Electronic Colloquium on Computational Complexity, Report No. 53 (2000)

# Parallel Read Operations Without Memory Contention [*]

Alexander E. Andreev
LSI Logic
(CA) USA

Andrea E. F. Clementi[†]
Dipartimento di Matematica
"Tor Vergata" University of Rome

Paolo Penna
Dipartimento di Matematica
"Tor Vergata" University of Rome

José D.P. Rolim
Centre Universitaire d'Informatique
University of Geneva

May 5, 2000

**Abstract**

We address the problem of organizing a set $T$ of shared data into the memory modules of a *Distributed Memory Machine* (DMM) in order to minimize memory access conflicts (i.e. memory contention) during read operations. Previous solutions for this problem can be found as fundamental subprocedures of the PRAM simulation methods on DMM presented during the last years. The efficiency of such solutions relies on the assumption that the set of shared data is relatively small. Indeed, each shared data is replicated in at least two copies; moreover, the number of processors and that of memory modules are polynomial in the number of the shared data. This assumption is reasonable to the aim of PRAM simulations (where the shared data consist only on the shared program variables) but it is not realistic in the case of parallel systems for large public-accessible databases where the number of available resources (such as processors and memory modules) is tipically significantly (say exponentially) smaller than the size of the database.

As for the latter case, we present a new randomized scheme that given any positive parameter $\beta$, with high probability, performs any set of $r$ unrelated read operations on the shared data set $T$ in $O(\log r + \log \beta)$ parallel time *with no memory contention* using $r$ processors (where each processor consists of $O(r^3 + \beta)$ Boolean gates of fan-in two). The set $T$ is distributed into $m$ DMM memory modules where $m$ is polynomial in $r$ and $\beta$, and the overall size of the shared memory used by our scheme is not larger than $(1 + 1/(\beta))|T|)$ (this means that the *data replication* can be made arbitrarily small). Our solution is thus very efficient in the case of a large number of shared data. Furthermore, the memory organization scheme and most part of all the computations *do not depend* on the read requests, so they can be performed once and for all during an off-line phase.

[*]This paper merges and revises the results presented at the *IV IEEE - Israel Symposium on Theory of Computing and Systems 1996 (ISTCS'96)* and at the *XVI Symposium on Theoretical Aspects of Computer Science 1999 (STACS'99)*.

[†]Contact author: E-mail: clementi@mat.uniroma2.it Fax ++39-6-854-1842.

ISSN 1433-8092

## 1 Introduction

Consider a shared-memory synchronous parallel machine in which a set of $p$ processors can access to a set of $b$ external *memory modules* (also called *banks*) in parallel, provided that a memory module is not accessed by more than one processor simultaneously. The $p$ processors are connected to the memory modules through a switching network. This parallel model, commonly referred to as *Distributed Memory Machine* (DMM) or *Module Parallel Computer*, is considered more realistic than the PRAM model and it has been the subject of several studies in the literature [Karp et al 96, Karlin et al 86, Mehlhorn et al 84, Pietracaprina et al 93, Upfal 84]. In an EREW PRAM, each of the $p$ processors can in fact access any of the $N$ memory words, provided that a word is not accessed by more than one processor simultaneously. To ensure such connectivity, the total number of the switching elements must be $\Theta(pN)$. For large shared memory, constructing a switching network of this complexity is very expensive or even impossible in practice.

One standard way to reduce the complexity of the switching network is to organize the shared memory in *modules* [Kumar et al 95], each of them containing several words. A processor switches between modules and not between individual words. So, if the total number of modules is $b << N$ we then need only $\Theta(pb)$ switching elements to realize the interconnection network. There are two fundamental problems that always arise when the DMM model is adopted. The first one concerns the construction of feasible switching networks and related routing algorithms that must guarantee a full connectivity between processors and memory modules with a sufficiently small delay. Several randomized and deterministic solutions of this problem have been derived over the last years (see [Leighton 92] for a good survey).

Once the routing problem is efficiently solved, the shared data have to be distributed (and, if necessary, replicated) among the set of memory modules so that the $p$ processors can access them avoiding simultaneous reading accesses on the same memory module. This second problem is sometimes referred to as the *granularity problem*. The importance of this problem lies in the fact that reading conflicts on the same shared-memory module (and, in general, any operation that generates conflicts in the use of shared external devices) can only be solved at the cost of a significant time delay. So, the *memory contention*, i.e. the maximum number of shared accesses simultaneously mapped into the same module, is one of the most important factors of the overall time complexity of a DMM algorithm.

In this paper, we address the granularity problem by assuming that we have at hand a sufficiently good solution for the routing problem and thus processors - memory modules communication is efficiently achieved. We also assume that our DMM model is provided by *memory interleaving* technology [Kumar et al 95] that allows the CPU of any processor to send access requests to more than one memory module simultaneously, provided that each of these modules is not used by other processors.

Based on the above assumptions, several works have been devoted to the design of efficient solutions for the granularity problem. In particular, randomized solutions have been presented in [Karp et al 96, Mehlhorn et al 84, Upfal 84], while deterministic but less efficient solutions have been introduced in [Pietracaprina et al 93, Upfal et al 87]. Most of these works concern the problem of simulating a PRAM algorithm on a DMM model. Further relevant applications of the granularity problem concern the design of parallel systems for Private Information Retrieval (PIR) on public-accessible databases [Chor et al 97, Chor et al 95, Gertner et al 98], parallel routing-table computations [Konstantinidou et al 90, Pluris 98, WPluris 98] for IP lookup, and matrix representations in parallel image processing (skewing schemes) [Liu et al 92, Frailong et al 85].

Let $T$ be the table of the binary data to be shared and $r$ ($1 \leq r \leq p$) be the number of parallel read requests to be satisfied. The performance analysis of any solution of the granularity problem on the DMM model should mainly address the following aspects. *i*) The size of the shared memory required to implement the original table $T$. *ii*) The number of memory modules in which $T$ is distributed. *iii*) The *memory contention*, i.e., the maximum number of simultaneous read requests sent to a single memory module. *iv*) The parallel time complexity. *v*) The circuit-size complexity of the processors.

Clearly, the role of the first aspect becomes crucial when large data tables have to be shared (this is the case, for instance, of public databases accessible, say, on WWW servers). On the other hand, in the case of PRAM simulations, it is reasonable to assume that the number of shared data is relatively small [Pietracaprina et al 93]. The parallel time complexity depends on the *local computations* performed by each processor, and on the memory contention. As discussed before, it is reasonable to assume that the latter represents a dominant factor.

To our knowledge, the best randomized solution for the granularity problem in the case of PRAM simulation on the DMM model (notice that in the randomized case, it is assumed that the DMM processors have access to a public pseudo-random generator) has been introduced by Karp *et al* in [Karp et al 96]. They indeed derived a randomized method that simulates a PRAM with $p \log\log p \log^* p$ processors by using a DMM with $p$ processors with optimal expected delay time $O(\log\log p \log^* p)$ per step of simulation. In one (the best) version of their solution the memory contention is $O(\log\log p)$ and each of the shared data is replicated in $c \geq 2$ copies and mapped to $p$ memory modules by means of *c hash* functions (see also [Karlin et al 86]). It is important to remark that such hash functions have sufficiently good random properties (and, thus, they work correctly) only if the size of the domain of the mapped data is polynomial in $p$. This is one of the reasons for which their randomized simulation determines a partition of the generic PRAM algorithm into a sequence of consecutive phases to be executed by the DMM; each phase simulates $O(p^{1/10})$ steps of the PRAM algorithm and all the variables (observe that this number is at most $O(p^{11/10})$) used during these steps are distributed into the memory modules by using new randomly selected hash functions. This random distribution is called the *cleaning up* task. Such a solution thus requires the *a priori* knowledge of the data used during the generic phase and which must be assigned to the memory modules.

From the above discussion, it follows that Karp *et al* solution is unfeasible when: *i*) the number of shared data is much (say exponentially) larger than the number $r \leq p$ of parallel read requests; *ii*) it is not possible to determine which is the set of shared data that will be requested in the next phase.

These are the typical situations that arise in the case of concurrent accesses to large data structures such as public database available on WWW servers and IP routers databases where non predictable read requests have to be satisfied in "real time"[1]. Finally, we emphasize that an application of Karp *et al*'s method for these instances of the granularity problem would imply a new assignment of the sharing data to the memory modules (during the cleaning up task) for each new set of read requests.

[1] Observe that in these cases read operations on the memory modules are much more frequent than write operations.

## 1.1 Our Results

We address the problem of organizing a binary database $T$ of size $N = 2^n$ into the shared memory of a $p$-processors DMM in order to perform any set of $r \leq p$ unrelated read accesses on $T$ by a subset of $r$ processors. A trivial deterministic solution that has no memory contention would be that of storing $r$ copies of $T$ into $r$ different memory modules. In this case we would have memory size $rN$ which is clearly unfeasible for large databases. One of our constraints, instead, is that of using a shared memory of optimal size (i.e. $O(N)$).

Our solution consists of two main parts: an "off-line" *memory organization scheme* and an "on-line" randomized *reading procedure* .
The first part provides a parameterized deterministic method that distributes $T$ into the set of memory modules. Given any "data-replication" parameter $\beta > 1$, the overall memory size is $(1 + \frac{1}{\beta})N$ and the number of memory modules is polynomial in $r$, $\beta$ and the security parameter $\delta$ of the randomized reading procedure (so, it *does not depend* on $N$). This implies that an eventual implementation of a larger binary database requires only to increase the size of every memory module while keeping the same number of them. Furthermore, the memory organization (i.e. the distribution of $T$ into the memory modules) *does not depend* on the input requests and on the binary values of $T$ (so, the updating of $T$ never requires a new organization of the shared memory).
The second part consists of a randomized reading scheme that, given any security parameter $0 < \delta < 1/2$ and the parameter $\beta$ , performs, with probability at least $(1 - \delta)$, any set of $r$ unrelated read requests on the input table $T$ within the following complexity

- there is no memory contention;
- $O(\log r + \log \beta + \log(1/\delta))$ parallel time;
- the processors have circuit size $O((\beta/\delta)r^3)$;
- $O(\log r + \log \delta)$ random bits.

When $N$ (i.e. the amount of shared data) is large, the fact that *all the complexity measures do not depend on $N$* make our solution more efficient than those proposed for PRAM simulations. We also note that the procedure has the same performances even when all the $r$ processors ask the same bit. In the "unlikely" case of error (i.e. memory contention greater than 0), the randomized procedure does not fail to compute the read requests, however, the correct computation of the $r$ requests is delayed.

The on-line computation performed by any of the $r$ processors in order to select the subsets of memory modules which have to be accessed to satisfy its read request is simple and involves only elementary linear algebra on finite fields (more precisely, it is required to compute the set of points that belong to a line specified by one of its points and the parameters of its direction). Then, by using a simple multicast protocol, each of the $r$ processors sends the same request to *all* its selected modules.

We also provide a possible derandomization of our method. However, removing randomness from our algorithm yields a significant increasing of the circuit size and depth of the processors (and, thus, of the parallel time during the "on-line" phase). The main interest of the deterministic version lies in its consequences in circuit complexity. In fact, by re-stating the overall algorithm in terms of Boolean circuits, this version provides an efficent solution for computing an arbitrary finite Boolean function on a set of unrelated inputs, a fundamental problem which has been extensively studied in the literature [Bshouty 89, Nisan et al 94] (the problem is known as the *Direct Sum Problem*). Ulig [Ulig 74] proved that if $L(r,n)$ denotes the worst-case, circuit-size complexity to compute an $n$-variable boolean function on $r$ unrelated inputs then we have $L(r,n) = (1+o(1))L(1,n)$ for any $r = 2^{o(n/\log n)}$. Then, the deterministic version of our algorithm implies that Ulig's result holds for the wider case $r = 2^{o(n)}$ and that, for any $0 < \epsilon < \frac{1}{2}$ and $r \leq 2^{n(\frac{1}{2}-\epsilon)}$, it holds $L(r,n) = O(L(1,n))$. This theoretical consequence of our method is out of the scope of our paper, we will thus omit it and we refer the reader to [Andreev et al 96] for a detailed description. On the other hand, our randomized method can be efficiently used to implement *Private Information Retrieval* (PIR) schemes [Chor et al 97, Chor et al 95, Gertner et al 98] on the DMM model. This application will be discussed in Section 5.

## 2 The Memory Organization Scheme

Let $T$ be the binary database to be shared, and let $|T| = 2^n$ for some integer $n > 0$, we then consider $T$ as the output table of a finite Boolean function $f : \{0,1\}^n \to \{0,1\}$. According to this terminology, our problem is that of computing the function $f$ on a set of $r$ unrelated inputs.

In this section we introduce an algebraic method to represent finite Boolean functions. This method will be then used to distribute the shared database $T$ into the memory modules of the DMM. To this aim, we split the input space $\{0,1\}^n$ in the direct product of two subsets:

$$\{0,1\}^n = \{0,1\}^{4k} \times \{0,1\}^{n-4k}$$

(the correct choice of $k$ will be given later). The set $\{0,1\}^{4k}$ is here considered as the 4-dimensional linear space $GF(q)^4$ over the finite field $GF(q)$ where $q = 2^k$. It follows that $f$ can be written as $f(A,B)$, where $A \in GF(q)^4$, $B \in \{0,1\}^{n-4k}$, and our problem is now to compute the set of values

$$f(A_1,B_1), f(A_2,B_2), \ldots, f(A_r,B_r)$$

for arbitrary pairs $(A_i,B_i)$, $i = 1,\ldots,r$.

Let $l(A,u)$ be the line passing through $A$ and parallel to the vector $\vec{h}(u) = (1,u,u^2,u^3)$. Notice that the parameter $u$ determines the direction of the line. Let $U$ be any subset[2] of $GF(q)$. The term $SL_4(U)$ denotes the set of all lines $l(A,u)$ such that $A \in GF(q)^4$ and $u \in U$. We also define the set of points

$$l^{\#}(A,u) = l(A,u) \setminus \{A\}\ .$$

We are now ready to introduce our organization of $f$ in the shared memory. For any $A \in GF(q)^4$, consider the function $f_A : \{0,1\}^{n-4k} \to \{0,1\}$ defined as

$$f_A(B) \ = \ f(A,B)$$

furthermore, for each line $l \in SL_4(U)$, define the function $g_l : \{0,1\}^{n-4k} \to \{0,1\}$ as

$$g_l(B) = \bigoplus_{A \in l} f_A(B)\ .$$

[2] In our algorithm we will give a suitable choice of this subset, however, the results of this section hold for any $U \subseteq GF(q)$.

Each output table of these functions is stored in one single memory module of the DMM. Processors can specify the input of one of these Boolean functions and get one output bit. The time required to perform this external operation is denoted as extime. The maximum number of concurrent accesses on the same module is denoted as *memory contention*. Notice that the construction (more precisely, the size and the structure of the function tables) is independent of $f$ and of the sequence $(A_i, B_i)$, $i = 1, \ldots, r$. So, it can be performed off-line in a preliminary phase once and for all.

## 3 The Reading Procedure

In what follows, we will assume that the output table $T$ of $f$ is stored in the shared memory modules by means of functions $g_l$ and $f_A$ as described in the previous section. Let us first observe that, for any $A \in l$, it holds

$$f_A(B) = g_l(B) \bigoplus \left( \bigoplus_{A^* \in l \setminus \{A\}} f_{A^*}(B) \right), \ \forall\, B \in \{0,1\}^{n-4k}.$$

Informally speaking, the idea of our solution is that we can compute $f$ on a given input $(A_i, B_i)$ without using the memory module that contains $f_{A_i}$. Indeed, if we consider a set $\{u_1, \ldots, u_r\}$ of elements from $GF(q)$ then we can compute $f$ on $(A_i, B_i)$, $i = 1, \ldots, r$, by applying the following simple procedure :

```
• Procedure READ.
• input:
    – $(A_i, B_i)$, $i = 1, \ldots, r$;
    – $\{u_1, \ldots, u_r\}$ $(u_i \in GF(q))$ $i = 1, \ldots, r$;
• begin
• for any $i = 1, \ldots, r$ do
•     – begin
      – read $g_{l(A_i,u_i)}(B_i)$;
      – for any $A^* \in l^{\#}(A_i, u_i)$, read $f_{A^*}(B_i)$;
      – end
• for any $i = 1, \ldots, r$ compute
```

$$f(A_i, B_i) = g_{l(A_i,u_i)}(B_i) \bigoplus \left( \bigoplus_{A^* \in l^{\#}(A_i,u_i)} f_{A^*}(B_i) \right) ; \quad (1)$$

```
• return $f(A_i, B_i)$, for any $i = 1 \ldots r$.
• end.
```

The system in Eq. 1 suggests us a way to avoid memory contention in READ: it suffices to find a set of elements $\{u_1, \ldots, u_r\}$ such that any function of type $f_A$ or $g_l$ (and so any memory module) can participate only in one equation of the system. More formally, we need a sequence of elements $\{u_1, \ldots, u_r\}$ such that

$$\text{for any} \quad i \neq j \;\; l^{\#}(A_i, u_i) \bigcap l^{\#}(A_j, u_j) = \emptyset. \tag{2}$$

In the next section, we provide an efficient randomized solution for this searching problem while a possible derandomization of it will be discussed in Section 4.2.

## 3.1 Avoiding Memory Contention Via Randomness

The randomized algorithm that, on any input sequence $A_1, \ldots, A_r$, returns an output sequence $u_1, \ldots, u_r$ satisfying the condition in Eq. 2, enjoys of the following result.

**Lemma 3.1** *Let $d \geq 1$ and let $U$ be any subset of $GF(q)$ such that $|U| \geq dr^3$. Let*

$$M = [u_{i,j}], \;\; i = 1, \ldots, dr^2, \;\; j = 1, \ldots, r$$

*be a matrix of pairwise distinct elements from $U$. The probability that a randomly chosen index $i_0$ satisfies the property*

$$\textit{for any } j_1 \neq j_2, \;\; l^{\#}(A_{j_1}, u_{i_0,j_1}) \bigcap l^{\#}(A_{j_2}, u_{i_0,j_2}) \;=\; \emptyset \tag{3}$$

*is greater than $1 - \frac{1}{2d}$.*

*Proof.* The proof is by contradiction. Let us define the subset

$$\mathrm{BAD} = \{i \in \{1, \ldots, dr^2\} \mid \exists j_1(i) \neq j_2(i), l^{\#}(A_{j_1}, u_{i,j_1(i)}) \cap l^{\#}(A_{j_2}, u_{i,j_2(i)}) \neq \emptyset\}.$$

Assume that for some $U \subseteq GF(q)$ with $|U| \geq dr^3$ and for some matrix $M$ of $dr^3$ distinct elements of $U$ we have that

$$|\mathrm{BAD}| \geq \frac{dr^2}{2d} = \frac{r^2}{2}. \tag{4}$$

For any $i \in \mathrm{BAD}$, consider two indexes $j_1(i) \neq j_2(i)$ for which

$$l^{\#}(A_{j_1(i)}, u_{i,j_1(i)}) \cap l^{\#}(A_{j_2(i)}, u_{i,j_2(i)}) \neq \emptyset \tag{5}$$

(observe that, from the definition of BAD, these two indexes must exist).

Since $u_{i,j_1(i)} \neq u_{i,j_2(i)}$ (remind that $M$ consists of pairwise distinct elements from $U$), Eq. 5 easily implies that $A_{j_1(i)} \neq A_{j_2(i)}$. Since the number of possible pairs $(A_{j_1(i)}, A_{j_2(i)})$ with $A_{j_1(i)} \neq A_{j_2(i)}$ is

$$\frac{1}{2}r(r-1) < \frac{1}{2}r^2 \leq |BAD|$$

then at least two different $i_1$ and $i_2$ exist for which $A_{j_1(i_1)} = A_{j_1(i_2)}$ and $A_{j_2(i_1)} = A_{j_2(i_2)}$. Let

$$A_1 \;=\; A_{j_1(i_1)} \;=\; A_{j_1(i_2)}, \quad \text{and } A_2 \;=\; A_{j_2(i_1)} \;=\; A_{j_2(i_2)}\;.$$

Define also

$$C_1 = l(A_1, u_{i_1,j_1(i_1)}) \bigcap l(A_2, u_{i_1,j_2(i_1)}) , \; C_2 = l(A_1, u_{i_2,j_1(i_2)}) \bigcap l(A_2, u_{i_2,j_2(i_2)}) .$$

Consider now the four vectors

$$V_1 = C_1 - A_1 , \qquad V_2 = A_2 - C_1 , \qquad V_3 = C_2 - A_2 , \qquad V_4 = A_1 - C_2 .$$

It is easy to verify that they are linearly dependent, i.e. $V_1 + V_2 + V_3 + V_4 = 0$. Furthermore, we have that

$$\begin{array}{lllll} V_1 \mid\mid \vec{h}(u_1), & V_2 \mid\mid \vec{h}(u_2), & V_3 \mid\mid \vec{h}(u_3), & V_4 \mid\mid \vec{h}(u_4), & \text{where} \\ u_1 = u_{i_1,j_1(i_1)}, & u_2 = u_{i_1,j_2(i_1)}, & u_3 = u_{i_2,j_2(i_2)}, & u_4 = u_{i_2,j_1(i_2)} . & \end{array}$$

At least two of the above vectors $V_1$, $V_2$, $V_3$, $V_4$ are not zero. It follows that vectors $h(u_i), i = 1, \ldots, 4$ should be linearly dependent. But this is not true: these vectors constitute the well known *Wandermonde* determinant which is always positive for pairwise distinct values of $u_i$, $i = 1, \ldots, 4$. This implies that $|\text{BAD}| < \frac{dr^2}{2d}$ and the lemma is proved.

$\square$

Informally speaking, this lemma states that if we randomly choose a row of $M$ then, with high probability, the $r$ elements contained in this row can be used to compute the system in Eq. 1 without reading conflicts on memory modules. The formal randomized procedure is described in the next section.

## 4 The Overall Solution

Let us consider a DMM with $r$ processors. In order to run randomized algorithms, we assume that a public *pseudo-random generator* of bits is available to all processors.

1. **The Pre-Processing Task: The Shared Memory Partition.** This task receives the output table $T$ of $f$, the data-replication parameter $\beta$, and the security parameter $\delta = \frac{1}{2d}$. Let

   $$k = \lceil \log d + 3 \log r + \log \beta \rceil , \quad \text{and } q = 2^k$$

   Consider the field $GF(q)$ using its standard binary representation. Define $U$ as the first $dr^3$ elements in $GF(q)$ (any ordering of the field works well). Then, we store the subtables $f_A$ (for any $A \in GF(q)^4$) and $g_l$ (for any $l \in SL_4(U)$) in the shared memory modules (each of them is stored in one single memory module).

2. **The On-Line Reading Procedure.** The procedure receives a set of $r$ read requests $\{X_i = (A_i, B_i), \; i = 1, \ldots, r\}$.

   **2.a. The Randomized Procedure** RAND.

   - Define the $(dr^2 \times r)$-matrix $M$ where

     $$M(i,j) \text{ is the } ((i-1)r + j)\text{-th element of } U$$

     (note that we don't need to store $M$ in the shared memory)

- Choose uniformly at random an index $i_R$ from the set $1, \ldots, dr^2$;
- For any $j = 1, \ldots, r$ processors $p_j$ reads $i_R$ and computes

  $$u_j = M(i_R, j) \ .$$

**2.b. The Procedure** READ. Apply procedure READ using the sequence $(u_1, \ldots, u_r)$.

### 4.1 Performance Analysis

**1 The Pre-Processing Task: The Shared Memory Partition.** Since we have defined $k = \lceil \log d + 3 \log r + \log \beta \rceil$ and $q = 2^k$, it follows that

$$\begin{aligned} |SL_4(U)| &= \frac{|GF(q)^4||U|}{|GF(q)|} \\ &= 2^{4k}\frac{|U|}{2^k} \\ &\le 2^{4k}\frac{1}{\beta} \ . \end{aligned}$$

The total size of the shared memory used to implement $f$ is thus the following

$$\begin{aligned} |GF(q)^4|2^{n-4k} + |SL_4(U)|2^{n-4k} &\le \left(2^{4k} + 2^{4k}\frac{1}{\beta}\right)2^{n-4k} \\ &= \left(1 + \frac{1}{\beta}\right)2^n. \end{aligned}$$

The number of memory modules is $|GF(q)^4| + |SL_4(U)| = O(d^4 r^{12} \beta^4)$. So, if we have to implement a larger table while keeping the same values of the other parameters, the number of the memory modules does not increase (only their size increases). Furthermore, if some bit in the Table $T$ of $f$ has to be changed, we just need to update the contents of some memory modules but we do not need to update the distribution of $T$ in the memory.

**2.a. The Procedure** RAND. We first perform $\lceil \log r + \log d \rceil$ calls of the public pseudo-random generator to select $i_R$. Then, the processor $p_j$ ($j = 1, \ldots, r$) returns the element $u_j$ by computing the $((i-1)r + j)$-th element of $U$. This computation in $GF(q)$ requires $O(k)$ time using $O(k^2)$ processors.

**2.b. The Procedure** READ. In order to compute $f$ we apply the procedure READ. For any $j = 1, \ldots, r$, $p_j$ computes the function

$$f(A_j, B_j) = g_{l(A_i,u_i)}(B_i) \bigoplus \left( \bigoplus_{A^* \in l\#(A_i,u_i)} f_{A^*}(B_i) \right) .$$

Assume now that the sequence $u_1, \ldots, u_r$ verifies the property in Eq. 3 (from Lemma 3.1, this happens with probability greater than $(1 - 1/(2d))$). In this case, each memory module receives at most one query, so the memory contention is 0. It follows that the task of each processor $p_j$ can be performed in $O(\log r + \log d + \log \beta) + \mathsf{extime}$ parallel time using a number of processors (each of these processors is in fact a Boolean gate of fan-in two) that satisfies the following bound

$$O(|l^{\#}(A,u)|) = O(2^k) = O(dr^3\beta)$$

Finally, we emphasize that processor $p_j$ asks the same bit (i.e. that having address $B_j$) to all "its" memory modules, so these requests can be sent by a simple multicast protocol.

We can now state the following theorem.

**Theorem 4.1** *Given any $\beta, d \geq 1$, the above system distributes a set $T$ of $2^n$ binary data and computes, with probability greater than $(1 - 1/(2d))$, any set of $r$ read requests within the following complexity*

- $(1 + \frac{1}{\beta})2^n$ *memory size (no additional shared memory for extra-computation);*
- $O(\log r + \log \beta + \log d) + \mathsf{extime}$ *parallel time (with no memory contention);*
- *$r$ processors each of them having circuit size $O(dr^3\beta)$.*
- $O(\log r + \log d)$ *random bits.*

Finally, we emphasize that if the procedure RAND returns a "bad" sequence of elements from $GF(q)$ the algorithm still computes the correct output: in this case, however, more than one read operations have to be performed on the same memory module.

## 4.2 The deterministic version

The algorithm ALG shown in the previous section can be derandomized by selecting one correct sequence $\{u_1, \ldots, u_r\}$ that satisfies Eq. 3 in a deterministic way. More precisely, it is possible to apply a procedure INIT that considers the matrix $M(i,j)$ and then computes the first row $i_0$ of $M$ that satisfies Eq. 3. Clearly, the existence of a row in $M$ satisfying Eq. 3 is an immediate consequence of Lemma 3.1. Then, the rest of the procedure is the same of Algorithm ALG.

The procedure INIT must check Eq. 3 on at least a fraction $1/2c + 1$ of the $cr^2$ rows and then determine the index $i_0$. Notice that, once $i_0$ is determined, since the $M$ elements are fixed, each processor $p_j$ can compute its own parameter $u_j$ independently. It is not hard to see that the checking task requires a global Boolean circuit of size $O(r^4k^2)$ and depth $O(\log r + \log k)$. Indeed, Eq. 3 can be verified by checking the feasibility of a linear system in $GF(q)^4$ ($q = 2^k$) having two equations. This can be done with a Boolean circuit of $O(k^2)$ gates and depth $O(\log k)$. This checking must be performed in parallel for all possible pairs of distinct elements in every row (such pairs are $r(r-1)/2$) and for at least $(1/2c + 1)cr^2$ rows.

From the above description it should be clear that the deterministic version is much more expensive (in terms of circuit size of the processors and parallel time) than the randomized version (even though its complexity is still independent of the size of the sharing data). We also observe that the procedure INIT must be run for every new set of read requests since it depends on the prefixes $A_1, \ldots, A_r$. The only practical reason for which the deterministic version can be chosen rather than the randomized one is thus the lack of a good pseudorandom generator.

On the other hand, the main interest of the deterministic version lies in its consequences in circuit complexity. In fact, by re-stating the whole algorithm in terms of Boolean circuits, our deterministic method provides an efficent solution for computing an arbitrary Boolean function on a set of unrelated inputs (the problem is known as the *Direct Sum Problem* [Nisan et al 94, Bshouty 89]). Indeed, Ulig [Ulig 74] proved that if $L(r,n)$ denotes the worst-case, circuit-size complexity to compute an $n$-variable boolean function on $r$ unrelated inputs then we have $L(r,n) = (1+o(1))L(1,n)$ for any $r = 2^{o(n/\log n)}$. The deterministic version of our algorithm implies that Ulig's result holds for the wider case $r = 2^{o(n)}$, and that, for any $0 < \epsilon < \frac{1}{2}$, for any $r \leq 2^{n(\frac{1}{2}-\epsilon)}$, we have $L(r,n) = O(L(1,n))$.

This theoretical consequence of our method is out of the scope of this paper, we thus refer the reader to [Andreev et al 96] for a detailed description of this part.

## 5 An Application: Parallel Private Information Retrieval Using the DMM Model

The problem of *Private Information Retrieval (PIR)* consists of designing distributed models and related algorithms that enable users to perform queries (i.e. read operations) to databases while keeping these queries secret from the database owner.

The rapid development of distributed databases and all kinds of data-services available on WWW servers is one of the major motivations of several recent works on this problem [Chor et al 97, Chor et al 95, Gertner et al 98]. All the existing solutions for PIR achieve information theoretic privacy by replicating the database into several copies managed by independent agents that are not allowed to communicate with each other. This allows the user to make different queries to different copy agents and combine their answers to compute his original query, without giving any information about it to the database agents.

This problem is usually formalized (see [Chor et al 95]) by considering the database (or any copy of it) as the output table of a function $f : \{0,1\}^n \to \{0,1\}$ and any user request as the specification of an input $x \in \{0,1\}^n$. A PIR system must return the value $f(x)$ to the user without giving any information about $x$ to the database owner (or to the copy owners).

Since public databases are expected to be used by a large number of users, it is natural to design parallel systems that allows the computation of more queries from different users while minimizing the memory contention. A feasible solution for this task relies on the use of the DMM model to implement each copy of the database function $f$ and to satisfy the user queries in parallel. Each agent copy of $f$ is distributed on the memory modules of one DMM according to our memory scheme (i.e. each copy of the database is the table $T$ in our solution). It is then immediate to verify that our randomized solution can be efficiently applied in order to implement any PIR system that uses a set of $l$ copies of the original database by using $l$ DMM. We also emphasize that the best available protocols [Chor et al 95, Gertner et al 98] for PIR are based on the computation of xor (linear summations in $GF(2^k)$) among different queries from different copies of the original database. This linear algebra is exactly the same used in our solution in order to determine the correct read accesses from the memory modules of one DMM. So, we do not need a more sophisticated technology than that commonly adopted by such protocols and in several "real-world" databases (see [Chin 86, Dobkin et al 79, Ullman 82]).

A further motivation in designing parallel systems for PIR arises from a recent work by Gertner *et al* [Gertner et al 98]. They indeed introduced a new method to achieve both user and database privacy without making copies of the original database. Their method relies on the use of *universal random servers* containing random sequences that are accessed by the users to derive the answers of their queries. The main novelty here is the fact that the same random sequence of a universal server can be used by the users of several independent databases (and, thus, by more independent PIR systems). This clearly provides a further motivation in designing parallel systems for PIR applications.

## References

[Andreev et al 96] A.E. Andreev, A.E.F. Clementi, J. D.P. Rolim (1996), On the parallel computation of Boolean functions on unrelated inputs. Proc. of the *IV IEEE Israel Symposium on Theory of Computing and Systems (ISTCS'96)*, IEEE, pp. 155–161.

[Bshouty 89] Bshouty N. H. (1989), On the Extended Direct Sum Problem Conjecture, *Proceedings, 21th ACM STOC*, pp. 177-185 .

[Chin 86] Chin F. (1986), Security Problems on Inference Control for SUM, MAX and MIN Queries. *J. of ACM*, 33(3), pp. 451–464.

[Chor et al 97] Chor B., and Gilboa N. (1997), Computationally Private Information Retrieval. *Proc. of ACM STOC*, p. 304–313.

[Chor et al 95] Chor B., Goldreich O., Kushilevitz E., and Sudan M. (1995), Private Information Retrieval. *Proc of IEEE FOCS*, pp. 41-50.

[Dobkin et al 79] Dobkin D., Jones A. K., Lipton R.J. (1979), Secure Databases: Protection Against User Influence, *ACM Trans. on Database Systems*, 4(1), pp. 97–106.

[Frailong et al 85] Frailong J.M., Jalby W. and Lenfant J. (1985), XOR-schemes: a flexible data organization in parallel memories. *Proc. of Intern. Conf. on Parallel Processing*, pp. 276–283.

[Gertner et al 98] Gertner Y., Goldwasser S., and Malkin T. (1998), A Random Server Model for Private Information Retrieval. *Technical Report* MIT-LCS-TR-715. (also in *Proc. of RANDOM '98*

[Gertner et al 98] Gertner Y., Ishai Y., Kushilevitz E., and Malkin T. (1998), Protecting Data Privacy in Private Information Retrieval Schemes. *Proc. of ACM STOC.*

[Karp et al 96] Karp R. M., Luby M., and Meyer auf der Heide F. (1996), Efficient PRAM Simulation on a Distributed Memory Machine. *Algoritmica*, 16, pp. 517-542 (Extended Abstract in *ACM STOC 1992*).

[Karlin et al 86] Karlin A. and Upfal E. (1986), Parallel hashing - an efficient implementation of shared memory. *Proc. ACM STOC*, 160-168.

[Kruskal et al 90] Kruskal C.P., Rudolph L., and Snir M. (1990), A Complexity Theory of Efficient Parallel Algorithms. *Theoret. Comput. Sci*, 71, p. 95–132.

[Konstantinidou et al 90] S. Konstantinidou and L. Snyder (1990), Chaos router: a practical application of randomisation in network routing. *Proc. SPAA*, pp. 21-30.

[Kumar et al 95] Kumar V., Grama A., Gupta A., and Karypis G. (1995), *Introduction to Parallel Computing.* Benjamin/Cummings Publ. Company.

[Leighton 92] T. Leighton (1992), *Introduction to parallel algorithms and architectures: arrays, trees, hypercubes.* Morgan Kaufmann Publishers, san Mateo, CA.

[Liu et al 92] Liu Z., Li X., and You J. (1992), On storage schemes for parallel array access. *Proc. ACM ICS*, pp. 282–291.

[Mehlhorn et al 84] Mehlhorn K. and Vishkin U. (1984), Randomized and Deterministic Simulation of PRAM by Parallel Machines with Restricted Granularity of Parallel Memoriesm. *ACTA Informatica*, 21, pp. 339–374.

[Nisan et al 94] Nisan N., Rudich S., and Saks M. (1994), Products and Help Bits in Decision Trees, *Proceedings, 35th IEEE FOCS*, pp. 318-329.

[Paul 76] Paul W.J. (1976), Realizing Boolean functions on Disjoint Set of Variables. *Theoret. Comput. Sci.* **2**, pp. 383-396.

[Pietracaprina et al 93] Pietracaprina A., and F. P. Preparata (1993), A Practical Constructive Scheme for Deterministic Shared-Memory Access. *Proc of ACM SPAA*, p. 100–109.

[Pluris 98] Pluris Inc. (1998), Pluris Massively Parallel Routing. *Technical Report* available at *www.pluris.com/wp/index.html.*

[WPluris 98] Pluris Inc. (1998), Parallel Routing, *Technical report* available at *www.pluris.com.*

[Tannenbaum 94] Tannenbaum A.(1994), *Computer Networks.* Prenctice Hall, III Edition.

[Ulig 74] Ulig D. (1974), On the synthesis of self-correcting schemes from functional elements with a small number of reliable elements, *Notes Acad. Sci. USSR* **15**, pp. 558-562.

[Ullman 82] Ullman J. D. (1982) *Principles of Database Systems.* II edition.

[Upfal 84] Upfal E. (1984), Efficient Schemes for Parallel Communication. *J. of the ACM*, 31 (3), pp. 507–517.

[Upfal et al 87] Upfal E. and Wigderson A. (1987), How to share memory in a distributed system, *J. of the ACM*, 34, pp. 116–127.