c-Perfect Hashing Schemes for Arrays, with Applications to Parallel Memories

Gennaro Cordasco Alberto Negro Vittorio Scarano

Dipartimento di Informatica ed Applicazioni "R.M. Capocelli"

Università di Salerno, 84081, Baronissi (SA) – Italy

E-mail: {cordasco,alberto,vitsca}@dia.unisa.it

Arnold L. Rosenberg*

Dept. of Computer Science

University of Massachusetts at Amherst

Amherst, MA 01003, USA

E-mail: rsnbrg@cs.umass.edu

Abstract

We study the problem of mapping array-structured data to an ensemble of parallel memory modules, by allowing at most c conflicts (i.e. simultaneous access by up to c processors to the same memory module). We seek the smallest ensemble that will allow us to store *any* n-vertex instance of three array-like data structures with no more than c array-vertices stored on the same module. For the most general family, *chaotic* arrays, we prove that $\Theta(n^2/c^2)$ memory modules are needed to achieve this bound on conflicts. Similarly tight bounds are found for the other two families, *ragged* and *rectangular* arrays.

1 Introduction

Overview. We study the problem of mapping array-structured data onto a collection of memory modules that can be accessed concurrently by the processors of a parallel computing system (e.g., a multiprocessor or cluster). Motivated by application areas as diverse as relational databases, sparse matrix calculations, and computer vision (see, e.g., [15, 16, 17]), we interpret the qualifier "array-structured" quite liberally, as we shall see imminently.

Mapping array-structured data onto a collection of memory modules poses a challenge to an algorithm designer, because such parallel memory systems are typically single-ported: simultaneous accesses (i.e., conflicts) are queued up, hence incur delays. Consequently, the designer must allocate the arrays in a way that minimizes the number of conflicts for any possible set of

*Research supported in part by US NSF Grant CCR-00-73401.

n items accessed from the array. Of course, the mapping strategy to be used depends on the detailed structure of the arrays to be stored, as well as on how algorithms will access data items. For instance, mapping strategies for regular (e.g., row-wise or column-wise) access patterns in a rectangular array will be much simpler than strategies that accommodate irregular access patterns (which we call *templates*) in a nonrectangular array.

Related Work. Research in this field originated with strategies for mapping two-dimensional arrays into parallel memories. One can view Euler's "latin square" [11] as the first strategy for mapping a square matrix onto a parallel memory, where rows, columns and diagonals can be accessed without conflicts. Research has since moved on to seeking mapping algorithms that extend the "versatility" of latin squares by allowing conflict-free access for other templates, such as blocks and diagonals [19]. Several schemes have been proposed [3, 4, 10, 12, 14] that offer conflict-free access to several templates, including rows, columns, diagonals and submatrices. While the strategies in these sources provide conflict-free mappings, such was not their primary goal; they were actually designed to accommodate as many templates as possible.

There has also been work on conflict-free mappings for tree-structured data. Early research [9, 22] considered mainly conflict-free access for one "elementary" template—either complete subtrees or root-to-leaf paths or levels—at a time. Some work has also been done (e.g., [1]) following the cited array-related sources in seeking *maximally versatile* mapping strategies. Our recent paper [7] seeks conflict-free mappings for arbitrary n-node trees.

Similar objectives to ours are seen in work on parallel *secondary* memories [20, 21]. Their focus differs from ours because of the nature of the devices (block-access devices, as opposed to cell-access) as well as the emphasis on designing efficient algorithms rather than generic storage mappings.

Templates. In contrast to the cited earlier sources, we study templates that can be viewed as generalizing array-blocks and "paths" originating from the origin vertex $\langle 0,0\rangle$. We deal with the following three templates.

1. A (*two-dimensional*) *chaotic array* C is an undirected graph whose vertex-set V_C is a subset of[1] $N \times N$ that is *order-closed*, in the sense that for each vertex $\langle r,s\rangle \neq \langle 0,0\rangle$ of C, the set $V_C \cap \{\langle r-1,s\rangle, \langle r,s-1\rangle\} \neq \emptyset$. C's edges comprise all 2-element subsets of V_C of the form $(v,\ v+\alpha)$ where $\alpha \in \{\langle 0,1\rangle, \langle 1,0\rangle\}$.

2. A (*two-dimensional*) *ragged array* R is a chaotic array whose vertex-set V_R satisfies the following.

- if $\langle v_1, v_2\rangle \in V_R$, then $\{v_1\} \times [v_2] \subseteq V_R$;
- if $\langle v_1, 0\rangle \in V_R$, then $[v_1] \times \{0\} \subseteq V_R$.

[1] N denotes the nonnegative integers. For each $n \in N$, $[n]$ denotes the set $\{0, 1, \ldots, n-1\}$.

3. A (*two-dimensional*) *rectangular array* S is a ragged array whose vertex-set has the form $[a] \times [b]$ for some $a, b \in N$.

We denote by $\mathbf{C}_n$ (resp., $\mathbf{R}_n$) (resp., $\mathbf{S}_n$) the family of chaotic arrays (resp., ragged arrays, rectangular arrays) having n or fewer vertices.

2 Background

For $v = \langle v_1, v_2 \rangle \in N \times N$, we define $\Sigma(v) = v_1 + v_2$. The ith *diagonal* of an array[2] A is the set $D_i^{(A)} \stackrel{\text{def}}{=} \{v \in V_A \mid \Sigma(v) = i\}$. For any array A: $\text{Size}\,(A) = |V_A|$, and $\text{Size}\,(A, i) = |D_i^{(A)}|$; easily, $\text{Size}\,(A, i) \leq i + 1$.

For any integer $c > 0$, a *c-contraction* of an array A is a graph G that is obtained from A as follows.

1. Rename A as $G^{(0)}$. Set $k = 0$.
2. Pick a set S of $\leq c$ vertices of $G^{(k)}$ that were vertices of A. Replace these vertices by a single vertex v_S; replace all edges of $G^{(k)}$ that are incident to vertices of S by edges that are incident to v_S. The graph so obtained is $G^{(k+1)}$.
3. Iterate step 2 some number of times; G is the final $G^{(k)}$.

A graph $G_n = (V_n, E_n)$ is *c-perfect-universal* for the family $\mathbf{C}_n$ (resp., $\mathbf{R}_n$, $\mathbf{S}_n$) if every c-contraction of an n-vertex chaotic (resp., ragged, rectangular) array is a *labeled-subgraph* of G_n. In other words: Given any c-contraction $G^{(k)} = (V, E)$ of an n-vertex chaotic (resp., ragged, rectangular) array A, the fact that $G^{(k)}$ is a subgraph of G_n is observable via a mapping $f : V \to V_n$ for which each $v \in V$ is a subset of $f(V)$.

The notion of 1-perfect-universality (of more general graph families) was invented in [6] to unite the well-studied areas of universal graphs [2, 5] and perfect hashing [8], toward the end of studying efficient storage mappings for important families of data structures. A 1-perfect-universal graph for $\mathbf{C}_n$ (resp., $\mathbf{R}_n$, $\mathbf{S}_n$) can be viewed as a way to map each vertex of an chaotic (resp., ragged, rectangular) array $A \in \mathbf{C}_n$ (resp., $\mathbf{R}_n$, $\mathbf{S}_n$) to a distinct memory location, in such a way that: no two array-vertices share the same location; each path in A translates to a sequence of memory locations. A c-perfect-universal graph for $\mathbf{C}_n$ (resp., $\mathbf{R}_n$, $\mathbf{S}_n$) plays a similar role, with $\leq c$ vertices of A allowed to share the same memory location.

Remark. *Intuitively, the vertices of a c-perfect-universal graph represent the modules of a parallel memory. The preservation of vertex labels ensures simple address calculations.*

The simplest 1-perfect-universal graph for the families $\mathbf{C}_n$ and $\mathbf{R}_n$ is the ragged array $\mathcal{R}_n$ with vertex-set $V_{\mathcal{R}_n} = \{v \in N \times N \mid \Sigma(v) < n\}$. $\mathcal{R}_n$'s perfect-universality is witnessed by the identity map of the vertices of any n-vertex chaotic or ragged array to the vertices of $\mathcal{R}_n$. Of

[2] A can be a chaotic or ragged or rectangular array.

course, $\mathcal{R}_n$ is rather "inefficient," in that it has $n(n+1)/2$ vertices, whereas each $A \in \mathbf{C}_n$ (resp., $\mathbf{R}_n$) has only n vertices. It is natural to wonder how much smaller a 1-perfect-universal graph for $\mathbf{C}_n$, $\mathbf{R}_n$ and $\mathbf{S}_n$ can be. The *perfection number* of $\mathbf{C}_n$ (resp., $\mathbf{R}_n$, $\mathbf{S}_n$), denoted $\mathrm{Perf}(\mathbf{C}_n)$ (resp., $\mathrm{Perf}(\mathbf{R}_n)$, $\mathrm{Perf}(\mathbf{S}_n)$), is the size of the smallest such graph. These numbers are determined exactly in [6].

Theorem 1 ([6]) *For each integer $n > 0$:* **(a)** $\mathrm{Perf}(\mathbf{C}_n) = \lceil n/2 \rceil (\lfloor n/2 \rfloor + 1)$*;* **(b)** $\mathrm{Perf}(\mathbf{R}_n) = \frac{1}{2}(\lfloor n/3 \rfloor + 1)(3\lceil 2n/3 \rceil - n)$*;* **(c)** $\mathrm{Perf}(\mathbf{S}_n) = n$.

Our Results. Let $\mathrm{Perf}_c(\mathbf{X})$ denote the c-collision analogue of $\mathrm{Perf}(\mathbf{X})$, where $\mathbf{X}$ ambiguously denotes $\mathbf{C}, \mathbf{R}, \mathbf{S}$. We present close bounds on the quantities $\mathrm{Perf}_c(\mathbf{X})$, the upper bounds being supported by the first strategies for mapping array-structured data onto a parallel memory in such a way that any n-vertex chaotic array, ragged array or rectangular array can be accessed with at most c conflicts. These strategies lead to the following bounds.

Theorem 2 *For all integers c and n:*

$$\frac{1}{c}\left(\left\lfloor \frac{n}{2} \right\rfloor \cdot \left\lfloor \frac{n}{2(c+1)} \right\rfloor - 1\right) \;\leq\; \mathrm{Perf}_c(\mathbf{C}_n) \;\leq\; \left(\left\lceil \frac{n+1}{c} \right\rceil\right)^2$$

$$\frac{1}{c}\left(\left\lfloor \frac{n}{2} \right\rfloor \cdot \left\lfloor \frac{n}{2(c+1)} \right\rfloor - 1\right) \;\leq\; \mathrm{Perf}_c(\mathbf{R}_n) \;<\; \frac{n \cdot (n+1)}{c \cdot (c+1)} + n \;=\; \frac{n^2}{c^2} + \text{l.o.t.}$$

$$\mathrm{Perf}_c(\mathbf{S}_n) \;=\; \left\lceil \frac{n}{c} \right\rceil.$$

We now proceed to prove the various parts of Theorem 2.

3 A Lower Bound on $\mathrm{Perf}_c(\mathbf{C}_n)$ and $\mathrm{Perf}_c(\mathbf{R}_n)$

Theorem 3 *For all integers c and n, letting $\mathbf{X}$ ambiguously denote $\mathbf{C}_n$ and $\mathbf{R}_n$, we have*

$$\mathrm{Perf}_c(\mathbf{X}) \;\geq\; \frac{1}{c}\left(\left\lfloor \frac{n}{2} \right\rfloor \cdot \left\lfloor \frac{n}{2(c+1)} \right\rfloor - 1\right). \tag{1}$$

Proof : Let G be a c-perfect-universal graph for $\mathbf{C}_n$. Easily, one can view G as being obtained from the naive universal graph $\mathcal{R}_n$ by identifying vertices. Define U_n as follows (see also Fig. 1):

$$U_n \;\stackrel{\text{def}}{=}\; \left\{ \langle v_1, v_2 \rangle \in V_{\mathcal{C}_n} \;\middle|\; \left(v_1 < \lfloor n/2 \rfloor \text{ and } v_2 \leq \left\lfloor \frac{n}{2(c+1)} \right\rfloor\right) \text{ or } \left(v_1 = \lfloor n/2 \rfloor \text{ and } v_2 < \left\lfloor \frac{n}{2(c+1)} \right\rfloor\right) \right\}.$$

Observe that any set of $c+1$ vertices $\{x^{(1)}, x^{(2)}, \ldots, x^{(c+1)}\} \subseteq U_n$ can coexist within the same n-vertex chaotic array. To wit, if one send out paths to "capture" all of the $x^{(i)}$, then one ends up with a chaotic array C of size

$$\mathrm{Size}\,(C) \;\leq\; \lfloor n/2 \rfloor + \left(\left\lfloor \frac{n}{2(c+1)} \right\rfloor \cdot (c+1)\right) \;\leq\; n.$$

This fact implies that no vertex of G can "contain" more than c vertices of U_n: any such overloaded vertex would violate c-perfect-universality. This implies inequality (1), since $|U_n| = (\lfloor n/2 \rfloor \cdot \lfloor n/(2(c+1)) \rfloor - 1)$. The reader will note that our "bad" chaotic array C is also a ragged array, so that this lower bound also applies to $\mathrm{Perf}_c(\mathbf{R}_n)$. □

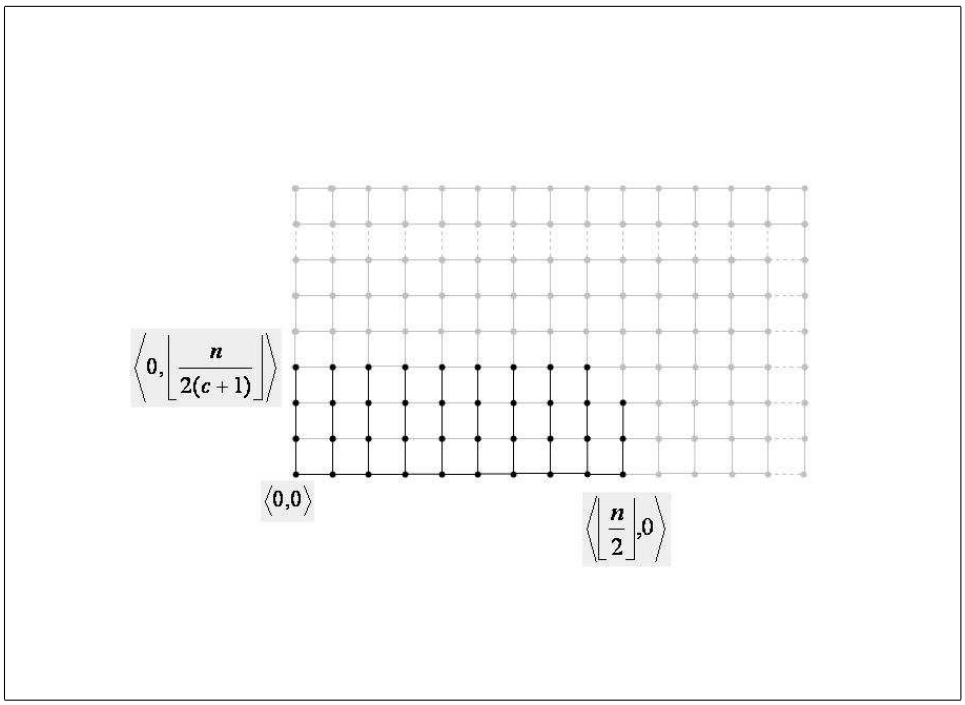

Figure 1: The set of vertices U_n.

4 An Upper Bound on $\mathrm{Perf}_c(\mathbf{C}_n)$

Theorem 4 *For all integers n and c,*

$$\mathrm{Perf}_c(\mathbf{C}_n) \;\leq\; \left\lceil \frac{n+1}{c} \right\rceil^2. \tag{2}$$

We prove Theorem 4 by explicitly constructing a graph G_c that is c-perfect-universal for $\mathbf{C}_n$. We construct G_c via an algorithm $\mathcal{A}_c$ that colors the vertices of the naive perfect graph $\mathcal{R}_n$ in such a way that the natural vertex-label-preserving embedding of any n-vertex chaotic array C into $\mathcal{R}_n$ (which witnesses C's being a subgraph of $\mathcal{R}_n$) never uses more than c vertices of any given color as homes for C's vertices. When we identify each like-colored set of vertices of $\mathcal{R}_n$—i.e., contract each such set to a single vertex—we obtain the graph G_c whose size provides our upper bound on $\mathrm{Perf}_c(\mathbf{C}_n)$.

For each vertex $v \in \mathcal{R}_n$, we denote by $color\,(v)$ the color assigned to v by $\mathcal{A}_c$. Let $d = \lceil (n+1)/c \rceil$, and let $V_d = \{\langle v_1, v_2 \rangle \in \mathcal{R}_n \mid v_1 < d \text{ and } v_2 < d\}$.
Algorithm $\mathcal{A}_c$ assigns a color to each vertex $\langle v_1, v_2 \rangle \in \mathcal{R}_n$ as follows:

if $v = \langle v_1, v_2 \rangle \in V_d$, **then** $color\,(v) = v_1 \cdot d + v_2$;

if $v = \langle v_1, v_2 \rangle \notin V_d$, **then** $color\,(v) = color\,(\langle v_1 \bmod d, v_2 \bmod d \rangle)$.

Simple counting and reasoning yield the following observations.

Observation 1. *Algorithm $\mathcal{A}_c$ uses exactly $d^2 = \lceil (n+1)/c \rceil^2$ colors to color $\mathcal{R}_n$.*

Let $\delta\,(u, v)$ denote the (Manhattan) distance between nodes $u = \langle u_1, u_2 \rangle$ and $v = \langle v_1, v_2 \rangle$ of $N \times N$; i.e., $\delta\,(u, v) = |v_1 - u_1| + |v_2 - u_2|$.

Observation 2. *If $\mathcal{A}_c$ assigns the same color to vertices $u, v \in \mathcal{R}_n$—i.e., if $color\,(u) = color\,(v)$—then $\delta\,(u, v) \geq d$.*

Proof of Theorem 4: We claim that the coloring of $\mathcal{R}_n$ by Algorithm $\mathcal{A}_c$ illustrates how to embed any $C \in \mathbf{C}_n$ into $\mathcal{R}_n$ with at most c collisions. Indeed, if a chaotic array $C \in \mathbf{C}_n$ cannot be so embedded, then its size must exceeds n. To wit, let us ignore the part of C that connects the origin of $\mathcal{R}_n$ with the vertex of C closest to the origin. An easy generalization of Observation 2, shows that any connected set of $c+1$ like-colored vertices created by Algorithm $\mathcal{A}_c$ must have at least $c \cdot \lceil (n+1)/c \rceil > n$ vertices (see Fig. 2). An appeal to Observation 1 now establishes the claimed bound on $\mathrm{Perf}_c(\mathbf{C}_n)$. □

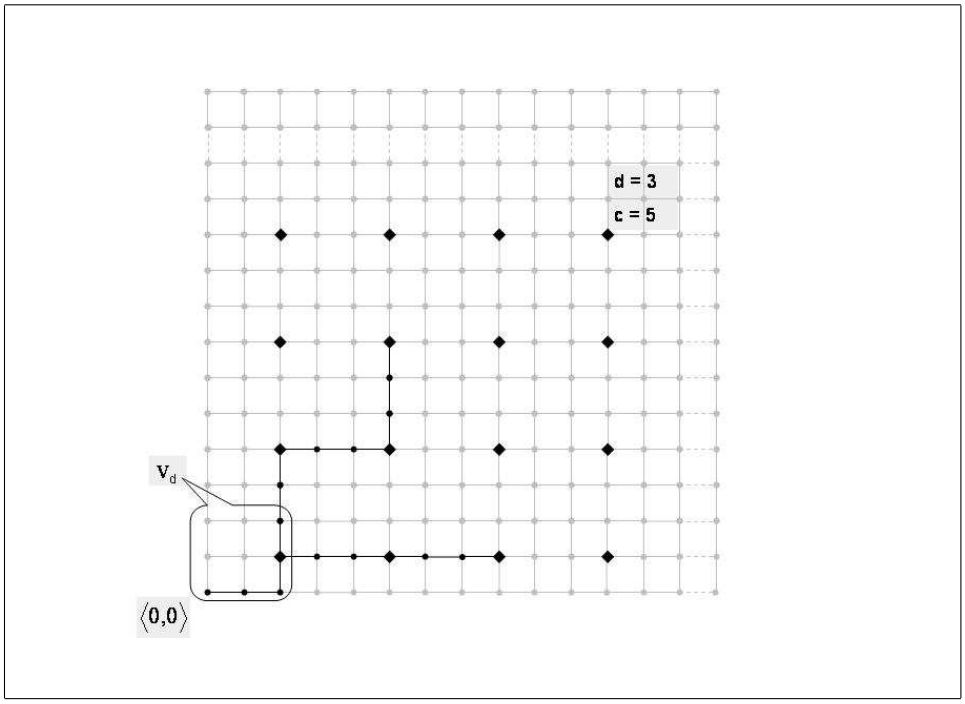

Figure 2: An example of $C \in \mathbf{C}_n$ with at most $c = 5$ collisions. notice that the set V_d is colored with distinct colors.

5 An Upper Bound on $\mathrm{Perf}_c(\mathbf{R}_n)$

Theorem 5 *For all integers n and c,*

$$\mathrm{Perf}_c(\mathbf{R}_n) \;<\; \frac{n\cdot(n+1)}{c\cdot(c+1)} + n \;=\; \frac{n^2}{c^2} + \text{l.o.t.}. \tag{3}$$

Proof : We derive our upper bound on $\mathrm{Perf}_c(\mathbf{R}_n)$ by explicitly constructing a graph H_c that is c-perfect-universal for $\mathbf{R}_n$. We construct H_c via an algorithm $\mathcal{B}_c$ that colors the vertices of $\mathcal{R}_n$ in such a way that the natural vertex-label-preserving embedding of any n-ragged array R into $\mathcal{R}_n$ (which witnesses R's being a subgraph of $\mathcal{R}_n$) never uses more than c vertices of any given color as homes for R's vertices. When we identify each like-colored set of vertices of $\mathcal{R}_n$—i.e., contract each set to a single vertex in the obvious way—we obtain the c-perfect-universal graph H_c, whose size is clearly an upper bound on $\mathrm{Perf}_c(\mathbf{R}_n)$. Algorithm $\mathcal{B}_c$ proceeds via sequential passes along each diagonal of $\mathcal{R}_n$ in turn, assigning a unique set of colors, L_i, to the vertices of each diagonal $D_i^{(\mathcal{R}_n)}$ in a round-robin fashion. $\mathcal{B}_c$ thereby distributes the $i+1$ vertices of $D_i^{(\mathcal{R}_n)}$ among the $|L_i|$ diagonal-i vertices of H_c. Clearly, then, $\mathrm{Size}\,(H_c) = \sum_i |L_i|$.

The remainder of the section is devoted to estimating how big the sets L_i must be in order for H_c to be c-perfect-universal for $\mathbf{R}_n$.

Let us begin by considering the following ragged subarrays of $\mathcal{R}_n$. For positive integers m, i and φ with $(m-1)\cdot\varphi \;\le\; i \;\le\; n$, the ragged array $RA_{(i,m,\varphi)}$ consists of:

- the $i+1$ vertices that connect vertices $\langle 0,0\rangle$ and $\langle i,0\rangle$;
- for $1 \le j \le m-1$, the $j\varphi$ vertices that connect vertices $\langle i-j\varphi,\ 1\rangle$ and $\langle i-j\varphi,\ j\varphi\rangle$.

Note (see Fig. 3) that $RA_{(i,m,\varphi)}$ has m vertices on diagonal $D_i^{(\mathcal{R}_n)}$ and none on diagonals $D_j^{(\mathcal{R}_n)}$ for $j > i$. Moreover, by simple counting, $RA_{(i,m,\varphi)}$ has

$$i+1+\sum_{i=1}^{m-1}\varphi i \;=\; \binom{m}{2}\cdot\varphi + i + 1$$

nodes. The following source of our interest in the ragged arrays $RA_{(i,m,\varphi)}$ is readily verified.

Lemma 1 *For all integers $i, m, \varphi > 0$, $RA_{(i,m,\varphi)}$ is the smallest ragged array $R \in \mathbf{R}_n$ that has m vertices on diagonal $D_i^{(\mathcal{R}_n)}$ and size $\le \frac{1}{2}\varphi m(m-1) + i + 1$.*

Lemma 2 *Let $\mathcal{R}_n$ be colored by algorithm $\mathcal{B}_c$ using κ_i colors at each level i. If each*

$$\kappa_i \;\ge\; \varphi_i \;\stackrel{def}{=}\; \left\lceil \frac{2(n-i)}{c\cdot(c+1)} \right\rceil, \tag{4}$$

then any $R \in \mathbf{R}_n$ can be embedded into $\mathcal{R}_n$ with at most c collisions.

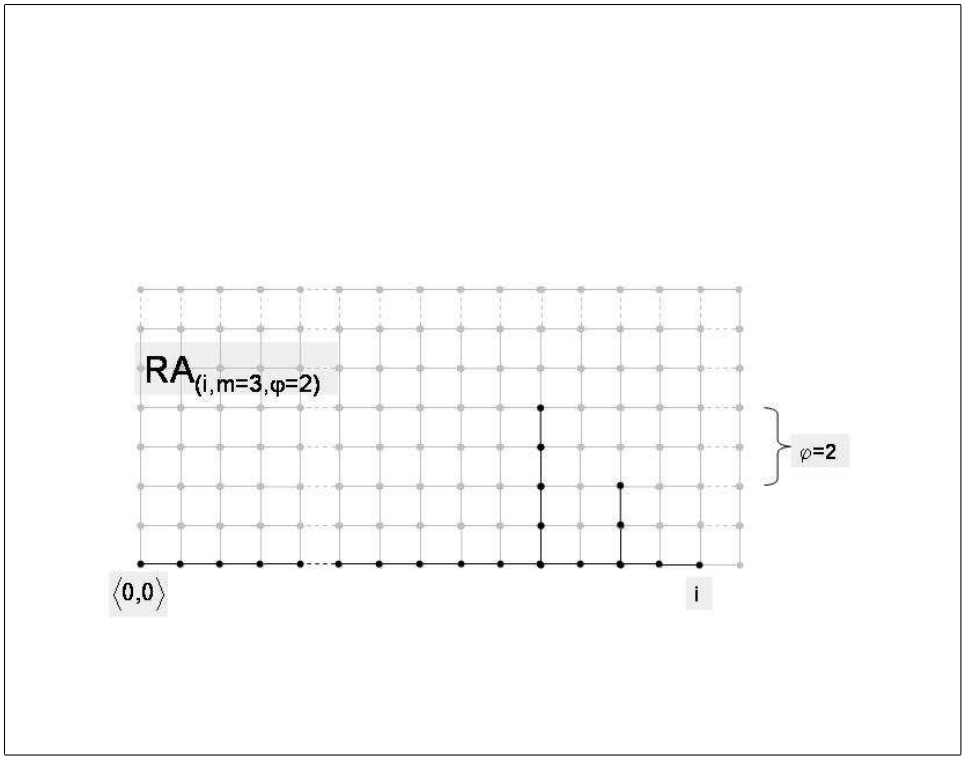

Figure 3: $RA_{(i,m,\varphi)}$

Proof : We show that if a ragged array $R \in \mathcal{R}_n$ engenders more than c collisions, then $\text{Size}\,(R) > n$. Note that each collision that occurs must involve vertices from the same diagonal of $\mathcal{R}_n$, because $\mathcal{B}_c$ uses a distinct set of colors at each diagonal. Let us, therefore, consider a shallowest (in terms of number of diagonals touched) ragged array R that engenders $c+1$ collisions at diagonal i of $\mathcal{R}_n$. Easily, the offending ragged array R has $c+1$ vertices from diagonal i of $\mathcal{R}_n$, and, by the design of Algorithm $\mathcal{B}_c$, these vertices must be at distance κ_i from one another. We can, therefore, combine Lemma 1 (with $m = c+1$), and the bound (4) to bound from below the size of the offending ragged array R.

$$\begin{aligned}
\text{Size}\,(R) \;\geq\; \text{Size}\left(RA_{(i,c+1,\varphi_i)}\right) &= \varphi_i \cdot \binom{c+1}{2} + i + 1 \\
&= \left\lceil \frac{2(n-i)}{c\cdot(c+1)} \right\rceil \cdot \left(\frac{c\cdot(c+1)}{2} \right) + i + 1 \\
&\geq n - i + i + 1.
\end{aligned}$$

Since we are interested only in ragged arrays having $\leq n$ vertices, the lemma follows. □

Combining Lemmas 1 and 2, we thus have the following bound on the size of our c-universal graph H_c.

$$\text{Size}\,(H_c) \;=\; \sum_{i=0}^{n-1} \kappa_i \;=\; \sum_{i=0}^{n-1} \left\lceil \frac{2(n-i)}{c\cdot(c+1)} \right\rceil$$

$$< \quad n + \frac{2}{c \cdot (c+1)} \cdot \sum_{i=0}^{n-1} (n-i)$$

$$= \quad \left(\frac{n \cdot (n+1)}{c \cdot (c+1)} \right) + n.$$

The claimed upper bound (3) follows. □

6 The Value of $\mathrm{Perf}_c(\mathbf{S}_n)$

Theorem 6 *For all integers n and c,* $\mathrm{Perf}_c(\mathbf{S}_n) = \lceil n/c \rceil$.

Proof (sketch): The rectangular array with vertices $[n-1] \times \{0\}$ indicates that $\mathrm{Perf}_c(\mathbf{S}_n) \geq \lceil n/c \rceil$. Since by Theorem 1(c), $\mathrm{Perf}(\mathbf{S}_n) = n$, a simple "blocking" strategy shows that $\mathrm{Perf}_c(\mathbf{S}_n) \leq \lceil n/c \rceil$. □

References

[1] V. Auletta, S. Das, A. De Vivo, M.C. Pinotti, V. Scarano, "Optimal tree access by elementary and composite templates in parallel memory systems." *IEEE Trans. on Parallel and Distr. Systs. 13*, 2002.

[2] S.N. Bhatt, F.R.K. Chung, F.T. Leighton, A.L. Rosenberg. "Universal graphs for bounded-degree trees and planar graphs." *SIAM J. Discr. Math. 2*, 1989, 145–155.

[3] P. Budnik, D.J. Kuck. "The organization and use of parallel memories." *IEEE Trans. Comput. C-20*, 1971, 1566–1569.

[4] C.J. Colbourn, K. Heinrich. "Conflict-free access to parallel memories." *J. Parallel and Distributed Comput. 14*, 1992, 193–200.

[5] F.R.K. Chung and R.L. Graham. "On universal graphs for spanning trees." *Proc. London Math. Soc. 27*, 1983, 203–211.

[6] F.R.K. Chung, A.L. Rosenberg, L. Snyder. "Perfect storage representations for families of data structures." *SIAM J. Algebr. Discr. Meth. 4*, 1983, 548–565.

[7] G. Cordasco, A. Negro, A.L. Rosenberg, V. Scarano. "c-perfect hashing schemes for binary trees, with applications to parallel memories." *Intl. Conf. on Parallel and Distr. Computing (Euro-Par'03)*, to appear, 2003.

[8] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein. *Introduction to Algorithms* (2nd Edition). MIT Press, Cambridge, Mass., 1999.

[9] R. Creutzburg, L. Andrews, "Recent results on the parallel access to tree-like data structures – the isotropic approach." *Intl. Conf. on Parallel Processing*, Vol. 1, 1991, 369-372.

[10] S.K. Das, F. Sarkar, "Conflict-free data access of arrays and trees in parallel memory systems." *6th IEEE Symp. on Parallel and Distr. Processing*, 1994, 377–383.

[11] L. Euler. "Recherches sur une espéce de carrés magiques." *Commentationes Arithmeticae Collectae, vol. II*, 1849, 302–361.

[12] D.H. Lawrie. "Access and alignment of data in an array processor." *IEEE Trans. on Computers, C-24*, 1975, 1145–1155.

[13] R.J. Lipton, A.L. Rosenberg, A.C. Yao, "External hashing schemes for collections of data structures." *J. ACM 27*, 1980, 81–95.

[14] K. Kim, V.K. Prasanna. "Latin squares for parallel array access." *IEEE Trans. on Parallel and Distr. Systs 4*, 1993, 361–370.

[15] D.L. Milgram and A. Rosenfeld, "Array automata and array grammars" In *Information Processing 71*, North-Holland, Amsterdam, 1971, 69–74.

[16] A.L. Rosenberg, "Computed access in ragged arrays." In *Information Processing 74*, North-Holland, Amsterdam, 1974, 642–646.

[17] A.L. Rosenberg, "On storing ragged arrays by hashing." *Math. Syst. Th. 10*, 1976/77, 193–210.

[18] A.L. Rosenberg and L.J. Stockmeyer, "Hashing schemes for extendible arrays." *J. ACM 24*, 1977, 199–221.

[19] V. Scarano, "Versatile access to parallel memory systems." Proc. of *1998 Wkshp. on Distributed Data and Structures*, Orlando (Florida) USA, Carleton Press,1998.

[20] J.S. Vitter, E.A.M. Shriver. "Algorithms for parallel memory, I: Two-level memories." *Algorithmica 12(2/3)*, 1994, 110-147.

[21] J.S. Vitter, E.A.M. Shriver. "Algorithms for parallel memory II: Hierarchical multilevel memories.". *Algorithmica 12(2/3)*, 1994, 148-169.

[22] H.A.G. Wijshoff, "Storing trees into parallel memories." *Parallel Computing*, 1986, 253-261.