LUISS

IRIS - Luiss Guido Carli Research Institutional Repository | Catalogo istituzionale della ricerca Luiss Guido Carli

Conflict-free star-access in parallel memory systems

Post-print version of the following publication: | Versione post-print della seguente pubblicazione:

Original Citation/Citazione:
Sajal K., Das; Finocchi, Irene; Petreschi, Rossella. (2006). Conflict-free star-access in parallel memory systems. JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING, (ISSN: 0743-7315), 66:11, 1431-1441. Doi: 10.1016/j.jpdc.2006.06.004.

Availability/Disponibilità:
This version is available at: 11385/192643 since: 2020-02-15T13:58:31Z - Questa versione è disponibile alla pagina: 11385/192643 dal: 2020-02-15T13:58:31Z

Publisher/Casa editrice:

Published version/Pubblicato:
DOI: https://dx.doi.org/10.1016/j.jpdc.2006.06.004

License/Licenza:
DRM (Digital rights management) non definiti

Availability/Termini d'uso:

The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. Works made available under a Creative Commons license can be used according to the terms and conditions of said license. For all terms of use and more information see the publisher's website. | I termini e le condizioni relativi al riutilizzo della presente versione della pubblicazione sono disciplinati dalla politica editoriale. Le opere messe a disposizione con licenze Creative Commons possono essere utilizzate conformemente ai termini e alle condizioni previste da tali licenze. Per l'insieme delle condizioni di utilizzo e per ulteriori informazioni si rinvia al sito web dell'editore.

This item was downloaded from IRIS Luiss (https://iris.luiss.it/). When citing, please refer to the published version. | Questo documento è stato scaricato da IRIS Luiss (https://iris.luiss.it/). Per la citazione, fare riferimento alla versione pubblicata sul sito dell'editore.

(Article begins on next page | Il contributo inizia nella pagina successiva)

05 September 2026

Conflict-Free Star-Access in Parallel Memory Systems *

Sajal K. Das§
Department of Computer Science and Engineering
The University of Texas at Arlington
Arlington, TX 76019-0015, USA
e-mail: das@cse.uta.edu

Irene Finocchi
Department of Computer Science
University of Rome "La Sapienza"
Via Salaria 113, 00198 Rome, Italy
e-mail: finocchi@di.uniroma1.it

Rossella Petreschi
Department of Computer Science
University of Rome "La Sapienza"
Via Salaria 113, 00198 Rome, Italy
e-mail: petreschi@di.uniroma1.it

Abstract

We study conflict-free data distribution schemes in parallel memories in multiprocessor system architectures. Given a host graph G, the problem is to map the nodes of G into memory modules such that any instance of a template type T in G can be accessed without memory conflicts. A *conflict* occurs if two or more nodes of T are mapped to the same memory module. The mapping algorithm should (i) be fast in terms of data access (possibly mapping each node in constant time); (ii) minimize the required number of memory modules for accessing any instance in G of the given template type; and (iii) guarantee load balancing on the modules. In this paper, we consider conflict-free access to *star* templates, i.e., to any node of G along with all of its neighbors. Such a template type arises in many classical algorithms like breadth-first search in a graph, message broadcasting in networks, and nearest neighbor based approximation in numerical computation. We consider the star-template access problem on two specific host graphs – tori and hypercubes – that are also popular interconnection network topologies. The proposed conflict-free mappings on these graphs are fast, use an optimal or provably good number of memory modules, and guarantee load balancing.

*This work has been partially supported by MIUR, the Italian Ministry of Education, University and Research, under Project ALGO-NEXT ("Algorithms for the Next Generation Internet and Web: Methodologies, Design and Experiments") and by the University of Rome "La Sapienza" (project "Parallel and Distributed Codes"). A preliminary version of this work appeared in the *Proceedings of the 18th International Parallel and Distributed Processing Symposium* (IPDPS'04), Santa Fe, New Mexico, April 2004 [15].

§Part of this work was done while this author was visiting the Department of Computer Science of the University of Rome "La Sapienza".

1 Introduction

The CPU speed has been traditionally increasing at a much faster rate than the memory speed. This results in higher memory latency and lower bandwidth or memory access rate that put limits on the overall performance of algorithm execution on a given architecture, be it uniprocessor or multiprocessor. Such a bottleneck can be overcome by one of two ways: (i) organizing the memory hierarchically by adding several levels of *caches*, or (ii) partitioning the memory into *multiple modules* (also called memory *banks*). The concept of multibank partition has gained popularity, particularly in shared-memory multiprocessors [8], and is sometimes used along with multi-level caches. Systems with multiple memory banks are often called *parallel memory systems*: the memory modules can be accessed in parallel as long as the processors request access to distinct modules. In other words, *simultaneous* access to the same module by different processors leads to what is called a *memory conflict*. Conflicting access requests must be queued, which increases the memory access *latency*. Clearly, the *efficiency* of data distribution schemes into modules is crucial in minimizing conflicts, thus exploiting the data parallelism offered by multibank architectures and improving the overall performance [30]. Data distribution schemes should also be [1, 3]:

- *balanced*, by storing an almost equal number of data items in each module;
- *direct*, allowing the module to which a given item is assigned to be determined analytically;
- *scalable*, with the number of available memory modules.

In this paper, we focus on the problem of distributing data onto a parallel memory system such that a high degree of data-parallelism is achieved. Assuming that the multiprocessor architecture is able to request a number of data items from the memory subsystem, our goal is to store data items in such a manner that they can be accessed with as few (ideally, zero) conflicts as possible. As a consequence, we keep transparent the interconnection topology for inter-processor communication as well as the algorithm implementation details. Informally, the *data distribution* problem is defined as follows.

> Let G be a *host* graph representing the data structure corresponding to an application. Let T be a given *template* type in G. Instances of T are a family of subsets of nodes, that describe which elements (or subgraphs) of G are to be accessed together. The data distribution problem is to design an efficient algorithm for mapping the data items onto memory modules such that either of the following two objectives are met: (i) minimize the number of conflicts for accessing the specified template when a fixed number of memory modules is given; (ii) minimize the number of modules for accessing the specified template when conflict-freeness must be guaranteed.

We deal with objective (ii) in this paper. In general, minimizing the conflicts for arbitrary templates or memory access patterns is computationally hard, since this problem can be reduced to a variant of hypergraph coloring [24], where a color represents a module and nodes in a template instance must have different colors to guarantee conflict-freeness. A natural way to tackle this problem is by restricting it to special host graphs (e.g., arrays, circular lists, trees, hypercubes) on which we can characterize structured templates useful in practice [35].

1.1 Related Work

Over the last two decades, significant attention has been paid on the conflict-free mapping and access to two-dimensional array data structures, in which templates of interest are rows, columns, diagonals, and subarrays [9, 12, 19, 25, 35]. Mapping of non-numeric data structures, such as trees and graphs, is also investigated in [14, 18, 19, 21]. The main focus of the majority of existing research has been to guarantee conflict-free access with as few memory modules as possible. The proposed mapping schemes consider mostly trees as hosts and elementary templates like paths [7, 17, 18, 34] or subtrees [16, 17, 19]. In particular, conflict-free access to t-ary subtrees of a complete k-ary tree and to subtrees of a binomial tree is proposed in [17], where leaf-to-root path access in trees is also studied. Subcubes of a binary or generalized hypercube are investigated in [16]. These works show that the overlapping of template instances (of a given type) in the data structure plays a significant role in determining the minimum number of memory modules needed to achieve conflict-free mappings.

The mapping algorithm due to [4] can be seen as a first step toward a "unifying" approach that maps a complete binary tree onto memory modules for efficient access to several types of elementary templates. Subsequently, improved and "versatile" algorithms have been proposed [3] for mapping complete trees when accessed by elementary or composite, as well as variable size templates. This is significant, since in a multiprocessor environment, the number of available memory modules for a single cluster of processors may not be fixed and the cluster can be (temporarily) forced to use only a part of the available memory modules for executing a program. Therefore, when the number of modules varies, the template or the host size changes, the mapping scheme should adapt to such change without being recomputed from scratch.

1.2 Our Contributions

In this paper, we investigate efficient data distribution schemes for *conflict-free* access to *star* templates that consists of a node of G along with all of its neighbors. The goal is to color the nodes of the host data structure in such a way that nodes in any arbitrary star are assigned different colors and the size of all color classes is balanced. We call this as the *star-coloring* problem. In the rest of this paper, we will use the terms 'color' and 'memory module' interchangeably. We will consider tori and hypercubes as the host graphs.

Star templates arise in many classical algorithms. Graph algorithms based on breadth-first search, for instance, explore in each iteration a node's immediate neighbors yet to be visited. Message broadcasting in a network by breadth-first tree or binomial spanning tree (as in hypercube topologies [26]) requires the access to star templates. Performance of parallel algorithms for nearest neighbor computation in finite element meshes and numerical approximation of diffusion equations can be enhanced by efficient distribution of data items in various kinds of grids and tori.

Similar coloring problems have been also studied in the context of channel assignment to the base stations of wireless networks, where the same channel can be reused (without signal interference) by two stations that are spaced sufficiently apart [5, 11]. The minimum distance at which channels can be reused is called the co-channel reuse distance, denoted by σ. Our star-coloring problem is related to the case where $\sigma = 3$ [2, 22]. Such a problem has been shown to be intractable [28] and optimal results have been found on special networks, such as rings, grids, complete trees. A survey of results on this topic is presented in [10]. We notice that star-colorings have additional constraints with respect to colorings arising in the

Host graph		Memory modules	Lower bound	Mapping time
Torus $C_h \times C_k$	$\{h,k\}\ mod\ 5 = 0$	5	5	$O(1)$
	$h\ mod\ 5 \neq 0$ or $k\ mod\ 5 \neq 0$	8	6	
Binary Hypercube Q_d		$2^{\lfloor \log d \rfloor + 1}$	$d+1$	$O(d)$

Table 1: Summary of our results. All mappings guarantee balanced load on memory modules. The mapping related to hypercubes is a 2-approximation, likely to be optimal.

context of channel assignment, for example, they should be balanced and should allow the determination of color assignments analytically. As far as we know, neither non-trivial lower bounds nor optimal solutions on tori are known: the only results can be derived from [20] for the torus $C_3 \times C_3$ by looking at it as the product of two complete graphs. With respect to hypercubes, a near-optimal algorithm for conflict-free channel assignment is presented in [33]. This algorithm guarantees load balancing and requires $O(d \log d)$ time and space to compute the channel assigned to any node, where d is the dimension of the hypercube.

Our contributions on star-coloring of tori and hypercubes are summarized in Table 1. With respect to tori, we show how to access star templates using at most 8 memory modules: this is optimal within an additive factor ≤ 2. Our algorithm yields a closed formula that allows it to color each node in only constant time. Deriving the torus coloring needs to consider some technical issues that may not appear evident at a first thought: namely, the presence of wrap-around edges makes the problem of coloring tori much more difficult with respect to coloring grids (many results are indeed available in the literature for grids, but none of them extends to tori without using too many additional colors). With respect to hypercubes, we present a mapping algorithm that guarantees conflict-free access to every star template in a d-dimensional binary hypercube, Q_d, using $2^{\lfloor \log d \rfloor + 1}$ modules. The mapping time and space are $O(d)$, thus improving over [33]. As in [33], our result is a 2-approximation that we conjecture to be optimal. As a by-product, our approach also solves the conflict-free access problem to Q_2 templates in Q_d, thus generalizing the results presented in [16]. Both mappings in Table 1 guarantee balanced load on memory modules.

The remainder of this paper is organized as follows. Section 2 introduces graph-theoretic definitions and terminology specific to the conflict-free data access problem. Section 3 discusses basic properties of star-colorings. Section 4 and Section 5 present our contributions on tori and hypercubes, respectively. Section 6 sums up and suggests possible research directions.

2 Preliminaries

This section introduces preliminary notations and concepts that will be useful throughout the paper. We rely on standard graph-theoretic terminology [6]. Let paths and cycles with k nodes, for $k \geq 0$, be denoted as P_k and C_k, respectively. The length of a path is the number of edges in it. The distance between any pair of nodes in a graph $G(V, E)$ is the length of a shortest path between the nodes. The diameter of G is the maximum distance over all pairs of nodes of G.

2.1 Graphs as Cartesian Products

Given two graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$, their Cartesian product $G_1 \times G_2$ is the graph $G(V, E)$ such that $V = V_1 \times V_2$ and $E = \{(u_1u_2, v_1v_2) \mid \text{either } u_1 = v_1 \text{ and } (u_2, v_2) \in E_2, \text{ or } u_2 = v_2 \text{ and } (u_1, v_1) \in E_1\}$. Interconnection topologies like square meshes, tori, and hypercubes can be obtained as Cartesian product of paths, cycles, and hypercubes of smaller dimensions, respectively.

A *square mesh* $M_{h,k}$ consists of $n = h \cdot k$ nodes arranged into an $h \times k$ grid such that a node $v_{i,j}$ is adjacent to nodes $v_{i\pm1,j}$ and $v_{i,j\pm1}$, whenever they exist. A *torus* $T_{h,k}$ is obtained from a square mesh $M_{h,k}$ by adding wrap-around edges that connect nodes on the boundary of the mesh. Clearly, $M_{h,k} = P_h \times P_k$ and $T_{h,k} = C_h \times C_k$. A d-dimensional *hypercube* (Q_d) is a d-regular graph with 2^d nodes, numbered from 0 to $2^d - 1$, each having a binary label of d bits. The least and most significant bits in the node label have indexes 1 and d, respectively. Two nodes in Q_d are adjacent if and only if their binary labels differ in exactly one position (i.e., their Hamming distance is 1). Note that $Q_d = Q_{d-i} \times Q_i$, for $0 \leq i \leq d$, and in particular $Q_d = Q_{d-1} \times Q_1$.

2.2 Conflict-free Data Access vs. Node Coloring

Terminology specific to the conflict-free data access problem includes the following. Given a host graph G and a *mapping* m of its nodes into memory modules, let $m(v)$ denote the memory module to which node v is assigned. The *load* on a module is defined as the number of nodes assigned to it. A mapping is *load-balanced* if it evenly distributes the nodes of G onto the modules, and *perfectly load-balanced* if the loads on any two modules differ by at most 1. A mapping is *direct* if the module $m(v)$ can be computed in $O(1)$ time for each node v, and *optimal* if it distributes the nodes of G onto the minimum number of memory modules such that any occurrence of the specified template type in G can be accessed without conflicts.

We note that perfectly load-balanced mappings are also known in graph theory as equitable colorings [27]. Moreover, since the nodes on any path of length 2 in star-coloring must be assigned different colors, it is easy to prove the following lemma:

Lemma 1 *A node coloring of a graph G is a star-coloring if and only if each pair of nodes at distance ≤ 2 in G are assigned with different colors.*

Let $\chi_2(G)$ be the number of colors necessary in order to assign different colors to any two nodes of G at distance ≤ 2. (In the literature, $\chi_2(G)$ is sometimes denoted as $\chi(G^2)$, where G^2 is the square of graph G, containing the same set of vertices as G and edges (u, v) if and only if u and v have distance ≤ 2 in G.) Then, by Lemma 1, the minimum number of memory modules that allows conflict-free access to any occurrence of a star template in G corresponds exactly to $\chi_2(G)$.

3 Star-coloring and Maximum Degree

In this section we discuss some basic properties of a star-coloring. As with the traditional node-coloring problem [23], there is a strong relationship between $\chi_2(G)$ and the maximum degree Δ of the graph. Namely:

$$\chi_2(G) \geq \Delta + 1 \tag{1}$$

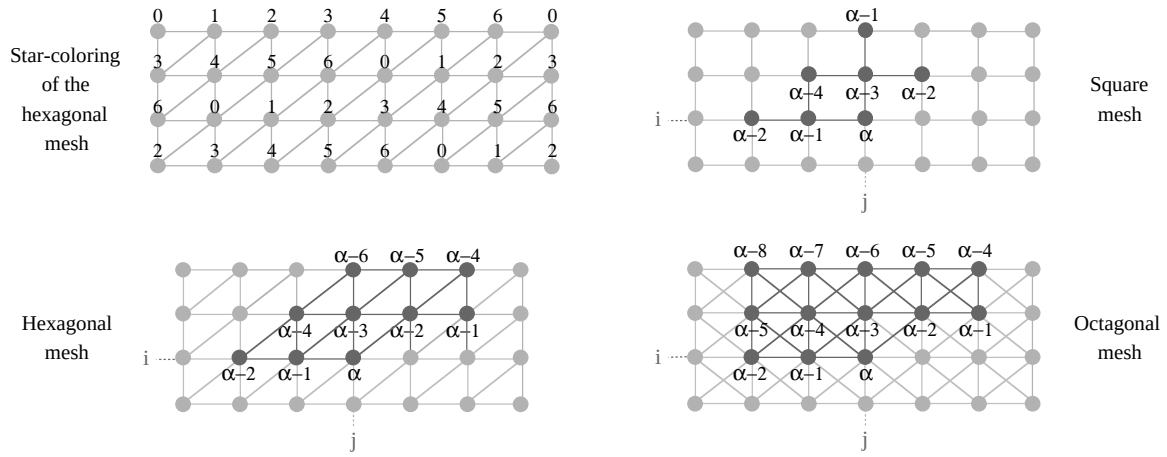


Figure 1: Star-coloring of square, hexagonal, and octagonal meshes.

This is because $(\Delta+1)$ is the maximum size of a star template. Interestingly, this lower bound is not tight even for very simple host graphs, such as cycles: this depends on the overlapping of the template instances in the host graph, and is in line with the results obtained for other kinds of templates (see, e.g., [3]). In the following, we show examples in which either $(\Delta+1)$ colors are sufficient or as many as (Δ^2+1) colors are necessary.

A good example. Square, hexagonal, and octagonal meshes can be star-colored with $(\Delta+1)$ colors, where $\Delta = 4, 6, 8$, respectively (hexagonal and octagonal meshes can be obtained from the square mesh by adding "diagonal" edges as shown in Figure 1). We now show a mapping m that works on all these meshes using the maximum degree Δ as the only distinguishing parameter. This generalizes the result presented in [7], which studies conflict-free access to length-2 paths in square meshes only. We assume that a node in the mesh is identified by (i, j) and the row and column indices start from 0. The mapping is as follows:

$$m(i,j) = \begin{cases} j \ mod \ (\Delta+1) & \text{if } i = 0 \\ m(i-1, j+3) = (m(i-1,j)+3) \ mod \ (\Delta+1) & \text{otherwise} \end{cases} \tag{2}$$

The coloring of the first row is obtained by cyclically repeating the pattern $(0, 1, \ldots, \Delta)$. The coloring of a generic row i, for $i > 0$, is obtained by shifting the coloring of row $(i-1)$ to the left by three positions. (An example of star coloring on the hexagonal mesh is shown in Figure 1.) The mapping is optimal, as it uses exactly $(\Delta+1)$ colors, and can be easily made direct by unrolling the recursion, thus obtaining:

$$m(i,j) = (j+3i) \ mod \ (\Delta+1) \tag{3}$$

Theorem 1 *The direct mapping m given in Equation (3) is an optimal star-coloring of square, hexagonal, and octagonal meshes with h rows and k columns. The coloring is balanced, and the cardinality γ of each color class satisfies $h \cdot \lfloor \frac{k}{\Delta+1} \rfloor \leq \gamma \leq h \cdot \lceil \frac{k}{\Delta+1} \rceil$.*

Proof. To prove that mapping m yields a star-coloring, let α be the color assigned to a node (i, j) in row i and column j. We say that node (r, s) precedes (i, j) in the mesh if either $r < i$, or $r = i$ and $s < j$. Let $N(i, j)$ be the set of nodes that precede (i, j) and have

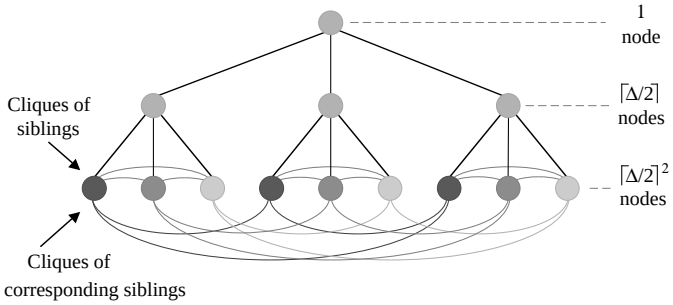


Figure 2: Class of graphs for which $\chi_2(G) = \Theta(\Delta^2)$. In this example, $\Delta = 5$.

distance ≤ 2 from it. Clearly, the larger the degree Δ, the larger is the size of $N(i,j)$. We now prove that α is different from the colors of all nodes in $N(i,j)$. Figure 1 shows the colors of all the nodes in $N(i,j)$ as a function of α. As far as the mapping works and due to the properties of the modulo operator, a node with label $(\alpha - c)$ in the figure is assigned with color $(\alpha - c)$ *mod* $(\Delta + 1)$ by the mapping m. Since $c \in [1, \Delta]$, the color α will be different from those of all other nodes in $N(i,j)$.

With respect to load balancing, it is sufficient to observe that every color appears on each of the h rows either $\lfloor \frac{k}{\Delta+1} \rfloor$ or $\lceil \frac{k}{\Delta+1} \rceil$ times. □

A bad example. There exists an infinite class of graphs for which $\Theta(\Delta^2)$ colors or memory modules are necessary to guarantee conflict-freeness. The skeleton of these graphs is shown in Figure 2. Each graph is a $\lceil \frac{\Delta}{2} \rceil$-regular tree consisting of three levels with leaves interconnected to form $\Theta(\Delta)$ cliques, each consisting of $\lceil \frac{\Delta}{2} \rceil$ nodes. Each leaf belongs to two different cliques: one is formed by its siblings in the base tree; while the other one consists of leaves in different subtrees but in equal position with respect to the leaves ordering in each subtree (i.e., the clique of all the first leaves, the clique of all the second leaves, and so on). Such a graph has $\Theta(\Delta^2)$ nodes and diameter 2. Since any graph G with n nodes and diameter 2 has $\chi_2(G) = n$, it is easy to show that $n = \Theta(\Delta^2)$ colors are necessary in our example in order to avoid conflicts.

4 Star Template in Tori

In this section we describe a mapping algorithm to access without conflicts any star template in a torus $C_h \times C_k$ with h rows and k columns. Our algorithm yields a direct, balanced mapping using at most 8 colors, which are optimal within an additive factor ≤ 2. We assume that nodes in the torus are identified by pairs (i,j) where $i, j \geq 0$. Throughout this section, the color of a node u in position (i,j) will be denoted either as c_u or as $m(i,j)$, where m is the mapping.

4.1 Lower Bound

We start by discussing when the lower bound on $\chi_2(G)$ given in Inequality (1) is tight, i.e., by characterizing tori for which there exist $(\Delta + 1)$ star-colorings. The following observation is useful for this purpose.

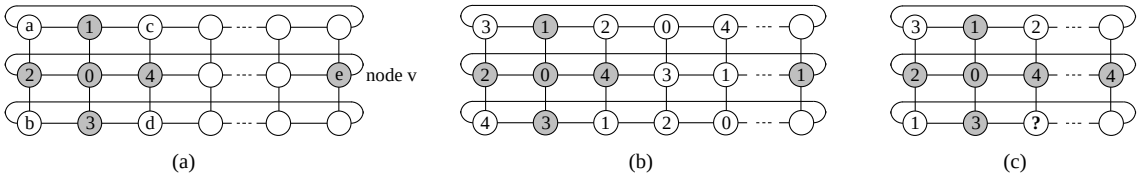


Figure 3: Colorings of the topology $P_3 \times C_t$ used in the proof of Lemma 2.

Lemma 2 *The topology $P_3 \times C_t$, for $t \geq 3$, can be star-colored using five colors only if $t \ mod \ 5 = 0$.*

Proof. In any star-coloring of the Cartesian product $P_3 \times C_t$, the nodes in any star must be assigned with 5 different colors, as shown in Figure 3(a). Let e be the color of node v. Now e can be either 1, or 3, or 4. If $e = 1$, then $b = 4$ and proceeding left-to-right, the choice for the color of any other node is mandatory, as shown in Figure 3(b). It is not difficult to see that the coloring can be completed without conflicts only if $t \ mod \ 5 = 0$. The case $e = 3$ is symmetric. If $e = 4$, colors a, b, and c are forced to be 3, 1, and 2, respectively, but d cannot be assigned with any of the colors 0, 1, 2, 3, 4 (see Figure 3(c)). □

Lemma 3 *Any star-coloring of the torus $C_h \times C_k$, for $h, k \geq 3$, requires at least 5 colors. Furthermore, $\chi_2(C_h \times C_k) = 5$ if and only if $h \ mod \ 5 = k \ mod \ 5 = 0$.*

Proof. $\chi_2(C_h \times C_k) \geq 5$ in view of Inequality (1). If both dimensions h and k are multiples of 5, then $C_h \times C_k$ can be star-colored with five colors as follows:

$$m(i, j) = (j + 2i) \ mod \ 5 \tag{4}$$

This mapping guarantees conflict-freeness. Indeed, the mapping repeats a pattern of five different colors in each row, and thus $m(i, j-1) \neq m(i, j) \neq m(i, j+1)$ for each i and j, where $1 \leq i \leq k-1$ and $1 \leq j \leq h-1$. Furthermore, since $m(i, j) = m(i-1, j+2) = m(i+1, j-2)$, it follows that $m(i-1, j) \neq m(i, j) \neq m(i+1, j)$. Boundary conditions can also be easily checked since $k \ mod \ 5 = h \ mod \ 5 = 0$.

Let us now assume that either h or k is not a multiple of 5. The impossibility of coloring $C_h \times C_k$ with five colors under this hypothesis follows from Lemma 2 and from observing that $C_h \times C_k$ contains $P_3 \times C_t$ as a subgraph both vertically ($t = h$) and horizontally ($t = k$). □

Lemma 3 implies that $\chi_2(C_h \times C_k) \geq 6$ if h or k is not multiple of 5. It remains an open problem to derive a general lower bound as a function of h and k.

4.2 Algorithm Torus-Coloring

In the following, we show how to star-color any torus $C_h \times C_k$ using at most 8 colors. Recall that a torus is obtained from a square mesh by adding wrap-around edges that connect boundary nodes, and that a square mesh can be easily star-colored with the help of the mapping presented in Lemma 3 (see also [7]). Unfortunately, the presence of wrap-around edges prevents the coloring of the square mesh from being extended to tori unless both h and k are multiples of 5.

We start with an informal sketch of the proposed algorithm `Torus-Coloring`. The algorithm colors $C_h \times C_k$ row by row, by using two different sets of colors on even and odd

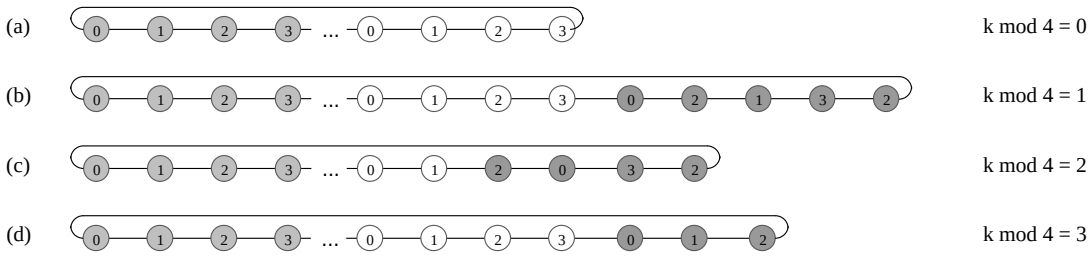


Figure 4: Star-coloring of cycle C_k: the mapping $\widetilde{m}$ described in this figure is used as a building block of torus colorings and can be easily computed in $O(1)$ time.

rows, respectively. Let m be the mapping and let r and s be the row and column indices, respectively. Rows r, $(r+2)$, and $(r+4)$, for any r, pick colors from the same set. In order to avoid conflicts, it must be $m(r,s) \neq m(r+2,s)$, $m(r+2,s) \neq m(r+4,s)$, while it can be $m(r,s) = m(r+4,s)$. Therefore, the coloring of row $(r+2)$ is obtained by suitably *shifting* the coloring of row r, which can instead be repeated in row $(r+4)$. This gives rise to a pattern consisting of four differently colored rows. Due to the vertical wrap-around edges, the last two rows are treated differently.

Basic Row Coloring. The basic row coloring is obtained by using the mapping $\widetilde{m}$ described in Figure 4. This mapping is conflict-free and load-balanced, but not optimal. The reason for using a non-optimal mapping as a basic row coloring will be clear later on.

We now describe formally the algorithm `Torus-Coloring`. For simplicity, we assume that the torus $C_h \times C_k$ has at least 6 columns (i.e., $k \geq 6$). If $k \leq 5$ and $h \geq 6$, we work on the transposed torus $C_k \times C_h$. We omit the coloring of constant-size instances with less than 6 rows and 6 columns.

Repeated Pattern. Let us first focus on the coloring of the four uppermost rows of $C_h \times C_k$. Since the basic row coloring uses four different colors, we assume that rows 0 and 2 get colors from $\{0,1,2,3\}$, while rows 1 and 3 get colors from $\{4,5,6,7\}$. The coloring of row 2 is obtained by cyclically shifting the coloring of row 0 to the left by one position. Similarly, for rows 1 and 3. The pattern is then repeated for all rows except for the last two. Therefore, the mapping of node (i,j), for $0 \leq i < h-2$ and $0 \leq j \leq k-1$, is as follows:

$$m(i,j) = \begin{cases} \widetilde{m}(j) & \text{if } i \bmod 4 = 0 \\ 4 + \widetilde{m}(j) & \text{if } i \bmod 4 = 1 \\ \widetilde{m}((j+1) \bmod k) & \text{if } i \bmod 4 = 2 \\ 4 + \widetilde{m}((j+1) \bmod k) & \text{if } i \bmod 4 = 3 \end{cases}$$

Although the above description and recurrence relation better resemble the algorithmic idea, the mapping m is actually a direct mapping and can be expressed concisely as follows:

$$m(i,j) = 4 \cdot (i \bmod 2) + \widetilde{m}\left(\left(j + \left\lfloor \frac{i \bmod 4}{2} \right\rfloor\right) \bmod k\right) \tag{5}$$

Therefore, the color of any node can be computed in $O(1)$ time since the basic row mapping $\widetilde{m}$ is also direct.

Wrapping Around. Due to the existence of vertical wrap-around edges, the pattern described above cannot be used to color the last two rows for arbitrary values of h and k.

Nevertheless, the vertical cycles can be closed without conflicts and without using new colors. To show this, let us distinguish two cases depending on the parity of h and k.

▶ h **is even:** the last two rows can use the same coloring as rows 0 and 1, respectively, cyclically shifted to the left by 2 positions (called *2-move shift*):

$$m(h-2,j) = \widetilde{m}((j+2)\ mod\ k)$$
$$m(h-1,j) = 4 + \widetilde{m}((j+2)\ mod\ k)$$

Note that a 2-move shift is needed because row $(h-2)$ may be at vertical distance 2 both from a row colored like row 0 and from a row colored like row 2. A similar argument holds for rows $(h-1)$, 1, and 3.

▶ h **is odd:** to deal with this case, let us define the following two mappings:

$$m_1(i,j) = 4 + (j+3)\ mod\ 4$$
$$m_2(i,j) = 4 + (j+1)\ mod\ 4$$

Note that mappings m_1 and m_2 are 1-move shifts of the basic pattern $4,5,6,7$ to the right and to the left, respectively. We color row $(h-2)$ using mapping m_1, and row $(h-1)$ using mapping m_2, except for the case when $k\ mod\ 4 \neq 0$. In this case, at most ten boundary nodes in the last two rows are assigned with different colors. The appropriate coloring of such nodes is shown in Figure 5 and can be computed in $O(1)$ time based on the values of k and h.

4.3 Analysis

The following theorem proves that the coloring described in Section 4.2 is conflict-free and near-optimal with respect to the number of colors used and also load balanced.

Theorem 2 *Algorithm* `Torus-Coloring` *yields a direct, conflict-free, and load-balanced mapping for accessing any occurrence of a star template in a host torus $C_h \times C_k$. The mapping uses 8 colors and is optimal within an additive factor ≤ 2. The load λ on any memory module satisfies*

$$\left\lfloor \frac{h}{2} \right\rfloor \left\lfloor \frac{k}{4} \right\rfloor \leq \lambda \leq \left\lceil \frac{h}{2} \right\rceil \left(\left\lfloor \frac{k}{4} \right\rfloor + 2 \right)$$

Proof. If both h and k are multiples of 5, the mapping in Equation (4) in the proof of Lemma 3 solves the star-coloring problem optimally. Otherwise, $\chi_2(C_h \times C_k) \geq 6$. Since algorithm `Torus-Coloring` uses at most 8 colors, the additive factor is at most 2.

To prove conflict-freeness, let u and v be any two nodes at distance ≤ 2. In view of Lemma 1, it is sufficient to show that the colors c_u and c_v of u and v, respectively, are different. To begin with, observe that if h is odd, $k\ mod\ 4 \neq 0$, and either u or v is any of the grey nodes shown in Figure 5, then their colors do not conflict, as shown in the same figure. In the other cases, let r_u and r_v be the rows corresponding to nodes u and v, respectively.

- If $r_u = r_v$, then $c_u \neq c_v$ because the basic cycle coloring derived from mapping $\widetilde{m}$ is conflict-free.

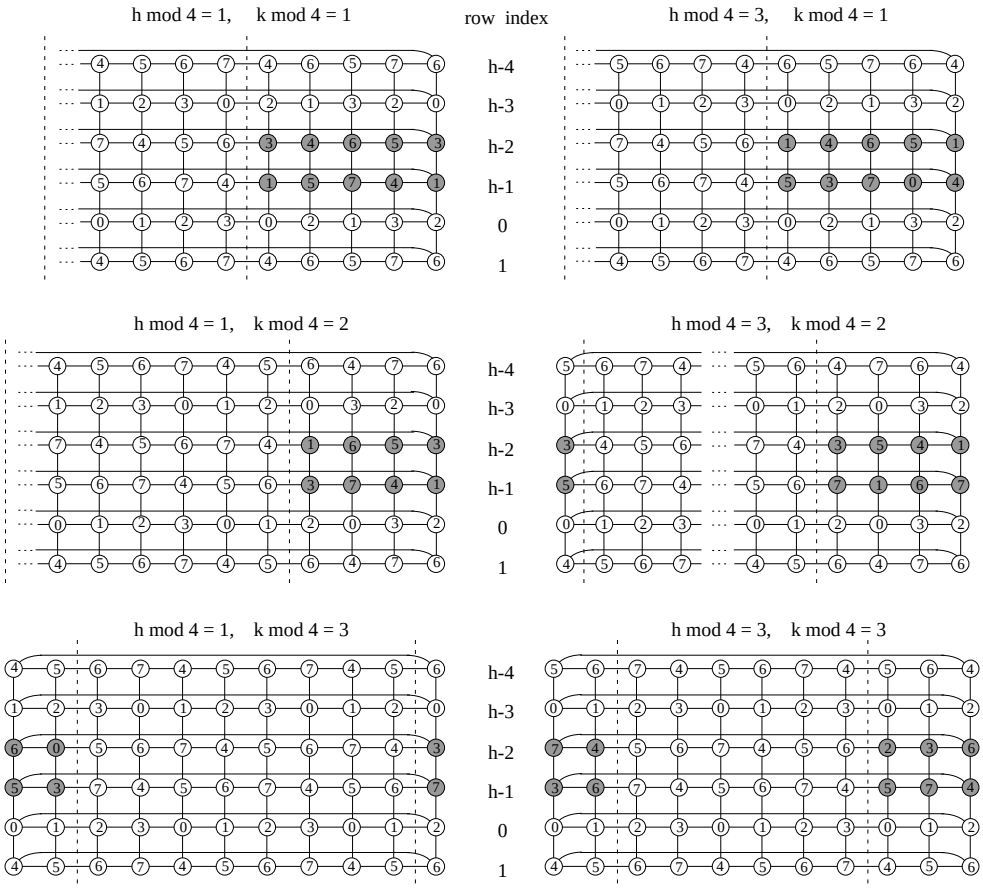
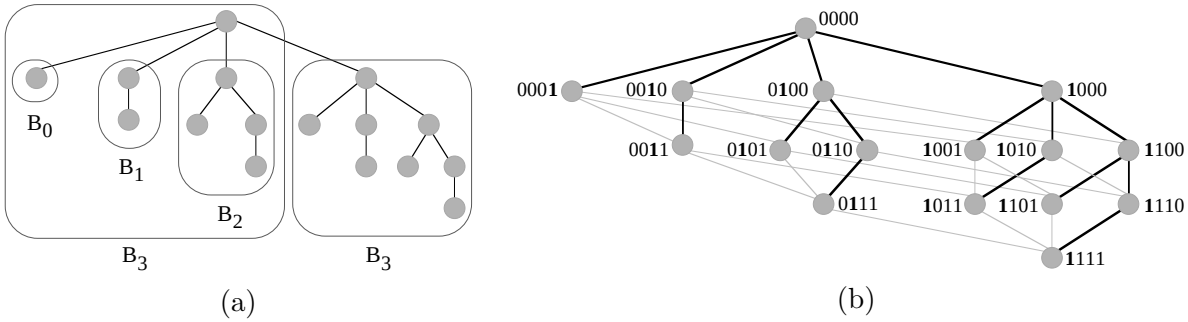


Figure 5: Closing vertical cycles without introducing new colors when $k \ mod \ 4 \neq 0$ and $h \ mod \ 2 = 1$. The left and right columns represent the cases where $h \ mod \ 4 = 1$ and $h \ mod \ 4 = 3$, respectively. Rows 0 and 1 are shown below row $h - 1$.

- If $r_v = (r_u + 1) \ mod \ h$, the colors of nodes u and v are drawn from disjoint sets, and thus cannot conflict. The only exception is when h is odd, $r_u = h - 2$, and $r_v = h - 1$. In this case, it is easy to see that $m_1(h-2, j) \neq m_2(h-1, j)$ as far as mappings m_1 and m_2 are defined. (The case where $r_u = (r_v + 1) \ mod \ h$ is symmetric.)

- If $r_v = (r_u + 2) \ mod \ h$, then u and v must be on the same column, since by hypothesis they are at distance ≤ 2. If rows r_u and r_v get their colors from the same set, as far as the algorithm works, one of these rows must be obtained from the other one by a 1-move or a 2-move shift. Thus, node v takes the same color of a node in row r_u which is at distance ≤ 2 from u: such a color is different from c_u due to the conflict-freeness of $\widetilde{m}$. (The case where $r_u = (r_v + 2) \ mod \ h$ is symmetric.)

With respect to load balancing, note that $\lfloor h/2 \rfloor$ rows use colors from $[0, 3]$ and the remaining $\lceil h/2 \rceil$ rows use colors from $[4, 7]$. Moreover, for any two colors in the same set, the number of times they appear on the same row differs by at most 2. Hence, the load λ on any memory module is bounded as in the statement of the theorem. □



(a) (b)

Figure 6: (a) The binomial tree B_4. (b) The hypercube Q_4 and its binomial spanning tree B_4: edges in B_4 are black and solid.

5 Star Template in Hypercubes

This section addresses the problem of distributing the nodes of a d-dimensional binary hypercube, Q_d, into memory modules so as to guarantee conflict-free access to any star template in Q_d. In particular, we present a linear time algorithm for conflict-free star access that uses $2^{\lfloor \log d \rfloor + 1}$ colors and guarantees perfectly balanced load on memory modules. From the algorithm we derive a mapping that computes the memory module assigned to any node of Q_d in $O(d)$ time. As a by-product, our approach also solves the conflict-free access to Q_2 templates in Q_d. This generalizes the result in [16], which presents a coding theoretic approach to the Q_2-access problem and is applicable only if codes with certain features exist (e.g., Hamming perfect codes or maximum distance separable codes).

Before describing the algorithm, let us review the concept of binomial trees. A binomial tree B_k of order k is an ordered, rooted tree recursively defined as follows [13]: (a) B_0 is a single node; (b) for $k \geq 1$, B_k consists of two binomial trees of order $(k-1)$, where the root of one tree is the rightmost child of the root of the other, as shown in Figure 6(a). Clearly, B_k has 2^k nodes, depth k, and $\binom{k}{i}$ nodes at level i, for $0 \leq i \leq k$. The hypercube Q_d has a binomial spanning tree B_d of order d [26] (Figure 6(b) shows the hypercube Q_4 together with its binomial spanning tree). In particular, if we hang Q_d at node 0, the root of B_d has d different children $q_0, \ldots, q_{d-1}$, where $q_i = 2^i$ (see Figure 7). For each $i \in \{0, 1, d-1\}$, node q_i is the root of the binomial spanning tree B_i of a subcube Q_i of dimension i. The subgraph induced by node 0 and by the nodes in all the subcubes Q_j, with $j < i$, is in turn a hypercube Q_i. Recall that $Q_d = Q_{d-1} \times Q_1$, i.e., Q_d is obtained by connecting two hypercubes

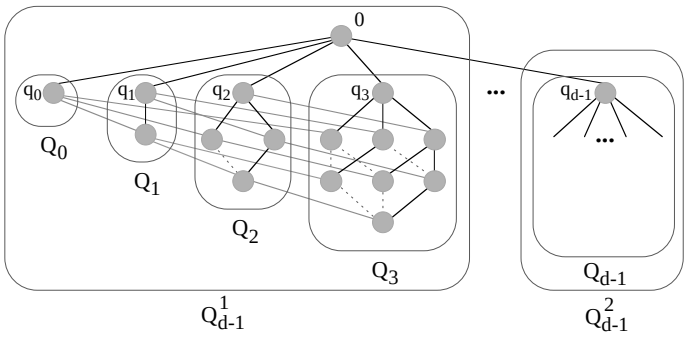


Figure 7: The binomial spanning tree B_d of hypercube Q_d: edges in B_d are black and solid.

```
    Algorithm Hypercube-Coloring (Q_d)
1.  begin
2.      m(0) ← 0                                  {m is the mapping computed by the algorithm}
3.      for i = 0 to d − 1
4.          for v = 2^i to 2^{i+1} − 1
5.              m(v) ← (i + 1) ⊕ m(v − 2^i)       {Note that m(q_i) = i + 1}
6.      return m
7.  end
```

Figure 8: Conflict-free star-coloring of the d-dimensional hypercube, Q_d.

of dimension $(d-1)$, say Q^1_{d-1} and Q^2_{d-1}, as shown in Figure 6(b). Let us assume that the nodes in Q^1_{d-1} have labels $< 2^{d-1}$: each node in Q^2_{d-1} has a unique neighbor in Q^1_{d-1} and the binary labels of these two nodes differ exactly in position d.

5.1 Algorithm Hypercube-Coloring

The pseudocode of algorithm `Hypercube-Coloring` is given in Figure 8. It considers the binomial spanning tree B_d obtained by hanging Q_d at node 0; nodes and subcubes are named as above. Subcubes Q_i, for $0 \leq i \leq d-1$, are sequentially colored. In order to color Q_i, the star-coloring of the subgraph $\bigcup_{j=0}^{i-1} Q_j$ is used. In particular, the color of the root q_i is assigned with $m(q_i) = i+1$. The color of any other node v in Q_i, for $2^i < v < 2^{i+1}$, is obtained from the bitwise **xor** operation, $\oplus$, between $m(q_i)$ and the color of node $u = v - 2^i$. Observe that u is a node in $\bigcup_{j=0}^{i-1} Q_j$ and also that u is a neighbor of v, since their binary labels differ only in position i. For homogeneity, in our pseudocode, $m(q_i)$ is also obtained from a **xor** operation with value $m(q_i - 2^i) = 0$. The following two lemmas discuss properties of the coloring generated by the algorithm that will be used in the proof of Theorem 3.

Lemma 4 *Let $d > 0$ and $\varphi_d = 2^{\lfloor \log d \rfloor + 1}$. Algorithm* `Hypercube-Coloring`*$(Q_d)$ uses all the colors in $[0, \varphi_d - 1]$.*

Proof. (By induction on d.) The claim is easily verified for $d = 1$. By the inductive hypothesis, assume that algorithm `Hypercube-Coloring` applied to Q_{d-1} uses all the colors from 0 to $\varphi_{d-1} - 1$. Thus, $\lfloor \log(d-1) \rfloor + 1$ bits are necessary and sufficient to represent the colors. If $d = 2^k$ for some k, then $k = \lfloor \log(d-1) \rfloor + 1$, but $(k+1)$ bits are necessary to represent the color d that is assigned to node q_{d-1}. Since colors of nodes in Q^2_{d-1} are obtained by means of a **xor** operation with value d, in this case the number of used colors doubles. If d is not a power of 2, then $\lfloor \log(d-1) \rfloor + 1$ bits are still sufficient to represent color d, and in this case the **xor** operation does not introduce any new color. □

Lemma 5 *Let u and v be two adjacent nodes of Q_d (for $d > 0$) whose binary labels differ in position i, for $1 \leq i \leq d$. The mapping m computed by the algorithm* `Hypercube-Coloring` *yields $m(u) = m(v) \oplus i$.*

Proof. The statement can be easily verified on hypercubes of dimension $d \leq 2$. By induction, assume that the coloring produced by the algorithm on any hypercube of dimension $\leq d-1$ satisfies the claim. Let Q_d be obtained from Q^1_{d-1} and Q^2_{d-1}, let the root of Q^2_{d-1} be q_{d-1} and let $m(q_{d-1}) = d$ (see line 5 of algorithm `Hypercube-Coloring`). Without loss of generality, assume that $v < u$. We distinguish three cases, according to the relative positions of u and v in the subcubes Q^1_{d-1} and Q^2_{d-1}:

1. $v \in Q^1_{d-1}$ and $u \in Q^2_{d-1}$: in this case, $i = d$ and thus $m(u) = m(q_{d-1}) \oplus m(v) = d \oplus m(v) = i \oplus m(v)$.
2. $u, v \in Q^1_{d-1}$: the claim holds by the inductive hypothesis.
3. $u, v \in Q^2_{d-1}$: in this case, $m(u) = d \oplus m(\tilde{u})$ and $m(v) = d \oplus m(\tilde{v})$, where $\tilde{u} = u - 2^{d-1}$ and $\tilde{v} = v - 2^{d-1}$, respectively. Clearly, $\tilde{u}, \tilde{v} \in Q^1_{d-1}$. If $m(u) \neq m(v) \oplus i$, then $m(\tilde{u}) = m(u) \oplus d \neq m(v) \oplus i \oplus d = m(\tilde{v}) \oplus i$, that contradicts the inductive hypothesis on Q^1_{d-1}. Therefore, $m(u) = m(v) \oplus i$. □

Theorem 3 *Algorithm* `Hypercube-Coloring` *computes in linear time a perfectly balanced star-coloring of hypercube Q_d using $2^{\lfloor \log d \rfloor + 1}$ colors.*

Proof. With respect to conflict-freeness, by Lemma 1 it is sufficient to prove that any pair of nodes u and v at distance ≤ 2 are assigned different colors. Based on the distance between u and v in Q_d, we consider two cases:

- If u and v are at distance 1, their binary labels differ in a position i and, by Lemma 5, $m(u) = m(v) \oplus i \neq m(v)$ since $i > 0$.
- If u and v are at distance 2, their binary labels differ in two positions i and j, with $i \neq j$. Let w be a common neighbor of both u and v such that the i-th bits in the binary labels of w and v are the same. By Lemma 5, $m(u) = m(w) \oplus i$ and $m(v) = m(w) \oplus j$, from which $m(u) = m(v) \oplus j \oplus i$. Thus, $m(u) \neq m(v)$ since $i \oplus j \neq 0$.

By Lemma 4, the number of colors used by the algorithm is $\varphi_d = 2^{\lfloor \log d \rfloor + 1}$. Let us now prove the load balancing property. We again use induction on d and assume by inductive hypothesis that the coloring of Q^1_{d-1} is perfectly balanced. If $d = 2^k$, the number of colors doubles and perfect balancing is consequently maintained. Otherwise, the colors in Q^2_{d-1} are the same as the colors in Q^1_{d-1} and appear with exactly the same multiplicity. This is due to the **xor** operator and the fact that all the colors from 0 to $\varphi_d - 1$ are already used in Q^1_{d-1} according to Lemma 4. Hence the claim. □

To conclude, we observe that the color of any node v can be computed quickly in a distributed way. In other words, mapping a node to its memory module does not require to color the entire hypercube Q_d, but can be simply obtained from the binary label of the node itself. The mapping works as follows. After initializing $m(v)$ to 0, consider the binary label $b_d b_{d-1} \ldots b_1$ of node v: for each position i such that $b_i = 1$, update $m(v)$ with the result of the **xor** operation between its current value and i. Thus, at most d bitwise **xor** operations are required to compute the color assigned to node v. It is not difficult to see that this mapping assigns any node v with the same color as assigned to v by algorithm `Hypercube-Coloring`. Conflict-freeness, load balancing and the number of colors used thus follow from Theorem 3. In summary we have:

Theorem 4 *The module $m(v)$ assigned to a node v of hypercube Q_d can be computed from the binary label of v in time $O(d)$ using $O(d)$ bits of space.*

This result improves by a multiplicative factor $\Theta(\log d)$ both the mapping time and the space required by the algorithm described in [33], and yields a simple distributed algorithm

for star-coloring of binary hypercubes. Notice that the recursive construction in algorithm `Hypercube-Coloring` results in a sequential star-coloring algorithm faster by a factor $\Theta(d)$ than the sequential algorithm that would be obtained from the iterated application of Theorem 4.

5.2 Remarks on the Lower Bound for $\chi_2(Q_d)$

Since each node of a hypercube Q_d has exactly d neighbors, $(d+1)$ is an obvious lower bound on $\chi_2(Q_d)$. If $(d+1)$ is a power of two, the upper bound is $2^{\lfloor \log d \rfloor + 1} = d + 1$ and hence our star-coloring algorithm is optimal. It has been conjectured (see for instance, [29, 33]) that $2^{\lfloor \log d \rfloor + 1}$ is a lower bound on $\chi_2(Q_d)$ for every d: this would imply the optimality of our solution on every instance. In this section we prove that, if an optimal solution for Q_d exists satisfying an *optimal substructure property*, then the lower bound would hold. The optimal substructure property states that, in an optimal coloring of Q_d, the coloring of the two subgraphs Q_{d-1} from which Q_d is obtained is also optimal.

Lemma 6 *If there exists an optimal solution to the star-coloring problem on Q_d which satisfies the optimal substructure property, then $\chi_2(Q_d) \geq 2^{\lfloor \log d \rfloor + 1}$.*

Proof. It is sufficient to show that the claim holds if the dimension of the hypercube is a power of two. Otherwise, let $\hat{d}$ be the largest power of two such that $\hat{d} \leq d$. We have: $\chi_2(Q_d) \geq \chi_2(Q_{\hat{d}})$ and $2^{\lfloor \log d \rfloor + 1} = 2^{\lfloor \log \hat{d} \rfloor + 1}$. Hence, $\chi_2(Q_{\hat{d}}) \geq 2^{\lfloor \log \hat{d} \rfloor + 1}$ implies that $\chi_2(Q_d) \geq 2^{\lfloor \log d \rfloor + 1}$. In the following we therefore assume $d = 2^k$, for $k > 0$, and prove that any optimal and sub-optimal star-coloring of Q_d must use at least $2d = 2^{\lfloor \log d \rfloor + 1}$ colors.

Let Q^1_{d-1} and Q^2_{d-1} be the two subcubes of dimension $(d-1)$ from which Q_d is obtained. Without loss of generality, let the most significant bit of the binary label of every node in Q^1_{d-1} be 0. By the sub-optimality property, there exists an optimal coloring of Q_d such that the coloring of Q^1_{d-1} is also optimal. Therefore, the coloring of Q^1_{d-1} uses exactly $2^{\lfloor \log(d-1) \rfloor + 1} = d$ colors. We will now show that the coloring of Q_d obtained from the optimal coloring of Q^1_{d-1} must use at least $2d$ colors.

First observe that the star centered at any node of Q_d must use $(d+1)$ different colors. In particular, this holds for node 0. In this case, node 0 and its $(d-1)$ neighbors in Q^1_{d-1} use all the colors from $[0, d-1]$, because the coloring of Q^1_{d-1} is optimal. The unique neighbor of node 0 in Q^2_{d-1} is a node v that must use a color larger than $(d-1)$, say d.

Let x be any neighbor of node v in Q^2_{d-1} and let u be the unique neighbor of x in Q^1_{d-1}. Since the coloring of Q^1_{d-1} is optimal, the star centered at u in Q^1_{d-1} uses all the colors from $[0, d-1]$. Thus, the color of node x must be larger than $(d-1)$. Furthermore, since x is a neighbor of v in Q^2_{d-1}, its color cannot be d. Thus, we have to introduce a new color for x, say $(d+1)$. We can apply the same reasoning on the entire neighborhood of node v in Q^2_{d-1}, introducing a new color for each neighbor. Since the number of such neighbors is $(d-1)$, at the end we will be using all the colors from $[0, 2d-1]$. □

Lemma 6 implies that, if the conjecture were false, then there exist a dimension d such that every optimal coloring of Q_d does not satisfy the optimal substructure property.

6 Conclusions and Open Problems

In this paper we have studied conflict-free access to star templates in such host graphs as tori and hypercubes that are interconnection topologies of multiprocessor architectures,

which implements parallel memory subsystems. The proposed mapping algorithms are fast, load-balanced, and use an optimal or near-optimal number of memory modules to guarantee conflict-freeness. As part of future work, we plan to investigate the mapping of data (or graph) structures on secondary memory or multi-level memory hierarchies [31, 32] with a goal to design efficient external memory algorithms in parallel.

In general, as explained earlier, mapping arbitrary graphs to memory modules so as to guarantee conflict-free access to arbitrary templates is extremely challenging. A number of open questions arise. We have seen that expressing a graph as Cartesian product of smaller graphs may help in designing efficient mappings: it would be therefore interesting to see whether color bounds for two graphs G and H can be useful to obtain color bounds for their Cartesian product $G \times H$. More in general, deriving non-trivial color bounds for graphs obtained from simpler component graphs by using suitable composition operations appears to be a natural approach to the problem (the case of Cartesian products of complete graphs has been already addressed in [20]). Exploring the use of randomization to get conflict-free mappings for general graphs, studying templates in directed graphs, and generalizing our results to k-distance neighborhood graphs or spanning trees represent additional interesting research directions.

References

[1] J. M. Abello and J. S. Vitter. *External Memory Algorithms*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 50, AMS, 1999.

[2] G. Agnarsson and M. M. Halldorson. Coloring Powers of Planar Graphs. *Proc. 11th ACM-SIAM Symp. on Discrete Algorithms* (SODA 00), 654–662, 2000.

[3] V. Auletta, S. K. Das, A. De Vivo, M. C. Pinotti, and V. Scarano. Optimal Tree Access by Elementary and Composite Templates in Parallel Memory Systems. *IEEE Transactions on Parallel and Distributed Systems*, 13:4, 399–412, 2002.

[4] V. Auletta, A. De Vivo, V. Scarano. Multiple Template Access of Trees in Parallel Memory Systems. *Journal of Parallel and Distributed Computing*, 49:1, 22–39, 1998.

[5] R. Battiti, A. Bertossi and M. A. Bonuccelli. Assigning Codes in Wireless Networks: Bounds and Scaling Properties. *Wireless Networks*, 5, 195–209, 1999.

[6] C. Berge. *Graphs and Hypergraphs*. North Holland, Amsterdam, 1973.

[7] A. Bertossi and M. C. Pinotti. Mappings for Conflict-Free Access of Paths in Bidimensional Arrays, Circular Lists, and Complete Trees. *Journal of Parallel and Distributed Computing*, 62, 1314–1333, 2002.

[8] G. E. Blelloch, P. B. Gibbons, Y. Mattias, and M. Zagha. Accounting for Memory Bank Contentions and Delay in High-bandwidth Multiprocessors. *IEEE Trans. on Parallel and Distributed Systems*, 8, 943–958, 1997.

[9] P. Budnik, and D. J. Kuck. The Organization and Use of Parallel Memories. *IEEE Trans. on Computers*, 20, 1566–1569, 1971.

[10] T. Calamoneri, and R. Petreschi. On the Radiocoloring Problem. *Proc. 4th Int. Workshop on Distributed Computing, Mobile and Wireless Computing* (IWDC'02), LNCS 2571, 118–127, 2002.

[11] I. Chlamtac, and S. S. Pinter. Distributed Nodes Organizations Algorithm for Channel Access in a Multihop Dynamic Radio Network. *IEEE Trans. on Computers*, 36, 728–737, 1987.

[12] C. J. Colbourn and K. Heinrich. Conflict-Free Access to Parallel Memories. *Journal of Parallel and Distributed Computing*, 14, 193–200, 1992.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* McGraw-Hill, 2001.

[14] R. Creutzburg and L. Andrews. Recent Results on the Parallel Access to Tree-like Data Structures – The Isotropic Approach. *Proc. Int. Conference on Parallel Processing*, vol. 1, 369–372, 1991.

[15] S. K. Das, I. Finocchi and R. Petreschi. Star-Coloring of Graphs for Conflict-Free Access in Parallel Memory Systems. *Proc. 18th International Parallel and Distributed Processing Symposium* (IPDPS 04), IEEE CS Press, 2004.

[16] S. K. Das and M. C. Pinotti. Conflict-Free Access to Templates of Trees and Hypercubes in Parallel Memory Systems. *Proc. 3rd Int. Conference on Computing and Combinatorics* (COCOON 97), LNCS 1276, 1–10, 1997.

[17] S. K. Das and M. C. Pinotti. Optimal Mappings of q-ary and Binomial Trees into Parallel Memory Modules for Fast and Conflict-Free Access to Path and Subtree Templates. *Journal of Parallel and Distributed Computing*, 60:8, 998–1027, 2000.

[18] S. K. Das, M. C. Pinotti, F. Sarkar. Optimal and Load Balanced Mapping of Parallel Priority Queues in Hypercubes. *IEEE Trans. on Parallel and Distributed Systems*, 7:6, 555–564, 1996.

[19] S. K. Das, F. Sarkar. Conflict-Free Data Access of Arrays and Trees in Parallel Memory Systems. *Proc. 6th IEEE Symp. on Parallel and Distributed Processing*, Dallas (Texas) USA, 377–383, 1994.

[20] J. P. Georges, D. W. Mauro, M. I. Stein. Labeling products of complete graphs with a condition at distance two. *SIAM Journal on Discrete Mathematics*, 14, 28–35, 2000.

[21] M. Gössel and B. Rebel. Memories for Parallel Subtree Access. *Proc. Int. Workshop on Parallel Algorithms and Architectures*, LNCS 299, 122–130, 1987.

[22] J. R. Griggs and R. K. Yeh. Labelling Graphs with a Condition at Distance 2. *SIAM Journal on Discrete Mathematics*, 5:586–595, 1992.

[23] T. R. Jensen and B. Toft. Graph Coloring Problems. John Wiley and Sons, New York, 1995.

[24] D. Kaznachey, A. Jagota and S. K. Das. Neural Network-based Heuristic Algorithms for Hypergraph Coloring Problems with Applications. *Journal of Parallel and Distributed Computing*, 63:9, 786–800, 2003.

[25] K. Kim and V. K. Prasanna Kumar. Latin Squares for Parallel Array Access. *IEEE Transactions on Parallel and Distributed Systems*, 4:4, 361–370, 1993.

[26] T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees and Hypercubes*. Morgan Kaufmann Publishers, 1992.

[27] K. W. Lih. The Equitable Coloring of Graphs. In D.-Z. Du and P.M. Pardalos eds., *Handbook of Combinatorial Optimization*, 3:543–566, Kluwer, Boston, 1998.

[28] S. T. McCormick. Optimal Approximation of Sparse Hessians and its Equivalence to a Graph Coloring Problem. *Mathematical Programming*, 26, 153–171, 1983.

[29] H. Q. Ngo, D. Z. Du, and R. L. Graham. New Bounds on a Hypercube Coloring Problem. *Information Processing Letters*, 84:5, 265–269, 2002.

[30] H. D. Shapiro. Theoretical Limitations on the Efficient Use of Parallel Memories. *IEEE Trans. on Computers*, 17, 421–428, 1978.

[31] J. S. Vitter and E. A. M. Shriver. Algorithms for Parallel Memory I: Two-level Memories. *Algorithmica*, 12(2/3), 110–147, 1994.

[32] J. S. Vitter and E. A. M. Shriver. Algorithms for Parallel Memory II: Hierarchical Multilevel Memories. *Algorithmica*, 12(2/3), 148–169, 1994.

[33] P. J. Wan. Near-Optimal Conflict-Free Channel Set Assignments for an Optical Cluster-Based Hypercube Network. *Journal of Combinatorial Optimization*, 1, 179–186, 1997.

[34] H. A. G. Wijshoff. Storing Trees into Parallel Memories. *Parallel Computing*, Elsevier Science Publishers, 253–261, 1986.

[35] H. A. G. Wijshoff. *Data Organization in Parallel Memories*. Kluwer Academic Publishers, 1989.