

PARALLEL MULTIPLICATION OF A VECTOR BY A KRONECKER PRODUCT OF MATRICES (PART II)

CLAUDE TADONKI * AND BERNARD PHILIPPE †

Abstract The paper provides a generalization of our previous algorithm for the parallel multiplication of a vector by a Kronecker product of matrices. For any p , a factor of the problem size, our algorithm runs on p processors with a minimum number of communication steps and memory space. Specifically, on p processors with global communication, we show that the multiplication requires at least $\Theta(\log(p))$ communication steps, assuming that there is no computation redundancy. This complexity is revised according to the underlying topology, and some performance results on the CRAY T3E are given.

keywords Kronecker product, scheduling, communication, complexity.

AMS subject classifications. 15A15, 15A09, 15A23

1. Introduction. The *Kronecker product* is a well-known matrix algebra operation [2] which appears as a powerful tool for modelling and designing algorithms in number of fields [5]: *Stochastic Automata Networks* [3], *Fast Fourier Transform* (FFT), *Fast Poisson Solver* (FPS) [10] and *Quantum Computation* (QC) [7]. The problem of performing the multiplication of a vector by a Kronecker product of matrices seems to be of particular interest.

It is well-known that for N square matrices of order $n_i, i = 1, \dots, N$, the total number of required floating point multiplications is given by

$$\rho_N = \prod_{i=1}^N n_i \times \sum_{i=1}^N n_i.$$

which is a lower complexity than $\left(\prod_{i=1}^N n_i\right)^2$ which is the number of operations if the matrix was fully assembled. That complexity is achieved by using a stepwise computation scheme following from a popular factorization of the principal matrix. An overview of some related work can be found in [10]. Most authors consider a block decomposition of working arrays and deal with an explicit data shuffling which unfortunately exacerbates the communication overheads. A cyclic distribution is successfully considered in [8, 9].

In this paper, we extend the set of positive integers that can be considered for the number of processors on which our parallel algorithm can be mapped. Indeed, in [8], that number of processors must be a multiple of $n_{s+1}n_{s+2}\dots n_N$ and a divisor of $n_s n_{s+1}\dots n_N$ for a given $s \in \{1, \dots, N\}$. In this paper we allow it to be any divisor of $n_1 n_2 \dots n_N$. We study the interprocessor communications [6] and show that our algorithm involves lowest communication overheads. Experimental results on the CRAY T3E show that the algorithm is quite efficient.

The remainder of the paper is organized as follows. Section 2 provides the necessary background and formalism. Next, we give a general splitting model and present

* University of Yaounde I, 812 Yaounde, Cameroon, (cmtado@uycdc.uninet.cm) and University of Rennes I, Campus de Beaulieu, Rennes, France, (ctadonki@irisa.fr), supported by the French agency "Aire Developpement" and by an Inria/NSF agreement.

† INRIA/IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France, (philippe@irisa.fr).

a stepwise approach for the multiplication. Section 4 describes our parallel schedule. In section 5, we give an explicit representation of the algorithm. Section 6 presents a complexity analysis and is followed by some experimental results in section 7. We conclude in section 8.

2. Background and Formalism. The *Kronecker product* is a matrix algebra operation which can be defined as follows.

DEFINITION 2.1. (*Kronecker product*). If $A \in R^{n_A \times m_A}$ and $B \in R^{n_B \times m_B}$ then the *Kronecker product* $C = A \otimes B$ is the n_A -by- m_A block matrix

$$(2.1) \quad C = (a_{ij}B) \in R^{n_A n_B \times m_A m_B}.$$

Some properties of the *Kronecker product* can be found in [10]. Because of associativity, the Kronecker product $A^{(1)} \otimes A^{(2)} \otimes \dots \otimes A^{(N)}$ of N matrices $A^{(i)}, i = 1, \dots, N$, denoted by $\otimes_{i=1}^N A^{(i)}$, is well defined and we have the following canonical *factorization*:

$$(2.2) \quad \otimes_{i=1}^N A^{(i)} = \prod_{s=1}^N (I_{n_1} \otimes \dots \otimes I_{n_{s-1}} \otimes A^{(s)} \otimes I_{n_{s+1}} \otimes \dots \otimes I_{n_N}),$$

where the matrix multiplication is commutative with these special factors.

Operating with such matrices involves vectors of length $n_1 n_2 \dots n_N$ whose components can be indexed with N -dimensional patterns of the form (i_1, i_2, \dots, i_N) , where $1 \leq i_s \leq n_s, s = 1, \dots, N$. A connection with the corresponding ordinal index is given by the following relation:

$$(2.3) \quad \begin{array}{ccc} & \text{ord} & \\ & \longrightarrow & \\ (i_1, i_2, \dots, i_N) & & (i_1 - 1)r_1 + (i_2 - 1)r_2 + \dots + (i_N - 1)r_N + 1, \\ & \longleftarrow & \\ & \text{lex} & \end{array}$$

where $r_s = n_{s+1} \dots n_N$.

For any divisor p of $n_1 \dots n_N$, there are N positive integers $d_i, i = 1, \dots, N$ such that $p = d_1 d_2 \dots d_N$ and d_i divides n_i for any $i \in \{1, \dots, N\}$. Given p and (n_1, \dots, n_N) , it is obvious that there are generally more than one possible decomposition. Associating the integer p to one of its decompositions (d_1, \dots, d_N) , we derive a multidimensional indexation as previous for p objects.

Let n be a positive integer and $I = \{i_1, \dots, i_d\}, d \leq n$, a subset of $\{1, \dots, n\}$ with $1 \leq i_k < i_{k+1} \leq d = |I|$ for any $k \in \{1, \dots, d-1\}$, then we denote $e_I = [e_{i_1}, e_{i_2}, \dots, e_{i_d}]$, where e_i is the i^{th} column vector of the canonical basis of R^n . Therefore, for a $n \times n$ matrix A , if $I = \{i_1, \dots, i_d\}$, then we have $Ae_I = [Ae_{i_1}, Ae_{i_2}, \dots, Ae_{i_d}]$, which corresponds to the matrix A restricted to selected columns (the square brackets denote a concatenation of columns).

3. Expressions of the multiplication. Given N square matrices $A^{(i)}$ of order $n_i, i = 1, \dots, N$, and a vector $x \in R^L$ with $L = n_1 \dots n_N$, our concern is the multiplication $x(A^{(1)} \otimes \dots \otimes A^{(N)})$ on a distributed memory parallel machine. First note that if $z = x(A^{(1)} \otimes \dots \otimes A^{(N)})$, then for a given index (i_1, i_2, \dots, i_N) , we have

$$(3.1) \quad z(i_1, i_2, \dots, i_N) = x(A^{(1)} e_{i_1} \otimes A^{(2)} e_{i_2} \otimes \dots \otimes A^{(N)} e_{i_N}).$$

Hence, given N nonempty subsets $Q_i \subseteq \{1, \dots, n_i\}, i = 1, \dots, N$, if we consider $z(Q_1, Q_2, \dots, Q_N) = \{z(i_1, i_2, \dots, i_N), i_k \in Q_k, k = 1, \dots, N\}$, then

$$(3.2) \quad z(Q_1, Q_2, \dots, Q_N) = x(A^{(1)} e_{Q_1} \otimes A^{(2)} e_{Q_2} \otimes \dots \otimes A^{(N)} e_{Q_N}).$$

This relation is useful to express a specific part of the multiplication from a static splitting. We now present a recursive way to perform the computation. Because

$$(3.3) \quad \otimes_{i=1}^N A^{(i)} = \prod_{s=1}^N (I_{n_1} \otimes \dots \otimes I_{n_{s-1}} \otimes A^{(s)} \otimes I_{n_{s+1}} \otimes \dots \otimes I_{n_N}),$$

the recursion

$$(3.4) \quad \begin{cases} V^{(N+1)} = x \\ V^{(s)} = V^{(s+1)} (I_{n_1} \otimes \dots \otimes I_{n_{s-1}} \otimes A^{(s)} \otimes I_{n_{s+1}} \otimes \dots \otimes I_{n_N}) \quad 1 \leq s \leq N \end{cases}$$

obviously leads at the last step to $V^{(1)} = x \otimes_{i=1}^N A^{(i)}$.

Since the vectors $V^{(s)}$ are of length $L = n_1 \dots n_N$, they can be addressed with patterns of the form (i_1, \dots, i_N) , where $1 \leq i_s \leq n_s, 1 \leq s \leq N$. From this multidimensional addressing, the previous recursion is explicitly expressed by the following algorithm [8].

For $(i_1, \dots, i_N) = (1, \dots, 1)$ **to** (n_1, \dots, n_N) **do**
 $V^{(s)}(i_1, \dots, i_{s-1}, i_s, i_{s+1}, \dots, i_N) \leftarrow \sum_{t=1}^{n_s} A^{(s)}(t, i_s) V^{(s+1)}(i_1, \dots, i_{s-1}, t, i_{s+1}, \dots, i_N)$
end do

Alg. 1 : Recursion step of the multiplication

Observe that such a step involves $n_1 \dots n_N \times n_s$ floating point multiplications, the complete algorithm achieves the computation with the classical complexity $(n_1 + \dots + n_N) \times n_1 \dots n_N$.

4. Parallel schedule. Recall that our concern is the parallel computation of $z = x(A^{(1)} \otimes \dots \otimes A^{(N)})$, where $A^{(i)}$ are square matrices of order $n_i, i = 1, \dots, N$ and x a vector in R^L with $L = n_1 n_2 \dots n_N$. Given p processors (assuming that p divides L), we proceed as follows. We first compute a sequence of N positive integers $d_i, i = 1, \dots, N$ such that $p = d_1 \dots d_N$ and d_i divides $n_i, i = 1, \dots, N$ (we will later focus on how to obtain such integers). Next, for each $i \in \{1, \dots, N\}$, we consider a partition $Q_{ij}, j = 1, \dots, d_i$ of $Q_i = \{1, \dots, n_i\}$, i.e.

$$(4.1) \quad \bigcup_{j=1}^{d_i} Q_{ij} = Q_i \text{ and } Q_{ij} \neq \emptyset, j = 1, \dots, d_i,$$

and

$$(4.2) \quad (\forall i \in \{1, \dots, N\})(\forall j, k \in \{1, \dots, p_i\}, j \neq k)(Q_{ij} \cap Q_{ik} = \emptyset).$$

Hence, our schedule is given by the following allocation:

$$(4.3) \quad \overbrace{(w_1, w_2, \dots, w_N)}^{\text{processor index}} \leftarrow \overbrace{z(Q_{1w_1}, Q_{2w_2}, \dots, Q_{Nw_N})}^{\text{allocated portion}},$$

where $1 \leq w_i \leq d_i, i = 1, \dots, N$.

Such a computation can be achieved in a similar recursive way as the overall one, through the vectors $V^{(s)}(Q_{1w_1}, Q_{2w_2}, \dots, Q_{Nw_N}), 1 \leq s \leq N+1$. This yields *Alg. 2* for processor (w_1, w_2, \dots, w_N) .

For $(i_1, \dots, i_N) \in Q_{1w_1} \times Q_{2w_2} \times \dots \times Q_{Nw_N}$ **do**
 $V^{(s)}(i_1, \dots, i_{s-1}, i_s, i_{s+1}, \dots, i_N) \leftarrow \sum_{t=1}^{n_s} A^{(s)}(t, i_s) V^{(s+1)}(i_1, \dots, i_{s-1}, t, i_{s+1}, \dots, i_N)$
end do

Alg. 2 : Splitting recursion.

It follows from *Alg. 2* that data communication is required if and only if $t \notin Q_{sw_s}$. The algorithm does not involve computation redundancy since we have considered a *partition* instead of a cover for each level of the computation.

A straightforward method to derive a balanced partition is to perform a block or cyclic splitting of the given set. As we shall see, such partitions lead to a *well balanced computation*, meaning that each processor has the same amount of computation and communication. We now give an algorithm for the decomposition.

Given a sequence of N positive integers (n_1, \dots, n_N) , and another integer p that divides $n_1 n_2 \dots n_N$, the following algorithm performs the desired decomposition.

$(d_1, d_2, \dots, d_N) \leftarrow (1, 1, \dots, 1)$
 $i \leftarrow 1; c \leftarrow p$
while $(c > 1)$ **do**
 $d_i \leftarrow \gcd(c, n_i)$
 $c \leftarrow \frac{c}{d_i}$
 $i \leftarrow i + 1$
end while

Alg. 3. Algorithm for the decomposition.

From the invariant property of the loop expressed by

$$(4.4) \quad (c \times \prod_{i=1}^N d_i = p) \wedge (\forall i \in \{1, \dots, N\} \ d_i \text{ divides } n_i),$$

the algorithm is guaranteed to be correct. Table 4.1 displays the trace of *Alg. 3* with $p = 60$ and $(n_1, n_2, n_3, n_4) = (10, 9, 8, 16)$. Note that terms n_i can be selected in any

i	c	(d_1, d_2, d_3, d_4)
1	60	(1, 1, 1, 1)
2	6	(10, 1, 1, 1)
3	2	(10, 3, 1, 1)
4	1	(10, 3, 2, 1)

TABLE 4.1

Trace of *Alg. 3* for $p = 60$ with $(10, 9, 8, 16)$.

order and this would provide different decompositions. However, as we shall see, these decompositions are equivalent in terms of computation cost when there is no special assumption about the target architecture.

5. Description of the algorithm. Here, we consider the case of balanced partitions, i.e. $|Q_{ij}| = \frac{n_i}{d_i}, 1 \leq i \leq N, 1 \leq j \leq d_i$. Such a partition can be naturally obtained by a *block* or *cyclic* splitting of each set $Q_i = \{1, 2, \dots, n_i\}$. First note that the computation expressed by *Alg. 2* can be written in the following compact form :

$$(5.1) \quad V^{(s)}(Q_{1w_1}, \dots, Q_{sw_s}, \dots, Q_{Nw_N}) = \Psi_s(Q_{sw_s}) V^{(s+1)}(Q_{1w_1}, \dots, Q_s, \dots, Q_{Nw_N}),$$

Id	Time				
	1	2	3	4	5
1	1	2	3	4	5
2	2	3	4	5	1
3	3	4	5	1	2
4	4	5	1	2	3
5	5	1	2	3	4

TABLE 5.1

Identities table of target processors.

where $\Psi_s(Q_{sw_s}) = e_{Q_{1n_1}} \otimes \dots \otimes e_{Q_{s-1,n_{s-1}}} \otimes A^{(s)} e_{Q_{sw_s}} \otimes e_{Q_{s+1,n_{s+1}}} \otimes \dots \otimes e_{Q_{Nn_N}}$.

Hence, from the fact that $Q_s = \bigcup_{T=1}^{p_s} Q_{sT}$, each processor (w_1, \dots, w_N) should broadcast its local portion $V^{(s+1)}(Q_{1w_1}, \dots, Q_{sw_s}, \dots, Q_{Nw_N})$ to all processors of the set $\Gamma(w, s) = \{(w_1, \dots, w_{s-1}, T, w_{s+1}, \dots, w_N), T = 1, \dots, d_s\}$. We denote this communication by $\text{broadcast}(w, s)$. This yields the overall algorithm *Alg. 4*.

```

For  $s \leftarrow N$  downto 1 do
   $V^{(s)} \leftarrow 0$ 
   $\text{broadcast}(w, s)$ 
  For  $(i_1, \dots, i_N) \in Q_{1w_1} \times Q_{2w_2} \times \dots \times Q_{Nw_N}$  do
    For  $T = 1$  to  $d_s$  do
       $V^{(s)}(i_1, \dots, i_{s-1}, i_s, i_{s+1}, \dots, i_N) \leftarrow V^{(s)}(i_1, \dots, i_{s-1}, i_s, i_{s+1}, \dots, i_N) +$ 
       $+ \sum_{t \in Q_{sT}} A^{(s)}(t, i_s) V^{(s+1)}(i_1, \dots, i_{s-1}, t, i_{s+1}, \dots, i_N)$ 
    end do
  end do
end do

```

Alg. 4 : Overall SPMD algorithm with local broadcasts.

This algorithm provides an efficient communication organization since it only involves *broadcast* routines. However, there is an significant bufferization which can be avoided by overlapping communication and computation. Indeed, instead of performing a broadcast at the beginning of a step, each processor could perform a communication loop while operating on received data. Because operation performed with subvectors is commutative, they can be received in any order. Hencefore, an *all-to-all* communication onto processors subset $\Gamma(w, s) = \{(w_1, \dots, w_{s-1}, T, w_{s+1}, \dots, w_N), T = 1, \dots, d_s\}$ can be achieved in d_s steps following a *Hankel matrix* for target processors selection. Table 5.1 displays an example for $|\Gamma(w, s)| = 5$ processors identified with the component T . Each row is assigned to the corresponding processor and for each time (in column), it gives the identity of the target processor. In general, the table is a *Hankel* matrix based on the set $\{1, \dots, n\}$, where n is the number of involved processors. For each processor, the first value of T is w_s , and this means that it uses the portion stored in its local memory. This is an important feature to overlap communication and computation by a suitable anticipation of *send* and *receive* routines. Let us turn to a formal analysis of the communication on a subset of d processors. At time t , if $\alpha_t(w)$ (resp. $\beta_t(w)$) is the identity of the processor to (resp. from) whom processor w should send a message (resp. receive a message), then the following recurrence

holds:

$$(5.2) \quad \begin{cases} \alpha_1(w) &= w \\ \alpha_t(w) &= \alpha_{t-1}(w) \bmod d+1 \quad : 1 < t \leq d \end{cases}$$

$$(5.3) \quad \begin{cases} \beta_1(w) &= w \\ \beta_t(w) &= d - (d - \beta_{t-1}(w) - 1) \bmod d \quad : 1 < t \leq d \end{cases}$$

We are now ready to express our algorithm. For this purpose, let us consider the following commands. The command **send**(T) means that processor $w = \text{ord}(w_1, \dots, w_{s-1}, w_s, w_{s+1}, \dots, w_N)$ sends its portion $V^{(s+1)}(Q_{1w_1}, \dots, Q_{sw_s}, \dots, Q_{Nw_N})$ to processor $(w_1, \dots, w_{s-1}, T, w_{s+1}, \dots, w_N)$ with the label T , whereas the command **recv**(T) means that processor w receives the portion $V^{(s+1)}(Q_{1w_1}, \dots, Q_{sT}, \dots, Q_{Nw_N})$ with label T from processor $(w_1, \dots, w_{s-1}, T, w_{s+1}, \dots, w_N)$.

```

For  $s \leftarrow N$  downto 1 do
   $V^{(s)} \leftarrow 0$ 
   $\alpha \leftarrow w$ ;  $\beta \leftarrow w$ ;  $d \leftarrow d_s$ 
  For  $(i_1, \dots, i_N) \in Q_{1w_1} \times Q_{2w_2} \times \dots \times Q_{Nw_N}$  do
     $V^{(s)}(i_1, \dots, i_{s-1}, i_s, i_{s+1}, \dots, i_N) \leftarrow V^{(s)}(i_1, \dots, i_{s-1}, i_s, i_{s+1}, \dots, i_N) +$ 
     $+ \sum_{t \in Q_{s\alpha}} A^{(s)}(t, i_s) V^{(s+1)}(i_1, \dots, i_{s-1}, t, i_{s+1}, \dots, i_N)$ 
  end do
  For  $T = 2$  to  $d_s$  do
     $\alpha \leftarrow \alpha \bmod d+1$ 
     $\beta \leftarrow d - (d - \beta - 1) \bmod d$ 
    send( $\alpha$ )
    recv( $\beta$ )
    For  $(i_1, \dots, i_N) \in Q_{1w_1} \times Q_{2w_2} \times \dots \times Q_{Nw_N}$  do
       $V^{(s)}(i_1, \dots, i_{s-1}, i_s, i_{s+1}, \dots, i_N) \leftarrow V^{(s)}(i_1, \dots, i_{s-1}, i_s, i_{s+1}, \dots, i_N) +$ 
       $+ \sum_{t \in Q_{s\beta}} A^{(s)}(t, i_s) V^{(s+1)}(i_1, \dots, i_{s-1}, t, i_{s+1}, \dots, i_N)$ 
    end do
  end do
end do

```

Alg. 5 : Overall SPMD algorithm with communication loops.

More explicit details on this algorithm are given in section 7. We now turn to the complexity analysis.

6. Complexity. We emphasize the fact that we only consider non redundant computation with balanced partitions. Since our algorithm involves global communications, we restate some results on their optimal cost (in terms of communication steps) depending on the underlying topology. These results and more can be found in [6]. The number of communication steps necessary to broadcast a message to p processors is :

- $O(\log(p))$ on tree, hypercube and Bruijn topologies.
- $O(\sqrt{p})$ on two dimensional grid.
- $O(p)$ on linear and ring topologies.

Let $\Gamma(p)$ denote this value. It is easy to prove that if a and b are two integers such that $a, b > 1$, then $\Gamma(a) + \Gamma(b) \leq \Gamma(ab)$ (the equality occurs in the logarithmic

case). Since in our algorithm, each processor broadcasts its entire local portion to the subset in which it belongs, we just have to consider the function $\Gamma(p)$ multiplied by the factor $\beta + \tau L/p$, where L/p is the size of the message and β, τ the transmission parameters. We will therefore make abstraction of this factor.

THEOREM 6.1. *Algorithm Alg. 4 performs the multiplication with at least $O(\Gamma(p))$ parallel transmissions.*

Proof. At step s , each processor w broadcasts its portion to $|\Gamma(w, s)| = d_s$ processors. Since this can be performed in $\Gamma(d_s)$ communicating steps, the overall number of parallel transmissions is

$$(6.1) \quad \sum_{i=1}^N (\Gamma(d_i)) \leq \Gamma\left(\prod_{i=1}^N d_i\right) = \Gamma(p).$$

□

The following result shows that it is not possible to further reduce that number.

THEOREM 6.2. *Given N square matrices $A^{(i)}$ of order $n_i, i = 1, \dots, N$, and a vector $x \in R^L$ with $L = n_1 \dots n_N$, multiplication $x(A^{(1)} \otimes \dots \otimes A^{(N)})$ performed in parallel on p processors needs at least $\Theta(\log(p))$ communication steps.*

Proof. From the relation $z(i_1, i_2, \dots, i_N) = x(A^{(1)}e_{i_1} \otimes A^{(2)}e_{i_2} \otimes \dots \otimes A^{(N)}e_{i_N})$, it follows that the computation of each entry of the resulting vector involves the entire input vector x . Hence, assuming that the input vector x is distributed without duplication of entries and that there is no idle processor nor redundant computation, the communication graph derived from any parallel computation is a strongly connected graph. This implies at least $\Theta(\log(p))$ communication steps.

□

In Alg. 4, the memory requirement per processor is $Max\{d_i\} \times \frac{L}{p}$, where $L = n_1 \dots n_N$. This is the consequence of performing all required communications before computing. The solution provided by Alg. 5 reduces this requirement to $3\frac{L}{p}$. This reduction is an important feature for computations involving very large arrays since each fraction may then fit in the local memory of processor and furthermore, the overall task would require only one *read-access* to disk for the input vector x and only one *write-access* to the disk for the output vector z . However, the number of communication steps becomes $(d_1 - 1) + \dots + (d_s - 1) + \dots + (d_N - 1)$ instead of $\Gamma(p)$ in general, and this raises the problem of finding a decomposition with a minimum sum to achieve the best performance. We now focus on this question. Observe that, given N integers $n_i, i = 1, \dots, N$ and another integer p , if (d_1, \dots, d_N) is a valid decomposition of p with a minimum sum, then for $i, j, 1 \leq i < j \leq N$, $(d_i, d_{i+1}, \dots, d_j)$ is a valid decomposition of $\pi_{ij} = d_i d_{i+1} \dots d_j$ towards $(n_i, n_{i+1}, \dots, n_j)$ with a minimum sum. This is an optimality result which provides a deterministic procedure following a *dynamical programming* model to find the required decomposition. Unfortunately, this does not help to avoid an exhaustive exploration of all valid decompositions, which may be expensive. To illustrate the difficulty[4] of this problem, consider the case when $N = p$ and $n_i = p, i = 1, \dots, N$. Since an integer p cannot have more than p factors and that each of them is a factor of any n_i (in this case), it follows that the decomposition which gives a minimal sum is composed of prime factors of p . Hence in this trivial case, our problem is as hard as the integer factorization [1]. This forces to turn to consider heuristics like *genetic algorithms* and others. A simple solution may be expressed as follows :

```

 $(d_1, d_2, \dots, d_N) \leftarrow (1, 1, \dots, 1)$ 
 $i = 1$ 
while  $(p > 1)$  do
   $c = \max(n_i, n_{i+1} \dots n_N)$ 
   $e \leftarrow \gcd(p, c)$ 
   $d_i \leftarrow \frac{p}{e}$ 
   $p \leftarrow e$ 
  if  $(c = n_i)$  then  $d_i \leftrightarrow p$  { swap }
   $i \leftarrow i + 1$ 
end while

```

Alg. 6 : Heuristic algorithm for the *minimum sum* decomposition.

An execution on our example $p = 60$ and $(10, 9, 8, 16)$ gives the decomposition $(5, 3, 1, 4)$ with the sum equal to 13, whereas the optimal decomposition is $(5, 3, 2, 2)$ with the sum equal to 12. Note that, not splitting the factor 4 has been the main reason to have missed the optimal decomposition. Such a splitting may provide a possible refinement of our algorithm, but we will not go farther in our investigation on this topic.

7. Implementation and performance.

7.1. Implementation. First, observe that in a balanced partition, the components of $V^{(s)}(Q_{1w_1}, \dots, Q_{sw_s}, \dots, Q_{Nw_N})$ can be stored in a contiguous vector of length

$$(7.1) \quad |Q_{1w_1} \times \dots \times Q_{sw_s} \times \dots \times Q_{Nw_N}| = \prod_{i=1}^N |Q_{iw_i}| = \prod_{i=1}^N \frac{n_i}{d_i} = \frac{\prod_{i=1}^N n_i}{\prod_{i=1}^N d_i} = \frac{L}{p},$$

where $L = n_1 \dots n_N$. Let $c_i = \frac{n_i}{d_i}$ and consider a block partition . Hence, for $i, 1 \leq i \leq N$ and $j, 1 \leq j \leq d_i$, we have

$$(7.2) \quad Q_{ij} = \{\alpha_i + 1, \alpha_i + 2, \dots, \alpha_i + c_i\},$$

where $\alpha_i = (j-1)c_i$. Furthermore, from the correspondance provided by 2.3, we have

$$(7.3) \quad \text{ord}(i_1, \dots, t, \dots, i_N) = \text{ord}(i_1, \dots, i_s, \dots, i_N) + (t - i_s) \times c_{s+1} \dots c_N,$$

$$(7.4) \quad \text{ord}(w_1, \dots, T, \dots, w_N) = \text{ord}(w_1, \dots, w_s, \dots, w_N) + (T - w_s) \times d_{s+1} \dots d_N.$$

If $w = \text{ord}(w_1, \dots, w_s, \dots, w_N)$ then

$$(7.5) \quad w_s = [(w - 1) \mathbf{div}(d_{s+1} \dots d_N)] \mathbf{mod}(d_s) + 1,$$

where \mathbf{div} denotes the division of integers.

```

 $\pi \leftarrow 1; r \leftarrow 1; \ell \leftarrow c_1 c_2 \dots c_N = L/p \quad /* c_i = \frac{n_i}{d_i} */$ 
 $y \leftarrow x(Q_{1w_1}, Q_{2w_2}, \dots, Q_{Nw_N})$ 
For  $s \leftarrow N$  downto 1 do
   $\ell \leftarrow \ell / c[s]$ 
   $ws = [w \mathbf{div}(\pi)] \mathbf{mod}(d[s]) + 1$ 
   $e \leftarrow (ws - 1) \times c[s]$ 
   $v \leftarrow 0$ 
   $i \leftarrow 1$ 
  For  $a \leftarrow 1$  to  $\ell$  do
    For  $j \leftarrow e + 1$  to  $e + c[s]$  do
      For  $b \leftarrow 1$  to  $r$  do
        For  $t \leftarrow e + 1$  to  $e + c[s]$  do
           $v[i] \leftarrow v[i] + A(s, t, j)y[I + (t - j)r]$ 
        end do
         $i \leftarrow i + 1$ 
      end do
    end do
  end do
  If  $(ws = 1)$  then  $H \leftarrow d$  else  $H \leftarrow ws - 1$ 
  For  $T = ws + 1$  to  $ws + d[s] - 1$  do
     $G \leftarrow \mathbf{mod}(T - 1, d[s]) + 1$ 
     $idest \leftarrow w + (G - ws) \times \pi$ 
     $isender \leftarrow w + (H - ws) \times \pi$ 
    send $(y, idest, ws)$ 
    recv $(u, isender, H)$ 
     $e \leftarrow (H - 1) \times c[s]$ 
     $i \leftarrow 1$ 
    For  $a \leftarrow 1$  to  $\ell$  do
      For  $j \leftarrow 1$  to  $c[s]$  do
        For  $b \leftarrow 1$  to  $r$  do
          For  $t \leftarrow e + 1$  to  $e + c[s]$  do
             $v[i] \leftarrow v[i] + A(s, t, j)u[I + (t - j)r]$ 
          end do
           $i \leftarrow i + 1$ 
        end do
      end do
    end do
    If  $(H = 1)$  then  $H \leftarrow d[s]$  else  $H \leftarrow H - 1$ 
  end do
   $r \leftarrow r \times c[s]$ 
   $\pi \leftarrow \pi \times d[s]$ 
  If  $(s > 1)$  then  $y \leftarrow v$ 
end do
 $z(Q_{1w_1}, Q_{2w_2}, \dots, Q_{Nw_N}) \leftarrow v$ 

```

Alg. 7: Implementation of the global algorithm.

7.2. Performance. The target machine is a CRAY T3E, which is a standard distributed memory parallel machine with up to 256 processors, each running at 300 MHZ. We have considered the case of $N = 6$ square matrices, each of order 12 (i.e. $L = 2,985,984$). Best efficiencies are obtained on power of two since it implies only

p	(d_1, \dots, d_6)	$T(sec)$	σ	e
1	(1, 1, 1, 1, 1, 1)	19.2713	1	1
2	(1, 1, 1, 1, 1, 2)	9.7235	1.98	0.99
3	(1, 1, 1, 1, 1, 3)	6.6964	2.87	0.95
4	(1, 1, 1, 1, 2, 2)	4.8267	3.99	0.99
6	(1, 1, 1, 1, 2, 3)	3.4404	5.60	0.93
8	(1, 1, 1, 2, 2, 2)	2.4755	7.78	0.97
9	(1, 1, 1, 1, 3, 3)	2.3436	8.2	0.91
12	(1, 1, 1, 2, 2, 3)	1.8949	10.17	0.84
16	(1, 1, 2, 2, 2, 2)	1.3376	14.40	0.90
18	(1, 1, 1, 2, 3, 3)	1.3247	14.54	0.81
24	(1, 1, 2, 2, 2, 3)	0.9019	21.36	0.89
27	(1, 1, 1, 3, 3, 3)	0.8130	23.70	0.98
32	(1, 2, 2, 2, 2, 2)	0.6423	30.00	0.93

TABLE 7.1

Performances on CRAY T3E (σ : speedup, e : efficiency).

one transmission per step for each processor (more efficient on hypercube topology). Timings are reported in Table 7.1. The table includes the speedup (ratio between the sequential time and the parallel time) and the efficiency (the speedup divided by the number of used processors). The average efficiency is 0.9. Finally, we note the fact that, by using the algorithm of [8], we would have missed the subset $\{8, 9, 16, 18, 27\}$ for the number of processors. In general, this difference would be considerable, and the present solution gives us more chance to fully use a given number of available processors. However, in the set of values that can be used for the number of processors in the two cases, the scheduling is the same and that is why we do not need to give experimental performance for the algorithm of [8].

8. Conclusion. The purpose of this paper was to provide an efficient parallel algorithm for the multiplication of a vector by a Kronecker product of matrices. We have extended the idea in [8] to all numbers of processors that are factors of the length of the problem. This extension is significant since the present requirement is the minimum one for a well balanced computation. We have shown that our algorithm performs the multiplication with a minimum number of communication steps and is efficient because each of them does not involve any preprocessing. A difficult combinatorial problem seems to be the price to achieve the best performance with communication loop. However, since the size of matrices may not be too large, heuristic algorithms may succeed in the determination of suitable decompositions. These algorithms remain to be designed.

REFERENCES

- [1] R. P. Brent, *Some Parallel Algorithm for Integer Factorization.*, Europar'99 Parallel Processing, Lecture Note in Computer Science, No. 1685, pages 1-22, Springer Verlag, 1999.
- [2] M. Davio, *Kronecker Products and Shuffle Algebra.*, IEEE Trans. Comput., Vol. C-30, No. 2, pp. 116-125, 1981.
- [3] P. Fernandes, B. Plateau, and W. J. Stewart, *Efficient Descriptor-Vector Multiplications in Stochastic Automata Networks*, INRIA Rapport de recherche interne No. 2935, July 1996.
- [4] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Compagny, July 1996.

- [5] J. Granta, M. Conner, and R. Tolimieri, *Recursive fast algorithms and the role of the tensor product*, IEEE Transaction on Signal Processing, 40(12):2921-2930, December 1992.
- [6] J. De Rummeur, *Communications dans les réseaux de processeurs*, Masson, 1994.
- [7] P. W. Shor, *Quantum Computing*, Proceeding of the ICM Conference, 1998.
- [8] C. Tadonki and B. Philippe, *Parallel Multiplication of a Vector by a Kronecker Product of Matrices*, Journal of Parallel and Distributed Computing and Practices, To appear, 1999.
- [9] C.Tong and P. N. Swarztrauber, *Ordered Fast Fourier Transforms on Massively Parralel Hypercube Multiprocessor*, Journal of Parallel and Distributed Computing 12, 50-59, 1991.
- [10] C. Van Loan, *Computational Framework for the Fast Fourier Transform*, SIAM, 1992.