Parallel Newtonian Optimization without Hessian Approximation

Khalil K. Abbo
College of Computer sciences and Mathematics
University of Mosul, Iraq

Received on: 01/09/2005 **Accepted on: 26/12/2005**

ABSTRACT

The purpose of this paper is to introduce parallel algorithms based on the Newton method for solving non-linear unconstrained optimization problem in (MIMD) parallel computers by solving linear system in parallel using Gaussian Elimination method rather than finding inverse Hessian matrix to avoid the errors caused by evaluating the inverse matrix and also to increase computing power and reduce run time.

Keywords: non-linear unconstrained optimization, Newton method, parallel algorithm, Hessian matrix.

أمثلية نيوتن الموازية بدون تقريب هيسيان

خليل خضر عبو
كلية علوم الحاسوب والرياضيات، جامعة الموصل

تاريخ الاستلام: 2005/09/01 **تاريخ القبول: 2005/12/26**

الملخص

الغرض من هذا البحث هو اقتراح خوارزمية تعتمد على طريقة التوازي تستند إلى طريقة نيوتن لحل مسائل الامثلية اللاخطية باستخدام حاسبات متوازية من النوع (MIMD) وذلك بحل نظام معادلات خطية بطريقة الحذف لكاوس بشكل متوازٍ بدلاً من إيجاد معكوس مصفوفة هيسيان لكي نتجنب الخطأ الناتج من حساب معكوس المصفوفة ولتزيد من قوة العمليات الحسابية وكذلك لتقليل الزمن اللازم لحل المسالة.

الكلمات المفتاحية: الامثلية اللاخطية غير المقيدة، طريقة نيوتن، خوارزمية التوازي، مصفوفة هيسيان.

1. Introduction:

This paper is concerned with finding a local minimum x^* of the unconstrained optimization problem

$$\min \ f(x) \qquad (1)$$

where $x \in R^n$ and the objective $f: R^n \rightarrow R$ is twice continuously differentiable.

Methods for unconstrained optimization are generally iterative methods in which the user typically provides an initial estimate of x^*, and possibly some additional information. A sequence of iterates $\{x_i\}$ is then generated according to some algorithm. Usually the algorithm is such that the sequence of function values f_i is monotonically decreasing (f_i denotes $f(x_i)$). Due to practical applications of the unconstrained optimization problem, a considerable amount of effort has been expanded on the development of efficient sequential algorithms for the solution of this problems. Recent advances in parallel computing technology have made it possible to solve the optimization problem more effectively and increasing computing power for this reason it is natural that there is an increased interest in designing parallel algorithms for various types of applications. In this paper, we will discuss parallel numerical algorithm namely Newton algorithm for the general unconstrained optimization problem using multiprocessors or (MIMD) parallel machines which consist of a number of fully programmable processors capable of simultaneously performing entirely different operations on different data, where each processor has its own local memory.

Concurrent computation can be done at different levels. Our focus is on using (MIMD) computers where a number of processors communicate and cooperate to solve a common computational problem. Multiprocessor parallel computation involves three key ingredients hardware, software (programming language, operating system and compiler) and parallel algorithms (See [8] or [11]).

A popular class of methods for solving problem (1) is the Quasi-Newton (QN) methods.

Assume that at the ith iteration, an approximation point x_i and n×n matrix H_i are available, then the methods proceed by generating a sequence of approximation points via the equation

$$x_{i+1} = x_i + \alpha_i d_i \qquad (2)$$

where $\alpha_i > 0$ is the step- size which is calculated to satisfy certain line search conditions and d_i is an n-dimensional real vector representing the search direction. For QN methods, d_i is defined by :

$$d_i = -H_i g_i \qquad (3)$$

where $g_i = \nabla f(x_i)$ is the gradient vector of $f(x)$ evaluated at point $x=x_i$ and H_i is an approximation of G_i^{-1} (where $G = \nabla^2 f(x)$) which is corrected or updated from iteration to iteration in general H_i is symmetric and positive definite , there are different choices of H_i , we list here some must popular forms (See [5],[6]or[4]).

$$H_{i+1} = H_i + \frac{(\delta_i - H_i r_i)(\delta_i - H_i r_i)^T}{(\delta_i - H_i r_i)^T r_i} \qquad (4)$$

Where $\delta_i = x_{i+1} - x_i$ and $r_i = g_{i+1} - g_i$ is called rank- one formula, the other forms are BFGS and DFP where

$$\underset{i+1}{\overset{BFGS}{H}} = \underset{i}{H} + \left(1 + \frac{r_i^T H_i r_i}{\delta_i^T r_i}\right)\frac{\delta_i \delta_i^T}{\delta_i^T r_i} - \left(\frac{\delta_i r_i^T H_i + H_i r_i \delta_i^T}{\delta_i^T r_i}\right) \qquad (5)$$

$$\underset{i+1}{\overset{DFP}{H}} = \underset{i}{H} + \frac{\delta_i \delta_i^T}{\delta_i^T r_i} - \frac{H_i r_i r_i^T H_i}{r_i^T H_i r_i} \qquad (6)$$

QN methods mentioned above have two disadvantages: one of which *QN* algorithms require line search which is expensive in practice and The second is the accumulation of error in evaluating the approximate Hussein matrix H_i at each iteration.

To overcome these disadvantages we use parallel Newton method without approximating Hussein matrix and neglecting line search

2-Newton Method

Newton method uses first and second derivatives, the idea behind this method is as follows:

Given a starting point x_0, we construct a quadratic approximation to the objective function that matches the first and the second derivative values at the point. We then minimize the approximate (quadratic) function instead of the original objective function. We use the minimizer of the objective function as the starting point in the next step and we repeat the procedure iteratively. If the objective function is quadratic, then the approximation is exact, and the method yields the true minimizer in one step. If on the other hand, the objective function is not quadratic then the approximation will provide only an estimate for the position of the true minimizer.

We can obtain a quadratic approximation to the given twice continuously differentiable function *f(x)* using the Taylor series expansion of *f* about the current point x_i, neglecting terms of order three and higher in Δx_i for simplicity we let $\Delta x_i = \delta_i$

$$f(x_i + \delta_i) \approx q(\delta_i) = f + \delta_i^T + \frac{1}{2}\delta_i^T G_2 \delta_i \qquad (7)$$

where δ_i, $x_i \in R^n$ and the matrix G usually positive definite when x_i is in some neighborhood of x^*. A unique minimizer of $q(\delta_i)$ exists if and only if G is positive and Newton method is only well defined in this case. (See [5]). Then (δ_i) is obtained by finding the stationary point at $q(\delta_i)$, which requires the solution of the linear system

$$G_i\delta_i = -g_i \qquad (8)$$

the next step is then defined by

$$x_{i+1} = x_i + \delta_i \qquad (9)$$

We then prove that the sequence $\{x_i\}$ convergence and the order of convergence is two (See [13]). These local convergence properties represent the ideal local behavior which other algorithms aim be evaluated as far as possible (See [6]). In fact, super linear convergence of any algorithm is obtained if and only if the step Δx_i is asymptotically equal to the step given by solving (8). This fundamental results due to Dennis and More (1974) emphases the importance of the Newton step for local convergence.

Two quite different classes of methods for solving the linear system (8) are at interest: direct methods and iterative methods. In direct method, the system is transformed to a system of simpler form e.g. triangular or diagonal form, which can be solved in an elementary way. The most important direct method is the Gaussian elimination, we will use it to solve the linear system (8).

3. Gaussian Elimination Method

A fundamental observation in Gauss elimination is the elementary row operations, which can be performed on the system without changing the set of solution. These operations are:

1- adding a multiple of the ith equation to the jth equation

2- Interchange two equations.

The idea behind Gaussian elimination is to use such elementary operations to eliminate the unknowns in the system (8) in a symmetric way, so that at the end an equivalent upper triangular system is produced, which is then solved by back – substitution.

If $a_{11} \neq 0$. (where $a_{ij} \in G$). Then in the first step we eliminate δ_1 from the last (n-1) equation by subtracting the multiple

$$L_{i1} = \frac{a_{i1}}{a_{11}} \qquad i = 2,3,....,n$$

of the first equation from the ith equation. This produces a reduces system of (n-1) equations in the (n-1) unknowns $\delta_2, \delta_3, ..., \delta_n$, where the new coefficients are given by :

$$\tilde{a}_{ij} = a_{ij} - L_{i1}\ a_{1j}\ ; \quad \tilde{g}_i = g_i - L_{i1}g_1 \quad i = 2,3,,......,n$$

If $a_{22} \neq 0$, we can eliminate δ_2 from the last (n-2) equations. After m-1 steps, $m \leq n$ of Gaussian elimination the matrix G has been reduced to the form:

$$\overset{(m)}{G} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & a_{2n}^{2} \\ \vdots & 0 & \ddots & \vdots \\ 0 & 0 & & a_{nn}^{(m)} \end{bmatrix} ; \; \overset{(m)}{g} = \begin{bmatrix} \overset{(1)}{g_1} \\ \overset{(2)}{g_2} \\ \vdots \\ \overset{(m)}{g_n} \end{bmatrix}$$

Where $\overset{(m)}{a_{ij}}$ is the reduced element produced from row operation at steps m=1,2,…,n .

The elements $\overset{(m)}{a_{ii}} \; (i = 1,......, n)$ are called pivotal elements. There are two difficulties in the Gaussian elimination method. The first if a zero pivotal element is encountered i.e $\overset{(m)}{a_{ii}} = 0$ for some $m \leq n$, then we cannot proceed. But this case does not occur in our problem, which is stated in (8). Since G is square matrix and positive definite, this means that all diagonal elements of G are non-zero. Second a zero pivot in exact arithmetic will almost invariably be polluted by rounding errors in such a way that it equals some small non-zero number, unfortunately, there is no general rule which can be used to decide when a pivot should be taken to be zero. What tolerance to use in such a test should depend on the context. This question and the stability (without these two difficulties) of Gaussian method treated in [Laxxus, 2000]

Gaussian Elimination Algorithm

Given a matrix $G = G^{(1)} \in R^{n\times n}$ and a vector $g = g^{(1)} \in R^{n}$ the following algorithm reduces the system $G\delta$=g to upper triangular form $a_{ii} \neq 0$, $\mathrm{i} = 1,2,..,\mathrm{n}$, since $\underset{n\times n}{G}$ is positive and symmetric)

```
For k=1:n-1
For i=k+1:n
    $L_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$
for j=k+1:n
$a_{ij}^{(k+1)} = a_{ij}^{(k)} - L_{ik} a_{kj}^{(k)}$;
end
```

```
$g_i^{(k+1)} = g_i^{(k)} - L_{ik} g_k^{(k)}$
end
end
```

4- Parallel algorithms for finding Newton direction:

In order for the algorithm to be as efficient as possible, we must be able to exploit the memory hierarchy of the architecture we are running on. The memory hierarchy refers to different kinds of memories inside a computer. These range from very fast, expensive and therefore small memory at the top of the hierarchy, down to slow. Cheap and very large memory at the bottom (for more detail see [13]). Useful floating point operations can only be done at the top of hierarchy, in the registers. Since an entire large matrix cannot fit inside the registers, it must be moved up and down through the hierarchy. This takes time and for maximum performance should be optimized for each architecture.

The Basic linear Algebra Subprograms BLAS are high performance routines for performing basic vector and matrix operations. The BLAS are specifically machine optimized to exploit the memory hierarchy. Level 1 BLAS does vector-vector operation, Level 2 BLAS does matrix-vector operations and Level 3 BLAS does matrix-matrix operations (see [7]).

In order to speed up Gaussian elimination, we wish to use as high a level of BLAS as possible.

There are different ways to parallelize the Gaussian elimination method. In this paper, we consider two types of parallel Gaussian elimination, the first is based on the rows of the matrix G and the second uses columns of G.

4-1: Parallel Gaussian Elimination Based on Rows (PGER)

The most popular form of Gaussian elimination is defined by subtracting multiple of row of the matrix *G* from other rows in order to reduce (8) to an upper triangular system, which is then solved directly by back substitution. We assume that the computing system consists of *n* (where *n* is the dimension of the problem) identical independent processors also assume that the ith row of *G* is assigned to processors *i*. At the first stage, the first row of G is sent to all processors and then the elimination of the first elements of rows 2 to n can be done in parallel in the processors $P_2, \ldots\ldots, P_n$.

The computation is continued by sending the new second row of the reduced matrix from processors $P_3, \ldots\ldots, P_n$, then doing the calculations in parallel ; and so on [see figure, 1]

P1

1- $save\ x^{(1)}, g^{(1)}, G^{(1)}$

2.find

$l_{j1}=\frac{a^{(1)}_{j1}}{a^{(1)}_{11}}$, j=2,...n,

and send l_{j1} to p_j

3- $send$ first $R_1^{(1)}\ to\ p_2,..., p_n$

4. do for $i=2,...,n$

$g_i^{(1)} - L_{2i}g_1^{(1)} = g_i^{(2)}$ and send the vector $g^{(2)}$ to P_n

5..recieve L_{j2} from p_2 and repeat (4) for $g^{(2)}$

6.receive the vector $\delta^{(1)}$ to find $x^{(2)} = x^{(1)} + \delta^{(1)}$ and find new g from $x^{(2)}$

$g^{(1)} = g$ and $G^{(1)} = G^{(2)}$

6.- if $\|g_1\| < \varepsilon$ stop , otherwise

$x^{(1)} = x^{(2)}$;Go to 1

P2

1- save second row $R_2^{(1)}$

2- receive $R_1^{(1)}$ and L_{21} from p_1 to find $R_2^{(2)}$ as follows

3- do for i=1,…,n

$a_{2i}^{(1)} - L_{21}a_{1i}^{(1)} = a_{2i}^{(2)}$

4. recieve $R_i^{(2)} (i = 3,...,n)$ and find

$L_{i2} = \frac{a^{(2)}_{i1}}{a^{(2)}_{22}}$

send it to $P_3,……,P_n$

…… *Pn*

1-save $R_n^{(1)}$ and receive $R_1^{(1)}\ and\ L_{21}$

2- do for i=1,…,n to find the reduced $R_n^{(2)}$ from

$a_{ni}^{(1)} - L_{n1}a_{1i}^{(1)} = a_{ni}^{(2)}$

3 -receive $R_2^{(2)},...., R_{n-1}^{(2)}$ from processors $P_2...P_{n-1}$

$G^{(2)} = \begin{bmatrix} R_1^{(2)} \\ R_n^{(2)} \end{bmatrix}$

4- If G upper triangular use back substitution to find $\delta^{(1)}$ from

$\delta_i = g_i^{(1)} - \frac{a_{i,i+1}}{a_{ii}}\delta_{i+1}$

$i = (n-1)-(n-2),....,1$

and $\delta_n^{(1)} = \frac{g_n^{(1)}}{a_{nn}^{(2)}}$

5- send $\delta^{(1)}$ to P_1

Figure (1) tasks of processors in PGER

This approach has two major drawbacks (See [9]) (i) there is a considerable communication of data between the processors at each stage (ii) the number of active processors decreases by one at each stage.

4-2: Parallel Gaussian Elimination Based on Columns (PGEC)

Let $x^{(1)}, g^{(1)} \in R^n$ and $G = G^{(1)} \in R^{n \times n}$ are given and assume that the computing system consists of n+1 identical independent processors and fully communicated processors, also assume that the *jth* column (C_j). of the matrix $G^{(1)}$ is assigned to processor *j* and column $g^{(1)}$ assigned to P_{n+1} . The first step is to find $L_{i1} = \frac{a_{i1}^{(1)}}{a_{11}^{(1)}}$; $i = 2,......n$ in P_1 then the values of L_{i1} can be sent to processor P_j (j=2,…..,n+1) , then the computation can be done in parallel in all processors as follows :
for each processors j =1,……,n

$a_{ij}^{(1)} - L_{i1}a_{jj}^{(1)} = a_{ij}^{(2)} \qquad ; i = 2,....,n$

and processor n+1 compute the vector $g^{(2)}$ from

$g_m^{(1)} - L_{m1}g_1^{(1)} = g_m^2 \qquad ; m = 2,...., n$

for next step (see figure (2) we can find

$L_{i1} = \frac{a_{i1}^{(1)}}{a_{11}^{(1)}}$; $i = 3,......n$ in p_2 and then send to processors P_3,……,P_{n+1} and the process is repeated until the matrix G is transformed to upper triangular matrix, we denote it $G^{(k)}$ latter k steps and then backsubstitution is used to find the vector δ .

The new point x^{k+1} is found from equation (9). Again in this algorithm the number of active processors decreases by one at each stage. The main advantage of this algorithm is that we optimized data transfer between processors since we need only transfer elements instead of vectors, this will reduce the time required to complete the computations.

p_1

Step(1)

1.save

$x^{(1)}, g^{(1)}, C_1^{(1)} = \left[a_{i1}^{(1)} \right]$

2- find

$L_{i1} = \frac{a_{i1}^{(1)}}{a_{11}^{(1)}} \; ; \quad i = 2,\dots,n$

and send it to $P_2,\dots\dots,P_n$

3- do for i=2,….,n

$a_{i1}^{(1)} - L_{i1}a_{11}^{(1)} = a_{i1}^{(2)} = 0$

$C_1^{(2)} = \left[a_{11}^{(2)} \quad 0 \quad 0 \right]^T ; \; a_{11}^{(1)} ; a_{11}^{(2)}$

4.step (k):

1- receive

from

$C_i^k \; (i = 2,\dots\dots,n+1)$

$P_2,\dots\dots,P_n$; is

$G^k = \left[C_1^2 \quad C_2^3 \quad C_n^k \right]$

triangular?

2- use back substitution to find δ

$x^{(2)} = x^{(1)} + \delta;$ to find

new g at stop otherwise

$\|g\| < \varepsilon$ 3.If

4-

$x^{(2)}, x^{(1)}, g^{(1)}, g \, ; G^{(1)} = G^{(k)}$

repeat from step (1)

p_2

1- save

$C_2^{(1)} = \left[a_{i2} \right]$

2- receive L_{i1} from P_1

3- do

$a_{i2}^{(1)} - L_{i1}a_{12}^{(1)} = a_{i2}^{(2)}$

$C_2^{(2)} = \left[a_{in}^{(2)} \right] i = 1,\dots\dots,n$

step(2)

1- find

$L_{i1} = \frac{a_{i2}^{(2)}}{a_{22}^{(2)}}; \; i = 3,\dots,n$

and repeat the process

2- send L_{i2} to $P_2,\dots\dots,P_n$

3- repeat (3) for $C_2^{(2)}$

and L_{i2} to find

$L_2^{(3)} = \left[a_{21} \quad a_{22} \quad 0 \quad 0 \right]^T$

$C_2^{(2)}$ to $P_1$4- send

p_n

1- save

$C_n^{(1)} = \left[a_{in} \right]$

2- receive L_{i1} from P_1

3- do for i=2,…,n

$a_{in}^{(1)} - L_{i1}a_{1n}^{(1)} = a_{in}^{(2)}$

$C_n^{(2)} = \left[a_{in}^{(2)} \right]$ send it to P_{n-1}

step (2)

1- receive L_{32} from P_2

2- repeat (3) for $C_n^{(2)}$

and L_{i2}

3- the process repeated until $G^{(k)}$ triangular

4- send $C_n^{(k)}$ to P_1

p_{n+1}

1- save $g^{(1)};$

2- receive L_{i1}

i=1,2,……n-1 from P_1

3- do for i=1.2,…,n-1

$g_n^{(1)} - L_n g_1^{(1)} = g_n^{(2)}$

:

:

step (k)

1- $C_{n+1}^{(k)} = \left[g_i^k \right] \; i = 1,\dots\dots,n$

2- send $C_{n+1}^{(k)}$ to P_1

Figure (2) tasks of processors in PGEBC

Numerical Example:

We give an example from [1] to illustrate the tasks of processors in the algorithms PGER and PGEC

1-Minimize the function

$$f(x_1, x_2, x_3) = x_1^2 + 2x_2^2 + x_3^2 - 4x_1x_2 + 3x_1x_3 + 2x_2x_3. \qquad (10)$$

$x_0^T = [0\;0\;\tfrac{1}{2}]^T$; $g_0 = [\tfrac{3}{2}\;1\;1]^{\mathrm{T}}$ where

$$g(x) = \begin{bmatrix} 2x_1 - 4x_2 + 6x_3 \\ 12x_2 - 4x_1 + 20x_3 \\ 46x_3 + 6x_1 + 20x_2 \end{bmatrix} ;\; G = \begin{bmatrix} 2 & -4 & 3 \\ -4 & 4 & 2 \\ 3 & 2 & 2 \end{bmatrix}$$

1. Using PGER: We need 3 processors , see figure (3)

P_1	P_2	P_3
1- save $x^{(1)}, g^{(1)}, G^{(1)}$ 2- find $L_{21}\frac{a_{21}^{(1)}}{a_{11}} = \frac{-4}{2} = -2$ and $L_{31}\frac{a_{31}^{(1)}}{a_{11}} = \frac{3}{2}$, send these value to P_2, P_3 3- send $R_1^{(1)} = [2 \quad -4 \quad 3]$ to P_2, P_3 4- do for i=1,2,3 -1-(-2)(3/2)=-4 -1-(-3/2)(3/2)=5/4 $g^{(2)} = [\frac{-3}{2} \quad -4 \quad \frac{5}{4}]^T$ 5-recieve L_{22} from P_3 6- do (5/4)-(-2)(-4)=-27/4 $g = \begin{bmatrix} -3/2 \\ -4 \\ -27/4 \end{bmatrix}$ and send to P_1 7- $x^2 = \begin{bmatrix} 0 \\ 0 \\ 1/2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -1/2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ 8-$\lVert g \rVert < \varepsilon$ stop	1- save $R_2^{(1)} = [-4 \quad 4 \quad 2]$ 2- receive $R_1^{(1)} = [2 \quad -4 \quad 3]$ and L_{21}=-2 3- do i=1,2,3 -4 – (-2)(2)=0 2-(-2)(3)=8 $\therefore R_2^{(2)} = [0 \quad -4 \quad 8]$ 4 – send $R_2^{(2)}$ to P_3	1-save $R_3^{(1)} = [3 \quad 2 \quad 2]$ 2- receive $R_1^{(1)}$ and L_{31} 3- do for i=1,2,3 3-(3/2)(2)=0 2-(3/2)(-4)=8 2-(3/2)(3)=5/2 4 – receive $R_2^{(2)}$ and find $L_{22} = \frac{8}{-4}$ send L_{22} to P_1 5- repeat step (3) for $R_3^{(2)}$ 8-(-2)(-4)=0 (5/2)-(-2)8=37/2 6-$G^{(2)} = \begin{bmatrix} 2 & -4 & 3 \\ 0 & -4 & 8 \\ 0 & 0 & 27/2 \end{bmatrix}$ 7-receive $g^{(2)}$ from P_1 8- use back substitution to find δ and send it to P_1

Figure (3) Tasks of processors using PGER algorithm

2- Using PGEC:

We need 4 processors to solve the problem given (10) as shown in figure (4)

P_1

Step(1)

1-save

$X^{(1)} = [0 \quad 0 \quad 1/2]^T$

$-g = [-3/2 \quad -1 \quad -1]^T$

$C^{(1)} = [2 \quad -4 \quad 3]$

2- find

$L_{21} = \frac{-4}{2},\ L_{31} = \frac{3}{2}$

and send then to P2,,P3,P4

3- do

4 -2(2)=0

3-(3/2)(2)=0

$C_1^{(2)} = [2 \quad 0 \quad 0]^T$

step(2)

1- receive

$C_2^{(3)}, C_3^{(3)}, g^{(3)}$

2- $G^{(2)} = \begin{bmatrix} 2 & -4 & 3 \\ 0 & -4 & 8 \\ 0 & 0 & 27/2 \end{bmatrix}$

$g^3 = \begin{bmatrix} -3/2 \\ -4 \\ -27/4 \end{bmatrix}$

3- use back substitution to find

$\delta = \begin{bmatrix} 0 \\ 0 \\ -1/2 \end{bmatrix}$

$x^2 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$

P_2

1- save

$C_2^{(1)} = [-4 \quad 4 \quad 2]^T$

2- receive L21=-2,,L31=3/2

3- do

4-(-2)(-4)=-4

2-(3/2)(-4)=8

$C_2^{(2)} = [-4 \quad -4 \quad 8]$

step(2)

1- find L22=8/-4=-2 and send it to P3,,P4I

2- do

8-(-2)(-4)=0

$\because C^{3}_{(3)} = [-4 \quad -4 \quad 0]$

send it to P1

P_3

1-save

$C_3^{(1)} = [3 \quad 2 \quad 2]^T$

2-receive L21 ,L31

3-do

2- (-2)(3)=8

2-(3/2)(3)=-5/2

$C_3^{(2)} [3 \quad 8 \quad -5/2]^T$

step(2)

1- receive L22=2

2- do

-5/2-(-2)(8)=27/2

$\therefore C_3^{(2)} = [3 \quad 8 \quad 27/2]^T$

send it to P1

P_4

1- save

$-g^{(1)} = [-3/2 \quad -1 \quad -1]^T$

2-receive L21 ,L31

3- do

-1-(-2)(-3/2)=-4

-1-3/2(-3/2)=5/4

$\therefore g^{(2)} = [-3/2 \quad -4 \quad 5/4]^T$

step(2)

1- receive L22

2- do

5/4 -(-2)(-4)=-27/4

$\therefore g^{(3)} = [-3/2 \quad -4 \quad -27/4]$

send it to P1

Figure (4): Tasks of processors for PGEC algorithm

5-Numerical experiments:

Both the amount of time required to complete the calculations and the subsequent round-off error depend on the number of floating arithmetic operations needed to solve a routine problem. The a mount of time required to perform multiplication or division on a computer is approximately the same and is considerably greater than that required to perform an addition or subtraction (see [3]). The total number of arithmetic operation depends on the size n as follows:

Multiplications / divisions: $\frac{n^3}{3} + n^2 - \frac{n}{3}$

Additions / subtractions: $\frac{n^3}{3} + \frac{n^2}{2} - \frac{5n}{6}$

For large n, the total number of Multiplications and divisions is approximately $\frac{n^3}{3}$ as is the total number of additions and subtractions.

Hence if the all operations done on sequential computer then the accumulation error will be large. On the other hand, dividing the operation on different processors will reduce the amount of errors.

As for the amount of time, If we assume t_1 is time required to complete the calculations in sequential computer, we expect the time required to perform the same calculations in the parallel algorithms proposed is $\frac{t_1}{p-1}$ where *p* number of processors.

In the algorithms PGER and PGEC, there is no need to line search and matrix inversion, all operations are done using ordinary arithmetic operations. Also, the number of function evaluations are only (2), since the Gaussian method gives exact solutions shown in the given example.

<u>Conclusion:</u>

In this paper we proposed two parallel algorithms for solving unconstrained optimization problem. These methods can be extended to any non-homogenous linear system with positive definite matrix. These methods do not require to vector multiplication and matrix inversion as shown in the numerical example.

REFERENCES

[1] Bundy B.. (1984) **A Basic optimization methods** London .Edward Arnold Bdford squar.

[2] Dennis J. and J. More (1974) " A characterization of super linear converge and its application to qusi-Newton Methods*"* **Maths. of Computation.** 28 ,pp.249-860.

[3] Douglas J. and B. Richard (2003) **Numerical Methods** third edition [Brooks/ Cole] Thomson learning: Inc

[4] Edwin K. and H. Stanislaw (2003) **An introduction to optimization** John Wiley & Sons. Inc

[5] Fletcher R. (1987) **Practical optimization** second edition, John Wiley & Sons. Ltd

[6] Fletcher R. (1993) "*An overview of unconstrained optimization*" **Numerical Analysis Report NA/49.**

[7] James, W.; T. Michael,; A. Henk, and V.Van, (1992) "Parallel Numerical Linear Algebra" ,**Mathematical Institute university**.

[8] Jordan, H. and G. Alaghband, (2003) "Fundamentals of parallel processing", USA: New Jersey, Prentice Hall.

[9] Khalaf M. 1990 "Parallel Numerical Algorithms for solving ODEs" Ph.D. Thesis **leads university**.

[10] Laxxuss (2000) " Numerical Mathematics and Scientific computation" **Ake-Bjorlck** Vol. 2.

[11] Managasarian O. (1994) "*Parallel Gradient Distribution in unconstrained optimization*" http://www.google.com /search?hl= en&sa=X&oi=spell&resnum=0&ct=result&cd=1 &q=Mangasarian+ O. +1994+%E2%80%9CParallel+Gradient+Distrib

[12] Michael, W. 2004 "Parallel Gaussian Elimination" **http: // www . student.cs.uwaterloo.ca/~cs775/project/index.html**

[13] Robert, M. (2004) "*Newton's Method for unconstrained optimization*", Cs775, University of Waterloo, **Massachusetts Institute of Technology.**