Accelerating Block-tridiagonal Matrix Inversion on the GPU

Bemnet Demere
Faculty Advisor: Peter Yoon
Department of Computer Science, Trinity College, Hartford, CT

Introduction

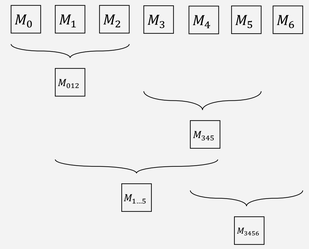
Block tridiagonal matrices are found in quantum physics. Specifically, the non equilibrium Green's function found in a an area of quantum method involves repeated inversions of large block tridiagonal matrices. A matrix containing say 100 blocks with dimensions of say 1000 (Petersen [1]) may need to be computed. Hence, given that those specific types of matrices are in the integral part of the aforementioned field and that they may also be of a large dimensions, enhancement in the speed of the inversion algorithms of such matrices will have a significant value.

GPU implementation

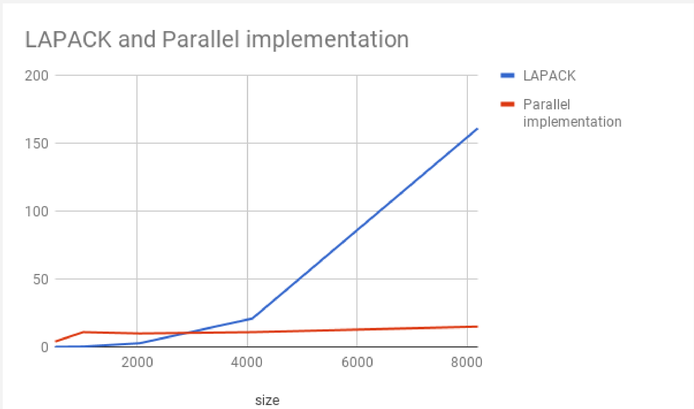
- The algorithm straightforwardly parallelizable routines such as matrix multiplication, addition and inversion of small blocks
- Block multiplication and additions
  - In the downward and upward sweeps to generated two pairs of set of matrices- ($\boldsymbol{d^L}$, $\boldsymbol{c^L}$) and ($\boldsymbol{d^R}$, $\boldsymbol{c^R}$) – multiplications of small block matrices are performed using cuBlas library
- Block inversions
  - In the two sweeps and in the computation of the diagonals, magma is used to perform sub-matrix inversions shown below

Computing blocks in the upper and lower triangles

- Computing the upper and lower triangles required a series of matrix multiplications as shown below
- Most of the execution time of the serial version is spent on repeatedly multiplying a series of matrices
- An amortization strategy is used to attain speedup.



Performance



Specifications
Intel Xeon E5-2620 CPU
NVIDIA Tesla P100 GPU, 16GB memory

Future work

Currently, the improvements are made by straightforward application of cuBlas and MAGMA and by the use of amortization in Step 3. Going forward, multiple GPUs may be used to compute the first two steps in parallel, and also more research should be done in finding ways to parallelize Step 3.

Acknowledgement

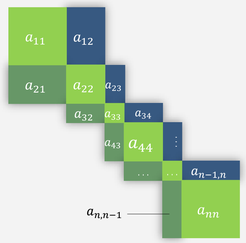
- Trinity College, Computer Science Department

References

REFERENCES
[1] Skelboe, Stig. "The Scheduling of a Parallel Tiled Matrix Inversion Algorithm." (2010).
[2] https://docs.nvidia.com/cuda/cublas/index.html
[3] http://icl.cs.utk.edu/projectsfiles/magma/doxygen/

Procedures of the algorithm

The algorithm used to compute the inverse of block-tridiagonal matrices in this project can be summarized into four procedures. For the following block tridiagonal matrix



Step 1: 'Downward sweep' ($d_{11}^L = a_{11}$)

$$c_{i-1}^L = -a_{i,i-1}\left(d_{i-1,i-1}^L\right)^{-1}, d_{ii}^L = a_{ii} + c_{i-1}^L a_{i-1,i}$$
$$for\ i = 2,3,\dots,n$$

Step 2: 'Upward sweep' ($d_{nn}^R = a_{nn}$)

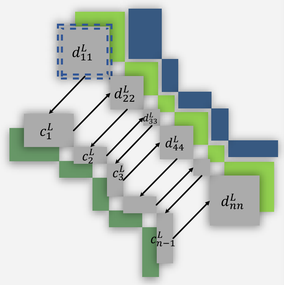
$$c_{i+1}^R = -a_{i,i+1}\left(d_{i+1,i+1}^R\right)^{-1}, d_{ii}^R = a_{ii} + c_{i+1}^R a_{i+1,i}$$
$$for\ i = n-1,\dots,1$$

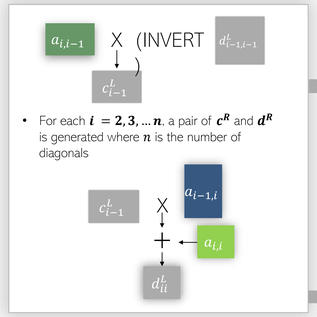
Step 3: Compute diagonals

$$g_{ii} = \left(-a_{ii} + d_{ii}^L + d_{ii}^R\right)^{-1} where\ i = 1,2,\dots n.$$

Step 4: Compute upper and lower triangles

$$g_{ij} = g_{ii} c_{i+1}^R c_{i+2}^R \dots c_j^R, for\ i < j$$
$$g_{ij} = g_{ii} c_{i-1}^L c_{i-2}^L \dots c_j^L, for\ i > j$$





- A matrix being pointed by the arrow will be computed using the matrix from which the arrow originates

Downward sweep:

- In this procedure, two sets of matrices necessary to compute the lower triangle of the inverse matrix are computed starting from the first $\boldsymbol{d^L}$

Upward sweep:

- Similarly, another sets of matrices used to compute the upper triangle of the inverse matrix are computed but this time starting from the last $\boldsymbol{d^R}$