This is a pre-final version. The final paper is available at: Efthimios Tambouris and Peter Van Santen (2000) "Scalability analysis of parallel systems with multiple components of work", Proc. of 8th Euromicro workshop on Parallel and Distributed Processing, IEEE Computer Society, pp. 182-189.

Scalability Analysis of Parallel Systems with Multiple Components of Work

E. Tambouris[1] and P. Van Santen
Department of Electronic and Computer Engineering
Brunel University
Uxbridge, UB8 3PH Middlesex, UK
{Efthimios.Tambouris, Peter.Van.Santen}@brunel.ac.uk

Abstract

In this paper, the generic fixed-value efficiency (fve) method is proposed to study the scalability of parallel algorithms with multiple components of work. The generic fve method is based on the isoefficiency method. Unlike isoefficiency however this method may be applied to parallel algorithm-machine combinations (parallel systems) where the relationship between the total work and its components is not predetermined by the decomposition method or any other factor. The objective of the method is to derive the relationships between the total work and its components in order for the efficiency of the parallel system to be preserved. The use of the method is demonstrated by analysing the impact of the sparsity of the input data on the scalability of a static state-estimator for power systems.

1. Introduction

Scalability analysis allows predicting performance of large machine and problem instances based on the known performance of small ones. A large number of metrics has been proposed for analysing scalability including isoefficiency function [1] scaled speedup [2] and isospeed [10]. It is now well-established that no single metric is better than the others as different metrics are more suitable under different conditions [3] [5] [9] [13]. For a comprehensive view of scalability the use of a set of metrics instead of a single one has been also proposed [5] [13]. In certain cases however a rapid evaluation of scalability is required. For these cases, isoefficiency analysis has been proved to be a useful tool [1] [4] [14].

The main objective of the isoefficiency analysis is to determine whether a parallel system can preserve its efficiency by increasing the work as the number of processors is scaled. Parallel systems are termed scalable if they can preserve efficiency and unscalable otherwise. Isoefficiency analysis is suitable for parallel algorithms where the relationship between the total work and its components is predetermined. Examples are some parallel algorithms derived from perfectly balanced domain decompositions. In these cases the work assigned to each processor cannot change. The objective of isoefficiency analysis is to determine whether the parallel system under investigation is scalable or not. For a certain number of processors, the analyst cannot experiment with different assignments of work to processors in order to increase efficiency.

For a certain class of parallel systems however the relationship between the total work and its components may not be implied by the decomposition method or any other factor. Examples are some parallel algorithms derived from a functional decomposition of the serial algorithm, such as the decomposed state-estimator for power systems [8]. The use of the isoefficiency method is not practical here, as all possible assignments of work to processors have to be defined and in turn analysed.

In this paper, the generic fixed-value efficiency (fve) is proposed. The objective of this method is to determine the relation that should hold between the growth rates of the total work and its components for the parallel system under investigation to preserve efficiency as the number of processors is scaled.

This paper is organised as follows. In section 2 the isoefficiency method is reviewed. In section 3, the model of parallel systems with multiple components of work is outlined. In section 4, the generic fixed-value efficiency (fve) method is introduced. In section 5, the proposed method is applied to analyse the scalability of a state-estimator algorithm for power systems. Conclusions and final comments are given in section 6.

2. Isoefficiency method

The size W of a problem may be defined as the work performed by an optimal (or the best known) sequential algorithm to solve the given problem on a single

[1] Now at Archetypon S.A., Sygrou Ave, GR- 176 72, Hellas. E-mail: tambouris@archetypon.gr

This is a pre-final version. The final paper is available at: Efthimios Tambouris and Peter Van Santen (2000) "Scalability analysis of parallel systems with multiple components of work", Proc. of 8th Euromicro workshop on Parallel and Distributed Processing, IEEE Computer Society, pp. 182-189.

processor. The overhead function $T_o(W, p)$ represents the sum total of all the overhead incurred by all the p processors during the parallel execution of an algorithm. The efficiency of a parallel system is given by

$$E = S / p = 1 / (1 + T_o(W, p) / W t_c)$$

where t_c is a machine-dependent constant. For a class of parallel systems, called scalable parallel systems, the efficiency can be maintained at a desirable value (between 0 and 1) provided W is increased as p is scaled. In order to maintain a fixed efficiency, W should be proportional to $T_o(W, p)$ or the following relation must be satisfied [1]:

$$W = K T_o(W, p) \quad (1)$$

where $K = E / ((1-E) t_c)$ is a constant depending on the efficiency to be maintained. The isoefficiency function of a parallel system is determined by abstracting W as a function of p through algebraic manipulations is equation (1).

For parallel systems where the overall overhead function T_o has multiple terms it can be very difficult or even impossible to obtain the isoefficiency function as a closed function of p. However, as p and W increase, the efficiency does not decrease if none of the terms of T_o grow faster than W. Therefore, if T_o has multiple terms, W is balanced against each term of T_o and the respective isoefficiency function for each individual term is computed. If at least one term yields no isoefficiency function the analysis terminates and the system is considered as unscalable. If all terms yield an isoefficiency function then the component of T_o that requires W to grow at the highest rate w.r.t. p determines the overall asymptotic isoefficiency function of the parallel system.

3. Parallel systems with multiple components of work

For a fairly large number of parallel systems, the amount of work at the parallel algorithm consists of multiple components. Parallel systems in this category include those where work is not equally balanced between processors, those with a serial component and those following the master-slave paradigm. They also include parallel algorithms derived from a functional decomposition of the best known sequential algorithm for the same problem. In these cases, assuming that the total work at the parallel algorithm is equal to the problem size W (i.e. there are no extra computations in the parallel algorithm) it is:

$$W = \sum_{i=1}^{p} W_i \quad (2)$$

where the growth rate of W_i is no faster than the growth rate of W (the term *no faster* as well as other order analysis terms which are used in this paper are defined in Appendix A). In this paper, it is assumed that the growth rates of the work components W_i with respect to W are not predetermined.

Other relationships between the problem size and the work components are also possible depending on the parallel system under investigation. For example, in a parallel system with a serial W_1 and a perfectly parallel W_p component, it is $W = W_1 + W_p$.

4. Generic fixed-value efficiency method

The generic fixed-value efficiency (fve) method can be applied to parallel systems where the relationship between the total work and its components is not predetermined. As a result, the total work may be distributed to the available processors in may different ways. The objective of the generic fve method is to derive the relationships between the total work and its components for the parallel system under investigation to be scalable.

In parallel systems with multiple components, the total work is related with its components as shown in equation (2). The overhead function T_o , which is a function of p and W, becomes a function of p and the components W_1 , W_2 , ..., W_p , i.e. $T_o(p, W_1 , W_2 , ..., W_p)$. Thus the main relationship for the generic fve method, i.e. $W = K T_o$ becomes:

$$W = K T_o(p, W_1 , W_2 , ..., W_p) \quad (3)$$

For generic fve analysis, the following set of relationships is examined:

$$W = K T_o(p, W_1 , W_2 , ..., W_p) \quad (4a)$$

$$W = W_1 + W_2 + ... + W_p \quad (4b)$$

There is a significant difference in the analysis of these two relationships. Equation (4a) is utilised to derive the isoefficiency function, i.e. the growth rate of W that is *required* for the parallel system under investigation to preserve efficiency. On the other hand, equation (4b) shows the *actual* growth rate of W thus posses a constraint on the relative growth rate of the work components in the parallel algorithm.

Equation (4b) suggests that W grows at the same rate as the fastest between the components W_i ($1 \le i \le p$). As the fastest component is unknown all possible cases are in turn examined. This gives a total of p cases where in case i the component W_i is assumed to be the fastest.

For each case, equation (4a) is then examined to reveal the conditions for the parallel system to be scalable. These conditions however should not violate relationship (4b).

This is a pre-final version. The final paper is available at: Efthimios Tambouris and Peter Van Santen (2000) "Scalability analysis of parallel systems with multiple components of work", Proc. of 8th Euromicro workshop on Parallel and Distributed Processing, IEEE Computer Society, pp. 182-189.

5. Scalability analysis of a power systems state-estimator

A Decomposed State Estimator (DSE) program for power systems was made available by the Brunel Institute of Power Systems [6] for evaluation purpose of the fve method. This program implements the DSE algorithm reported by J. F. Marsh et al. [8]. The algorithm itself has two important characteristics. Firstly, it is based on the master-slave paradigm; additionally the amount of work in the master and each slave are not determined by the algorithm but rather by the input data. Secondly, different variations of the algorithm can be implemented based on the expected sparsity of the input data.

In this section, the generic fve method is used to analytically predict the effect of input data sparsity on scalability. The predictions are also verified by timing measurements obtained from porting the DSE program to a Parsytec GCel 3/512.

For clarity the relevant material relating to the power systems state-estimator reported by J. F. Marsh and M. Azzam [7] and J. F. Marsh et al. [8] is summarised in sections 5.1 through 5.3. A detailed account can be found in the appropriate references [6][7][8]. The performance analysis conducted to construct all necessary models is given in section 5.4. The generic fixed-value efficiency method is presented in section 5.5 while the convergence between theoretical predictions and timing measurements is shown in section 5.6.

5.1 The conventional least-squares estimator

In a steady state analysis of power systems operation, the objective is to estimate the states x of the system given a set of measurements z and an algebraic model relating z to x. Network states are nodal voltage magnitudes and phases while measurements consist of a selection of nodal voltage magnitudes, real/reactive nodal power injections and line flows. The power system is represented by a graph that essentially contains all available information on the relationship between z and x. This graph consists of n nodes interconnected by lines.

The relationship between measurements and states is represented by a set of algebraic non-linear equations of the form:

$$z = h(\chi) + u$$

where χ is the actual states of the system, h is the measurement function and u the measurement uncertainty ($\chi \in \Re^n$; $z, u \in \Re^m$ and $m>n$).

Assuming that measurements are assigned weights then the problem of Weighted Least-Squares estimation of χ is equivalent to finding the local minimum of the performance index:

$$J = \|z - h(x)\|^2_{R^{-1}}$$

by choice of estimate x. If the non-linear function $h(x)$ is approximated by a linear form obtained from a truncated Taylor-series expansion, then the conventional least-squares estimator (CSE) for the system states is given by convergence of

$$H^T(j)\, R^{-1}\, H(j)\, \Delta x\,(j) = H^T(j)\, R^{-1}\, \Delta z\,(j) \qquad (5)$$

where $\Delta x\,(j) = x(j+1) - x(j)$, $\Delta z\,(j) = z - h(x(j))$ and $H = \partial h/\partial x$ is the Jacobian matrix of $h(x)$. H and $h(x)$ are evaluated at $x = x(j)$ and j is an iteration index. The procedure terminates when a pre-defined function of $\Delta x(j)$ (such as $max \parallel \Delta x(j) \parallel$) falls below a predetermined threshold A_o .

5.2 The decomposed state estimator

The problem of reducing run times and also improving the robustness of state estimation can be addressed by hierarchical methods, such as the decomposed state estimator (DSE) [8]. In the DSE, the overall graph of n nodes is decomposed into k non-overlapping areas (figure 1). Lines which inter-connect areas are known as 'ties'. Areas that are directly interconnected by ties are said to be 'adjacent'. Ties are terminated at nodes termed 'boundary nodes'. The total number of boundary nodes is represented by n_b while the boundary states are represented by the vector x_b . In each area, all nodes other than boundary nodes are termed 'internal nodes'. For area i , the number of internal nodes is represented by n_{ir} while the internal states are represented by the vector x_{ir} .

The CSE method presented in section 5.1 is equivalent to the DSE method shown in figure 4 (Appendix C). Details on the derivation of the DSE and the mathematical equivalence of the two methods are reported by J. F. Marsh and M. Azzam [8].

In the DSE, the problem of state estimation is solved in two levels: the upper level is assigned to a single processor while the lower level is assigned to $(p - 1)$ processors working in parallel. At the upper level, the states x_b of the boundary nodes are computed. At the lower level the states x_{ir} of the internal area i are computed for all i; furthermore matrices required for computing the new x_b during the next iteration are also calculated. Communication is required between the upper and lower level but not within the lower level. The DSE consists of the repetition of consecutive entries to lower and upper level, continued to a defined convergence.

An appropriate model for implementing the DSE is the master-slave model where the master is assigned all computations at upper level while each slave is assigned the computations at each area of the lower level.

This is a pre-final version. The final paper is available at: Efthimios Tambouris and Peter Van Santen (2000) "Scalability analysis of parallel systems with multiple components of work", Proc. of 8th Euromicro workshop on Parallel and Distributed Processing, IEEE Computer Society, pp. 182-189.

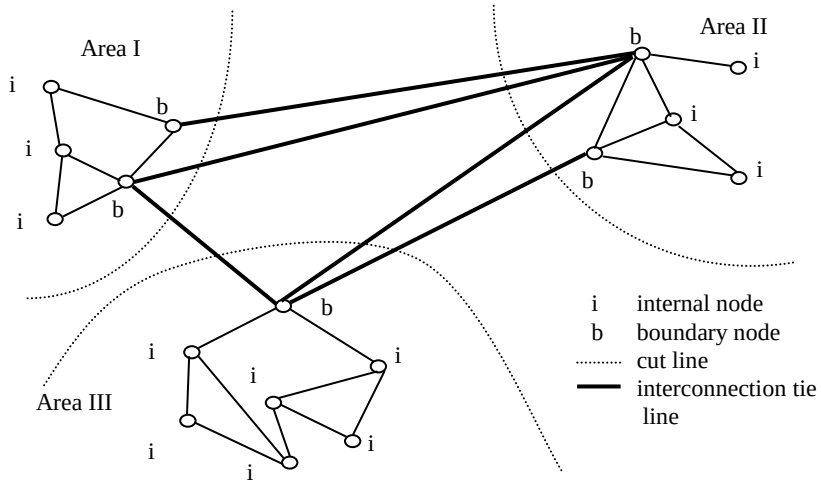

Fig. 1 Example of nodal definitions for network partitioned intro three areas

5.3 Sparsity of the input data

In most real-life power systems, each node is directly connected to a small number of other nodes hence the number of lines per node is small, usually around 3. Consequently, the Jacobian matrix *H* formed at the CSE is sparse and therefore programming techniques for sparse matrices can be used for the implementation of the CSE.

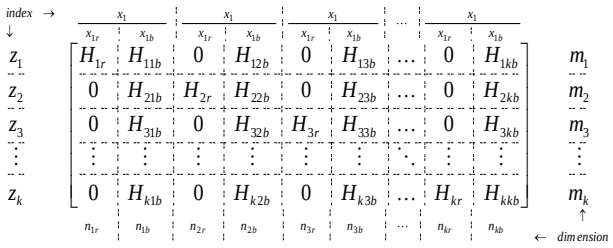

Fig. 2 Block structure of the Jacobian matrix *H*

The Jacobian *H* has a block structure, shown in figure 2, which is fully exploited in the DSE [7]. As a result, computations at lower level are based on H_{ir}, the internal Jacobian matrix for area *i*, and H_{ib}, the boundary Jacobian matrix for area *i*, whereas computations at upper level are based on the boundary Jacobian matrix H_b which has the structure:

$$H_b = \begin{bmatrix} H_{11b} & \cdots & H_{1kb} \\ \vdots & \ddots & \vdots \\ H_{k1b} & \cdots & H_{kkb} \end{bmatrix}$$

It is expected that for 'clearly decomposable' graphs, i.e. those where $n_r >> n_b$, the Jacobian H_{ir} will be sparse thus programming techniques for sparse matrices could be employed at the DSE's lower level. Furthermore, the partitioning of the graph is unlikely to result in practice in any one subsystem being interconnected with more than four others [7]. As a result, when the number of areas is large (k>6) the Jacobian H_{ib} and consequently H_b are also expected to be block-sparse. Hence, programming techniques for sparse matrices could be also employed at the DSE's upper level.

In summary, sparsity programming techniques may be employed for implementing both the CSE and DSE depending on the structure of the global graph and the technique used to partition this graph into areas. Three cases have particular practical interest.

1. The global graph is large and is partitioned into a small number of large areas. Here, sparsity programming can be employed for the CSE and at the DSE's lower level but not at the DSE's upper level.
2. The global graph is very large and is partitioned into a large number of large areas. Here, sparsity programming techniques can be employed for the CSE and at both the DSE's upper and lower levels.
3. The global graph is dense and is partitioned into dense areas. Here, programming techniques for dense matrices are employed at both the CSE and DSE's upper and lower levels.

The employment of sparsity programming affects the performance of both the CSE and DSE. The storage required for a sparse matrix is usually proportional to the number of non-zero elements in the matrix. The execution time of computations, such as matrix multiplication or those required for solving a system of equations, depends on the number as well as the position of the non-zero elements in the participating matrices. For example, the result of a matrix multiplication between two sparse matrices may be a dense matrix as a result of the relative position of non-zero elements in the two matrices. In the general case, the execution time of computations, such as matrix multiplication or those required for solving

This is a pre-final version. The final paper is available at: Efthimios Tambouris and Peter Van Santen (2000) "Scalability analysis of parallel systems with multiple components of work", Proc. of 8th Euromicro workshop on Parallel and Distributed Processing, IEEE Computer Society, pp. 182-189.

a system of equations, is no slower (meaning linear or faster) to the number of non-zero elements in the participating matrices. It is therefore clear that the input data significantly affect the performance of sparsity programming methods.

5.4 Performance models

For simplicity, it is assumed that all areas have the same number of internal nodes n_r . Hence, it is:

$$n = n_b + (p-1)\, n_r \quad (6)$$

The amount of work at the CSE is represented by W, the amount of work at the master by W_b and the amount of work at each slave by W_r. The total work W is a function of the number of nodes in the total graph n, the work W_b is a function of the number of boundary nodes n_b and the work at each slave W_r is a function of the number of internal nodes n_r and also a function of n_b. The exact relationship between work and input size depends on the sparsity of the input data. As all work involved various matrix operations it follows that work is in the best case $\Theta(n)$ for matrixes $n \times n$ that are sparse and $\Theta(n^3)$ for matrices that are dense [4].

The overheads in the DSE are due to load imbalance, communication and computation that is not performed by the CSE.

The overhead due to load imbalance consists of the overhead due to the slaves being idle while the master is performing computation, which is $(p\text{-}1)\ W_b$, and the overhead due to the master being idle while the slaves are performing computation, which is W_r . Hence:

$$T_o^{load\_im} = \Theta(p\ W_b) + \Theta(W_r)$$

The communication overhead is due to the two single-node broadcasts (steps 2* and 4* in figure 4) and the two global combine operations (steps 1* and 2a and 3* and 4). The performance of the global combine operation in steps 1* and 2a dominates all others. In this operations, matrix G is transferred thus the size of the message is $\Theta(n_b^2)$. If $\Theta(n_b)$ packets are formed where each has size $\Theta(n_b)$ then by using the recursive splitting algorithm [4] the communication overhead per processor is $\Theta(n_b^2) + \Theta(n_b \log p)$. Summing up over all processors, the total communication overhead is:

$$T_o^{comm} = \Theta(n_b^2\, p) + \Theta(n_b\, p \log p)$$

Extra computation in the DSE that is not performed in the CSE occurs in each slave. This is due to the inversion of matrix Ω (step 1 in figure 4) which is not required in the CSE and is $\Theta(W_r)$. Extra computation can also sometimes occur in the master. This is when programming techniques for sparse matrices are employed at the CSE but not at the upper level of the DSE (due to the fact that H_b may not be sparse enough). This contributes $\Theta(W_b)$ to the overhead function which however is slower than the $\Theta(p\ W_b\ )$ in $T_o^{load\_im}$ and thus is omitted. Hence, the overhead due to extra computation is:

$$T_o^{e\_work} = \Theta(p\ W_r)$$

Summing up all overheads and keeping dominant terms only, the total overhead is:

$$T_o = \Theta(p\ W_b) + \Theta(p\ W_r) + \Theta(n_b^2\, p) + \Theta(n_b\, p \log p) \quad (7)$$

5.5 Generic fixed-value efficiency analysis

For generic fve analysis the expression $W = K\ T_o$ is examined. Since T_o contains multiple terms, W is balanced against each individual term (section 2). The following set of expressions is derived:

$$W = \Theta(p\ W_b) \quad (8a)$$

$$W = \Theta(p\ W_r) \quad (8b)$$

$$W = \Theta(n_b^2\, p) \quad (8c)$$

$$W = \Theta(n_b\, p \log p) \quad (8d)$$

In the DSE, the total work W, the work at the master W_b and the work at each slave W_r are not independent. Instead, they are related by equation (6) giving the relationship between n , n_b and n_r , i.e.

$$n = \Theta(n_b) + \Theta(p\ n_r) \quad (8e)$$

Three cases are now examined based on the expected sparsity of the input data as suggested in section 5.3.

Theorem 1 presents the results from fve analysis when sparsity programming is employed at CSE and both the upper and lower level of the DSE. Since all matrices are sparse, the amount of work at the CSE is $W = \Theta(n)$, the amount of work at the DSE's upper level is $W_b = \Theta(n_b)$ and the amount of work at the DSE's lower level is $W_r = \Theta(n_r) + \Theta(n_b)$.

Theorem 1. The DSE is scalable when $n_r\ (p)$ is no slower than $n_b^2\ (p)$ for $n_b\ (p) = \Omega(\log p)$ and when $n_r\ (p)$ is no slower than $n_b \log p$ for $n_b\ (p) = o(\log p)$. The isoefficiency function is $\Theta(p\ n_r)$ which is $\Theta(p\ n_b^2)$ for $n_b(p) = \Omega(\log p)$ and $\Theta(n_b\, p \log p)$ for $n_b\ (p) = o(\log p)$.

Proof: The proof is given in Appendix B.

Theorem 2 presents the results from fve analysis when sparsity programming is employed in the CSE and DSE's lower level but not at the DSE's upper level (the proof is similar to the proof of theorem 1 and is omitted). The amount of work at the CSE is $W = \Theta(n)$. Matrix H_b is dense thus the amount of work at the DSE's upper level is $W_b = \Theta(n_b^3)$. Finally, matrix H_r is sparse thus the amount of work at the DSE's lower level is $W_r = \Theta(n_r) + \Theta(n_b)$.

This is a pre-final version. The final paper is available at: Efthimios Tambouris and Peter Van Santen (2000) "Scalability analysis of parallel systems with multiple components of work", Proc. of 8th Euromicro workshop on Parallel and Distributed Processing, IEEE Computer Society, pp. 182-189.

Theorem 2. The DSE is scalable when n_r (p) is no slower than $n_b{}^3$ (p) for n_b $(p) = \Omega(\log^{1/2} p)$ and when $n_r(p)$ is no slower than n_b log p for n_b $(p) = o(\log^{1/2} p)$. The isoefficiency function is $\Theta(p\ n_r)$ which is $\Theta(p\ n_b{}^3)$ for $n_b(p) = \Omega(\log^{1/2} p)$ and $\Theta(n_b\ p \log p)$ for n_b $(p) = o(\log^{1/2} p)$.

Theorem 3 presents the results from fve analysis when programming techniques for dense matrices are employed in both the CSE and DSE (the proof is similar to the proof of theorem 1 and is omitted). Since all matrices are dense, the amount of work at the CSE is $W = \Theta(n^3)$, the amount of work at the DSE's upper level is $W_b = \Theta(n_b{}^3)$ and the amount of work at the DSE's lower level is $W_r = \Theta(n_r{}^3) + \Theta(n_b{}^3)$.

Theorem 3. The DSE is scalable when n_r (p) is no slower than n_b (p) / $p^{2/3}$ for n_b $(p) = \Omega(\log^{1/2} p)$ and when n_r (p) is no slower than $(n_b \log p / p^2)^{1/3}$ for n_b $(p) = o(\log^{1/2} p)$. The isoefficiency function is $\Theta(p^3\ n_r{}^3\ (p))$ which is $\Theta(p\ n_b{}^3\ (p))$ for n_b $(p) = \Omega(\log^{1/2} p)$ and $\Theta(n_b\ p \log p)$ for $n_b(p) = o(\log^{1/2} p)$.

It can be reasonably assumed that the number of boundary nodes is linear to the number of areas (as each area contains on average the same number of boundary nodes). Since the number of areas is linear to p it follows that n_b $(p) = \Theta(p)$. In this case, for preserving efficiency, the number of internal nodes has to increase as $\Theta(p^2)$ (theorem 1), as $\Theta(p^4)$ (theorem 2) and as $\Theta(p^{1/3})$ (theorem 3) depending on the sparsity of the input data. These isoefficiency functions provide some interesting results on the scalability of the DSE and its relationship with the sparsity of the input data. It is first observed that none of the three cases examined is unscalable although the overheads incurred are large. It is also observed that the best scalability is achieved when all data are dense. In this case, the number of internal nodes has to increase at $\Theta(p^{1/3})$ to preserve the parallel system's efficiency. For example, it is assumed that a parallel system with $p = 4$ and $n_r = 20$ has a desirable efficiency. When p is scaled to 8 then this efficiency is preserved only if n_r is increased to 25. The scalability of the case where all data are sparse is not so good. For the same example, when p scales from 4 to 8 then n_r has to increase from 20 to 80 for the system's efficiency to be preserved. Finally, the scalability of the case where data are sparse at the CSE and DSE's lower level but not at the DSE's upper level is very poor. In the same example, n_r has to increase from 20 to 320 for the system's efficiency to be preserved.

5.6 An implementation of the DSE

An implementation of the CSE and DSE [6] was provided by the Brunel Institute of Power Systems for evaluation purposes. This program, initially written for a shared-memory parallel computer, was ported to the Parsytec GCel 3/512, a distributed-memory parallel computer with a mesh interconnection topology. More specifically, the parts of the program related to computations (initially written in FORTRAN) were ported with minor only changes. On the other hand, the parts to accomplish inter-processor communication were initially written using the PVM model, which has poor performance for the target architecture. To improve performance, the recursive-splitting algorithm was implemented using the more efficient message-passing model.

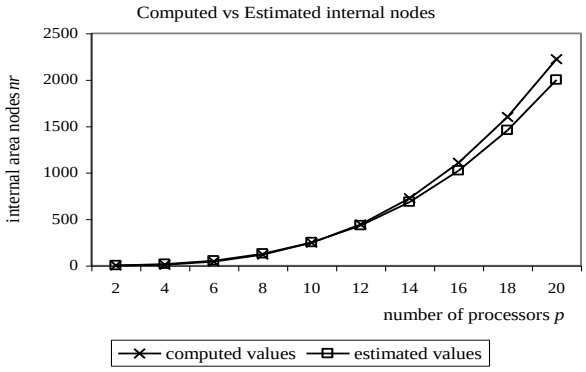

Fig. 3 Computed vs. estimated values of n_r

Timing measurements were obtained from solving the IEEE-30 node standard network and performance models were constructed for all sources of overhead. These models allowed computing an analytical expression for the number of internal nodes n_r that is required to preserve efficiency as p is scaled. This expression was thereafter projected to compute the values of n_r for large input sizes and number of processors. These computed values are presented in figure 3 together with values estimated from theorem 2. The convergence of these values illustrates the analytical power of the generic fixed-value efficiency method.

6. Conclusions

In this paper, the generic fixed-value efficiency (fve) method was proposed to evaluate the scalability of parallel systems with multiple components. The generic fve method is an extension of isoefficiency that is suitable for parallel systems where the growth rates of total work and its components are not predetermined. The method

This is a pre-final version. The final paper is available at: Efthimios Tambouris and Peter Van Santen (2000) "Scalability analysis of parallel systems with multiple components of work", Proc. of 8th Euromicro workshop on Parallel and Distributed Processing, IEEE Computer Society, pp. 182-189.

can be utilised to reveal the relation between the total work and its components for the parallel system to preserve efficiency. This method was demonstrated for a decomposed-state estimator for power systems on a distributed-memory parallel computer with a mesh interconnection topology. The results allowed the impact on scalability of the input data sparsity to be analytically estimates. These results were verified through timing measurements.

7. Acknowledgements

The authors would like to acknowledge the Brunel Institute for Power Systems (BIPS) for providing the source code of the CSE and DSE programs. Also Prof. Halatsis at the University of Athens for granting access to the Parsytec GCel 3/512 parallel computer. Finally, Dr. Marsh for his comments on the CSE and DSE algorithms.

8. References

[1] Gupta A. *Analysis and Design of Scalable Parallel Algorithms for Scientific Computing*. Ph.D. Thesis, Department of Computer Science, University of Minnesota, Minneapolis, MN. 1995

[2] Gustafson J. L. "The Consequences of Fixed Time Performance Measurement". *Proceedings of the 25th Annual Hawaii International Conference on System Sciences*, vol. 2, 1992, pp. 113-124.

[3] Kumar V., Gupta A. "Analyzing scalability of parallel algorithms on parallel architectures". *Journal of Parallel and Distributed Computing*, **22**, 1994, pp. 379-391.

[4] Kumar V., Grama A., Gupta A., Karypis G. *Introduction to Parallel Computing. Design and Analysis of Algorithms*, USA, Benjamin/Cummings, 1994.

[5] Martin I., Tirado F. "Relationships between efficiency and execution time of full multigrid methods on parallel computers". *IEEE Transactions on Parallel and Distributed Systems*, **8** (6), 1997, pp. 562-573.

[6] Marsh J. F., Zitouni S., Irving M. R. "Computational Aspects of Distributed State-Estimators for Power Systems". *12th IFAC World Congress*, 1993.

[7] Marsh J. F. "Structure of measurement Jacobian matrices for power systems". *IEE Proc., Pt. C.*, **6**, 1989, pp. 407-413.

[8] Marsh J. F, Azzam M. "MCHSE-a versalite framework for the design of 2-level power system state-estimators". *IEE Proc., Pt. C.*, **4**, 1988, pp. 291-298.

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

[10] Sun X-H., Rover D.T. "Scalability of Parallel Algorithm-Machine Combinations". *IEEE Transactions on Parallel and Distributed Systems*, **5** (6), 1994, pp. 599-613.

[11] Sun X-H. "The Relation of Scalability and Execution Time". *International Processing Symposium*, 1996. Also as NASA ICASE Report 95-62, Sep. 1995.

[12] Tambouris E., van Santen P., Marsh J. "Performance Evaluation of a Processor-Farm Model for Power Systems State-Estimators on a Distributed-Memory Computer". *Computing Systems in Engineering*, **5** (4-6), 1994, pp. 479-487.

[13] Tambouris E., van Santen P. "A Methodology for Performance and Scalability Analysis". *Lecture Notes in Computers Science*, vol. 1012, 1995, pp. 475-480.

[14] Tambouris E., Dimas E. "Scalability analysis of multiprocessor systems using stochastic models", *International Conference on Parallel and Distributed Processing Techniques and Applications*, vol. III, 1997, pp. 1448-1452.

Appendix A: Order Analysis Notation

Order analysis is used to classify functions by their growth rates. All functions analysed are from positive integers to positive reals. The order notation used in this paper is summarised in table 1.

Table 1 Order analysis notation

notation	description
$f = o(g)$	f is *slower* than g
$f = O(g)$	f is *no faster* than g
$f = \Theta(g)$	f is *about as fast* as g
$f = \Omega(g)$	f is *no slower* than g
$f = \omega(g)$	f is *faster* than g

The formal definitions of order notation follows:

- $o(g)$ is the set of f such that f/g approaches 0; i.e. $\lim_{n\to\infty}\{f(n)/g(n)\} = 0$.
- $O(g)$ is the set of f such that f/g is bounded above; that is, there exists C such that for all positive n, $f(n) < C\,g(n)$.
- $\Theta(g)$ is the set of f such that f/g is bounded above and bounded away from zero; that is, there exist C_1, C_2 such that for all n, $0 < C_1\,g(n) < f(n) < C_2\,g(n)$.
- $\Omega(g)$ is the set of f such that f/g is bounded away from 0; that is, there exists C such that for all n, $0 < C\,g(n) < f(n)$.
- $\omega(g)$ is the set of f such that f/g approaches infinity; i.e. $\lim_{n\to\infty}\{f(n)/g(n)\} = \infty$.

Appendix B: Proof of theorem 1

Since all matrices are sparse, the amount of work at the CSE is $W = \Theta(n)$, the amount of work at the DSE's upper level is $W_b = \Theta(n_b)$ and the amount of work at the DSE's lower level is $W_r = \Theta(n_r) + \Theta(n_b)$. Now equations (8) become:

$$W = \Theta(p\,n_b) \quad (9a)$$

$$W = \Theta(p\,n_r) + \Theta(p\,n_b) \quad (9b)$$

$$W = \Theta(n_b^2\,p) \quad (9c)$$

$$W = \Theta(n_b\,p\log p) \quad (9d)$$

This is a pre-final version. The final paper is available at: Efthimios Tambouris and Peter Van Santen (2000) "Scalability analysis of parallel systems with multiple components of work", Proc. of 8th Euromicro workshop on Parallel and Distributed Processing, IEEE Computer Society, pp. 182-189.

$W = \Theta(n_b) + \Theta(p\ n_r)$ (9e)

Keeping dominant terms only these equations are reduced to the following set:

$W = \Theta(p\ n_r)$ (10a)

$W = \Theta(p\ n_b^2)$ (10b)

$W = \Theta(n_b\ p \log p)$ (10c)

$W = \Theta(n_b) + \Theta(p\ n_r)$ (10d)

The isoefficiency functions derived from equations (10a-c) are $f_1(p) = \Theta(p\ n_r)$, $f_2(p) = \Theta(p\ n_b^2)$ and $f_3(p) = \Theta(n_b\ p \log p)$ respectively. The growth rate of W, which is represented by $f_W(p)$, is conveyed from equation (10d), hence depends on the relative growth of n_b and $p\ n_r$. Therefore, two cases are examined.

(i) $p\ n_r$ is slower than n_b i.e. $n_r = o(n_b / p)$

In this case, the term n_b dominates the r.h.s. of equation (10d) and the growth rate of W is:

$f_W(p) = \Theta(n_b)$

We observe that one of the isoefficiency functions, namely $f_2(p) = \Theta(p\ n_b^2)$, is faster than $f_W(p)$ for all possible forms of $f_b(p)$. Therefore, the parallel system is fve unscalable.

(ii) $p\ n_r$ is no slower than n_b i.e. $n_r = \Omega(n_b / p)$

In this case, the term $p\ n_r$ dominates the r.h.s. of equation (10d) and the growth rate of W is:

$f_W(p) = \Theta(p\ n_r)$

The condition for the parallel system to be scalable is that $f_W(p)$ grows no slower than all isoefficiency functions. One of the isoefficiency functions, namely $f_1(p) = \Theta(p\ n_r)$, is always no slower than $f_W(p)$. Hence, the condition for the parallel system to be scalable is:

$f_W(p)$ is no slower than $\max\{f_2(p), f_3(p)\} \Rightarrow$

$p\ n_r = \Omega(\max\{p\ n_b^2, n\ p \log p\}) \Rightarrow$

$n_r = \Omega(\max\{n_b^2, n_b \log p\})$

Now there are two different cases depending on which isoefficiency function is faster:

(a) n_b^2 is no slower than $n_b \log p$, i.e. $n_b = \Omega(\log p)$

Hence, $n_r = \Omega(n_b)$ and the isoefficiency function is $\Theta(p\ n_b)$

(a) n_b^2 is no slower than $n_b \log p$, i.e. $n_b = o(\log p)$

Hence, $n_r = \Omega(log\ p)$ and the isoefficiency function is $\Theta(n_b\ p \log p)$. •

Appendix C: The Decomposed State Estimator

	Program DSE_upper_level	**Program DSE_lower_level**
	/* executed in one processor (the master)	/* executed in $(p-1)$ parallel processors (the slaves) /* here shown for processor i
	Begin	Begin
0		Initialise $x_{ir}(0)$, $x_b(0)$. Read z_i. Set $j = 0$.
1		Compute: $h_i(x_{ir}(j), x_b(j))$; $H_{ir}(j)$; $H_{ib}(j)$; $\Delta z_i(j) = z_i - h_i$; $\Omega_{ir}(j) = H_{ir}^T R_i^{-1} H_{ir}$; $W_{ir}(j) = I - H_{ir}\Omega_i^{-1} H_{ir}^T R_i^{-1}$; $G_i = [W_{ir}\ H_{ib}]^T R_i^{-1} [W_{ir}\ H_{ib}]$; $g_i = [W_{ir}\ H_{ib}]^T R_i^{-1} \Delta z_i$.
1*	Receive $[G_i(j): g_i(j)]$ from lower level.	Send $[G_i(j): g_i(j)]$ to upper level
2a	$[G : g] = \sum [G_i : g_i]$ and $[G_i(j): g_i(j)]$.	
2b	Compute: $\Delta x_b(j)$ from $G(j)\ \Delta x_b(j) = g(j)$	
2*	Send $\Delta x_b(j)$ to lower level	Receive $\Delta x_b(j)$ from upper level
3		Compute: $\Delta r_{ib}(j) = \Delta z_i - H_{ib}(j)\ \Delta x_b(j)$ and $\Delta x_{ir}(j)$ from $\Omega_{ir}(j)\ \Delta x_{ir}(j) = H_i^T(j)\ R_i^{-1}\ \Delta r_{ib}(j)$. Set $j = j + 1$. Compute $x_{ir}(j)$; $x_b(j)$; If $max\ \|\Delta x_{ir}(j)\| < A_o$ then slave_conv_flag:=TRUE
3*	Receive slave_conv_flag from lower level	Send slave_conv_flag to upper level
4	Compute: global convergence flag	
4*	Send global convergence flag to lower level	Receive global flag from upper level.
5	If global convergence not reached, then go to 1* else stop	If global convergence not reached, then go to 1 else $x_{ir}(j) = \mathbf{x}_{ir}$ and $x_b(j) = \mathbf{x}_b$
	End	End

Fig. 4 The decomposed state estimator (DSE)