# Thesis Planning GPU/CUDA basics Histogram kernel

Tsapatsaris Panagiotis (Ntua)
Professor:
Dimitrios Soudris (Ntua)
Advisors:
Konstantinos Iliakis (Ntua,Cern)
Sotirios Xydis (Ntua)

# Diploma Thesis Planning

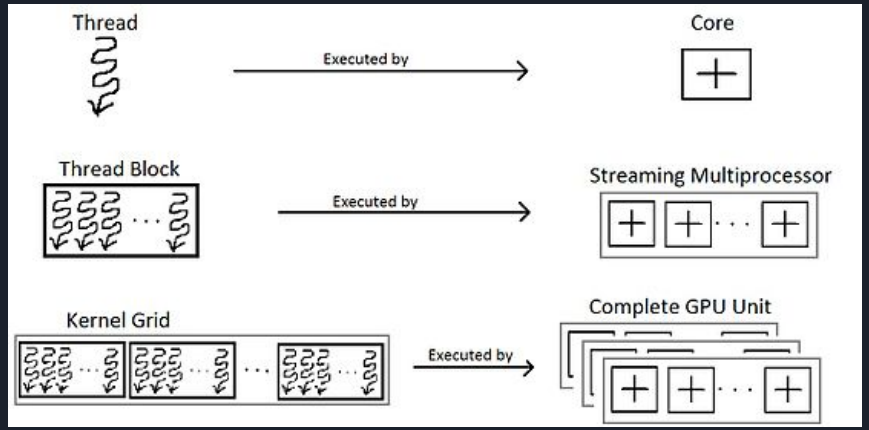
- November to January — studying(GPU architecture , Cuda programming model)
- February — break for exams
- March to …. — implementing and optimizing

Implementing And Optimizing in 2 Phases

- implementing and optimizing each kernel alone
- merge kernels and optimize again

# Comparison between CPU and GPU Vector Add

```
for (int i=0; i<size; i++)
        c[i] = a[i] + b[i]
```




```
int tid = threadIdx.x + blockIdx.x*blockDim.x
if (tid<size)
        c[tid] = a[tid] + b[tid]
```

We call a kernel like that:

```
kernel_name<<<grid,block>>>
```

grid and block can be 1d, 2d or 3d

What if threads are lesser than size?

```
for (int i=tid; i<size; i=i+blockDim.x*gridDim.x)
        c[i] = a[i] + b[i]
```

# GPU Basic Operations

Memory Operations

- Allocate memory in the GPU
- Copy from CPU to GPU (host to device)
- Execute the computations
- Copy the results from the device to host

# Histogram Function

Input

- n particles each with a value
- k bins (k ~n/1000)
- cut_right , cut_left

Histogram function measure the frequency of these particles, so for each particle we

- check if it is inside the limits
- find the target bin
- increase its frequency by 1

# Why do we need atomic operations

starting value of i is 0

thread a thread b

i++; i++;

After these threads complete their execution what is the value of i?

Answer: We do not know Example

thread a thread b final value of i is 1

read i(0)

add 1 (1)

read i(0) That is the reason
need atomic oper

store i(1)

add 1(1)

store i(1)

# Histogram Kernel

**Only atomic version**

each thread atomically increase target bin's frequency by 1

Problem:
The code is being serialized because of conflicts

**Shared Memory version**

Each block of threads computes a local histogram and then add it to the global histogram atomically

Pros:
Less conflicts in the shared memory
Cons:
More operations

# So what if the local histogram can not fit in the shared memory?

**Hybrid version**

Each block of threads has a local histogram for a subgroup of bins and increase the rest of them atomically

Example:

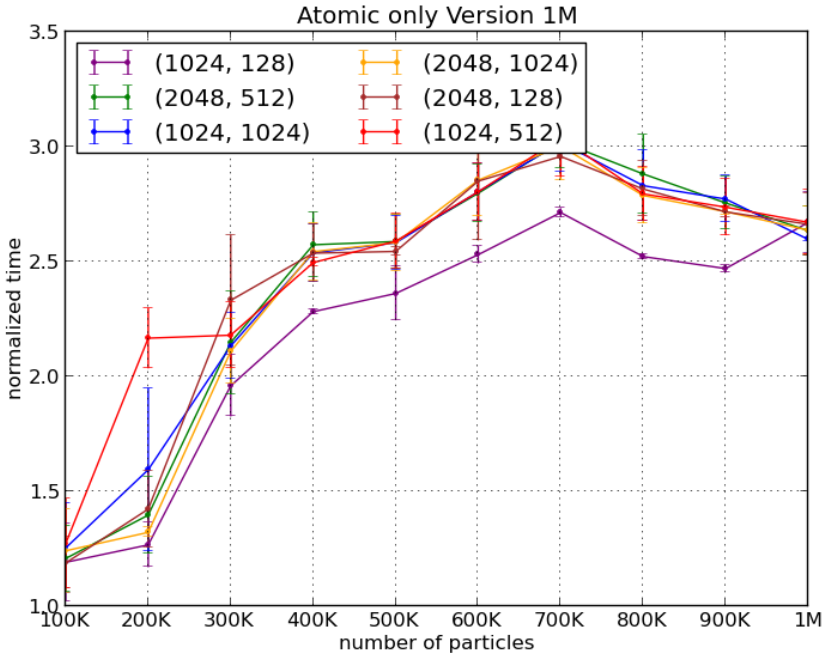
shared memory capacity = 4
histogram bins = 10
block 0 → [0,3] locally and the rest globally
block 1 → [4,7] locally
block 2 → [8,10] locally
block 3 → [0,3] locally

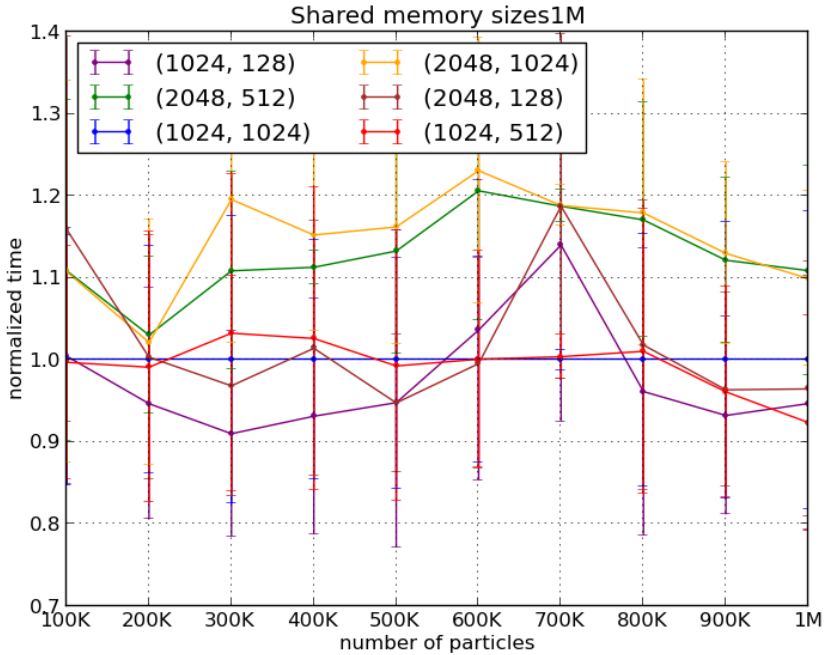
# Parameters for optimizing

We need to find the optimal values for block size and grid size

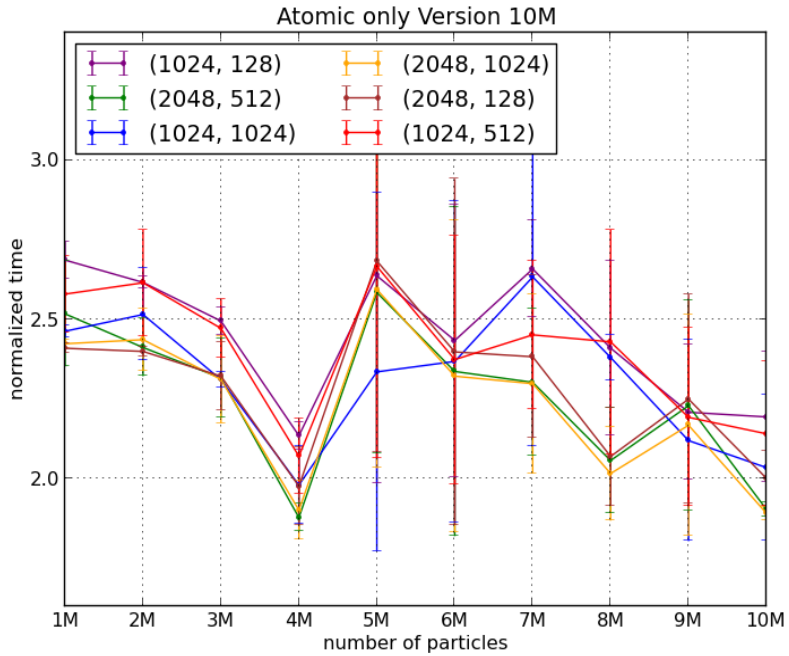
We also need to decide which algorithm to choose

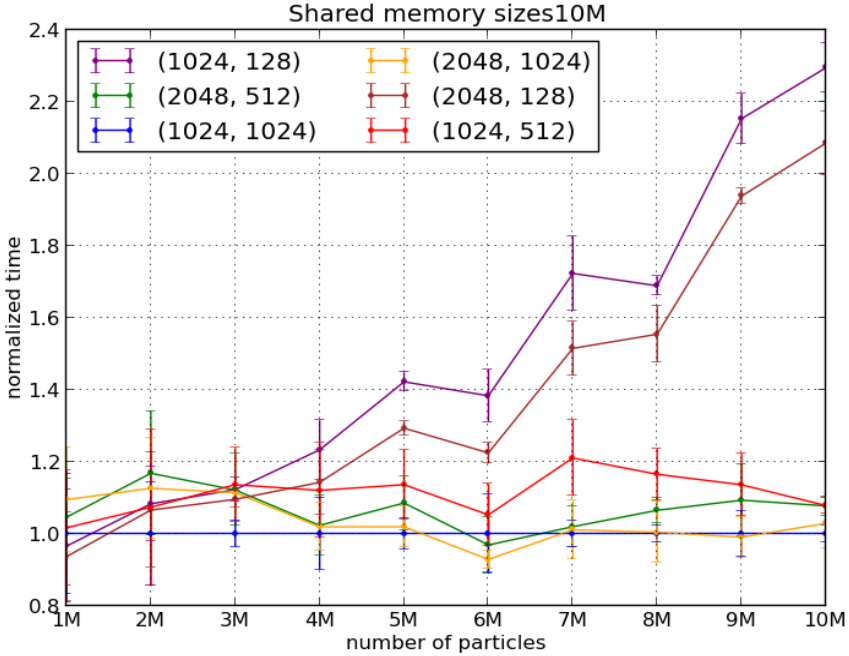
# Results for small sizes
# normalized with (1024,1024) shared memory
# 1000 turns, slices ~particles/1000



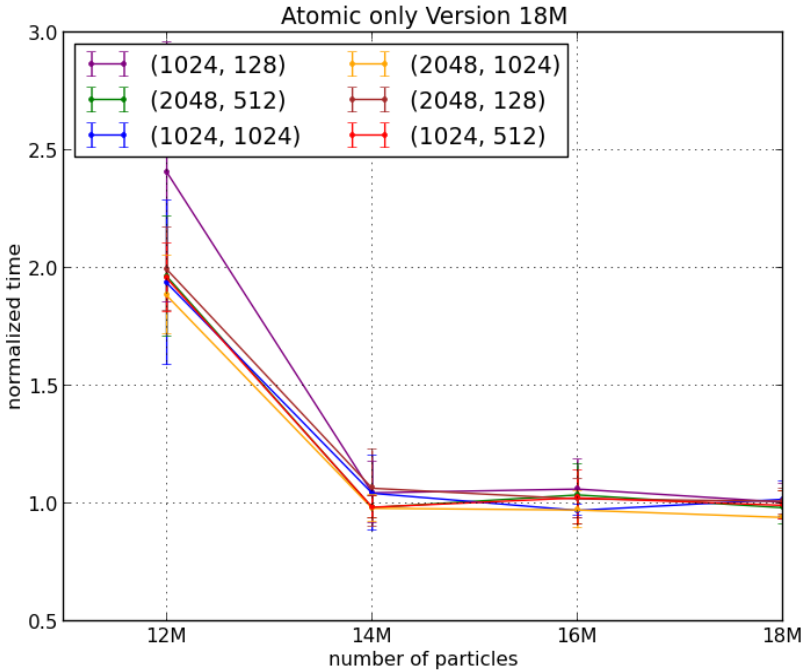



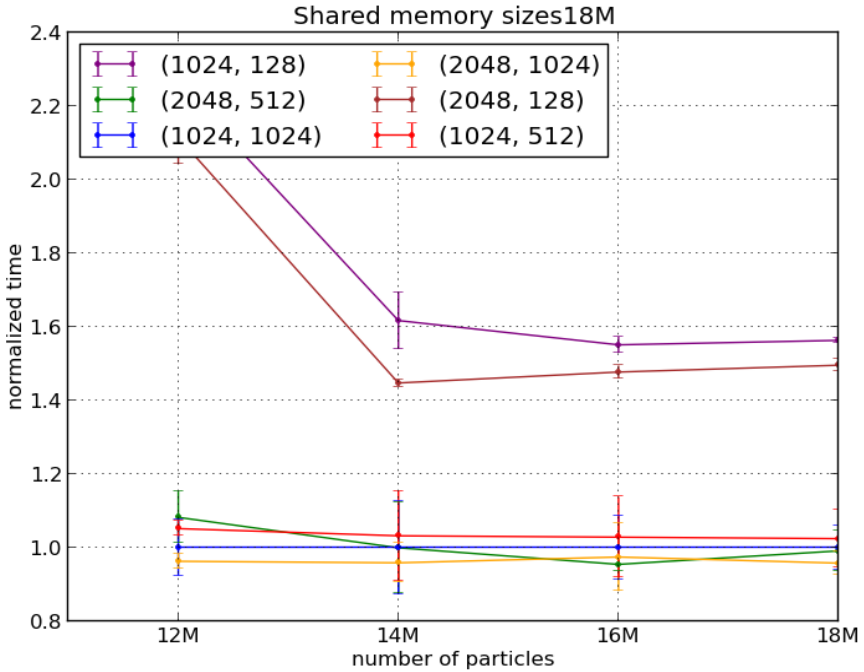
# Results for big sizes
# normalized with (1024,1024) shared memory
# 1000 turns, slices ~particles/1000



# Results for huge sizes
# normalized with (1024,1024) shared memory
# 1000 turns, slices ~particles/1000



# Future work

- try to find dynamically the best couple of (grid size,block size)
- test on normal distribution
- test different number of bins

Thank you very much!

Questions