Toward a Universal Mapping Algorithm for Accessing Trees in Parallel Memory Systems

Vincenzo Auletta[1] Sajal K. Das[2] Amelia De Vivo[1]
M. Cristina Pinotti[3] Vittorio Scarano[1]

[1]Dipartimento di Informatica ed Applicazioni "R.M. Capocelli"
Università di Salerno, 84081, Baronissi (SA) – Italy

[2]Department of Computer Sciences
University of North Texas, Denton, TX 76203, USA

[3]IEI, Consiglio Nazionale delle Ricerche
Via S. Maria 46, 56126 Pisa – Italy

Abstract

We study the problem of mapping the N nodes of a complete t-ary tree on M memory modules so that they can be accessed in parallel by templates, *i.e. distinct sets of nodes. Typical templates for accessing trees are subtrees, root-to-leaf paths, or levels which will be referred to as* elementary *templates.*

In this paper, we first propose a new mapping algorithm for accessing both paths and subtrees of size M with an optimal number of conflicts (i.e., only one conflict) when the number of memory modules is limited to M. We also propose another mapping algorithm for a composite *template, say $\mathcal{V}$ (as* versatile*), such that its size is not fixed and an instance of $\mathcal{V}$ is composed of any combination of c instances of elementary templates. The number of conflicts for accessing an S-node instance of template $\mathcal{V}$ is $O\left(\frac{S}{\sqrt{M \log M}} + c\right)$ and the memory load is $1 + o(1)$ where load is defined as the ratio between the maximum and minimum number of data items mapped onto each memory module.*

1. Introduction

Let $\mathcal{D}$ be a data structure stored in a distributed fashion (i.e., without replication) into $\mathcal{M}$ memory modules of a multiprocessor machine. The processors read/write from/into the memory modules through a shared bus or an interconnection network topology and concurrently access different sets of nodes (called *templates*) of $\mathcal{D}$. A *conflict* occurs if several processors attempt to access the same memory module simultaneously to retrieve different nodes of the data structure. We say that an instance of a template has k conflicts if at most $k+1$ different processors try to access the same memory module at the same time.

Given an application problem, let a parallel algorithm be such that access to $\mathcal{D}$ is according to given templates. Since different processors are not allowed to access the same memory module at the same step, the time to access the template nodes is proportional to the maximum number of conflicts in any module. Therefore, the overall performance of the parallel algorithm is affected by the mapping procedure used to store the data items of $\mathcal{D}$ onto the memory modules.

We study the problem of mapping the nodes of an N-item data structure $\mathcal{D}$ onto M memory modules such that they can be accessed in parallel according to *templates*, i.e. distinct sets of nodes. For fixed M, this problem can be viewed as a coloring problem where the distribution of nodes into the memory modules is done by *coloring* the nodes with a color from the set $\{0, 1, 2, \ldots, M-1\}$. The *size* S of an instance of the template will be given by the number of nodes in it.

Let $\mathcal{T}$ be the set of template types in the input data structure $\mathcal{D}$, let U be an algorithm/mapping for a single template type $\hat{T} \in \mathcal{T}$ or for the set $\mathcal{T}$ itself. We define the cost $C_U(\hat{T}, M)$ of the mapping U for template $\hat{T}$

when the number of memory modules is limited to M as the maximum over all instances I of $\hat{T}$ of the number of conflicts achieved by U on I using M memory modules; the *cost* of a mapping U for the set $\mathcal{T}$ of template types when the number of memory modules is limited to M will be defined as

$$Cost(\mathrm{U},\ \mathrm{M}) \stackrel{\text{def}}{=} \max_{\hat{T} \in \mathcal{T}} \{C_U(\hat{T}, M)\}$$

For a mapping algorithm U, let MAX(U) and MIN(U) be, respectively, the maximum and minimum number of nodes mapped into the same memory module. We define

$$Load(\mathrm{U}) \stackrel{\text{def}}{=} \frac{\mathrm{MAX(U)}}{\mathrm{MIN(U)}}.$$

A mapping U^* for a set of templates $\mathcal{T}$ (or for a single template thereof) is defined as *M-optimal* if it guarantees the minimum number of conflicts for accessing $\mathcal{T}$ under the constraint that only M memory modules are available. That is, $Cost(\mathrm{U}^*,\ \mathrm{M}) \stackrel{\text{def}}{=} \min_{U \in \mathcal{U}} Cost(\mathrm{U},\ \mathrm{M})$, where $\mathcal{U}$ stands for the set of all mapping algorithms.

We call a mapping U for $\mathcal{T}$ *conflict-free* if there exists some value of M for which U avoids memory conflicts, that is $\exists M : Cost(\mathrm{U}, M) = 0$. Clearly, among all possible conflict-free mappings of a given data structure with respect to specified template types, the more interesting ones are those that also minimize the number of memory modules necessary. Such mappings are called *optimal*.

Straightforwardly, there are no conflict-free mappings using fewer than S memory modules for a template type of size S. More precisely, using M memory modules, any mapping algorithm results in at least $\lfloor S/M \rfloor$ conflicts for each template instance (from the given type) of size S. However, it is not true in general that an *S-optimal* mapping for a template of size S is a conflict-free mapping. In other words, not only the template size S but also the overlapping of template instances in the data structure are crucial, as shown by Das and Pinotti [5, 6], in the context of determining the minimum number of memory modules necessary to guarantee an optimal, conflict-free access.

To summarize, the following properties are desirable for mapping a data structure into parallel memory systems.

(i) *Efficiency*: the number of conflicts for each instance of the considered template type(s), i.e., the number of memory read/write operations should be minimized.

(ii) *Versatility*: should allow efficient data access by an algorithm that uses a set of template types at once.

(iii) *Memory Load Balance*: the load (i.e., the number of data items stored) of each memory module should be balanced.

(iv) *Fast Memory Address Retrieval*: algorithms for retrieving the memory module where a given data item is stored should be simple and fast.

1.1. Previous Work

The problem of conflict-free mapping and access to two-dimensional array data structures have received significant attention over the last decade, where the templates of interest are rows, columns, diagonals, and subarrays. For recent results on arrays, refer to [3, 8, 15]. However, the problem becomes computationally hard when the data structure is an arbitrary graph and templates to be accessed are subgraphs. In fact, the conflict-free template access problem can be reduced to a variant of hypergraph coloring problem [8, 14]. Therefore, a natural way to deal with this problem is to restrict ourselves to special or structured graphs (e.g., trees and hypercubes) and simple templates that can be characterized elegantly.

As compared to arrays, mapping tree data structures in parallel memory systems has received more attention only recently [4, 8, 9, 10, 17, 7]. Moreover, the focus has been mostly on single template types such as T-template (any complete subtree), P-template (any root-to-leaf path) and L-template (all nodes on a level). In the sequel, we refer to these templates as "*elementary*" as opposed to "*composite*" templates. We also emphasize that all the elementary templates considered in the literature have fixed sizes, either explicitly (as in the M-node subtree for T-template [5]) or implicitly by limiting the height of the complete binary tree (as in the P-template[10, 7]).

In particular, Das et al. [8, 9, 10, 5] have proposed several optimal algorithms for accessing complete binary trees according to tree/level/path templates. However, the mapping for T- and L-templates are different from that of P-template. Auletta et al. [1] recently described a mapping algorithm for accessing an M-node subtree, M adjacent nodes in the same level or M consecutive nodes of a root-to-leaf path such that the number of conflicts is given by $O(\sqrt{M/\log M})$. This algorithm can be seen as a first step toward a "unifying" approach that maps an N-node complete binary tree on M memory modules providing efficient access to tree, path and level templates.

Das and Pinotti [5, 6] have also provided optimal algorithms to access t-ary subtrees of a complete k-ary tree, subcubes of a binary or generalized hypercube and subtrees of a binomial tree. The main emphasis

of this work is to show that the overlapping of template instances (of a given type) in the data structure play a significant role in obtaining optimal conflict-free mappings.

1.2. New Results

This paper follows the route traced by [1, 5, 6] and provide further improvements to mapping trees into memory modules with respect to versatility of template types. The goal is to have a *universal* mapping algorithm that allows efficient access to any set of nodes of a tree structure.

Denoting the T-template and P-template of size M by $T(M)$ and $P(M)$ respectively, our first result, described in Section 2, is an M-optimal mapping, called $U(M, TP)$, for both templates $T(M)$ and $P(M)$, leading to only 1 conflict. Due to the results in [1], the number of conflicts achieved by this algorithm is tight. We obtain this result with the help of a new template type, called *tree-path template* or TP-template for short, which captures the complexity of the conflicts due to the overlapping of both the tree- and path-templates. This new template is also of interest on its own.

Next the generality on templates is obtained by relaxing their sizes and shapes. In particular, we consider a template type, call it $\mathcal{V}$ (as *versatile*), whose instances can have any size S and consist of any combination of disjoint instances of elementary templates such as subtrees/paths/levels, the combination of which need not be connected.

In fact, the versatile template $\mathcal{V}$ adequately models the data access for the well-known range-query problem on a t-ary search tree such that an instance is composed of a path (from the root) attached to subtrees. This template can also be used to model parallel access to distinct templates from a set of processors since, by definition, instances of $\mathcal{V}$ can be obtained by composition of instances of elementary templates that need not be connected.

Although the mapping $U(M, TP)$ yields $O(S/M+c)$ conflicts for an S-node instance of the template $\mathcal{V}$ consisting of c disjoint instances of elementary templates, it suffers from two drawbacks: the memory load is unbalanced and the data address retrieval is costly.

Therefore, we propose a new algorithm, described in Section 3, for mapping complete t-ary trees with N nodes onto M memory modules and obtain the following results:

- the number of conflicts for accessing an S-node instance of template $\mathcal{V}$ consisting of c disjoint instances of elementary templates is $O\left(\frac{S}{\sqrt{M \log M}} + c\right)$,
- the load ratio between the maximum and minimum number of data items mapped onto each module is $1 + o(1)$,
- the time complexity for retrieving the module where a given data item is stored is $O(1)$ if a preprocessing phase of space and time complexity $O(\log N)$ is executed, or $O(\log \log N)$ if no preprocessing is allowed.

2. An Optimal Algorithm for Both T-template and P-template

For complete binary trees, we propose an M-optimal mapping algorithm for accessing two template types, $T(M)$ and $P(M)$, and solve an open problem suggested in [1]. Our algorithm allows the access to both subtrees and paths of size M with only 1 conflict using M memory modules.

We say that a *tree-conflict* (resp. *path-conflict*) is a conflict that occurs while accessing an instance of a subtree (resp. path) template. It has been proved in [1] that mapping algorithms that are optimal with regard to $T(M)$ (resp. $P(M)$) have $\Omega\left(\frac{M}{\log M}\right)$ conflicts when data are accessed using the path (resp. subtree) template. For the sake of completeness, let us state the following lemma from [1].

Lemma 1 *Given M colors, a complete binary tree B, and an optimal algorithm U for $T(M)$ (resp. $P(M)$), there is at least one instance of $P(M)$ (resp. $T(M)$) in B with $\lfloor M/\log M \rfloor$ path-conflicts (resp. tree-conflicts).*

Before proceeding further, let us define a few notations. Indicated by $m = \log(M + 1)$ the height of $T(M)$, for $N > m$, let $TP(M, N)$ be the *tree-path* template, which consists of any two instances of $T(M)$ and $P(N)$ such that the root of the $T(M)$ instance is the uppermost node of the $P(N)$ instance. The size S of the $TP(M, N)$-template is $M + N - m$.

For $j \geq 0$ and $0 \leq i \leq 2^j - 1$, let (i, j) denote the i-th node at level j of a complete binary tree. Let the i-th *block* at level j be the set of 2^{m-1} consecutive nodes $\{i2^{m-1}, \cdots, (i+1)2^{m-1} - 1\}$. In other words, for $j \geq m$, the i-th block at level j consist of the leaves of the subtree of height m rooted at the node $(i, j-m+1)$.

Given a complete binary tree B of height N, the sets of colors $\mathcal{Z} = \{0, \cdots, M + N - m - 1\}$ and $\mathcal{Z}' = \{M, \cdots, M + N - m - 1\}$, let us color B using the following algorithm for accessing TP-templates.

Procedure $U(M+N-m, TP(M,N))$**-mapping**;

Step 1. For $0 \leq j \leq m-1$ and $0 \leq i \leq 2^j - 1$, each node (i,j) in the uppermost m levels of B is assigned to the memory module $2^j - 1 + i$.

Step 2. For $m \leq j \leq N-1$, and $0 \leq k \leq 2^{j-m+1} - 1$, assign node $(k2^{m-1}, j)$ to the $(j-m+1)$-th memory module from the set $\mathcal{Z}'$. That is, the leftmost node in the k-th block at level j is colored with the $(j-m+1)$-th color in $\mathcal{Z}'$.

Step 3. For $1 \leq j \leq N-m$ and $0 \leq i \leq 2^j - 1$, consider the uncolored nodes of the i-th block at level $j+m-1$ in left to right order, while the nodes of the subtree of height $m-1$ rooted at the sibling of the node (i,j), in level by level order. Observing that these two sets have the same size, and that the above subtree is already colored, we assign the k-th node of the i-th block to the same memory module to which is assigned the k-th node of the subtree.

In words, we color level by level the uppermost m levels of the complete binary subtree, B, with the colors $\{0, \cdots, M-1\}$ and divide the nodes of level $j \geq m$ in blocks of size 2^{m-1}. Having assigned a new color to the leftmost node of each block (Step 2), the remaining nodes of the block are colored by reusing (Step 3) the colors employed in the subtree of height $m-1$, which is rooted $m-1$ levels above and at the sibling of the node $(i, j-m+1)$. Although it is immaterial the order by which the colors of that subtree are assigned to the block of nodes, we color the block from the second (the leftmost node is already colored) through the rightmost node using the colors in the same order as they appear while visiting that subtree level by level.

Lemma 2 *The $U(M+N-m, T(M,N)$-mapping procedure for a complete binary tree B of height $N > m = \log(M+1)$ accesses optimally TP-templates without conflicts.*

Proof : Let us first observe that any two nodes assigned to the same memory module (i.e., color) cannot belong to the same $T(M)$-template instance since their lowest common ancestor is at a distance of at least $m+1$ from one of them.

With respect to the paths, by construction, any path starting from the root and of length $m+1$ is conflict-free. This forms the basis of an induction proof. By inductive hypothesis, let us assume that any path $P(N-1)$ of length $N-1$ is conflict-free. While coloring level N, one of the two nodes assigned the same color belongs to the left subtree and the other to the right subtree of their lowest common ancestor, and therefore, there are no conflicts.

Since the number of memory modules used is equal to the template size, the procedure $U(M+N-m, T(M,N))$-mapping is optimal. □

Let us now generalize the above procedure so as to color complete binary trees of any height h. We consider a complete binary tree B of height $h > N$ as a special 2^{N-m}-ary tree $G_{\lceil h-m/N-m \rceil}$, where $m = \log(M+1)$. The nodes of G are complete binary tree of height N in B. Recursively, the root of $G_{\lceil h-m/N-m \rceil}$ consists of the uppermost N levels of B, and a $G_{\lceil h-m/N-m \rceil - 1}$ tree is rooted to each node at level $N-m$ of B. The specialty of G is that nodes of B at levels l, where $jN - jm \leq l \leq jN - jm + m$ and $1 \leq j \leq \lceil \frac{h-m}{N-m} \rceil$, are considered to be the nodes of G at both levels $j-1$ and j.

Finally, to color G, the procedure $U(M+N-m, TP(M,N))$-mapping is applied to the root of G using the colors in the set $\mathcal{Z} = \{0, \cdots, M+N-m-1\}$. Then, for each child x of the root of G, define $\mathcal{Z}'$ as the set of all the colors in $\mathcal{Z}$ that have not been already assigned to the uppermost m levels of node x while coloring the parent of x. Hence, Steps 2 and 3 of the procedure $U(M+N-m, T(M,N))$-mapping are applied to color the uncolored nodes in x reusing the colors in the set $\mathcal{Z}'$. Proceeding this way, the tree G is colored in a level-by-level order. (Note that the set $\mathcal{Z}'$ changes at each node of G.) Thus,

Lemma 3 *The procedure $U(M+N-m, T(M,N))$-mapping is an optimal mapping for any instance of the tree-path template $TP(M,N)$ in any complete binary tree.*

For $N = M$, this mapping can be applied to access c disjoint instances of subtrees $T(S)$ or paths $P(S)$ with $O(S/M + c)$ conflicts. However, since the mapping is recursive in nature and the color of a node depends on all previous node assignments, the retrieval process is costly. Also, with regards to the memory load, we can only claim that the number of occurrences of a given color at most doubles at each level.

Hence, we restrict our attention on this mapping to obtain an optimal access for both $T(M)$- and $P(M)$-template types. For this purpose, let consider a complete binary tree colored by the $U(M, TP(\lfloor M/2 \rfloor, \lceil M/2 \rceil + m))$-mapping, which allows conflict-free accesses to $TP(\lfloor M/2 \rfloor, \lceil M/2 \rceil + m)$ templates, using M memory modules. We obtain the following result.

Theorem 1 *At most two nodes of any instances of $T(M)$ or $P(M)$ template are assigned to the same memory module, that is at most 1 conflict occurs.*

Proof : Any path of length $\lceil M/2 \rceil + m$ and any tree of size $\lfloor M/2 \rfloor$ are conflict-free. Also, any instance of $P(M)$ have at most 2 nodes assigned to the same memory module. Now, for any instance I of $T(M)$, the uppermost $m-1$ levels of I constitue an instance of $T(\lfloor M/2 \rfloor)$ and are conflict-free. Whereas the level m in I consists of two blocks in which we reuse all the colors used in the levels $2, \ldots m-1$ of I. Therefore, at most 1 conflict occurs. □

Recalling that by Lemma 1, no conflict-free mapping exists for both $T(M)$ and $P(M)$ templates using M memory modules, we conclude:

Corollary 1 *If a complete binary tree of arbitrary height is colored by the $U(M, TP(\lfloor M/2 \rfloor, \lceil M/2 \rceil + m))$-mapping, both the $T(M)$ and $P(M)$ template types can be accessed with only 1 conflict. Such a mapping is M-optimal.*

Finally, Lemma 1 can be generalized as follows to t-ary trees.

Lemma 4 *Given M colors, a t-ary tree, and an optimal algorithm U for t-ary tree templates of size M, (resp., for path templates of size M) there is at least one path of length M (resp., one tree of length M) on the t-ary tree with $\lfloor M/\log M \rfloor$ path-conflicts (resp., tree-conflicts).*

It can be shown that, using M memory modules, $t-1$ conflicts occur while accessing both trees and paths of size M in the t-ary tree. However, no kind of optimality can be stated when $t > 2$.

3. The Algorithm GEN-LABEL-TREE

We propose in this section a new mapping algorithm GEN-LABEL-TREE for complete t-ary trees with N nodes onto M memory modules such that the number of conflicts for instances of template $\mathcal{V}$ consisting of c disjoint instances of elementary templates is $O\left(\frac{S}{\sqrt{M \log M}} + c\right)$, where S is the size of the instance; the algorithm has load $1 + o(1)$ and the time complexity for retrieving the module where a given data item is stored is $O(1)$ if a preprocessing phase of space and time complexity $O(\log N)$ is executed, or $O(\log \log N)$ if no preprocessing is allowed.

$\mathcal{V}$ template can be a real step forward a more general model for the following reasons.
(i). Templates not always have the same size: parallel access to data structures is often done passing through a *cache*. In literature, this aspect is not taken into account and then, implicitly, it is assumed that cache has size M i.e. the number of available memory modules. In this section we assume that the (implicitly defined) cache can have size S (not related to M) and therefore, access can be done by asking more than one item per module, even if optimal access can be done.
(ii). Previous papers have focussed on basic templates (or *elementary*, as we call them) such as subtrees, level and paths. Recently, in [5, 6] templates that can be obtained through composition have been presented. In this case, it is possible, for example, to define a template that is a composition of a path (from the root) down to a given level (say to node x) and then, include the nodes in that level starting from x.

We now propose a more general way of composing templates. Our composition scheme covers several realistic applications such as the data access scheme for the well-known range-query problem on a t-ary search tree, where the template is composed by a path (from the root) attached to subtrees.

We allow (as in [5, 6]) templates obtained by composition to be not connected and this models the following scenario: assume that access to the data structure (i.e. the t-ary complete tree) stored on a parallel memory system is done by P processors but that the processors belong to a SIMD/MIMD parallel machine where each processor can be accessing a template of its own. Any not connected template allows to model such a situation since no *a priori* structure can be seen by the instance as a whole but it is just the effect of a *parallel* access by many processors to the same data structure.

3.1. The Algorithm

The idea of the algorithm is similar to [1], generalized in order to take into account the unlimited height of the tree and the arity t of the tree.

Recall that S is the size of the template, N is the size of the t-ary tree and M is the number of available memory modules. We set $m \stackrel{\text{def}}{=} \lfloor \log_t((t-1)M + 1) \rfloor$ and $n \stackrel{\text{def}}{=} \lfloor \log_t((t-1)N + 1) \rfloor$ and denote by $\mathcal{M}$ the whole set of M colors.

Roughly speaking, the algorithm divides horizontally the t-ary tree B in blocks of m levels and color the subtrees in each block with a path-optimal strategy that uses a set of k colors, k to be specified later.

The algorithm divides the color set $\{0, \ldots, M-1\}$ in disjoint groups of k colors and assigns a color group to each subtree in the blocks in such a way that groups are reused as later as possible in the path.

More formally, we partition horizontally B into $\lceil M/m \rceil$ blocks, named $B_0, B_1, \ldots, B_{\lceil n/m \rceil - 1}$. Each block is composed by m levels, except the bottom one, which may contain a smaller number of levels. So each

B_i consists of complete subtrees of height m whose root is on the level $i \cdot m$. We also call the j^{th} subtree of block B_i the subtree rooted at the j^{th} node left-to-right on level $i \cdot m$.

Then, we divide the color set into p subsets, $G_0, G_1, ..., G_{p-1}$ of k or $k+1$ colors, k to be specified later.

Now, B can be viewed as a complete t^m-ary *"macro-tree"* $\mathcal{B}$ of height $H = \lceil n/m \rceil$, where every node (but the last level) is a complete *"micro"* subtree of B of height m as follows: B_0 is the root of $\mathcal{B}$, and for each $i = 1, \ldots, H-1$ the j^{th} node of level i in $\mathcal{B}$ corresponds to the j^{th} subtree of B_i. The GEN-LABEL-TREE algorithm (see below) uses a procedure called MACRO-LABELING to assign color groups to the macro-tree in a path-optimal way and then calls procedure MICRO-LABELING to color the nodes of each micro-tree by using a subset of colors of the color group previously assigned by MACRO-LABELING. The subset of used colors in a color group is built by procedure ROTATE so as to guarantee that the colors of the group are equally used to color the nodes of each level of the tree i.e. each color is used almost the same number of times and the distribution of nodes with the same color in the same level is uniform. This is crucial, first, to keep the load low and, secondly, to bound the number of conflicts in subtree templates.

```
GEN-LABEL-TREE(B, M)
 1  let B be the t^m-ary macro-tree obtained from B
 2  L = ⌊log_t(⌈√(Mm)⌉(t − 1) + 1)⌋
 3  h = (t^L − 1)/(t − 1)
 4  ℓ = h + (t^{m−L} − 1)/(t − 1)
 5  p ← ⌊M/ℓ⌋
 6  k ← ⌊M/p⌋
 7  divide M in G_0, G_1, ..., G_{p−1} subsets of k ....
 8  ..... or k + 1 colors
 9  MACRO-LABELING(B, {G_0, G_1, ..., G_{p−1}, M})
10  label the subtree of the root of B with ...
11  ...group(root), one different color per node
12  counter ← 0
13  for each node x ≥ 1 of B
14     do let X be the m-level subtree related to x
15        let ances be the ancestor of x on level 1
16        let v be the level of x in B
17        offset(x) ← ⌊ances/p⌋
18        MICRO-LABELING(X, ROTATE(group(x),
19                                 offset+ counter))
20        counter ← (counter+ 1) mod (t^m)^{v−1}
```

In the sequel we describe the building blocks of the algorithm.

3.1.1 The Macro Labeling

In this section, we describe the MACRO-LABELING algorithm to color the macro-tree, i.e. the t^m-ary tree $\mathcal{B}$, using as color set $G = \{G_0, G_1, \ldots, G_{p-1}, \mathcal{M}\}$.

The algorithm colors macro-tree $\mathcal{B}$ level by level in a path-optimal way: it assigns $\mathcal{M}$ to the root B_0 and assigns color G_i to all the nodes in levels j where $i = (j+1) \bmod p$.

```
MACRO-LABELING(T, S)
1  group(root) ← s_p
2  for each node i in level 1
3     do group(i) ← s_{i mod p}
4  for each node x in level > 1
5     do let s_j be the color group of parent of x
6           group(x) ← s_{(j+1) mod p}
```

It is trivial to see that the following holds.

Lemma 5 *Algorithm* MACRO-LABELING *color $\mathcal{B}$ in such a way that on every descending simple path of length S on macro-tree $\mathcal{B}$ there are at most S/p conflicts.*

3.1.2 Rotate procedure

In previous subsection, we showed that MACRO-LABELING assigns the same color group to all the subtrees in a block. Nodes of a subtree are colored using a subset of the color group assigned to the corresponding macro-node. In order to guarantee that colors are equally used, two subtrees (in the same block) using the same subset of colors should be as far as possible. This is done by the ROTATE procedure.

```
ROTATE(G_i, offset)
1  G' ← φ
2  let num be the number of colors in G_i
3  let G = {g_0, ..., g_{num−1}}
4  for j = 0 to num − 1
5     do insert g'_j = g_{(j+offset) mod num} in G'
6  return G'
```

Lemma 6 *Let G be the color group assigned to all the nodes in block B_i for any i. Then two subtrees of B_i colored with the same subset of colors of G are $k = |G|$ far apart.*

3.1.3 Micro-tree labeling

The algorithm MICRO-LABELING takes as input a complete subtree S of height $m = \lfloor \log_t((t-1)M + 1) \rfloor$ and a color set $F = \{f_0, \ldots, f_{\ell-1}\}$, where

$$\ell = h + \frac{t^{m-L} - 1}{t - 1},$$

with $L = \left\lfloor \log_t(\lceil\sqrt{Mm}\rceil(t-1)+1)\right\rfloor$ and $h = \frac{t^L-1}{t-1}$.

MICRO-LABELING(S, F)
1 let $F = \{f_0, \ldots, f_{\ell-1}\}$
2 $h \leftarrow \frac{t^L-1}{t-1}$
3 **for** $n = 0$ **to** $h-1$
4 **do** $color(n) \leftarrow f_n$
5 $L \leftarrow \left\lfloor \log_t(\lceil\sqrt{Mm}\rceil(t-1)+1)\right\rfloor$
6 **for** $n = 0$ **to** $\frac{t^{m-L}-1}{t-1} - 1$
7 **do** let c be the index in F of n's color
8 **for** $j = 1$ **to** t
9 **do** $c_j \leftarrow nt + j$
10 **for** $j = 1$ **to** t
11 **do** let i be the leftmost node within...
12 .... level $L-1$ of c_j-subtree
13 $color(i) \leftarrow f_{c+h}$
14 SUBTREE-LABEL(j)

The algorithm assigns a distinct color to each of the nodes of the $L = \log_t(h(t-1)+1)$ top levels of S.

To color the remaining part of S the algorithm traverses the tree level by level starting from the root.

Each time a vertex n is visited, if level $L-1$ of the subtrees rooted at children of n exists, it proceeds to color the nodes at this level. It is easy to verify that such a level exists for the first $(t^{m-L}-1)/(t-1)$ nodes of S.

Let us define the x-subtree as the subtree rooted at node x and let the i-th node of an x-subtree be the node labelled with label i in the standard level by level, left-to-right visit of the x-subtree. Let $c_1, c_2, \ldots, c_t$ be the first, second, ..., last child of n, respectively. To color the nodes at level $L-1$ of c_j-subtree, the algorithm uses a new color for the first node to be colored (that is the leftmost at level $L-1$) and then calls the procedure SUBTREE-LABEL that takes as input j.

This procedure assigns to remaining nodes on level $L-1$ of c_j-subtree, from left to right, the color sequence SEQ_j read by a level by level, left-to-right visit on c_1-subtree, ..., c_{j-1}-subtree, c_{j+1}-subtree, ..., c_t-subtree. We denote by $\text{SEQ}_j(i)$ the i^{th} element of SEQ_j. Since every t-ary tree of height $L-1$ has $(t^{L-1}-1)/(t-1)$ nodes, sequence SEQ_j contains $t^{L-1}-1$ colors, that is the number of remaining nodes to be colored at level $L-1$ of c_j-subtree.

It is easy to see that the following lemma holds.

Lemma 7 *For each subtree in a block labeled by* MICRO-LABELING *there are no path-conflicts and for each level of the subtree of size b at most $\frac{b}{\sqrt{M \log M}}$ level conflicts.*

3.2. Analysis

Theorem 2 *Algorithm* GEN-LABEL-TREE *for template $\mathcal{V}$ has*

$$Cost(\text{GEN-LABEL-TREE}) = O\left(\frac{S}{\sqrt{M \log M}} + c\right).$$

Proof sketch: We first can prove that every S-node instance of an elementary template has at most $O(S/\sqrt{M \log M})$ conflicts. We can, in fact, generalize the proofs in [1] along the following guidelines:
Level template: by Lemmas 6 and 7, the number of conflicts is $\leq 2S/\sqrt{M \log M}$.
Path template: by Lemma 7 there are no path conflicts in the micro-tree, therefore, since color groups are disjoint, we need to count only the path conflicts on the macro-tree. By Lemma 5 a simple descending path of length S/m (in the macro-tree) has at most $S/(mp)$ path-conflict and therefore a path of length S on the tree has at most $S/p = \frac{S}{\sqrt{M \log M}}$ path-conflicts.
Subtree template: if $S \leq M$ then the result follows from [1]. If $S > M$, the instance spans more blocks: we bound the conflicts on each level of the tree and add up all the terms to get the result. By Lemma 7, in every level a color is used at most $\frac{\#\text{ nodes in a level}}{\sqrt{M \log M}}$. Since the maximum cardinality of a level is $(S(t-1)+1)/t$ (the number of leaves on the last level) we have at most

$$\frac{S(t-1)+1}{t\sqrt{M \log M}} \leq \frac{S}{\sqrt{M \log M}}$$

conflicts on a level. Now, by adding up the power series upward the tree, the total number of conflicts is at most $\frac{2S}{\sqrt{M \log M}}$.

What we are left with, is to upper bound the number of conflicts for a template composed by c disjoint instances of elementary templates of size $S_1, S_2, \ldots, S_c$, where $\sum_{i=1}^{c} S_i = S$. The idea is that each elementary template has (by previous results) at most $\left\lceil \frac{2S_i}{\sqrt{M \log M}} \right\rceil$ conflicts. Summing up all the elementary templates we have: $\sum_{i=1}^{c} \left\lceil \frac{2S_i}{\sqrt{M \log M}} \right\rceil \leq \frac{2S}{\sqrt{M \log M}} + c$. □

Theorem 3 *Algorithm* GEN-LABEL-TREE *for template $\mathcal{V}$ has load $1 + o(1)$ and the the time complexity for retrieving the module where a given data item is stored is $O(1)$ if a preprocessing phase of space and time complexity $O(\log N)$ is executed, or $O(\log\log N)$ if no preprocessing is allowed.*

The proof of the theorem (omitted because of space limitation) is quite technical and can be easily obtained by generalizing a similar proof in [1].

Range query templates. As an example of the applications of $\mathcal{V}$ we can sketch the proof that by using GEN-LABEL-TREE it is possible to access range-query template with $O\left(\frac{S}{\sqrt{M \log M}}\right)$ conflicts.

In fact, in this case, the template is a path plus as many complete subtrees as the height of the path. Notice that the topmost complete subtree is the largest and that the height of the complete subtrees decreases by one as we go down the path. Then, the number of nodes in the complete subtrees is at most twice the number of nodes in the topmost complete subtree (geometric series). Thus the length of the path is logarithmic in the number of nodes and therefore, the number of elementary pieces is logarithmic in the number of nodes S as well.

References

[1] V. Auletta, A. De Vivo, V. Scarano, "Multiple Template Access of Trees in Parallel Memory Systems". *Proc. of Intern. Parallel Processing Symp.* (IPPS), Geneva, Switzerland, Apr 1997, pp. 694-701.

[2] M.J. Atallah, M.T. Goodrich, K. Ramaiyer, "Biased Finger Trees and Three-Dimensional Layers of Maxima". *Proc. of Symp. on Computational Geometry*, 1994

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

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

[5] S.K. Das, M.C. Pinotti, "Conflict-Free Template Access in k-ary and Binomial Trees", *Proc. of 11th ACM Intern. Conf. on Supercomputing*, Vienna, Austria, July 1997, pp. 237-244.

[6] S.K. Das, M.C. Pinotti, "Load Balanced Mapping of Data Structures in Parallel Memory Modules for Fast and Conflict-Free Templates Access", *Proc. of 5th Intern. Workshop on Algorithms and Data Structures* (WADS), Halifax, Canada, Aug 1997. In *Lecture Notes in Computer Science*, Vol. 1272, pp. 272-281.

[7] S.K. Das, M.C. Pinotti, and F. Sarkar, "Optimal and Load Balanced Mapping of Parallel Priority Queues in Hypercubes,", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 7, No. 6, June 1996, , pp. 555-564.

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

[9] S.K. Das, F. Sarkar, M.C. Pinotti, "Conflict-free Access of Trees in Parallel Memory Systems and Its Generalizations", *Tech Rep.* CRPDC-94-21, Department of Computer Sciences, University of North Texas, Nov 1994.

[10] S.K. Das, F. Sarkar, M.C. Pinotti, "Conflict-free Path Access of Trees in Parallel Memory Systems with Application to Distributed Heap Implementation", *Proc. of 24th International Conference of Parallel Processing*, Vol. III, pp. 164-167, 1995.

[11] L.J. Guibas, E.M. McCreight, M.F. Plass, J.R. Roberts, "A New Representation for Linear Lists". *Proc. of 9th ACM Symp. on Theory of Computing*, 1977, pp. 49-60.

[12] S. Huddlestone, K. Mehlhorn, "A New Data Structure for Representing Sorted Lists", *Acta Informatica* **17**, 1982, pp. 157-184.

[13] D.H. Lawrie, C.R. Vora, "The Prime Memory System for Array Access", *IEEE Transactions on Computers*, vol. C-31, no. 5, May 1982, pp. 435-441.

[14] D. Kaznachey, A. Jagota and S. K. Das, "Primal-Target Neural Net Heuristics for the Hypergraph k-Coloring Problem," *Proceedings of the International Conference on Neural Networks* (ICNN'97), Houston, Texas, pp. 1251-1255, June 1997.

[15] K. Kim, V.K. Prasanna, "Latin Squares for Parallel Array Access", *IEEE Transactions on Parallel and Distributed Systems*, vol.4, no.4, April 1993, pp. 361-370.

[16] R. Kosaraju, "Localized Search in Sorted Lists", in *Proc. of 13th Annual ACM Symp. on Theory of Computing*, 1981, pp. 62-69.

[17] H.A.G. Wijshoff, "Storing Trees into Parallel Memories", *Parallel Computing*, Elsevier Science Publ., 1986, pp. 253-261