**PARALLEL IMPLEMENTATION OF THE ITERATIVE CONJUGATE GRADIENT METHOD**

**LLOYD MAWUNYEGA DZOKOTO**

**GLOBAL SOFTWARE DEVELOPMENT , HOCHSCHULE FULDA**

**TABLE OF CONTENTS**

1. **ABSTRACT**
2. **GLOSSARY**
3. **INTRODUCTION**
4. **CONJUGATE GRADIENT METHOD**
5. **PARALLELIZING THE CONJUGATE GRADIENT METHOD**
6. **EXPERIMENTAL RESULTS**
7. **SUMMARY**
8. **REFERENCES**
9. **SOURCE CODES**

**1. Abstract**

The objective of this project is to implement using C programming language both the serial and parallel Conjugate Gradient Method. The parallel variant of CG Method would be implemented using Message Passing Interface (MPI) and an experiment will be conducted to compare the performance of the serial to the parallel Conjugate Gradient (CG) Method.

**Introduction**

Cg?

Why paralelize?

Why mpi?

**4. Conjugate Gradient (CG) Method**

The CG method is widely considered by a large community of scientists worldwide as the most popular iterative method for solving large systems of numerical linear equations of the form

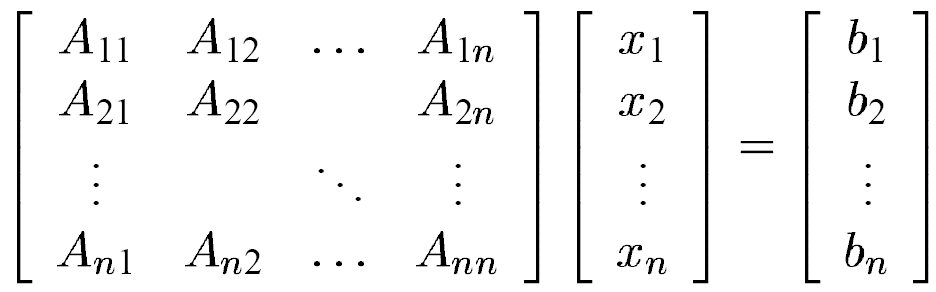
AX = B (1)

Where:

A is a known square , symmetric , positive-definite (or positive-indefinite) matrix.

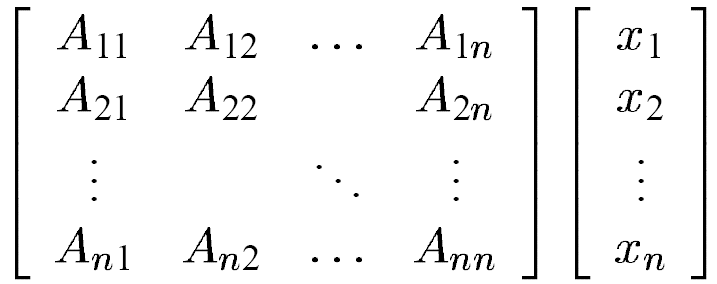
X is an unknown column vector.

B is a known column vector.



A positive-definite matrix for nonzero vector X is defined as:

XT AX > 0 (2)



*>0*

[ *x1 x2 … xn* ]

Most of the iterative algorithms require a stopping criteria to stop the algorithm from executing indefinitely. The vector X then approximates the solution at this point. However, the CG method terminates after n iterations and the vector X is then the approximate solution.

**4.1 Procedure**

The CG method begins with an initial guess of the solution vector *x*0

The residual r0 = B - A *x*0 is computed.

A gradient αi is computed at each iterate which is used to improve better estimates.

The method iterates to determine better successive estimates of the solution *x*1, *x*2, *x*3… using αi .

An improved residual is computed using αi .

An improved direction towards the solution is determined by computing a new gradient βi .

**4.2 Algorithm**

* x0 = 0, r0 = b, p0 = r0
* **for** k = 0, 1, 2, 3, . . .n
* αk = (rTkrk) / (pTkApk) step length
* xk+1 = xk + αk pk approximate solution
* rk+1 = rk – αk Apk residual
* βk = (rTk+1 rk+1) / (rTkrk) improvement
* pk +1 = rk+1 + βk pk search direction
* **end**