Deriving Parallelism and GPU Acceleration Of Algorithms With Inter-Dependent Data Fields

** A Substitute for Building Sparse Connection Matrix and Sparse Matrix-Vector Multiplication **

Jaideep Singh, Ipseeta Aruni, Indian Institute of Technology Roorkee, INDIA

Abstract

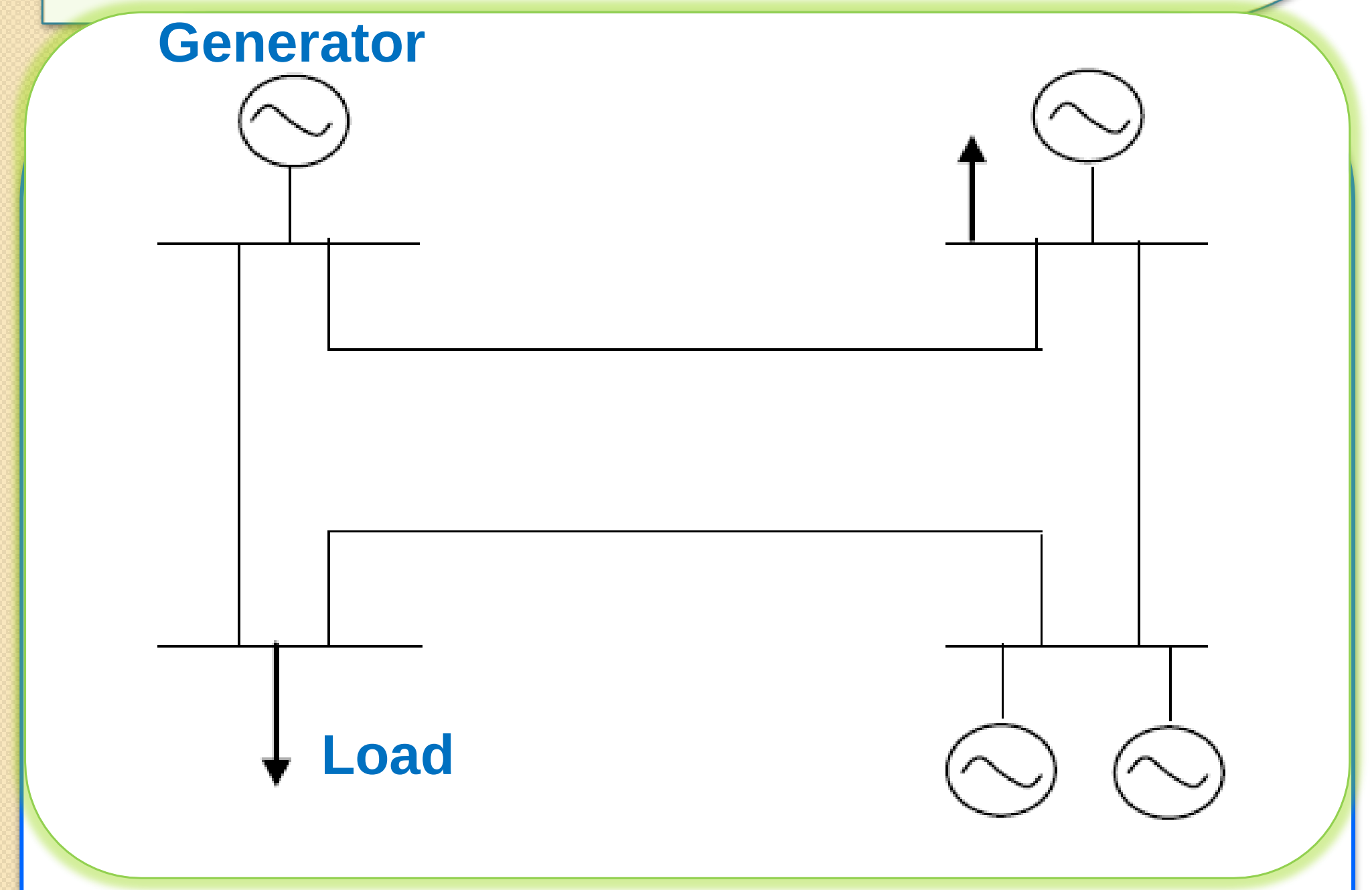
This poster presents an approach to derive parallelism in algorithms that involve building sparse matrix that represents relationships between inter-dependent data fields and enhancing its performance on the GPU. This work compares the algorithm performance on the GPU to its CPU variant that employs the traditional sparse matrix-vector multiplication (SpMV) approach. We have also compared our algorithm performance with CUSP SpMV [1] on GPU. The softwares used in this work are MATLAB and Jacket – GPU engine for MATLAB [2].

Introduction

To introduce the concept, we consider a problem from Power System Engineering domain. In this study, we consider a system of 'N' buses or nodes and variable number of generators connected at each bus. The aim is to compute the total power generation at each bus using generator power generation data. This is one of the starting steps in performing the Power Flow study. Power Flow study is performed to determine :

- Power flows and voltages in a system
- Power flows are within equipment ratings
- Voltages are within acceptable operating limits

Power Flow studies are performed for normal as well as contingency operating conditions.



A power system with variable generators and load at each bus

Classical Approach

This is done building a sparse connection matrix that has a non-zero value corresponding to generator connected at a particular bus. Let us assume that the system has 3 nodes and 5 generators with generator data given as:

Generator	Bus Number	Power Generation (MW)
1	1	20.09
2	1	30.52
3	2	40.80
4	2	20.34
5	3	10.03

To compute total power generated at each bus, we build a connection matrix of size 3x5 as follows:

Sparse Connection Matrix

Bus	Gen1	Gen2	Gen3	Gen4	Gen5
1	1	1	0	0	0
2	0	0	1	1	0
3	0	0	0	0	1

This matrix is built as a sparse matrix storing only the non-zero elements and corresponding indices. This sparse matrix when multiplied with the power generation vector gives the total power generation at each bus. ***This approach involves an additional multiply operation per non-zero element but the desired result is addition of data fields to give total power generation.***

We have devised a technique to do this in parallel without using sparse matrix approach. This is useful in Power Flow study as well as other algorithms exhibiting similar behavior.

Why GPUs in Power Flow Studies

- For a power system composed of thousands of buses and lines, to successfully compute the power flow in real time for these types of systems requires high computational throughput.
- Previous approaches to parallelize the power flow algorithms have been in the form of deploying heterogeneous network of workstations [3, 4] for real-time applications. These required specialized hardware and could not be commercialized due to investment constraints.
- On the other hand, commodity computer graphics processing units (GPU) are probably today's most powerful computational hardware per dollar installed on general purpose PC [5].

Our Algorithm : Par4Sp

To perform the same operation without building connection matrix and sparse matrix-vector multiplication, we build two vectors, viz. noOfGen and cumNoOfGen. This is demonstrated in the following diagram :

Parallel threads compute total power generation

Bus	noOfGen	cumNoOfGen
1	2	1
2	2	3
3	1	5

If we launch threads equal to number of buses, each thread can now fetch the generator count from noofGen for the number of times it has to loop (fetch generation data and add) and the starting location in the generator data is given by the cumNoOfGen vector. This technique avoids any additional multiply inherent in the spare matrix approach.

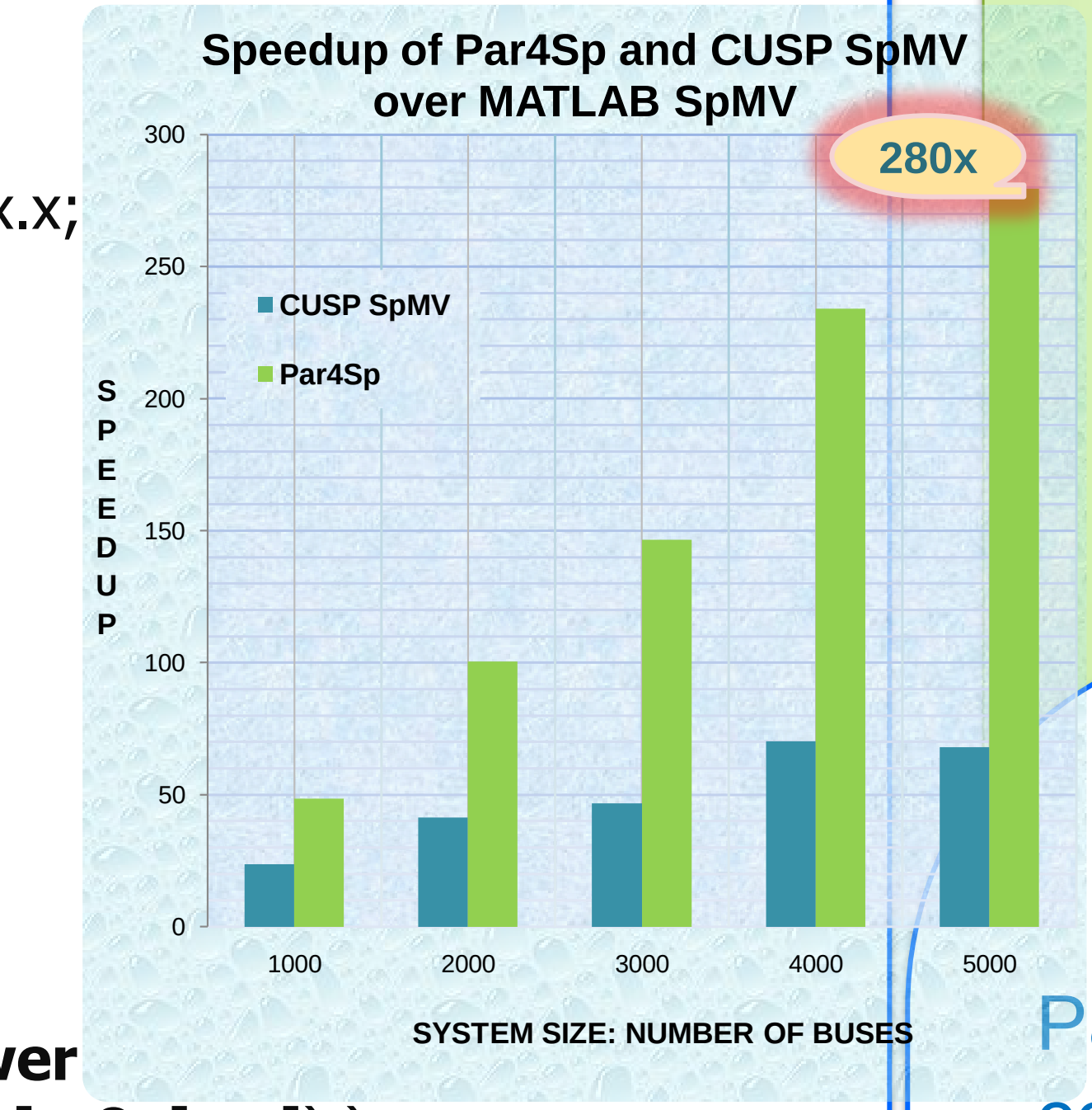
```
int idx = blockDim.x * blockIdx.x + threadIdx.x;
if ( idx >= noOfBuses )
return;

float2 temp = make_float2( 0.0f, 0.0f );

int ntimes = noOfGen[idx];
int pos      = cumNoOfGen[idx] -1;

for ( int i = 0 ; i < ntimes ; ++i )
{

// Pg and Qg hold real and reactive power
temp = cuCaddf ( temp, make_float2( Pg[pos],  Qg[pos]) );
pos++;
 }
```



Scope for Improvement

- This approach suffers from uncoalesced memory access patterns that limits the performance of Par4Sp.
- Due to random nature of generator count, threads in a warp have unbalanced computation load.
- → The noOfGen vector is sorted to have nearly balanced computation load for threads in a warp.
- → A special matrix containing power generation of each generator is maintained that allows coalesced memory access for threads in a warp.

Improved Par4Sp

Improved Par4Sp has enhanced performance and well adapted for GPU for the following reasons :

- **Synchronization-free parallelism between thread blocks** : Allocation of one thread per bus to compute total power generation at that bus independent of other buses in which a one-dimensional grid of thread blocks and a one-dimensional block of threads are used.
- **Optimized Global Memory Access** : A special data matrix stores the power generation corresponding to each generator as shown below:

Maximum generator count

Threads in a warp access consecutive data elements

Bus Number	Power Generation (MW)	Power Generation (MW)
1	20.09	30.52
2	40.80	20.34
3	10.03	x

- **Optimized thread mapping** : Sorted noOfGen vector allows for threads in a warp to have similar computation load.

Performance and Conclusion

Par4Sp was tested in MATLAB. The GPU version was coded using JACKET SDK tools [6]. MATLAB supports sparse matrices and allows sparse matrix-vector multiplication. We have also compared Par4Sp performance with CUSP, an optimized library for sparse matrix linear algebra on GPU. Number of generators and power generation values were randomly generated for each case. The following table shows the simulation results and the speedup obtained using Par4Sp and CUSP SpMV over CPU SpMV in MATLAB.

Buses (N)	% Non-zero(nz) elements = (nz /(N*N))	Time(ms) MATLAB CPU: EE2200 @2.2GHz	Time(ms) CUSP SpMV GPU : Tesla C1060	Time(ms) Par4Sp GPU : Tesla C1060	Speedup CUSP SpMV	Speedup Par4Sp
1000	5.4242	2.5678	0.108	0.053	23.77	48.449
2000	5.4511	9.5352	0.231	0.095	41.27	100.37
3000	5.9413	23.153	0.496	0.158	46.679	146.53
4000	5.7415	46.563	0.662	0.199	70.336	233.98
5000	5.6068	67.596	0.994	0.242	68.02	279.32

The above results demonstrate remarkable speedup for computing power generation on GPU. Compared with the MATLAB CPU version, Par4Sp is nearly 280 times faster for large power systems and outperforms CUSP SpMV on GPU, thus speeding up Power Flow studies and acting as a suitable substitute for similar problems.

References :

[1] http://code.google.com/p/cusp-library/

[2] http://wiki.accelereyes.com/wiki/index.php?title=Main_Page

[3] V. C. Ramesh, "On distributed computing for on-line power system applications," International Journal of Electrical Power & Energy Systems, vol. 18, 1996, pp. 527-533.

[4] N. Balu, T. Bertram, A. Bose, V. Brandwajn, G. Cauley, D. Curtice, A. Fouad, L. Fink, M. G. Lauby, B. F. Wollenberg, and J. N. Wrubel, "Online Power-System Security Analysis," Proceedings of the IEEE, vol. 80, 1992, pp. 262-280.

[5] N. Govindaraju, M. Harris, J. Krüger, A. E., Lefohn, and T. J. Purcell. —A Survey of General-Purpose Computation on Graphics Hardware|| In Eurographics 2005, State of the Art Reports, August 2005, pp. 21-51.

[6] http://wiki.accelereyes.com/wiki/index.php/Jacket_SDK

We express our gratitude to the AccelerEyes team for their support