PARALLEL ALGORITHMS

Design and analysis of Algorithms

By:

HASNAT AMIR, 210752
MUHAMMAD NOUMAN IMRAN, 215192
MUHAMMAD ZAKIULLAH USMAN, 208655

CONTENT:

Introduction:

The parallel algorithm is an approach to solve real-world problems as well as high-level complex data management problems with ease & high efficiency. It works by dividing a bigger problem into sub smaller problems and then simultaneously executing the problem on different processors and combining the data into a final result.

Every day we face many task & challenges which requires high volume if data processing and by following the regular approach we cannot solve these tasks or even if some algorithm manages to do so then it will be a big challenge for the CPU to process and generate the Result.

This is where Parallel algorithm plays its role. These types of algorithms are carefully designed to manage a large volume of data and then rather than putting everything onto a single CPU, it divides the problem into subtask that runs the same algorithm but on a smaller scale with a lesser volume of data and executes all the subtasks simultaneously on every CPU core. In other words, multiple processors are used to running the same program with instructions divided across. Then collect the result and generate the final result for the task.

As a result, the running time of the program is smaller than the number of instructions executed.

This report describes the impact of such parallel algorithms on some problems that are common and useful in our everyday life.

Matrix Multiplication:

For our everyday data management issues matrix is an excellent solution. You can convert many types of problems into the matrix and solve them efficiently with ease. This section explains how matrix multiplication in parallel is better than in sequential programming.
Following are is the most efficient matrix multiplication algorithm that use a parallel approach to solve the problem.

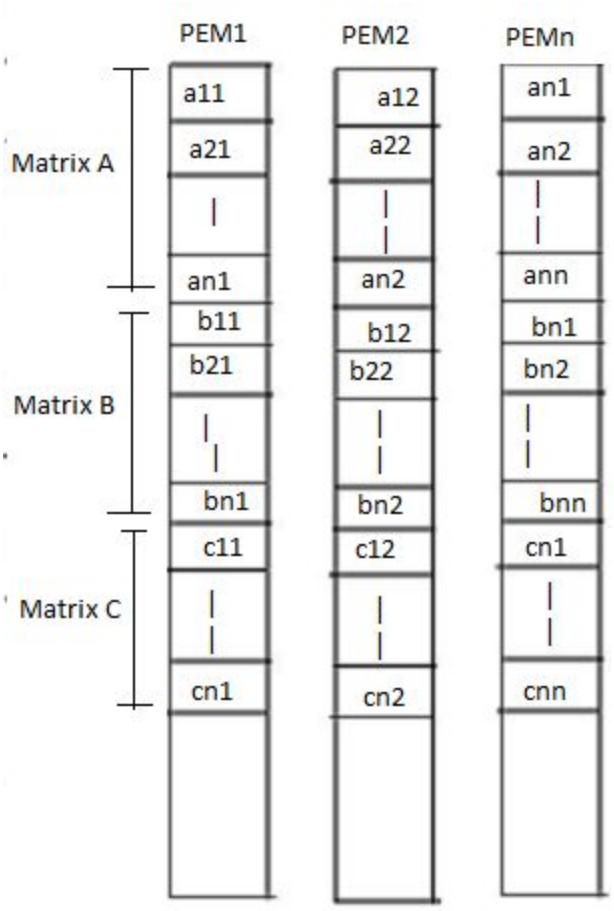
Matrix multiplication using Mesh network:

Matrix multiplication using SIMD i.e single instruction multiple data is the advance and faster version of matrix-matrix multiplication with time complexity of **$O(n^2)$**. The previous method of doing this was by following a SISD i.e single instruction single data technique which leads to a time complexity $O(n^3)$. But by using new and improved method faster and more efficient results are achieved. The method of doing so is explained as:

We have considered a 2D mesh network SIMD model having wraparound connections. We will design an algorithm to multiply two n × n arrays using n^2 processors in a particular amount of time.

Matrices A and B have elements a_{ij} and b_{ij} respectively. Processing element PE_{ij} represents a_{ij} and b_{ij}. Arrange the matrices A and B in such a way that every processor has a pair of elements to multiply. The elements of matrix A will move in left direction and the elements of matrix B will move in upward direction. These changes in the position of the elements in matrix A and B present each processing element, PE, a new pair of values to multiply.

We divide the matrix and assign it to multiple processors where the number of processors is equal to order to the matrix. The key feature of this progress is by calculating the multiplication simultaneously on a parallel processor which leads to time complexity of O (1), as parallel processing shows linear time complexity.



Complexity:

The following algorithm shows a time complexity of $O(n^2)$. The parallel loops takes a complexity of O(1) as it completes all the execution in a single cycle which results to a final complexity of n^2.

```
for i=1 to n do
    par for k=1 to n do
        c_ik = 0 [vector load]
    for j=1 to n do
        par for k=1 to n do
            c_ik = c_ik + a_ij * b_jk
            [Vector Multiply]
    end of j loop
end of i loop
```

Algorithm:

Procedure meshMultiply

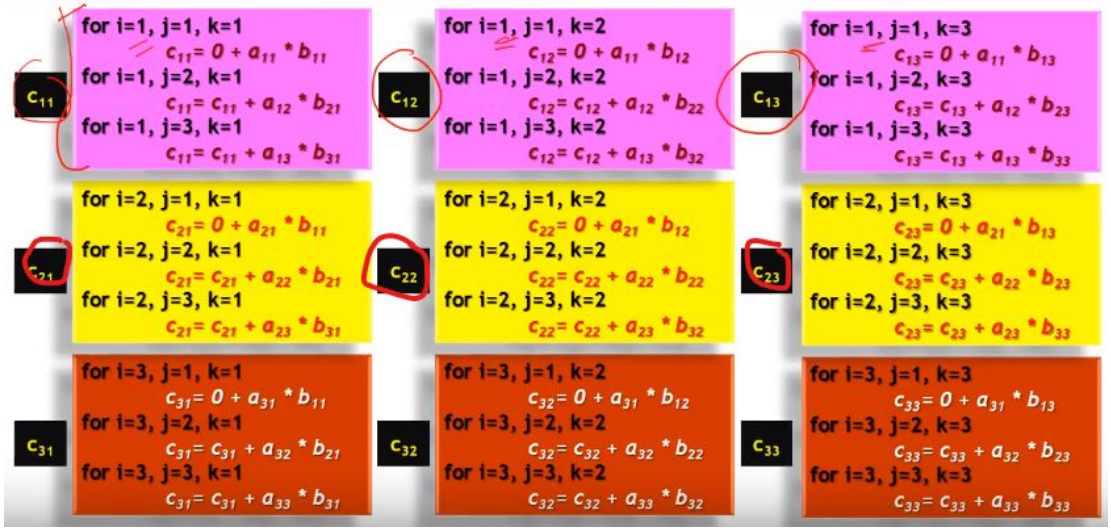
Begin
  for k = 1 to n-1

  for all Pij; where i and j ranges from 1 to n
    ifi is greater than k then
      rotate a in left direction
    end if

  if j is greater than k then
    rotate b in the upward direction
  end if

  for all Pij ; where i and j lies between 1 and n
    compute the product of a and b and store it in c
  for k= 1 to n-1 step 1
  for all Pi;j where i and j ranges from 1 to n
    rotate a in left direction
    rotate b in the upward direction
    c=c+aXb

End



Limitation:

The SIMD mesh approach is considered to be as the most efficient and widely used technique for matrix multiplication. The only limitation this algorithm has is that we require more number of processing elements as the matrix size also as the size of

matrix increases too large, in case of large images the we may need an even better technique for faster results.

Image Processing

It is a fast parallel processing algorithm. It consists of two sub-iterations. It is also known as a thinning algorithm. One sub-iteration is aimed at deleting the south-east boundary points and the north-west corner points while the other one is aimed at deleting the north-west boundary points and the south-east corner points.

Introduction

The method of extracting the distinctive features from the patterns is based on the efficiency and effectiveness of extracting these features from a pattern. This is one of the ways through which the general problem of pattern recognition is handled. The stroke analysis is an important method through which the recognition of certain types of digital patterns such as alphanumeric characters and ideographs is handled. The distortions may be different thinned by hardware or software. Hence, the strokes thinned by hardware and software are accompanied by distortions in the range [1-5, 7-12]. Similarly, different thinning algorithms produce different degrees of distortions.

The definition of thinness has no general agreement. Pavlidis has described a thinning algorithm that determines skeletal pixels by local operations. At the exact time, the pixels of graphics are labeled so that the original image can be reconstructed from its skeleton.

Reason

The reason for studying this algorithm is to find a faster and more efficient parallel thinning algorithm. Although many other algorithms exist and can also be studied, the scope of this domain and its applications in AI/ML allow us to explore a better algorithm that can handle this task of thinning image processing through parallel processing. The distortion should be as little as possible. Experimental results indicate that this method can be used to thin a variety of digital patterns.

Explanation

We will discuss the two most important concept involved when studying image processing through parallel techniques. These are parallel image processing and thinning algorithm. We will discuss them one by one.

Parallel Image processing

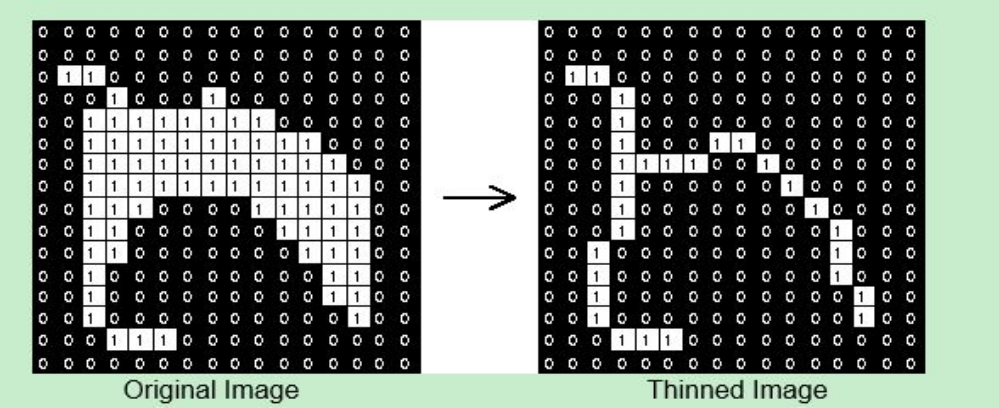
A binary digitized image is presented as a Matrix. Let this matrix is represented by *T.* Each pixel can be defined as an element of the *T(i,j).* The pixel can either be 1 or 0. If the pixel has value 1, then those points having value 1 will form a pattern. Each stroke in the pattern is more than one element thick. Iterative transformations are applied to matrix *T.* These transformations are produced point by point according to the values of a small set of points. It is assumed that the neighbors of the point (i, j) are (i - 1, j), (i - 1, j + 1), (i, j + 1), (i + 1, j + 1), (i + 1, j), (i + 1, j - 1), (i, j - 1), and (i - 1, j - 1).

P_9 $(i-1, j-1)$	P_2 $(i-1, j)$	P_3 $(i-1, j+1)$
P_8 $(i, j-1)$	P_1 (i, j)	P_4 $(i, j+1)$
P_7 $(i+1, j-1)$	P_6 $(i+1, j)$	P_5 $(i+1, j+1)$

This table is showing the designations of nine-pixel in a 3x3 window. In a parallel image processing, the new value given to a point at the nth iteration depends on its own value as well as those of its eight neighbors at the (n - 1) iteration. This is because of the reason that the image point can simultaneously be processed. It is assumed that one point on the table let (i,j) is connected with eight other points and the window size is selected as 3x3. The algorithm requires only simple computations to find the points involved in image processing.

THINNING Algorithm

Themethod for extracting the skeleton of an image is removing all the surrounding points of the image except those points that belong to the skeleton. In order to preserve the final shape of the skeleton, each iteration is divided into two sub-iterations.



Considering the first sub-iteration, the contour point P1 is deleted from the digital pattern. This point is deleted on the basis of 4 conditions. These include:

1. 2 <= B(P1) <= 6
2. A(P1)= 1
3. P2*P4*P6 = 0
4. P4*P6*P8 = 0

Where A(P1) = The number of 01 patterns in the ordered set P2, P3, P4, ……. P8, P9 that are the eight neighbors of P1,

B (P1) = The number of nonzero neighbours of P1 or B (P1) = P2 + P3 + P4 + • • • + P8 + P9,

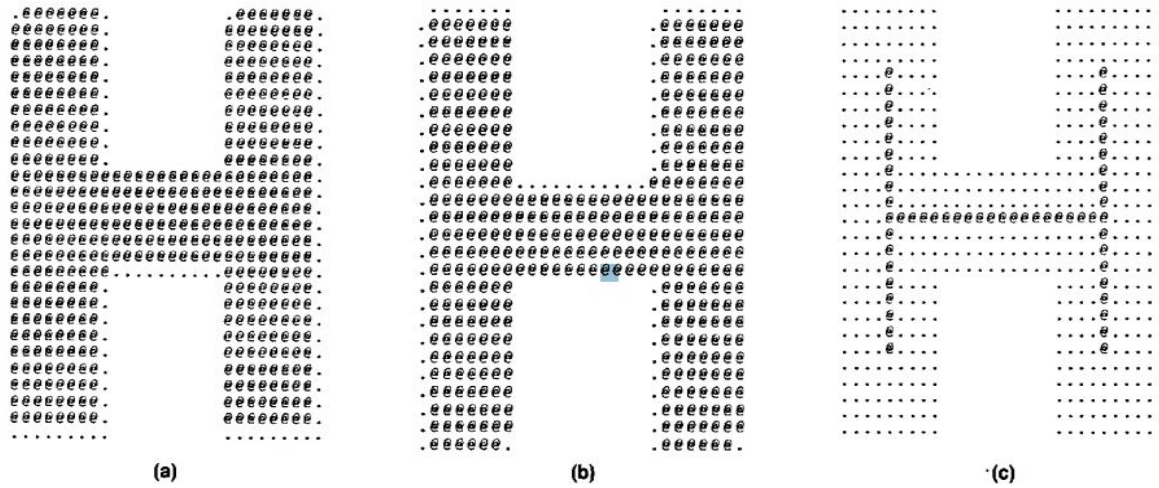
Let if any of the condition is not satisfied, the values of P2, P3, P4……. P9, then A(P1) = 2

Therefore, P1 is not deleted from the image.

Now, consider the second sub-iteration, only the condition c and d are changed and the new conditions are c' and d'. The remaining remains the same.

(c'). P2*P4*P8 = 0
(d') P2*P6*P8 = 0

By conditions (c) and (d) of the first sub-iteration, it will be shown that the first sub-iteration removes only the south-east boundary points and the north-west corner points which do not belong to an ideal skeleton.



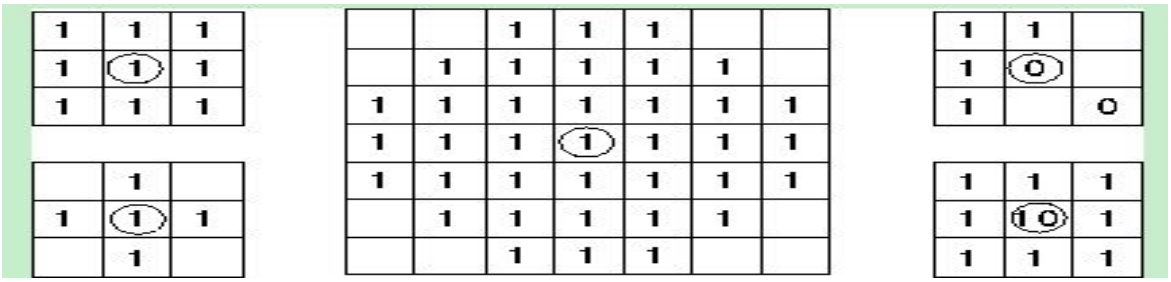
The results of processing the character “H” by both sub-iterations are shown in the figure above. The points which are shown by “.” are removed.

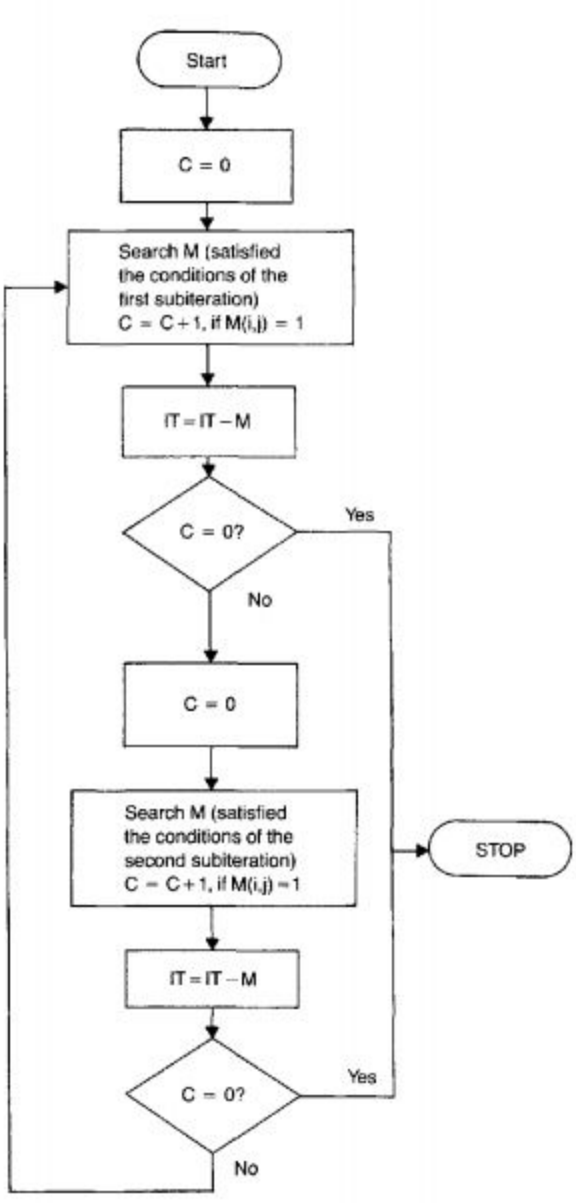
Mathematical Formulas

$$A \oplus B = \left\{ x : (\hat{B})_x \cap A \neq \varnothing \right\} = \bigcup_{x \in B} A_x$$

This equation simply means that *B* is moved over *A* and the intersection of *B* reflected and translated with *A* is found. Usually, *A* will be the signal or image being operated on and *B* will be the ‘**Structuring Element**’.

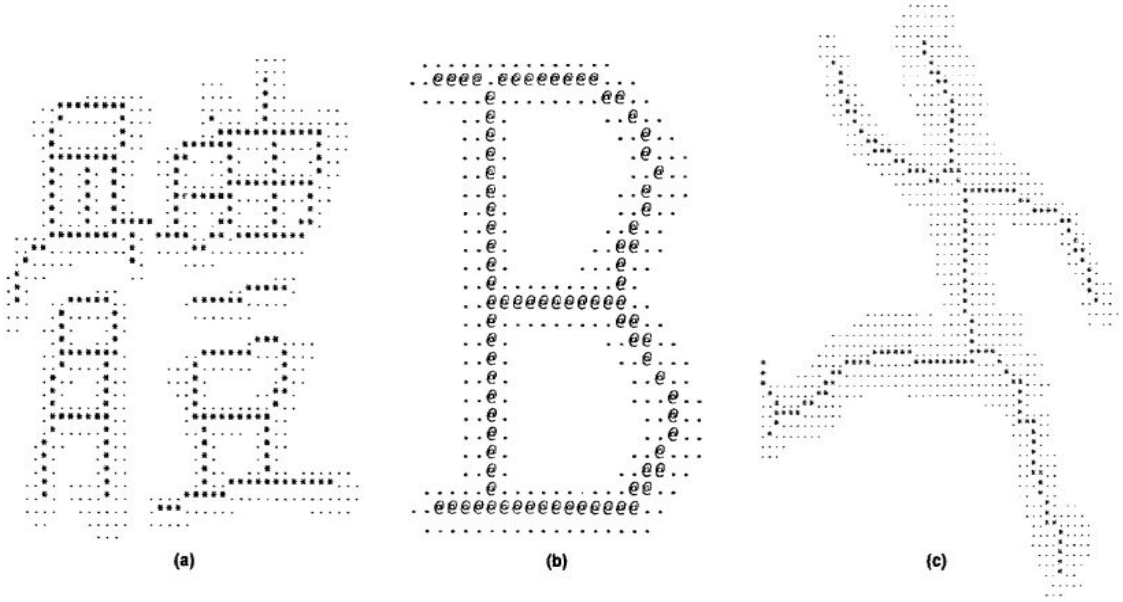
$$A \ominus B = \left\{ x : (B)_x \subseteq A \right\} = \bigcap_{x \in B} A_x$$





This flowchart is explaining the flow of the algorithm. At the start, a counter is initialized by 0. Initially, the original picture is stored in matrix *T*. The results are stored in matrix *T.* Only two matrices are used in our computations T and M. At the start, the counter is set to zero and if the condition of first sub-iteration is satisfied

i.e. if M(i,j)=1 then C=C+1 and the transformed matrix becomes T-M and the cycle continues.



This image shows the thinning of some other patterns like B, moving body etc.

Complexity

The above algorithm yields very good results. With respect to both connectivity and contour noise immunity, this algorithm is performing really nice. Similarly, the conditions for searching those points that should be deleted from the pattern are very simple.

Limitations

There are limitations in all Parallel Algorithms. These limitations usually come in the form of Inherent Sequentiality. Inherent Sequentiality of a problem is the tasks that must be performed in series. To make this concept clear, an interesting analogy is discussed.

Suppose it takes 50 men 200 days in order to complete the construction of a house. To sum it all up, that is 10000 days of work if each man's work is considered. If the number of men is increased to 100, it will take them 100 days to complete the house. That's double the rate. But can 10,000 men build the same house in just one day? The answer is no. This is because there are some tasks that must be done sequentially because they depend on their output. For instance, the foundation must

be laid before the walls. Moreover, the roof can only be built if there are supporting walls. And similarly, the first floor can only be built if there is a ground floor already in place.

Similarly, this is the case in computers. When we add more processors or more machines to perform the same job, we can get enhanced performance but only up to a limit. Now let's take an example for computer instruction.

Suppose there is a problem, P which can be solved in T(n) time with just one processor. Where n is the size of the problem. Suppose we have p number of processors which can solve the problem in Tp(n) time. The speed-up factor here is represented by T(n)/Tp(n). We expect it to be p. However, due to the inherent sequentiality, the time p is impossible to achieve.

There is one more limitation as far as Parallel algorithms are concerned. Unlike serial algorithms, which deal with the resource of time and space, these algorithms have to factor in the delay in the communications between the processors.

Why and how to improve?

Most parallel algorithms suffer from the issue of Load Balancing. This is the technique of making sure that each processor is given a similar load of work rather than the same input size. For example, the task is to find prime numbers in the range 0-200. This task is divided as the processor A is given 0-100 while the processor B is given 100-200. The input size is the same for both but it is easier to calculate primality of lower numbers. As a result processor, A will be idle for a time that processor B is still busy because it has a bigger computational load.

This is why we should improve any parallel algorithm to make sure each processor carries a similar computational load.

There is a room for improvement when some assumptions are made. Moreover, to combat the Inherent Sequentiality the program can be split into portions which must be performed in a sequence and other portion in which order of instruction execution is not relevant.

A sort of hybrid system can be used in which parallel algorithm behave just like sequential algorithms where they have to and switch back to parallel whenever possible. These were some of the ways an improvement can be made.

Conclusions

Parallel Algorithms have a number of applications which are growing. While the idea is closely linked with parallel computer architecture, there are differences as well. With the development of parallel algorithms, many sorting algorithms are improved

further like the Parallel Merge Sort and the Hyper Quick Sort. We learned that there can be many other applications of this class of algorithms as well, however, these require special parallel programming languages which feature data structures such as hypercubes for communications between the processors.

It is expected that this field will grow and be implemented in Machine Learning Algorithms.

References

[[1]] Kai Hwang, Faye A. Briggs, —Computer Architecture and Parallel Processing || McGraw Hill, 1985.
[[2]] Junjie Li Sanjay Ranka Sartaj Sahni, —Strassen's Matrix Multiplication on GPUs || , National Science Foundation
[[3`4]] M. Krishnan, J. Nieplocha, || SRUMMA: A Matrix Multiplication Algorithm Suitable for Clusters and Scalable
[[5]] Shared Memory Systems || , to appear in Proceedings of International Parallel and Distributed Processing Symposium
(IPDPS'04), Santa Fe, NM, 2004.
[[6]] Robert A. van de Geijn, Jerrell Watts, —SUMMA: Scalable Universal Matrix Multiplication Algorithm || , NAG5-2497
[[7]] John Gunnels Calvin Lin Greg Morrow Robert van de Geijn, || A Flexible Class of Parallel Matrix Multiplication
[[8]] Algorithms || , PRISM project (ARPA grant P-95006) and the Environmental Molecular Sciences construction project at

---

[1] "Computer architecture and parallel processing / Kai Hwang, Fayé A ...." https://trove.nla.gov.au/work/19437920. Accessed 3 May. 2019.
[2] "Strassen's Matrix Multiplication on GPUs - UF CISE - University of Florida." https://www.cise.ufl.edu/~sahni/papers/strassen.pdf. Accessed 3 May. 2019.
[3] "SRUMMA: a matrix multiplication algorithm suitable for clusters and ...." 20 Oct. 2018, https://www.researchgate.net/publication/4077686_SRUMMA_a_matrix_multiplication_algorithm_suitable_for_clusters_and_scalable_shared_memory_systems. Accessed 3 May. 2019.
[4] "A Matrix Multiplication Algorithm Suitable for Clusters and Scalable ...." http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.78.71&rep=rep1&type=pdf. Accessed 3 May. 2019.
[5] "Mahmut Taylan Kandemir - Computer Science and Engineering." http://www.cse.psu.edu/~mtk2/publications.html. Accessed 3 May. 2019.
[6] "SUMMA: Scalable Universal Matrix Multiplication Algorithm." https://courses.cs.washington.edu/courses/csep524/02au/summa.pdf. Accessed 3 May. 2019.
[7] "A Flexible Class of Parallel Matrix Multiplication Algorithms." https://www.cs.utexas.edu/~lin/papers/ipps98.pdf. Accessed 3 May. 2019.
[8] "PNNL: News - PNNL to build new energy sciences research building ...." 22 Jan. 2019, https://www.pnnl.gov/news/release.aspx?id=4542. Accessed 3 May. 2019.

Pacific Northwest National Laboratory.