ISSN:0975-9646
Juby Mathew et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 2 (6) , 2011, 2796-2799

# The Performance of Parallel Algorithms by Amdahl's Law, Gustafson's Trend

[1]Juby Mathew, [2]Dr.R Vijayakumar

[1]*Department of CSE, Amaljyothi College of Engg. Kanjirapally*
[2]*School of Computer Science, Mahatma Gandhi University Kottayam,Kerala, India*

**Abstract-Parallelization is a core strategic-planning consideration for all software makers, and the amount of performance benefit available from parallelizing a given application (or part of an application) is a key aspect of setting performance goals for the parallelization process. Theoretical discussions of performance potential are necessarily the starting point for understanding the critical issues involved, before moving to practical issues associated with a given project. Amdahl's Law and its modification by Gustafson's trend give us the basic means to understand what's possible for a given application, and tools and best practices give us the means to decide how to use that information in practice.**

***Key words: Parallel computing; parallel processing; parallel speedup; parallel efficiency; Gustafson's Trend***

## OVERVIEW

Parallel computers consisting of thousands of processors are now commercially available. These computers provide many orders of magnitude more raw computing power than traditional supercomputers at much lower cost. They open up new frontiers in the application of computers—many previously unsolvable problems can be solved if the power of these machines is used effectively Analyzing the performance of a given parallel algorithm/architecture calls for a comprehensive method that accounts for scalability: a measure of a parallel system's capacity to effectively utilize an increasing number of processors. There has been extensive work in investigating the performance and scalability properties of large scale parallel systems and several laws governing their behavior have been proposed.
This paper provides an overview of Amdahl's Law and Gustafson's Trend, placing them in the context of current development considerations

## THE PERFORMANCE OF PARALLEL ALGORITHMS EXECUTED ON MULTIPROCESSOR SYSTEMS

The first criterion taken into consideration when the performances of the parallel systems are analyzed is the speedup used to express how many times a parallel program works faster than a sequential one, where both programs are solving the same
problem. The most important reason of parallelization a sequential program is to run the program faster.[1]
The speedup formula is

$$S = \frac{T_s}{T_p}$$

Where **Ts** is the execution time of the fastest sequential program that solves the problem

**Tp** is the execution time of the parallel program used to finalize the same problem.

If a parallel program is executed on a computer having ***p*** processors, the highest value that can be obtained for the speedup is equal with the number of processors from the system. The maximum speedup value could be achieved in an ideal multiprocessor system where there are no communication costs and the workload of processors is balanced. In such a system, every processor needs **Ts/p** time units in order to complete its job so the speedup value will be as the following:

$$S = \frac{T_s}{\frac{T_s}{p}} = p$$

There is a very simple reason why the speedup value cannot be higher than **p** – in such a case, all the system processors could be emulated by a single sequential one obtaining a serial execution time lower than **Ts**. But this is not possible because **Ts** represents the execution time of the fastest sequential program used to solve the problem.

According to the *Amdahl law*, it is very difficult, even into an ideal parallel system, to obtain a speedup value equal with the number of processors because each program, in terms of running time, has a fraction α that cannot be parallelized and has to be executed sequentially by a single processor. The rest of **(1 - α)** will be executed in parallel.

The parallel execution time and the speedup will become:

$$T_p = T_s \cdot \alpha + T_s \cdot (1-\alpha)/p$$

$$S = \frac{T_s}{T_p} = \frac{T_s}{T_s \cdot \alpha + T_s \cdot (1-\alpha)/p} = \frac{1}{\alpha + (1-\alpha)/p} = \frac{p}{\alpha \cdot (p-1) + 1}$$

When $p \longrightarrow \infty$, we have

$$\lim_{p \to \infty} S = \frac{1}{\alpha}$$

The maximum speedup that could be obtained running on a parallel system a program with a fraction **α** that cannot be parallelized is **1/ α**, no matter of the number of processors from the system.

For example, if a program fraction of 20% cannot be parallelized on a four processors system, the parallel execution time and the speedup will be equal with:

$$T_p = T_s \cdot 0.2 + T_s \cdot 0.8/4 = 0.4 \cdot T_s$$

$$S = \frac{T_s}{0.4 \cdot T_s} = \frac{1}{0.4} = 2.5$$

The parallel execution time will be 40% of the serial execution time and the parallel program will be only 2.5 times faster than the sequential one because 20% of the program cannot be parallelized (figure 1). The maximum speedup that we can obtain is 1/0.2 = 5 and this means that the parallel execution time will never be shorter than 20% of the sequential execution time even in a system with an infinite number of processors.

Amdahl law concludes it is very important to identify the fraction of a program than cannot be parallelized and to minimize it.
The *parallel efficiency* quantifies the number of the valuable operations performed by the processors during the parallel program execution. The parallel efficiency could be expressed as the following:

$$E = \frac{S}{p}$$

Where **S** is the speedup and **p** represents the number of the processors or cores from the system.

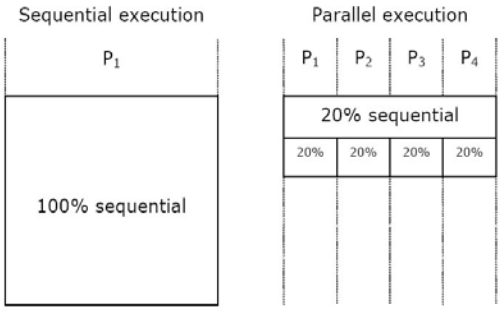



Fig.1. Parallel execution on an ideal system

Due to the fact the speedup value is lower than the number of processors; the parallel efficiency will be always located between 0 and 1.
Another important indicator is the execution cost representing the total processor time used to solve the problem. For a parallel application, the parallel cost could be calculated according with the following formula:[2]

$$C_p = p \cdot T_p$$

For a sequential program, its cost (sequential cost) will be equal with the total execution time:

$$C_s = T_s$$

For this reason, the parallel efficiency could be also expressed as the following:

$$E = \frac{S}{p} = \frac{T_s / T_p}{p} = \frac{T_s}{p \cdot T_p} = \frac{C_s}{C_p}$$

Finally, the supplementary cost of parallel processing indicates the total processor times spent for secondary operations not directly connected with the main purpose of the program that is executed. Such a cost cannot be identified for a sequential program.

$$C_{sup\,l} = C_p - C_s = p \cdot T_p - T_s$$

The figure 2 presents the way in which a parallel program will be executed on a real 4 processor system. This time, the program contains a fraction of 20% that cannot be parallelized, the load of the processors is not balanced and the communications times are not neglected anymore.

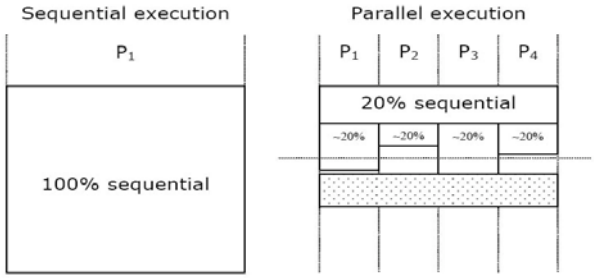



Fig.2. Parallel execution on a real system

The source of this type of cost is represented by the following elements:

- **Load imbalance** – is generated by the unbalanced tasks that are assigned to different processors. In such a case, some processors will finish the execution earlier so they need to wait in an idle state for the other tasks to be completed. Also, the presence of a program fraction that cannot be parallelized generates load imbalance because this portion of code should be executed by a single processor in a sequential manner.
- **Supplementary calculations** – generated by the need to compute some value locally even if they are already calculated by another processor that is, unfortunately, busy at the time when these data are necessary.-
- **Communication and synchronization between processors** – the processors need to communicate each others in order to obtain the final results. Also, there are some predefined execution moments when some processors should synchronize their activity.

In order to obtain a faster program, we can conclude we need to reduce to the minimum the fraction that cannot be parallelized, to assure the load balance of the tasks at the processor level and also to minimize the times dedicated for communication and synchronization.

**Gustafson: Adding Due Consideration for Large-Scale Resources and Tasks**

Amdahl shows that increasing the parallelism of the computing environment by some number **N** (e.g., providing N times the number of processors or cores) can never increase performance by a factor of N. The two main factors contribute to this limitation are the presence of the inherently serial portion of the computational load (the performance of which cannot be improved by parallelization) and the overhead associated with parallelization. That overhead consists of such factors as creating and destroying threads, locking data to prevent multiple threads from manipulating it simultaneously, and synchronizing the computations performed among various threads to obtain a coordinated result.
Amdahl's Law [3], quantifies the theoretical speedup that can be obtained by parallelizing a computational load among a set number of processors. Another way of expressing that relationship is given in Equation 1.

**Equation 1. Representation of Amdahl's Law**

$$\text{Speedup}(N) = \frac{1}{S + \frac{1-S}{N}} - O_N$$

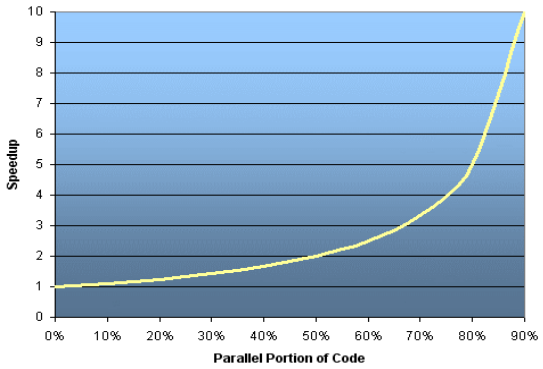
Where:
N = Number of processor cores
S = Serial percentage of workload (expressed as a decimal in the range 0-1)
$O_N$ = Parallelization overhead for N threads

A simplified case of Equation 1 helps to illuminate the relationship being shown.

Consider the equation with **S** (the serial, un-parallelizable portion of the workload) equal to zero, meaning that the workload is fully parallelizable; in this case, the speedup is equal to $\mathbf{N + O_N}$.

Further simplifying that expression by setting $\mathbf{O_N}$ equal to zero (removing the parallelization overhead) reduces the equation to **Speedup (N) = N**. Therefore, for example, if one neglects both the serial component of the workload and the parallelization overhead (the ideal case), the speedup from splitting a workload from one processor onto two processors produces a speedup of **2x**, splitting it onto eight cores would yield a speedup of **8x**, etc.

Further, viewing Equation 1 as the subtraction of $\mathbf{O_N}$ from a complex fraction, the complex fraction represents the speedup without being adjusted for threading overhead.




To illustrate the limitations on possible performance gains from parallelizing workloads (which was Amdahl's actual intent), consider the effect on Equation 1 when **N** tends toward infinity and $\mathbf{O_N}$ tends toward zero. That represents the case where infinitely parallel processing capacity is available, without any overhead from parallelization, and it therefore demonstrates the theoretical upper limit to the performance increase available from parallelization. As N becomes infinitely large, the expression **(1 - S) / N** becomes infinitely small, so that the specialized case of Equation 1 with infinitely parallel resources and zero parallelization overhead is reduced to the expression shown in Equation 2.

**Equation 2. A specialized case of Amdahl's Law with infinitely parallel execution resources and zero parallelization overhead**

$$\text{Speedup (upper limit)} = \frac{1}{S}$$

Where:
S = Serial percentage of workload (expressed as a decimal in the range 0-1)

In 1988, John Gustafson, working with E. Barsis, helped to refine Amdahl's model by adjusting some of its underlying assumptions. That is, whereas Amdahl's Law indicates that the speedup from parallelizing any computing problem is inherently limited by the presence of serial (non-parallelizable) portions, Gustafson's Trend posits that this is an incomplete relationship. Gustafson argues that, as processor power increases, the size of the problem set also tends to increase. To cite one obvious example: as mainstream computational resources have increased, computer games have become far more sophisticated, both in terms of user-interface characteristics and in terms of the underlying physics and other logic.

Simply, as compute resources increased, the problem size also increased, and the inherently serial portion became much smaller as a proportion of the overall problem. Because Amdahl's Law cannot address this relationship, [5] ***Gustafson modifies Amdahl's work according to the precept that the overall problem size should increase proportionally to the number of processor cores (N), while the size of the serial portion of the problem should remain constant as N increases***. The result is shown in Equation 3

**Equation 3. A computational representation of Gustafson's Trend**

$$\text{Speedup}(N) = \frac{S + N(1-S)}{S + (1-S)} - O_N$$

Where:
N = Number of processor cores
S = Serial percentage of workload (expressed as a decimal in the range 0-1)
$O_N$ = Parallelization overhead for N threads

In this equation, note first that **S** represents the serial proportion of the *unscaled* workload; that is, unlike in Amdahl's Law (Equations 1 and 2), S remains steady in the numerator versus denominator as a quantity of work, rather than as a proportion of the overall work. That is, while the parallel portion of the workload $\mathbf{(1 - S)^2}$ scales with the number of processor cores in the numerator of the equation, the serial portion **(S)** does not. Obviously, Equation 3 can be easily simplified by adding the components of the denominator together, and by doing so as well as eliminating (for the moment) the effect of parallelization overhead, Gustafson's trend[4] reduces to the relationship shown in Equation 4.

**Equation 4. A simpler representation of Gustafson's Trend**

$$\text{Speedup}(N) = S + N(1-S)$$

Where:
N = Number of processor cores
S = Serial percentage of unscaled workload (expressed as a decimal in the range 0-1)

Taking the most extreme case first, according to this simplified version of Gustafson's trend, scaling the number of processor cores toward infinity should result in a speedup that also scales toward infinity. Of course, infinite numbers of cores are not directly relevant to real-world implementations, but this relationship is instructive as a comparison with Amdahl's Law. To see more clearly what the effect of increasing the number of cores on a specific workload might be, consider a computational load that is 10 percent serial, where the serial portion remains a fixed size and the parallel portion increases in size proportionally to the number of processor cores, as called for in Gustafson's Trend. Table 1 shows the projected result as the number of processor cores applied to the theoretical problem is increased.

**Table 1**. Gustafson's Trend applied to a hypothetical problem being scaled to various numbers of processors

| # cores | Computation | Speedup | Efficiency (speedup / # cores) |
|---|---|---|---|
| 2 | 0.1 + 2 ( 1 -0 .1) | 1.9x | 95.00% |
| 4 | 0.1 + 4 ( 1 -0 .1) | 3.7x | 92.50% |
| 32 | 0.1 + 32 ( 1 -0 .1) | 28.9x | 90.31% |
| 1024 | 0.1 + 1024 ( 1 -0 .1) | 921.7x | 90.01% |

Clearly, these calculations show that the performance result continues to scale upward as more processor cores are applied to the computational load. It's also worth noting that the per-core efficiency trends downward as additional cores are added, although the data in Table 1 shows the decrease in per-core efficiency between the two-core case and the four-core case to be greater than the entire decrease between four cores and 1024 cores. On the other hand, this relationship does not take parallelization overhead into account, which obviously increases dramatically as the number of threads (and therefore the complexity of the associated thread management) increases.

## CONCLUSION

Amdahl's and Gustafson's theoretical constructs about the performance limits of parallelization are an important foundation to our understanding of how future software will manifest the power of future hardware. Placed in the context of real-world considerations about the overheads associated with software multi-threading, they illuminate the possibilities that multi-core hardware affords individual applications that have been properly parallelized.

## REFERENCES/ADDITIONAL RESOURCES

The following materials, some of which are referred to in the text of this paper, provide a point of departure for further research on this topic:

[1] Grama, A. et al, an Introduction to Parallel Computing: Design and Analysis of Algorithms,Addison Wesley, 2nd edition, 2003

[2] Jordan, H. F., Jordan, H. E. Fundamentals of Parallel Computing, Prentice Hall, 2002

[3] G.M. Amdahl, Validity of the single-processor approach to achieving large scale computing capabilities. In AFIPS Conference Proceedings vol. 30 (Atlantic City, N.J., Apr. 18-20). AFIPS Press, Reston, Va., pp. 483-485.

[4] John Gustafson, Reevaluating Amdahl's Law, Communications of the ACM 31(5),reposted at http:// www.scl.ameslab.gov/ Publications/Gus/ AmdahlsLaw/Amdahls.html.

[5]. http://www.springer.com/978-1-4419-9738-8