Analytical Modeling of Parallel Programs

Hardik K. Molia
Master of Computer Engineering,
Department of Computer Engineering
Atmiya Institute of Technology & Science,
Rajkot- 360005,
Gujarat, India

hardik.molia@gmail.com

***Abstract*— A parallel program should be evaluated to determine its efficiency, accuracy and benefits. This paper defines how parallel programs differ by sequential programs. A brief discussion on the effect of increasing number of processors on execution time is given. Some of the important measurement units which are used for the purpose of measuring performance of a parallel program are discussed. Various performance laws -Amdahl's Law, Gustafson's Law to measure speedup are discussed.**

***Index Terms*— Speedup, Efficiency, Granularity, Excess Computations Amdahl's Law, Gustafson's Law**

I. Introduction

A sequential program has a single flow of instructions and thus the execution flow is sequential. It can be executed on a single processor system. A parallel program may have multiple flows of instructions that can be executed on all or some of the available processors together, inhibiting the base condition of being a parallel program. A parallel program can be executed in the system which supports multiple executions at a time called multi processor computer where multiple processors are available for the computation purpose.

Example of Real Life Parallelism

100 students have participated in a quiz. If we hire only a single person to assess all the 100 answer sheets, we get all the answer sheets assessed after (100 * t) time where t is the average time to assess a single answer sheet. This approach is the sequential approach. Lets hire 5 people for the same work, if we divide 20 answer sheets to each of them, we get all the answer sheets assessed after (20 * t) time where t is the average time to assess a single answer sheet.

Here we can see that the parallelism has improved the speed by which we can complete the task, but at the same time it becomes costly as we need to hire 5 people at a time. We need to define rules of work distribution as well as final gathering phase.

Example of Computerised Parallelism

A sequential approach to calculate addition of n numbers requires (n) time units. If we take k processors (assuming k is a factor of n) each of them gets n/k numbers to perform addition, we can say the execution time will be decreased to n/k. But some additional time is required for the purpose of data distribution and for the calculation of final sum from the partial sums available with all the k processors.

Evaluation of Parallelism

One natural question arise in our mind is that "if we use two processors then will our program run twice faster?" the answer is "It depends upon what the algorithm is, but in reality increasing number of processors, affects the speed of execution but not necessarily linearly.

A sequential program is evaluated by its execution time, as a function of input size by using asymptotic notations. The main concentration here is to take number of primitive operations required to be executed into count. Various notations, Big O, Big Ω, Big Θ corresponding to the worst case, best case and average (expected) case complexities are considered as measuring tools of sequential programs.

A parallel program is evaluated by its execution time as a function of input size, number of processors and some communication parameters. The next sections describe how the increase in the number of processors affects the execution time and how to evaluate a parallel program efficiently.

II. SOURCES OF OVERHEADS

We discussed a natural question, "if we use two processors then will our program run twice faster?"
In this section we are going to define the answer precisely.

In reality, it is the rarely case where the execution time decreases linearly with respect to the increase in the number of processors. The reason behind this logic is the presence of various overheads. Overheads are the extra activities in which a computer remains busy other than the actual execution.

Inter process Communication

A parallel program divides and distributes the input to the available processors. Every processor has a process to manipulate/process/compute its local data set .At the end of the working with local data, every process has certain results those are required to one of the participating processes mostly master process, so that the master process can combine results of all the processes and produce the final result.

Let we want to perform addition of {1, 2, 3, 4, 5, 6}
Let's have three processors each of them is executing a process. 6 data elements are equally distributed across 3 processors (processes $P_{0,}$ $P_{1,}$ P_2 with P_0 as the master-parent process which will combine all the partial values) as shown below. the summation process performs by following these steps. It doesn't show the sequence as some of the steps can be executed in parallel manner.

1.Process P_0 has assigned elements {1,2}
2.Process P_1 has assigned elements {3,4}
3 Process P_2 has assigned elements {5,6}
4.Process P_0 calculates partial sum {1+2=3}
5.Process P_1 calculates partial sum {3+4=7}
6.Process P_2 calculates partial sum {5+6=11}
7.process P_1 sends its partial sum {3} to P_0
8.process P_2 sends its partial sum {7} to P_0
9.P_0 calculates the final results as 3+7+11 = 21

Other than the time required to distribute data across the processors (step 1,2,3), in step 7 and 8, we are transferring data between processors. Time required to perform such data distribution and then communication is known as inter process communication. As it is in addition of the actual execution time, we consider it as an overhead.

Load Imbalance

In a parallel computer, all the processors should loaded with the work equally, the scenario when some of the processors are extremely busy while some other processors are idle or comparatively very less busy is an example of imbalance of load. In case of a symmetric multiprocessor system
(All processors are of same architecture), input data must be evenly distributed over the processors. In case of asymmetric multiprocessor system (where different processor may have different architecture), data elements shouldn't be strictly distributed but processors should find data to process from pool of work.

Process migration is very important task in a multiprocessor system. Time required for the load balance must be less than the improved execution time otherwise it introduces more overhead. Load balancing should be fast at the same time efficient, accurate and effective too.

Excess Computation

A parallel program may have some operations which are not part of the main execution, it has no direct connection with the final results but it helps to find the results precisely. In shared memory architecture, processes have to maintain consistency while writing shared values. The code which keeps trying to enter into a critical region is an example of excess computation where the goal is to improve the accuracy of parallelism. We can define the difference between the computation required by sequential approach and computation required by parallel approach as excess computation.

III. Speedup and Efficiency

Speedup is a measurement to get information about the amount of performance gain we can achieve by replacing a sequential program by a logically equivalent parallel program. Speedup is measured by comparing execution times of two implementations of an algorithm. Let Ts – time required to execute most efficient sequential program and Tp – time required to execute most efficient parallel program.

General Definition

Speedup can be defined as the ratio of Ts and Tp, time required to execute a most efficient sequential program on sequential computer divided by time required to execute a parallel program on parallel computer.

.

$$\mathrm{Sp} = \frac{\mathrm{Ts}}{\mathrm{Tp}}$$

Architecturally Bounded Definition

Sequential computers and parallel computers differ drastically in terms of architectures. Even sequential computers of different manufacturers as well as of different generations differ by the way they compute. So to measure the speedup efficiently, it is required to analyse the sequential and parallel programs on same architecture. With architecturally bounded definition, we modify the times with,

Ts -Time required to execute most efficient sequential program on parallel computer.

Tp -Time required executing most efficient parallel program on parallel computer.

Here the programs are different but parallel computer on which we execute them is same.

Algorithmically & Architecturally Bounded Definition

There are certain differences in the way sequential and parallel programs can be written for a problem. Sequential programs are easy to write as they don’t have implementation of inter process communication, shared memory access, or mutual exclusion handling. Parallel programs have communication overheads other than the actual execution.

This definition states the speedup as a measurement to find the relationship between execution of a parallel program as a sequential as well as a parallel one. we can redefine the definitions of speedup with.

Ts -Time required to execute most efficient parallel program on a single processor of a parallel computer.

Tp -Time required to execute most efficient parallel program on all processors of a parallel computer.

Here the program as well as the parallel computer on which we execute them is same.

Efficiency

Efficiency measures how effectively resources of a parallel computer (multiple processors) can be used to solve a problem. Speedup defines how fast a parallel program can be executed with reference of its sequential implementation. Efficiency defines how efficiently resources are utilised. E is a measure of the time a specific processor is usefully used.

$$\mathrm{E} = \frac{\mathrm{Sp}}{\mathrm{p}}$$

Sp = Speedup, P = No of processors

Example

An arithmetic summation can be performed for n elements in (n) time sequentially. If we use the parallel reduction approach than the same problem can be solved in ($\log_2 n$) time with n processors on hypercube organization. Speedup Sp and Efficiency E can be defined as follow.

$$\mathrm{Sp} = \frac{\mathrm{n}}{\log_2 \mathrm{n}}$$

$$E = \frac{n/\log_2 \mathrm{n}}{n} = \frac{1}{\log_2 \mathrm{n}}$$

IV. EFFECT OF IMPROVING RESOURCES

It is not the always case when increasing number of processors will increase the performance. It may happen for a while, up to certain level of increase, but after a saturation point, increasing resources yields splitting and distributing a problem more and more which degrades the overall performance.

Effect of Number of Processors on Communication Time

As the number of processors increases, more communication is required to work together. Creation, control, coordination and communication tasks require more time.

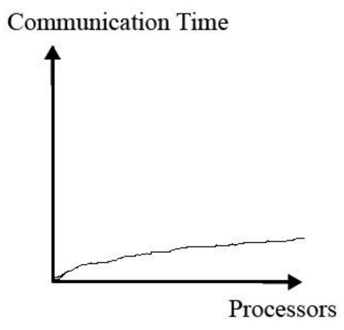

Fig. 1- Communication Time Vs Processors

Effect of Number of Processors on Execution Time

Ideally, execution time should be decreased linearly with respect to the increase in number of processors. But due to increase in communication time, it doesn't.

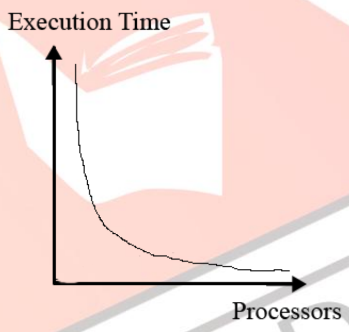

Fig. 2- Execution Time Vs Processors

Effect of Number of Processors on Speedup

As number of processors increase, idly speedup should also increase linearly. But because of overheads speed up doesn't increase linearly. In reality we get sub linear speed up only. The three categories of speedup are shown in figure

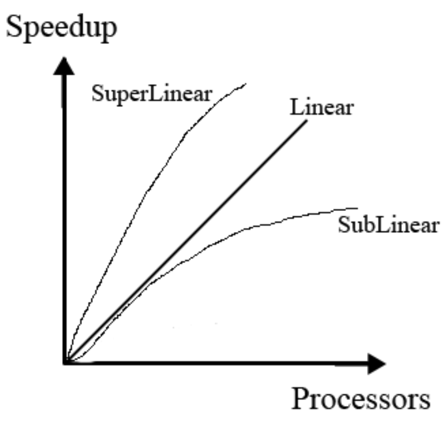

Fig. 3- Speedup Vs Processors

Sub Linear Speedup

Communication overhead is one of the main reasons of getting sub linear speedups in reality. As increase in number of processors (assuming all the processors are involved in execution of a parallel program), increases the inter process communication and restrict the total execution time.

Another reason is the memory. The rate at which processors can process the date is generally faster than the rate at which memory is able to supply the data. Use of cache memory and following spatial locality for static data strictures in programs may improve the speedup.

Inherently sequential nature may degrade the performance even if the resources are free. Poor load balancing can be a factor to restrict speedup.

Super Linear Speedup

In reality we get sub linear speedup. But there are certain cases, or certain instances of programs where we may get speedup which is super linear.

To find speedup through general or architecturally bounded definitions, we use the most efficient sequential program. Here the meaning of the most efficient is the one which requires less time of execution. But in reality, sometimes efficiency of a program depends upon the input instance. So in a case when a program gives worst result, speedup can become super linear even if the parallel program is average efficient.

A single processor system has a single cache and suffers from moderate miss ration. A parallel system has multiple processors and having their own cache memories. This may decrease the total miss ration as the data is distributed across the processors. All processors have small fraction of data that can be completely loaded in cache, but the whole data can't be loaded into a single cache in case of single processor system.

V. EFFECT OF GRANULARITY

A parallel program is an implementation of an algorithm where a problem is divided-decomposed into a set of sub tasks each of them can be performed simultaneously. Granularity refers to the number and size (number of elements to process, number of resources required, memory requirement). A decomposition of large number of small tasks is called fine grained and a decomposition of small number of large tasks is called course grained.

Let's revised the addition example. For the simplicity of discussion, we assume that we are having infinite number of free processors.

A Fine Grained Decomposition

Data= {1, 2, 3, 4, 5, 6}. Let's take 6 processors and distribute single element to each of them. Arithmetic addition of Data will be performed in following way.

Step 1- Data Distribution

$P_1 = 1$	$P_2 = 2$
$P_3 = 3$	$P_4 = 4$
$P_5 = 5$	$P_6 = 6$

Step 2- Parallel Reduction. Every even number processor will send its partial sum to previous (as per label as well as which has latest data) processor to sum up.

$P_2 = 2$ ------→ $P_1 = 1+2 = 3$
$P_4 = 4$ ------→ $P_3 = 3+4 = 7$
$P_6 = 6$ ------→ $P_5 = 5+6 = 11$
$P_3 = 7$ ------→ $P_1 = 3+7 = 10$
$P_5 = 6$ ------→ $P_1 = 10+11 = 21$

Here we can see that the communication overhead is comparatively high.

A Corse Grained Decomposition

Data= {1, 2, 3, 4, 5, 6}. Let's take 3 processors and distribute 6/3=2 elements to each of them. Arithmetic addition of Data will be performed in following way.

Step 1- Data Distribution
$P_1 = 1,2$ $P_2 = 3,4$ $P_3 = 5,6$

Step 2- Local sum
$P_1 = 3$ $P_2 = 7$ $P_3 = 11$
Step 3- Parallel Reduction. Every even number processor will send its partial sum to previous (as per label as well as which has latest data) processor to sum up.

$P_2 = 7$ ------→$P_1 = 3+7 =10$
$P_3 = 11$ ------→$P_1 = 10+11 =21$

Here we can see that though local computation is increased, the communication overhead is less.

Scaling Down a Parallel System

Using less number of processors than the maximum possible is known as scaling down a parallel system. Consider the situation where n number of elements are required to be processed by p processors (n>p). Each processor can assign n/p elements to process. As the number of processors is decreased by factor n/p. Local computation at any processor is increased by factor n/p.

VI. Measuring Speedup

The goal of improving the number of processors can be either to decrease execution time or to increase accuracy of computation. Based on the purpose of using multiple processors, speedup can be measured by following rules.

Amdahl's Law

Amdahl's law defines the effect of increasing number of processors on execution time of a parallel program. The main goal of doing Amdahl analysis is to find how much execution time we can decrease by increasing number of processors.

Every parallel program has some operations which are inherently sequential (reading or writing files, sequential device access, inter dependent code execution, data distribution across the processors, final results calculations). These operations require to be executed in sequential approach. Let the time required to execute sequential operations is Ts- a fraction f ($0 < f \leq 1$).

The rest of the operations can be executed in parallel. The factor (1-f) denotes the time, Tp to execute operations in parallel. The speedup Sp can be defined as,

$$\mathrm{Sp} \leq \frac{1}{f + \frac{(1-f)}{n}}$$

Total time execution is T(1) which can be considered as sequential program's execution time. Fraction (f) defines the inherent sequential part which can't be parallelised. Fraction (1-f) defines the part which can be parallelised and with n processors it can be executed by factor n.
The effect of varying amount of inherently sequential operations on speedup, with increase in number of processors is shown in figure below.

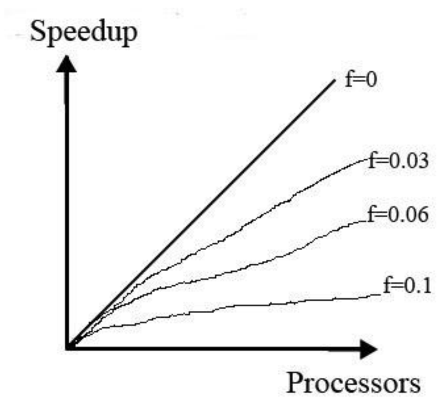

Fig. 4 – Effect of sequential fraction on speedup

Gustafson's Law

Amdahl's law focuses on faster execution by distributing work load to more processors. Gustafson's law focuses on increasing accuracy, precision and efficiency of calculations by using more processors.

Let Ts is the time to execute sequential operations and Tp is the time to execute parallel operations. W is total workload and n is number of available processors The speedup Sp can be defined as,

$$\mathrm{Sp} = \frac{\mathrm{Ts} + \mathrm{Tp}(1, \mathrm{W})}{\mathrm{Ts} + \mathrm{Tp}(\mathrm{n}, \mathrm{W})}$$

Where Tp(n,W) is the time required to perform W workload by processors assuming that the parallel operations achieve a linear speedup, n processors take 1/nth of the total execution time.

$$\mathrm{Tp}(1, \mathrm{W}) = \mathrm{n} * \mathrm{Tp}(\mathrm{n}, \mathrm{W})$$

Let f be the fraction which defines inherent sequential execution, f can be defined as,

$$\mathrm{f} = \frac{\mathrm{Ts}}{\mathrm{Ts} + \mathrm{Tp}(\mathrm{n}, \mathrm{W})}$$

The speedup can be rewritten as,

$$\mathrm{Sp} = \frac{\mathrm{Ts} + \mathrm{n} * \mathrm{Tp}(\mathrm{n}, \mathrm{W})}{\mathrm{Ts} + \mathrm{Tp}(\mathrm{n}, \mathrm{W})}$$

$$\mathrm{Sp} = \frac{\mathrm{Ts}}{\mathrm{Ts} + \mathrm{Tp}(\mathrm{n}, \mathrm{W})} + \frac{\mathrm{n} * \mathrm{Tp}(\mathrm{n}, \mathrm{W})}{\mathrm{Ts} + \mathrm{Tp}(\mathrm{n}, \mathrm{W})}$$

$$\mathrm{Sp} = \mathrm{f} + \mathrm{n} * (1 - \mathrm{f})$$

$$\mathrm{Sp} \approx \mathrm{n} * (1 - \mathrm{f}) \qquad \text{(For larger n)}$$

VII. CONCLUSIONS

Problem size of an algorithm is defined as the number of primitive operations in the most efficient sequential program. For a given problem size, increasing the number or processors decreases the speedup and efficiency because of increase in communication overheads. In many cases, speedup and efficiency increase with increase in the problem size while keeping the number of processors constant. The conclusion is that we should develop our program which keeps speedup and efficiency constant with increase in problem size and the number of processors.

Increasing problem size is equivalent to increasing number of input values to process. Generally increasing input size affects the number of inherent sequential operations but it affects more on the number of parallel operations. And so we can control the speedup and efficiency at a level while increasing problem size.

There are some algorithms whose performances depend upon instances of them at any particular time. Such algorithms are difficult to model analytically as they don't have certain regularity to model. Probabilistic and subjective analysis helps to model such algorithms.

REFERENCES

[1] Gene M. Amdahl –"Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities"- AFIPS Spring Joint Computer Conference -1967

[2] Hwang, K. and Zu, Z. –"Scalable Parallel Computing, Technology, Architecture, Programming". WCB/McGraw-Hill USA, 1998

[3] Gustafson, J.L. -"Reevaluating Amdahl's Law", Communications Of ACM Vol 31, No. 5 May 1998,

[4] Hwang K. -"Advanced Computer Architecture: Parallelism, Scalability, and Programmability." McGraw-Hill Inc. Singapore 1993.

[5] Quinn Michael J. -"Parallel Computing, Theory And Practice. "McGraw-Hill Inc India 2002