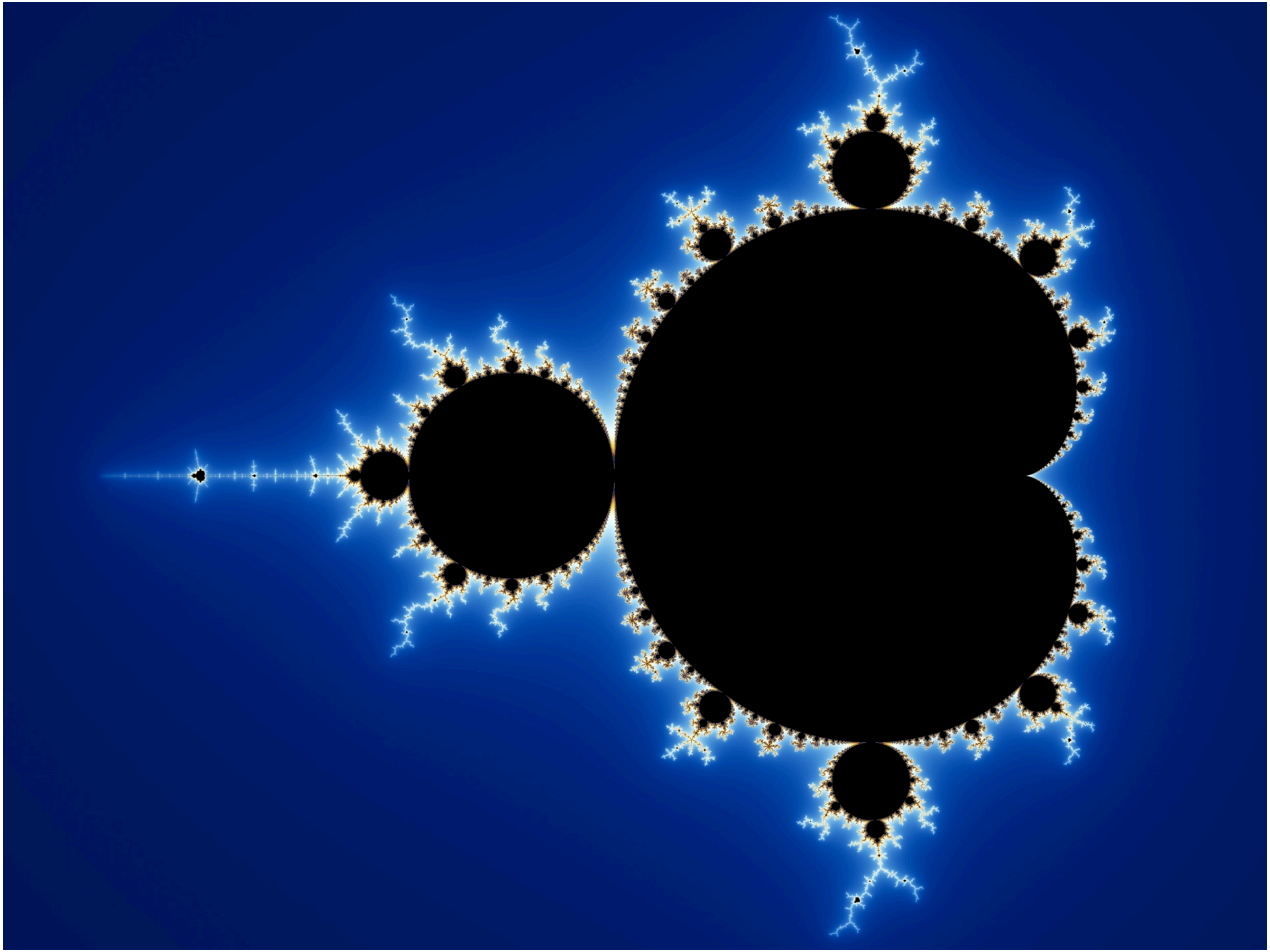
HEIGHT MAP GENERATION WITH GPU's

Ravi Kiran Chilakapati
CSE 633: Parallel Algorithms
Fall 2011

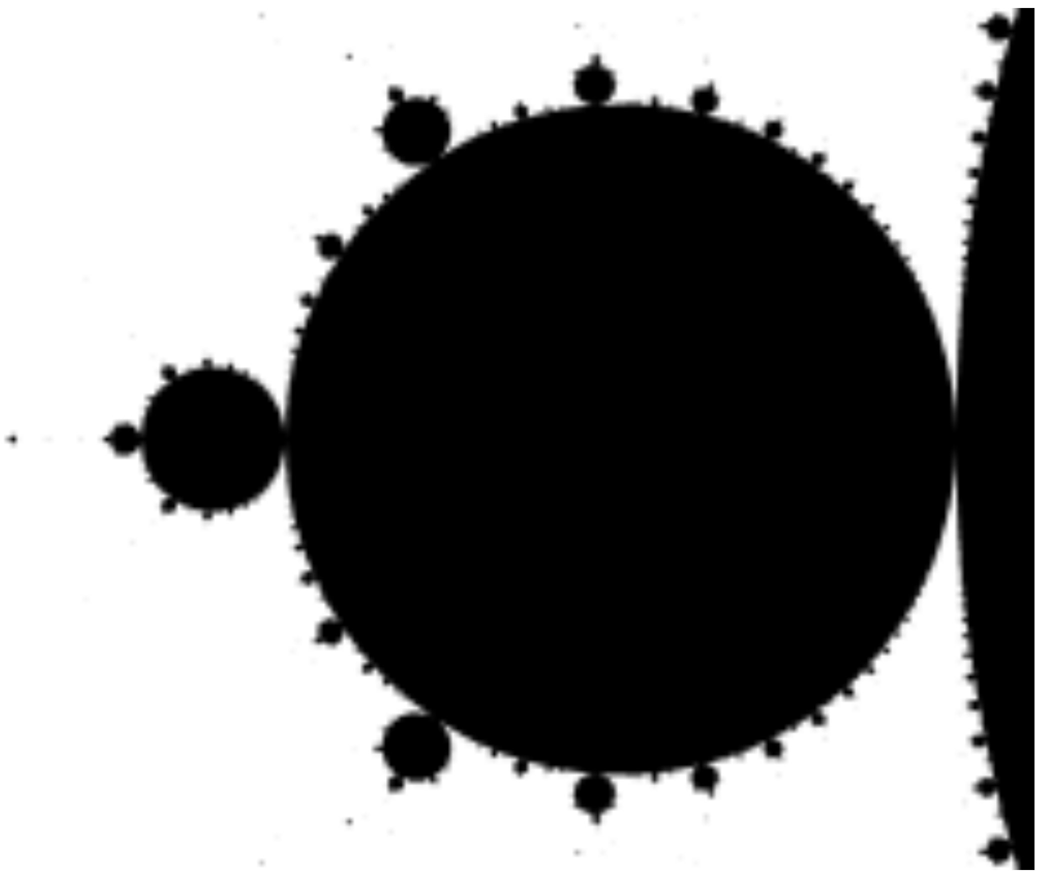
FRACTALS

- Self-Similarity
- Fine structure at arbitrarily small scales
- Simple and recursive definition
- Everywhere in nature – snowflakes, clouds, mountain ranges, lightning bolts and even in vegetables!



Source: Wikipedia

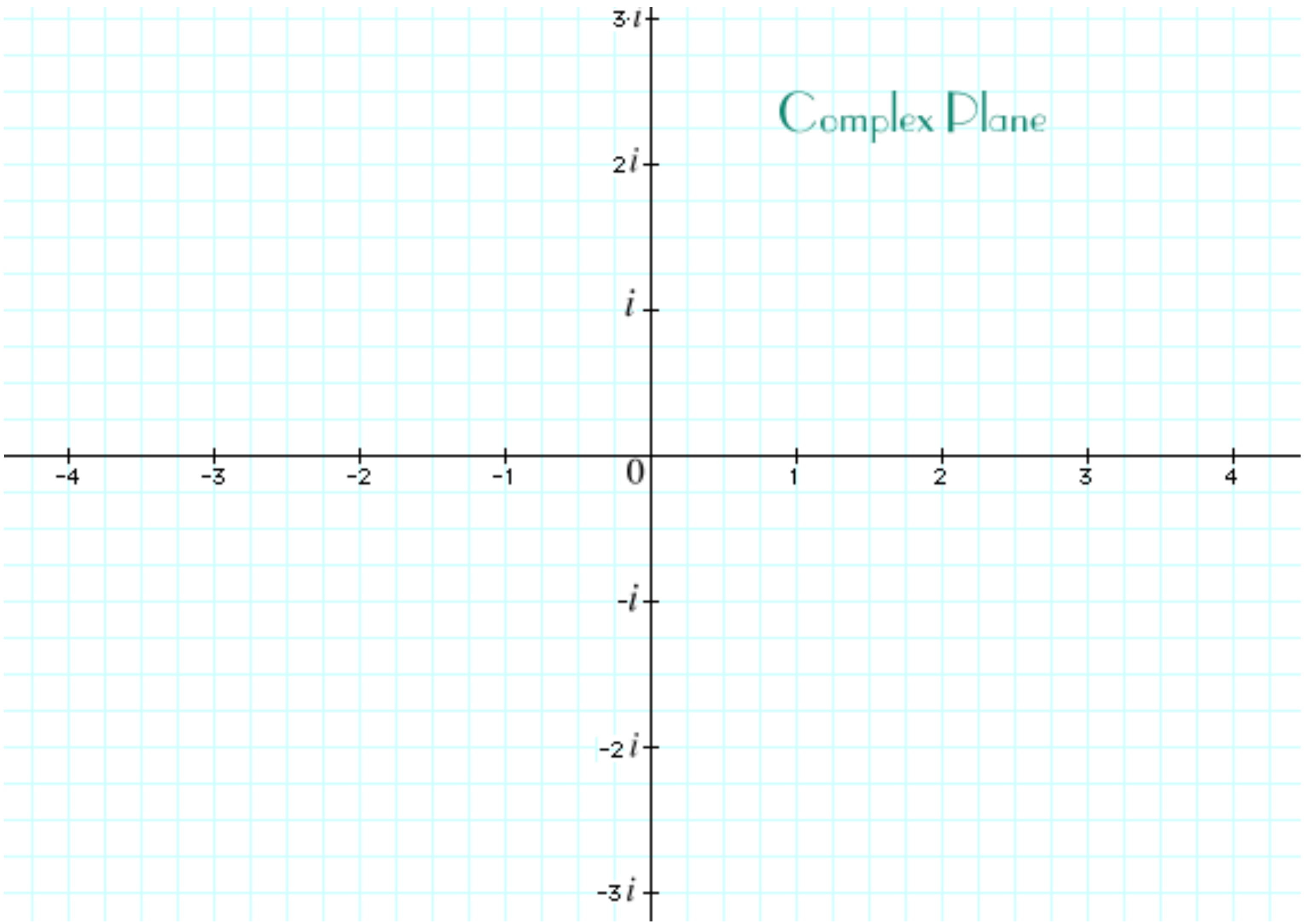
SELF SIMILARITY



Source: Wikipedia

FRACTAL GENERATION

- The "simple" part is exaggerated!
- Recursion perfectly captures the self-similarity of fractals
- Usually generated in a complex plane
- Escape-time; Iterated functions; Random fractals; Strange attractors; L-systems
- Julia set, Mandelbrot set, Nova fractal, Sierpinski carpet, Koch snowflake, Brownian Tree



Source: http://www.clarku.edu/~djoyce/complex/plane.html

APPLICATION OF FRACTALS

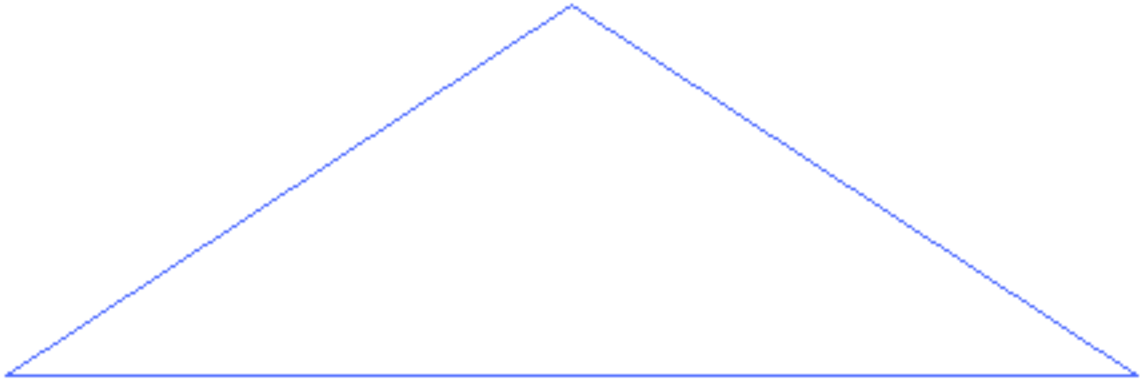
- Lossy compression (sounds and images)
- Seismology
- Fractal antennas
- Computer network design
- Fractal landscape generation
- Financial analysis
- Computer graphics
- Art

MANDELBROT SET

- A visualization of an iterative function in a complex plane
- $y(z) = z^2 + c$
- c is used as a bounding constant to ensure that y(z) does not exceed that value as we perform an increasing number of iterations
- Fine detail even on infinite magnification

TERRAIN MAPPING

- Used to generate mountainous or futuristic terrain
- Practical use in the entertainment/graphics sector
- Far Cry 2, Left4Dead, .kkrieger, Borderlands, Diablo

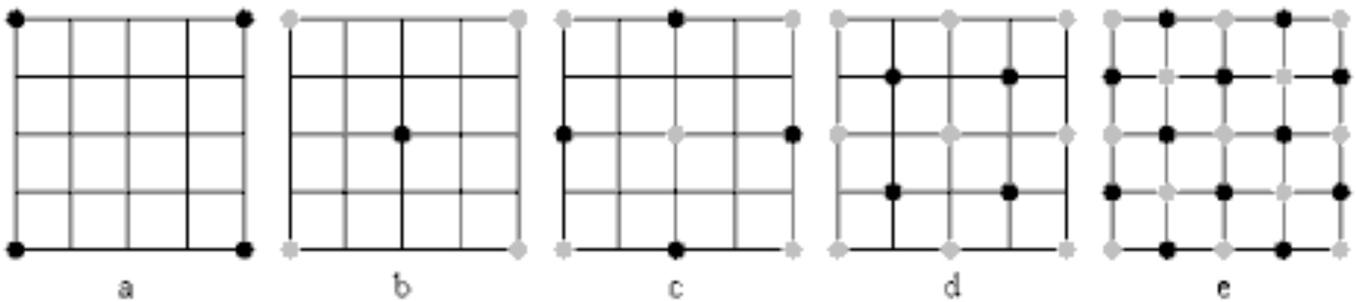


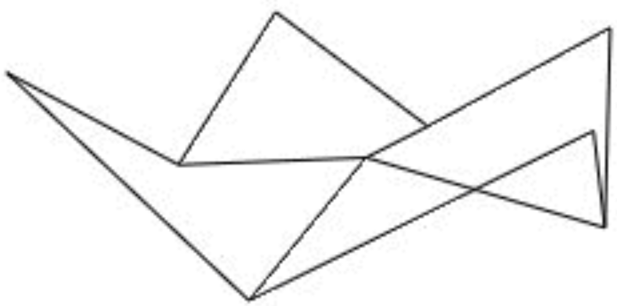
Source: Wikipedia

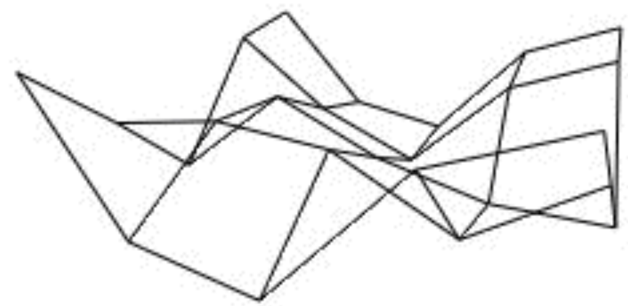
MID POINT DISPLACEMENT ALGORITHM

- Take a line segment on the X-axis
- Calculate the mid-point of the segment
- Add some random noise to generate the y co-ordinate
- Reduce the range of the random noise
- Recursively call the above steps for all the line segments

DIAMOND SQUARE ALGORITHM







Source: http://gameprogrammer.com/fractal.html#diamond

PROBLEMS

- The number of squares increases exponentially after every round
- 2^{X+2} squares after X iterations
- Large terrains require a huge number of calculations

GOALS

- Write a Mandelbrot set visualization program using CUDA/C
  - Generate the elements of a Mandelbrot set
  - Visualize them using OpenGL
- Create a random terrain map generation program using CUDA/C
  - Generate random height maps using fractals
  - Visualize them using OpenGL/pass height maps as inputs to existing 3D renderers

APPROACH

- Divide and Conquer!
- Assign a thread/block/processor to each individual pixel value that needs to be calculated
- Run the fractal generation/diamond square algorithm
- Because the number of pixels to be calculated differs in each step for DS, dynamically allocate
- CUDA did not support recursive device calls for < CUDA 3.1 (roughly compute capability 2.0)

CUDA ABSTRACTION FOR DEVELOPERS

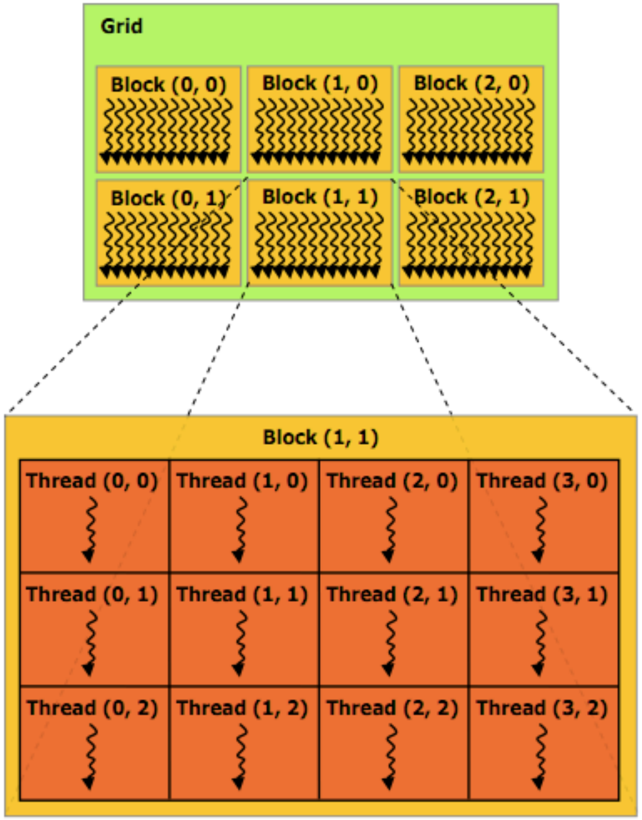


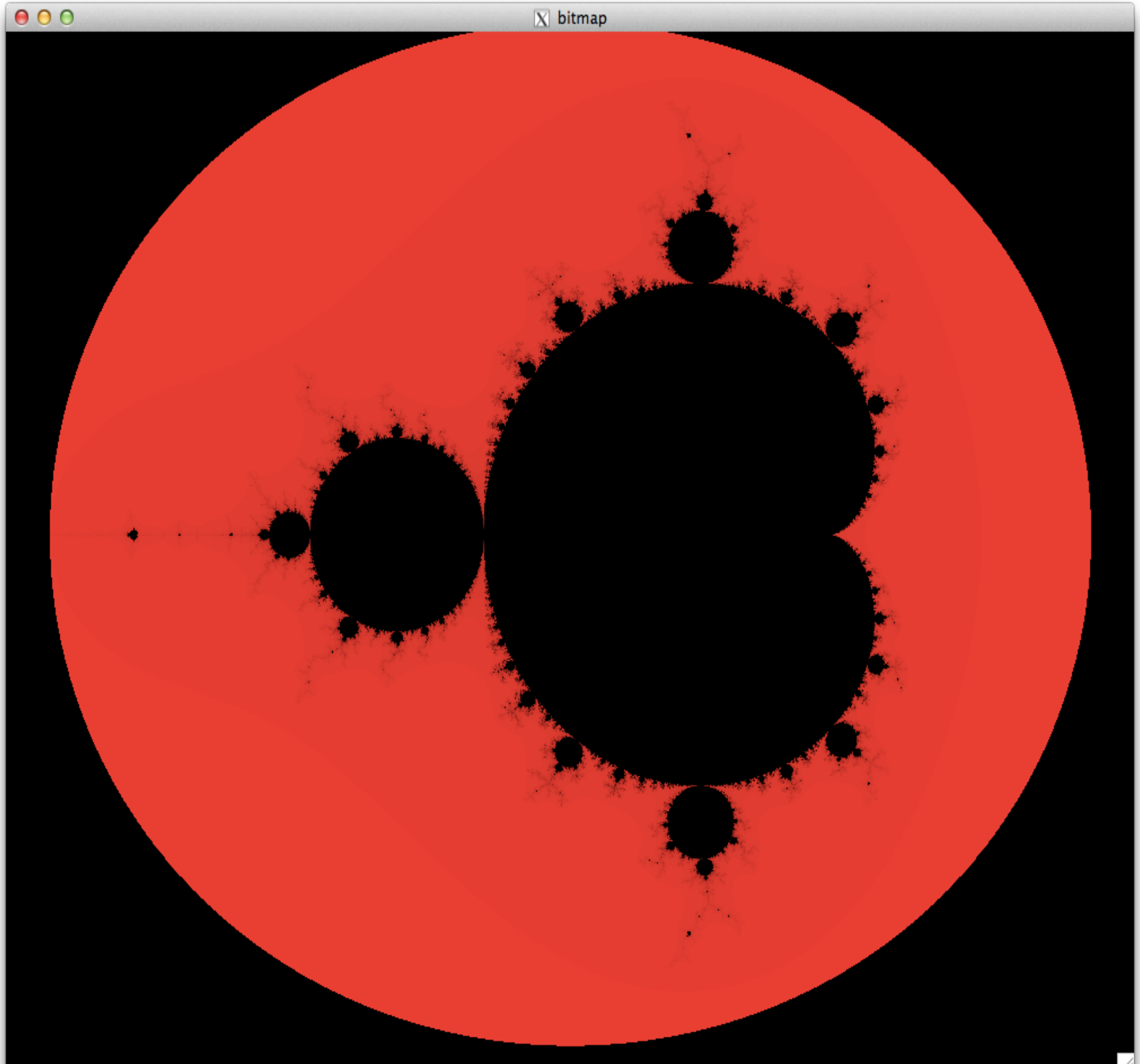
Figure 2-1. Grid of Thread Blocks

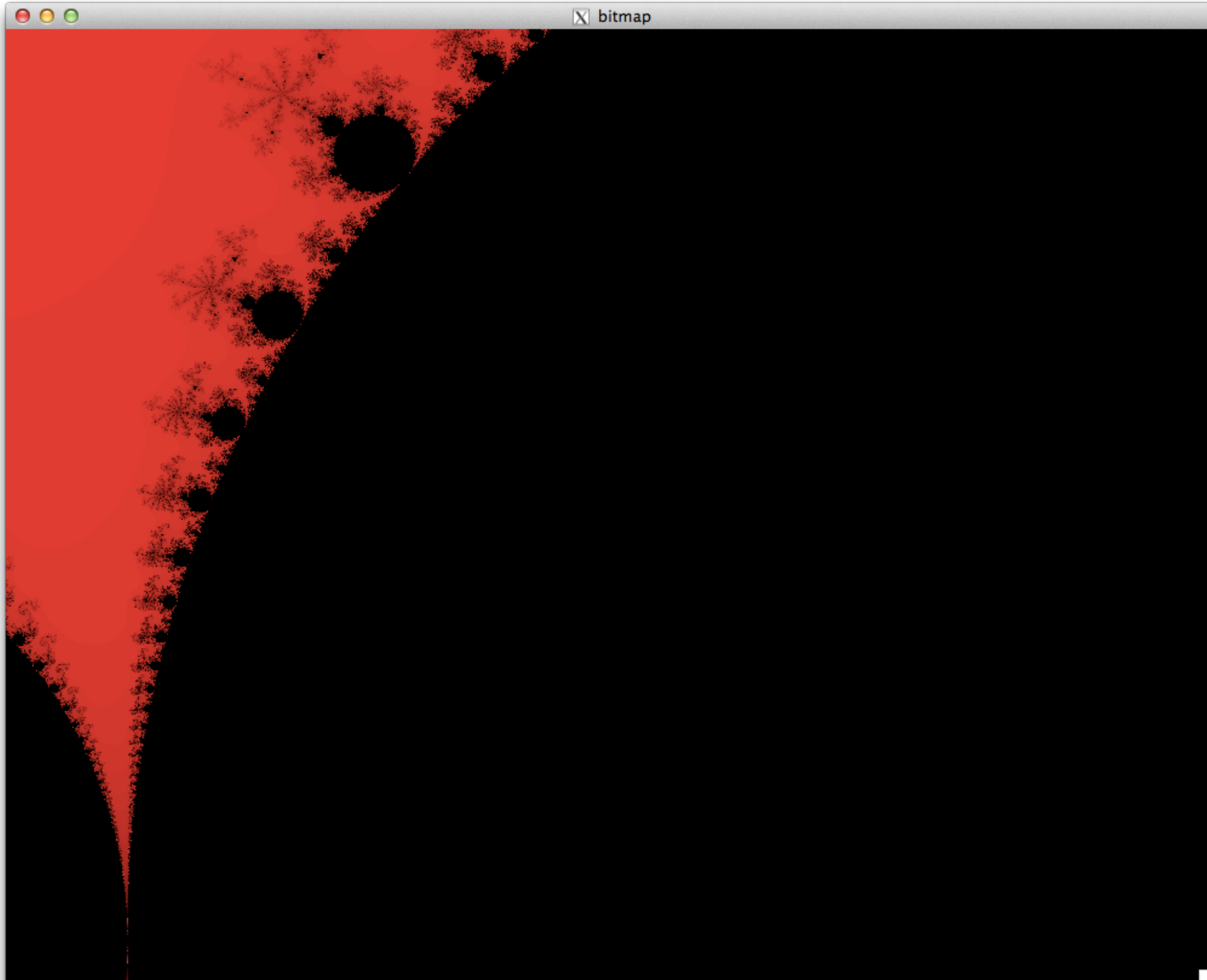
Source: CUDA C Programming Guide

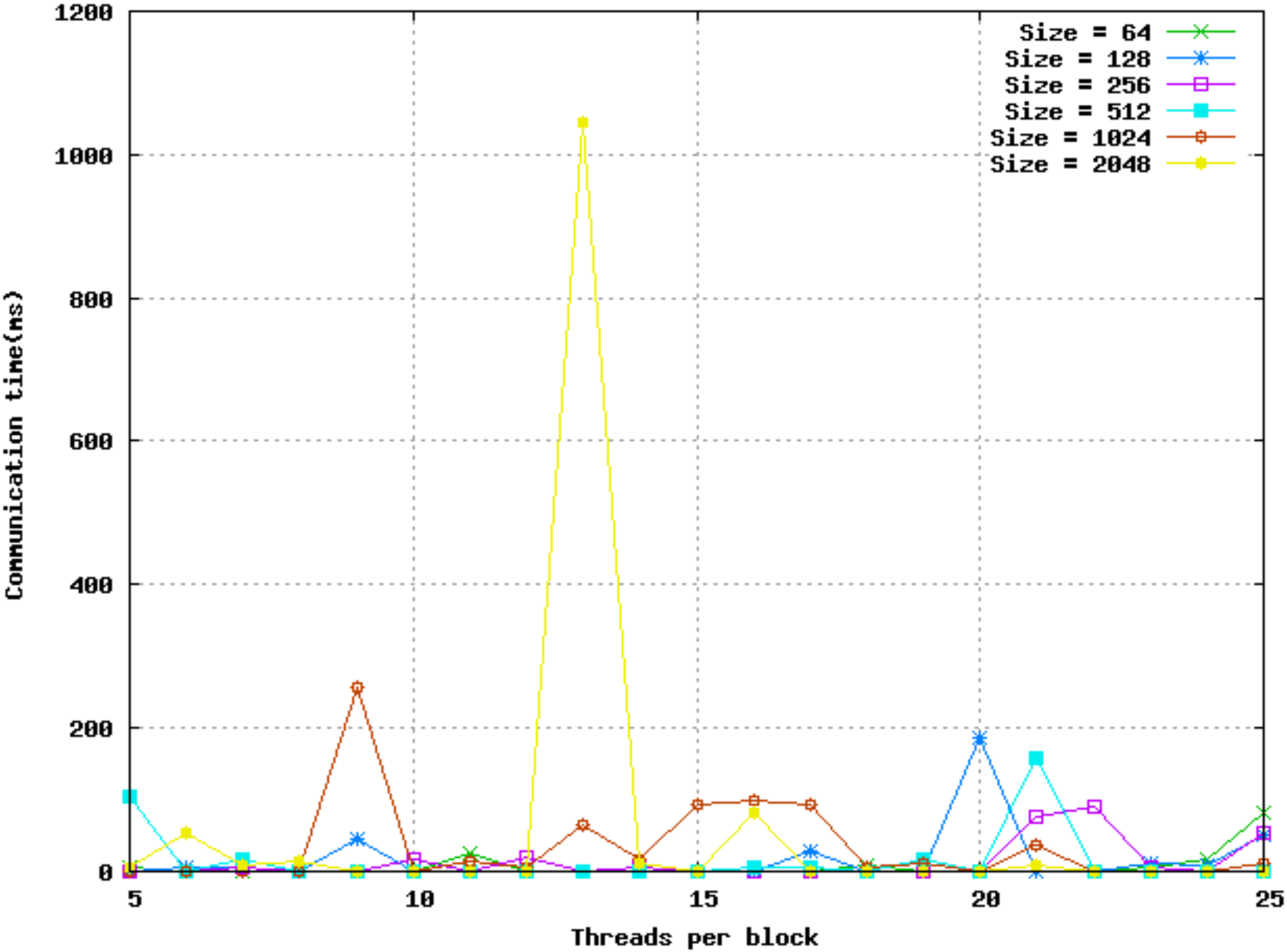
- Though GPU's inherently support multi-threading, there are several hardware constraints
- Eg: CCR
  - Name: Tesla M2050
  - CUDA Version: 2.0
  - Shared memory per block: 49152
  - Total constant memory: 65536
  - Regs per block: 32768
  - Max threads per block: 1024
  - Max threads per dim: 1024,1024,64
  - Max grid size: 65535,65535,65535
  - Multi processor count: 14

MANDELBROT SET

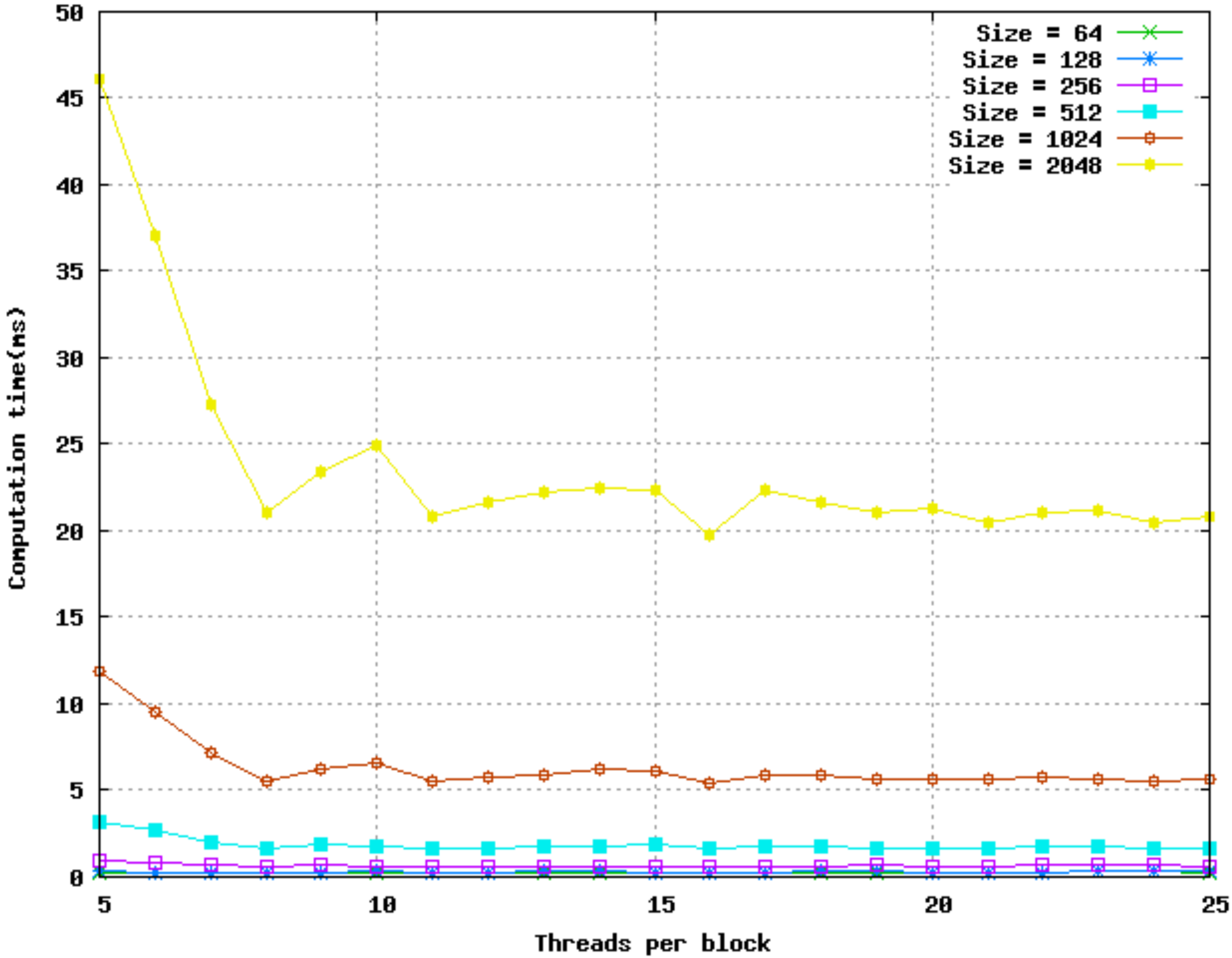
- Straight-forward implementation
- Challenge was in writing the program in CUDA
- Use varying number of threads and blocks
- Number of blocks dependent on the number of threads
- I learnt the hard way about the hardware limitation on the number of threads
- numBlocks*numThreadsPerBlock <= maxThreadsPerDim
- numThreadsPerBlock <= maxThreadsPerBlock
- Based on input number of threads, dynamically allocating number of blocks to be created by the GPU
- Communication time => Time taken to transfer initial array to GPU or to allocate memory on GPU
- Computation time => Time taken by the GPU to finish computation


bitmap


bitmap



Size = 64
Size = 128
Size = 256
Size = 512
Size = 1024
Size = 2048
Communication time(ms)
Threads per block
0
200
400
600
800
1000
1200
5
10
15
20
25

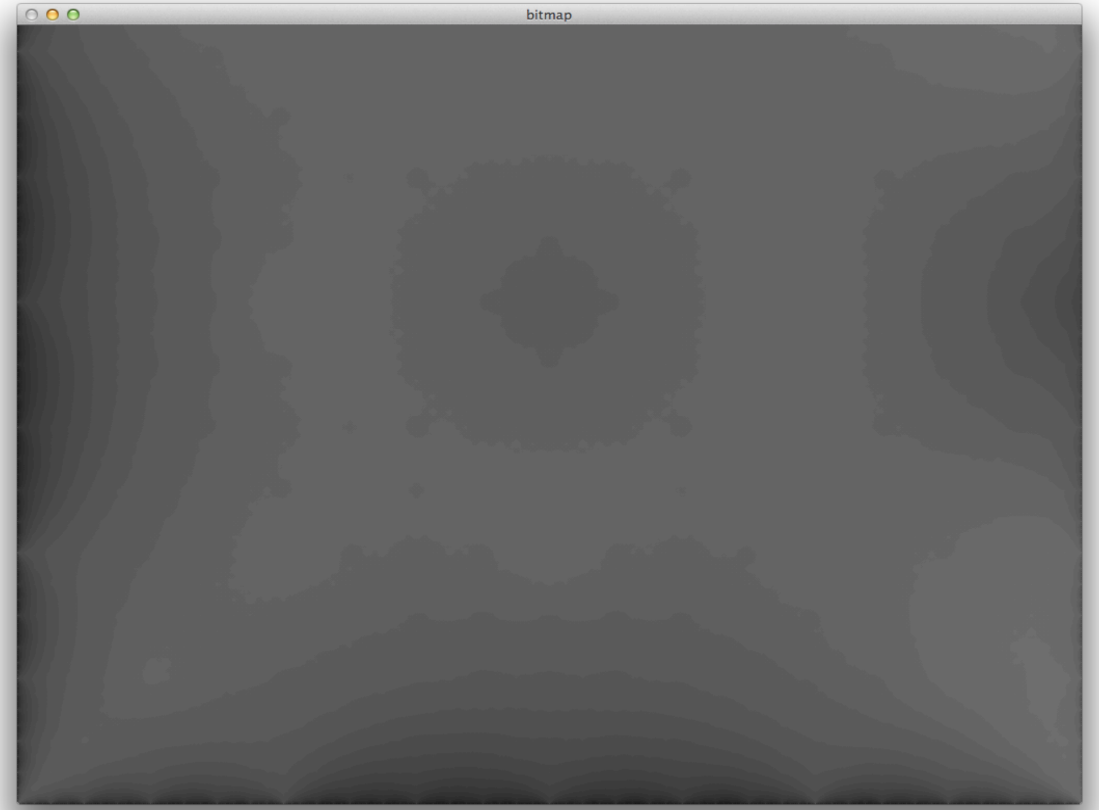


Size = 64
Size = 128
Size = 256
Size = 512
Size = 1024
Size = 2048
Computation time(ms)
Threads per block
0
5
10
15
20
25
30
35
40
45
50
5
10
15
20
25

HEIGHT MAPS WITH GPU's

- Sample height map available on the internet

- Sample height map from my code

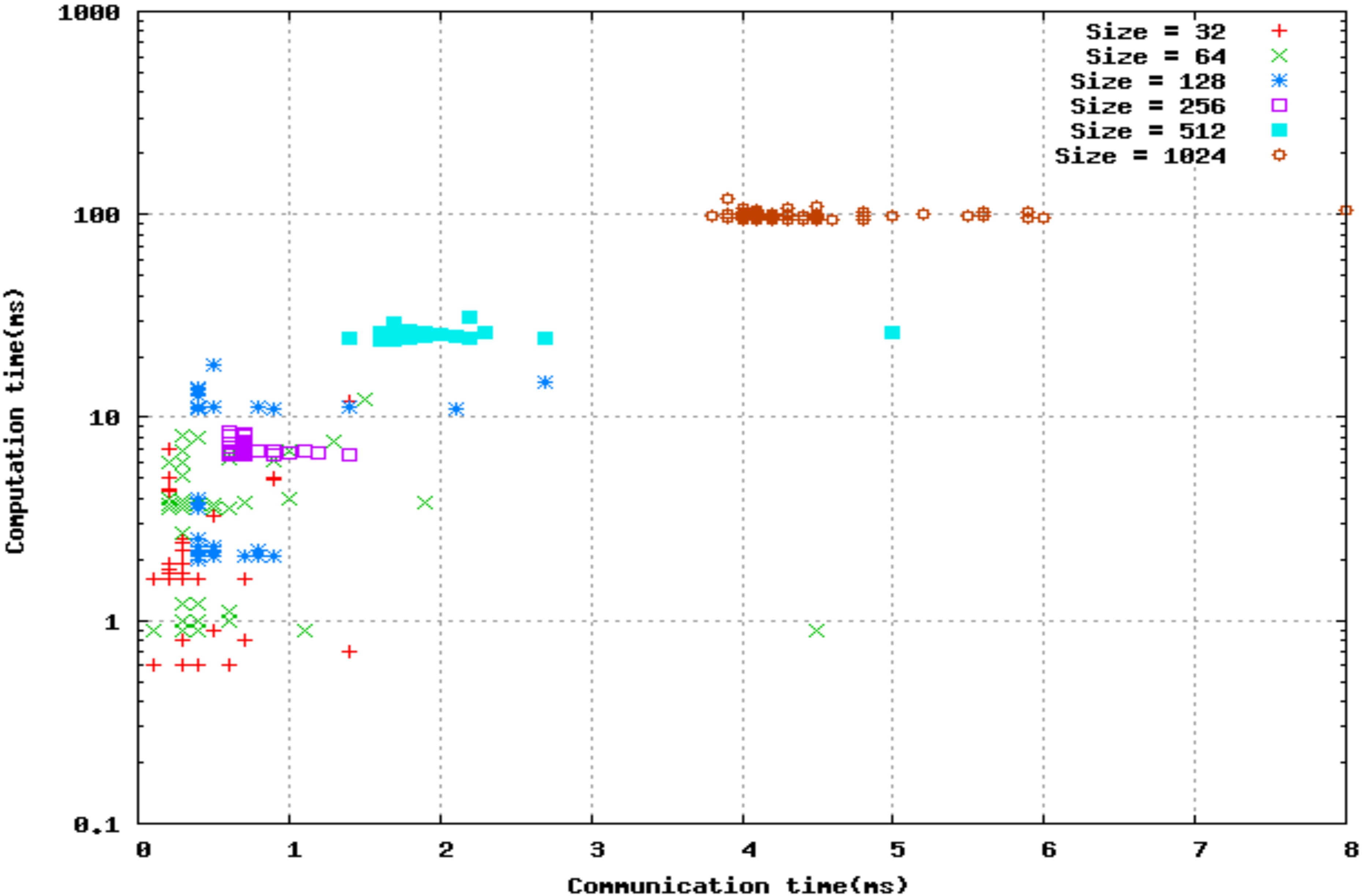


- Flatten the array and pass it to the GPU
- Communication time – Time take to initialize GPU with initial height map with seeded values
- Computation time – Time taken by the GPU to calculate height values for ALL the points in the given 2D array
- Runs
  - Sequential run on CPU
  - Single thread with multiple blocks
  - Single block with multiple threads
  - Multiple blocks with multiple threads

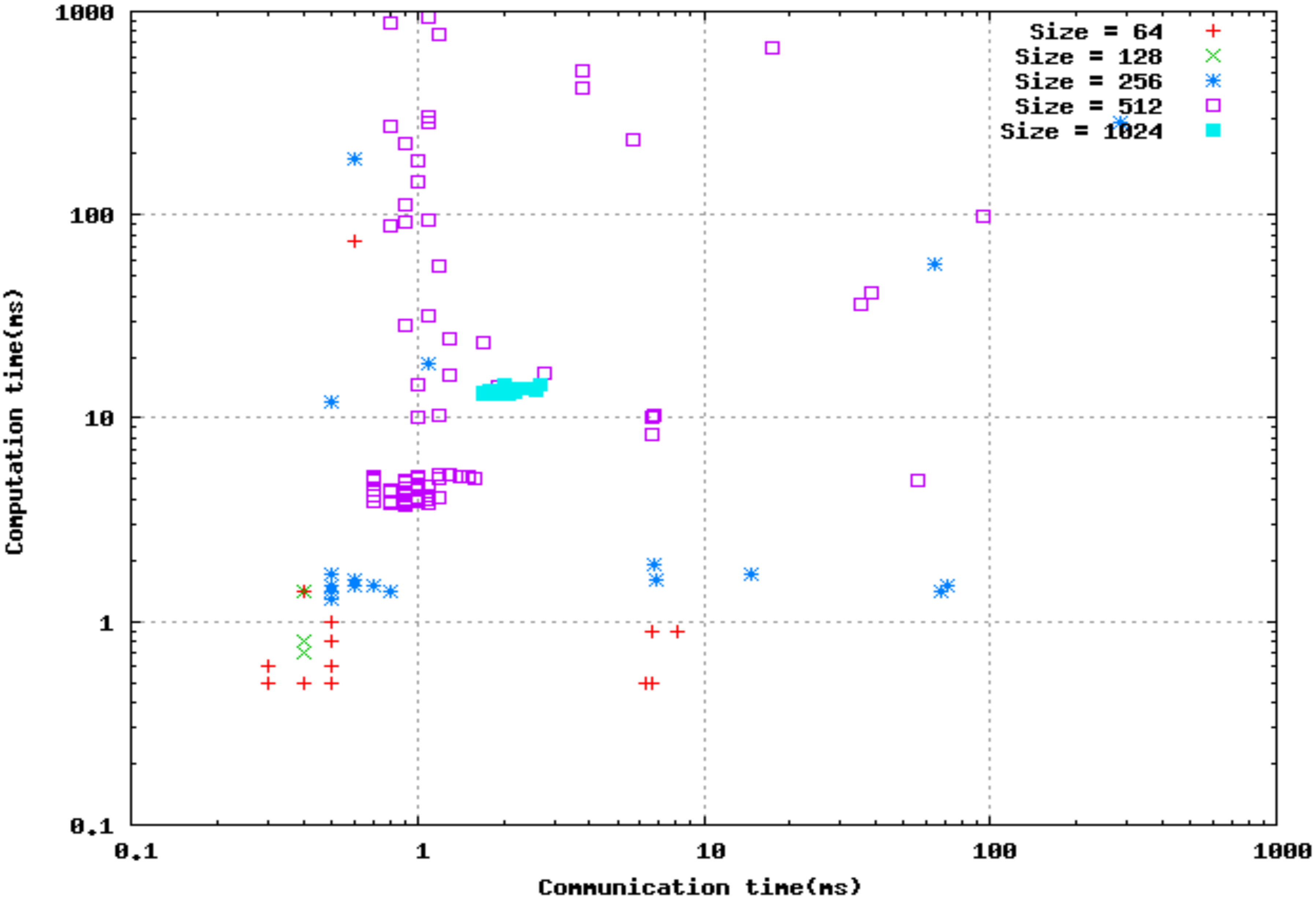
SEQUENTIAL CPU RUN

- The previous image did not use random values in the diamond square algorithm
- Grid size: 1025*1025
- Average running time (with random value generation) was 64.9 ms (100 runs)
- Difference in running on integrated graphics memory vs. dedicated GPU (CCR machine)
- For the parallel runs, number of blocks/threads = dimension of image
- If hardware limit is smaller, assign multiple pixel values to each block/thread

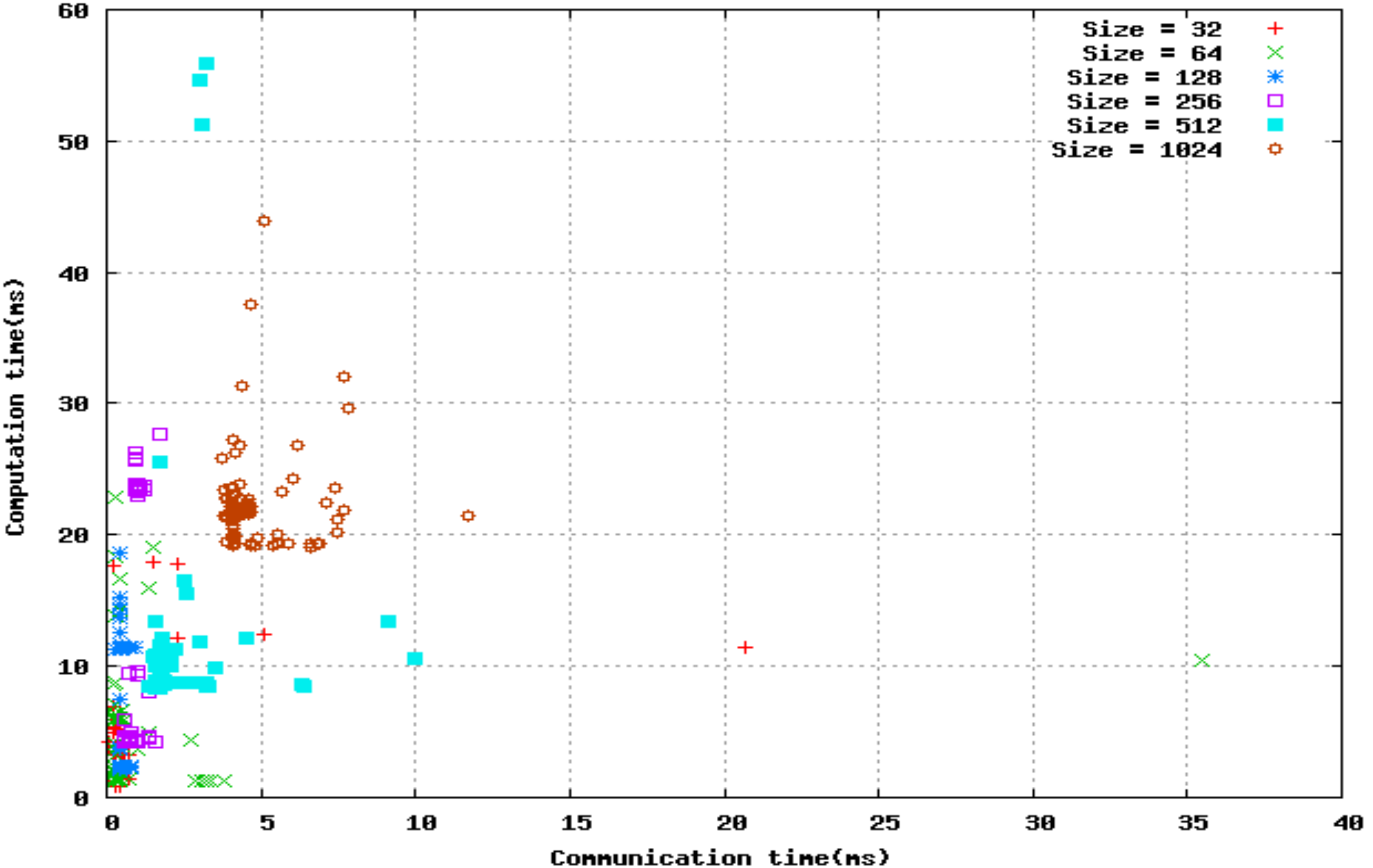
SINGLE THREAD WITH MULTIPLE BLOCKS (INTEGRATED GRAPHICS MEMORY)



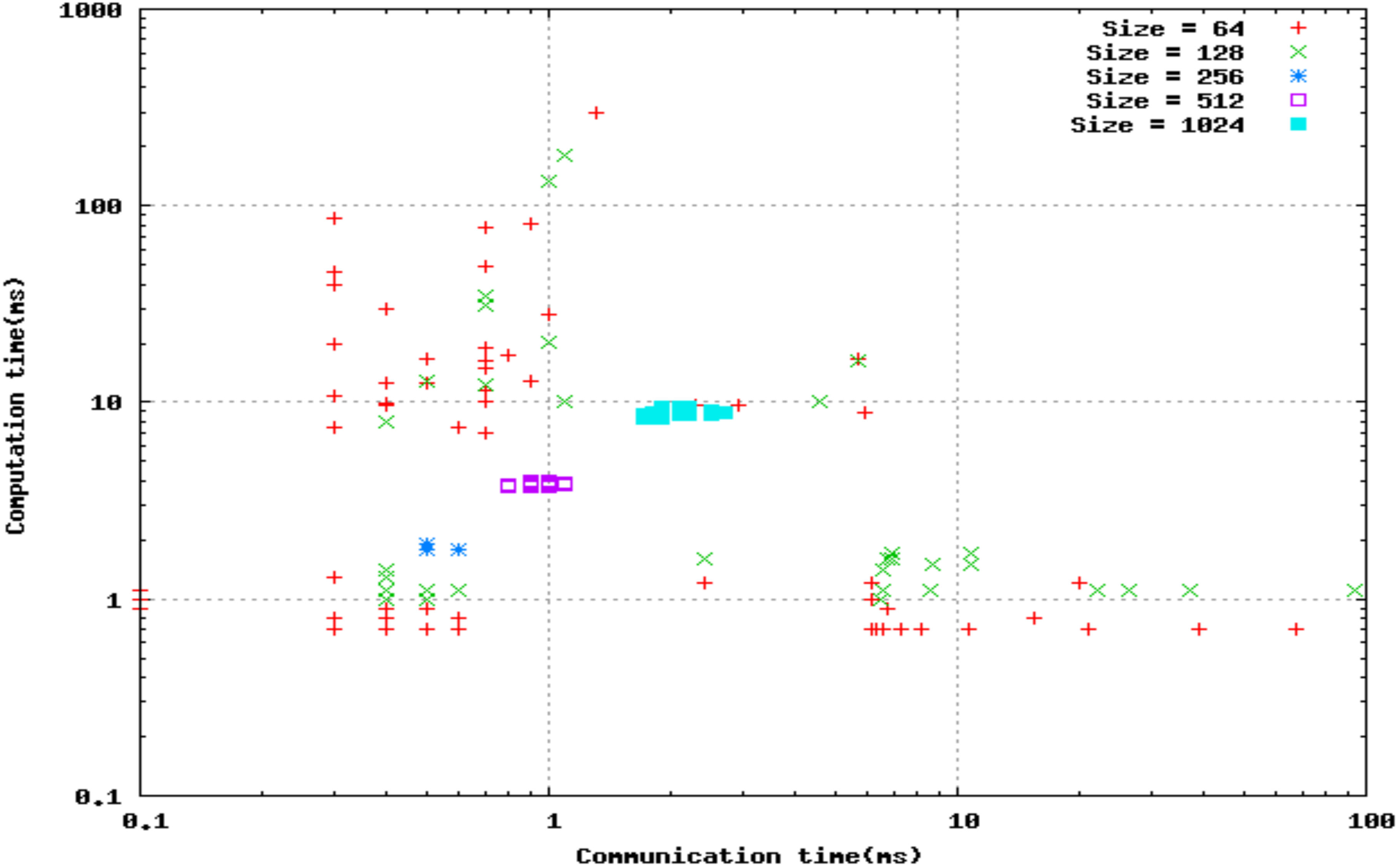
SINGLE THREAD WITH MULTIPLE BLOCKS (CCR)



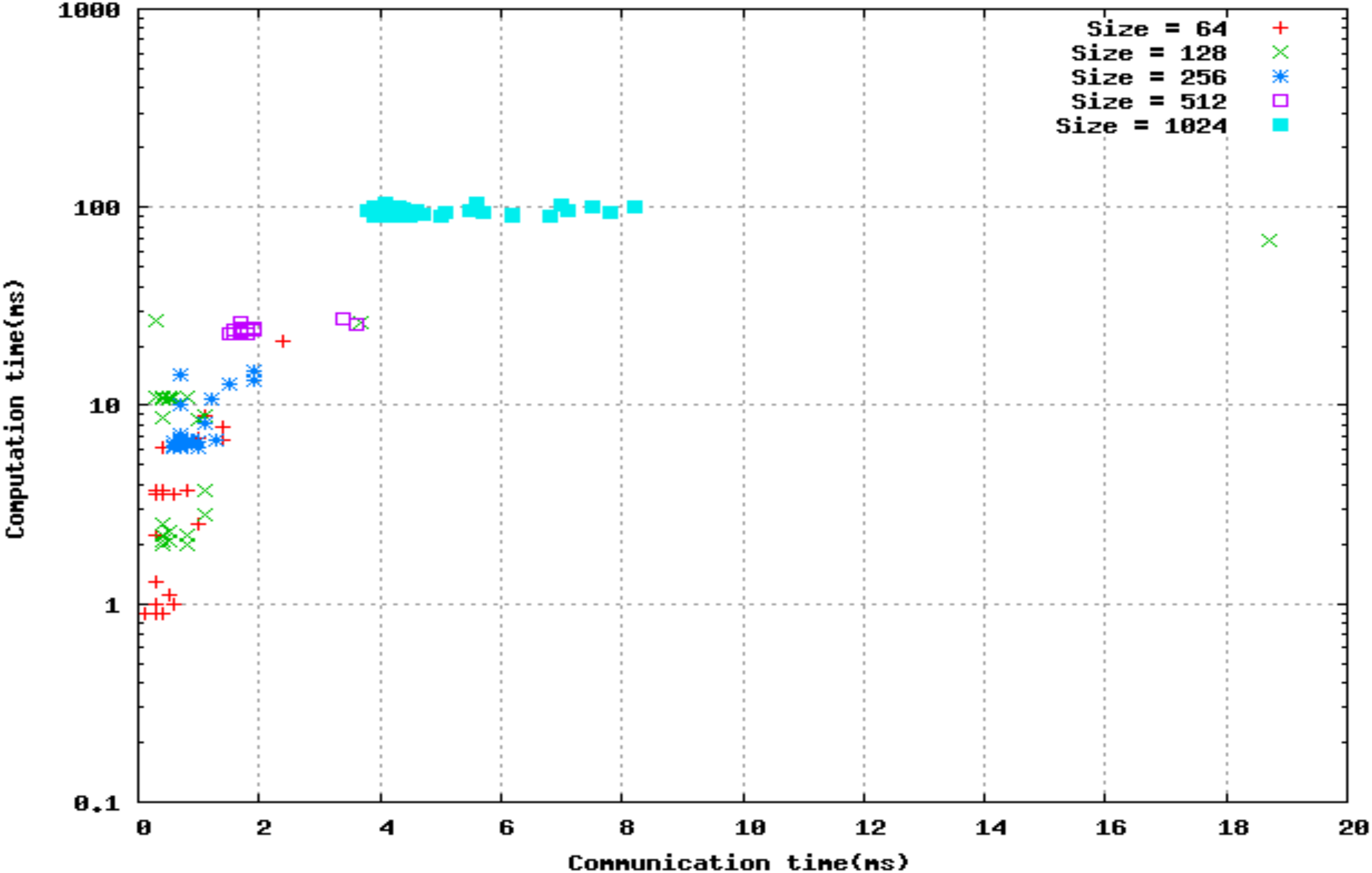
SINGLE BLOCK WITH MULTIPLE THREADS (INTEGRATED GRAPHICS MEMORY)



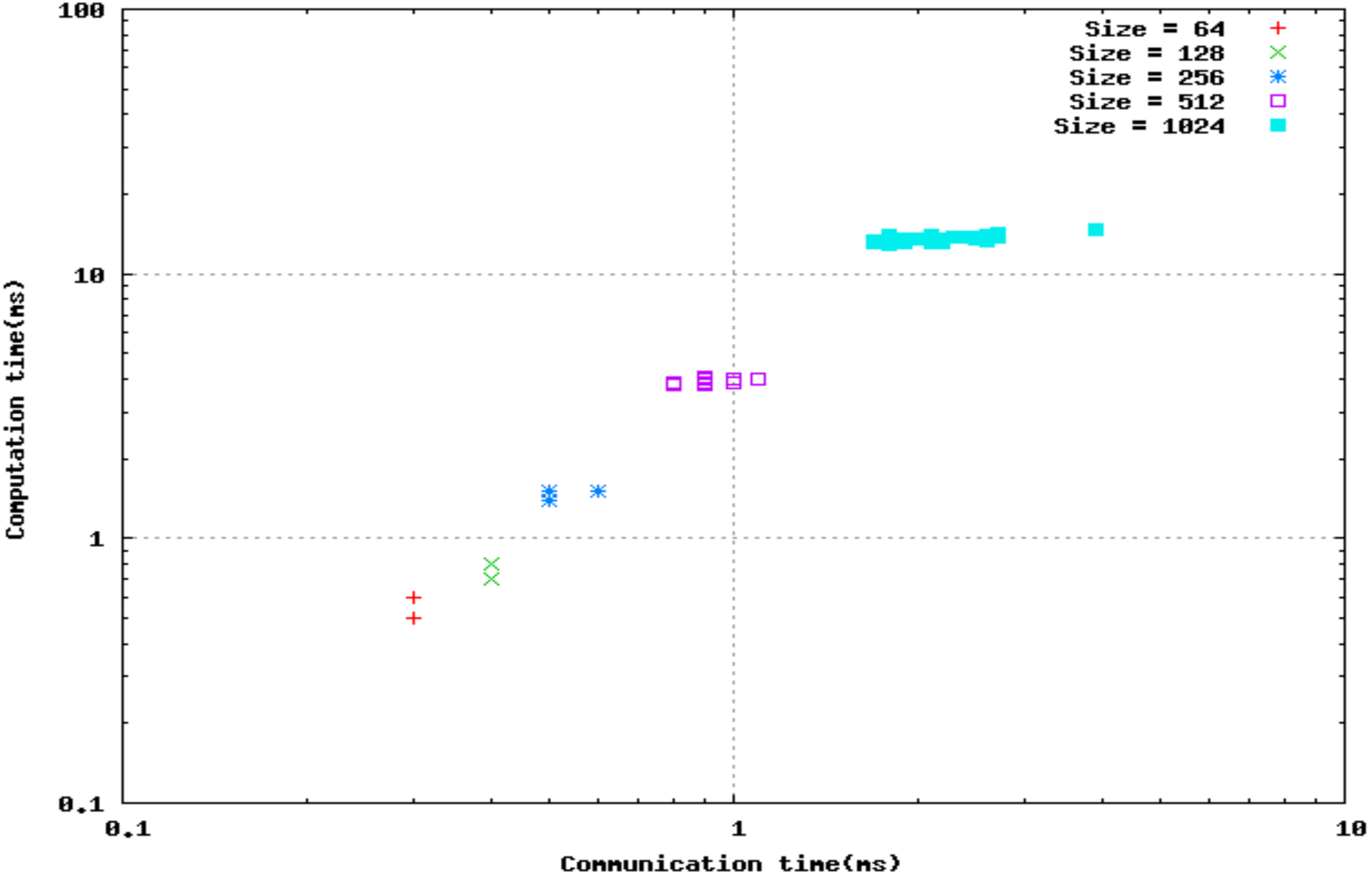
SINGLE BLOCK WITH MULTIPLE THREADS (CCR)



MULTIPLE BLOCKS WITH 4 OR LESS THREADS - (INTEGRATED GRAPHICS MEMORY)



MULTIPLE BLOCKS WITH 4 OR LESS THREADS - (CCR)



COMMENTS

- When we have multiple blocks, assign each square step to a single thread
- Can’t always launch 4 threads (hardware limitation!)
- Based on the maxThreadsPerDim property of the CUDA enabled device
- Dynamic creation of threads and blocks maybe creating considerable overhead
- Computation time involves some communication (random numbers) to the GPU
- Using more advanced features of the GPU – streams, DMA, Shaders, maybe even multiple CUDA enabled devices – might considerably lower the running time

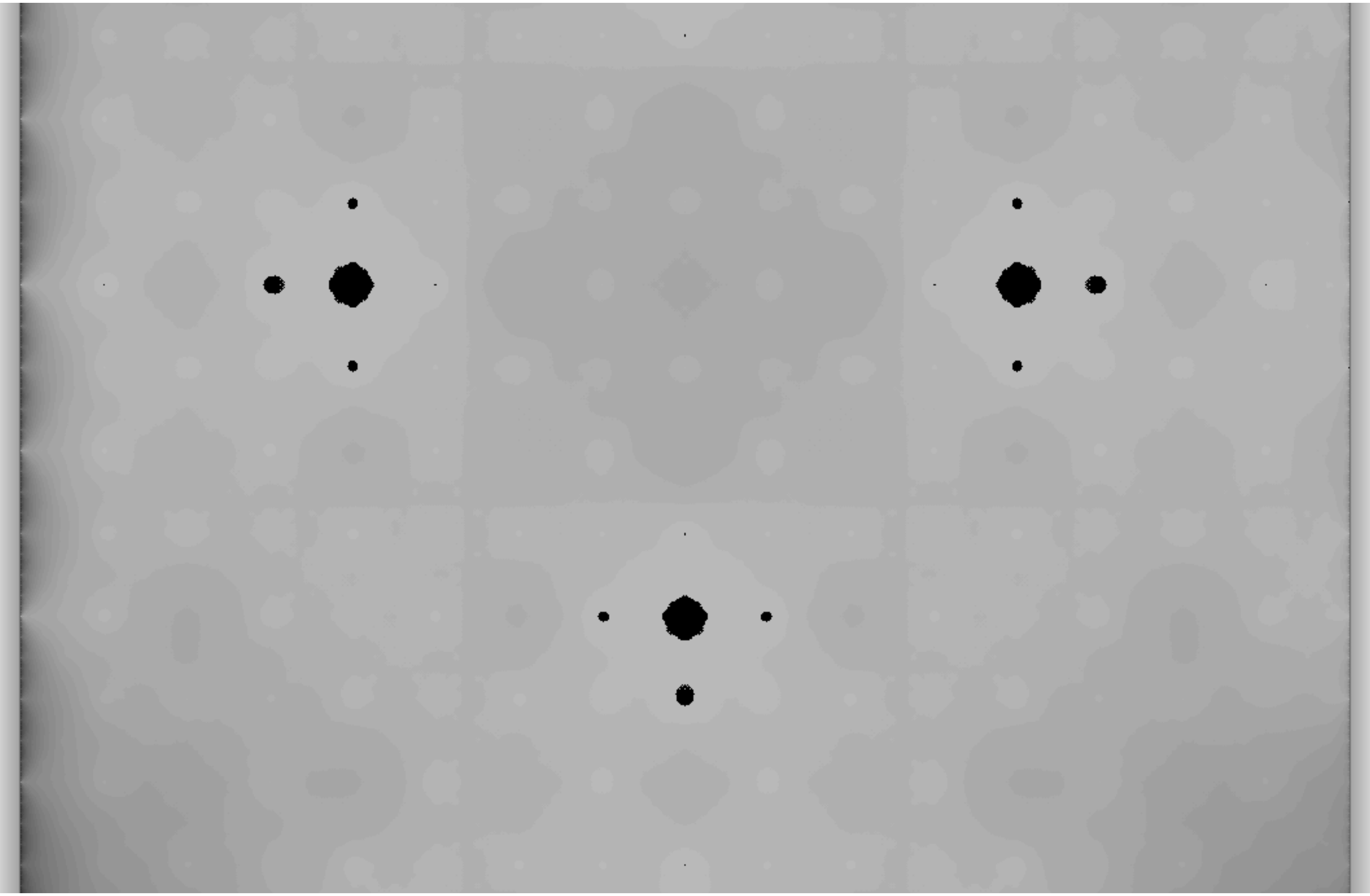
FUTURE WORK

- Make use of 2D CUDA functions like cudaMemcpy2D, cudaMallocPitch etc.
- Utilize advanced features – streams, DMA, Shaders
- Divide into smaller problems of constant size and solve each problem on a separate device in parallel – will help in static assignment of number of threads/blocks
- Use better terrain generation algorithms
- Add rendering code – mesh and full color

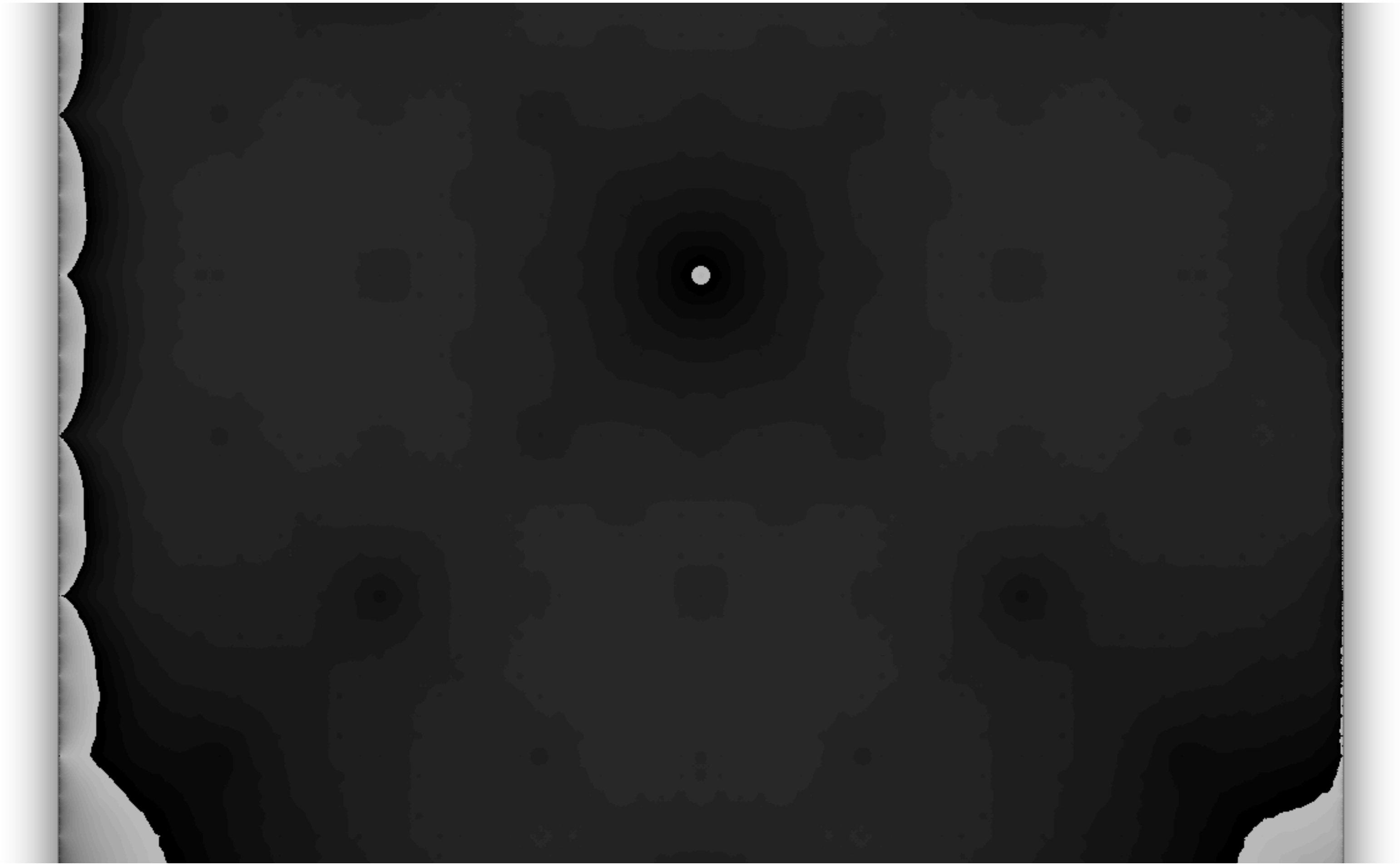
QUESTIONS?

SAMPLE HEIGHT MAPS GENERATED

SAMPLE HEIGHT MAPS GENERATED



SAMPLE HEIGHT MAPS GENERATED



SAMPLE HEIGHT MAPS GENERATED