ISSN No. 0976-5697

Volume 1, No. 3, Sept-Oct 2010

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Efficient Algorithm for Multi-Dimensional Matrix Multiplication Operations Representation

Satya Prakash*
Birla Institute of Technology, Mesra Ranch (Ext. Centre Noida)
E-mail: spsinghbit@yahoo.co.in

Anil K Ahlawat
Department of Computer Science& Engineering,
Ajay Kumar Garg Engineering College Ghaziabad
E-mail: a_anil2000@yahoo.co.in

***Abstract*:** Multi-dimensional arrays are widely used in a lot of scientific studies but still some issues have been encountered regarding efficient operations of these multi-dimensional arrays. In this paper, the extended Karnaugh Map representation (EKMR) scheme has been proposed as an alternative to the traditional matrix representation (TMR) which caused the multi-dimensional array operation to be inefficient when extended to dimensions higher than two. EKMR scheme has managed to successfully optimize the performance of the multi-dimensional array operations to the nth dimension of the array. The basic concept EKMR is to transform the multi-dimensional array in to a set of two-dimensional arrays. EKMR scheme implies Karnaugh Map which is a technique used to reduce a Boolean expression. It is commonly represented with the help of a rectangular map which holds all the possible values of the Boolean expression. Then the efficient data parallel algorithms for multi-dimensional matrix multiplication operation using EKMR are presented in this study which outperformed those data parallel algorithms for multi-dimensional matrix multiplication operation which used the TMR scheme. The study encourages designing data parallel algorithms for multi-dimensional dense and sparse multi-dimensional arrays for other operations as well using the EKMR scheme since this scheme produces the efficient performance for all dimensions and for all operations of the arrays.

***Keywords*:** Matrix multiplication Algorithm, EKMR, TMR.

I. INTRODUCTION

Multi-dimensional arrays which are also referred as tensors or n-ways arrays are usefully applied to a wide range of studies or methods such as climate modeling, finite element analysis (FEA), molecular dynamic and many more but still many issues have been encountered regarding efficient operations of these multi-dimensional arrays. Most of the proposed methods are successful in case of two-dimensional arrays which do not show accurate results when applied to the extended form of tensors. This occurred due to the traditional matrix representation (TMR) which is an array representation scheme that is commonly used to represent the multi-dimensional dense or sparse array. Dense and sparse are the two categories of the array form which are provided through the various data parallel programming languages [2] for instance, Vienna Fortran, High Performance Fortran, etc. If all or most of the array elements are non-zero values then it is called a dense array. On the other hand, if most of the elements of the array are zero then it is called a sparse array. When an operation is applied on a dense array then it is executed on elements of the dense array whereas in case of the sparse array, an operation is exercised only on the non-zero elements in order to optimize the performance [1]. Coming back to the flaws of the TMR which is also known as canonical data layouts, there are three reasons found for the failure of the TMR scheme when applied on a dense array which has three or more than three dimensions. First reason is the increase in the cost of packing/unpacking of the elements of the dense tensor in relation to its dimensions, second is the increase in its cost of the index computations with the increase of its dimensions. Third reason is the increase in the rate of cache miss for an operation with the increase of the dimensions of the dense tensor since more cache lines are acquired [5] [6] [7]. Due to these three drawbacks, TMR scheme has turned out to be a difficult and less tractable for designing efficient data parallel algorithms for tensor operations.

In case of designing the parallel programs for operations on sparse tensors, the programming languages usually use compressed row storage (CRS) and compressed column storage (CCS) as the data compression scheme to compress the sparse arrays with respect to the TMR scheme due to which operation is only performed over non-zero elements of the sparse arrays in order to improve performance and reduce memory space. But still parallel array operations with respect to CRS or CCS for higher dimensional tensors have also failed to produce good performance merely, because of the following two reasons. First reason is that more of the single dimensional matrices are required with the increase of dimensions of the tensors in order to store the resultant extra indices of non-zero elements which further increase the time and the required storage space. The second reason is that with the increase in the dimensions of the tensors, the cost of indirect data access [4] and the cost of index comparisons increase for parallel operations on sparse tensors.

Thus, this dissertation is aimed towards providing a new, effective and efficient array representation scheme and data compression scheme for dense and sparse tensors, respectively. These new array representation scheme and data compression scheme would then be used to design a parallel algorithm for multi-dimensional matrix multiplication operation. The new array representation scheme provided in this dissertation is called the Extended Karnaugh Map Representation (EKMR) which is based on the concept of representing a multi-dimensional array as a set of two-dimensional arrays [14]. This scheme is appropriate for both dense and sparse tensors. Thus, it has become easier to design an efficient parallel algorithm for tensors of higher dimensions with the help of EKMR. The

theoretical and experimental analysis proved that the EKMR scheme is better than the TMR scheme.

II. EKMR SCHEME

"Chun-Yuan Lin" from Institute of Molecular and Cellular Biology, National Tsing Hua University, Hsinchu, 300, Taiwan, "Yeh-Ching Chung" from Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, 300 and "Jen-Shiuh" Liu from Department of Information Engineering and Computer Science, Feng Chia University, Taichung, Taiwan, 407 proposed the scheme of Extended Karnaugh Map Representation which is primarily based on the Karnaugh Map. The Karnaugh Map is a technique used to reduce a Boolean expression. It is commonly represented with the help of a rectangular map which holds all the possible values of the Boolean expression. The n variables are used to hold memory space and 2^n possible combinations are represented for an n-input Karnaugh Map. If n is less than or equal to 4 then the Karnaugh Map can be shown as a two-dimensional array and thus, it can be easily represented on a plane. EKMR(1) is a single input Karnaugh Map that is a simple one-dimensional array or a vector. Similarly, EKMR(2), EKMR(3) and EKMR(n) are two-dimensional, three-dimensional and n-dimensional arrays, respectively where n is the number of inputs. Thus, for n equals to 1 and 2, EKMR(n) and TMR(n) exhibit the same array representation. Therefore, we will take in to account EKMR(n) where n is greater than 2.

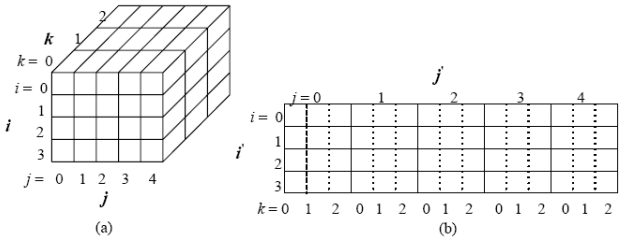

Figure 2.1 (a) 3x4x5 array represented by TMR (b) 4x15 array represented by EKMR

Figure 2.1 represents the TMR(3) and EKMR(3) where (a) is a 3x4x5 array represented by TMR(3) and (b) is a 4x15 array represented by EKMR(3). Practically, a multi-dimensional array requires linear memory storage for programming languages supporting multi-dimensional array. Programming languages copy the array index space in to the linear memory address. Thus, an array A[k][i][j] which is represented by TMR(3), has the memory address L_{RM} (k,i,j;3,4,5) (that is the row major data representing function) and L_{CM} (k,i,j;3,4,5) (that is the column major data representing function) for the array element in the third dimension 'k', row dimension 'i' and column dimension 'j' with respect to the starting memory address of a 3x4x5 size array.

A 3-input Karnaugh Map represents the TMR(3) array A[k][i][j] as a 2-dimensional array with respect to EKMR(3). Figure 2.1(b) shows the relative EKMR(3) representation of A[k][i][j]. Figure 2.1(b) is a 4x15 size 2-dimensional array. The EKMR(3) is also given by the row major data representing function L'_{RM}(i',j';4,15) which is equivalent to i'x15+j' or the column major data representing function L'_{CM}(i',j';5,12) which is equivalent to j'x5+i'.The placement of the elements of the array with respect to the direction given by the index 'k' is the primary difference between the two data representation schemes which are TMR(3) and EKMR(3).

III. PARALLEL ALGORITHM FOR MULTIDIMENSIONAL MATRIX

The design of a parallel algorithm usually has three phases which are: data distribution, local computation and result collection [9]. Further on, we will analyze the issues encountered in these three phases which are concerned with the design of the data parallel algorithms of the multi-dimensional dense matrix operations with respect to the row major data layout using Karnaugh Map that is EKMR scheme. These data parallel algorithms are not regarded to be based on the recursive data layout [10]. The reason behind this consideration is the selection of the recursive data layout with respect to the TMR scheme tensor operation algorithm in order to optimize the performance [11]. In our design example, we will only use multiplication operation on the matrices although there are other array operations as well. Our example is using the distribution of one dense array since the distribution of more dense arrays for an operation will design a very complicated parallel algorithm.

A. The Data Distribution Phase

In this section, we will illustrate the row and the column which are the distribution methods for dense arrays. A dense global array is distributed to the processor in three steps. In step number one, the global array is divided in to the local dense arrays on the basis of the data partition method. In step number two, the elements of the local dense array from step number one, are collected in the form of a batch and then distributed to the relevant processor in the step number three. The cost of the row and the column data distribution methods are the same for the first and the third steps [3] so we will discuss the cost of packing in the second step in order to analyze these data distribution methods. The hypothesis is that A[k][i][j] is an n x n x n dense array and P are the given processors.

a)The Row Data Distribution Method: According to the definition of EKMR(3), A' comprises of n rows and each row has n^2 elements. As per the row distribution method, A'[i'][j'] is distributed to the processors by diving A' in to 2-dimensional arrays P in the direction i'. The elements of the same row are saved in the sequential memory addresses in the EKMR(3), so it is not necessary to pack them before distribution as shown in figure 3.1(b). Therefore, the row size is same for TMR(3) and EKMR(3).

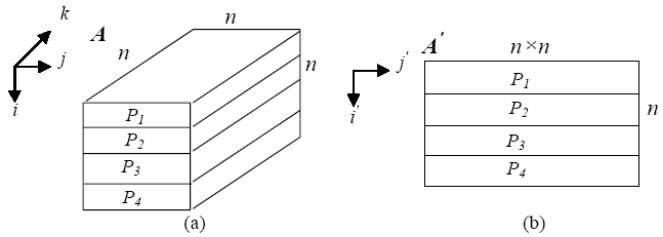

Figure 3.1: Row data distribution method for A and A' to 4 processors (a)TMR(3) (b) EKMR(3).

Figure 3.1 shows the row data distribution method for A and A' to 4 processors where (a) is based on TMR(3) and (b) is based on EKMR(3).

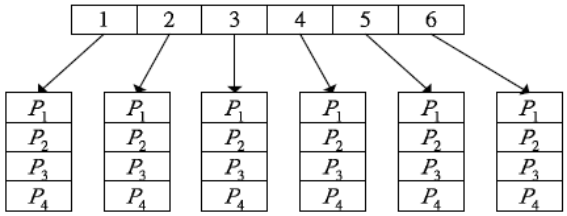

Figure 3.2 shows the row data distribution method for A' to 4 processors for EKMR(6).

The number of non-consecutive data blocks is zero for a single processor as well as for multiple processors. For the EKMR(4), elements of the partially dense arrays allocated to each processor are saved to the sequential memory addresses. So, they are packed before distribution as shown in figure 3.2. Now A' is the corresponding EKMR(n) of the n^n dense array. The data parallel algorithm for the EKMR(n) with respect to the row data distribution method of the data distribution phase is given below in figure 3.3.

Algorithm row_data distribution method_EKMR(n)

1. *for* (P_id = 0; P_id< P; P_id ++)
2. *l* = 0;
3. offset = P_id × row _size; /* row_size = $\left\lceil \frac{n^2}{P} \right\rceil$ or $\left\lfloor \frac{n^2}{P} \right\rfloor$ */
4. *for*(x = 0; x < n^{n-4}; x++)
5. *for* (i =0; i < row_size; i++)
6. *for*(j = 0; j < n^2; j++)
7. D[*l*] = A_x[i + offset] [j]; /*Pack array elements into buffer D*/
8. *l*++ ;
9. Distribute D[] to appropriate processor;

*End_of_row_data distribution method*_EKMR(n)

Figure 3.3

The EKMR(n) row size for the first r processors = $\left\lceil \frac{n^2}{P} \right\rceil$ and

EKMR(n) row size for the remaining P - r processors = $\left\lfloor \frac{n^2}{P} \right\rfloor$

According to figure 3.2 and figure 3.3, P x n^{n-4} is the number of non-consecutive data blocks on P processors. Thus, the number of non-continuous data blocks in the EKMR scheme for row data distribution method is less than that of TMR scheme.

b) The Column Data Distribution Method

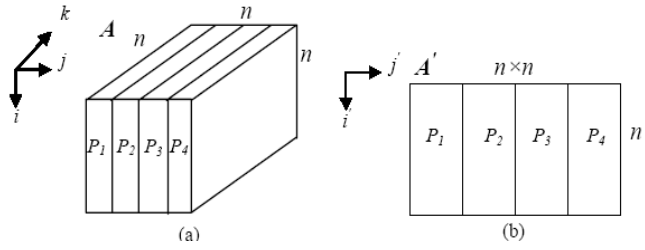

Figure 3.4 shows column data distribution method for A and A' to n=4 processors where (a) represents TMR(3) and (b) represents EKMR(3).

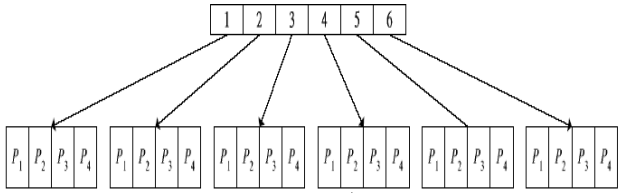

Figure 3.5 shows the column data distribution method for A to 4 processors according EKMR(6)

Algorithm column_data distribution method_EKMR(n)

1. *for* (P_id = 0; P_id < P ; P_id ++)
2. *l* = 0;
3. offset = P_id ×column _size; /* row_size = $\left\lceil \frac{n^2}{P} \right\rceil$ or $\left\lfloor \frac{n^2}{P} \right\rfloor$ */
4. *for*(x = 0; x < n^{n-4}; x++)
5. *for* (i =0; i < n ; i++)
6. *for* (j =0; j < column_size; j++)
7. D[*l*] = A_x [i][j + offset] ; /*Pack array elements into buffer D*/
8. *l*++
9. Distribute D[] to appropriate processor ;

*End_of_coloumn_data distribution method*_EKMR(n)

Figure 3.6

Figure 3.6 shows the algorithm column data distribution method for EKMR(n).

According to figure 3.4 and figure 3.6, P x n^{n-2} is the number of non-consecutive data blocks on P processors. Thus, the number of non-continuous data blocks in the EKMR scheme for column data distribution method is less than that of TMR scheme.

B. The Local Computation Phase

When single dense array is distributed to the processors then the next phase is to perform local computation on these distributed arrays. Assume A[m_{n-4}] [m_{n-3}]… [m_1][l][k][i][j] and B[m_{n-4}] [m_{n-3}]… [m_1][l][k][i][j] are two dense arrays with size n^n for the TMR(n) then A' and B' are the relative EKMR(n) of A(m_{n-4}, m_{n-3},.. m_1) and B(m_{n-4}, m_{n-3},.. m_1) ; C for TMR(n) and C' for EKMR(n) are the local dense arrays in each processor. The parallel algorithm of the computational phase for multidimensional matrix multiplication operation for EKMR(n) with respect to the row distribution method (this algorithm is also similar for column data distribution method) is shown is figure 3.7.

```
Algorithm local computation_row_data distribution method_EKMR(n)
1.  For (P_id = 0; P_id < P ;  P_id ++)
2.      for (x = 0; x<n^(n-4); x++)
3.        for(i =0; i< row_size ; i++)
4.          t = i×n;
5.            for (j =0; j <  n ; j++)
6.            v = j×n;
7.            for (l =0; l <  n ; l++)
8.          w = t + l;
9.         u =  l+ v;
10.   for (m =0;  m  < n;   m++)
11.    r = m ×n ;
12.   for (k = 0; k < n;  k++)
13. for (k =0; k<n ; k++) C_x[w][k+r]= C_x[w][k+r]+A_x[w][k+v]×B_x[u][k+r];
    end_of_local computation_row_data distribution method_EKMR(n)
```

Figure 3.7

The cost of computing index of elements and dense array multiplication operations for TMR(n) = $(\frac{3n^2-3n+2}{2})\alpha m^{n+1}+(3n-2)\beta m^{n+1}$

The cost of computing index of elements and dense array multiplication operations for EKMR(n) = $(7\alpha+10\beta)m^{n+1}+\alpha m^{n}+2\beta m^{n-1}+\alpha m^{n-2}+\alpha m^{n-3}$

The loopcost(l) for parallel algorithms of the local computation phase of multiplication operations for EKMR(n) with inner most loop index J = $(2\left\lceil\frac{m^2}{r}\right\rceil+\left\lceil\frac{m}{r}\right\rceil)\times m^{n-1}$

The loopcost(l) for parallel algorithms of the local computation phase of multiplication operations for TMR(n) with inner most loop index J = [12]. $(2\left\lceil\frac{m}{r}\right\rceil+1)\times m^{n}$

C. The Result Collection Phase

The results generated from the computation phase which are dispersed among processors must be gathered to produce the final result. Generally, the processor that is used for distributing the data is also used for collecting the results. Different ways are adopted by the host processor to process the interim results for various dense array operations by collecting the interim results to produce the final result. For matrix multiplication operation (and for other such operations), the host processor unpacks the partial or interim results, which have been collected from each processor, in to relevant memory addresses to get the final result. This phase is similar to the data distribution. The result collection phase may have different implementations for different dense array operations. The data parallel algorithm for the result collection phase for multi-dimensional matrix multiplication operation for the EKMR(n) with the row distribution method (the similar algorithm will be used for the column data distribution method) is given below in figure 3.8.

```
Algorithm column_data distribution method_EKMR(n)
1.  for (P_id = 0; P_id < P ;  P_id ++)
2.     l = 0;
3.     offset = P_id ×column _size;      /* row_size = ⌈n^2/P⌉ or ⌊n^2/P⌋ */
4.     for(x = 0; x< n^(n-4);  x++)
5.       for (i =0; i <  n ; i++)
6.         for (j =0; j < column_size; j++)
7.           D[l]  = A_x [i][j + offset] ;        /*Pack array elements into buffer D*/
8.           l++
9.  Distribute D[] to appropriate processor ;
End_of_coloumn_data distribution method_EKMR(n)
```

Figure 3.8

IV. PERFORMANCE COMPARISON BETWEEN TMR AND EKMR SCHEMES

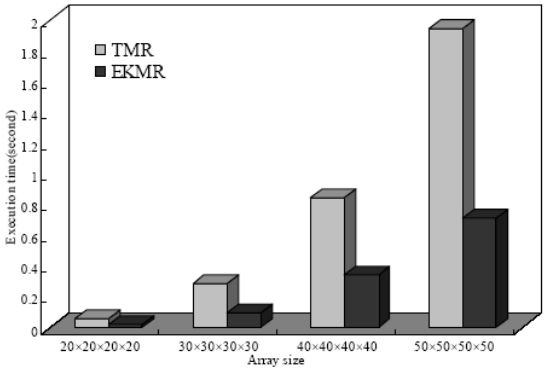

Figure 4.1 shows the time of the data distribution phase of data parallel algorithms with respect to the row data distribution method for TMR(4) and EKMR(4) on 16 processors. It illustrates that EKMR scheme takes less time than TMR scheme for data parallel algorithms in the distribution phase of the row distribution method.

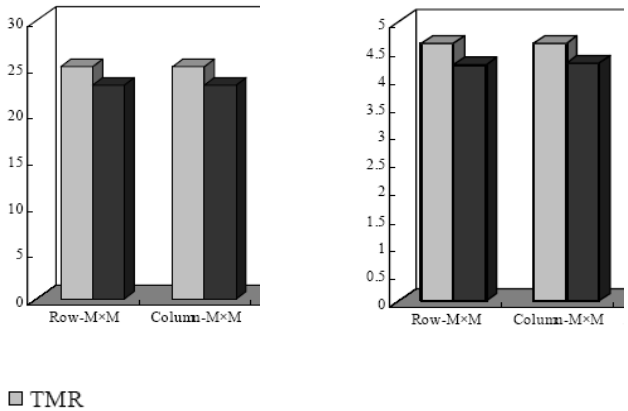

Figure 4.2 (a)Without Compiler Optimization (b)With Compiler Optimization

The figure 4.2 shows the time taken for local computation phase of data parallel algorithms with matrix multiplication by TMR(3) and EKM(3) schemes for array size 200 x 200 x 200 on 16 processors where (a) represents without complier optimization option and (b) represents with complier optimization option.

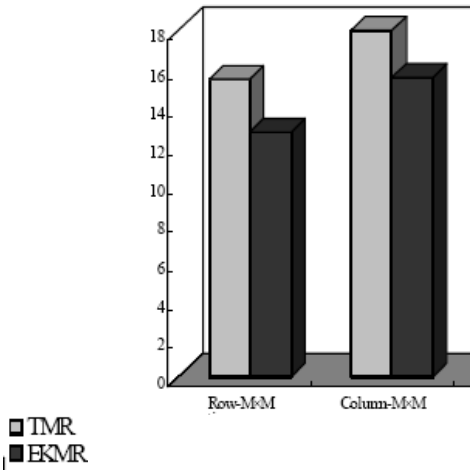

Figure 4.3 shows time taken by the TMR(3) and EKMR(3) for the result collection phase of data parallel algorithms of matrix multiplication of array size 200 x 200 x 200 on 16 processors.

The above performance comparison graphs between TMR and EKMR schemes are taken in the context of this study that is data parallel algorithms for matrix multiplication. The graphs portray that EKMR is a better and efficient scheme than the TMR scheme in all phases due to the lesser values obtained in the EKMR scheme for the following: the number of non-continuous data blocks (with various data distribution methods), the packing time, the cost of the index computation of array elements, the number of cached lines accessed for operations of dense arrays and the unpacking time. This efficiency of EKMR in comparison to TMR is also true for all operations, all array sizes including both dense and sparse arrays [13] and all algorithms.

V. CONCLUSION

In this paper, we first identified and then discussed the issues faced in relation to the efficient operations of the multi-dimensional arrays. It was found that the most of the proposed methods do not perform well for extended form of tensors although these methods show good performance when applied to two-dimensional arrays. We discussed the flaws of the traditional matrix representation (TMR) and then proposed the Extended Karnaugh Map Representation (EKMR) as a new scheme which ruled out the drawbacks of the TMR scheme. EKMR is based on the Karnaugh Map. The basic concept of the EKMR technique is to represent the multi-dimensional array in to the form of a set of two-dimensional arrays. Thus, the extended Karnaugh map representation made it easier to design the efficient data parallel algorithms for multi-dimensional arrays having more than two dimensions. We analyzed the data parallel algorithms for multi-dimensional matrix multiplication using the Karnaugh map that is EKMR and concluded that EKMR is better than TMR in all aspects. The concepts given by O' Boyle to design the loop re-permutation have been applied in this report to design the data parallel algorithms for multi-dimensional array multiplication operation using the EKMR scheme [16]. This report focused on the application of the EKMR on the dense multi-dimensional array, however we have discussed that EKMR is equally effective in case of sparse multi-dimensional arrays.

With the help of the parallel algorithms for multi-dimensional matrix multiplication operation using the Karnaugh map, it was proved that the cost of computing index of elements with EKMR scheme is less than that of TMR scheme and the number of lines cached which the dense array operations have accessed for EKMR scheme is less than that of TMR scheme. These were the flaws of the TMR scheme which previously caused the inefficient performance when the dimensions of the arrays exceeded the value of 2. Thanks to the EKMR scheme which optimized the performance even to the nth dimension of the tensors.

VI. REFERENCES

[1] A.J.C. Bik and H.A.G. Wijshoff, "Compilation Techniques for Sparse Matrix Computations," Proc. ACM Int.l Conf. Supercomputing, 1993, pp. 416-424.

[2] A.J.C. Bik, P.M.W. Knijnenburg, and H.A.G. Wijshoff, "Reshaping Access Patterns for Generating Sparse Codes," Proc. Int.l Workshop Languages and Compilers for Parallel Computing, 1994, pp. 406-420.

[3] A.J.C. Bik and H.A.G Wijshoff, "Automatic Data Structure Selection and Transformation for Sparse Matrix Computations," IEEE Transactions on Parallel and Distributed Systems, vol. 7, no. 2, Feb. 1996, pp. 109-126.

[4] G. Bandera, P.P. Trabado, and E.L. Zapata, "Local Enumeration Techniques for Sparse Algorithms," Proc. IEEE Int.l Symp. Parallel Processing, 1998, pp.52-56.

[5] B.B. Fraguela, R. Doallo, E.L. Zapata, "Modeling Set Associative Caches Behaviour for Irregular Computations," Proc. ACM Int.l Conf. Measurement and Modeling of Computer Systems, 1998, pp.192-201.

[6] B.B. Fraguela, R. Doallo, E.L. Zapata, "Cache Misses Prediction for High Performance Sparse Algorithms," Proc. Int.l Euro-Par Conf., 1998, pp.224-233.

[7] B.B. Fraguela, R. Doallo, E.L. Zapata, "Cache Probabilistic Modeling for Basic Sparse Algebra Kernels Involving Matrices with a Non-Uniform Distribution," Proc. IEEE Euromicro Conf., 1998, pp. 345-348.

[8] B.B. Fraguela, R. Doallo, E.L. Zapata, "Automatic Analytical Modeling for the Estimation of Cache Misses," Proc. Int.l Conf. Parallel Architectures and Compilation Techniques, 1999, pp. 221-231.

[9] B. Kumar, C.H. Huang, R.W. Johnson, and P. Sadayappan, "A Tensor Product Formulation of Strassen's Matrix Multiplication Algorithm with Memory Reduction," Proc. Int.l Symp. Parallel Processing, 1993, pp. 582-588.

[10] T.R. Chung, R.G. Chang, and J.K. Lee, "Sampling and Analytical Techniques for Data Distribution of Parallel Sparse Computation," Pro. SIAM Conf. Parallel Processing for Scientific Computing, 1997.

[11] S. Chatterjee, A.R. Lebeck, P.K. Patnala, and M. Thottethodi, "Recursive Array Layouts and Fast Matrix Multiplication," IEEE Transactions on Parallel and

Distributed Systems, vol. 13, no. 11, Nov. 2002, pp.1105-1123.

[12] G.H. Golub and C.F.V. Loan, Matrix Computations, 2nd Edition, The John Hopkins University Press, Baltimore, Maryland 21218, 1989.

[13] J.B. White and P. Sadayappan, "On Improving the Performance of Sparse Matrix-Vector Multiplication," Proc. Int.l Conf. High-Performance Computing, 1997, pp. 711-725.

[14] C.Y. Lin, J.S. Liu, and Y.C. Chung, "Efficient Representation Scheme for Multi-Dimensional Array Operations," IEEE Transactions on Computers, vol. 51, no. 3, Mar. 2002, pp. 327-345.

[15] C.Y. Lin, Y.C. Chung, and J.S. Liu, "Efficient Data Parallel Algorithms for Multi-Dimensional Array Operations Based on the EKMR Scheme for Distributed Memory Multicomputers," Accepted by IEEE Transactions on Parallel and Distributed Systems, 2003.

[16] M.F.P. O'Boyle and P.M.W. Knijnenburg, "Integrating Loop and Data Transformations for Global Optimization," Proc. Int.l Conf. ParallelArchitectures and Compilation Techniques, 1998, pp. 12-19.