Parallel Computing

Linh Nguyen '16

A (very) Short Introduction to CS

- Use computers to solve problems
- Is applicable in many different disciplines
- Has many areas
- Is fun

- NOT about using softwares (such as Word or Excel)

Why Parallel?

GTX 680

Single-threaded performance has plateaued

Why Parallel?

GTX 680

Terminology

- Central Processing Unit = CPUs
- Graphics Processing Unit = GPUs
- Threads = processors that can do arithmetic computations.

A survey of parallel hardware

GTX 680

Personal Mobile Device

2 CPU cores/
3 GPU cores

4 CPU cores/
4 GPU cores

iPhone 5

Galaxy S3

A survey of parallel hardware

GTX 680

Desktop Space

A survey of parallel hardware

GTX 680

Warehouse Space

2012

8

A survey of parallel hardware

GTX 680

Warehouse Space

 2508 CPU cores
 187264 GPU cores

CPUs vs GPUs

What is a CPU?

- CPU
  - SR71 Jet
- Capacity
  - 2 passengers
- Top Speed
  - 2200 mph

CPUs vs GPUs

GTX 680

What is a GPU?

- GPU
  - Boeing 747
- Capacity
  - 605 passengers
- Top Speed
  - 570 mph

CPUs vs GPUs

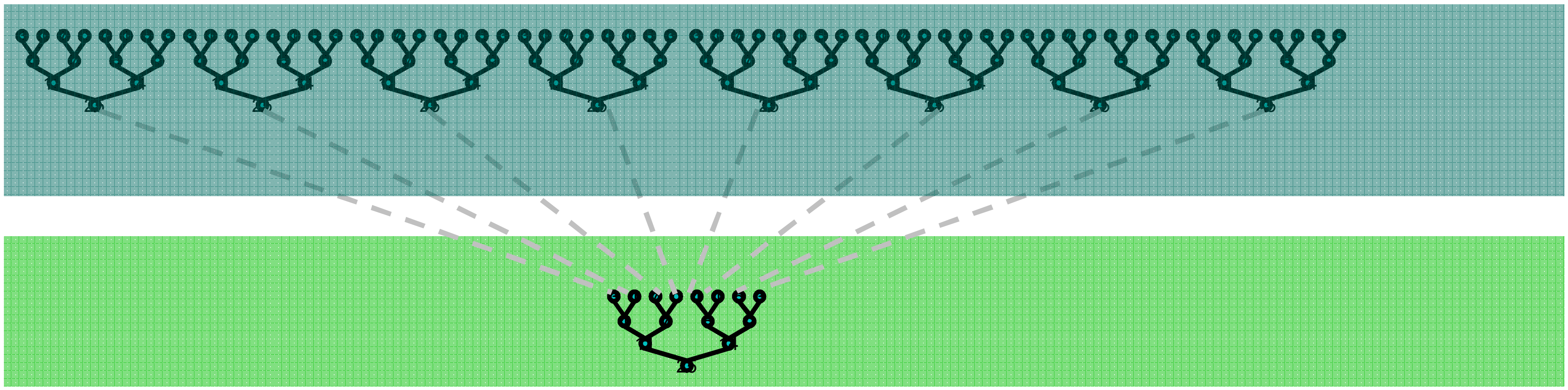
GTX 680

	Capacity (passengers)	Speed (mph)	Throughput (passengers * mph)
“CPU” Fighter Jet	2	2200	4400
“GPU” 747	452	555	250,860

- Assign yourself the number 1
- Pair off with someone standing
- Add your numbers together and adopt the sum as your new number
- One of the pair sits down
- Repeat

Parallel Reduction

- For a large array
  - Each thread adds a pair of numbers
  - Write partial sums to the temporary array
  - Repeat until done



The BIG idea behind CUDA

- Replace loops with a functions (a kernel) excecuting at each point in a problem domain
  - E.g., process a 1024x1024 image with one kernel invocation per pixel or 1024x1024=1,048,576 kernel executions.

Traditional loops

```
void
vecAdd(const int n,
       const float *a,
             const float *b,
             float *c)
{
  int i;
  for (i = 0; i < n; i++)
    c[i] = a[i] * b[i];
}
```

Data Parallel CUDA

```
__global__ void
vecAdd(const float *a, const float *b,
            float *c)
{
  int id = threadIdx.x + \
      blockDim.x * blockIdx.x;
  c[id] = a[id] * b[id];
}
// many instances of the kernel,
// called threads, execute
// in parallel
```

What is GPGPU?

- General Purpose computation using GPU in applications (other than 3D graphics)
  - GPU accelerates critical path of application

- Data parallel algorithms leverage GPU attributes
  - Large data arrays, streaming throughput
  - Low-latency floating point (FP) computation

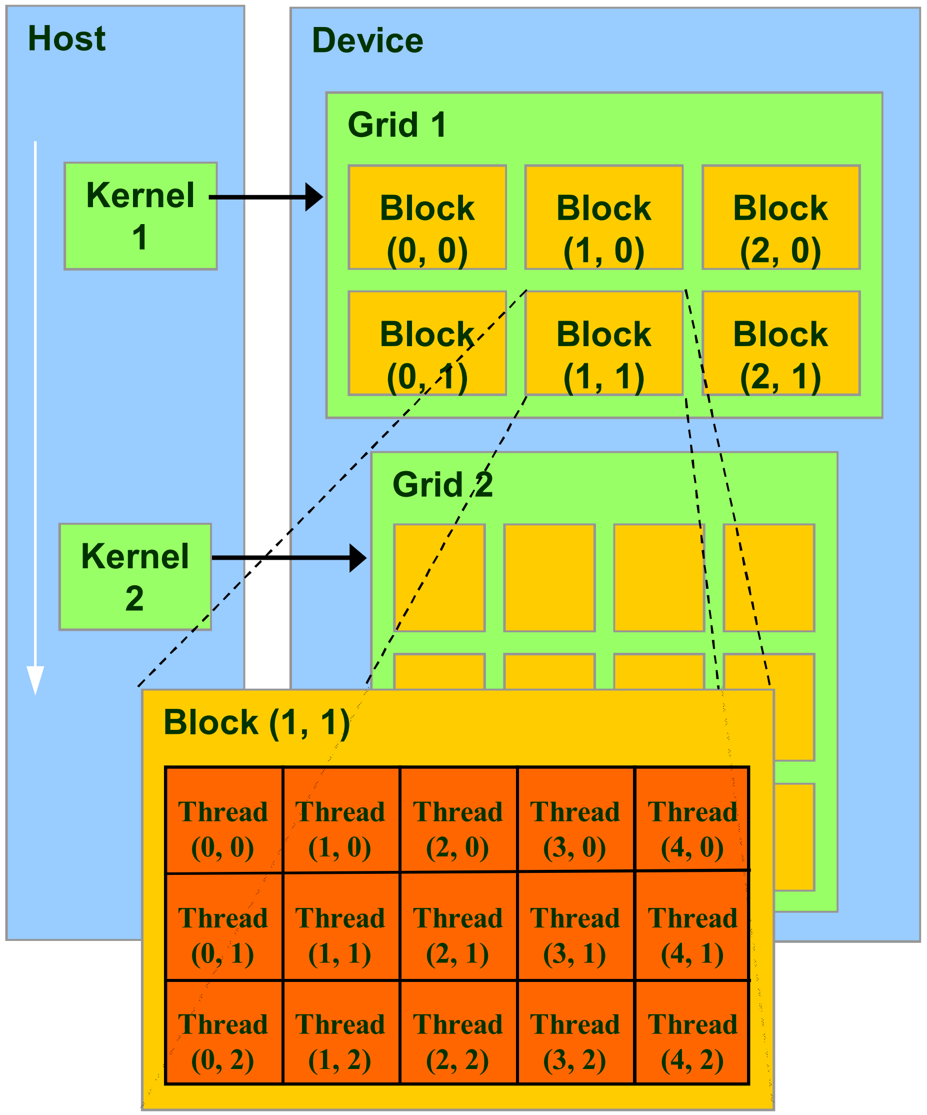
- Applications – see //GPGPU.org
  - Game effects (FX) physics, image processing
  - Physical modeling, computational engineering, matrix algebra, convolution, correlation, sorting

CUDA Programming Model

- The GPU is viewed as a compute device that:
  - Is a coprocessor to the CPU or host
  - Has its own DRAM (device memory)
  - Runs many threads in parallel
    - Hardware switching between threads (in 1 cycle)
- Data-parallel portions of an application are executed on the device as kernels which run in parallel on many threads

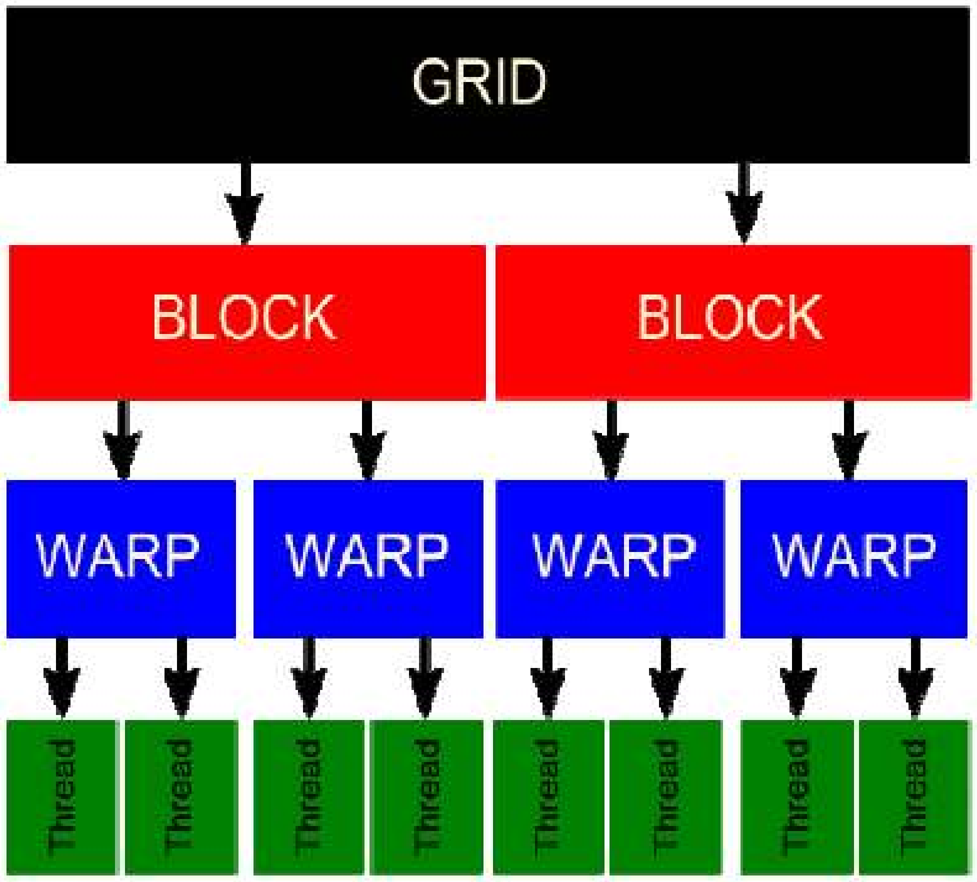
Thread Scheduling: Grids and Blocks

- Kernel executed as a grid of thread blocks
  - All threads share data memory space
- Thread block is a batch of threads, can cooperate with each other.
- In each thread block there are warps of 32 threads.
- Threads and blocks have IDs



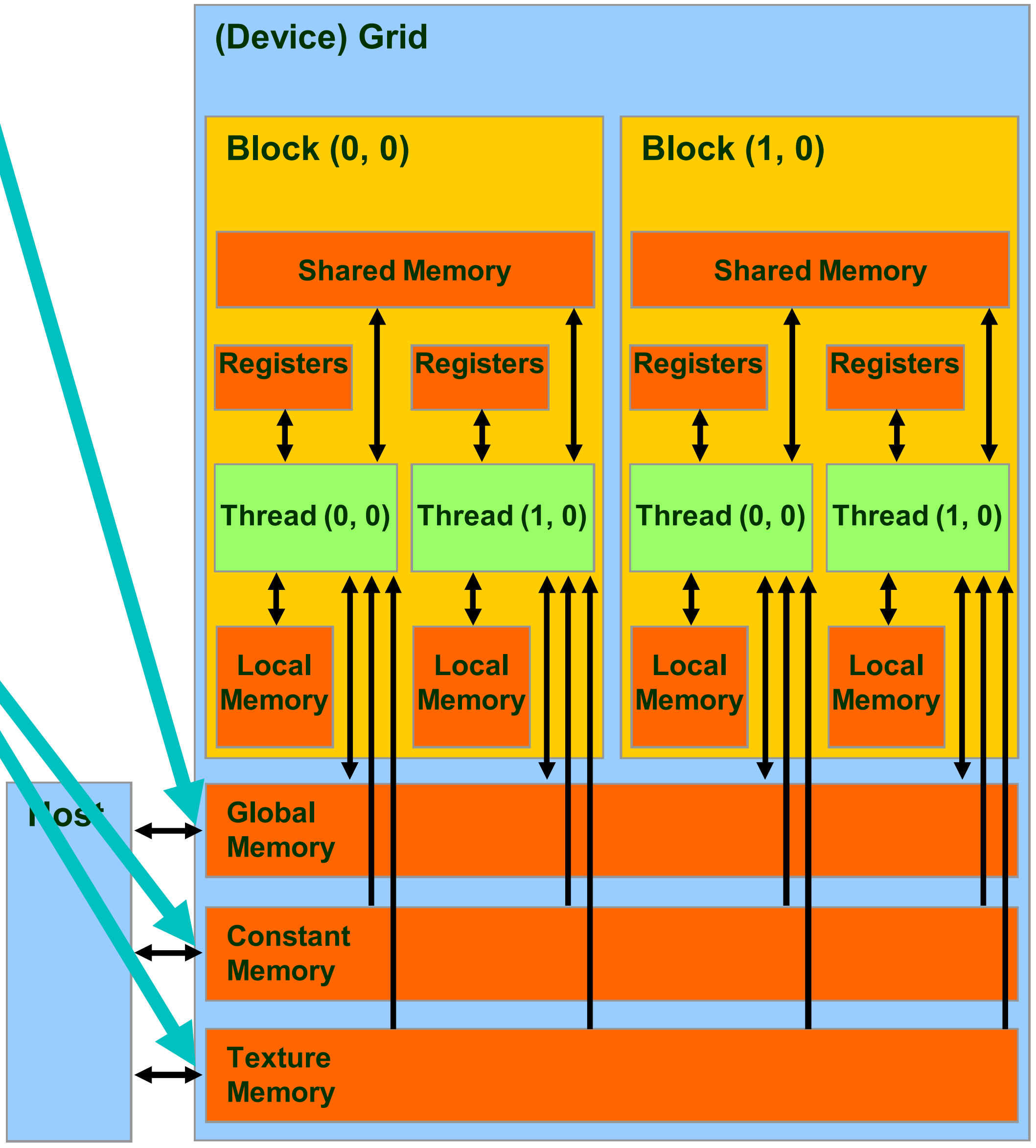
Courtesy: NDVIA

Execution model



Memory Hierarchy

- Global memory
  - Main means of communicating R/W Data between host and device
  - Contents visible to all threads
- Texture and Constant Memories
  - Constants initialized by host
  - Contents visible to all threads



Courtesy: NDVIA

HotSpot

- Based on a well-known duality between electric current and heat flow
- Construct a network of thermal “resistances” and “capacitances”
- Solve standard RC circuits with finite element analysis

The circuit

- Divide the chip into small “blocks”
- Each node has a capacitor to model transient state
- The stencil equation

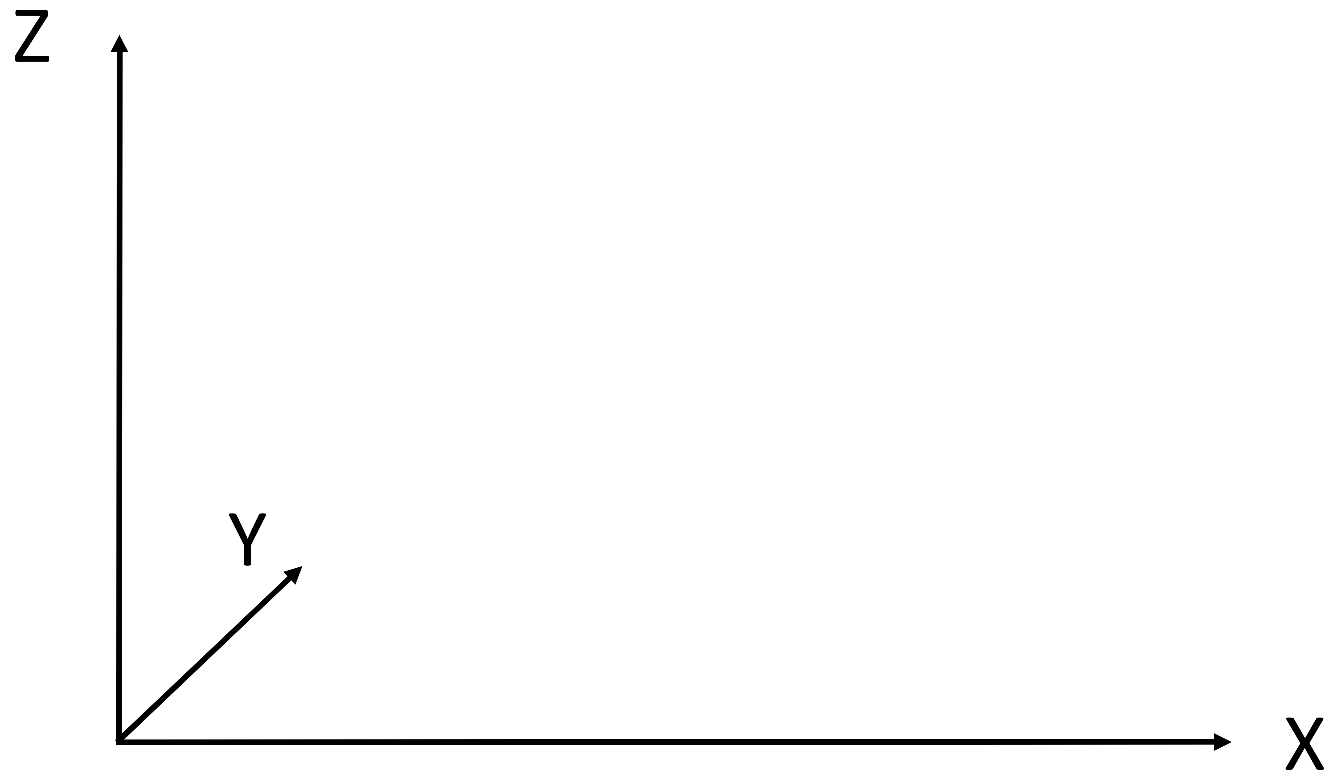
$$\left(\frac{(T_N + T_S - 2\mathrm{T}_{i,j,k})}{R_x} + \frac{(T_E + T_W - 2\mathrm{T}_{i,j,k})}{R_y} + \frac{(T_A + T_B - 2\mathrm{T}_{i,j,k})}{R_z} + P_{i,j,k}\right) \times \frac{\Delta t}{C} = \Delta T$$

Profiling HotSpot

- 512x512 thermnal grid
- Intel Xeon(R) X5550 2.67GHz
- Gprof 2.22

Accelerating Slope Function

- Each thread sweeps points in the z direction, calculating the temperatures using the 7-point stencil
- X and Y dimensions are blocked with AxB thread blocks
- A and B are user-configurable parameters



Testing the slope kernel

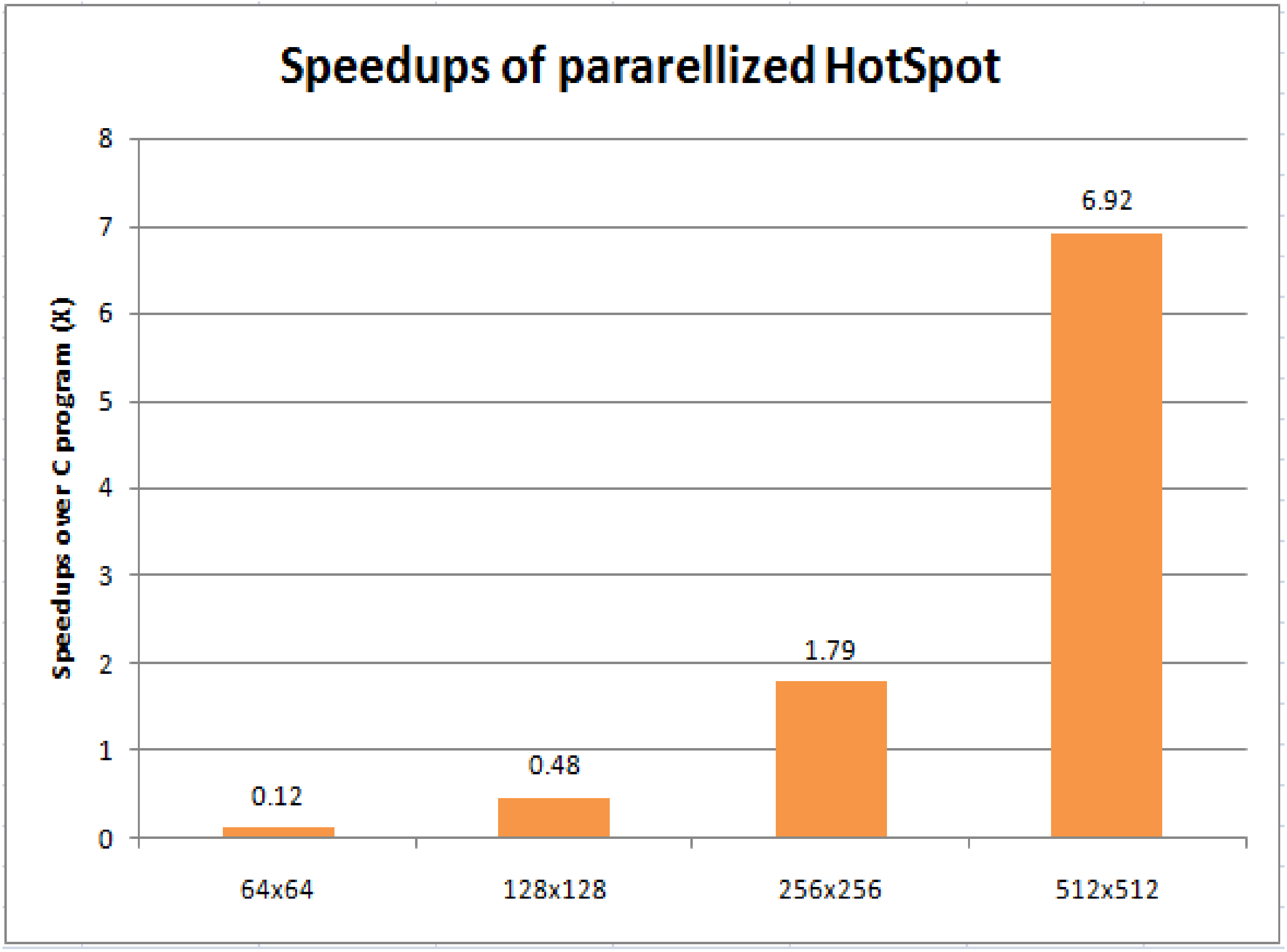
- CUDA toolkit 5.5
- Geforce GT 630M, Tesla C2050 and Tesla K20c

Testing the HotSpot Program

GPU Solver

CPU Solver

Testing the HotSpot Program


Speedups of pararellized HotSpot
Speedups over C program (X)
64x64
128x128
256x256
512x512