Journal of Computer Science 6 (1): 75-79, 2010
ISSN 1549-3636

Parallel Calculation Sensitivity Function for Multi Tasking Environments

Hamed Al Rjoub and Ahmed Al-Sha'or
Department of Computer Science, Umm Alqura University, Kingdom of Saudi Arabia

Abstract: Problem statement: Calculating sensitive functions for a large dimension control system to find the unknowns vectors for a linear system in both single and multi processors, is not considered internally compatible with multi tasking environments, so breaking the process can cost time and memory and it couldn't be paused, resumed and saved as patterns for later continuity. This study is an attempt to solve this problem in parallel to reduce the time factor needed and increase the efficiency by using parallel calculation sensitivity function for multi tasking environments (PSME) algorithm. **Approach:** calculate in parallel sensitivity function using n-1 processors where n is a number of linear equations which can be represented as TX = W, where T is a matrix of size $n_1 \times n_2$, $X = T^{-1}W$, is a vector of unknowns and $\partial X/\partial h = T^{-1}((\partial T/\partial h)-(\partial W/\partial h))$ is a sensitivity function with respect to variation of system components h. The algorithm (PSME) divides the mathematical input model into two partitions and uses only (n-1) processors to find the vector of unknowns for original system $x = (x_1,x_2,\ldots,x_n)^T$ and in parallel using (n-1) processors to find the vector of unknowns for similar system $(x')^t = d^tT^{-1} = (x_1',x_2',\ldots x_n')^T$ by using Net-Processors, where d is a constant vector. Finally, sensitivity function (with respect to variation of component $\partial X/\partial h_i = (x_i \times x_i')$ can be calculated in parallel by multiplication unknowns $X_i \times X_i'$, where i = 0,1,…n-1. **Results:** The running time t is reduced to O(t/n-1) and, the performance of (PSME) was increased by 30-40%. **Conclusion:** Hence, used (PSME) algorithm reduced the time to calculate sensitivity function for a large dimension control system and the performance was increased.

Key words: Sensitivity function, parallel, linear equations, variation, running time, mathematical model

INTRODUCTION

Mathematical models of sensitive system that has a matrix of unknowns exists in all practical knowledge such as biology, physics, geology and all other applied life areas and it is possible to simulate those models for pre use expectation systems to examine those systems before applying it on real world solutions, specially when dangerous human life matters can happen on fractional mistakes when wrong model applied, like airplane mechanical and electrical systems and space shuttles systems calculations and another application when we need computer with mathematical decisions for next moves on any practical system, beside those sensitivity systems parallelism approach must considered on proper with needed speed.

The use of iterative methods has increased substantially in many application areas in the last years (Duff and van de Vorst, 1999). One reason for that is the advent of parallel Computing and its impact in the overall performance of various algorithms on numerical analysis (Duff and van de Vorst, 1999). The use of clusters plays an important role in such scenario as one of the most effective manner to improve the computational power without increasing costs to prohibitive values. However, in some cases, the solution of numerical problems frequently presents accuracy issues increasing the need for computational power. Verified computing provides an interval result that surely contains the correct result. Numerical applications providing automatic result verification may be useful in many fields like simulation and modeling. Finding the verified result often increases dramatically the execution time (Ogita *et al*., 2005). However, in some numerical problems, the accuracy is mandatory. The requirements for achieving this goal are: Interval arithmetic, high accuracy combined with well suitable algorithms. The interval arithmetic defines the operations for interval numbers, such that the result is a new interval that contains the set of all possible solutions. The high accuracy arithmetic ensures that the operation is performed without rounding errors and

Corresponding Author: Hamed Al Rjoub, Department of Computer Science, Umm Alqura University, Kingdom of Saudi Arabia

rounded only once in the end of the computation. The requirements for this arithmetic are: The four basic operations with high accuracy, optimal scalar product and direct rounding. This arithmetic's should be used in appropriate algorithms to ensure that those properties will be hold. There is a multitude of tools that provide verified computing; among them an attractive option is C-XSC (C for extended Scientific Computing) (Li and Coleman, 1989). CXSC is a free and portable programming environment for C and C++ programming Languages, offering high accuracy and automatic verified results. This programming Tool allows the solution of several standard problems, including many reliable numerical parallel algorithms. The need to solve systems of linear algebraic equations arises frequently in scientific and engineering applications, with the solution being useful either by itself or as an intermediate step in solving a larger problem. In practical problems, the order, n, may in many cases be large (100-1000) or very large (many tens or hundreds of thousands). The cost of a numerical procedure is clearly an important consideration-so too is the accuracy of the method. Let us consider a system of linear algebraic equations:

$$AX = B \quad (1)$$

Where:
$A = \{aij\}^{n}_{i,j=1}$ is a given matrix
$B = (b1, \ldots, bn)^{t}$ is a given vector

It is well known (Duff, 2000) that the solution, x, $x \epsilon R^n$, when it exists, can be found using-direct methods, such as Gaussian elimination and LU and Cholesky decomposition, taking $O(n^3)$ time; -stationary iterative methods, such as the Jacobi, Gauss- Seidel and various relaxation techniques, which reduce the system to the form:

$$x = Lx+f \quad (2)$$

and then apply iterations as follows:

$$x^{(0)} = f, x^{(k)} = Lx^{(k-1)} + f, k = 1, 2 \quad (3)$$

until desired accuracy is achieved; this takes $O(n^2)$ time per iteration. Monte Carlo methods (MC) use independent random walks to give an Approximation to the truncated sum (3):

$$x^{(1)} = \sum_{k=0}^{1} L^k f \quad (4)$$

Taking time O(n) (to find n components of the solution) per random step. Keeping in mind that the convergence rate of MC is $O(N^{-1/2})$, where N is the number of random walks, millions of random steps are typically needed to achieve acceptable accuracy. The description of the MC method used for linear systems can be found in (Pan and Reif, 1989; Eisentat and Heath, 1988; Holbig and Morandi, 2004). Different improvements have been proposed, for example, including sequential MC techniques (Duff and van de Vorst, 1999), resolve-based MC methods (Duff and van de Vorst, 1999) and have been successfully implemented to reduce the number of random steps. In this study we study the Quasi-Monte Carlo (QMC) approach to solve linear systems with an emphasis on the parallel implementation of the corresponding algorithm. The use of quasirandom sequences improves the accuracy of the method and preserves its traditionally good parallel efficiency.

MATERIALS AND METHODS

Solution of large (dense or sparse) linear systems is considered an important Part of numerical analysis and often requires a large amount of scientific computations (Duff and van de Vorst, 1999; Saad, 1996). More specifically, the most time consuming operations in iterative methods for solving linear equations are inner products, vector successively updates, matrix-vector products and also iterative refinements (Eisentat and Heath, 1988). Tests pointed out that the Newton-like iterative method presents a iterative refinement step and uses a inverse matrix obtained through the backward/forward substitution (after LU decomposition), which are the most time consuming operations. The parallel solutions for linear solvers found in the literature explore many aspects and constraints related to the adaptation of the numerical methods to high performance environments (Li and Coleman, 1989). However, the proposed solutions are not often realistic and mostly deal with unsuitable models for high performance environments of distributed memory as clusters of workstations. In many theoretical models (such as the PRAM family) the transmission cost to data exchange is not considered (Ogita *et al*., 2005), but in distributed memory architectures this issue is crucial to gain performance. Nevertheless, the difficulty in parallelizing some numerical methods, mainly iterative schemes, in an environment of distributed memory, is the interdependency among data (e.g., the LU decomposition) and the consequent overhead needed to perform Inter Process Communication (IPC) (Li and Coleman, 1989). Due to this, in a first approach some modifications were done in the backward/forward

substitution procedure (Liu and Cheung, 1997) to allow less Communications and independent computations over the matrix. Another possible optimization when implementing for such parallel environments is to reduce communication cost through the use of load balance techniques, as we can see in some recent parallel solutions for linear systems solvers (Saad, 1996). Anyway, their focus was toward the issues related to MPI implementation through a theoretical performance analysis. Few researches were found related to numerical analysis of parallel implementations of iterative solvers, mainly using MPI. Moreover, some interesting papers found present algorithm, which allow the use of different parallel environments (Eisentat and Heath, 1988). However, those papers (like others) do not deal with verified computation. We also found some works which focus on verified computing (Liu and Cheung, 1997) and both verified computing and parallel implementations (Feng, 2002), but these thesis implement other numerical problems or use a different Parallel approach. Another concern is the implementation of self-verified numerical solvers, which allow high accuracy operations. The researches already made, show that the execution time of the algorithms using this kind of routines is much larger than the execution time of the algorithms, which do not use it (Saad, 1996; Feng, 2002). The C-XSC library was developed to provide functionality and portability, but early researches indicate that more optimizations may be done to provide more efficiency, due to additional computational cost in sequential and consequently for other environments as Itanium clusters. Some experiments were conducted over Intel clusters to parallelize self-verified numerical solvers that use Newton-based techniques but there are more tests that may be done.

Sensitivity analysis defines the relative sensitivity function for time independent parameters as:

$$S_{i,j} = \partial x_i / \partial h_i \quad (5)$$

Where:

X_i = The i-th state variable

h_j = The element of the parameter vector

Hence the sensitivity is given by the so-called sensitivity matrix S, containing the sensitivity coefficient $S_{i,j}$, Eq. 5 The direct approach of numerically differentiating by means of numerical field calculation software will lead to diverse difficulties (Duff and van de Vorst, 1999; Li and Coleman, 1989). Therefore, some ideas to overcome those problems aim at performing differentiations necessary for sensitivity analysis prior to any numerical treatment. Further calculations are then carried out with a commercially available field calculation program. Such approach has already been practical successfully (Eisentat and Heath, 1988).

We consider the linear system (1) where A is a tridiagonal matrix of order n of the form shown in (6):

Step 1: Build first diagonal Matrix V size(m×n) for the input matrix A(nxm)

Step 2: Build C1 vector of the unknowns

Step 3: Calculate $C1_{(1xm)}$ vector of unknowns where:

$$C1_{(1)} = (A_{(n,1)} * V1_{(2,m)}) / (A_{(n,1)} * V1_{(1,m)})$$

.

.

.

.

$$C_{(m)} = (A_{(n,m)} * V1_{(n,m)}) / (A_{(n,m)} * V1_{(n-1,m)})$$

Step 4: Build and calculate vector V2 where:

$$V2_{(2,m)} = V1_{(2,m)} - C1_{(2)} * V1_{(1,m)}$$

.

.

.

.

Step log m: Build and calculate matrix Vx(2×2) where:

$$Vx_{(2,m)} = V(x-1)_{(2,m)} - Cx_{(2)} * Vx_{(1,m)}$$

*when we reach to final V we will find the solution of the unknowns already solved in the last v matrix.

Numerical example: The linear equation:

$$2x_1 + 4x_2 = 4$$
$$2x_1 + 2x_2 = 5$$

Could be represents as follow:

$$A = A = \begin{bmatrix} 2 & 4 & -4 \\ 2 & 2 & -5 \end{bmatrix}$$

We build the diagonal working area for the previous matrix A:

$$V = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Where:

$$v1^0 = \begin{bmatrix}1\\0\\0\end{bmatrix}$$

$$v2^0 = \begin{bmatrix}0\\1\\0\end{bmatrix}$$

$$V3^0 = \begin{bmatrix}0\\0\\1\end{bmatrix}$$

In the first cycle and in parallel we calculate $C2^1$, $C3^1$ values:

$C2^1 = A1.V2^0/A^1.V1^0$
$C3^1 = A1.V3^0/A1.V1^0$

Where A1 is the first row of matrix A:

$$C2^1 = \begin{bmatrix}2 & 4 & -4\end{bmatrix}\cdot\begin{bmatrix}0\\1\\0\end{bmatrix} \div \begin{bmatrix}2 & 4 & -4\end{bmatrix}\cdot\begin{bmatrix}1\\0\\0\end{bmatrix} = 2$$

$$C3^1 = \begin{bmatrix}2 & 4 & -4\end{bmatrix}\cdot\begin{bmatrix}0\\0\\1\end{bmatrix} \div \begin{bmatrix}2 & 4 & -4\end{bmatrix}\cdot\begin{bmatrix}1\\0\\0\end{bmatrix} = -2$$

Now we calculate in parallel:

$$v2^1 = v2^0 - c2^1.\, v1^0 = \begin{bmatrix}0\\1\\0\end{bmatrix} - \begin{bmatrix}2\\0\\0\end{bmatrix} = \begin{bmatrix}-2\\1\\0\end{bmatrix}$$

$$v3^1 = v3^0 - c3^1.\, v1^0 = \begin{bmatrix}0\\0\\1\end{bmatrix} - \begin{bmatrix}-2\\0\\0\end{bmatrix} = \begin{bmatrix}2\\0\\1\end{bmatrix}$$

Calculate:

$$c3^2 = A2\,.\,V3^1 \;/\; A2\,.\,V2^1 = \begin{bmatrix}2 & 2 & -5\end{bmatrix}\cdot\begin{bmatrix}2\\0\\1\end{bmatrix} \div \begin{bmatrix}2 & 2 & -5\end{bmatrix}\cdot\begin{bmatrix}-2\\1\\0\end{bmatrix} = 1/2$$

Now we find the solution (X1, X2):

$$V3^2 = V3^1 - C3^2\,.v2^1 = \begin{bmatrix}2\\0\\1\end{bmatrix} - \begin{bmatrix}-1\\1/2\\0\end{bmatrix} = \begin{bmatrix}3\\-1/2\\1\end{bmatrix} = \begin{bmatrix}x1\\x2\\1\end{bmatrix}$$

RESULTS

To calculate the accurate time and performance we repeat the process m times then we divide the measured time on m for both single and multi thread versions, for single thread we start basic multiplication division and subtraction inside the Matrix until we get the upper of that matrix, for multi threading we use R-1 threads where R is the count of desired Matrix rows, we measure the longest thread which is the last one in our case, then every thread takes a part of the Matrix basic operations and we do that in parallel for origin and similar systems.

Table 1 shows the time results done on Pentium Due 1.8 GHZ processor with 1 GB Ram and shows the time when we use one processor (single thread) and the time when we use a multi processors in parallel (multi thread) to calculate the unknowns vector. From the Table 1, Fig. 1 and 2, we can see that performance increase with respect to the size of matrix, which represents the linear system.

Table 1: Comparison between single and multithread

Matrix dimension	Single thread, time/M_S	Multi thread, time/M_S	Performance
2×3	0.000119	0.000347	0.34293948
3×4	0.000425	0.000314	1.35350318
4×5	0.001073	0.000234	4.58547009
5×6	0.001718	0.000308	5.57792208

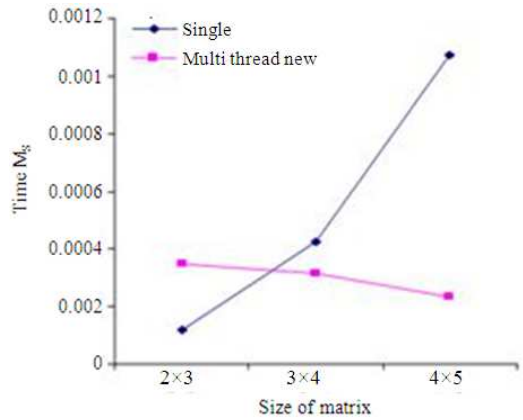

Fig. 1: Time comparison between single and parallel to calculate unknown's vector

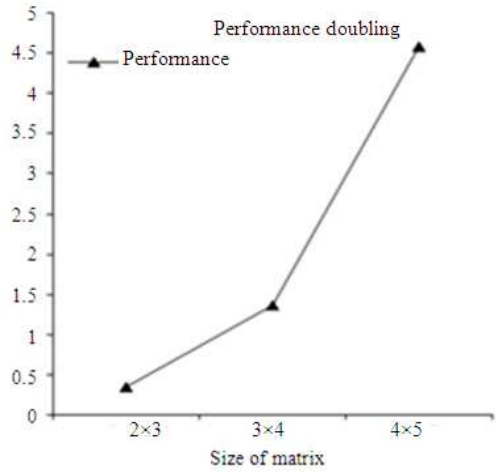

Fig. 2: System performance chart

DISCUSSION

The application of high performance programming techniques for solution of electric power systems problems has been increasing. Particularly, parallel processing present's very remising perspectives when heavy amputation is required. It may consist in a feasible alternative for solution of several large-scale problems, which are not well conditioned for a sequential approach. Despite its potentiality in engineering software development, parallel algorithm philosophy is quite different from that adopted by sequential programs. This work presents investigations regarding the application of parallel processing to calculate sensitivity function for a large dimension control system, which we can write its mathematical model as a system of linear equations.

PSME algorithm description: The main goal of PSME algorithm is resolving in parallel linear equations which represents as AX = W and calculate sensitivity function of electric power systems to obtain the result with respect to variation any component of output function F with respect to any component of electric power systems $h(\partial f / \partial h)$. PSME algorithm contains the next stages: distribution data (rows matrix T and components vector W) to the p processors where p = n-1 (n is the number of equations) which represents the mathematical model of electric system and calculate in parallel unknown vector for origin system

CONCLUSION

The (PSME) algorithm to find the vector of unknowns for calculated in parallel sensitivity function and one thread was simulated and proved that (PSME) algorithm is more efficient. The running time was reduced to O(t/n-1) and the efficiency was increased by 40-55%.

REFERENCES

Duff, I.S. and H.A. van de Vorst, 1999. Developments and trends in parallel solution of linear systems. Parall. Comput., 25: 1931-1970. DOI: 10.1016/S0167-8191(99)00077-0

Duff, I.S., 2000. The impact of high performance computing in the solution of linear systems: Trend and problems. J. Comput. Applied Math., 123: 515-530. DOI: 10.1016/S0377-0427(00)00401-5

Eisentat, S.C. and M.T. Heath, 1988. Modified cyclic algorithm for solving triangular system on distributed-memory multiprocessor. SIAM J. Stat. Comput., 9: 589-600. http://scitation.aip.org/getabs/servlet/GetabsServlet?prog=normal&id=SJOCE3000009000003000589000001&idtype=cvips&gifs=yes

Feng, T., 2002. A message-passing distributed-memory Newton-GMRES parallel power flow algorithm. Proceeding of the IEEE Meeting on Power Engineering Society Summer, July 25-25, Chicago, IL., USA., pp: 1477-1482.

Holbig, C.A. and P.S. Morandi, 2004. Selfverifying solvers for linear systems of equations in C-XSC. Lecture Notes Comput. Sci., 3019: 292-297. DOI: 10.1007/b97218

Li, G. and T.F. Coleman, 1989. A new method for solving triangular system on distributed memory message-passing multiprocessors. SIAM J. Sci. Stat. Comput., 10: 382-396.

Liu, Z. and D.W. Cheung, 1997. Efficient parallel algorithm for dense matrix LU decomposition with pivoting on hypercubes. Comput. Math. Applied, 33: 39-50.

Ogita, T., S.M. Rump and S. Oishi, 2005. Accurate sum and dot product. SIAM. J. Sci. Comput., 26: 1955-1988. http://www.ti3.tu-harburg.de/paper/rump/OgRuOi05.pdf

Pan, V. and J. Reif, 1989. Fast and efficient parallel solution of dense linear system. Comput. Math. Applied, 17: 1481-1491.

Saad, Y., 1996. Iterative Methods for Sparse Linear Systems. Proceeding of the 99th ACM Symposium on FPGAs. ACM Press, New York, pp: 157-166.