Name: A. Shiva Surya Saran
Roll No.: BE-B 20

High Performance Computing

Assignment 4

Unit 4: Analytical Model of Parallel Program

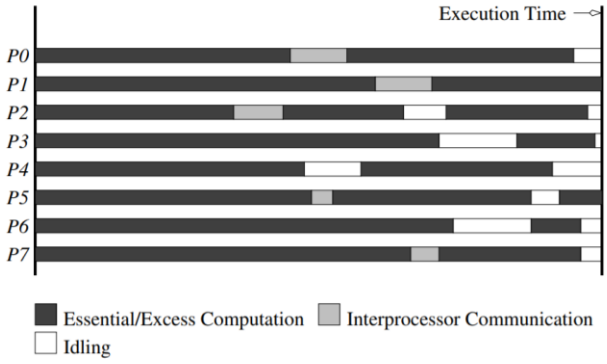
1. *Explain the basic Analytical Models with sources of overhead in Parallel Programs.*

Basic Analytical Models

- A sequential algorithm is evaluated by its runtime (in general, asymptotic runtime as a function of input size).
- The asymptotic runtime of a sequential program is identical on any serial platform.
- The parallel runtime of a program depends on the input size, the number of processors, and the communication parameters of the machine.
- An algorithm must therefore be analysed in the context of the underlying platform.
- A parallel system is a combination of a parallel algorithm and an underlying platform.

Sources of Overhead in Parallel Programs

Several overheads, including wasted computation, communication, idling, and contention cause degradation in performance.



The execution profile of a hypothetical parallel program executing on eight processing elements. Profile indicates times spent performing computation (both essential and excess), communication, and idling.

Inter-process interactions: Processors working on any non-trivial parallel problem will need to talk to each other.

Idling: Processes may idle because of load imbalance, synchronization, or serial components.

Excess Computation: This is computation not performed by the serial version. This might be because the serial algorithm is difficult to parallelize, or that some computations are repeated across processors to minimize communication.

2. *Describe the effects of Granularity on Performance.*

- Using fewer processors often improves performance of parallel systems.
- Using fewer than the maximum possible number of processing elements to execute a parallel algorithm is called scaling down a parallel system.
- A naive way of scaling down is to think of each processor in the original case as a virtual processor and to assign virtual processors equally to scaled down processors.
- Since the number of processing elements decreases by a factor of n/p, the computation at each processing element increases by a factor of n/p.
- The communication cost should not increase by this factor since some of the virtual processors assigned to a physical processor might talk to each other. This is the basic reason for the improvement from building granularity.

3. *Explain the Scalability of Parallel System with Amdahl's Law.*

- A parallel architecture is said to be scalable if it can be expanded (reduced) to a larger (smaller) system with a linear increase (decrease) in its performance (cost). This general definition indicates the desirability for providing equal chance for scaling up a system for improved performance and for scaling down a system for greater cost-effectiveness and/or affordability.
- Scalability is used as a measure of the system's ability to provide increased performance, for example, speed as its size is increased. In other words, scalability reflects the system's ability to efficiently utilize the increased processing resources.
- The scalability of a system can be manifested as in the forms, speed, efficiency, size, applications, generation, and heterogeneity.
- In terms of speed, a scalable system can increase its speed in proportion to the increase in the number of processors.
- In terms of efficiency, a parallel system is said to be scalable if its efficiency can be kept fixed as the number of processors is increased, provided that the problem size is also increased.
- Size scalability measures the maximum number of processors a system can accommodate.
- Application scalability; refers to the ability of running application software with improved performance on a scaled-up version of the system.

Amdahl's law

Amdahl's law is a formula which gives the theoretical speedup in latency of the execution of a task at a fixed workload that can be expected of a system whose resources are improved. It is a formula used to find the maximum improvement possible by just improving a particular part of a system. It is often used in parallel computing to predict the theoretical speedup when using multiple processors.

Speedup is defined as the ratio of performance for the entire task using the enhancement and performance for the entire task without using the enhancement or speedup can be defined as the ratio of execution time for the entire task without using the enhancement and execution time for the entire task using the enhancement.

If Pe is the performance for entire task using the enhancement when possible, Pw is the performance for entire task without using the enhancement, Ew is the execution time for entire task without using the enhancement and Ee is the execution time for entire task using the enhancement when possible then,

Speedup = Pe/Pw or Speedup = Ew/Ee

Amdahl's law uses two factors to find speedup from some enhancement –

Fraction enhanced –
The fraction of the computation time in the original computer that can be converted to take advantage of the enhancement. For example- if 10 seconds of the execution time of a program that takes 40 seconds in total can use an enhancement , the fraction is 10/40. This obtained value is Fraction Enhanced. *Fraction enhanced is always less than 1.*

Speedup enhanced –
The improvement gained by the enhanced execution mode; that is, how much faster the task would run if the enhanced mode were used for the entire program. For example – If the enhanced mode takes, say 3 seconds for a portion of the program, while it is 6 seconds in the original mode, the improvement is 6/3. This value is Speedup enhanced. *Speedup Enhanced is always greater than 1.*

The overall Speedup is the ratio of the execution time: -

$$\text{Overall Speedup} = \frac{\text{Old execution time}}{\text{New execution time}}$$

$$= \frac{1}{\left((1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right)}$$

4. Describe the Cost Optimality and Optimal Execution time.

We are often interested in knowing how fast a problem can be solved, or what the minimum possible execution time of a parallel algorithm is, provided that the number of processing elements is not a constraint.

As we increase the number of processing elements for a given problem size, either the parallel runtime continues to decrease and asymptotically approaches a minimum value, or it starts rising after attaining a minimum value. We can determine the minimum parallel runtime T_P^{min} for a given W by differentiating the expression for T_P with respect to p and equating the derivative to zero (assuming that the function T_P (W, p) is differentiable with respect to p). The number of processing elements for which T_P is minimum is determined by the following equation:

$$\frac{\mathrm{d}}{\mathrm{d}p} T_P = 0$$

5. Describe the Dense Matrix Algorithm and matrix vector multiplication.

Algorithms involving matrices and vectors are applied in several numerical and non-numerical contexts. Due to their regular structure, parallel computations involving matrices and vectors readily lend themselves to data-decomposition. Depending on the computation at hand, the decomposition may be induced by partitioning the input, the output, or the intermediate data.

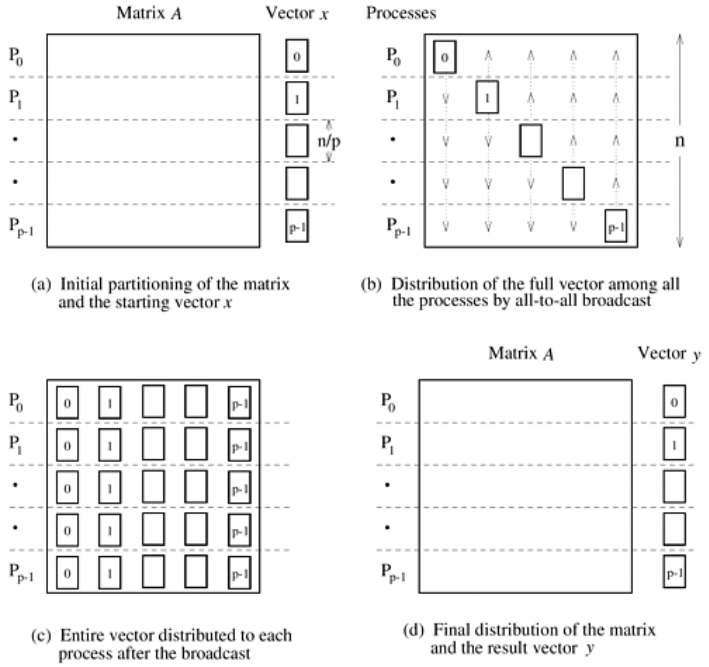
Matrix-Vector Multiplication

A serial algorithm for multiplying an n x n matrix A with an n x 1 vector x to yield an n x 1 product vector y.

```
1.   procedure MAT_VECT ( A, x, y)
2.   begin
3.      for i := 0 to n - 1 do
4.      begin
5.         y[i]:=0;
6.         for j := 0 to n - 1 do
7.             y[i] := y[i] + A[i, j] x x[j];
8.      endfor;
9.   end MAT_VECT
```

Rowwise 1-D Partitioning

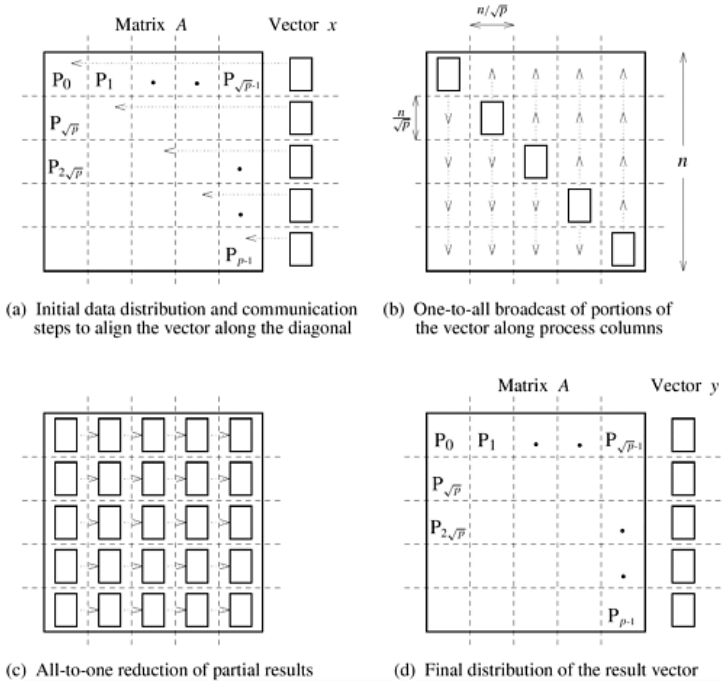
The parallel algorithm for column wise block 1-D partitioning is similar and has a similar expression for parallel run time. Figure describes the distribution and movement of data for matrix-vector multiplication with block 1-D partitioning.



Multiplication of an n x n matrix with an n x 1 vector using rowwise block 1-D partitioning. For the one-row-per-process case, p = n.

2-D Partitioning

Parallel matrix-vector multiplication for the case in which the matrix is distributed among the processes using a block 2-D partitioning. Figure shows the distribution of the matrix and the distribution and movement of vectors among the processes.



Matrix-vector multiplication with block 2-D partitioning. For the one-element-per-process case, p = n2 if the matrix size is n x n.

Matrix-Matrix Multiplication

Parallel algorithms for multiplying two n x n dense, square matrices A and B to yield the product matrix C = A x B. If we assume that an addition and multiplication pair take unit time, then the sequential run time of this algorithm is n3. Matrix multiplication algorithms with better asymptotic sequential complexities are available, for example Strassen's algorithm. However, for the sake of simplicity, we assume that the conventional algorithm is the best available serial algorithm.

The conventional serial algorithm for multiplication of two n x n matrices.

```
1.   procedure MAT_MULT (A, B, C)
2.   begin
3.      for i := 0 to n - 1 do
4.          for j := 0 to n - 1 do
5.              begin
6.                 C[i, j] := 0;
7.                 for k := 0 to n - 1 do
8.                     C[i, j] := C[i, j] + A[i, k] x B[k, j];
9.              endfor;
10.  end MAT_MULT
```