Parallel implementation of Particle-Mesh mapping using CUDA enabled GPUs

Guide: Prof Bhasker Chaudhury

201101102 Brijesh Kumar

Objective

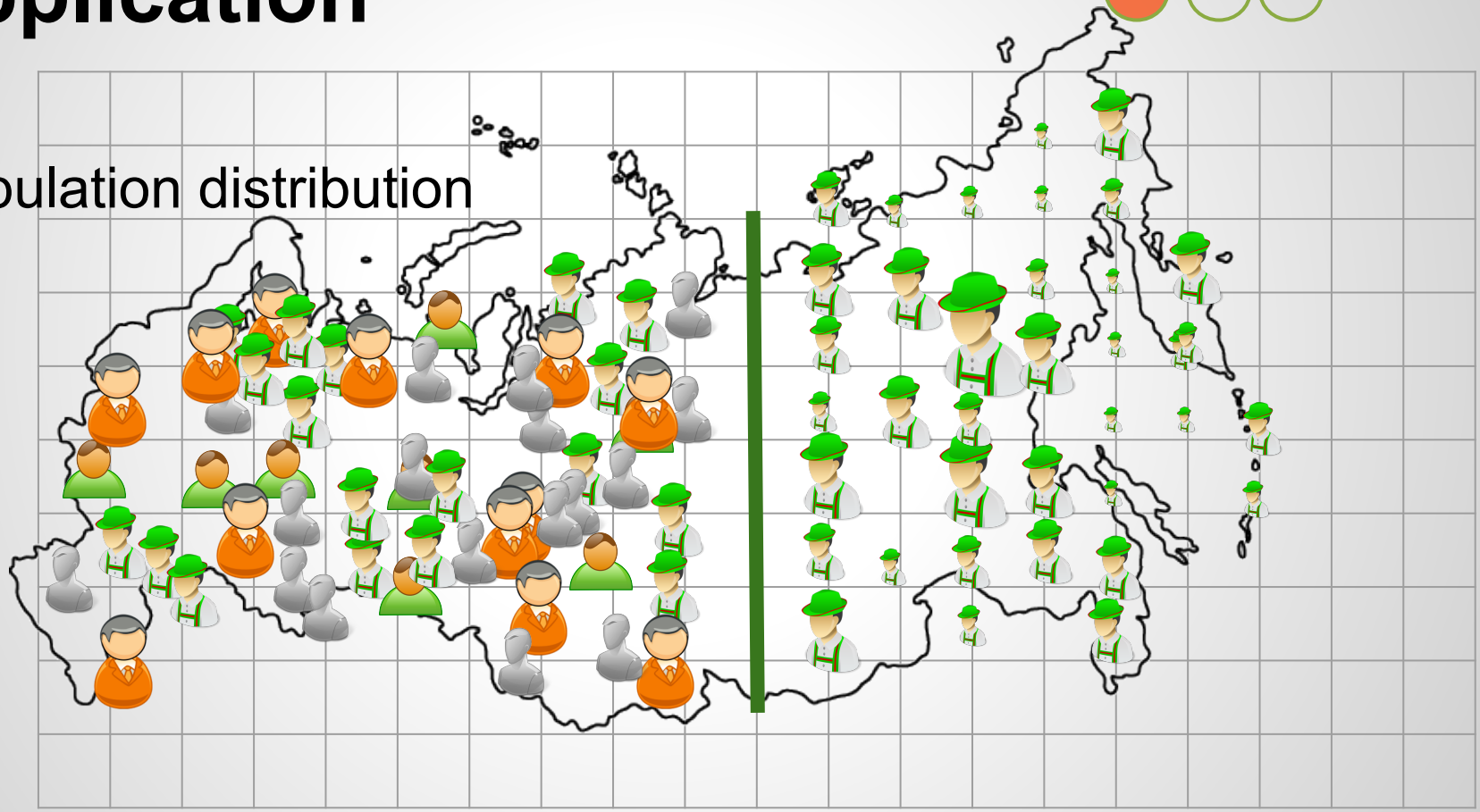
Particle to grid/mesh interpolation is an important technique which is used in several computational problems, such as the kinetic simulation of plasmas using Particle in cell algorithms. This interpolation procedure is computationally very expensive when the number of particle is more than few millions.

Objective

In this project we are trying to map randomly scattered particle to discrete system. We will use serial and parallel implementation to compare the results.

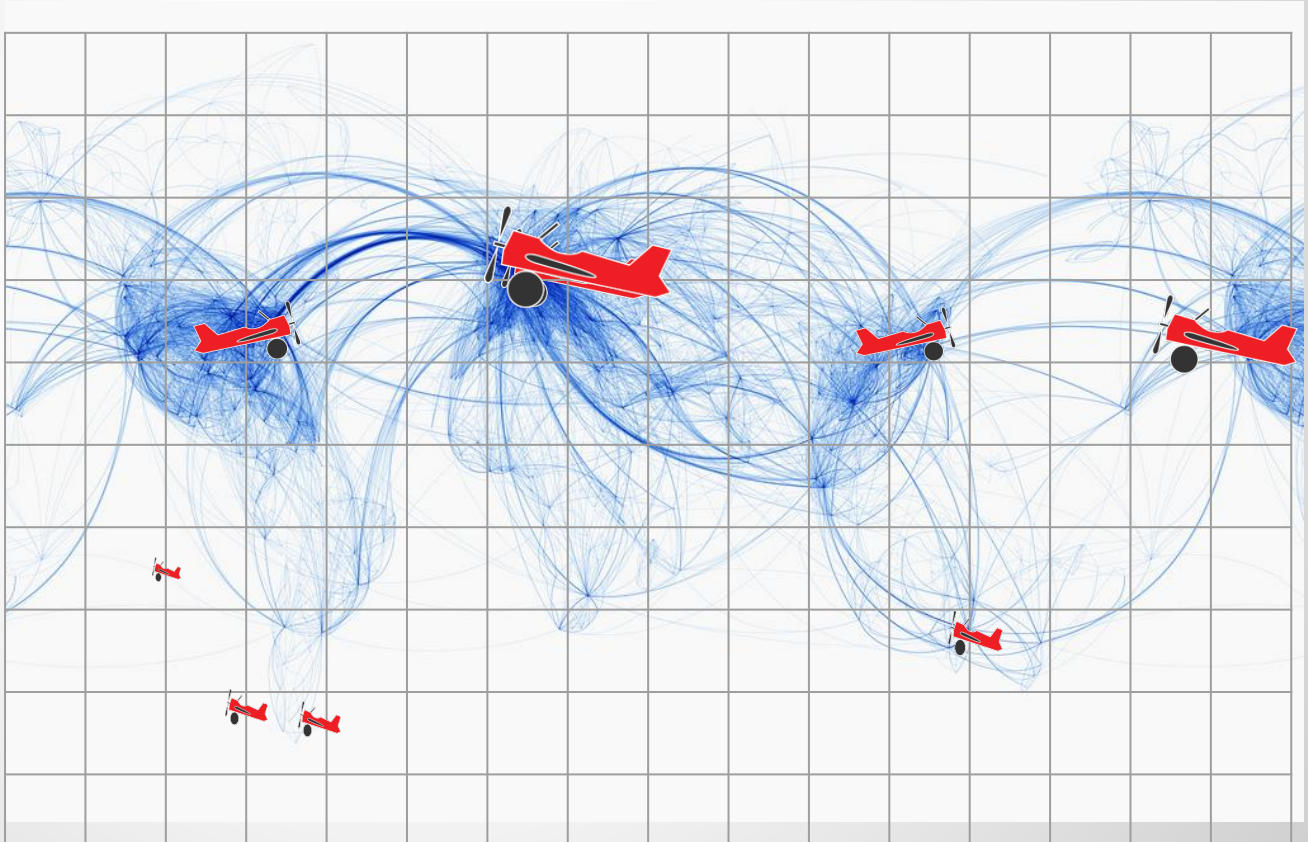
Application

Population distribution



Application

Air traffic control

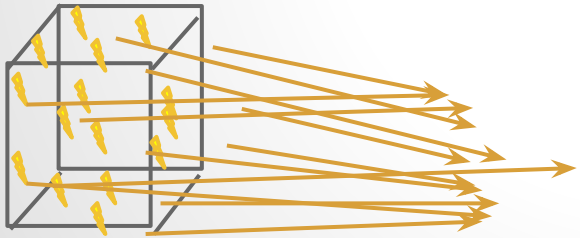


Application

Physics

gauss law

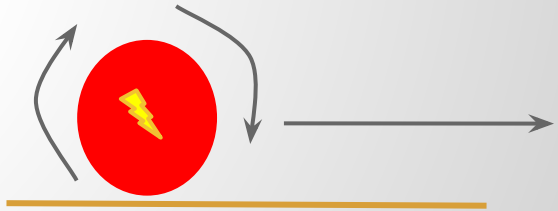
$\nabla \cdot \mathbf{E} = \rho/\varepsilon 0$



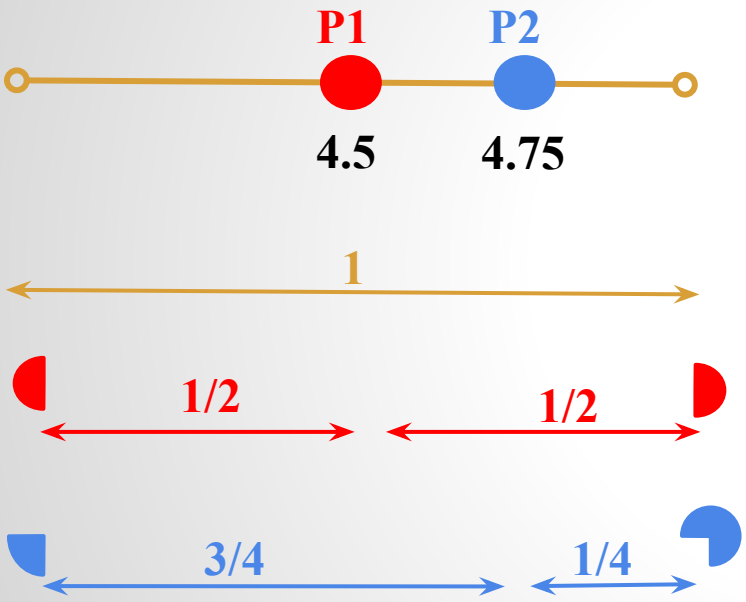
Lorentz equation

$F=q\mathbf{E}$

Electrodynamics Motion



Mapping Algorithm



for each (P1 & P2) {

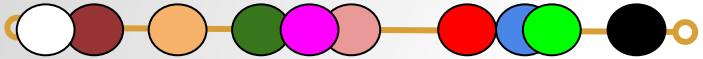
	P1	P2
Left point = (Floor).position	4	4
Right Point = Left+1	5	5
fLeft = position - Left	1/2	3/4
fRight = Right - position	1/2	1/4

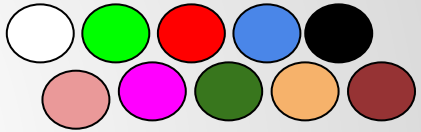
value (left) = (own value) + (fR x magnitude)

value (right) = (own value) + (fL x magnitude)

}

Mapping Algorithm





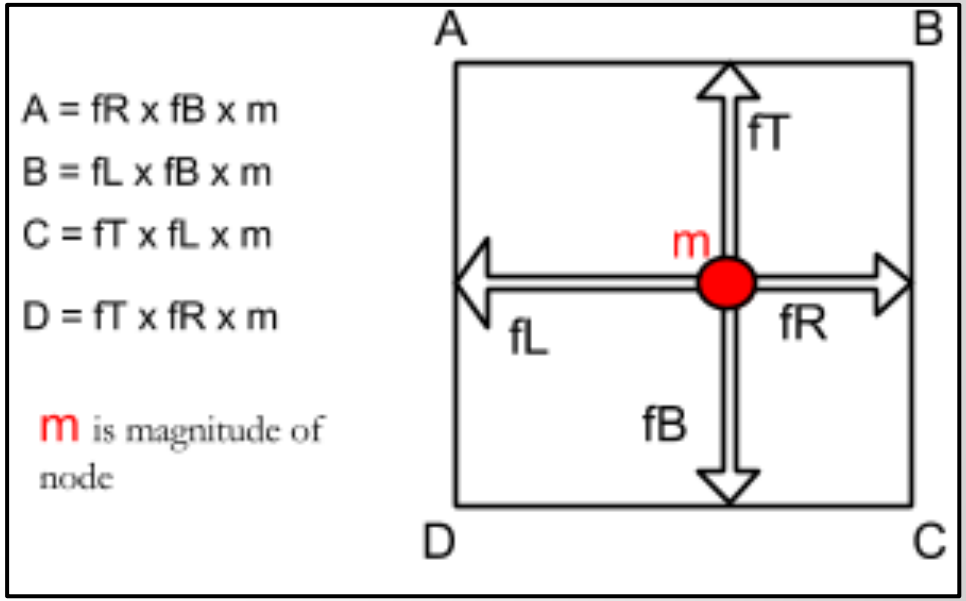
```
for each particle  {
Left point  = (Floor).position //coordinate search
Right Point = Left+1
fLeft    = position - Left   // " f " fractional distance
fRight = Right - position
//Update coordinate
value (left) = (own value) + (fR x magnitude)
value (right) = (own value) + (fL x magnitude)
}
```

Mapping Algorithm

***foreach* particle p(i) in list**

```
  left = (floor).p(i).x ; //Coordinate search
  bottom = (floor).p(i).y ;
  right = left+1;
  top = bottom+1;
//Fractional distance
  fL = p(i).x – left ;     fR = 1-fL;
  fB = p(i).y-bottom ; fT = 1-fB ;
// updating coordinates
(left, top)     = A;    (right, top)=B;
(left,bottom)=C;    (right,bottom)=D;
```

end



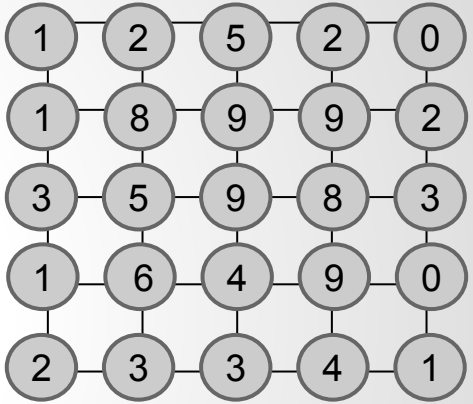
For each particle mapping approx 10 flops cycle is required

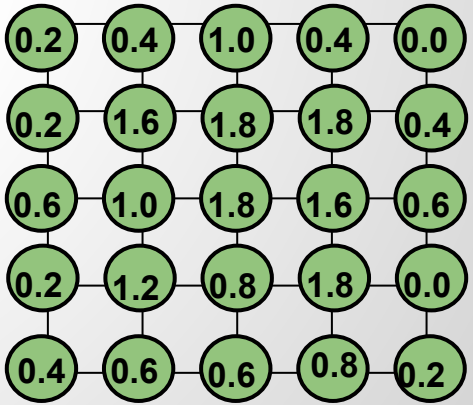
Mapping Algorithm

Normalisation

Variable average = (total particle) / (total mesh coordinates);
foreach position in grid
node value = (own value) / (average);
end
// if ratio > 1 means node value is more than average;
// if ratio < 1 means node value is less than average;

E.g. Total particle = 125
Total mesh node = 25
Average particle per node = 5





Serial Implementation

```
foreach particle i
        p(i).x = random position.x;
        p(i).y = random position.y;
end
foreach position in mesh
        value mesh( x,y ) = 0.0;
end
```

Mapping algorithm

Normalization algorithm

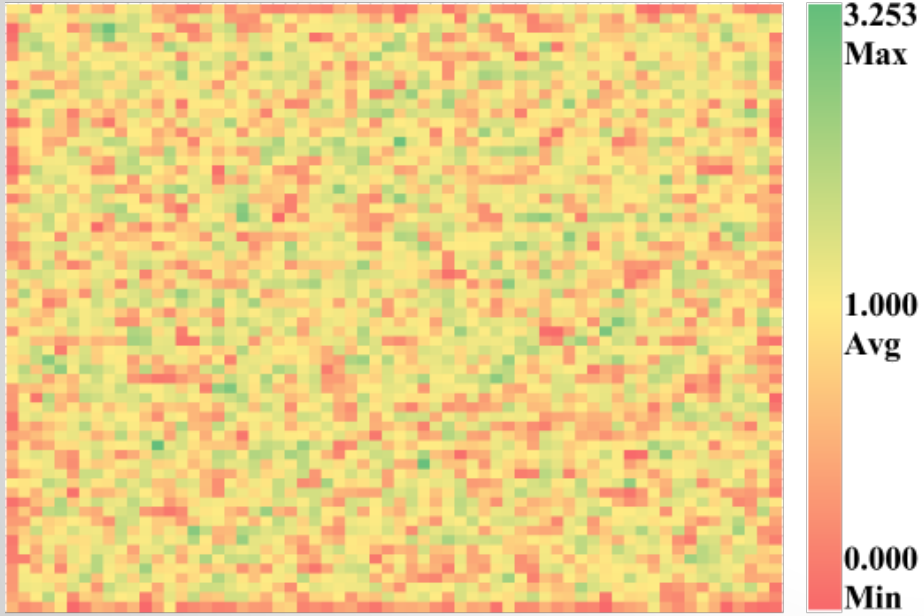
open file;

store information;

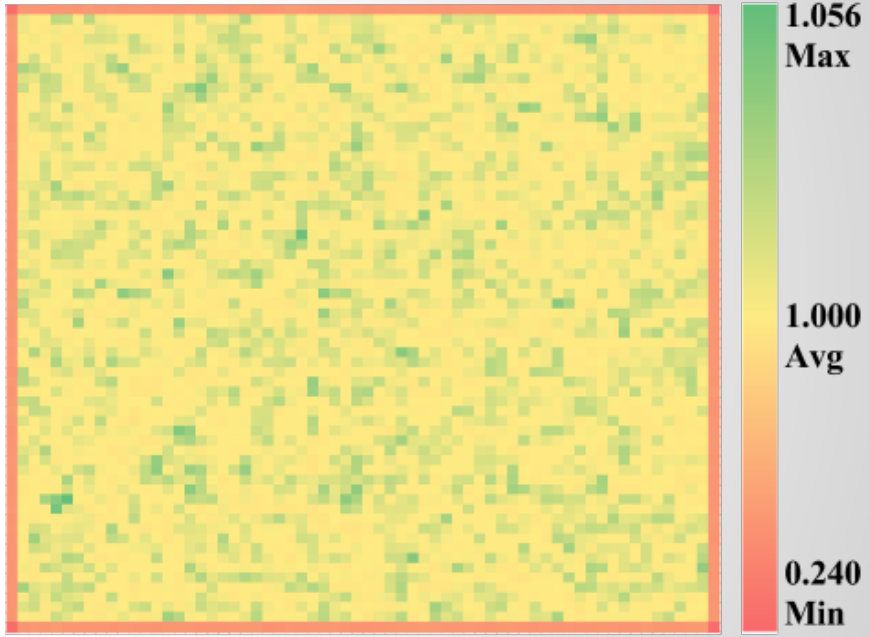
close file;

Property	Details
Cache	3.0MB
Clock Speed	2.90 GHz
Number of cores	4
Thermal design power	28 W
Max memory bandwidth	25.6GB/S

Serial Implementation

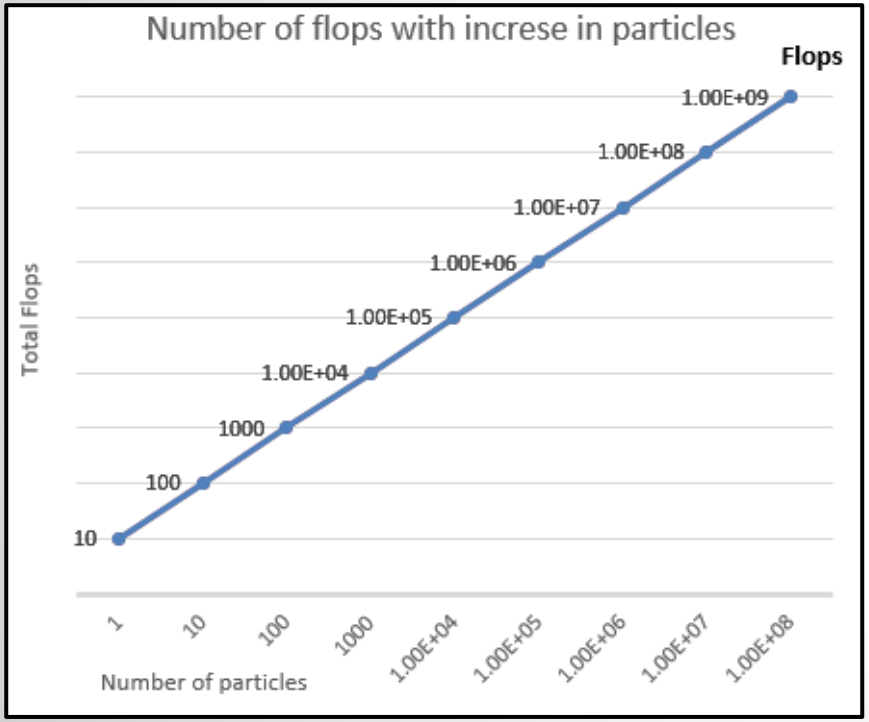


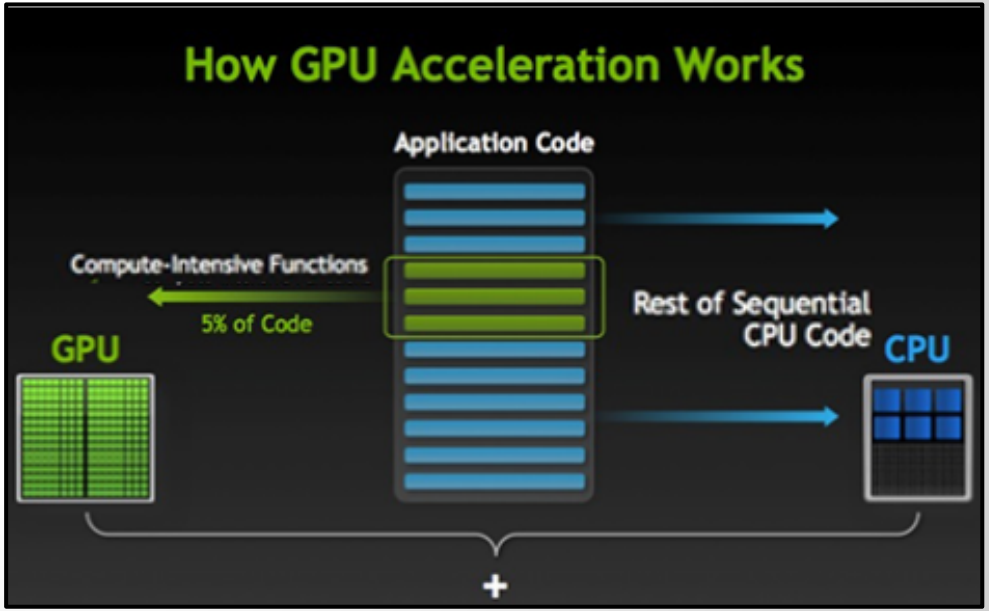
Testing 64X64 grid 8.5×10^3 particles



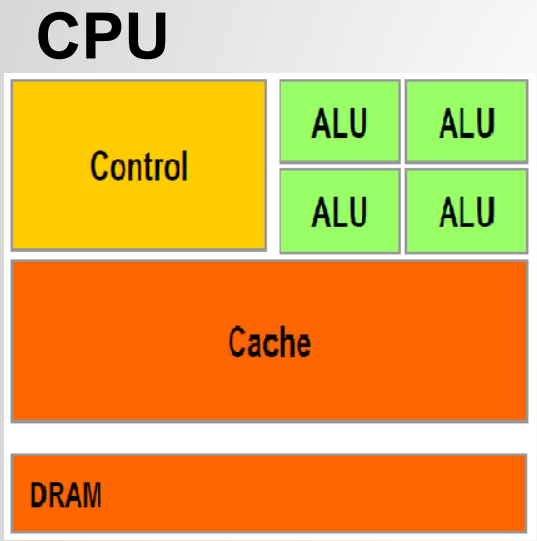
Testing 64X64 grid 8.5×10^6 particles

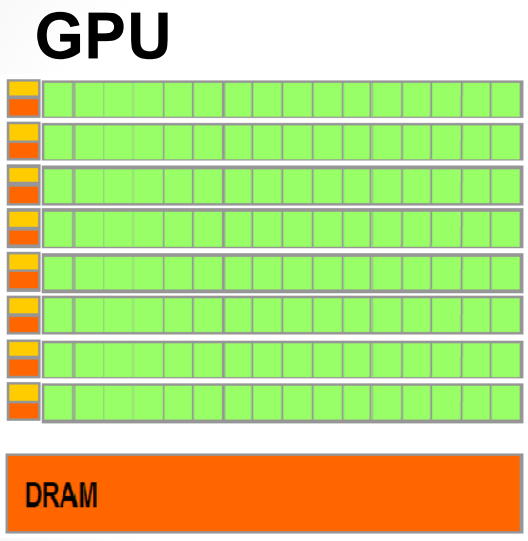
Serial Implementation

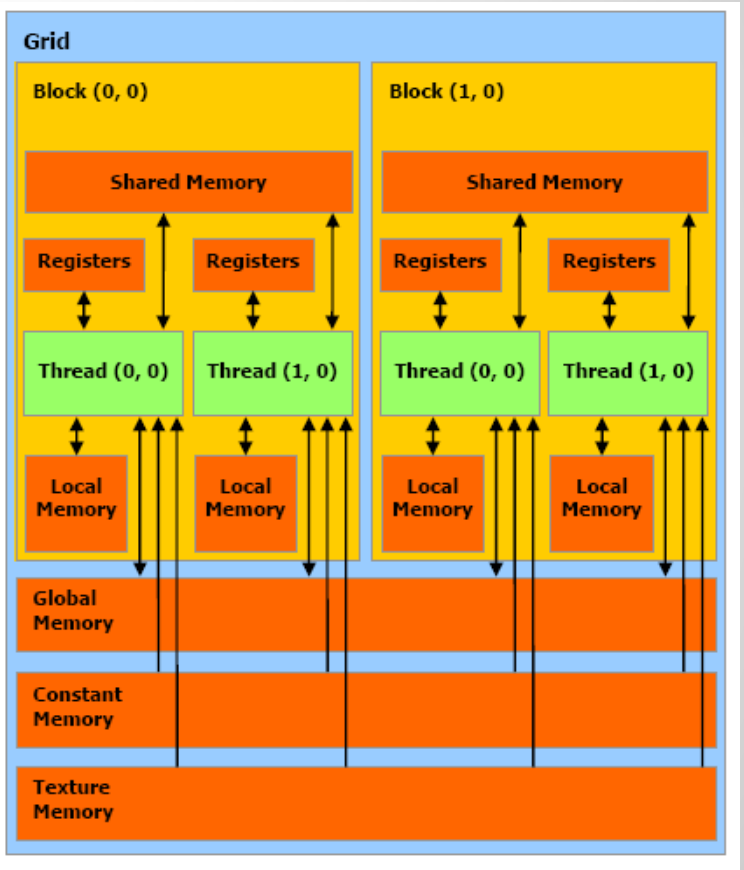




CUDA enabled GPU





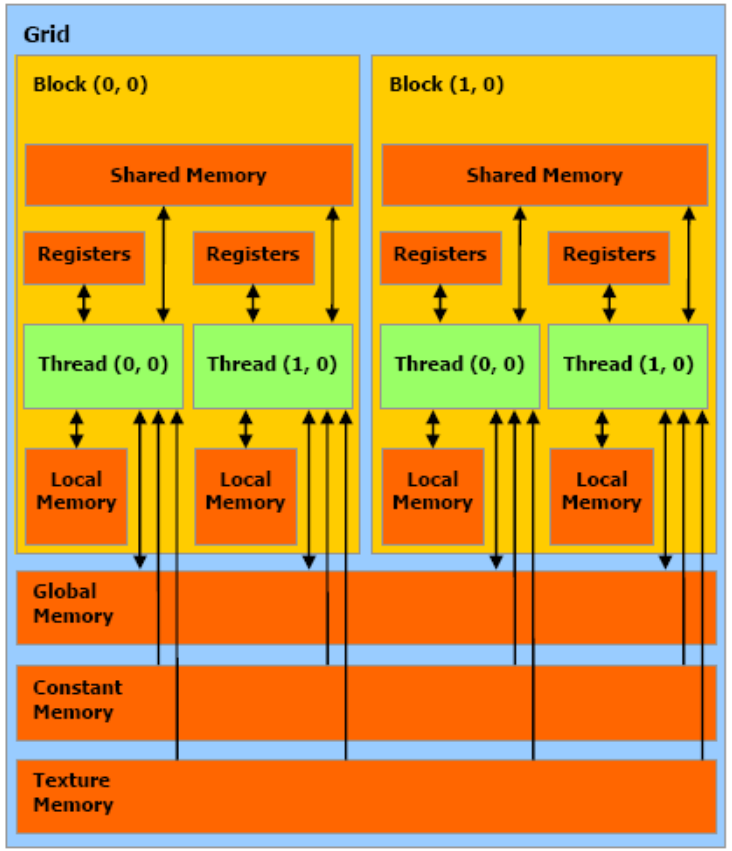


Memory

CUDA enabled GPU

Process

- Declare Host variable
- Declare Device Variable
- Allocate device memory
- Transfer data to device memory
- launch parallel threads
- process data
  - Find thread id
  - Mapping
  - normalisation
- Transfer data from device memory to cpu
- store information in file
- Deallocate memory



Parallel Implementation

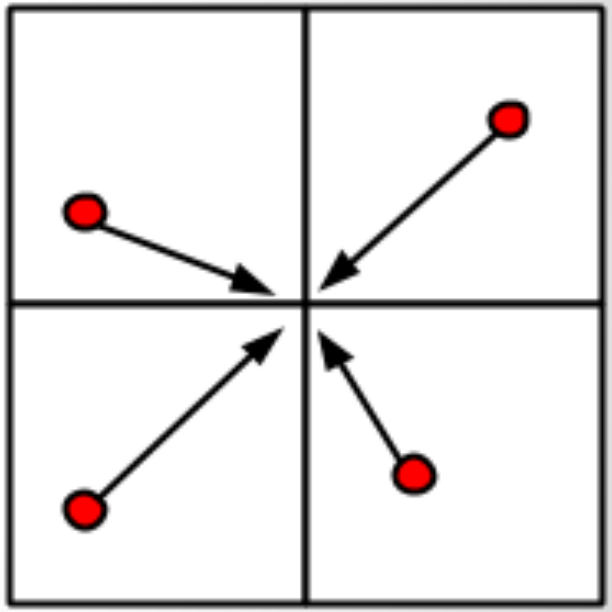
Card	K40
Max warp per MP	64
Max thread per MP	2048
Max thread block per MP	16
Registers per MP	65536
Max register per thread	255
MP count	15

```
__global__ void parMap( parameters ){
thread ID= blockDim.x*blockIdx.x +
threadIdx.x;
        // Mapping algorithm
        // Normalization algorithm
}
```

```
Int main(){
  Declare host variable;
  Declare host variable;
// initialize host variable.
    cudaMalloc( );
    cudaMemcpy( host to device);
// define block dimension
        Dim3 bD;   //block dimension;
        Dim 3 gD;   //grid dimension;
Launch kernel<< bD, gD>> (parameters);
        cudaMemcpy(from device to host);
        Store data into file;
}
```

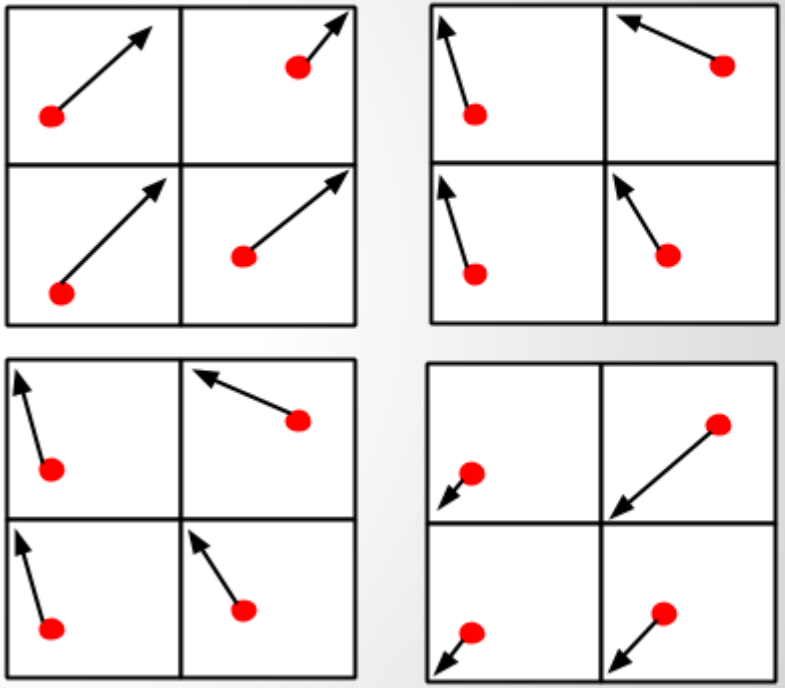
Parallel Implementation

- Collison



Parallel Implementation

- Cyclic memory write



Result

Number of particles	Timing		
	CPU	GPU	Ratio
256 K	13.1	0.7	18.7
512 K	21.9	1.3	16.8
1 M	42.1	2.7	15.6
2 M	77.2	5.3	14.5
4 M	151.1	11.2	13.4
8 M	297.2	24.1	12.3
16 M	589.6	51.2	11.5

Result

```
==1466== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 88.88%  1.76606s       999  1.7678ms  1.7472ms  1.7985ms  parMap(float*, float*, int)
 10.94%  217.30ms      1000  217.30us  215.68us  1.0644ms  [CUDA memcpy HtoD]
  0.18%  3.6073ms         1  3.6073ms  3.6073ms  3.6073ms  [CUDA memcpy DtoH]
==1466== API calls:
 70.81%  1.92092s      2998  640.73us  3.0620us  1.7994ms  cudaEventSynchronize
 13.62%  369.53ms      1001  369.16us  336.37us  4.6231ms  cudaMemcpy
  9.85%  267.20ms      1000  267.20us  162.25us  724.20us  cudaMalloc
  3.67%  99.650ms         8  12.456ms  1.3830us  99.639ms  cudaEventCreate
  0.03%  829.41us       166  4.9960us     350ns  174.95us  cuDeviceGetAttribute
  0.03%  784.52us       999     785ns     627ns  2.9240us  cudaConfigureCall
  0.02%  427.34us         2  213.67us  171.67us  255.66us  cudaFree
```

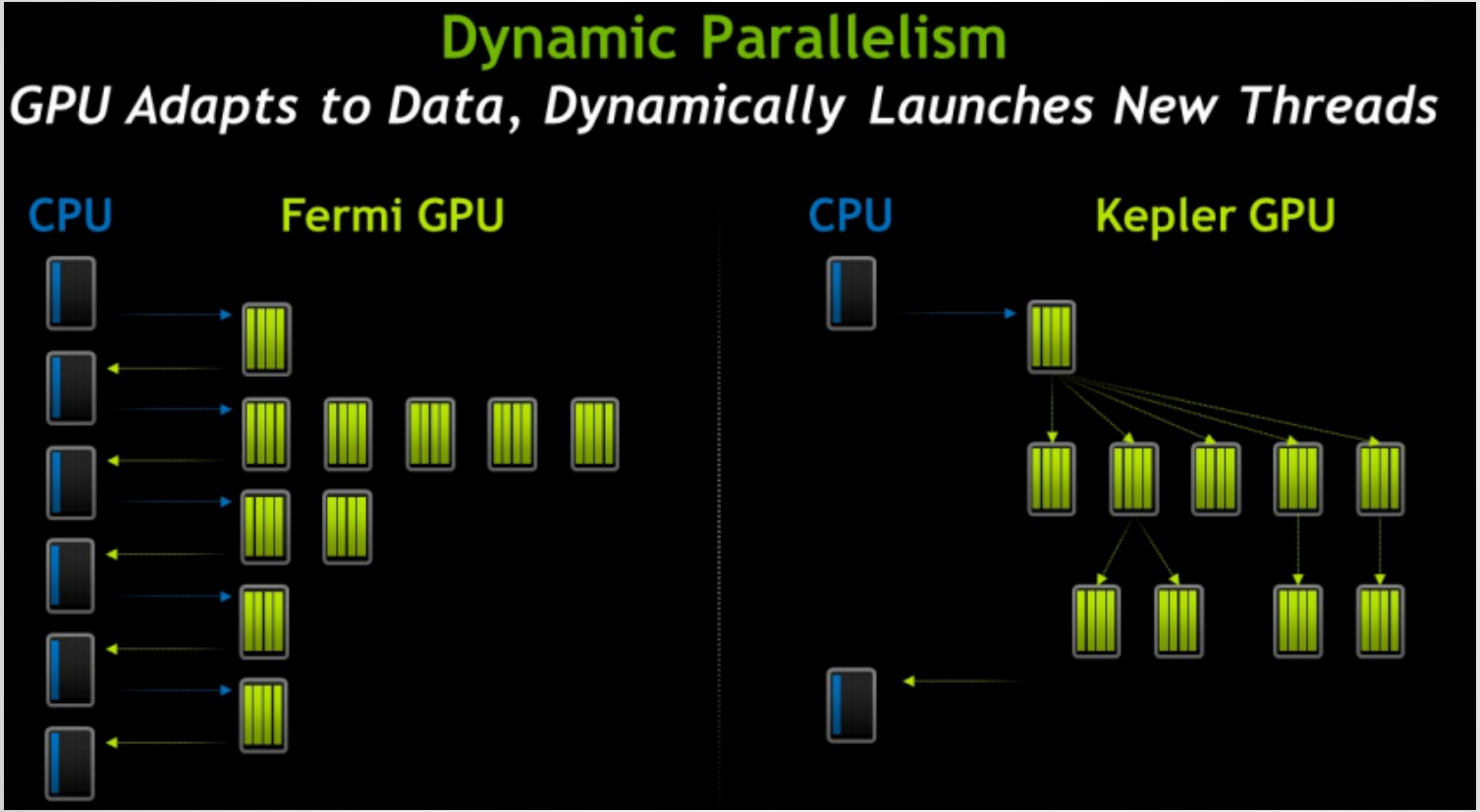
Learning’s

- GDB compiler
- Memory limit
- Functions not available: basic host fuction dont work in device
  - rand
  - floor
  - #include <math.h>
- range of variables
- server lag

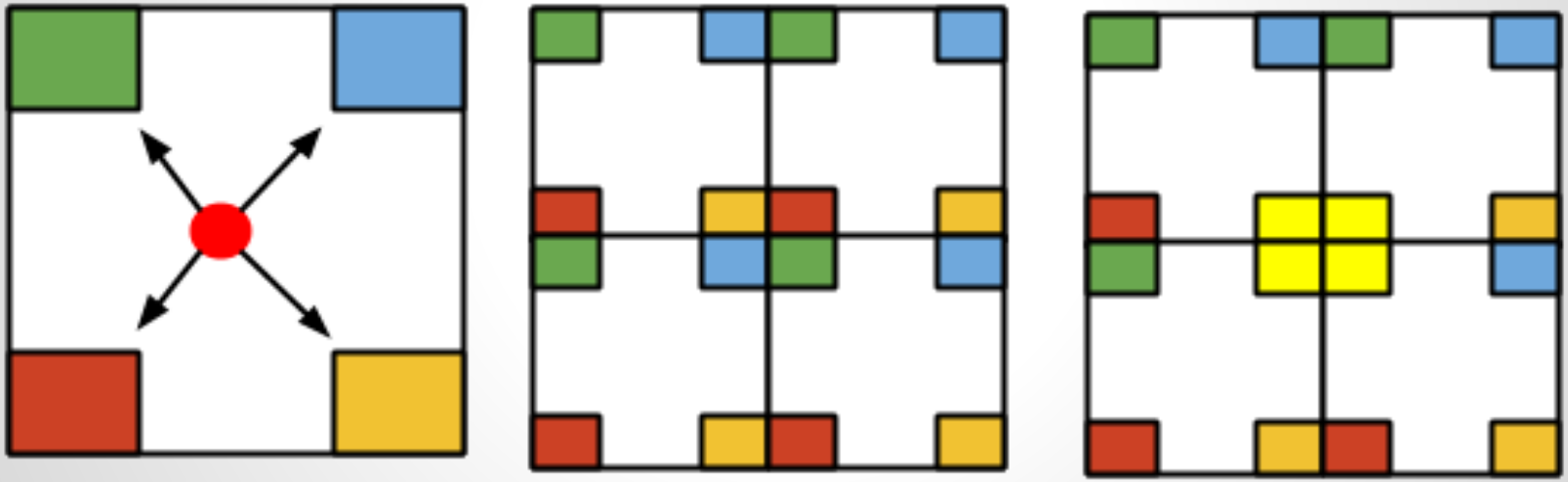
Learning’s

- ssh
- mobaxtream
- sftp
- nvcc compiler
- ocupancy calculater
- nvprof

Future work


Dynamic Parallelism
GPU Adapts to Data, Dynamically Launches New Threads
CPU
Fermi GPU
CPU
Kepler GPU

Future work



Acknowledgement

For my project I have used following machines

- 10.100.71.159 DAIICT,India
- 103.21.126.139 GSU, US
- 131.96.5.210 IIT-B,India
- 147.97.156.17 ASU,US

Acknowledgement

I have used following references

1. Shane Cook Book ‘CUDA programming , a developer guide’.
2. David B. Kirk, book ‘Programming massively with parallel processor’
3. F. Buyukkececi, O. Awile, I.F. Sbalzarini, A portable openCL implementation of generic particle-mesh and mesh-particle interpolation in 2D and 3D, sciencedirect 2013
4. G.Stantchev, W. Dorland,N.Gumerov, Fast parallel particle-to-grid interpolation for plasma PIC simulation on the GPU, J.Parallel Distrib, comput (2008)

Acknowledgement

I would like to thanks my guide Prof. Bhaskar Chaudhury for his clear guidance.

Thank You