Calculation in Parallel Sensitivity Function Using Vector Presentation Algorithm (VPA)

Hamed Al Rjoub
Umm Al-Qura University
Computer Science Department, Makkah-Saudi Arabia

Abstract

This paper presents a new algorithm to solve in parallel linear equations which represent a mathematical model for a large dimension control system and calculates in parallel sensitivity function using n-1 processors where n is a number of linear equation that can be represented as TX=W, where T is a matrix of size $n_r \times n_c$, $X=T^{-1}W$,is a vector of unknowns, and $\partial X/\partial h=-T^{-1}((\partial T/\partial h)X-(\partial W/\partial h))$ is a sensitivity function with respect to variation of system components h. The algorithm (VPA) divides the mathematical input model into two partitions and uses only (n-1) processors to find out 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 $(x^{|})^t=-d^tT^{-1}=(x^{|}_1,x^{|}_2,\ldots,x^{|}_n)^T$ where d is a constant vector .Finally, the sensitivity function (with respect to variation of any component $\partial X/\partial h_i = (x_i \times x^{|}_i)$ can be calculated in parallel by multiplication unknowns $x_i \times x^{|}_i$ respectively, where i=0,1,…n-1 .The running time t is reduced by O(t/2) and the efficiency of (VPA) is increased by 50-60% .

Key words: *Parallel processing, Vector Presentation, Sensitivity Function, Matrix, Variation, Running Time, Mathematical Model.*

1. Introduction

The ability to develop mathematical models in Biology, Physics, Geology and other applied areas has pulled and has been pushed by the advances in High Performance Computing. Moreover, the use of iterative methods has increased substantially in many application areas in the last years [9, 5]. One reason for that is the advent of parallel Computing and its impact in the overall performance of various algorithms on numerical analysis[1].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 [6]. 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 [2]. 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 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. These arithmetics should be used in appropriate algorithms to ensure that those properties will be held. There is a multitude of tools that provides verified computing, among them an attractive option is C-XSC (C for extended Scientific Computing) [3]. 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:

$$Ax = b, \ldots\ldots\ldots\ldots\ldots\ldots\ldots \ldots\ldots. \quad (1)$$

Where $A = \{aij\}^{n}_{i,j=1}$ is a given matrix, and $b = (b1, \ldots, bn)^{t}$ is a given vector. It is well known (see, for example, [4, 5]) that the solution, x, $x \in R^{n}$, 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:

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

and then apply iterations as follows

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

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)

$$x^{(l)} = \sum_{k=0}^{l} L^k f \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots(4)$$

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 [6], [7], [8]. Different improvements have been proposed, for example, including sequential MC techniques [5], resolve-based MC methods [1], etc., and have been successfully implemented to reduce the number of random steps. In this paper 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. The paper is organized as follows: gives the background - MC for linear systems and a brief description of the quasirandom sequences we use, describes parallel strategies, presents some numerical results and presents conclusions and ideas for parallel processing.

2. RELATED WORK

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 [9, 10]. 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 [11, 12]. 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 [3]. 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, 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) [3]. Due to this, in a first approach some modifications were done in the backward/ forward substitution procedure [7] 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 [8]. Anyway, their focus was toward the issues related to MPI implementation through a theoretical performance analysis. Few works 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 [9]. However, those papers (like others) do not deal with verified computation. We also found some works which focus on verified computing [5] and both verified computing and parallel implementations, but this 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 [11, 13]. 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 [2,14].

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

$$S_{i,j} = \partial X_i/\partial h_j \text{ ,……………………(5)}$$

Where X_i represents the i-th state variable, h_j is the element of the parameter vector. Hence the sensitivity is given by the so-called sensitivity matrix S, containing the sensitivity coefficient Si,j ,equation 5 .The direct approach of numerically differentiating by means of numerical field calculation software will lead to diverse difficulties [1,3]. 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 [7]. As it considered that the linear system (1) where A is a tridiagonal matrix of order n of the form shown in (6), $x=(x_0,x_1,\ldots,x_{n-1})^T$ is the vector of unknowns , and $d=(d_0,d_1,\ldots,d_{n-1})^T$ is a vector of dimension n.

$$A=\begin{pmatrix} b_0 & c_0 & & & \\ a_1 & b_1 & c_1 & & \\ & a_2 & b_2 & c_2 & \\ & & .. & .. & \\ & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & a_{n-1} & b_{n-1} \end{pmatrix} \text{ … (6)}$$

In the LU factorization A, is decomposed into a product of two bidiagonal matrices L and U as A=LU, where

$$L=\begin{bmatrix} 1 & & & & & \\ h_1 & 1 & & & & \\ & .. & 1 & & & \\ & & .. & .. & & \\ & & & h_{n-2} & 1 & \\ & & & & h_{n-1} & 1 \end{bmatrix}$$

$$U=\begin{bmatrix} u_0 & c_0 & & & & \\ & u_1 & c_1 & & & \\ & & . & . & & \\ & & & . & . & \\ & & & & u_{n-2} & c_{n-2} \\ & & & & & u_{n-1} \end{bmatrix}$$

The LU algorithm to solve the linear system (1) then proceeds to solve for y from Ly=d and then finds vector X in parallel:

Step 1. Compute the decomposition of A given by

$u_0 = b_0$,
$h_i=a_i /u_i-1$, $1<=i<=n-1$,
$u_i= b_i - h_i * c_{i-1}$, $1<=i<=n-1$,

Step 2. Solve for y from Ly =d using

$y_0 = d_0$,
$y_i = d_i - h_i * y_{i-1}$, $1<=i<=n-1$.

Step 3. Compute X by solving ux = y using

$x_{n-1} = y_{n-1} / u_{n-1}$,
$x_i = (y_i - c_i * x_{i+1})/u_i$, $0<=i<=n-2$.

First we consider the parallelization of the LU decomposition part of the LU algorithm to solve (1), i.e. Step 1 above. Once the diagonal entries $u_0,u_1,\ldots,u_n$ of U have been calculated, $h_1,h_2,\ldots,h_{n-1}$can subsequently be computed in a single parallel step with n-1 processors. Thus it concentrates on the computation of the u_i's

3. Parallel Algorithm to Calculate Sensitivity Function Using VP Algorithm

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.

3.1 VPA Description

The main goal of **VP** 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 ($\partial f/\partial h$) . VP algorithm contains the next stages: distribution data(rows matrix A and components vector W) to the p processors where p= n (n is the number of rows ,m is the number of columns) which represents the mathematical model of electric system, and calculate in parallel unknown vector for origin system $X=(x_1,x_2,\ldots,x_n)^T$. Distribution data (at the same time) to p processors, and calculate unknown vector for similar system $(x^{|})^t=-d^tT^{-1}=(x^{|}_1,x^{|}_2,\ldots,x^{|}_n)^T$. Multiplication operation for unknown $x_i \times x^{|}_i$ respectively using n-1 processors to find in parallel sensitivity function for a large dimension system.

3.2 Distribution data stage

In this stage, we defined vectors $v_1^0, v_2^0 \ldots v_n^0$. Figure 1, illustrates this stage.

$v_1^0 \rightarrow$	1	0	…..	0
$v_2^0 \rightarrow$	0	1	…..	0
$v_n^0 \rightarrow$	0	0	…..	1

Fig. 1: defined vectors $v_1^0, v_2^0 \ldots v_n^0$.

Given A_1 first row matrix A, A_2 second row matrix A, and A_n last row matrix A with unknown vector w. Figure. 2, illustrates the mentioned above.

$A_1 \rightarrow$	a_{11}	a_{12}	…..	w_1
$A_2 \rightarrow$	a_{21}	a_{22}	…..	w_2
…				
$A_n \rightarrow$	A_{m1}	A_{m2}	…..	w_m

Fig. 2: Distribution rows stage matrix A with unknown vector w .

3.3 Multiplication and division Stages

In this stage we find unknown:
$C_2^1 = (A^1 * v_2^0) / (A^1 * v_1^0)$ and in parallel we calculate variable C until C_m :

$$C^{n-1}{}_m = A_{n-1} * V^1{}_m / A_{n-1} * V^1{}_{m-1}$$

And in parallel we find vector $V_2^1 \ldots V_m^1$:

$$V_2^1 = V_2^0 - C_2^1 * V_1^0$$

…….

$$V_m^1 = v_m^0 - v_m^1 * v_1^0$$

Finally we calculate the equations:

$$C_m^{n-1} = A_{n-1} * V_m^1 / A_{n-1} * V_{n-1}^1,$$

and find unknown $x_1, x_2 .. x_n$ for original system.

$$V_m^{n-1} = V_m^1 - C_m^{n-1} * V_{n-1}^1.$$

$$V_m^{n-1} = (x_1, x_2, \ldots, x_n)^T.$$

3.4 Distribution data for similar system

Distribute data to p= n-1 processors, and calculate unknown vector for similar system $(x^|)^t = -d^t T^{-1} = (x^|_1, x^|_2, \ldots, x^|_n)$, (we do that at the same time when we calculated unknown vector for original system $X = (x_1, x_2, \ldots, x_n)^T$ as mentioned above).

3.5 Calculate in parallel sensitivity function algorithm

Step 1. Compute unknown vector for similar system $X^| = (x^|_1, x^|_2, \ldots, x^|_n)$ using next equation :

$$(x^|)^t = -d^t T^{-1} \ldots\ldots\ldots\ldots\ldots\ldots(7)$$

Step 2.multiplicate equation (6) from the right side by matrix T and transpose left and right side to obtain a system with respect to $X^|$:

$$T^t x^| = d \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(8)$$

Step 3 . Calculate:

$$\partial X/\partial h = -T^{-1} (\partial T/\partial h) X - (\partial W/\partial h) .. (9)$$

Step 4. Find sensitivity Function f with respect to h:

$$\partial f/\partial h = -d^t T^{-1} (\partial T/\partial h\, X - \partial W/\partial h) .. (10)$$

Step 5. Put the expression (7) in (10) then:

$$\partial f/\partial h = (x^|)^t \partial T/\partial h X - (x^|)^t W/\partial h \ldots..(11)$$

To implement the expression (11) we just need to resolve in parallel the tow linear systems (1) and (8) by using VP algorithm.

4. A numerical Example

Figure. 3, illustrates the electric circuit, in which we want to calculate in parallel the sensitivity function of the output potential v_{out} with respect to resistance g_2, condensers c_1, and c_3, respectively, the mathematical model for this circuit is :

$$\begin{array}{|c|c|}\hline G_1+G_2+sC_1+ +sC_2 & G_2 - sC_2 \\ \hline G_2 - sC_2 & G_2+G_3+sC_2+ +sC_3 \\ \hline \end{array} \times \begin{array}{|c|}\hline V_1 \\ \hline V_2 \\ \hline \end{array} = \begin{array}{|c|}\hline 1 \\ \hline 0 \\ \hline \end{array}$$

Using VP algorithm in parallel, we find unknowns vector X for original system:

$$x = \begin{array}{|c|}\hline v_1 \\ \hline v_2 \\ \hline \end{array} = \begin{array}{|c|}\hline (3 - j)/5 \\ \hline (2+j)/5 \\ \hline \end{array}$$

At the same time we find unknowns vector X^{l} for similar system :

$$x^{l} = \begin{array}{|c|}\hline v^{l}_1 \\ \hline v^{l}_2 \\ \hline \end{array} = \begin{array}{|c|}\hline -(2+j)/5 \\ \hline (-3+j)/5 \\ \hline \end{array}$$

Finally we just do the multiplication operation to find the sensitivity function as follows:

$$\partial v_{out} / \partial C_1 = {}_s v^{l}_1 v_1 = 1-j7/25,$$

$$\partial v_{out} / \partial G_2 = (v^{l}_1 - v^{l}_2)(v_1 - v_2) = -3-j4/25,$$

$$\partial v_{out} / \partial C_3 = {}_s v^{l}_2 v_2 = 1-j7/25.$$

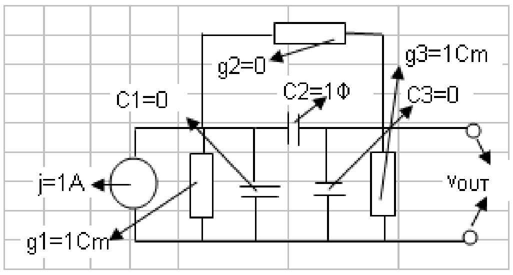

Fig. 3: electric circuit to calculate sensitivity function for v_{out} with respect to variation parameters (C1, G_2, C3).

5. 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 measured the longest thread which is the last one in this present case, then every thread take 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 used one processor (single thread) and the time when used a multi processors in parallel (multi thread) to calculate the unknown vector. From the table 1, figure 4 and 5 show that time and performance is increased with respect to the size of matrix, which represents the linear system.

Table 1: Comparison between single and multithread

performance	Multi thread, MS	Single thread, MS	Matrix Dimension
1.25	0.000002	0.000005	1x2
6.933	0.00001	0.000105	2x3
13.741	0.000025	0.000325	3x4
29.83	0.000033	0.000809	4x5
53.267	0.000027	0.001718	5x6

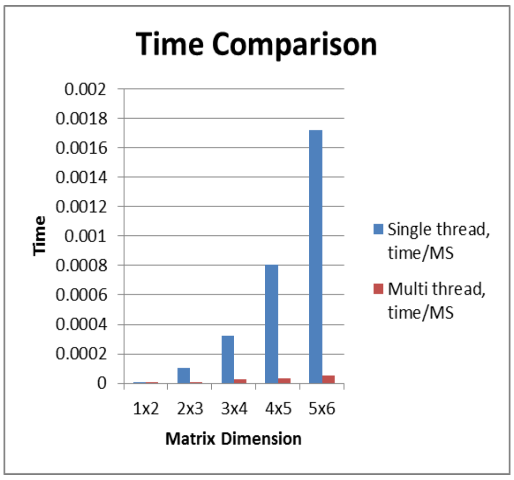

Fig 4: Time comparison between single and parallel to calculate unknown's vector

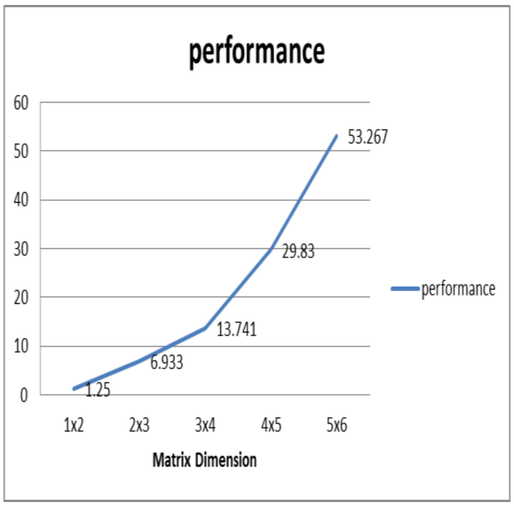

Fig 5: System performance with respect to matrix dimension.

6. CONCLUSION

One of the most important fields in sciences and techniques is to calculate a sensitivity function for large dimension control systems, this research demonstrates new parallel algorithm to do that, the idea of this algorithm is to distribute the coefficient mathematical model studies system to a number of processors and use the parallelism to reduce the running time for solving linear equations of a big size by using original and similar systems which working in parallel, and that is confirmed in the description of VP algorithm , The running time is reduced by O(t/2) and, The efficiency of VP algorithm is increased by 50-60%.

REFERENCES

[1] Duff, I.S. and H.A. van de Vorst, 1999.Developments and Trends in parallel Solution of Linear Systems. Technical Report RAL TR-1999-027.
[2] Ogita, T, S. M. Rump, and S. Oishi, 2005.Accurate Sum and Dot Product.SIAM Journal on Scientific Computing, 26(6):1955-1988.
[3] Klatte, R., U.Kulisch, and A. Wiethoff.1993.C-XSC-A C++ Class Library for Extended Scientific Computing.Spriger-Verlag.
[4] Duff, I.S., 1999.The Impact of High Performance Computing in the Solution of Linear Systems: Trend and Proplems.Technical Report RAL TR-1999-072.
[5] Facius, A., 2000.Iterative solution of linear systems with improved arithmetic and result verification .PhD thesis, University of Karlsruhe.
[6] Bohlender, G., 1990.What Do Need Beyond IEEE Arithmetic? Computer Arithmetic and Self-validation Numerical Methods.Academic Press Professional,Inc.,San Diego,CA.
[7]Eisentat,S.C.,M.T.Heath,1988.Modified cyclic algorithm for solving triangular system on distributed-memory multiprocessor.SIAM J.Stat.Comput.,9(3):589-600.
[8]Holbig,c.a.,P.S.Morandi,2004.Self verifying Solvers for Linear Systems of Equations in C-XSC.In Proceeding of Parallel and Distributed Programming (PPAM),volume 3019,pages 292-297.
[9] Cunha, R.D. and T.Hopkins, 1991.The parallel solution of triangular systems of linear equations. Technical Report 86*, University of kent, Cantenbary, UK.
[10] Saad, Y., 1995.Iterative Methods for Sparse Linear Systems.Boston:PWS Publishing Company.
[11] Feng, T., 2002.A Message-Passing Distributed-Memory Newton-GMRES Parallel Power Flow Algorithm. Volume 3, pages 1477-1482.

[12] Heath, J.W., 1993.Parallel Numerical Linear Algebra.Technical Report UCB CSD-92-703.
[13] Hedayat, G.A., 1993.Numerical Linear Algebra and Computer Architecture. Technical Report UMCS-93-1-5.
[14] Lo, C.G. and D.W.Cheunge, 1997.Efficient Parallel Algorithm for Dense Matrix LU Decomposition with Pivoting on Hypercubes.Computer & Mathematics with Applications, 33(8):39-50.

Hamed Khaled Alrjoub
Alrjoub is an assistant professor in computer science department, deanship of preparatory year, Um Alqura University/ Saudi Arabia. He was a head of computer science department, Irbid national university / Jordan. He got a PhD from international civil aviation university, Kiev / Ukraine, his area of interest is Parallel processing, E-commerce, e-government, system analysis, database, data mining, machine learning.