Journal of Mathematical Chemistry, Vol. 40, No. 1, July 2006 (© 2006)
DOI: 10.1007/s10910-006-9125-6

Analyzing scalability of Neville elimination

P. Alonso*

Department of Mathematics, University of Oviedo, Campus de Viesques, E-33203 Gijón, Spain
E-mail: hpc@aic.uniovi.es

R. Cortina, I. Díaz and J. Ranilla

Artificial Intelligence Center, University of Oviedo, Campus de Viesques, E-33203 Gijón, Spain

The scalability of a parallel system is a measure of its capacity to effectively use an increasing number of processors. Several performance evaluation metrics have been developed to study the scalability of parallel algorithms and architectures. The isoefficiency function is one of those metrics. It relates the size of the problem being solved to the number of processors required to maintain efficiency at a fixed value. This work studies the scalability of Neville elimination, which is a method to solve a linear equation system. This process appears naturally when the Neville strategy of interpolation is used to solve linear systems. The scalability behavior of some algorithms of this method is studied on an IBM SP2 and also over a network of personal computers using the isoefficiency function and the scaled efficiency.

KEY WORDS: Neville elimination, scalability, isoefficiency function, performance

Mathematics Subject Classification (2000): MSC 65F05, MSC 65Y20, MSC 68W10

1. Introduction

It is accepted that the fastest algorithm for solving a given problem is the best one in sequential computing. However, parallel computing introduces additional sources of complexity: if we are interested in building "good" programs we also need to manage the execution of thousands of processes and to coordinate millions of interprocess interactions.

As early as 1967, Amdahl [4] made the observation that if the sequential component of an algorithm accounts for s percent of program execution time, then the maximum possible speedup that can be achieved on a parallel computer is $1/s$. This statement, now popularly known as Amdahl's Law, has been used by Amdahl and others to argue against the usefulness of large scale parallel computers.

However, the time taken by a parallel algorithm depends on many and more complex multifaceted issues than the serial fraction s, like, for example, the

*Corresponding author.

degree of locality, the communication model, the degree of concurrency, latency, throughput, the temporal relationship between the concurrent events or potential for reuse. The relative importance of these diverse metrics will vary according to the nature of the problem at hand.

Thus, the time taken by a parallel algorithm to solve a problem instance provides only limited information. In fact, an algorithm that yields good performance for a selected problem on a fixed number of processors on a given machine may perform poorly if any of these parameters are changed. Moreover, the best solution may differ from that suggested by existing sequential algorithms.

Hence, the evaluation of a parallel algorithm on a parallel computer requires a more comprehensive analysis. A good performance model, like a good scientific theory, is able to explain available observations and predict future circumstances, while disregarding unimportant details. We can use metrics like speedup or efficiency to explore and refine a parallel algorithm design, and thus, perform qualitative analyses of performance without further refinement. We can also perform a quantitative analysis, which is obtained by substituting machine specific numeric values for the various parameters in performance models.

However, large parallel computers are frequently used not only to solve fixed-size problems faster, but also to solve larger problems. This observation encourages a different approach to algorithm analysis, whereby we do not consider how efficiency varies with the number of processors (p), but rather how the amount of computation performed must scale with p to keep the efficiency constant. This approach is based on a measurement called scalability.

A set of metrics has been developed to study the scalability of parallel algorithms and architectures. Kumar and Gupta [16] provide a comprehensive survey of different methods of scalability analysis. The isoefficiency function is one such metric. It relates the size of the problem being solved to the number of processors required to maintain efficiency at a fixed value. This function allows the degree of scalability of a parallel system to be determined with respect to the number of processors, their speed, and the communication bandwidth of the interconnection network (see [12,13]).

This paper models and measures just one aspect of Neville elimination performance: parallel scalability. We focus on this issue because it frequently figures among the more problematic aspects of parallel program design, yet it is easily formalized in mathematical models.

Neville elimination is a method to solve a linear equation system, which appears naturally when the Neville strategy of interpolation is used for resolving linear systems. This elimination works by making zeros in a matrix column by adding to each row an adequate multiple of the previous one. This process is an alternative to Gaussian elimination and proved to be better than the latter when working with totally positive matrices, sign-regular matrices or other related types of matrices (see [8,11]). For example, the computational cost is smaller for

Neville elimination than for Gaussian elimination when working with the inverse matrix of a band matrix (see Theorem 2.7 of [10]).

Solution of the matrix equation and calculation of the matrix inverse of a square matrix are recurrent tasks handled by molecular modeling software (see [5]). The calculation of the inverse of positive matrices is a common task in Computational Chemistry. Thus, in Quantum Chemistry, inversion of the overlap matrix between basis functions is required to obtain the electronic energy and to perform charge density analyses. Similarly, inversion of the hessian matrix, which stores the second derivatives of molecular energy with respect to nuclear coordinates, is frequently demanded by optimization techniques and normal mode analyses. Hence, the availability of more efficient direct methods for solving matrix equations could be of particular interest.

This work is organized as follows: section 2 introduces a number of metrics commonly used to evaluate the performance of parallel systems. Section 3 briefly describes Neville elimination. Section 4 focuses on the analysis of the isoefficiency function applied to the Neville method when different data distributions are considered. In this section, the result of this analysis is compared to the same known analysis performed with Gaussian method. Finally, section 5 shows how the hardware related parameters and the number of processors affect the isoefficiency function.

2. Performance metrics for parallel systems

Certain metrics that are commonly used to measure the performance of parallel systems are introduced in this section (see [12,16]). For a more detailed study of these parameters, see [13].

This work focuses on the study of the scalability of a parallel system. A parallel system is said to be scalable if its efficiency can be kept fixed as the number of processors is increased, provided that the problem size is also increased.

For different parallel systems, problem size must increase at different rates in order to maintain a fixed efficiency as the number of processors is increased. This rate determines the degree of scalability of the parallel system.

The problem size (W) is defined as the number of basic operations needed to solve the problem by the fastest known sequential algorithm on a single processor, while the overhead function of a parallel system is defined as the part of its cost that is not incurred by the fastest known serial algorithm on a sequential computer ($T_0 = pT_p - W$). Therefore,

$$W = KT_0(W, p), \tag{1}$$

where K is a constant depending on efficiency. This function dictates the growth rate of W required to keep efficiency fixed as p increases. This is called the isoefficiency function of the parallel system.

The isoefficiency function determines the ease with which a parallel system can maintain a constant efficiency, which is achieved if the ratio T_0/W remains fixed.

If T_0 has multiple terms, we balance W against each term of T_0 and compute the respective isoefficiency functions for individual terms.

3. The Neville method

A system of equations $Ax = b$ is usually solved in two stages ($A = LU$). First, through a series of algebraic manipulations, the original system of equations is reduced to an upper-triangular system $Ux = y$. In the second stage, the upper-triangular system is solved by a procedure known as back-substitution.

If A is a square matrix of order n, the Neville elimination procedure consists of $n-1$ successive major steps (see [10] for a detailed and formal introduction), resulting in a sequence of matrices as follows

$$A = A^{(1)} \to A^{(2)} \to \cdots \to A^{(n)} = U, \tag{2}$$

where U is an upper-triangular matrix. If A is non-singular the matrix $A^{(k)} = (a_{ij}^{(k)})1 \leq i, j \leq n$ has zeros below its main diagonal in the $k-1$ first columns, and the following relationship is also satisfied:

$$a_{ik}^{(k)} = 0, \quad i \geq k \Rightarrow a_{hk}^{(k)} = 0, \quad \forall h \geq i. \tag{3}$$

To get $A^{(k+1)}$ from $A^{(k)}$ we produce zeros in the kth column below the main diagonal, subtracting a multiple of the ith row from the $(i+1)$th for $i = n-1,\ n-2,\ \ldots, k$, according to the formula:

$$a_{ij}^{(k+1)} = \begin{cases} a_{ij}^{(k)}, & \text{if } 1 \leq i \leq k, \\ a_{ij}^{(k)} - \frac{a_{ik}^{(k)}}{a_{i-1,k}^{(k)}} a_{i-1,j}^{(k)} & \text{if } k+1 \leq i \leq n \quad \text{and } a_{i-1,k}^{(k)} \neq 0, \\ a_{ij}^{(k)}, & \text{if } k+1 \leq i \leq n \quad \text{and } a_{i-1,k}^{(k)} = 0. \end{cases} \tag{4}$$

In the same way, the vector of independent terms is modified stage by stage. In the algorithms we are next going to study, we will focus on the coefficient matrix of a system of equations.

Let $Ax = b$ be a non-singular system of linear equations. Starting from the last expressions, we obtain the following algorithm:

Algorithm 1. Serial Neville elimination

For $j = 1$ **to** $n-1$ **do**
Compute the $(n-j)$ multipliers

Update the active part of the matrix
End j

Let us consider the case in which Neville elimination can be performed without changing rows, which happens, for example, when A is a non-singular totally positive matrix. The work presented in [8] shows that row changes are not necessary in A when the Neville elimination process is applied to a totally positive matrix. This is particularly useful in maintaining band structure when working with this kind of matrix. For example, when an interpolation problem is solved by using B-splines, their collocation matrices are totally positive band matrices.

The same work also proves that the multipliers obtained after transforming A to U are the same as those obtained from converting L to I in Neville as well as Gaussian elimination.

However, in the case of Neville elimination, those multipliers are opposite to those obtained from getting L^{-1} to I; this is not true for Gaussian elimination.

A lot of computations become unnecessary if L^{-1} is a band matrix in Neville elimination as well as Gaussian elimination. However, L is usually a dense lower-triangular matrix and so it is not possible to save computations when Gaussian elimination is performed on L or on A. On the other hand, the cost of Neville elimination for L or A is the same as for L^{-1}. In general, the number of non-zero multipliers of Neville elimination for A is the minimum of the number of non-zero multipliers needed for Gaussian elimination applied to L and L^{-1}.

Regarding the use of pivoting strategies, Gasca [9] proves that in exact arithmetic the Neville elimination does not need row exchange when scaled partial pivoting is used and A is a non-singular totally positive matrix. The same result holds, for sufficiently high-precision arithmetic, for a class of totally positive matrices, which nevertheless includes the most interesting of them, i.e., B-spline collocation matrices, Hurwitz matrices, etc.

On the other hand, it is known that with finite arithmetic, Gauss elimination without pivoting produces a small component-wise relative backward error when totally positive matrices are used (see [14]). In [1], it is proved that the backward error bounds obtained by Neville elimination are quite similar to those obtained by other authors for Gauss elimination.

Let us consider an elimination process without row changes. Neville elimination involves approximately $n^2/2$ divisions and $n^3/3$ subtractions and multiplications. If the cost associated to the back-substitution is rejected, the sequential run time of the procedure is:

$$T_s = \frac{4n^3 - 3n^2 - n}{6} t_c \approx \frac{2n^3}{3} t_c, \tag{5}$$

where t_c is the time spend carrying out float point operation. This cost coincides with the cost of sequential Gaussian elimination.

4. The isoefficiency function of Neville elimination

This section studies the isoefficiency function for parallel Neville elimination for different data distributions of the matrix A. To do this, a MIMD Distributed-Memory machine with p processors will be considered.

Let us consider the block-cyclic-striped mapping, where the matrix is divided into blocks of m consecutive rows (columns). These blocks are distributed among the processors cyclically, so each processor has assigned h blocks, where $h = n/mp$ and $h \in [1, n/p]$. This algorithm is known as row-wise block-striped partitioning (column-wise block-striped partitioning) if $h = 1$ and row-wise cyclic-striped partitioning (column-wise cyclic-striped partitioning) if $h = n/p$.

4.1. Row-wise striping

If a row-wise block-cyclic striping data distribution among processors is considered, the Neville method can be described by the following algorithm:

Algorithm 2. Row-wise block-cyclic striping in Neville elimination

For j = 1 **to** n–1 **do**
For each processor l (being l an active processor) **do**
Send the last row of each block to the next processor (l + 1)
Compute the multipliers
Update the active part of the matrix
End j

Due to communications being between neighboring processors (directly connected), the cost of sending a message of size n is

$$t_s + n\ t_w, \tag{6}$$

where t_s denotes the startup time and t_w is the transmission time of a floating-point number.

Let us now analyze the cases $h = 1$ and $h = n/p$.

Case 1. If $h = 1$ (row-wise block-striped), the results obtained in [2] guarantee that the overall parallel run time is:

$$T_p^{\text{row/block}} \approx \frac{n^3}{p} t_c + n\ t_s + \frac{n^2}{2} t_w. \tag{7}$$

According to the definition of the size of the problem given in section 2, we can state that W is $\Theta(n^3)$ if Neville elimination is used to solve a linear equation system of order n.

The relation $T_0 = pT_p - W$ gives the following expression for the total overhead function:

$$T_0^{\text{row/block}} \approx \frac{1}{3}n^3 t_c + np\ t_s + \frac{1}{2}n^2 p\ t_w. \tag{8}$$

Thus, the communication overhead is as follows:

$$np\ t_s + \frac{1}{2}n^2 p\ t_w. \tag{9}$$

As the approximation of communication overhead has multiple terms, W is balanced against each individual term of (9) to compute the respective isoefficiency function.

As for the term t_s, the following expression must be satisfied:

$$W = np = W^{1/3}p \Rightarrow W^{2/3} = p \Rightarrow W = p^{3/2} \tag{10}$$

Similarly, to determine the isoefficiency term due to t_w, n^3 has to be proportional to $n^2 p t_w$. Therefore,

$$W = n^2 p = W^{2/3}p \Rightarrow W^{1/3} = p \Rightarrow W = p^3. \tag{11}$$

On the other hand, if the communication costs are ignored in (8), the cost of this algorithm is n^3. Thus, the cost of the parallel algorithm is higher than the sequential run time (5) by a factor of 3/2. This inefficiency with row-wise block-striped partitioning is due to processor idling resulting from an uneven load distribution.

As a result of the estimations carried out above, we can conclude that the overall asymptotic isoefficiency function of this parallel system is $\max\{p^{3/2},\ p^3\}$, which is $\Theta(p^3)$.

Case 2. If $h = n/p$ (row-wise cyclic-striped), by applying the same argument as in the previous case, we obtain:

$$T_p^{\text{row/cyclic}} \approx \frac{2n^3}{3p}t_c + n\ t_s + \left(\frac{n^3}{3p} + \frac{n^2}{4}\right)t_w. \tag{12}$$

In this case, the processors have the same work load due to the cyclic-striped mapping and, therefore, the following expression for the overhead function is obtained:

$$T_0^{\text{row/cyclic}} \approx np\ t_s + \left(\frac{n^3}{3} + \frac{n^2 p}{4}\right)t_w. \tag{13}$$

As in case 1, the isoefficiency term with respect to message startup time is $\Theta(p^{3/2})$.

Since W is n^3, the term $n^3 t_w$ will always be balanced with respect to W. This term is independent of p and does not contribute to the isoefficiency function. The other term of t_w, $n^2 p$, yields the following isoefficiency function for the algorithm:

$$W = n^2 p = W^{2/3} p \Rightarrow W^{1/3} = p \Rightarrow W = p^3. \tag{14}$$

Thus, the final isoefficiency function is $\Theta(p^3)$.

4.2. Column-wise striping

As it can be seen in [3], the algorithm of a column-wise block-cyclic striping in Neville elimination is the following:

Algorithm 3. Column-wise block-cyclic striping in Neville elimination

For $j = 1$ **to** $n - 1$ do
$q = ((j - 1)$ DIV $m)$ MOD $p + 1$
In processor q do
Compute the $(n - j)$ multipliers
Send the multipliers to the rest of the active processor
For each processor l (l being an active processor) **do**
Update the active part of the matrix
End j

As regards communication time, a one-to-all broadcast is required to send the multipliers from a processor to the rest of the active processors. The cost of this operation in our platforms (IBM SP2 and PC Cluster using MPI) is:

$$(t_s + n\ t_w) \log(p), \tag{15}$$

where n is the size of the message and the logarithm is in base two (see [7,17]).

Case 1: If $h = 1$ (column-wise block-striped), the total parallel run time is:

$$T_p^{\text{column/block}} \approx \frac{n^3}{p} t_c + n \log(p)\ t_s + \frac{n^2 \log(p)}{2} t_w. \tag{16}$$

Observe the existence of the uneven load distribution as in row-wise block-striped.

As it has already been noted, the size of the problem is $W \in \Theta(n^3)$, which is independent of the algorithm used. Furthermore, the communication overhead is:

$$np \log(p)\ t_s + \frac{n^2 p\ \log(p)}{2} t_w. \tag{17}$$

Table 1
Isoefficiency function.

Method	Row/block	Row/cyclic	Column/block	Column/cyclic
Gauss	$\Theta(p^3\ \log^3(p))$	$\Theta(p^3\ \log^3(p))$	$\Theta(p^3\ \log^3(p))$	$\Theta(p^3\ \log^3(p))$
Neville	$\Theta(p^3)$	$\Theta(p^3)$	$\Theta(p^3\ \log^3(p))$	$\Theta(p^3\ \log^3(p))$

To directly calculate its isoefficiency function we can obtain the term that dominates in T_0, that is,

$$np\log(p)\ t_s + \frac{1}{2}n^2 p\ \log(p)t_w = np\log(p)\left(t_s + \frac{1}{2}n\ t_w\right) \approx n^2 p\log(p), \quad (18)$$

thereby,

$$W = n^2 p\log(p) = W^{2/3} p\log(p) \Rightarrow W = (p\log(p))^3. \quad (19)$$

Hence, the isoefficiency function is $\Theta(p^3\ \log^3(p))$.

Case 2: If $h = n/p$ (column-wise cyclic-striped)

$$T_p^{\text{column/cyclic}} \approx \frac{2n^3}{3p}\ t_c + n\log(p)\ t_s + \frac{n^2\ \log(p)}{2}\ t_w. \quad (20)$$

The associated overhead function in this case is:

$$T_0^{\text{column/cyclic}} \approx np\log(p)\ t_s + \frac{n^2 p\ \log(p)}{2}\ t_w. \quad (21)$$

The isoefficiency term due to t_s is $\Theta(p^{3/2}\ \log^{3/2}(p))$ and the relative one to t_w is $\Theta(p^3\ \log^3(p))$. Therefore, overall asymptotic isoefficiency of this parallel system is $\max\{p^{3/2}\ \log^{3/2}(p),\ p^3\ \log^3(p)\}$, which is $\Theta(p^3\ \log^3(p))$.

4.3. Neville versus Gauss

In this section, the isoefficiency functions computed in the previous one are compared with those obtained from the Gaussian method under the same conditions.

The differences between Neville and Gauss depend on the communications. In a row-wise striping distribution, whilst Neville realizes communications between neighboring processors, Gauss carries out one-to-all broadcasts. In case of columns, the communications are the same.

As shown by table 1, the isoefficiency function for the Neville method is lower than the Gaussian one when a row-wise striping is performed. This function has the same value when the partitioning is column-wise.

For example, the total run time of Gauss elimination when the matrix is partitioned among the processors by using cyclic-striped mapping is:

$$T_p^{\text{row/cyclic}} \approx \frac{2n^3}{3p}\ t_c + n\log(p)\ t_s + \frac{1}{2}n^2\ \log(p)\ t_w. \tag{22}$$

The isoefficiency functions due to t_s and t_w are $(p\ \log(p))^{3/2}$ and $(p\ \log(p))^3$, respectively. Hence, the overall isoefficiency function due to the communication overhead is $\Theta(p^3\ \log^3(p))$.

5. Numerical results

The previous sections studied the scalability of the Neville method using different partitioning strategies from a theoretical point of view. Thus, section 4 shows that the theoretical isoefficiency function ranges from $\Theta(p^3)$ and $\Theta(p^3\ \log^3(p))$, according to the kind of partitioning applied. In addition, it is shown how the isoefficiency expression is qualified by the constants t_c, t_s, and t_w which depend on the computer used.

This section will now analyze the influence of the values of these constants. Obviously, the order of the function will not change, but the real (empirical) needs of the problem size growth depend on these values. To analyze the influence of t_c, t_s and t_w constants, they have been estimated in our platforms: IBM SP2 (*thin*2 processors at 1.6 MHz and *TrailBlazer*3 switch) and PC Cluster (Intel *Pentium III* at 733 MHz and Fast Ethernet). An adequate distribution of Message Passing Interface (MPI) is used when necessary.

Equation (23) shows the estimations obtained for the IBM SP2

$$t_c = 1.6 \times 10 - 8\ \text{s}, \quad t_s = 3.1 \times 10 - 5\ \text{s}, \quad t_w = 1.2 \times 10 - 7\ \text{s} \tag{23}$$

and equation (24), those for the PC Cluster

$$t_c = 3.8 \times 10 - 8\ \text{s}, \quad t_s = 8.8 \times 10 - 5\ \text{s}, \quad t_w = 1.3 \times 10 - 6\ \text{s}. \tag{24}$$

All these values were obtained using a least square approximation. Similar studies can be seen in [6] and [15].

As we have seen in section 4, the isoefficiency function for the case of row-wise block-striped is $\Theta(p^3)$. This means that increasing the number of processors by a factor of $\bar{p}/p$ requires W to be increased by a factor of $\bar{p}^3/p^3$. Using the estimations of (23), let us suppose we are solving a system of size 960^3 using four processors. At this point, the efficiency is equal to 0.65. We want to analyze the scalability when the machine size increases to eight processors. To keep efficiency constant (at 0.65 value), the system size needs to grow to 1920^3.

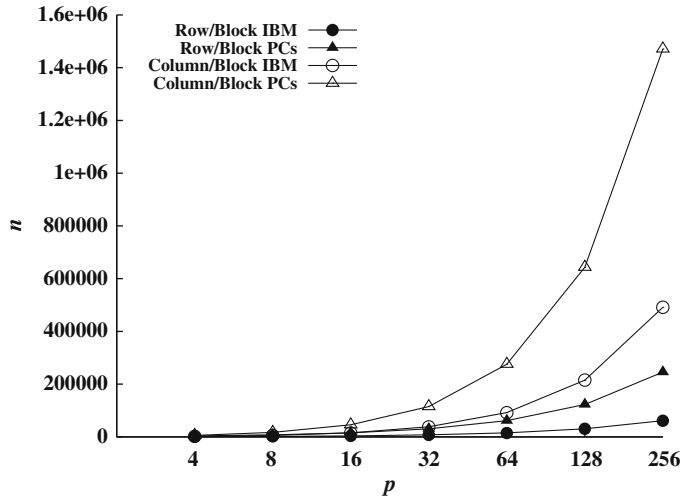

Figure 1. IBM SP2 and PC Cluster isoefficiency curves for $E = 0.65$.

In the case of column-wise block-striped, with the same parameters, the efficiency of the system is 0.64. The isoefficiency function given by (19), determines that if the number of processors increases to 8, the system size must increase to 2880^3.

In figure 1, theoretical isoefficiency curves $\Theta(p^3)$ and $\Theta(p^3 \log^3(p))$ are plotted for the IBM SP2 and the PC Cluster when the efficiency is 0.65. These curves show that the isoefficiency function is sensitive to changes in the constants. For example, the IBM SP2 reaches efficiency 0.65 with a problem size of 960^3, four processors and row/block distribution, while in the PC Cluster this efficiency is obtained for a system size of 3000^3 and the same number of processors. The isoefficiency function is much more sensitive to changes in t_w. Thus, a hypothetical machine with t_c and t_s from IBM SP2 and t_w from PC Cluster reaches an efficiency of 0.65, with a problem size of 7000^3, four processors and row/block distribution. If we change the value of t_s in the IBM SP2 to that from the PC Cluster, the size of the problem must be 1300^3. On the other hand, if the replaced constant is t_s the size of the problem must be 500^3.

Let us now consider scaled efficiency [18] to compare all the distributions analyzed in this paper. Table 2 shows the efficiency evolution of the IBM SP2 if the problem size and processors are doubled. As expected, efficiency worsens for increasing p, but we can say that for row distributions it decreases more slowly than for column distributions.

The scaled efficiency data shown in Figure 2 has been computed for the IBM SP2, the PC Cluster and block and cyclic distributions (rows and columns). Again, we can observe the difference between both environments for the same distribution.

Table 2
Scaled efficiency.

P	n^3	Row/block	Row/cyclic	Column/block	Column/cyclic
4	960^3	0.65	0.21	0.64	0.93
8	1210^3	0.64	0.21	0.60	0.86
16	1524^3	0.63	0.21	0.55	0.76
32	1920^3	0.62	0.21	0.48	0.63
64	2419^3	0.60	0.20	0.39	0.48
128	3048^3	0.56	0.20	0.29	0.34
256	3840^3	0.52	0.20	0.20	0.23

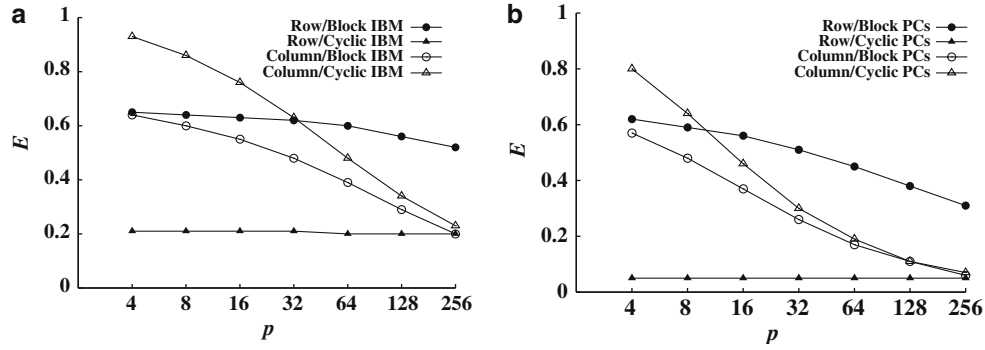

Figure 2. IBM SP2 and PC Cluster scaled efficiency.

Acknowledgments

The research reported in this paper has been supported in part under MEC and FEDER grant TIN2004-05920.

References

[1] P. Alonso, M. Gasca and J.M. Peña, Backward error analysis of Neville elimination, Appl. Numer. Math. 23 (1997) 193–204.
[2] P. Alonso, R. Cortina, V. Hernández and J. Ranilla, A study the performance of Neville elimination using two kinds of partitioning techniques, Linear Algebra Appl. 332–334 (2001) 111–117.
[3] P. Alonso, R. Cortina, I. Díaz, V. Hernández and J. Ranilla, A columnwise block striping in Neville elimination, Lect. Notes in Comput. Sci. 2328 (2002) 379–386.
[4] G.M. Amdahl, Validity of the single processor approach to achieving large scale computing capabilities. In: Proc. AFIPS Conference, (AFIPS Press, Reston, VA, 1967), pp. 483–485.
[5] C.J. Cramer, Essentials of Computational Chemistry: Theories and Models, 2nd Edn. (Wiley, England, 2004) p. 596.

[6] J.J. Dongarra, Performance of Various Computers Using Standard Linear Equations Software (Linpack Benchmark Report), University of Tennessee Computer Science Technical Report, CS-89–85 (2001).
[7] A. Farazdel, G.R. Archondo-Callao, E. Hocks, T. Sakachi and F. Vagnini, Understanding and Using the SP Switch, IBM International Technical Support Organization, SG24-5161-00 (1999).
[8] M. Gasca and J.M. Peña, Total positivity and Neville elimination, Linear Algebra Appl. 165 (1992) 25–44.
[9] M. Gasca and J.M. Peña, Scaled pivoting in Gauss and Neville elimination for totally positive systems, Appl. Numer. Math. 13 (1993) 345–356.
[10] M. Gasca and J.M. Peña, A matricial description of Neville elimination with applications to total positivity, Linear Algebra Appl. 202 (1994) 33–45.
[11] M. Gasca and C.A. Michelli, Total Positivity and its Applications (Kluwer Academic Publishers, Netherlands, 1996) p. 518.
[12] A. Grama, A. Gupta and V. Kumar, Isoefficiency Function: A Scalability Metric for Parallel Algorithms and Architectures, IEEE Parallel Distrib. Technol. Special Issue Parallel Distrib. Syst. Theory Practice, 1–3 (1993) 12–21.
[13] A. Grama, G. Karypis, V. Kumar and A. Gupta, Introduction to Parallel Computing, 2nd Ed. (Addison-Wesly, Boston, 2003) p. 656.
[14] N.J. Higham, Bounding the error in Gaussian elimination for tridiagonal systems, SIAM J. Matrix Anal. Appl. 11(4) (1990) 521–530.
[15] R. Hockney and M. Berry, Public International Benchmarks for Parallel Computers, PARKBENCH Committee: Report (1996) (http://www.netlib.org//parkbench).
[16] V. Kumar and A. Gupta, Analyzing scalability of parallel algorithms and architectures, J. Parallel Distr. Com. 22(3) (1994) 379–391.
[17] G.R. Luecke, M. Kraeva, J. Yuan and S. Spanoyannis, Performance and scalability of MPI on PC clusters, Concurr. Comput. Practice Experience 16 (2004) 79–107.
[18] M. Prieto, R.S. Montero, I.M. Llorente and F. Tirado, A parallel multigrid solver for viscous flows on anisotropic structured grids, Parallel Computing 29 (2003) 907–923.