

# Parallel Computer Architecture

## Final Exam

### Iterative Jacobi Solver

Prof. Naga Kandasamy  
ECE Department  
Drexel University

March 12, 2018

This problem, worth 25 points, is due March 24, 2018, by 11:59 pm via BBLearn. You may work on this problem in a team of up to two people. One submission per group will suffice. Please submit original work. Solutions copied from other students or from online sources will result in a grade of zero for the entire exam. You may discuss high-level approaches to solve the problem with Shihao or me, but we will not help you with the code itself.

Consider the following system of linear equations  $Ax = b$  with  $n$  equations and  $n$  unknowns:

$$\begin{array}{cccccc} a_{0,0}x_0 & + a_{0,1}x_1 & + \cdots & + a_{0,n-1}x_{n-1} & = & b_0, \\ a_{1,0}x_0 & + a_{1,1}x_1 & + \cdots & + a_{1,n-1}x_{n-1} & = & b_1, \\ \cdot & \cdot & & \cdot & & \cdot \\ a_{i,0}x_0 & + a_{i,1}x_1 & + \cdots & + a_{i,n-1}x_{n-1} & = & b_i, \\ \cdot & \cdot & & \cdot & & \cdot \\ a_{n-1,0}x_0 & + a_{n-1,1}x_1 & + \cdots & + a_{n-1,n-1}x_{n-1} & = & b_{n-1}, \end{array}$$

where the unknowns are  $x_0, x_1, x_2, \dots, x_{n-1}$ . We have seen from the previous question that Gaussian elimination is one way to solve for these unknowns. Another method is by iteration. In the above system, the  $i^{\text{th}}$  equation

$$a_{i,0}x_0 + a_{i,1}x_1 + \cdots + a_{i,i}x_i + \cdots + a_{i,n-1}x_{n-1} = b_i$$

can be rearranged as

$$x_i = \frac{1}{a_{i,i}} (b_i - (a_{i,0}x_0 + a_{i,1}x_1 + a_{i,2}x_2 + \cdots + a_{i,i-1}x_{i-1} + a_{i,i+1}x_{i+1} + \cdots + a_{i,n-1}x_{n-1})),$$

or as

$$x_i = \frac{1}{a_{i,i}} \left( b_i - \sum_{j \neq i} a_{i,j}x_j \right) \quad (1)$$

This equation gives  $x_i$  in terms of the other unknowns and can be used as an iteration formula for each of the unknowns to obtain better approximations. The iterative method used in the reference code provided to you is the *Jacobi iteration*. It can be shown that the Jacobi method will converge if the diagonal values of  $a$  have an absolute value greater than the sum of the absolute values of the other  $a$ 's on the row; that is, the array of  $a$ 's is *diagonally dominant*. In other words, convergence is guaranteed if

$$\sum_{j \neq i} |a_{i,j}| \leq |a_{i,i}|.$$

The matrix  $A$  provided to you satisfies the above condition. Note however, that this condition is a sufficient but not a necessary condition; that is, the method may converge even if the array is not diagonally dominant. The iteration formula in (1) is numerically unstable, however, in that it will not work if any of the diagonal elements are zero because it would require dividing by zero—but you do not have to worry about this issue for the assignment. An example of how the iterative method works to solve the system of equations is illustrated using a small example in the file called `jacobi_example.pdf`.

Given arrays `a[][]` and `b[]` holding the constants in the equation, `x[]` holding the unknowns, and a user-defined tolerance for convergence, the sequential code that implements the Jacobi iteration can be developed as follows.

```
for (i = 0; i < n; i++) /* Initialize the unknowns. */
    x[i] = b[i];
done = 0;
while (!done){
    /* Loop iterates over each unknown. */
    for (i = 0; i < n; i++){
        sum = 0;

        /* Implement Equation 1. */
        for (j = 0; j < n; j++){
            if (i != j)
                sum += a[i][j] * x[j];
        }
        new_x[i] = (b[i] - sum)/a[i][i];
    }
    /* Update the unknown values. Test for convergence. */
    ssd = 0;
    for (i = 0; i < n; i++){
        ssd += (x[i] - new_x[i])^2;
        x[i] = new_x[i];
    }
    if (sqrt(ssd) < TOLERANCE)
        done = 1;
} /* End while. */
```

The convergence criteria used in the above code snippet is to test the square root of the sum of the

squared differences of the  $x$  values from two consecutive iterations—the current iteration and the one before—against a user-defined error tolerance as

$$\sqrt{\sum_{i=0}^{n-1} (x_i^k - x_i^{k-1})^2} \leq \text{error tolerance},$$

where  $k$  is the iteration number.

The reference code given to you takes no input arguments. The solution provided by the GPU is compared to that generated by the CPU by printing out the relevant statistics. The configurable parameters such as the dimensions of the matrices, the error tolerance, thread block sizes, and number of thread blocks can be found within the `.h` file.

Complete the functionality of the equation solver on the GPU by editing the following functions: `compute_on_device()` and `jacobi_iteration_kernel()`. You may add additional kernels as well as host-side helper and timing functions as needed to your code. For maximum points, optimize the performance of your GPU code via efficient use of the GPU memory hierarchy. Ensure that your accesses to the matrix elements in global memory are coalesced. You may have to use double precision operations on the GPU to minimize any mismatch to the reference result.

Upload all files needed to run your code as a single zip file. Submit a short report describing: (1) the design of your kernels using code or pseudocode to clarify the discussion; (2) the speedup obtained over the serial version for matrix sizes of  $512 \times 512$ ,  $1024 \times 1024$ , and  $2048 \times 2048$ , including the overhead due to CPU/GPU communication; and (3) sensitivity of your kernels to thread-block size in terms of the execution time.