Calculation of the Acceleration of Parallel Programs as a Function of the Number of Threads

GEORGE POPOV[1], NIKOS MASTORAKIS[2], VALERI MLADENOV[3]

[1]Faculty of Computer Systems and Control
Technical University of Sofia; Sofia 1000, "Kliment Ohridski" blvd. 8; BULGARIA
e-mail: popovg@tu-sofia.bg

[2]English Language Faculty of Engineering
Technical University of Sofia; Sofia 1000, "Kliment Ohridski" blvd. 8; BULGARIA
e-mail: mastor@wseas.org

[3]Department of Theoretical Electrical Engineering
Technical University of Sofia; Sofia 1000, "Kliment Ohridski" blvd. 8; BULGARIA
e-mail: valerim@tu-sofia.bg

Abstract: - The purpose of this study is to determine analytically what and how acceleration from paralleling execution of a task depends. It is reasonable if level of parallelism is increased, the costs of synchronization will be increased also and upon reaching a certain degree of granulation acceleration of multi-program execution starts to decrease.

Key-Words: - Multiprogramming, parallel execution, granulation, acceleration, speed up, process, thread, Amdahl's Law

1 Introduction

Splitting a task into subtasks that run in parallel (on different types of parallel systems) is a basic way to reduce the time of its implementation. However, the use of parallel algorithms has its price – there are loses of processing time for system operations that hosts the parallel execution of the tasks – parallelization and synchronization, i.e. computer system will be often in a system mode instead of performing user tasks.

In academic and scientific literature there are number of studies showing that increasing the number of processors (respectively the tasks performed on these parallel processes or threads) does not result in direct proportion to increase productivity [1,2,3,4,5,6,8]. Similar statements are applied in the theory of organization of human's productivity – if n times more people are engaged for certain operation, it won't be finished n times faster.

This dependency is shown at Fig.1.

For acceleration of parallel treatment Amdahl suggests the following equation [7,9,10], which is known as Amdahl's Law:

$$(1) \qquad Acc = \frac{1}{\left(S + \frac{(1-S)}{n}\right)},$$

where:

S is the percentage of the work that cannot be parallelized;

n is number of processors.

Free Amdahl's Law simulator is suggested for use in [11].

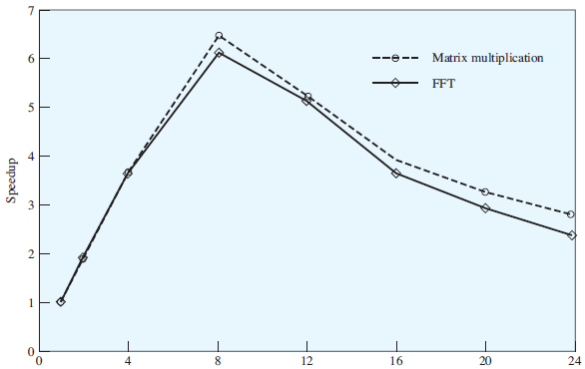


Fig.1. Application speedup as a function of number of threads (Figure is captured from [1], p 464, fig.10.4)

What other factors does the acceleration depend on? If it is possible to split a task indefinitely, where is the optimum? What is right amount of the threads? If parallel segments become very small, the cost of the system resources for their management will be significant. This problem is described in details in the scientific literature, but unfortunately there aren't any derived mathematical formulas to support the conclusions.

2 Parallel Processing Model

The following abstractions are made in this paper:

• task (program, process) can be divided into an unlimited number of parallel sections (processes, threads);

• computer system is used with an unlimited number of processing elements – cores, processors or separate computer systems arranged in a cluster architecture, only relevant system costs are recorded and the actual system architecture is not discussed;

• tasks are not blocked (they aren't waiting for resources used by other tasks);

• the level of parallelism is not specified in the current calculations – processes or threads, the conclusions made are relevant for both of them.

By increasing the coefficient of parallelism of treatment is possible to decrease the productivity as the volume of the executed system code increases.

Fig. 2 gives a Gantt's chart for the parallel implementation of the hypothetical task.

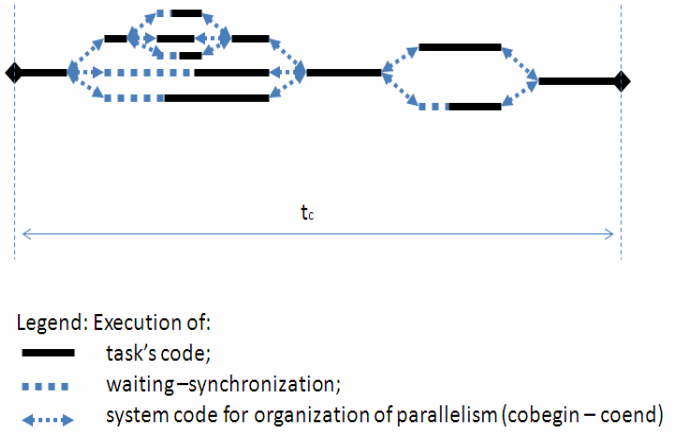


Fig.2. Typical implementation of the parallel task

Assuming (Fig. 3) that a segment of a task can be divided into a number n of parallel sections with a duration t_p, then execution time will be determined by the longest stretch system costs plus time for the organization of parallelism (primitives such as fork) and synchronization (waitings) (see equation (2)).

Fig.3. A parallel segment of the task

$$t_t = \max(t_1, t_2, ..., t_n) + t_s, \quad (2)$$

where:

$t_1, t_2, ..., t_n$ are durations for separated sections;

t_s are system costs for the organization of parallelism.

If the equation is applied for the general case, as illustrated on Fig.1 it will be written as:

$$t_t = \sum_{i=1}^{n} \max(t_1, t_2, ..., t_n)_i + t_{si} \quad (3)$$

,

where:

$\max(t_1, t_2, ..., t_n)_i$ is the maximum length of each segment;

t_{si} are system costs for organization of parallelism.

Equation (3) corresponds to Amdahl's Law, but in this case $n = 1$ implies a single section.

If the volume of the source code of the individual sections is Qi, and the systematic tools for organizing the parallelism is Qs and productivity of the system is P, equation (3) will be:

$$t_t = \sum_{i=1}^{n} \max\left(\frac{Q_1}{P}, \frac{Q_2}{P}, ..., \frac{Q_n}{P}\right)_i + \frac{Q_s}{P} i \quad (4)$$

or

$$t_t = \frac{1}{P} \sum_{i=1}^{n} \max(Q_1, Q_2, ...Q_n)_i + Q_{si} \quad (5)$$

If the entire task would be implemented in a system without multi-tasking, it would be executed for:

$$t_t = \sum_{i=1}^{n} t_i = \frac{1}{P} \sum_{i=1}^{n} Q_i = \frac{Q}{P}, \quad (6)$$

where Q is the volume of entire task code

Acceleration coefficient K_a (speed up) at the parallel execution of this task is:

(7) $$K_a = \frac{\frac{1}{P}\sum_{i=1}^{n} Q_i}{\frac{1}{P}\sum_{i=1}^{n} \max(Q_1, Q_2, ...Q_n)_i + Q_{s\,i}}$$

or

(8) $$K_a = \frac{\sum_{i=1}^{n} Q_i}{\sum_{i=1}^{n} \max(Q_1, Q_2, ...Q_n)_i + Q_{s\,i}}$$

In the ideal case, the task is divided into equal number of sections n and equation (8) can be written in the following form:

(9) $$K_a = \frac{Q}{\frac{Q}{n} + nQ_s},$$

where Q is the volume of the whole task.

After processing, the equation (9) can be presented as:

(10) $$K_a = \frac{nQ}{Q + n^2 Q_s}$$

The results of equation (10) at **Q** = 10 000, Q_s = 20 and n (1,2, .., 65) are shown on Fig.4.

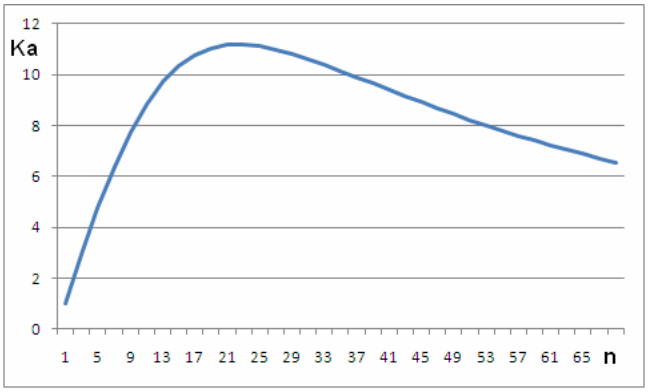


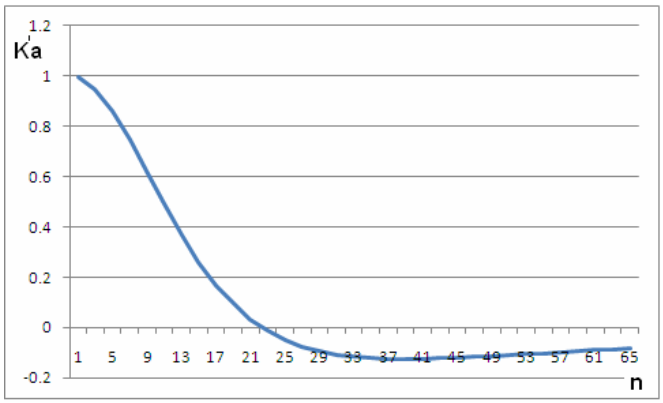
Fig.4. Graphical view of Ka = f (n)
(dependence on formula (9))

It is clear that in the case of mono program implementation the acceleration factor K_a is equal to 1. By increasing the level of parallel processing (up to n = 22) system acceleration increases to 11 and then gradually declines.

The first derivative of (10) gives n to change the effectiveness of the task's splitting:

(11) $$K'_a = \frac{Q(Q + n^2 Q_s) - 2n^2 QQ_s}{(Q + n^2 Q_s)^2} = \frac{Q^2 - n^2 QQ_s}{(Q + n^2 Q_s)^2} = Q\frac{(Q - n^2 Q_s)}{(Q + n^2 Q_s)^2}$$

The result is shown at fig.5. (at Q=10000, Q_s=20 и $n \in (1,2,..,65)$)



As can be observed for values of n> 22 the acceleration starts to decrease.

If the expression (11) is aligned to zero, the value of maximum acceleration can be determined analytically which can be obtained by splitting the task of n parts (at fixed volume of system's primitives).

(12) $$Q\frac{(Q - n^2 Q_s)}{(Q + n^2 Q_s)^2} = 0$$

(13) $$Q - n^2 Q_s = 0$$

(14) $$n = \sqrt{\frac{Q}{Q_s}}$$

If formula (14) is substituted with the values used for plotting the dependences of Fig.3 and Fig.4 (Q = 10,000, Q_s = 20), it gives:

(15) $$n = \sqrt{\frac{10000}{20_s}} = 22{,}36068,$$

which confirms the above charts.

Experiments are carried on a multicomputer parallel platform comprising 7 workstations (Intel Pentium 4, 3.2GHz, 1G RAM, Hyper-Threading) connected via Fast Ethernet switch (100 Mbps) running program

implementation of genetic algorithms. In first case there are 32 iterations in each thread, and in the second there are 128 iterations.

Fig.6. Experimental results with 7 workstations

In this case there is a maximum of acceleration at number of threads equal to 7. It can be seen the effectiveness of parallel treatment is greater at larger number of iterations.

4 Conclusion

Analytical dependence determining the maximum acceleration that can be obtained by dividing a task into subtasks is derived in this article, taking into account the size of the subtasks and the volume of code for system calls, used for realization of the parallelism.

The calculations are particularly relevant for clusters in which the relatively weak connectivity between computer systems (as opposed to systems with shared memory) leads to significant synchronization costs.

The obtained results are approximated – an assumption is made that the main task can be divided into any number of subtasks of equal length, which is practically impossible.

References:

[1] W. Stallings, *Operating Systems: Internals and Design Principals 6/e,* Pearson Education, 2009, Singapore.

[2] Silberschatz, Galvin, Gagne,Greg. *Operating Systems Concepts,* John Wiley & Sons, 2008.

[3] A.Tannenbaum, *Modern Operating Systems*, Prentice Hall, 2009

[4] Arch D. Robison, Why Too Many Threads Hurts Performance, and What to do About It? *Intel Corporation*, April 6, 2007

[5] Aviad Ezra, Concurrency Levels Tuning with Task Parallel Library (How Many Threads to Use?) October 18, 2009, http://aviadezra.blogspot.com/2009/10/how-many-threads-tpl-concurrency.html

[6] Joe Duffy, Using concurrency for scalability, *MSDN Magazine* , Issues, 2006, September

[7] Amdahl, G.M. 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., 1967, pp. 483-485.

[8] http://youandunix.com, scaling limitation factors, August 18th, 2009

[9] Jardin V., How should Amdahl's law drive the redesigns of socket system calls for an OS on a multicore CPU?, http://www.multicorepacketprocessing.com, May 21st, 2010

[10] Gustafson, J.L., Reevaluating Amdahl's Law, *CACM*, 31(4. 5), 1988. pp. 532-533.

[11] http://demonstrations.wolfram.com/AmdahlsLaw/

[12] A. Srinivasan and J. Anderson. Optimal rate-based scheduling on multiprocessors. *In Proceedings of the 34th ACM Symposium on Theory of Computing*, pages 189–198, May 2002.

[13] S. Lauzac, R. Melhem, and D. Mosse. An efficient RMS admission control and its application to multiprocessor scheduling. *In Proceedings of the 12th International Symposium on Parallel Processing*, pages 511–518, April 1998.