Optimal Tree Access by Elementary and Composite Templates in Parallel Memory Systems

Vincenzo Auletta, Sajal K. Das, *Member*, *IEEE*, Amelia De Vivo,
M. Cristina Pinotti, *Member*, *IEEE Computer Society*, and Vittorio Scarano

Abstract—In this paper, we study efficient strategies for mapping onto parallel memory systems complete trees that are accessed by fixed templates (like complete subtrees, paths, or any combinations their of). These mappings are evaluated with respect to the following criteria: 1) the largest number of data items that can be accessed in parallel without memory conflicts; 2) the number of memory conflicts that can occur when accessing templates of size equal to the number of available memory modules, thereby exploiting the full parallelism of the system; 3) the complexity of the memory addressing scheme, i.e., the cost of retrieving the module where a given data item is mapped. We show that there exist trade-offs between these three criteria and the performance of different mapping strategies depends on the emphasis given on each of these criteria. More specifically, we describe an algorithm for mapping complete binary trees of height H onto M memory modules and prove that it achieves the following performance results: 1) conflict-free access to complete subtrees of size K and paths of size N such that $N + K - \lceil \log K \rceil \leq M$; 2) at most 1 conflict in accessing complete subtrees and paths of size M; 3) $O(\frac{K}{M} + c)$ conflicts when accessing a composite template of K nodes consisting of c disjoint subsets, each subset being a complete subtree, or a path or a set of consecutive nodes in a level of the tree. Furthermore, we show that an existing mapping algorithm results in a larger number, namely $O\left(\frac{K}{\sqrt{M \log M}} + c\right)$, of conflicts when accessing a composite template. However, such an algorithm maps each single node in $O(1)$ time, while the new algorithm requires $O(H/N - \log K)$ time.

Index Terms—Complete trees, composite templates, conflict-free access, elementary templates, mapping scheme, parallel memory system.

1 Introduction

In this paper, we study efficient strategies for mapping data structures onto parallel memory systems. A parallel memory system consists of several modules that can be accessed in parallel; however, it is not allowed to perform simultaneous accesses by different processors to the same module (called a *memory conflict*) and, therefore, these accesses must be queued.

In a multiprocessor environment, the cost of an operation can be distributed among several processors. A considerable amount of research has been done to reengineer algorithms for a parallel environment. For example, a large body of literature exists on designing parallel algorithms for a variety of machine models [4], [12], [13].

Most of these works concentrate on optimizing the time complexity, by splitting the computation work among the processors and/or optimizing the cost of communications. In particular, to reduce the communication costs, algorithms are reengineered in order to exploit the properties of the interconnection network topology such that each processor communicates only with its neighbors.

In an idealized model, such as a parallel random access machine (PRAM), all the processors can access the memory concurrently and obtain their values. In reality, however, the memory consists of several distinct blocks, called *modules*, accessed by the processors through a bus or an interconnection network, where each module can be accessed by only one processor at any instant. Therefore, the cost of implementing an operation on a parallel machine architecture depends not only on how the parallel algorithm is able to share the workload among the processors, but also on how the items of the data structure are mapped onto the memory modules. For improved performance of frequently accessed data structures, it is helpful to add a high degree of *data-parallelism* in memory accesses.

- *V. Auletta, A. De Vivo, and V. Scarano are with the Dipartimento di Informatica ed Applicazioni "RM Capocell," Università di Salerno, 84081, Baronissi (SA), Italy. E-mail: {auletta, amedev, vitsca}@dia.unisa.it.*
- *S.K. Das is with the Department of Computer Science & Engineering, University of Texas at Arlington, Arlington, TX 76019-0015. E-mail das@cse.uta.edu.*
- *M.C. Pinotti is with the Dipartimento di Informatica e Telecomunicazioni, Università di Trento, 38050 Povo (TN), Italy. E-mail pinotti@science.unitn.it.*

Manuscript received 24 Jan. 2000; accepted 8 Oct. 2001.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 111320.

In this paper, we focus on the problem of mapping data onto a parallel memory system such that a high degree of data-parallelism is achieved. Assuming that the multiprocessor machine 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 fewer (ideally, zero) conflicts occur in accessing memory. As a consequence, we are not particularly interested in how the processors communicate, nor do we worry about the actual implementation of the proposed algorithm on a real machine.

Informally, the problem we aim to solve is defined as follows: Given a data structure and the operations allowed on it or, equivalently, given a graph representing the data structure and a family of subsets of nodes, called "*templates*," that describe which elements of the data structure are to be accessed together, our objective is to design an algorithm for mapping data items on to the memory modules.

The mapping strategy must be 1) *efficient*, meaning that it has to achieve a minimum number of conflicts on given templates and compute memory addresses easily and 2) *versatile*, meaning that it has to allow efficient access to different types of templates.

1.1 Problem Definition

Mapping a data structure $\mathcal{D}$ onto a parallel memory system consisting of M modules can be viewed as a *coloring* problem, where the distribution of nodes of $\mathcal{D}$ into the memory is nothing but assigning them colors from the set $\{0, 1, 2, \ldots, M-1\}$.

Let $G_{\mathcal{D}} = (V, E)$ be the graph underlying $\mathcal{D}$. A *template* $\mathcal{I}$ of $\mathcal{D}$ is defined as a subgraph of $G_{\mathcal{D}}$, and any occurrence of $\mathcal{I}$ in $\mathcal{D}$ will be called a *template instance*. After coloring $\mathcal{D}$, we say k *conflicts* occur if $k+1$ nodes of a template instance are assigned to the same color (module). Thus, the problem of mapping $\mathcal{D}$ on M memory modules, when $\mathcal{D}$ is accessed by template $\mathcal{I}$, can be formulated as a *coloring problem* as follows: find an M-coloring of $G_{\mathcal{D}}$ such that the maximum number of nodes of any template instance that have the same color is minimized.

In the general case, this problem is computationally hard [15]. A natural way to deal with it is to restrict the problem to special or structured graphs on which simple but useful templates can be characterized easily. In this paper, we restrict our attention to data structures represented by trees.

Let T be a complete binary tree [1] and let $\{\mathcal{I}_1, \mathcal{I}_2, \cdots, \mathcal{I}_d\}$ be a family of *templates*. We consider a data structure $\mathcal{D}$ that has T as underlying graph and each operation on $\mathcal{D}$ requires to access a subset of T that belongs to a template $\mathcal{I}_i$, for some i. We aim to solve the coloring problem for accessing $\mathcal{D}$ by templates $\{\mathcal{I}_1, \mathcal{I}_2, \cdots, \mathcal{I}_d\}$. More specifically, we focus on four templates for tree data structures:

1. complete subtrees (called S-template),
2. ascending or leaf-to-root paths (called P-template),
3. tree levels (called L-template),
4. any combination of the above templates (called C-template).

Throughout this paper, by trees and subtrees, we mean only complete binary trees and complete binary subtrees. In the sequel, we will refer to S-templates, L-templates, and P-templates as *elementary* templates, while C-templates as *composite* ones. Example templates are illustrated in Fig. 1.

1. A tree is *complete* if and only if all of its leaves are at the same level.

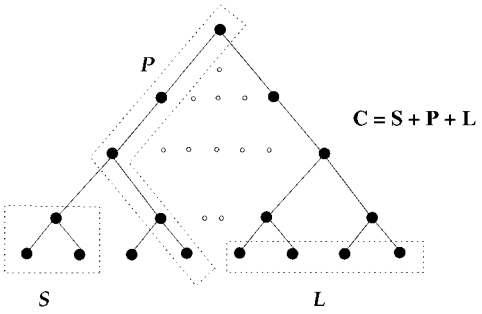

Fig. 1. Templates for tree data structures.

It can be easily seen that most of the meaningful operations for concurrent access to sets of items in tree data structures belong to one of these four templates. As an example, heaps and dictionaries are among the two most popular data structures implemented with trees.

In a binary tree min-heap, operations like insertion of a new key and decrease-key are traditionally implemented by accessing all the nodes of a leaf-to-root path (P-template) of the tree. Furthermore, as shown in [9], [14], the deletion of the minimum can also be implemented (storing proper information at each heap node) by accessing all the nodes of a suitable leaf-to-root path.

In a B-tree, implemented as a complete tree, a range query means accessing (in parallel) all the nodes whose keys belong to a given range; that is, the set of nodes to be accessed can be partitioned into a composite template (C-template) consisting of a set of complete subtrees (S-template) and a path of cardinality no larger than the height of the B-tree.

1.2 Previous Work

Over the last two decades, the problem of conflict-free mapping and access to two-dimensional array data structures have received significant attention, where templates of interest are rows, columns, diagonals, and subarrays. For recent results, refer to [3], [10], [16].

In contrast, mapping of tree structures has been considered only recently [5], [9], [10], [11], [17]. The focus of most of this research has been to guarantee conflict-free access while using as few memory modules as possible. Most of the proposed mappings considers only one kind of elementary templates at a time, such as the P-template or S-template [9], [11], [8]. In particular, Das et al. [6], [10], [11] proposed several conflict-free algorithms for accessing complete binary trees according to S-templates or P-templates that use as few memory modules as possible. However, the mapping strategy for S-templates is substantially different from that of P-templates.

Das and Pinotti [6], [7] provided conflict-free mappings for accessing t-ary subtrees of a complete k-ary tree, subtrees of a binomial tree, and subcubes of a binary or generalized hypercube. In those works, they 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.

Auletta et al. [2] described a mapping algorithm for accessing an M-node subtree, M adjacent nodes in the same level of the tree, or M consecutive nodes of a leaf-to-root 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 onto M memory modules, providing efficient access to several types of elementary templates.

1.3 New Results

This paper follows the route traced by [1], [2] and provides "better" algorithms for mapping complete trees into parallel memory systems when access is performed according to a variety of templates.

We extend the family of templates with respect to their structure and size. We believe that allowing variable size of the template leads to more significant results. In fact, in a multiprocessor environment the number of available memory modules for a single cluster of processors is not fixed and the cluster can be (temporarily) forced to use only a part of all the available resources (M memory modules) for executing a program. Therefore, the mapping algorithm must scale with the number of memory modules.

We measure the performance of our mapping algorithm by two important criteria: the maximum number of conflicts for any template instance and the "versatility" (i.e., possibility of accessing by different templates). Additional criteria include:

1. *Data-Parallelism*. The number of items that can be accessed in parallel should be as large as possible. Note that more than M items, where M is the number of memory modules, cannot be accessed in parallel;
2. *Fast Addressing*. Algorithms for computing the memory module, where a given data item is stored, should be simple and fast.

We show that there exist trade-offs between all the above criteria. In fact, it is possible to reduce the number of memory conflicts by reducing the number of items that have to be accessed in parallel and by using costly algorithms for retrieving memory addresses. On the other hand, if we want to use all the available parallelism of the memory system (i.e., access subsets of size equal to the total number of modules) and design fast algorithms for retrieving data addresses, then the presence of conflicts in accessing the memory is unavoidable.

Our versatile mapping algorithms are presented in the same order as the performance criteria mentioned above. In particular, we first present a conflict-free algorithm that achieves limited parallelism for accesses by S-templates and P-templates, and prove that no algorithm can achieve the same result using fewer memory modules. Next, we show that the same algorithm achieves maximum parallelism at the expense of one conflict for the same templates. Hence, we study the performance of two different mapping algorithms on the (larger) composite template. Both algorithms obtain maximum parallelism. However, the first one has fewer conflicts and an expensive address retrieval, while the second one offers constant-time address retrieval but induces more conflicts.

The rest of the paper is organized as follows: Section 2 introduces some definitions and notations. Section 3 presents the algorithm $\mathrm{COLOR}(T, N, K)$ to map a complete binary tree T on $N + K - \lceil \log K \rceil$ memory modules, guaranteeing conflict-free access to P-templates of size N and S-templates of size K. Thus, we also prove that the minimum number of memory modules needed to guarantee conflict-free access to S-templates and P-templates of size M is $2M - \lceil \log M \rceil$. This settles an open question proposed in [2].

Section 4 measures the performance of the algorithm COLOR when we force maximum parallelism, i.e., when we want to access templates of size equal to the number of available memory modules M. In this case, we prove that the COLOR algorithm is optimal since it guarantees at most one conflict. Moreover, it improves on the algorithm LABEL-TREE, proposed in [2], that accesses S- and P-templates of size M with $O(\sqrt{M/\log M})$ conflicts. However, this improvement is obtained at the expense of a more difficult and expensive data addressing scheme.

In Section 5, we study the versatility properties of the COLOR algorithm, by considering its performance on the C-template. We prove that it yields $O(K/M + c)$ conflicts on instances of the C-template of size K that consist of c disjoint instances of elementary templates. However, this algorithm suffers from two drawbacks: It overloads some memory modules and its intrinsic recursive nature makes the computational cost of the mapping quite expensive.

Finally, Section 6 considers the LABEL-TREE algorithm, discussed in [2], and proves that this algorithm yields

$$O\left(\frac{K}{\sqrt{M \log M}} + c\right)$$

conflicts on instances of the C-template of size K that consist of c disjoint instances of elementary templates. Although this algorithm produces a larger number of memory conflicts than the COLOR algorithm, it allows computing the module where a given data item is stored in $O(1)$ time, if a preprocessing phase of space and time complexity $O(M)$ is executed, or in $O(\log M)$ time, if no preprocessing is allowed [2].

Another interesting property of the LABEL-TREE algorithm is that it equally distributes data items among the memory modules (balanced memory load). Precisely, the ratio between the maximum and minimum number of data items mapped onto the same module is $1 + o(1)$. This result shows an interesting trade-off between the number of conflicts, the simplicity of the addressing scheme, and the balancing of load among memory modules.

2 Preliminaries

Let T be a tree of height H and let $\mathcal{I} = \{\mathcal{I}_1, \mathcal{I}_2, \cdots, \mathcal{I}_d\}$ be a family of templates used to access T. A mapping of T onto an M-module parallel memory system consists of coloring each element of T with a color r from the set $\{0, \cdots, M-1\}$. For each mapping algorithm U and for each instance I of a template $\mathcal{I}_i$, we define the cost $\mathcal{C}_U(T, I, M)$ of U as the number of conflicts produced by U on I, that is:

$$\mathcal{C}_U(T, I, M) = \max_{0 \le r \le M-1} \left| \left\{ u \in I : \text{u is colored by } r \right\} \right| - 1.$$

The cost of U on the template $\mathcal{I}_i$ is then defined as

$$\mathcal{C}_U(T,\mathcal{I}_i,M)=\max_{I\in\mathcal{I}_i}\mathcal{C}_U(T,I,M).$$

The cost of coloring the tree T by the algorithm U under the hypothesis that all accesses are according to the family of templates $\mathcal{I}$, is defined as

$$Cost(T,U,\mathcal{I},M)=\max_{\mathcal{I}_i\in\mathcal{I}}\Big\{\mathcal{C}_U(T,\mathcal{I}_i,M)\Big\}.$$

The mapping of the tree T on an M-module parallel memory system produced by the algorithm U is *M-conflict-free* (in short, M-CF) on the family of templates $\mathcal{I}$ if

$$Cost(T,U,\mathcal{I},M)=0.$$

An M-CF mapping U for accessing $\mathcal{I}$ is *M-CF-optimal* on $\mathcal{I}$ if, for any $M'<M$, there is no M'-CF mapping algorithm for $\mathcal{I}$. In other words, an M-CF-optimal mapping achieves the maximum possible data-parallelism possible with M memory modules: if we consider templates of larger size (or equivalently, fewer colors), no M-CF mapping is possible.

A mapping U^* is *M-optimal* on $\mathcal{I}$ if

$$Cost(T,U^*,\mathcal{I},M)=\min_{U\in\mathcal{U}}Cost(T,U,\mathcal{I},M),$$

where $\mathcal{U}$ denotes the set of all mapping algorithms. In other words, an M-optimal algorithm guarantees to access templates of size M with the minimum number of conflicts.

In the remaining sections, we will omit the notations T and U when they are clear from the context.

Let us emphasize the use of M in the above definitions. In M-CF-optimal mapping, M stands for the number of memory modules available; whereas in M-optimal notation, M stands for the size of templates. Notice that if $\mathcal{I}$ contains a subset of size $K>M$, any mapping U has $Cost(U,\mathcal{I},M)\geq\lceil\frac{K}{M}\rceil-1$. Moreover, it is worthy to point out that the number of memory modules necessary to have an optimal conflict-free mapping on $\mathcal{I}$ depends not only on the size of the instances contained in $\mathcal{I}$ but also on their structures. In particular, if the instances overlap, then a number of memory modules than M is required for optimal conflict-free mapping [6], [7].

2.1 Notations and Terminology

Finally, let us introduce few notations that will be used from now on. For each node $v\in T$, the *level* of v is defined as the distance of v from the root of the tree (the root is at level 0). The level j of T, denoted by $\mathrm{LEV}_T(j)$, is the subset of all nodes of T at level j, where the nodes at a level are in the left-to-right order and the first node is indexed as 0. Thus, the node i of $\mathrm{LEV}_T(j)$ is the $(i+1)$st node visited while traversing $\mathrm{LEV}_T(j)$ in left-to-right order. This node will be denoted as $v_T(i,j)$.

Let $\mathrm{ANC}_T(i,j,k)$ be the kth ancestor of $v_T(i,j)$. For instance, if T is a binary tree, then

$$\mathrm{ANC}_T(i,j,1)=v\left(\left\lfloor\frac{i}{2}\right\rfloor,j-1\right)\text{ and}$$
$$\mathrm{ANC}_T(i,j,k)=v\left(\left\lfloor\frac{i}{2^k}\right\rfloor,j-k\right).$$

For each node $v_T(i,j)$, we consider the following subsets of nodes in T:

- $S_K^T(i,j)$ is the complete subtree of size K rooted at $v_T(i,j)$,
- $L_K^T(i,j)$ is the set (of size K) of consecutive nodes $v_T(i+h,j)$ at level j for $0\leq h<K$,
- $P_K^T(i,j)$ is the set of nodes belonging to the path (of size K) starting from $v_T(i,j)$ to $\mathrm{ANC}_T(i,j,K-1)$.

For each $K=2^k-1$, $k\geq1$, we define the elementary template $S^T(K)$ as the family of all the complete subtree instances of T having size K, i.e.,

$$S^T(K)=\bigcup_{\substack{0\leq j\leq H-k\\0\leq i<2^j}}S_K^T(i,j),$$

where H is the height of the tree.

For each $K>0$, let us define the elementary template $L^T(K)$ as the family of all the instances of K consecutive nodes in a level of T, i.e.,

$$L^T(K)=\bigcup_{\substack{j\geq\lceil\log K\rceil\\0\leq i\leq 2^j-K}}L_K^T(i,j).$$

Similarly, for each $K\leq H$, we define the elementary template $P^T(K)$ as the family consisting of all the ascending paths of length K in T, i.e.,

$$P^T(K)=\bigcup_{\substack{K-1\leq j\leq H\\0\leq i<2^j}}P_K^T(i,j).$$

For each $K>0$ and $c>0$, we define the composite template $C^T(K,c)$ as the family consisting of all the subsets of size K that can be partitioned into $C_1,C_2,\cdots,C_c$, where each C_i is an instance of an elementary template.

In the sequel, we omit the name of the tree T (in the superscript) whenever it is clear from the context.

3 A CF–Optimal Mapping for Elementary Templates

Let $K=2^k-1$ and $N\geq k$. In this section, we describe a mapping algorithm, called COLOR, to access without conflicts all subtrees of size K and paths of size N of a complete binary tree using $(N+K-k)$ memory modules. We also prove that COLOR is $(N+K-k)$-CF-optimal.

The main idea of COLOR is to split the tree into a set of intersecting complete subtrees of height N and color each subtree separately. Thus, COLOR uses a subroutine BASIC-COLOR(B,N,K) to color a height-N subtree B using $N+K-k$ colors in such a way to minimize the number of conflicts on accessing the templates of $P^B(N)$ and $S^B(K)$.

3.1 Algorithm Basic-Color

Let B be a complete binary tree of height N that is accessed by templates $\mathcal{I}=\{S^B(K),P^B(N)\}$. Notice that $P^B(N)$ consists of all the paths from the leaves of B up to the root.

```
BASIC-COLOR(B, N, K)
1  Let Σ = {0, 1, ···, K − 1} and Γ = {K, K + 1, ···, N + K − k − 1}
2  for each j = 0 to k − 1
3  do
4     for each i = 0 to 2^j − 1
5     do
6        color v(i, j) with color 2^j + i − 1 ∈ Σ
7  BOTTOM(B, N, K, Γ)

BOTTOM(B, N, K, Z)
1  for each j = k to N − 1
2  do
3     for each h = 0 to 2^j − 1
4     do
5        let b_0, b_1, ···, b_{2^{k−1}−1} be the nodes of block(h, j)
6        let v_1 = ANC_B(h · 2^{k−1}, j, k − 1), let v_2 be the sibling of v_1 and S_2 be the
         subtree of size K rooted at v_2
7        color node b_i with the color assigned to the (i + 1)st node of S_2 (level-
         by-level, left-to-right order)
8        color b_{2^{k−1}−1} with the (j − k + 1)th color of Z
```

Fig. 2. Algorithm BASIC-COLOR.

For each $j \geq k$, we partition $\mathrm{LEV}_B(j)$ into 2^{j-k+1} blocks, each of 2^{k-1} nodes. For $0 \leq h \leq 2^{j-k+1}-1$, $block(h,j)$ consists of the nodes $v(i,j)$ with $h2^{k-1} \leq i < (h+1)2^{k-1}$. It is worthy to note that $block(h,j)$ consists of leaves of the subtree $S_K(h, j-k+1)$.

Without loss of generality, let $\{0,1,\cdots,N+K-k-1\}$ be the set of colors used to color the binary tree B. The algorithm BASIC-COLOR(B,N,K), presented in Fig. 2, constructs two lists of colors $\sum = \{0,1,\cdots,K-1\}$ and $\Gamma = \{K, K+1,\cdots,N+K-k-1\}$. Then, the coloring of the tree is computed in two phases. In the first phase, each node in the first k levels of B is colored with a distinct color of $\sum$. In the second phase, performed by the algorithm BOTTOM(B,N,K,Γ), the remaining levels of B are colored in a top down order using colors from both the sets $\sum$ and Γ. Each level j is colored blockwise, starting from $block(0,j)$.

Consider $block(h,j)$ containing nodes $b_0, b_1, \ldots, b_{2^{k-1}-1}$ with $0 \leq h \leq 2^j - 1$ and $k \leq j \leq N-1$. Let $v_1 = \mathrm{ANC}_B(h \cdot 2^{k-1}, j, k-1)$ be the $(k-1)$st ancestor of the nodes in $block(h,j)$, and let v_2 be the sibling of v_1. Let S_1 and S_2 denote the subtrees of size K that are rooted at v_1 and v_2, respectively (see Fig. 3). By construction, when the algorithm colors the nodes of $block(h,j)$, all the nonleaf nodes of S_1 and S_2 have already been colored. The algorithm colors the first $2^{k-1}-1$ nodes of $block(h,j)$ with the colors assigned to the nonleaf nodes of S_2. More formally, node b_0 gets its color from the already colored node

$$v_2 = (h + (-1)^{h \bmod 2}, j-k+1),$$

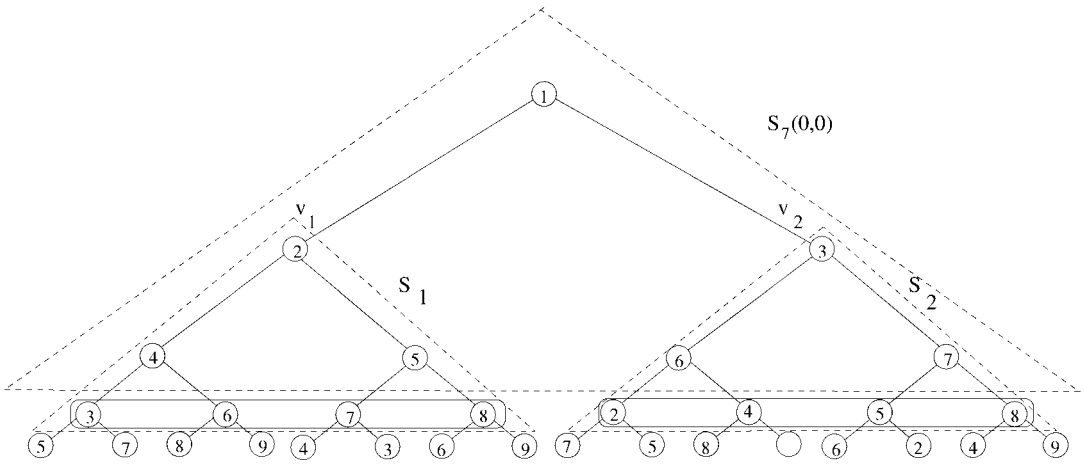

Fig. 3. Description of BASIC-COLOR.

```
RETRIEVING-BASIC-COLOR(B, v(i,j), UP)
1  if UP[v(i,j)] = v(r,s)
2     then color v(i,j) with color 2^s + r - 1
3     else  UP[v(i,j)] = *
4           color v(i,j) with color j - k + K
```

Fig. 4. Algorithm RETRIEVING-BASIC-COLOR.

and node b_i, with $1 \leq i \leq 2^{k-1} - 2$, such that $i = 2^{r-1} - 1 + s$, with $0 \leq s \leq 2^{r-1} - 1$ and $r \geq 1$, gets the same color as node

$$v(2^r(h + (-1)^{h \bmod 2}) + s, j - k + r + 1) \in S_2.$$

Finally, node $b_{2^{k-1}-1}$ gets color $j - k + K$, which is the $(j - k + 1)$th color of Γ. Therefore, lines 6 and 7 of the BOTTOM procedure can be performed in constant time, and the BASIC-COLOR algorithm colors the entire tree B in $O(2^N)$ time.

Although the coloring of the tree takes time proportional to the size of the tree, $O(N - k)$ time is needed to retrieve the color assigned to a given node. Indeed, let $z = v(i, j)$ be a leaf in the second half of a block (at level $N - 1$). By previous remarks, z gets its color from a node at level $N - 2$ that may belong to the right half of a block and, therefore, takes the color from a node at level $N - 3$. This process climbs up the tree, until a node, say $w = v(r, s)$, belonging to one of the k uppermost levels of B, is reached. Notice that w is directly colored in the first phase of BASIC-COLOR and node z inherits w's color. Thus, to retrieve the color of z we need to discover w and this takes $O(N - k)$ time.

Let us also point out that retrieval cost can be reduced to constant time if an appropriate preprocessing step, called PREBASIC-COLOR, is performed to build a table UP of size $O(2^N)$. For each node z of B, UP$[z]$ stores either the special mark * if z belongs to one of the first k levels of the tree or it is colored with a color from the list Γ (i.e., if z is the last node in a block); otherwise, it stores the node w of B which v inherits its color from. Once UP is available, each single node of B is colored as in Fig. 4.

In order to prove that the BASIC-COLOR algorithm is $(N + K - k)$-CF on $S(K)$ and $P(N)$, we will consider a new family of subsets of B, that contains both $S(K)$ and $P(N)$, and prove that BASIC-COLOR is $(N + K - k)$-CF on this larger family too.

For each node $v(i, j)$ of B, we define the subset $TP_K(i, j)$ as consisting of the nodes lying on the path from the root of B to $v(i, j)$ and the nodes of the complete subtree of size K rooted at $v(i, j)$. (Note that if $j > N - k$, subtree rooted at v(i,j) has size smaller than K.) We define the family $TP(K, j)$ as

$$TP(K, j) = \{TP_K(i, j - 1) : 0 \leq i < 2^{j-1}\}.$$

Lemma 1. *The BASIC-COLOR algorithm yields mappings of binary trees of height N that are $(N + K - k)$-CF on $TP(K, j)$, for each $j \leq N$.*

Proof. In order to prove the lemma, we will show that, for each $i = 0, 1, \cdots, 2^{j-1}$, no two nodes of $TP_K(i, j)$ are colored with the same color.

The proof is by induction on the length of the path segment of the subset TP. The basis holds trivially. In fact, $TP(K, 1)$ contains only the subset $TP_K(0, 0) = S_K(0, 0)$ that is colored in the first phase of the BASIC-COLOR algorithm, assigning to each node a different color of $\sum$. Since all the colors of $\sum$ are distinct, it follows that $TP_K(0, 0)$ is conflict-free. Now, suppose that all subsets in $TP_K(i, j - 1)$ are conflict-free, for $j > 1$. We will prove that, for each $i = 0, 1, \cdots, 2^j - 1$, the subset $TP_K(i, j)$ is also conflict-free.

Observe that, if $j > N - k$ then

$$TP_K(i, j) \subseteq TP_K(\lfloor i/2 \rfloor, j - 1)$$

and, by inductive hypothesis, it is conflict-free. On the other hand, if $j \leq N - k$, we can partition $TP_K(i, j)$ into two parts (see Fig. 5) as follows: 1) a bottom part T_b, containing all the leaves of the subtree of size K rooted at $v(i, j)$ and 2) a top (upper) part T_u, containing the remaining nodes. In order to prove that $TP_K(i, j)$ is conflict-free, we will show that both T_b and T_u are conflict-free and colors used for T_b are different from those used to color T_u.

T_u is conflict-free since $T_u \subseteq TP_K(\lfloor i/2 \rfloor, j - 1)$ and, by inductive hypothesis, this subset is conflict-free.

Observe that $T_b = block(i, j + k - 1)$ and the ith node of the block is colored with either the color of a node in $S_K(i', j)$ (Step 7) where $v(i', j)$ is the sibling of $v(i, j)$, or with a color of Γ (Step 8), when $i = 2^{k-1} - 1$.

Thus, the algorithm defines a one-to-one correspondence between the colors of the first $2^{k-1} - 1$ nodes of T_b and the colors of the first $k - 1$ levels of $S_K(i', j)$. Since the first $k - 1$ levels of $S_K(i', j)$ are contained in $TP_K(\lfloor i/2 \rfloor, j - 1)$, by inductive hypothesis it follows that

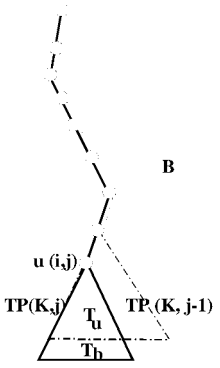

Fig. 5. Description of TP template.

there is no conflict between the first $2^{k-1}-1$ nodes of T_b. Moreover, these colors are different from the colors used in T_u. Finally, the last node of T_b is colored with a color of Γ that was never used before. Since Γ contains all distinct colors, this new color is different from all the other colors assigned to nodes of $TP(i,j)$. □

Theorem 1. *The* BASIC-COLOR *algorithm yields mappings of binary trees of height N that are $(N+K-k)$-*CF *on $S(K)$ and $P(N)$.*

Proof. We observe that $P(N) \in TP(K,N)$ and, for each $S_K(i,j) \in S(K)$, $S_K(i,j) \subseteq TP_K(i,j)$. Thus, by Lemma 1 we have that the mapping is $(N+K-k)$-CF on $P(N)$ and $S(K)$. □

Theorem 2. *Any mapping of binary trees of height N that is M-CF on $S(K)$ and $P(N)$ requires $M \geq N+K-k$ colors. Then, the* BASIC-COLOR *algorithm is $(N+K-k)$-*CF-optimal.

Proof. Observe that all subsets included in $TP(K,N-k)$ have size equal to $N+K-k$. Thus, any mapping algorithm that is M-CF on $TP(K,N-k)$ must use $M \geq N+K-k$ memory modules. We prove the theorem by showing that each mapping algorithm that is M-CF on $S(K)$ and $P(N)$ is M-CF on $TP(K,N-k)$.

Let U be an algorithm for mapping binary trees that is M-CF on $S(K)$ and $P(N)$ and let $Z \in TP(K,N-k)$. For each pair of nodes v_1 and v_2 in Z, we prove that they are colored with different colors by U. We distinguish two cases, depending on the positions of v_1 and v_2. If v_1 and v_2 belong to the tree part of Z, then they are colored differently, since U is M-CF on $S(K)$. If, instead, at least one of the two vertices, say v_1, does not belong to the tree part of Z, then there exists a path in T of length N that contains both v_1 and v_2. Since U is M-CF on $P(N)$, then v_1 and v_2 have different colors. □

Theorem 1 proves that BASIC-COLOR is M-CF on $S(K)$ and $P(N)$. A similar result does not hold for accessing L-templates. However, we can prove that, using the same number of memory modules, the number of conflicts on $L(K)$ is at most 1.

Lemma 2. *The* BASIC-COLOR *algorithm yields mappings of binary trees of height N on $(N+K-k)$ memory modules that have cost at most 1 on $L(K)$.*

Proof. Let $L \in L(K)$ be a set of K consecutive nodes of LEV(j). It can be easily seen that the mapping cost on L yielded by the BASIC-COLOR algorithm is at most 2. In fact, L spans at most three consecutive blocks of LEV(j). Each block contains all the leaves of a subtree of $S(K)$ and, by Theorem 1, it is conflict-free. Thus, the number of nodes with the same color in L is upper bounded by the number of blocks spanned by L.

In the rest of the proof, we show how to improve this result and prove that each color is used to color at most two nodes of L. In particular, we show that the color taken from Γ to color the last node of a block is the unique color that is used in all the three blocks. Since L has size K, then either L contains nodes of only two blocks or it does not contain the last node of the third block. Thus, the number of nodes of L with the same color is 2 (and, hence, there is only one conflict).

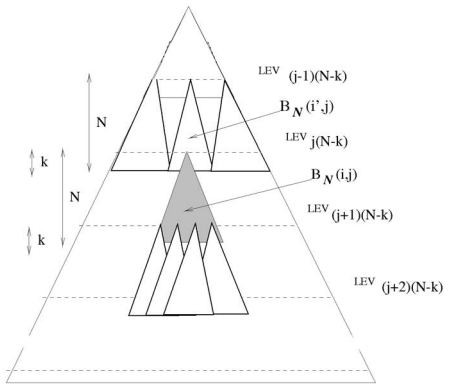

Fig. 6. Description of the template family $\mathcal{B}(N)$.

Assume that the first node of L belongs to $block(h,j)$ and let v_0, v_1, v_2 be the $(k-1)$st ancestors of the nodes of $block(h,j)$, $block(h+1,j)$ and $block(h+2,j)$, respectively. We observe that one among v_0 and v_2 (say v_0) is the sibling of v_1. We prove that the unique pair of nodes of $block(h,j) \cup block(h+1,j)$ that is colored with the same color is the pair containing the last nodes of the two blocks. Since this pair is colored with a color taken from Γ that is not used to color the first $2^{k-1}-1$ nodes of $block(h+2,j)$, we obtain that the number of conflicts for the template L is 1.

The BASIC-COLOR algorithm colors $block(h,j)$ with the colors used in the first $k-1$ levels of the subtree rooted at v_1, and $block(h+1,j)$ with the colors used in the first $k-1$ levels of the subtree rooted at v_0. Since there exists a tree in $S(K)$ that contains all the nodes of the first $k-1$ levels of the trees rooted at v_0 and v_1, by Theorem 1, there is no conflict on the first $2^{k-1}-1$ nodes of $block(h,j)$ and $block(h+1,j)$. □

3.2 Algorithm Color

In the previous section, we presented an algorithm to map binary trees of height N that is $(N+K-k)$-CF-optimal on $S(K)$ and $P(N)$. In the following, we show how this algorithm can be used as a building block for an algorithm to color binary trees of any height that is $(N+K-k)$-CF-optimal on $S(K)$ and $P(N)$.

Let T be a binary tree of height $H = h(N-k)+N$, for $h \geq 1$ (if $H \neq h(N-k)+N$, dummy levels are added). We split T into a family of (not necessarily disjoint) subtrees of height N and color each subtree as described in the previous subsection.

Consider the family of subtrees

$$\mathcal{B}(N) = \{S_{2^N-1}(i, j(N-k)) | 0 \leq j \leq h, 0 \leq i < 2^{j(N-k)}\}.$$

For brevity, let us denote the subtree $S_{2^N-1}(i, j(N-k))$ as $B_N(i,j)$. Notice that $\mathcal{B}(N)$ is not a partition of T (see Fig. 6). In particular, $B_N(i,j)$ has nonempty intersection with $2^{N-k}+1$ distinct subtrees of the family: It shares its first k levels with $B(i', j-1)$, where $i' = \lfloor i 2^{-(N-k)} \rfloor, j-1)$,

```
COLOR(T, N, K)
1  Consider the family B(N) = {B_N(i,j)}, where B_N(i,j) = S_{2^N-1}(i, j(N-k))
2  BASIC-COLOR(B_N(0,0), K, N)
3  for each j = 1 to ⌊H/(N-k)⌋
4  do
5     for each i = 0 to 2^{j(N-k)} - 1
6     do
7        Let Γ(i,j) be the list, in top-down order, of colors used for nodes lying
         on the path from the root of B_N(⌊i2^{-(N-k)}⌋, j-1) to the root of B_N(i,j)
8        BOTTOM(B_N(i,j), N, K, Γ(i,j))
```

Fig. 7. Algorithm COLOR.

and a subtree of height k with $B_N(h, j+1)$, for each $i \cdot 2^{N-k} \leq h < (i+1) \cdot 2^{N-k}$.

The algorithm COLOR(T, N, K), presented in Fig. 7, first constructs the family $\mathcal{B}(N)$ and then color its subtrees in a top down order, starting with $B_N(0,0)$ that is colored by calling the algorithm BASIC-COLOR$(B(0,0), N, K)$. Notice that when the algorithm starts coloring $B_N(i,j)$, for $j > 0$, the first k levels of this subtree have been already colored and only the bottom part of the tree needs to be colored. For this purpose, we use the algorithm BOTTOM$(B_N(i,j), N, K, \Gamma(i,j))$, where $\Gamma(i,j)$ is defined as the list of colors assigned to the nodes of $P_{N-k}(i,j)$, that is simply the path from the node $v_{\lfloor i2^{-(N-k)} \rfloor, j-1}$ to $v_{i,j}$, taken in a top down order.

Theorem 3. *Algorithm* COLOR *yields mappings of binary trees that are* $(N+K-k)$-CF-optimal *on* $S(K)$ *and* $P(N)$.

Proof. Consider first the template $S(K)$. Observe that, for each $S \in S(K)$, there exists a subtree $B \in \mathcal{B}(N)$ such that $S \subseteq B$. By Theorem 1, the B's coloring is $(N+K-k)$-CF on the template $S_B(K)$. Therefore, the mapping of COLOR is $(N+K-k)$-CF-optimal on $S(K)$.

Consider, now, the template $P(N)$ and let P be an instance of this template. It can be easily seen that there exist two not-disjoint subtrees $B_1, B_2 \in \mathcal{B}(N)$ such that all the nodes of P belong to B_1, B_2 or both. Without loss of generality, assume that the root of B_1 is an ancestor of the root of B_2.

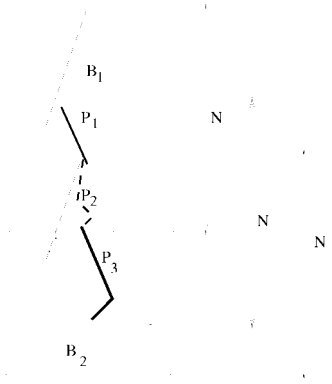

Fig. 8. Description of the splitting of P.

We split P into three parts: $P_1 \in B_1$, $P_2 \in B_1 \cap B_2$, and $P_3 \in B_2$ (see Fig. 8). By Theorem 1, the coloring of B_1 (respectively, B_2) is $(N+K-k)$-CF on $P_{B_1}(N)$ (respectively, $P_{B_2}(N)$). Thus, each of P_i, for $1 \leq i \leq 3$, is conflict-free and the colors assigned to the nodes of P_2 are distinct from those assigned to nodes of P_1 and P_3. It still remains to prove that there are no two nodes in P_1 and P_3 having the same color.

Observe that algorithm BOTTOM colors the last $N-k$ levels of B_2 using colors assigned to the nodes of the first k levels of B_2 and colors taken from a list $\Gamma_{(i,j)}$, consisting of the colors assigned to the nodes on the path (in top-down order) from the root of B_1 to the root of B_2. By Lemma 1, all the colors used in the first k levels of B_2 are different from those assigned to P_1. On the other hand, the hth color of $\Gamma_{(i,j)}$ is used only on jth level of B_2, for $j \geq k+h-1$. Thus, the nodes of P_1 can be colored only with the last $|P_1|$ colors of $\Gamma_{(i,j)}$, while nodes of P_3 are colored with the first $|P_3|$ colors of $\Gamma_{(i,j)}$. Since $|P_1| + |P_3| = N - k = |\Gamma_{(i,j)}|$, it follows that none of the colors assigned to the nodes of P_3 is used in P_1. Hence, the mapping is $(N+K-k)$-CF on $P(N)$. Optimality follows from Theorem 2. □

The COLOR algorithm requires $O(2^H)$ time, while $O(H)$ time is required to retrieve the memory module where a single node of T is mapped. In fact, BASIC-COLOR assigns to $v(r,s)$ the same color as one of the nodes of the uppermost k levels of subtree $B_N(i,j) \in \mathcal{B}(N)$ which $v(r,s)$ belongs to. However, the coloring of the uppermost k levels of $B_N(i,j)$ depends on the coloring of the tree $B_N(i', j-1)$ that shares the first k levels with $B_N(i,j)$.

This process climbs up the tree until subtree $B_N(0,0)$ is reached. Then, the algorithm must compute the coloring of $O(H/N-k)$ trees on the path from $B_N(i,j)$ (where $v(r,s)$ is) up to $B_N(0,0)$. By the analysis of BASIC-COLOR, each step costs $O(N-k)$ time and, therefore, the time for retrieving the memory module where $v(r,s)$ is mapped is $O(H)$.

Similarly to previous section, the retrieval time can be reduced to $O(H/N-k)$ if the PRE-COLOR algorithm is performed with a new preprocessing step, called PRE-COLOR algorithm.

```
RETRIEVING-COLOR(v(i,j), UP, NEW)
1  // Find the color assigned to the node v(i,j) by COLOR
2  if j < N
3     then return RETRIEVING-BASIC-COLOR(B(0,0), UP)
4  if UP[v(i,j)] = *
5     then return RETRIEVING-COLOR(ANC_B(i,j,N), UP, NEW)
6  if UP[v(i,j)] = v(m,n)
7     then return RETRIEVING-COLOR(NEW[v(m,n)],UP,NEW)
```

Fig. 9. Algorithm RETRIEVING-COLOR.

PRE-COLOR builds a table NEW of size $O(K)$ that stores, for each node $v(r,s)$ of the k uppermost levels of any $B_N(i,j)$, its relative address in the subtree $B_N(i',j-1)$ that shares the first k levels with $B_N(i,j)$. Notice that NEW$[v(r,s)]$ is a function of both the relative position of $v(r,s)$ in $B_N(i,j)$ and the root $v(i,j)$ of $B_N(i,j)$.

Formally, assuming that both preprocessing steps PRE-BASIC-COLOR and PRE-COLOR have been performed, the recursive RETRIEVING-COLOR algorithm can be described as in Fig. 9.

4 An Optimal Mapping for Elementary Templates

By Theorem 3, it is possible to obtain conflict-free access to $S(M)$ and $P(M)$ if and only if the memory system has at least $2M - \lceil \log M \rceil$ memory modules, i.e., the degree of memory parallelism is greater than the size of the subset to be accessed in parallel.

This section restricts our attention to mappings that exploit all the parallelism of the memory system. We prove that if M colors are available, letting $m = \lceil \log M \rceil$, COLOR$(T, 2^{m-1}-1, 2^{m-1}+m-1)$ yields a mapping that has unit cost on $S(M)$ and $P(M)$, and thus it is M-optimal on these set of templates. In the remainder of this section, COLOR stands for COLOR$(T, 2^{m-1}-1, 2^{m-1}+m-1)$.

Theorem 4. *For each integer $M = 2^m - 1$, the* COLOR *algorithm yields mappings of binary trees on M memory modules such that*

$$\max\{Cost(\text{COLOR}, S(M), M), \\ Cost(\text{COLOR}, P(M), M)\} = 1.$$

Proof. We evaluate separately the cost of the mapping on $S(M)$ and $P(M)$, and prove that in both the cases the cost is at most 1.

Let $P \in P(M)$. By Theorem 3, COLOR colors with no conflicts each path of length $\leq 2^{m-1}+m-1$. Since $M/2 < 2^{m-1}+m-1$, we can split P into two segments of size $\leq M/2$ such that each of them is conflict-free. However, the same color could be used to color a node in each of the two segments. Thus, there are at most two nodes in S colored with the same color and

$$Cost(\text{COLOR}, P(M), M) \leq 1.$$

Let $S \in S(M)$. By Theorem 3, the first $m-1$ levels of S are colored by COLOR with no conflicts. On the other hand, the last level of S contains the nodes of two consecutive blocks that are colored with $\lfloor \frac{M}{2} \rfloor - 1$ colors already used in the previous levels of S and a new color. Thus, there are at most two nodes in S colored with the same color and

$$Cost(\text{COLOR}, S(M), M) = 1.$$

□

Since there exists no algorithm [2] for mapping of binary trees that is M-CF on $\{S(M), P(M)\}$, we can state that

Theorem 5. *The algorithm COLOR is M-optimal on $\mathcal{I} = \{S(M), P(M)\}$.*

In Section 6, we introduce algorithm LABEL-TREE which exhibits constant time retrieval cost at the cost of a slightly larger number of memory modules. Intuitively, LABEL-TREE beats COLOR since it colors T by a family of disjoint subtrees.

5 Accessing Trees by Composite Templates

Let us study the efficiency of COLOR with respect to composite templates (C-templates) introduced in Section 2.

For this purpose, we first consider the cost of the mapping yielded by COLOR$(T, 2^{m-1}-1, 2^{m-1}+m-1)$ on elementary templates of any size D; then, we combine these results for deriving the cost of mapping instances of the composite templates $C(D,c)$, where D is the size of the template and c is the number of constituent elementary templates. As earlier, we set

$$K = 2^{m-1} - 1,$$
$$N = 2^{m-1} + m - 1,$$
$$M = 2^m - 1,$$

and we simply say COLOR for

$$\text{COLOR}(T, 2^{m-1}-1, 2^{m-1}+m-1).$$

For sake of simplicity, we assume in the rest of the paper that the parallel memory system consists of $M = 2^m - 1$ modules. In the general case, all results presented in this and next section still hold, but the number of conflicts increases by a constant factor.

Lemma 3. *For each integer $D \geq M$, the* COLOR *algorithm yields mappings of binary trees of height $H > D$ on a memory system consisting of $M = 2^m - 1$ modules such that*

$$Cost(\text{COLOR}, P(D), M) \leq 2\left\lceil \frac{D}{M} \right\rceil - 1.$$

Proof. Consider $P \in P(D)$ and split it into segments $P_1, P_2, \cdots, P_{\lceil \frac{D}{M} \rceil}$ in such a way that each segment, except for the last one, has length M.

By Theorem 4, each segment of P contains at most two nodes with the same color. Therefore, the cost of the mapping on P is at most $2\lceil \frac{D}{M} \rceil - 1$. □

Lemma 4. *For each integer $D \geq M$, the* COLOR *algorithm yields mappings of binary trees of size larger than $2D - 1$ on a memory system with M modules such that*

$$Cost(\text{COLOR}, L(D), M) \leq 4\left\lceil \frac{D}{M} \right\rceil.$$

Proof. Let $L \in L(D)$. Recall that the algorithm splits each level of the tree into blocks of size 2^{m-2} and color each block without conflicts. Thus, the number of conflicts in L is upper bounded by the number of blocks that contain nodes of L. Since L spans at most $\lceil \frac{D}{2^{m-2}} \rceil + 1$ consecutive blocks, the number of conflicts on L is at most

$$\left\lceil \frac{D}{2^{m-2}} \right\rceil \leq \left\lceil \frac{4D}{M} \right\rceil \leq 4\left\lceil \frac{D}{M} \right\rceil.$$

□

Lemma 5. *For each integer $D = 2^d - 1$, where $d \geq m$, the* COLOR *algorithm yields mappings of binary trees of size greater than D on a memory system with M modules such that*

$$Cost(\text{COLOR}, S(D), M) \leq 4\left\lceil \frac{D}{M} \right\rceil - 1.$$

Proof. Let $d = hm + i$ be the height of a tree $S \in S(D)$, for some $h > 0$ and $i < m$. We partition S into $i + 1$ subsets $S_0, S_1, \cdots, S_i$ such that S_0 consists of the first hm levels of S and S_j consists of the nodes of level $hm + j$, where $1 \leq j \leq i$. Observe that S_0 can be partitioned into l subtrees of size M, where $l = \frac{2^{hm}-1}{M}$. By Theorem 4, each of these trees has at most two nodes colored with the same color. Thus, the number of nodes of S_0 with the same color is at most $2l$.

Moreover, the number of nodes in S_j with the same color is upper bounded by $\frac{2^j}{2^{m-2}}$, that is the number of blocks at level $hm + j$. Summing over all S_j, the number of conflicts in S is at most

$$\begin{aligned}
2l + \sum_{j=1}^{i} \frac{2^{hm+j-1}}{2^{m-2}} - 1 &= 2l + \frac{2^{hm}}{2^{m-2}}(2^i - 1) - 1 \\
&= 2\left(\frac{2^{d-i} - 1}{M}\right) + \frac{2^{d-i}}{2^{m-2}}(2^i - 1) - 1 \\
&< 2\left(\frac{2^{d-i}}{M}\right) + 4\left(\frac{2^{d-i}}{M}\right)(2^i - 1) - 1 \\
&= 4\left(\frac{2^d}{M}\right) - 2\left(\frac{2^{d-i}}{M}\right) - 1 \\
&= 4\left(\frac{2^d + 1 - 1}{M}\right) - 2\left(\frac{2^{d-i}}{M}\right) - 1 \\
&= 4\left(\frac{D}{M}\right) - \left(\frac{1}{M} + 2\frac{2^{d-i}}{M}\right) - 1 \\
&< 4\left\lceil \frac{D}{M} \right\rceil - 1.
\end{aligned}$$

□

Theorem 6. *For each pair of integers $D \geq M$ and $c > 0$, the algorithm* COLOR *yields mappings of binary trees of size greater than D on a memory system with M modules such that the cost of accessing composite templates is given by*

$$Cost(\text{COLOR}, C(D, c), M) \leq 4\left(\frac{D}{M}\right) + c.$$

Proof. Consider a composite template instance $C \in C(D, c)$ and let $C_1, C_2, \cdots, C_c$ be disjoint instances of elementary templates that form C. By Lemmas 3, 4, and 5, the mapping of C_i has a cost no greater than $4\left\lceil \frac{|C_i|}{M} \right\rceil$, for each i. Thus, the cost of the mapping on C is at most

$$\sum_i 4\left\lceil \frac{|C_i|}{M} \right\rceil < \sum_i \left(4\frac{|C_i|}{M} + 1\right) = 4\left(\frac{D}{M}\right) + c.$$

□

For $c = O(\frac{D}{M})$, the algorithm

$$\text{COLOR}(T, 2^{m-1} - 1, 2^{m-1} + m - 1)$$

is M-optimal, within a constant factor, on C-templates.

6 Faster Algorithm for Accessing Composite Templates

The COLOR algorithm is intrinsically recursive in nature and the color of a node depends on the previously colored nodes. However, as the RETRIEVING-COLOR procedure shows, one can trade-off time with memory space and, in this case, reduce the time to compute the address of a node to $O(H/N)$.

In this section, we describe a different mapping algorithm for accessing complete trees through composite templates that distributes the nodes among the memory modules in a very balanced way and also allows retrieval of node addresses in constant time with a moderate size table. In [2], it is shown that LABEL-TREE maps complete binary

```
MICRO-LABEL(B, Σ)
 1  for each j = 0 to l − 1
 2  do
 3      for each i = 0 to 2^j − 1
 4      do
 5          color u(i, j) with (2^j − 1 + i)th color of Σ.
 6  for each j = l to m − 1
 7  do
 8      for each h = 0 to 2^{j−l+1} − 1
 9      do
10          let b_0, b_1, ···, b_{2^{l−1}−1} be the nodes of block(h, j)
11          let u(h', j − l + 1) be the sibling of u(h, j − l + 1)
12          color b_i with the color assigned to the ith node of S_{2^{l−1}−1}(h', j − l + 1)
13          color b_{2^{l−1}−1} with the (2^l + 2^{j−l} + ⌊h/2⌋ − 1)th color of Σ.
```

Fig. 10. Algorithm MICRO-LABEL.

trees on M memory modules with the following performances:

Theorem 7 [2]. *Algorithm LABEL-TREE yields mappings of binary trees on M memory modules such that:*

- *there are $O\left(\sqrt{\frac{M}{\log M}}\right)$ conflicts on all instances of elementary templates of size M;*
- *the memory load is $1 + o(1)$;*
- *for each node of the tree, the address of the module where this node is mapped can be computed in $O(1)$ time if an $O(M)$ size table is precomputed, otherwise it takes $O(\log M)$ time.*

In this section, we extend the analysis in [2] and show that the algorithm LABEL-TREE scales well when the size of the templates grows. In particular, we prove that for each integer D, LABEL-TREE has cost $O\left(\frac{D}{\sqrt{M \log M}}\right)$ on instances of elementary templates of size D and, hence, a cost

$$O\left(\frac{D}{\sqrt{M \log M}} + c\right)$$

on composite templates $C(D, c)$. These results show an interesting trade-off between the number of conflicts and the simplicity of node-addressing, and balancing the memory load.

In the following, we first give an overview of the algorithm LABEL-TREE [2] for completeness, and then compute its cost on C-templates.

6.1 Algorithm LABEL-TREE

The basic idea of the algorithm LABEL-TREE is to divide the tree in a set of disjoint subtrees of height $m = \lceil \log M \rceil$ and color each subtree independently so that no conflict occurs on accessing paths of each subtree and a few conflicts occurs on levels. The set of colors is split in groups and then the following three steps are followed for each subtree B.

- **MACRO-LABEL.** Assign a group of colors to B such that if the roots of two subtrees lie on the same ascending path and are assigned the same group, then their distance is at least $\Omega(\sqrt{M \log M})$. The goal is to reduce conflicts for P-templates.
- **ROTATE.** Select a list of ℓ colors from among the colors of the group assigned to B in such a way that two subtrees at the same level with the same list of colors are as far as possible. Goals are 1) to reduce conflicts for L- and S-templates, 2) balance the load on memory modules.
- **MICRO-LABEL.** Color the nodes of B using the colors of the list assigned to B. It yields mappings of binary trees of height m that are ℓ-conflict-free (ℓ defined right below) on the path template $P(m)$ and have cost $O\left(\sqrt{\frac{M}{\log M}}\right)$ on the subtree template $S(M)$.

Since MICRO-LABEL is the core of our coloring strategy a more detailed description is needed. Let

$$l = \left\lfloor \log \left\lceil \sqrt{M \lceil \log M \rceil} \right\rceil \right\rfloor,$$

$\ell = 2^l + 2^{m-l} - 2$, and $p = \lfloor \frac{M}{\ell} \rfloor$. We partition the color set $\{0, 1, \cdots, M-1\}$ into p subsets $G_0, G_1, \ldots, G_{p-1}$, where $|G_j| = \left\lfloor \frac{M}{p} \right\rfloor$ or $\left\lfloor \frac{M}{p} \right\rfloor + 1$.

MICRO-LABEL is similar to BASIC-COLOR, presented in Section 3, but it uses a different number of colors. In fact, MICRO-LABEL uses more colors and is tailored to guarantee conflict-free access to $S(2^l - 1)$. Whereas, BASIC-COLOR uses less colors and guarantees conflict-free access to $S(M)$. Notice that, here, we are trading the number of colors and the number of conflicts with the balancing of loads on the elements of the parallel memory system.

Let B be a subtree of the forest $\mathcal{B}$ and let $\sum = \{f_0, \ldots, f_{\ell-1}\}$ be the list of colors assigned to B by the algorithm ROTATE. Algorithm MICRO-LABEL$(B, \sum)$, shown in Fig. 10, consists of two steps. In the first step it assigns a distinct color to each of the nodes of the l top levels of B; in the second step it colors the remaining levels in a top-down order, on a block basis (e.g. here the blocks have size 2^{l-1}). The coloring of the nodes of a block is similar to the coloring of BASIC-COLOR. Let $block(h, j)$ be the block that has to be colored, with $j \geq l$, and let $u(h, j - l + 1)$ and $u(h', j - l + 1)$ be the $(l-1)$th ancestor of the nodes in $block(h, j)$ and its sibling, respectively. As in

BASIC-COLOR we color $(2^{l-1}-1)$ nodes of $block(h,j)$ with the colors of the subtree rooted at $u(h', j-l+1)$ and the last node with the $(2^l + 2^{j-l} + \lfloor h/2 \rfloor - 1)$th color of $\sum$, that was never used in the previous levels of the tree.

Observe that the procedure MICRO-LABEL uses only colors of $\sum$ to color B. In fact, the largest index of a color taken from $\sum$ is $2^l + 2^{m-l} - 2 = \ell - 1$, for $j = m-1$ and $h = 2^{m-l} - 1$.

It can be proved that MICRO-LABEL gives a mapping of the subtree B that is ℓ-conflict-free on templates $P(m)$ and $S(2^l-1)$ (the proof is similar to that of Theorem 1). Moreover, the following property holds for the number of conflicts on the nodes of a level of B.

Lemma 6. *For each $D = 2^d$, with $d \geq l$,* MICRO-LABEL *yields mappings of binary trees such that*

$$Cost(\text{MICRO-LABEL}, L(D), \ell) = O\left(\frac{D}{\sqrt{M \log M}}\right).$$

Proof. We observe that MICRO-LABEL maps nodes of each block without conflicts. Thus the number of conflicts on each instance $L \in L(D)$, the level template, is upper bounded by the number of blocks spanned by L. □

6.2 Cost Analysis

The problem of computing the number of conflicts produced by LABEL-TREE on instances of elementary templates of size M has been studied in [2]. In this section, we extend the analysis to instances of the composite templates. Before proceeding further, we study the performance of the algorithm on instances of elementary templates of any size D.

Lemma 7. *For each integer D, it holds that*

1. $Cost(\text{LABEL-TREE}, L(D), M) = O\left(\frac{D}{\sqrt{M \log M}}\right)$.
2. $Cost(\text{LABEL-TREE}, P(D), M) \leq \left\lceil \frac{D}{\sqrt{M \log M}} \right\rceil + 1$.
3. $Cost(\text{LABEL-TREE}, S(D), M) = O\left(\frac{D}{\sqrt{M \log M}}\right)$.

Proof. The result is a nontrivial extension of similar results presented in [2]. Here, we prove only the first statement above and leave the others to the reader.

Let $L \in L(D)$ and assume that nodes of L belong to the hth level of B_j. Thus, there are at most $\lceil \frac{D}{2^h} \rceil + 1$ trees of B_j that contain nodes of L. As for the previous lemma, we divide L in segments, each consisting of the nodes contained in a tree of B_j.

We distinguish two cases, depending on the value of h. If $h \geq l$, then by Lemma 6, the algorithm MICRO-LABEL colors each segment of L with $O\left(\frac{2^h}{\sqrt{M \log M}}\right)$ conflicts. Thus, the number of conflicts on L is at most

$$\left(\left\lceil \frac{D}{2^h} \right\rceil + 1\right)\left(\frac{2^h}{\sqrt{M\lceil \log M \rceil}}\right) = O\left(\frac{D}{\sqrt{M \log M}}\right).$$

On the other hand, if $h < l$, a more accurate analysis occurs. In fact, for small h, the number of segments of L is greater than D. Observe that algorithms MICRO-LABEL and ROTATE assign different lists of colors to trees of B_j that are close. In particular, if B and B' are two consecutive trees of B_j that are assigned the same color group, ROTATE assigns lists of colors to the trees such that $\text{list}(B) = \{f_0, f_1, \cdots, f_{\ell-1}\}$ and $\text{list}(B') = \{f_1, f_2, \cdots, f_\ell\}$. Thus, if f is the color assigned to the last node of the first segment of L, then f is used in the first 2^h segments of L and afterwards it is never used until segment $\ell + 1$.

In general, if we group the segments of L into sets of ℓ consecutive segments, the number of conflicts in each set being at most 2^h. Thus, the total number of conflicts is

$$\left(\left\lceil \frac{D}{2^h} \right\rceil + 1\right)\left(\frac{2^h}{\ell}\right) = O\left(\frac{D}{\sqrt{M \log M}}\right).$$

□

Theorem 8. *For each pair of positive integers D and c, algorithm* LABEL-TREE *yields mappings of binary trees on M memory modules such that*

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

Proof. Let $C \in C(D,c)$ be a composite template, and let $C_1, C_2, \cdots, C_c$ be the components of C. Recall that each C_i is an instance of an elementary template of size at most D.

By Lemma 7, each component C_i is colored by LABEL-TREE so that the number of conflicts on C_i is $O\left(\left\lceil \frac{|C_i|}{\sqrt{M \log M}} \right\rceil\right)$. Thus, the number of conflicts on C is at most

$$\sum_{i=1}^{c} O\left(\left\lceil \frac{|C_i|}{\sqrt{M \log M}} \right\rceil\right) = O\left(\frac{D}{\sqrt{M \log M}} + c\right).$$

□

7 Conclusions

In this paper, we have presented several strategies for mapping complete binary tree data structures onto parallel memory systems and evaluated each strategy with respect to three criteria: 1) the number of memory conflicts that can occur when a parallel access to the data structure is performed, 2) the number of elements that can be accessed in parallel, and 3) the complexity of the addressing scheme. We have also shown that interesting trade-offs exist between these criteria. In fact, it is not possible to guarantee conflict-free access to both subtree and path templates of size equal to the number of memory modules. On the other hand, it is possible to have conflict-free access to subtrees and paths of size K if there are at least $2K - \log K$ memory modules.

Moreover, when considering parallel access to templates obtained by composition of subtrees, paths and levels, we have shown that there exists a trade-off between the number of conflicts and the complexity of the addressing scheme, measured in terms of the time needed by a processor to compute the address of a node in the memory.

ACKNOWLEDGMENTS

The authors would like to acknowledge the editor and the anonymous referees for their valuable comments that contributed to greatly improve the paper. Preliminary versions of this paper appeared in [1], [2]. This work has been partially supported by Progetto MURST 40% Algoritmi, Modelli di Calcolo e Strutture Informative, Texas Advanced Research grant TARP-003594-013 and Texas Advanced Technology grant TATP-003494-031.

REFERENCES

[1] V. Auletta, S.K. Das, A. De Vivo, M.C. Pinotti, and V. Scarano, "Towards a Universal Mapping Algorithms for Accessing Trees in Parallel Memory Systems," *Proc. Int'l Parallel Processing Symp. (IPPS/SPDP),* pp. 447-459, Apr. 1998.

[2] V. Auletta, S.K. Das, M.C. Pinotti, and V. Scarano, "Optimal Tree Access by Elementary and Composite Templates in Parallel Memory Systems," *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS),* Apr. 2001.

[3] V. Auletta, A. De Vivo, and V. Scarano, "Multiple Template Access of Trees in Parallel Memory Systems," *J. Parallel and Distributed Computing,* vol. 49, np. 1, pp. 22-39, Feb. 1998.

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

[5] T. Cormen, C. Leiserson, and R. Rivest, *An Introduction to Algorithms.* Mit Press, 1990.

[6] R. Creutzburg and L. Andrews, "Recent Results on the Parallel Access to Tree-Like Data Structures—The Isotropic Approach," *Proc. of Int'l Conf. Parallel Processing,* vol. 1, pp. 369-372, 1991.

[7] S.K. Das and M.C. Pinotti, "Conflict-Free Template Access in *k*-Ary and Binomial Trees," *Proc. 11th ACM Int'l Conf. Supercomputing,* pp. 237-244, July 1997.

[8] S.K. Das and M.C. Pinotti, "Load Balanced Mapping of Data Structures in Parallel Memory Modules for Fast and Conflict-Free Templates Access," *Proc. Fifth Int'l Workshop Algorithms and Data Structures,* vol. 1272, pp. 272-281, Aug. 1997.

[9] 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," *J. Parallel and Distributed Computing,* vol. 60, no. 8, pp. 998-1027, 2000.

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

[11] S.K. Das and F. Sarkar, "Conflict-Free Data Access of Arrays and Trees in Parallel Memory Systems," *Proc. Sixth IEEE Symp. Parallel and Distributed Processing,* pp. 377-383, 1994.

[12] S.K. Das, F. Sarkar, and M.C. Pinotti, "Conflict-free Path Access of Trees in Parallel Memory Systems with Application to Distributed Heap Implementation," *Proc. 24th Int'l Conf. Parallel Processing,* vol. III, pp. 164-167, 1995.

[13] J. JaJa, *An Introduction to Parallel Algorithms.* Addison Wesley, 1992.

[14] F.T. Leighton, *Introduction to Parallel Algorithms and Architectures.* Morgan Kaufman, 1992.

[15] M.C. Pinotti and G. Pucci, "Parallel Algorithms for Priority Queue Operations," *Theoretical Computer Science,* vol. 148, pp. 171-180, 1995.

[16] D. Kaznachey, A. Jagota, and S. K. Das, "Primal-Target Neural Net Heuristics for the Hypergraph k-Coloring Problem," *Proc. Int'l Conf. Neural Networks (ICNN '97),* pp. 1251-1255, June 1997.

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

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

Vincenzo Auletta received the Laurea degree (summa cum laude) in computer science from the University of Salerno, Italy, in 1987. He received the doctorate degree in applied mathematics and computer science from the University of Naples, Italy, in 1992. He is an associate professor of computer science in the Department of Computer Science of the University of Salerno. His research interests are in the areas of design and analysis of algorithms, distributed/parallel processing, network architectures and protocols, security.

Sajal K. Das received the BTech degree from Calcutta University, 1983, the MS degree from the Indian Institute of Science, Bangalore, and the PhD degree in 1988 from the University of Central Florida, Orlando, all in computer science. Currently he is a full professor of computer science and engineering and also the founding director of the Center for Research in Wireless Mobility and Networking (CReWMaN) at the University of Texas at Arlington (UTA). Until 1999, he was a professor of computer science at the University of North Texas (UNT), Denton, where he founded the Center for Research in Wireless Computing (CReW) in 1997, and also served as the director of the Center for Research in Parallel and Distributed Computing (CRPDC) from 1995 to 1997. He is a recipient of the UNT Student Association's Honor Professor Award in 1991 and 1997 for best teaching and scholarly research, the UNT Developing Scholars Award in 1996 for outstanding research, and the UTA Outstanding Senior Faculty Research Award in Computer Science in 2001. He has visited numerous universities, research organizations, and industry research labs for collaborative research and invited seminar talks. He was a Visiting Scientist at the Council of National Research in Pisa, Italy, and Slovak Academy of Sciences in Bratislava, and was also a Visiting Professor at the Indian Statistical Institute, Calcutta. He is frequently invited as a keynote speaker at international conferences and symposia. His current research interests include resource and mobility management in wireless networks, mobile computing, QoS provisioning and wireless multimedia, mobile Internet, network architectures and protocols, distributed/parallel processing, performance modeling and simulation. He has published more than 185 research papers in these areas, directed several projects funded by industry and government, and filed four US patents in wireless moble networks. He is a member of the IEEE and the IEEE Computer Society. He received the Best Paper Awards for significant research contributions in the ACM Fifth International Conference on Mobile Computing and Networking (MobiCom'99), Third ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2000), and ACM/IEEE International Workshop on Parallel and Distributed Simulation (PADS '97). He serves on the editorial boards of the *Journal of Parallel and Distributed Computing*, *Parallel Processing Letters*, *Journal of Parallel Algorithms and Applications*, and *Computer Networks*. Each year he serves on numerous IEEE and ACM conferences as technical program committee member, program chair, or general chair. He is a member of the IEEE TCPP executive committee and advisory boards of several cutting-edge companies.

Amelia De Vivo received the degree in computer science from University of Salerno in 1993. In 1997, she achieved the masters degree in information technology from Istituto Internazionale per gli Asti Studi Scientifici (IIASS), Italy. In 1998, she joined Quadrics Supercomputers World, where she worked on a parallel compiler for the APE100 SIMD machine. Currently, she is working on the PhD degree in computer science at the University of Salerno. Her research interests include parallel compilers, parallel computing, and communication systems for high performance cluster networking.