UNIVERSITAT POLITÈCNICA DE VALÈNCIA

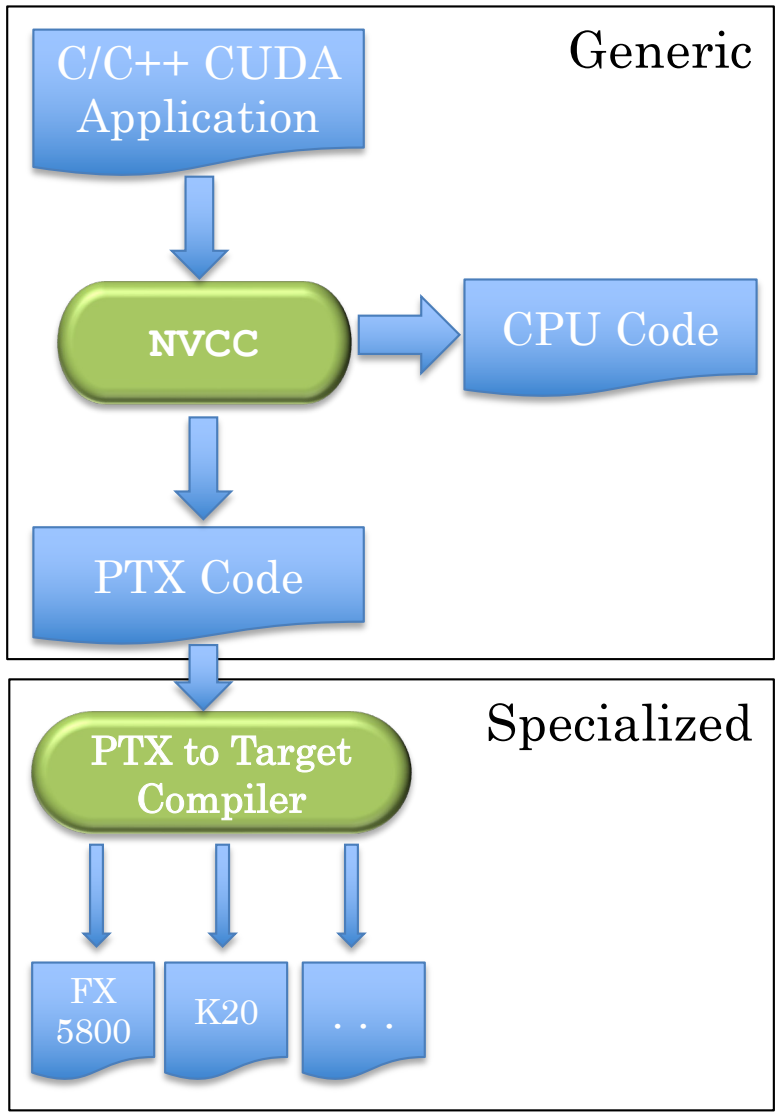
# CUDA: Programming Interface

**Adrián Castelló y Pedro Alonso**

**Universitat Politècnica de València**

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# Compilation with **nvcc**



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# Compilation with **nvcc**

- Kernels can be written in **ptx** (Parallel Thread eXecution)
- **nvcc** is a compiler driver that simplifies the process of compiling *C* or *PTX* code
- Compilation workflow: **nvcc**'s basic workflow consists in separating device code from host code
  - compiling the device code into an assembly form (*PTX* code) and/or binary form (*cubin* object),
  - and modifying the host code by replacing the **<<<...>>>** syntax by the necessary CUDA C runtime function calls to load and launch each compiled kernel from the *PTX* code and/or *cubin* object.
- Just-in-Time Compilation:
  - Any *PTX* code loaded by an application at runtime is compiled further to binary code by the device driver. (This is controlled by an environment variable).

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# Compilation with **nvcc**

- Binary Compatibility
  - Binary code is architecture-specific.
  - A *cubin* object is generated using the compiler option **-code** that specifies the targeted architecture (**-code=sm_13** -- compute capability 1.3).
- PTX Compatibility
  - Some *PTX* instructions are only supported on devices of higher compute capabilities
  - The **–arch** compiler option specifies the compute capability that is assumed when compiling C to *PTX* code (**-arch=sm_13** for double-precision)
- Application Compatibility
  - *PTX* and binary code embedded in a CUDA C application is controlled by the **-arch** and **-code** compiler options or the **-gencode** compiler option
  - For example, **arch=sm_13** is a shorthand for **arch=compute_13 code=compute_13,sm_13**, which is the same as **gencode arch=compute_13,code=\'compute_13,sm_13\'**.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# Compilation with **nvcc**

- C/C++ Compatibility
  - The compiler processes CUDA source files according to C++ syntax rules
- 64-Bit Compatibility
  - The 64-bit version of **nvcc** compiles device code in 64-bit mode (the same for 32-bit version)

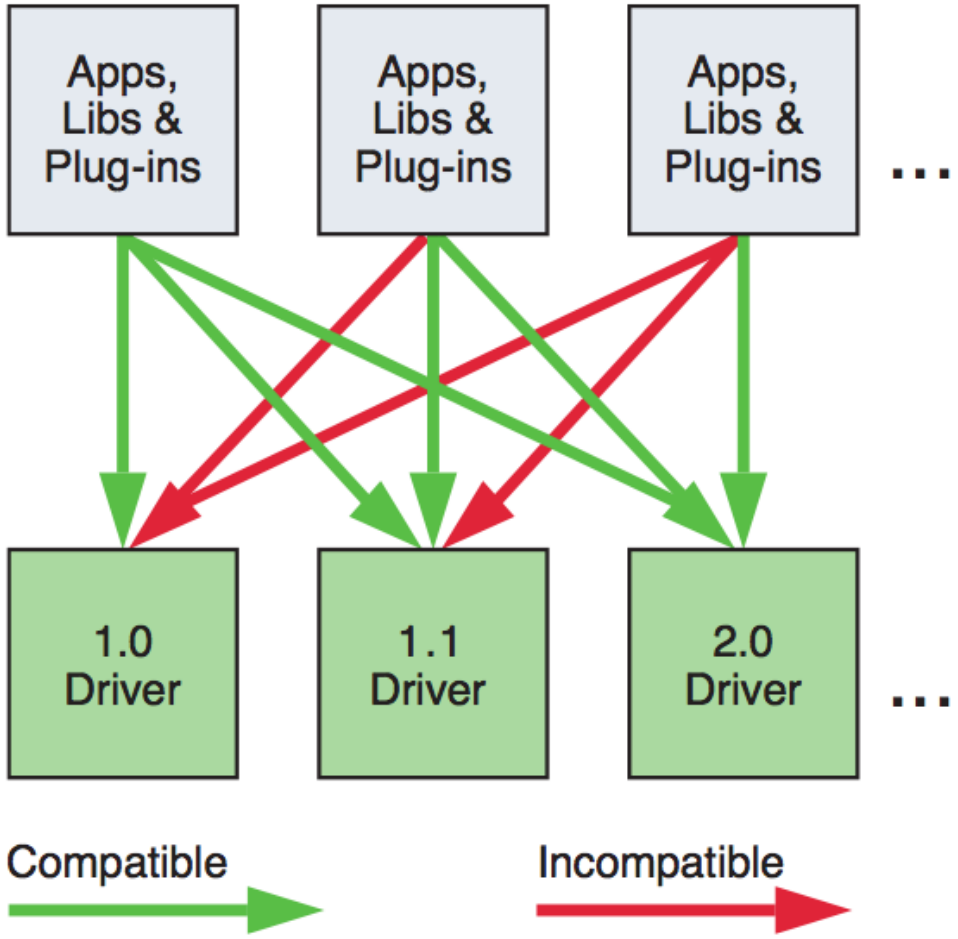
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# Versioning and Compatibility

- Two version numbers:
  - The compute capability (of the device), and
  - the version of the CUDA driver API (software: driver and runtime).
- The version of the driver API is defined in the driver header file as `CUDA_VERSION`:
  - All applications, plug-ins, and libraries on a system must use the same version of the CUDA driver API, since only one version of the CUDA device driver can be installed on a system.
  - All plug-ins and libraries used by an application must use the same version of the runtime.
  - All plug-ins and libraries used by an application must use the same version of any libraries that use the runtime (such as CUFFT, CUBLAS, ...).

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# Versioning and Compatibility



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# CUDA C Runtime: Initialization

- The runtime is implemented in the **`cudart`** library which is typically included in the application installation package (`cudart.lib`, `libcudart.a`, `cudart.dll`, `libcudart.so`).
- All its entry points are prefixed with **`cuda`**.
- Initialization
  - There is no explicit initialization function for the runtime.
  - It initializes the first time a runtime function is called.
- A CUDA context is analogous to a CPU process.
  - All resources and actions performed within the driver API are encapsulated inside a CUDA context.
  - Besides objects such as modules and texture or surface references, each context has its own distinct address space.
  - During initialization, the runtime creates a CUDA context for each device in the system (*primary context*).
  - The context is shared among all the host threads of the application.
  - When a host thread calls **`cudaDeviceReset()`**, this destroys the primary context.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# CUDA C Runtime: Device Memory

- Device memory can be allocated either as *linear memory* or as *CUDA arrays*.
- Linear memory:
  - exists on the device in a 40-bit address space of devices of compute capability 2.x
  - **`cudaMalloc()`**, **`cudaFree()`**, **`cudaMemcpy()`**
  - **`cudaMallocPitch()`**, **`cudaMalloc3D()`**
- CUDA arrays are opaque memory layouts optimized for texture fetching

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# CUDA C Runtime: Device Memory

```
printf("Allocating memory in GPU...\n");
double *d_A, *d_B, *d_C;
cudaError_t cudaError;
cudaError = cudaMalloc((void **) &d_A, m*n*sizeof(double) );
cudaError = cudaMalloc((void **) &d_B, m*n*sizeof(double) );
cudaError = cudaMalloc((void **) &d_C, m*n*sizeof(double) );
if (cudaError != cudaSuccess) {
  fprintf(stderr,"CUDA: error occurred in cudaMalloc. Exiting...\n");
  exit(cudaError);
}
printf("Tranferring data CPU-GPU...\n");
cudaMemcpy(d_A, A, m * n * sizeof(double), cudaMemcpyHostToDevice);
cudaMemcpy(d_B, B, m * n * sizeof(double), cudaMemcpyHostToDevice);

// Computation...

printf("Tranferring result GPU-CPU...\n");
cudaMemcpy(D, d_C, m * n * sizeof(double), cudaMemcpyDeviceToHost);

//More operations...

cudaFree(d_A);
cudaFree(d_B);
cudaFree(d_C);
```

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# CUDA C Runtime: Device Memory

- Device memory can be allocated either as *linear memory* or as *CUDA arrays*.
- Linear memory:
  - exists on the device in a 40-bit address space of devices of compute capability 2.x
  - `cudaMalloc()`, `cudaFree()`, `cudaMemcpy()`
  - `cudaMallocPitch()`, `cudaMalloc3D()`
- CUDA arrays are opaque memory layouts optimized for texture fetching

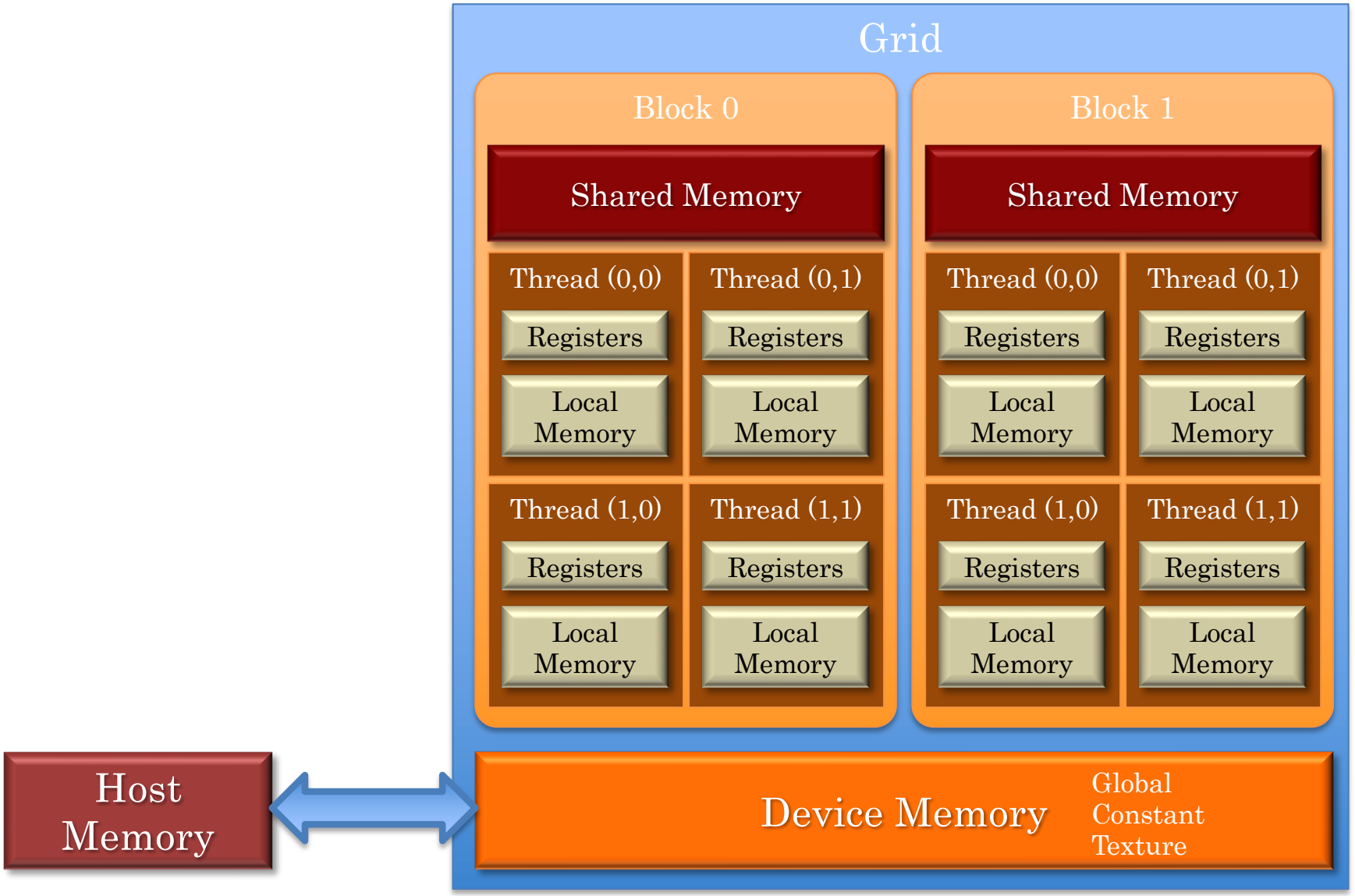
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# Performance issues: Device Memory Accesses

- An instruction that accesses addressable memory might need to be re-issued multiple times depending on the distribution of the memory addresses across the threads within the warp.
- How the distribution affects the instruction throughput this way is specific to each type of memory
- A general rule: the more scattered the addresses are, the more reduced the throughput is.
- Global memory resides in device memory and device memory is accessed via 32-, 64-, or 128-byte memory transactions.
- Segments of device memory that are aligned to their size (i.e., whose first address is a multiple of their size) can be read or written by memory transactions.
- *When a warp executes an instruction that accesses global memory, it coalesces the memory accesses of the threads within the warp*.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# Memory Spaces



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# Performance with and without Shared Memory

- Matrix Multiplication without Shared Memory:

```
[Matrix Multiply Using CUDA] - Starting...
GPU Device 0: "Tesla K20c" with compute capability 3.5

MatrixA(320,320), MatrixB(640,320)
Computing result using CUDA Kernel...
done
Performance= 88.14 GFlop/s, Time= 1.487 msec, Size= 131072000 Ops, WorkgroupSize= 1024
threads/block
Checking computed result for correctness: Result = PASS
```

- Matrix Multiplication with Shared Memory:

```
[Matrix Multiply Using CUDA] - Starting...
GPU Device 0: "Tesla K20c" with compute capability 3.5

MatrixA(320,320), MatrixB(640,320)
Computing result using CUDA Kernel...
done
Performance= 246.11 GFlop/s, Time= 0.533 msec, Size= 131072000 Ops, WorkgroupSize= 1024
threads/block
Checking computed result for correctness: Result = PASS
```

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# CUDA C Runtime: Page-Locked Host Memory (*pinned*)

- As opposed to regular pageable host memory **`malloc()`**, **`cudaHostAlloc()`** and **`cudaFreeHost()`** allocate and free *page-locked* (or *pinned*) host memory.
- Benefits:
  - Copies between page-locked host memory and device memory can be performed concurrently with kernel execution.
  - Page-locked host memory can be mapped into the address space of the device (flag **`cudaHostAllocMapped`**)
  - Bandwidth between host memory and device memory is higher.
- Constraint:
  - Page-locked host memory is a scarce resource however: consuming too much page-locked memory reduces overall system performance.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# CUDA C Runtime: Asynchronous Concurrent Execution

- Concurrent Execution between Host and Device
  - Kernel launches;
  - Memory copies between two addresses to the same device memory;
  - Memory copies from host to device of a memory block of 64 KB or less;
  - Memory copies performed by functions that are suffixed with **`Async`**;
  - Memory set function calls.
- Overlap of Data Transfer and Kernel Execution
  - Copies between page-locked host memory and device memory concurrently with kernel execution.
- Concurrent Kernel Execution
  - Devices of compute capability >= 2.x.
- Concurrent Data Transfers
  - Copy from page-locked host memory to device memory concurrently with a copy from device memory to page-locked host memory.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# CUDA C Runtime: Streams

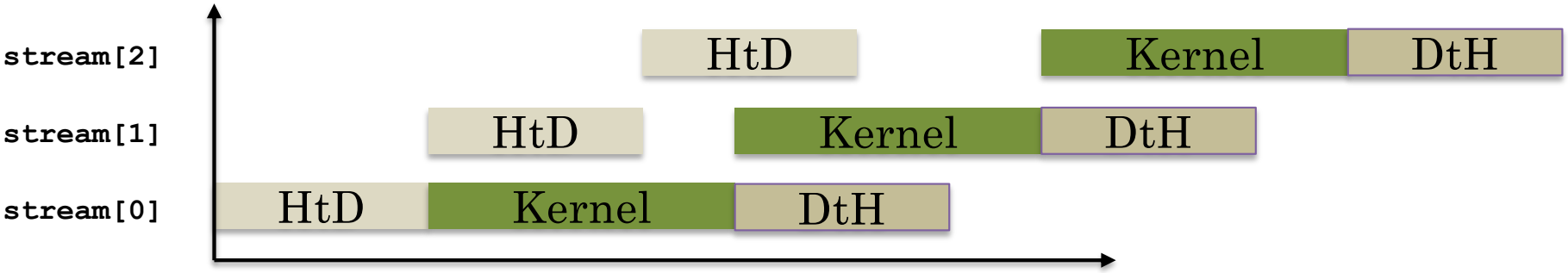
- Applications manage concurrency through *streams*.
- Streams: A stream is a sequence of commands that execute in order.
- Creation and Destruction

```
// allocate and initialize an array of stream handles
cudaStream_t *streams = (cudaStream_t*) malloc(nstreams * sizeof(cudaStream_t));
for(int i = 0; i < nstreams; i++)
    cutilSafeCall( cudaStreamCreate(&(streams[i])) );

// asynchronously launch nstreams kernels, each operating on its own portion of data
for(int i = 0; i < nstreams; i++)
    init_array<<<blocks, threads, 0, streams[i]>>>(d_a + i * n / nstreams, d_c, niterations);

// asynchronoously launch nstreams memcopies.  Note that memcopy in stream x will only
//   commence executing when all previous CUDA calls in stream x have completed
for(int i = 0; i < nstreams; i++)
    cudaMemcpyAsync(a + i * n / nstreams, d_a + i * n / nstreams, nbytes / nstreams,
                                                    cudaMemcpyDeviceToHost, streams[i]);

// release resources
for(int i = 0; i < nstreams; i++)
    cudaStreamDestroy(streams[i]);
```

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# The pipeline of Streams





UNIVERSITAT POLITÈCNICA DE VALÈNCIA

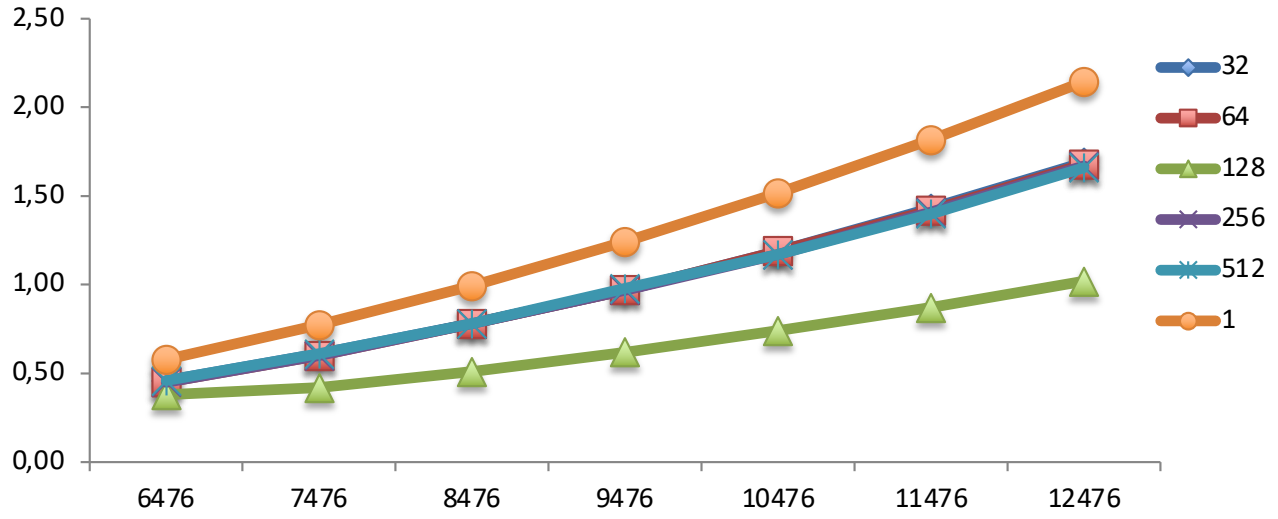
# CUDA C Runtime: Streams

```
cudaStream_t stream[2];
for( int i = 0; i < 2; ++i )
    cudaStreamCreate(&stream[i]);
float* hostPtr;
cudaMallocHost(&hostPtr, 2 * size);

for( int i = 0; i < 2; ++i ) {
  cudaMemcpyAsync(inputDevPtr + i * size, hostPtr + i * size,
                  size, cudaMemcpyHostToDevice, stream[i]);
  MyKernel <<<100, 512, 0, stream[i]>>> (outputDevPtr + i * size, inputDevPtr + i * size, size);
  cudaMemcpyAsync(hostPtr + i * size, outputDevPtr + i * size,
                  size, cudaMemcpyDeviceToHost, stream[i]);
}

for( int i = 0; i < 2; ++i )
    cudaStreamDestroy(stream[i]);
```

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# Streams timing example

| Streams/ N | 1 | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|---|
| 6476 | 0,58 | 0.46 | 0.45 | 0.38 | 0.45 | 0.46 |
| 7476 | 0,77 | 0.61 | 0.60 | 0.42 | 0.60 | 0.61 |
| 8476 | 1,00 | 0.78 | 0.78 | 0.51 | 0.78 | 0.78 |
| 9476 | 1,24 | 0.97 | 0.97 | 0.62 | 0.97 | 0.98 |
| 10476 | 1,52 | 1.19 | 1.19 | 0.74 | 1.17 | 1.17 |
| 11476 | 1,82 | 1.43 | 1.42 | 0.87 | 1.41 | 1.40 |
| 12476 | 2,15 | 1.69 | 1.68 | 1.02 | 1.67 | 1.66 |



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# CUDA C Runtime: Multi-Device System

- Device Enumeration

```
int deviceCount;
cudaGetDeviceCount(&deviceCount);
int device;
for (device = 0; device < deviceCount; ++device) {
     cudaDeviceProp deviceProp;
     cudaGetDeviceProperties(&deviceProp, device);
     printf("Device %d has compute capability %d.%d.\n",
          device, deviceProp.major, deviceProp.minor);
}
```

- Device Selection: **`cudaSetDevice()`**

```
size_t size = 1024 * sizeof(float);
cudaSetDevice(0);              // Set device 0 as current
float* p0;
cudaMalloc(&p0, size);     // Allocate memory on device 0
MyKernel<<<1000, 128>>>(p0);   // Launch kernel on device 0
cudaSetDevice(1);              // Set device 1 as current
float* p1;
cudaMalloc(&p1, size);     // Allocate memory on device 1
MyKernel<<<1000, 128>>>(p1);   // Launch kernel on device 1
```