Matrix Factorizations for Parallel Integer Transforms *

Yiyuan She[1,2] Pengwei Hao[1,2] Yakup Paker[2]
[1]*Center for Information Science, Peking University, Beijing, 100871, China*
[2]*Department of Computer Science, Queen Mary, University of London, E1 4NS, UK*
E-mail: {yyshe, phao, paker}@dcs.qmul.ac.uk

Abstract

Integer mapping is critical for lossless source coding and the techniques have been used for image compression in the new international image compression standard, JPEG 2000. In this paper, from block factorizations for any nonsingular transform matrix, we introduce two types of parallel elementary reversible matrix (PERM) factorizations which are helpful for the parallelization of perfectly reversible integer transforms. With improved degree of parallelism (DOP) and parallel performance, the cost of multiplication and addition can be respectively reduced to O(*logN) and* O(*log*^{2}*N) for an N-by-N transform matrix. These make PERM factorizations an effective means of developing parallel integer transforms for large matrices. Besides, we also present a scheme to block the matrix and allocate the load of processors for efficient transformation.*

1. Introduction

Due to the limitation of computational precision and storage capacity, transforms used in data compression should be integer reversible. Integer transform (or integer mapping) is such a type of transform that maps integers to integers with perfect reconstruction (PR). People had long before explored in this area, and their early work, such as S transform [1], TS transform [2] and S+P transform [3], suggests a promising future of reversible integer mapping in image compression, region-of-interest (ROI) coding, and unified lossy/lossless compression systems. However, not until lifting scheme (LS) [4] was proposed for constructing the second generation wavelets did people try to break away from various specific transforms and roundings and to build generic integer wavelet transforms [6] based on the simplified ladder structure [5]. Afterwards, research in this area is enhanced and the technique is widely adopted in applications.

For finite dimensional signal, the transform matrix can be simplified from a polyphase matrix consisting of Laurent polynomials [7] to a constant matrix of finite dimension. By matrix factorization, Hao and Shi first considered reversible integer implementations for such invertible linear transforms in a finite dimensional space [8], and later obtained an optimal factorization of minimum number of matrices [9]. The technique [10] has been included in the new international image compression standard, JPEG 2000.

However, the computational efficiency of the inverse integer transform based on their matrix factorizations still remains a problem, especially for large matrices, due to the recursiveness of the reconstruction. To overcome this drawback, in this paper we introduce two new block factorizations that are easier for computation optimization and parallel design. Actually, even for the sequential computation they may be preferred. To differentiate from block factorizations, we take the element matrix factorizations (or block size of 1-by-1) as point factorizations hereinbelow.

Section 2 recalls point factorization and block factorization. In Section 3, based on the block TERM and SERM factorizations [11, 12], we introduce two types of PERM factorizations for parallel integer transform. Section 4 is a discussion of computational complexity, and we present an efficient scheme for matrix blocking and multiprocessor arrangement in Section 5. At the end of this paper, we conclude in Section 6.

2. Point and Block Factorizations

The basic matrix factors for reversible integer transformation are called elementary reversible matrices (ERMs), including triangular ERMs (TERMs) and single-row ERMs (SERMs). A TERM is defined as a special triangular matrix whose diagonal elements belong to the unit group of an integral domain. For instance, they are ± 1 and $\pm i$ on the set $\{a+bi \mid a,b \in Z\}$, the so-called integer factors in [9]. A SERM is a matrix with integer factors on the diagonal and only one row possibly nonzero. Obviously, a SERM can be converted to a simple TERM by a row and a column permutation. Furthermore, a unit TERM is actually a unit triangular matrix and a unit SERM associated with the *i*-th row can be formulated as $\boldsymbol{S}_i = \boldsymbol{I} + \boldsymbol{e}_i \boldsymbol{s}_i^T$, where $\boldsymbol{e}_i$ is an elementary vector with the *i*-th element 1 and all others 0, and $\boldsymbol{s}_i$ is a vector whose *i*-th element is zero.

The reversible integer mapping can be implemented

* This work was supported by the foundation for the authors of National Excellent Doctoral Dissertation of China, under Grant 200038.

via a series of TERMs, or SERMs equivalently. Let $\boldsymbol{A}=(a_{i,j})$ be a lower TERM of size N with a diagonal of integer factors $j_1,\cdots,j_N$. Then the forward integer transform for $\boldsymbol{y}=\boldsymbol{A}\boldsymbol{x}$ is computed as follows:

$$\begin{cases} y_1 = j_1 x_1 \\ y_m = j_m x_m + \left\lfloor \sum_{n=1}^{m-1} a_{mn} x_n \right\rfloor \quad 2 \le m \le N \end{cases} \tag{1}$$

while its inverse should be executed in a recursive way like forward elimination:

$$x_1 = \frac{y_1}{j_1}; \quad x_m = \frac{1}{j_m}\left(y_m - \left\lfloor \sum_{n=1}^{m-1} a_{mn} x_n \right\rfloor \right), \quad m = 2,\cdots,N \tag{2}$$

where $\lfloor\ \rfloor$ is a rounding arithmetic. The computation is analogous for an upper TERM, except that the computational ordering of the inverse should be upward. It is easy to see the following characteristics of the above transform computations: (i) mapping integers to integers; (ii) perfect reconstruction; (iii) in-place computation. All these are attractive for lossless data compression.

Given an $N\times N$ nonsingular matrix $\boldsymbol{A}$, there are two SERM factorizations in [9]: (i) if the leading principal minors of $\boldsymbol{A}$ are all 1's, $\boldsymbol{A}=\boldsymbol{L}\boldsymbol{U}=\boldsymbol{S}_N\cdots\boldsymbol{S}_1$, denoted by SERM$^{(0)}$ below; (ii) if $\det(\boldsymbol{A})$ is an integer factor, then $\boldsymbol{P}^T\boldsymbol{A}=\boldsymbol{S}_N\cdots\boldsymbol{S}_1\boldsymbol{S}_0$, denoted by SERM$^{(1)}$, where $\boldsymbol{P}$ is a permutation matrix, $\boldsymbol{S}_0,\boldsymbol{S}_1,\cdots,\boldsymbol{S}_N$ are unit SERMs, and $\boldsymbol{S}_0$ is associated with the last row (also a lower TERM). If $\det(\boldsymbol{A})$ is an integer factor, after a scaling modification and a few permutations, the integer transform of $\boldsymbol{A}$ of size $N\times N$ can be implemented by no more than $N+1$ SERMs. The number of the scalar floating-point multiply-add operations are respectively N^2-N and N^2-1 for SERM$^{(0)}$ and SERM$^{(1)}$ integer transforms.

Observing that a unit SERM can be trivially generalized to a unit block SERM (for notation simplicity, we still use $\boldsymbol{S}_i$ to denote afterwards): $\boldsymbol{S}_i=\boldsymbol{I}+\boldsymbol{e}_i\boldsymbol{s}_i^T$, where $\boldsymbol{e}_i$ is an elementary block matrix of which the *i*-th block is $\boldsymbol{I}$ and $\boldsymbol{s}_i$ is a block matrix with the *i*-th block zero, we studied block factorizations in [11, 12]. By contrast with point SERM factorizations, block SERM factorizations boost the degree of parallelism and make it possible that the factorization and transforms are carried out at the block level. Such block approaches are more appropriate for efficient integer implementation of large matrices, let alone those with natural block structures originated from underlying physical backgrounds.

For example, given a 2-by-2 block unit lower SERM $\boldsymbol{A}=\begin{bmatrix}\boldsymbol{I} & \boldsymbol{0}\\ \boldsymbol{M} & \boldsymbol{I}\end{bmatrix}$, to reconstruct $\boldsymbol{x}=\begin{bmatrix}\boldsymbol{x}_1\\ \boldsymbol{x}_2\end{bmatrix}$ from $\boldsymbol{y}=\begin{bmatrix}\boldsymbol{y}_1\\ \boldsymbol{y}_2\end{bmatrix}$, the integer transform for $\boldsymbol{A}\boldsymbol{x}$, we can use the block formula below instead of the one-by-one reconstruction of (2)

$$\boldsymbol{x}=\begin{bmatrix}\boldsymbol{x}_1\\ \boldsymbol{x}_2\end{bmatrix}=\begin{bmatrix}\boldsymbol{y}_1\\ \boldsymbol{y}_2-\lfloor \boldsymbol{M}\boldsymbol{x}_1\rfloor\end{bmatrix} \tag{3}$$

where $\lfloor\ \rfloor$ is a rounding operator for all elements in the vector.

Generalizing the point factorizations to block factorizations is not so straightforward due to the difficulty of the scaling modification and the possibility that some crucial blocks may not have full rank in factorization. In [11], in an almost arbitrary partition manner, we defined a generalized determinant matrix function "**DET**" and studied the block LU (**BLU**) factorization $\boldsymbol{A}=\boldsymbol{PLDU}$, where $\boldsymbol{P}$ is a permutation matrix, $\boldsymbol{L}$, $\boldsymbol{U}$ are unit lower and unit upper block triangular matrix respectively, and $\boldsymbol{D}$ is a block diagonal matrix. We also discussed how to convert them into block unit SERM factorizations in [11]. In the case that all blocks are of the same size [12], we redefined the generalized determinant matrix function "**DET**" and obtained a **BLUS** factorization $\boldsymbol{A}=\boldsymbol{PLDUS}_0$, where $\boldsymbol{S}_0$ is a unit block SERM associated with the last block row, $\boldsymbol{D}=diag(\boldsymbol{I},\boldsymbol{I},\cdots,\boldsymbol{I},\mathbf{DET}(\boldsymbol{P}^T\boldsymbol{A}))$. Thus $\boldsymbol{S}_0$ is also a unit lower TERM. We proposed a practical algorithm in [12] as a generalization of point TERM factorization [9]. We also proved that block SERM factorization $\boldsymbol{A}=\boldsymbol{PS}_n\cdots\boldsymbol{S}_1\boldsymbol{S}_0$ exists if and only if $\mathbf{DET}(\boldsymbol{P}^T\boldsymbol{A})$ is a diagonal matrix and all the diagonal elements are integer factors.

In the following discussions, we assume uniform blocking, and mainly use the basic block SERM forms of **BLU** and **BLUS** factorizations -- $\boldsymbol{PDS}_n\cdots\boldsymbol{S}_1$ and $\boldsymbol{PD'S}_n\cdots\boldsymbol{S}_1\boldsymbol{S}_0$, where $\boldsymbol{P}$ is a permutation matrix at the element level, $\boldsymbol{D}$ is a block diagonal matrix, and $\boldsymbol{D}$' is a block diagonal matrix with only one diagonal block not to be $\boldsymbol{I}$ (in this paper it's supposed to be the bottom-right block). Throughout the rest of this paper, let $\boldsymbol{A}$ be the original transform matrix in a finite dimensional space, *n* the number of blocks in a row or column, *m* the size of the each block, and $\boldsymbol{A}_n$ the corresponding block matrix of $\boldsymbol{A}$.

3. Parallel ERM (PERM) factorizations

For parallel computing, a linear transform of an $N\times N$ block SERM, $\boldsymbol{S}_i=\boldsymbol{I}+\boldsymbol{e}_i\boldsymbol{s}_i^T$, with the *i*-th block of $\boldsymbol{s}_i$ being zeros and of block size $m\times m$, can be implemented by parallel multiplications and parallel additions. The main difficulty of applying block factorizations to efficient parallel computing lies in $\boldsymbol{D}$ (or $\boldsymbol{D}'$), the residue. Row and column permutations alone are not capable of converting $\mathbf{DET}(\boldsymbol{A}_n)$ into $\boldsymbol{I}$. For $\boldsymbol{D}'$, $\mathbf{DET}(\boldsymbol{A}_n)$ is a non-identity block. See [11, 12] for detailed definitions of **DET**. Therefore, we exploit recursive factorizations. For a

matrix of size N_1, at the k-th level, partition the residue of the last level into $n^{(k)}$ blocks of size $m^{(k)}$, till the block size reduces to N_2. This process is denoted as

$$N_1 = m^{(0)} \xrightarrow{n^{(1)}} m^{(1)} \xrightarrow{n^{(2)}} m^{(2)} \cdots m^{(K-1)} \xrightarrow{n^{(K)}} m^{(K)} = N_2 \quad (4)$$

Take **BLU** factorization as an example. At the k-th level, each diagonal block of $\boldsymbol{D}^{(k-1)}$ is further partitioned into $n^{(k)} \times n^{(k)}$ blocks of block size $m^{(k)} \times m^{(k)}$. Then we apply BLU and block SERM factorizations to factorize $\boldsymbol{D}^{(k-1)}$ into $n^{(k)}$ block SERMs, formally denoted as $\boldsymbol{S}_j^{(k)} (1 \le j \le n_k)$, and a non-ERM block diagonal matrix $\boldsymbol{D}^{(k)}$. Repeat this process recursively till all the non-ERM blocks are reduced to single elements (see Figure 1 for an illustration), and finally we obtain

$$\begin{aligned} \boldsymbol{A} &= \boldsymbol{P}^{(1)}\boldsymbol{P}^{(2)}\cdots\boldsymbol{P}^{(K)}\boldsymbol{D}^{(K)}(\boldsymbol{L}^{(K)}\boldsymbol{U}^{(K)})\cdots(\boldsymbol{L}^{(1)}\boldsymbol{U}^{(1)}) \\ &= \boldsymbol{P}\boldsymbol{D}\prod_{k=K}^{1}\boldsymbol{S}_{n^{(k)}}^{(k)}\cdots\boldsymbol{S}_1^{(k)} \end{aligned} \quad (5)$$

where $\boldsymbol{D} = diag(d_1, \cdots, d_N)$, and K is the number of factorization levels. It's not difficult to see that $\prod_{j=1}^{i} d_j$ is the i-th leading principal minor of $\boldsymbol{P}^T\boldsymbol{A}$.

Similarly, successively applying **BLUS** to factorize the last diagonal block of previously remained non-identity sub-matrix as shown in Figure 2, we obtain

$$\boldsymbol{A} = \boldsymbol{P}\boldsymbol{D}\prod_{k=K}^{1}\boldsymbol{S}_{n^{(k)}}^{(k)}\cdots\boldsymbol{S}_1^{(k)}\boldsymbol{S}_0^{(k)} \quad (6)$$

where $\boldsymbol{D} = diag(1, \cdots, 1, \det(\boldsymbol{P}^T\boldsymbol{A}))$.

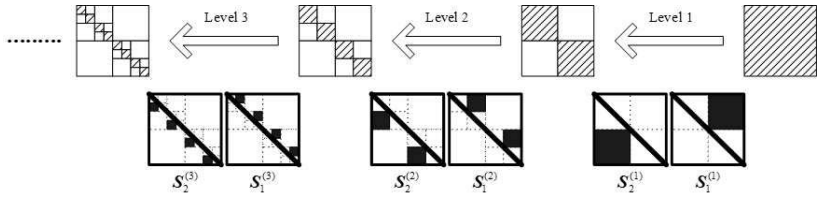

Figure 1 PERM$^{(0)}$ factorization (Suppose $n^{(1)} = n^{(2)} = n^{(3)} = 2$)

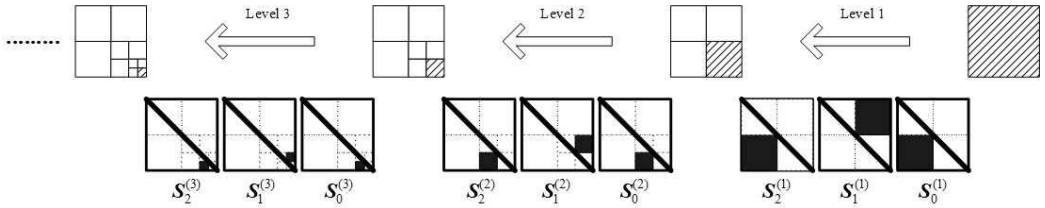

Figure 2 PERM$^{(1)}$ factorization (Suppose $n^{(1)} = n^{(2)} = n^{(3)} = 2$)

To realize perfect integer-reversible transform, we need to make a scaling modification to the original transform matrix as suggested in [9]. For factorization formula (5), we can left-multiply $\boldsymbol{A}$ by $\boldsymbol{P}\boldsymbol{D}^{-1}\boldsymbol{P}^T$, where the leftmost $\boldsymbol{P}$ is to maintain the order. Since the scaling values here are perhaps only meaningful in mathematics, formula (5) may be of limited use in real-world applications, although it has fewer factor matrices. By contrast, multilevel factorization (6) has one more term at each level, but the less restrictive modification provides more flexibility and practicability: we are free to choose any rows or columns for scaling, as long as the final determinant turns out to be an integer factor. This property plays an important role in keeping proportions of the transform matrix and adjusting the dynamic ranges of data (see Section VIII of [9]). Of course, **BLU** and **BLUS** can be combined in factorization. Analogously, we can draw similar conclusions from right-permutation block factorizations.

Hereafter, the scaled formulas (5) and (6) which are appropriate for perfectly reversible integer transform are referred to as parallel ERM (PERM) factorizations and are denoted by PERM$^{(0)}$ and PERM$^{(1)}$, respectively, as a counterpart of SERM$^{(0)}$ and SERM$^{(1)}$. From the scaling process we easily see that it is sufficient to investigate unit PERM and unit SERM factorizations.

4. Parallel computational complexity

For PERM$^{(0)}$, if $N_1 = N$, $N_2 = 1$, there are

$$\sum_k \left(\frac{m^{(k-1)}}{n^{(k)}} \cdot \frac{N}{m^{(k-1)}}\right) \cdot \left(m^{(k-1)}\left(1 - \frac{1}{n^{(k)}}\right)\right) \cdot n^{(k)} = N^2 - N \quad (7)$$

multiplications and additions, equal to those of SERM$^{(0)}$. For PERM$^{(1)}$, the number is

$$\sum_k m^{(k)}(m^{(k-1)} - m^{(k)})(n^{(k)} + 1) = N^2 - 1 \quad (8)$$

also the same as SERM$^{(1)}$. Thus the sequential computational complexity of PERM does not increase in the least, and does not take advantage of the factorization, either. In fact, since the computation can be now focused onto blocks (see formula (3) for an example), the performance can be improved more by using some

mathematical packages, such as BLAS. Moreover, with nontrivial elements to be $m(N-m) \geq N-1$, degree of parallelism increases and more processors (up to $N^2/4$) can be involved in computing. We notice that the additional freedom of row partitioning in the two dimensional data structure helps cutting down the computation cost for parallel computing, owing to the independent reconstruction of all the intra-block rows in the inverse PERM integer transform.

In a block SERM transformation, all the multiplications can be efficiently done in parallel by using as many processors as possible, so the total computational time of multiplications in parallel is $\lceil m(N-m)/p \rceil$ times multiplications if the number of processors is *p*. However, additions are not so simple. For dual additions, *p* processors can only implement addition (summation) of *n* numbers in $\lceil \log_2 n \rceil$ times addition if $n < 2p$. For $n \geq 2p$, the computational time of additions is $\lceil (n-p)/p + C \log_2 p \rceil$, where $0 < C \leq 1$. Therefore, the computational time of additions in parallel for a block SERM transform is $\lceil \log_2(N-m) \rceil$ if $m(N-m) < 2p$, and $\lceil (m(N-m)-p)/p + C(\log_2 p - \log_2 m) \rceil$, or simply $\lceil m(N-m)/p + C(\log_2 p - \log_2 m) \rceil$, if $m(N-m) \geq 2p$.

Theoretically, the multiplication time of the parallel integer transform (4) with PERM$^{(0)}$ is

$$T^*_{PERM^{(0)}} = \sum_{k=1}^{K} n^{(k)} \left\lceil m^{(k)}(m^{(k-1)} - m^{(k)})/p \right\rceil \frac{N_1}{m^{(k-1)}} \approx \frac{N_1}{p} \sum_{k=1}^{K} (m^{(k-1)} - m^{(k)}) = \frac{N_1(N_1 - N_2)}{p} \quad (9)$$

where $n^{(k)} m^{(k)} = m^{(k-1)}$, $m^{(0)} = N_1$, $m^{(K)} = N_2$.

Analogously, the multiplication time of (4) with PERM$^{(1)}$ is

$$T^*_{PERM^{(1)}} = \sum_{k=1}^{K} (n^{(k)}+1) \left\lceil m^{(k)}(m^{(k-1)} - m^{(k)})/p \right\rceil \approx \frac{1}{p} \sum_{k=1}^{K} \left((m^{(k-1)})^2 - (m^{(k)})^2 \right) = \frac{N_1^2 - N_2^2}{p} \quad (10)$$

where $n^{(k)} m^{(k)} = m^{(k-1)}$, $m^{(0)} = N_1$, $m^{(K)} = N_2$.

From (9) and (10), we see that the multiplication time has nothing to do with *n* for PERM$^{(0)}$ and PERM$^{(1)}$.

If all $n^{(k)}$ are equal to n, then $m^{(k)} = m^{(k-1)}/n = m^{(0)}/n^k = N_1/n^k$, and we have $m^{(K)} = N_1/n^K = N_2$ or $K = \log_n(N_1/N_2)$.

However, PERM factorizations are not perfect. Supposing all $n^{(k)}$ are equal to n, the total number of rounding operations of PERM$^{(1)}$ is

$$\sum_{k=1}^{K} (n^{(k)}+1) m^{(k)} = (n+1) N_1 \sum_{k=1}^{K} \frac{1}{n^k} = (n+1) N_1 \frac{n^K - 1}{n^K (n-1)} = (N_1 - N_2) \frac{n+1}{n-1} \quad (11)$$

which is a decreasing function of *n* and achieves its minimum when $n = N_1/N_2$ and $K = 1$. For PERM$^{(0)}$, the total number is KN. Hence, as the block size or the number of factorization levels grows, the rounding operations also increase, which will probably result in higher transform error though integer reversibility is still guaranteed.

The additions cannot be done all in parallel, so the addition time is theoretically more complicated than multiplication time.

For PERM$^{(0)}$, if there are *p* processors and as many processors as possible are used in computation, the parallel addition time can be estimated as

$$T^+_{PERM^{(0)}} = \sum_{k=1}^{K_p} n^{(k)} \left\lceil \begin{matrix} \left(m^{(k)}(m^{(k-1)} - m^{(k)}) - p \right)/p \\ + C\left(\log_2 p - \log_2 m^{(k)} \right) \end{matrix} \right\rceil \frac{N_1}{m^{(k-1)}} + \sum_{k=K_p}^{K} n^{(k)} \left\lceil \log_2(m^{(k-1)} - m^{(k)}) \right\rceil \frac{N_1}{m^{(k-1)}} \quad (12)$$

For PERM$^{(1)}$, the parallel addition time can be

$$T^+_{PERM^{(1)}} = \sum_{k=1}^{K_p} (n^{(k)}+1) \left\lceil \begin{matrix} \left(m^{(k)}(m^{(k-1)} - m^{(k)}) - p \right)/p \\ + C\left(\log_2 p - \log_2 m^{(k)} \right) \end{matrix} \right\rceil + \sum_{k=K_p}^{K} (n^{(k)}+1) \left\lceil \log_2(m^{(k-1)} - m^{(k)}) \right\rceil \quad (13)$$

Above time estimations are related to a turning point, K_p, where $m^{(K_p)}(m^{(K_p-1)} - m^{(K_p)})$ should be a number closest to but less than 2*p*. And as level *k* steps higher, the problem size decreases to such an extent that the speed-up reaches its limit and cannot be improved further. However, in order to minimize the computational time, we can split the whole task into several phases and use different processor allocation schemes in different phases.

5. Matrix blocking strategy

How to partition the matrix and allocate the data to processors is a practical problem to apply PERM factorizations, for it determines the parallel complexity of the corresponding integer transform. Generally speaking, appropriate blocking strategy is made according to specific optimization principles. Ignoring other factors like communication and multiprocessor architecture, we just consider the computation time of parallel multiplications and parallel additions as the metrics to evaluate the block structure of PERM$^{(1)}$.

Because there exists a turning point in parallel implementation, it is necessary to consider the block structure in the case of a small matrix or abundant processors. Besides, row distribution should be given first priority if only a few processors are available, for it leads

to a higher degree of parallelism of addition. From the above discussion we propose a three-phase blocking strategy:

(i) If $N \geq 2p$, then factorize the matrix recursively in the first phase till the block size is reduced to p, i.e., $N \to \cdots \to p$. In this phase data is allocated in rows to multiprocessor. To minimize the transform error, we can employ immediate one-level block factorization of N/p blocks;

(ii) If $2\sqrt{p} \leq N < 2p$, then perform $N \to \cdots \to \sqrt{p}$ in this phase. In mapping the data onto processors, we still take priority in row distribution. Again, a straightforward factorization is reasonable with the block size $\sqrt{p}$;

(iii) If $N \leq 2\sqrt{p}$, then $N \to \cdots \to 1$. Processors are excessive in this phase and thus the availability drops. To minimize the parallel cost of multiplication, or equivalently, the number of matrices, we have

$$\sum_{k=1}^{K}(n^{(k)}+1) = (n+1)K = (n+1)\log_n N \tag{14}$$

where $n = n^{(1)} = m^{(0)}/m^{(1)} = n^{(2)} = m^{(1)}/m^{(2)} = \cdots = n^{(K)} = m^{(K-1)}/m^{(K)}$. It follows that the minimum value can be obtained at $n = 4$, i.e., partitioning into 4 blocks at each level is the best solution. In such case, the parallel computation time of additions and multiplications is:

$$T^{*}_{\mathrm{PERM}^{(1)}} = \sum_{k=1}^{K}(n^{(k)}+1) = 5\log_4 N \tag{15}$$

$$T^{+}_{\mathrm{PERM}^{(1)}} = \sum_{k=1}^{K}(n^{(k)}+1)\log_2\left((n^{(k)}-1)m^{(k)}\right) = \sum_{k=1}^{K} 5\log_2\frac{3N}{4^k} = 5\log_4 N\left(\log_4 9N - 1\right) \tag{16}$$

We now draw a comparison on the computation time between the above blocking scheme and the direct parallelization of SERM$^{(1)}$. Let $T^{*/+}_{\mathrm{PERM}^{(1)}}$ and $T^{*/+}_{\mathrm{pSERM}^{(1)}}$ denote the time complexity of parallel addition/multiplication with PERM factorization and SERM factorization respectively. For above blocking strategy, we have

$$T^{*}_{\mathrm{PERM}^{(1)}}(N,p) = \begin{cases} f_1^{*}(N,p) = \left(\frac{N}{p}+1\right)\cdot\left(\frac{N}{p}-1\right)\cdot p + f_2^{*}(p,p), & p \leq \frac{N}{2} \\ f_2^{*}(N,p) = \left(\frac{N}{\sqrt{p}}+1\right)\cdot\left(\frac{N}{\sqrt{p}}-1\right) + f_3^{*}(\sqrt{p},p), & \frac{N}{2} < p < \frac{N^2}{4} \\ f_3^{*}(N,p) = 5\log_4 N, & p \geq \frac{N^2}{4} \end{cases} \tag{17}$$

$$T^{+}_{\mathrm{PERM}^{(1)}}(N,p) = \begin{cases} f_1^{+}(N,p) = \left(\frac{N}{p}+1\right)\cdot\left(\frac{N}{p}-1\right)\cdot p + f_2^{+}(p,p), & p \leq \frac{N}{2} \\ f_2^{+}(N,p) = \left(\frac{N}{\sqrt{p}}+1\right)\cdot\left(\frac{N}{\sqrt{p}}-1+C\log_2\sqrt{p}\right) + f_3^{+}(\sqrt{p},p), & \frac{N}{2} < p < \frac{N^2}{4} \\ f_3^{+}(N,p) = 5\log_4 N\left(\log_4 9N - 1\right), & p \geq \frac{N^2}{4} \end{cases} \tag{18}$$

while $$T^{*}_{\mathrm{pSERM}^{(1)}}(N,p) = (N+1)\frac{N-1}{p} \tag{19}$$

$$T^{+}_{\mathrm{pSERM}^{(1)}}(N,p) = (N+1)\left(\frac{N-1}{p} + C\log_2 p\right) \tag{20}$$

First, the effective processors can be up to $N^2/4$ for PERM integer transform. From (17) and (18), it's easy to show that the costs of multiplication and addition are both $O(N)$ when $p = O(N)$, are $O(\log N)$ and $O(\log^2 N)$ respectively when $p = O(N^2)$. By contrast, the effective processors can not exceed N for SERM$^{(1)}$ transform; and when either $p = O(N)$ or $p = O(N^2)$, the time of multiplication and addition is $O(N)$ and $O(N \log N)$, respectively. Just as demonstrated in Table 1, Figure 3 and Figure 4 for $N = 64$ and $C = 1$, PERM$^{(1)}$ is more efficient than SERM$^{(1)}$ for parallel computation.

Table 1 Time Complexity Comparison

Operation \ p		$O(N)$	$O(N^2)$
Multiplications	SERM$^{(1)}$	$O(N)$	$O(N)$
	PERM$^{(1)}$	$O(N)$	$O(\log N)$
Additions	SERM$^{(1)}$	$O(N\log N)$	$O(N\log N)$
	PERM$^{(1)}$	$O(N)$	$O(\log^2 N)$

Of course, regardless of communication and other overheads, the above blocking strategy is only demonstrative in nature. In practice, the blocking may be flexible due to different requirements. For instance, to accommodate as many processors for parallel computing as possible, we may use multilevel binary partitioning.

Distinct from that of PERM$^{(1)}$, although total problem size of PERM$^{(0)}$ also drops (yet slower) as the level increases, the number of effective rows in each matrix can remain unchanged: at level *k*, there are altogether $N/n^{(k)}$ components updated in a single step, whereas the number for PERM$^{(1)}$ is $N/\left(n^{(1)}\cdots n^{(k)}\right)$. This trait is conducive to row allocation to efficiently utilize processor resources. For instance, assuming each $N/n^{(k)}$ is a multiple of p, the parallel complexity of multiplication and addition is both $(N^2 - N)/p$.

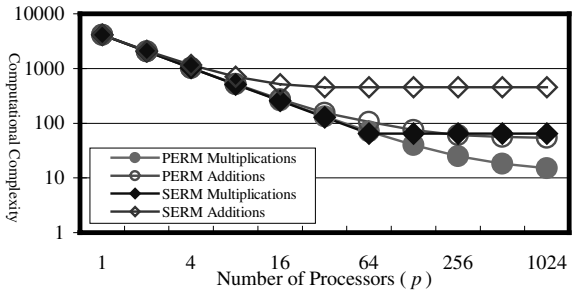

Figure 3 Computation cost of PERM$^{(1)}$ and parallel SERM$^{(1)}$ transforms (N = 64, C = 1)

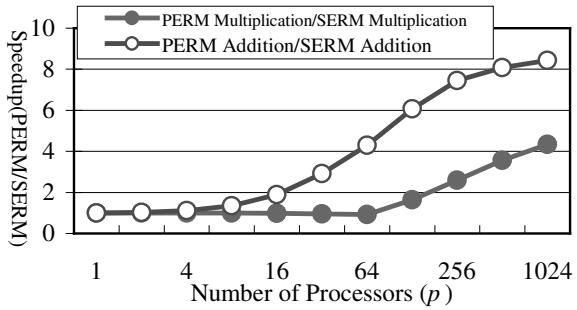

Figure 4 Relative speedup of PERM$^{(1)}$ over parallel SERM$^{(1)}$ integer transforms (N = 64, C = 1)

6. Concluding remarks

In the above discussions, we have presented PERM factorizations for parallel reversible integer transforms based on block factorizations. Compared with SERM factorizations, they improve the parallel performance. Particularly, they increase the degree of parallelism and thus accommodate more processors. Since the PERM factorizations and the corresponding integer transforms can all be calculated at the block level, we also expect the efficiency in sequential computation with special matrix computation software such as BLAS can speedup the block operations. Consequently, the PERM factorizations are attractive for large matrix integer transforms. In consideration of the flexibility of the scaling modification, PERM$^{(1)}$ may be more promising in real-world applications.

One drawback brought in by the PERM factorizations is that larger block size and more factorization levels result in more rounding operations, and possibly greater transform error. However, noticing that $\lfloor\ \rfloor$ in (1) and (2) can actually be any nonlinear operators, we may keep more decimal digits (e.g., rounding to hundredths or thousandths) to effectively reduce the transform error. The error bound given in Section VII of [9] can be used to determine the precision. Another disadvantage is that the problem size gradually drops with the accretion of level k will probably reduce the availability of processors. This can not be ignored especially when PERM$^{(1)}$ is employed with relatively more processors.

The key to applying the PERM factorizations is the blocking strategy. Including other necessary factors such as the communication, our future work is to study this problem systematically and test the performance by further experimentation.

7. References

[1] H. Blume and A. Fand, Reversible and irreversible image data compression using the S-transform and Lempel-Ziv coding, *Proceedings of SPIE*, 1989, 1091, pp. 2-18.

[2] A. Zandi, J. D. Allen, E. L. Schwartz and M. Boliek, CREW: Compression with reversible embedded wavelets, in *Proceedings of IEEE Data Compression Conference*, 1995, pp. 212-221.

[3] A. Said and W. A. Pearlman, An image multiresolution representation for lossless and lossy compression, *IEEE Transactions on Image Processing*, 1996, 5, pp. 1303-1310.

[4] W. Sweldens, The lifting scheme: A custom-design construction of biorthogonal wavelets*, J. of Applied and Computational Harmonic Analysis*, 1996, 3, pp. 186-200.

[5] F. A. M. L. Bruekers, A. W. M. van den Enden, New networks for perfect inversion and perfect reconstruction, *IEEE J. on Selected Areas in Communications*, 1992, 10, pp. 130-137.

[6] I. Daubechies, W. Sweldens, Factoring wavelet transforms into lifting steps*, J. of Fourier Analysis and Applications*, 1998, 4, pp. 247-269.

[7] A. R. Calderbank, I. Daubechies, W. Sweldens and B.-L. Yeo, Wavelet transform that map integers to integers, *J. of Applied and Computational Harmonic Analysis*, 1998, 5, pp. 332-369.

[8] P. Hao and Q. Shi, Invertible linear transforms implemented by integer mapping, *Science in China*, Series E (in Chinese), 2000, 30, pp. 132-141.

[9] P. Hao and Q. Shi, Matrix factorizations for reversible integer mapping, *IEEE Trans. Signal Processing*, 2001, 49 pp. 2314-2324.

[10] P. Hao and Q. Shi, Proposal of reversible integer implementation for multiple component transforms, *ISO/IEC JTC1/SC29/WG1N1720*, Arles, France, 2000.

[11] Y. She and P. Hao, A new block factorization of nonsingular matrices for integer transform, submitted to *Linear Algebra and Its Applications*, 2003.

[12] Y. She and P. Hao, A block TERM factorization of nonsingular uniform block matrices, *Science in China*, 2004, 34(2).