Sorbonne Université

Introduction to CUDA

Projet : Solving a tridiagonal system on GPUs

Code and short presentation are due on 08/01/2021

The source code should be working, readable, and commented

Duration of presentation (code + slides) is not more than 10 minutes per pair

The special case of a system of linear equations is tridiagonal that has nonzero elements only on diagonals: the main diagonal and plus one above and one below it. Such tridiagonal system is shown in (1).

$$\begin{pmatrix} d_1 & c_1 & & & & \\ a_2 & d_2 & c_2 & & 0 & \\ & a_3 & d_3 & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & 0 & & \ddots & \ddots & c_{n-1} \\ & & & & a_n & d_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \tag{1}$$

Thomas method

For tridiagonal systems, the procedures of LU decomposition (aka Gaussian elimination), namely forward and backward substitutions, take only $O(n)$ operations, and the solution can be derived relatively easy. The forward substitution for the system of linear equations (1) is as follows

$$c_1' = \frac{c_1}{d_1},\ y_1' = \frac{y_1}{d_1},\ c_i' = \frac{c_i}{d_i - a_i c_{i-1}'},\ y_i' = \frac{y_i - a_i y_{i-1}'}{d_i - a_i c_{i-1}'} \text{ when } i = 2, ..., n. \tag{2}$$

The corresponding backward phase is

$$x_n = y_n',\ x_i = y_i' - c_i' x_{i+1} \text{ when } i = n-1, ..., 1. \tag{3}$$

The forward (6) and backward (7) phases are also called the **Thomas method**.

Obviously, it is not worthy to store the entire $n \times n$ matrix but rather the corresponding three vectors a, b, c. Note that a_1 and c_n are not defined in (1) and, hence, are not in use in (6) and (7).

Cyclic reduction

Like the Thomas method, the **Cyclic reduction** (CR) method consists of two phases, forward reduction and backward substitution. However, CR is suitable for parallelization. The forward phase successively reduces a system to a smaller system with half the number of unknowns, until a system of 2 unknowns is reached. The backward phase successively determines the other half of the unknowns using the previously solved values.

In each step of forward reduction, we update all even-indexed equations in parallel with the equation i of the current system as a linear combination of equations $i, i+1, i-1$, so that we derive a system of only even-indexed unknowns. Equation i has the form $a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i$. The updated values of a_i, b_i, c_i and d_i are

$$a_i' = -a_{i-1}k_1, d_i' = d_i - c_{i-1}k_1 - a_{i+1}k_2 \tag{4}$$

$$c_i' = -c_{i+1}k_2, y_i' = y_i - y_{i-1}k_1 - y_{i+1}k2 \tag{5}$$

$$k_1 = \frac{a_i}{d_{i-1}}, k_2 = \frac{c_i}{d_{i+1}} \tag{6}$$

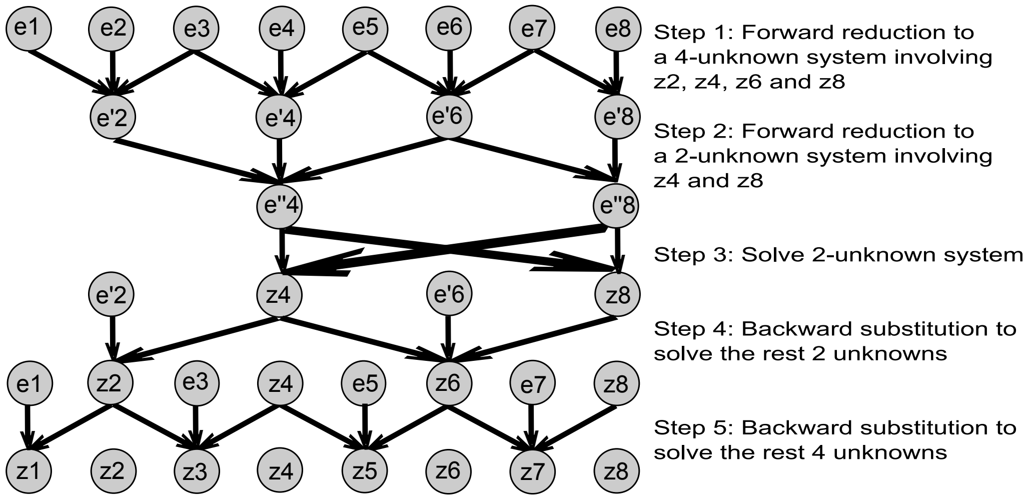
e1
e2
e3
e4
e5
e6
e7
e8
e'2
e'4
e'6
e'8
e"4
e"8
z4
z8
z2
z6
z1
z3
z5
z7
Step 1: Forward reduction to a 4-unknown system involving z2, z4, z6 and z8
Step 2: Forward reduction to a 2-unknown system involving z4 and z8
Step 3: Solve 2-unknown system
Step 4: Backward substitution to solve the rest 2 unknowns
Step 5: Backward substitution to solve the rest 4 unknowns

Figure 1: Communication pattern for CR in the 8-unknown case, showing the dataflow between each equation, labeled $e1$ to $e8$. Letters $e^{'}$ and $e^{''}$ stand for updated equation.

In each step of backward phase, we solve all odd-indexed unknowns x_i by using the already computed x_{i-1} and x_{i+1} values

$$x_i = \frac{(y_i' - a_i' x_{i-1} - c_i' x_{i+1})}{d_i'}. \tag{7}$$

Note that for the sake of simplicity, in the above description, we disregard the special treatment of the last equation and the first unknown respectively in the two algorithm phases. Also, we solve a 2-unknown system between the two algorithm phases.

Figure 1 provides a graphical representation of the communication pattern in CR, where the variables are denoted with a 'z' instead of 'x'.

Parallel cyclic reduction

The **Parallel Cyclic Reduction** (PCR) is a variant of CR. In contrast to CR, PCR only has the forward reduction phase. Although the reduction mechanism and formula are the same as those of CR, in each reduction step, PCR reduces each of the current systems to two systems of half size. Figure 2 shows PCR on an example of an 8-unknown system: we reduce it to two 4-unknown systems in step 1, then further reduce the two 4-unknown systems to four 2-unknown systems in step 2, and finally solve the four systems in step 3.

Tasks

In this project, we ask you to

1. find out how many steps both CR and PCR requires as well as how many operations
2. implement these three methods: the Thomas Method, Cyclic Reduction (CR), and Parallel Cyclic Reduction (PCR) with CUDA
3. implement also a version of PCR for matrices of size that is no power of two, e.g. 1023×1023
4. optimize the code with shared memory and registers, possible with the shuffle instructions. NB: better to have multiple versions of the code and use version control like `git`
5. compare performance among best performing implementations of the three methods
6. find application of these methods and show how it works

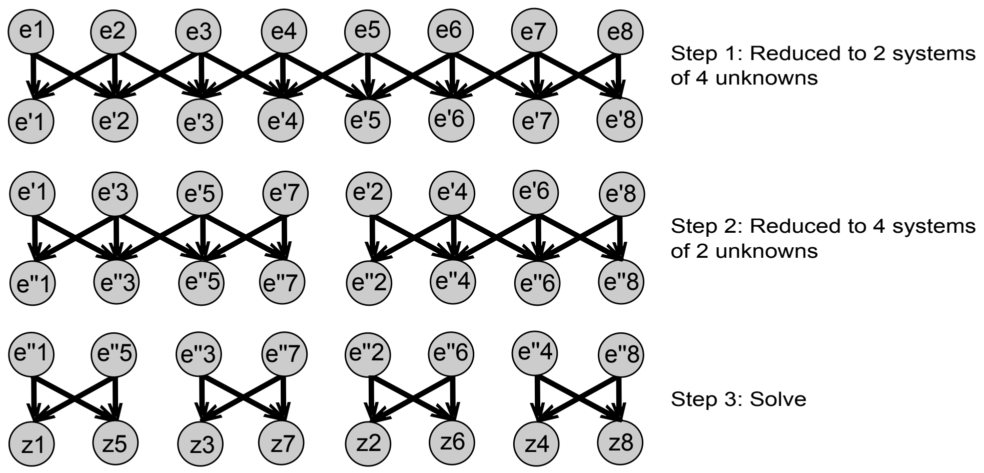

Figure 2: Communication pattern for PCR in the 8-unknown case, showing the dataflow between each equation, labeled $e1$ to $e8$. Letters $e^{'}$ and $e^{''}$ stand for updated equations.

References

[1] W. H. Press et al. Numerical Recipes in C: The art of scientific computing. Section 2.4 Tridiagonal and Band Diagonal Systems of Equations

[2] Y. Zhang, J. Cohen and J. D. Owens. Fast Tridiagonal Solvers on the GPU