Evaluation of Parallel Programs by Measurement of Its Granularity

Jan Kwiatkowski

Computer Science Department, Wroclaw University of Technology
50-370 Wroclaw, Wybrzeze Wyspianskiego 27, Poland
kwiatkowski@ci-1.ci.pwr.wroc.pl

Abstract. In the past years computing has been moving from the sequential world to the parallel one, from centralised organisation to a decentralised. In parallel programming the goal of the design process cannot be reduced to optimise a single metrics like for example speed. While evaluating a parallel program a problem specific function of execution time, memory requirements, communication cost, implementation cost, and others have to be taken into consideration. The paper deals with the use of an idea of program granularity in the evaluation of parallel programs. The obtained results suggest that the presented method can be used for performance evaluation of parallel programs.

1 Introduction

In the past years computing has been moving from the sequential world to the parallel one, from a centralised organisation to a decentralised. Different computer architectures have been proposed. Although there is a large diversity of parallel computer organisations, generally two different computer organisations can be distinguished: the shared memory and the distributed memory organisations [1,2,7]. A parallel program on shared memory computer shares data by storing it in globally accessible memory. When there are no shared variables one can use the message passing programming paradigm in distributed memory computer organisation. With the development of computer networks, parallel programming using networked computers became one of the most attractive and cheap ways to increase the computing power. This is why parallel programming on a distributed system stays so popular. In the message passing programming paradigm, programmers view their programs as a collection of processes with private local variables (memory), and the ability to send and receive data between processes by passing messages.

Depending on the computer organisation used, different algorithms and different ways to support their parallelisation can to be utilised. This leads to the need of developing new programming methodologies and methods for program evaluation. Performance evaluation is one of the main problems during parallel program developing. The performance analysis can be carried out analytically or through experiments. In parallel programming the goal of the design process is not to optimise a single metrics like for example speed. A good design has to take into consideration the problem specific function of execution time, memory requirements, implementation cost, and others. The paper deals with the use of the idea of program granularity in evaluation of parallel programs.

The paper is organised as follows. Section 2 briefly describes different metrics used during performance evaluation. The idea of using granularity in performance evaluation is presented in section 3. Section 4 illustrates the experimental results obtained during evaluation of different programs using method presented. The comparison of obtained results with results of standard methods is also included. Finally, section 5 outlines the work and discusses ongoing work.

2 Performance Analysis

During performance evaluation of parallel programs different metrics are used [1,4,7,9]. The first one is the parallel run time. It is the time from the moment when computation starts to the moment when the last processor finishes its execution. The parallel run time is composed as an average of three different components: computation time, communication time and idle time. The computation time (T_{comp}) is the time spent on performing computation by all processors, communication time (T_{comm}) is the time spent on sending and receiving messages by all processors, the idle time (T_{idle}) is when processors stay idle. The parallel run time of a parallel algorithm depends not only on the size of the problem but also on the complexity of the interconnection network and the number of processors used. The next commonly used metric is speedup, which captures the relative benefit of solving a given problem using a parallel system. There exist different speedup definitions [1,10]. Generally the speedup (S) is defined as the ratio of the time needed to solve the problem on a single processor to the time required to solve the same problem on a parallel system with "p" processors. Depending on the way in which sequential time is measured we can distinguish absolute, real and relative speedups. Theoretically, speedup cannot exceed the number of processors used during program execution, however, different speedup anomalies can be observed.

Both above mentioned performance metrics do not take into account the utilisation of processors in the parallel system. While executing a parallel algorithm processors spend some time on communicating and some processors can be idle. Then the efficiency (E) of a parallel program is defined as a ratio of speedup to the number of processors. In the ideal parallel system the efficiency is equal to one but in practice efficiency is between zero and one, however because of different speedup anomalies, it can be greater than one. The next measure, which is often used in the performance evaluation of parallel programs, is the cost of solving a problem by the parallel system. The cost is usually defined as a product of the parallel run time and the number of processors. The next useful measure is the scalability of the parallel system. It is a measure of its capacity to increase speedup in proportion to the number of processors. We say that a system is scalable when the efficiency is the same for increasing the number of processors and the size of the problem [4].

Concluding the above short description of different performance metrics we can say that during experimental performance evaluation of parallel programs we need to measure the run time of sequential and parallel programs. However, there is a question: Is it possible to evaluate a parallel program using the above metrics by executing only a parallel version of the program on a parallel computer?

3 Using Granularity for Performance Analysis

A study of granularity is important if one is going to choose the most efficient architecture of parallel hardware for the algorithm at hand. In general the granularity of a parallel computer is defined as a ratio of the time required for a basic communication operation to the time required for a basic computation operation [5], and for parallel algorithms as the number of instructions that can be performed concurrently before some form of synchronisation needs to take place.

On the other hand, granularity of a parallel algorithm can be defined as the ratio of the amount of computation to the amount of communication within a parallel algorithm implementation ($\mathrm{G}=\mathrm{T}_{comp}/\mathrm{T}_{comm}$) [6]. This definition of granularity will be used in this paper.

The above definition will be used for calculating the granularity of a single process executed on the single processor as well as for the whole program using total communication and computation times of all program processes. Let us calculate parallel program granularity using the above definition. For this aim we defined the overhead function, which determines all overheads in the parallel algorithm compared with the best serial algorithms.

The overhead function is a function of problem size and the number of processors and is defined as follows [4]:

$$T_o(W, p) = p * T_p - W \tag{1}$$

where W denotes the problem size, T_p denotes time of parallel program execution and p is the number of processors. The problem size is defined as the number of basic computation operations required to solve the problem using the best serial algorithm. Let us assume that a basic computation operation takes one unit of time. Thus the problem size is equal to the time of performing the best serial algorithm on a serial computer. Based on the above assumptions after rewriting the equation (1) we obtain the following expression for parallel run time:

$$T_p = \frac{W + T_o(W, p)}{p} \tag{2}$$

Then the resulting expression for efficiency takes the form:

$$E = \frac{1}{1 + \frac{T_o(W,p)}{W}} \tag{3}$$

Recall that the parallel run time consists of computation time, communication time and idle time. If we assume that the main overhead of parallel program execution is communication time (idle time can be added to the communication time during run time measurement) then equation (3) can be rewritten as follows:

$$E = \frac{1}{1 + \frac{T_{total\_comm}}{W}} \tag{4}$$

The total communication time is equal to the sum of the communication time of all performed communication steps. Assuming that the distribution of data among processors is equal then the communication time can be calculated using equation T_{total_comm}= p * T_{comm}. Note that the above is true when the distribution of work between processors and their performance is equal. Similarly, the computation time is the sum of the time spent by all processors performing computation. Then the problem size W is equal to p * T_{comp}.

Finally, substituting the problem size and total communication time in equation (4) by using above equations we get:

$$E = \frac{1}{1 + \frac{T_{comm}}{T_{comp}}} = \frac{1}{1 + \frac{1}{G}} = \frac{G}{G+1} \tag{5}$$

It means that using granularity we can calculate the efficiency and speedup of parallel algorithms. So, it is possible to evaluate a parallel program using such metrics like efficiency and speedup by executing only a parallel version of a program on a parallel computer.

4 Experimental Results

To confirm the theoretical results some experiments were performed. During the experiments three classes of algorithms were used: algorithms with frequent communication, algorithms which do not require frequent communication and algorithms for which it is not possible to determine the frequency of communication (for example graph algorithms). Later we used our method for evaluation of different real applications. During the experiments two different hardware platforms were used: the SP2 supercomputer with 15 nodes and 10 general-purpose RISC workstations (four HP 712/60 and six SunSparc 4) connected via a local area network. Consequently two different communication networks were utilised; a dynamic interconnection network while using the SP2 supercomputer, and a static network during experiments on the computer network. As the software environment, PE (Parallel Environment) and PVM (Parallel Virtual Machine) were used respectively. Because the experiments were performed in a multi-user environment and the performance of computers was different, the execution times strongly depended on computer load. This means that the same application with the same input data may run slower or faster depending on the time of the experiment. In some instances, when workstations were overloaded it, could be many times slower in comparison with the case when all computers were underloaded. Therefore, the presented results are the averages from the series of 5 to 10 identical experiments performed under various conditions. Additionally,

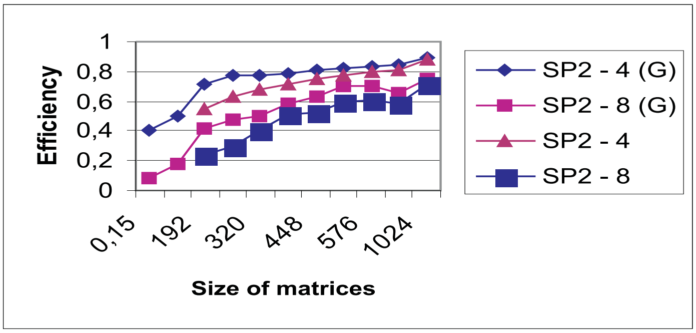

Fig. 1. Efficiency of Cannon's matrix multiplication

while experiments were performed using heterogeneous network (Sun and HP computers), results obtained were normalised, using the idea of the virtual processor. First the number of virtual processors were calculated. Next the efficiency (speedup) was calculated as the ratio of the measure *speedup* and the number of virtual processors. During these calculation the following formulas are used:

$$VP(p) = \frac{\sum_{i=1}^{p} S_i}{S_1}\ , \quad E_H = \frac{Speedup}{VP(p)}$$

where, S_i is the computational capacity of the ith processor and S_1 is the computational capacity of the processor that executed sequentially. The speedup and efficiency use during evaluation were relative - it means that sequential time (Tseq) is the execution time of parallel algorithm executing on one of the processors in a parallel computer. Taking into account the possibility that during run time measurement the processors may not be equally balanced (processor idle time can occur), the granularity was calculated using the following expression (isogranularity): $G = T_{comp} / (T_{comm} + T_{idle})$

The results of the experiments are summarised in figures below. The results obtained using granularity analysis (indicated by *G* in legend) are compared with the results obtained by using standard methods of speedup or efficiency calculation. For the first two algorithms analysed: Cannon's matrix multiplication

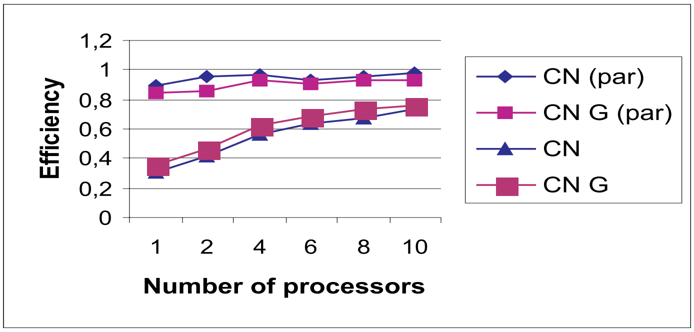

Fig. 2. Efficiency for bucket sort algorithm.

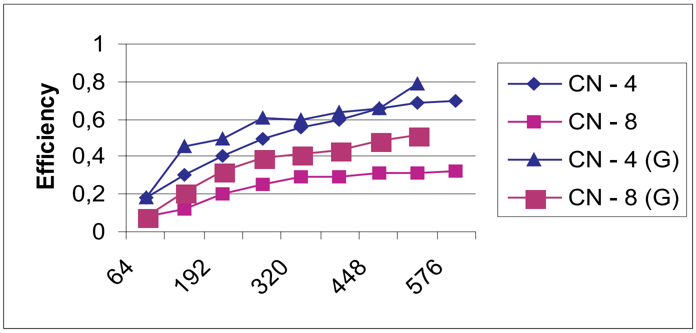

Fig. 3. Efficiency for the parallel part of Floyd algorithm.

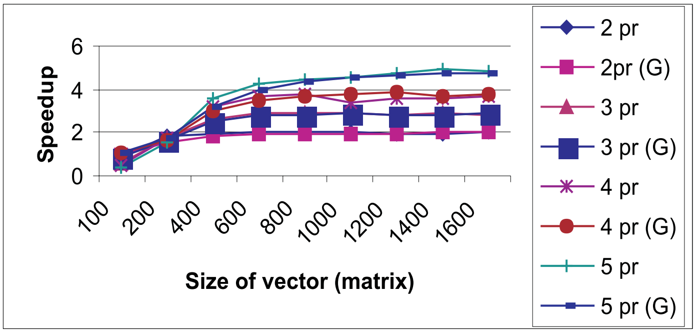

Fig. 4. Speedup for Matrix-Vector multiplication

(PE environment) and bucket sort (PVM environment) results are presented in figures 1 and 2. Comparing the obtained results, the better result was obtained for bucket sort algorithm for the parallel part of algorithm as well as for the whole algorithm, the deviation between efficiency calculated by the standard method and the new one is less then 10%. Similar results were obtained for Cannon's matrix multiplication algorithm. The shape of the diagrams is similar in both cases, however efficiency calculated using the granularity concept is higher then using the classical method. For the greater matrices the differences between both methods are greater. The reason for these differences is probably computer overload (anomalies obtain for the matrices of size 1024*1024).

Figure 3 shows results obtained for the Floyd algorithm for solving the all-pairs shortest paths using PVM environment. The results are worth because of the speedup anomalies which can be observed for graph algorithms. However, the shape of the diagrams is similar and the efficiency is greater when using granularity analysis. Figure 4 show results of experiments for matrix-vector mul-

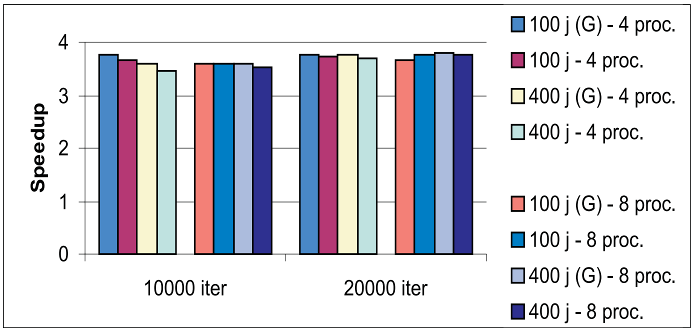

Fig. 5. Speedup for Mandebrot set

tiplication algorithm for PVM environment using from 2 to 5 processors. The results obtained are similar to the previous examples, the shape of the diagrams is similar and the speedup calculated using the granularity analysis is higher then in the classical method. Figures 5 and 6 show the results obtained for the fractal decoding algorithm. The analysed fractal was the Mandebrot set (the "whole" set – figure 5, and the part of it – figure 6). The experiments were carried out using the image of size 1000*1000 under PVM environment. The number of executed in parallel processes was 100 or 400 and depended on the way of domain partitioning of the source image (100 * 100 and 50 *50). It leads to the maximum number of performed iterations. The results obtained are similar to previouslly presented the differences between both methods are less then 10 %, and the speedup calculated using granularity analysis is higher then using the classical method.

5 Conclusions

In the paper a new way of calculating speedup and efficiency of parallel algorithms is presented. The method is based on the idea of granularity and makes it possible to calculate the efficiency and speedup of parallel algorithm by executing

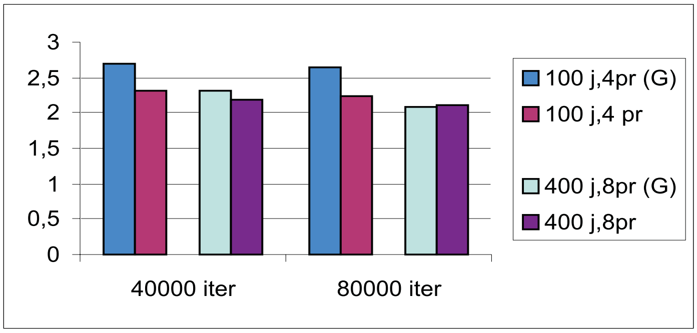

Fig. 6. Speedup for Mandebrot set (small part)

only the parallel version of a program on a parallel computer. The experimental results confirm that the presented method can be used for all investigated algorithms. However, it cannot be used for algorithms with speedup anomalies. The best results were obtained for algorithms which do not need frequent communication. However, efficiency and speedup calculated by using this method are higher than those obtained by the classical method. For all analysed algorithms and execution environments the results obtained are similar: the shape of diagrams is similar and the value of speedup and efficiency are mainly higher when using the granularity analysis. So, results obtained can be treated as an upper bound. The results obtained during the experiments at SP2 supercomputer are worse than obtained when using a computer network. Further research should investigate the possibility of using the granularity for evaluation algorithms with speedup anomalies, as well as the possibility of using granularity for scalability analysis. (The parallel system is scalable if it maintain granularity at a fixed value).

References

1. Cosnard M., Trystan D., *Parallel* Algorithms and Architectures, International Thomson Publishing Company, London 1995.
2. Foster I., Designing and Building Parallel Programs, Addison-Wesley Pub., 1995 (also available at http://www.mcs.anl.gov/dbpp/text/book.html).
3. Gustafson J.L., Reevaluating Amdahl's Law, Communication of the ACM, May 1988, pp. 532-533
4. Grama A.Y., Gupta A., Kumar V., Isoefficiency: Measuring the Scalability of Parallel Algorithms and Architectures, IEEE Parallel & Distributed Technology, August 1993, pp. 12-21.
5. Huzar Z., Kwiatkowski J., Magott J., Dataflow Processing Modeling in Performance Extension of LOTOS, *Proceedings of the IASTED International Conference Parallel and Distributed Processing Systems - Euro-PDS'97*, Barcelona, Spain - 1997, pp. 335-339.

6. Konieczny D., Kwiatkowski J., Skrzypczynski G., Parallel Search Algorithms for the Distributed environments, *Proceedings of the 16th IASTED International Conference APPLIED INFORMATICS*, Garmisch-Partenkirchen, Germany - 1998, pp. 324-327.
7. Kwiatkowski J., Performance Evaluation if Parallel Programs*, Proceedings of the International Conference Parallel Processing and Applied Mathematics PPAM'99*, Kazimierz Dolny, Poland 1999, pp. 75-85
8. Kumar V., Grama A., Gupta A., Karypis G., Introduction to Parallel Computing, The Benjamin/Cummings Pub. Comp., Inc., 1995
9. Lewis T, Revini H., Introduction to Parallel Computing, Prentice-Hall, 1992
10. Peterson D., Chamberlain D., Beyond Execution Time: Expanding the Use of Performance Models, IEEE Parallel & Distr. Technology, summer 1994, pp. 37-49
11. Sahni S., Thanvantri V., Performance Metrics: Keeping the Focus on Runtime, IEEE Parallel & Distributed Technology, spring 1996, pp. 43-56