Optimization of Symmetric Tensor Computations

Jonathon Cai
Department of Computer Science, Yale University
New Haven, CT 06520
Email: jonathon.cai@yale.edu

Muthu Baskaran, Benoît Meister, Richard Lethin
Reservoir Labs Inc.
New York, NY 10012
Email: {baskaran,meister,lethin}@reservoir.com

Abstract—For applications that deal with large amounts of high dimensional multi-aspect data, it is natural to represent such data as tensors or multi-way arrays. Tensor computations, such as tensor decompositions, are increasingly being used to extract and explain properties of such data. An important class of tensors is the symmetric tensor, which shows up in real-world applications such as signal processing, biomedical engineering, and data analysis.

In this work, we describe novel optimizations that exploit the symmetry in tensors in order to reduce redundancy in computations and storage and effectively parallelize operations involving symmetric tensors. Specifically, we apply our optimizations on the matricized tensor times Khatri Rao product (mttkrp) operation, a key operation in tensor decomposition algorithms such as INDSCAL (individual differences in scaling) for symmetric tensors. We demonstrate improved performance for both sequential and parallel execution using our techniques on various synthetic and real data sets.

I. INTRODUCTION

Real-world data are often multi-dimensional, with multiple aspects. Tensors or multi-dimensional arrays are a natural fit to represent data with multiple attributes and dimensionality. Tensor analysis (through decomposition or factorization) using algorithms from multi-linear algebra, a generalization of linear algebra to higher dimensions, is increasingly being used to extract and explain properties of multi-attribute data. The application areas of tensor decompositions include signal processing, data mining, computer vision, numerical linear algebra, numerical analysis, and graph analysis [1].

Symmetric tensors form an important class of tensors. The list of areas where symmetric tensor decomposition is applied includes (but is not limited to) speech, mobile communications, machine learning, factor analysis of k-way arrays, biomedical engineering, psychometrics, and chemometrics [2]. In addition, the decomposition of symmetric tensors plays an important role in independent component analysis (ICA) [3] and also in the process of blind identification of underdetermined mixtures (UDM), linear mixtures with more inputs than observable outputs.

Algorithms have been derived specifically for symmetric tensors, reducing computational cost and sometimes improving convergence [4], [5], [6]. We focus on optimizing a key operation in factorizing symmetric tensors, the matricized tensor times Khatri Rao product (mttkrp) operation. To our knowledge, there have been no papers describing optimizations to the mttkrp operation that exploit symmetry of the symmetric tensors.

In this work, we implement a lexicographical storage format, providing storage benefits and efficiency for performing symmetric tensor computations. In addition, we propose novel optimizations that exploit symmetry in order to reduce redundant computations in the mttkrp operation. Furthermore, our optimizations lead to effective parallelization of the mttkrp operation applied over symmetric tensors. We demonstrate performance improvements of our implementation in an INDSCAL (individual differences in scaling) decomposition algorithm that is used for decomposing symmetric tensors. We demonstrate improved performance for both sequential and parallel execution using our optimizations on various synthetic and real data sets.

II. BACKGROUND

In this section, we give some background information that will help with understanding our techniques and experimental study.

a) General Tensor Background: A tensor is a multi-dimensional array and the order of a tensor is the number of dimensions, also called modes, of the tensor. In particular, a cubical tensor, a tensor with every mode of the same size, is supersymmetric if its elements remain constant under any permutation of the indices. For instance, a three-way tensor \( X \in \mathbb{R}^{I \times J \times I} \) is supersymmetric if \( x_{ijk} = x_{ikj} = x_{jki} = x_{kij} = x_{kj i}, \forall i, j, k = 1, \ldots, I \). Tensors can be partially symmetric in two or more modes as well. For example, a three-way tensor \( X \in \mathbb{R}^{I \times J \times K} \) is symmetric in modes one and two if all its frontal slices are symmetric, i.e., \( x_{ijk} = x_{jki}, \forall i, j = 1, \ldots, I, \forall k = 1, \ldots, K \).

The fibers of a tensor result from holding all but one index constant. For a matrix, they are the rows and columns. In a third-order tensor, the fibers are tubes. Using MATLAB notation, three possible fibers are \( \mathcal{X}(i, j, :) \), \( \mathcal{X}(i, :, k) \), and \( \mathcal{X}(:, j, k) \). For a mode-\( n \) fiber, \( n \) refers to the index that is not held constant.

b) Tensor Computations: A tensor can be matricized, or unfolded, into a matrix along any of its modes. In the mode-\( n \) matricization, the mode-\( n \) fibers are used to form the columns of the resulting matrix. The mode-\( n \) unfolding of \( \mathcal{X} \) is denoted by \( X_{(n)} \).

Figure 1 shows an example of an unfolded tensor.

There are several matrix products that are used in tensor decomposition methods. One of the matrix products that is used in the methods discussed in this work is the Khatri-Rao
and Chang claim that when the CP algorithm finally converges, the first two factor matrices will be equivalent up to scalar multiplication [7]. Ten Berge et al. show that for a contrived set of matrices, nonequivalence for the first two factor matrices can arise [9]. For all practical purposes, however, the CP decomposition guarantees convergence [10].

The best method for computing INDSCAL is still an open question [11]. In this work, we ensure that the first two factor matrices are equal for each iteration of the algorithm, effectively calculating a solution for only one of them. The motivation for this is the claim that in practice [10], the first two factor matrices converge to equivalence.

e) CP-OPT Methods: Acar et. al [12] have developed gradient-based optimization methods (called CP-OPT) for computing the CP decomposition. The methods are motivated by observing the drawbacks of two popular CP decomposition approaches: the alternating least squares (ALS) approach and the nonlinear least squares (NLS) formulation. The ALS approach is fast, but often fails to find the underlying structure of the data, especially in the case of overfactoring. Compared with ALS, the NLS algorithm is superior in terms of finding the underlying structure, but it is significantly slower. The memory and time overhead associated with NLS makes the NLS algorithm intractable for large data sets. Acar et al. find that gradient-based optimization methods are more accurate than ALS and faster than NLS in terms of computation time. They implement two different gradient-based methods (with similar performance): nonlinear conjugate gradient (NCG) method and limited memory quasi-Newton method using BFGS updates (L-BFGS).

f) Matricized Tensor Times Khatri-Rao Product: The matricized tensor times Khatri Rao product (mttkrp) operation with respect to a tensor Z and a CP decomposition with factor matrices \( A^{(1)}, \ldots, A^{(N)} \) is defined as

\[
Z(\mathbf{n})(A^{(N)} \circ \ldots \circ A^{(n+1)} \circ A^{(n-1)} \circ \ldots \circ A^{(1)}),
\]

where \( n \) refers to the factor matrix being updated. The mttkrp operation is a common operation used in CP decomposition algorithms, such as CP-ALS and CP-OPT.

III. SYMMETRIC TENSOR STORAGE FORMAT

Before discussing our storage format for symmetric tensors, we discuss some of the existing works in literature (e.g. [13], [14]) that define storage for symmetric tensors. Schatz et al. [13] design Block Compact Symmetric Storage, building on the work in \texttt{libflame} [15]. Ballard et al. [14] introduce a lexicographical storage format, storing only unique values of a supersymmetric tensor. The storage format introduced by Ballard et al. is natural, and we extend it to the more general case of partially symmetric tensors.

We describe our storage format for partially symmetric tensors of order three, but the format generalizes to tensors of higher order and for supersymmetric tensors. For further discussion, let us consider a tensor \( Z \in \mathbb{R}^{l_x \times l_y \times l_z} \) that is symmetric in the first two modes. Since the tensor is symmetric, there are values that are repeated and need not be stored...
be stored redundantly. We define a tensor index as an array of three indices corresponding to one entry of the tensor, and we define an index class as a set of tensor indices such that the corresponding tensor entries all share a value due to symmetry. For example, in \( \mathbb{Z} \), the tensor indices \([1, 2, 1]\) and \([2, 1, 1]\) are in the same index class since \( \mathbb{Z}_{211} = \mathbb{Z}_{121} \). We choose a unique representative of an index class by selecting the tensor index whose indices are in non-decreasing order in the first two modes.

Assuming \( \mathbb{Z} \) is dense, we choose an underlying ordering on the unique entries to avoid storing any index information. We use lexicographic order of the index classes, increasing with respect to the index representation, placing the elements in a linear array. That is, the index class with index representation \([i_1, i_2, \ldots, i_m]\) is listed before \([j_1, j_2, \ldots, j_m]\) if \( i_1 < j_1 \) or if \( i_1 = j_1 \) and \( i_2 < j_2 \), and so on.

For a tensor of dimensions \( I_n \times I_n \times I_k \), the total storage is reduced to \( \frac{I_n \times (I_n + 1)}{2} \times I_k \). Geometrically, the format represents a stack of upper triangular slices. To get to the next \( Z \) metric tensors via a modification of CP-OPT method. For the least squares fit function \( A + \) takes its gradient function as

\[
\frac{\partial f}{\partial A(n)} = -Z(n)(A(N) \odot \ldots \odot A^{(n+1)} \odot A^{(n-1)} \odot \ldots \odot A(1)) + A^{(n)} \Gamma^{(n)} \text{, where } \Gamma^{(n)} = y^{(1)} \ast \ldots \ast y^{(n-1)} \ast y^{(n+1)} \ast \ldots \ast y^{(N)} \text{ and } y^{(n)} = A^{(n)^T} A(n). \]

The formula for the gradient is derived in [12]. It is to be noted that the fit function is the same function that is minimized for the CP-ALS algorithm. Further, the mttkrp operation is the dominant operation for IND-OPT because it is the most expensive operation in the gradient, which is calculated numerous times.

V. OPTIMIZATION AND PARALLELIZATION OF \( \text{mttkrp} \) OPERATION FOR SYMMETRIC TENSORS

In this section, we discuss our techniques to optimize and parallelize the \( \text{mttkrp} \) operation, namely

\[
Z(n)(A(N) \odot \ldots \odot A^{(n+1)} \odot A^{(n-1)} \odot \ldots \odot A(1)),
\]

for partially symmetric tensors. In order to compute the \( \text{mttkrp} \) it is necessary to multiply and add a combination of factor matrix elements and tensor values. As long as the final result is the same as the formal definition of the \( \text{mttkrp} \), we can change the order of the operations to exploit symmetry and redundancies.

Our optimizations are based on the following two important ideas.

1) We compute intermediate results on-the-fly instead of storing them in a large matrix.

2) Instead of computing the Khatri-Rao product in full and then multiplying it by the matricized tensor, we fuse the computations and compute partial products that represent a combination of those two operations.

These two key ideas enable minimal intermediate storage and effective parallelization of the \( \text{mttkrp} \) operation.

For ease of understanding, we present our techniques with respect to a partially symmetric tensor of order three, symmetric in the first two modes. However the techniques are generalizable to higher order partially symmetric and supersymmetric tensors. For a partially symmetric tensor \( Z \in \mathbb{R}^{I_n \times I_n \times I_k} \), with the factor matrices \( A \in \mathbb{R}^{I_n \times R} \) and \( B \in \mathbb{R}^{I_n \times R} \), the gradient function in IND-OPT method leads to the calculation of the \( \text{mttkrp} \) operation in two cases: \( Z(3)(A \odot A) \) and, without loss of generality, \( Z(2)(B \odot A) \). Note that we need not compute \( Z(1)(B \odot A) \) since it is equivalent to \( Z(2)(B \odot A) \). In the following discussion, we refer to the Khatri-Rao product as KRP.

A. First Case: \( \text{mttkrp} \) in non-symmetric mode

First, we consider the computation of \( Z(3)(A \odot A) \). We call this the “non-symmetric” case, since the unfolded mode is non-symmetric.

The computation can be viewed as an unfolded tensor, \( Z(3) \), multiplied with a KRP, \( A \odot A \). For brevity, let \( A \odot A \) be \( K \). The naive way to compute \( K \) is to compute it in full, storing it in a matrix of size \( (I_n)^2 \times R \). As noted in [18], computing the Khatri-Rao product in full leads to “intermediate data explosion problem” since \( K \) is large and dense. The following optimizations allow us to avoid the intermediate blow-up problem.

We index the rows of \( K \) by pairs \((i, j)\), where \( i = 1, \ldots, I_n \) and \( j = 1, \ldots, I_n \). By definition of the Khatri-Rao product, the \((i, j)\)-th row is formed by multiplying component-wise the \( i \)-th and \( j \)-th rows of matrix \( A \). Note that because we are computing the Khatri-Rao product of identical matrices, we only need to compute the rows of \( K \) corresponding to upper triangular pairs. For example, the row in \( K \) corresponding to the pair \((2, 3)\) consists of component-wise multiplication of the second row of \( A \) by the third row of \( A \), and this is equivalent to the row in \( K \) corresponding to the pair \((3, 2)\). Hence, we can store the KRP in a matrix of size \( \frac{I_n \times (I_n + 1)}{2} \times R \).

Consider a row in \( Z(3) \). Let the first-pair tuple refer to the first two members of the index of a tensor element. A first-pair tuple \((a, b)\) is permutable if \( a \) is not equal to \( b \). Due to symmetry in the first two modes, the values that correspond to symmetric first-pair tuples, with the same third index, are equal. Furthermore, these values are in the same row of \( Z(3) \). Hence, both of these elements send their contributions, when
multiplying by the appropriate rows of the Khatri-Rao product, to the same row in the final result matrix. Now consider the KRP rows with which these elements are multiplied, in the outer product fashion, to compute the final result. In the case of elements with indices of $[1, 2, 3]$ and $[2, 1, 3]$, they will be multiplied by the rows of $K$ corresponding to tuples $(1, 2)$ and $(2, 1)$, respectively. Recall that these rows are equivalent.

If the first-pair tuple is permutable, we only need to do one of these row multiplications and multiply the result by two, to compute the total contributions from both elements on the final result matrix. For first-pair tuples that are not permutable, we do not multiply the result by two.

We further optimize this operation by computing rows of the KRP on-the-fly, in order to reduce the intermediate storage. We compute each row of the KRP, of size $R$, and find the appropriate values in the tensor to compute the final result. The algorithm is parallelizable because each row, and in fact each element, of the final result matrix can be independently computed, making the algorithm parallelizable along the loop that iterates over the third mode.

We implement the above optimizations in Algorithm 1.

Algorithm 1 Non-Symmetric mttkrp

1: Initialize the result matrix $S$ of size $I_k \times R$, setting all values to 0.
2: for upper triangular 2-tuples with maximum size $(I_n, I_n)$, letting the tuple be $(a, b)$ do
3: Compute the row of the KRP corresponding to $(a, b)$. Let this row be denoted by $h$.
4: for $i = 1, \ldots, I_k$ do
5: Let the tensor value corresponding to tuple $(a, b, i)$ be $v$.
6: for $r = 1, \ldots, R$ do
7: if $a \neq b$ then
8: $S(i, r) += 2vh(r)$
9: else
10: $S(i, r) += vh(r)$
11: end if
12: end for
13: end for
14: end for

B. Second Case: mttkrp in symmetric mode

Second, we consider the computation of $Z_{(2)}(B \odot A)$. We call this the “symmetric” case, since the unfolded mode is symmetric.

We apply techniques that are different from the non-symmetric case. We first make the observation that tensor elements with a common third mode are multiplied by the same elements in $B$. This redundancy can be exploited.

As before, the computation can be viewed as an unfolded tensor, $Z_{(2)}$, multiplied with a Khatri-Rao product, $B \odot A$. For brevity, let $B \odot A$ be $K$. Instead of computing $K$ on its own, we compute an intermediate that represents a fused version of the both the matrix multiplication operations, between $Z_{(2)}$ and $K$, and the operations needed to compute the KRP. We describe three schemes for storing this intermediate, with sizes of $(I_n)^2 \times R$, $I_n \times (I_n + 1) \times R$, and $R$, respectively.

The first scheme computes a matrix $I$ of size $(I_n)^2 \times R$, where each row of the matrix is indexed by a triplet $(i, j, m)$, where $(i, j)$ are upper triangular over $I_n$ and $m \in \{i, j\}$. The $r$-th element in the $(i, j, m)$-th row of the intermediate represents the partial product $\sum_k Z_{ijk}B_{(k, r)}A(r, r)$ where $k \in 1, \ldots, I_k$. When computing the final result matrix, we compute the $q$-th row by adding the relevant rows of $I$, namely the rows with $m = q$. Since the rows of the result matrix can be computed independently, we can parallelize over a loop of $I_n$.

The second scheme computes a matrix $I$ of size $I_n \times (I_n + 1) \times R$, where each row of the matrix is indexed by a pair $(i, j)$, where $(i, j)$ are upper triangular over $I_n$. The $r$-th element in the $(i, j)$-th row of the intermediate represents the partial product $\sum_k Z_{ijk}B_{(k, r)}$, where $k \in 1, \ldots, I_k$. When computing the final result matrix, we compute the $q$-th row by adding the relevant rows of $I$, namely the rows with $i = q$ or $j = q$. Depending on if $i = q$ or $j = q$, we multiply the corresponding row of $I$ component-wise by $A(i, :)$ or $A(j, :)$, respectively. Since the rows of the result matrix can be computed independently, we can parallelize over a loop of $I_n$.

The third scheme computes an intermediate vector of size $R$, by computing each row of $I$ in the second scheme on-the-fly and then updating the corresponding rows in the result matrix. For brevity, only the third scheme is shown in Algorithm 2. If $i \neq j$, two rows in the result matrix are updated; if $i = j$, then one row in the result is updated. The parallelism in this scheme is limited due to the dependence introduced by the loop involving the permutations. Consequently, for parallel tests, we ran experiments with the second scheme. For sequential tests, we ran experiments with the third scheme.

VI. Experiments

We carried out experiments to evaluate the following: (1) efficiency of a single optimized mttkrp operation and (2) efficiency of the IND-OPT algorithm for the sequential and parallel implementations. We performed the experiments on a quad socket 8-core system with Intel Xeon E5-4620 2.2 GHz processors (Intel Sandy Bridge microarchitecture chips) which supports 32 concurrent threads. Each of the 8 cores in a socket has a 32 KB private L1 instruction cache, a 32 KB private L1 data cache, and a 256 KB private L2 cache. Each core also shares a 16 MB L3 cache. The system has 128 GB of DRAM.

A. Testing a single mttkrp operation

We construct a representative problem, synthetically generating a $120 \times 120 \times 50$ partially symmetric tensor. The sequential performance achieved is shown in Table I. From the results in Table I, we observe an approximate 30x speed up
Algorithm 2 Symmetric \texttt{mttkrp}

1: Initialize the result matrix $S$ of size $I_n \times R$, setting all values to 0.
2: for upper triangular 2-tuples with maximum size of $(I_n, I_n)$, let the tuple be $(a, b)$
3: Initialize a row vector of the intermediate, $m$, of size $R$, setting all values to 0.
4: for $i = 1, \ldots, I_p$ do
5: Let the tensor value corresponding to tuple $(a, b, i)$ be $v$.
6: for $r = 1, \ldots, R$ do
7: \hspace{1em} $m(r) \leftarrow v * B(i, r)$
8: end for
9: end for
10: for permutations of $(a, b)$, let the permutation be $(x, y)$
11: for $r = 1, \ldots, R$ do
12: \hspace{1em} $S(y, r) \leftarrow A(x, r) m(r)$
13: end for
14: end for
15: end for

using the optimized \texttt{mttkrp} as opposed to the un-optimized version of the \texttt{mttkrp} operation that does not exploit the tensor’s symmetry.

<table>
<thead>
<tr>
<th>Version</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-symmetric, optimized</td>
<td>0.0052</td>
</tr>
<tr>
<td>Non-symmetric, dense</td>
<td>0.1828</td>
</tr>
<tr>
<td>Symmetric, optimized</td>
<td>0.0000</td>
</tr>
<tr>
<td>Symmetric, dense</td>
<td>0.0183</td>
</tr>
</tbody>
</table>

**TABLE I**

PERFORMANCE OF \texttt{mttkrp} KERNEL

B. Testing the IND-OPT algorithm

1) Data Sets: The IND-OPT algorithm was run on five data sets - one real and four synthetic data sets. The synthetic tensors were generated using the MATLAB Tensor Toolbox [19]. The synthetic data sets used in our experiments (along with their size and decomposition rank used for the data sets in the experiments) are listed in Table II.

<table>
<thead>
<tr>
<th>Data set</th>
<th>Size</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syn-120</td>
<td>$120 \times 120 \times 50$</td>
<td>6</td>
</tr>
<tr>
<td>Syn-1000</td>
<td>$1000 \times 1000 \times 10$</td>
<td>10</td>
</tr>
<tr>
<td>Syn-500</td>
<td>$500 \times 500 \times 300$</td>
<td>6</td>
</tr>
<tr>
<td>Syn-200</td>
<td>$200 \times 200 \times 1000$</td>
<td>8</td>
</tr>
</tbody>
</table>

**TABLE II**

SYNTHETIC DATA SETS IN OUR EXPERIMENTS

Initial guesses for the synthetic partially symmetric tensors were supplied by the create_guess function in [19] with a perturbation parameter of 1E-1.

The real data set is a modified version of a Facebook data set from [20]. We call our modified data set facebook-m. In the original Facebook dataset, information about various social networking activities, such as number of wall posts and messages, was collected over 1591 days in a large 63891 \times 63891 array. We select a subset of 500 users and 300 days for our data set, facebook-m, which has dimensions $500 \times 500 \times 300$ and decompose over a rank of 6. For the real tensor, initial guesses were generated randomly.

2) Parameters: The memory parameter for the IND-OPT algorithm was set to 5. For the L-BFGS component of IND-OPT, the maximum number of iterations was set to 10,000, the maximum number of gradient evaluations was set to 100,000, the relative function tolerance was set to 1E-16, and the stop tolerance was set to 1E-12.

3) Sequential Results: The sequential performance achieved is shown in Table IV. The INDSCAL time represents the time it takes for an un-optimized INDSCAL algorithm to run. The un-optimized INDSCAL uses a version of the \texttt{mttkrp} operation that does not exploit the tensor’s symmetry.

The \texttt{mttkrp} operation dominates the running time of the algorithm. Table III shows the percentage of time spent in the \texttt{mttkrp} kernel during IND-OPT for our five data-sets, for the sequential implementation.

From the results in Table IV, we observe speed-ups using our optimized \texttt{mttkrp} and IND-OPT. For the different data sets, we observe a range of different speed-ups (from 21x for Syn-120 data set, up to 39x for Syn-200 data set).

<table>
<thead>
<tr>
<th>Data Set</th>
<th>\texttt{mttkrp} % Time</th>
<th>Remaining % Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syn-120</td>
<td>99.45</td>
<td>0.55</td>
</tr>
<tr>
<td>Syn-1000</td>
<td>99.95</td>
<td>0.07</td>
</tr>
<tr>
<td>Syn-500</td>
<td>99.99</td>
<td>0.01</td>
</tr>
<tr>
<td>Syn-200</td>
<td>99.95</td>
<td>0.07</td>
</tr>
<tr>
<td>facebook-m</td>
<td>99.99</td>
<td>0.01</td>
</tr>
</tbody>
</table>

**TABLE III**

PERCENTAGE OF TIME SPENT IN \texttt{mttkrp} KERNEL

4) Parallel Results: Figure 2 shows results achieved for the parallel implementation of the \texttt{mttkrp} kernel in the IND-OPT algorithm on the Syn-500 data set. The number of threads was varied from 1 to 16 in powers of 2. Though we observe a speed-up with increasing number of threads, the performance scaling is sub-linear (1.96x for 2 threads vs 4.64x for 16 threads for the non-symmetric \texttt{mttkrp} kernel, and 1.31x for 2 threads vs 7.49x for 16 threads for the symmetric \texttt{mttkrp} kernel) due to increased NUMA memory conflicts.

Figure 3 shows results achieved for the parallel implementation of the \texttt{mttkrp} kernel in the IND-OPT algorithm on the
we do not compute the full Khatri-Rao product explicitly, but rather in rows or in fused computations. The work described in [23] specifies novel tensor storage formats for storing sparse and semi-sparse tensors. However, our symmetric tensor format is specifically tuned for supersymmetric and partially symmetric tensors.

VIII. CONCLUSIONS AND FUTURE WORK

We have described novel optimizations for improving the performance of computations involving symmetric tensors. Specifically, we have optimized a key operation in tensor decompositions, namely, the \texttt{mttkrp} operation, and implemented our optimizations in the IND-OPT algorithm, both for sequential and parallel execution, with respect to partially symmetric tensors. We have demonstrated the efficiency of our optimizations using synthetic and real data sets.

While these optimizations were implemented by hand to provide immediate benefit in implementations of algorithms such as IND-OPT, we expect their major impact will be obtained when they are implemented automatically in the context of automatic high-level optimizations, such as those that extract parallelism, improve locality, and perform other format improvements (for e.g. automatically implementing the optimizations through R-Stream compiler [24]).

REFERENCES


