PARALLEL CYCLIC REDUCTION OF TRIDIAGONAL LINEAR SYSTEM ON GPU

JOHN S GEORGE (JOHNS15) *

Abstract. Parallel Cyclic Reduction (PCR) algorithm, for solving tridiagonal linear systems, was implemented on GPU. The chosen tridiagonal system arised out of finite difference descretization of one dimensional Laplace equation. The performance of PCR algorithm was compared with a serial implementation of Cholesky factorization in the Eigen [1] numerical library. Results show that the parallel implementation on GPU outperforms the serial version both in terms of accuracy and kernel computation time.

1. Introduction. Tridiagonal linear systems often arises from the finite difference discretization of Poisson equation. For the purpose of this project, linear system arising out of discretization of one-dimensional Laplace equation with Dirichlet boundary conditions will be considered, i.e.,

$$\frac{d^2T}{dx^2} = 0 \tag{1.1}$$

The discretized form of equation 1.1, at a grid point i, is:

$$T_{i-1} - 2T_i + T_{i+1} = 0 \tag{1.2}$$

Differential eq 1.1 has an exact solution given by:

$$T(x) = -100x + 373.15 \tag{1.3}$$

where T is the temperature in Kelvin. The discrete solution was computed over the domain $x \in (0, 1)$. Since the discretization eq.1.2 is second-order accurate, the solution of the system of eqs.1.2 is exact. This allows for the validation of results from the implementation.

2. Serial Solvers. Tri-diagonal linear systems can be solved using either LU decomposition or Cholesky factorization. Since the matrix considered in this project is symmetric positive definite, Cholesky factorization can be used to solve the system. Eigen [1], which is a high-level C++ library for numerical solvers was used to solve the system using its sparse LDL variant of Cholesky decomposition. Another serial algorithm for solving tridiagonal system is Thomas algorithm, but it is only applicable to diagonally dominant or symmetric positive definite matrix, and also have stability issues.

3. Parallel Cyclic Reduction. Parallel Cyclic Reduction (PCR) is a form of divide and conquer algorithm. Linear combination of adjacent equations are used to eliminate alternate unknowns. For the tridiagonal system i^{th} equation,

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = y_i$$

is transformed into

$$\bar{a}_i x_{i-2} + \bar{b}_i x_i + \bar{c}_i x_{i+2} = \bar{y}_i$$

where

*johns15@illinois.edu

$$\bar{a}_i = \alpha_i a_{i-1}, \qquad \bar{b}_i = b_i + \alpha_i c_{i-1} + \beta_i a_{i+1}$$

$$\bar{c}_i = \beta_i c_{i+1}, \qquad \bar{y}_i = y_i + \alpha_i y_{i-1} + \beta_i y_{i+1}$$

with,

$$\alpha_i = -a_i / b_{i-1}, \qquad \beta_i = -c_i / b_{i+1}$$

Each equation is transformed as above, by handling the first two and last two equations as special cases. The resulting system of equations can be reordered, such that, the unknowns with odd indices are placed before even indices, as shown below.

$$\begin{bmatrix} \bar{b}_1 & 0 & \bar{c}_1 & & & & \\ 0 & \bar{b}_2 & 0 & \bar{c}_2 & & & \\ \bar{a}_3 & 0 & \bar{b}_3 & 0 & \bar{c}_3 & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & \bar{a}_{n-2} & 0 & \bar{b}_{n-2} & 0 & \bar{c}_{n-2} \\ & & & \bar{a}_{n-1} & 0 & \bar{b}_{n-1} & 0 \\ & & & & \bar{a}_n & 0 & \bar{b}_n \end{bmatrix} \longrightarrow \begin{bmatrix} \bar{b}_1 & \bar{c}_1 & & & & & \\ \bar{a}_3 & \bar{b}_2 & \ddots & & & & \\ & \ddots & \ddots & \bar{c}_{n-3} & & & \\ & & \bar{a}_{n-1} & \bar{b}_{n-1} & 0 & & \\ & & & 0 & \bar{b}_2 & \bar{c}_2 & \\ & & & & \bar{a}_4 & \bar{b}_4 & \ddots \\ & & & & & \ddots & \ddots & \bar{c}_{n-2} \\ & & & & & & \bar{a}_n & \bar{b}_n \end{bmatrix}$$

This breaks the system into two independent tridiagonal system that can be solved simultaneously. Each resulting system can in turn be broken down using the same technique, recursively, until it is reduced to multiple single equation systems. This process is known as forward reduction and hence eliminates the need for back-substitution. The simultaneous transformation of system of equations and solution of multiple tridiagonal system is the source for parallelism.

For a system of n equations, PCR takes $12n \log_2(n)$ operations and $\log_2(n)$ steps to finish on a parallel computer with n processors. The communication pattern for PCR with 8 unknowns is shown in Figure 3.1.

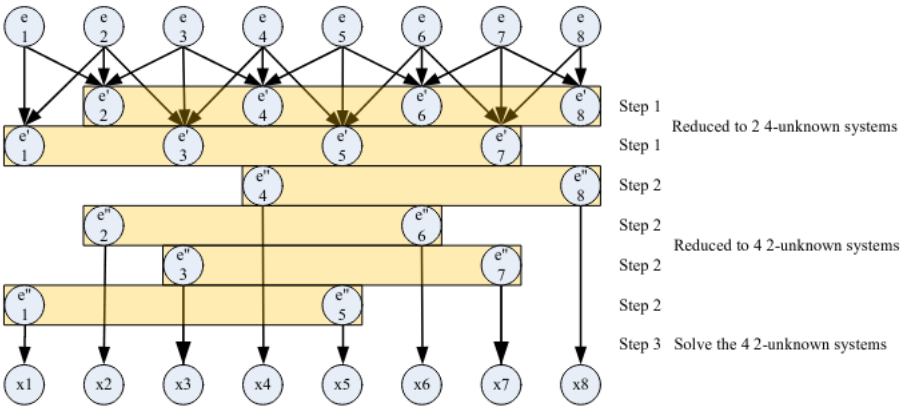


FIG. 3.1. *Communication pattern for PCR for a system of 8 equations. The yellow rectangle forms a system of equation. [2]*

4. GPU Implementation. PCR algorithm was implemented using the CUDA programming model on Nvidia GEFORCE GTX 1060 (6 GB), which has 10 Streaming Multiprocessors (SMs) and 128 CUDA cores per SM. One of the first step towards optimizing any tridiagonal solver is to take advantage of the sparse and banded nature of the matrix by storing only the non-zero entries on the three matrix diagonals and ignoring the zeros. This increases performance due to increased spatial and temporal locality even in a serial implementation on a CPU. The same storage pattern can also be used on GPUs to take advantage of the memory coalescing feature of the GPU's shared memory. In addition to matrix diagonals, the right-hand side and the solution vector is also stored in arrays. Since all the matrix equations are modified and accessed by the threads in each step of reduction, race conditions are possible. To avoid this, in the current implementation, 4 additional arrays are used for the matrix diagonals and the right-hand side vector.

Threads per block is taken to be 256 and the number of blocks are determined such that each equation is handled by one thread. The data is kept on the device memory during the entire process shown in Figure 3.1. The solvers are written to handle only power-of-two systems for the ease of implementation.

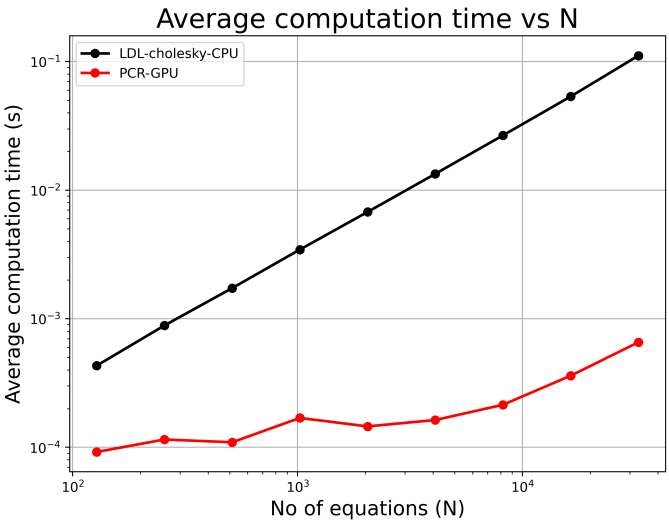
5. Performance Evaluation. In this section the performance results and the accuracy of PCR and LDLT factorization is compared for varying system sizes. Figure 6.1a shows the log-log plot of the average time taken by each implementation, excluding the time taken for GPU-CPU memory transfer. Figure 6.1b shows the speedup achieved by GPU in solving the equations, for number of unknowns varying from 2^7 to 2^{15}. The plot indicates that PCR outperforms LDLT factorization for all system sizes. Figure 6.2 shows the maximum residual error for each algorithm and indicates that PCR is not only faster but also more stable that Cholesky factorization. Figure 6.3 shows the time taken to allocate and transfer memory to GPU, suggesting that even though kernel computation is much faster on GPU, the time taken for memory transfer and allocation could adversely affect the overall performance.

6. Future Improvements. The current implementation uses temporary arrays to avoid race conditions during forward reduction stages of PCR algorithm. However, GPU's shared memory can be used instead and will save time on memory allocation and transfer. In addition, shared memory access is faster on a GPU compared to global memory access, which could further enhance the performance.

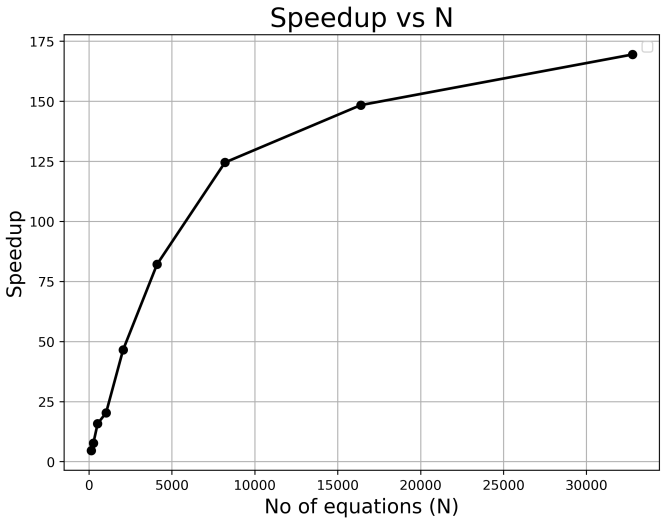
Hybrid versions of PCR algorithm which combines PCR with Cyclic Reduction(another parallel algorithm to solve tridiagonal sytem) can provide better performance [2]. Also, a more comprehensive analysis of computation time, by differentiating time taken for global and shared memory access from actual computation, could reveal further areas of improvement.

REFERENCES

[1] G. Guennebaud, B. Jacob, et al., *Eigen v3*. http://eigen.tuxfamily.org, 2010.
[2] Y. Zhang, J. Cohen, and J. Owens, *Fast tridiagonal solvers on the gpu*, vol. 45, 05 2010, https://doi.org/10.1145/1837853.1693472.



(a) Kernel computation time vs N



(b) Speedup vs N

FIG. 6.1.

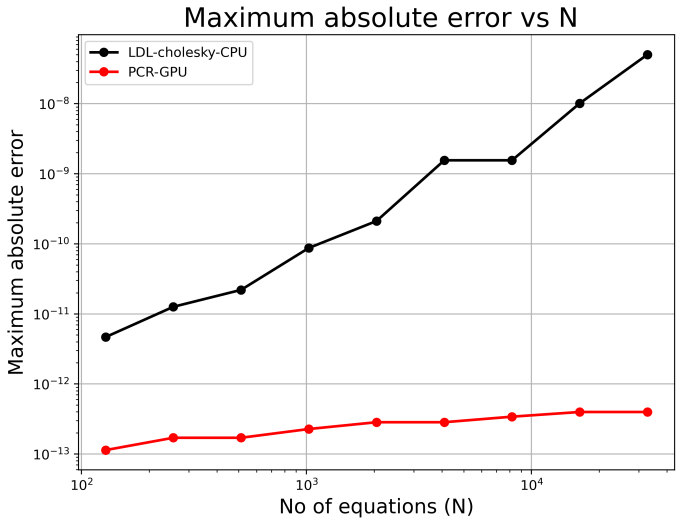


FIG. 6.2. *Maximum absolute error for varying number of unknowns*

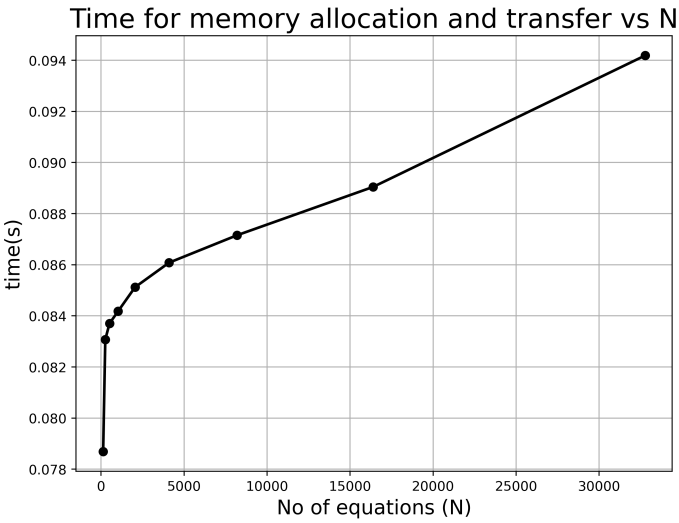


FIG. 6.3. *Time taken for memory transfer and allocation on GPU vs N*