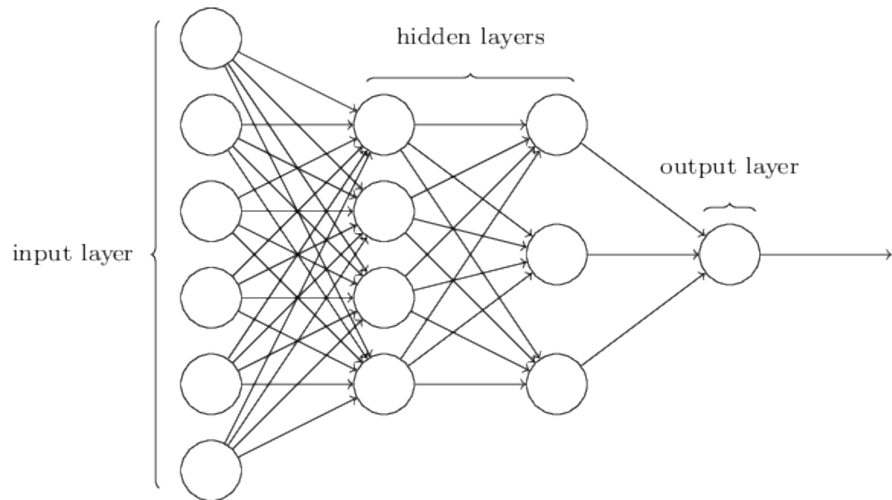
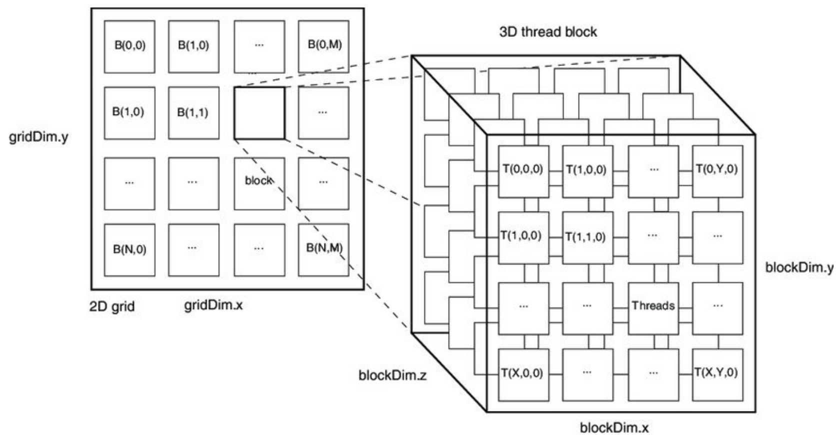
THE UNIVERSITY
of NORTH CAROLINA
at CHAPEL HILL

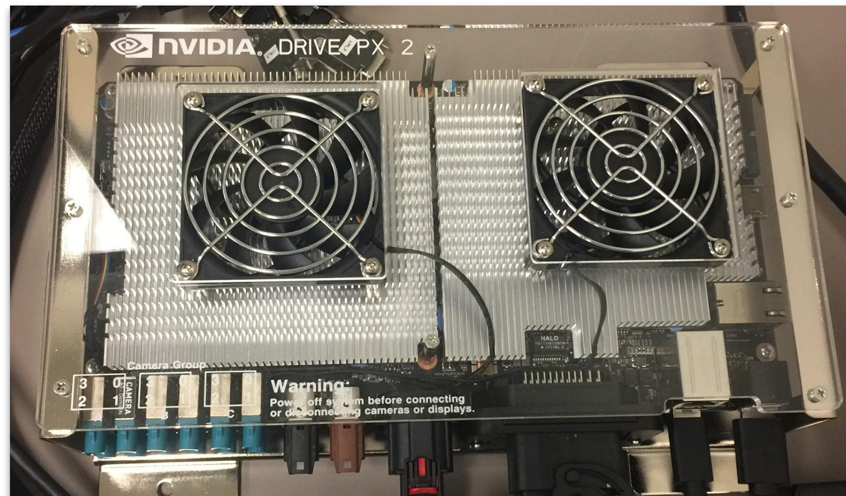
Avoiding Pitfalls when Using NVIDIA GPUs for Real-Time Tasks in Autonomous Systems

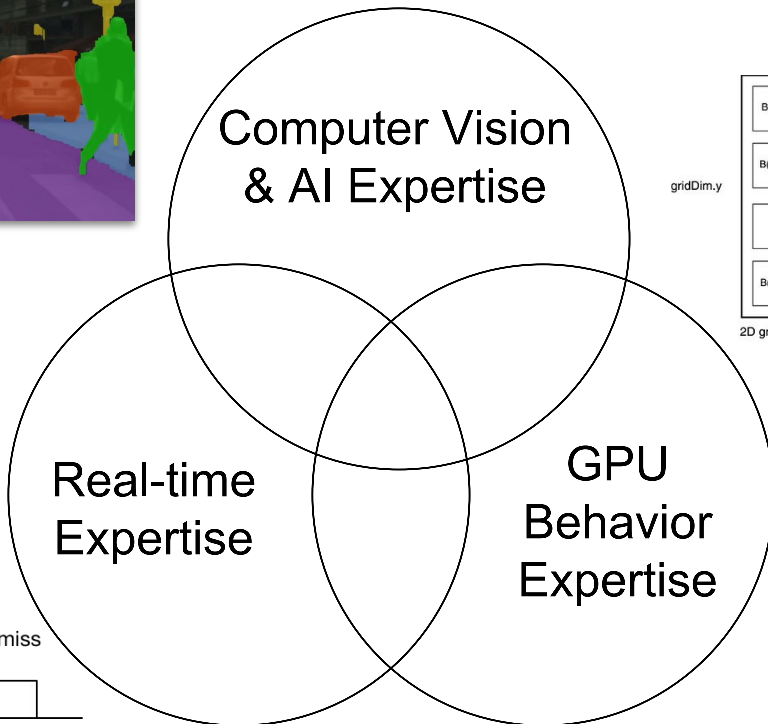
Ming Yang, **Nathan Otterness**, Tanya Amert, Joshua Bakita, James H. Anderson, F. Donelson Smith

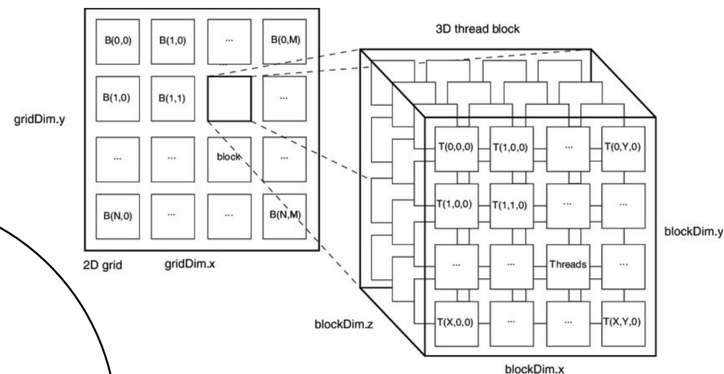
All image sources and references are provided at the end.

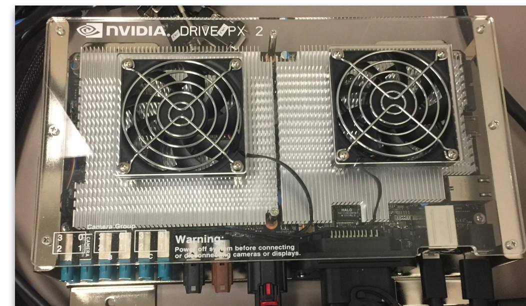

hidden layers
output layer
input layer

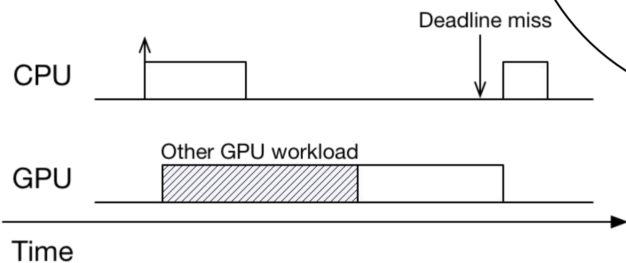

3D thread block
B(0,0)
B(1,0)
B(0,M)
B(1,0)
B(1,1)
block
B(N,0)
B(N,M)
gridDim.y
2D grid
gridDim.x
T(0,0,0)
T(1,0,0)
T(0,Y,0)
T(1,0,0)
T(1,1,0)
Threads
T(X,0,0)
T(X,Y,0)
blockDim.y
blockDim.z
blockDim.x


NVIDIA. DRIVE PX 2
Warning:


Computer Vision
& AI Expertise
Real-time
Expertise
GPU
Behavior
Expertise


B(0,0)
B(1,0)
...
B(0,M)
B(1,0)
B(1,1)
block
B(N,0)
B(N,M)
gridDim.y
2D grid
gridDim.x
3D thread block
T(0,0,0)
T(1,0,0)
T(0,Y,0)
T(1,0,0)
T(1,1,0)
Threads
T(X,0,0)
T(X,Y,0)
blockDim.y
blockDim.z
blockDim.x


NVIDIA. DRIVE PX 2


Deadline miss
CPU
Other GPU workload
GPU
Time

Pitfalls for Real-Time GPU Usage

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

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 (*kernel* = 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 (*kernel* = 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 (*kernel* = 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 (*kernel* = 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 (*kernel* = 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);

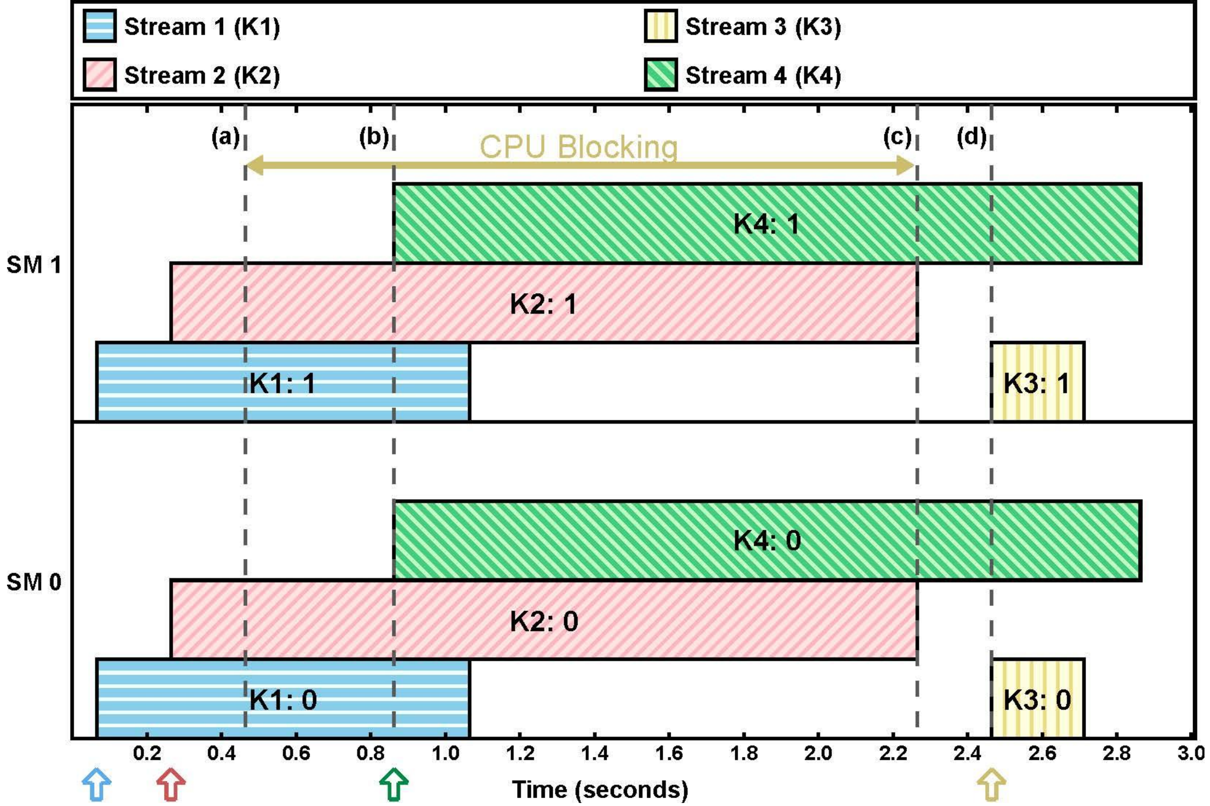
CUDA Programming Fundamentals

(i) Allocate GPU memory	`cudaMalloc(&devicePtr, bufferSize);`
(ii) Copy data from CPU to GPU	`cudaMemcpy(devicePtr, hostPtr, bufferSize);`
(iii) Launch the kernel (*kernel* = 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);`

Pitfalls for Real-Time GPU Usage

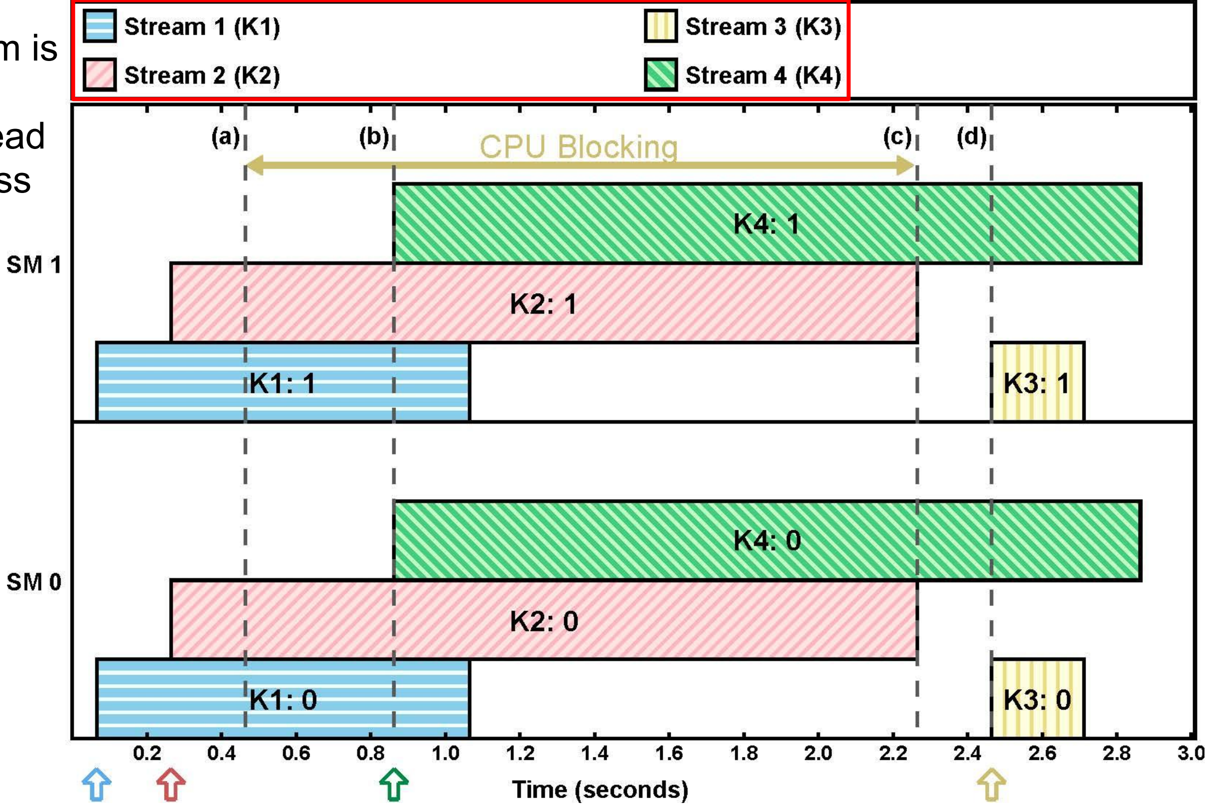
- Synchronization and blocking
- GPU concurrency and performance
- CUDA programming perils

Explicit Synchronization

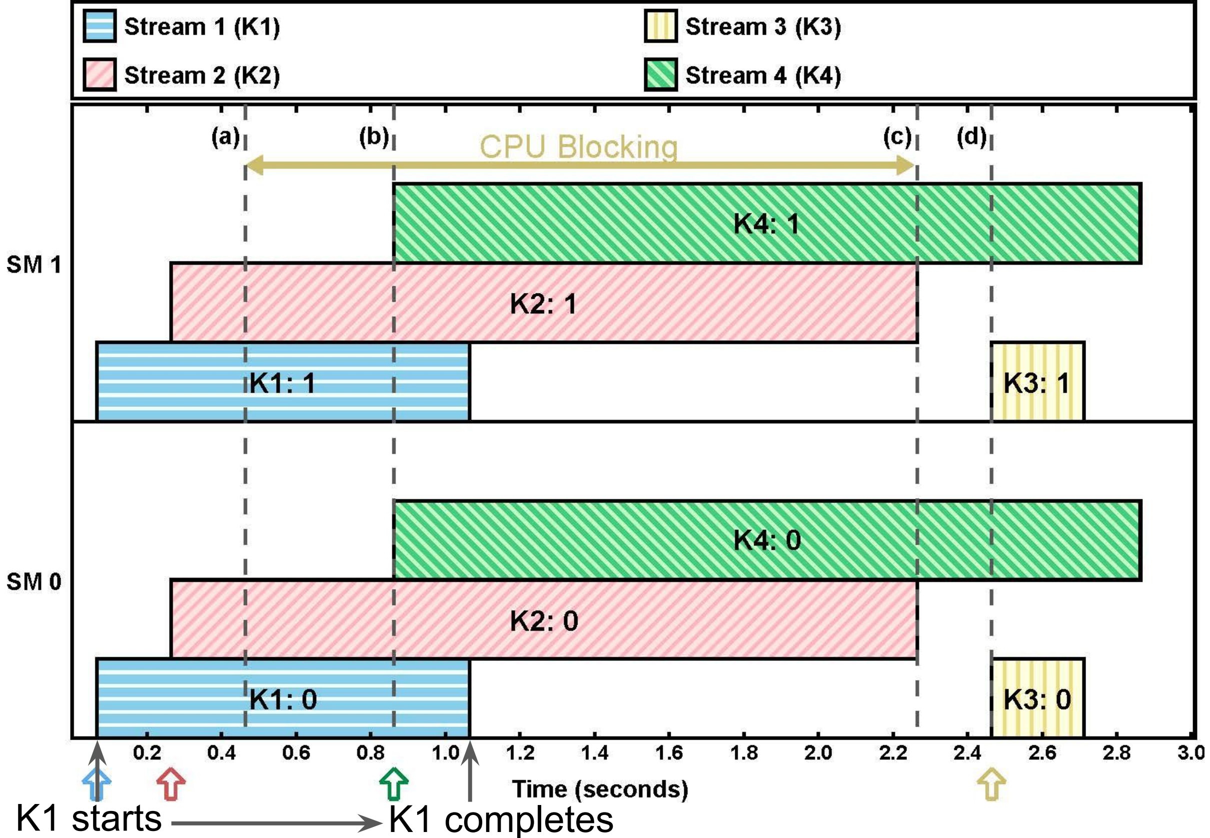


Explicit Synchronization

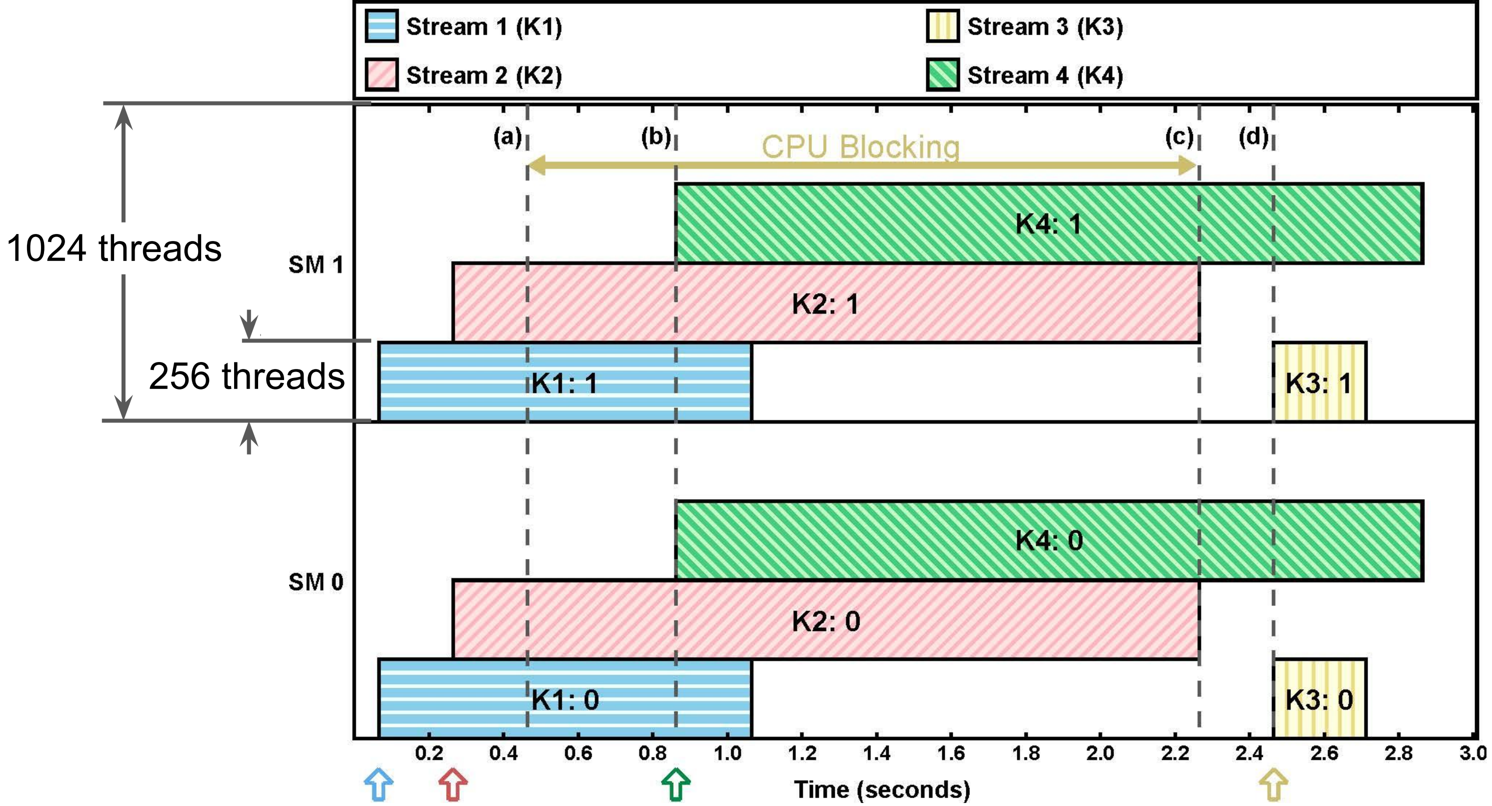
Each CUDA stream is managed by a separate CPU thread in the same address space.



Explicit Synchronization



Explicit Synchronization

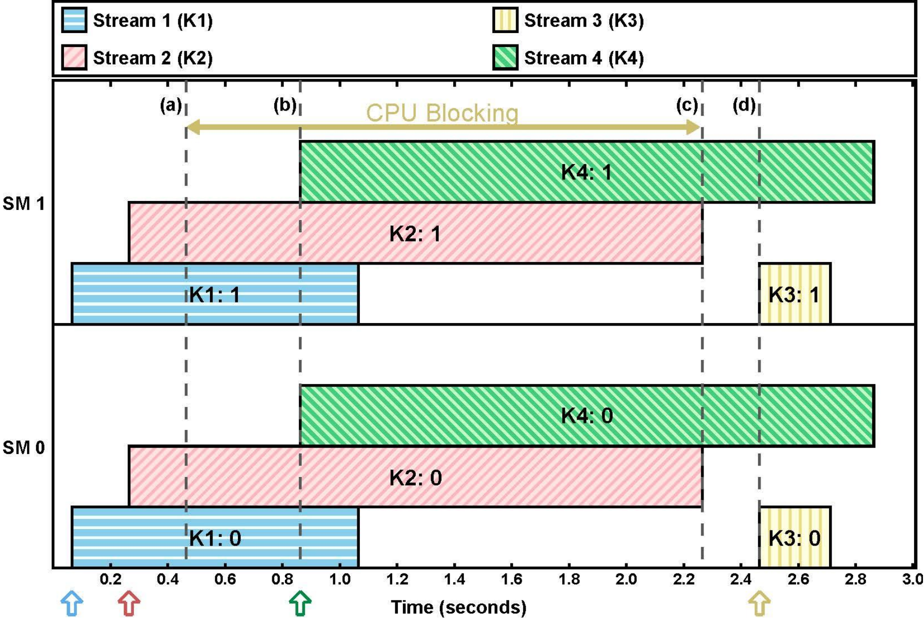


Explicit Synchronization

1. Thread 3 calls `cudaDeviceSynchronize` (explicit synchronization). **(a)**

2. Thread 3 sleeps for 0.2 seconds. **(c)**

3. Thread 3 launches kernel K3. **(d)**



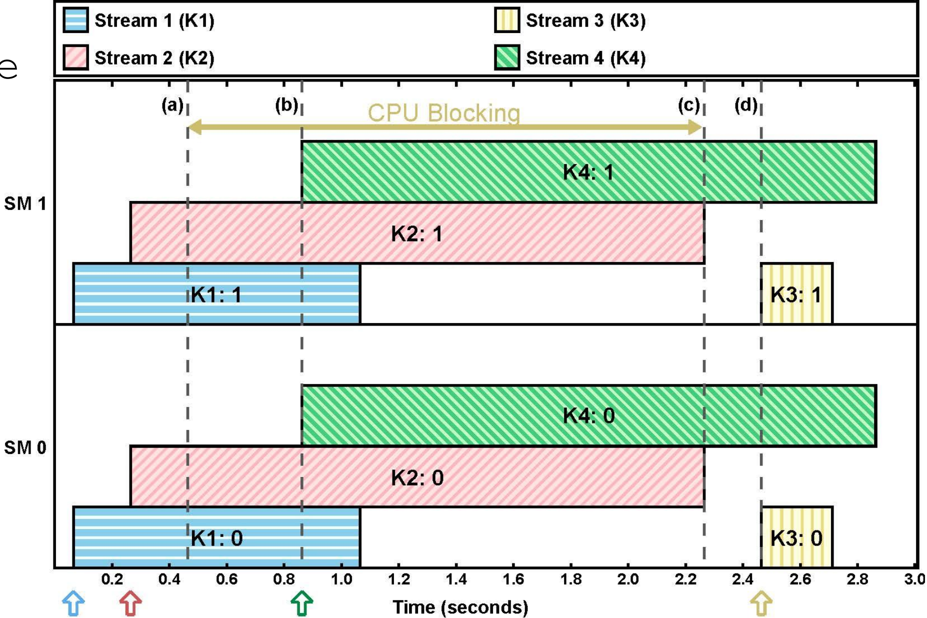
Explicit Synchronization

1. Thread 3 calls `cudaDeviceSynchronize` (explicit synchronization). **(a)**

2. Thread 4 launches kernel K4. **(b)**

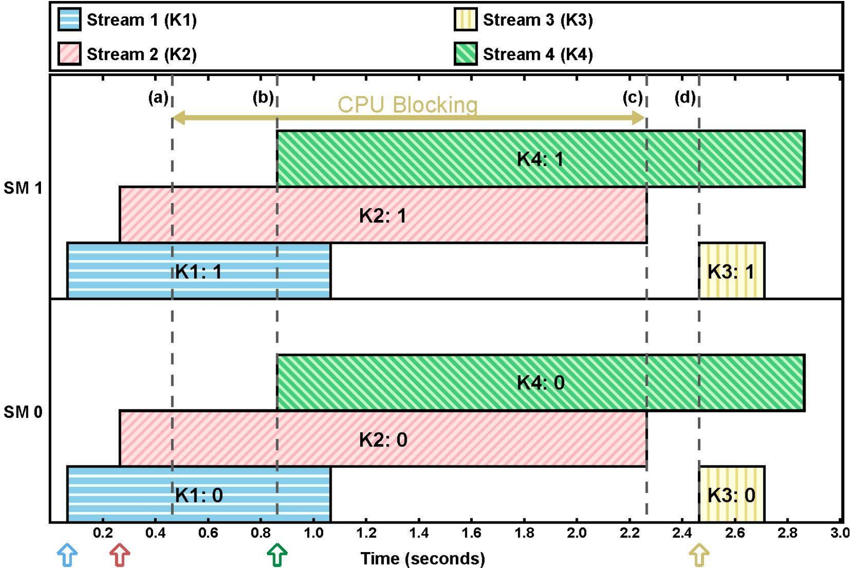
3. Thread 3 sleeps for 0.2 seconds. **(c)**

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]:

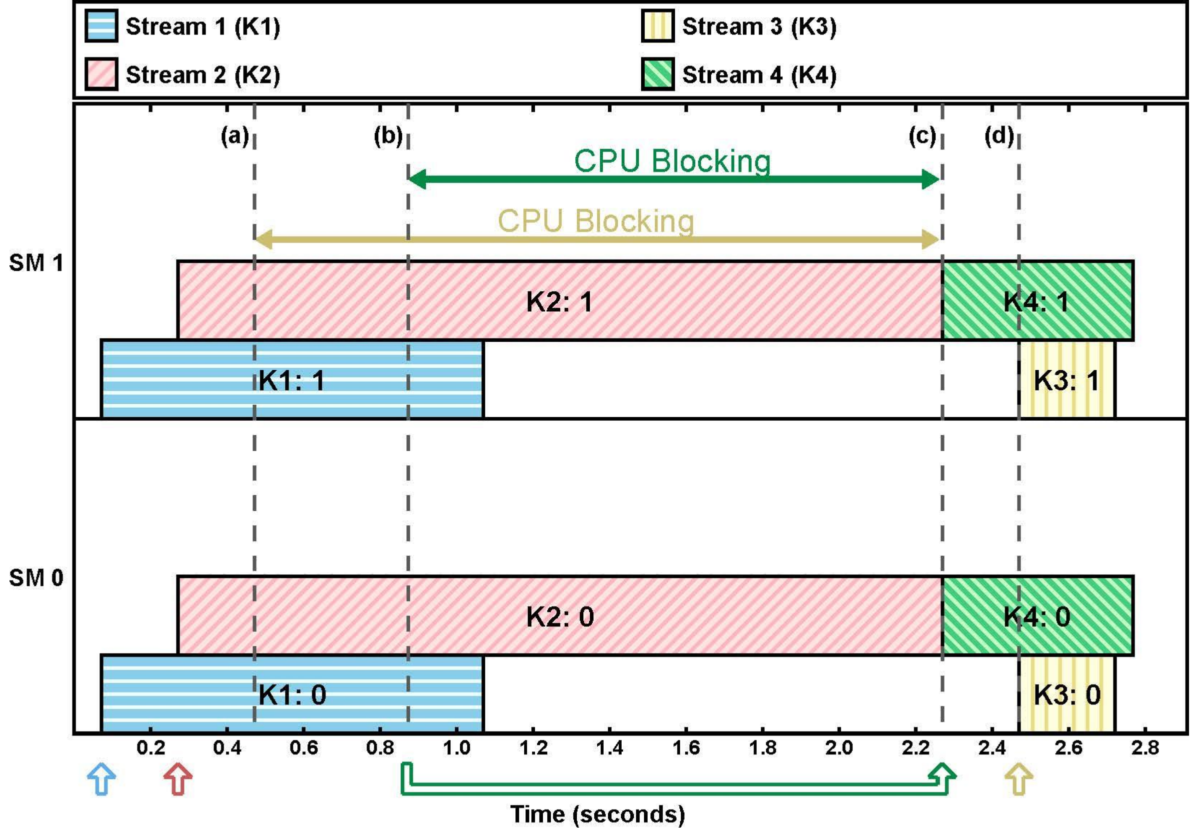
> 1. A page-locked host memory allocation
> 2. A device memory allocation
> 3. A device memory set
> 4. A memory copy between two addresses to the same device memory
> 5. Any CUDA command to the NULL stream

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

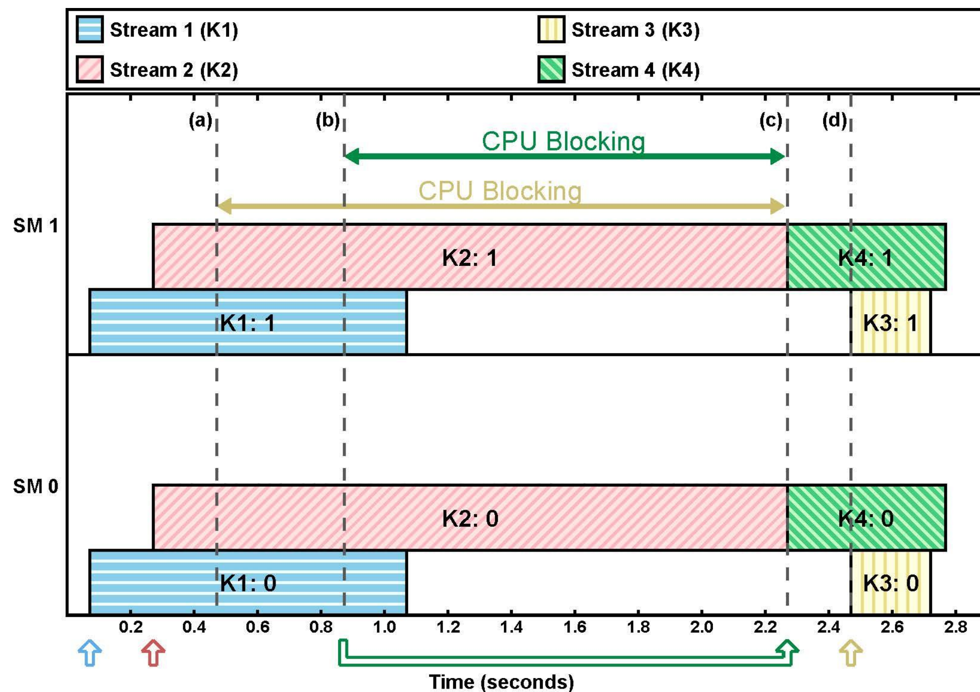


Implicit Synchronization

1. Thread 3 calls `cudaFree`. **(a)**

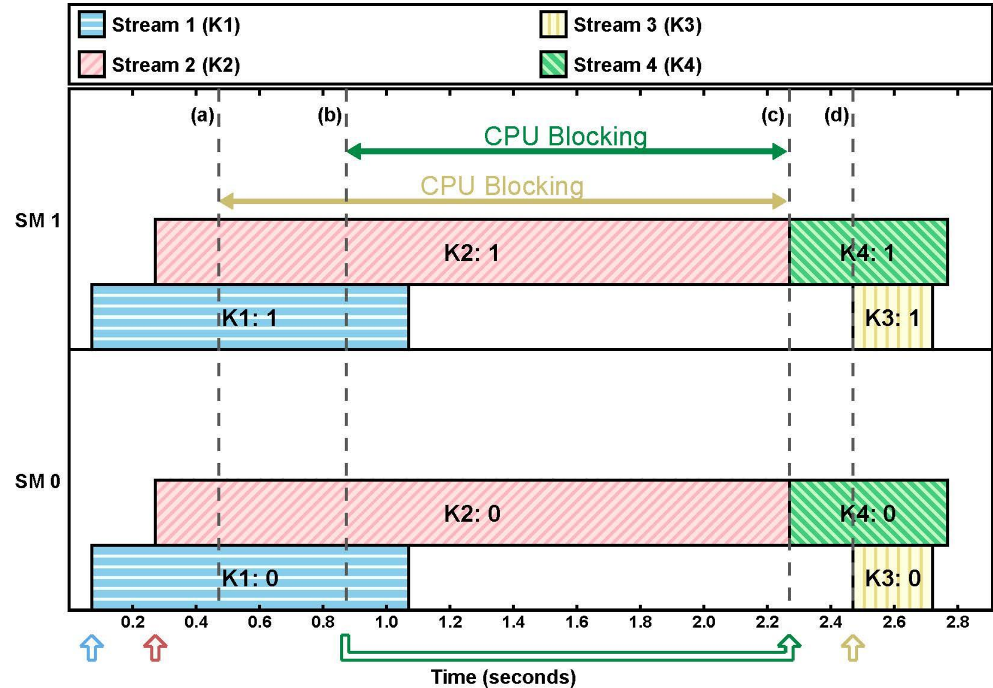
2. Thread 3 sleeps for 0.2 seconds. **(c)**

3. Thread 3 launches kernel K3. **(d)**



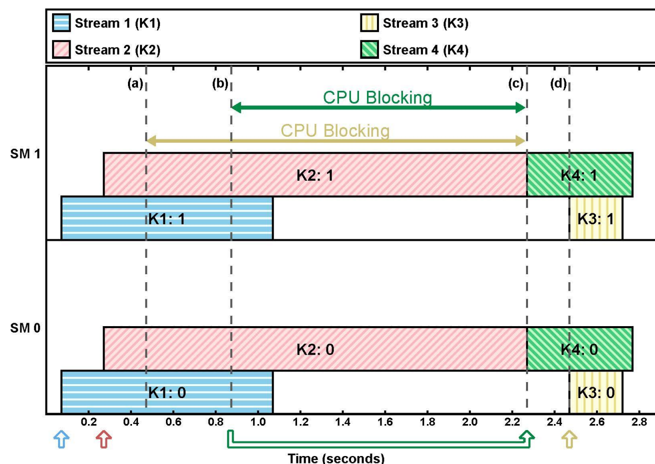
Implicit Synchronization

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 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
- CUDA programming perils

GPU Concurrency and Performance

- Implicit synchronization penalty = Processes with MPS vs. Threads

GPU Concurrency and Performance

- Implicit synchronization penalty = Processes with MPS vs. Threads

- GPU concurrency benefit = Processes with MPS vs. Processes without MPS

GPU Concurrency and Performance

- Implicit synchronization penalty = Processes with MPS vs. Threads
- GPU concurrency benefit = Processes with MPS vs. Processes without MPS
- MPS overhead = Threads vs. Threads with MPS (not in plots)

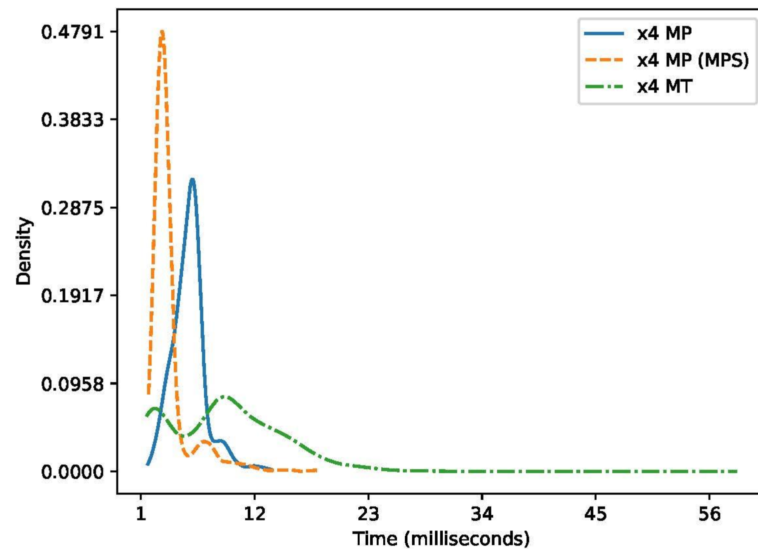
GPU Concurrency and Performance

VisionWorks Samples	Scenarios	Max	$99^{th}\%$	$90^{th}\%$	Mean	Median
Video Stabilization	MP	17.55	12.88	5.43	3.31	2.69
	MP (MPS)	36.73	**11.12**	**5.37**	**2.81**	**2.06**
	MT	**17.0**	13.87	8.94	4.72	3.63
Feature Tracking	MP	**5.64**	**3.87**	**1.45**	**1.08**	**0.96**
	MP (MPS)	14.73	6.04	1.51	1.31	1.09
	MT	31.11	20.86	11.51	4.68	2.68
Motion Estimation	MP	**28.64**	**21.25**	17.33	16.75	17.24
	MP (MPS)	33.05	22.66	**15.75**	**14.3**	**14.89**
	MT	42.86	26.12	16.53	15.07	15.14
Hough Transform	MP	**13.56**	**11.61**	7.28	5.68	5.7
	MP (MPS)	18.35	11.66	**6.44**	**3.74**	**3.18**
	MT	58.65	22.64	15.82	9.12	8.94
Stereo Matching	MP	75.13	50.54	30.42	24.14	24.77
	MP (MPS)	**59.73**	**45.05**	**26.87**	22.59	24.41
	MT	125.96	58.82	34.36	**20.75**	**18.95**

GPU Concurrency and Performance

VisionWorks Samples	Scenarios	Max	$99^{th}\%$	$90^{th}\%$	Mean	Median
Video Stabilization	MP	17.55	12.88	5.43	3.31	2.69
	MP (MPS)	36.73	**11.12**	**5.37**	**2.81**	**2.06**
	MT	**17.0**	13.87	8.94	4.72	3.63
Feature Tracking	MP	**5.64**	**3.87**	**1.45**	**1.08**	**0.96**
	MP (MPS)	14.73	6.04	1.51	1.31	1.09
	MT	31.11	20.86	11.51	4.68	2.68
Motion Estimation	MP	**28.64**	**21.25**	17.33	16.75	17.24
	MP (MPS)	33.05	22.66	**15.75**	**14.3**	**14.89**
	MT	42.86	26.12	16.53	15.07	15.14
Hough Transform	MP	**13.56**	**11.61**	7.28	5.68	5.7
	MP (MPS)	18.35	11.66	**6.44**	**3.74**	**3.18**
	MT	58.65	22.64	15.82	9.12	8.94
Stereo Matching	MP	75.13	50.54	30.42	24.14	24.77
	MP (MPS)	**59.73**	**45.05**	**26.87**	22.59	24.41
	MT	125.96	58.82	34.36	**20.75**	**18.95**

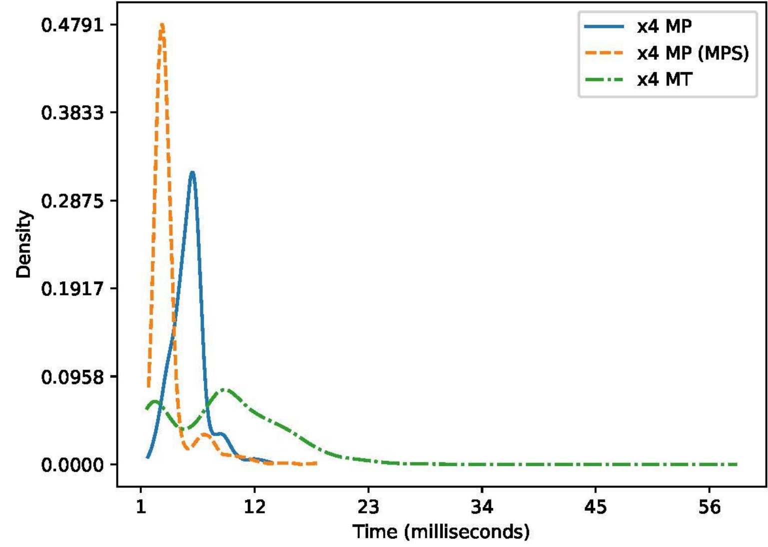
GPU Concurrency and Performance



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

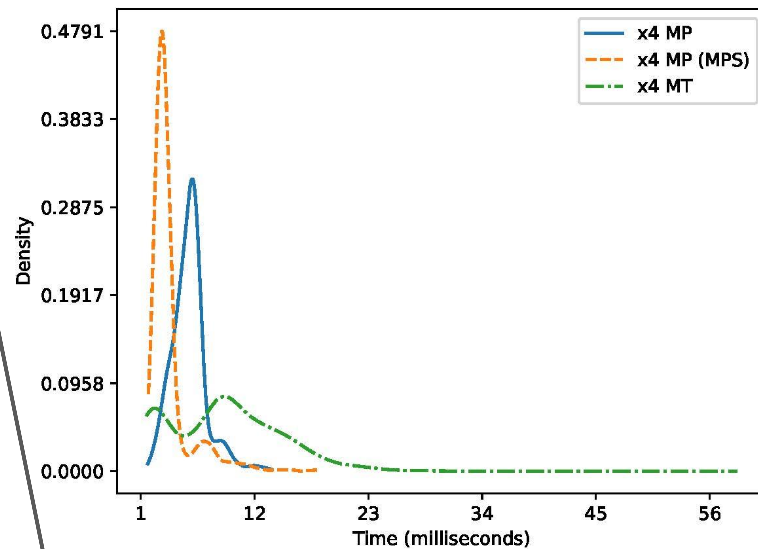
GPU Concurrency and Performance

100
80
60
40
20
0
% <= X
1 12 23 34 45 56
Time (milliseconds)
x4 MP
x4 MP (MPS)
x4 MT



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

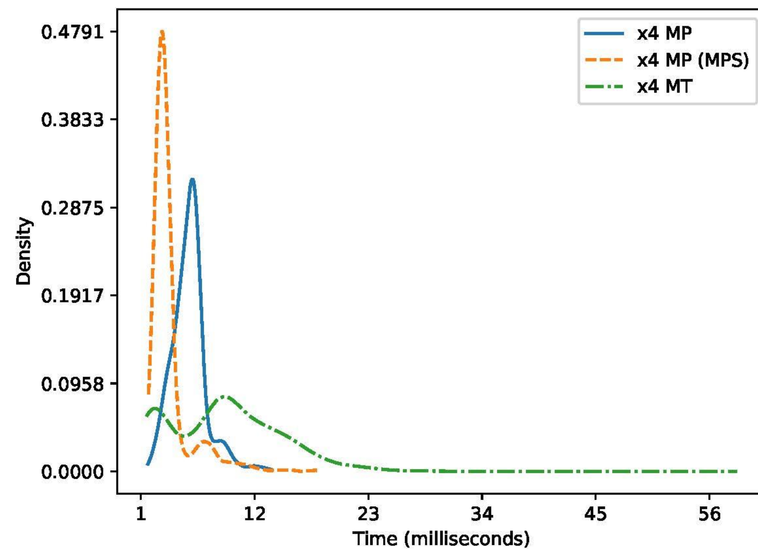
GPU Concurrency and Performance



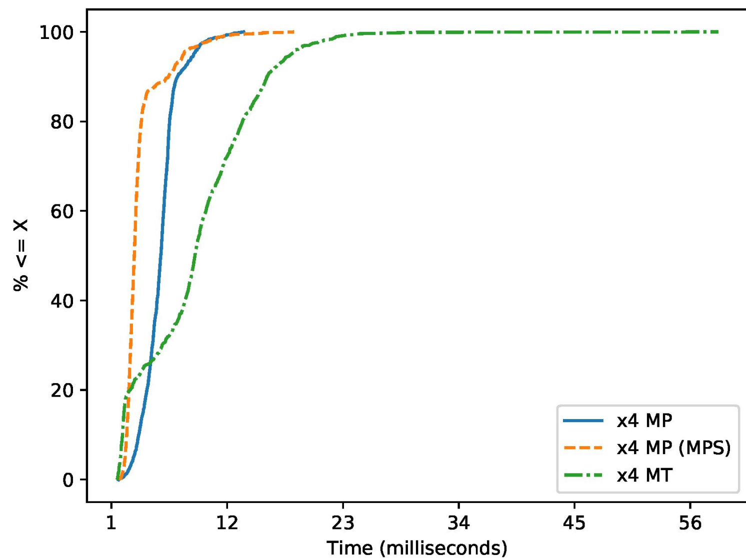
The observed WCET using threads was over 4x the WCET using multiple processes.

GPU Concurrency and Performance

100
80
60
40
20
0
% <= X
1
12
23
34
45
56
Time (milliseconds)
x4 MP
x4 MP (MPS)
x4 MT



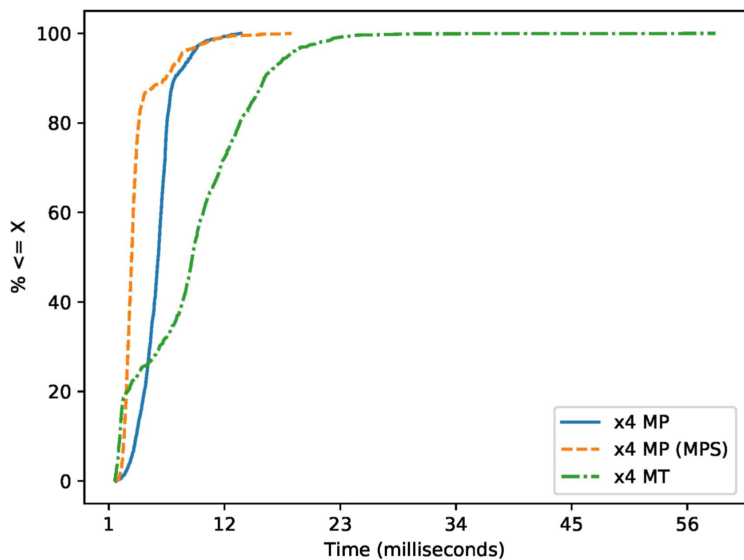
GPU Concurrency and Performance

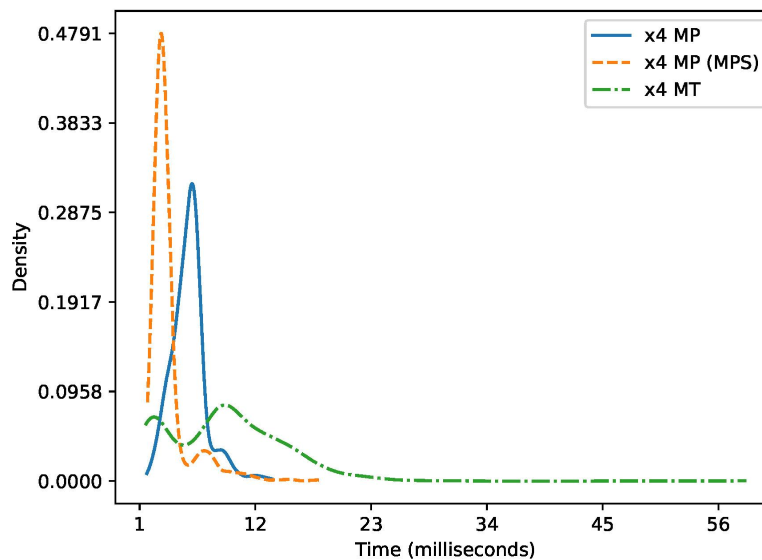


0.4791
0.3833
0.2875
0.1917
0.0958
0.0000
Density
1
12
23
34
45
56
Time (milliseconds)
x4 MP
x4 MP (MPS)
x4 MT

GPU Concurrency and Performance

➔ **Pitfall 5.** The suggestion from NVIDIA's documentation to exploit concurrency through user-defined streams may be of limited use.





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 Servi
- 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;
}
```

Why does this cause implicit synchronization?

Synchronous Defaults

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

- The CUDA docs say that memset causes implicit synchronization...

Synchronous Defaults

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

- The CUDA docs say that memset causes implicit synchronization...
- But didn't slide 20 say memset *doesn't* cause implicit synchronization?

Synchronous Defaults

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

```
if (!CheckCUDAError(
    cudaMemsetAsync(
    state->device_block_smids,
    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.

	Observed Behavior			Documented Behavior	
Source	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_fig1_320806445?_sg=ziaY-gBKKiKX4pljRq4vJSWZvDvdOidZ2aCRYnD1QVFBJDxIx3MEO1I03cI31e1It6pUr53qaS1L1w4Bt5fd8w