# OPTIMIZATION OF COMPUTATION IN PARALLEL COMPUTING BASED ON FINITE PROJECTIVE GEOMETRY

Lokendra Gour, Akhilesh A. Waoo
Assistant Professor, Associate Professor and HOD
Department of Computer Science, Department of Computer Science Applications and Technology
RGCCAT Satna(MP) India, AKS University Satna(MP) India.

***Abstract:*** Various engineering and scientific computational problems involve both sparse and dense matrices. Problems related to dense matrix can be resolved easily by parallelizing matrix computations. For problems related to sparse matrix, Karmaker suggested a new parallel architecture which is based on finite projective geometry. For such kinds of computations, it is essential to utilize parallel architectures. All contemporary computer systems are equipped with multi-core model in addition to multithreading. These multiple cores which are physical processing units provide parallelism. These cores may be homogeneous or heterogeneous. These cores operate in parallel and each performs separate computations by using different instruction streams on different data streams.

*Keywords:* **Concurrent computing, parallel computing, distributed computing, parallel algorithm, Projective geometry, Load balancing, Load assignment, Perfect pattern, perfect sequence.**

## I INTRODUCTION

In parallel computing, it is extremely necessary to speed up computations. Following the two major problems arise in parallel architecture: However parallel computations suffer following major problems:

- Data Distribution: It describes the assignment of data to the appropriate processing unit to enhance the performance of the system.
- Expression Evaluation: Expression can be broken down into sub-expression for further computation.
- Load Balancing: It is achieved by distributing the computations in such a manner that each processor occupies equal amount of computational load.
- Memory Access Conflicts: In parallel architecture, whenever two or more processors compete to access the same memory location this lead to memory access conflicts.

The computations assigned to a processor depend on the projective geometry and the incidence relations. Since the projective geometry possesses the symmetric nature, the computation load on each processor is balanced. The pattern of these geometries defines the interconnections between processors and memories, and also helps to solve difficult tasks such as load balancing, bandwidth matching, avoiding conflicts, data routing etc. (The automorphism governing these geometries are used to develop 'perfect-access patterns' and 'perfect-access sequences', which confirms that all the processors and memories are simultaneously involved in conflicts free communication of data.

## II PARALLEL COMPUTING

Traditionally software has been written for serial computations:

- To be run on a single computer having a single Central Processing Unit (CPU)
- A problem is broken into a discrete set of instructions
- Instructions are executed one after another
- Only one instruction can be executed at any moment in time

In the simplest sense, parallel computing is the simultaneous use of multiple compute resources to solve a computational problem:

- To be run using multiple CPUs
- A problem is broken into discrete parts that can be solved concurrently
- Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different CPUs

### 2.1 Parallel Algorithm

An algorithm is a sequence of steps that take inputs from the user and after some computation, produces an output. A parallel algorithm is an algorithm that can execute several instructions simultaneously on different processing devices and then combine all the individual outputs to produce the final result.

### III Concurrent Processing

The easy availability of computers along with the growth of Internet has changed the way we store and process data. We are living in a day and age where data is available in abundance. Every day we deal with huge volumes of data that require complex computing and that too, in quick time. Sometimes, we need to fetch data from similar or interrelated events that occur simultaneously. This is where we require concurrent processing that can divide a complex task and process it multiple systems to produce the output in quick time.

Concurrent processing is essential where the task involves processing a huge bulk of complex data. Examples include − accessing large databases, aircraft testing, astronomical calculations, atomic and nuclear physics, biomedical analysis, economic planning, image processing, robotics, weather forecasting, web-based services, etc.It is not easy to divide a large problem into sub-problems. Sub-problems may have data dependency among them. Therefore, the processors have to communicate with each other to solve the problem.

It has been found that the time needed by the processors in communicating with each other is more than the actual processing time. So, while designing a parallel algorithm, proper CPU utilization should be considered to get an efficient algorithm to design an algorithm properly, we must have a clear idea of the basic model of computation in a parallel computer.

### IV Model of Computation

Both sequential and parallel computers operate on a set (stream) of instructions called algorithms. These set of instructions (algorithm) instruct the computer about what it has to do in each step. Depending on the instruction stream and data stream, computers can be classified into following four categories:

- Single Instruction stream, Single Data stream (SISD) computers
- Single Instruction stream, Multiple Data stream (SIMD) computers
- Multiple Instruction stream, Single Data stream (MISD) computers
- Multiple Instruction stream, Multiple Data stream (MIMD) computers

### SISD Computers

SISD computers contain **one control unit, one processing unit,** and **one memory unit**.


Figure: SSID computers

In this type of computers, the processor receives a single stream of instructions from the control unit and operates on a single stream of data from the memory unit. During computation, at each step, the processor receives one instruction from the control unit and operates on a single data received from the memory unit.

### SIMD Computers

SIMD computers contain one control unit, multiple processing units, and shared memory or interconnection network.

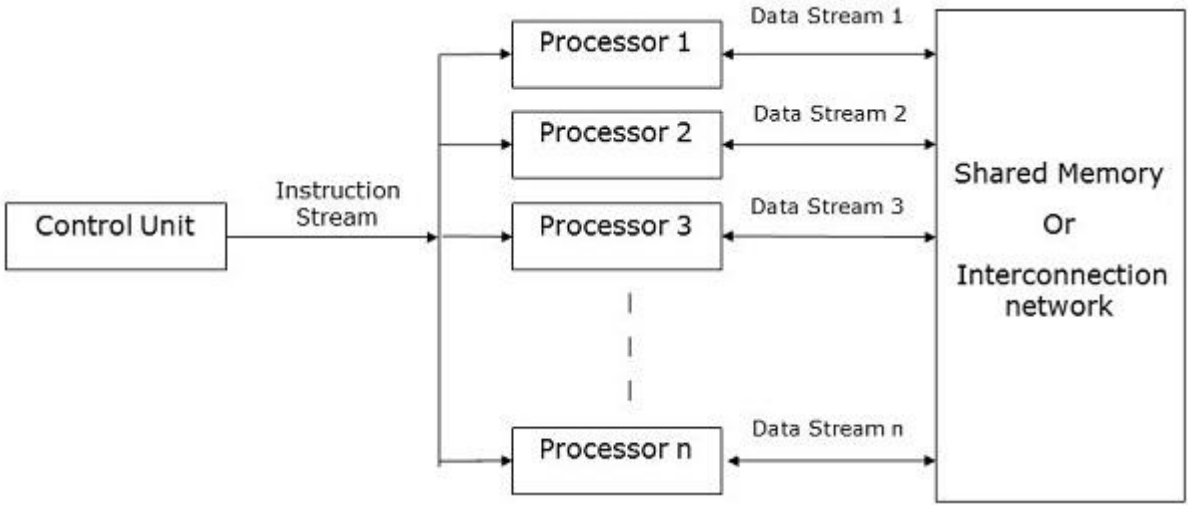


Figure: SIMD computers

Here, one single control unit sends instructions to all processing units. During computation, at each step, all the processors receive a single set of instructions from the control unit and operate on different set of data from the memory unit.Each of the processing units has its own local memory unit to store both data and instructions. In SIMD computers, processors need to communicate among themselves. This is done by shared memory or by interconnection network. While some of the processors execute a set of instructions, the remaining processors wait for their next set of instructions. Instructions from the control unit decides which processor will be **active** (execute instructions) or **inactive** (wait for next instruction).

**MISD Computers**

As the name suggests, MISD computers contain multiple control units, multiple processing units, and one common memory unit.

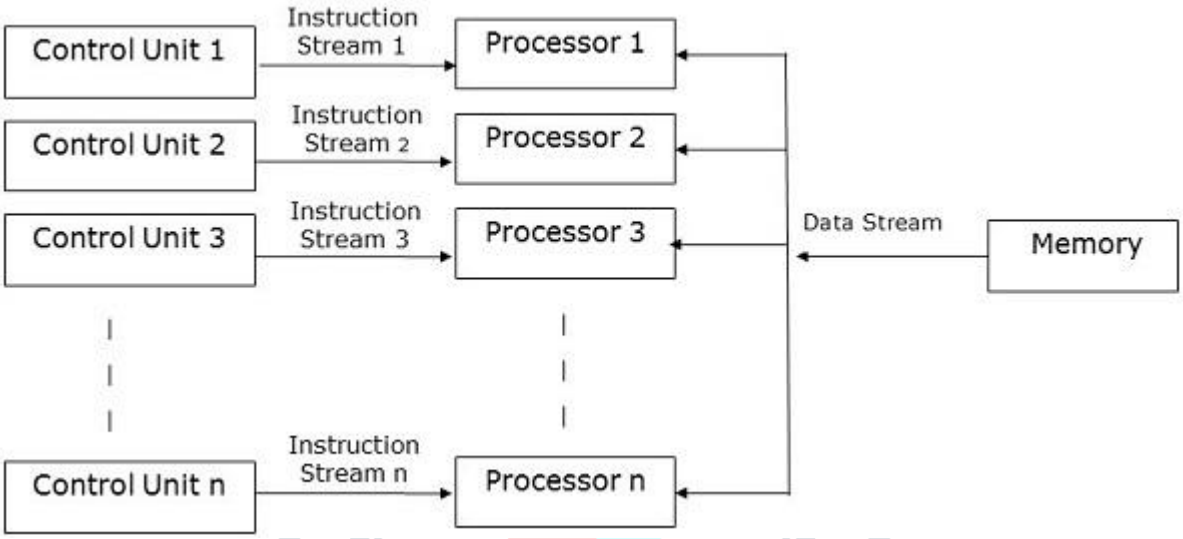


Figure: MISD computers

Here, each processor has its own control unit and they share a common memory unit. All the processors get instructions individually from their own control unit and they operate on a single stream of data as per the instructions they have received from their respective control units. This processor operates simultaneously.

**MIMD Computers**

MIMD computers have multiple control units, multiple processing units, and a shared memory or interconnection network.

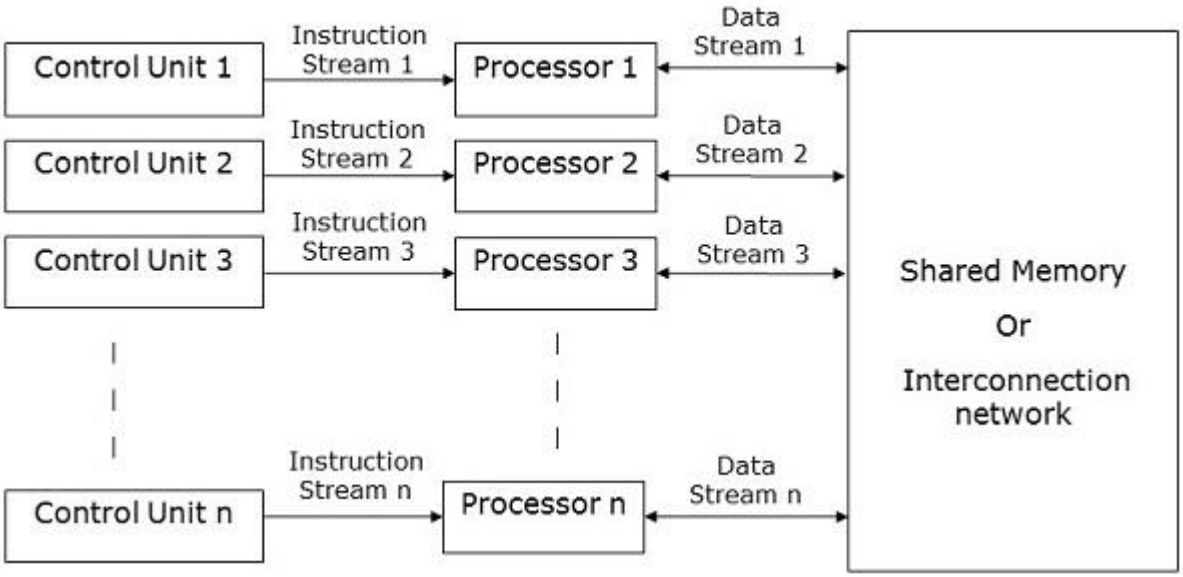


Figure: MIMD computers

Here, each processor has its own control unit, local memory unit, and arithmetic and logic unit. They receive different sets of instructions from their respective control units and operate on different sets of data.

- An MIMD computer that shares a common memory is known as multiprocessors, while those that uses an interconnection network is known as multi-computers.
- Based on the physical distance of the processors, multi-computers are of two types −
  - Multicomputer − When all the processors are very close to one another (e.g., in the same room).
  - Distributed system − When all the processors are far away from one another (e.g.- in the different cities)

**V Projective Geometry**

A geometry is denoted by $G = (\Omega, I)$, where $\Omega$ is set and I is a relation which is both symmetric and reflexive. The relation on geometry is called the incidence relation, For example consider the traditional Euclidian geometry. In this geometry, the objects of the set $\Omega$ are points and lines. A point is incident to a line if it lies on that line, and two lines are incident if they have all points in common – only when they are the same line.

A point in a finite projective plane $PG(2, P^n)$, may be denoted by the symbol $(x_1, x_2, x_3)$ , where the coordinates $x_1, x_2, x_3$ are marks of a Galois field of order $P^n$ , $GF(P^n)$. The symbol $(0,0,0)$ is excluded, and if k is a nonzero mark of the $GF(P^n)$, the symbols $(x_1, x_2, x_3)$ and $(kx_1, kx_2, kx_3)$ are to be thought of as the same point. The totality of points whose coordinates satisfy the equation $u_1 x_1 + u_2 x_2 + u_3 x_3 = 0$, where $u_1$ , $u_2$ , $u_3$ are marks of the $GF(P^n)$, not all zero , is called a line. The plane then consists of $P^{2n} + P^n + 1 = q$ points and $q$ lines: each line contains $P+1$ point.

**5.1 Finite Projective Geometries**

There is a mathematical construct known as finite projective geometry, which plays an important role in defining the parallel architecture. The structure of these geometries is helpful in efficiently solving several difficult problems encountered in the design of parallel systems, such as load balancing, data routing, memory access conflicts, etc.

Consider a finite field, $F_s = GF(s)$ which has $s = p^k$ elements , where p is a prime, k is a positive integer. A projective geometry of dimension d, denoted by $P^d(F_s)$, is the set of all one dimensional subspaces of the (d+1)-dimensional vector space $F_s^{d+1}$ over the field $F_s$. A one dimensional subspace of $F_s^{d+1}$ generated by $x, x \in F_s^{d+1}, x \neq 0$ , is the set of all nonzero elements of the form $\lambda x, \lambda \in F_s$. These subspaces are the points of the projective geometry. Since there are $(s^{d+1}-1)$ nonzero elements in $F^{d+1}$, and $(s-1)$ nonzero elements in $F_s$, the number of points in the geometry, $n_d$, is given by $(s^{d+1} - 1)/(s - 1)$. Similarly, an m-dimensional subspace of the projective geometry consists of all one dimensional subspaces of an (m + 1)-dimensional subspace of $F^{d+1}$. If $\{b_0, b_1, \ldots b_m\}$ forms a basis of this vector subspace, then the elements of the subspace are of the form

$$\sum_{i=0}^{m} \alpha_i b_i \text{ , where } \alpha_i \in F_s$$

The number of elements in the subspace, $n_m$ , is given by $(s^{m+1} - 1)/(s - 1)$. The set of all m-dimensional projective subspaces of $P^d(F_s)$ is denoted by $\Omega_m$. Now $\Omega_0$ represents the set of all the points of the projective space, $\Omega_1$ is the set of all lines, $\Omega_2$ is the set of all planes and so on. For $n \geq m$, to count the number of elements in each of these sets, we define the function

$$\Phi(n, m, s) = \frac{(s^{n+1} - 1)(s^n - 1) \ldots (s^{n-m+1} - 1)}{(s^{m+1} - 1)(s^m - 1) \ldots (s - 1)}$$

Let $0 \leq l < m \leq d$. Then the number of l-dimensional subspaces of $P^d(F_s)$ contained in a given m-dimensional subspaces is given by $\phi(m, l, s)$, and the number of m-dimensional subspaces of $P^d(F_s)$ containing a given l-dimensional subspaces is given by $\phi(d-l-1, m-l-1, s)$.

**VI Description of Karmakar's Architecture**

As mentioned earlier, Karmakar's architecture defines the interconnection patterns between processors and memories based on finite projective geometry. A finite projective geometry of dimension d consists of a set of points S, which form the zero-dimensional subspaces. These points can be grouped together to form subspaces of higher dimensions (1, . . ., d). The subspaces of dimension 1 are called lines, 2-dimensional subspaces are called planes and the d-1-th dimensional subspaces are called hyper-planes. Once the appropriate geometry for a problem has been identified, a pair of dimensions $d_m$ and $d_p$ are chosen. The processors are associated in one-to-one correspondence with the subspace of dimension $d_p$ while the memories are associated with subspaces of dimension $d_m$ and a connection between a processor and memory is established if the corresponding subspaces have a non-trivial intersections.

The access of memory is done in a structured fashion. By applying the symmetry of the geometry, it is possible to identify processor-memory pairs, involving all the processors and memories, which can communicate in a conflict-free manner. Each such set

of processor-memory pairs forms a perfect-access pattern. A collection of all such patterns together forms a perfect-access sequence, which ensures that every processor gets to communicate with every memory it is directly connected to.

For the distribution of computational work between processors, first the problem is broken down into atomic computations and operations that can be carried out parallel are considered together. Then the memories, which conflict the operands needed for a particular operation, are identified and the operation is assigned to the processor connected to these relevant memories, which is unique for most operations and depends on the problem and the underlying geometry.

The symmetry of geometry ensures that a balance is maintained in the distribution of load among the processors. Thus, the data required for computations is brought in parallel using parallel access sequence and the computations are then carried out parallel on each processor, ensuring efficient use of resources while avoiding conflicts and maintaining load balance.

**VII Computational Environments**

The proposed architecture can be used as an attached accelerator to a general purpose host processor. The accelerator and the host share a global memory system. The main program runs on the host, while computationally intensive subroutines are to be executed on the attached accelerator. The shared memory consists of partitioned memory modules, which are shared by processors in the accelerator over an interconnection network. Since the host and the accelerator share the memory system, it is not necessary to communicate large amount of data between the two separate I/O buses.

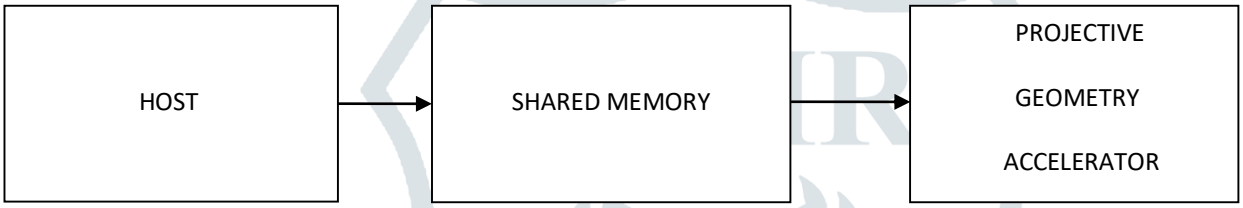


Figure 1: Computational Environment

Only certain structural information such as base address of arrays needs to be communicated from host processor to the attached accelerator before invoking a subroutine to be executed on the attached accelerator.

**7.1 Interconnection Scheme**

There is a finite projective geometry of dimension d, we describe the architecture as. Select a pair of dimensions $\mathbf{0 \le d_m < d_p \le d}$. Put the processors in the system in a one-to-one correspondence with subspace of dimension $\mathbf{d_p}$, and put memory modules in a one-to-one correspondence with subspaces of dimension $d_m$. From a connection between a processor and a memory module if an d only if(iff) the subspace corresponding to the processor contains the subspace corresponding to the memory module. By reference to above discussion, the number of processors in the system will be $\mathbf{\varphi(d,d_p,s)}$, and the number memory modules will be $\mathbf{\varphi(d,d_m,s)}$. Each processor will be connected to $\mathbf{\varphi(d_p,d_m,s)}$ memory modules, and each memory modules will be connected to $\mathbf{\varphi(d-d_m-1,d_p-d_m-1,s)}$ processors. If we are interested in a symmetric architecture, with an equal number of processors and memory modules, then we must choose $\mathbf{d_p}$ and $\mathbf{d_m}$ such that $\mathbf{d = d_p+d_m+1}$

**7.2 Load Assignment**

With the above correspondence between subspace of geometry and processors (and memories), the assignment of computational load to processor can automatically be done at a fine-grain level. To illustrate this, consider a binary operation

$$\mathbf{o \leftarrow a\ o\ b}$$

Suppose operand a is in memory module $M_l$, and b is in memory module $\mathbf{M_j}$. Then we associate an index pair **(i, j)** with this operation. (Similarly, we associate an index triplet with a ternary operation). The processor $P_i$ responsible for doing this operation is determined by a function ***f*** that depends on the geometry:

$$\boldsymbol{l = f(i, j)}$$

Thus operations having the same associated index pairs (or triplets) always assigned to the same processor. Furthermore, the function ***f*** is compatible with structure of the geometry i.e. processor $P_i$ has connection to memory modules $\mathbf{M_i}$ and $\mathbf{M_j}$.

**7.3 Perfect Pattern and Perfect Sequences**

Now we introduce the concepts of perfect pattern and perfect sequences which restrict the combination of words that can be accessed in one cycle. These combinations are designed so that no conflicts can arise in either accessing the memories or in sending the accessed data through the interconnection network. We define a perfect access pattern for a symmetric architecture based on a 4-dimensional geometry. In this architecture, memory modules are in a one-to-one correspondence with lines, and processors are in one-to-one correspondence with planes. A memory module is connected to a processor if and only if the line corresponding to the memory module lies on the plane corresponding to the processor.

Suppose the number of lines (and hence the number of planes) is n. A perfect access pattern P is a collection of n non-collinear triplets,

$$P = \{(p_i,q,r_i) \mid p_i, q_i, r_i \in \Omega_0, dim(p_i,q_i\, r_i) = 2, i = 1 \ldots n\}$$

Satisfying the following properties:

1. Let $u_i$ , i = 1…n denote the lines generated by first two points of each triplet

   $u_i = < p_i , q_i >$

   Then the collection of lines $\{u_1 \ldots u_n\}$ forms a permutation of all the lines of the geometry.
2. Let $v_i$, i = 1…n denotes the lines

   $V_i = < q_i , r_i >$

   Then the collection of lines $\{v_i \ldots v_n\}$ form a permutation of all the lines of the geometry.
3. Let $w_i$ , i = 1…n denotes the lines

   $W_i = < r_i , p_i >$

   Then the collection of lines $\{w_i \ldots w_n\}$ forms a permission of all the lines of the geometry.

4. Let $h_i$ , i= 1…n denotes the planes

   $H_i = < p_i , q_i , r_i >$

   Then the collection of planes $\{ h_i \ldots h_n \}$ forms a permutation of all the planes of the geometry.

Since there is a connection between a memory module M and a processor **P** iff the line **α** Corresponding to M is contained in the plane β corresponding to P, we can denote a connection by the ordered pair **(α , β )**. Let C be the collection of all processor-memory connections:

$$C = \{(\alpha, \beta) \mid \alpha \Omega_1 , \beta \Omega_2 , \alpha \subseteq \beta\}$$

Then, if ($p_i$ , $q_i$ , $r_i$ ) is a triplet in a pattern P, $u_i$ , $v_i$ , $w_i$ are the corresponding lines and $h_i$ is the corresponding plane , we say the perfect pattern P exercises the connections ($u_i$, $h_i$), ($v_i$, $h_i$ ) and ($w_i$, $h_i$).

A sequence of perfect patterns is called a perfect sequence if each connection in C is exercised the same number of times collectively by the patterns in the sequence. It follows that if such perfect sequences form the basis for instruction executed on the architecture, it leads to a uniform utilization of even the wires connecting processors and memories. It is then possible to connect the processors and memories so that the number of wires in the system only grows linearly with the number of processors. A definition of perfect pattern for 2-dimentional geometries and a discussion on how to generate perfect pattern based on automorphisms of the underlying groups

Using the automorphisms, we develop perfect matching sequence, which are bijective mappings between lines and planes. This is possible because we have the same number of planes and lines, both of which have been generated using the same automorphisms. Of these, the mappings that are relevant to our work are those that map a plane to one of the 7 lines that lie on it.

Consider the first sequence, $S_1$: $\Omega_2 \rightarrow \Omega_1$.

$$S_1(p) = L_x^a(\phi^b(0,1,18)), \text{ if } p = L_x^a(\phi^b(0,1,2,5,11,18,19))$$

(The automorphism of a line or a plane is the set formed by the automorphisms of individual points on that line or plane)

**VIII Problem Mapping Strategies**

There are 2 different schemes that we have discussed here. The two schemes are based on the complete geometry using all the 155 lines and planes. There are 155 processors and 155 memory modules in each case. However they differ in their architecture and in the distribution of computational load over the processors.

For all the strategies, the number of blocks in each row and column in matrix A is multiple of 31, as the block indices are associated with the points and there are 31 points in all. The indices are taken to be zero based. The block indices are mapped to points by taking their remainder modulo 31. Therefore we have the mapping function f: block indices → points as

$$F(b) = b(mod\ 31)$$

**IX Algorithm Mapping Scheme**

In this design, we have 155 processors and 155 memory modules and we use the entire P(4,GF(2)) geometry in defining the interconnection network. Each processor is connected to its own exclusive memory module; this processor-memory pair is associated with a line. In addition, the processor is also associated with plane mapped to the line through perfect matching $S_1$. The processor is

directly connected to processor-memory pair representing other lines on its plane. Hence, each processor is connected to 12 other processors- 6 processors that lie on its own plane and 6 other processors in whose plane it lies.

### 9.1 Data Distribution

The data distribution in the memory modules depends on the indices of the matrix block and the triplet of points representing each module. Consider the following function **M( i,j)** from point doublets (elements of $\Omega_0 \times \Omega_0$) to $\Omega_1$, which specifies the memory module for $\mathbf{A_{p,q}}$ if **f(p) = I** and **f(q) = j.**

$M(\alpha\ \beta)$ = *line joining points* $\alpha$ *and* $\beta$ *all* $\alpha, \beta\ \mathcal{E}\ 0,1,\ldots 30$ *and* $\alpha \neq \beta$
$M(i,j) = L_x^i(\phi^n(0,1,18))$, $a\ \varepsilon\ 0,1,2,3,4$

As can be seen from these equations, every block $A_{i,j}$, with distinct i, j , gets stored ij the memory module with the corresponding points in its 3-tuple representation. For example, the block $A_{0,1}$ and $A_{32,31}$ go into the module (0,1,18). This specifies the storage for all the non-diagonal blocks.

### 9.2 Distribution of Computations

The computation, which are represented by triplets of block indices are first converted to a triplet of points using the f map. These point triplets are distributed according to the following map $P(\alpha,\beta,\gamma) : \Omega_0 \times \Omega_0 \times \Omega_0 \rightarrow \Omega_2$..

P(α,β,γ) = plane through non-collinear points α,β and γ
P(α, α , β) = $S_1^{-1}$(line joining α,β)
P(α, β, α ) = $S_1^{-1}$(line joining α,β)
P(α, β, β ) = planes passing through α and the lines M(β, β)

As can be seen in the above equation, the computations corresponding to non-collinear triplets are allocated to the processor associated with the plane passing through that triplet. The column updates for the i-th iteration are carried out on the processors obtained by using the perfect matching sequence $S_1^{-1}$ on each of 15 memory modules associated with lines passing through point i. The update of a diagonal block is done along with the update of other blocks stored.

## X Conclusions

We can implement implicit and explicit parallelism to exploit speed-up computations. By applying language's constructs we achieve implicit parallelism and by applying special purpose directives and system calls which are inherent in operating systems, we achieve explicit parallelism.

Degree of parallelism can be enhanced by adopting both the multi-core and multi-threaded computation model. Projective geometry plays a significant role in parallel computing by suitably assigning the processes to the appropriate processors. Perfect pattern and perfect matching techniques can enhance the performance of the parallel system.

## REFERENCES


1. Aberger C. R., Lamb A., Tu S. , Nötzli A., Olukotun K., and Re C. 2017. Emptyheaded: A relational engine for graph processing. ACM Trans. Database Syst.
2. Aggarwal A., Anderson R. J., and Kao M.-Y. 1989. Parallel depth-first search in general directed graphs. In STOC.
3. Alon N., Babai L., and Itai A. 1986. A fast and simple randomized parallel algorithm for the maximal independent set problem. J. Algorithms.
4. Anderson R. and Mayr E. W. A 1984. P-complete problem and approximations to it. Technical report.
5. Bader D. A. and Cong G. 2006. Fast shared-memory algorithms for computing the minimum spanning forest of sparse graphs. JPDC.
6. Beamer S., Asanovic K., and Patterson D. A. 2015. The GAP benchmark suite. CoRR, abs/1508.03619.
7. Ben-David N., Blelloch G. E., Fineman J. T., Gibbons P. B., Gu Y., McGuffey C., and Shun J. 2018. Implicit decomposition for write-efficient connectivity algorithms. In IPDPS.
8. Birn M., Osipov V., Sanders P., Schulz C., and Sitchinava N. 2013. Efficient parallel and external matching. In Euro-Par.
9. Blelloch G. E. 1993. Prefix sums and their applications. Synthesis of Parallel Algorithms.
10. Blelloch G. E. and Dhulipala L. 2018. Introduction to parallel algorithms 15-853: Algorithms in the real world.
11. Blelloch G. E., Fineman J. T., and Shun J. 2012. Greedy sequential maximal independent set and matching are parallel on average. In SPAAs.
12. Blelloch G. E., Gu Y., J. Shun, and Sun Y. 2016 Parallelism in randomized incremental algorithms. In SPAA.
13. Blelloch G. E., Y. Gu, and Y. Sun. 2017. A new efficient construction on probabilistic tree embedding. In ICALP.
14. Blelloch G. E., R. Peng, and Tangwongsan K.. 2011 Linear-work greedy parallel approximate set cover and variants. In SPAA.
15. Blelloch G. E., Simhadri H. V., and Tangwongsan K.. 2012. Parallel and I/O efficient set covering algorithms. In SPAA.
16. Blumofe R. D. and Leiserson C. E. Sept. 1999. Scheduling multithreaded computations by work stealing. J. ACM, 46(5).
17. Boldi P. and Vigna S. 2004. The Web Graph framework I: Compression techniques. In WWW.
18. Borůvka O. O jistém. 1926. problému minimálním. Práce Mor. Přírodověd. Spol. v Brně III.
19. Brandes U. 2001. A faster algorithm for betweenn ess centrality. Journal of mathematical sociology, 25(2).

20. 1666-2005 — IEEE Standard System C Language Reference Manual. 2006. doi: 10.1109/IEEESTD.2006.99475
21. (http:/ / dx. doi. org/ 10. 1109/ IEEESTD. 2006. 99475). ISBN 0-7381-4871-7.
22. 1666-2011 — IEEE Standard for Standard SystemC Language Reference Manual. 2012. doi:
23. 10.1109/IEEESTD.2012.6134619 (http:/ / dx. doi. org/ 10. 1109/ IEEESTD. 2012. 6134619).
24. ISBN 978-0-7381-6801-2.
25. Grötker T., Liao S., Martin G., Swan S. 2002. System Design with SystemC. Springer, ISBN 1-4020-7072-1
26. A SystemC based Linux Live CD with C++/SystemC tutorial (http:/ / sclive. wordpress. com/ )
27. Bhasker J., 2004. A SystemC Primer, Second Edition, Star Galaxy Publishing. ISBN 0-9650391-2-9
28. Black D. C., Donovan J., SystemC: Springer 2009 From the Ground Up, 2nd ed.,. ISBN 0-387-69957-0
29. Ghenassia Frank (Editor), Transaction-Level Modeling with SystemC: TLM Concepts and Applications for
30. Embedded Systems, Springer 2006. ISBN 0-387-26232-6
31. Liao Stan Y., Tjiang Steven W. K., Gupta Rajesh K.: An Efficient Implementation of Reactivity for Modeling
32. Hardware in the Scenic Design Environment. DAC 1997: 70-75
33. Hwang Kai: 2001. Advanced Computer Architecture: Parallelism, Scalability, Programmability (2001), Tata McGraw Hill.
34. Henessy J. L. and Patterson D. A. Computer Architecture: A Qualititative Approach, Morgan Kaufman (1990)
35. Rajaraman V. and Murthy C Shive Ram. Parallel Computer: Architecture and Programming: Prentice Hall of India
36. Salim G. Parallel Computation, Models and Methods: Prentice Hall of India