CUDA Implementation of The Jacobi Method

Biyang Fu

School of Computing

Clemson University

Clemson, US

Biyangf@clemson.edu

一、Introduction

In numerical linear algebra, the Jacobi method is an iterative algorithm used to determine the solution of strictly diagonally dominant linear equations. Solve for each diagonal element and insert an approximation. Then repeat the process until convergence. This algorithm is a simplified version of the Jacobi transformation method of matrix diagonalization. The Jacobian iteration method is the earlier and simpler one among many iteration methods, and its name is also to commemorate the famous Prussian mathematician Jacobi. The calculation formula of the Jacobian iteration method is simple. It only needs to calculate the multiplication of the matrix and the vector once for each iteration, and the original matrix A is always unchanged during the calculation process, which is easier to calculate in parallel.

二、Description

Iterative method is suitable for solving large-scale sparse linear equations (high order and many zero elements). As a general method for solving numerical problems, the basic idea of the iterative method is unified, that is, the iterative sequence generated by the designed iterative formula is used to gradually approximate the exact solution.

Suppose there are linear equations Ax = b, where A is not singular, and the equation system has a unique solution x∗. For the matrix Ax = b, the solution can be written in the form of a system of equations $[a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1]$, etc., after sorting, we can write it in the following form:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

Has a unique solution. And the coefficient matrix has no zeros on its main diagonal, namely, $a_{11}, a_{22}, \ldots, a_{nn}$ are nonzeros.

To begin, solve the 1st equation for x_1, the 2nd equation for x_2 and so on to

obtain the rewritten equations:

$$x_1 = \frac{1}{a_{11}}(b_1 - a_{12}x_2 - a_{13}x_3 - \cdots - a_{1n}x_n)$$
$$x_2 = \frac{1}{a_{22}}(b_2 - a_{21}x_1 - a_{23}x_3 - \cdots - a_{2n}x_n)$$
$$\vdots$$
$$x_n = \frac{1}{a_{nn}}(b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{n,n-1}x_{n-1})$$

Then make an initial guess of the solution $x^{(0)} = (x_1^{(0)}, x_2^{(0)}, x_3^{(0)}, \ldots x_n^{(0)})$. Substitute these values into the right hand side the of the rewritten equations to obtain the first approximation, $(x_1^{(1)}, x_2^{(1)}, x_3^{(1)}, \ldots x_n^{(1)})$. This accomplishes one iteration.

In the same way, the second approximation $(x_1^{(2)}, x_2^{(2)}, x_3^{(2)}, \ldots x_n^{(2)})$ is computed by substituting the first approximation's x-vales into the right hand side of the rewritten equations.

By repeated iterations, we form a sequence of approximations $x^{(k)} = (x_1^{(k)}, x_2^{(k)}, x_3^{(k)}, \ldots x_n^{(k)})^t$, k = 1, 2, 3, …

So, for each k $\geq$ 1, generate the components $x_i^{(k)}$ of $x^{(k)}$ from $x^{(k-1)}$ by

$$x_i^{(k)} = \frac{1}{a_{11}}\left[\sum_{\substack{j=1,\\ j\neq i}}^{n}\left(-a_{ij}x_j^{(k-1)}\right) + b_i\right], \qquad \text{for i = 1, 2, … n}$$

Consider to solve an n $\times$ n size system of linear equations Ax=b with $A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}$ and $b = \begin{matrix} b_1 \\ \vdots \\ b_n \end{matrix}$ for $x = \begin{matrix} x_1 \\ \vdots \\ x_n \end{matrix}$. We split A into

$$A = \begin{bmatrix} a_{11} & 0 & \ldots & 0 \\ 0 & a_{22} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & a_{nn} \end{bmatrix} - \begin{bmatrix} 0 & \ldots & 0 & 0 \\ -a_{21} & \ldots & 0 & 0 \\ \vdots & & \ddots & \vdots \\ -a_{n1} & \ldots & -a_{n,n-1} & 0 \end{bmatrix} - \begin{bmatrix} 0 & -a_{12} & \ldots & -a_{1n} \\ 0 & 0 & & \vdots \\ \vdots & \vdots & \ddots & -a_{n-1,n} \\ 0 & 0 & \ldots & 0 \end{bmatrix} = D - L - U$$

Ax=b is transformed into (D – L – U)x = b, so Dx = (L + U)x + b.

Assume D^{-1} exists and $D^{-1} = \begin{bmatrix} \frac{1}{a_{11}} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{1}{a_{nn}} \end{bmatrix}$, Then x = $D^{-1}(L+U)x + D^{-1}b$

The matrix form of Jacobi iterative method is

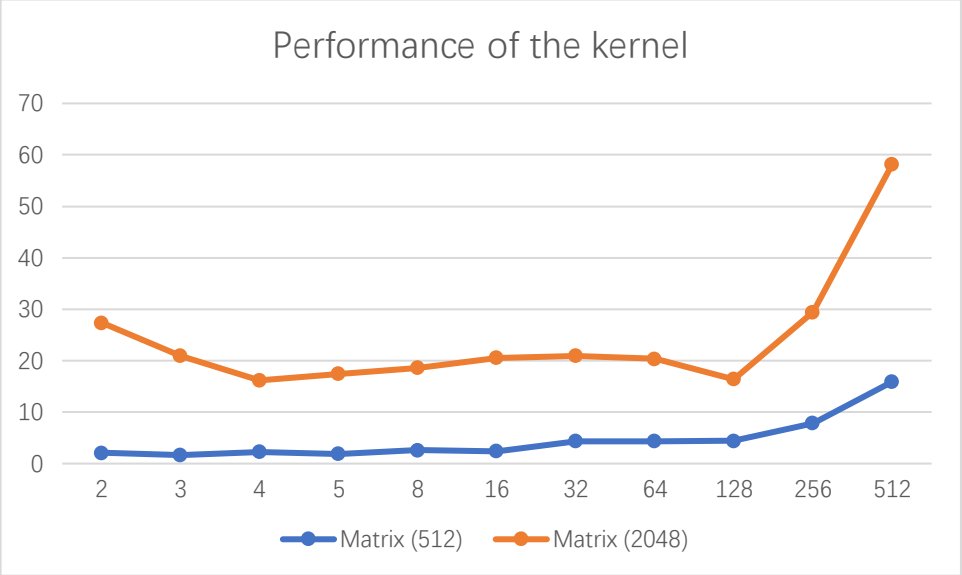
$$x^{(k)} = D^{-1}(L+U)x^{(k-1)} + D^{-1}b \qquad \text{k = 1, 2, 3, …}$$

三、Implementation

We implement the Jacobi method through serial CPU functions, unoptimized CUDA kernels and optimized versions of CUDA kernels. The CPU serial implementation is implemented in C according to the pseudo code on Wikipedia. The unoptimized kernel runs an inner loop in each thread, where all threads belong to a block. Then two adjustments were made to optimize this kernel. First, the index is pre-calculated and stored in the local register. This avoids one multiplication in each iteration in each thread. Second, flatten the threads into blocks. This can be extended to larger problem sizes.

四、Results

We tested all the implementations on 512 matrix and 2048 matrix respectively, and recorded the time of 10,000 iterations of the Jacobi method. The runtime for the serial CPU implementation is 18.34s for the 512 matrix and 283.23s for the 2048 matrix. About the optimized GPU implementation, as shown in the following figure:



So we can see that the shortest runtime is 1.72s on the 512 matrix with a tile size of 3 and 15.93s on the 2048 matrix with a tile size of 4. In the 512 matrix problem, the GPU parallel operation is about ten times faster than the CPU, and in the larger 2048 matrix problem, its running speed is nearly 18 times that of the CPU.

五、References

1. Saad, Yousef (2003). Iterative Methods for Sparse Linear Systems (2 ed.). SIAM. p. 414. ISBN 0898715342.
2. Eichler, M. and Zagier, D., 1985. The theory of Jacobi forms (Vol. 55, pp. v+-148). Boston: Birkhäuser.
3. Mirco Michel, CudaJacobi. https://github.com/MMichel/CudaJacobi. Mar 3, 2015.
4. Xinyuan Huang, Jacobian iteration. https://zhuanlan.zhihu.com/p/30965284. Aug. 24, 2019.
5. Zhang, Z., Miao, Q., & Wang, Y. (2009, December). CUDA-based Jacobi's iterative method. In 2009 International Forum on Computer Science-Technology and Applications (Vol. 1, pp. 259-262). IEEE.