the above results, number of colors needed to color a binary de Bruijn graph is 3 and each pair of adjacent nodes have different colors while Algorithm 1 is executed. This completes the proof. ∎

Figures 2 shows $dB(2,4)$ and the color assignment after Algorithm 1 is executed. In Figures 2, the numbers 1, 2 and 3 denote three different colors. It is obvious that the minimum number of colors used for coloring nodes $(0000), (1000)$ and (0001) in $dB(2,4)$ is 3. The $dB(2,m)$ graph can not be 2-colored. Hence, Algorithm 1 is optimal.

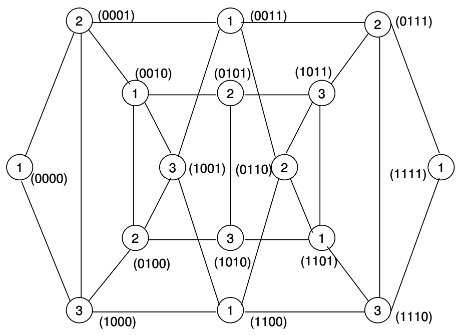

Figure 2: The color assignment for $dB(2,4)$

3 The optimal design for constructing 1-fair alternator

An alternator [8, 12] is consisted of a network of concurrent processors which would like to satisfy the following two conditions. First, no neighbors can execute the critical step while a processor is performing the critical section. Second, each processor performs the critical step infinitely often. A system is 1-fair alternator if the second condition is modified as: 1. Processor v executes two successive critical steps at steps $s1$ and $s2$. 2. All neighbors of processor v execute the critical step at most once between $s1$ and $s2$.

Some research efforts on concurrent system have focused on the design for alternator. In Gouda and Haddix's design [8], each processor is admitted into the critical section once per $2d - 1$ steps, where d is the length of the longest simple cycle of the network. It is obvious that their design is 1-fair, but may not be optimal for all networks. A counter example is given as follows. In an 8-processor binary hypercube, the length of the longest simple cycle is 8. Thus, each one executes the critical step once per 15 steps. But, a hypercube network can be viewed as a bipartite graph. Hence, an optimal design for hypercube would be asked to achieve each processor to execute the critical step once per 2 steps. Huang [12] proposes two optimal designs: one for binary hypercubes, the other for $D \times D$ meshes with odd D.

In this section, we propose a simple approach for constructing optimal 1-fair alternator on binary de Bruijn networks. In our approach, in order to hold the first condition for 1-fair alternator, the processors are divided into several groups such that no pair of processors are adjacent in the same group. This can be done by Algorithm 1. The processors in the same group are assigned the same color. And, the number of groups is 3. The next thing to do is to stabilize phases synchronous [2, 9, 11, 13, 17] where the number of phases is 3. Finally, each processor is allowed to execute the

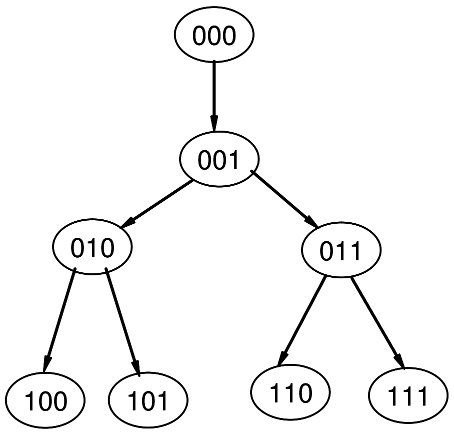

Figure 3: The rooted tree in $dB(2,3)$

critical step only when its color number matches the phase number. The computation steps considered are synchronous: in each computation step, all the processors take one step synchronously. The computation consists of a sequence of computation units, where all processors participate in each unit of computation. We suppose that during one unit of computation, a processor can inspect the state of each processor connected to it by a link and then change its state.

In our design, we assume the identity of each processor is given and fixed, the color assigned to a processor is decided by Algorithm 1. We use 1, 2 and 3 instead of the color number $c1, c2$ and $c3$, respectively. After Algorithm 1 is executed, each processor gets its color. The shortest-path spanning tree rooted at node $(0\cdots 00)$ is used again. For each processor v with identity $(v_m \cdots v_2 v_1)$, $F(v)$ denotes its neighbor with identity $(0v_m \cdots v_2)$, which is the father of v in the tree. Note that $F(0\cdots 00) = (0\cdots 00)$. We suppose that there exists a pointer pointing from $F(v)$ to v, then we get the tree rooted at $(0\cdots 00)$. Figure 3 shows the rooted tree in $dB(2,3)$.

The phase synchronization can be done by letting the processor $(0\cdots 00)$ play the leadership role. When a processor v receives the phase value of $F(v)$, it sends the value to its sons immediately and then evaluates its phase value. Each processor v maintains a phase variable $PHASE(v)$ of values 1 or 2 or 3. Our design is then as follows.

At each step, each processor v executes the following statement:

```
PHASE(v) = PHASE(F(v)) mod 3 + 1
If PHASE(v) = C(v) then execute the critical step
                 else skip
```

In an asynchronous system, each processor executes phases arbitrarily far ahead of other processors. Our design is also valid if the phase synchronization in asynchronous systems provides: 1. No processor executes a phase until all other processors have successfully executed the previous phase. 2. After all processors have successfully executed a phase, each of them eventually execute the next phase.

4 Concluding remarks

This paper presents an optimal design for the coloring problem on binary de Bruijn graphs. A shortest-path spanning tree with initiate node $(0\cdots 00)$ is used to achieve our design. Our algorithm finds that the binary de Bruijn graphs are 3-colorable. In other words, our algorithm is optimal.

The design for constructing an alternator is divided into two major parts: one is the coloring, the other is phase synchronization. Since there exists phase synchronization protocols for general, uniform networks [2, 9], an optimal design can be achieved if the coloring is optimal. The rest is the execution of critical step, and it becomes easily: if the color number matches the phase number then processor v executes the critical step.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structures and Algorithms*. Addison-Wesley Publishing Company, 1983.

[2] A. Arora, S. Dolev, and M. Gouda, “Maintaining digital clocks in step,” *Parallel Processing Letters*, Vol. 1, pp. 11–18, 1991.

[3] S. Baase, *Computer Algorithms: Introduction to Design and Analysis*. Addison-Wesley Publishing Company, 1988.

[4] J. C. Bermond and P. Fraigniaud, “Broadcasting and gossiping in de Bruijn networks,” *SIAM Journal on Computing*, Vol. 23, pp. 212–225, Feb. 1994.

[5] J. C. Bermond, Z. Liu, and M. Syska, “Mean eccentricities of de Bruijn networks,” *Networks*, Vol. 30, pp. 187–203, 1997.

[6] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*. The Macmillan Press Ltd., 1976.

[7] G. Brassard and P. Bratley, *Algorithmics: Theory and Practice*. Prentice-Hall, Inc., 1988.

[8] M. G. Gouda and F. Haddix, "The alternator," *Proc.* 19th *IEEE International Conference on Distributed Computing Systems*, pp. 48–53, 1999.

[9] T. Herman and S. Ghosh, "Stabilizing phase-clocks," *Information Processing Letters*, Vol. 54, pp. 259–265, 1995.

[10] D. F. Hsu and D. S. L. Wei, "Efficient routing and sorting schemes for de Bruijn networks," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, pp. 1157–1170, Nov. 1997.

[11] S. T. Huang, "Leader election in uniform rings," *ACM Trans. Programming Language Systems*, Vol. 15, pp. 563–573, 1993.

[12] S. T. Huang and B. W. Chen, "Optimal 1-fair alternators," *Information Processing Letters*, Vol. 80, pp. 159–163, 2001.

[13] S. Kulkarni and A. Arora, "Multitolerance barrier synchronization," *Information Processing Letters*, Vol. 64, pp. 29–36, 1997.

[14] J. C. Lin and N. C. Hsien, "Reconfiguring binary tree structures in a fault supercube with unbounded expansion," *Parallel Computing*, Vol. 28, pp. 471–483, 2002.

[15] Z. Liu and T. Y. Sung, "Routing and transmitting problems in de Bruijn networks," *IEEE Transactions on Computers*, Vol. 45, pp. 1056–1062, Sep. 1996.

[16] J. W. Mao and C. B. Yang, "Shortest path routing and fault-tolerant routing on de Bruijn networks," *Networks*, Vol. 35, pp. 207–215, 2000.

[17] J. Misra, "Phase synchronization," *Information Processing Letters*, Vol. 38, pp. 101–105, 1991.

[18] N. Tsuda, "Fault-tolerant processor arrays using additional bypass linking allocated by graph-node coloring," *IEEE Transactions on Computers*, Vol. 49, pp. 431–442, May 2000.