Journal of Computer Science 4 (11): 928-933, 2008
ISSN 1549-3636

Calculate Sensitivity Function Using Parallel Algorithm

Hamed Al Rjoub
Irbid National University, Irbid, Jordan

Abstract: Problem statement: To calculate sensitivity functions for a large dimension control system using one processor, it takes huge time to find the unknowns vectors for a linear system, which represents the mathematical model of the physical control system. This study is an attempt to solve the same problem in parallel to reduce the time factor needed and increase the efficiency. **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 $n_1 x n_2$, $X = T^{-1} W$, 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 parallel algorithm divided 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 $(x')^t = d^t T^{-1} = (x_1',x_2',\ldots x_n')^T$ 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 was reduced to O(t/n-1) and, The Performance of parallel algorithm was increased by 40-55%. **Conclusion:** Used parallel 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

The ability to develop mathematical models in Biology, Physics, Geology and other applied areas has pull 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]. 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. 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. 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)[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:

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

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

It is well known (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:

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

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[6-8]. Different improvements have been proposed, for example, including sequential MC techniques[5], resolve-based MC methods[1] 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[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[11,7]. 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[2], 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[5] 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[10]. 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[7]. 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[11], 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[11,10]. 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[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].

We consider the linear system (1) where A is a tridiagonal matrix of order n of the form shown in (6), $x = (x_0, x_1, ..., x_{n-1})^T$ is the vector of unknowns and $d = (d_0, d_1, ..., d_{n-1})^T$ is a vector of dimension n:

$$A = \begin{bmatrix} b_o & c_o & & & \\ a_1 & b_1 & c_1 & & \\ & a_2 & b_2 & c_2 & \\ & & \ddots & \ddots & \\ & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & a_{n-1} & b_{n-1} \end{bmatrix} \quad (6)$$

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

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

$$U = \begin{bmatrix} U_0 & C_0 & & & \\ & U_1 & C_1 & & \\ & & \ddots & \ddots & \\ & & & U_{n-2} & C_{n-2} \\ & & & & U_{n-1} \end{bmatrix}$$

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 we concentrate on the computation of the u_i's.

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 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
1 X 2	0.000005	0.000004	1.250
2 X 3	0.000119	0.000015	7.933
3 X 4	0.000425	0.000027	15.741
4 X 5	0.001073	0.000047	22.830
5 X 6	0.001718	0.000030	57.267

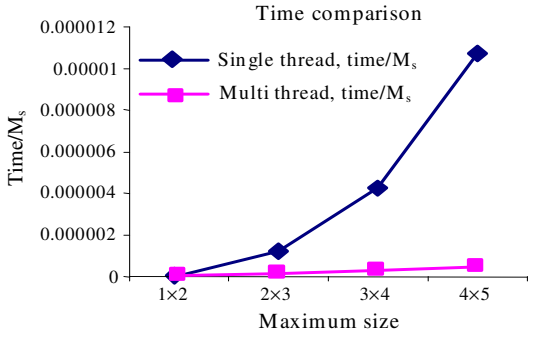

Fig. 1: Time comparison between single and parallel to calculate unknowns vector

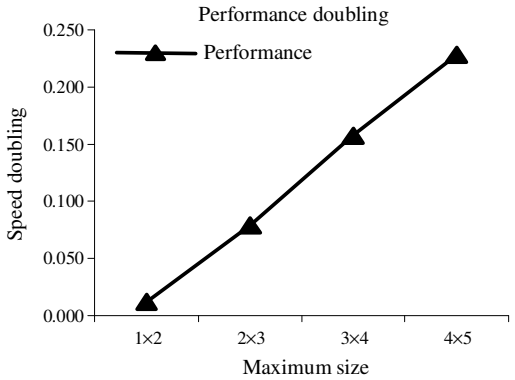

Fig. 2: System performance chart

DISCUSSION

The time to find in parallel unknown vector was decreased with respect to the increased size of the matrix, which represent the mathematical model of physical system (Table 1),and with respect to the single thread , the performance was increased (Fig. 1 and 2). The main goal of Parallel 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 $h(\partial f / \partial h)$. Parallel 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 $X=(x_1, x_2, ..., x_n)^T$. Distribution data (at the same time) to p processors and calculate unknown vector for similar system $(x^I)^t = -d^t T^{-1} = (x^I_1, x^I_2, ..., x^I_n)^T$. Multiplication operation for unknown's $x_i \times x^I_i$ respectively using p processors to find in parallel sensitivity function for a large dimension system.

Distribution data stage: In this stage, given first row matrix T and the first component of right side linear equation TX = W (w_1)to the first processor p_1 ,and first row with component w_1 to second processor p_2 and first row with component w_1 to third processor ,and first row with component w_1 to the p_{n-1} processor. Figure 3 shows this stage.

Given p_1 second row matrix T with component w_2, p_2 third row matrix T with component w_3 and p_{n-1} last row matrix T with component w_n (Fig. 4).

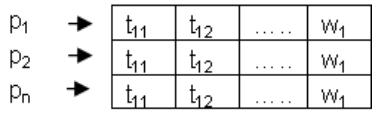

Fig. 3: Distribution first row matrix T and w_1

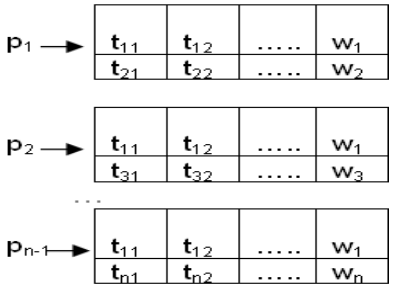

Fig. 4: Distribution rows stage matrix T, where p = n-1, (working in parallel)

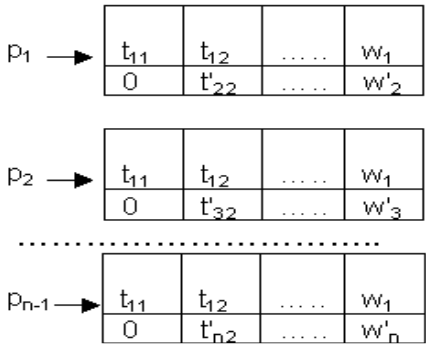

Fig. 5: Multiply first rows (p_1, p_2, p_{n-1}) by Constants c_1,c_2,c_{n-1} respectively

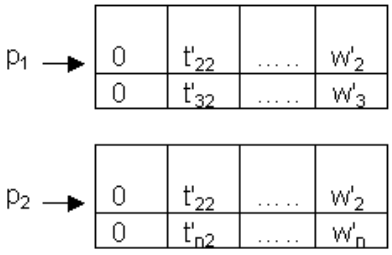

Fig. 6: Distribution rows between processors, where p = n-2 (working in parallel)

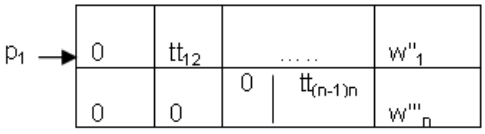

Fig. 7: Representation of the final step on processor p_1

Multiplication Stage: Multiply the first row processors p_1, p_2, p_{n-1} by constants c_1,c_2,c_{n-1} respectively and subtract second row components from the result to obtain zero in the first component of the second rows Fig. 5.

The second row processor p_1 become the first row for all processors, where the number of processors is equal n-2 (Fig. 6).

After n-1 cycles of multiplication operation and distribution rows between processors, just on p_1 we obtained a system, which contains tow rows. Figure 7 shows the final step.

Finally we get the mathematical model for original system as triangular equations:

$$\begin{matrix} t_{11} & t_{12} & \dots & t_{1n} \\ 0 & t'_{22} & \dots & t'_{2n} \\ 0 & 0 & \dots & t'_{3n} \\ 0 & 0 & 0 & tt_{(n-1)n} \end{matrix} \times \begin{matrix} X_1 \\ X_2 \\ \dots \\ X_n \end{matrix} = \begin{matrix} W_1 \\ W'_2 \\ W''_3 \\ W'''_n \end{matrix}$$

The above triangular equations are solved by back substitution. From the last equation, we immediately have $x_n = W'''_n / tt_{(n-1)n}$. By substituting this value in the n-1 equation, we find x_{n-1} and so on we find unknown vector for original system $X=(x_1,x_2,...,x_n)^T$.

Distribution data for similar system: Distribute data to p = n-1 processors and calculate unknown vector for similar system $(x^I)^t = d^t T^I = (x^I_1, x^I_2, ..., x^I_n)$, (we do that at the same time when we calculated unknown vector for original system $X=(x_1,x_2,...,x_n)^T$ as mentioned above).

Calculate in parallel sensitivity function algorithm:

Step 1: Compute unknown vector for similar system $X^I=(x^I_1,x^I_2,...,x^I)$ using next equation:

$$(x^I)t = -dT^{-1} \quad (7)$$

Step 2: Multiplicate Eq. (7) from the right side by matrix T and transpose left and right side to obtain a system with respect to x^I:

$$T^t X^I = -d \quad (8)$$

Step 3: Calculate:

$$\partial X / \partial h = -T - 1(\partial T / \partial h)X - (\partial W / \partial h) \quad (9)$$

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

$$\partial j / \partial h = -d^t T^{-1}(\partial T / \partial W / \partial h) \quad (10)$$

Step 5: Put the expression (6) in (9) then:

$$\partial j / \partial h = (x^I)^t \partial T / \partial hX - (x^I)^t \partial W / \partial h \quad (11)$$

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

A numerical example: Figure 8 shows the electric circuit, in which we wont 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:

$$\begin{matrix} G_1 + G_2 + {}_S C_1 + {}_S C_2 & G_2 - {}_S C_2 \\ G_2 - {}_S C_2 & G_2 + G_3 + {}_S C_2 + {}_S C_3 \end{matrix} \times \begin{matrix} V_1 \\ V_2 \end{matrix} = \begin{matrix} 1 \\ 0 \end{matrix}$$

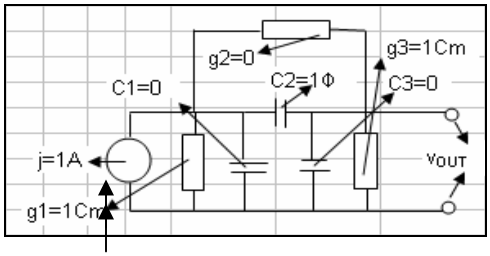

Fig. 8: Electric circuit to calculate sensitivity function for v_{out} with respect to variation parameters (C_1, G_2, C_3)

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

$$x = \begin{matrix} v_1 \\ v_2 \end{matrix} = \begin{matrix} (3-j)/5 \\ (2+j)/5 \end{matrix}$$

At the same time we find unknowns vector X^I for similar system:

$$x^I = \begin{matrix} v^I_1 \\ v^I_2 \end{matrix} = \begin{matrix} -(2+j)/5 \\ (-3+j)/5 \end{matrix}$$

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

$$\partial V_{out} / \partial C_1 = s V^I_1 v_1 = 1 - j7/25$$

$$\partial V_{out} / \partial G_2 = (V^I_1 - v^I_2)(V_1 - v_2) = -3 - j4/25$$

$$\partial V_{out} / \partial C_3 = s V^I_2 v_2 = 1 - j7/25$$

CONCLUSION

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

ACKNOWLEDGMENT

The researcher acknowledge the financial support (Fundamental Research Scheme) received from Irbid National University, Irbid, Jordan.

REFERENCES

1. Duff, I.S. and H.A. van de Vorst, 1999. Developments and trends in parallel solution of linear systems. Paral. Comput., 25: 1931-1970. DOI: 10.1016/S0167-8191(99)00077-0
2. 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
3. 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. DOI: 10.1137/0910025
4. Duff, I.S., 2000. The Impact of High Performance Computing in the solution of linear systems: Trend and proplems. J. Comput. Applied Math., 123: 515-530. DOI: 10.1016/S0377-0427(00)00401-5
5. Liu, Z. and D.W. Cheung, 1997. Efficient parallel algorithm for dense matrix LU decomposition with pivoting on hypercubes. J. Comput. Math. Appl., 33: 39-50. DOI: 10.1016/S0898-1221(97)00052-7
6. Pan, V. and J.Reif, 1989. Fast and efficient parallel solution of dense linear system. Comput. Math. Appl., 17: 1481-1491. http://cat.inist.fr/?aModele=afficheN&cpsidt=7227240
7. Eisentat, S.C. and M.T. Heath, 1988. Modified cyclic algorithm for solving triangular system on distributed-memory multiprocessor. SIAM J. Sci. Stat. Comput., 9: 589-600. DOI: 10.1137/0909038
8. 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. http://cat.inist.fr/?aModele=afficheN&cpsidt=15811200
9. Singer, M.F., 1988. Algebraic relations among solutions of linear di-erential equations: Fano's theorem. Am. J. Math., 110: 115-143. http://www.jstor.org/pss/2374541
10. Saad, Y., 1996. Iterative methods for sparse linear systems. Proceeding of the Symposium on FPGAs, (FPGAs'96), ACM Press, New York, USA., pp: 157-166.
11. Feng, T., 2002. A message-passing distributed-memory newton-GMRES parallel power flow algorithm. Proceeding of the Summer Meeting Power Engineering Society, July 25, IEEE Computer Society, Washington DC., USA., pp: 1477-1482. DOI: 10.1109/PESS.2002.1043638