Final Project Report
Transient Stability of Power System (Programming Massively Parallel Graphics Multiprocessors Using CUDA)

Amirhassan Asgari Kamiabad, Student Number: 996620802

***Abstract*—Electric power system is large interconnected network of generators, transmission and distribution facilities and electrical loads. Such a huge system is prone to many kinds of disturbances which may lead to undesirable effects on the network, such as blackouts or loss of synchronism in generators. Transient stability analysis examines the dynamic behavior of a power system for as much as several seconds following a power disturbance. Computational complexity of transient stability analysis have kept them from being run in real-time to support decision making at the time of a disturbance and prevent cascading failures. Parallel processing is a promising technology for the speed up of the dynamic simulation required in transient stability. This project presents the transient stability analysis performed on the graphics multiprocessors as an emerging general purpose parallel platform.**

The analysis involves solution of extremely large systems of differential and algebraic equations. Various integration techniques are available to transform the differential equations to non-differential, non linear system of equations. However, the core of the resulting nonlinear equations from any integration schemes is the solution of a large sparse linear system. It is proven that direct methods (e.g. LU decomposition) have a poor performance on the parallel platforms, especially massively parallel structures such as GPU; therefore, Conjugate Gradient method is implemented for the linear solver. The fact that CG methods involve just matrix addition and multiplication propose speed up anticipation for the whole process.

In order to improve the convergence rate of the conjugate gradient method, Chebychev Polynomial Preconditionner is implemented on GPU. The preconditionner effects the condition number of jacobian matrix and reduce the total number of CG iterations. Number of iterations before and after preconditioning, execution time and speed up for CG and preconditionner process is reported.

***Index Terms*—Transient stability analysis, Differential algebraic equation, Iterative linear solver, General purpose graphics processor.**

I. Introduction

Traditionally, power system transient stability analysis has been performed off-line to understand the system's ability to withstand specific disturbance and the systems response characteristics as system returns to normal operation. To date, the computational complexity of transient stability problems have kept them from being run in real-time to support decision making at the time of disturbance. In order to improve the performance of the program either faster hardware or more efficient algorithms can be exploited.

Since most of the previous attempts in order to improve the efficiency of underlying algorithms were focused on sequential methods, direct approaches have been studied tremendously in this field. However, it is proven that significant portions of the direct approaches do not show apparent parallelism. Krylof subspace methods are a class of iterative linear solvers which form an orthogonal basis of the sequence of successive matrix powers times the initial residual (the Krylov sequence) and approximate the solution by minimizing the residual over the subspace formed. [1] provides simple explanation of the principle Krylov subspace methods. The robustness and efficiency of these methods has attracted lots of attentions during past few years. [2] to [7] are similar attempts to use Krylof methods in order to perform power system analysis on parallel platforms. In [8], Gopal, Niebur, et al. use Cg language to perform power system analysis based on Gauss Jacobi method on graphic cards.

This project implements conjugate gradient method on GPU using CUDA to solve the sparse linear system resulted from Newton linearization. It also computes a preconditionner matrix on GPU to improve the conjugate gradient efficiency. This process is evaluated against CPU version and speed up is reported for various matrix sizes.

II. Algorithms & GPU Implementation

A. Chebychev Preconditioning

The iterative methods such as conjugate gradient are guaranteed to converge in N (matrix size) iterations. N is usually a large number (around 10k) and conjugate gradient method suffers from slow rate of convergence. A preconditionner matrix is an approximation to the inverse of

A. Asgari Kamiabad is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON Canada (e-mail: amirhassan.asgarikamiabad@utoronto.ca).

the coefficient matrix, A. The prefect preconditionner is the exact inverse of coefficient matrix, however calculating the exact inverse is expensive process.

$$Ax = b \qquad (1)$$

Chebychev method is an iterative method for approximating the inverse of a scalar number. In [9], it is shown that matrix valued Chebychev method can be used as a preconditionner matrix. This idea is implemented on GPU in this project. In contrast to more conventional preconditioning methods, Chebychev method only uses matrix-matrix or matrix-vector multiplication and addition.

$$T_k(Z) = 2Z(T_{k-1}(Z)) - T_k(Z) \qquad (2)$$

$$A^{-1} = \frac{c_0}{2} I + \sum_{1}^{r} c_k T_k(Z) = M \qquad (3)$$

(2) Shows the Kth Chebychev polynomial, Z is the scaled Jacobian matrix. The first and second Chebychev polynomials are I (Identity Matrix) and Z. The few first Chebychev polynomials are added to form the preconditionner matrix.

The figure 1 describes the implementation of this process on GPU:

1-Load the Jacobian matrix to GPU and Build Z matrix
2-For k = 1 to r Do:
3- For J = 1 to N Do:
4- Multiply Jth column of T(k-1) by Z
5- Subtract Jth column of T(k-2)
6- Update Jth column of T(k) and M
7- Save dense updates in sparse
8- EndDo
9- Change matrix pointers
10-EndDo
11-Save Preconditionner matrix

Fig. 1. Chebychev Preconditioning Algorithm on GPU

In 4th line, both matrix and vector are in sparse format, the algorithm for this multiplication is shown figure 2.

```
One thread per row: Multiply each element in the row
   for (int jj = row_start; jj < row_end; jj++)
Find corresponding element:
 while (kk < col_end) && (indice[jj]>=vec1_indice[kk])
   if found, perform the multiplication
      if (indice[jj] == vec1_indice[kk])
         sum += data[jj] * vec1_data[kk];
```

Fig. 2. Matrix Vector Multiplication (both in sparse)

In order to save the dense result in sparse format, a pointer vector indicates the nonzero elements in the vector. The pointer vector is scanned to find the number of nonzero elements and their position in sparse format. The figure 3 illustrates this algorithm.

Save the result of multiplication in dense format:

0	0	0	A	0	0	B	C	0	0	0	0	0	D	0

If thread holds zero, write zero, otherwise write 1:

0	0	0	1	0	0	1	1	0	0	0	0	0	1	0

Perform scan over the pointer vector:

0	0	0	1	1	1	2	3	3	3	3	3	3	4	4

```
if (ptr[tid] - ptr[tid-1]){
   data[ptr[tid] -1] = data[tid];
   indice[ptr[tid] -1] = tid;}
```

Sparse data:

A	B	C	D

Sparse Pointer:

4	7	8	14

Fig. 3. Dense to Sparse transformation

B. Conjugate Gradient

Conjugate gradient method has been extensively used to solve a given set of linear equations instead of direct methods. This algorithm is shown in figure 4.

1-Set initial values and calculate residual $r_0=b-Ax_0$
2-While not converged Do:
3- Multiply residual by preconditionner z = M*r
4- Compute inner product of (r,z) = ρ_i
5- $\beta = \rho_{i/} \rho_{i-1}$ and p = z + βp
6- Compute $\alpha = \rho_i/(p,A*p)$
7- Update solution x += αp
8- Update residual r -= αp
9- Compute norm of residual
11-If norm of residual is small: Converged

Fig. 4. Conjugate Gradient Algorithm on GPU

The main kernels are vector norm calculation, vector inner product and vector addition. The matrix vector multiplication is the same as previous section, but the vector is in dense format, therefore the while-loop can be eliminated. Next parts present the evaluation of the individual kernels and whole process.

III. METHODOLOGY

A. Chebychev Preconditioning

λ is an Eigen-value of matrix A, if there exists a nonzero vector x such that Ax = λx. The range of Eigen-value is an important factor in determining the number of iterations in CG algorithm until convergence. If Eigen-values are concentrated in a short range (e.g. [0-1]) the CG converges in few (e.g. less than 20) iterations. The real matrices in power system has very large range of Eigen-values (e.g [1- 3000]). A good preconditionner should reduce the range of Eigen-values of coefficient matrix. The histogram of Eigen-values will be the best metrics for evaluating preconditionner efficiency. In order to show the effect of preconditioning, sparse matrices of various sizes from real applications are imported to the

program and range of Eigen-values before and after preconditioning are reported. The effect on total number of iterations in CG method is also reported.

The execution time for each kernel is reported for detailed study of algorithm. A similar code for preconditionner and CG is developed in Matlab in order to compute the speed-up over sequential version.

The matrices for evaluation are sparse matrices in SCR (sparse compressed row) from real applications. The matrices and right hand side vector are taken from Matrix Market website [12].

IV. Evaluation

All the GPU experiments are performed on ug75.eecg.toronto.edu running GTX280/1G NVIDIA card and a quad core Intel Core2 duo processor with 4G of memory. The CPU code is implemented in MATLAB and performed on Intel Core2 Quad Q9450 running at 2200MHZ with 4GB of memory with 6MB L2 cash.

Figure 5 shows the effect of preconditioning on IEEE-57 test system. The first figure shows the range of Eigen-values for the original matrix. The next figure is the same matrix with scaled elements (it is multiplied by its main diagonal) and the last one is both scaled and preconditioned. The result from GPU is verified by CPU version of the algorithm. The preconditioned matrix has considerably smaller range of Eigen-values (figure 5 – c) which guarantee the effectiveness of the Chebychev preconditioning technique.

Table 1 shows the specification of the matrices used for the rest of evaluation. These matrices and the right hand side vectors are available on Matrix Market website [12]. The condition number shows whether the system is well-conditioned or ill-conditioned. A well-conditioned system has smaller condition number ($k < 10$) and an ill-conditioned matrix has larger condition number ($k > 1000$). If the condition number is higher, the CG takes more iteration to converge.

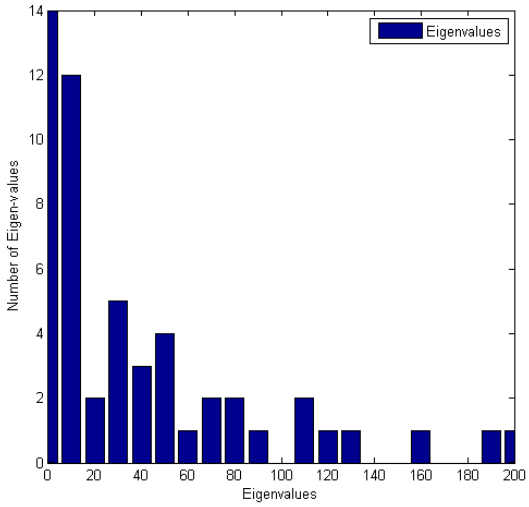
TABLE I
Test Matrices Properties

Matrix name	Matrix size	Total Nonzero	Condition Number
E05R	236	5856	6e+04
E20R	4241	131556	5.45e+10
E30R	9661	306335	3.47e+11

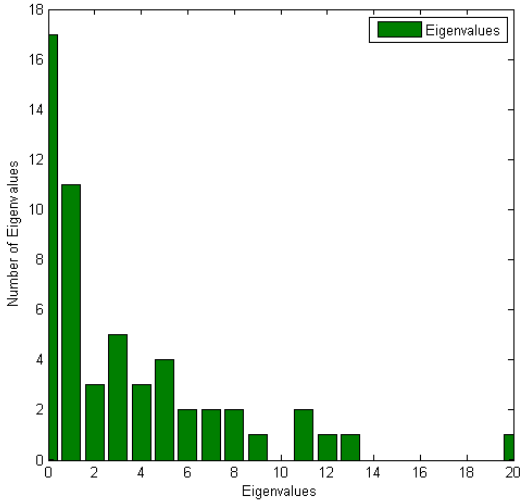
Figure 6 present the various kernel execution times for the test matrices. For smaller size matrices, the overhead of kernel initialization and data transfer limit the efficiency. The Dot Product kernel has to load two vectors from global memory. The global memory access time shows its effect on the execution time when the matrix size is increased considerably.

Preconditioning effect on total number of iterations is shown in figure 7. Different graphs show various matrix sizes. The horizontal axis shows the total number of iterations spent in Chebychev algorithm (r). When r = 0, the coefficient matrix is not preconditioned.

a. Without preconditioning



b. With scaling



c. With scaling and preconditioning

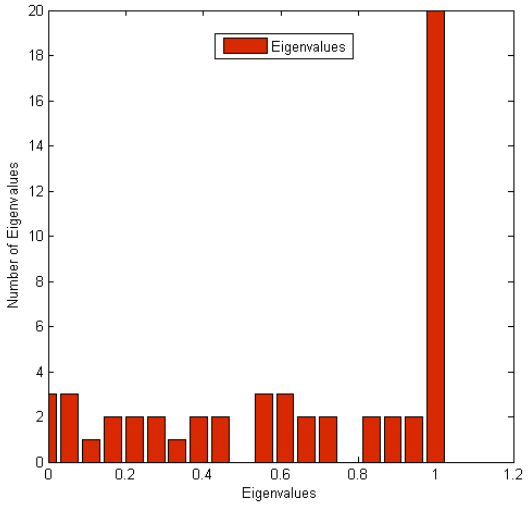


Fig. 5. The effect of preconditioning on IEEE-57 test system

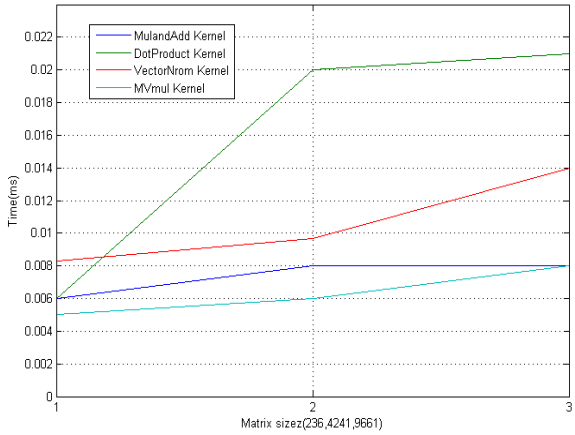


Fig. 6. Main kernels execution time for various input matrix size

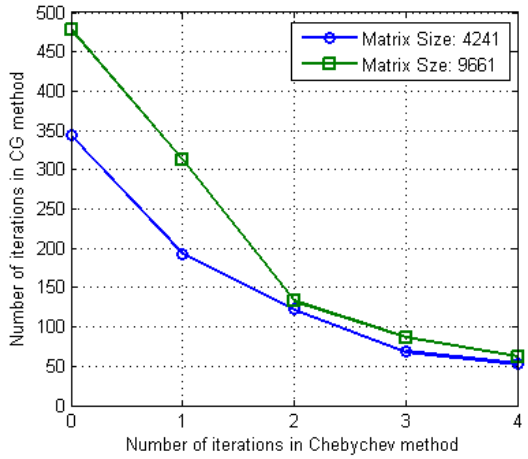


Fig. 6. Main kernels execution time for various input matrix size

Table II summarizes the result for preconditionner and CG for various matrix sizes. The 8x speed up in whole process is achieved for the largest matrix size. The execution time is comparable with the most optimized serial codes available for solving linear systems. This result proves the GPU's potential for reducing the computation time of transient stability process. However, transient stability process consists of several sequential parts and large data access and storage which limit the overall speed up.

This project uses sparse matrices which usually appear in real simulation. Dealing with sparse matrices adds more levels of complexity to the implementation, in particular on GPU platform. This process promises higher speed up in similar engineering applications which need solution of linear systems but use dense matrices. Another factor which may result in shorter execution time is branching factor. This factor puts a limit on number of nonzero in each row of the sparse matrix. This may reduce the generality of the process but will result in much simpler code and reduction in execution time. The algorithms provided in this project do not consider the branching factor and perform under any condition. The specific system configurations will be studied in future works.

TABLE II
TEST MATRICES PROPERTIES

Matrix(size)	Chebychev	CG	Total GPU	Total CPU	Speed Up
E05R(236)	033	1.51	1.84	4.67	2.54
E20R(4241)	1.78	23.31	25.09	154.78	6.16
E30R(9661)	5.41	90.8	96.21	756.3	7.86

V. CONCLUSION

The main achievement of this project is the reduction in computation time of transient stability analysis. This will optimistically enable the real-time or faster than real-time simulation of the power system which will have significant impact on the future design and security of power network.

Almost all of the scientific and engineering problems require solution of linear system. Direct methods, which are usually used for solving large linear systems, cannot be implemented properly on parallel machines; therefore, this project presents methods which are suitable for parallel processing and can be implemented on any other parallel platform. The preconditionner part can be used for most iterative solver based of Krylov subspace method. It can also be used when GPU performs as a co-processor for a large cluster of CPUs solving the main linear system.

Different variations of Krylov subspace methods such as biconjugate gradient (BiCG), and generalized minimum residual method (GMRES) appear in various engineering studies and use the same kernels developed in this project. The preconditionner algorithm remains the same for these algorithms. The GMRES method is more general method than CG and performs well for any matrices without any condition. The future work will concentrate on GMRES method.

VI. REFERENCES

[1] Shewchuk, Jonathan R."An Introduction to the Conjugate Gradient Method without the Agonizing Pain" 2007-10-09.

[2] Decker, I.C.; Falcao, D.M.; Kaszkurewicz, E." Conjugate gradient methods for power system dynamic simulation on parallel computers," Power Systems, IEEE Transactions on, Volume 11, Issue 3, Aug 1996 Page(s):1218 – 1227.

[3] Khaitan, S.K. and McCalley, J.D. and Qiming Chen" Multifrontal Solver for Online Power System Time-Domain Simulation," Power Systems, IEEE Transactions on, vol. 23, pp. 1727-1737, Nov. 2008.

[4] Chaniotis, D.; Pai, M.A., "Iterative solver techniques in the dynamic simulation of power systems," Power Engineering Society Summer Meeting, 2000. IEEE , vol.1, no., pp.609-613 vol. 1, 2000

[5] Anupindi, K.; Skjellum, A.; Coddington, P.; Fox, G., "Parallel differential-algebraic equation solvers for power system transient stability analysis," Scalable Parallel Libraries Conference, 1993, Proceedings of the , vol., no., pp.240-244, 6-8 Oct 1993

[6] de Leon, F. "A new preconditioned conjugate gradient power flow," Power Systems, IEEE Transactions on, Volume 18, Issue 4, Nov. 2003 Page(s): 1601 – 1609.

[7] Flueck, A.J.; Hsiao-Dong Chiang "Solving the nonlinear power flow equations with an inexact Newton method using GMRES," Power Systems, IEEE Transactions on, Volume 13, Issue 2, May 1998 Page(s):267 – 273.

[8] Gopal, A.; Niebur, D.; Venkatasubramanian, S. "DC Power Flow Based Contingency Analysis Using Graphics Processing Units," Volume, Issue, 1-5 July 2007 Page(s):731 – 736.

[9] Dag, H.; Semlyen, A., "A new preconditioned conjugate gradient power flow," Power Systems, IEEE Transactions on , vol.18, no.4, pp. 1248-1255, Nov. 2003.

[10] Yousef Saad, Iterative Methods for Sparse Linear Systems, Second Edition, Society for Industrial and Applied Mathematics, April 30, 2003.

Technical Reports:

[11] D. P. Koester and S. Ranka and G. C. Fox" Power Systems Transient Stability- A Grand Computing Challenge," NPAC Technical Report-SCCS 549, 31 Aug 1992.

[12] Matrix Market website: http://math.nist.gov/MatrixMarket/