Instituto de Ciencias Aplicadas y Tecnología

BioComLab

INTRODUCCIÓN A CUDA

Alfonso Gastélum Strozzi, Ph.D.
alfonso.gastelum@icat.unam.mx
http://www.biocomlab.com/
Universidad Nacional Autónoma de México
Instituto de Ciencias Aplicadas y Tecnologías
Instrumentación científica e Industrial
Dispositivos Biomédicos

Web of Science: **M-8874-2016**
SCOPUS: **57214152908**

WHAT IS THE PARALLEL PARADIGM?

- Computing is evolving from "central processing" on the CPU to "co-processing" on the CPU and GPU.

- “If you were plowing a field, which would you rather use: two strong oxen or 1024 chicken?”

Seymour Cray

WHAT IS THE PARALLEL PARADIGM?

- Parallel computing is a form of computation in which many calculations are carried out simultaneously.

- The paradigm operating principles resides in the ability to divide large problems into smaller components. These components can then be solved concurrently, in parallel.

WHAT IS THE PARALLEL PARADIGM?

- Depending on the architecture and hardware there are several different forms of parallel computing:
  - Bit-level
  - Instruction level
  - Data
  - Task parallelism

WHAT IS THE PARALLEL PARADIGM?

The most popular parallel architectures and template libraries are:

- CUDA: Compute Unified Device Architecture. Created by NVIDIA and implemented by the graphics processing units (GPUs) that they produce.

  http://www.nvidia.com/object/cuda_home_new.html

- TBB: Intel Threading Building Blocks. Parallelization on multi-core processors.

  https://www.threadingbuildingblocks.org/

WHAT IS THE PARALLEL PARADIGM?

The most popular parallel architectures and template libraries are:

- OpenCL: Open Computing Language. Open, royalty-free standard for cross-platform, parallel programming of modern processors found in personal computers, servers and handheld/embedded devices.

  http://www.khronos.org/opencl/

- OpenMP: Open Multi-Processing. API that supports multi-platform shared memory multiprocessing programming

  http://openmp.org/

WHAT IS THE PARALLEL PARADIGM?

The most popular parallel architectures and template libraries are:

- Open MPI: is a Message Passing Interface (MPI) library project combining technologies and resources from several other projects. MPI is a language-independent communications protocol used to program parallel computers.

  http://www.open-mpi.org/

WHAT IS CUDA?

From NVIDIA:

- CUDA is NVIDIA’s parallel computing architecture that enables dramatic increases in computing performance by harnessing the power of the GPU (graphics processing unit).

- With millions of CUDA-enabled GPUs sold to date, software developers, scientists and researchers are finding broad-ranging uses for CUDA, including image and video processing, computational biology and chemistry, fluid dynamics simulation, CT image reconstruction, seismic analysis, ray tracing, and much more.

WHAT IS CUDA?

General-purpose computing on graphics processing units (GPGPU) was initially difficult.

- Needed to write programs using graphics APIs: DirectX, OpenGL, Cg using shaders, textures.
- NVIDIA released the Compute Unified Device Architecture (CUDA) in 2006.
- A hardware architecture and programming model designed for GPU computing.

WHAT IS CUDA?

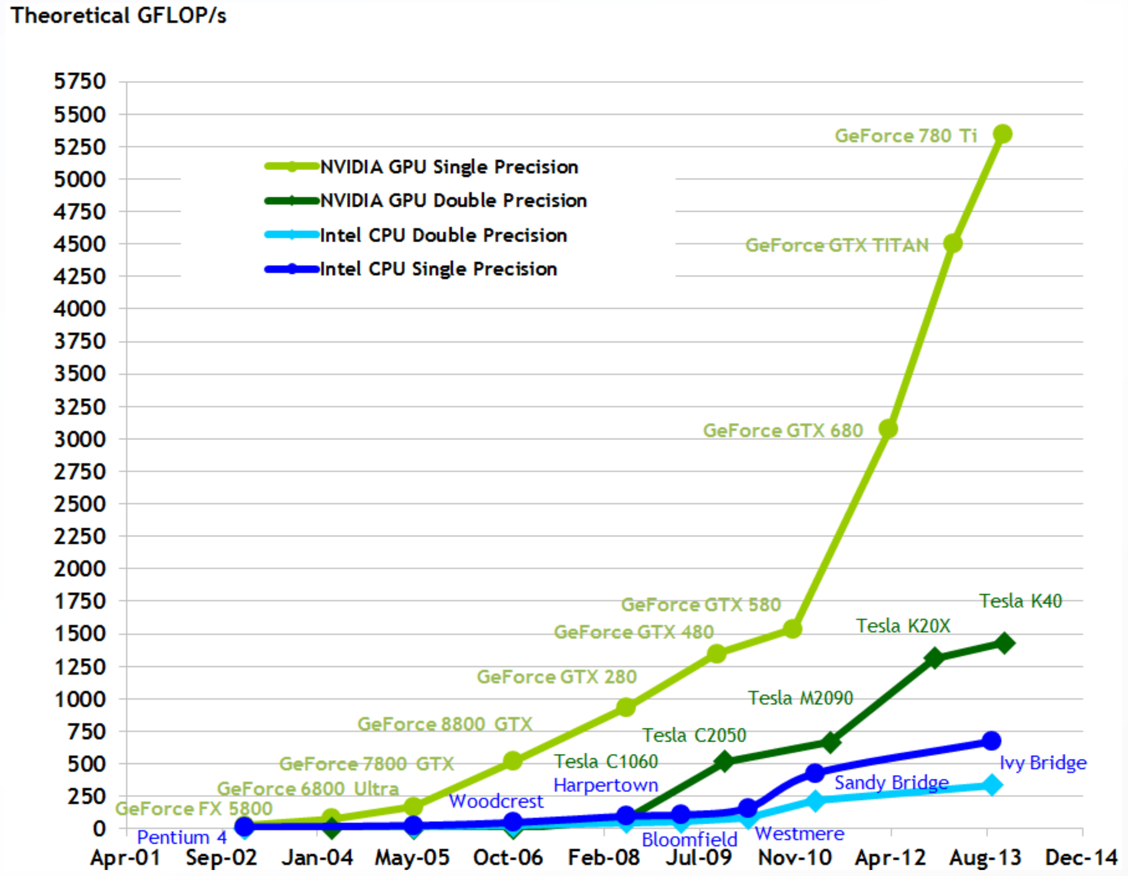
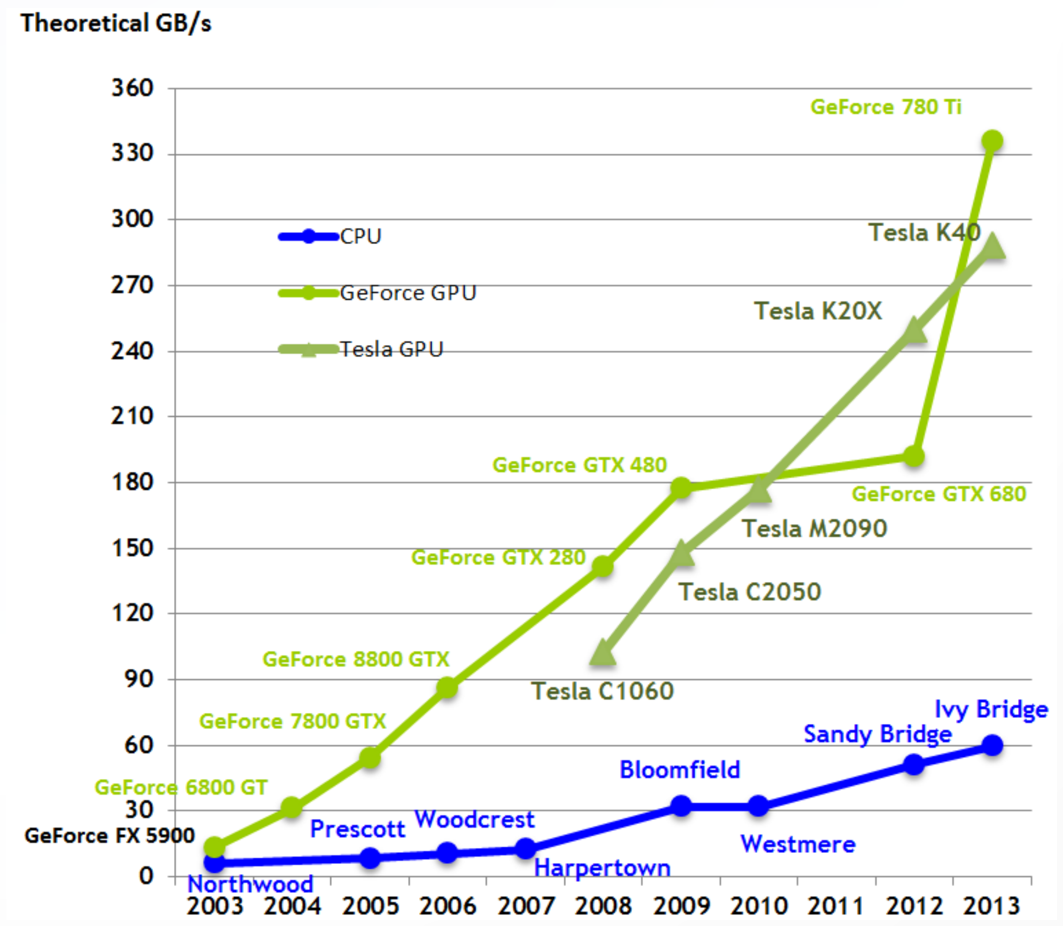


Figure 1. Floating-Point Operations per Second for the CPU and GPU

Read more at: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#ixzz3S6YypYMq
Follow us: @GPUComputing on Twitter | NVIDIA on Facebook

WHAT IS CUDA?



- Figure 2. Memory Bandwidth for the CPU and GPU

  Read more at: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#ixzz3S6ZEPVaL
  Follow us: @GPUComputing on Twitter | NVIDIA on Facebook

WHAT IS CUDA?

- Extends popular programming languages: C, C++, Java, Fortran
- Based on single instruction multiple thread (SITM) model.
- Threads perform the same task on different data.
- Programs will execute on any CUDA compatible NVIDA GPU.

WHAT IS CUDA?

The NVIDIA GPU architecture is built around a scalable array of multithreaded Streaming Multiprocessors (SMs). When a CUDA program on the host CPU invokes a kernel grid, the blocks of the grid are enumerated and distributed to multiprocessors with available execution capacity.

The threads of a thread block execute concurrently on one multiprocessor, and multiple thread blocks can execute concurrently on one multiprocessor. As thread blocks terminate, new blocks are launched on the vacated multiprocessors.

WHAT IS CUDA?

The multiprocessor creates, manages, schedules, and executes threads in groups of 32 parallel threads called warps.

Individual threads composing a warp start together at the same program address, but they have their own instruction address counter and register state and are therefore free to branch and execute independently.

WHAT IS CUDA?

When a multiprocessor is given one or more thread blocks to execute, it partitions them into warps and each warp gets scheduled by a warp scheduler for execution.

The way a block is partitioned into warps is always the same; each warp contains threads of consecutive, increasing thread IDs with the first warp containing thread 0

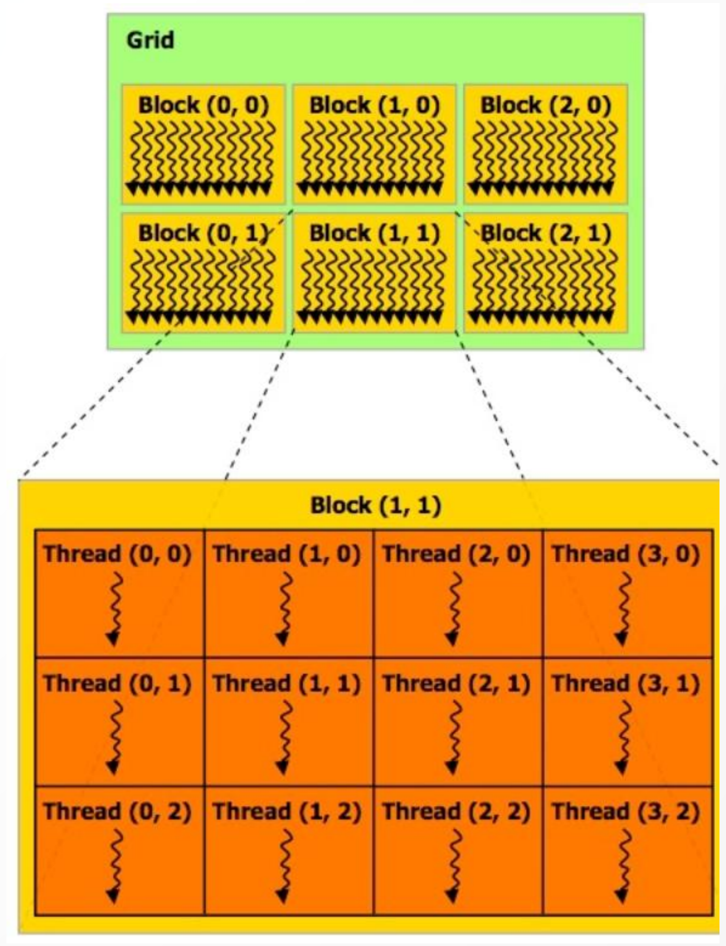
WHAT IS CUDA?

A warp executes one common instruction at a time, so full efficiency is realized when all 32 threads of a warp agree on their execution path.

If threads of a warp diverge via a data-dependent conditional branch, the warp serially executes each branch path taken, disabling threads that are not on that path, and when all paths complete, the threads converge back to the same execution path.

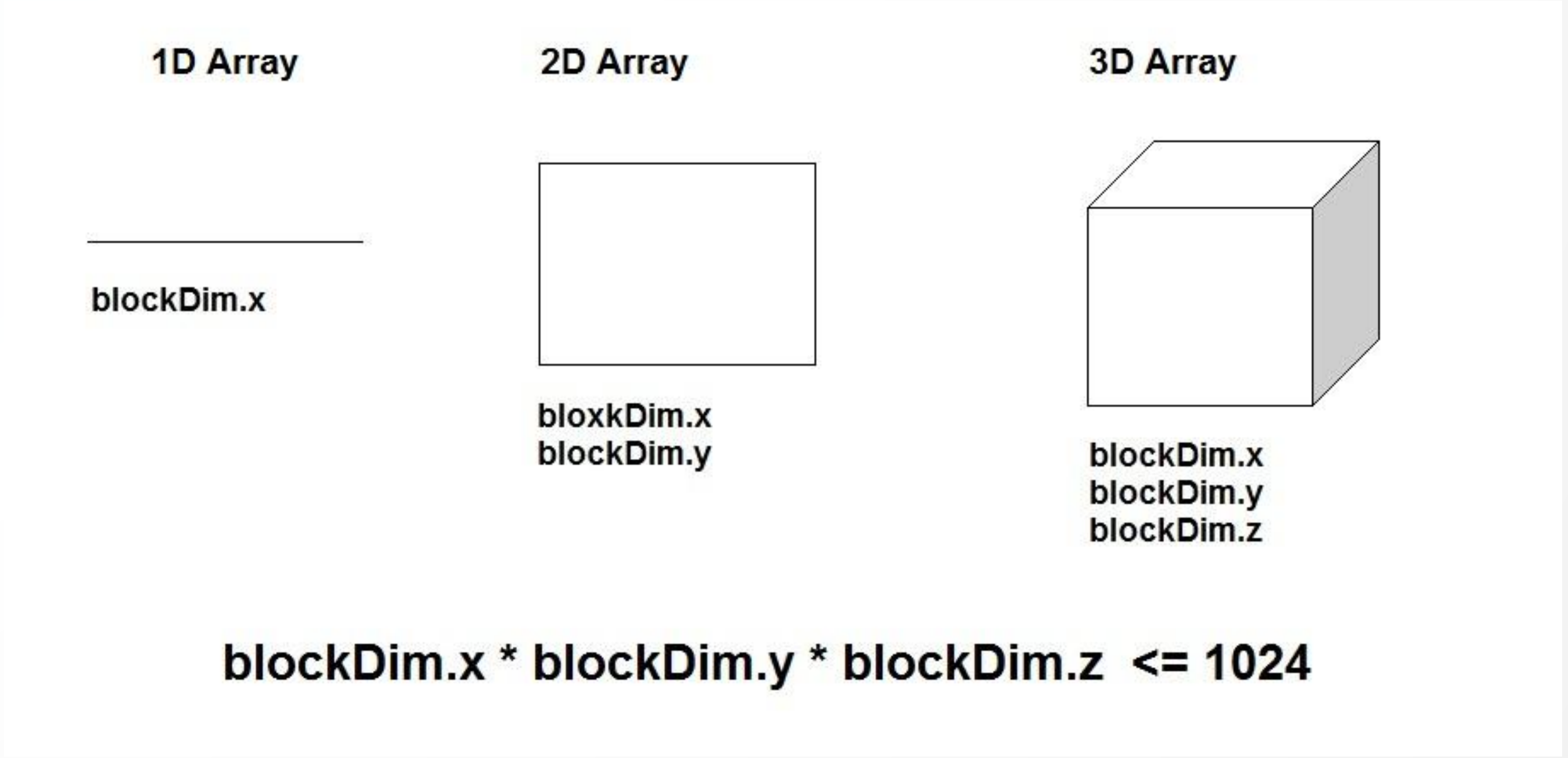
Branch divergence occurs only within a warp; different warps execute independently regardless of whether they are executing common or disjoint code paths.

WHAT IS CUDA?

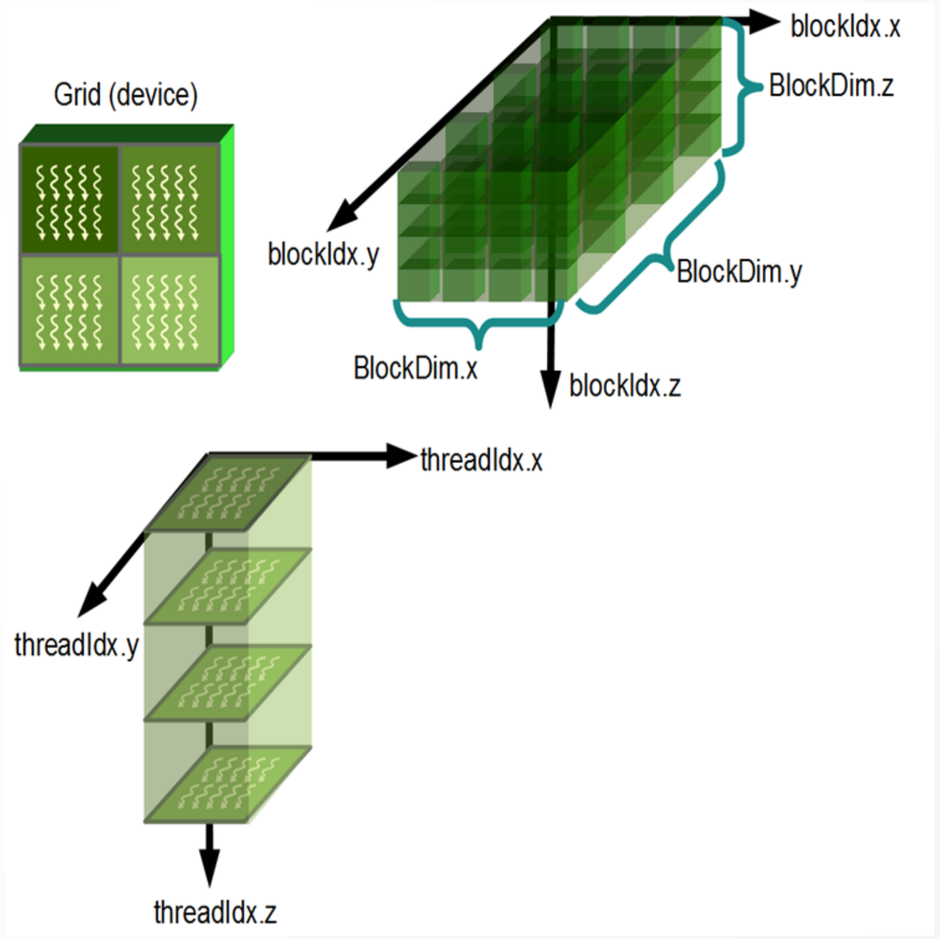

Grid
Block (0, 0)
Block (1, 0)
Block (2, 0)
Block (0, 1)
Block (1, 1)
Block (2, 1)
Block (1, 1)
Thread (0, 0)
Thread (1, 0)
Thread (2, 0)
Thread (3, 0)
Thread (0, 1)
Thread (1, 1)
Thread (2, 1)
Thread (3, 1)
Thread (0, 2)
Thread (1, 2)
Thread (2, 2)
Thread (3, 2)

WHAT IS CUDA?

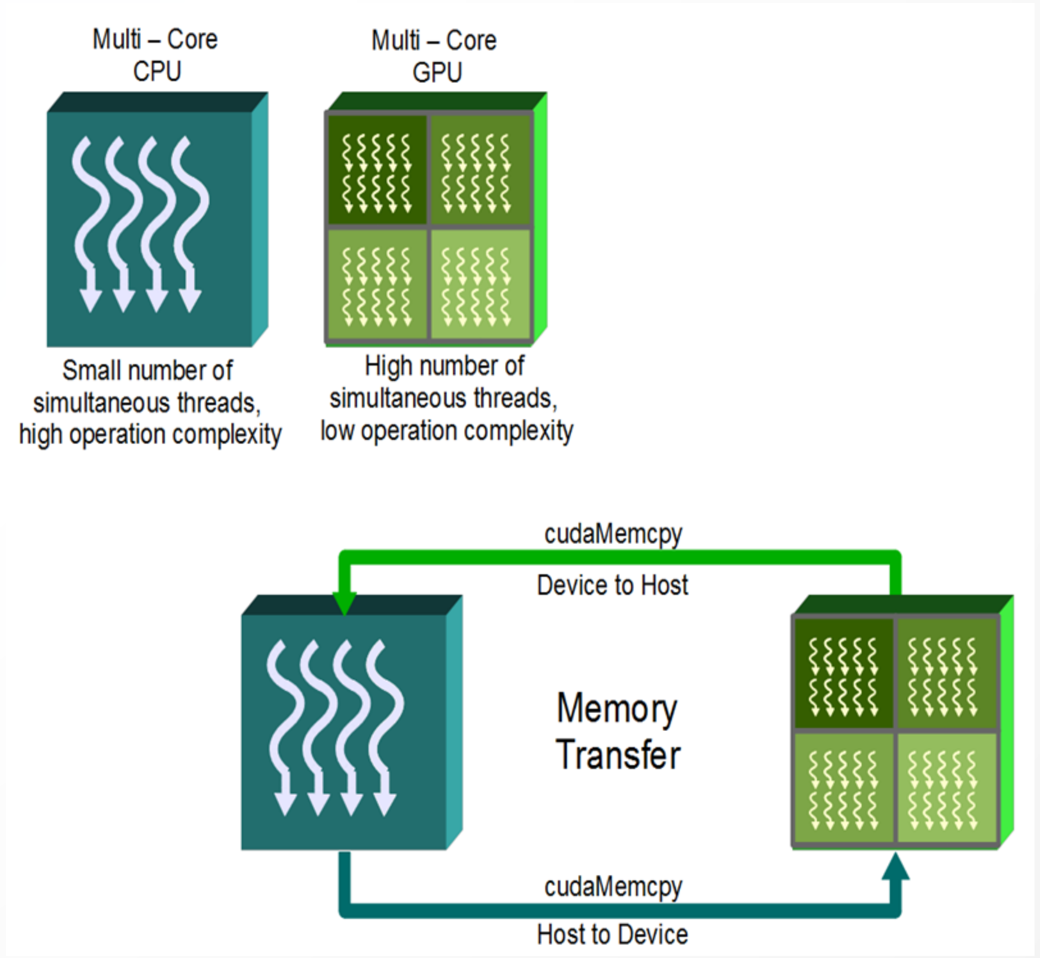
CUDA threads are executed in blocks. Blocks are an easy way to map threads to data structures. The SM can manage multiple blocks. Thread blocks are organised into grids.



WHAT IS CUDA?

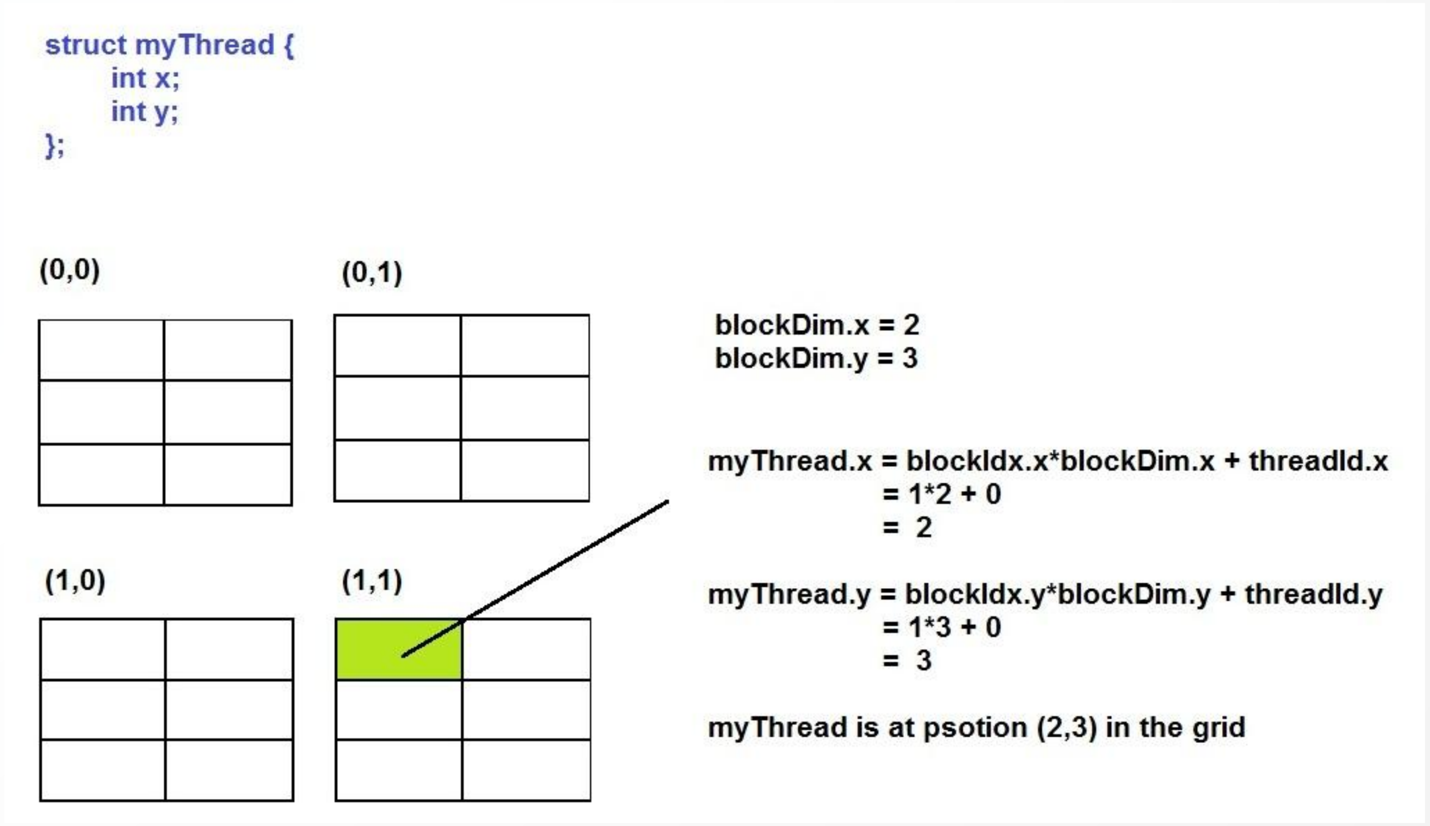

Grid (device)
blockIdx.x
BlockDim.z
blockIdx.y
BlockDim.y
BlockDim.x
blockIdx.z
threadIdx.x
threadIdx.y
threadIdx.z

WHAT IS CUDA?


Multi – Core
CPU
Multi – Core
GPU
Small number of simultaneous threads, high operation complexity
High number of simultaneous threads, low operation complexity
cudaMemcpy
Device to Host
Memory Transfer
cudaMemcpy
Host to Device

WHAT IS CUDA?

Suppose we have a 2x2 grid with 2x3 blocks. How is a thread specified?



```
struct myThread {
        int x;
        int y;
};
```

blockDim.x = 2
blockDim.y = 3

```
myThread.x = blockIdx.x*blockDim.x + threadId.x
           = 1*2 + 0
           =  2

myThread.y = blockIdx.y*blockDim.y + threadId.y
           = 1*3 + 0
           =  3
```

myThread is at psotion (2,3) in the grid

Test

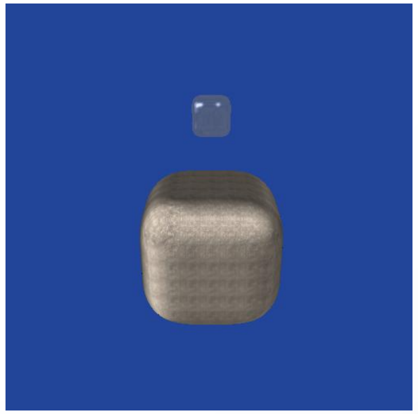
Numerical simulated drop of water falling towards a solid.

Number of solid particles constant.

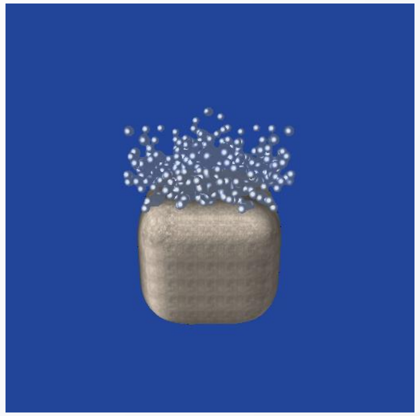
Number of fluids increased in each case.

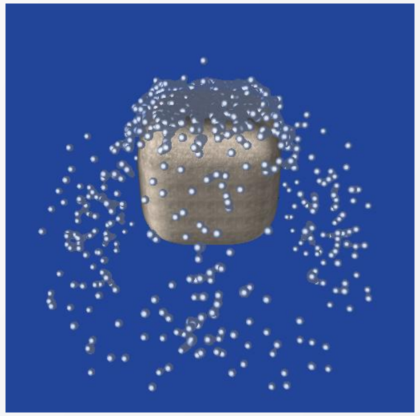
Time taken to complete physics calculations measured

Serial: Pentium i7 2.86 GHz, Parallel: GTX480


Increasing

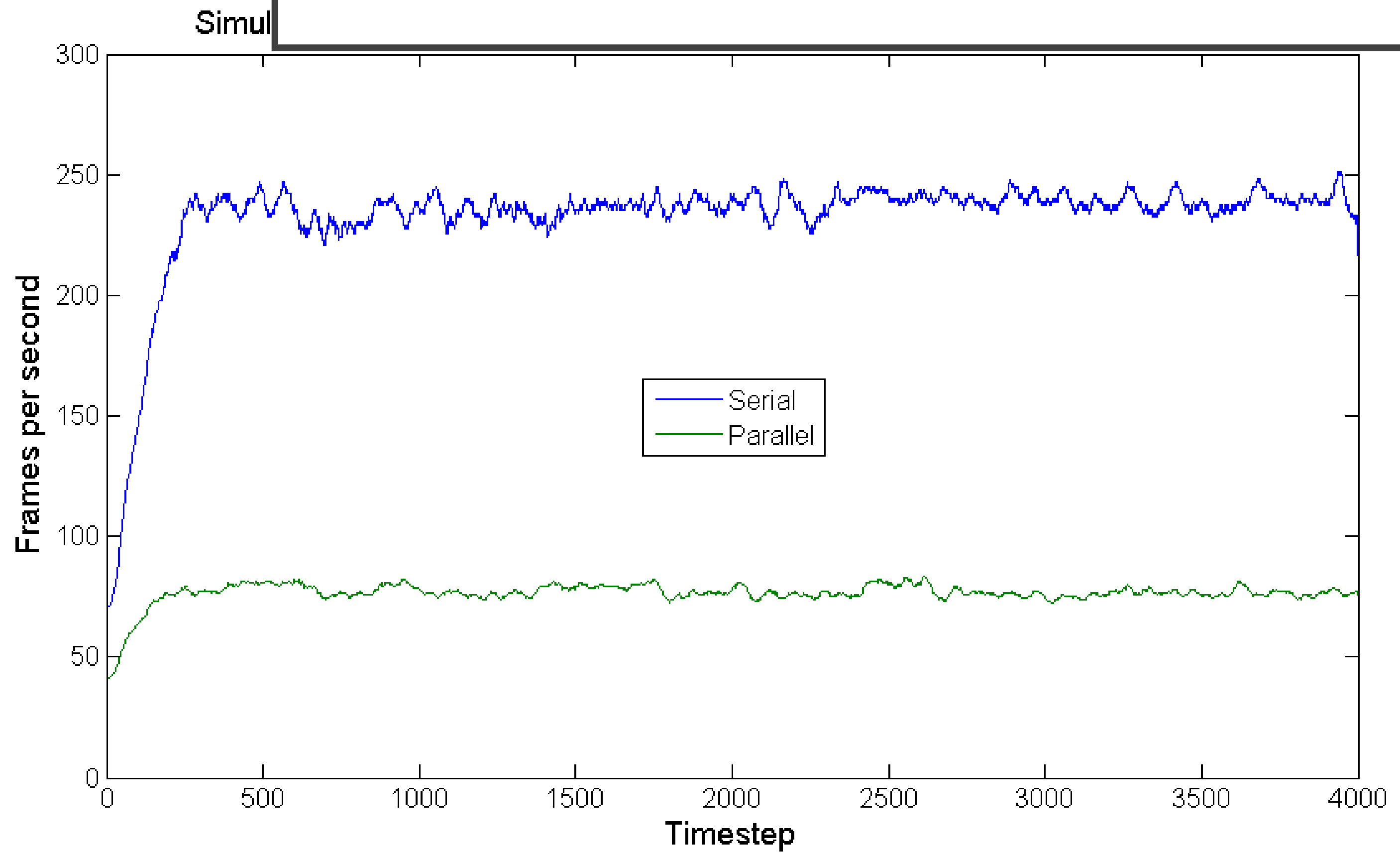
Decreasing


Increasing


Constant

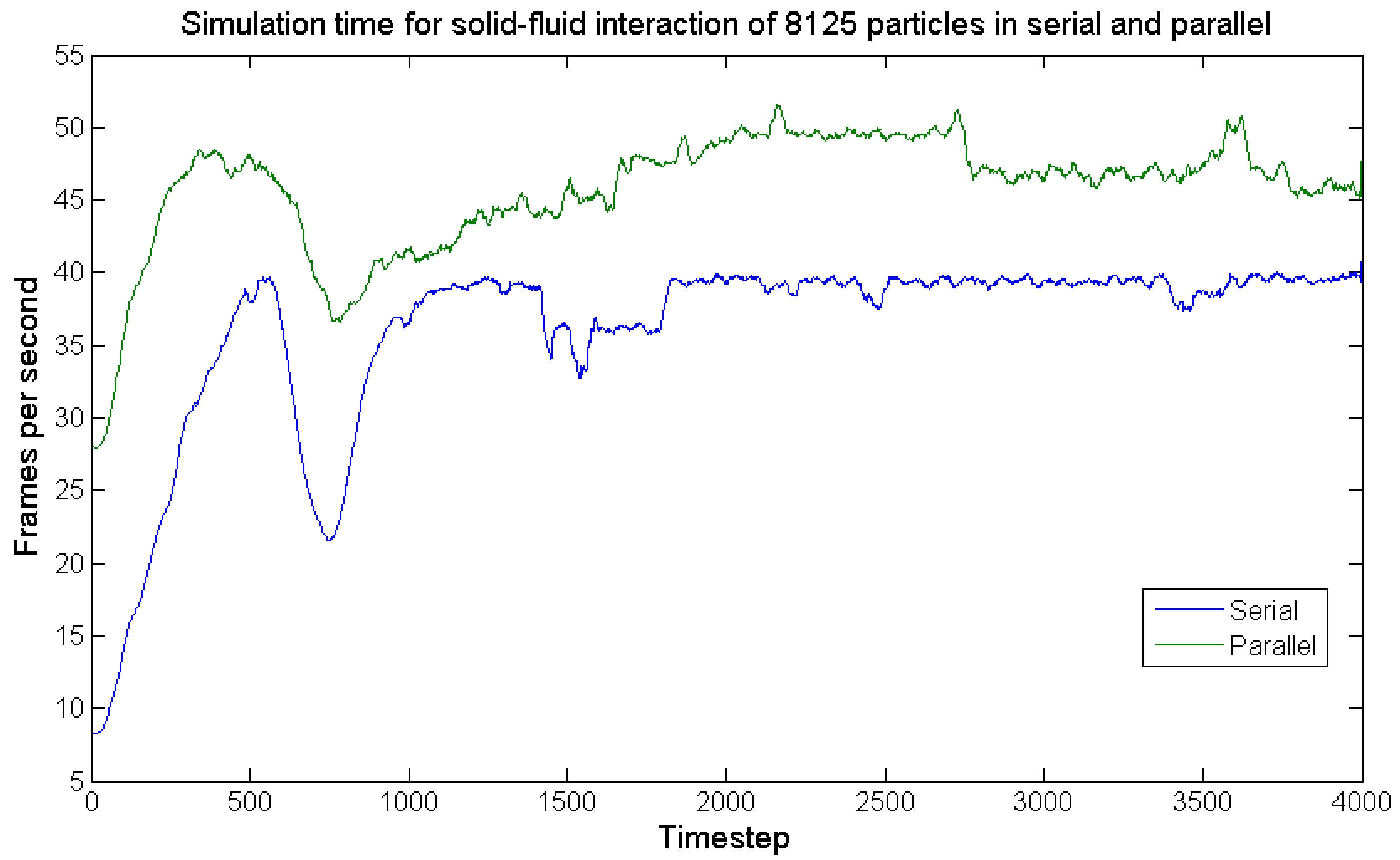
Performance over time

SMALL: 1,125 PARTICLES



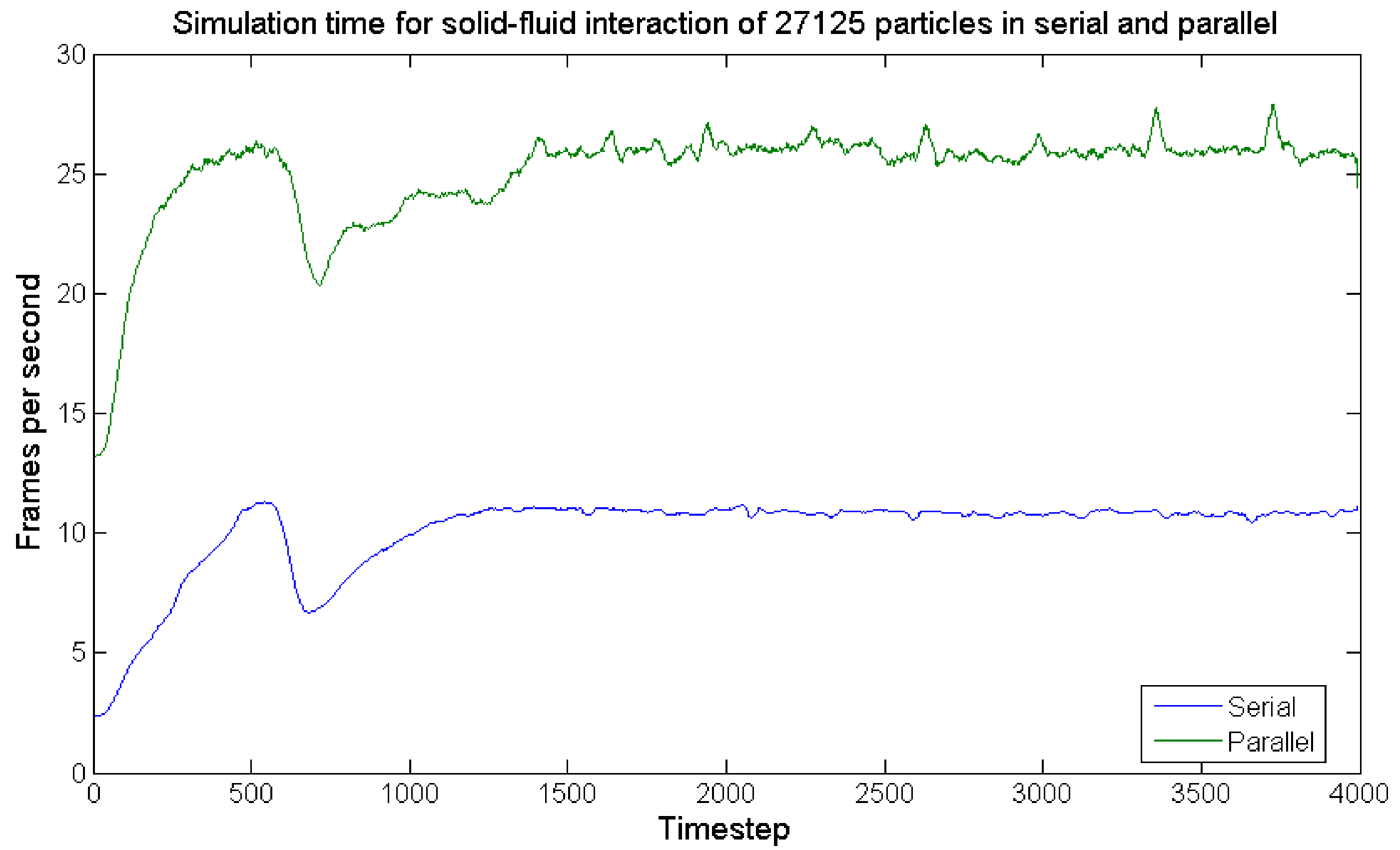
- Time taken: Serial 17.77 s , Parallel 52.85 s
- Serial is 2.97x faster
- Why? Memory transfer latencies to GPU

MEDIUM: 8,125 PARTICLES



- Notice how performance changes during the simulation
- Serial: 120.32s Parallel: 88.49 s, Parallel is 1.36x faster
- Performance gains from multithreading exceeding transfer costs

LARGE: 27,125 PARTICLES



- Serial: 425.70 s , Parallel: 161.40 s, Parallel is 2.63x faster
- Problem size is large enough for performance gains to show

CPU VS GPU

CPU: optimized for fast single-thread execution

- Cores designed to execute 1 thread or 2 threads concurrently
- Large caches attempt to hide DRAM access times
- Cores optimized for low latency cache accesses
- Complex control logic for speculation and out-of-order execution

GPU: optimized for high multi-thread throughput

- Cores designed to execute many parallel threads concurrently
- Cores optimized for data-parallel, throughput computation
- Chips use extensive multithreading to tolerate DRAM access times

CPU VS GPU

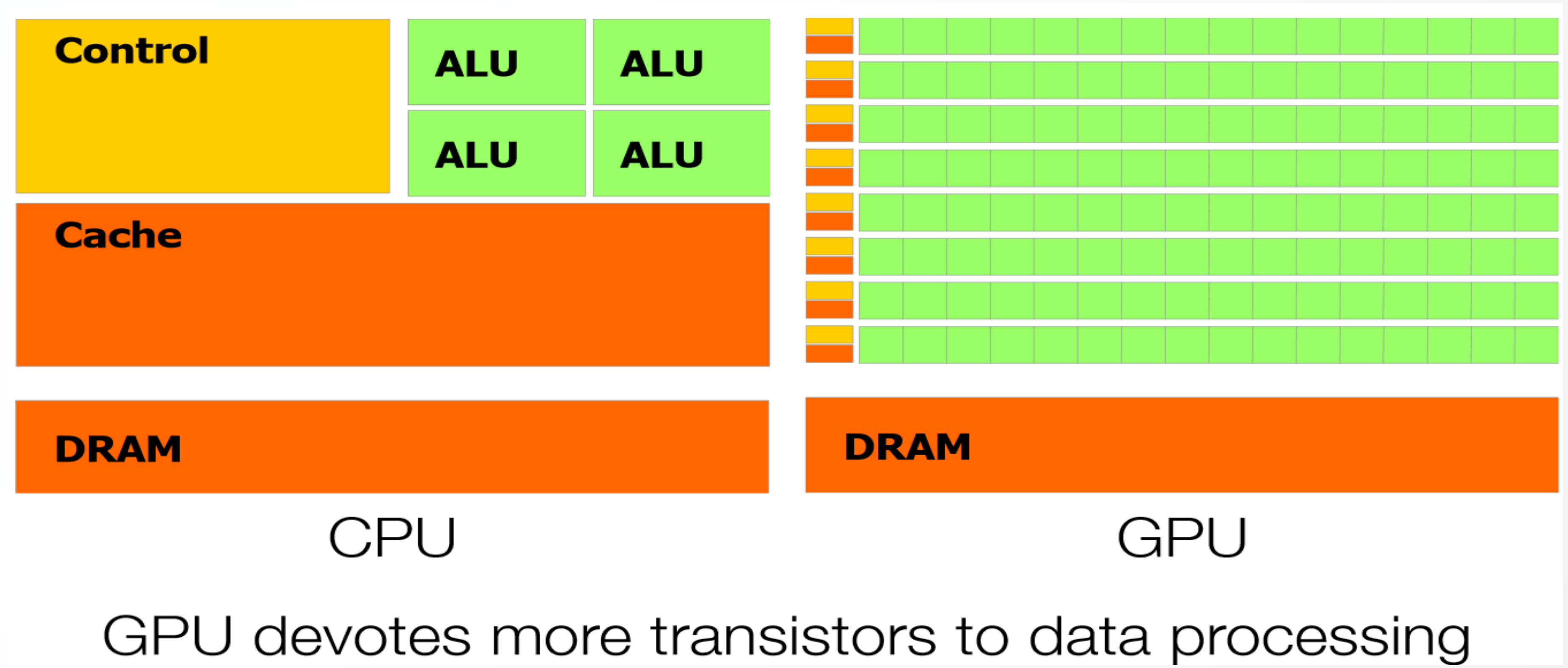
CPU: optimized for fast single-thread execution

- Cores designed to execute 1 thread or 2 threads concurrently
- Large caches attempt to hide DRAM access times
- Cores optimized for low latency cache accesses
- Complex control logic for speculation and out-of-order execution

GPU: optimized for high multi-thread throughput

- Cores designed to execute many parallel threads concurrently
- Cores optimized for data-parallel, throughput computation
- Chips use extensive multithreading to tolerate DRAM access times

CPU VS GPU



In computing, an arithmetic and logic unit (ALU) is a digital circuit that performs integer arithmetic and logical operations.

LEARN TO CHOOSE, THE BEST SOLUTION FOR DIFFERENT PROBLEMS

When developing numerical solutions is important to understand the mathematics, data size and computational time required.

The parallel programing starts with pen and paper, is important to first been able to separated the problems into components.

Depending on the data size and the form of separation a parallel paradigm, architecture or library can be chosen.