PERFORMANCE ANALYSIS OF PARALLEL ALGORITHMS

Felician ALECU[1]
PhD, University Lecturer, Economic Informatics Department,
Academy of Economic Studies, Bucharest, Romania

E-mail: alecu.felician@ie.ase.ro

Abstract: *A grid is a collection of individual machines. The goal is to create the illusion of a powerful computer out of a large collection of connected systems sharing resources. Some resources may be used by all users of the grid while others may have specific restrictions. The most common resource is computing cycles provided by the processors. Grid computing represents unlimited opportunities in terms of business and technical aspects. The main reason of parallelization a sequential program is to run the program faster. The first criterion to be considered when evaluating the performance of a parallel program is the speedup used to express how many times a parallel program works faster than the corresponding sequential one used to solve the same problem. When running a parallel program on a real parallel system there is an overhead coming from processors load imbalance and from communication times needed for changing data between processors and for synchronization. This is the reason why the execution time of the program will be greater than the theoretical value.*

Key words: grid computing; grid network; parallel processing; performance analysis; parallel speedup; parallel efficiency

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.

The *speedup* formula is

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

where

- T_s is the execution time of the fastest sequential program that solves the problem;

- T_p 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 T_s/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 T_s. But this is not possible because T_s 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 $p \to \infty$, 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 $1/\alpha$, 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 low 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 from the system.

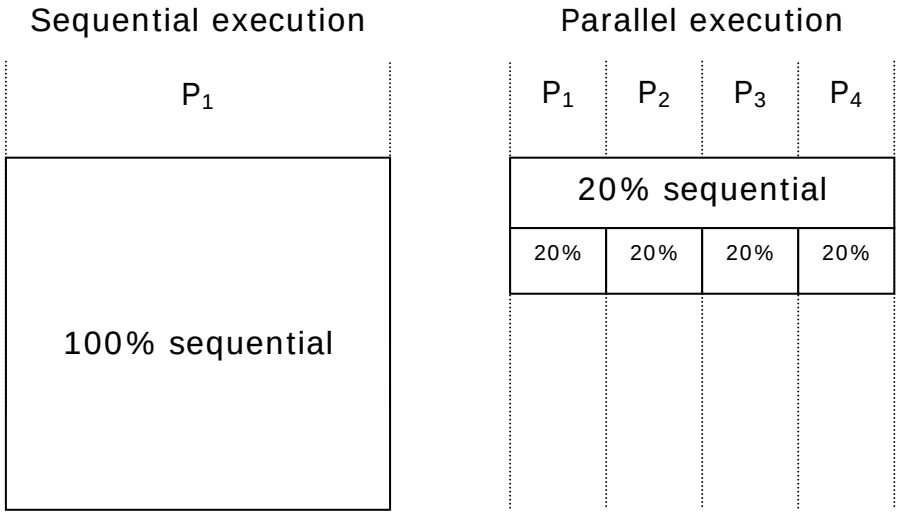

Figure 1. Parallel execution on an ideal system

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:

$$C_p = p \cdot T_p$$

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.

$$C_{supl} = C_p - C_s = p \cdot T_p - T_s$$

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.

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.

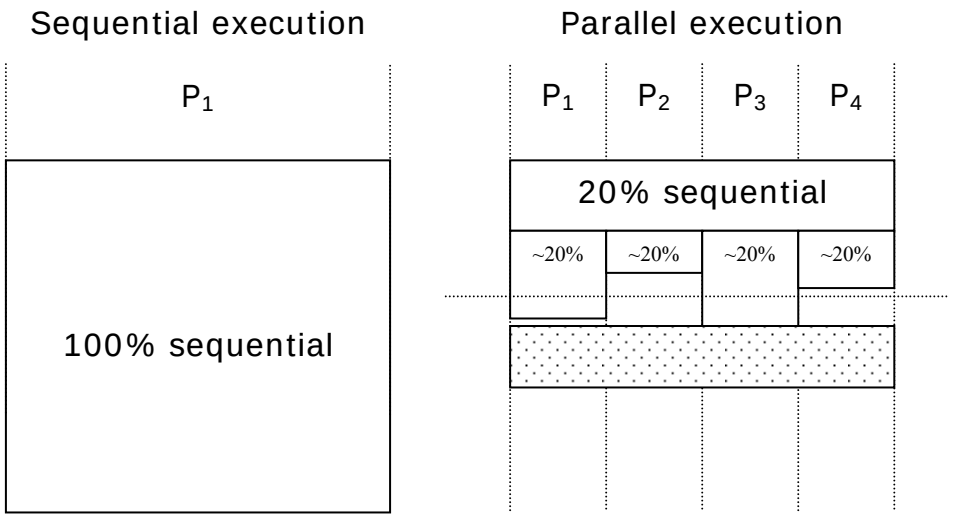

Figure 2. Parallel execution on a real system

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.

The performance of parallel algorithms executed on grid systems

A grid is a collection of machines that contribute any combination of resources as a whole. Basically, grid computing represents a new evolutionary level of distributed computing. It tries to create the illusion of a virtual single powerful computer instead of a large collection of individual systems connected together. These systems are sharing various resources like computing cycles, data storage capacity using unifying file systems over

multiple machines, communications, software and licenses, special equipments and capacities.

The use of the grid is often born from a need for increased resources of some type. Grids can be built in all sizes, ranging from just a few machines in a department to groups of machines organized in hierarchy spanning the world. The simplest grid consists of just few machines, all of the same hardware architecture and same operating system, connected on a local network. Some people would call this a cluster implementation rather than a grid. The next step is to include heterogeneous machines but within the same organization. Such a grid is also referred to as an *intragrid*. Security becomes more important as more organizations are involved. Sensitive data in one department may need to be protected from access by jobs running for other departments. Dedicated grid machines may be added to increase the service quality. Over time, a grid may grow to cross organization boundaries and may be used for common interest projects. This is known as an *intergrid*.

We will consider a parallel program that is executed in a time of T_p on a grid network composed by *p* computers numbered from *1* to *p*. Also, the sequential execution time of the program on an individual station will be T_s^i.

The *speedup* of a parallel program that runs on the cluster of stations can be computed by dividing the best sequential time by the parallel one:

$$S_{grid} = \frac{\min_{i=1}^{p} T_s^i}{T_p}$$

The individual computers of the grid network are not identical so they will have different processing powers. The ratio between the power of an ordinary computer and the most powerful one can be expressed can be expressed by the formula:

$$P_i = \frac{\min_{j=1}^{p} T_s^j}{T_s^i}, \; i = 1..p$$

Each proportion will satisfy the following relation: $P_i \leq 1$.

Based on these ratios, we can calculate the *heterogeneity factor* of the computers being part of the cluster of stations by using the differences in power that exist between them:

$$HF = \frac{\sum_{i=1}^{p} (1 - P_i)}{p}$$

During a program execution, the degree of parallelism will vary and this will generate the load imbalance of the processors from the system. Basically, the degree of parallelism is equal with the number of processors that are participating to the program execution.

The *average degree of parallelism* is defined as being the average number of stations that were active during the entire execution of the program, as the following:

$$GP_m = \frac{\sum_{i=1}^{p} T_p^i}{T_p}$$

where T_p^i represents how much time the station *i* was active.

The speedup formula can be now obtained based on the heterogeneity of the stations that are part of the grid network and using the average degree of parallelism of the program that is executed:

$$S_{grid} = GP_m \cdot (1 - GE).$$

In **conclusion** in order to obtain a faster parallel program, there is the need to reduce to the minimum the fraction that cannot be parallelized, to assure the load balance of the tasks at the processors level and also to minimize the amount of data used for communication and synchronization.

References

1. Grama, A. et al, **An Introduction to Parallel Computing: Design and Analysis of Algorithms,** Addison Wesley, 2nd edition, 2003
2. Gropp, W. et al, **The Sourcebook of Parallel Computing,** Morgan Kaufmann, 2002
3. Jordan, H. F., Jordan, H. E. **Fundamentals of Parallel Computing,** Prentice Hall, 2002
4. Joseph, J., Fellenstein, C., **Grid Computing,** Prentice Hall, 2003
5. Ladd, S., **Guide to Parallel Programming,** Springer-Verlag, 2004
6. Tanenbaum, A. S. **Distributed Operating Systems,** Prentice Hall, 1995
7. Wyrzykowski, R., **Parallel Processing And Applied Mathematics,** Springer, 2004

---

[1] Alecu Felician has graduated the Faculty of Cybernetics, Statistics and Economic Informatics in 2000 and he holds a PhD diploma in Economics from 2006.
Currently he is lecturer of Economic Informatics within the Department of Economic Informatics at Faculty of Cybernetics, Statistics and Economic Informatics from the Academy of Economic Studies.
He is the author of more than 20 journal articles in the field of parallel computers, grid computing and distributed processing.