A Distributed Approach for Solving a System of Linear Equations

Dhruv Matani

Computer Engineering, The DJ Sanghvi College of Engineering [Vile Parle (W), Mumbai, India]
53 Mint Road, 2nd Floor, Fort, Mumbai, Maharashtra 400001, India, dhruvbird@yahoo.com

Abstract: This paper discusses an algorithm and its implementation for solving a system of linear equations on a distributed system. The algorithm used is The Cramer's rule for solving a large number of linear equations simultaneously. Hence, a pivotal part of this algorithm is solving a determinant of a square matrix in a distributed manner. This topic of determinant solving is discussed in detail, followed by some results of running the program with different number of computers on the network. [The Journal of American Science. 2005;1(2):1-8].

Key Words: Determinant; Distributed System; Distributed Algorithm; Linear equations

Contents

1 Introduction

A system of linear equations has many applications in the fields of chemistry, physics, mathematics, and also in our day to day lives. Solving slightly non-trivial problems become very easy if we can express them as a system of linear equations, and solve them simultaneously. They provide just the abstraction that the human mind needs to solve the original problem without taking into account other unrelated data.

We describe in this paper a method for solving a large number of such simultaneous linear equations quickly and efficiently on a distributed system. The rationale for choosing a distributed system lies in the fact that they are very cheap to build, because a large number of inexpensive computers connected by a high-speed network can form a very powerful distributed network.

We first describe a distributed approach for solving the determinant of a square matrix, and then apply Cramer's rule to solve the system of linear equations simultaneously. We shall show how these two tasks are very much inter-related, and closely coupled for the correct functioning of the final algorithm.

We have implemented this program on a Linux based system, and have measured the timings for runs of the program for solving the same number of equations, but with a different number of clients connected to the main server.

2 Practical applications of solving a system of linear equation.

(1) Digital signal processing (**DSP**) is the study of signals in a digital representation and the processing methods of these signals. DSP and analog signal processing are subsets of signal processing. It has three major subfields: audio signal processing, digital image processing and speech processing.

In DSP, engineers most commonly study digital signals in one of the following domains: time domain (one-dimensional signals), spatial domain (multidimensional signals), frequency domain, autocorrelation domain, and wavelet domains. They choose the domain in which to process a signal by making an educated guess (or trying out different possibilities) as to which domain best represents the

essential characteristics of the signal. A sequence of samples from a measuring device produces a time or spatial domain representation, whereas a discrete Fourier transform produces the frequency domain information. The autocorrelation is, loosely speaking, defined as the expected value of correlation of the signal with itself on some distance in time or spatial distance. http://www.fact-index.com/d/di/digital_signal_processing.html

(2) Linear programming is the process of solving a system of linear equalities and linear inequalities over a set of unknown real variables, along with a linear objective function to be maximized.
Many practical problems in operations research can be expressed as linear programming problems. For instance, if x1 is the number of acres planted with wheat and x2 is the number planted with corn, and a farmer has a limited number of acres A, and has a limited permissible amount F of fertilizer and P of insecticide which can be used, each of which is required in different amounts per acre (F1, F2, P1, P2) for wheat and corn respectively, and knows the selling price of wheat S1 and the selling price of corn S2, then the optimal number of acres to plant with wheat vs corn can be expressed as a linear program. http://www.fact-index.com/l/li/linear_programming.html.

(3) Numerical analysis is that branch of applied mathematics, which studies the methods and algorithms to find (approximate) numerical solutions to various mathematical problems, using a finite sequence of arithmetic and logical operations. Most solutions of numerical problems build on the theory of linear algebra. http://www.fact-index.com/n/nu/numerical_analysis_1.html

(4) Nonlinear Least Squares Curve Fitting: Simple linear curve fitting deals with functions that are linear in the parameters, even though they may be nonlinear in the variables. For example, a parabola *y=a+b*x+c*x*x* is a nonlinear function of x (because of the *x*-squared term), but fitting a parabola to a set of data is a relatively simple linear curve-fitting problem because the parameters enter into the formula as simple multipliers of terms that are added together. Another example of a linear curve-fitting problem is *y= a+b*Log(x)+c/x*; the terms involve nonlinear functions of the independent variable *x*, but the parameters enter into the formula in a simple, linear way. http://members.aol.com/johnp71/nonlin.html.

3 Existing Solutions

Many solutions currently exist for solving such a large number of simultaneous linear equations. Many of them are iterative methods like the Gauss-Jordan iterative method. Others include the Gauss elimination, and matrix-methods which involve manipulation of the contents of a matrix to get the final result. The matrix methods including the Gauss elimination method essentially use the following relation:

$$\boldsymbol{Ax = B}$$

where,
A --> The coefficient matrix.
x --> The list of variables.
B --> The column matrix representing the RHS of the equations.

The **Gauss-Jordan** method, which is an iterative method uses the values of the unknowns computed at the previous step in its next step while refining the values. thus, there is a dependence of one step on the previous step making this method not very attractive for implementing on a distributed system.

The **Gauss elimination** method involves converting the original matrix into an upper or lower triangular matrix, and then computing the individual values by re-substituting them into the resulting equations. This method too suffers from the defect that each step is dependent on the previous step. there is one more dependency introduced in the fact that the same operations must be performed on the column matrix on the other side as is being performed on the original matrix. Thus, making it unsuitable for our purposes. [MFA2001]

The **matrix-inversion** method again involves converting the original matrix to a unit matrix, and suffers from the same defects as the Gauss-elimination method, making it unsuitable for our purposes.

4 A distributed approach for solving a system of linear equations

In our distributed approach, we use the Cramer's rule for solving the system of linear equations. Cramer's rule is defined as:

Given a system of 'n' linear equations in 'n' distinct unknowns represented as:

$a_1x + a_2y + ... + a_nz = k_1$
$b_1x + b_2y + ... + b_nz = k_2$
$c_1x + c_2y + ... + c_nz = k_3$

We can find the unknowns x, y, ... , z using the following formulae:
Let

Δ = **det**(coefficient matrix)
Δ_x = **det**(coefficient matrix, with the first column replaced by RHS column matrix)
Δ_y = **det**(coefficient matrix, with the second column replaced by RHS column matrix)
and so on...
Δ_z = **det**(coefficient matrix, with the n^{th} column replaced by RHS column matrix)
Thus,
$x = \Delta_x / \Delta$
$y = \Delta_y / \Delta$
$z = \Delta_z / \Delta$

As you can clearly see, the process of calculating the values Δ, Δ_x, Δ_y, ..., Δ_z is independent of any other process, and also each of the above values can be calculated independently of each other. So, this method seems to be very attractive for implementing on a distributed system, since it involves a series of steps which can be performed independently of each other. So, in our approach, we use the Cramer's rule for solving the system of linear equations simultaneously.

As you might have noticed, now that we know that each of the above operations can be performed independently of each other the fact still remains that the procedure for calculating the determinant of a square matrix is a very time consuming process, and should be done efficiently for the efficient operation of our algorithm. We describe in detail in the following section the procedure for calculating the determinant of a square matrix. Also described is the algorithm for calculating the same in a distributed environment. [DAB2002] [LN012003].

5 A distributed approach for finding the determinant of a square matrix

The process of calculating the determinant of a square matrix is a recursive one, as we shall show shortly. Thus, it should be possible to break up the process into distinct independent units, each operating in isolation from the others.
Consider a 1 X 1 matrix given by:
a
The determinant of this matrix is trivial to find, and is the value 'a' itself.
Now, consider a 2 X 2 matrix given by:
a b
c d
The determinant of the above 2 X 2 matrix is given by the formula:
a(d) - b(c)
Now, consider a 3 X 3 matrix given by:
a b c
d e f
g h i

To calculate the determinant of the above matrix, we use the formula:
a(e(h) - f(h)) - b(d(i) - f(g)) + c(d(h) - e(g))

As you can see, the above step of calculating the determinant of a 3 X 3 matrix involves calculating the determinant of a 2 X 2 matrix 3 times. And, the process of calculating the determinant of a 2 X 2 matrix involves calculating the determinant of a 1 X 1 matrix 2 times.

Generalizing, we conclude that the process of calculating the determinant of an n X n matrix involves calculating the determinant of an (n-1) X (n-1) matrix 'n' times. This shows us that the process of calculating the determinant of a square matrix is in fact a recursive process.

Now, coming to the point of calculating the determinant of a square matrix in a distributed environment, we make the following assumptions:

1. The computer containing the n X n matrix whose determinant is to be calculated is called the 'master'.

2. The 'master' is connected to at least 'n' client machines by means of a high speed network connection. Each of these 'n' clients is capable of calculating the determinant of a square matrix.

3. Each of these 'n' clients may be connected to any number of sub-clients, which may in turn be connected to sub-sub-clients, and so on and so forth... Thus, the client acts as a server for the sub-client, and this kind of relationship is replicated at each level.

4. It is not necessary for the correctness of the algorithm for the points (1..3) to hold, but they are necessary for showing that the proposed algorithm in such a scenario is very efficient in solving the determinant of an n X n matrix.

Once the above environment is set up, the 'master' starts the process of calculating the determinant of the n X n matrix by delegating to each of the 'n' clients the task of calculating the determinant of each of the 'n' (n-1) X (n-1) matrices. These 'n' clients may in turn delegate part of their work-load to sub-clients connected to them, and so on and so forth... Thus, which each client has finished performing the task that its master assigned it, the master is passed back the result from each of its clients, where it is consolidated and the final result is computed by the master.

We can express the algorithm as:

1. The master delegates the task of calculating the determinant of 'n' (n-1) X (n-1) matrices to each of the 'n' clients connected to itself.

2. Each of the clients in turn delegate the work to clients connected to them, and this continues till there is no client with a client connected to it. At this stage, the determinant is calculated locally on that machine.

3. These clients at various stages in the tree-hierarchy then pass the computer results up the tree to their respective masters.

4. The masters who are waiting for the computer results accept them, and consolidate the computed determinants of the 'n' sub-matrices, and compute the value of the determinant of the square matrix that they have been assigned.

6 Complexity analysis

The complexity for finding the determinant of an n X n matrix can be expressed by the following recurrence:

O(n) = n + n(O(n-1)) _____(1)

The first 'n' stands for the 'n' multiplication operations performed after each of the 'n' (n-1) X (n-1) determinants have been computed, and the n(O(n-1)) stands for the complexity of calculating the value of the 'n' (n-1) X (n-1) determinants 'n' times.

Solving the above recurrence, we get:

$\mathbf{O(n) = n^n}$ _____(2)

The above complexity represents that for a serial execution of the determinant algorithm on a uni-processor system, or more simply said, on a single machine.

To reduce the complexity, we have decided to use the distributed approach for computing the determinants.

The complexity for finding the determinant of an n X n matrix in a distributed environment is quite different from that in a non-distributed one.

We notice that each of the (n-1) X (n-1) determinants is being computed simultaneously (in parallel) on the various clients that have been assigned the mutually independent tasks of computing the determinants.

Thus, in the eqn.(1), the term **n(O(n-1)** becomes simply **O(n-1)**.

The new recurrence for a distributed system can be given as:

O(n) = n + O(n-1) _____(3)

Solving the above recurrence, we get:

$\mathbf{O(n) = n^2}$ _____(4)

Cramer's rule solves the determinant of an n X n matrix 'n' times, so the first serial(non-distributed) method gives us an asymptotic upper bound of:

O(n) = n x (2)

$\mathbf{O(n) = n \times n^n}$

$\mathbf{O(n) = n^{(n+1)}}$ _____(5)

However, for a distributed system, the asymptotic upper bound is given by:

O(n) = n + (4)

$\mathbf{O(n) = n + n^2}$

$\mathbf{O(n) = n^2}$ _____(6)

Thus, the asymptotic upper bound for solving 'n' linear equations in 'n' unknowns is: $\mathbf{n^2}$.

7 The implementation in detail

The ideas expressed in the document up to now have been implemented in standard C++ in the form of a computer program, which can be compiled and run on a Computer system. This section discusses the implementation in detail, highlighting some of the features and pit-falls of the current implementation.

The implementation is currently split up into 7 logical units.

(1) client

- The client is the part of the code that runs on the client machines, and is responsible for connection to the central server.
- The client accepts requests like **'start'**, **'reset_client'**, **'reset_subclient'**and **'quit'** from the server, and acts on them. See below for a complete list of commands supported by the client, and how each one of them is acted upon by the client.

(2) Server

The server is responsible for polling continuously for new connections from clients. Once a client tries to connect to the server, and the server accepts this connection, the client is said to have been connected to the server, and the server stores the unique ID associated with this connection locally within its memory.

This server also partially manages the task of distributing the work-load of solving the determinant remotely along with the determinant module. It most be noted that the server code is executed on both the server (master) machine as well as the client machines. This is because the client may acts as a server for other sub-clients, etc...

(3) File parser

a. The file parser defines a set of rules for the syntax of equations that can be understood by the parser, and reads them off the input file. These set of rules

are in the form of a CFL (Context Free Language) represented by a set of productions.

b. The rules are given below.

(4) Determinant module

a) The determinant module is the main part of the Equation Cruncher application which is responsible for solving the determinants both remotely and locally.

b) It contains various helper functions to assist itself in the correct and efficient execution of the determinant solving logic.

c) This piece of code makes heavy use of threads, and a lot of the operations being mutually independent can be carried out concurrently. So, having the master as an SMP system would be highly beneficial from the efficiency and speed of execution point of view.

d) This module interacts extensively with both the Socket module as well as the Node manager.

e) It extracts free clients from the Node manager, and instructs them to carry out various operations. A list of operations and how these interactions take place are given below.

f) The interaction of this module with the Socket module is also very large, and there is strong coupling between these two modules, as is the case with this module and the Node manager.

(5) Node manager

a) The Node manager is responsible for effectively and correctly maintaining the list of all clients currently connected to the master after the server has recognized such clients and has accepted their connections.

This module contains functions that send matrices across the network to clients, and on the client side, another set of functions receive the data, and re-create the matrix in its original form from the extra data provided in the command header. A description of the packer header is given below.

b) It contains atomic operations for getting a client socket, and replenishing it.

c) This piece of code is tightly coupled with the server code.

(5) Socket manager

a) This contains basic wrapper functions around standard UNIX socket functions, and implements basic error checking.

b) It throws exceptions if there is any error in the execution of the underlying functions.

c) The list of exceptions is given below.

(6) Synchronization manager

a) This is just a thin wrapper around the POSIX thread API, and provides a solitary Mutex class which allows programs to have mutually exclusive access to any piece of shared data.

b) It throws exceptions if there is any error in the execution of the underlying functions.

c) The list of exceptions is given below.

(7)Synchronization manager

a) This is just a thin wrapper around the POSIX thread API, and provides a solitary Mutex class which allows programs to have mutually exclusive access to any piece of shared data.

You can download the current version of the program named “Equation Cruncher” at: http://www.geocities.com/dhruvbird/eqnc/.

List of commands accepted by the client, and actions taken when these commands are received

Command	Action taken
start	When the client receives a **start** command, it first receives the complete matrix from its server, and then starts evaluating the determinant of that square matrix received. When the determinant calculation is complete, it sends the server a **result** command in return along with the actual result.
reset_client	When the client receives a **reset_client** command, it performs 2 main functions. Firstly, it sends all its clients a **reset_subclient** command, and secondly it disconnects itself from its server by closing the open socket, and then starts waiting for a connection once again on the same IP address and port as

	before.
reset_subclient	When the SubClients receive a **reset_subclient** command, they just display the appropriate message to the user, and propagate the message to their clients.
quit	When the client receives a **quit** command, it closes the connection with all its clients, and its server, and gracefully exits.

Set of productions defining the Context Free Language (syntax) of the equations appearing in the input file.

1. **EQN --> LHS = RHS**
2. **DOUBLE -->(0..9)* | (0..9)*.(0..9)$^+$**
3. **RHS --> (+ | - | e)DOUBLE**
4. **LHS --> LHS(+ | -)TERM | TERM**
5. **TERM --> (+ | - | e)DOUBLE_VAR**
6. VAR --> (a..z)$^+$

Structure of the Packet Header

```
struct pkt_header
  {
    int size; // Amount of data after the packer header gets over.
    int type; // Type of packet: { data, command }
    int code; // Command issued: { quit, start, result, reset_client }
    double reserved; // Reserved for the implementation to use as it feels right.
  };
```

List of exceptions thrown by the Socket module

Exception Name	Thrown When?	Public Base Classes
eqn_cruncher::socket_error	Any general socket error while creating the socket, connecting to the server, listening for connections, accepting connections, sending/receiving data.	**std::runtime_exception**
eqn_cruncher::host_error	Host name lookup error.	**eqn_cruncher::socket_error**
eqn_cruncher::port_error	Error binding to the specified port number.	**eqn_cruncher::socket_error**

List of exceptions thrown by the File parser

Exception Name	Thrown When?	Public Base Classes
eqn_cruncher::file_error	Input file does not exist.	**std::runtime_exception**
eqn_cruncher::parse_error	Error encountered during parsing the input file. This error is a syntax error encountered during scanning/reading the input file data.dat. This exception suggests that the format of the equations is not correct, and does not adhere to the list of productions given above.	**std::runtime_exception**

8 Results of tests conducted measuring the improvements in time required for solving the system of linear equations

The measurements were taken on systems running Mandrake Linux - 9. They had a hardware configuration of Intel Celeron 2400MHz CPU with 128MB RAM. The various machines were connected to each other by a 10/100MBPs fast ethernet connection. In all, there were 9 systems connected to each other, with one acting as the server, and the other 9 as clients.

We solved 10 simultaneous linear equations in 10 unknowns on these systems increasing the number of clients each time from 0(zero) right up to 8(eight). The results are given below in a tabular format, followed by a graphical representation.

As you can see, the time taken to solve the equations reduces as the number of clients connected to the server increases. Thus, showing that the implementation is fairly scalable, and will perform well under various loads with many clients connected to the server(master) machine.

Also shown below is the tree-hierarchy structure for the arrangement of clients writ the server (master) machine that will result in optimal performance.

Number of clients connected to the server.	Time taken (in seconds) to solve 10 equations.
0	340.525
1	132.95
2	80.2574
3	37.3861
4	20.8911
5	13.2871
6	9.9505
7	7.56436
8	5.38614

9 Future work

These are the areas which we shall be concentrating on in the near future. Any help in these or other related areas is greatly welcomed, and will be highly appreciated.

(1) Optimizations:

- **Network speed test:** This should test the speed between each client, and the server by sending ping like packets.
- **Client speed(CPU speed) test:** This should test the speed of the client in terms of CPU speed.
- **Client memory(RAM) capacity test:** This should test whether the client machine has enough memory to handle the request being sent to it without making sufficient use of it's swap file.

(2) Load Balancing:

This algorithm should balance the load between the various clients by relinquishing some unused clients in some server to some other client who needs them(and their processing speed) to distribute its work-load.

(3) Fault Tolerance: (Done!)

- If some client goes down while evaluating the determinant, then the server should immediately recognize this situation, and take remedial steps such as sending the request to another client if there exists a free client or queuing up the request for a client as soon as some client gets free, or to evaluate the determinant itself.

(4) Security:

- If some spurious application tries to log on to the network, and pretend to be an actual client, we have no way of knowing this fact! This was pointed out to me by Sandesh Singh (sandesh247@gmail.com). If such a situation arises, then the application can get badly mistaken, and can produce wrong results if the spurious application sends back junk data. We need some method by which the clients are authenticated before they are allowed to establish a connection with the server, and get problems to solve.

10 Conclusion

We have seen the implementation of an algorithm which solves a system of linear equations on distributed systems. The test results show that it is more efficient compared to the serial algorithm, which executes on a single machine. We have also computed the asymptotic upper bound for both the approaches, and have seen that given a greater number of processors, the asymptotic upper bound for out distributed algorithm tends towards $\mathbf{n^2}$, and compared to $\mathbf{n^n}$ for the serial algorithm.

We have also seen the timings for solving 9 equations in 9 unknowns on a distributed system, each containing a different number of clients. We observe that as the number of clients increases, the speed of the

algorithm also increases, thus showing that the algorithm is scalable. We would have liked to perform the tests with an SMP/server system as the master, but such a computer was not available at the time of testing.

Future work in the areas of optimization, Load balancing, and creating a Fault tolerant system will be encouraged, and greatly appreciated. We hope to accomplish the points mentioned in (9) in the year to follow.

There is some scope for improving the algorithm of finding the determinant, and improvements which use properties of determinants to eliminate redundant calculations will be encouraged, and implemented. For example, one optimization already in place is that if the element with which the determinant of a sub-matrix is to be multiplied is 0, then the value of the determinant is not calculated, thus saving a lot of time. This will prove highly beneficial for sparse matrices. Another area that has been explored is cutting down the number of iterations, and network data flow. When we wish to calculate the determinant of a 3 X 3 matrix or a square matrix of a smaller dimension, then the formula is used directly instead of using the recursive function.

We would also like to explore the possibility of using a grid based approach instead of the current tree-like hierarchy based approach, since we believe that it would alleviate the problem of load balancing to some extent.

11 Acknowledgments

The author would like to thank Siddharth Shah for introducing him to the world of clusters, grids, and helping him out with the basics of Networking in computers, without which this project would never have been possible. This project was actually made because Siddharth wanted to do something related to networking, and I wanted to do something related to Algorithms. So, we reached a compromise, and this baby was born! I would also like to thank Apurva Mehta, who helped me test the initial client-server network model, which I had implemented in C++. Thanks to Aniket for proof reading this document, reporting the mistakes and helping me explain certain concepts in a better manner. My mom needs a special mention, because she was the one who reminded me that there existed something called food while I was busy thinking about the whole concept and later implementing the ideas.

We would like to pay our tributes to Dr. Cramer, without whom this project would never have been possible!

12 References

[1] [MFA2001] Mark F. Adams. A Distributed Memory Unstructured Gauss-Seidel Algorithm for Multigrid Smoothers. August 17, 2001.

[2] [DAB2002] David A Bader, Bernard M.E. Moret, and Peter Sanders. Algorithm Engineering for Parallel Computation.

[3] [LN012003] Lecture notes. Csanky's Algorithm: matrix determinant € NC^2. 10/14/2003.