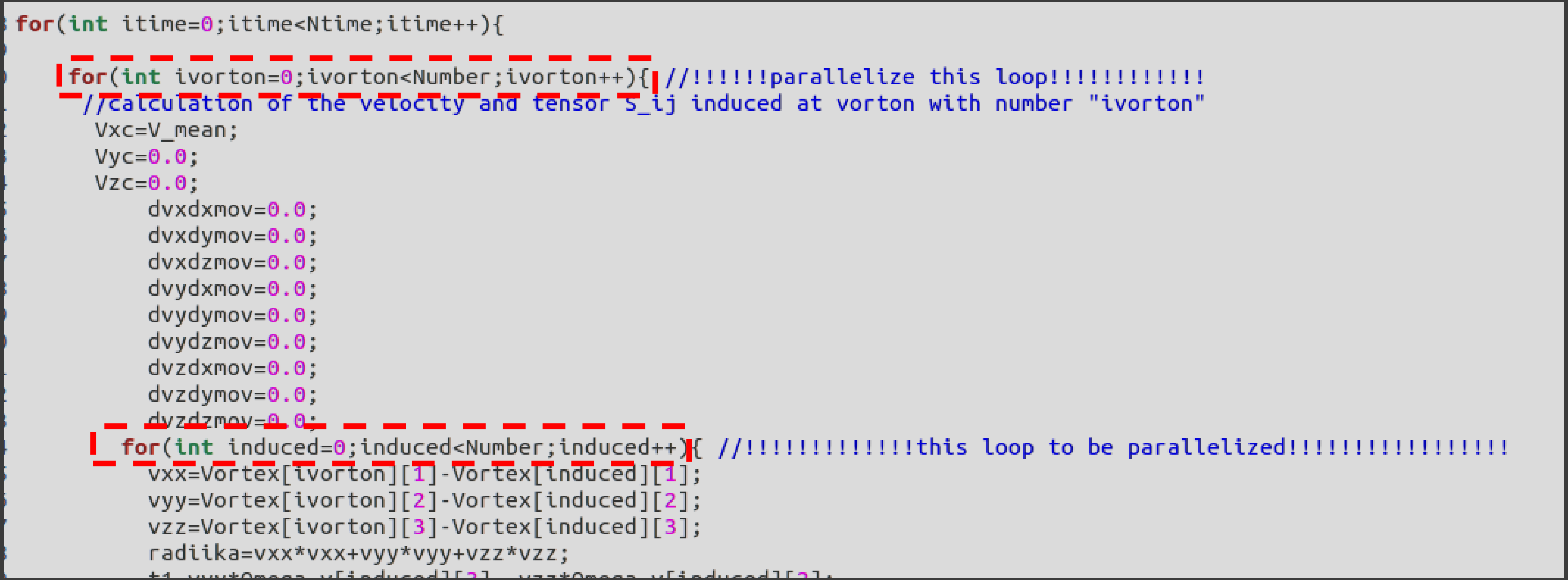
Strömungssimulation auf GPUs

Kami Reddy Koti Reddy -- 216100231
Atul Udaivir Singh -- 216100191
Kamineni Akhil -- 216100197

Problem statement

The task is to accelerate the computations using GPU, OMP and MPI options. The part of the code to be parallelized contains two embedded loops (see comments ”this loop can be done parallel”).



```
for(int itime=0;itime<Ntime;itime++){

   for(int ivorton=0;ivorton<Number;ivorton++){ //!!!!!!parallelize this loop!!!!!!!!!!!
    //calculation of the velocity and tensor S_ij induced at vorton with number "ivorton"
     Vxc=V_mean;
     Vyc=0.0;
     Vzc=0.0;
         dvxdxmov=0.0;
         dvxdymov=0.0;
         dvxdzmov=0.0;
         dvydxmov=0.0;
         dvydymov=0.0;
         dvydzmov=0.0;
         dvzdxmov=0.0;
         dvzdymov=0.0;
         dvzdzmov=0.0;
       for(int induced=0;induced<Number;induced++){ //!!!!!!!!!!!!!this loop to be parallelized!!!!!!!!!!!!!!!!
         vxx=Vortex[ivorton][1]-Vortex[induced][1];
         vyy=Vortex[ivorton][2]-Vortex[induced][2];
         vzz=Vortex[ivorton][3]-Vortex[induced][3];
         radiika=vxx*vxx+vyy*vyy+vzz*vzz;
```

N-Body problems

Simulated with particle systems, where each particle interacts with all other particles according to the laws of physics

Fundamentally, point vortices correspond to singularities in an ideal irrotational flow, which in turn characterize the flow itself.

E.g:-Jupiter's red spot, intensifying hurricanes and plasma flows. Each vortex generates a rotational velocity field that advects all other vortices.

Problem Description

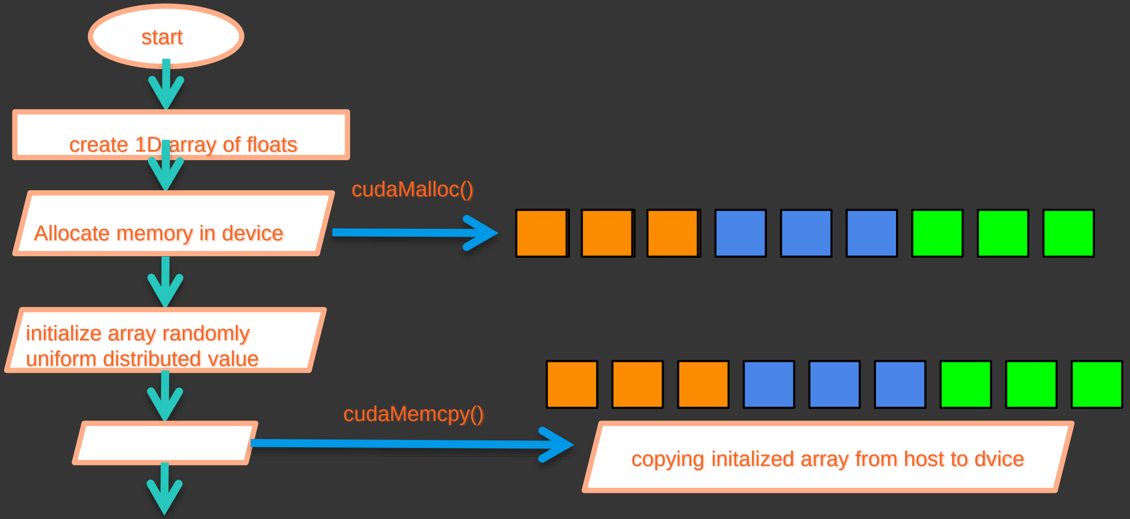
- We have a cube of dimension 1 x 1 x 1 units. Where 700 Vortices are filled in cube, Each vortex has data of its position and angular velocities in x,y,z directions.
- Once the discretized form of below equations are applied, then the position and angular velocity of vortex is changed for first time step.

$$\mathbf{V}(\mathbf{y}) = \sum_{j=1}^{N} \mathbf{r} \times \mathbf{\Gamma}_j e^{-\pi\rho/2}; \qquad \frac{d\mathbf{\Gamma}_k}{dt} = (\mathbf{\Gamma}_k \cdot \nabla) \sum_{j=1}^{N} \mathbf{r} \times \mathbf{\Gamma}_j e^{-\pi\rho/2},$$

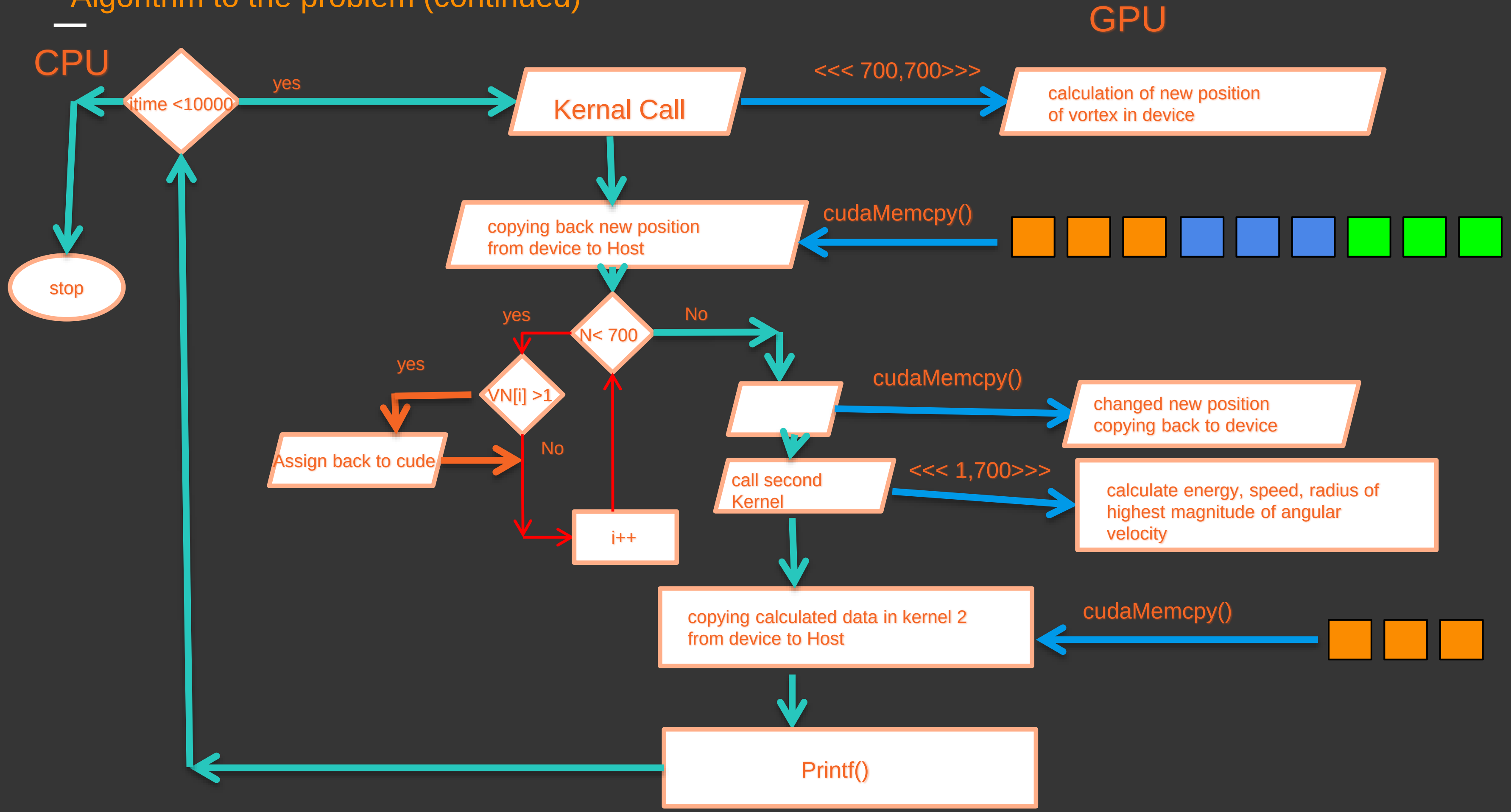
$$\sigma_k(t+\Delta t) = \sigma_k^*(t+\Delta t) + 2\nu\pi\Delta_t; \mathbf{\Gamma}_k(t+\Delta t) = \mathbf{\Gamma}_k^*(t)\left(\frac{\sigma_k^*(t+\Delta t)}{\sigma_k(t+\Delta t)}\right)^5$$

- If the new positioned vortex are out of box, Assign new position to it in the box and new angular velocity to it.
- Find the maximum magnitude of angular velocity, Speed, Radius of Vortex.
- The same procedure is continued for N time steps.

Algorithm to the problem (Flowchart)

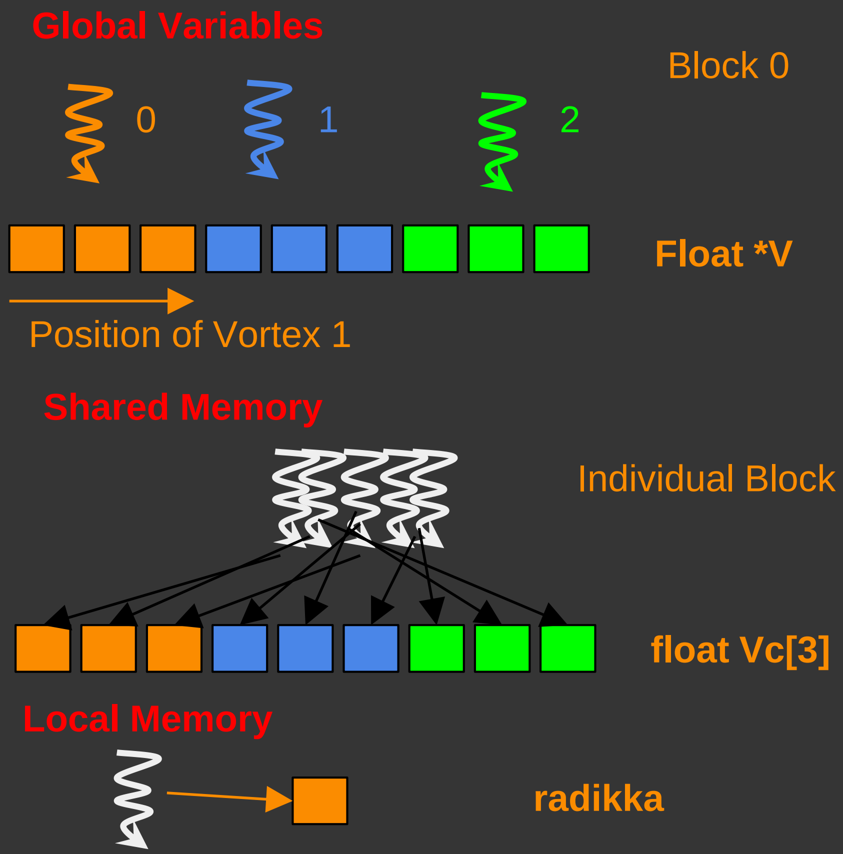

start
create 1D array of floats
Allocate memory in device
cudaMalloc()
initialize array randomly uniform distributed value
cudaMemcpy()
copying initalized array from host to dvice

Algorithm to the problem (continued)


GPU
CPU
itime <10000
yes
Kernal Call
<<< 700,700>>>
calculation of new position of vortex in device
stop
copying back new position from device to Host
cudaMemcpy()
yes
N< 700
No
yes
VN[i] >1
No
Assign back to cude
i++
cudaMemcpy()
changed new position copying back to device
call second Kernel
<<< 1,700>>>
calculate energy, speed, radius of highest magnitude of angular velocity
copying calculated data in kernel 2 from device to Host
cudaMemcpy()
Printf()

Parallelization to the problem (Continued...)

```
__global__

void NewVortexDistrub (float *V, float *O, float *VN,
float *ON, float *S, int N) {

                __shared__ float Vc[3];

                __shared__ float dVx[3];

                __shared__ float dVy[3];

                __shared__ float dVz[3];

                __shared__ float domdt[3];

 int tx = threadIdx.x;

 int bx =  blockIdx.x;

float radiika, dssss_dr;

float ssss;

float t1,t2,t3;
```



Parallelization to the problem (Continued...)

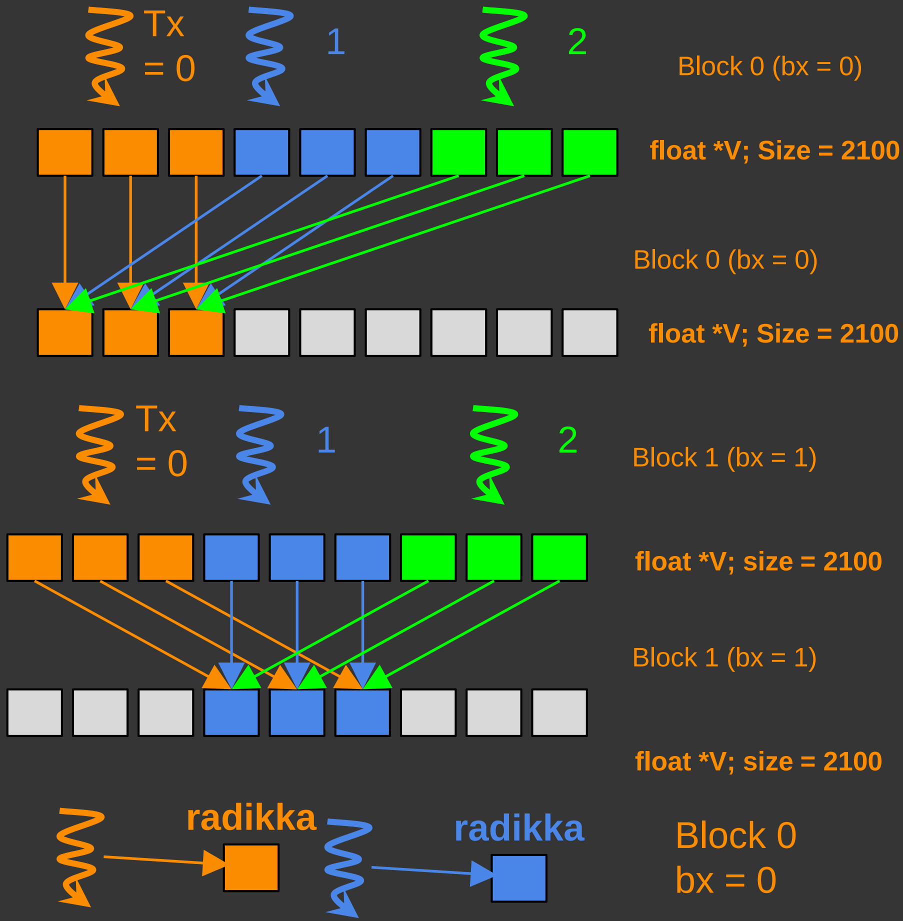
Parallel Code :

Number Blocks : 700 ; Number of threads in each block = 700

```
radiika = powf (V[(bx * 3) + 0] - V[(tx * 3) + 0], 2) + powf ( V[(bx * 3)
+ 1] - V[(tx * 3) + 1], 2 ) + powf ( V[(bx * 3) + 2] - V[(tx * 3) + 2], 2 );

dssss_dr = expf (-((3.1416f * 2.0f) / (S[tx]* S[tx] ) ) )*expf ( ( -
radiika ) * ( ( 3.1416f * 2.0f ) / ( S[tx] * S[tx] ) ) ) ;
```

Sequential code:

```
for (int ivorton = 0; ivorton<2100; ivorton++) {
    for (int induced = 0; induced<2100; induced++){

     vxx = Vortex[ivorton * 3 + 0] - Vortex[induced * 3 + 0];

     vyy = Vortex[ivorton * 3 + 1] - Vortex[induced * 3 + 1];

     vzz = Vortex[ivorton * 3 + 2] - Vortex[induced * 3 + 2];

     radiika = vxx*vxx + vyy*vyy + vzz*vzz;
```

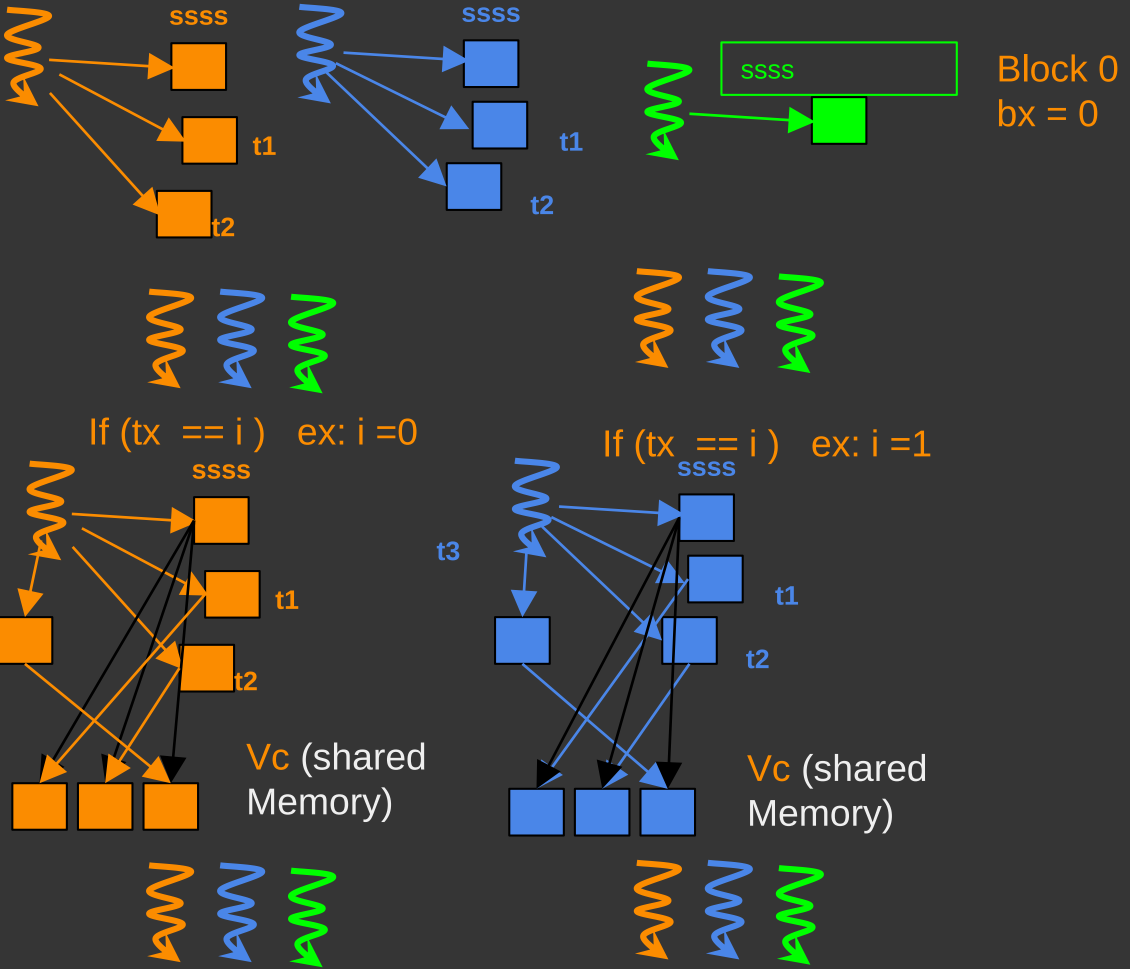


Parallelization to the problem (Continued...)

Parallel Code :

Addition of Vc[3], dVx[3], dVy[3], dVz[3], domdt [3] by all threads in a Block :

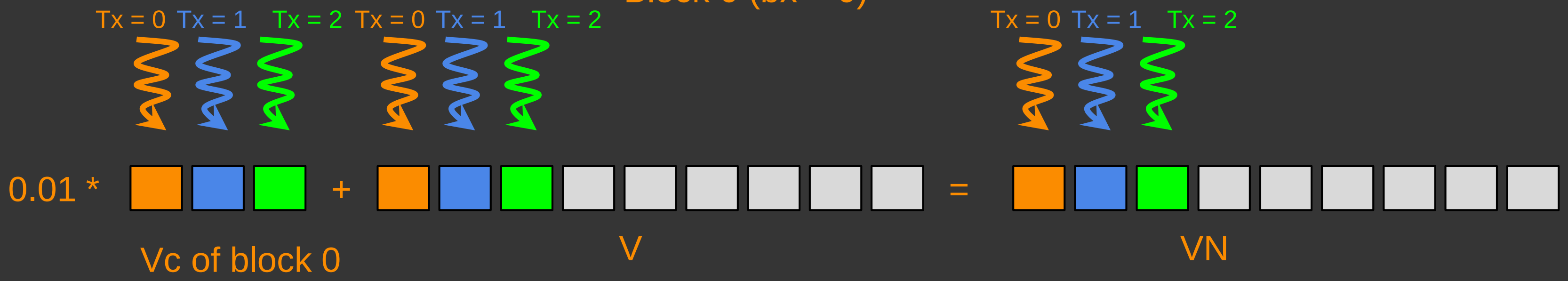
```
for (int i = 0; i<N; i++) {

            if (tx == i) {

            Vc[0] = Vc[0] + ssss * t1;

            Vc[1] = Vc[1] + ssss * t2;

            Vc[2] = Vc[2] + ssss * t3;
            ...............................
            ...............................
            ...............................

                  }

__syncthreads();

}
```



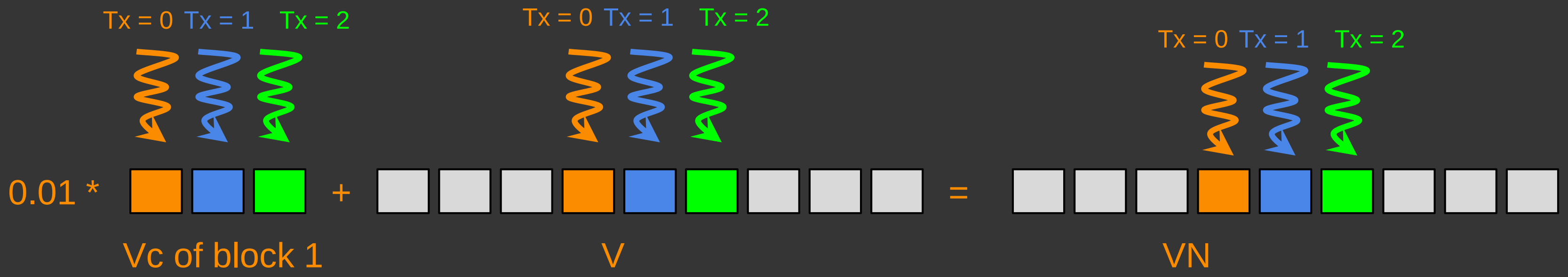
Parallel code:

```
if (tx < 3) {
            VN[(bx * 3) + tx] = V[(bx * 3) + tx] + 0.01f * Vc[tx];
            domdt[tx] = dVx[0] * O[bx * 3 + 0] + dVx[1] * O[bx * 3 + 1] + dVx[2] *
O[bx * 3 + 2];
            ON[bx * 3 + tx] = O[bx * 3 + tx] + domdt[tx] * 0.01f;
        }
```

Block 0 (bx = 0)



Block 1 (bx = 1)



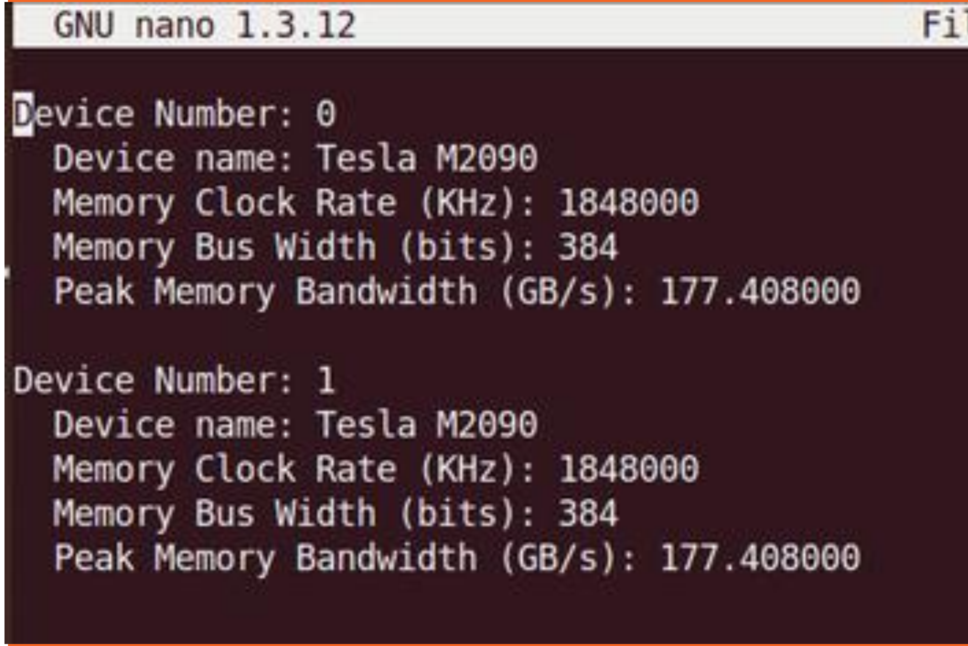
Tesla M2090

- The Tesla M2090 , a July 2011 launch is built on the 40 nm process, and based on the GF110 graphics processor, with DirectX 12.0.

- Compute Capability is of 2.0

- Connected with a PCIe 2.0 x 16 interfaces.

Device Query

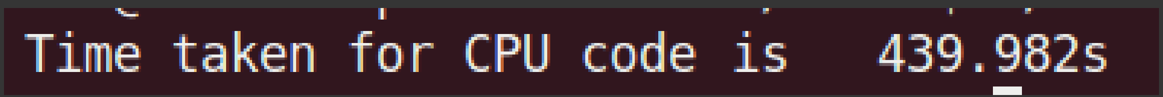


```
  GNU nano 1.3.12                                       Fi

Device Number: 0
  Device name: Tesla M2090
  Memory Clock Rate (KHz): 1848000
  Memory Bus Width (bits): 384
  Peak Memory Bandwidth (GB/s): 177.408000

Device Number: 1
  Device name: Tesla M2090
  Memory Clock Rate (KHz): 1848000
  Memory Bus Width (bits): 384
  Peak Memory Bandwidth (GB/s): 177.408000
```

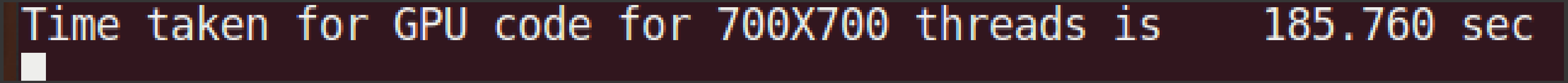
Memory Handling

- The Cudamalloc can allocate limited memory in device at once.
- For the current cuda device in server neptun1, we managed to allocate 2100 floats at once for in cudaMalloc.
- For 3000 floats and more, the cuda fails to allocate memory.
- Therefore, only 700 vortices are consider than 1000 vortices in this project.

Sequential V/s Parallel code



```
Time taken for CPU code is   439.982s
```



```
Time taken for GPU code for 700X700 threads is    185.760 sec
```

Percentage difference:- 57%

Note:- The higher time in GPU can be attributed to the less number of threads and blocks assigned for data transfer between GPU and CPU.. For a higher number of threads, the time could have been lesser because of the PCIe Bus.

Future Improvements

- A Gpu with compute capability 3.5 or higher will yield better results, as they offer Dynamic Parallelism and Unified memory Programming.
- In case the system allows the cudaMalloc to allocate huge memory in device at Once could be helpful in handling large data.
- Further improvements can be done in code by using cudaMallocPitch() and cudaMemcpy2D() for 2 Dimensional arrays.

References

[1]https://devblogs.nvidia.com/parallelforall/even-easier-introduction-cuda/

[2]https://github.com/harrism/mini-nbody/blob/master/cuda/nbody-block.cu

[3]https://www.olcf.ornl.gov/wp-content/uploads/2013/02/Intro_to_CUDA_C-TS.pdf

[4] David Kirk , Wen-mei hwu , Programming massively parallel processors, a hands on approach

[5]Jason Sanders , Edward Kandrot, Cuda by example, a general Purpose GPU Programming

Thank you !!