

Programming Schemata for Tensor Products

R. W. Johnson*

Department of Computer Science
St. Cloud State University
St. Cloud, MN 56301

J. R. Johnson[†]

Department of Mathematics and Computer Science
Drexel University
Philadelphia, PA 19104

*Supported in part by Defense Advanced Research Projects Agency DARPA Order No. 7898, monitored by NIST under contract No. 60NANB1D1151.

Abstract

Previously [7, 8], we presented a methodology for translating mathematical formulas involving matrix operations, tensor products and a special class of permutations, into programming constructs suitable to a variety of parallel and vector machines. In this paper we discuss an implementation of this methodology. As an application we derive a program for computing an arbitrary multi-dimensional finite Fourier transform using the CRAY Scientific Library routines for the CRAY Y-MP.

Keywords

Linear computation, tensor product, multi-dimensional Fourier transform, parallel and vector computation, CRAY Y-MP

1 Introduction

In previous work [7, 8], we presented a methodology for translating mathematical formulas involving matrix operations, tensor products and a special class of permutations, into programming constructs suitable to a variety of parallel and vector computers. This methodology is suitable to the design and implementation of algorithms that can be described by mathematical formulas of this type. Important examples are the finite Fourier Transform [7, 9, 10] and block recursive matrix algorithms such as Strassen's matrix multiplication algorithm [8]. In this paper we discuss an implementation of this methodology.

Assuming that we are given code segments implementing the computation $y = A_i x$ our goal is to produce a program implementing the linear computation

$$y = (A_1 \otimes A_2 \otimes \cdots \otimes A_t) x = \left(\bigotimes_{i=1}^t A_i \right) x, \quad (1)$$

where \otimes denotes the tensor product. There are many different implementations, called tensor product schemata, of Equation 1 corresponding to different matrix factorizations of $(A_1 \otimes \cdots \otimes A_t)$. We present a translation scheme that can be used to map different factorizations into different programming schemata. Each schema has different performance characteristics

due to different addressing requirements, different memory locality, and varying amounts of vectorization and parallelism.

An important aspect of this paper is a description of the addressing requirements needed to combine code segments into an arbitrary tensor product schema. It is interesting to note that extra stride parameters, not usually available, are needed to make the code sufficiently general to be used in an arbitrary tensor product factorization. Provided that a code segment has appropriate addressing parameters it can be inserted into any of our program schemata. Therefore this approach provides a general mechanism for combining linear programs using the tensor product. This flexibility makes it easy to mix and match different algorithms, incorporate improved algorithms, and to reorganize a program to adapt to new architectures. Moreover, the ability to easily combine a large collection of program modules in this manner is necessary to implement complicated algorithms such as Fourier transform algorithms that respect crystallographic symmetry [3, 2].

2 Tensor Product Factorizations

Let

$$A = \begin{pmatrix} a_{1,1} & \cdots & a_{1,n_1} \\ \vdots & \ddots & \vdots \\ a_{m_1,1} & \cdots & a_{m_1,n_1} \end{pmatrix},$$

and

$$B = \begin{pmatrix} b_{1,1} & \cdots & b_{1,n_2} \\ \vdots & \ddots & \vdots \\ b_{m_2,1} & \cdots & b_{m_2,n_2} \end{pmatrix}.$$

The tensor (or Kronecker) product, $A \otimes B$, of A and B is the block matrix obtained by replacing each element of $a_{i,j}$ by the matrix $a_{i,j}B$:

$$\begin{pmatrix} a_{1,1}B & \cdots & a_{1,n_1}B \\ \vdots & \ddots & \vdots \\ a_{m_1,1}B & \cdots & a_{m_1,n_1}B \end{pmatrix}.$$

The tensor product satisfies the following basic properties, where I_n is the $n \times n$ identity matrix, indicated inverses exist, and matrix dimensions are such that all products make sense.

1. $(\alpha A) \otimes B = A \otimes (\alpha B) = \alpha(A \otimes B)$.
2. $(A + B) \otimes C = (A \otimes C) + (B \otimes C)$.
3. $A \otimes (B + C) = (A \otimes B) + (A \otimes C)$.
4. $1 \otimes A = A \otimes 1 = A$.
5. $A \otimes (B \otimes C) = (A \otimes B) \otimes C$.
6. ${}^t(A \otimes B) = {}^tA \otimes {}^tB$.
7. $(A \otimes B)(C \otimes D) = AC \otimes BD$.
8. $A \otimes B = (I_{m_1} \otimes B)(A \otimes I_{n_2}) = (A \otimes I_{m_2})(I_{n_1} \otimes B)$.
9. $(A_1 \otimes \cdots \otimes A_t)(B_1 \otimes \cdots \otimes B_t) = (A_1 B_1 \otimes \cdots \otimes A_t B_t)$.
10. $(A_1 \otimes B_1) \cdots (A_t \otimes B_t) = (A_1 \cdots A_t \otimes B_1 \cdots B_t)$.
11. $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$.
12. $I_m \otimes I_n = I_{mn}$.

All of these identities follow from the definition or simple applications of preceding properties (see [5]).

Associated with the tensor product are a class of permutations called tensor permutations. If π be a permutation of $\{1, \dots, t\}$, then the tensor permutation associated with π is the permutation matrix defined by the property $P_\pi(A_1 \otimes \cdots \otimes A_t) = (A_{\pi(1)} \otimes \cdots \otimes A_{\pi(t)})P_\pi$. The special case when there are only two factors is called the commutation theorem and the associated permutation is called a stride permutation.

Definition 1 (Stride Permutation) *Let x be a vector of length m and y be a vector of length n .*

$$L_n^{mn}(x \otimes y) = y \otimes x.$$

The notation indicates that elements of a vector of length mn are loaded into n segments each at stride n . If ${}^t x = (x_0, x_1, \dots, x_{mn-1})$, the transpose of the column vector x , then

$${}^t(L_n^{mn} x) = (x_0, x_n, \dots, x_{(m-1)n}, \dots, x_{n-1}, x_{2n-1}, \dots, x_{mn-1}).$$

Listed below are some of the important properties of stride permutations. Proofs easily follow from the definition. A more complete discussion of tensor permutations and their properties can be found in [6].

1. $L_s^{rst} L_t^{rst} = L_{st}^{rst}$.
2. $L_n^N L_{N/n}^N = I_N$.
3. $L^{rst} = (L_t^{rt} \otimes I_s)(I_r \otimes L_t^{st})$.

Theorem 1 (Commutation) *If A is an $m_1 \times n_1$ matrix, and B is an $m_2 \times n_2$ matrix, then*

$$A \otimes B = L_{m_1}^{m_1 m_2} (B \otimes A) L_{n_2}^{n_1 n_2} \quad (2)$$

If the input vector x is grouped into segments x_i^n of length n beginning with the element x_i , then the operation $A \otimes I_n$ can be viewed as a vector operation on vectors of length n .

$$\begin{aligned} (A \otimes I_n)x = & \begin{pmatrix} a_{0,0}I_n & \cdots & a_{0,m-1}I_n \\ \vdots & \ddots & \vdots \\ a_{m-1,0}I_n & \cdots & a_{m-1,m-1}I_n \end{pmatrix} \begin{pmatrix} x_0^n \\ x_n^n \\ \vdots \\ x_{(m-1)n}^n \end{pmatrix} = \\ & \begin{pmatrix} a_{0,0}x_0^n + a_{0,1}x_n^n + \cdots + a_{0,m-1}x_{(m-1)n}^n \\ \vdots \\ a_{m-1,0}x_0^n + a_{m-1,1}x_n^n + \cdots + a_{m-1,m-1}x_{(m-1)n}^n \end{pmatrix}. \end{aligned}$$

This operation consists of scalar vector multiplications $a_{i,j}x_{jn}^n$ and vector additions.

The operation $y = (I_m \otimes B)x$ can be interpreted either as a loop or as a parallel operation.

$$(I_m \otimes B)x = \begin{pmatrix} B & & \\ & B & \\ & & \ddots \\ & & & B \end{pmatrix} \begin{pmatrix} x_0^n \\ x_n^n \\ \vdots \\ x_{(m-1)n}^n \end{pmatrix}$$

consists of m copies of B acting on the disjoint data x_{in}^n . It is possible to exchange parallel operations for vector operations, and vice versa, with the use of the commutation theorem.

Let A^{m_i, n_i} be an $m_i \times n_i$ matrix. The general tensor product operation

$$y = (A^{m_1, n_1} \otimes \dots \otimes A^{m_t, n_t})x$$

can be written in many different ways, leading to many different implementations. We present three different factorizations that can easily be proven using induction and properties of the tensor product and stride permutations. Let $N(i) = n_1 \cdots n_i$, for $1 \leq i \leq t$, $N = N(t)$, and $N(0) = 1$. Also, let $\bar{N}(i) = N/N(i)$. Similarly, let $M(i) = m_1 \cdots m_i$, $M = M(t)$, $M(0) = 1$, and $\bar{M}(i) = M/M(i)$.

Theorem 2 (Fundamental Factorization)

$$A^{m_1, n_1} \otimes \dots \otimes A^{m_t, n_t} = \prod_{i=1}^t \left(I_{N(i-1)} \otimes A^{m_i, n_i} \otimes I_{\bar{M}(i)} \right).$$

The special case when $A^{m_1, n_1} = \dots = A^{m_t, n_t}$ is highlighted in the following corollary.

Corollary 1 *Let A be an $m \times n$ matrix. Then*

$$\bigotimes_{i=1}^t A = A \otimes \dots \otimes A = \prod_{i=1}^t (I_{n^{i-1}} \otimes A \otimes I_{m^{t-i}}).$$

If $m_i = n_i$ then all of the factors in the Fundamental factorization commute and the factorization is true for any ordering of the product. However, if the matrices are rectangular, the factors no longer commute as seen in Property 8 of the tensor product. In this case Property 8 can be used to obtain the following Corollary of Theorem 2.

Corollary 2 (Fundamental Factorization) *Let π be a permutation of the set $\{1, \dots, t\}$, and let $p(i, j) = m_j$ if $\pi(i)$ comes before $\pi(j)$ in the sequence $(\pi(1), \dots, \pi(t))$ and n_j otherwise. Then*

$$A^{m_1, n_1} \otimes \dots \otimes A^{m_t, n_t} = \prod_{i=1}^t \left(I_{P(i)} \otimes A^{m_{\pi(i)}, n_{\pi(i)}} \otimes I_{\bar{P}(i)} \right),$$

where $P(i) = \prod_{j=1}^{\pi(i)-1} p(i, j)$ and $\bar{P}(i) = \prod_{j=\pi(i)+1}^t p(i, j)$.

The fundamental factorization can be modified to obtain several equivalent forms.

Theorem 3 (Parallel Factorization)

$$A^{m_1, n_1} \otimes \dots \otimes A^{m_t, n_t} = \prod_{i=1}^t L_{m_i}^{N^{(i-1)}\bar{M}^{(i-1)}} \left(I_{N^{(i-1)}\bar{M}^{(i)}} \otimes A^{m_i, n_i} \right).$$

Theorem 4 (Vector Factorization)

$$A^{m_1, n_1} \otimes \dots \otimes A^{m_t, n_t} = \prod_{i=1}^t \left(A^{m_i, n_i} \otimes I_{\bar{M}^{(i)}N^{(i-1)}} \right) L_{n_i}^{\bar{M}^{(i)}N^{(i)}}.$$

3 Indexing Notation

Let e_i^m be the vector of length m with a one in the i -th position and zeros elsewhere. The set $\{e_i^m, i = 0, \dots, m - 1\}$ form the standard basis for the vector space, \mathbf{F}^m , of m -tuples of elements in the field \mathbf{F} . These basis elements serve as placeholders for the elements of the vector $x = \sum_{i=0}^{m-1} x_i e_i^m$, and since we are interested in linear computations, the computation $y = Ax$ is completely described in terms of these basis elements.

Closely associated with the basis elements e_i^m are the basis elements, ${}^t e_i^m$ of the dual space, which serve as access operators. If X is an array of size m containing the elements of the vector x , the array access $X[i]$ is equivalent to applying the access operator ${}^t e_i^m$ to the vector x (i.e. ${}^t e_i^m x = x_i$). We will use the notation $X[{}^t e_i^m]$ to denote the evaluation of the access function ${}^t e_i^m$ applied to the vector x . Storing an element in the i -th position of a vector, y , is the dual operation and is denoted by $Y[e_i^m]$. The indexing operations of a linear computation can be described using access and storage operators.

Since we are interested in writing programs to compute $y = (A^{m_1, n_1} \otimes \dots \otimes A^{m_t, n_t})x$ we need bases of \mathbf{F}^M and \mathbf{F}^N (where $M = m_1 \dots m_t$ and $N = n_1 \dots n_t$) that are compatible with $A^{m_1, n_1} \otimes \dots \otimes A^{m_t, n_t}$. Using the bilinearity of the tensor product, it is easy to see that the collection of vectors $x_1 \otimes \dots \otimes x_t$, where $x_i \in \mathbf{F}^{n_i}$, form a vector space, $\mathbf{F}^{n_1} \otimes \dots \otimes \mathbf{F}^{n_t}$, called the tensor product of the vector spaces \mathbf{F}^{n_i} , which is isomorphic to \mathbf{F}^N . The

benefit of using this representation of \mathbf{F}^M is that it is compatible with the tensor product. Property 9 of the tensor product shows that $(A^{m_1, n_1} \otimes \cdots \otimes A^{m_t, n_t})(x_1 \otimes \cdots \otimes x_t) = A^{m_1, n_1} x_1 \otimes \cdots \otimes A^{m_t, n_t} x_t$.

Definition 2 (Tensor Basis) *The collection of elements $e_{i_1}^{m_1} \otimes \cdots \otimes e_{i_t}^{m_t}$ form a basis, called the tensor basis, for $\mathbf{F}^{m_1} \otimes \cdots \otimes \mathbf{F}^{m_t}$ and hence a basis for \mathbf{F}^M .*

The connection between this basis and the standard basis for \mathbf{F}^M is given by the equation

$$e_{i_1}^{m_1} \otimes \cdots \otimes e_{i_t}^{m_t} = e_{i_1 \bar{M}(1) + \cdots + i_{t-1} \bar{M}(t-1) + i_t \bar{M}(t)}^M. \quad (3)$$

This equation is equivalent to lexicographically ordering the elements of the tensor basis. Just as we interpreted the basis elements e_i^m and the dual basis elements ${}^t e_i^m$ as storage and access operators, the elements of the tensor basis can be viewed as indexing operators and Equation 3 shows that these operators are equivalent to storing and accessing a multi-dimensional array that is ordered lexicographically.

3.1 Block Access Operators

The notation

$$X[{}^t e_{i_1}^{m_1} \otimes \cdots \otimes {}^t e_{i_k}^{m_k} \otimes \cdots \otimes {}^t e_{i_t}^{m_t}]_{i_k=0}^{m_k-1}$$

denotes a vector of length m_k whose k -th component is $X[i_1 \bar{M}(1) + \cdots + i_k \bar{M}(k) + \cdots + i_t \bar{M}(t)]$ for $i_k = 0, \dots, m_k - 1$. This subvector obtained from X can be conveniently described using a based vector with stride.

A based vector with stride,

$$x_{b,s}^n = \{x + b, x + b + s, x + b + 2s, \dots, x + b + (n - 1)s\}$$

is a vector of length n with stride s and base b . We have the notationally useful defaults: if there is no n , assume $n = 1$, if no s , $s = 1$, and if no b , $b = 0$. Note this is consistent with the usual coordinate notation; x_0, x_1, \dots, x_{m-1} is a collection of m vectors of length 1.

In the example above, the operator that accesses m_k elements of the vector x is $(e_{i_1}^{m_1} \otimes \cdots \otimes I_{m_k} \otimes \cdots \otimes e_{i_t}^{m_t})$. The particular based vector

with stride obtained by applying this block access operator is given by the following equation.

$${}^t(e_{i_1}^{m_1} \otimes \cdots \otimes I_{m_k} \otimes \cdots \otimes e_{i_t}^{m_t})X = X_{i_1 \bar{M}(1) + \cdots + i_{k-1} \bar{M}(k-1) + i_{k+1} \bar{M}(k+1) + \cdots + i_t \bar{M}(t), \bar{M}(k)}$$

These considerations lead to an even more general notation which is useful in describing our programming schemata.

$$x_{b,u,s}^{m,n} = \{x + b, x + b + s, x + b + 2s, \dots, x + b + (n-1)s, \\ x + b + u, x + b + u + s, x + b + u + 2s, \dots, x + b + u + (n-1)s, \\ x + b + 2u, x + b + 2u + s, x + b + 2u + 2s, \dots, x + b + 2u + (n-1)s, \\ \cdot \\ \cdot \\ \cdot \\ x + b + (m-1)u, x + b + (m-1)u + s, x + b + (m-1)u + 2s, \dots, x + b + (m-1)u + (n-1)s\}$$

where u is called the base stride. This is a vector of length mn ; it is often convenient to think of it as m vectors of length n , but this is still one vector (i.e. it is raveled in the way indicated.)

3.2 Examples

The following two examples illustrate the use of this notation. Consider the linear computation corresponding to the stride permutation. The stride permutation can be described by its action on the tensor basis $L_n^{mn}(e_i^m \otimes e_j^n) = e_j^n \otimes e_i^m$, and the corresponding program is described by the evaluation of the access and storage operators $Y[e_j^n \otimes e_i^m] = X[{}^t(e_i^m \otimes e_j^n)]$. The program is given by a nested loop implementing the following sum.

$$y = L_n^{mn} x = L_n^{mn} \left(\sum_{i,j} X[{}^t(e_i^m \otimes e_j^n)](e_i^m \otimes e_j^n) \right) \\ = L_n^{mn} \left(\sum_{i,j} x_{in+j} (e_i^m \otimes e_j^n) \right)$$

$$\begin{aligned}
&= \sum_{i,j} x_{in+j} (e_j^n \otimes e_i^m) \\
&= \sum_{i,j} x_{in+j} e_{jm+i}^{mn}.
\end{aligned}$$

In the second example we derive the indexing needed to implement $y = (I_p \otimes A \otimes I_q)x$, where A is an $m \times n$ matrix. In this case the computation is described by the following equation.

$$Y[e_i^p \otimes I_m \otimes e_k^q] = AX[t(e_i^p \otimes I_n \otimes e_k^q)].$$

This equation, interpreted using the based vector with stride notation, yields

$$Y_{imq+k,q}^m = AX_{inq+k,q}^n.$$

The notation can be justified by the following basis computation.

$$\begin{aligned}
(I_p \otimes A \otimes I_q)(e_i^p \otimes e_j^n \otimes e_k^q) &= (e_i^p \otimes Ae_j^n \otimes e_k^q) \\
&= e_i^p \otimes \sum_{j'=0}^{m-1} A_{j',j} e_{j'}^m \otimes e_k^q \\
&= \sum_{j'=0}^{m-1} A_{j',j} (e_i^p \otimes e_{j'}^m \otimes e_k^q)
\end{aligned}$$

A program implementing the linear computation $y = (I_p \otimes A \otimes I_q)x$ would perform the computation $Y_{imq+k,q}^m = AX_{inq+k,q}^n$ for each value $i = 0, \dots, p-1$ and $k = 0, \dots, q-1$.

Schema 1 *The linear computation*

$$y = (I_p \otimes A^{m,n} \otimes I_q)x$$

can be implemented as follows:

```

for  $i = 0, \dots, p-1$ 
  for  $k = 0, \dots, q-1$ 
     $Y_{imq+k,q}^m \leftarrow AX_{inq+k,q}^n$ 

```

4 Programming Schemata

A programming schema for the linear computation $y = (A^{m_1, n_1} \otimes \dots \otimes A^{m_t, n_t})x$ is a template containing all of the indexing information needed to implement the computation. The schema can be executed provided programs implementing $y = A^{m_i, n_i}x$ are provided that are compatible with the indexing requirements of the schema. In this section we derive programming schemata based on the different factorizations of $A^{m_1, n_1} \otimes \dots \otimes A^{m_t, n_t}$ presented in Section 2. All of the factorizations have factors of the form $I \otimes A \otimes I$ which can be implemented using the schema in Section 3.

A tensor product factorization is evaluated by applying each factor in turn, (**for** $i = t$ **downto** 1), starting with the input vector x , using appropriate temporaries, to ultimately produce the output vector y . The indexing for each factor is determined by selecting a compatible input and output basis. Note that each factor is tensor-compatible with the next, in the sense that the output of one stage has a compatible tensor basis with the input of the next.

We begin by considering the fundamental factorization (Theorem 2). A generic element of the input basis for the i -th stage is $e_{j_i}^{N(i-1)} \otimes e_{k_i}^{n_i} \otimes e_{l_i}^{\bar{M}(i)}$ and a generic element of the output basis for the i -th stage is $e_{j_i}^{N(i-1)} \otimes e_{k'_i}^{m_i} \otimes e_{l_i}^{\bar{M}(i)}$. Therefore the code segment for the i -th stage can be denoted by

$$Y[e_{j_i}^{N(i-1)} \otimes I_{m_i} \otimes e_{l_i}^{\bar{M}(i)}] = A^{m_i, n_i} X[(e_{j_i}^{N(i-1)} \otimes I_{n_i} \otimes e_{l_i}^{\bar{M}(i)})],$$

which, using based vector with stride notation, can be written

$$Y_{j_i m_i \bar{M}(i), \bar{M}(i)}^{m_i} = A^{m_i, n_i} X_{j_i m_i \bar{M}(i), \bar{M}(i)}^{n_i}.$$

Separate subscripts for each stage are not necessary since the output basis of the i -th stage has the same form as the input basis for the $(i - 1)$ -st stage. An output basis for the i -th stage is

$$e_{j'_i}^{N(i-1)} \otimes e_{k'_i}^{m_i} \otimes e_{l'_i}^{\bar{M}(i)}$$

and an input basis for the $(i - 1)$ -st stage is

$$e_{j_{i-1}}^{N(i-2)} \otimes e_{k_{i-1}}^{n_{i-1}} \otimes e_{l_{i-1}}^{\bar{M}(i-1)}.$$

Since $\bar{N}(i-1) = n_i \bar{N}(i-2)$ we can split the element $e_{j'_i}^{\bar{N}(i-1)}$ as

$$e_{j'_i}^{\bar{N}(i-1)} = e_{\alpha}^{\bar{N}(i-2)} \otimes e_{\beta}^{n_{i-1}},$$

where $j'_i = \alpha n_{i-1} + \beta$. In this way, we can rewrite the basis of the output of i -th stage as

$$e_{\alpha}^{N(i-2)} \otimes e_{\beta}^{n_{i-1}} \otimes e_{k'_i}^{m_i} \otimes e_{l'_i}^{\bar{M}(i)}.$$

Similarly, we can rewrite the basis for the input of the $(i-1)$ -st stage as

$$e_{j_{i-1}}^{N(i-2)} \otimes e_{k_{i-1}}^{m_i} \otimes e_{\delta}^{m_i} \otimes e_{\gamma}^{\bar{M}(i)},$$

where $j_{i-1} = \delta \bar{M}(i) + \gamma$.

Composing the code sequences for the i stages from $i = t$ down to 1 we obtain a schema implementing the factorization given by Theorem 2.

Schema 2 *The linear computation*

$$y = (A^{m_1, n_1} \otimes \dots \otimes A^{m_t, n_t})x$$

can be implemented as follows:

$$\begin{aligned} &\mathbf{for} \ i = t, t-1, \dots, 1 \\ &\quad \mathbf{for} \ j = 0, \dots, N(i-1) - 1 \\ &\quad \quad \mathbf{for} \ l = 0, \dots, \bar{M}(i) - 1 \\ &\quad \quad \quad Y_{j m_i \bar{M}(i) + l, \bar{M}(i)}^{m_i} \leftarrow A^{m_i, n_i} X_{j n_i \bar{M}(i) + l, \bar{M}(i)}^{n_i} \end{aligned}$$

In the notation for Schema 2 we have obscured the fact that the output of one stage must be the input of the next stage.

The special case, where each of the A_i 's are the same, is of such importance that we state it explicitly.

Schema 3 *Let A be an $m \times n$ matrix. The linear computation*

$$y = \left(\bigotimes_{i=1}^t A \right) x$$

can be implemented as follows:

```

for  $i = t, t - 1 \dots, 1$ 
  for  $j = 0, \dots, n^{i-1} - 1$ 
    for  $l = 0, \dots, m^{t-(i+1)} - 1$ 
       $Y_{jm^{t-i+1}+l, m^{t-i}}^m \leftarrow AX_{jnm^{t-i}+l, m^{t-i}}^n$ 

```

Similar schema can be developed for other factorizations. As a second example, we derive the schema for Theorem 4. An input basis for the i -th stage of this factorization is $e_j^{\bar{M}(i)N(i-1)} \otimes e_k^{n_i}$ and an output basis for the i -th stage is $e_{k'}^{m_i} \otimes e_j^{\bar{M}(i)N(i-1)}$. Therefore the code for the i -th stage can be denoted by

$$Y[I_{m_i} \otimes e_j^{\bar{M}(i)N(i-1)}] = A^{m_i, n_i} X[(e_j^{\bar{M}(i)N(i-1)} \otimes I_{n_i})],$$

which using based vector with stride notation is equivalent to

$$Y_{j, \bar{M}(i)N(i-1)}^{m_i} = A^{m_i, n_i} X_{jn_i, 1}^{n_i}.$$

Schema 4 *The linear computation*

$$y = (A^{m_1, n_1} \otimes \dots \otimes A^{m_t, n_t})x$$

can be implemented as follows:

```

for  $i = t, t - 1 \dots, 1$ 
  for  $j = 0, \dots, \bar{M}(i)N(i-1) - 1$ 
     $Y_{j, \bar{M}(i)N(i-1)}^{m_i} \leftarrow A_i^{m_i, n_i} X_{jn_i, 1}^{n_i}$ 

```

4.1 Vector Schemata

In Section 2 we indicated that the tensor product operation $A \otimes I_n$ could be interpreted as a vector operation on vector of length n . In this section we show how to include this vector interpretation in our programming schemata. The key idea is to apply the linear computation $y = Ax$ to appropriately accessed vectors.

Consider the indexing notation $Y[e_i^p \otimes I_m \otimes e_k^q] = AX[e_i^p \otimes I_n \otimes e_k^q]$, used to describe the linear computation $y = (I_p \otimes A \otimes I_q)x$, where A is an $m \times n$ matrix. We can view this code applied to vectors of length q by combining the q copies of the computation $e_i^p \otimes Ae_j^m \otimes e_k^q$ for $k = 0, \dots, q-1$ into a single vector operation. The subvector accessed by $X[e_i^p \otimes e_j^n \otimes I_q] = X_{inq+jq,1}^q$ and the subvector needed for the vector computation $A \otimes I_q$ is $X[e_i^p \otimes I_n \otimes e_k^q]_{k=0}^{q-1} = X[e_i^p \otimes I_n \otimes I_q] = X_{inq,q,1}^{n,q}$. Therefore we use the notation

$$Y[e_i^p \otimes I_m \otimes I_q] = (A \otimes I_n)X[e_i^p \otimes I_n \otimes I_q],$$

which stated using based vector with stride notation is

$$Y_{imq,q,1}^{m,q} = (A \otimes I_n)X_{inq,q,1}^{n,q}.$$

Schema 5 *The linear computation*

$$y = (I_p \otimes A^{m,n} \otimes I_q)x$$

can be implemented using the following vector schema:

$$\begin{aligned} &\mathbf{for} \ i = 0, \dots, p-1 \\ &\quad Y_{imq,q,1}^{m,q} \leftarrow (A \otimes I_q)X_{inq,q,1}^{n,q} \end{aligned}$$

Using this idea, the Schemata 2 and 4 can be converted into vector schemata.

Schema 6 *The linear computation*

$$y = (A^{m_1, n_1} \otimes \dots \otimes A^{m_t, n_t})x$$

can be implemented as follows:

$$\begin{aligned} &\mathbf{for} \ i = t, t-1, \dots, 1 \\ &\quad \mathbf{for} \ j = 0, \dots, N(i-1) - 1 \\ &\quad \quad Y_{jm_i \bar{M}(i), \bar{M}(i), 1}^{m_i, \bar{M}(i)} \leftarrow (A^{m_i, n_i} \otimes I_{\bar{M}(i)})X_{jn_i \bar{M}(i), \bar{M}(i), 1}^{n_i, \bar{M}(i)} \end{aligned}$$

Schema 7 *The linear computation*

$$y = (A_1^{m_1, n_1} \otimes \dots \otimes A_t^{m_t, n_t})x$$

can be implemented as follows:

$$\begin{aligned} & \mathbf{for} \ i = t, t-1, \dots, 1 \\ & \quad Y_{0, \bar{M}(i)N(i-1), 1}^{m_i, \bar{M}(i)N(i-1)} \leftarrow (A_i^{m_i, n_i} \otimes I_{\bar{M}(i)N(i-1)}) X_{0, 1, n_i}^{n_i, \bar{M}(i)N(i-1)} \end{aligned}$$

5 Computer Implementation

Returning to our original goal of combining code segments of the form $y = A_i x$ into a program implementing the linear computation

$$y = \left(\bigotimes_{i=1}^t A_i \right) x,$$

we now show how the code segment $y = A_i x$ must be parameterized to be used in the schemata derived in Section 3 and Section 4.

For example, let's consider the vector Schema 5 in Section 4. Let A be an $m \times n$ matrix and

$$y = (I_p \otimes A \otimes I_q)x$$

coded as

$$\begin{aligned} & \mathbf{for} \ i = 0, \dots, p-1 \\ & \quad y = (A \otimes I_q)x \\ & \quad \equiv \\ & \quad \mathbf{for} \ i = 0, \dots, p-1 \\ & \quad \quad y_{imq, q, 1}^{m, q} \leftarrow (A \otimes I_q)x_{inq, q, 1}^{n, q} \end{aligned}$$

Where for each i we need new vectors of length q at base inq on input and imq on output. The general set of parameters needed to implement $(A \otimes I_q)$

in this context is

```

A(m, n, y, a, u, s, x, b, v, t, d) ≡
    for α = 0, ..., m - 1
        ya+αu,sd ← 0
        for β = 0, ..., n - 1
            ya+αu,sd ← ya+αu,sd + Aα,βxb+βv,td

```

where m and n are the output and input dimensions of the matrix A and y, a, u, s, d describes the vector notation

$$y_{a,u,s}^d$$

a vector y of length d , with base a , base stride u , and stride s . This code is just a matrix-vector product with several additional parameters needed to be used in our vector schemata. We sketch a FORTRAN fragment, showing how this might be coded. We use the `complex` versions of the BLAS routines [4, 1] `copy` and `saxpy` to emphasize the embedded vector instructions.

```

do ia = 0,m-1
    call ccopy(d,(0.0,0.0),0,y(a+ia*u),s)
    do ib = 0,n-1
        call csaxpy(d,aa(ia,ib),x(b+ib*v),t,y(a+ia*u),s)
    end do
end do

```

Where `ia` is α , `ib` is β , and `aa` is the matrix of coefficients corresponding to A .

6 Applications

In this section we will derive a program to compute the t -dimensional Fourier transform. This problem arises in programming Fourier transforms that respect crystallographic symmetry [2] on the CRAY Y-MP. The t -dimensional Fourier transform can be written

$$y = F_{n_1} \otimes \cdots \otimes F_{n_t}$$

where F_n is the n -point Fourier transform. Therefore, we can use the schemata developed in this paper to implement this computation. Since in the application on the CRAY Y-MP we have vector instructions available, we shall use the vector Schema 7.

In the case of the t -dimensional Fourier transform all the matrices are square, so that $m_i = n_i$ and the index computations are greatly simplified. We first show an example implementation in FORTRAN of the Fourier transform with sufficient parameters to be used in our schema. This follows the example fragment in Section 5. It computes the Fourier transform directly from the definition and is not meant to be efficient, but simply to show the use of the parameters. Also, we have reversed the positions of the input and output vectors to conform with the BLAS and CRAY Library conventions.

```

        subroutine fnv(n,
&           x, bx, inc1x, inc2x,
&           y, by, inc1y, inc2y,
&           d
&           )
*
*   computes the Fourier Transform  $y = F_n(x)$ 
*   from the definition.
*
*   includes sufficient parameters to be used in
*   our vector schema.
*
        integer n
*
        complex x(0:*)
        integer bx          ! base
        integer inc1x      ! base stride
        integer inc2x      ! vector stride
*
        complex y(0:*)
        integer by          ! base
        integer inc1y      ! base stride
        integer inc2y      ! vector stride
*

```

```

*
*****

      complex iota
      parameter(iota = (0.0,1.0))
      real pi
      complex w
      complex ca

      integer basex, basey
      integer iy,ix

      pi = atan2(0.0, -1.0)
      w = exp((2*pi*iota/n))

      do iy = 0, n-1
         basey = by + iy*inc1y
         call ccopy(d,(0.0,0.0),0,y(basey),inc2y) !zero y
         do ix = 0, n-1
            ca = w**(ix*iy)
            basex = bx + ix*inc1x
            call caxpy(d,ca,x(basex),inc2x,y(basey),inc2y)
         end do ! ix
      end do ! iy

      return
      end ! fnv

```

This subroutine can now be used to implement schema 7. Here is a FORTRAN fragment showing this implementation. (Letting $nn = N$.)

```

      call ccopy(nn,x,1,t,1)
      do i = t,1,-1
         call fnv(n(i),t,0,1,n(i),y,0,nn/n(i),1,nn/n(i))
         call ccopy(nn,t,1,y,1)
      end do

```

In the scientific libraries that Cray Research distributes with UNICOS 6.1, there is a Fast Fourier Transform routine, `MCFFT`, that can be used to efficiently implement the arbitrary tensor product of Fourier Transforms. With the machinery we have developed in the first part of this paper, we will show how to compute the appropriate calls to `MCFFT`.

We first abstract the *man* page [1] for the routine. The library routine `MCFFT` applies a multiple multitasked complex Fast Fourier Transform. Its synopsis is

```
CALL MCFFT(isign, n, m, scale,
           x, inc1x, inc2x, y, inc1y, inc2y,
           table, ntable, work, nwork)
```

`MCFFT` computes the Fourier Transform of each column of the complex matrix `x`, and stores the results in the columns of the matrix `y`. If `x` and `y` were dimensioned as:

```
complex x(0:n-1, m), y(0:n-1, m)
```

then `MCFFT` computes F_n for each of the `m` columns of `x` and stores the result in the corresponding columns of `y`. The routine is vectorized along the columns and therefore, in our terminology, is an implementation of $F_n \otimes I_m$. Furthermore, if `n` is sufficiently large, `MCFFT` will multitask.

The important parameters for our schema are `x` and `y` together with their increment parameters `inc1x`, `inc2x`, `inc1y`, and `inc2y`. The first increment `inc1x` is the `x` increment in the first dimension. That is the distance between successive complex array elements within each column of the input array `x`. The second increment parameter, `inc2x`, is the `x` increment in the second dimension. That is the distance between successive complex array elements within each row of the input array `x`. And similarly for the array `y`. Suitably interpreted, these parameters can be used to implement our vector Schema 7.

For the sake of completeness we sketch the meaning of the other parameters: `table` is a real array of size `ntable` which is used to hold the initialized trigonometric constants between calls to Fourier Transforms of the same size `n`. The real array `work` of size `nwork` is a temporary working array supplied by the caller. The integer `isign` is a switch: if 0, then simply initialize `table`, if 1, compute the forward transform, and if -1, compute the inverse transform. The real `scale` is used to scale the output `y` so that the true inverse can be obtained.

With this introduction then the following call to MCFFT, corresponds to our call to `fnv` in the program fragment above.

```
call mcfft(1,n(i),nn/n(i),1.0,  
&          t,1,n(i),y,nn/n(i),1,  
&          table, ntable, work, nwork)
```

Acknowledgements

We would like to thank several people at Cray Research, Inc: John Mertz, a collaborator of the first author in developing crystallographic programs, for many conversations, ideas, and access to the Cray Research's computer facilities; (It was these discussions that lead to the actual implementation problem in Section 6.) Mike Merchant for help with the library FFT's; and Mike Heroux, for help with the BLAS.

References

- [1] *Volume 3: UNICOS Math and Scientific Library Reference Manual*. Cray Research, Inc., Mendota Heights, MN 55120, 1991. SR-2081.
- [2] L. Auslander, R. W. Johnson, and M. Vulis. Evaluating Finite Fourier Transforms that respect group symmetries. *Acta Cryst.*, A44:467–478, 1988.
- [3] L. Auslander and M. Shenefelt. Fourier transforms that respect crystallographic symmetries. *IBM J. Res. Develop.*, 31(2):213–223, March 1987.
- [4] Thomas F. Coleman and Charles Van Loan. *Handbook for Matrix Computations*. *Frontiers in Applied Mathematics*, Society for Industrial and Applied Mathematics, Philadelphia, 1988. Vol. 4.
- [5] Roger A. Horn and Charles R. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, Cambridge, 1991.

- [6] J. R. Johnson, C.-H. Huang, and R. W. Johnson. Tensor permutations and block matrix allocation. 1992. Preprint.
- [7] J. R. Johnson, R. W. Johnson, D. Rodriguez, and R. Tolimieri. A methodology for designing, modifying, and implementing Fourier Transform algorithms on various architectures. *Circuits Systems Signal Process.*, 9(4):449–500, 1990.
- [8] R. W. Johnson, C.-H. Huang, and J. R. Johnson. Multilinear algebra and parallel programming. *The Journal of Supercomputing*, 5:189–217, 1991. A Condensed version of this paper appeared previously in the *Proceedings of Supercomputing '90*.
- [9] R. Tolimieri, M. An, and C. Lu. *Algorithms for the Discrete Fourier Transform and Convolution*. Springer-Verlag, New York, 1989.
- [10] Charles Van Loan. *Computational Frameworks for the Fast Fourier Transform*. *Frontiers in Applied Mathematics*, Society for Industrial and Applied Mathematics, Philadelphia, 1992. Vol. 10.