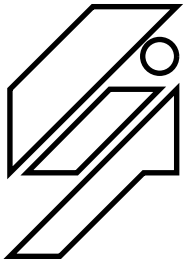
A note on the XRAM and PRAM models.

Pierre Fraigniaud



Laboratoire de l'Informatique du Parallélisme

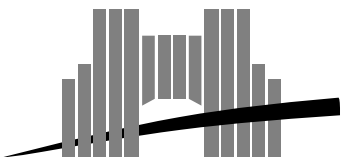
Ecole Normale Supérieure de Lyon
Unité de recherche associée au CNRS n°1398

A note on the XRAM and PRAM models

Pierre Fraigniaud

January 1996

Research Report N^{O} 96-03



Ecole Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : (+33) 72.72.80.00 Télécopieur : (+33) 72.72.80.80
Adresse électronique : lip@lip.ens−lyon.fr

A note on the XRAM and PRAM models

Pierre Fraigniaud

January 1996

Abstract

In this paper, we deal with the XRAM model introduced in [3]. We mainly show that the original definition of the XRAM model was not consistent, and must be slightly modified. Therefore, we modify the definition of the XRAM model to make it consistent, and we study the consequence of this modification on the complexity theory developed in the XRAM model. The new model modifies, in particular, the definition of a *problem* on a XRAM, and thus on a PRAM and on a RAM since these two models are particular cases of the XRAM. However, we show that, though theoretically important, this modification has no practical consequence on the complexity theory developed on the XRAM model.

Keywords: PRAM, complexity

Résumé

Cet article traite du modèle XRAM introduit dans [3] et de ses implications sur le modèle PRAM. Il rectifie en particulier la définition originelle du modèle XRAM pour rendre ce modèle robuste vis-à-vis de l'isomorphisme de graphe.

Mots-clés: PRAM, complexité

A note on the XRAM and PRAM models

Pierre Fraigniaud *

Laboratoire de l'Informatique du Parallélisme – CNRS
Ecole Normale Supérieure de Lyon
69364 Lyon Cedex 07, France

email: pfraign@lip.ens-lyon.fr

Abstract

In this paper, we deal with the XRAM model introduced in [3]. We mainly show that the original definition of the XRAM model was not consistent, and must be slightly modified. Therefore, we modify the definition of the XRAM model to make it consistent, and we study the consequence of this modification on the complexity theory developed in the XRAM model. The new model modifies, in particular, the definition of a *problem* on a XRAM, and thus on a PRAM and on a RAM since these two models are particular cases of the XRAM. However, we show that, though theoretically important, this modification has no practical consequence on the complexity theory developed on the XRAM model.

*Supported by the research programs PRS and ANM of the CNRS, and by the DRET of the DGA.

1 Introduction

This paper deals with the XRAM model introduced by Cosnard and Ferreira in [3]. The XRAM model generalizes the PRAM model [8] by taking into account the several possible interconnection topologies of the existing distributed memory parallel computers (these ones are not fully connected in general [10]).

A random access machine (RAM) [9] consists of

- a memory with a potentially infinite number of locations, and
- a processor capable of loading and storing data from and into the memory, executing arithmetic and logical operations using a finite number of internal registers, and operating under the control of a program stored in a control unit.

In one step requiring a unit of time, the processor can

1. read a datum from an arbitrary location in memory into one of its internal registers,
2. perform a computation on the content of one or two registers, and
3. write the content of one register into an arbitrary memory location.

The parallel RAM (PRAM) [8] consists of an arbitrary large number n of RAMs, all sharing the same common memory. Every step of a PRAM consists of three phases (all along the paper, we restrict ourselves to the *exclusive read, exclusive write* (EREW) model):

1. all processors read simultaneously from n different locations in the shared memory (one for each processor), and each processor stores the obtained value in one of its internal registers;
2. all processors perform a computation on the content of one or two local registers;
3. all processors write simultaneously into n different locations in the shared memory (one for each processor).

Cosnard and Ferreira generalized the PRAM model by introducing the XRAM model as follows:

Definition 1 (From [3])
Let X_i, $i = 0, \ldots, n-1$, be a collection of subsets of $\{0, \ldots, n-1\}$. An XRAM(P, M, X) is an undirected bipartite graph such that $P = \{P_i, i = 0, \ldots, n-1\}$ and $M = \{M_i, i = 0, \ldots, n-1\}$ are the two partitions (representing the processors and the memory locations respectively) and such that P_i is connected to M_j if and only if $j \in X_i$. X is the corresponding interconnection network. Each computation step of an XRAM satisfies the same constraints as the PRAM excepted that the memory locations that can access processor P_i are limited to $M_j, j \in X_i$, $i = 0, \ldots, n-1$.

For instance, the *hypercube* RAM is defined by the sets $X_i = \{j \in \{0, \ldots, n-1\} \mid$ the binary expressions of i and j differ in at most one bit position$\}$, $i = 0, \ldots, n-1$. The hypercube RAM is denoted by HRAM in the following. The PRAM is a special case of Definition 1 where $X_i = \{0, \ldots, n-1\}$ for every $i = 0, \ldots, n-1$ (although the number of memory location is limited to n

instead of being infinite, but this can be easily solved by allowing $|P|$ and $|M|$ to be different, and $|M|$ to be arbitrarily large).

Though this model is quite attractive, and draws a bridge between PRAMs and distributed memory computers, it suffers of a major default that is two virtually equivalent topologies are not comparable. This default is pointed out in the next section, and a new definition that corrects it is proposed. The main consequence of this new definition is the freedom of the initial and final placements of data and results. This also forces a new definition of the complexity of a problem, and then we can prove that two isomorphic topologies have indeed the same computational power. Hopefully, we also show that our new model does not modify the hierarchy of the complexity classes since all results that were previously derived based on Definition 1 are still valid up to an additive factor corresponding to the time of the permutation routing problem. In Section 3, we discuss about several properties of our new model. In particular, we show that a problem can be naturally decomposed in subproblems whereas such a formal decomposition was not easy in the former model. We also discuss about separation theorems, and show that most of the ones proved in [3] still hold in our new model. We also revisit the speed-up folk theorem and the simulation theorem on a PRAM and prove that a speed-up of $2p-1$ is possible on a p-processors PRAM, even if there is no constraint on the memory location of the data and the results. We generalize in this way the result obtained in [1], where input and output memory locations were part of the problem. Finally, we conclude the paper in Section 4 by some comments about the XRAM model as a practical and/or theoretical model for parallel computation.

2 A new definition of the XRAM model

2.1 Comparability must be reflexive

We adopt the same terminology as in [3]: given a problem $\mathcal{P}$ and two models of computation M_1 and M_2, $M_1(\mathcal{P}) \leq M_2(\mathcal{P})$ (resp. $M_1(\mathcal{P}) < M_2(\mathcal{P})$) if the complexity of $\mathcal{P}$ in the model M_2 is smaller (resp. strictly smaller) than the complexity of $\mathcal{P}$ in the model M_1. We will say that the model M_1 is less powerful than the model M_2 if $M_1(\mathcal{P}) \leq M_2(\mathcal{P})$ for every problems $\mathcal{P}$. This is denoted by $M_1 \leq M_2$. Moreover, if $M_1 \leq M_2$, and if there exists a problem $\mathcal{P}$ such that $M_1(\mathcal{P}) < M_2(\mathcal{P})$, then we say that M_1 is *strictly* less powerful than M_2, that is denoted by $M_1 < M_2$.

The two models PRAM and HRAM are separated in [3] as follows. (Of course HRAM $\leq$ PRAM.) Let us consider the *cyclic shift* problem defined by: $C[M_i] \leftarrow C[M_{(i-1) \bmod n}], i = 0, \ldots, n-1$, where $C[M_i]$ denotes the content of the ith memory location. This problem can be solved in one step on a PRAM. On the other hand, solving this problem on a HRAM requires at least $\Omega(\log n)$ steps since $C[M_{n-1}]$ must be "sent" to processor P_0 that is at distance $\Omega(\log n)$ from M_{n-1} in the hypercubic network induced by the HRAM.

Even if this proof is virtually correct, one can argue against it because it also proves that the HRAM is strictly less powerful than...itself! Indeed, let us consider two isomorphic copies G_1 and G_2 of an Hamiltonian graph G. For instance, graphs (a) and (b) of Figure 1 are two isomorphic copies of the 2-dimensional hypercube Q_2. Assume that the vertices of the two copies are arbitrarily labeled. These two graphs induce two XRAMs. For instance, XRAMs (c) and (d) of Figure 1 are obtained from the graphs (a) and (b), respectively. Now, in a same way as $(0, 1, 3, 2)$ is an Hamiltonian cycle of graph (a) in Figure 1 but not of graph (b), it is likely true that one can find a permutation $\sigma_1 \in \Gamma_n$ such that the ordered set $\{\sigma_1(i), i = 0, \ldots, n-1\}$ is an Hamiltonian cycle in

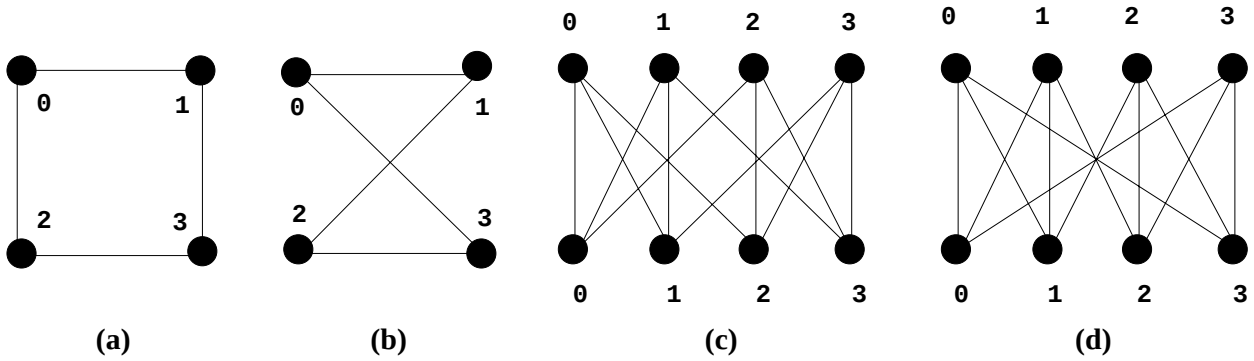


Figure 1: Two isomorphic XRAMs obtained from two isomorphic copies of Q_2.

G_1 but not in G_2, and a permutation $\sigma_2 \in \Gamma_n$ such that the ordered set $\{\sigma_2(i), i = 0, \ldots, n-1\}$ is an Hamiltonian cycle in G_2 but not in G_1. Hence, following the same arguments as the separation proof for the HRAM and the PRAM, one can prove that the two XRAMs obtained from two isomorphic copies of the same graph G are incomparable:

$$G_1\text{RAM}(\mathcal{P}) < G_2\text{RAM}(\mathcal{P}) \text{ and } G_2\text{RAM}(\mathcal{P}') < G_1\text{RAM}(\mathcal{P}')$$

where $\mathcal{P}$ and $\mathcal{P}'$ are two different versions of the cyclic shift problem adapted to the corresponding Hamiltonian cycles of G_1 and G_2.

Therefore, the classification based on the comparator "$<$" defined before does not produce a partial order because it is not reflexive. As we will see later, it may also produce some strange results but, in the following section, we first modify the definition of the XRAM so that "$<$" produces a partial order. This will be enough to avoid inconsistent results that can be obtained with Definition 1.

2.2 A new definition of the XRAM model

We propose the following new definition for the XRAM model. To make a distinction between the definition of Cosnard and Ferreira, and the new definition, we denote our model by *io*-XRAM (for *input-output* XRAM).

Definition 2 (New definition of the XRAM: *io*-XRAM)
Let G be any graph of p vertices. An io-XRAM of topology G consists in a set P (for processor) of p RAMs, a set M (for memory) of p memory blocks, each block being potentially infinite as does the memory of a RAM, and two sets $\mathcal{I}$ (for input) and $\mathcal{O}$ (for output) of n memory locations. The p RAMs of P and the p memory blocks of M are connected as the incident bipartite graph of G.

Computation on a io-XRAM are performed as follows:

1. ***Input the data.*** *Data are initially stored in $\mathcal{I}$. They are "loaded" in M using an input function $\phi_{\mathcal{I}} = (\phi^1_{\mathcal{I}}, \phi^2_{\mathcal{I}}) : \{0, \ldots, n-1\} \to \{0, \ldots, p-1\} \times \mathbb{N}$ that maps $\mathcal{I}$ to M. The mapping $\phi_{\mathcal{I}}$ depends on the problem solved (but not on the values of the data): the ith data, that is the one stored in position i of $\mathcal{I}$, is stored in the memory block $M_{\phi^1_{\mathcal{I}}(i)}$ at the address in this memory block specified by $\phi^2_{\mathcal{I}}(i)$.*

2. ***Computation.*** *This is done exactly in the same way as seen before for the PRAM or the XRAM: computation proceeds step by step, each step being composed of the three phases described in Section 1 where, for each processor, data can be loaded and stored from/to adjacent memory blocks following the connections defined by the graph G.*

3. ***Output the results.*** *Results must be placed in $\mathcal{O}$. They are loaded from M using an output function $\phi_{\mathcal{O}} = (\phi^1_{\mathcal{O}}, \phi^2_{\mathcal{O}}) : \{0, \ldots, n-1\} \to \{0, \ldots, p-1\} \times \mathbb{N}$ that maps $\mathcal{O}$ to M. As $\phi_{\mathcal{I}}$, the mapping $\phi_{\mathcal{O}}$ depends on the solved problem (but not on the values of the results): the ith result, that is the one that must be placed in position i of $\mathcal{O}$ is stored in memory block $M_{\phi^1_{\mathcal{O}}(i)}$ at the address $\phi^2_{\mathcal{O}}(i)$.*

The two functions $\phi_{\mathcal{I}}$ and $\phi_{\mathcal{O}}$ allow to take into account that two XRAMs defined from two isomorphic copies of the same graph are the same: even if the two sets of nodes are labeled in a different way, the choice of the adapted functions $\phi_{\mathcal{I}}$ and $\phi_{\mathcal{O}}$ will allow to execute the same code for solving the same problem on the two machines. We will formally prove this fact soon but, before, we need to define what is the complexity of a problem in the *io*-XRAM model.

2.3 Complexity of a problem

An instance of a *problem* on an *io*-XRAM is defined as a function from $\mathcal{I}$ to $\mathcal{O}$ whereas it was defined on a XRAM as a function from M to M. We are free to choose the best adapted input and output functions $\phi_{\mathcal{I}}$ and $\phi_{\mathcal{O}}$ but this choice is, generally, of no help because it depends on the problem and not on its instances. For instance, one cannot choose the functions $\phi_{\mathcal{I}}$ and $\phi_{\mathcal{O}}$ such that $\phi^{-1}_{\mathcal{O}} \circ \phi_{\mathcal{I}}$ systematically sorts any set of keys.

Of course the load of the data from the input set $\mathcal{I}$ to the memory, and the store of the result from the memory to the output set $\mathcal{O}$ are only virtual operations. It is simply a way to say where are initially the data and where can be obtained the results. Therefore, in the computation process, phases 1 and 3 are for free, and only phase 2 is costly.

More precisely, given a problem $\mathcal{P}$, and given $\phi_{\mathcal{I}}$ and $\phi_{\mathcal{O}}$, let $\mathcal{A}$ be an algorithm solving $\mathcal{P}$, that is for any instance of $\mathcal{P}$, $\mathcal{A}$ transforms the contents of the memory locations according to the rules of the *io*-XRAM computation such that if, for every i, the ith component of the data is placed in memory block $M_{\phi^1_{\mathcal{I}}(i)}$ at the address $\phi^2_{\mathcal{I}}(i)$, then, for every j, the jth component of the result is placed in $M_{\phi^1_{\mathcal{O}}(j)}$ at the address $\phi^2_{\mathcal{O}}(j)$. As usual, the complexity of the algorithm $\mathcal{A}$ is the maximum, taken over all the instances of $\mathcal{P}$, of the number of steps of $\mathcal{A}$ required to solve a given instance of $\mathcal{P}$. Given $\phi_{\mathcal{I}}$ and $\phi_{\mathcal{O}}$, the complexity of a problem $\mathcal{P}$ is the minimum, taken over all the algorithms $\mathcal{A}$ solving $\mathcal{P}$, of the complexity of $\mathcal{A}$. It is denoted by $comp_{\phi_{\mathcal{I}},\phi_{\mathcal{O}}}(\mathcal{P})$.

However, Definition 2 introduces a new degree of freedom, and solving a problem $\mathcal{P}$ on an *io*-XRAM consists in:

1. finding $\phi_{\mathcal{I}}$ and $\phi_{\mathcal{O}}$;
2. given $\phi_{\mathcal{I}}$ and $\phi_{\mathcal{O}}$, finding the fastest algorithm $\mathcal{A}$ solving $\mathcal{P}$.

Therefore, the complexity of a problem $\mathcal{P}$ is denoted by $comp(\mathcal{P})$ and satisfies

$$comp(\mathcal{P}) = \min_{\phi_{\mathcal{I}},\phi_{\mathcal{O}}} comp_{\phi_{\mathcal{I}},\phi_{\mathcal{O}}}(\mathcal{P}).$$

We can now prove the following result that was not true with Definition 1:

Theorem 1 *Let G_1 and G_2 be two isomorphic copies of a graph G and let X_1 and X_2 be the two io-XRAMs obtained from G_1 and G_2 respectively. X_1 and X_2 have the same power.*

Proof. Let ℓ_1 and ℓ_2 be two arbitrary labelings of the nodes of G_1 and G_2 respectively. (ℓ_1 and ℓ_2 then also label the processors and the memory locations of X_1 and X_2.) These labeling, plus the isomorphism ψ between G_1 and G_2, induce a permutation $\sigma \in \Gamma_p$, $\sigma = \ell_2 \circ \psi \circ \ell_1^{-1}$. Let $\phi_{\mathcal{I}}$ and $\phi_{\mathcal{O}}$ be the "best" input and output functions for solving a problem $\mathcal{P}$ on X_1, and let $\mathcal{A}$ be the "best" algorithm used to solved $\mathcal{P}$ on X_1, given $\phi_{\mathcal{I}}$ and $\phi_{\mathcal{O}}$. Then choose the input function $(\sigma \circ \phi_{\mathcal{I}}^1, \phi_{\mathcal{I}}^2)$ and the output function $(\sigma \circ \phi_{\mathcal{O}}^1, \phi_{\mathcal{O}}^2)$ for X_2, and apply the algorithm $\mathcal{A}'$ on X_2 where $\mathcal{A}'$ is obtained from $\mathcal{A}$ by replacing each instruction "P_i accesses M_j at the address k" by "$P_{\sigma(i)}$ accesses $M_{\sigma(j)}$ at the address k". $\mathcal{A}$ and $\mathcal{A}'$ have the same complexity. □

Remark. The execution of the algorithm $\mathcal{A}'$ in the proof of Theorem 1 can also be done using the XRAM model (Definition 1) excepted that the data are not placed initially at their correct positions and therefore $\mathcal{A}'$ will not produce the correct answer.

Note also that, roughly speaking, the *io*-PRAM model and the PRAM model are identical because two labelings of the vertices of the complete graph cannot be distinguished. The unique difference lies on the statement of problems in these two models: in the *io*-PRAM, a problem is defined in terms of input and output, and not in term of memory location.

To definitively convince that the input and output functions must be included in the definition of the XRAM, let us consider the following example: let C_n be the cycle of n vertices and $Q_{\log n}$ be the hypercube of n vertices (we assume n to be a power of 2). Label the vertices of C_n from 0 to $n-1$ in the clockwise direction. Label the vertices of the hypercube as usual, that is the labeling obtained using the recursive construction of the cube: vertex i is joined by and edge to vertex j if and only if the binary expressions of i and j differ of exactly one bit. Now, consider the cyclic shift problem $\mathcal{P}$ as defined in Section 2.1 under the XRAM model. It allows to prove that $Q_{\log n}(\mathcal{P}) < C_n(\mathcal{P})$! Does it mean that the cycle is more powerful than the hypercube? Of course not, again the several ways of labeling the vertices are not taken into account in Definition 1, and induce inconsistent results. In fact the new definition of the XRAM model allow to prove the following theorem that sounds quite natural but that was not true with the former definition:

Theorem 2 *Let $G = (V, E)$ be any graph, and $G' = (V, E')$ be a subgraph of G, $E' \subset E$. Then the io-XRAM of topology G' is less powerful that the io-XRAM of topology G: io-G'RAM $\leq$ io-GRAM.*

Proof. Let ℓ and ℓ' be two arbitrary labeling of the nodes of G and G' respectively. Since G' is a subgraph of G, one can define $\psi = \ell \circ \ell'^{-1}$. Let $(\phi'_{\mathcal{I}}, \mathcal{A}', \phi'_{\mathcal{O}})$ be the placements and the algorithm solving a problem $\mathcal{P}$ on G'. Using the relabeling function ψ as we did in the proof of Theorem 1, one can construct an algorithm $\mathcal{A}$ and placements $\phi_{\mathcal{I}}$ and $\phi_{\mathcal{O}}$ that directly apply to G. Therefore *io*-G'RAM$(\mathcal{P}) \leq$ *io*-GRAM$(\mathcal{P})$. □

Remark. Why this straightforward proof did not applied in the model of Definition 1? Simply because the labeling of the RAMs and the memory locations is more or less forced in Definition 1 whereas it is not considered in Definition 2.

Note also that there exist many conditions for which *io*-G'RAM $<$ *io*-GRAM, where G' is a subgraph of G. For instance it might be the case if the diameter or the girth of G' turn to be

much larger than the ones of G. However such conditions must be studied in detail because one must also find a problem for which these structural modifications really induce an increase in the problem complexity.

2.4 XRAM versus *io*-XRAM

It is known that sorting on hypercube is in $\Omega(\log n)$ and in $O(\log n \log\log^2 n)$ [5]. Now, can we prove that the complexity of sorting on an *io*-HRAM is in this range? Such a question is meaningful because a problem on an *io*-XRAM does not map the memory to itself, but an input set $\mathcal{I}$ to an output set $\mathcal{O}$, where $\mathcal{I}$ and $\mathcal{O}$ are both isomorphic to the memory space M, and where the choice of the isomorphisms $\mathcal{I} \mapsto M$ and $\mathcal{O} \mapsto M$ are free. Of course, the answer of this question is yes, though up to the price of a permutation on the machine. More precisely:

Theorem 3 *Let us consider an arbitrary p-processor XRAM of topology G. For any problem $\mathcal{P}$, we have*

$$comp_{Id,Id}(\mathcal{P}) = O(comp(\mathcal{P}) + \max_{\sigma \in \Gamma_p} comp_{Id,Id}(\mathcal{P}_\sigma))$$

where $\mathcal{P}_\sigma$ is the problem that consists to permute any array A stored in $\mathcal{I}$ ($A[i]$ in position i) following σ, and to obtain the result in $\mathcal{O}$ ($A[i]$ in position $\sigma(i)$). Moreover, this bound is tight.

This theorem shows that although the virtual spaces $\mathcal{I}$ and $\mathcal{O}$, and the functions that map these spaces to the memory, must be introduced to keep consistent the formal definition of an abstraction of a distributed memory computer, the complexity of a problem can be computed practically in fixing arbitrarily the input and output position of the data. Note that the bound of Theorem 3 is tight because for every $\sigma \in \Gamma_p$, $comp(\mathcal{P}_\sigma) = O(1)$. For instance, on a p-processor hypercube, any permutation can be off-line routed in $O(\log p)$ steps [10]. Therefore all the result for the hypercube that were previously derived are valid in the *io*-HRAM model up to an additive logarithmic factor.

3 General properties of the *io*-XRAM model

3.1 Decomposition of a problem in subproblems

As we have seen, functions $\phi_\mathcal{I}$ and $\phi_\mathcal{O}$ were introduced to insure the reflexivity of the comparability by taking into account the possible graph isomorphisms. As we said, such functions cannot be used for solving a problem because they depend on the problem only, and not on its instances. However, one can be tempted to cheat by decomposing a problem in subproblems. For instance, consider the problem of adding matrices in the following order:

1 $C \leftarrow A + B$;

2 $D \leftarrow A^t + B$. ($*$ A^t denotes the transposition of $A*$)

where A and B are stored in $\mathcal{I}$ in row major order, and C and D are stored in $\mathcal{O}$ in row major order. This problem implies to transpose A. This cannot be done using the input and output functions: once the data have been loaded, intermediate results cannot be output during the computation in order to be loaded again in different memory locations after. Indeed, the complexity of a problem

is evaluated once the data are loaded, and before they are output. Therefore, if a problem $\mathcal{P}$ can be decomposed into two successive subproblems $\mathcal{P}_1$ and $\mathcal{P}_2$, then

$$comp_{\phi_{\mathcal{I}},\phi_{\mathcal{O}}}(\mathcal{P}) \leq comp_{\phi_{\mathcal{I}},Id}(\mathcal{P}_1) + comp_{Id,\phi_{\mathcal{O}}}(\mathcal{P}_2). \quad (1)$$

However, it could be interesting to redistribute the data between the execution of $\mathcal{P}_1$ and $\mathcal{P}_2$. This redistribution might be costly, but may also allow to place the data in the right position so that $\mathcal{P}_2$ can be executed rapidly. For instance, if $comp(\mathcal{P}_1) = comp_{\phi^*_{\mathcal{I}},\phi^*_{\mathcal{O}}}(\mathcal{P}_1)$ and $comp(\mathcal{P}_2) = comp_{\psi^*_{\mathcal{I}},\psi^*_{\mathcal{O}}}(\mathcal{P}_2)$, then

$$comp(\mathcal{P}) \leq comp(\mathcal{P}_1) + comp_{Id,Id}(\psi^*_{\mathcal{I}} \circ {\phi^*_{\mathcal{O}}}^{-1}) + comp(\mathcal{P}_2). \quad (2)$$

In Equation 2, $comp_{Id,Id}(\psi^*_{\mathcal{I}} \circ {\phi^*_{\mathcal{O}}}^{-1})$ is the time necessary to perform the permutation of the data from their positions after the execution of $\mathcal{P}_1$ to the positions chosen to perform $\mathcal{P}_2$ optimally. It is not clear whether or not the upper bound 2 is better than 1. In fact, there is a tradeoff between, on one hand, the time to perform $\mathcal{P}_1$ and $\mathcal{P}2$ given the input and output positions of the data, and, on the other hand, the time to permute the data between P_1 and P_2. Therefore, we can state the following general upper bound:

$$comp(\mathcal{P}) \leq \min_{\phi_{\mathcal{I}},\phi_{\mathcal{O}},\psi_{\mathcal{I}},\psi_{\mathcal{O}}} \left[comp_{\phi_{\mathcal{I}},\phi_{\mathcal{O}}}(\mathcal{P}_1) + comp_{Id,Id}(\psi_{\mathcal{I}} \circ \phi_{\mathcal{O}}^{-1}) + comp_{\psi_{\mathcal{I}},\psi_{\mathcal{O}}}(\mathcal{P}_2) \right].$$

More generally, if a problem $\mathcal{P}$ can be decomposed into a succession of k subproblems $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k$, $k > 2$, that we denote by $\mathcal{P} = \mathcal{P}_1|\mathcal{P}_2|\ldots|\mathcal{P}_k$, then $comp(\mathcal{P})$ is equal to

$$\min_{k \geq 1} \quad \min_{\substack{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k \\ \mathcal{P} = \mathcal{P}_1|\mathcal{P}_2|\ldots|\mathcal{P}_k}} \quad \min_{\substack{\phi^{(i)}_{\mathcal{I}}, \phi^{(i)}_{\mathcal{O}} \\ i = 1, \ldots, k}} \left[\begin{array}{c} \sum_{i=1}^{k-1} \left(comp_{\phi^{(i)}_{\mathcal{I}},\phi^{(i)}_{\mathcal{O}}}(\mathcal{P}_i) + comp_{Id,Id}(\phi^{(i+1)}_{\mathcal{I}} \circ {\phi^{(i)}_{\mathcal{O}}}^{-1}) \right) \\ + comp_{\phi^{(k)}_{\mathcal{I}},\phi^{(k)}_{\mathcal{O}}}(\mathcal{P}_k) \end{array} \right].$$

The reader may find interesting to refer to practical experiments where redistributing the data between the several phases of a problem yields better results than the direct algorithm [4, 12]. This is typically the case in the parallel implementations of the ScaLAPack subroutines for linear algebra [2].

Remark. Such a decomposition in subproblems was not so clear in the former XRAM model. Let us take an example: finding the eigenvalues of a matrix is a well defined problem, but nobody will never understand the sentence "finding the eigenvalues of a matrix that is stored on a hypercube such that row 1 is stored on processor 4, block $2 \ldots 7, 3 \ldots 5$ is stored on processor 13, column 2 is stored on processor 31,..." as a *problem*. Indeed, the problem is "finding the eigenvalues of a matrix" and the other part of the sentence is just indications about the initial storage of the data. Such a distinction between *problem* and *storage* formally appears in the *io*-XRAM model.

3.2 About separation theorems

We have seen that the HRAM and the PRAM can be separated in the XRAM model. This result still holds in the *io*-XRAM model. Indeed, let us consider the *permutation* problem defined by: $C[\mathcal{O}_i] \leftarrow C[\mathcal{I}_{\sigma(i)}], i = 0, \ldots, p-1$ where σ is an arbitrary permutation of Γ_p stored in $\mathcal{I}$ between positions p and $2p-1$. This problem can be solved in one step on a PRAM. However, whatever is the choice of the input and output functions, there exists a permutation σ_0 such that the memory

block $M_{\phi_I^1(i)}$ and the memory block $M_{\phi_I^1(\sigma_0(i))}$ are at unbounded distance in the hypercube. (Indeed, only a constant amount of data can be stored in each block, otherwise it would already take an unbounded time just to access locally the data.) Therefore, it is true that

$$io\text{-HRAM} < \text{PRAM}$$

when we restrict our study to the EREW model. Other separation results have been proved in [3]. Most of them separate not only topologies but also memory access constraints (EREW, CREW, CRCW). They stay true in the *io*-XRAM model because proofs use arguments based on problems not defined in term of memory location, but in fact in terms of input and output (like searching or prefix computation).

Tom Leighton deeply investigates in [10] the computational power of several topologies including cycles (linear arrays), meshes, meshes of trees, and hypercubes and related networks. We refer the reader to his book for the several simulation and separation results that link these topologies. He showed in particular that the butterfly network is *universal* in the sense that it can emulate every bounded degree network with a constant slowdown in the computation time. The XRAM model give a general framework to such results.

3.3 Speed-up and simulation

Definition 2 applies to the *io*-XRAM of topology K_p (the complete graph of p vertices), and therefore to the PRAM model. Of course it does not imply any modification of the PRAM theory because, as we said, two labelings of the vertices of K_p cannot be distinguished. However, we need to go through the proofs of theorems based on problems described in terms of data movement inside the memory (the memory locations of the data and the results are specified as part of the problem). As an example, we consider the speed-up folk theorem that says that the speed-up of a parallel algorithm using p processors cannot be greater than $O(p)$. Of course, super linear speed-up can be obtained in practice (that is on real parallel machines) because a processor which deals with less data may avoid problems as, for instance, cache miss, that might strongly slow down the sequential computation on a large amount of data. However, it is often said that a speed-up larger than p cannot be achieved on PRAM. Akl, Cosnard and Ferreira [1] have shown that it is not true and that a speed-up of $2p - 1$ can be achieved on a PRAM of p processors. This result holds mainly because one must keep in mind that each RAM has a *finite* number of registers, and therefore a PRAM of p processors has p times more registers than a single RAM. This is why a p-processors PRAM is more than p times faster than a RAM.

The proof in [1] lies on two arguments (in the following, we assume that each processor has a unique register; the generalization to an arbitrary number of registers can be found in [1]):

1. there exists a problem that can be solved in one step on a p-processors PRAM, and that cannot be solved in less than $2p - 1$ steps on a single RAM (Theorem 3.2 in [1]);
2. each step of a p-processors PRAM can be simulated in $2p - 1$ steps on a RAM (Theorem 6.1 in [1]).

The second argument stays true even under the model of Definition 2. However, the first argument used a problem of the class named *data-movement intensive problem* that is defined in terms of memory location as follows:

Problem 1 *Let $I_1, \ldots, I_p$ be p distinct integers in the range $]-\infty, p]$ stored in an array A in such a way that $A[i] = I_i, i = 1, \ldots, p$. It is required to modify A so that it satisfies the following condition:*

$$\begin{cases} A[i] = i & \text{if there exists } j \text{ such that } I_j = i; \\ A[i] = I_i & \text{otherwise.} \end{cases} \tag{3}$$

Problem 1 requires 1 step on a p-processors PRAM, whereas it requires at least $2p-1$ steps on a RAM. Indeed, the memory location of the input and the output is imposed, that is the data $A[i]$ is given in memory location i, and the result $A[i]$ must be returned in memory location i, with a risk of overwriting an unread data. We could now imagine to store the results elsewhere to avoid this problem. Indeed, what is important is that we must *know* where is the result, but why the memory location of the result should be specified in advance? In fact it does not correspond to Definition 2. Problem 1 under the PRAM model is translated in the following problem in the *io*-PRAM model:

Problem 2 *We are given an array A stored in $\mathcal{I}$ ($A[i]$ in position i). We want to modify it according the rule of Problem 1, and we want the result stored in $\mathcal{O}$ ($A[i]$ in position i).*

The two sentences "stored in $\mathcal{I}$" and "stored in $\mathcal{O}$" just mean "we give you the data", and "we want the result", but the position where are stored and loaded the data in the memory is not part of the problem, it is part of the algorithm solving the problem. Anyway, one can still prove that Problem 2 cannot be solved in less than $2p-1$ steps on an *io*-PRAM, that is even with a total freedom on the memory locations of the data and the results.

Lemma 1 *Problem 2 requires at least $2p-1$ steps on an io-PRAM.*

Proof. Let i, $1 \leq i \leq p$, and let $i^* = \phi_{\mathcal{O}}(i)$. More precisely, i^* denotes the memory location where can be found the result $A[i]$ after modifications specified by Equation 3 in Problem 1: $C[M_{i^*}] = i$ if there exists j such that $I_j = i$, and $C[M_{i^*}] = I_i$ otherwise. It means that the last instruction "write" at the address i^* must follow at least p instructions "read" because p reads are necessary to check whether or not there exists j such that $I_j = i$. Therefore, for every i, $1 \leq i \leq p$, each final write at the address $\phi_{\mathcal{O}}(i)$ must be preceded by p reads. That is a total of at least $2p-1$ steps are necessary (one step can be economized because one can read and then write in the same step). □

Therefore, we get the following result:

Theorem 4 *The speed-up of a p-processors io-PRAM over an io-RAM cannot exceed $2p-1$, and this bound is tight.*

Proof. The tightness of the bound is given by lemma 1. The simulation theorem 6.1 in [1] shows that any step of a p-processors PRAM can be simulated on a RAM in at most $2p-1$ steps. □

4 XRAM: yet another model?

Everybody can state the simple but primordial requirement about what must satisfy a computer model: it must be *simple* and must *reflect the behavior of real computers*. Of course quite a few

models satisfy both requirements. To conclude this paper, let us analyze the XRAM model in terms of these two conditions.

The XRAM model clearly satisfies the first condition. It is just a formal way to express the fact that we work on a machine that has some particular connection properties. The huge amount of results obtained in this framework (see for instance [10]) proves the fruitfulness of such a model.

Concerning the second requirement, the approach followed by theoreticians proving theorems in the XRAM framework is justified by the fact that real parallel computers are indeed not fully connected, and that many machines were built with topologies as hypercube, meshes, trees, etc. Now, we can point out many defaults of the XRAM model. For instance:

- *Computation times* and *communication times* are not distinguished in the XRAM model whereas the elementary computation time and the elementary communication time often differ by many order of magnitude in real parallel computers [13].
- *Computation steps* and *communication steps* are linked in such a way that the efficient and practical method that consists to overlap communications with computations [6] cannot be easily expressed in the XRAM model.
- The several models of *communication costs* that apply to parallel machines [7] are difficult to handle with the XRAM model because they make distinction between start-up times, commutation times, propagation time, etc.
- The *routing mode* related to the XRAM model is packet-switching whereas circuit-switching or wormhole routing are often preferred on the last generation of parallel computers [11].
- The *computation grain* of the XRAM model is fine whereas new parallel computers are often composed of few very powerful processors [13].

Does it mean that the XRAM model is useless? Of course not. First we can argue against some of the previously listed defaults. For instance, the fine grained approach is quite efficient because it is often the good manner to derive a parallel algorithm: a fine grained algorithm can easily be transformed in a coarse grained algorithm [10]. Other defaults can be treated in modifying the model by including a *router* attached to each processor. Doing this requires to consider many factors as elementary communication time, communication constraints, routing mode, etc. Moreover, even if such a model will closely approach the behavior of real machines, it will turn to be so complicated that quite a few powerful theoretical results will be possible to derive on it. So is it a vicious circle?

The answer is no because one should not oppose simplicity and practicability. Theoreticians must know about some part of applied computer science so that their models and results can be used for practical applications. Ingineers must know about some of the main theoretical results derived under abstract models so that they can adapt these results and apply them to real programming environments. From that point of view, we claim that the XRAM model is definitively a *good* model.

Now, it is true that, in order to derive efficient algorithms and to compute lower bounds on the complexities of problems, all the formal environment provided by the input and output sets, and by the input and output functions could be relaxed, and a model like the one defined in [10](Chapter 1.1.5) could be certainly prefered.

References

[1] S. Akl, M. Cosnard, and A. Ferreira. Data-movement-intensive problems: two folk theorems in parallel computation revisited. *Theoretical Computer Science*, 95:323–337, 1992.

[2] E. Anderson, A. Benzoni, J. Dongarra, S. Moulton, S. Ostrouchov, B. Tourancheau, and R. Van de geijn. LAPACK for distributed memory architecture. In *Fifth SIAM Conference on Parallel Processing for Scientific Computing*, USA, 1991.

[3] M. Cosnard and A. Ferreira. Designing parallel non numerical algorithms. In Joubert Evans and Liddell, editors, *Parallel Computing'89*, pages 3–18. Elsevier Science, 1991.

[4] M. Cosnard, M. Loi, and B. Tourancheau. A framework for data migrations on the hypercube. In *NATO Advanced Research Workshop - Software for Parallel Computation (Cetraro)*, 1992.

[5] R. Cypher and G. Plaxton. Deterministic sorting in nearly logarithmic time on the hypercube and related computers. In *Twenty second annual ACM Symposium on Theory of Computing*, pages 193–203, 1990.

[6] F. Desprez, J.J. Dongarra, and B. Tourancheau. Performance Complexity of LU Factorization with Efficient Pipelining and Overlap on a Multiprocessor. *Parallel Processing Letters*, 5-II, 1995.

[7] P. Fraigniaud and E. Lazard. Methods and Problems of Communication in Usual Networks. *Discrete Applied Mathematics*, 53:79–133, 1994.

[8] A. Gibbons and W. Rytter. *Efficient parallel algorithms*. Cambridge University Press, 1988.

[9] J. Hopcroft and J. Ullman. *Introduction to automata, languages and computation*. Addison-Wesley, 1979.

[10] T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann, 1992.

[11] L.M. Ni and P.K. McKinley. A survey of wormhole routing techniques in direct networks. *Computers*, 26(2):62–76, feb 1993.

[12] Loïc Prylli and Bernard Tourancheau. Efficient block cyclic data redistribution. Research Report RR 2766, INRIA/LIP, Laboratoire de l'Informatique du Parallélisme, ENS-Lyon, France, 1996.

[13] Jean De Rumeur. *Communication dans les réseaux de processeurs*. Collection Etudes et Recherches en Informatique. Masson, 1994. (English version to appear).