

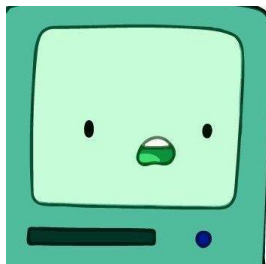
Day Final

Team 8 ELSA Robotics

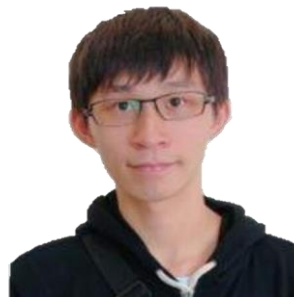
Team Members

Mentors

From NVIDIA



Frank Lin



Johnson Sun

Team Members

From NTHU ELSA



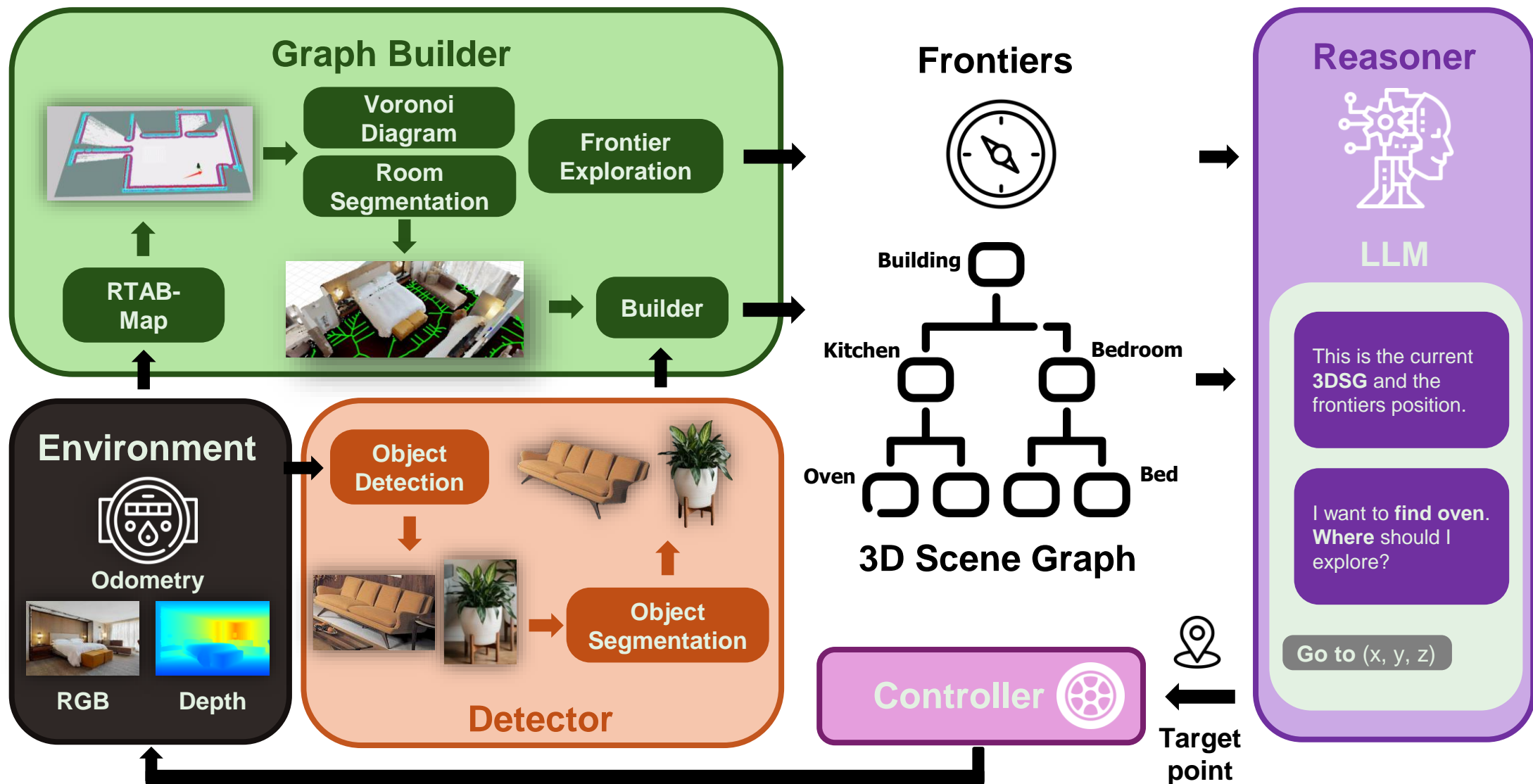
Yu-Zhong Chen



Yung-Shun Zhan



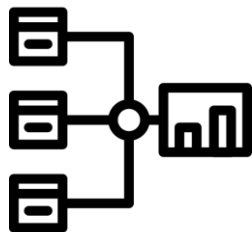
Chon-Hang Lam





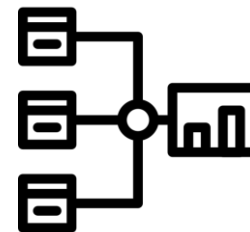
-

Total Speedup



Original pipeline

0.5 fps



Optimized pipeline

5.5 fps

Faster 11x

Optimized Objective

- (1) Depth to Point Cloud
- (2) YoloWorld

Evolution and Strategy

(1) Depth to Point Cloud

```

1  for i in range(len(points)):
2      # Transform the 2D point to a 3D point in map frame
3      map_point = np.array([0.0, 0.0, 0.0, 1.0], dtype=np.float32)
4      map_point[0] = (points[i][1] - cx) * depth[i] / fx
5      map_point[1] = (points[i][0] - cy) * depth[i] / fy
6      map_point[2] = depth[i]
7
8      # Apply the transformation
9      map_point = np.dot(transform, map_point)

```

Original

- Convert depth point to 3D
- Dot product for **each point**

$$([p_{i1} \ p_{i2}] - [c_x \ c_y]) \times \begin{bmatrix} \frac{d_i}{f_x} & 0 \\ 0 & \frac{d_i}{f_y} \end{bmatrix}$$

- Need to optimize for-loop

Evolution and Strategy

(1) Depth to Point Cloud - Optimization 1 / 3

```
1 # Precompute constants for the coordinate transformation
2 x_offsets = (points[:, 1] - cx) / fx
3 y_offsets = (points[:, 0] - cy) / fy
4
5 # Construct 3D points before transformation
6 map_points = np.vstack((
7     x_offsets * depth,
8     y_offsets * depth,
9     depth,
10    np.ones_like(depth),
11)).T
12
13 # Apply the transformation matrix
14 map_points = (transform @ map_points.T).T
```

Vectorization

- Calculate point offset for input image (**constant** part)
- Construct map point matrix
- Perform dot product for **all the point** in the matrix

Evolution and Strategy

(1) Depth to Point Cloud - Optimization 2 / 3

```
1  # Precompute constants for the coordinate transformation
2  x_offsets = (points[:, 1] - cx) / fx
3  y_offsets = (points[:, 0] - cy) / fy
4
5  # Transfer data to GPU
6  depth = cp.asarray(depth)
7  x_offsets = cp.asarray(x_offsets)
8  y_offsets = cp.asarray(y_offsets)
9  transform = cp.asarray(transform)
10
11 # Construct 3D points before transformation
12 map_points = cp.vstack((
13     x_offsets * depth,
14     y_offsets * depth,
15     depth,
16     cp.ones_like(depth),
17 )) .T
18
19 # Apply the transformation matrix
20 map_points = (transform @ map_points.T) .T
```

Vectorization with CuPy

- Replace Numpy with CuPy

Evolution and Strategy

(1) Depth to Point Cloud - Optimization 3 / 3

```

1  # CUDA kernel
2  kernel_code = """
3  extern "C" __global__
4  void Transform2DPointsToMap(const int* points, const float* depth, ...) {
5      // Get the index of the point
6      int idx = blockIdx.x * blockDim.x + threadIdx.x;
7      float point[4] = {0.0, 0.0, 0.0, 1.0};
8      point[0] = (points[idx * 2 + 1] - cx) * depth[idx] / fx;
9      point[1] = (points[idx * 2] - cy) * depth[idx] / fy;
10     point[2] = depth[idx];
11
12     // Dot product
13     float map_point[4] = {0.0, 0.0, 0.0, 1.0};
14     for (int i = 0; i < 4; i++) {
15         map_point[i] =
16             transform[i * 4 + 0] * point[0] +
17             transform[i * 4 + 1] * point[1] +
18             transform[i * 4 + 2] * point[2] +
19             transform[i * 4 + 3] * point[3];
20     }
21
22     // Store the result
23     map_points[idx * 3 + 0] = map_point[0];
24     map_points[idx * 3 + 1] = map_point[1];
25     map_points[idx * 3 + 2] = map_point[2];
26 }
27 """
28
29 # Compile the CUDA kernel
30 module = cp.RawModule(code=kernel_code, backend="nVRTC")

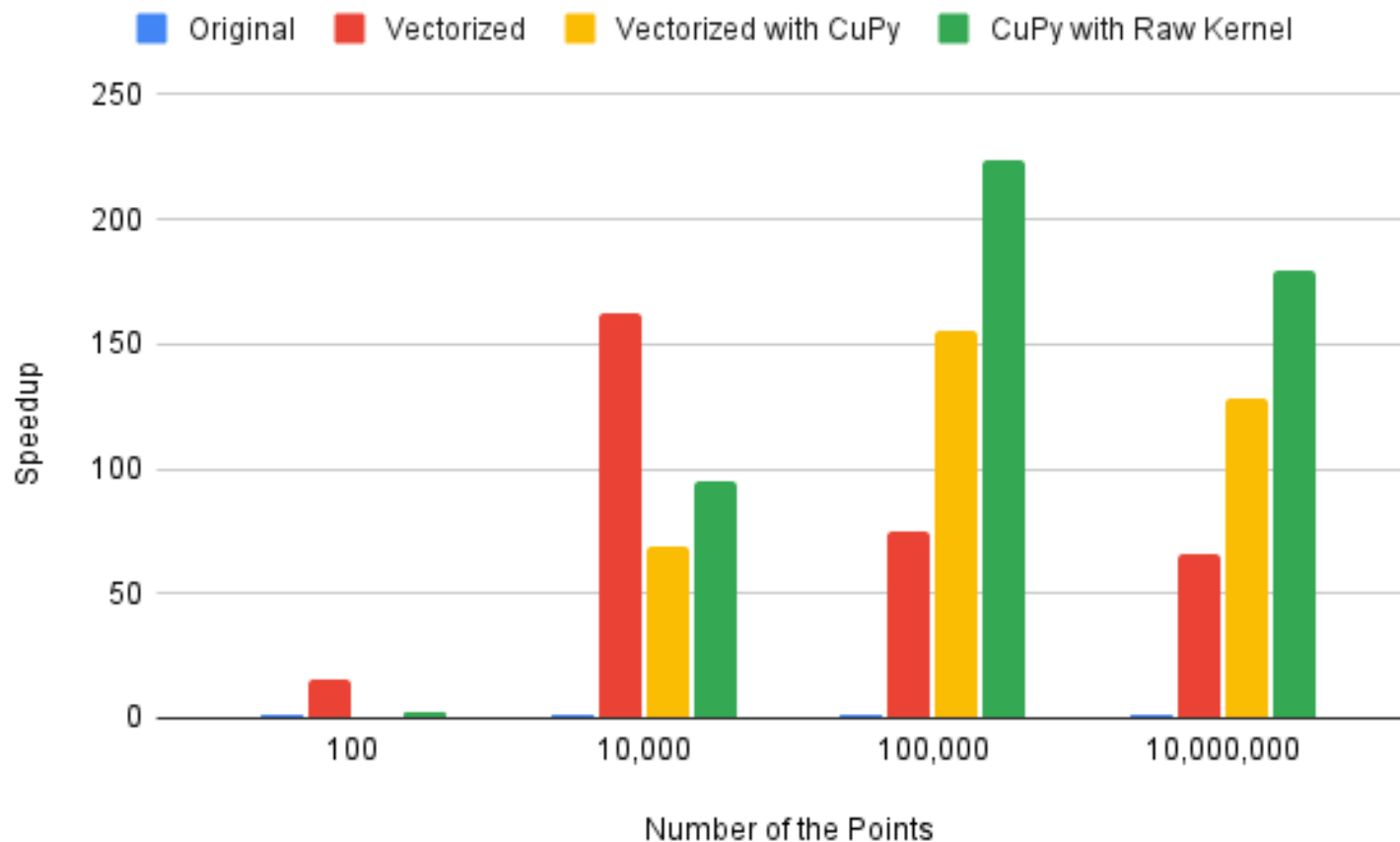
```

CuPy with Raw Kernel

- Optimized the original for-loop with raw kernel

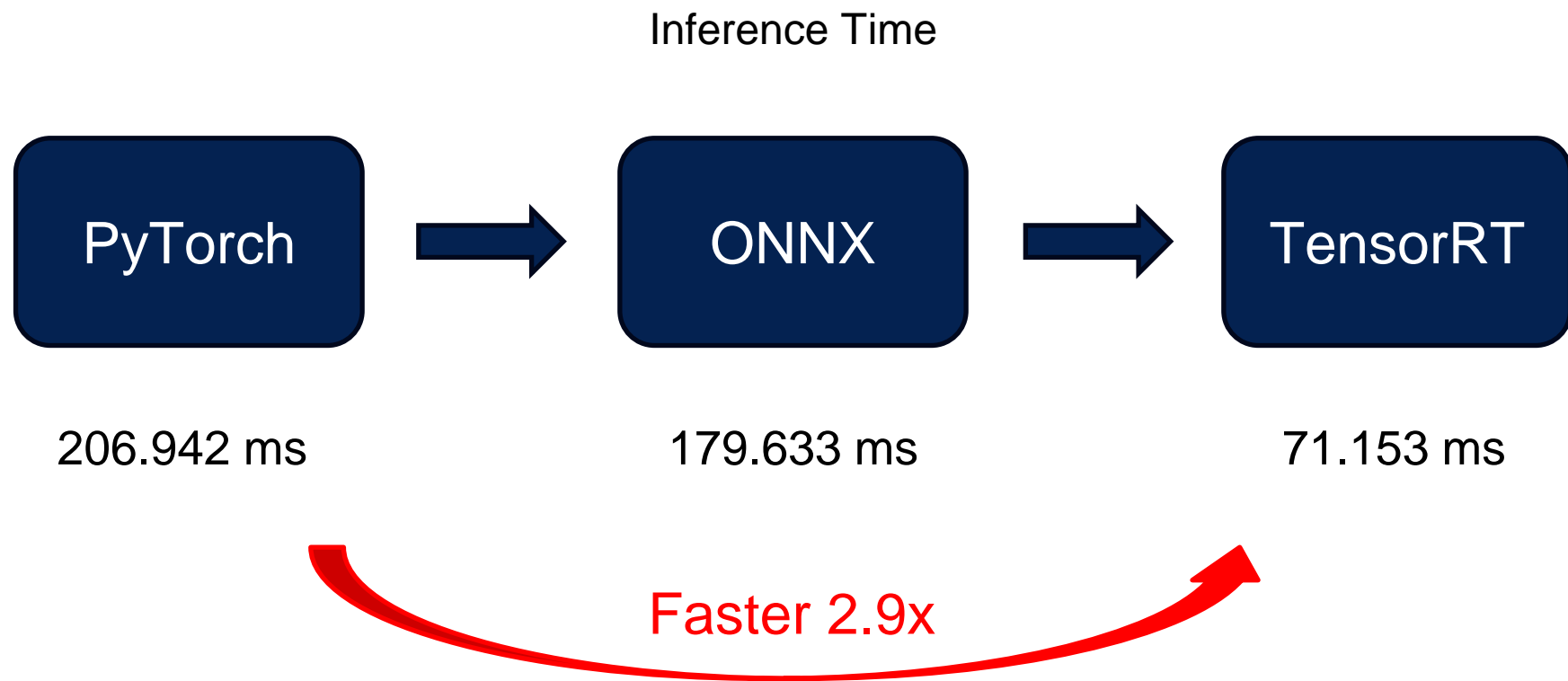
Evolution and Strategy

Result (Speedup)



Evolution and Strategy

(2) YoloWorld



Evolution and Strategy

(2) YoloWorld - Before optimization

```
1 # Allocate Memory
2 d_input = cuda.mem_alloc(1 * 3 * 640 * 640 * 4)
3 d_output = cuda.mem_alloc(1 * 8400 * 13 * 4)
4 np_output = np.empty(1 * 8400 * 13, dtype=np.float32)
5
6 # Set Input and Output Addresses
7 context.set_tensor_address(tensor_names[0], int(d_input))
8 context.set_tensor_address(tensor_names[1], int(d_output))
9
10 # Inference
11 cuda.memcpy_htod_async(d_input, image, stream) # Transfer data to device
12 context.execute_async_v3(stream.handle) # Execute model
13 cuda.memcpy_dtoh_async(np_output, d_output, stream) # Transfer predictions back to host
14
15 # Create Torch Tensor from NumPy
16 torch_output = torch.from_numpy(output).to("cuda:0")
17
18 # Postprocess ...
```

GPU

CPU

- Memory allocation

Evolution and Strategy

(2) YoloWorld - Before optimization

```
1 # Allocate Memory
2 d_input = cuda.mem_alloc(1 * 3 * 640 * 640 * 4)
3 d_output = cuda.mem_alloc(1 * 8400 * 13 * 4)
4 np_output = np.empty(1 * 8400 * 13, dtype=np.float32)
5
6 # Set Input and Output Addresses
7 context.set_tensor_address(tensor_names[0], int(d_input))
8 context.set_tensor_address(tensor_names[1], int(d_output))
9
10 # Inference
11 cuda.memcpy_htod_async(d_input, image, stream) # Transfer data to device
12 context.execute_async_v3(stream.handle) # Execute model
13 cuda.memcpy_dtoh_async(np_output, d_output, stream) # Transfer predictions back to host
14
15 # Create Torch Tensor from NumPy
16 torch_output = torch.from_numpy(output).to("cuda:0")
17
18 # Postprocess ...
```

- Memory allocation
- Setting address

Evolution and Strategy

(2) YoloWorld - Before optimization

```
1 # Allocate Memory
2 d_input = cuda.mem_alloc(1 * 3 * 640 * 640 * 4)
3 d_output = cuda.mem_alloc(1 * 8400 * 13 * 4)
4 np_output = np.empty(1 * 8400 * 13, dtype=np.float32)
5
6 # Set Input and Output Addresses
7 context.set_tensor_address(tensor_names[0], int(d_input))
8 context.set_tensor_address(tensor_names[1], int(d_output))
9
10 # Inference
11 cuda.memcpy_htod_async(d_input, image, stream) # Transfer data to device
12 context.execute_async_v3(stream.handle) # Execute model
13 cuda.memcpy_dtoh_async(np_output, d_output, stream) # Transfer predictions back to host
14
15 # Create Torch Tensor from NumPy
16 torch_output = torch.from_numpy(output).to("cuda:0")
17
18 # Postprocess ...
```

- Memory allocation
- Setting address
- Perform inference and Move back to CPU

Evolution and Strategy

(2) YoloWorld - Before optimization

```
1 # Allocate Memory
2 d_input = cuda.mem_alloc(1 * 3 * 640 * 640 * 4)
3 d_output = cuda.mem_alloc(1 * 8400 * 13 * 4)
4 np_output = np.empty(1 * 8400 * 13, dtype=np.float32)
5
6 # Set Input and Output Addresses
7 context.set_tensor_address(tensor_names[0], int(d_input))
8 context.set_tensor_address(tensor_names[1], int(d_output))
9
10 # Inference
11 cuda.memcpy_htod_async(d_input, image, stream) # Transfer data to device
12 context.execute_async_v3(stream.handle) # Execute model
13 cuda.memcpy_dtoh_async(np_output, d_output, stream) # Transfer predictions back to host
14
15 # Create Torch Tensor from NumPy
16 torch_output = torch.from_numpy(output).to("cuda:0")
17
18 # Postprocess ...
```

- Memory allocation
- Setting address
- Perform inference and Move back to CPU
- Create Torch Tensor (GPU) and post-processing

Evolution and Strategy

(2) YoloWorld - After optimization

```
1  # Allocate Memory
2  d_input = torch.zeros([1 * 3 * 640 * 640], dtype=torch.float32, device="cuda:0")
3  d_output = torch.zeros([8400, 13], dtype=torch.float32, device="cuda:0")
4
5  # Set Input and Output Addresses
6  context.set_tensor_address(tensor_names[0], d_input.data_ptr())
7  context.set_tensor_address(tensor_names[1], d_output.data_ptr())
8
9  # Inference
10 d_input.copy_(torch.from_numpy(image)) # Transfer input data to device
11 context.execute_async_v3(stream_handle=torch.cuda.current_stream().cuda_stream) # Execute model
12
13 # Postprocess ...
```

- Memory allocation

Evolution and Strategy

(2) YoloWorld - After optimization

```
1  # Allocate Memory
2  d_input = torch.zeros([1 * 3 * 640 * 640], dtype=torch.float32, device="cuda:0")
3  d_output = torch.zeros([8400, 13], dtype=torch.float32, device="cuda:0")
4
5  # Set Input and Output Addresses
6  context.set_tensor_address(tensor_names[0], d_input.data_ptr())
7  context.set_tensor_address(tensor_names[1], d_output.data_ptr())
8
9  # Inference
10 d_input.copy_(torch.from_numpy(image)) # Transfer input data to device
11 context.execute_async_v3(stream_handle=torch.cuda.current_stream().cuda_stream) # Execute model
12
13 # Postprocess ...
```

- Memory allocation
- Setting address

Evolution and Strategy

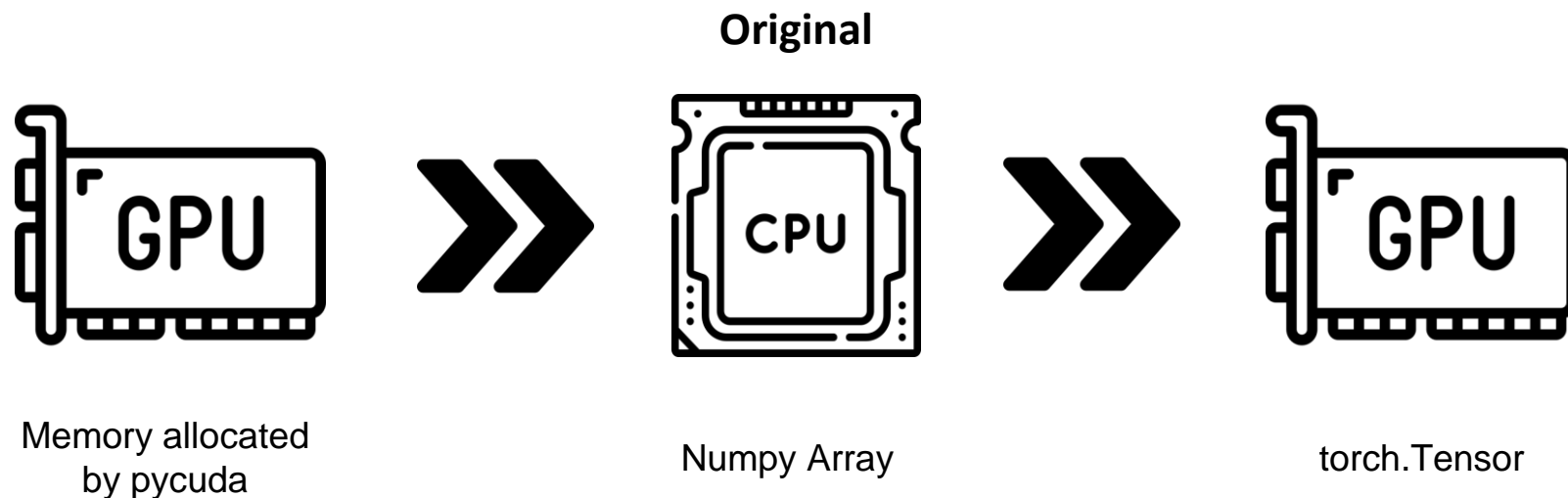
(2) YoloWorld - After optimization

```
1  # Allocate Memory
2  d_input = torch.zeros([1 * 3 * 640 * 640], dtype=torch.float32, device="cuda:0")
3  d_output = torch.zeros([8400, 13], dtype=torch.float32, device="cuda:0")
4
5  # Set Input and Output Addresses
6  context.set_tensor_address(tensor_names[0], d_input.data_ptr())
7  context.set_tensor_address(tensor_names[1], d_output.data_ptr())
8
9  # Inference
10 d_input.copy_(torch.from_numpy(image)) # Transfer input data to device
11 context.execute_async_v3(stream_handle=torch.cuda.current_stream().cuda_stream) # Execute model
12
13 # Postprocess ...
```

- Memory allocation
- Setting address
- Perform inference and post-processing

Evolution and Strategy

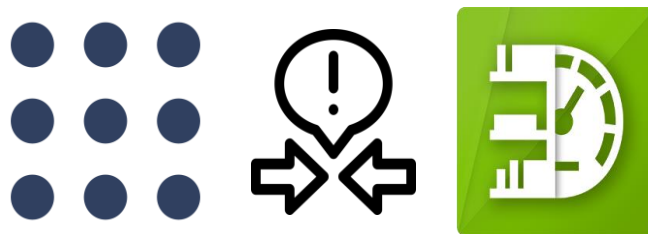
(2) YoloWorld



> Solve with only use PyTorch, and use `tensor.data_ptr()` to get the pointer.

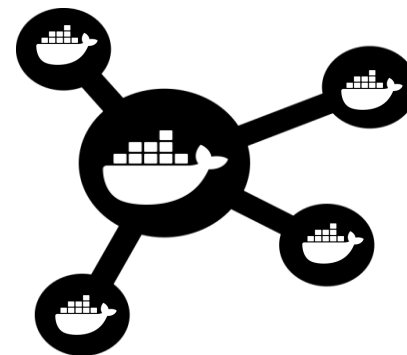
2.9x → 3.1x

What problems have you encountered?



Hard to profile ROS with Nsight System

- Profiling with NVTX tag



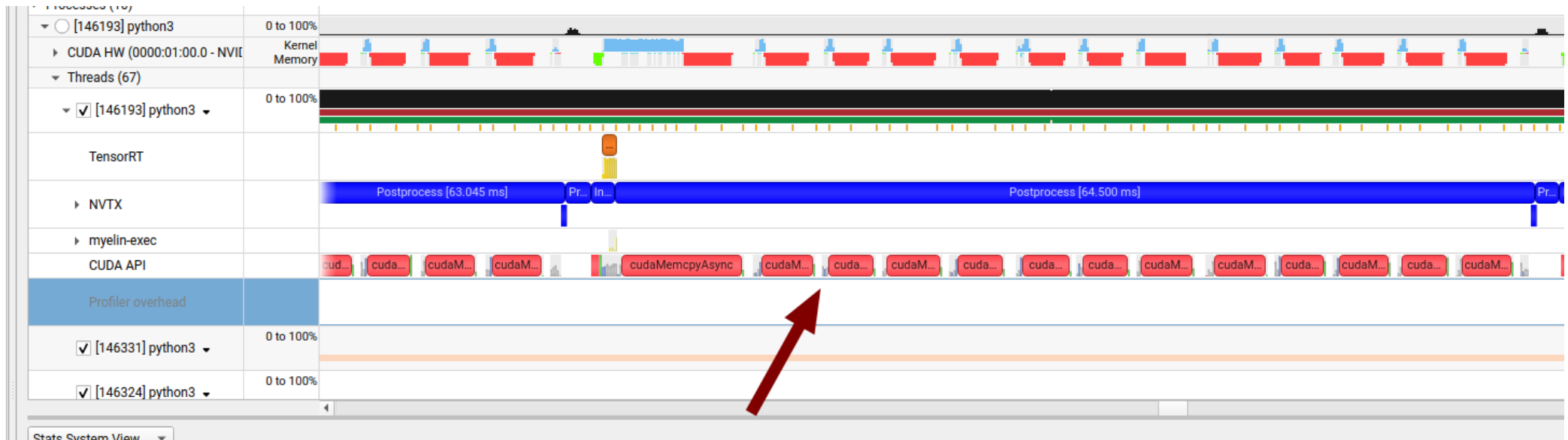
Profiling across multi-container

- Profiling each process with each container

Future Work

YoloWorld

Image post-processing was time-consuming because it moved data between the GPU and the CPU periodically.

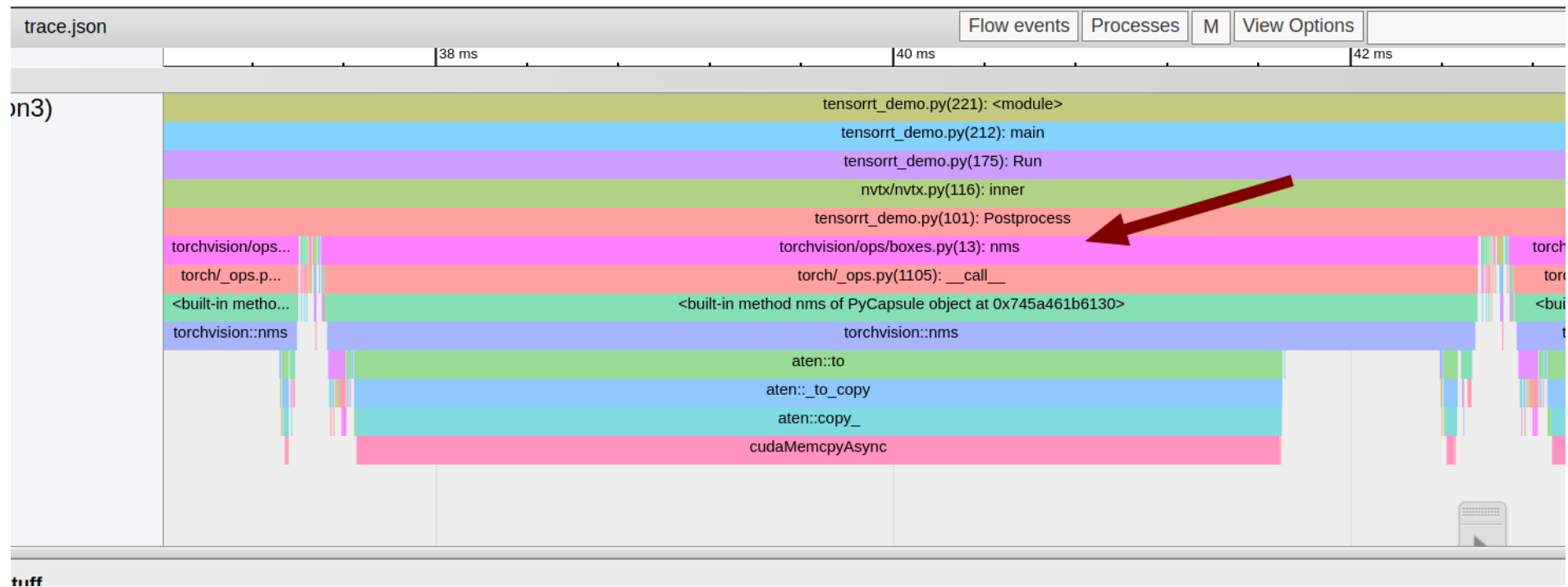


Need to remove the bubble between these commands.

Future Work

YoloWorld

By profiling result, the **cudaMemcpyAsync** was executed from Torchvision's NMS.





Thank You

OpenACC
More Science. Less Programming