A Methodology for User-Oriented Scalability Analysis[1]

Dolors Royo, Miguel Valero-García, Antonio González and Carme Marí
Departament d'Arquitectura de Computadors
Universitat Politècnica de Catalunya
c/ Jordi Girona 1-3, Campus Nord - Edifici D6
E-08034 Barcelona (Spain)
Phone: +34-3-401 7189
E-mail: {dolors, miguel, antonio}@ac.upc.es

Abstract

Scalability analysis provides information about the effectiveness of increasing the number of resources of a parallel system. Several methods have been proposed which use different approaches to provide this information. This paper presents a family of analysis methods oriented to the user. The methods in this family should assist the user in estimating the benefits when increasing the system size. The key issue in the proposal is the appropriate combination of a scaling model, which reflects the way the users utilize an increasing number of resources, and a figure of merit that the user wants to improve with the larger system. Another important element in the proposal is the approach to characterize the scalability, which enables quick visual analyses and comparisons. Finally, three concrete examples of methods belonging to the proposed family are introduced in this paper.

1 Introduction

In this paper we focus on the problem of analysing the scalability of a parallel system. The term parallel system is used to refer to a particular parallel architecture running a particular parallel algorithm.

The scalability of a parallel system is a measure of its capability to effectively utilize an increasing number of resources [2]. Since parallelism is a key issue in today's supercomputing trends, it is not surprising that scalability analysis has motivated a large amount of research work.

Scalability analysis is important for system architects since it may reveal system bottlenecks and may help in the improvement of the parallel architecture and/or the parallel algorithm. Scalability analysis can also show to the user the benefits of increasing the system size. In this paper we focus on scalability analysis oriented to the user' point of view.

Many authors have proposed methods to evaluate the scalability of parallel systems. See [5] for a review of proposals. In general, these methods can be characterized by three elements:

(a) An scaling model, that describes how the user utilizes an increasing number of nodes.
(b) A figure of merit, that is the performance measure that the user wants to improve when increasing the system size.

1. This work has been supported by the Ministry of Education and Science of Spain (CICYT TIC-95/0429) and the European Center for Parallelism in Barcelona (CEPBA).

(c) The scalability characterization, that is the approach to relate the figure of merit and the increase in the number of nodes.

In the following, two concrete examples are used to illustrate the above elements.

In the isoefficiency method proposed in [2] it is assumed that, when increasing the system size, the problem size is also increased so that the parallel efficiency remains constant. This is, therefore, the scaling model associated to the isoefficiency method. The figure of merit is the problem size. The scalability is characterized by the function $f_E(p)$ that determines the problem size required to maintain a certain efficiency E when using p nodes in the parallel system.

A second example is the CMP method proposed in [1], which uses the memory-constrained scaling model. Under this scaling model, an increasing number of nodes is used to increase the problem size as much as possible, with the only limitation of the system memory. The figure of merit in the CMP method is the scaled speed-up. The scalability is characterized by the function that describes the asymptotic growth of the scaled speed-up, when the number of nodes goes to infinity.

Scalability analysis methods such as isoefficiency and CMP have been used to analyse concrete systems and some interesting results have been obtained. However, we claim that, although these results can be useful for system architects, they may have limited interest for system users. The reasons for this claim is that the components which characterize the scalability analysis method (scaling model and figure of merit), do not reflect the user interest when using the parallel system. Again, isoefficiency and CMP methods provide concrete examples to clarify this claim as shown below.

It could be argued that increasing the problem size to maintain a constant efficiency is not a frequent way to use an increasing number of processors. More frequently, the user wants more processors to increase as much as possible the problem size to obtain a more precise result, or to reduce as much as possible the execution time of a fixed-size problem.

On the other hand, it could be also argued that scaled speed-up can be a figure of interest for system architects but probably not for system users. More frequently, system users are interested in improving figures of merit such as execution time or problem size.

To conclude, isoefficiency and CMP analysis may be of little use for system users, the former because it does not assume a reasonable scaling model and the later because it does not use a figure of merit of interest for users.

In this paper, a methodology for analysing the scalability of parallel systems is proposed, which is aimed at helping the system user. This is the reason for using the term user-oriented scalability analysis.

In terms of the three elements which characterizes an scalability analysis method, the main contribution of the proposed methodology are the following:

(a) Only figures of merit relevant to the user and "real" scaling models are used. In particular, we use figures of merit such as execution time or problem size, (but not scaled speed-up, as in CMP), and scaling models such as fixed-problem size, time-constrained or memory-constrained [4] (but not "fixed-efficiency" as in isoefficiency analysis). Moreover, only useful combinations of scaling model and figure of merit are used. As an example, analysing the execution time under the memory-constrained model is not useful. However, analysing the problem size under the same scaling model could help the user in determining if the cost of increasing the system size would be amortized by the increment in the problem size.

(b) A new approach to characterize the scalability of the system is used, which enables a quick visual analysis and comparisons.

By following the methodology proposed in this paper many user-oriented analysis methods can be proposed. In this paper, three concrete methods will be proposed, each of them targeted to a different type of user.

It should be expected that the conclusions of user-oriented methods are different, and more relevant to the users than the conclusions of other methods. To illustrate this fact, section 2 presents and example in which the scalability of a concrete parallel system (FFT on a hypercube) is analysed using the isoefficiency method and a user-oriented method which combines the time-constrained scaling model and the problem size figure of merit. It is shown that the conclusions of both analysis methods are significantly different. After this motivating example, in section 3 the proposed methodology is formally described. and three different user-oriented methods are introduced, which are targeted to different ways of using an increase in the number of nodes. Section 4 shows the application of two of the proposed methods to a concrete parallel system. The emphasis is not in the conclusions about the scalability of the system but in showing the potential of the proposed methods for quick a analysis and comparison among systems. Finally, in section 5 the main conclusions of this paper are drawn.

2 Motivating example

A scalability analysis of several parallel algorithms to compute the FFT on a hypercube and a mesh, using the isoefficiency method was presented in [3]. In this section we briefly review the conclusions of this analysis for the particular case of the binary-exchange algorithm on a hypercube. We present then an alternative scalability analysis for the same parallel system, based on the approach introduced in this paper. The conclusions of this alternative analysis are significantly different from those of the isoefficiency method.

2.1 The binary-exchange algorithm for computing the FFT on a hypercube

Assume we have to compute the FFT of a n-point complex sequence ($n = 2^r$) on a p-node hypercube ($p = 2^d$). The well known binary-exchange algorithm is a parallel implementation of the radix-two Cooley-Tukey algorithm to solve the problem. In the following the binary-exchange algorithm is outlined.

The i-th element of the input sequence is initially stored in node i *div* p. The binary-exchange algorithm consists of r iterations. In each of the first d iterations every node communicates with one of its neighbours in the hypercube and then performs some computation. Specifically, the communication step consists of the exchange of 2^{r-d-1} complex elements and the computation consists of 2^{r-d-1} butterflies. Every butterfly takes two complex elements as input operands and produces two complex elements of a new sequence. In each of the r-d last iterations every node performs 2^{r-d-1} butterflies without requiring any communication. Several variations of this algorithm have been identified in the literature. The difference among them are not relevant to the purpose of this paper.

In the following, the term elementary computation will be used to refer to the computation of each of the two complex numbers produced by a butterfly. The time required by an elementary computation will be denoted by t_c (i.e. the time required by a butterfly is $2*t_c$). In addition, the time required to send a message containing N real numbers to a neighbour node in the hypercube is assumed to be $t_s + N*t_w$ (t_s is the communication start-up and t_w is the transmission time per real number).

Putting all this together, the time required by the binary-exchange algorithm in a hypercube is:

$$T_p = r(2^{r-d}t_c) + d(t_s + 2^{r-d}t_w)$$

In the following it is assumed that every node has a local memory of size R real numbers ($R=2^s$). Therefore, it will be considered that the largest FFT which can be solved in a system with 2^d nodes has 2^{d+s-1} complex points. If the system does not have virtual memory, larger FFT would not fit in the system memory. If the system has virtual memory, larger FFT would incur a significant page fault overhead, which is not modelled in the above expression.

Finally, it is important to note that if the input sequence has n complex points then the largest system which can be used to compute the FFT using the binary-exchange algorithm is $n/2$.

2.2 Isoefficiency analysis

In this section we review briefly the conclusions of the isoefficiency analysis proposed in [3] for the parallel system which executes the binary-exchange algorithm in a hypercube. In the isoefficiency method, the scalability of a parallel system is characterized by the function which determines the problem size required to maintain a certain efficiency, when increasing the number of nodes of the system. The problem size, denoted by W, is defined in this paper as the number of elementary computations required to solve the problem using the best-known sequential algorithm. This definition is slightly different from that given in [3], where W is not defined as the amount of elementary computations but as the time required to perform them. This difference should not make the comparison of results difficult.

In the following, t^i_e denotes the time spent by node i in useful computation and t^i_o as the time spent by node i in communication, idle periods and in general, any activity which would not exist if the problem were solved in a single node using the best-known sequential algorithm. Therefore, we have that: $W \cdot t_c = \sum_i t^i_e$

In the case of the FFT computation, we take the radix-two Cooley-Tukey algorithm as the best-known sequential algorithm. Therefore, we have $W = nr$.

If T_p is defined as the time to solve the problem using a p-node parallel system, the parallel efficiency $E(p)$ is defined as:

$$E(p) = \frac{W \cdot t_c}{T_p \cdot p}$$

The parallel overhead, denoted by T_o, is defined as $T_o = \sum_i t^i_o$. Since $T_p = t^i_e + t^i_o$ for any i, we have that $W \cdot t_c + T_o = p \cdot T_p$, and therefore, we can write:

$$E(p) = \frac{1}{1 + T_o / W \cdot t_c}$$

It is obvious that if we wish to maintain a certain parallel efficiency E when increasing p, the ratio $T_o/W \cdot t_c$ should remain constant. Since, in general T_o increases with p, the ratio $T_o/W \cdot t_c$ will remain constant only if the problem size W is increased. In particular, the following equation must hold:

$$W = K\frac{T_o}{t_c}$$

where $K=E/(1-E)$.

The degree of scalability is characterized by the isoefficiency function $f_E(p)$ which determines the value of W required to maintain a certain efficiency E for a given number of nodes p. Applying the isoefficiency analysis to the binary exchange algorithm in the hypercube we have the following:

$$t^i_o = d(t_s + 2^{r-d} t_w)$$

$$T_o = \sum_i t^i_o = pd(t_s + 2^{r-d} t_w)$$

$$W = nr$$

If $W=KT_o/t_c$ then we have:

$$nr = \frac{Kpd(t_s + 2^{r-d} t_w)}{t_c} \quad (1)$$

The requirement on W due to the first term in (1) is given by:

$$W = nr = K\frac{t_s}{t_c}pd = O(p\log p) \quad (2)$$

The requirement on W due to the second term in (1) is given by:

$$r = K\frac{t_w}{t_c}d \Rightarrow$$

$$W = nr = p^{Kt_w/t_c}K\frac{t_w}{t_c}d = O\left(p^{Kt_w/t_c}\log p\right) \quad (3)$$

From expressions (2) and (3) it can be concluded that the values t_w and t_c determine to a great extent the degree of scalability of the system. In particular, when $Kt_w/t_c < 1$, the growth in the problem size is dominated by (2). Therefore, taking into account that $K=E/(1-E)$, maintaining an efficiency lower that $E=t_c/(t_c+t_w)$ requires a $O(p\log p)$ growth in W, which can be considered as a moderate growth. It could be said that in this case the system is highly scalable. However, when $Kt_w/t_c > 1$, the growth in the problem size is dominated by (3). To keep an efficiency greater than $E=t_c/(t_c+t_w)$ a growth $O(p^{Kt_w/t_c}\log p)$ in W is required. In this case, we can conclude that the parallel system has a low degree of scalability. It is even possible that the efficiency cannot be maintained. This will happen when the problem required to maintain the efficiency does not fit in the system memory. In particular, it should be taken into account that the size of the largest FFT which can be solved in a 2^d-node hypercube with 2^s-words (real values) of main memory per node is 2^{d+s-1}. Therefore, from expression (3) it can be concluded that efficiency E can be maintained only when:

$$d \leq \frac{s-1}{K(t_w/t_c)-1} \quad (4)$$

To illustrate the above conclusions, figures 1a, 1b and 1c show some results obtained from the analytical models used in this section. These plots show the length of the FFT required to maintain a certain efficiency for a given number of nodes. In particular we plot the values of d in the X-axis and the values of r in the Y-axis (for clarity, the Y-axis plots r instead of the problem size, which would be $r2^r$). We consider four different values for the reference efficiency: E =0.2, 0.4, 0.6 and 0.8. The corresponding values of K are 0.25, 0.66, 1.5 and 4. Finally, we consider three different values for the ratio t_w/t_c: 0.2, 1.28 and 10.7. These are the values used in [3] which correspond to the nCUBE1, nCUBE2 and iPSC/RX respectively.

In every plot, one of the curves (max-size) shows the largest FFT which can be solved in the system assuming a memory per node of $2^s = 2^{24}$ real values (128 Mbytes per node). This curve allows to identify quickly the cases in which the efficiency cannot be maintained since the required problem does not fit in the system memory.

Figure 1a shows that when $t_w/t_c = 0.2$ all the considered efficiencies can be maintained with a growth of r proportional to d, since $Kt_w/t_c < 1$ for all four values of K.

Figure 1b shows that when $t_w/t_c = 1.28$, and K=4 (E=0.8) the system has a poor scalability ($Kt_w/t_c=5.12$). Moreover, for values of d greater than 5 the efficiency cannot be maintained, as predicted by expression (4). For the rest of values of E the system is scalable.

Finally, figure 1c shows that when $t_w/t_c = 10.7$ the system is scalable only for E=0.2 and only up to d=13.

It is important to remark that varying the memory size per node does not introduce any new conclusion to the analysis.

2.3 Time-constrained scalability analysis

In this section we propose an alternative analysis of the binary-exchange algorithm on a hypercube, using a user-oriented analysis method that utilizes the time-constrained scaling model [4] and the problem size figure of merit. This method is a particular example of what we call a user-oriented scalability analysis method. It will be one of the methods formally introduced in section 3. In this section however, the emphasis is in showing that the results of the analysis are different from and more relevant than those of isoefficiency analysis.

In the time-constrained scaling model, when increasing the number of nodes the user increases the problem size as much as possible but without exceeding a certain reference execution time *T*. The scalability analysis determines to what extent the problem size can be increased. Under this model, a system is scalable if the increase in the problem size can be proportional to the increase in the number of nodes. Below, we highlight the main steps of this analysis.

In the scalability analysis that follows, it is assumed that the reference time *T* is the time required to solve the largest possible problem in a single node. Assuming a main memory of $R=2^s$ real numbers per node the largest FFT has length 2^{s-1} and therefore:

$$T = (s-1)\,2^{s-1}t_c$$

If we have a 2^d -node system, the length ($n = 2^r$) of the largest FFT that can be solved without exceeding a time *T* is given by:

$$T_p = T \Rightarrow r2^{r-d}t_c + dt_s + d2^{r-d}t_w = (s-1)\,2^{s-1}t_c \tag{5}$$

The requirement on *W* due to the term $r2^{r-d}t_c$ in (5) is given by:

$$W = r2^r = (s-1)\,2^{d+s-1} = O(p) \tag{6}$$

The term dt_s in (5) does not impose any requirement on W.

The requirement on W due to the term $d2^{r-d}t_w$ in (5) is given by:

$$2^r = \frac{(s-1)\,2^{d+s-1}t_c}{dt_w} \Rightarrow$$

$$r = d+s-1+\log_2\frac{(s-1)\,t_c}{dt_w} \Rightarrow$$

$$W = r2^r = (s-1)\,2^{s-1}\frac{t_c}{t_w}2^d\left(1-\frac{L(d)}{d}\right) \tag{7}$$

where:

$$L(d) = \log_2\frac{dt_w}{(s-1)\,t_c}+1-s$$

Expression (6) corresponds to the ideal case (perfect scalability) in which increasing the number of nodes allows a proportional increase in the problem size, without exceeding *T*. Expression (7) reflects to what extent the real case differs from the ideal case. In particular, when $L(d) \le 0$ we are in the ideal case in which $W=O(p)$. This will happen when:

$$d \le \frac{(s-1)\,t_c}{t_w}2^{s-1} \tag{8}$$

On the other hand, when $L(d) > 0$, that is, when (8) does not hold, an increase in problem size proportional to the increase in the number of nodes is not possible. In this case, there is also a limit in the number of nodes that can be used to solve the problem under the scaling model. To evaluate that limit, we have to take into account that in a 2^d -node system with 2^s real values of main memory per node, the smallest FFT that can be solved has length 2^{d-s+1}. For a certain value of *d*, the computation of the smallest FFT will take longer that *T*. This value of *d* is the limit in the system size. That limit can be obtained from the following expression:

$$d+s-1+\log_2\frac{(s-1)\,t_c}{dt_w} \ge d-s+1$$

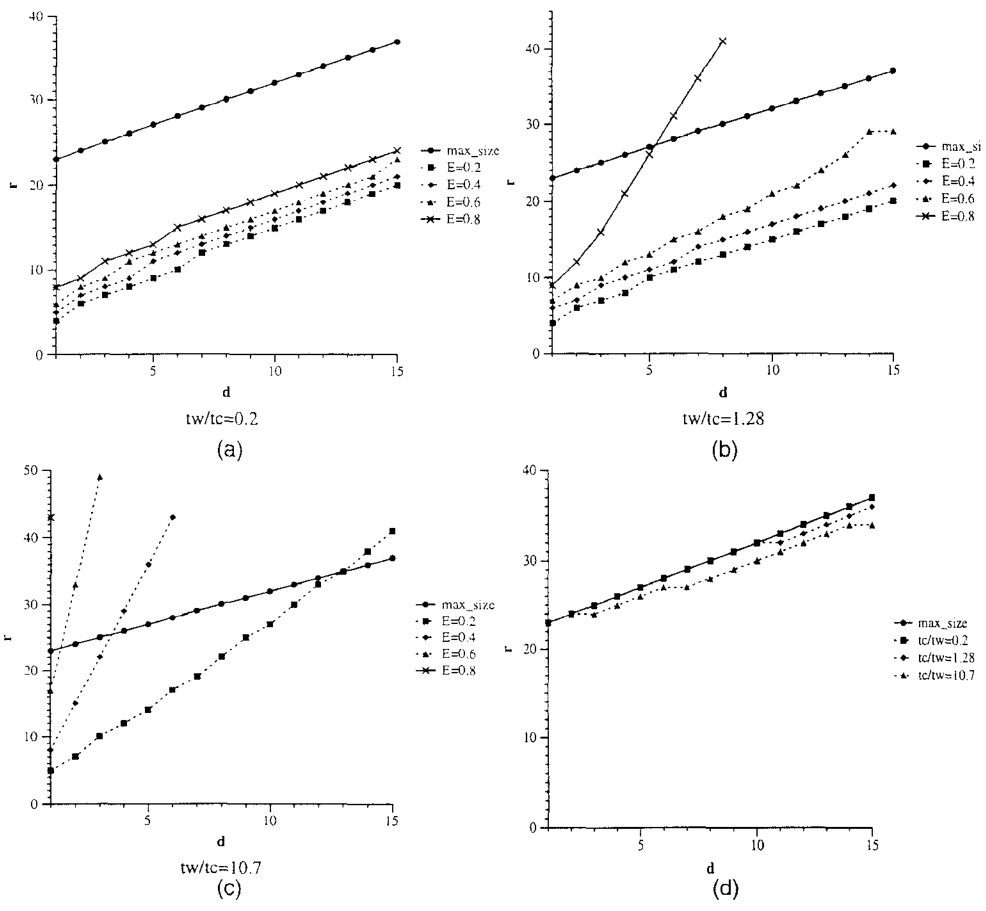


Figure 1: (a), (b) and (c): Length of FFT (2^r) required to maintain a constant efficiency E when increasing the number of nodes (2^d), for different values of t_w/t_c. (d) Length of the FFT which can be solved without exceeding the reference time, when increasing the number of nodes.

The left hand of the inequality corresponds to the length of the FFT which takes a time T in the parallel system. This length should be greater than or equal to that of the smallest FFT. Otherwise, the smallest FFT would take longer than T. From the above expression the following limit for d can be derived:

$$d \leq \frac{(s-1)\,t_c}{t_w} 2^{2s-2} \tag{9}$$

Anyway, it can be observed that, for the range of (reasonable) values we are considering for d, t_w/t_c and s, conditions (8) and (9) are always satisfied, and therefore $W=O(p)$. We can conclude that the system is scalable in all the considered cases.

We illustrate also these conclusions with some plots. Figure 1d shows the largest FFT (values of r in the Y-plot) that can be solved without exceeding time T, using 2^d nodes (values of d in X-axis). We have considered the same three different values for t_w/t_c (0.2, 1.28 and 10.7) and s=24

(128 Mbytes per node), as in the previous section. The plots show that the system is quite scalable in all cases.

As a final conclusion of this section, we have shown that isoefficiency analysis and time-constrained scalability analysis give different conclusions about the scalability of a particular system. In addition, the conclusions of the time-constrained scalability analysis are more relevant to the system user. The definition of scalability analysis methods that are of interest for users is the main motivation of the methodology for scalability analysis which is formally described in the next section.

3 A methodology for scalability analysis

The two key issues in the proposed methodology are: (a) the appropriate combination of the scaling model and the figure of merit and, (b) the proposed scalability characterization, which is common to all the methods in the user-oriented family, and which is aimed at facilitating the analysis and comparison for systems.

In this section we first introduce the proposed scalability characterization. Then, three concrete user-oriented methods are proposed. These methods use the following combinations of scaling model and figure of merit: (fixed-problem size, speed-up), (time-constrained, problem size) and (memory-constrained, problem size).

3.1 Scalability characterization

A method for scalability analysis characterizes the scalability of the system in terms of the evolution of a figure of merit under a certain scaling model. We denote by *F(p)* the figure of merit and define $\Delta F(p,m)$ as:

$$\Delta F(p, m) = \frac{F((1+m)p) - F(p)}{F(p)}$$

$\Delta F(p,m)$ represents the increment in the figure of merit *F(p)* when increasing the number of nodes a m*100%. As an example, if $\Delta F(100,0.8)=0.3$, then the figure of merit *F(p)* has an increment of *30%* when moving from a system with 100 nodes to a system with 180 nodes (a increase of *80%* in the system size).

Finally, to characterize the scalability of the system we propose to plot the function *H(p,m)*, defined as:

$$H(p, m) = \frac{\Delta F(p, m)}{m}$$

Again as an example, if $\Delta F(100,0.8)=0.3$ then $H(100,0.8)= 0.375$. Function *H(p,m)* relates the increase in the figure of merit with the increase in the number of nodes. The ideal situation (high scalability) is $H(p,m) = 1$ which means that the increase in the system size is fully amortized in terms of figure of merit improvement. It is even possible that $H(p,m) > 1$, reflecting a situation of superscalability. Finally, if $H(p,m) < 0$ then the system size increment results in a reduction in the figure of merit. A quick visual inspection of *H(p,m)* gives a global overview of the scalability of the system.

The characterization of the scalability must also show the limits in the number of nodes which can be used under the associated scaling model. Section 2 provided some examples of these limitations. When we used the isoefficiency analysis we showed that the reference efficiency cannot be maintained for systems larger than a certain limit, since the required problem does not fit in the available memory, assuming no virtual memory. On the other hand, when we used the time-constrained scaling model, we showed that for a certain number of nodes the smallest possible problem has an execution time larger than the reference time which cannot be exceeded.

To conclude this section, we introduce some definition which will allow later to characterize, for each scalability analysis method, the limits in the number of nodes. Some of these definitions have been informally used in section 2.

We denote by n the size of the input data of the problem to be solved (size of vectors, matrices, etc.).

Let $W(n)$ be the number of elementary operations required to solve a problem with input data size n, when using the optimal sequential algorithm. In the following, we refer to $W(n)$ as the problem size. We denote by t_c the time required to perform an elementary operation.

In this work, we assume that W depends exclusively on n and, therefore, increasing the problem size is considered equivalent to increasing the input data size. Other authors have pointed out that the input data size is not always the only parameter which determines the problem size [6]. Examples of other parameters which influence the problem size are the numerical accuracy or the interval between timesteps. Anyway, we believe that the proposals of this paper can be easily extended to a more general view of the problem size.

We denote by R the size of the main memory per node (measured in terms of real values). As stated before, we assume in this work that the systems does not have virtual memory.

Let $M(n,R)$ be the minimum number of nodes required to solve a problem with input data size n, assuming a main memory of size R per node.

Let $M^{-1}(p,R)$ be the input data size of the largest problem which can be solved in a system with p nodes, each with a main memory of size R.

We denote by $\Gamma\,(n)$ the maximum number of nodes which can be used in the parallel system to solve a problem with input data size n.

Let $\Gamma^{-1}\,(p)$ be the input data size of the smallest problem which can be solved in a system with p nodes.

3.2 Three examples of user-oriented scalability analysis methods

In the following, we propose three different methods for scalability analysis which are based on the methodology introduced in this paper. Each method is targeted to a different way of using an increase in the number of nodes.

Fixed-problem size scalability analysis. In this case, it is assumed that the system is used to solve a given problem with fixed input data size n, as fast as possible (fixed-problem size scaling model). The figure of merit in this case is the speed-up, defined as:

$$F(p) = S(p,n) = \frac{T(1,n)}{T(p,n)} = \frac{W \cdot t_c}{T(p,n)}$$

where $T(p,n)$ is the time required to solve a problem with input data size n using a system with p nodes.

Under the fixed-problem size scaling model, the number of nodes has the following limitations:

$$M(n,R) \le p \le \Gamma(n)$$

This means that the number of nodes must be greater than or equal to the minimum required to solve the problem with input data size n, assuming a memory of R words per node. On the other hand, the number of nodes cannot exceed the maximum which can be used to solve the problem with input data size n.

Time-constrained scalability analysis. In this case, when increasing the number of nodes, the user increases the problem size as much as possible but without exceeding a given running time T (time-constrained scaling model).

We define $n_T(p)$ as the input data size of the problem which takes time T in a system with p nodes. That is:

$$T_p(n_T(p)) = T$$

In general, it is possible that the memory required by a problem with input data size $n_T(p)$ exceeds the available main memory. We define now $n_L(p)$ as the input data size of the greatest problem that can be solved in a system with p nodes, without exceeding time T and without exceeding the available main memory:

$$n_L(p) = min(M^{-1}(p, R), n_T(p))$$

The figure of merit is defined in this case as:

$$F(p) = W(n_L(p))$$

We must take into account that $n_L(p)$ must satisfy the following condition:

$$\Gamma^{-1}(p) \le n_L(p)$$

The above condition is not satisfied when the smallest problem that can be solved in a system with p nodes takes a time greater than T. This condition establishes the limit to the number of nodes of the system under the time-constrained scaling model.

Memory-constrained scalability analysis. In this case, when increasing the number of nodes the user increases the problem size as much as possible without exceeding the main memory of the system (memory-constrained scaling model). In this case, the figure of merit $F(p)$ is simply the size of the greatest problem which can be solved in a system of p nodes, with main memory R per node:

$$F(p) = W(M^{-1}(p, R))$$

The largest problem should be greater than or equal than the smallest problem which can be solved in a system with p nodes. Otherwise, the increase in the number of nodes does not enable an increase of the problem size. Again, this condition establishes a limit to the number of nodes of the system. This results in the following constraint:

$$M^{-1}(p, R) \ge \Gamma^{-1}(p)$$

4 Example of application

In this section we illustrate the proposed methodology by applying it to the binary-exchange algorithm on a hypercube (see section 2). The plots which characterize the scalability of the system for the cases of fixed-problem size and time-constrained methods are presented. The emphasis is not in the conclusions about the scalability of the analysed system but in showing that the plot which characterizes the scalability is a useful approach for quick analysis and comparisons.

To obtain $H(p,m)$ we take into account that a hypercube has a number of nodes that is a power of 2. Therefore, after increasing the system size, the number of nodes is always a power of 2 times the previous one. This means that the possible values for m are 1,3,7,... and, in general, 2^i-1. Therefore $H(p,m)$ can be expressed as:

$$H(p,m) = H(2^d, 2^i - 1) = \frac{\Delta F(2^d, 2^i - 1)}{2^i - 1} = \frac{F(2^{d+i}) - F(2^d)}{F(2^d)(2^i - 1)}$$

Assuming that $n=2^r$ and $R=2^s$, the expressions defined in section 3.1 take the following values:

$$W(2^r) = r2^r$$

$$T(2^d,2^r) = r2^{r-d}t_c+d(t_s + 2^{r-d}t_w)$$
$$M(2^r,2^s) = 2^{r+1-s}$$
$$M^{-1}(2^d,2^s) = 2^{d+s-1}$$
$$\Gamma(2^r)= 2^r$$
$$\Gamma^{-1}(2^d) = 2^d$$

Figures 2a and 2b plot the function H for the case of fixed-problem size analysis and time-constrained analysis. For the sake of simplicity, we plot d and i in the axes (instead of p and m). We consider three values for the ratio t_w/t_c (0.2, 1.23 and 10.7, corresponding to the cases of the nCUBE1, nCUBE2 and iPSC/RX, respectively). In all cases we assume s=24.

Figures 2a and 2b show that a plot of H provides a quick visual information about the scalability of the system. The plot corresponding to the time-constrained analysis and t_w/t_c = 0.2 shows a case of perfect scalability. The rest of plots show situations with a almost perfect scalability situation, since in all cases, H is close to 1.

More detailed information on the scalability of the system may require a two-dimensional cut of H, fixing a value of d or i. Figure 2c shows two examples, which compare results of the considered analysis methods when doubling the number of nodes (fixing i=1) and when fixing the number of nodes to d=10. It can be readily seen, for instance, that the system is slightly more scalable under the time-constrained model.

5 Conclusions

This paper has shown that different analysis methods can give different conclusions about scalability of the same parallel system. This is not a new observation. In [1] for instance, it was shown that CMP and isoefficiency methods may produce very different conclusions. In fact, different analysis methods may serve different purposes. Some methods can help the system architect in determining hardware and/or software bottleneck. Others can assist the user in estimating the benefits when increasing the system size. In this case of user-oriented analysis, different methods may be oriented to different types of users.

This paper proposes a simple approach to conceive user-oriented scalability methods. To illustrate the methodology, three concrete methods are proposed, targeted to different ways of using the system. Many other methods can be defined by establishing the scaling model and the figure of merit to be improved when increasing the system size.

The proposed methodology characterizes the scalability by using a three-dimensional plot which enables a quick visual analysis and comparison of systems.

References

[1] M.A. Fienup and S.C. Kothari, "A Memory-Constrained Scalability Metric," *Proc. Int'l Conf. on Parallel Processing*, pp. III-1, III-7, 1994

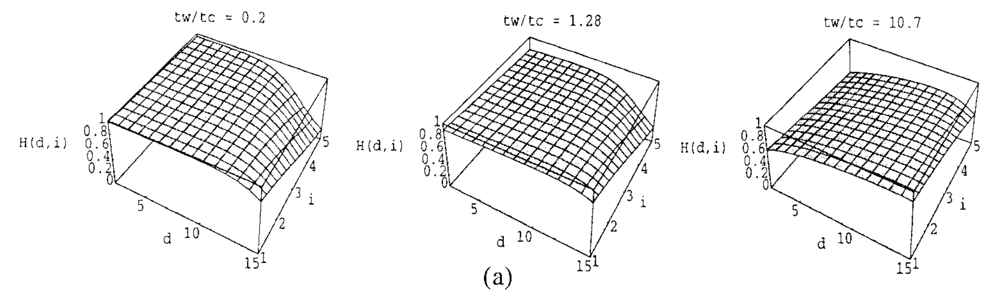
[2] A.Y. Grama, A. Gupta and V. Kumar, "Isoefficiency: Measuring the Scalability of Parallel Algorithms and Architectures," *IEEE Parallel & Distributed Technology*, pp. 12-21, August 1993.

[3] A. Gupta and V. Kumar, "The Scalability of FFT on Parallel Computers," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 4, No. 8, pp. 922-932, 1993

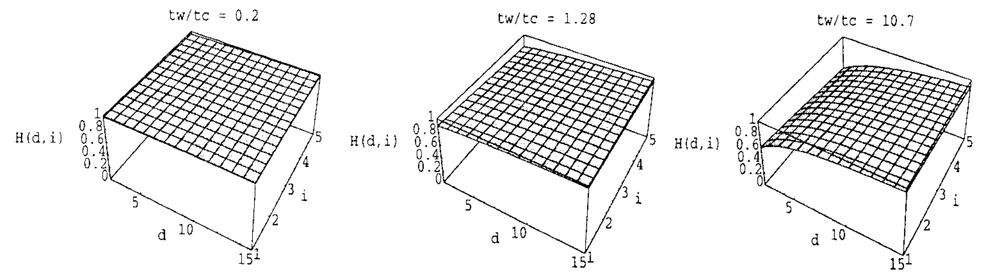
[4] K. Hwang, "Advanced Computer Architecture. Parallelism, Scalability and Programmability," *Morgan Kaufmann Publishers INC*, 1990.

[5] V. Kumar and A. Gupta, "Analysing Scalability of Parallel Algorithms and Architectures," *Journal of Parallel and Distributed Computing* 22, 379-391 (1994)

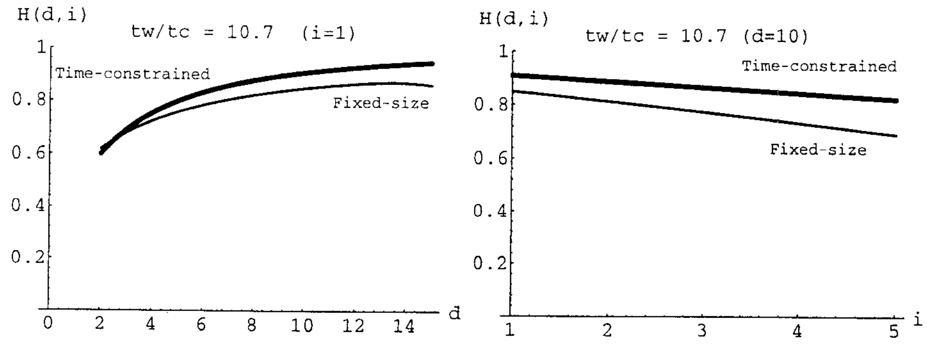
[6] J.P. Singh, J.L. Hennessy and A. Gupta, "Scaling Parallel Programs for Multiprocessors: Methodology and Examples," *IEEE Computer*, pp. 42-50, July 1993.



(a)



(b)



(c)

Figure 2: (a) Scalability plots corresponding to the fixed-problem method. (b) Scalability plots corresponding to the time-constrained method. (c) Two-dimensional cuts of H, fixing a value of i or a value of d.