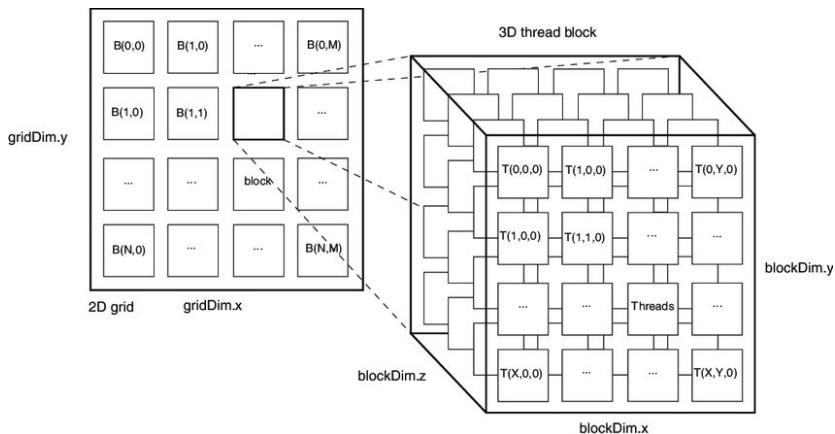
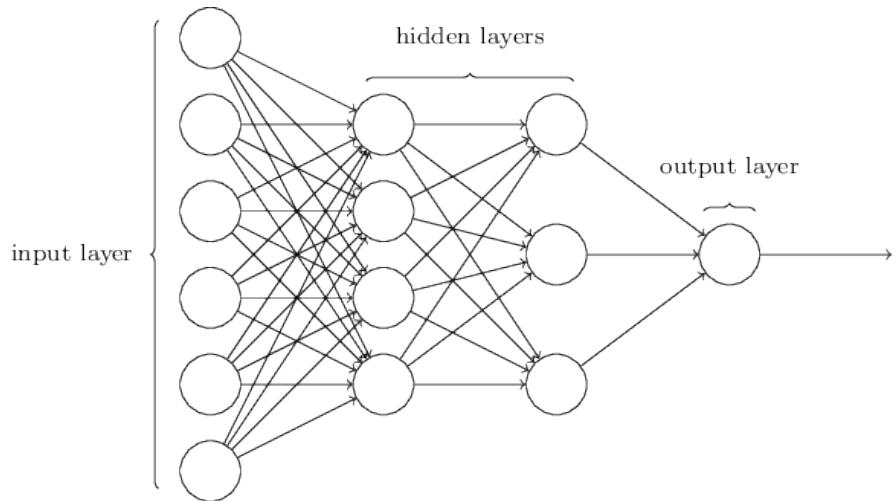


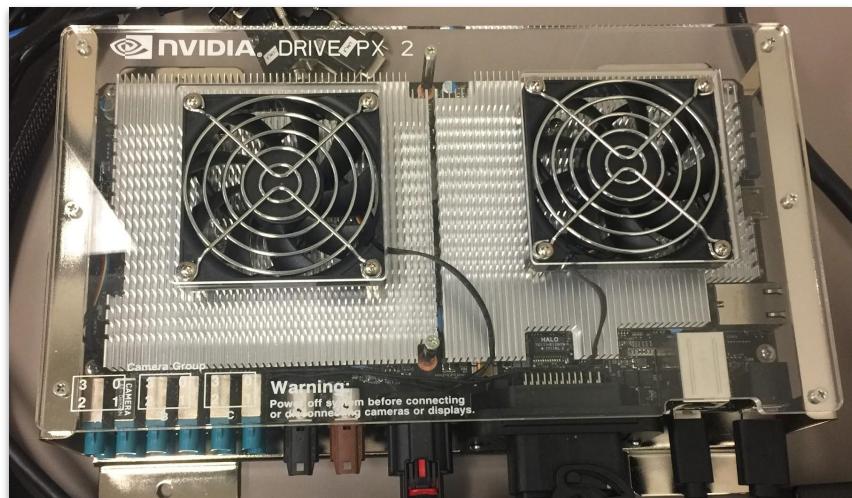
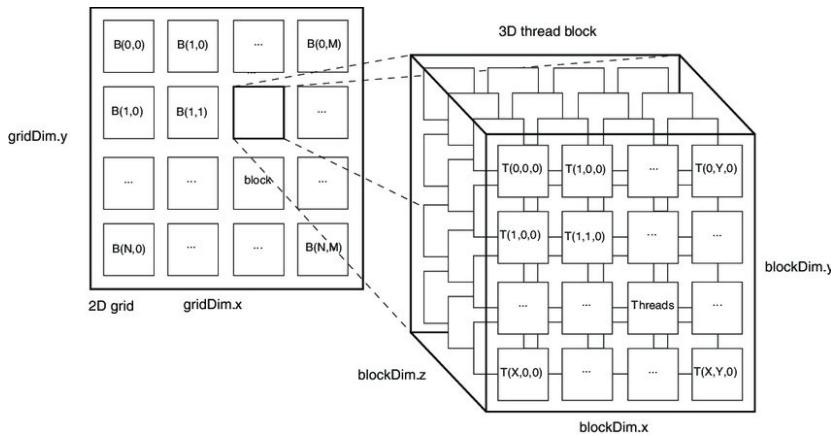
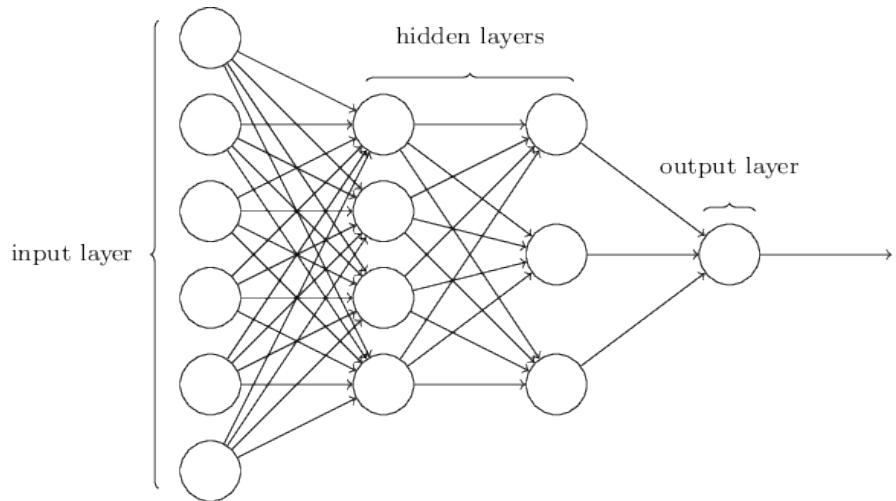


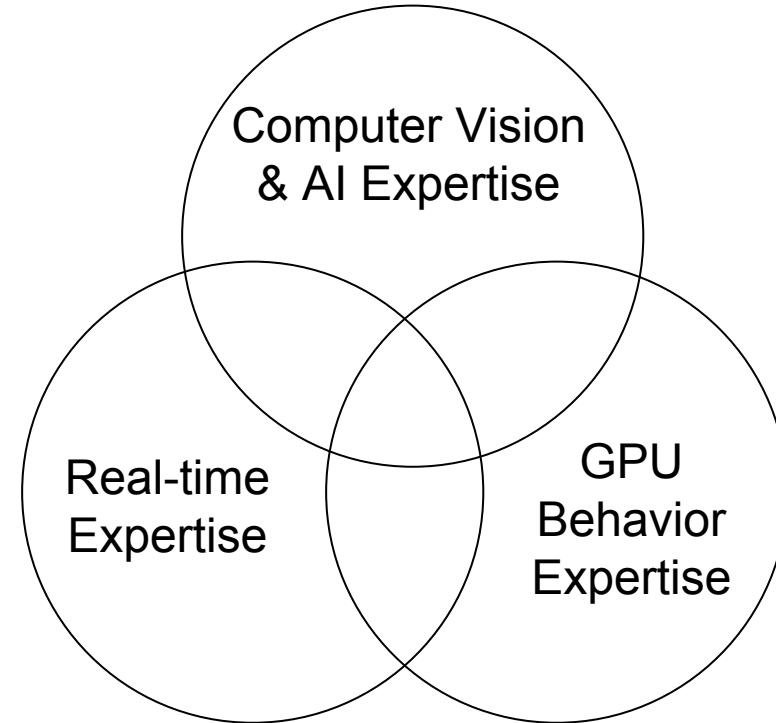
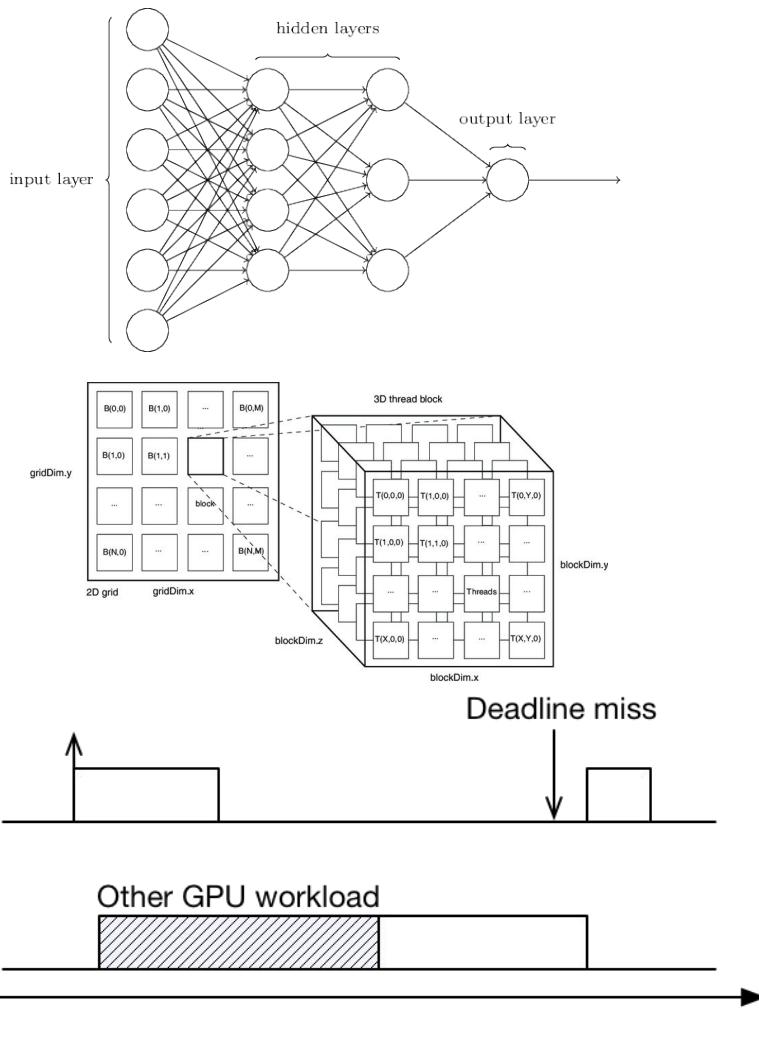
THE UNIVERSITY
of NORTH CAROLINA
at CHAPEL HILL

Avoiding Pitfalls when Using NVIDIA GPUs for Real-Time Tasks in Autonomous Systems

Ming Yang, **Nathan Otterness**, Tanya Amert, Joshua Bakita,
James H. Anderson, F. Donelson Smith







Pitfalls for Real-Time GPU Usage

- Synchronization and blocking
- GPU concurrency and performance
- CUDA programming perils

CUDA Programming Fundamentals

(i) Allocate GPU memory	<code>cudaMalloc (&devicePtr, bufferSize);</code>
(ii) Copy data from CPU to GPU	<code>cudaMemcpy (devicePtr, hostPtr, bufferSize);</code>
(iii) Launch the kernel <i>(kernel = code that runs on GPU)</i>	<code>computeResult<<<numBlocks, threadsPerBlock>>>(devicePtr);</code>
(iv) Copy results from GPU to CPU	<code>cudaMemcpy (hostPtr, devicePtr, bufferSize);</code>
(v) Free GPU memory	<code>cudaFree (devicePtr);</code>

CUDA Programming Fundamentals

(i) Allocate GPU memory	<code>cudaMalloc (&devicePtr, bufferSize);</code>
(ii) Copy data from CPU to GPU	<code>cudaMemcpy (devicePtr, hostPtr, bufferSize);</code>
(iii) Launch the kernel <i>(kernel = code that runs on GPU)</i>	<code>computeResult<<<numBlocks, threadsPerBlock>>>(devicePtr);</code>
(iv) Copy results from GPU to CPU	<code>cudaMemcpy (hostPtr, devicePtr, bufferSize);</code>
(v) Free GPU memory	<code>cudaFree (devicePtr);</code>

CUDA Programming Fundamentals

(i) Allocate GPU memory	cudaMalloc (&devicePtr, bufferSize);
(ii) Copy data from CPU to GPU	cudaMemcpy (devicePtr, hostPtr, bufferSize);
(iii) Launch the kernel <i>(kernel</i> = code that runs on GPU)	computeResult<<<numBlocks, threadsPerBlock>>>(devicePtr);
(iv) Copy results from GPU to CPU	cudaMemcpy (hostPtr, devicePtr, bufferSize);
(v) Free GPU memory	cudaFree (devicePtr);

CUDA Programming Fundamentals

(i) Allocate GPU memory	cudaMalloc (&devicePtr, bufferSize);
(ii) Copy data from CPU to GPU	cudaMemcpy (devicePtr, hostPtr, bufferSize);
(iii) Launch the kernel <i>(kernel = code that runs on GPU)</i>	computeResult<<<numBlocks, threadsPerBlock>>>(devicePtr);
(iv) Copy results from GPU to CPU	cudaMemcpy (hostPtr, devicePtr, bufferSize);
(v) Free GPU memory	cudaFree (devicePtr);

CUDA Programming Fundamentals

(i) Allocate GPU memory	cudaMalloc (&devicePtr, bufferSize);
(ii) Copy data from CPU to GPU	cudaMemcpy (devicePtr, hostPtr, bufferSize);
(iii) Launch the kernel <i>(kernel = code that runs on GPU)</i>	computeResult<<<numBlocks, threadsPerBlock>>>(devicePtr);
(iv) Copy results from GPU to CPU	cudaMemcpy (hostPtr, devicePtr, bufferSize);
(v) Free GPU memory	cudaFree (devicePtr);

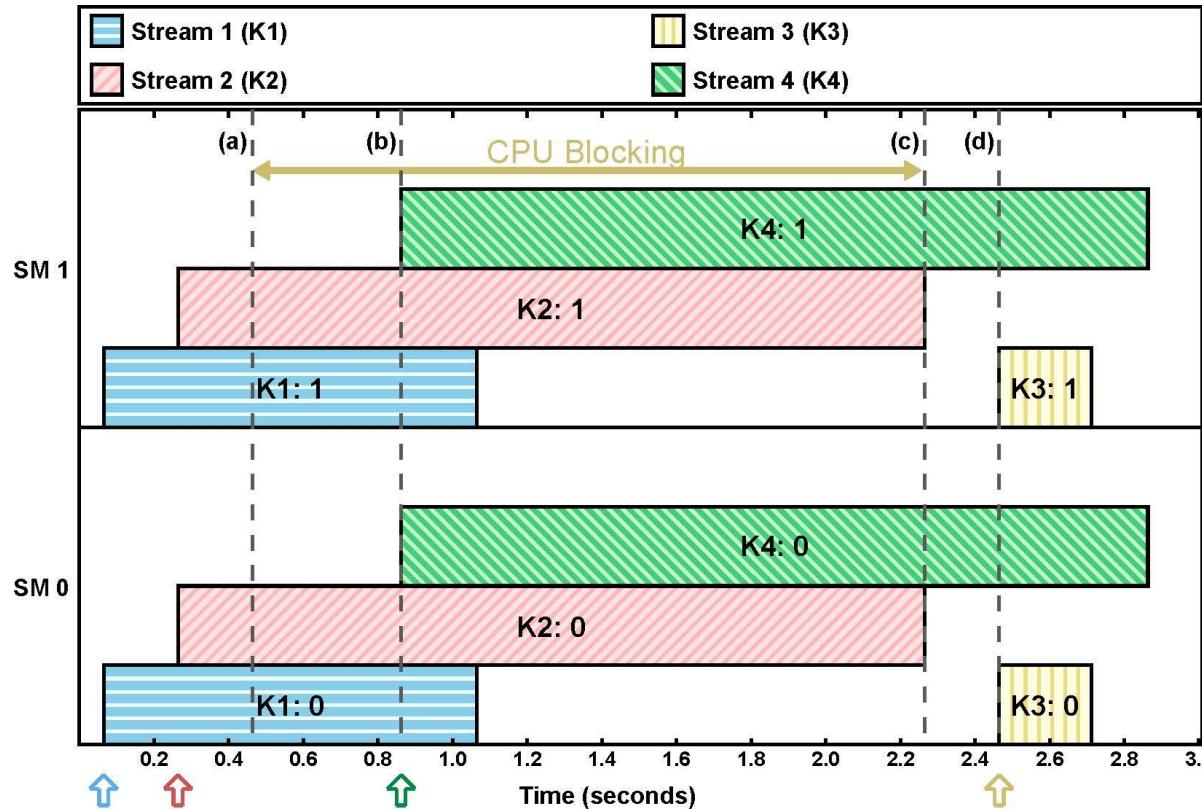
CUDA Programming Fundamentals

(i) Allocate GPU memory	<code>cudaMalloc (&devicePtr, bufferSize);</code>
(ii) Copy data from CPU to GPU	<code>cudaMemcpy (devicePtr, hostPtr, bufferSize);</code>
(iii) Launch the kernel <i>(kernel = code that runs on GPU)</i>	<code>computeResult<<<numBlocks, threadsPerBlock>>>(devicePtr);</code>
(iv) Copy results from GPU to CPU	<code>cudaMemcpy (hostPtr, devicePtr, bufferSize);</code>
(v) Free GPU memory	<code>cudaFree (devicePtr);</code>

Pitfalls for Real-Time GPU Usage

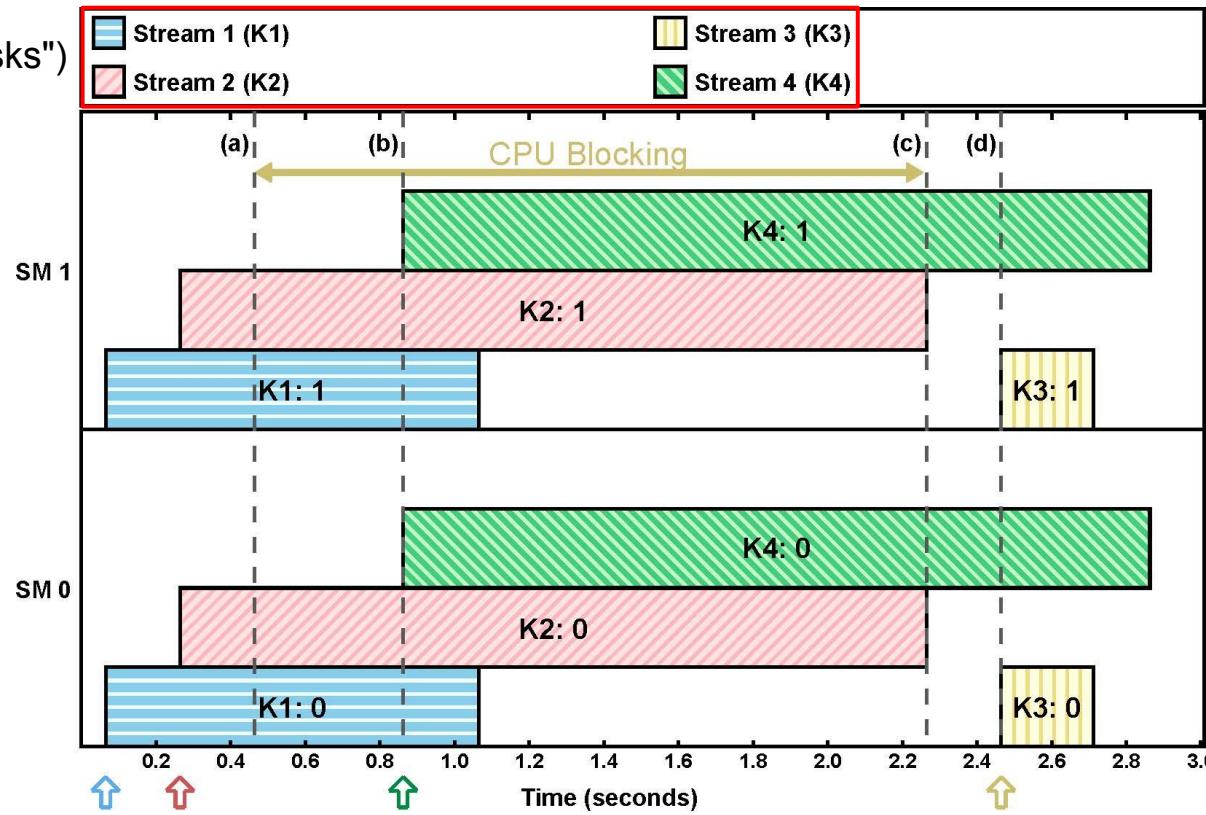
- Synchronization and blocking
- GPU concurrency and performance
- CUDA programming perils

Explicit Synchronization

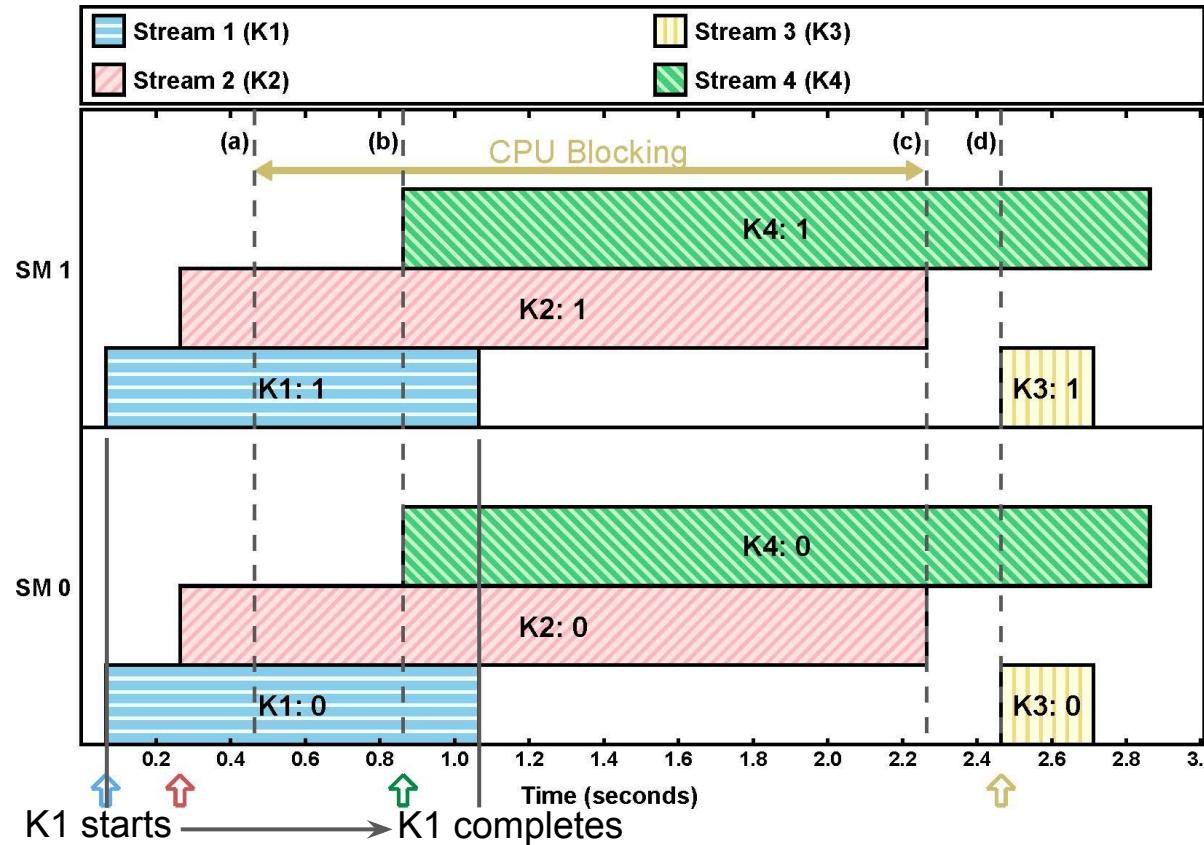


Explicit Synchronization

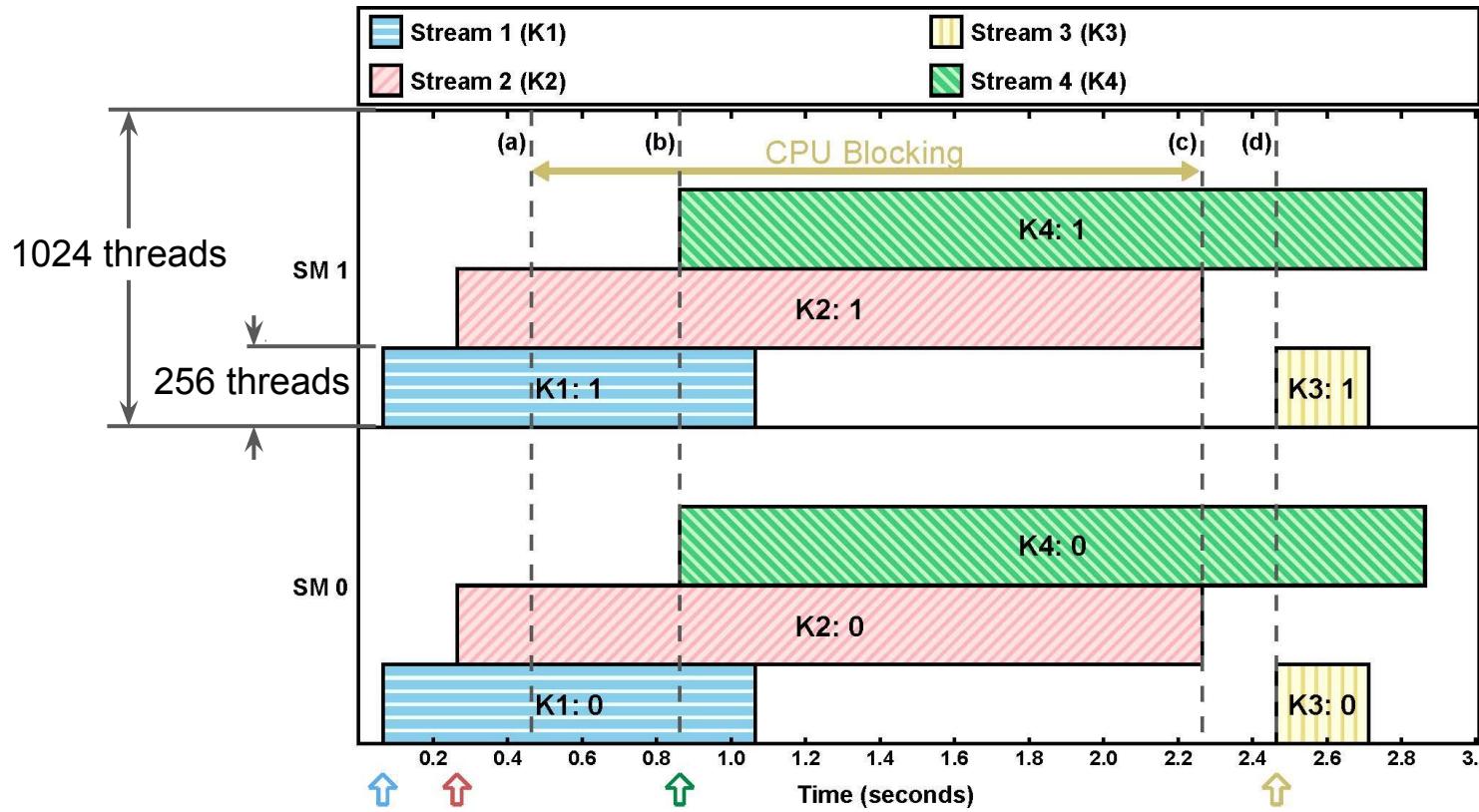
CPU threads ("tasks")



Explicit Synchronization



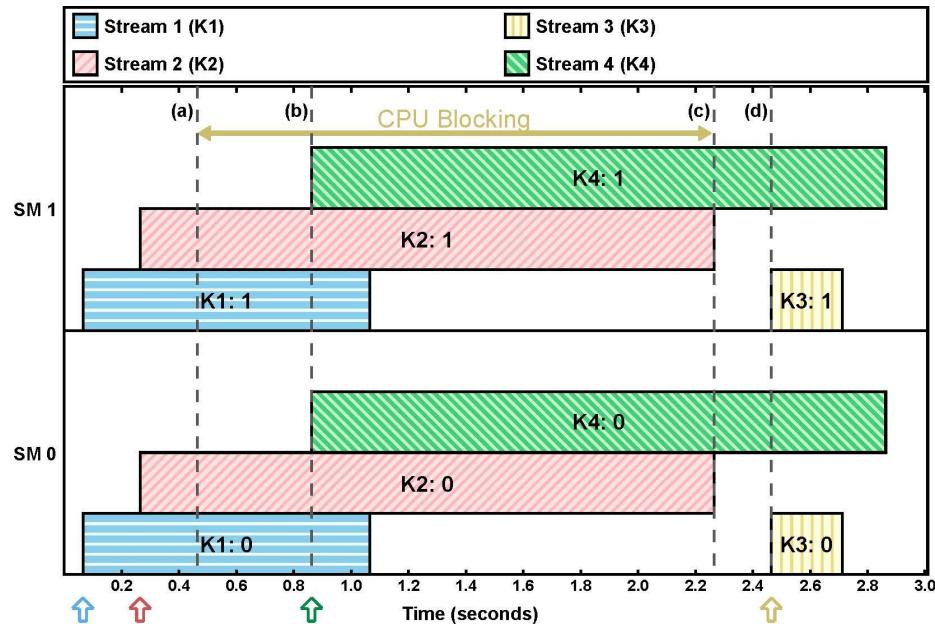
Explicit Synchronization



Explicit Synchronization

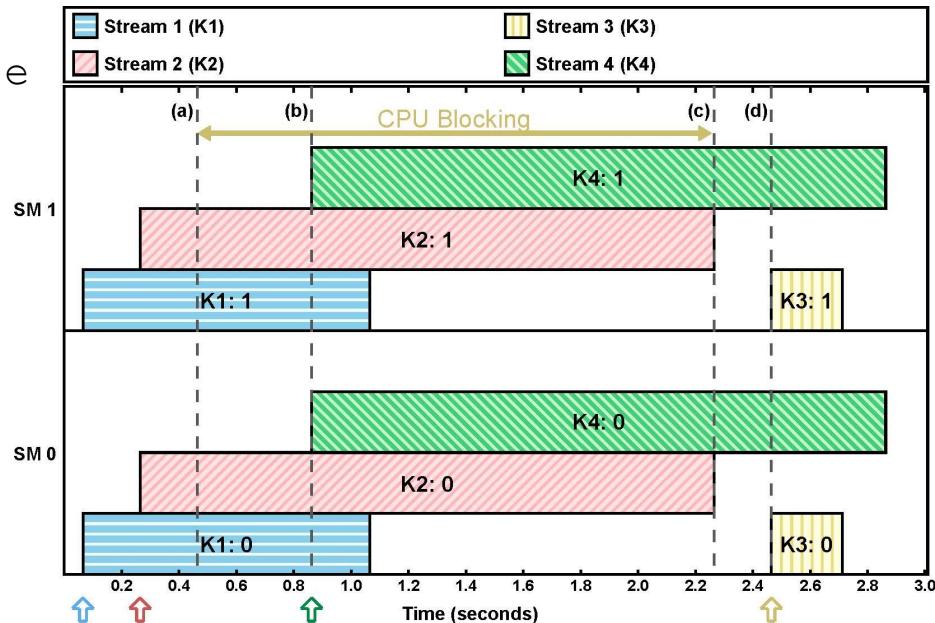
Thread 3

1. Call `cudaDeviceSynchronize` (explicit synchronization).
2. Sleep for 0.2 seconds.
3. Launch kernel K3.



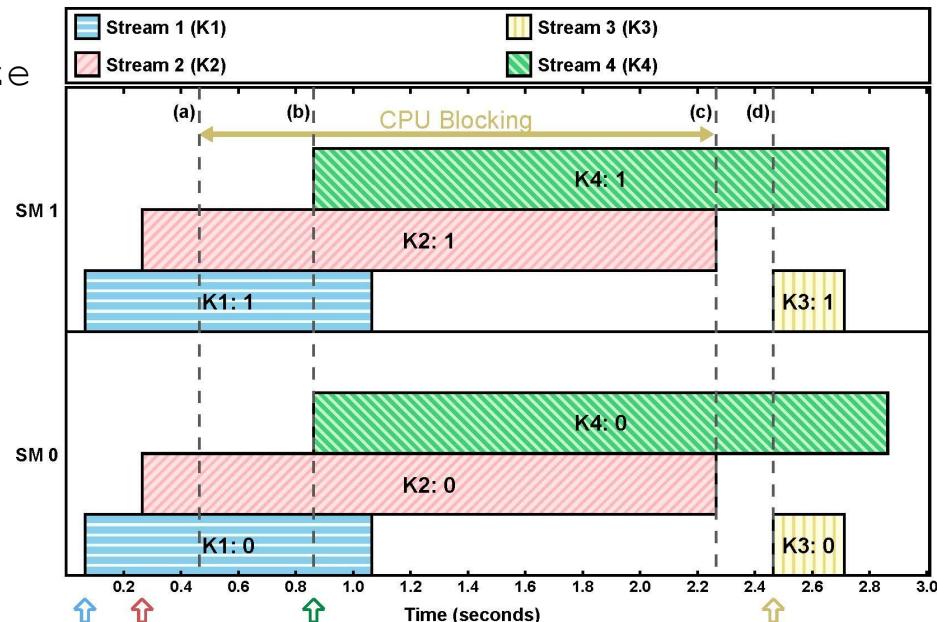
Explicit Synchronization

1. Thread 3 calls `cudaDeviceSynchronize` (explicit synchronization). (a)
2. Thread 3 sleeps for 0.2 seconds. (c)
3. Thread 3 launches kernel K3. (d)



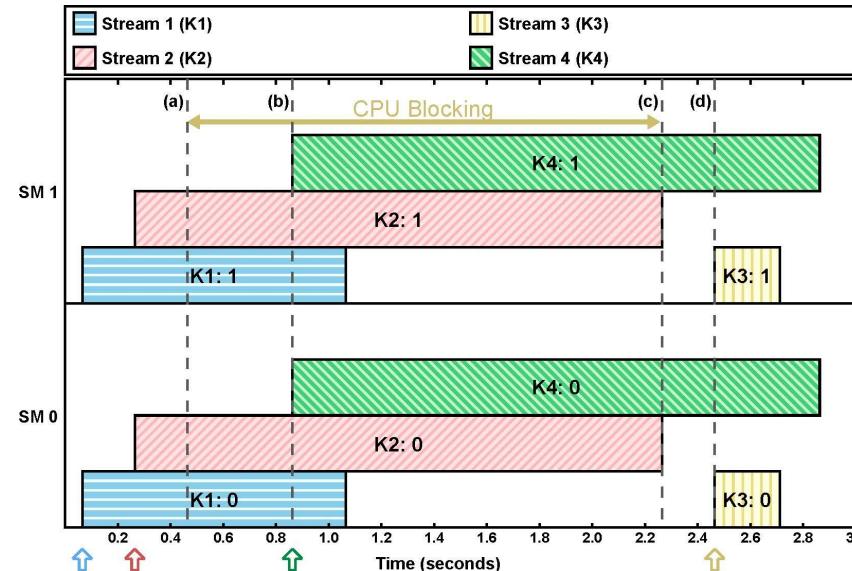
Explicit Synchronization

1. Thread 3 calls `cudaDeviceSynchronize` (explicit synchronization). **(a)**
2. Thread 4 launches kernel K4. **(b)**
3. Thread 3 sleeps for 0.2 seconds. **(c)**
4. Thread 3 launches kernel K3. **(d)**



Explicit Synchronization

Pitfall 1. Explicit synchronization does not block future commands issued by other tasks.



Implicit Synchronization

CUDA toolkit 9.2.88 Programming Guide, Section 3.2.5.5.4, "Implicit Synchronization":
Two commands from different streams cannot run concurrently [if separated by]:

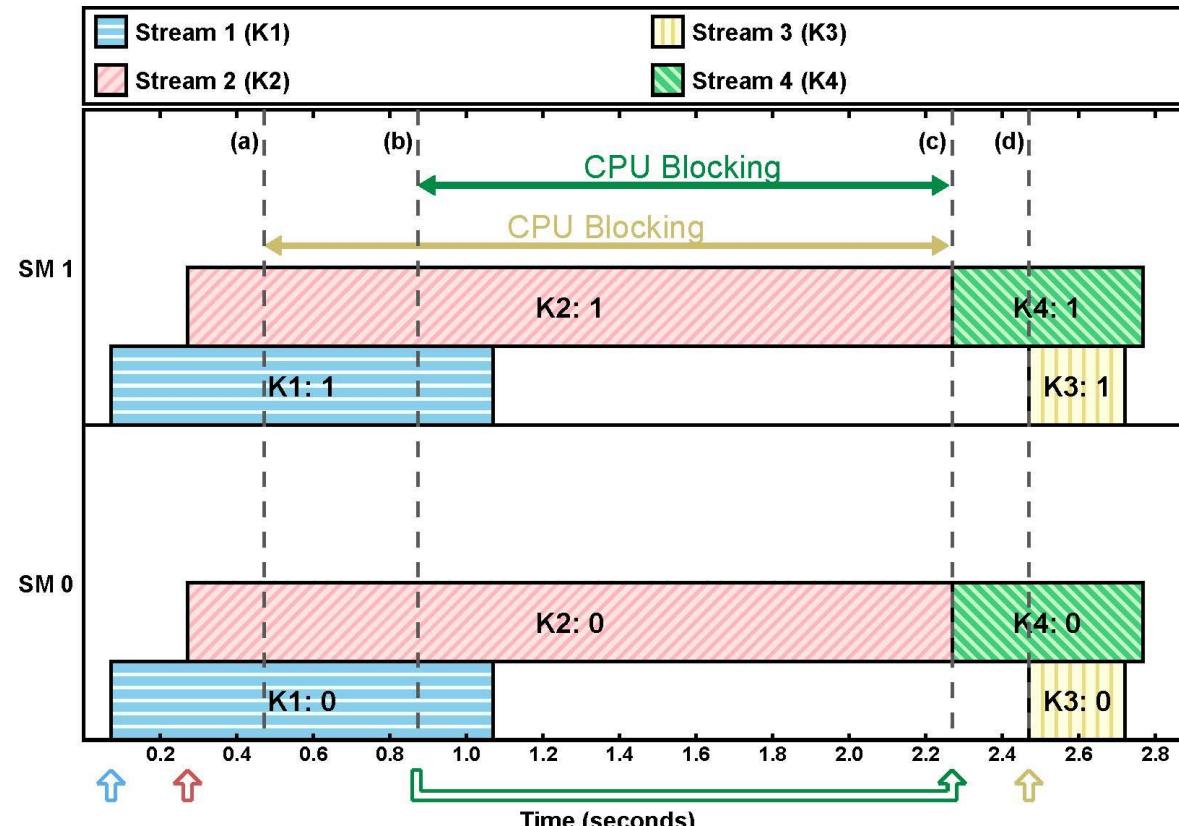
1. A page-locked host memory allocation
2. A device memory allocation
3. A device memory set
4. A memory copy between two addresses to the same device memory
5. Any CUDA command to the NULL stream

Implicit Synchronization

→ **Pitfall 2.** Documented sources of implicit synchronization may not occur.

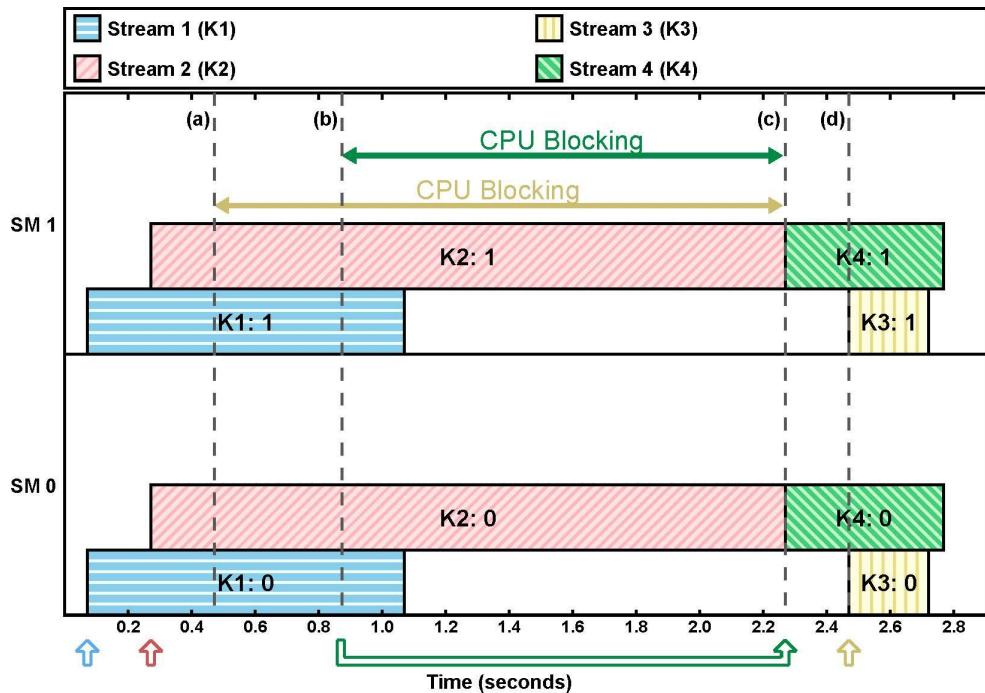
1. A ~~page locked host memory allocation~~
2. A ~~device memory allocation~~
3. A ~~device memory set~~
4. A ~~memory copy between two addresses to the same device memory~~
5. Any CUDA command to the NULL stream

Implicit Synchronization



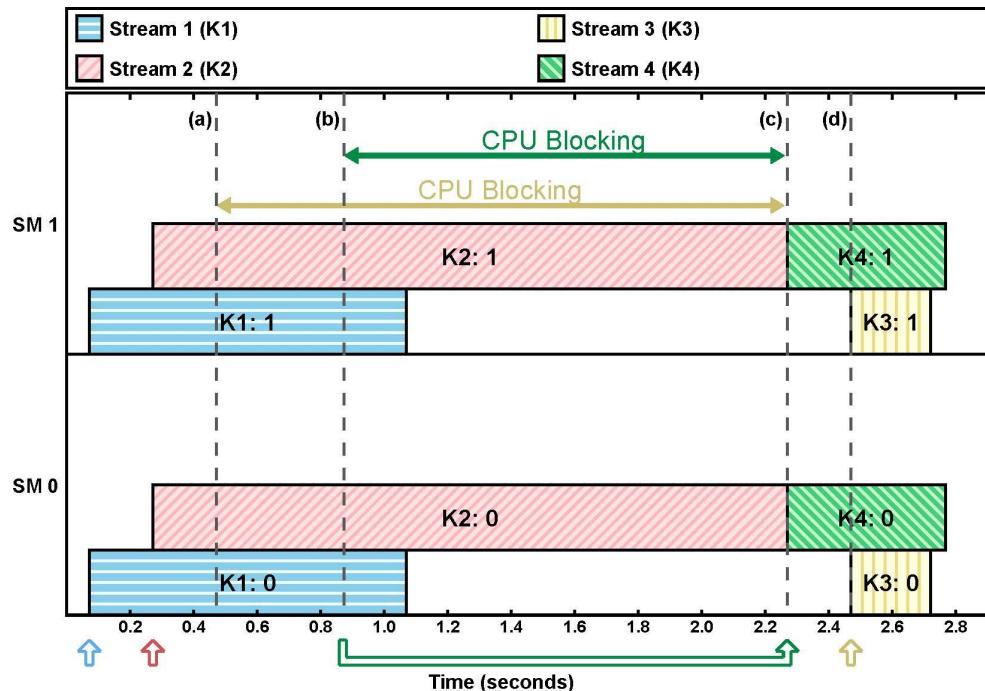
Implicit Synchronization

1. Thread 3 calls `cudaFree`. (a)
2. Thread 3 sleeps for 0.2 seconds. (c)
3. Thread 3 launches kernel K3. (d)



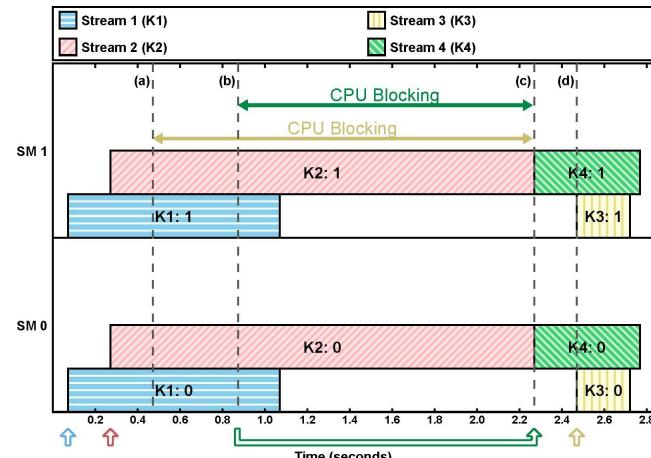
Implicit Synchronization

1. Thread 3 calls `cudaFree`. (a)
2. Thread 4 is blocked *on the CPU* when trying to launch kernel 4. (b)
3. Thread 4 finishes launching kernel K4, thread 3 sleeps for 0.2 seconds. (c)
4. Thread 3 launches kernel K3. (d)



Implicit Synchronization

- **Pitfall 3.** The CUDA documentation neglects to list some functions that cause implicit synchronization.
- **Pitfall 4.** Some CUDA API functions will block future, unrelated, CUDA tasks on the CPU.



Pitfalls for Real-Time GPU Usage

- Synchronization and blocking
 - Suggestion: use CUDA Multi-Process Service (MPS).
- GPU concurrency and performance
- CUDA programming perils

Pitfalls for Real-Time GPU Usage

- Synchronization and blocking
 - Suggestion: use CUDA Multi-Process Service (MPS).
- GPU concurrency and performance

	Multiple Process-based Tasks	Multiple Thread-based Tasks
Without MPS	MP	MT
With MPS	MP(MPS)	MT(MPS)

- CUDA programming perils

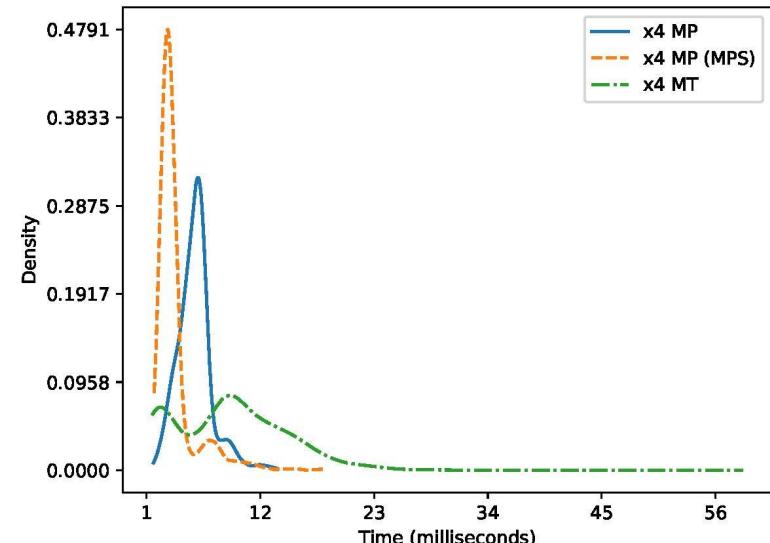
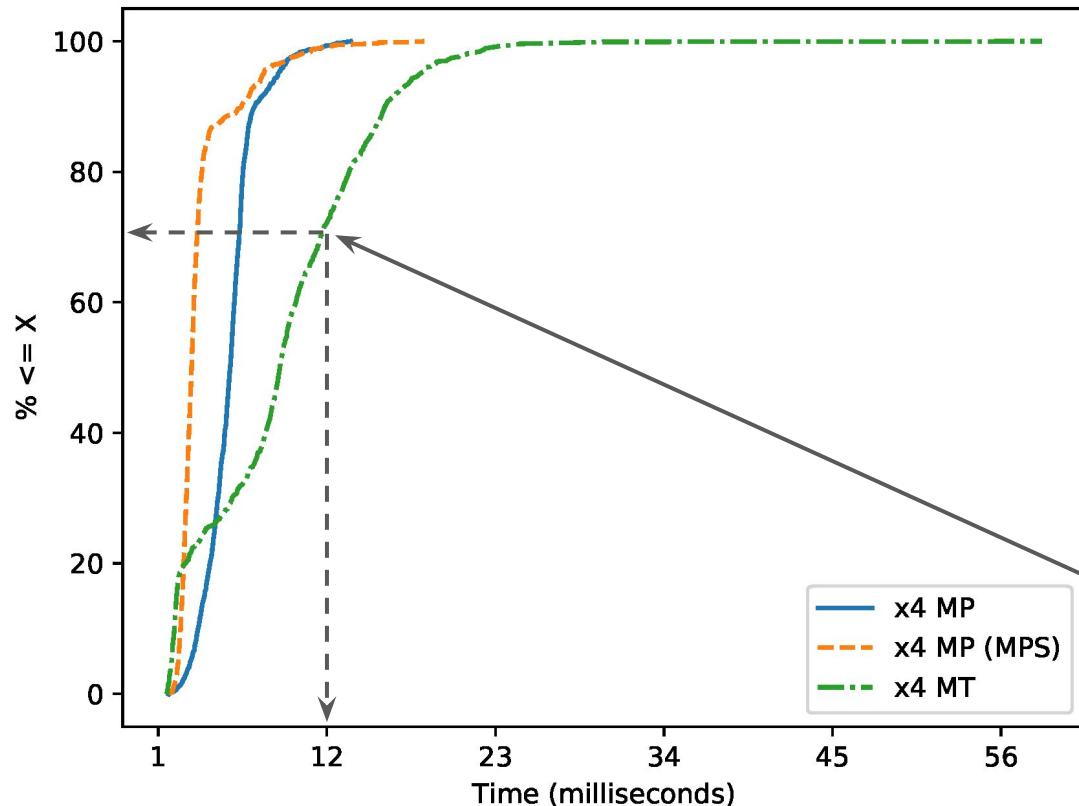
Pitfalls for Real-Time GPU Usage

- Synchronization and blocking
 - Suggestion: use CUDA Multi-Process Service (MPS).
- GPU concurrency and performance

	Multiple Process-based Tasks	Multiple Thread-based Tasks
Without MPS	MP	MT
With MPS	MP(MPS)	MT(MPS)

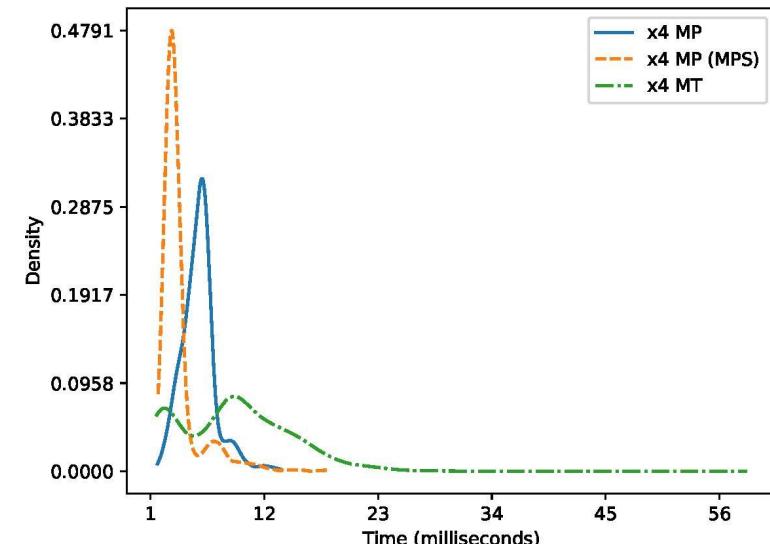
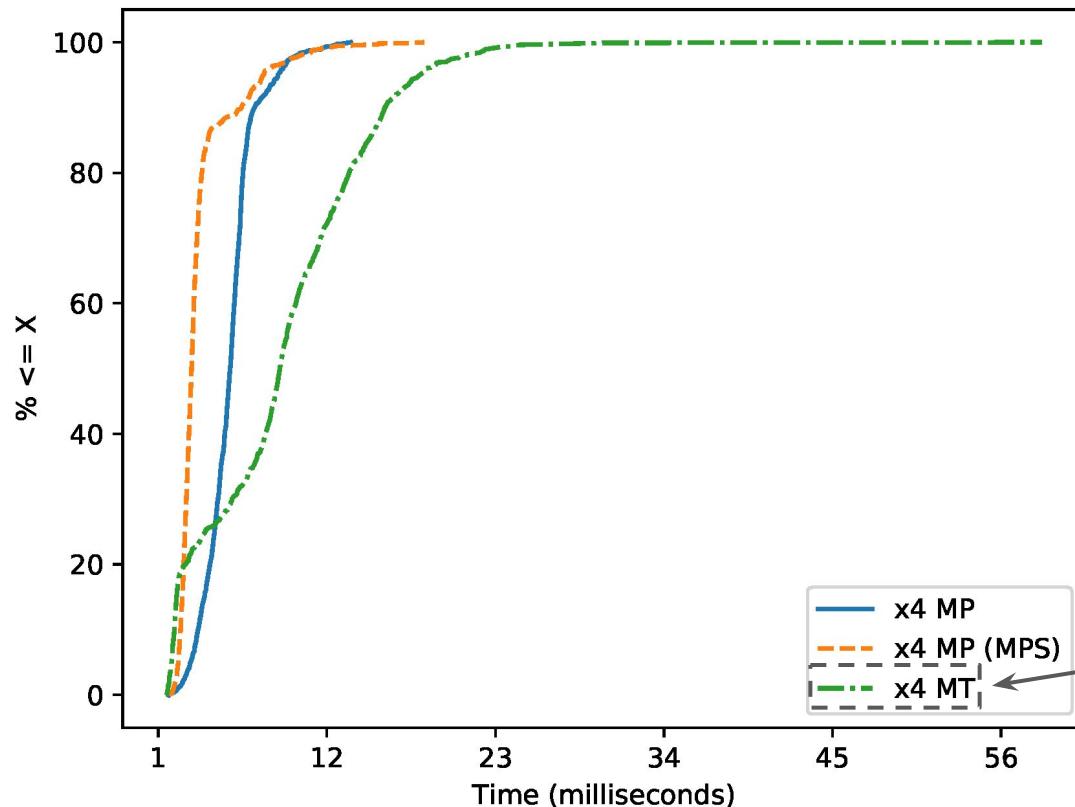
- CUDA programming perils

GPU Concurrency and Performance



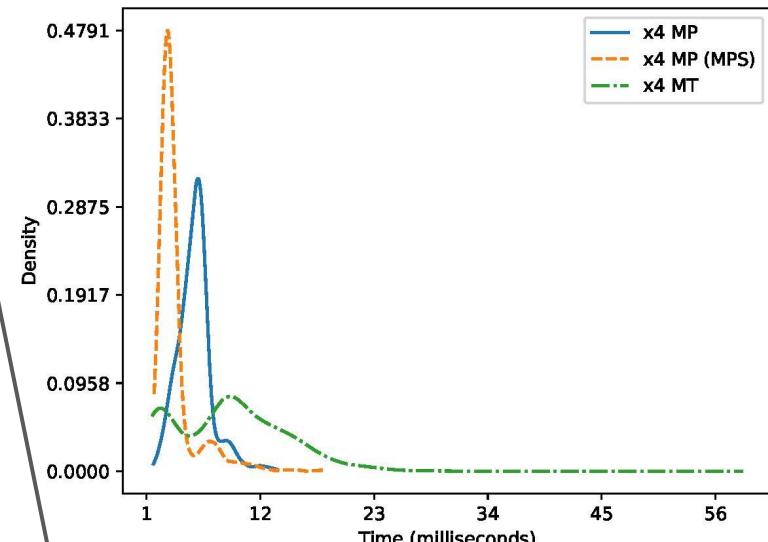
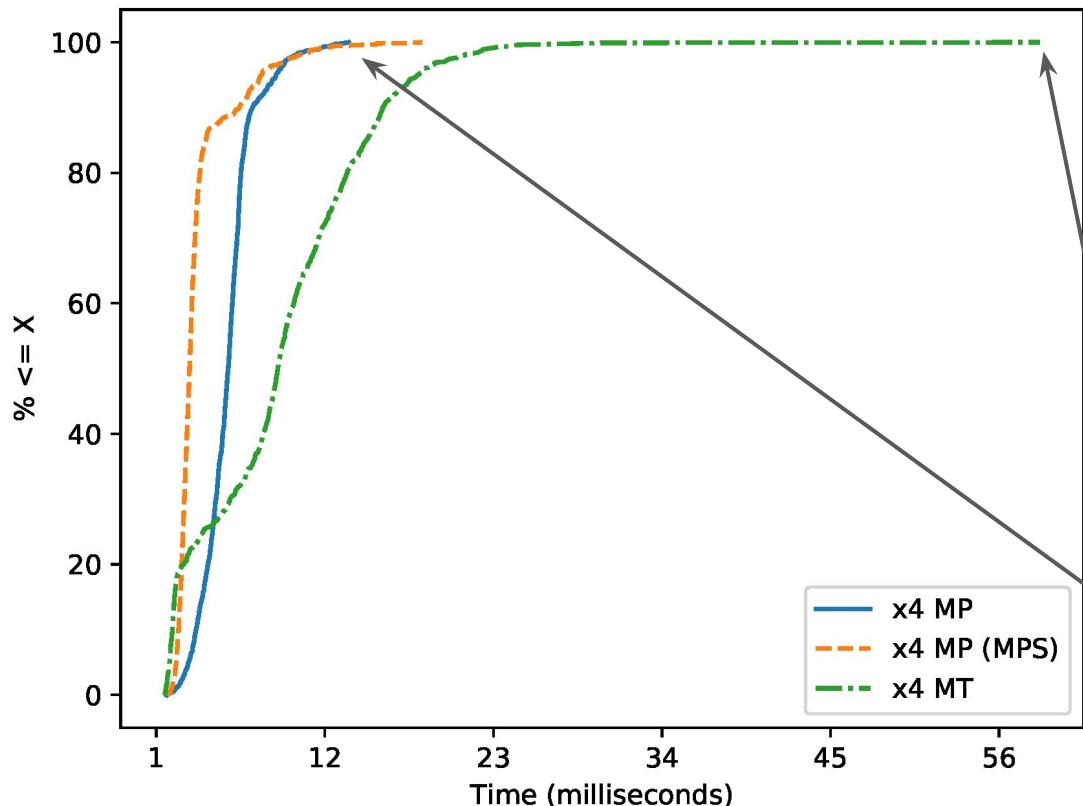
70% of the time, a single Hough transform iteration completed in 12 ms or less.

GPU Concurrency and Performance



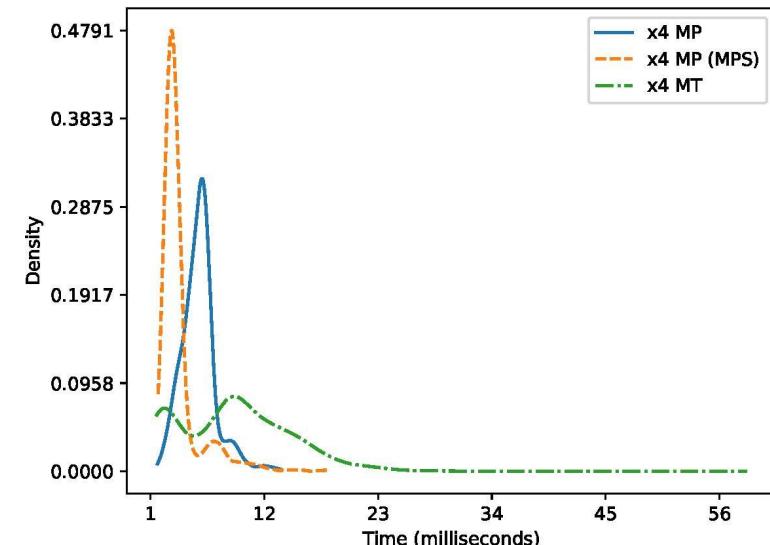
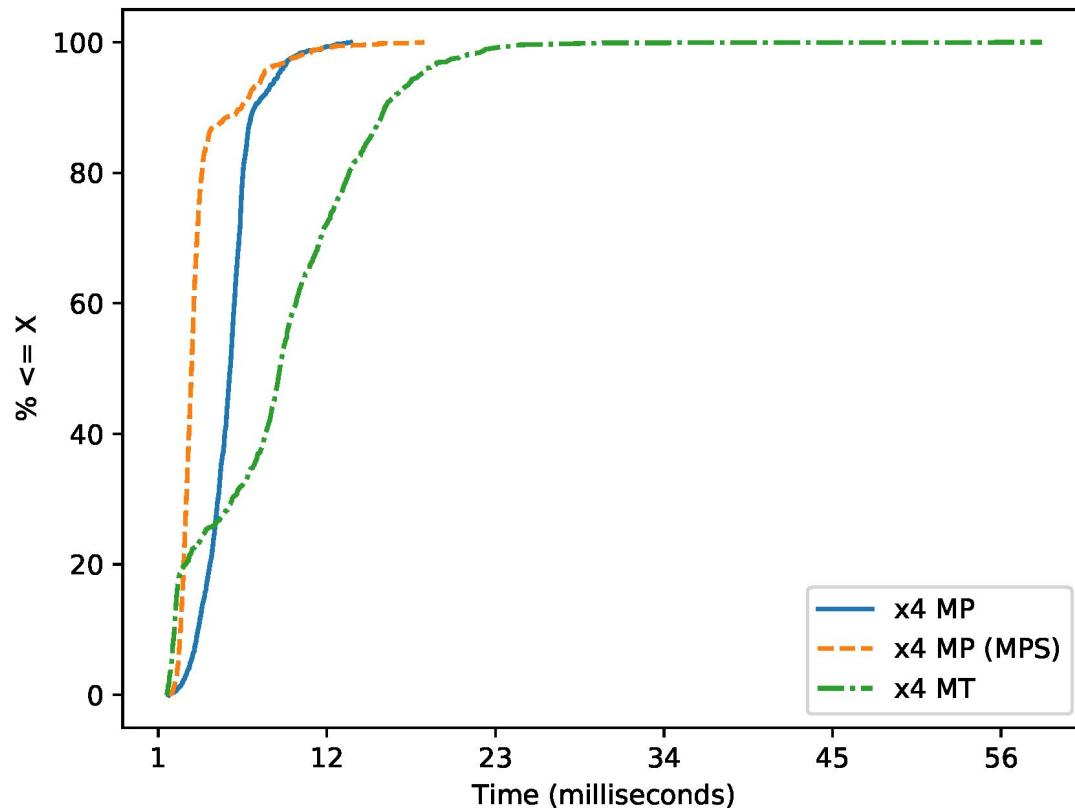
This occurred when four concurrent instances were running in separate CPU threads.

GPU Concurrency and Performance

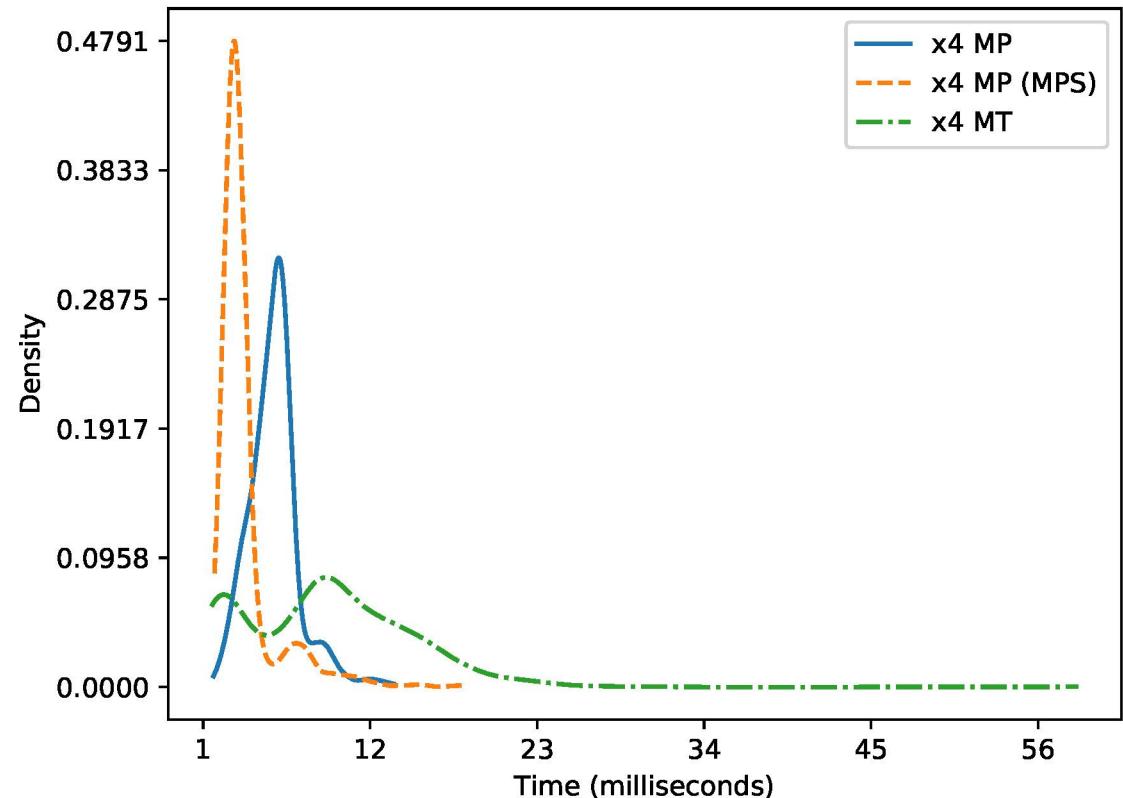
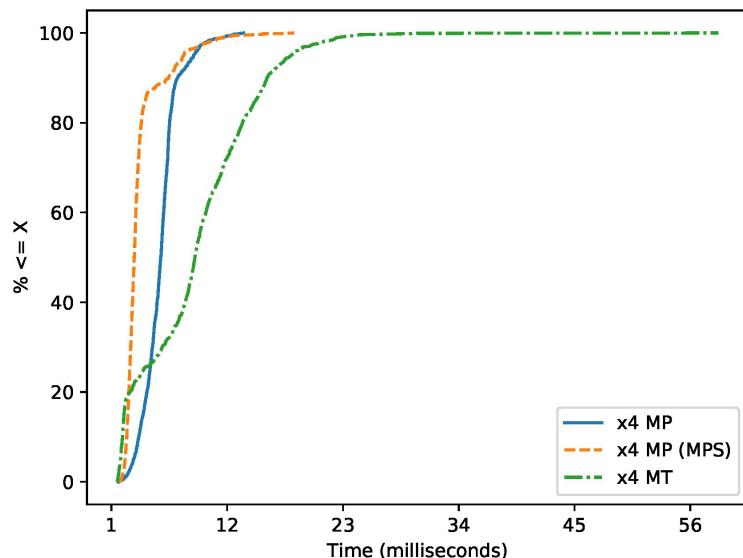


The observed WCET under MT was over 4x the WCET under MP.

GPU Concurrency and Performance

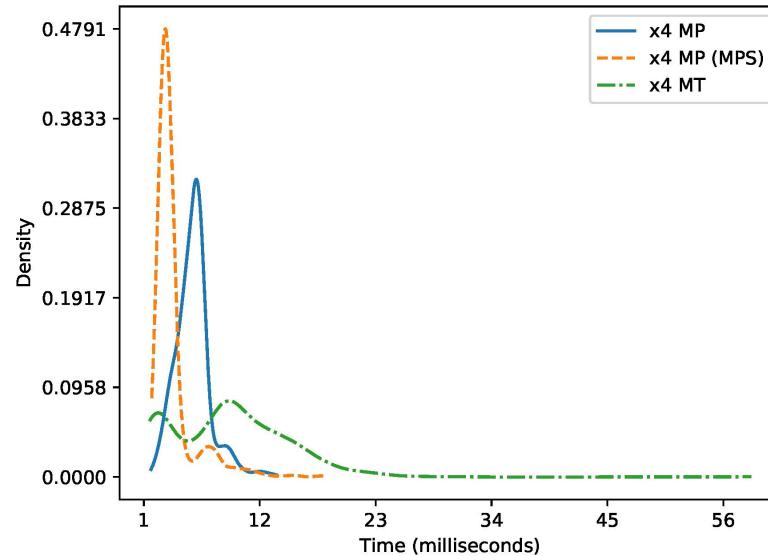
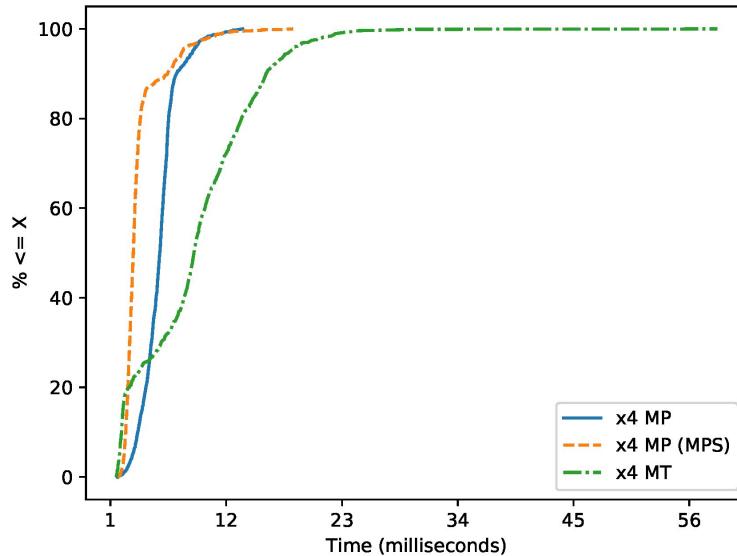


GPU Concurrency and Performance



GPU Concurrency and Performance

- **Pitfall 5.** The suggestion from NVIDIA's documentation to exploit concurrency through user-defined streams may be of limited use for improving performance in thread-based tasks.



Pitfalls for Real-Time GPU Usage

- Synchronization and blocking
 - Suggestion: use CUDA Multi-Process Service (MPS).
- GPU concurrency and performance
- CUDA programming perils

Pitfalls for Real-Time GPU Usage

- Synchronization and blocking

- Suggestion: use CUDA Multi-Process Synchronization.

- GPU concurrency and performance

- CUDA programming perils

Only on X86_64

Pitfalls for Real-Time GPU Usage

- Synchronization and blocking
 - Suggestion: use CUDA Multi-Process Service (MPS).
- GPU concurrency and performance
- CUDA programming perils

Synchronous Defaults

```
if (!CheckCUDAError(
    cudaMemsetAsync(
        state->device_block_smids,
        0, data_size))) {
    return 0;
}
```

Synchronous Defaults

```
if (!CheckCUDAError(  
    cudaMemsetAsync(  
        state->device_block_smids,  
        0, data_size))) {  
    return 0;  
}
```

- What about the CUDA docs saying that memset causes implicit synchronization?

Synchronous Defaults

```
if (!CheckCUDAError(  
    cudaMemsetAsync(  
        state->device_block_smids,  
        0, data_size))) {  
    return 0;  
}
```

- What about the CUDA docs saying that memset causes implicit synchronization?
- Didn't slide 22 say memset *doesn't* cause implicit synchronization?

Synchronous Defaults

```
if (!CheckCUDAError(          if (!CheckCUDAError(  
    cudaMemsetAsync(          cudaMemsetAsync(  
        state->device_block_smids,  
        0, data_size))) {      state->device_block_smids,  
        return 0;              0, data_size,  
    }                          state->stream))) {  
    }                          return 0;  
}
```

- **Pitfall 6.** Async CUDA functions use the GPU-synchronous NULL stream by default.

Other Perils

→ **Pitfall 7.** Observed CUDA behavior often diverges from what the documentation states or implies.

Source	Observed Behavior			Documented Behavior	
	Blocks Other CPU Tasks	Implicit Sync. (Sec. 3.1.2)	Caller Must Wait for GPU	Implicit Sync. (Sec. 3.1.2)	Caller Must Wait for GPU
cudaDeviceSynchronize	No	No	Yes	No	Yes
cudaFree	Yes	Yes	Yes	No (undoc.)	No (impl.)
cudaFreeHost	Yes	Yes	Yes	No (undoc.)	No (impl.)
cudaMalloc	?	No	No	Yes	No (impl.)
cudaMallocHost	?	No	No	Yes	No (impl.)
cudaMemcpyAsync D-D	No	No	No	Yes	No
cudaMemcpyAsync D-H	No	No	No	Yes*	No
cudaMemcpyAsync H-D	No	No	No	Yes*	No
cudaMemset (sync.)	No	Yes	No	Yes	No
cudaMemsetAsync	No	No	No	Yes	No
cudaStreamSynchronize	No	No	Yes	No	Yes

Other Perils

→ **Pitfall 8.** CUDA documentation can be contradictory.

Source	Observed Behavior			Documented Behavior	
	Blocks Other CPU Tasks	Implicit Sync. (Sec. 3.1.2)	Caller Must Wait for GPU	Implicit Sync. (Sec. 3.1.2)	Caller Must Wait for GPU
cudaDeviceSynchronize	No	No	Yes	No	Yes
cudaFree	Yes	Yes	Yes	No (undoc.)	No (impl.)
cudaFreeHost	Yes	Yes	Yes	No (undoc.)	No (impl.)
cudaMalloc	?	No	No	Yes	No (impl.)
cudaMallocHost	?	No	No	Yes	No (impl.)
cudaMemcpyAsync D-D	No	No	No	Yes	No
cudaMemcpyAsync D-H	No	No	No	Yes*	No
cudaMemcpyAsync H-D	No	No	No	Yes*	No
cudaMemset (sync.)	No	Yes	No	Yes	No
cudaMemsetAsync	No	No	No	Yes	No
cudaStreamSynchronize	No	No	Yes	No	Yes

Other Perils

- **Pitfall 8.** CUDA documentation can be contradictory.

CUDA Programming Guide, section 3.2.5.1:

*The following device operations are asynchronous with respect to the host:
[...] Memory copies performed by functions that are suffixed with Async*

CUDA Runtime API Documentation, section 2:

*For transfers from device memory to pageable host memory,
[cudaMemcpyAsync] will return only once the copy has completed.*

Other Perils

- **Pitfall 9.** What we learn about current black-box GPUs may not apply in the future.

Conclusion

- The GPU ecosystem needs clarity and openness!
- Avoid pitfalls when using NVIDIA GPUs for real-time tasks in autonomous systems
 - GPU synchronization, application performance, and problems with documentation

Thanks! Questions?

Figure sources:

<https://electrek.co/guides/tesla-vision/>

<https://www.quora.com/What-are-the-different-types-of-artificial-neural-network>

https://www.researchgate.net/figure/Compute-unified-device-architecture-CUDA-threads-and-blocks-multidimensional_fiq1_320806445?sq=ziaY-qBKKiKX4pljRq4vJSWZvDvdOidZ2aCRYnD1QVFBDxIx3MEO1I03cl31e1It6pUr53qaS1L1w4Bt5fd8w