Performance Analysis of Iterative Methods for Solving Linear Equations using Mobile Agent

Zarni Aung[#1], Thae` Nu Phyu[*2]

[#]*Computer University (Pakokku)*

[1]zarni.aung6cs1@gmail.com

Abstract— It is essential to solve linear equations in various fields such as electronic circuit simulation, structural programming, etc. If the matrix of a linear system is strictly diagonally dominant, iterative methods can be used. Even though the iterative methods are simple, they often require many computations. In this environment, parallel computing techniques are useful for solving numerical problems. This system analyses the performance of iterative methods in parallel computing and how mobile agents fit in parallel computing have been explored. Mobile agents are autonomous computer programs that can migrate between different network locations. Using the capabilities of mobile agent, this system proposes the iterative methods for solving linear equations and performance analysis of these iterative methods.

***Keywords*— — linear equations ,mobile agent ,parallel computing ,performance analysis ,iterative methods**

I. INTRODUCTION

Scientific computing uses computers to analyze and solve scientific and engineering problems based on mathematical models. The use of parallel computers can save time and solve larger problems. This system focuses on evaluating the performance of iterative methods applied for mathematical models computing. Among numerical problems, this system proposes solving the systems of linear equations by using Gauss Jordan, Gauss-Seidel and Jacobi iterative methods. Parallel computing for these numerical problems are developed by using Java Mobile Agent. In any scientific or engineering application of computers, it is usually required to solve a mathematical problem. Such applications span a wide range, from modelling the atmosphere in weather prediction to modelling hot plasmas in theoretical physics from the design of space stations, automobiles, and ground transportation networks.

One of the major foci of distributed computing is using networks of workstations to solve numerical problems efficiently. The most obvious way of achieving this is through parallel computation, when the problem lends itself to parallelization.[4]

In this system, we provide efficient implementations for solving numerical problem such as linear equations. To solve these problems, we use java mobile agents. Mobile agent technology permits dynamic, decentralized creation and execution modules that can execute on remote machines and spawn clones for parallel computation.

II. MOBILE AGENT

Mobile agents are autonomous computer programs that can act in the interest of an entity, migrating between different network locations, executing tasks locally and continuing their execution at the point where they stopped before migration.

Mobile agent has some key features:

Autonomy*:* A mobile agent is a separate computational unit with its own thread of execution, independent of other units.

Migration: It can suspend its execution at the hosting machine, transfer its current state (data) and resume its execution.

Communication: It can communicate with other agents on the local host or remote machines. [5]

III. PARALLEL COMPUTING

If several operations are performed simultaneously, then the time taken by a computation can be reduced. When a problem to be solved, it is broken into a number of sub-problems. All of these sub-problems are solved simultaneously. The results are combined to produce an answer to the original problem. The results are combined to produce an answer to the original problem. A given problem attempts to divide it into sub-problems which can be solved concurrently on different processors of a parallel computer. [4]

A. Parallel Algorithm Design

(i) Partitioning. A given problem attempts to divide it into sub-problems which can be solved concurrently on different processors of a parallel computer.

(ii) Communication. Communication coordinates task execution and defines appropriate communication structures and algorithms.

(iii) Agglomeration. The tasks and communication structures defined in the first two stages are evaluated with respect to performance, requirements and implementation costs. If necessary, tasks are combined into larger tasks to improve development costs.

(iv) Mapping. Each task is assigned to a processor in a manner that attempt to satisfy the completing goals. [3]

IV. SOLVING SYSTEMS OF LINEAR EQUATIONS

Given an **n x n** matrix **A** and an **n x 1** vector **b**, it is required to solve **Ax=b** for the unknown **n x 1** vector **x**. When **n=4**, for example, we have to solve the following system of linear equations for **x_1, x_2, x_3 and x_4**

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 = b_1,$$
$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 = b_2,$$
$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 = b_3,$$
$$a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 = b_4,$$

A .Iterative Methods

An iterative method to solve the linear system is as shown in equation:

Ax=b

where A= the matrix of coefficient values
x= the matrix of unknown variables
b= the matrix of solution values

It starts with initial approximation **x_0** to the solution x and generates a sequence of vectors **$\{x^k\}$** that converges to **x**. There are many types of methods for solving linear equations such as Gauss Jordan, Jacobi Iterative method, Gauss-Seidel Iterative method and SOR. Among these iterative methods, we will sketch about the Jacobi iterative method, Gauss-Seidel iterative method and Gauss Jordan method .[3]

B. Jacobi Iterative Method

Jacobi method assumes that the diagonal entries of A are all nonzero. Jacobi method requires the duplicate storage for x, since no component can be overwritten until all new values have been computed. Components of new iterate do not depend on each other, so they can be computed simultaneously. Jacobi method does not always converge, but it is guaranteed to converge under conditions that are often satisfied, through convergence rate may be very slow. [2]

```
Procedure Jacobi (old_x,new_x,A,b)
        For i=1 to n
          For j=1 to n
                  If (i!=j) then
                          b[i]= b[i]/a[i,i]
                          new_x[i]=old_x[i]
                          a[i,j]=a[i,j]/a[i,i]
                  end if
          end for
        end for
        Repeat
         For i=1 to n
                  old_x[i]=new_x[i]
                  new_x[i]=b[i]
         End for i
        For i=1 to n
         For i=1 to n
                  If (i!=j)then
                  new_x[i]=new_x[i]- a[i,j]*old_x[j]
                  `End if
         End for
        End for
        Until new_x and old_x converge to each other
```

C. Gauss Jordan Method

A well-known sequential algorithm for this problem is the Gauss-Jordan method. It consists in eliminating all unknowns but x_i from the *i*th equation. The solution is then obtained directly. A direct parallelization of the Gauss-Jordan method is now presented. It is designed to run on a CREW SM SIMD computer with n^2+n processors that can be thought of as being arranged in an n (n+1) array. The algorithm is given as procedure SIMD GAUSS JORDAN. In it we denote b_i by $a_{i,n+1}$. [4]

```
Procedure SIMD GAUSS JORDAN(A, b, x)
Step1:  for j=1 to n do
          for i=1 to n do in parallel
          for k=j to n+1 do in parallel
            if (i!=j) then
               $a_{ik} = a_{ik} - (a_{ij}/a_{jj})\, a_{jk}$
        end if
      end for
   end for
 end for
 Step2:  for i=1 to n do in parallel
               $x_i = a_{i,n+1}/a_{ii}$
         end for
```

D. Gauss-Seidel Method

This method converges for diagonal dominant matrices and symmetric positive definite matrices. It requires nonzero diagonal entries and does not require duplicate storage of x, since component values can be overwritten as they are computed. But each component depends on previous ones, so they must be computed successively. It does not converge, but it is guaranteed to converge under conditions. [4]

```
Procedure MIMD MODIFIED GS (A, x, b ,c)
Step 1: for i=1 to n do
          (1.1) $old_i = x_i^0$
          (1.2) $new_i = x_i^0$
          (1.3) create process i
          end for
```

```
Step 2: Process i
    (2.1) repeat
        (i)$old_i=new_i$
        (ii)$new_i=(b_i$-sum of k=1 to i-1
                ($a_{ik}*old_k$)-sum of k=i+1 to n
                ($a_{ik} * old_k))/a_{ii}$
      until  sum of i=1 to n abs(newi-oldi)<c
    (2.2) $x_i = new_i$
```

V. OVERVIEW DESIGN OF THE SYSTEM

The proposed system performs the parallel computing to solve linear equatiions with the master-worker pattern and the management of the input/output data and separates tasks to the machine based on the input data and user request. The system consists of three types of manager (i)IO Manager (ii) Client Manager (iii) Execution Manager.

(i)IO Manager

When the IO manager receives the input data and request from the users, the IO Manager requests the available client lists from the Client Manager. When the IO Manager accepts the available client list from Client Manager, the IO Manger sends the input data and the task plans according to the parallel techniques such as embarrassingly. IO Manager returns the output to user that receives from Execution Manager.

(ii) Client Manager

The Client Manager checks the available client list from the database to execute the parallel problems over these clients.

(iii)Execution Manager

The Execution Manager receives the data, the task plan and the available client lists from the IO Manager. The Execution Manager partitions the input data by using number of mobile agents and available client. Execution Manager takes the responsibility for initiating tasks to several slave processes. When these agents return after completion of task, the Execution Manager sends the result back to IO Manager. Therefore, Execution Manager acts as Master.

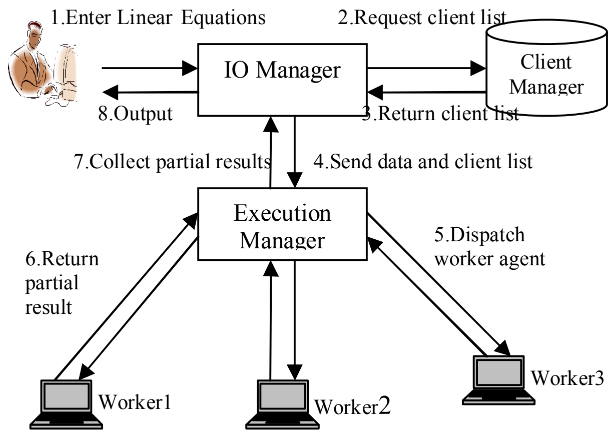


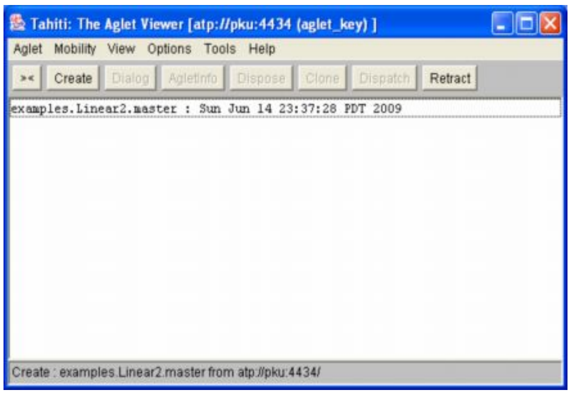
Fig.1: Overview Design of the system

VI. JAVA MOBILE AGENT

The Aglet Technology is a framework for programming mobile network agents in Java. The aglet technology was developed by the IBM Japan research group. From a technical point of view, the IBM's mobile agent called 'Aglet' is a lightweight Java object that can move autonomously from one computer host to another for executing, carrying along its program code and state as well as the so far obtained data. An aglet can exist and execute tasks forever.

An aglet can be dispatched to any remote host that supports the Java Virtual Machine. This requires from the remote host to have preinstalled Tahiti, a tiny aglet server program implemented in Java. A running Tahiti server listens to the host's ports for incoming aglets, captures them, and provides them with an aglet context in which they can run their code from the state that it was halted before they were dispatched. Within its context, an aglet can communicate with other aglets, collect local information and when convenient halt its execution and be dispatched to another host. An aglet can also be cloned or disposed.

The following figure is Tahiti Aglet Viewer. When the user clicks "Create" button, the aglet creates the master agent as follows.

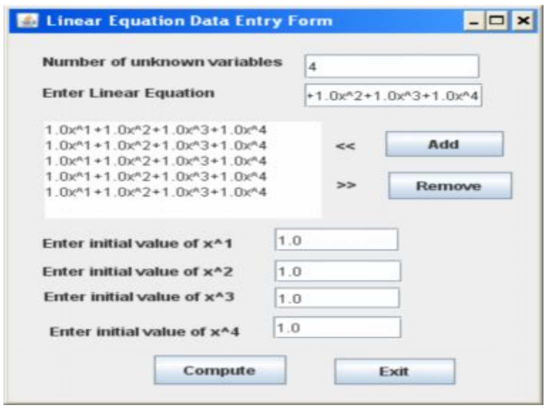

Tahiti: The Aglet Viewer [atp://pku:4434 (aglet_key)]
Aglet Mobility View Options Tools Help
>< Create Dialog AgletInfo Dispose Clone Dispatch Retract

```
examples.Linear2.master : Sun Jun 14 23:37:28 PDT 2009
```

```
Create : examples.Linear2.master from atp://pku:4434/
```

Fig.2: Tahiti Aglet Viewer

The following figure is linear equation data entry form. The user must enter the number of unknown variables, linear equations to solve and initial values. When the user click "Compute" Button, the solving linear equation form is displayed.



Number of unknown variables: 4

Enter Linear Equation: +1.0x^2+1.0x^3+1.0x^4

```
1.0x^1+1.0x^2+1.0x^3+1.0x^4
1.0x^1+1.0x^2+1.0x^3+1.0x^4
1.0x^1+1.0x^2+1.0x^3+1.0x^4
1.0x^1+1.0x^2+1.0x^3+1.0x^4
1.0x^1+1.0x^2+1.0x^3+1.0x^4
```

Enter initial value of x^1: 1.0

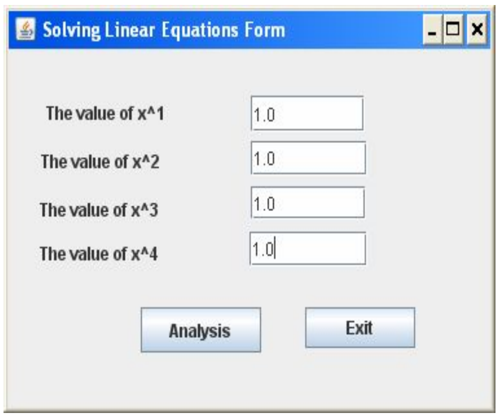
Enter initial value of x^2: 1.0

Enter initial value of x^3: 1.0

Enter initial value of x^4: 1.0

Fig 3: Linear Equation Data Entry Form

Fig.4 is the result form of solving linear equations. The user can see the result values of unknown variables as shown in figure.



The value of x^1: 1.0

The value of x^2: 1.0

The value of x^3: 1.0

The value of x^4: 1.0

Fig.4: Result Form of solving linear equations

When the user clicks the "Analysis" button, the following figure is displayed. This figure is performance evaluation of iterative methods with time in milliseconds.

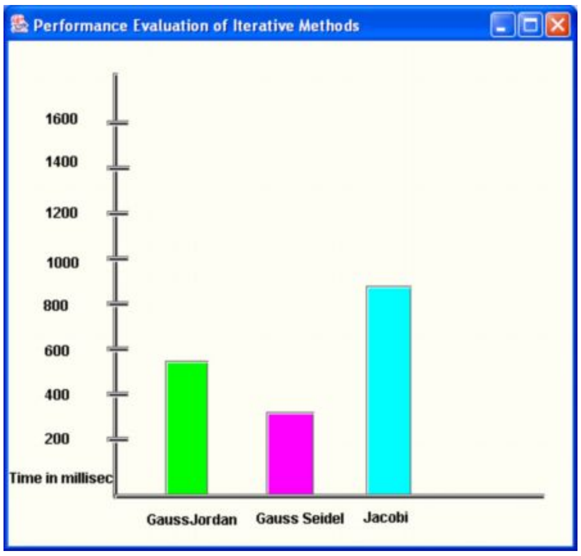


Fig.5: Performance Evaluation of iterative methods form

VII. CONCLUSION

In this system, parallel iterative methods for linear equations have been proposed. The application is implemented with the java mobile agent and the agent carries the code and data to the destination host. The result of computing sends back to stationary host by messaging.

An agent successfully dispatched to the destination hosts performed the parallel computing as expected when tested from Mobile Agent Procedure.

REFERENCES

[1] ERWIN KREYSZIG "Advanced Engineering Mathematics" Eighth Edition Ohio State University,Columbus, Ohio

[2] Myung Kyu Kim, Soo Hoan Chae " A Jacobi Iterative Algorithm for solving Linear System Equations With Large Sparse Matrices on Heterogeneouw Distributed Systems" Hankuk Aviation University, Korea

[3] N.N.Oo, K.M.L.Tun " Parallel Computing in Solving Numerical Problems Using Java Mobile Agent" (paper) Yangon Technological University, Myanmar

[4] Selim G. AkI "The Design and Analysis of Parallel Algorithms" Prentice Hall, Englewood Cliffs, New Jersey 07632, 1989

[5] Z.Z.Wint, K.M.L.Tun "Performance Evaluation of Mobile Agent based Parallel Iterative Methods" (paper) Yangon Technological University, Myanmar