CS 359 PROJECT REPORT

PARALLEL IMPLEMENTATION OF CONJUGATE GRADIENT METHOD

ASHISH ARUN GAWAI 160001009

RISHABH VERMA 160001049

INTRODUCTION

To solve different kind of problems .like to solve large linear algebraic equation as well as to solve optimization problem .But parallel implementation of this method mainly are not suitable for physical prohe conjugate gradient method is one of popular method generally applied to blem because of convergence rate of such kind of problems strongly depends on coefficient. As we know for symmetric and positive definite matrices there are a lot of parallel implementation .But if problem is not positive definite and symmetric then we are applying few method so that we can do parallel implementation easily

REFERENCES

1. A.Jordan R.P. Bycul "The parallel algorithn of conjugate gradien method " lecture notes on computer sciences vol 2326
2. M.Chikomaski "Gradient parallel method and evolutionary method of supercomputer SR-2201
3. IEEE Explorer

ALGORITHMS

Data Input: Matrix A which is coefficient of linear equation and vector b which is constant part of linear equation.

Matrix A will be stored in form of 2-D array and vector b will be stored in form of 1-D array in-order to solve

Data Output: In form of 1-D array representing the value of x

The algorithm is detailed below for solving Ax = b where A is a real, symmetric, positive-definite matrix. The input vector x_0 can be an approximate initial solution or 0. It is a different formulation of the exact procedure described above.

$$
\begin{aligned}
&\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0 \\
&\mathbf{p}_0 := \mathbf{r}_0 \\
&k := 0 \\
&\text{repeat} \\
&\qquad \alpha_k := \frac{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}{\mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k} \\
&\qquad \mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k\mathbf{p}_k \\
&\qquad \mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k\mathbf{A}\mathbf{p}_k \\
&\qquad \text{if } r_{k+1} \text{ is sufficiently small, then exit loop} \\
&\qquad \beta_k := \frac{\mathbf{r}_{k+1}^\mathsf{T}\mathbf{r}_{k+1}}{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k} \\
&\qquad \mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k\mathbf{p}_k \\
&\qquad k := k + 1 \\
&\text{end repeat} \\
&\text{The result is } \mathbf{x}_{k+1}
\end{aligned}
$$

$$\mathbf{r}_{k+1} = \mathbf{p}_{k+1} - \beta_k\mathbf{p}_k$$

,

$$\alpha_k = \frac{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}{\mathbf{r}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k} = \frac{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}{\mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k} \quad \beta_k = -\frac{\mathbf{r}_{k+1}^\mathsf{T}\mathbf{A}\mathbf{p}_k}{\mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k}$$

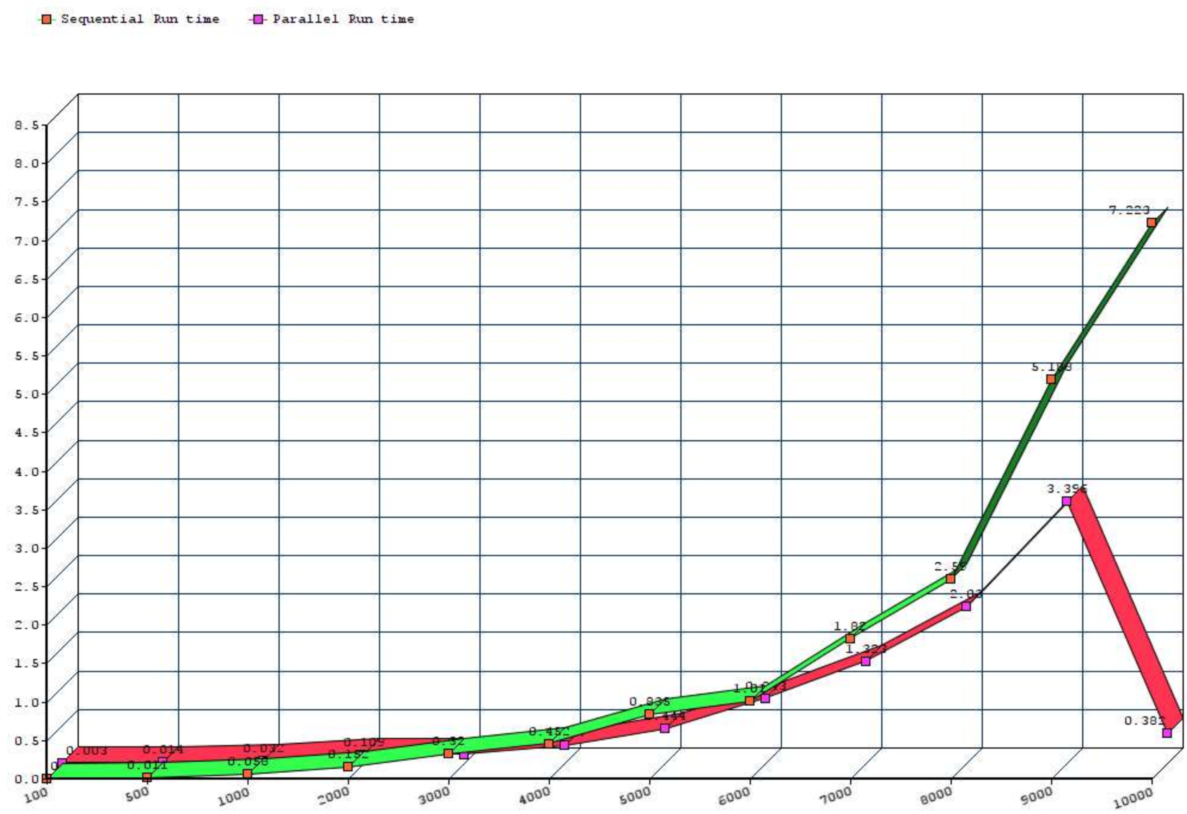
$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{p}_k$$

The parallel implementation of the Conjugate Gradient algorithm has been constructed basing on already verified papers mentioned in references assumption that it is necessary to compute only the most time consuming operations (matrix-vector, transposed matrix-vector multiplications and vector inner product) in parallel to get speedup of the computations.While the main loop will run in sequential .

ANALYSIS BETWEEN RUN TIME OF SEQUENTIAL AND PARALLEL CODE

N	Sequential Code Run Time	Parallel Code Run time
100	0.00000	0.0030
500	0.01100	0.01400
1000	0.05800	0.032000
2000	0.15200	0.10900
3000	0.32000	0.11000
4000	0.45200	0.227000
5000	0.83500	0.44400
6000	1.01000	0.843000
7000	1.81200	1.32300
8000	2.59000	2.03300
9000	5.18800	3.39600
10000	7.22300	4.38200

GRAPH REPRESENTATION BETWEEN SEQUENTIAL AND PARALLEL RUN TIME



GRAPH REPRESENTATION BETWEEN SPEED UP VS N

Theoretical Analysis

Using n processor

Sequential Time complexity - $O(n) + O(n) + O(q * (n + n^2)$

Parallel Time Complexity - $O(1) + O(q * (n * \log(n) + \log(n))$

Theoretical Speed Up - $O(n/\log(n)$

Sequential Cost - $O(q * n^2)$

Parallel Cost – $O(q * n * \log(n) * n)$

Experimental Analysis

Using 4 threads

Average Speeds up obtained: 1.61

Conclusion:

- Maximum speed up that can be obtained is order of n/log(n) and the above implementation will be can made cost optimal using processors equals to n/log(n)
- As output of next iteration is dependent on previous thus, it is not made parallel completely, even thought we can achieve reasonable speed up by doing its sub steps of one iteration in parallel.