A New Parallel Algorithm for EREW PRAM Matrix Multiplication

S. Vollala[1], K. Geetha[2], A. Joshi[1] and P. Gayathri[3]

[1] Department of Computer Science and Engineering, National Institute of Technology, Tiruchirappalli
[2] SASTRA University, Thanjavore
[3]Dept. of Mathematics, A. V. C. College (Autonomous), Mayiladuthurai, Tamil Nadu, India
{satya4nitt,geethavalavan, amitjoshi233}@gmail.com; pgayathri @avccollege.net

Abstract. This work presents a new parallel matrix multiplication algorithm using an exclusive-read and exclusive-write (EREW), parallel random access memory (PRAM) model for a fixed number of processors. This algorithm used for computing the matrix multiplication analyses the logical pattern that exists for accessing the elements of the matrix efficiently. The proposed algorithm PEMM (Parallel EREW algorithm for Matrix Multiplication) works with the time complexity of O(n), but takes less number of iterations when compared with the existing work. It can also run on parallel machines other than the EREW PRAM.

Keywords: ***EREW; PRAM; Parallel Algorithms; Matrix Multiplication; Theoretical Time Complexity.***

1 Introduction

The implementation of massive parallel machines are now a days easily facilitated by recent advancements in hardware technology. These machines make use of multi-dimensional arrays to solve scientific and engineering problems. In massively parallel systems parallelism can be exploited by performing concurrent operations. Though CREW access is a desirable property in parallel machines, the performance improvement can be easily attained by mapping the code to an underlying EREW machines [1]. There are several well-known, different fast parallel algorithms for matrix-matrix multiplication, but most of them use the CRCW (Concurrent Read Concurrent Write) model. The parallel computing model used in this work is a synchronous fine-grained shared-memory model where all the processors can read and write into a common memory, provided, there are no read and write conflicts. This means all reads and writes should happen exclusively. No memory location is allowed to access simultaneously by more than one processor at the same time. This model is known as the exclusive-read exclusive-write parallel random access memory (EREW PRAM).
Parallel computers can be modeled by means of parallel random access machine. It consists of sequential processors that will share a global memory. Each processor can also access its private memory. Communication and synchronization issues are captured property to promote parallel computing [2] [3]. PRAM has four variants identified based on the type of access mode to the shared memory cell. The Read and Write operations can be done concurrently or exclusively. Hence PRAM is classified as EREW, ERCW, CREW and CRCW. Here C represents concurrent operation, E represents exclusive operation, R and W represents read and write operations as shown in Fig. 1. CRCW is a strongest model, whereas EREW is the weakest model, but more realistic.

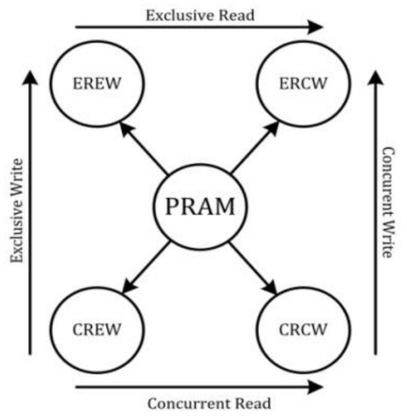


Fig. 1. PRAM Models

2 Related Works

Related Works: This Section discusses the performance of matrix multiplications implemented using MESH algorithm mentioned in [4] and CRCW multiplication cited in [2].

2.1 Mesh Algorithm or Matrix Multiplication

Mesh multiplication algorithm uses n^2 processors in mesh configuration to multiply two n x n square matrices. The two matrices are accessed starting from the boundary elements and linearly progresses in row wise for first matrix and column wise for the second matrix. This process is repeated until all product terms have been calculated by accessing the elements in both the matrices. Finally the result is updated by each processor in the corresponding location in result matrix simultaneously. Algorithm 1 shows the steps involved in MESH matrix multiplication [4]. Fig. 2 shows the illustration of MESH Matrix Multiplication Algorithm.

Algorithm 1 MESH_MATRIX_MULTIPLICATION

```
1.  $\textbf{for}\ (i = 1\ to\ m)\ \textbf{do}\ in\ parallel$
2.    $\textbf{for}\ (j = 1\ to\ k)\ \textbf{do}\ in\ parallel$
3.      $C_{ij} \leftarrow 0;$
4.      $\textbf{while}(P_{ij}\ receives\ two\ inputs\ a\ and\ b)\textbf{do}$
5.        $C_{ij} \leftarrow C_{ij} + a \times b;$
6.        $\textbf{if}(\ i < j)\textbf{then}$
7.          $Send\ b\ to\ P_{i+1j};$
8.        $\textbf{end if}$
9.        $\textbf{if}(\ j < k)\textbf{then}$
10.         $Send\ a\ to\ P_{i+1j};$
11.       $\textbf{end if}$
12.     $\textbf{end while}$
13.   $\textbf{end for}$
14. $\textbf{end for}$
```

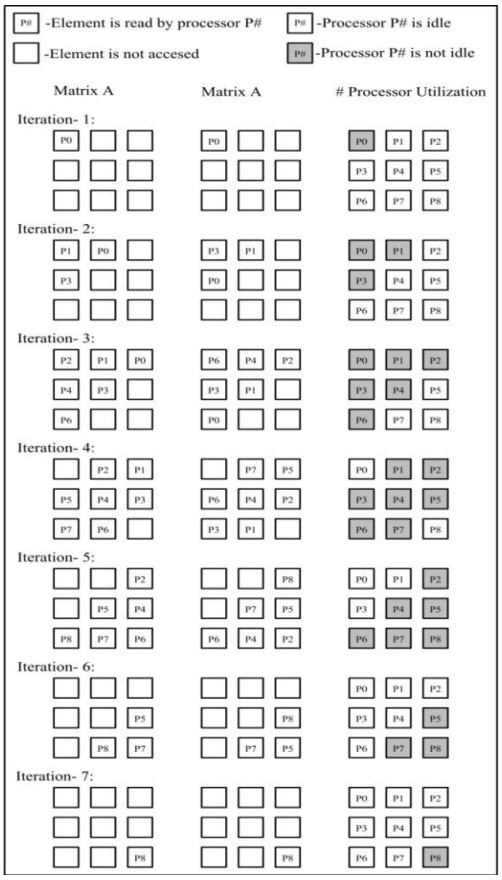


Fig. 2. An illustration of MESH Matrix Multiplication Algorithm

2.2 CRCW matrix multiplication

Matrix multiplication that can be performed in parallel approach has been designed to run SM SIMD computer on CRCW model. Write conflicts will arise when several processors tries to access the same memory location to perform write operation. Such conflicts can be resolved by storing the sum of numbers in that location. Sequential procedure adapted in matrix multiplication algorithm can be directly parallelized by use of $m \times n \times k$ processors to find the product of two matrices A, B of the order of $m \times n$ and $(n \times k)$. The multiplicand and multiplier matrices referred as A and B can be stored in a shared memory and after finding the product, the resultant matrix can be obtained in the same shared memory location [5]. The sequence steps involved in CRCW multiplication [4] is presented in Algorithm 2.

Algorithm 2 CRCW_MATRIX_MULTIPLICATION

```
1.  for (i = 1 to m) do in parallel
2.     for (j = 1 to k) do in parallel
3.        for (s = 1 to n) do in parallel
4.            $C_{ij} \leftarrow 0;$
5.            $C_{ij} \leftarrow C_{ij} + a_{is} \times b_{sj};$
6.        end for
7.     end for
8.  end for
```

2.3 Recent techniques for Matrix Multiplication

S K Das el al. devised new optimal algorithms for solving several problems on graph theory based on EREW PRAM model [6]. The parallel algorithms for finding the spanning forest and computing the connected components have been designed. They presented all the algorithms based on divide-and-conquer strategy. With each of the proposed algorithms they have achieved competent speed-up, scalability up to a certain number of processors. They have also analyzed the time complexity and derived the lower bound on the processor (time)^2. The same authors S K Das and Narsing Deo have given a short notes on Optimal parallel algorithms for solving some problems with EREW PRAM model based on divide-and-conquer [7]. Jong-Chuang Tsay and Sy Yuan designed combinatorial aspects of parallel algorithms for matrix multiplication [8]. Specially a systematic design of cylindrical array for matrix multiplication have been presented. Keqin Li et al. presented a brief material on parallel matrix multiplication with a reconfigurable pipelined bus system [9]. They claimed that for multiplication of 2 matrices of order $N \times N$ can be done in $O(\frac{N^\alpha}{p} + \frac{N^2}{p^{2/\alpha}} \log p)$ time. They have used p-processor linear array with a reconfigurable pipelined bus system (LARPBS), where $1 \leq p \leq N$. They have also proved that the multiplication of two $N \times N$ matrices took only $O(\log N)$ time. C Yang et al. presented a processor architecture for optical vector matrix multiplication [10]. The main aim of authors is to optimize the data flow of vector matrix multiplication. A J Nicholas et al. proposed a new parallelization technique to compute the sparse matrix-vector multiplication in [11]. They have also discussed the existing techniques in detailed manner. The results were experimented on NUMA architecture and claimed that they have achieved 90% parallel efficiency for best case. Soydan R et al. presented a reconfigurable architecture to find the polynomial matrix multiplication by making use of convolution technique [12]. As per the literature, it is the first hardware architecture for computing polynomial matrix multiplication. They have achieved higher accuracy, less execution time and compatibility with the help of pipeline techniques. L. Yavits et al. achieved the higher parallelism to implement sparse matrix multiplication with an associative processor [13]. The authors have claimed that the time complexity is (nnz), where nnz is the number of non-zero entries. M O Karsavuran et al. exploited thread-level parallelization to evaluate the sparse matrix multiplication [14].

3 Proposed Parallel EREW Matrix Multiplication

One new algorithm has been presented here for matrix multiplication on EREW PRAM machine. This algorithm uses n^2 processors and requires $O(n)$ time. This algorithm has been compared with another EREW PRAM algorithm called MESH multiplication [4]. The MESH algorithm also uses n2 processors and requires O(n) time, but the proposed algorithm is three times faster. Both the algorithms work with square as well as rectangular matrices and requires n^2 processors.

The proposed PEMM algorithm uses the memory efficiently by performing memory access in parallel, thereby utilizing the processors effectively to enhance the overall speedup. MESH algorithm performs only sequential access to memory thereby ending up in poor utilization of recourses. Unlike the mesh algorithm, every processor in this model accesses two distinct elements each from the two matrices, in one iteration. All n^2 processors access the n^2 elements simultaneously from both the matrices. The algorithm exploits the parallelism that exists in the accessing pattern of the elements. The sequence of steps involved in computing matrix multiplication by PEMM are given in Algorithm 3, Fig. 3 shows an illustration of PEMM Algorithm.

Algorithm 3 PARALLEL_EREW_MATRIX_MULTIPLICATION

```
1.  $\boldsymbol{for}\ (i = 1\ to\ m)\ \boldsymbol{do}\ in\ parallel$
2.    $\boldsymbol{for}\ (j = 1\ to\ k)\ \boldsymbol{do}\ in\ parallel$
3.      $C_{ij} \leftarrow 0;$
4.      $\boldsymbol{for}\ (s = 1\ to\ n)\ \boldsymbol{do}\ in\ parallel$
5.        $read\ a\ from\ A_{i((j+1) mod\ n)};$
6.        $read\ b\ from\ B_{((i+1) mod\ n)j}$
7.        $C_{ij} \leftarrow C_{ij} + a \times b;$
8.      $\boldsymbol{end\ for}$
9.    $\boldsymbol{end\ for}$
10. $\boldsymbol{end\ for}$
```

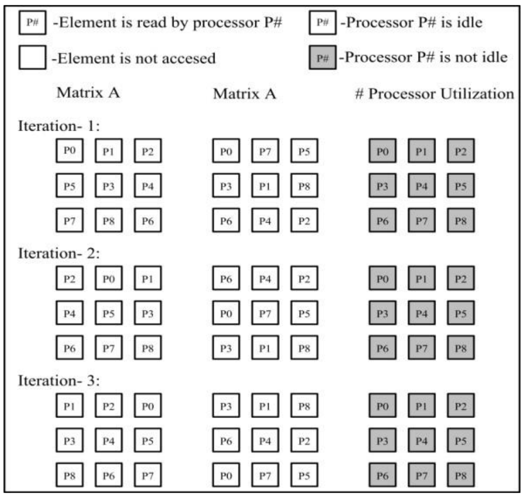


Fig. 3. An illustration of PEMM Algorithm

4 Time Complexity

Both the algorithms shown in Algorithm 1 and 2 requires m x k processors and multiplies two matrices $A(m \times n), B(n \times k)$ and writes the result into third matrix of the order $C(m \times k)$. MESH matrix multiplication takes $m + k + n - 2$ steps where the same operation is performed in just n steps by the PEMM algorithm. Assuming $m = k = n$, the time complexities are $O(3n - 2)$ *and* $O(n)$ respectively. Theoretical time complexity is $O(n)$ and it is same for both algorithms. But the efficient memory accessing pattern in proposed algorithm gives 100% processor utilization of the EREW model, where it is only 33% in MESH multiplication. This gives a theoretical speed up of three times faster than mesh multiplication.

5 Conclusion

For simulating the algorithms, it is implemented with n^2 threads in OpenMP, where each thread acts as a standalone processor. Synchronization constructs [15] are used to simulate the EREW logic, where on each iteration only one processor accesses any of one element each from both matrices. And it proceeds with next iteration when all the processors has completed the current iteration, which is done with the help of synchronization construct # pragma omp barrier [15]. The implementation of the both the algorithms, namely MESH and PEMM using synchronization constructs in OpenMP is presented in Appendix. A clear illustration of accessing the elements from both the matrices and processor utilization is shown in Fig. 6 and Fig. 7, where both matrices are of size 3 x 3.follow the given guidelines.

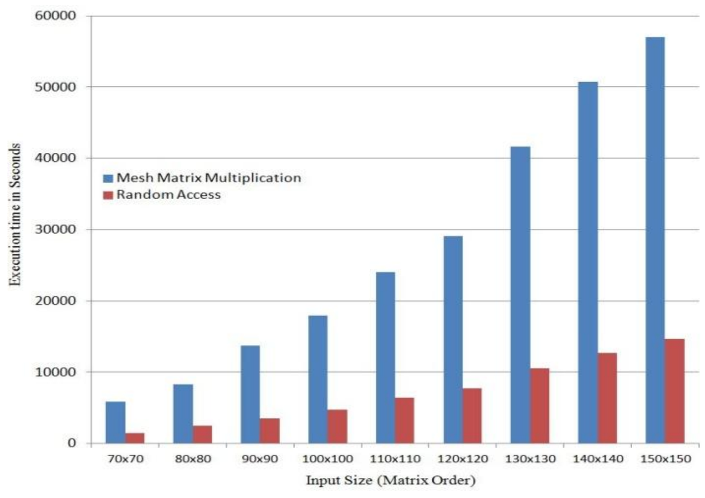


Fig. 4. Execution time for MESH and parallel EREW algorithms

Table 1. Performance comparison between MESH algorithm and proposed PEMM algorithm

Algorithm(→) / Matrix Size(↓)	Random Access	MESH Multiplication	Speed up (# of times)
10 x 10	5.33334	15.6667	2.94
20 x 20	36.33330	120.0000	3.30
30 x30	140.66700	411.6670	2.93
40 x 40	312.33300	1036.6700	3.32
50 x 50	661.33300	2401.9100	3.63
60 x 60	932.33300	3541.6700	3.80
70 x 70	1494.67000	5896.3300	3.94
80 x 80	2526.00000	8328.3300	3.30
90 x 90	3484.67000	13688.0000	3.93
100 x 100	4739.67000	17949.0000	3.79
110 x 110	6385.67000	24037.7000	3.76
120 x 120	7772.00000	29105.7000	3.75
130 x 130	10511.00000	41684.3000	3.97
140 x 140	12735.00000	50695.0000	3.98
150 x 150	14662.30000	56982.0000	3.89

6 Performance Comparison

Figure 5 and 6 shows the performance comparison between MESH algorithm and proposed PEMM algorithm in terms of execution time and speed up achieved for square matrices of size varying from 10x10 to 150x150 as given in Table 1. From the Figure # it can be inferred that there is a minor difference in execution time up to 50x50 and thereafter the execution time exponentially grows for matrix sizes beyond 50x50 for MESH in comparison with PEMM. Though the theoretical time complexity (Time complexity in terms of asymptotical notations) of both algorithms seems to be same, the number of iterations in matrix multiplication is reduced by $(m + k - 2)$ thus resulting in a speed up factor of 3 times as depicted in Fig. 5. The comparison between MESH and parallel EREW algorithms in terms of execution time is shown in Fig. 4.

7 Conclusion

The parallel EREW matrix multiplication algorithm is faster compared to the parallel MESH algorithm, and it also maintains constant speed up over a wide range of inputs. Since all the processors are utilized effectively (100%), it is possible to achieve speedup in the range of 3 to 4 times. Since, the proposed algorithm uses n2 processors effectively, the same can set a good bench mark to work in multi-core platform, exploiting the core utilization in an effective manner. This enhancement made can achieve better performance gain for image processing, graphics and animation related applications.

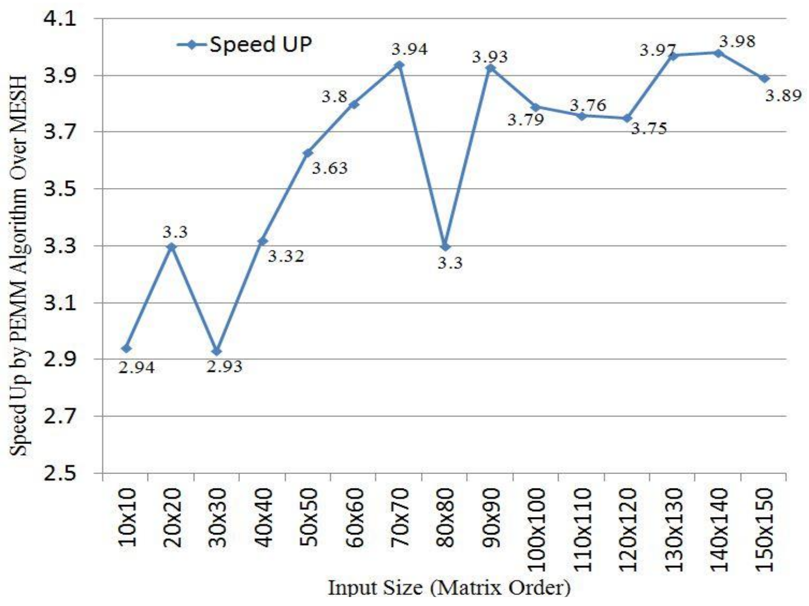


Fig. 5. Speed up over MESH multiplication algorithm

References

[1] Daniel Nussbaum and Anant Agarwal. Scalability of parallel machines. Communications of the ACM, 34(3):57-61, 1991.

[2] Rajeev Motwani and Prabhakar Raghavan. Randomized algorithms. ACM Computing Surveys (CSUR), 28(1):33-37, 1996.

[3] Rajeev Motwani and Prabhakar Raghavan. Randomized algorithms. Chapman & Hall/CRC, 2010.

[4] Selim G Aki. The design and analysis of parallel algorithms. 1989.

[5] Justin R Smith and Ali Smith. The design and analysis of parallel algorithms, volume 1090. Oxford University Press New York, 1993.

[6] Sajal K Das and Narsingh Deo. Divide-and-conquer-based optimal parallel algorithms for some graph problems on erew pram model. IEEE transactions on circuits and systems, 35(3):312-322, 1988.

[7] SK Das and N Deo. Notes ondivide-and-conquer-based optimal parallel algorithms for some graph problems on erew pram model'. IEEE transactions on circuits and systems, 37(7):962-965, 1990.

[8] Jong-Chuang Tsay and Sy Yuan. Some combinatorial aspects of parallel algorithm design for matrix multiplication. IEEE transactions on computers, 41(3):355-361, 1992.

[9] Keqin Li and Victor Y Pan. Parallel matrix multiplication on a linear array with a recon_gurable pipelined bus system. IEEE Transactions on Computers, 50(5):519-525, 2001.

[10] C Yang, GX Cui, YY Huang, L Wu, H Yang, and YH Zhang. Performance of an embedded optical vector matrix multiplication processor architecture. IET optoelectronics, 4(4):159, 2010.

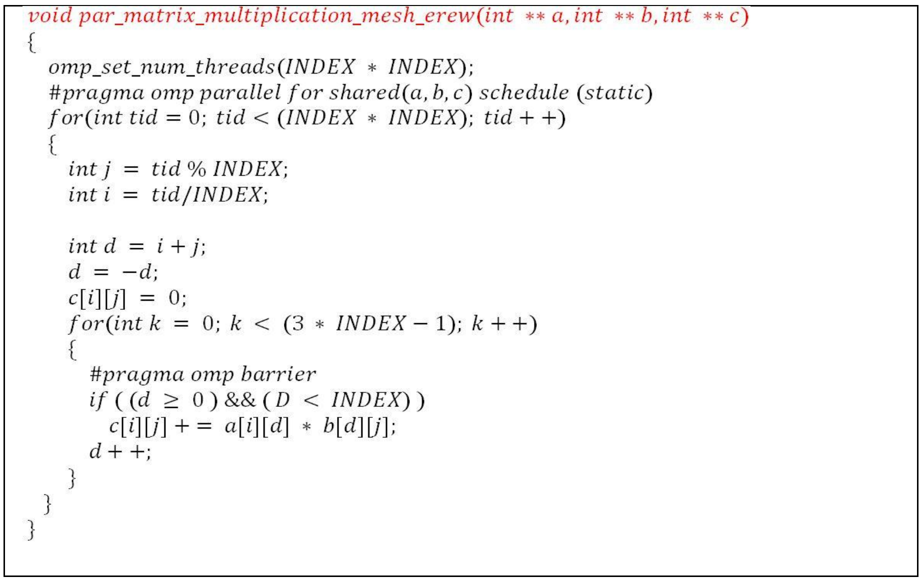
[11] Albert-Jan Yzelman and Dirk Roose. High-level strategies for parallel shared memory sparse matrix-vector multiplication. 2012.

[12] Soydan Redif and Server Kasap. Novel recon_gurable hardware architecture for polynomial matrix multiplications. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 23(3):454-465, 2015.

[13] Leonid Yavits, Amir Morad, and Ran Ginosar. Sparse matrix multiplication on an associative processor. IEEE Transactions on Parallel and Distributed Systems, 26(11):3175-3183, 2015.

[14] M Ozan Karsavuran, Kadir Akbudak, and Cevdet Aykanat. Locality-aware parallel sparse matrix-vector and matrix-transpose-vector multiplication on many-core processors. IEEE Transactions on Parallel and Distributed Systems, 27(6):1713-1726, 2016.
[15] ARB OpenMP. Openmp application program interface, v. 3.0. OpenMP Architecture Review Board, 2008.

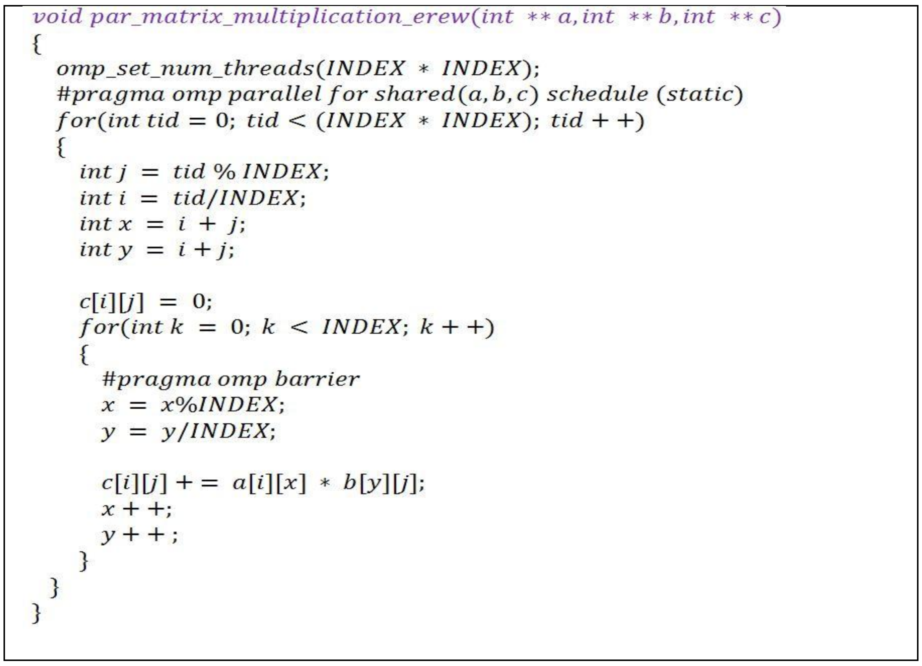
Appendix



```
void par_matrix_multiplication_mesh_erew(int ** a, int ** b, int ** c)
{
  omp_set_num_threads(INDEX * INDEX);
  #pragma omp parallel for shared(a, b, c) schedule (static)
  for(int tid = 0; tid < (INDEX * INDEX); tid + +)
  {
    int j = tid % INDEX;
    int i = tid/INDEX;

    int d = i + j;
    d = -d;
    c[i][j] = 0;
    for(int k = 0; k < (3 * INDEX - 1); k + +)
    {
      #pragma omp barrier
      if ( (d ≥ 0 ) && ( D < INDEX) )
        c[i][j] + = a[i][d] * b[d][j];
      d + +;
    }
  }
}
```

Fig. 6. Sample code for implementation of MESH matrix multiplication



```
void par_matrix_multiplication_erew(int ** a, int ** b, int ** c)
{
  omp_set_num_threads(INDEX * INDEX);
  #pragma omp parallel for shared(a, b, c) schedule (static)
  for(int tid = 0; tid < (INDEX * INDEX); tid + +)
  {
    int j = tid % INDEX;
    int i = tid/INDEX;
    int x = i + j;
    int y = i + j;

    c[i][j] = 0;
    for(int k = 0; k < INDEX; k + +)
    {
      #pragma omp barrier
      x = x%INDEX;
      y = y/INDEX;

      c[i][j] + = a[i][x] * b[y][j];
      x + +;
      y + + ;
    }
  }
}
```

Fig. 7. Sample code for implementation of PEMM matrix multiplication