International Journal of Early Childhood Special Education (INT-JECS)
DOI: 10.48047/INTJECSE/V14I7.291 ISSN: 1308-5581 Vol 14, Issue 07 2022

An Efficient Algorithm to multiply 'n' matrices on two-dimensional Mesh Network Parallel Architecture

Dr. Shanthi Makka
Professor, Department of Computer Science and Engineering
Vardhaman College of Engineering, Hyderabad, Telangana-501218, India
dr.shanthimakka@gmail.com
www.vardhaman.org
Dr. Gagandeep Arora
Professor and HOD, Department of AI and ML
Vardhaman College of Engineering, Hyderabad, Telangana-501218
gagandeeparora250379@gmail.com
www.vardhaman.org
Dr. S.Bharath Reddy
Associate professor, Department of AI and ML
Vardhaman College Of Engineering, Hyderabad, Telangana-501218
bharath.bittu945 @gmail.com
www.vardhaman.org

Abstract: In mathematics, matrix multiplication or matrix product is a binary operation that produces resultant matrix after multiplying two matrices concurrently and matrix multiplication is implemented for composition of linear maps, graphics-Graphic software such as Adobe Photoshop for rendering of images, Cryptography, wireless communication, Economics, for finding area of triangle, collinear points, for finding Solution of Linear Equations, Records, Engineering, Physics and machine learning. Multiple numbers of matrices along with dimensions are given, before multiplying we need to find an optimal sequence of matrices i.e., the order of matrices with a minimum number of multiplications, after that, we can proceed with multiplication process. Deciding of optimal order minimizes time complexity. Matrix multiplication follows the associative law, and if we alter the sequence of the matrices, it does not affect the final result. However, the cost of matrix multiplication greatly depends on the order in which the matrices are multiplied. The sequence which we follw to multiply 'n' number of matrices saves computational time. First we find out Optimal Sequence using Dynmaic approach and then we construct a complete binary tree for given 'n' number of matrices, then multiply chain of two matrices using a two-dimensional mesh network on a SIMD model having wrap-around connections, then designed an algorithm to multiply nxn matrices using "n^2" number of processors. The effectiveness of this approach is demonstrated with comparative study with other approaches.

Keywords: Two-dimensional mesh network, Dynamic Programming, Optimal Sequence, Parallelism, High performance, SIMD model

1. Introduction

Matrix multiplication plays vital role in day-to-day life. More precisely, if A is an $n \times m$ matrix and B is an $m \times p$ matrix, their matrix product AB is an $n \times p$ matrix, in which the m entries across a row of A are multiplied with the m entries down a column of B and summed to produce an entry of AB. When matrices represent two linear maps, then the matrix product represents the composition of the two maps. In geology, matrices are used for making seismic surveys. They are used for plotting graphs, statistics and also to do scientific studies and research in almost different fields. Matrices are also used in representing the real world data's like the population of people, infant mortality rate, etc. Multiplying 'n' number of matrices takes "n^4" amount of time that is very huge if you deal with larger amount of data and one should think of using parallelism i.e., by performing task parallel with advent availability of [1] parallel architectures, which are more economical nowadays. In the implementation of genetic algorithm using binary code as a matrix [17] for mobile robot navigation in both static and dynamic environment, basically path for navigation is implemented using trace theory as an optimum controller.

SIMD Model is a Single Instruction stream-Multiple Data stream parallel architecture [16] manifests excellent performance in a numerous application fields such as vision, scientific computing, geometric modeling, and artificial intelligence mainly because of the massive data parallelism. A detailed study and survey has been done on Parallel Algorithms and Architectures [4] for image processing with SIMD computers with without sharing of memory. The Parallel Architectures [3] can be categorized in to three types: mesh-connected

computers, pyramid computers, and hypercube and related computers. The mesh-connected network is array processor [5] has equal number of processing elements as number of pixels in an image.

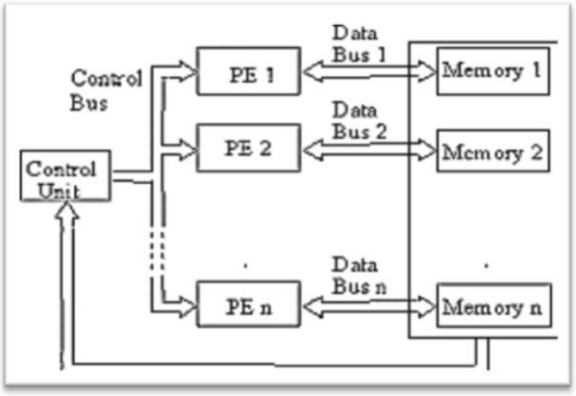


Fig1. Architecture of SIMD

It is a variant of Instructional level Parallelism and here same instruction is executed on multiple data sets to produce the output and no processor runs the same instruction in one clock cycle. Communication network permits synchronous communication among several memory modules.

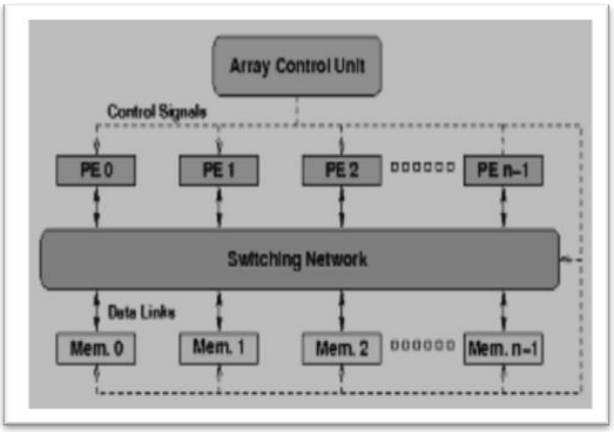


Fig 2. SIMD parallel Architecture

While a program is executed in SIMD Parallel Architecture, always we need a mask of a Program Element (PE) for doing processing along with autonomous control within PE. A mask bit within PE can be masked while processing of an instruction and mask bit in PE gets reset when PE receives instruction from control unit as no operation. Every PE has index registers, which are connected to global address space afford by CU Instruction and to reinforce data and address manipulations, the arithmetic logic unit has general-purpose registers and also contains pointer registers for additional support. In this paper we have used mesh-connected architecture, each node in this architecture uses four ports such as left, right, top and bottom ports. The instructions set in CU with PEs executes few instructions, which are fixed with P to stipulate that all instructions should execute in parallel. PEs has four bidirectional ports to communicate with four neighbors.

2. Matrix Chain Multiplication (MCM) Using Dynamic Programming

In dynamic programming approach, we divide given problem into smaller parts, find solutions for smaller modules, and store those solutions into a table and can be used while solving higher level modules and bottom up approach is used to establish solution for complex task.

2.1 Optimal Parenthesization

The Optimal Parenthesization for the given set of matrices can be determined by placing the parenthesis in all possible places, find out the cost of each placement and then return the minimum value. In a chain of ‘n’ matrices we can have set of parenthesis in ‘n-1’ possibilities. Let us consider an example of three matrices A1, A2, and A3 with dimensions 10X100, 100X50, and 50X20. If we multiply A1 and A2 first, then multiply with A3 leads to 60,000 multiplications. Another alternative is that multiply A2 and A3 together, then multiply with A1 leads to 1,20,000 multiplications. The number of multiplications is just double. That is the reason if we have an approach to deciding an optimal sequence of matrices before multiplications minimize the maximum amount of time.

2.1.1 Sequential Approach

```
Algorithm MATRIX-CHAIN-MULTIPLICATION (A)
{
//A is an array of dimensions of matrices
nm=length (A)-1
for (p=1 to nm) do
mul [p,p]=0
for(len=2 to nm) do
{ for p=1 to nm-len+1 do
        {       q=p+len-1
        mul[p,q]=∞
        for(r=p to q-1) do
        {       temp=mul[p,r]+mul[r+1,q]+A[p-1]*A[r]*A[q]
        if(temp<mul[p,q]) then
        {
                mul[i,j]=temp
                sep[p,q]=r
        } }}}
return mul, sep
}
PRINT-OPTIMAL-PARENTHESIS (sep, p, q)
{
If(p=q) then
        write 'Ai'
else
{
write '('
PRINT-OPTIMAL-PARENTHESIS (sep, p, sep (p, q))
PRINT-OPTIMAL-PARENTHESIS (sep, sep (p, q)+1,q)
write ')'
} }
```

2.1.2 Time Complexity: There is a nesting of three loops, one is for the length of matrices of the multiplications, the second loop is for row index, and column index cab calculated form the row index and length of the chain. The third loop is for 'r' i.e., numbers of alternatives where the matrix sequence can be get divided, and by going through all the possible values we consider the value of 'r', which gives the minimum number of multiplications. If 'n' is the total number of matrices, then the total time required to find the optimal sequence of matrices is O (n^3).

T (n)= O (n^3)

Sequential performance has been analyzed in Fig 3.

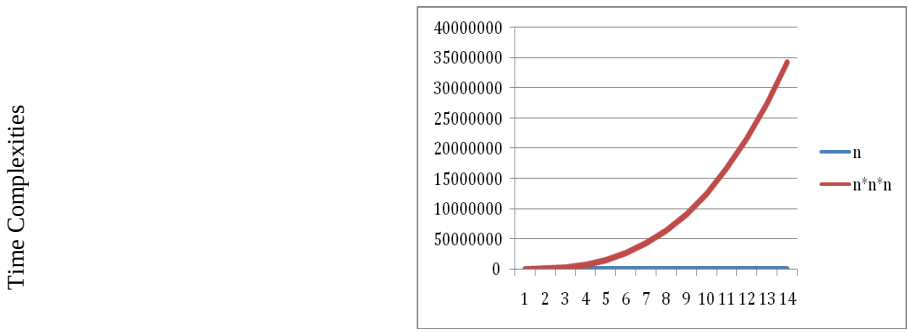


Fig 3. Sequential Analysis of MCM

2.2 Analysis by using Parallel Approach

The first polynomial time algorithm for the matrix chain product problem was got solved by Godbole [7] and it runs O (n^3) amount of time. Later Hu and Shing [8] gave another sequential algorithm, which runs in O (nlogn) time. This algorithm was later proven to be optimal [9]. Unfortunately, all of these algorithms are highly sequential in computation.

The first algorithm or an approach based on parallelism [2] for this problem is based on dynamic programming. Using Dynamic Programming, Valiant et al. [10] implemented an algorithm which runs in O ($\log^2 n$) time using n^9 processors and the time complexity got improved by Rytter [11] to n^6 / log n processors on a CREW PRAM. Huang et al. [12] and Galil and Park [13] modified Rytter's algorithm and further, they reduced the number of processors to $n^6/\log^5 n$ and $n^6/\log^6 n$ respectively. The most recent research on the parallel algorithms of these problems is based on Hu and Shing's sequential algorithm. A. Czumaj [14] and P. Ramanan [15] gave algorithms that run in O ($\log^3$ n) time using O ($n^2/ \log^3 n$) processors and O ($\log^4 n$) time using n processors, respectively.
The pictorial representations of the above facts are being plotted in Fig 4.

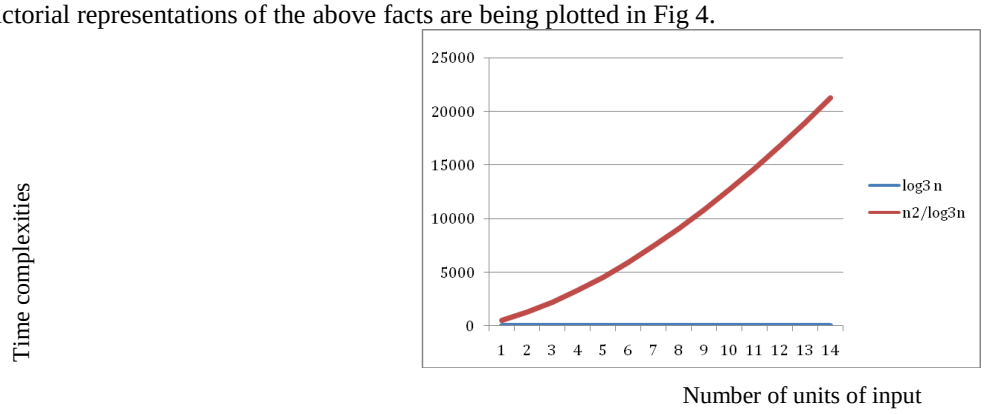


Fig 4. Parallel analysis of MCM using $n^2/\log^3 n$ processors

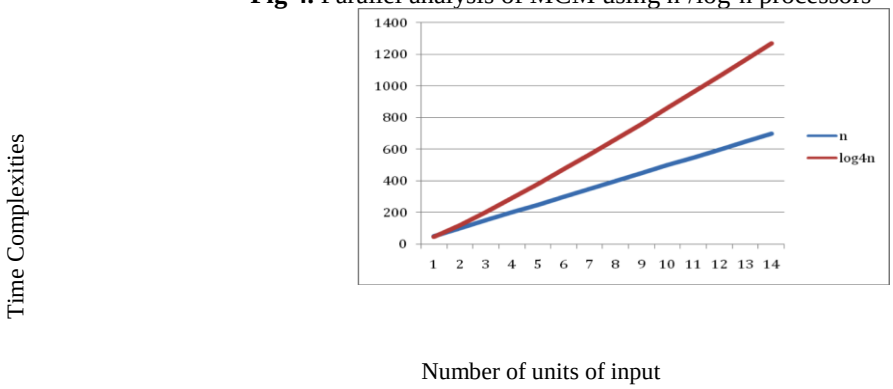


Fig 5. Parallel analysis of MCM using n processors

Sequential versus parallel approaches compared graphically in Fig 5. The blue color line represents sequential performance, red and green colored lines represents parallel approaches, which are discussed above paragraph.

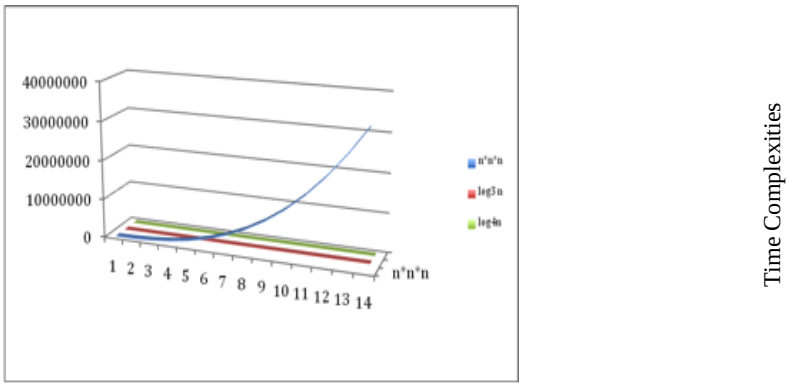


Fig 6. Comparative analysis of sequential with parallel analysis

DOI: 10.48047/INTJECSE/V14I7.291 ISSN: 1308-5581 Vol 14, Issue 07 2022

3. A new approach for solving 'n' number of Matrix Multiplication using parallel architecture

Consider an example of 'n' matrices A1*A2*A3*A4*A5……*An.

```
Algorithm Matrix Multiplication ()
{
//  1. Construct a complete binary tree
```

(i) Initially check whether the tree is empty or not. If it empty, then make new node as root. Else, delete front node of the queue and if the left node does not exist, then make new node as a left child. If left child exists, then make new node as a right child.

(ii) If deleted node has both left and right, then Deque () it, otherwise Enqueue () the new node.

```
SIZE =20     //initialization of capacity of the queue
// A structure definition for a tree node
struct node
{
    any data typeinfo;
   struct node *r,*l;
};
// For creation of a queue node
struct Queue
{
   struct node* *arr;
};
// structure to create a new node
struct node* new (info)
{    struct node* t = (struct node*) malloc (sizeof ( struct node ));
   t->info = data;
   t->l = t->r = NULL;
   return t;
}
// sub-routine to creation of a Queue
struct Queue* createQ(intsize)
{struct Queue* q = (struct Queue*) malloc(sizeof( struct Queue ));
   q->fr = q->re = -1;
   q->size = size;
   q->arr = (struct node**) malloc(q->size * sizeof( struct node* ));
   for i = 0 to size-1step by 1 do
      q->arr[i] = NULL;
   return q;
}
 //Algorithm to check whether queue is empty or full
Algorithm QEmpty (struct Queue* q)
{
   returnq->fr == -1;}
Algorithm QFull (struct Queue* q)
{
returnq->re == q->size - 1;
 }
 //If queue has single element
Algorithm single element (struct Queue* q)
{
Return q->fr == q->re;
  }
// Insert an element in to a queue
Algorithm Enqueue (struct node *root, struct Queue* q)
{
   if(QFull(q)) then
      return;
   q->arr[++q->re] = root;
   if(QEmpty(q)) then
      ++q->fr;
}
```

DOI: 10.48047/INTJECSE/V14I7.291 ISSN: 1308-5581 Vol 14, Issue 07 2022

```
struct node* Dequeue(struct Queue* q)
{
   if(QEmpty(q)) then
      return NULL;
   struct node* t = q->a[q->fr];
   if(singleelement(q)) then
      q->fr= q->re = -1;
   else
      ++q->fr;
   return t;
}
struct node* front(struct Queue* q)
{
return q->arr[q->fr];
 }
// Is tree has both the children
Algorithm twochildren (struct node* t)
{
   return t && t->le && t->ri;
}
// Insertion of a new node
Algorithm insert (struct node **root, int info, struct Queue* q)
{
   struct node *t = new(data);
   // If the tree is empty, make new node as root
   if(!*root) then
      *root = t;
   else
   {
      // go to the front of the queue.
      struct node* fr = front(q);
      // If left child does not exist, then make new node as left child to front node
      if(!fr->l) then
         fr->l = t;
       // If right child does not exist, then make new node as right child to front node
      else if(!fr->r) then
         fr->r = t;
      // If both children exist
      // Dequeue() it.
      if(twochildren(fr)) then
         Dequeue(q);
   }
   // Insert new node into the queue
   Enqueue(t, q);
}
// Complete binary tree
Algorithm binary tree (struct node* root)
{
   struct Queue* q = createQ(SIZE);
   Enqueue(root, q);
   while(!QEmpty(q)) do
   {
      struct node* t = Dequeue(q);
      write ( t->d);
      if(t->l) then
         Enqueue(t->l, q);
      if(t->r) then
         Enqueue(t->r, q);
   }
}
```

```
// Creation of tree starts here
Algorithm tree creation ()
{
   struct node* root = NULL;
   struct Queue* q = createQ(SIZE);
   for i = 1 to 12 step by 1 do
      insert(&root, i, q);
   binarytree(root);
}}
```

//2. Multiply matrices in bottom up fashion using n^2 processors to multiply every two adjacent matrices and number of steps are logn

```
Algorithm Matrix multiplication(   )
{
for h= 1 to logn step by 1do
{
```

//Multiplication of matrices A_iXA_{i+1}, $A_{i+2}XA_{i+3}$,……..$A_{n-1}XA_n$

```
for z= 1 to n-1 do
{
for all P(x,y) do
{
for x= 1 to n do
{
for y=1 to n do
{
if (x>z) then
rotate  matrix Ai in left direction
if (y>z) then
rotate  matrix Ai+1 in upward direction
} } }}

//compute the product of Ai and Ai+1
//store it in c
for z=1 to n-1 do
{
for all P(x,y) do
{
for x= 1 to n do
{
for y=1 to n do
{
rotate Ai in left direction
rotate Ai+1 in the upward direction
c=c+Ai*Ai+1
} }}
}
Algorithm Parallel_Multiplication(    )
{
call tree creation()
call Matrix multiplication(   ) //using cross wired mesh network using n^2 processors for every two matrices
}
```

Matrix Multiplication Using Mesh Network

Consider a two-dimensional mesh network on a SIMD model having wrap-around connections shown in Fig 7. Designed an algorithm to multiply nXn matrices using"n^2" number of processors.

Arrange the matrices Ai and Ai+1 in such a way that every processor pij has a pair of elements for multiplication. The elements of matrix Ai will move in a left direction and the elements of matrix Ai+1 will move in an upward direction. All these alternations of elements of Ai and Ai+1 stores in processing element 'p', which leads to a new pair for further multiplication. First, we stagger two matrices Ai and Ai+1. Secondly find multiplication of Ai [i, k] with Ai+1[k, j], and then we calculate the summation of the entries once the second step is completed.

Mesh Network is a topology that makes a collection of nodes to form a grid called mesh topology and all the

edges are parallel to the axis of the grid and also adjacent nodes communicate to each other through a network.

Total number of nodes in a network = (total number of nodes in a row) × (total number of nodes in a column)

The evaluation of mesh network can be by factors one is Diameter and second is Bisection width. A p-dimensional mesh network having k*p nodes has a diameter of p*(k–1). Bisection width is the minimum number of edges required from network to divide network into equal parts.

Ai+1[4,1]	Ai [1,4]	Ai [2,4]	Ai+1[4,2]	Ai+1[4,3]	Ai [3,4]	Ai [4,4]	Ai+1[4,4]
Ai+1[3,1]	Ai [1,3]	Ai [2,3]	Ai+1[3,2]	Ai+1[3,3]	Ai [3,3]	Ai [4,3]	Ai+1[3,4]
Ai+1[2,1]	Ai [1,2]	Ai [2,2]	Ai+1[2,2]	Ai+1[2,3]	Ai [3,2]	Ai [4,2]	Ai+1[2,4]
Ai+1[1,1]	Ai [1,1]	Ai [2,1]	Ai+1[1,2]	Ai+1[1,3]	Ai [3,1]	Ai [4,1]	Ai+1[1,4]

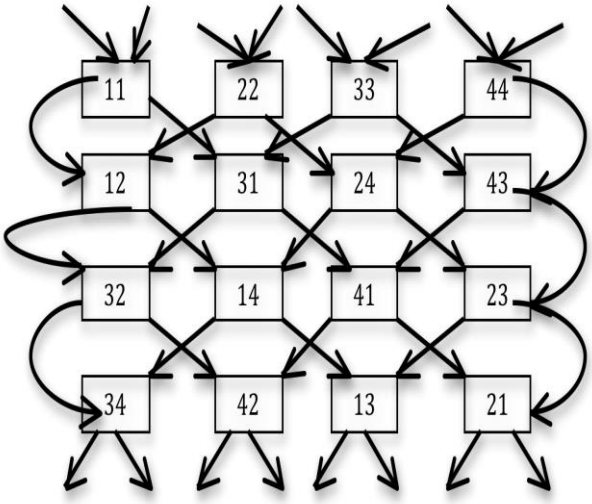


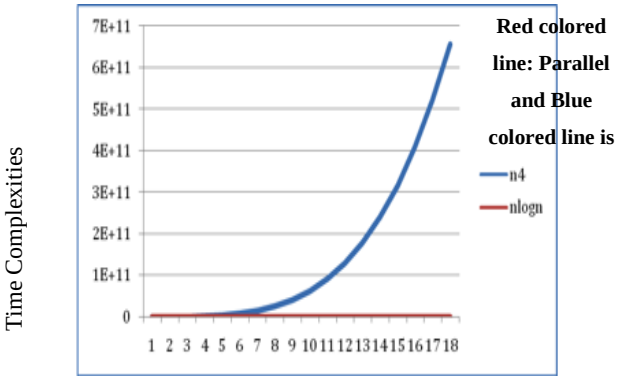
Fig 7. Matrix multiplication on the mesh network

Analysis:
The construction of complete binary tree using linked list takes O (n) time and Multiplication of two nXn matrices requires 2n-1 time using cross-wired mesh network by using n^2 processors. Multiplication process requires O (n) amount of time. And we need to multiply 'n' number of matrices that are organized in the form complete binary tree. A total step required to multiple 'n' numbers of matrices is logn. Our process requires O (nlogn) time to multiply 'n' number of matrices with log (n) number of step and n^2 number of processors to multiply every two matrices.

Time complexity=T (n)=O (nlogn)

Just imagine to multiply two matrices in a sequential machine requires n^3 time and to multiply 'n' number of matrices how much time does it requires? It will be O (n^4) time.

3.2 Comparison between sequential and our approach



Number of units of input
Fig 8. Sequential versus parallel to multiply 'n' number of matrices

Conclusion

Performance of an application merely depends upon the time and space complexity. With the advent of available architectures or resources, programmers are concentrating on execution time i.e., the execution speed of applications. A parallelization is an approach, which plays a major role in the improvement of performance of applications. If we use uniprocessor system to increase the performance of applications, we have to run the system at higher clock speeds that consume lots of power and further it generates a huge amount of heat and by making our programs parallel can eliminate these facts. We have identified parallel modules for multiplying set of matrices with the given dimensions using an architecture mesh network and its performance is compared with sequential approach.

Future Work

While this framework provides a new scenario to identify parallel modules. After identifying parallel modules according to our approach, we have considered only mesh network for execution of those modules. For this approach, we have designed an algorithm that can be further extended for practical implementation. Other interconnection topologies can also be considered for processing in place of the mesh network, further its performance can be compared with the existing approaches.

References

[1] Aldinucci, M., Danelutto, M., Kilpatrick, P., Meneghin, M., &Torquati, M. (2011, August). Accelerating code on multi-cores with fast flow. In European Conference on Parallel Processing (pp. 170-181). Springer, Berlin, Heidelberg.

[2] Benkner, S., Pllana, S., Traff, J. L., Tsigas, P., Dolinsky, U., Augonnet, C., ... &Osipov, V. (2011). PEPPHER: Efficient and productive usage of hybrid computing systems. IEEE Micro, 31(5), 28-41.

[3] Mens, T., & Tourwe, T.(2004). A survey of software refactoring. IEEE Transactions on software engineering, 30(2), 126-139.

[4] Cypher, R., & Sanz, J. L. (1989). SIMD architectures and algorithms for image processing and computer vision. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *37*(12), 2158-2174.

[5] Lee, S. Y., & Aggarwal, J. K. (1987). Parallel 2-D convolution on a mesh connected array processor. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (4), 590-594.

[6] Brown, C., Loidl, H. W., & Hammond, K. (2011, May). Paraforming: forming parallel haskell programs using novel refactoring techniques. In *International Symposium on Trends in Functional Programming* (pp. 82-97). Springer, Berlin, Heidelberg.

[7] Godbole, S. S. (1973). On efficient computation of matrix chain products. IEEE Transactions on Computers, 100(9), 864-866.

[8] Hu, T. C., & Shing, M. T. (1982). Computation of matrix chain products. Part I.*SIAM Journal on Computing*, *11*(2), 362-373.

[9] Ramanan, P. (1994). A new lower bound technique and its application: tight lower bound for a polygon triangulation problem. SIAM Journal on Computing, 23(4), 834-851.

[10] Valiant, L. G., Skyum, S., Berkowitz, S., & Rackoff, C. (1983). Fast parallel computation of polynomials using few processors. *SIAM Journal on Computing*, *12*(4), 641-644.

[11] Rytter, W. (1988). On efficient parallel computations for some dynamic programming problems. *Theoretical Computer Science*, *59*(3), 297-307.

[12] Wang, T. (1997). *Algorithms for parallel and sequential matrix-chain product problem* (Doctoral dissertation, Ohio University).

[13] Galil, Z., & Park, K. (1994). Parallel algorithms for dynamic programming recurrences with more than O (1) dependency. Journal of Parallel and Distributed Computing, 21(2), 213-222.

[14] Czumaj, A. (1993). Parallel algorithm for the matrix chain product and the optimal triangulation problems. *STACS 93*, 294-305.

[15] Ramanan, P. (1996). An efficient parallel algorithm for the matrix-chain-product problem. *SIAM Journal on Computing*, *25*(4), 874-893.

[16] Maresca, M., & Li, H. (1989). Connection autonomy in SIMD computers: a VLSI implementation. *Journal of Parallel and Distributed Computing*, *7*(2), 302-320.

[17] Patle, B.K. & Parhi, Dayal & Anne, Jagadeesh & Kashyap, Sunil Kumar. (2017). Matrix-Binary Codes based Genetic Algorithm for path planning of mobile robot. Computers & Electrical Engineering.67.10.1016/j.compeleceng.2017.12.011.