STUDIA UNIV BABEŞ-BOLYAI, MATHEMATICA XXXVIII 3, 1993

ON SOME MODELS OF PARALLEL PERFORMANCE

Daniela VĂSARU[*]

Dedicated to Professor Emil Muntean on his 60th anniversary

Received November 29, 1992
AMS Classification 65Y05, 68Q05, 68M20, 68Q22

REZUMAT Articolul "**Asupra unor modele de performanţă paralelă**" prezintă câteva dintre cele mai folosite modele de caracterizare a performanţei algoritmilor paraleli Acestea au ca trăsătură comună folosirea fracţiilor seriale şi paralele în studiul performanţei Paralel cu prezentarea lor sunt discutate atât calităţile şi defectele lor cât şi relaţiile existente între ele În finalul articolului este dat un exemplu de folosire al acestor modele în caracterizarea performanţei paralele

1. Introduction. New requirements in engineering and computational science had lead to a strong interest in constructing a "teraflop" computer Parallel processing is considered to be the great hope in obtaining such a performance Ideally, on $\mathbf{n}_p$ processors a program will run $\mathbf{n}_p$ times faster than on a single one Unfortunately this is rarely the case One reason is the great disproportion existing between the progress in hardware technology and the methods of programming the parallel computers In what concerns the software part, there are a lot of problems waiting to be solved Two of them are the inexistence of a common complexity model for parallelism and the difficulties encountered in analyzing the performance and corectness of parallel algorithms

This paper presents a number of models of parallel performance, models that have in common the use of serial and parallel fractions in characterizing the parallel algorithms,

[*] *"Babeş-Bolyai" University, Faculty of Mathematics and Computer Science, 3400 Cluj-Napoca, Romania*

showing the relations between them and how can we use them in predicting the parallel performance

2. Preliminaries. The most common used measures of parallel performance are **speedup** and **efficiency** [2,3] They are both functions of problem size **n** and number of processors $\mathbf{n_p}$ and formally can be described by

$$S(n,n_p) = \frac{T(n,1)}{T(n,n_p)} \quad , \quad E(n,n_p) = \frac{S}{n_p} \tag{1}$$

T(n,i) is the time spent to solve a problem of size **n** by **i** processors Because of the overhead introduced by parallelization, **T(n,i)** is considered relative to the best serial implementation

The influence that the two parameters **n** and $\mathbf{n_p}$ have on the speedup and efficiency is of great practical importance By varying one or both parameters, different models of parallel performance are coming out

In order to make more readable the article, we will not mention always the parameters of a function For example, we will write S instead of $\mathbf{S(n,n_p)}$ It should be clear from the context on which parameters a function depends In general, all the functions have two parameters In the case that one of them is fixed we will not mention it

3. Amdahl's Law Consider an algorithm solving a problem of given size **n** that has one part intrinsically sequential and the other part, 100% parallelizable, can be distributed equally among the available processors Now, if **s** is the fraction of time spent by a uniprocessor on the serial part of the algorithm (serial fraction) and **p** is the fraction of time spent on the parallelizable part by the uniprocessor, then the time spent by n_p processors on the same problem will be **$(s+p/n_p)*T(1)$** So, the speedup will be given by

$$S = \frac{(s+p)*T(1)}{(s+p/n_p)*T(1)} = \frac{1}{s+(1-s)/n_p} \tag{2}$$

This is a steep function of s near s = 0 For a fixed number of processors the speedup is increasing unbounded with the decreasing of s This case can be used in selecting the most efficient parallel algorithm (in the sense of efficient use of processors) among different algorithms solving the same problem the one with the minimum s is the best

What's happening if we have a single algorithm for a fixed-problem size and an increasing number of processors? Then the speedup is assimptotically bounded by **1/s**

$$S \rightarrow 1/s \quad as \quad n_p \rightarrow \infty \tag{3}$$

This is the performance forecast by Amdahl's Law if a computer has two speeds or modes of operation during a given calculation, the slow mode will limit overall performance even if the fast mode is infinitely fast [1,4] It means that if an algorithm has 2% sequential part, speedup greater than 50 one can not obtain even if it has thousands of processors

This result was used by Amdahl as an argument against building massively parallel systems

The limitation of speed given by (3), as we will see in the next sections, is valid only for the case under consideration, i e for fixed-size problems That's the reason why the model discussed is also called the **fixed-size model**

4. Moler's Law

Moler was one of the firsts to show that Amdahl's limit can be beated [1] He had proved that parallelism can attain desired speedup for sufficiently large computations

Instead of considering a fixed size problem and an increasing number of processors, he had study the case of a fixed number of processors and instances of the same problem but with different sizes He had shown that the serial fraction **s** is dependent on the input size **s = s(n)** So, s isn't constant (the main assumption in the fixed-size model) Even if S is bounded by **1/s**, this limit isn't fixed He define an effective parallel algorithm as one for which **s(n) → 0** when **n → ∞** In this case, for a fixed number of processors n_p, one would obtain

$$S = \frac{1}{s(n)+(1-s(n))/n_p} \rightarrow n_p \text{ , } for\ n \rightarrow \infty \tag{4}$$

It follows that for problems large enough, it can be obtained the desired speedup (the processors are used efficiently) In practice, **n** cannot grow to infinity but it can be made as big as the available memory allows

5. Sandia's Model The researchers from Sandia Laboratories had studied the variation of speedup starting from the following observation if one has more computing power, he usually don't use it to solve the same problem of fixed size but larger instances of the problem [1,6] The reason is obvious there is no point in using more processors than the concurrency of a problem because then, some of them will remain idle Also, by increasing the number of processors the overhead due to communication is increasing and if the problem size is fixed, than the computational time will remain fixed, while the communication time will grow, affecting the overall performance

By scaling the problem size proportionally with the number of processors, the serial fraction **s** can be made as little as we want The serial component of an algorithm is determined by the startup time, serial bottlenecks and I/O, which are not dependent on the problem size The parallelizable part of an algorithm varies with the input size It follows that **s** can be made to shrink under these circumstances

Adding more processors brings more memory and more speed How do we scale the problem size with memory or with speed? Most scientists scale the problem in order to occupy all the available memory This is called the **scaled model** and it is the one proposed by Sandia They assumed as a first approximation that the parallel part grows proportionally with the number of processors

The model proposed by Sandia as an alternative to the fixed-size model is, in fact, the inverse of the Amdahl's paradigm Instead of asking how fast a given serial program will run on $\mathbf{n_p}$ processors, it's asked how long it will take to run a given parallel program on an uniprocessor

If **s'** is the fraction of time spent by a multiprocessor machine with $\mathbf{n_p}$ processors on serial parts of a parallel program and **p'** the fraction of time spent by the same multiprocessor on the parallel part, the time to run the program on an uniprocessor will be $\mathbf{(s'+p'*n_p)*T(n_p)}$. Then , the scaled speedup will have the form

$$S_w = \frac{(s'+p'*n_p)*T(n_p)}{(s'+p')*T(n_p)} = s' + (1-s')*n_p \tag{5}$$

It is easy to see that the scaled speedup is a function of moderate slope $\mathbf{1-n_p}$ of s' (a line) and it grows with increasing $\mathbf{n_p}$

Another alternative is to scale the problem size in order to maintain constant execution time This is called the **fixed-time model** An example for the use of this model is the weather prediction It doesn't make sense to have an execution time greater than 24 hours in predicting the time for the next day

To illustrate the difference between these models (in fact, the fixed-time model is intermediary between the fixed-size model and the scaled model), we will present an example For the multiplication of two matrix (with dimensions **nxn**), the memory needed is $\mathbf{O(n^2)}$ but the number of operations is $\mathbf{O(n^3)}$ For the scaled model (problem size scales with memory) $\mathbf{n^2}$ grows proportionally with $\mathbf{n_P}$ but for the fixed-time model, $\mathbf{n^3}$ grows proportionally with $\mathbf{n_P}$ (i e $\mathbf{n^2}$ grows as $\mathbf{n_P^{2/3}}$)

6. General Model of Parallel Performance Carmona and Rice proposed a general model of parallel performance which capture the previous presented models [2]

They use the same criteria of characterizing the parallel algorithms, speedup and efficiency, but with some slights modifications of (1) Instead of considering running time as a measure of the complexity of algorithms, time beeing dependent on the architecture, they use as a measure of work the computational counts or unit counts based on the size of an indivisible task

If **wa** is the **work accomplished** by a parallel program and **we** the **work expended** by the same program, the efficiency can be expressed by $\mathbf{E = wa/we}$

The work accomplished is given by the number of operations done by the best serial implementation and it's not depending on the number of processors, only on the problem size In general, **wa** < **we** because the parallelization introduces some overhead, redundant operations, communication requirements not needed in the serial case

The difference **ww = we-wa** is called the **wasted work** It covers the time needed for the following activities waiting for other tasks to complete work, communication delays and/or memory contention (dependent on the particular architecture and the implementation of the algorithm), operation redundancies introduced by the implementation, including task activation/ termination and synchronization code **Ww** is a function of both problem size and number of processors

Under these considerations, the expressions for efficiency and speedup will be

$$E(n,n_p) = \frac{wa(n)}{we(n,n_p)} = \frac{wa(n)}{wa(n)+ww(n,n_p)} \qquad (6)$$

$$S(n,n_p) = E * n_p = \frac{wa(n)}{wa(n)+ww(n,n_p)} * n_p \qquad (7)$$

Using these work parameters, Rice and Carmona give also new interpretations for the serial fraction **s** and the scaled serial fraction **s'** From (2) and (7) it follows

$$s = \frac{ww}{wa} * \frac{1}{n_p-1} \quad , (n_p>1) \qquad (8)$$

So, **s** can be interpreted as the distribution across the additional processors of the ratio of work wasted to work accomplished Similarly, from (5) and (7)

$$s' = \frac{ww}{we} * \frac{n_p}{n_p-1} \quad , (n_p>1) \qquad (9)$$

Therefore, **s'** can be interpreted as a collective wasted effort **n_p*s1**, where **s1** is the distribution across the additional processors of the ratio of work wasted to work expended

From eqs (8) and (9) it follows that **s,s',p,p'** are functions both of problem size and the number of processors This modifies the previous points of view, i e **s** was considered constant for fixed-size problems as the number of processors increases, **s'** was considered only for scaled problems, with **$n=n(n_p)$** a increasing function of **n_p** These differences appear from the fact that the new definitions of **s** and **s'** incorporate wasted work

It is not difficult to see that the fixed size-model is a particular case of these new definitions if the wasted work has the form **ww = (n_p- 1) * w(n)**, where **w(n)** is a function only of **n**, then **s** will be constant for fixed-size problems Intuitively, **ww** has this form if each one of the new **n_p-1** processors contributes in equal part to the wasted work (with **w(n)**) and these contributions don't depend on the number of processors In a similar way we can show that the other described models are particular cases of this general one

Using eqs (6),(7),(8) and (9),it results the following law·

$$S = \begin{cases} s/s' & , ww>0 \\ n_p & , ww=0 \end{cases}$$
$$E = \begin{cases} p'/p & , ww>0 \\ 1 & , ww=0 \end{cases} \qquad (10)$$

This law relates s and **s'** for different combinations of **n** and **n_p**, while the previous models showed the trend in speedup when s and **s'** are varied for a given number of processors, or are held fixed and **n_p** is varied The law (10) also gives an interesting relation between the fixed-size and the scaled model, showing how can one predict the other From (2),(5) and (10) it's easy to derive

$$s' = \frac{s}{s+(1-s)/n_p} \qquad (11)$$
$$s = \frac{s'}{s'+(1-s')*n_p} \qquad (12)$$

These relations can be used in two ways for a given speedup, one can determine from the base scalar fraction the scaled serial fraction (or viceversa), secondly (and more important), from the serial fraction of a base problem **s** one can derive the scaled serial fraction s' (and, therefore, the scaled speedup) for a larger problem, by simply making s'=s in (5)

The general model proposed by Carmona and Rice is described by a group of assertions, assertions stating how the parameters influence each other on the curves of the form $n = n(n_p)$ These curves represent all possible relations between the problem size and the number of processors Given a function $f(n,n_p)$, the notation $f\uparrow$ (respectively $f\downarrow$) denotes that f increases (decreases) on some fixed curve $n = n(n_p)$ as n_p increases Also, $f\uparrow r$ (respectively $f\downarrow r$) denotes that f approaches the limit r on the curve as $n_p \rightarrow \infty$

The performance model is given by the following assertions

A1 $s'\downarrow \Rightarrow s\downarrow \Rightarrow S\uparrow$ (for any curve $n = n(n_p)$)

A2 $s\uparrow \Rightarrow s'\uparrow \Rightarrow E\downarrow$ (for any curve $n = n(n_p)$)

A3 Assume that $n = n(n_p)$ defines a constant s-curve Then $s'= \Theta(1)$ and $s'\uparrow 1$ Furthermore, $S\uparrow 1/c$ and $E\downarrow 0$, where $s(n(n_p),n_p)=c$ (constant s-curve)

A4 Assume that $n = n(n_p)$ defines a constant s'-curve Then $s = \Theta(1/n_p)$ and $s\downarrow 0$ Furthermore, $S = \Theta(n_p)$, $S\uparrow$ and $E\downarrow(1-c)$, where $s'(n(n_p),n_p) = c$ (constant s'-curve)

This general model provides a framework in which the various performance parameters can be compared and contrasted within a single unified view of speedup It is easy to see that assumption A3 is a generalization of the fixed-size model (Amdahl) and the assumption A4 of the scaled model (Sandia)

Now, one question easily arises why these differences between the general model and the previous ones with respect to the number of parameters on which s and s' depend? One reason it was given above The new definitions incorporate wasted work This is due to the fact that in all the other models the speedup was interpreted as the gain in time of a parallel implementation with respect to the serial implementation of the same algorithm, and not over the implementation of the best serial algorithm that solve the problem, as it is the case in the general model (best serial implementation)

7. **Example** To illustrate the use of these models in predicting the performance of the parallel algorithms, we will give an example The problem to be solved is the evaluation of a polynomial expression at a given point x

$$f(x) = \sum_{i=0}^{n} c_i x^i$$

It is well known that the standard serial algorithm takes 3n-1 unit counts (n additions and 2n-1 multiplications, considering that an addition and a multiplication take each a unit count) The best serial algorithm is the Horner scheme and it takes 2n unit counts (n additions and n multiplications)

A parallel algorithm for solving this problem using p processors, p ≤ n/2, is the following (see [5,7]) each processor i evaluates, using the Horner scheme, the following polynomial

$$g_i(x) = \sum_{j=0}^{[(n-i)/p]} c_{pj+i} x^{pj} \qquad i = 0, \ p-1$$

The value of the initial polynomial can be computed from the following expression

$$f(x) = \sum_{i=0}^{p-1} g_i(x) * x^i$$

This parallel algorithm takes (2n/p + 2*log p) unit counts (where the base of the logarithm is 2) For more details on the analysis of the complexity see [5]

In order to study the performance of the algorithm, we have to determine the serial fractions From above and from the general model of performance, we have

wa = 2n,

we = p(2n/p + 2*log p) = 2(n + p*log p) ,

$ww = 2p*\log p$

$s = (p*\log p)/(n*(p-1))$

$s' = (p^2*\log p)/((n + p*\log p)*(p-1))$

$S = n/(n/p + \log p)$

$E = n/(n + p*\log p)$

We can see that the parallel algorithm is efficient in the sense of Molei for a fixed number of processors, $s(n) \to 0$ when $n \to \infty$ and $S \to p$ It depends on our interests and on the available memory how much we will increase the dimension of the problem

From the restriction $p \le n/2$ it comes that we cannot increase to infinity the number of processors without increasing the dimension of the problem, if we want to make an efficient use of the processors

For a fixed problem size n, the speedup is an increasing function of p, when $1 < p \le n/2$ (it can be seen by studying the sign of the derivative) It follows that the optimal number of processors (in order to obtain a maximum speedup) is $p = n/2$ and the maximum obtainable speedup for fixed n is $S_{max} = n/(1+\log n)$ and the efficiency will be $E = 2/(1 + \log n)$ This efficiency is not very good, especially for big problems

If we want to find the optimal number of processors in order to obtain a maximum efficiency for a given problem size, we have to study the expression of E It is a decreasing function of p and so, if we want an optimal efficiency, it will be obtained for p=2 I this case, $E_{max}=n/(n+2)$ and $S = 2n/(n+2)$

We can see that maximizing the efficiency is not the same thing as maximizing the speedup Sometimes is better to find a way in between these two extremes

If neither the dimension of the problem, nor the number of processors is fixed, we can predict the performance of the parallel algorithm for various relations between these two parameters For example, if $n = c*p$ (with constant $c \geq 2$, from the restriction on the number of processors), we obtain

$S = n/(c + \log (n/c))$ and $E = c/(c + \log(n/c))$

It comes that the speedup is increasing with the dimension of the problem and the number of processors, but the efficiency is decreasing

There are many interesting conclusions that can be find out from the expressions above We will conclude with one of them

If we are interested in maintaining a fixed efficiency E, how do the parameters n and p need to be corelated? From the expression of the efficiency it comes out that

$n = (E*p*\log p)/(1-E)$

It means that we have to grow the dimension of the problem proportionally with p*logp (this is the isoefficiency function for the parallel algorithm, as defined in [4]) in order to maintain an efficient use of the processors

8. Final Remarks In conclusion, we will give a summary of the most important applications of these models:

- determining the best parallel algorithm for solving a fixed size problem on a given architecture (the one with the least scalar fraction),
- as the scalar fraction of an algorithm depends on the architecture used, we can determine the most appropriated architecture on which the parallel algorithm should be implemented or

viceversa (finding the minimum s),

- for a fixed size problem we can determine the optimal number of processors to be used in order to maximize the speedup or the efficiency,

- we can find out what relation has to exist between the dimension of the problem and the number of processors in order to maintain a fixed efficiency (called the isoefficiency function)

There are also other models for predicting the parallel performance, for a general view see [4] There isn't a best model, it depends on our interests which one should we use, each is appropriate for a different situation That is the reason why we had choose to present the models that have in common the use of serial fractions in this case, the general model of performance of Rice and Carmona is the best, as it is a generalization of all the others

REFERENCES

1 G F Carey (ed), Parallel Supercomputing Methods, Algorithms and Applications, WILEY Series in Parallel Computing 1989

2 E A Carmona, M D Rice, Modeling the Serial and Parallel Fractions of a Parallel Algorithm, Journal of Parallel and Distributed Computing, vol 13, no 3, p 286-298, 1991

3 Gh Coman, D L Johnson, Complexitatea algoritmilor, Universitatea "Babes-Bolyai", Facultatea de Matematica si Fizica, curs litografiat, 1987

4 V Kumar, A Gupta, Analyzing Scalability of Parallel Algorithms and Architectures, AHPCRC Preprint 92-020

5 D Văsaru, Calcul Paralel, Lucrare de Diploma, 1991, Univ "Babes-Bolyai", Fac de Matem , Cluj

6 H M Wacker, The Use of Amdahl's Scaled Law in Determining the Power of Vector Computers, 1989 CERN School of Computing, p 156-171

7 S Wilson, Numerical Recipes for Supercomputers, in Supercomputational Science, R G Evans,S Wilson (ed), Plenum Press, New-York and London, 1990, pp 81-109