Accelerating Scientific Computing Using Multiple GPUs

Aman Goyal (B19ME004)

Introduction: -

- Computational Fluid Dynamics (CFD) is the emerging field of fluid mechanics in which fluid flow problems are solved and analyzed using computational methods and numerical algorithms. The technologist in this field still uses heterogeneous computing to solve the fluid dynamics problems which includes more than one processor or cores.
- The problem with heterogeneous computing is, we have to transfer data from host to device and vice-versa which is a time-consuming process.
- To solve this problem, we can use multi-GPU environment and establish a direct communication between GPUs which can reduce time of data transfer and will increase efficiency.
- This technique has been used in other fields but not in fluid mechanic's field.

Objectives: -

- We write a 'C program' to solve the Poisson equation below using five-point stencil finite difference method with Dirichlet boundary condition on a square domain [-1,1] x [-1,1].

  Exact Solution U (x, y) = sin(pi*x) *cos(pi*y).

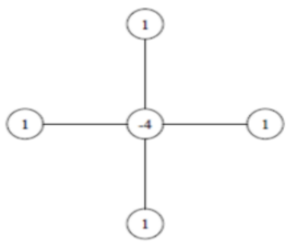
- We can approximate the Poisson equation to this-

Finite difference equation at grid point (i, j):

$$-\left(\frac{u_{i-1,j}-2u_{i,j}+u_{i+1,j}}{h^2}+\frac{u_{i,j-1}-2u_{i,j}+u_{i,j+1}}{h^2}\right)=f(x_i,y_j) \text{ or}$$

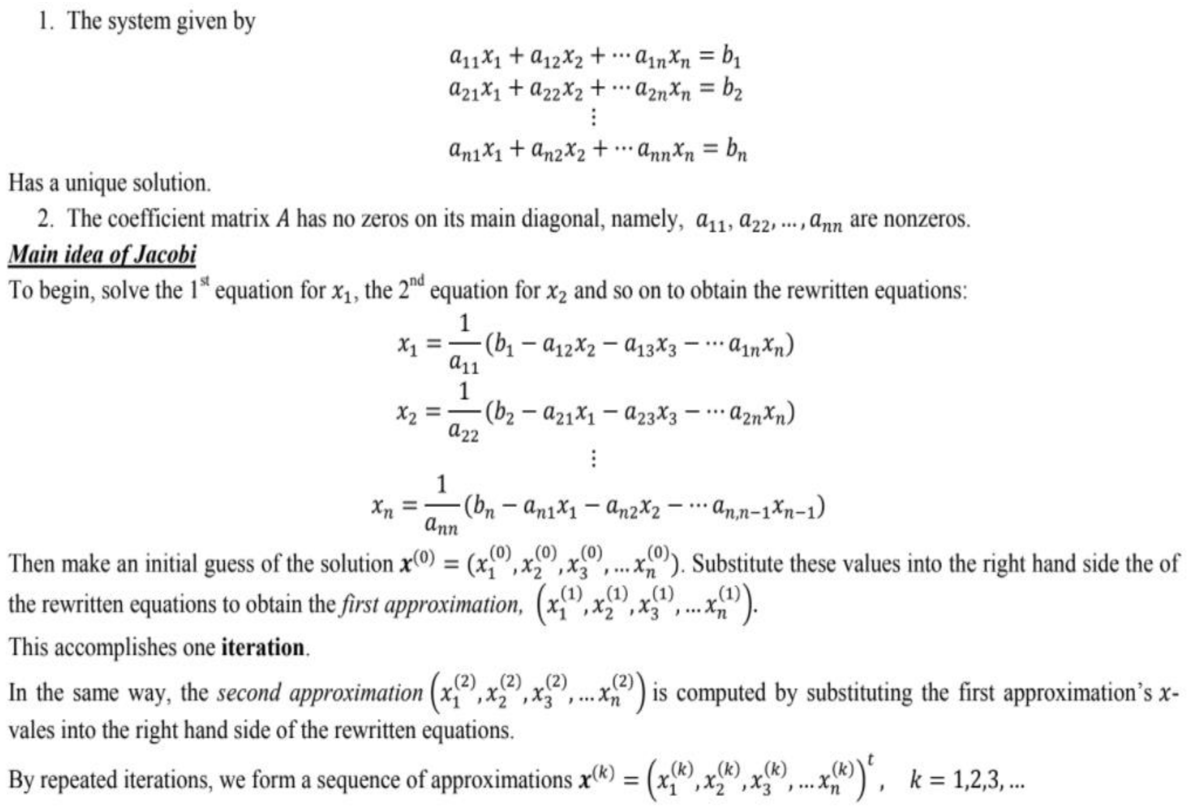
$$-u_{i,j-1}-u_{i-1,j}+4u_{i,j}-u_{i+1,j}-u_{i,j+1}=h^2 f(x_i,y_j)$$

- Five-point stencil of the finite difference approximation



- We can obtain a system of linear equations using this approximation. Further, we solve these equations using the Jacobi method.
- We solve equation using different numbers of total discretization points N in each direction.

Jacobi Method to solve system of linear equation-



1. The system given by

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

Has a unique solution.

2. The coefficient matrix A has no zeros on its main diagonal, namely, $a_{11}, a_{22}, \dots, a_{nn}$ are nonzeros.

Main idea of Jacobi

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

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

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

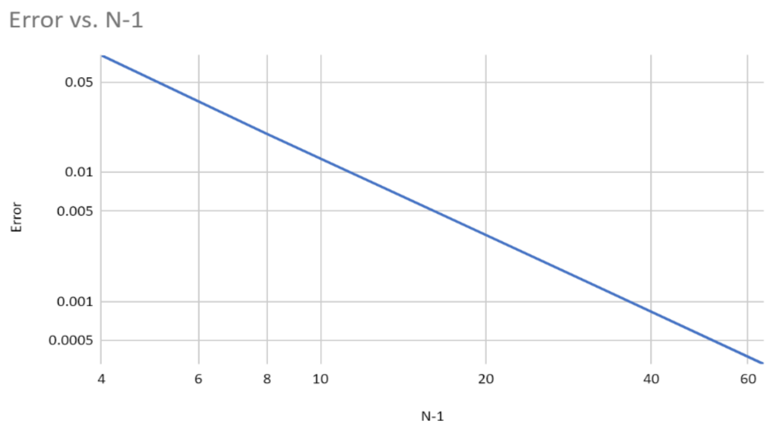
This accomplishes one **iteration**.

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

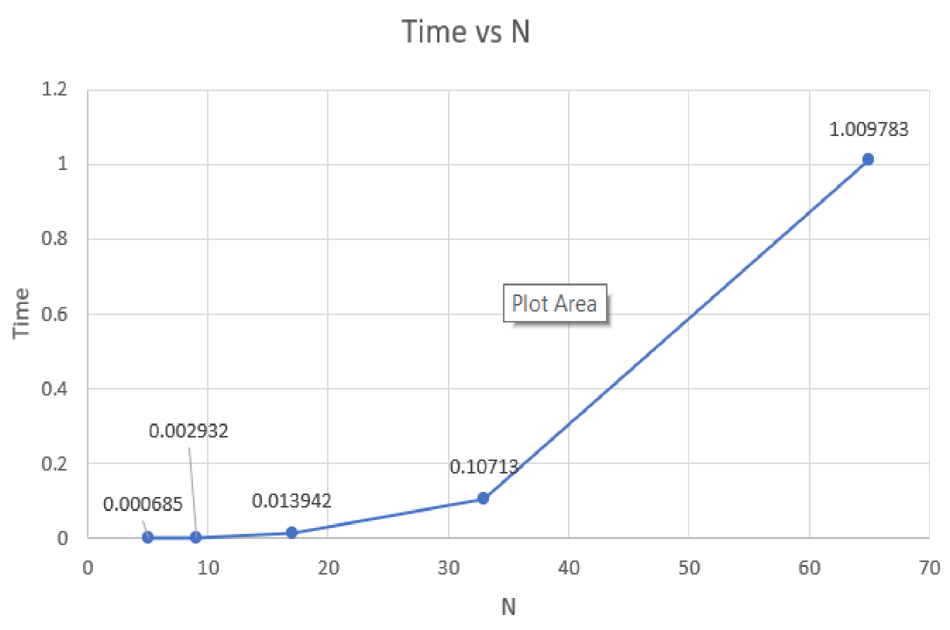
By repeated iterations, we form a sequence of approximations $\boldsymbol{x}^{(k)} = \left(x_1^{(k)}, x_2^{(k)}, x_3^{(k)}, \dots x_n^{(k)}\right)^t, \quad k = 1,2,3,\dots$

ONE CPU: –

We write a C program to solve the Poisson equation given above. We calculated error by the difference between exact and approximated value. Then, we run this program on the computer which has a CPU processor.



We also calculated the computational time taken by the process. Finally, we plot the graph of error v/s N and Time v/s N. The results we get are shown below.



As we can observe from these graphs, error is decreasing linearly with increasing N as we are getting more accurate results and time is also linearly increasing with N.

ONE CPU and ONE GPU: -

- A code is written in CUDA programming language to run on single GPU device. (One CPU, one GPU)

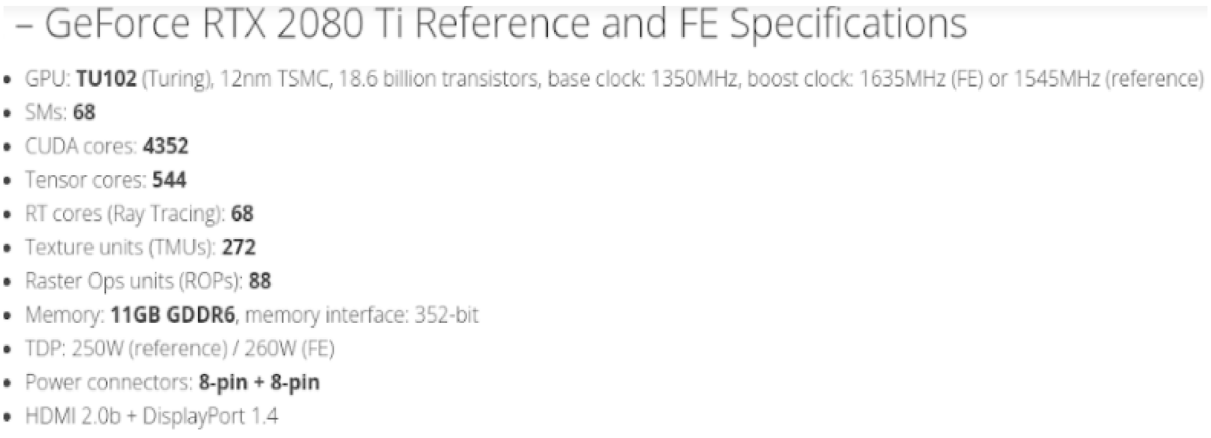
After running the code on the GPU Device, we get the following results-

Grid	Time (CPU in s)	Time (GPU in s)	Number of Iterations to converge
40×40	0.088 s	0.065 s	2048
80×80	1.192 s	0.225 s	7706
160×160	17.87 s	0.88 s	28368

A noticeable thing is that the GPU card has not yet saturated, so the time in GPU increases by 4 times with each increasing resolution because the number of iterations roughly increases by 4. The number of cells also increases by 4, but the number of threads launched also increases by 4.

The time of the CPU (1 processor) increases by 16 times with each increasing resolution because the number of iterations roughly increases by 4 and the number of cells also increases by 4. But the number of CPU processors is still 1.

Specifications of the GPU



– GeForce RTX 2080 Ti Reference and FE Specifications

- GPU: **TU102** (Turing), 12nm TSMC, 18.6 billion transistors, base clock: 1350MHz, boost clock: 1635MHz (FE) or 1545MHz (reference)
- SMs: **68**
- CUDA cores: **4352**
- Tensor cores: **544**
- RT cores (Ray Tracing): **68**
- Texture units (TMUs): **272**
- Raster Ops units (ROPs): **88**
- Memory: **11GB GDDR6**, memory interface: 352-bit
- TDP: 250W (reference) / 260W (FE)
- Power connectors: **8-pin + 8-pin**
- HDMI 2.0b + DisplayPort 1.4

SKILLS ACQUIRED

- CUDA Programming Language
- Learned Open-Source Debugging Software GDB