Jacobi iterative equation solver

$$Ax = b$$

$$\begin{array}{ccc} & \text{col} \rightarrow & \\ \text{row} \downarrow & A & \\ & \begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0\,n-1} \\ a_{10} & a_{11} & \cdots & a_{1\,n-1} \\ & & \ddots & \\ a_{n-1}a_0 & \cdots & & a_{n-1\,n-1} \end{bmatrix} & \end{array} \begin{array}{c} x \\ \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix} \end{array} = \begin{array}{c} b \\ \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{bmatrix} \end{array}$$

i

Each x_i can be calculated in parallel

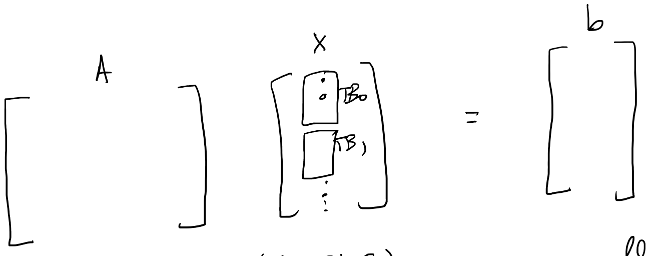
$$a_{i0}x_0 + a_{i1}x_1 + \dots + a_{ii}x_i + \cdots a_{in-1}x_{n-1} = b_i$$

$$a_{ii}x_i = b_i - \sum_{j \neq i} a_{ij}x_j$$

update rule

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

Version 1



thread block : (1, SIZE)
grid : $\left(1, \left\lceil \frac{\#\text{rows}}{\text{SIZE}} \right\rceil\right)$ ← covers all output elements of x

Host code :

```
1. Allocate and copy A, x, b, to device
                        (initialize to b)
2. Allocate for x_new and ssd
3. while (!done){
       cudamemset(ssd, 0)
                                  src   dest
       launch kernel (A, b, x, x_new, ssd)
       cudamemcpy(ssd)
       if ($\sqrt{ssd}$ < eps)
           done = 1
   } flip pointers to x, x_new
   cudamemcpy (x)
```

Use double precision for ssd values

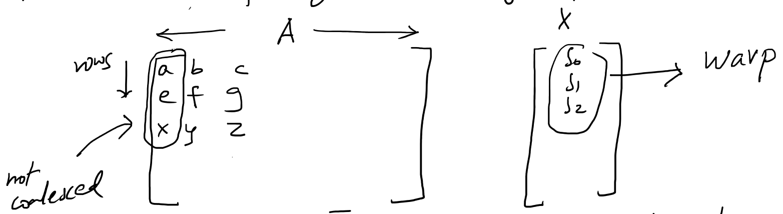
kernel code

```
1. Allocate shared memory for local_ssd values
2. old_value ← x[i]
3. new_value ← update
4. x_new[i] ← new_value
5. ~~lcl~~ssd = (old_value − new_value)$^2$
6. Store local_ssd to shared memory
7. Reduce local_ssds to single value at thread block level
8. if (threadIdx.x == 0)
       Accumulate reduced local ssd value into shared SSD value in GPU global mem
```

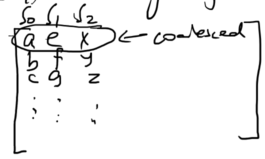
← use atomicAdd()

Optimized version:

- Make accesses from global memory of matrix A to be coalesced.



Convert A to $B = A^T$ on CPU s.t. elements of B are laid out in column major form



Rewrite update logic in kernel to perform update assuming colum-major layout.