Parallel Methods to solving Linear Equations Systems

Moreno Vera Felipe Adrian

20/08/2018

Abstract

In this work, I introduce different methods to solve a linear equations systems. Nowadays, exist a lot of methods to solve linear systems e.g. Gauss Elimination or Matrix decomposition methods like LU, QR and iterative methods and derivate from those. Matrix operations is commonly used to determinate the flops in a computer. SEL is used too in image processing and neuronal networks to get eigenvalues and eigenvectors.

1 Introduction

In this work, we present different methods to solve SEL and their parallelize form, we will compare the runtime of each method. For each method, we shows the algorithm and applications in Linear Algebra.

2 Methods and algorithms

Linear Systems can be solved using direct methods that operates over the matrix in each step or iterative methods that find a best solution for the system based on error calculation and convergence of the entry vector.

2.1 Direct Methods

Direct Methods consist in multiply to the original matrix A by Permutation Matrix and Operation Matrix in each step of the algorithms below.

2.1.1 Gauss Elimination

First, we present Gauss Elimination method, this method also can be used to determinate a determinant, the rank, the inverse of an invertible square matrix. The Gauss Elimination is adding a column b to the square matrix A, so we have the new matrix (nx(n+1)) Ab.

Gauss Elimination requires $\frac{n(n+1)}{2}$ divisions, $\frac{2n^3+3n^2-5n}{6}$ multiplications and $\frac{2n^3+3n^2-5n}{6}$ subtractions. its approximately $\frac{2n^3}{3}$. So, using Big-O notation, Gauss Elimination have an order $O(n^3)$.

Using row operations to convert a matrix into reduced row echelon form is sometimes called GaussJordan elimination. Some authors use the term Gaussian elimination to refer to the process until it has reached its upper triangular, or (unreduced) row echelon form.

For computational reasons, when solving systems of linear equations, it is sometimes preferable to stop row operations before the matrix is completely reduced. Because it takes a lot of extra operations.

```
Function A; nxn+1
for $i$ = $0$ to $n$ do
    // Search the maximum in this column (called infinity norm).
    maxEl = $|A_{i,i}|$
    maxRow = i
    for $k$ = $i+1$ to $n$ do
        if $|A_{k,i}| > maxEl$ then
            maxEl = $|A_{k,i}|$
            maxRow = k
        end
    end
    // Swap maximum row with current row
    for $k$ = $i$ to $n$ do
        tmp = $A_{maxRow,k}$
        $A_{maxRow,k} = A_{i,k}$
        $A_{i,k}$ = tmp
    end
    // Make all rows below this one 0 in current column
    for $k$ = $i+1$ to $n$ do
        C[i] = C[i] + C[i-1]
    end
end
// solve the equation X
X = 0
for $i$ = $n$ to $1$ do
    $X_i = \frac{A_{i,n+1}}{A_{i,i}}$
    for $k$ = $i-1$ to $1$ do
        $A_{k,n+1} - = A_{k,i}$
    end
end
return X
```

Algorithm 1: Gauss Elimination

Gauss Elimination creates Permutations Matrix and Operations Matrix called P and Q respectively in each iteration, these matrix can be used to transform a matrix A to U which is an upper triangular matrix. From these, methods called LU and QR was created.

2.1.2 LU Decomposition

LU decomposition factors a matrix as the product of a lower triangular matrix (L) and an upper triangular matrix (U). The product sometimes includes a permutation matrix as well. The LU decomposition can be viewed as the matrix form of Gaussian elimination.

```
Function A; nxn; b; x; n
Define a P vector that store all row exchange
for $i$ = $0$ to $n$ do
    // Search the maximum in this column (called infinity norm).
    maxA = 0
    imax = i
    for $k$ = $i$ to $n$ do
        if $|A_{k,i}| > maxEl$ then
            maxEl = $|A_{k,i}|$
            imax = k
        end
    end
    // Swap maximum row with current row
    if imax not i then
        swap(P[i], P[imax])
        P[n] = P[n] + 1
    end
    // Make all rows below this one 0 in current column
    for $j$ = $i+1$ to $n$ do
        $A_{j,i} = \frac{A_{j,i}}{A_{i,i}}$ for $k$=$i+1$ to $n$ do
            $A_{j,k} = A_{j,k} - A_{j,i} * A_{i,k}$
        end
    end
end
// solve the equation X
for $i$ = $0$ to $n$ do
    x[i]= b[P[i]]
    for $k$=$0$ to $i$ do
        x[i] = x[i] - $A_{i,k}$*x[k]
    end
end
for $i$ = $n-1$ to $0$ do
    for $k$=$i+1$ to $n$ do
        x[i] = x[i] - $A_{i,k}$*x[k]
    end
    x[i] = $\frac{x[i]}{A_{i,i}}$
end
return X
```

Algorithm 2: LU Decomposition

2.2 Iterative Methods

Iterative Methods consist in find the best solution or the most nearest solution in a System, you should choose a good entry vector to converge fast to the solution.

2.2.1 Jacobi Method

Jacobi method determine the solutions of a diagonally dominant system of linear equations. Each diagonal element is solved for, and an approximate value is plugged in. The process is then iterated until it converges.

Jacobi method requires about $O(kn^2)$ operations, where k is the number of iterations until err of the norm decrease to Error require.

```
Function Initial $x^0$, diagonal dominant matrix A; vector b, Error to be reached
k = 1
while $err > Error$ do
    $x^k = x^{x-1}$
    for $i = 0$ to $n$ do
        $xx^t = < A_{i,:} x^{x-1} >$
        $x^k[i] = \frac{(b[i]-xx^t)}{A_{i,i}} + x^{k-1}[i]$
    end
    k++
    err = $||x^k - x^{k-1}||$
end
return X
```

Algorithm 3: Jacobi Method

2.2.2 Gauss-Seidel Method

Also known as the Liebmann method or the method of successive displacement, Gauss-Seidel is used to solve a linear system of equations iteratively. Is similar to the Jacobi method but this method do the operations in the same vector x^{k+1} in each step instead of Jacobi that operate on the vector x^k and then it will be use to calculate x^{k-1}, with this technique Gauss-Seidel converge more faster than Jacobi method.

Though it can be applied to any matrix with non-zero elements on the diagonals, convergence is only guaranteed if the matrix is either diagonally dominant, or symmetric and positive definite.

Gauss-Seidel method requires about $O(kn^2)$ operations, where k is the number of iterations until err of the norm decrease to Error require.

```
**Function** Initial $x^0$, diagonal dominant matrix A; vector b, Error to be reached
k = 1
**while** $err > Error$ **do**
    $x^k = x^{x-1}$
    **for** $i = 0$ *to* $n$ **do**
        $xx^t = < A_{i,:} x^k >$
        $x^k[i] = \frac{(b[i]-xx^t)}{A_{i,i}} + x^k[i]$
    **end**
    k++
    err = $||x^k - x^{k-1}||$
**end**
**return** X
```

Algorithm 4: Gauss-Seidel Method

3 Results

We use OpenMP to parallelize all the previous algorithms, we measure the serial time, the parallel time, the efficiency and the speedup. We note that the elimination of Gauss is the worst of them, the execution time for different data grows polynomially.

In parallel mode, Gauss present the most significant optimization and reduction in runtime than the others.

We test with 1000, 2000, 3000 and 4000 and as you see, the time is growing as factor of n^3 in runtime.

Method	Serial time	Parallel time (n=2)	Parallel time (n=4)	Parallel time (n=8)
Gauss-Seidel	0.0729176	0.0670215	0.0362759	0.0152195
Jacobi	0.366089	0.358308	0.370612	0.0151156
Gauss	68.8792	34.4715	17.9784	17.0431
LU	71.7346	37.4299	19.5779	19.1613

Table 1: Execution time for 4000 items

Method	Serial time	Parallel time (n=2)	Parallel time (n=4)	Parallel time (n=8)
Gauss-Seidel	0.0411324	0.0410854	0.0239424	0.015187
Jacobi	0.208692	0.12308	0.0227526	0.00865889
Gauss	29.7034	14.9793	7.95358	7.45857
LU	30.4849	16.2177	8.57333	8.43204

Table 2: Execution time for 3000 items

Method	Serial time	Parallel time (n=2)	Parallel time (n=4)	Parallel time (n=8)
Gauss-Seidel	0.0184082	0.0109862	0.0080304	0.00389579
Jacobi	0.103014	0.100505	0.0680329	0.00385363
Gauss	8.78098	4.61284	2.30994	2.20196
LU	9.49935	4.97314	2.69314	2.48058

Table 3: Execution time for 2000 items

Method	Serial time	Parallel time (n=2)	Parallel time (n=4)	Parallel time (n=8)
Gauss-Seidel	0.00721745	0.00528055	0.0037653	0.00122341
Jacobi	0.0319083	0.059547	0.0340023	0.00098862
Gauss	1.16946	0.645061	0.321534	0.284147
LU	1.25657	0.622776	0.354914	0.318724

Table 4: Execution time for 1000 items

4 Conclusions

Depends of the matrix entry and the context and the kind of solution that we want (application) we could use different methods to solve a linear system, but some methods takes more computational costs than other also in parallel mode.

So, before to use which method could be the best, we need to know our data contidion and the behavior of the data (in this case matrix and vectors) when we apply multiplications o exchanges rows-columns.

Parallel programming help us to optimized and reduce runtime in algorithms that requires a high computational cost like Gauss-Elimination, but in other cases like Iterative methods, parallel algorithms optimized.

As we see, Gauss-Seidel is the best method to solve a linear system more faster than other because it has a strong way to converge to the real solution.