INTERNATIONAL COMPUTER SCIENCE INSTITUTE

1947 Center Street • Suite 600 • Berkeley, California 94704 • 1-510-642-4274 • FAX 1-510-643-7684

# Constructive Deterministic PRAM Simulation on a Mesh-Connected Computer*

A. Pietracaprina† G. Pucci‡§ J.F. Sibeyn¶

TR-93-059

October 1993

## Abstract

The PRAM model of computation consists of a collection of sequential RAM machines accessing a shared memory in lock-step fashion. The PRAM is a very high-level abstraction of a parallel computer, and its direct realization in hardware is beyond reach of the current (or even foreseeable) technology. In this paper we present a deterministic simulation scheme to emulate PRAM computation on a mesh-connected computer, a feasible machine where each processor has its own memory module and is connected to at most four other processors via point-to-point links. In order to achieve a good worst-case performance, any deterministic simulation scheme has to replicate each variable in a number of copies. Such copies are stored in the local memory modules according to a Memory Organization Scheme (MOS), which is known to all the processors. A variable is then accessed by routing packets to its copies. All deterministic schemes in the literature make use of a MOS whose existence is proved via the probabilistic method, but that cannot be efficiently constructed. We introduce a new constructive MOS, and show how to employ it to simulate an $n$-processor PRAM on an $n$-node mesh-connected computer. Our simulation achieves almost optimal slowdown for small memories. This is the first constructive deterministic PRAM simulation on a bounded-degree network.

*This research was supported in part, through the Leonardo Fibonacci Institute, by the Istituto Trentino di Cultura.

†Department of Computer Science, Brown University, Providence, RI 02912, USA. Email aap@cs.brown.edu.

‡Dipartimento di Elettronica e Informatica, Università di Padova, Padova I35131, Italy. Email geppo@art.dei.unipd.it. The work of the author was partially supported by MURST, Italy.

§International Computer Science Institute, Berkeley, CA 94708-1105, USA.

¶Max-Planck Institut für Informatik, Im Stadtwald, W-6600 Saarbrücken, Germany. Email jopsi@mpi-sb.mpg.de. The work of the author was partially supported by project ALTEC (basic research actions IC1000) of the EC.

# 1 Introduction

The PRAM is undoubtly the most attractive computational model for designing parallel algorithms. Its popularity mainly derives from the generality of its definition, that allows $n$ processors to read/write any set of $n$ cells (*variables*) of a shared memory, in constant time. This powerful feature is, however, unfeasible from a technological standpoint, and has challenged many authors, in the past decade, to simulate PRAM computation on more realistic models. A typical approach consists of partitioning the shared memory into $n$ modules, local to the processors, and having the processors connected by some network. In one time unit, each processor is now able to access a single variable from its module, or communicate with one of its neighbors. A PRAM step, where up to $n$ variables are requested, is simulated by sending a message through the network between each processor issuing an access to a variable and the processor storing that variable. The goal is to devise simulation schemes that are easy to implement and exhibit a good time performance. In order to minimize the time needed to simulate one PRAM step, one has to minimize both the memory contention, caused by requests addressed to the same module, and the congestion in the network, caused by the routing of the messages.

Several randomized simulation schemes have been presented in the literature, [MV84, KU88, LPP88, Ran91, Mey92, KLM92, DM93]. In all these schemes, the PRAM shared memory is distributed among the modules using one (or more) hash functions randomly drawn from a specific universal class [CW79]. One of the most significant results is Ranade's simulation of an $n$-processor PRAM step on an $n$-node Butterfly, in $O(\log n)$ time, with high probability [Ran91]. Recently, [Mey92, KLM92, DM93] showed that if each variable is replicated into a (small) number of copies, distributed among the modules by distinct hash functions, more efficient simulations are achievable. For example, with three copies per variable, an $n$-processor PRAM step can be simulated on a complete network of $n$ processors (*Module Parallel Computer* (*MPC*)) in time $O(\log\log n)$, with high probability.

On the other end, the development of efficient deterministic simulation schemes, that is, schemes that guarantee a worst-case bound on the access time, appears to be much harder. A simple argument shows that in order to avoid the trivial worst-case, where all the requested variables are stored in the same module, one has to use several copies for each variable, so that only a subset of "convenient" copies needs to be reached by each operation. The number of copies for each variable is called the *redundancy* of the simulation scheme. Mehlhorn and Vishkin [MV84] pioneered the multiple copy approach devising a PRAM simulation on the MPC, with redundancy $c$. In their scheme, only one copy is needed to read a variable, whereas all the copies have to be updated when the variable is written. An involved access protocol is given that satisfies a set of $n$ read in time $O(cn^{1-1/c})$, in the

worst-case, whereas for $n$ write operations, the protocol can take up to $O(cn)$ time.

Later, Upfal and Widgerson [UW87] proposed a more balanced use of the copies based the *m*ajority concept, previously adopted for databases [Gif79, Tho79]. Each variable is replicated into $2c-1$ copies. Each copy contains the value of the variable and a time-stamp indicating the last time that the copy has been accessed. Thus, a read/write operation needs to access only a majority $c$ of the copies to assure that the most recent value of the variable is always retrieved. The partitioning of the copies into modules is done according to a bipartite graph $G = (V, U; E)$, where $V$ represents the set of variables, $U$ the set of modules, and $2c-1$ edges connect each variable to the modules where its copies are stored. With $c \in \Theta(\log n)$, [UW87] show that there exist graphs $G$ with suitable expansion properties so that $n$ variables can be accessed in $O(\log n(\log\log n)^2)$ worst-case time on the MPC. They do not provide an explicit construction for $G$ but show that a random graph exhibits the desired properties, with high probability.

Several authors followed the ideas in [UW87] improving the time complexity and using bounded-degree networks instead of the MPC [AHMP87, HB88, Her89, LPP90, Her90b, AS90]. However, all these schemes are based on the *existence* of highly expanding graphs, which represents the basic shortcoming (maybe fatal from the practical standpoint) of this class of approaches, since the construction and testing of such graph is hard (see [PP93a]). Moreover, if one resorts to random graphs, the internal representation of the memory map becomes extremely space-inefficient [Her90a].

In two recent works, Pietracaprina and Preparata [PP93a, PP93b] presented the first explicit deterministic PRAM simulations that exhibit a sublinear time complexity for both read and write operations. Both results are for the MPC. In [PP93a], a simulation scheme for a PRAM with at most $n^2$ variables is given where, using constant redundancy, a set of $n$ (read/write) requests can be satisfied in $O(\sqrt{n})$ time in the worst-case. In [PP93b], an $O(n^{1/3}\log^* n)$ access time is achieved for a shared memory of at most $n^{1.5}$ variables, again using constant redundancy. In both cases, an approach similar to that by [UW87] is followed; however, the graph $G$ underlying the memory distribution is explicitly constructed, and the implementation of the memory map is simple and requires only constant internal storage in each processor.

Unfortunately, the MPC itself is an unrealistic model, since it assumes a complete interconnection among the processors. A simulation strategy on the MPC focuses on the memory contention problem, but completely ignores routing issues. Therefore, there is the need to develop efficient and explicit deterministic schemes that run on realistic interconnections, that is, networks of bounded degree. In this paper we present the first explicit deterministic scheme to simulate a PRAM with $n$ processors and $n^\alpha$ variables, $1 < \alpha < 2$, on an $n$-node square mesh. Each variable is replicated into $q^k$ copies, where $q \in O(1)$ is a prime power

and $k \geq 2$ an integer. The organization of the copies among the processors and an ad-hoc access protocol enable the processors to perform any set of $n$ read/write operations in the time bounds stated by the following theorem.

**Theorem 1** *Given an $n$-processor PRAM with a shared memory of $n^\alpha$ cells, $1 < \alpha \leq 2$, one computational step can be simulated in time*

$$T(n) \in \begin{cases} n^{\frac{1}{2}+\epsilon} & \text{if } 0 < \epsilon < 1 \text{ and } \alpha \leq \frac{3}{2}; \\ n^{\frac{1}{2}+\frac{\alpha-1}{16}} & \text{if } \frac{3}{2} \leq \alpha \leq \frac{5}{3}; \\ n^{\frac{1}{2}+\frac{2\alpha-3}{8}} & \text{if } \frac{5}{3} \leq \alpha \leq 2, \end{cases}$$

*with constant redundancy, and*

$$T(n) \in \quad n^{\frac{1}{2}} polylog(n) \quad \text{if } \alpha \leq \frac{3}{2},$$

*with polylog(n) redundancy.*

*The simulating machine is a square mesh of $n$ nodes. The scheme is deterministic and fully constructive.*

Note that an $\Omega(\sqrt{n})$ lower bound applies to any simulation on the mesh because of the network diameter.

The core of the simulation is a *Hierarchical Memory Organization Scheme (HMOS)* that governs the distribution of the copies among the processors. There are $k$ levels of logical modules, for a suitably chosen value $k \geq 1$. First, each variable is replicated into $q$ copies, assigned to the modules of the first level. Each one of these modules is in turn replicated into $q$ copies assigned to modules of the second level, and so on for $k$ times. Note that eventually each variable will be replicated into $q^k$ copies.

At any level $i$, the assignment of copies of modules of level $i-1$ to modules of level $i$ is done according to a *Balanced Incomplete Block Design*, which is the distribution graph underlying the memory organization scheme in [PP93a]. The construction of such graph is simple and allows a very efficient representation of the memory map.

As we will see in the following sections, as the level number grows higher, the modules become fewer and bigger. On the mesh, modules of the same level are mapped onto submeshes of the same size. Different levels correspond to different tessellations of the mesh into disjoint submeshes.

In order to read/write a set of $n$ variables, first a subset of their copies is selected according to a consistency rule that fits the hierarchical structure of our memory organization. These copies are also chosen so that contention within the modules of any level is conveniently bounded. Once selected, the copies are then accessed using a new routing strategy that takes each request packet through smaller and smaller submeshes, towards its

destination. The low congestion in each submesh, ensured by the copy selection mechanism, is crucial for the performance of the routing.

The relevance of our result is twofold.

1. It is the first explicit deterministic scheme for PRAM simulation on a bounded-degree network, and, when $\alpha \leq 3/2$, its time complexity is nearly optimal.

2. It introduces the HMOS, a new mechanism that enables to control both memory contention and network congestion, and, therefore, is suitable for deterministic simulations on bounded-degree networks.

The rest of the paper is structured as follows. In the next section, we present some results concerning the specific routing problems arising in the simulation. The PRAM simulation scheme is given in Section 3, which is subdivided into several subsections that describe in detail the memory organization, the selection of the copies, and the access protocol.

## 2 Routing on the Mesh

Consider an $(l_1, l_2)$-*routing problem* on an $n$-node mesh where each processor sends at most $l_1$ packets and receives at most $l_2$ packets. In [SK93] the following theorem is proved

**Theorem 2** *Any $(l_1, l_2)$-routing can be performed on an $n$-node mesh in $\sqrt{l_1 l_2 n} + O(l_1\sqrt{n})$ steps.*

In the general case, this result is optimal, however we will show now that under certain conditions a better routing time can be achieved. Subdivide the mesh into $n/m$ submeshes of $m$ nodes each, with $m \in o(n)$, and consider a $(l_1, l_2)$-routing problem where each submesh receives at most $\delta m$ packets, i.e., $\delta$ is the average number of packets that each processor receives in any given submesh. Call this problem an $(l_1, l_2, \delta, m)$-routing. An algorithm that first sends each packet to its destination submesh and then to its final destination, within the submesh, turns out to be more efficient that the general $(l_1, l_2)$-routing algorithm, for certain values of the parameters. More precisely, the algorithm works as follows.

**1.** Index the processors in each submesh from 0 to $m - 1$;

**2.** Sort and rank all packets in the mesh according to their destination submeshes;

**3.** Route all packets to their destination submeshes so that a packet with rank $i$ in its submesh is sent to the processor of index $i$ mod $m$ in the submesh;

**4.** Route all packets to their final destination.

Sorting and ranking can be done in $O(l_1\sqrt{n})$ steps (see for example [KSS94, Kun93]), and the two routing stages require, using Theorem 2, $\sqrt{l_1\delta n}+O(l_1\sqrt{n})$ and $\sqrt{\delta l_2 m}+O(\delta\sqrt{m})$ steps, respectively. Observing that $l_1 \leq \delta \leq l_2$, we have that the total time complexity is

$$O(\sqrt{\delta}(\sqrt{l_1 n} + \sqrt{l_2 m})).$$

Comparing the $O(\sqrt{l_1 l_2 n})$ complexity of the general $(l_1, l_2)$-routing with the above formula, we conclude that the $(l_1, l_2, \delta, m)$-routing algorithm is more profitable when $l_1, \delta \in o(l_2)$ and $\sqrt{\delta m} \in o(\sqrt{l_1 n})$.

These ideas will be exploited in the access protocol of our PRAM simulation scheme, where an $(l_1, l_2)$-routing problem has to be solved in order to access the copies of the variables. Several tessellations of the mesh into submeshes of different sizes are defined, and bounds on the number of packets destined to each submesh, ensured by a careful copy selection, allow us to adopt a more efficient routing strategy, where the packets are gradually routed to their destinations through a sequence of smaller and smaller submeshes.

# 3 PRAM Simulation Scheme

In this section we present an explicit deterministic scheme to simulate an $n$-processor PRAM on an $n$-node mesh. Suppose that the PRAM shared memory contains $n^\alpha$ variables, for some $\alpha > 1$. The goal is to distribute the shared memory among the processors of the mesh, so that any set of $n$ distinct variables can be efficiently accessed (read or written). To avoid incurring trivial lower bounds, we need to replicate each variable into a number of copies so that, when the variable is requested, we select an appropriate subset of the copies that minimizes the congestion in the network while maintaining consistency. The way the copies are organized in the network, the selection of the copies to access for each operation and the access protocol are presented in the following subsections.

## 3.1 Hierarchical Memory Organization Scheme

What we will use for our simulation is a *Hierarchical Memory Organization Scheme*, *HMOS*. Each variable is first replicated into $q$ copies, $q \in O(1)$, and the set of all $n^\alpha q$ copies is evenly distributed among $\Theta(n^{\alpha/2})$ *level-1 modules*. Each level-1 module is, in turn, replicated into $q$ copies, and the copies of all the level-1 modules are evenly distributed among $\Theta(n^{\alpha/4})$ *level-2 modules*. This process is iterated for $k$ levels. In general, there are $\Theta(n^{\alpha/2^i})$ *level-i modules* replicated into $q$ copies each, which are distributed among $\Theta(n^{\alpha/2^{i+1}})$ level-$(i+1)$ modules, for $0 \leq i < k$ (here, the variables are considered as *level-0 modules*). Level-$k$ modules are not replicated. Note that higher numbered levels consist of fewer modules. We will reserve the term *copy* for the copies of the original variables, and will call *level-i pages*

the copies of the *level-$i$ modules*. For the level-$k$ modules, the terms "module" and "page" will be used indistinguishably. It is easy to see that this setting generates $q^{k-i}$ level-$i$ pages for each level-$i$ module, for $0 \leq i < k$.

The HMOS can be conveniently represented by a $(k+1)$-partite graph $\mathcal{G} = (U_0, \ldots, U_k; E_1, \ldots, E_k)$, where $U_i$, $0 \leq i \leq k$, denotes the set of level-$i$ modules, and $E_i$, $1 \leq i \leq k$, the set of edges between $U_{i-1}$ and $U_i$. More precisely, $q$ edges connect each level-$i$ module to distinct level-$(i+1)$ modules containing its pages (see Figure 1).

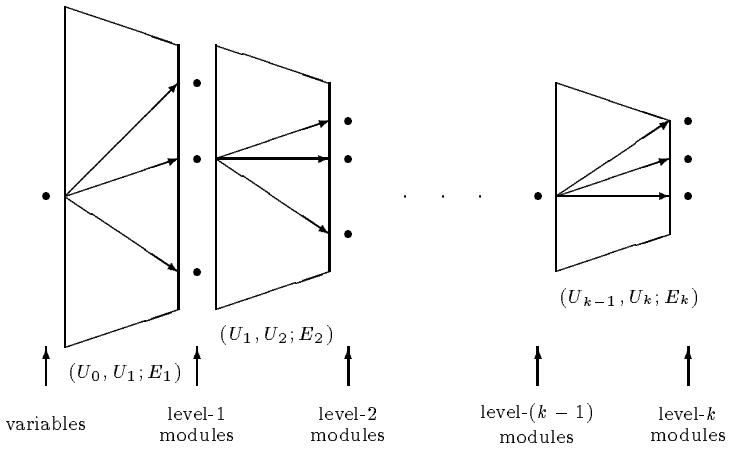


Figure 1: Structure of the HMOS.

The construction of the actual graph is based on a well known combinatorial structure, the *Balanced Incomplete Block Design* [Hal86], which has been first exploited for PRAM simulation on the MPC in [PP93a].

**Definition 1** *A Balanced Incomplete Block Design with parameters $m$ and $q$, a $(m,q)$-*BIBD*), is a bipartite graph $G = (W, U; E)$ such that*

- $|U| = m$;
- *The degree of any node in $W$ is $q$;*
- *For any two nodes $u_1, u_2 \in U$ there is exactly one node $w \in W$ connected to both*[1].

It immediately follows from 1 that $|W| = \frac{m(m-1)}{q(q-1)}$ and that the degree of each node in $U$ is $\frac{m-1}{q-1}$. An explicit construction of a $(q^d, q)$-BIBD, suitable for use in PRAM simulations, is given in [PP93a] for any prime power $q$ and integer $d \geq 1$.

One basic property of the BIBD, which we will heavily exploit in our simulation, is stated in the following lemma.

[1] The more general definition of BIBD found in the literature includes a third parameter $\lambda$ and requires that for any two nodes $u_1, u_2 \in U$ there are exactly $\lambda$ nodes in $W$ connected to both. However, we always have $\lambda = 1$, and omit the parameter $\lambda$.

**Lemma 1** *Let $G = (W, U; E)$ be an $(m, q)$-BIBD and let $u \in U$. Consider a subset $S \subset W$ such that for any $w \in S$, $(w, u) \in E$. For each $w$ in $S$ fix $k \leq q$ outgoing edges including the edge $(w, u)$ and let $\Gamma_k(S)$ denote the set of nodes of $U$ reached by these edges. Then*

$$|\Gamma_k(S)| = (k-1)|S| + 1.$$

**Proof:** By the property of the BIBD, the nodes of $S$ cannot share any neighbor other than $u$. Since for each $w \in S$ we fix $k \leq q$ outgoing edges including $(w, u)$, $\Gamma_k(S)$ must include $u$ and $(k-1)|S|$ other nodes. □

We refer to this property as the *strong expansion property.*

Let us return to the graph $\mathcal{G} = (U_0, \ldots, U_k; E_1, \ldots, E_k)$ underlying our memory organization. Let $q$ be a prime power and suppose that $n^\alpha = q^{d-1}\frac{q^d-1}{q-1}$, for some integer $d$. The subgraphs between consecutive levels are chosen as follows. Define the sequence of integers $d_1, \ldots, d_k$ as:

$$\begin{aligned} d_1 &= d, \\ d_{i+1} &= \lceil d_i/2 \rceil + 1, \qquad 1 \leq i < k, \end{aligned}$$

and set

$$\begin{aligned} |U_0| &= q^{d_1-1}\frac{q^{d_1}-1}{q-1} = n^\alpha, \\ |U_i| &= q^{d_i}, \qquad 1 \leq i < k. \end{aligned}$$

By induction it can be proven easily that

$$|U_i| = cn^{\alpha/2^i}, \qquad c \in [\frac{q}{2}, q^3], \tag{1}$$

which implies $|U_i| \in \Theta(n^{\alpha/2^i})$, for $0 \leq i \leq k$.

The graph $(U_i, U_{i+1}; E_{i+1})$, $0 \leq i \leq k$, is chosen as a subgraph of a $(q^{d_{i+1}}, q)$-BIBD, where $q^{d_{i+1}-1}\frac{q^{d_{i+1}}-1}{q-1} - q^{d_i}$ arbitrary input nodes are removed. Define $f(s)$ to be the size of the input of a $(q^s, q)$-BIBD:

$$f(s) \triangleq q^{s-1}\frac{q^s-1}{q-1}.$$

Note that for $i \geq 1$, $f(d_{i+1}-1) < q^{d_i} \leq f(d_{i+1})$. Therefore, the $(q^{d_{i+1}}, q)$-BIBD is the smallest BIBD with $q^{d_{i+1}}$ outputs and at least $q^{d_i}$ inputs.

The HMOS is physically mapped onto the network by defining $k$ tessellations of the mesh into submeshes of appropriate sizes. The processors of each submesh of the $i$th tessellation store one level-$i$ page. More details regarding the sizes of the submeshes in each tessellation and the actual mapping of the pages onto such submeshes will be given later when the access protocol is presented.

As observed before, a variable $v$ is replicated into $q^k$ copies. Such copies can be organized as a labeled, complete $q$-ary tree $\mathcal{T}_v$ of $k+1$ levels. The root is labeled with the name of the variable $v$, and its $q$ children with the names of the level-1 modules storing the copies of $v$. For $1 \leq i < k$, given an internal node at level $i$ of label $l$ ($l$ is the name of a level-$i$ module), its $q$ children are labeled with the names of the level-$(i+1)$ modules storing the pages of $l$. Thus, each copy of $v$ can be identified with a distinct leaf of $\mathcal{T}_v$, and the associated string of labels $\langle l_k, l_{k-1}, \ldots, l_0 \rangle$ of the nodes on the path from the leaf to the root.

When accessing any variable $v$ during the simulation of a PRAM step, consistency can be guaranteed by providing each copy with a timestamp, as customary in a multiple copy approach, and extending the majority rule of [Gif79, Tho79, UW87] to fit the HMOS, as follows.

**Definition 2** *A leaf of $\mathcal{T}_v$ is* accessed *if it is reached during a read or write step. A node of $\mathcal{T}_v$ at level $i$, $0 \leq i \leq k-1$ is* accessed *if a majority of its children is accessed.*

It is straightforward to show that any write protocol that accesses the root of $\mathcal{T}_v$ (i.e., a protocol that writes enough copies of $v$ so that the root of $\mathcal{T}_v$ is accessed) guarantees that any subsequent read protocol that accesses the root of $\mathcal{T}_v$ will always reach at least one updated copy of $v$, retrieved by looking for the most recent timestamp. Any set of copies which guarantees, when reached, that the root of $T_v$ is accessed will be called a *target set* for variable $v$. A *minimal target set* is a target set that does not properly include another target set. Note that when only the copies in a minimal target set are reached, the accessed nodes in $T_v$ form a complete $(\lfloor q/2 \rfloor + 1)$-ary sub-tree.

### 3.2 Copy Selection

Consider a PRAM step, where each of the $n$ processors wants to read or write a distinct variable, and let $R$ denote the set of requested variables. Suppose that each processor of the mesh emulates a PRAM processor and is in charge of one variable. The simulation of the PRAM step on the mesh is divided into two stages:

1. Copy selection.
2. Access protocol.

The copy selection stage determines, for each $v \in R$, a target set $C_v$. Such selection has to guarantee that the subsequent access protocol, where packets for the copies in $\bigcup_v C_v$ are routed through the network, can be performed efficiently.

The $C_v$ are selected by a procedure called CULLING. This procedure consists of $k$ iterations, in which for each variable $v \in R$ a conveniently chosen initial subset $C_v^0$ of copies of $v$ is progressively shrunk to reduce the congestion of the induced packet routing problem. Let

$C_v^i$, $0 \leq i \leq k$, denote the set of selected copies after the $i$th iteration of CULLING, where $C_v^k = C_v$ is the final set of copies that will be actually requested in the following access protocol. For reasons that will be made clear by the analysis, we need $C_v^i$ to contain enough copies of $v$ to guarantee *extensive access at level $i$* to $v$. This notion of access is defined on the nodes of $\mathcal{T}_v$ and coincides with Definition 2, for the nodes up to level $i-1$, but requires that a node at level $j \geq i$ be considered extensively accessed only when *more* than a majority of its children (i.e., at least $\lfloor q/2 \rfloor + 2$) are extensively accessed. A *level-$i$ target set* for a variable $v$ is a set of copies that grants extensive access at level $i$. A level-$i$ target set is called *minimal* if it does not properly include another level-$i$ target set. Clearly, a minimal level-$i$ target set contains a target set.

Procedure CULLING is executed in parallel by the $n$ processors of the mesh, and maintains, for each variable $v \in R$, the following invariants:

- $C_v^i \subseteq C_v^{i-1}$, for $0 < i \leq k$;
- $C_v^i$ is a minimal level-$i$ target set for $v$, for $0 \leq i \leq k$.

The code for the procedure is given below.

```
procedure CULLING;
  begin
    for each variable v ∈ R do
      Set C_v^0 to be a minimal level-0 target set for v;
    endfor;
    for i := 1 to k do
      for each level-i page P do
        Mark at most 2q^k n^{1-1/2^i} arbitrary copies of ⋃_v C_v^{i-1} belonging to P
      endfor;
      for each variable v ∈ R do
        Let M_v^i ⊆ C_v^{i-1} be the set of marked copies for v;
        if M_v^i contains a level-i target set for v
          then extract a minimal level-i target set C'^{i-1}_v from M_v^i
          else
            Mark a set S_v^i ⊆ C_v^{i-1} − M_v^i such that S_v^i ∪ M_v^i contains a level-i target set;
            Extract a minimal level-i target set C'^{i-1}_v from S_v^i ∪ M_v^i
        endif;
        C_v^i := C'^{i-1}_v;
      endfor
    endfor
    C_v := C_v^k
  end.
```

As we noticed before, the goal of the procedure is to select a target set for each variable so that when the copies are requested in the access protocol, the congestion in the network is conveniently bounded. More specifically, the $i$th iteration of CULLING aims at reducing the number of requests destined to any level-$i$ page, which is assigned to a submesh of the $i$th tessellation. This is stated in the following theorem:

**Theorem 3** *For each level-$i$ page $\mathcal{P}$, the number of copies in $\bigcup_v C_v^i$ belonging to $\mathcal{P}$ is at most $4q^k n^{1-1/2^i}$, $0 \leq i \leq k$.*

**Proof:** The proof goes by induction on $i \geq 0$. For $i = 0$ the statement is trivial. Suppose there is a level-$i$ page $\mathcal{P}$ containing more than $4q^k n^{1-1/2^i}$ copies in $\bigcup_v C_v^i$. Since the algorithm places at most $2q^k n^{1-1/2^i}$ copies from any page in $\bigcup_v M_v^i$, there are more than $2q^k n^{1-1/2^i}$ copies in $\mathcal{P}$ belonging to $\bigcup_v S_v^i$. Such copies are relative to variables $v'$ for which $M_{v'}^i$ does not contain a minimal level-$i$ target set.

Since each level-$(i-1)$ page, by the inductive hypothesis, contains at most $4q^k n^{1-1/2^{i-1}}$ copies of $\bigcup_v C_v^{i-1} \supseteq \bigcup_v S_v^i$, there are at least

$$\frac{2q^k n^{1-1/2^i}}{4q^k n^{1-1/2^{i-1}}} = \frac{n^{1/2^i}}{2}$$

level-$(i-1)$ pages in $\mathcal{P}$ that contain copies in $\bigcup_v S_v^i$. Note that these are pages of distinct level-$(i-1)$ modules. Consider any such module $u$. The fact that a page of $u$ contains copies in $\bigcup_v S_v^i$ implies that there exists a variable $v'$ such that a node in $\mathcal{T}_{v'}$ labeled with $u$ is not accessed when only the copies in $M_{v'}^i$ are reached, but it is extensively accessed when the copies in $C_{v'}^{i-1} \supseteq M_{v'}^i$ are reached (recall that $C_{v'}^{i-1}$ is a minimal level-$(i-1)$ target set for $v'$, whereas $C_{v'}^i$ has to be a minimal level-$i$ target set for $v'$). Therefore, there must be at least two pages of $u$ that contain copies of $v'$ that are in $C_{v'}^{i-1}$ but not in $M_{v'}^i$. Applying the strong expansion property (Lemma 1), we conclude that there are more than $n^{1/2^i}/2$ level-$i$ pages belonging to distinct level-$i$ modules containing copies in $\bigcup_v (C_v^{i-1} - M_v^i)$. Any such page contributes exactly $2q^k n^{1-1/2^i}$ distinct copies to $\bigcup_v M_v^i$, hence

$$|\bigcup_v M_v^i| > \frac{n^{1/2^i}}{2} \cdot 2q^k n^{1-1/2^i} = q^k n,$$

a contradiction. □

Before concluding this section, we analyze the time complexity of CULLING. As we said earlier, the procedure is executed in parallel by the $n$ processors of the mesh, each processor being in charge of a distinct variable. It is easy to see that the selection of the copies at the $i$th iteration can be accomplished by first sorting the copies according to their destination level-$i$ page and then ranking the copies destined to the same page. Since there are at most $nq^k$ copies, sorting and ranking take $O(q^k\sqrt{n})$ time. Also, each processor must check whether a level-$i$ target set for its variable is included in the selected copies, and then must extract a minimal target set. This can be done in $O(q^k)$ time. Since there are $k$ iterations, the time complexity of CULLING becomes

$$T_{\text{culling}} \in O(kq^k\sqrt{n}), \tag{2}$$

and each processor uses $O(q^k)$ internal storage.

### 3.3 The Access Protocol

After progressive culling is completed, the selected copies (set $\bigcup_v C_v$) have to be accessed. Each copy is requested by a distinct packet, routed from the requesting processor (*origin*) to the processor storing the copy (*destination*), and back to the origin. The idea is to route the packets in stages

so that they are moved gradually closer to their destinations, in accordance with the tessellations defined on the mesh. Such strategy results more profitable than simply sending the packets directly to their destinations, since the culling procedure provided us with bounds on the maximum number of packets destined to any level-$i$ page, $1 \leq i \leq k$, and such bounds can be exploited to limit the congestion in each stage.

We first describe in detail how the HMOS is mapped onto the nodes of the mesh. Let $m_i$ denote the number of level-$i$ modules (i.e., $m_i = |U_i|$), $1 \leq i \leq k$. For uniformity, set $m_0 = |V| = n^\alpha$. By (1), we have that, if $q$ is constant,

$$m_i \in \Theta(n^{\alpha/2^i}), \qquad 0 \leq i \leq k.$$

Also, let $p_i$ denote the number of level-$(i-1)$ pages contained in a level-$i$ module, $1 \leq i \leq k$, where level-0 pages are the copies of the variables. By the definition of $\mathcal{G}$, the graph between $U_{i-1}$ and $U_i$ is a subgraph of a $(q^{d_i}, q)$-BIBD, where some of the inputs nodes have been removed. In the Appendix, we will show how such subgraph can be chosen so that each level-$i$ module receives almost the same number of level-$(i-1)$ pages, that is

$$p_i \in \left\{ \lceil \frac{qm_{i-1}}{m_i} \rceil, \lfloor \frac{qm_{i-1}}{m_i} \rfloor \right\}. \tag{3}$$

Define the outermost tessellation as a subdivision of the mesh into *level-k submeshes* of size

$$t_k = \frac{n}{m_k} \in \Theta(n^{1-\alpha/2^k}),$$

associated with the distinct level-$k$ modules. Each such submesh $\mathcal{M}$ is in turn subdivided into $p_k$ *level-$(k-1)$ submeshes*, which are associated with the $p_k$ pages contained in the module assigned to $\mathcal{M}$. The size of each level-$(k-1)$ submesh is

$$t_{k-1} = t_k \frac{1}{p_k}.$$

Level-$(k-1)$ submeshes are in turn subdivided into *level-$(k-2)$ submeshes*, and so forth. In general, for $1 \leq i < k$, a level-$(i+1)$ submesh is subdivided into $p_{i+1}$ *level-$i$ submeshes* of size

$$t_i = \frac{n}{m_k} \frac{1}{p_k \cdot p_{k-1} \cdots p_{i+1}},$$

each assigned to a distinct level-$i$ page. By applying (1) and (3) we get

$$t_i \in \Theta\left(\frac{n}{q^{k-i} m_i}\right) = \Theta\left(q^{-(k-i)} n^{1-\alpha/2^i}\right) \qquad 1 \leq i \leq k. \tag{4}$$

Note that if $\alpha < 2\left(1 - \frac{k-1}{\log_q n}\right)$ then $t_i \geq 1$ for every $i \geq 1$. A level-1 page stores $p_1 \in \Theta(qn^{\alpha/2})$ copies of distinct variables, which are evenly distributed among $t_1$ processors; therefore, each processor stores $p_1/t_1 \in \Theta(q^k n^{\alpha-1})$ copies.

We are now ready to present the access protocol. First, each processor generates distinct request packets for the copies of its variable which have been selected by the culling procedure. Then, each

packet is routed from its origin to its destination, where the access takes place, and then back to the origin, carrying the result of the access. Consider the origin-destination part of the journey. It consists of $k+1$ routing stages, numbered, for convenience, from $k+1$ down to 1. Stage $i$, $k+1 \geq i \geq 1$, is executed in parallel and independently in every level-$i$ submesh (here, the original mesh is viewed as a level-$(k+1)$ submesh). In Stage $i$, for $k+1 \geq i > 1$, the packets are routed to arbitrary positions within the level-$(i-1)$ submeshes containing their destination, in such a way that the processors of each submesh receive, approximately, the same number of packets. This can be achieved by first sorting the packets according to their destination submeshes, and ranking the packets destined to the same submesh so that they can be evenly distributed among the processors of the submesh. Finally, in Stage 1, each packet is sent to its destination and its request is satisfied. Then the packets return to their origins following the same path through the $k$ intermediate nodes whose addresses have been recorded along the way.

We analyze now the time complexity of the origin-destination routing. Let $\delta_i$, $k+1 \geq i \geq 1$, denote the (maximum) number of packets held by any processor at the beginning of Stage $i$. Clearly, $\delta_{k+1} \leq q^k$. For $i \leq k$, $\delta_i$ can be seen as the number of packets received by any processor at the end of Stage $i+1$. By Theorem 3, a level-$i$ page is addressed by at most $4q^k n^{1-1/2^i}$ packets, and since there are $t_i$ processors assigned to each level-$i$ page, we have

$$\delta_i \leq \frac{4q^k n^{1-1/2^i}}{t_i} \in \Theta\left(q^{2k-i} n^{(\alpha-1)/2^i}\right) \qquad k \geq i \geq 1. \tag{5}$$

Set $t_{k+1} = n$ to denote the size of the entire mesh. For $k+1 \geq i > 1$ the time complexity of Stage $i$ is given by

$$\begin{aligned} T_i &\in O\left(\delta_i \sqrt{t_i} + \left(\sqrt{\delta_i \delta_{i-1}} + \delta_i\right) \sqrt{t_i}\right) \\ &\in O\left(\delta_i \sqrt{t_i} + \sqrt{\delta_i \delta_{i-1} t_i}\right), \end{aligned}$$

where the term $\delta_i \sqrt{t_i}$ is the complexity of sorting and ranking, and the term $\left(\sqrt{\delta_i \delta_{i-1}} + \delta_i\right) \sqrt{t_i}$ is the complexity of the $(\delta_i, \delta_{i-1})$-routing problem involved in this stage. As for the complexity of Stage 1, note that at the beginning each processor holds $\delta_1$ packets. Furthermore, since a level-1 page has at most $4q^k n^{1/2}$ packets (Theorem 3) and a processor stores $\Theta(q^k n^{\alpha-1})$ copies, each processor will receive at most

$$\delta_0 \in O(q^k \min(n^{1/2}, n^{\alpha-1})). \tag{6}$$

packets. Since this stage consists only of an instance of $(\delta_1, \delta_0)$-routing, its complexity is

$$T_1 \in O\left(\left(\sqrt{\delta_1 \delta_0} + \delta_1\right) \sqrt{t_1}\right).$$

Noting that the complexity of the first part of the routing (origin-destination) dominates that of the second part (destination-origin), we conclude that the complexity of the entire access protocol is

$$T_{\text{protocol}} = \sum_{i=k+1}^{1} T_i.$$

Assuming $1 < \alpha \leq 2\left(1 - \frac{k-1}{\log_q n}\right)$ and applying (4), (5) and (6), we get

$$\begin{aligned} T_{k+1} &\in O\left(q^k n^{\frac{1}{2}+\frac{\alpha-1}{2^{k+1}}}\right); \\ T_i &\in O\left(q^{\frac{3k-i+1}{2}} n^{\frac{1}{2}+\frac{2\alpha-3}{2^{i+1}}}\right) \qquad k \geq i \geq 2; \\ T_1 &\in O\left(q^k n^{\frac{1}{2}}\right). \end{aligned}$$

Therefore,

$$T_{\text{protocol}} \in O\left(q^k n^{\frac{1}{2}+\frac{\alpha-1}{2^{k+1}}} + \sum_{i=2}^{k}\left(q^{\frac{3k-i+1}{2}} n^{\frac{1}{2}+\frac{2\alpha-3}{2^{i+1}}}\right)\right). \tag{7}$$

Finally, the total simulation time becomes

$$\begin{aligned} T_{\text{sim}} &= T_{\text{culling}} + T_{\text{protocol}} \\ &\in O\left(q^k n^{\frac{1}{2}}\left(k + n^{\frac{\alpha-1}{2^{k+1}}} + q^{\frac{k+1}{2}} \sum_{i=2}^{k}\left(q^{-\frac{i}{2}} n^{\frac{2\alpha-3}{2^{i+1}}}\right)\right)\right). \end{aligned} \tag{8}$$

We are now able to state our main theorem.

**Theorem 4** *Given an n-processor PRAM with a shared memory of $n^\alpha$ cells, $1 < \alpha \leq 2$, one computational step can be simulated in time*

$$T(n) \in \begin{cases} n^{\frac{1}{2}+\epsilon} & \text{if } 0 < \epsilon < 1 \text{ and } \alpha \leq \frac{3}{2}; \\ n^{\frac{1}{2}+\frac{\alpha-1}{16}} & \text{if } \frac{3}{2} \leq \alpha \leq \frac{5}{3}; \\ n^{\frac{1}{2}+\frac{2\alpha-3}{8}} & \text{if } \frac{5}{3} \leq \alpha \leq 2, \end{cases}$$

*with constant redundancy, and*

$$T(n) \in n^{\frac{1}{2}} polylog(n) \quad \text{if } \alpha \leq \frac{3}{2},$$

*with polylog(n) redundancy.*

*The simulating machine is a square mesh of n nodes. The scheme is deterministic and fully constructive.*

**Proof:**

Fixed the value for $\alpha$, Formula (8) allows one to choose the parameters $q$ and $k$ so that the simulation time $T_{\text{sim}}$ and the redundancy $q^k$ (i.e., the number of copies per variable) are bounded as stated by the theorem. First note that both $T_{\text{sim}}$ and $q^k$ are increasing functions of $q$, therefore we use the smallest possible $q$ which, as it turns out from the previous analysis, is $q = 3$. Also recall that $1 < \alpha \leq 2\left(1 - \frac{k-1}{\log_q n}\right)$. In the following, the logarithms are taken to the base 2, unless differently specified.

Let us first consider schemes with constant redundancy. Fix $\alpha \leq \frac{3}{2}$ and let $\epsilon$ be a constant, $0 < \epsilon < 1$. Then if

$$k = \left\lceil \log\left(\max\left(2, \frac{\alpha-1}{2\epsilon}\right)\right)\right\rceil,$$

we get from (8)

$$\begin{aligned} q^k &\in \Theta(1); \\ T_{\text{sim}} &\in O\left(n^{\frac{1}{2}+\epsilon}\right). \end{aligned}$$

For $\alpha \geq \frac{3}{2}$ and $k = 2$, (8) instead yields

$$T_{\text{sim}} \in O\left(n^{\frac{1}{2}+\frac{\alpha-1}{8}}\right). \tag{9}$$

Note that if the largest possible value for $\alpha$, i.e., $\alpha = 2\left(1 - \frac{k-1}{\log_q n}\right)$, is used in the above example, we get that $n^\alpha \in \Theta(n^2)$, the redundancy is $q^2 = 9$ and $T_{\text{sim}} \in O(n^{5/8})$. For other values of $\alpha \geq \frac{3}{2}$, however, $T_{\text{sim}}$ is minimized for $k = 3$ and $q^3 = 27$ copies per variable. Indeed we have:

$$T_{\text{sim}} \in \begin{cases} O\left(n^{\frac{1}{2}+\frac{\alpha-1}{16}}\right) & \text{if} \quad \frac{3}{2} \leq \alpha \leq \frac{5}{3}, \\ O\left(n^{\frac{1}{2}+\frac{2\alpha-3}{8}}\right) & \text{if} \quad \alpha \geq \frac{5}{3}. \end{cases} \tag{10}$$

Faster running times can sometimes be obtained when more than constant redundancy is allowed. Namely, let $\alpha \leq \frac{3}{2}$ and let $k = \lceil k' \rceil$, where $k'$ is such that

$$q^{\frac{k'+1}{2}} = n^{\frac{\alpha-1}{2^{k'+1}}}.$$

Observe that $k \in O\left(\log\left(\frac{\log n}{\log\log n}\right)\right) \in O(\log\log n)$. We have

$$q^k \in O\left(\left(\frac{\log n}{\log\log n}\right)^{\log_2 3}\right) \simeq O\left(\left(\frac{\log n}{\log\log n}\right)^{1.59}\right),$$

and

$$T_{\text{sim}} \in O\left(n^{\frac{1}{2}}\left(\frac{\log n}{\log\log n}\right)^{\frac{3}{2}\log_2 3}\right) \simeq O\left(n^{\frac{1}{2}}\left(\frac{\log n}{\log\log n}\right)^{2.38}\right).$$

For slighly smaller memories we can do even better. Let $k = \lceil k' \rceil$, where $k'$ is such that

$$k' = n^{\frac{\alpha-1}{2^{k'+1}}}.$$

Observe that $k \in O\left(\log\left(\frac{\log n}{\log\log\log n}\right)\right) \in O(\log\log n)$. Then, for $\alpha \leq \frac{3}{2} - \frac{1}{\log_q k}$, we have

$$\begin{aligned} q^k &\in O\left(\left(\frac{\log n}{\log\log\log n}\right)^{\log_2 3}\right) \simeq O\left(\left(\frac{\log n}{\log\log\log n}\right)^{1.59}\right); \\ T_{\text{sim}} &\in O\left(n^{\frac{1}{2}}\left(\frac{\log n}{\log\log\log n}\right)^{\log_2 3}\log\log n\right) \simeq O\left(n^{\frac{1}{2}}\left(\frac{\log n}{\log\log\log n}\right)^{1.59}\log\log n\right). \end{aligned}$$

□

## Acknowledgements

This paper benefited from discussions with Matteo Frigo, Tim Harris and Franco Preparata.

## References


[AHMP87] H. Alt, T. Hagerup, K. Mehlhorn, and F.P. Preparata. Deterministic simulation of idealized parallel computers on more realistic ones. *SIAM J. on Computing*, 16(5):808–835, 1987.

[AS90] Y. Aumann and A. Schuster. Improved memory utilization in deterministic PRAM simulations. Manuscript, 1990.

[CW79] J.L. Carter and M.N. Wegman. Universal classes of hash functions. *J. of Computers and System Sci.*, 18:143–154, 1979.

[DM93] M. Dietzfelbinger and F. Meyer auf der Heide. Simple, efficient shared memory simulations. *Proc. of the 5nd ACM Symp. on Parallel Algorithms and Architectures*, pages 110–118, 1993.

[Gif79] D.K. Gifford. Weighted voting for replicated data. *Proc. of the 7th ACM Symp. on Operating System Principles*, pages 150–159, 1979.

[Hal86] M. Hall Jr. *Combinatorial Theory.* John Wiley & Sons, New York NY, second edition, 1986.

[HB88] K.T. Herley and G. Bilardi. Deterministic simulations of PRAMs on bounded degree networks. *Proc. of the 26th Annual Allerton Conference on Communication, Control and Computation*, pages 1084–1093, 1988.

[Her89] K.T. Herley. Efficient simulations of small shared memories on bounded degree networks. *Proc. of the 30th IEEE Symp. on Foundations of Comp. Sc.*, pages 390–395, 1989.

[Her90a] K.T. Herley. Deterministic simulation of shared memory on bounded degree networks. *Tech. Rep. TR90-1090, Cornell University, Ithaca*, 1990.

[Her90b] K.T. Herley. Space-efficient representations of shared data for parallel computers. *Proc. of the 2nd ACM Symp. on Parallel Algorithms and Architectures*, pages 407–416, 1990.

[KLM92] R. Karp, M. Luby, and F. Meyer auf der Heide. Efficient PRAM simulation on distributed machines. *Proc. of the 24th ACM Symp. on Theory of Comp.*, pages 318–326, 1992.

[KSS94] M. Kaufmann, J.F. Sibeyn, and T. Suel. Derandomizing routing and sorting algorithms for meshes. *Proc of the 5th ACM-SIAM Symp. on Discrete Algorithms*, 1994. To appear.

[KU88] A.R. Karlin and E. Upfal. Parallel hashing: An efficient implementation of shared memory. *J. ACM*, 35(4):876–892, 1988.

[Kun93] M. Kunde. Block gossiping on grids and tori: Deterministic sorting and routing match the bisection bound. *Proc. of the 1st European Symp. on Algorithms*, pages 272–283, 1993.

[LPP88] F. Luccio, A. Pietracaprina, and G. Pucci. A probabilistic simulation of PRAMs in VLSI. *Information Processing Lett.*, 28(3):141–147, 1988.

[LPP90] F. Luccio, A. Pietracaprina, and G. Pucci. A new scheme for the deterministic simulation of PRAMs in VLSI. *Algorithmica*, 5:529–544, 1990.

[Mey92] F. Meyer auf der Heide. Hashing strategies for simulating shared memory on distributed memory machines. *Proc. 1st Heinz Nixdorf Symp. on Parallel Architectures and their Efficient Use*, 1992. To appear in LNCS.

[MV84] K. Mehlhorn and U. Vishkin. Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memories. *Acta Informatica*, 9(1):29–59, 1984.

[PP93a] A. Pietracaprina and F.P. Preparata. An $O(\sqrt{n})$-worst-case-time solution to the granularity problem. *Proc. of the 10th Symp. on Theoretical Aspects of Comp. Sc.*, LNCS 665:110–119, 1993.

[PP93b] A. Pietracaprina and F.P. Preparata. A practical constructive scheme for deterministic shared-memory access. *Proc. of the 5nd ACM Symp. on Parallel Algorithms and Architectures*, pages 100–109, 1993.

[Ran91] A.G. Ranade. How to emulate shared memory. *J. of Computers and System Sci.*, 42:307–326, 1991.

[SK93] J.F. Sibeyn and M. Kaufmann. 1-$k$ routing on meshes, with application to hot-potato worm-hole. (submitted to STACS94), 1993.

[Tho79] R.H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *ACM Transactions on Databases Systems*, 4(2):180–209, 1979.

[UW87] E. Upfal and A. Widgerson. How to share memory in a distributed system. *J. ACM*, 34(1):116–127, 1987.

# Appendix

We show here how to contruct a bipartite graph $G = (V, U; E)$ which is a subgraph of a $(q^d, q)$-BIBD with the same number of output nodes, i.e., $|U| = q^d$, fewer input nodes, say $|V| = m$, with $1 \leq m < q^{d-1}\frac{q^d-1}{q-1}$, and such that each input $v \in V$ has degree $q$, as in the BIBD, and each output $u \in U$ has degree $\rho$,

$$\left\lfloor \frac{qm}{q^d} \right\rfloor \leq \rho \leq \left\lceil \frac{qm}{q^d} \right\rceil .$$

In other words, we want to select a subset of inputs from the BIBD, whose edges are distributed among the outputs as evenly as possible. This can be done by modifying the construction given in [PP93a]. Let $q$ be a prime power and let $\mathbb{F}_q$, denote the finite field with $q$ elements, with its elements represented by the integers $0, 1, \ldots, q-1$. Recall that the $(q^d, q)$-BIBD constructed in [PP93a] has the outputs associated with the set of $d$-dimensional vectors over $\mathbb{F}_q$, and the inputs with the $q^{d-1}\frac{q^d-1}{q-1}$ pairs ov vectors of kind

$$\begin{array}{ccccccccccccccc} ( & a_{d-2} & , & \ldots & , & a_h & , & 0 & , & a_{h-1} & , & \ldots & , & a_1 & , & a_0 & ) \\ ( & 0 & , & \ldots & , & 0 & , & 1 & , & b_{h-1} & , & \ldots & , & b_1 & , & b_0 & ). \end{array}$$

Let each such pair be denoted by $\Phi(h, A, B)$, where $h$ is an index between 0 and $d-1$, $A$ is the integer in $[0, q^{d-1})$ whose representation in base $q$ is $(a_{d-2} \ldots a_h a_{h-1} \ldots a_1 a_0)$, and $B$ is the integer in $[0, q^h)$ whose representation in base $q$ is $(b_{h-1} \ldots b_1 b_0)$. The graph $G$ is obtained from this BIBD by taking the same output set and selecting a subset of $m$ inputs, as follows. Let $l < d$ be the index such that

$$q^{d-1}\frac{q^l - 1}{q-1} \leq m < q^{d-1}\frac{q^{l+1} - 1}{q-1},$$

so that

$$m = q^{d-1}\left(\frac{q^l - 1}{q-1} + w\right) + z, \tag{11}$$

for some $w$, $0 \leq w < q^l$ and $z$, $0 \leq z < q^{d-1}$. The $m$ pairs $\Phi(h, A, B)$ that we select to represent the nodes of $V$ consist of the union of the three sets $V_1, V_2$ and $V_3$ defined below:

$$\begin{aligned} V_1 &= \left\{\Phi(h, A, B) : 0 \leq h < l,\ 0 \leq A < q^{d-1},\ 0 \leq B < q^h\right\}; \\ V_2 &= \left\{\Phi(h, A, B) : h = l,\ 0 \leq A < q^{d-1},\ 0 \leq B < w\right\}; \\ V_3 &= \left\{\Phi(h, A, B) : h = l,\ 0 \leq A < z,\ B = w\right\}. \end{aligned}$$

It is easy to verify that $|V_1| + |V_2| + |V_3| = m$.

The edges are those incident on the selected nodes in the original BIBD, that is, a node $v \in V$ associated with the pair

$$\begin{array}{ccccccccccccccc} ( & a_{d-2} & , & \ldots & , & a_h & , & 0 & , & a_{h-1} & , & \ldots & , & a_1 & , & a_0 & ) \\ ( & 0 & , & \ldots & , & 0 & , & 1 & , & b_{h-1} & , & \ldots & , & b_1 & , & b_0 & ), \end{array}$$

will be connected to the $q$ output nodes

$$(a_{d-2}, \ldots, a_h, x, a_{h-1} + x \cdot b_{h-1}, \ldots, a_1 + x \cdot b_1, a_0 + x \cdot sb_0)$$

for every $x \in \mathbb{F}_q$, where $+$ and $\cdot$ are operations in the field.

We have to show that the edges are evenly distributed among the outputs.

**Theorem 5** *Any node $u \in U$ is connected to $\rho$ nodes of $V$, where*

$$\left\lfloor \frac{qm}{q^d} \right\rfloor \leq \rho \leq \left\lceil \frac{qm}{q^d} \right\rceil .$$

**Proof:** Let $u$ be associated with the vector $(a_{d-1}, \ldots, a_0)$. We determine the value of $\rho$ by counting separately the contributions of the nodes in the three subsets $V_1, V_2$ and $V_3$. Consider $V_1$ and fix $h < l$. Using the properties of field operations, one can easily show that for any $B$, $0 \leq B < q^h$, there exists exactly one value $A$ such that the node $\Phi(h, A, B)$ is connected to $u$. Therefore, there are exactly $\sum_{s=0}^{l-1} q^h = \frac{q^l-1}{q-1}$ nodes of $V_1$ connected to $u$. A similar arguments shows that exactly $w$ nodes of $V_2$ are connected to $u$. Finally, it can be seen that the $z$ nodes of $V_3$ are connected to $qz$ distinct output nodes, therefore, according to whether $u$ is one of such nodes or not, we have that either $\rho = \frac{q^l-1}{q-1} + w$ or $\rho = \frac{q^l-1}{q-1} + w + 1$. By (11) we conclude that

$$\left\lfloor \frac{qm}{q^d} \right\rfloor \leq \rho \leq \left\lceil \frac{qm}{q^d} \right\rceil .$$

□