EDITORIAL COMMITTEE

M. Capovani, G. Capriz, F. Denuccio, S. Faedo, I. Galligani, F. Luccio,
E. Magenes, A. Tesei.

EDITORIAL BOARD

Fichera, L. Gatteschi, R. Glowinski, G. H. Golub, J. L. Lions, G. I. Marchuk,
Stancu, J. Stoer, G. Szegö, D. Trigiane.

CALCOLO

A QUARTERLY ON

NUMERICAL ANALYSIS AND THEORY OF COMPUTATION

ESTRATTO

CONSIGLIO NAZIONALE DELLE RICERCHE
ISTITUTO DI ELABORAZIONE DELLA INFORMAZIONE - CNR

GIARDINI EDITORI
E STAMPATORI
IN PISA
BASIC ALGORITHMS FOR THE MRMW PRAM MODEL

G. PRECIFE (1)

Key words. Parallel algorithms, models of computation, PRAM, MRMW PRAM.

1. Introduction

The mRMmMW (Multiple Read Multiple Write) PRAM has been defined for the first time in [3] as follows: an mRMmMW PRAM (shortly MRMW PRAM if m is understood) is such that at each step a processor P reads the contents of k \leq m memory cells, computes the next state and the output value as proper function of the read value (i.e., COMMON, PRIORITY etc.) and writes the output value in h \leq m memory cells. It constitutes a new mode of connecting processors with memory: mMR establishes a connection from \leq m memory cells to one processor, and mMW establishes a connection from one processor to \leq m memory cells. Although the problem of exclusive/concurrent read and write is not directly treated in [4], the assumption of exclusive read and write is implicitly taken in the paper.

In the same paper the authors show a lower bound of \Theta(\log n), with b = (m + 2 + \sqrt{m^2 + 4m})/2, in the MRMW model, for the problem of computing a function with a critical input f: \{0, 1\}^n \rightarrow \{0, 1\}. They also provide an optimal algorithm that computes the OR of n bits which runs in time

\begin{equation}
T(n) = \log\left(\frac{n}{c_1}\right),
\end{equation}

where \(c_1\) is a constant.

---

(1) Received: 15 October 1996.
(1) Dipartimento di Informatica, Corso Italia 40, 56100 Pisa, Italy.
Another result given in [4] is the equivalence between CRCW and MRMW PRAM. However, we can say that the MRMW is closer to practical parallel models than CRCW: in fact, data reduction is an abstruse operation in CW, while it can be simply performed by the processors in mMR.

We first observe that the equivalence of the two models holds only if the PRAM is semi-obvious (see [1] for the definition of semi-obviousness). For example, according to [4], the simulation of a CW operation is performed as follows. Let the CW operation be structured by processors $P_0, \ldots, P_n$ concurrently write into cell $M_{10}$. This is simulated on the MRMW with three steps.

1. **EW:** $P_0, \ldots, P_n$ write into the additional cells $M_{10}, \ldots, M_{1n}$;
2. **mMR:** $P_i$ reads from these cells;
3. **EW:** $P_j$ writes into $M_i$.

We note that, if the PRAM is not semi-obvious, the set of processors accessing the cell $M_{i0}$ at time $t$ is not known, since this set changes as the input changes. Consequently, we cannot know which processors read into the additional cells $M_{10}, \ldots, M_{1n}$, and which one executes the mMR and the final EW step. In conclusion, the simulation described in [4] works correctly only if the PRAM is semi-obvious.

In the next section, we describe some basic algorithms for which the MRMW model performs better than PRAM model, in the sense that the time complexity can be lowered by a factor of $m$. The first of these algorithms is an improvement of one given in [4], The other four are new.

We also recall that another attempt to limit parallelism in the PRAM model was made in [2], where the authors define the Concurrent-Read-Exclusive-Write PRAM. As in [4], the study of the model is limited to determine lower bounds to Boolean functions. Such bounds coincide with the lower bounds for MRMW, but no algorithm is given.

2. Efficient algorithms in the MRMW model

We present some basic algorithms on the MRMW PRAM. The Multiple Read operation is denoted by:

$$Y_i = \mathcal{R}_{\leq m} M_{i0}, \quad k \leq m,$$

where $Y_i$ is a local variable of the processor $P_i$ and $\mathcal{R}$ is a reduction operation, with:

$$\mathcal{R} \in \{\alpha, \Pi, \min, \max, \wedge, \vee, \oplus\},$$

with an extended definition of $\wedge, \vee, \alpha$ to any number of variables.

2.1. The OR of $n$ bits

In [4] an algorithm for computing the OR of $n$ bits on an MRMW is defined. However, some forbidden conflicts may occur in the read operation. These are eliminated in the following version of the algorithm.

Define the recurrences:

$$R_i = 1, \quad S_i = m_i,$$

and

$$(3) \quad R_i = R_{i-1} + S_{i-1},$$

$$S_i = S_{i-1} + mR_i.$$

Let $T = \log_2 n/c$, where $b = (m + 2 + \sqrt{m^2 + 4m})/2$ (see (1)). The OR problem can be solved with $n/m$ processors as follows:

**Algorithm OR**

Input: $n$ bits stored in the first $n$ memory cells. Initial conditions:

$$Y_i = 0, \quad 1 \leq i \leq n; \quad R_i = 0, \quad S_i = 0.$$

Output: the OR of the $n$ bits stored in the memory cell $M_i$.

Assumptions: $T > 1$

begin

for $i := 1$ to $n/m$ do in parallel

for $t := 1$ to $T$ do

\[ Y_i := Y_i \lor \left( \exists j \leq \frac{S_{i-1}}{m} \right) \]

\[ Y_{i-t} := Y_{i-t} \lor \left( \exists j \leq \frac{S_{i-1}}{m} \right) \]

end.
Algorithm Or is the same as in [4] except for the first if statement, this has been inserted to avoid conflicts in reading, which occur in the original algorithm if the memory addresses of the cells read by two processors, say i and k, are equal; namely from step (*) of algorithm Or:

\[
\begin{align*}
&i + R_{i,j} = j \cdot \left( \frac{S_{a,j}}{m} \right) + (m - 1 - j) \frac{n}{m} = \\
&= k + R_{i,j} = j \cdot \left( \frac{S_{a,j}}{m} \right) + (m - 1 - j') \frac{n}{m},
\end{align*}
\]

that can also be expressed as:

\[
i + \left( \frac{S_{a,j}}{m} \right) + (m - 1 - j) \frac{n}{m} = j + \left( \frac{S_{a,j}}{m} \right) + (m - 1 - j') \frac{n}{m},
\]

where \(1 \leq i, k \leq n/m\) and \(0 \leq j, j' \leq m - 1\). We have:

**Theorem 1.** Algorithm Or correctly computes the Or of n Boolean variables.

**Proof.** First we show that the inserted if statement avoids conflicts in reading. Analyze equation (4); two cases occur:

**Case i.** Suppose that \(j' = j + r, 0 \leq r \leq m - 1 - j\). If \(j = j'\) (i.e., \(r = 0\)) then \(i = k\) indicates the same processor and, obviously, there are not conflicts in reading; if \(j \neq j'\) we have \(0 \leq j \leq m - 2, 1 \leq j' \leq m - 1\) and \(1 \leq r \leq m - 1 - j\). In this case relation (4) becomes:

\[
k = i + j + \left( \frac{S_{a,j}}{m} \right) - j \cdot \left( \frac{S_{a,j}}{m} \right).
\]

Therefore the conflicts in reading are produced by the pair of processors \(i\) and \(k\) such that:

\[
i, k = i + r \cdot \left( \frac{S_{a,j}}{m} \right) - r \cdot \left( \frac{S_{a,j}}{m} \right).
\]

Since \(k\) is a processor index, then \(k \leq n/m\) and we have:

\[
i \leq r \cdot \left( \frac{S_{a,j}}{m} \right) - (r - 1) \cdot \frac{n}{m},
\]

hence \(1 \leq r \leq \left( \frac{S_{a,j}}{m} \right) - (r - 1) \cdot \frac{n}{m}\) and \(1 \leq r \leq m - 1 - j\). Since \(k = n/m\) and \(i > 0\), from relation (5) we obtain:

\[
\left( \frac{n - r \cdot \left( \frac{S_{a,j}}{m} \right)}{m} \right) < \frac{n}{m},
\]

and

\[
r < \frac{n}{n - S_{a,j}}.
\]

We want to show now that \(n/(n - S_{a,j}) < 2\). It is sufficient to show that for all \(T\) steps of the algorithm, where \(T = \log_{n}(n + c_{1}) = \log_{n} n + O(1)\), we have:

\[
S_{a,j} < \frac{n}{2}, \quad \forall t \leq T.
\]

We can assume that \(m < n/2\); otherwise assuming \(m \geq n/2\) we could compute the Or function in constant time with a constant number of processors. Then, \(S_{a,j} = m < n/2\). Moreover, the solution of the equation \(S_{a,j}\) in (5) is given by (see [4]):

\[
S_{a,j} = c_{1}b^{t-1} = \left( \frac{m + 2 + \sqrt{m^{2} + 4m}}{2} \right)^{c_{1}}.
\]

then \(S_{a,j} < n/2\), if the following relation holds:

\[
t < \log_{b}\left( \frac{n}{2c_{1}} \right) + 1 = \log_{b}\left( \frac{n}{c_{1}} \right) + 1 = \log_{b} 2 = T + \log_{b} b/2.
\]

Since \(b > 2\), we have that \(0 < \log_{b} b/2 < 1\); besides, during the execution of the algorithm, \(t \leq T\), therefore \(t < T + \log_{b} b/2\). In conclusion, we have that \(S_{a,j} < n/2\) during the whole execution of the algorithm, that is for any \(t \leq T\), and thus:

\[
\frac{n}{n - S_{a,j}} < 2.
\]
From (6), (8) and \( r > 0 \) we can conclude that \( r = 1 \); so the pair of processors that produce conflicts in reading is

\[
i, k = i + \frac{n}{m} - \left( \frac{S_{<i}}{m} \right)
\]

with \( 1 \leq i \leq \left( S_{<i}/m \right) \).

Case ii. \( j' = j - r \). Obviously is \( j = j' + r \), with \( 1 \leq r \leq m - 1 - j' \), and case ii is analogous to the previous one.

From (7) we have that \( k = i + n/m - S_{<i}/m \geq i \); then, to avoid conflicts in reading, it is sufficient that the processors with index \( \leq (S_{<i}/m) \) read in a previous step while the other ones (processor index > \( (S_{<i}/m) \)) read in the following one; this synchronization is obtained with the insertion of the new if statement. The proof proceeds as shown in [4].

The time complexity of algorithm \( Or \) is not affected by step (*) As in [4] such complexity is given by:

\[
T(n) = \log_2 n + O(1),
\]

with \( b = (m + 2 + \sqrt{m^2 + 4m})/2 \). Therefore algorithm \( Or \) is still optimal.

2.2. An optimal algorithm for the sum of \( n \) integers

ALGORITHM Sum

Input: An array \( X = \{X_1, X_2, ..., X_n\} \) of \( n = m^2 \) integers, with \( k \) a non-negative integer. Let \( X_i \) stored in \( M_i \).

Output: the sum of the \( X_i \), \( 1 \leq i \leq n \), stored in \( M_i \).

begin
for \( i = 1 \) to \( \log_2 n \) do
  for \( j = 1 \) to \( \frac{n}{m} \) do in parallel
    \[
    Y_j \leftarrow \sum_{i=1}^{m} M_{S_{ij} + (j-1)i}
    \]
    \[
    Y_j \leftarrow M_j
    \]
end.

The procedure \( \text{ExtSum}(S_1, ..., S_j) \) sums the elements of array \( S \) between index \( i \) and \( j \) and stores the result in \( S_i \). It is a simple extension of algorithm \( \text{Sum} \).

The step (*) produces conflicts in reading. To avoid this problem, \( n \) copies for each input value have to be produced as preprocessing with the following algorithm:

\[
W(n) = O(\frac{n}{m})
\]

2.3. Matrix Multiplication

Our algorithm for matrix multiplication uses an extension of algorithm \( \text{Sum} \).

ALGORITHM Matrix Multiplication

Input: two matrices \( A \) and \( B \) of \( n \times n \) elements.

Output: the matrix \( C = A \times B \).

begin
for \( i = 1 \) to \( n \) do in parallel
  for \( j = 1 \) to \( n \) do in parallel
    for \( k = 1 \) to \( n \) do in parallel
      \( S_{(i-1)n+i} \) := \( a_{ik} b_{kj} \)
      \( \text{ExtSum}(S_{(i-1)n+i}, ..., S_{(i-1)n+j}) \);
      \( c_{ij} := S_{(i-1)n+j} \);
end.
ALGORITHM Copy

Input an input value d and an array D of dimension n, with n = m^t.

Output the value d copied n times in the array D.

begin
  D := d, 1 ≤ j ≤ m;
for t := 2 to \log_m n do
  for 1 ≤ i ≤ m do in parallel
    D_{w^t(i-1)+j} := D_j, 1 ≤ j ≤ m - 1
end.

The previous version of the Matrix Multiplication algorithm can be easily extended to use algorithm Copy. Note that the latter algorithm works in O(\log_m n) time, with global work of O(n/m). These bounds do not affect the time and work complexity of the Matrix Multiplication algorithm. In conclusion we have:

T(n) = O(\log_m n),
W(n) = O(n^2).

2.4. Prefix Sums

The prefix sum problem is as follows [3]. Given an array X = \{X_1, X_2, ..., X_n\}, compute the array S = \{X_1, X_1 + X_2, ..., X_1 + X_2 + ... + X_n\}.

ALGORITHM Prefix Sums

Input an array of n = m^t elements X = \{X_1, X_2, ..., X_n\}, with k a non-negative integer.

Output the prefix sums S_i of X, with 1 ≤ i ≤ n.

begin
  if n ≤ m then
    for i := 1 to n do in parallel S_i := \sum_{k=1}^{i} X_k
  else
    for i := 1 to n/m do in parallel
      R_i := \sum_{k=(i-1)m+1}^{i m} X_k
end.

Steps (*) and (**) produce conflicts in reading the locations storing arrays Z and X, that can be avoided with the previous algorithm Copy without affecting the time and work complexity.

Algorithm PrefixSum is an immediate extension of the classical PRAM recursive algorithm. The recurrence equations for computing the time and work complexity of PrefixSum are:

T(n) = \left\lfloor \frac{n}{m} \right\rfloor + a,
W(n) = \left\lfloor \frac{n}{m} \right\rfloor + b n,

where a and b are constants. The solutions of these equations are:

T(n) = O(\log_m n),
W(n) = O(n).

3. Conclusion

In the previous section, we have presented some algorithms to solve basic problems in the MRMW model. These algorithms are faster than the ones on a standard CREW PRAM. However, not all problems benefit from multiple access-
es, and there are problems hard to solve on an MRMW PRAM. We can assert that good problems to be solved on this model are those that apply the same associative operation (i.e., $\exists$ the reduction operation, see (2)) to all data. Problems that do not have this feature seem to be hard for the model, as for example sorting, that requires only comparisons of elements. To read $m$ input values is then useless: in fact we do not reduce the $m$ read values into one value with an $\exists$ operation.

Another hard problem is list ranking [3]. Let $L$ a list of $n$ elements and $s[1 \ldots n]$ an array such that $s[i]$ is the pointer to the element following $i$ in $L$, for $1 \leq i \leq n$. The distance (rank) of each element from the end of the list has to be computed. PRAM algorithms solve this problem with the pointer jumping technique. Multiple accesses to memory cannot be used in this case, because the elements are linked in the list $L$, and the only way to know the position of each element is to follow the pointers of the list.

Although the hardness of such problems is not formally shown, the characteristics given before seem to seriously limit the efficiency of the MRMW model on solving them. These problems can be merely solved in the same time of the best algorithms in the PRAM model.

REFERENCES


