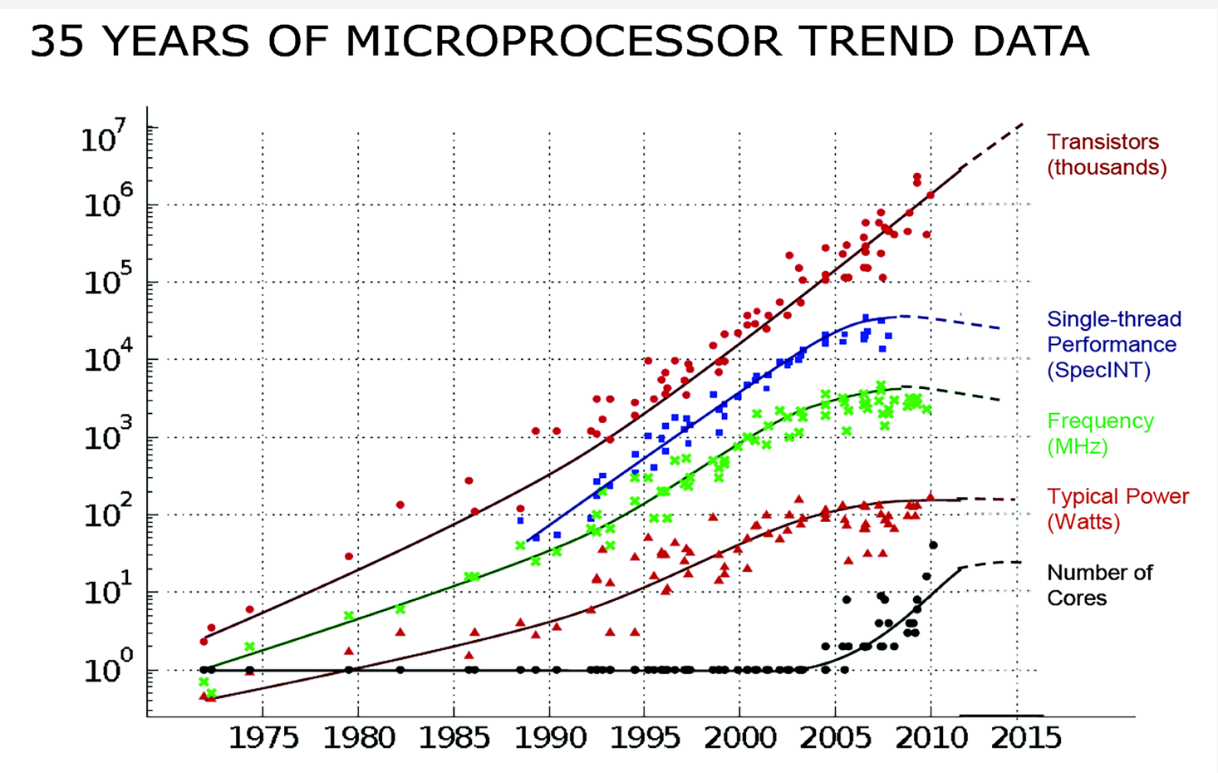
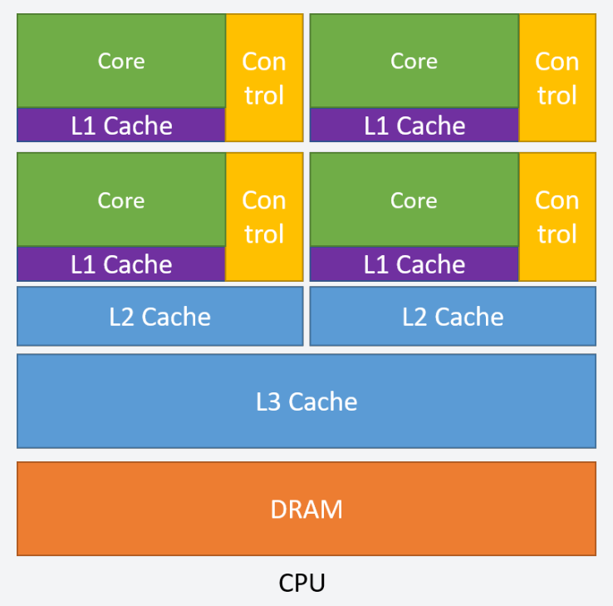
Efficiently Executing *NumPy* on GPUs via the *CUDAGraph* API

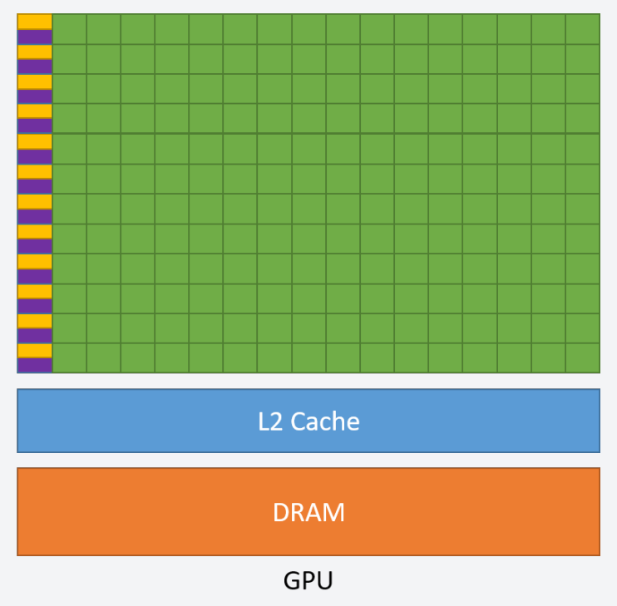
Mit Kotak · Kaushik Kulkarni · Andreas Klöckner

National Center for Supercomputing Applications · Urbana, IL

Why GPUs?







- CPU performance is flatlining
- GPUs devote more transistors to data processing

Abstract

- *NumPy* is the defacto standard for array-based numerical computations in scientific workloads (for ex. PDE solving, Image Processing, etc.)
- However, its ability to harness GPU acceleration is limited by its single-node, occasionally multi-threaded, CPU- only execution model
- Goal: Should be able to efficiently execute array operations on GPUs
- Challenges:
  - Saturating all available execution units to efficiently use the entire GPU
  - *NumPy* is a high-level programming interface, does not address performance directives such as:
    - Execution grid-size which affects data partitioning
    - User-specified local memory management
    - Kernel launch overhead costs
  - **Realizing concurrency across array operations**
- Our Approach: Use *CUDAGraph* API to concurrently launch array operations using tuned kernels that are accessed through *PyCUDA*
- Past work:
  - *Legate*: A runtime system for scheduling operations in a task graph
    - Representation lacks a global view of the program limiting the program optmizing space
  - Lazy evaluation: *Theano* | *JAX* | *PyTorch*
    - Requires expensive algorithms (ex. kernel/loop fusion)
    - Ex. *Theano*[3] claim to have super-linear codegeneration algorithm
  - Single Stream: *cuPy* | *GPUArrays.jl*

Why *CUDAGraphs*?

- *CUDA Streams*
  - Operations are enqueued in-order into the stream object
- *CUDAGraph* API
  - Takes in a task dependency graph where each node corresponds to a *CUDA* kernel
  - Scheduler realizes concurrency across nodes in the graph through multiple streams

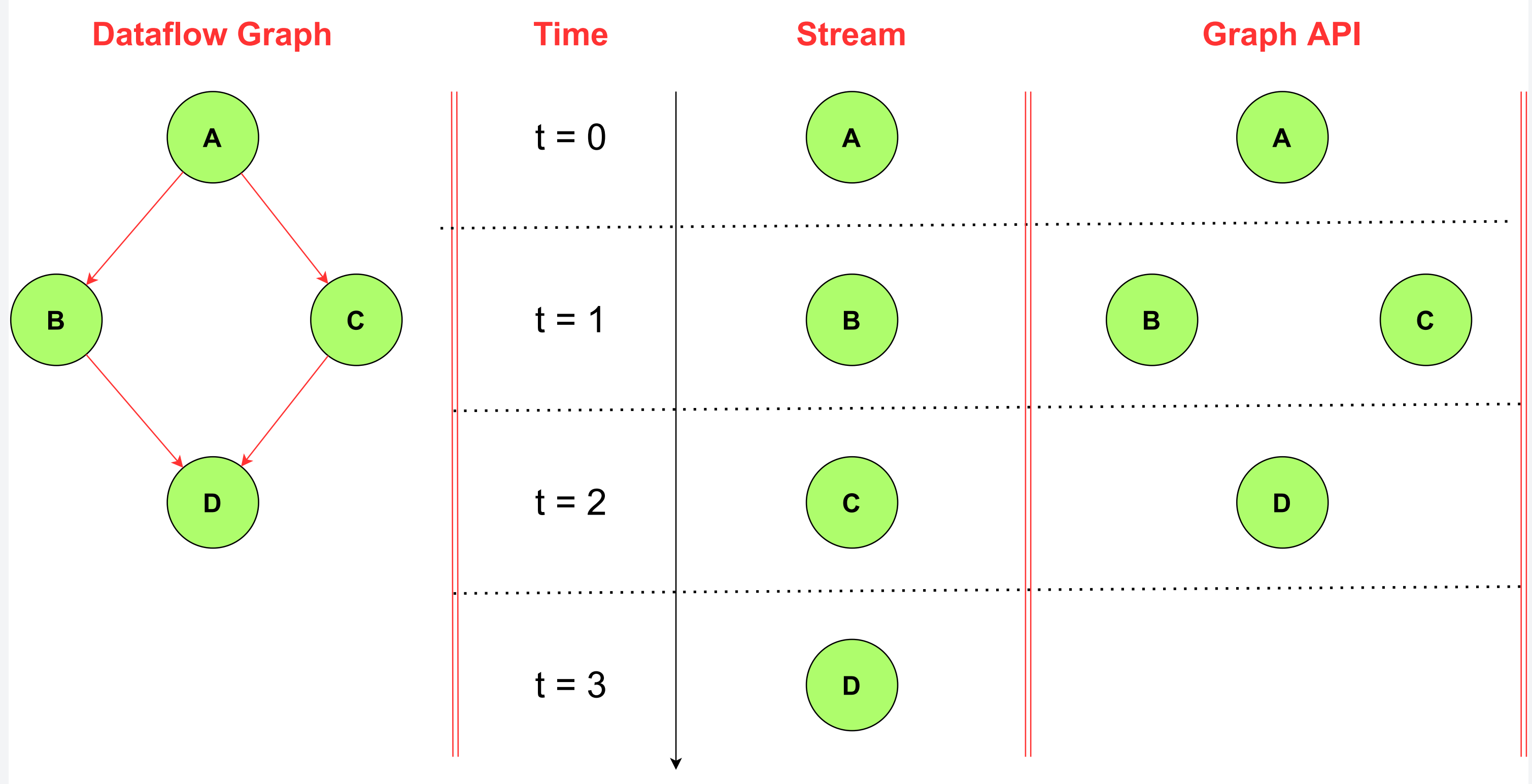
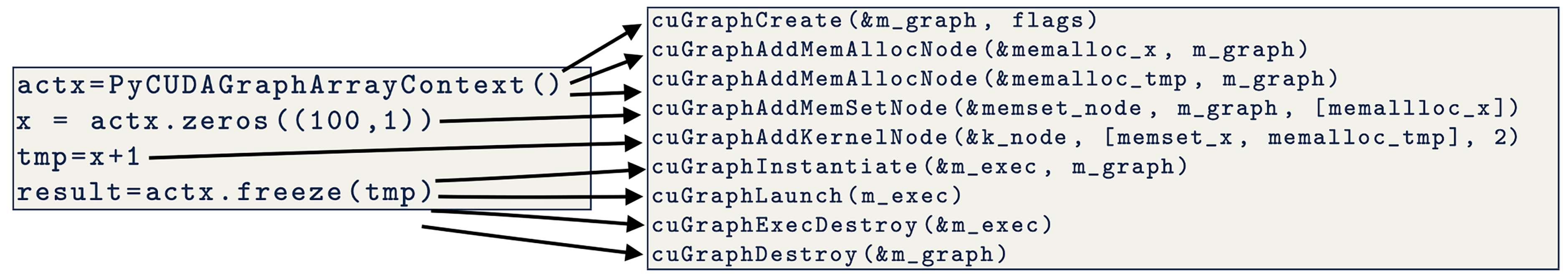


Figure: Single Stream vs CUDAGraphs scheduling

Interface Implementation



```
actx=PyCUDAGraphArrayContext()
x = actx.zeros((100,1))
tmp=x+1
result=actx.freeze(tmp)
```

```
cuGraphCreate(&m_graph, flags)
cuGraphAddMemAllocNode(&memalloc_x, m_graph)
cuGraphAddMemAllocNode(&memalloc_tmp, m_graph)
cuGraphAddMemSetNode(&memset_node, m_graph, [memallloc_x])
cuGraphAddKernelNode(&k_node, [memset_x, memalloc_tmp], 2)
cuGraphInstantiate(&m_exec, m_graph)
cuGraphLaunch(m_exec)
cuGraphExecDestroy(&m_exec)
cuGraphDestroy(&m_graph)
```

Figure: User Input Program

Figure: Driver C code

- Array operations realized as a composition of *CUDA* calls (memcpy, kernel_launches, memalloc) that are added onto a task dependency graph with precise edges.
- Object cleanup tied to lifetime of objects in array operations

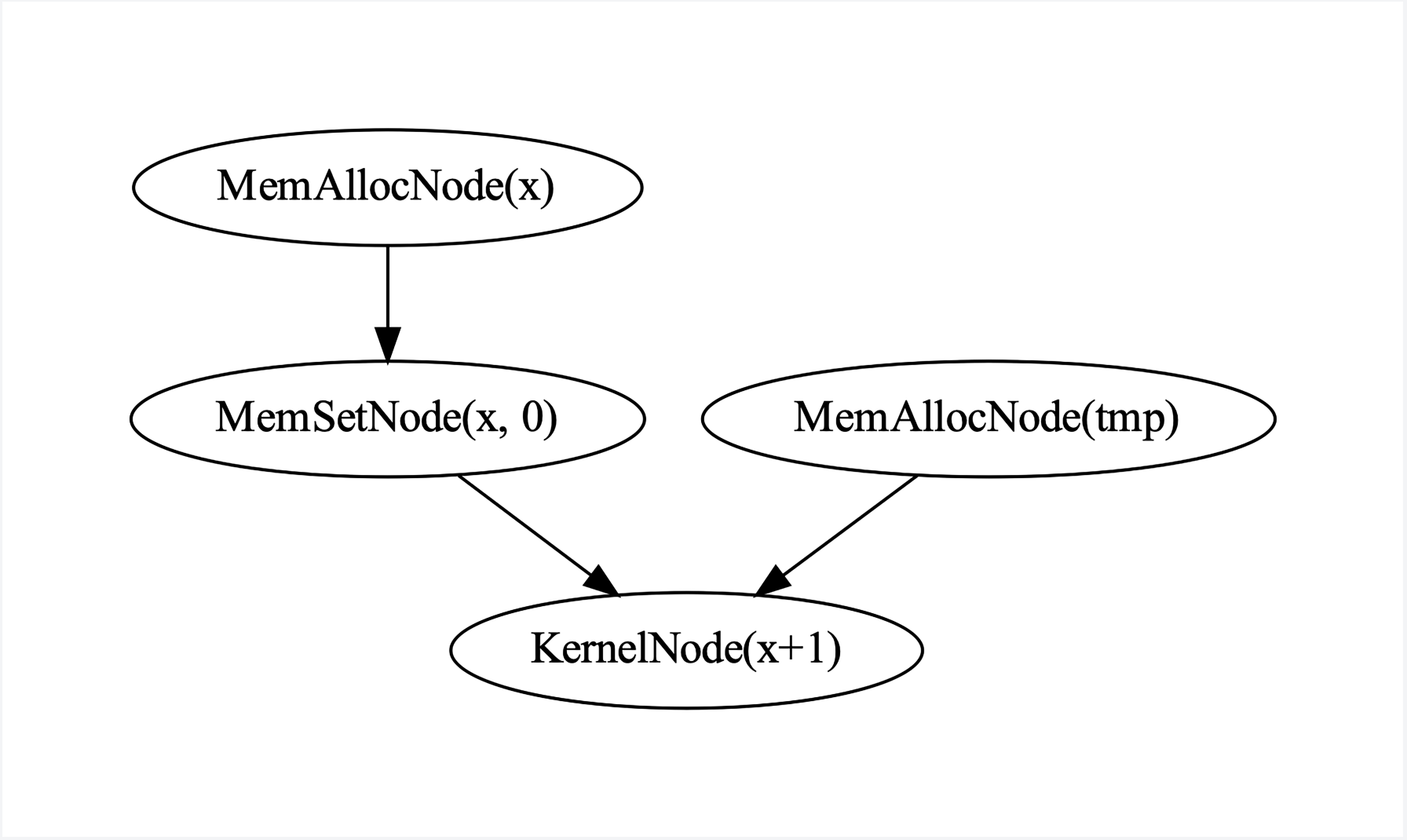
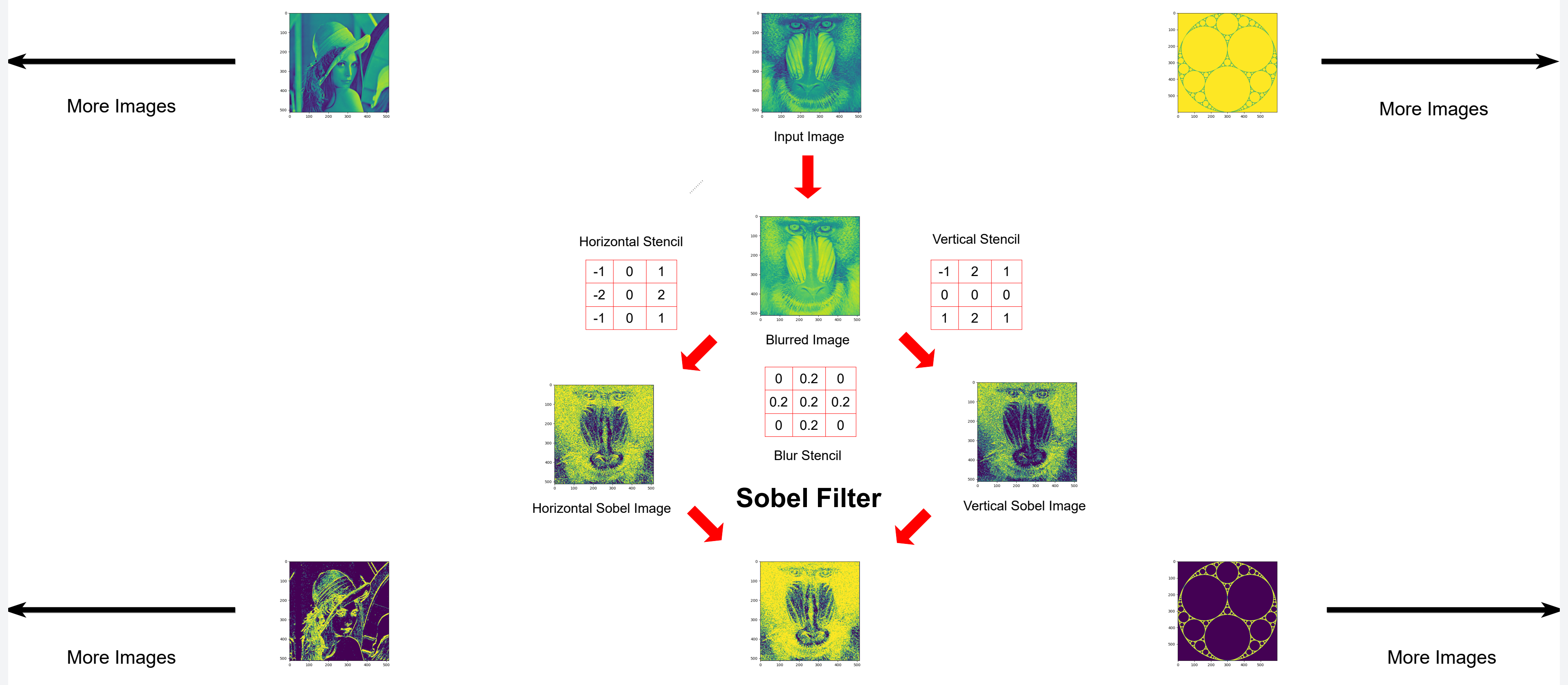


Figure: Generated CUDAGraph

Experimental Setup

We ran a set of image processing algorithms with and without *CUDAGraph* API on different image batches



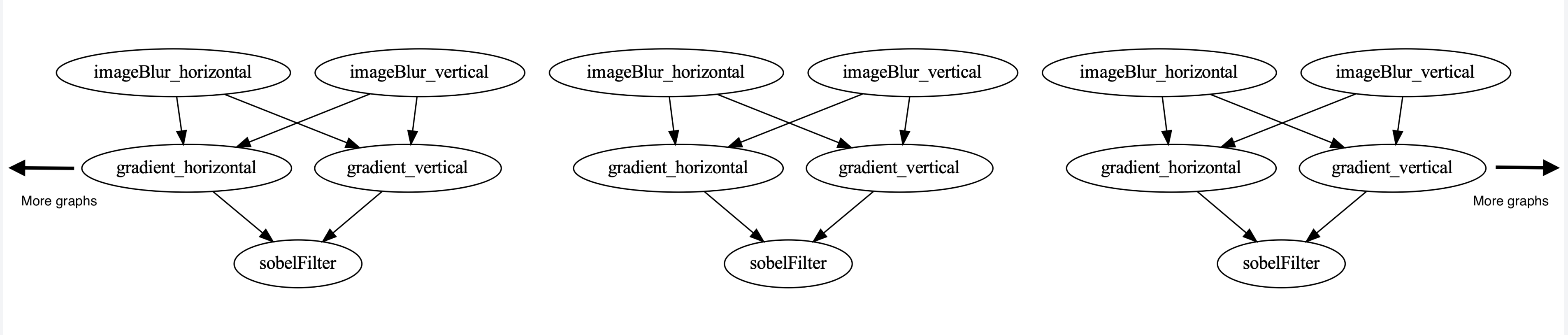


Figure: Sobel Filter Graph

Results

- We observed a speed up of upto 15% on NVIDIA Titan V for smaller problems which was attributed to high task parallelism
- We observed overlapping kernels across multiple streams for the *CUDAGraph* API program

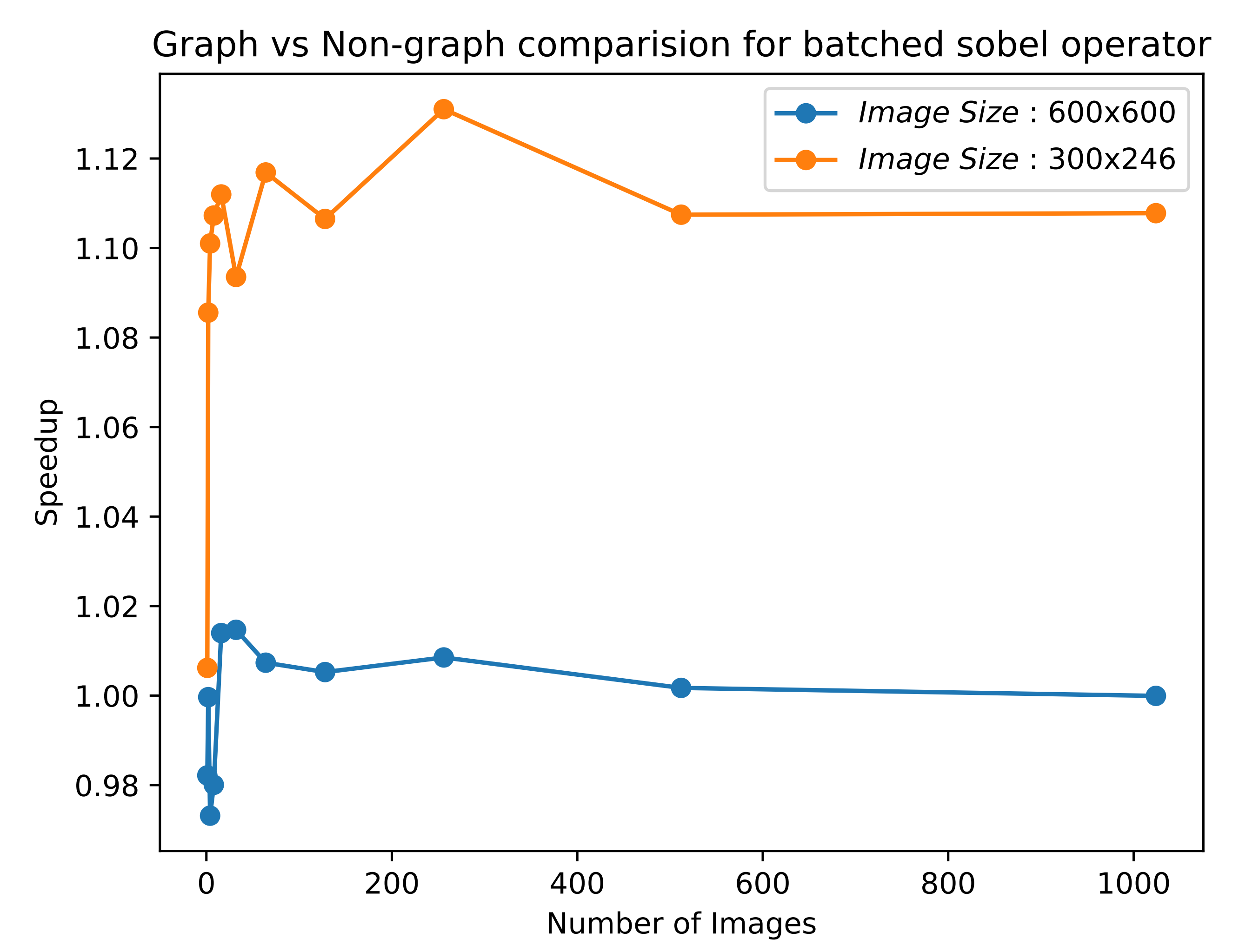


Figure: Sobel Filter on Image Batches

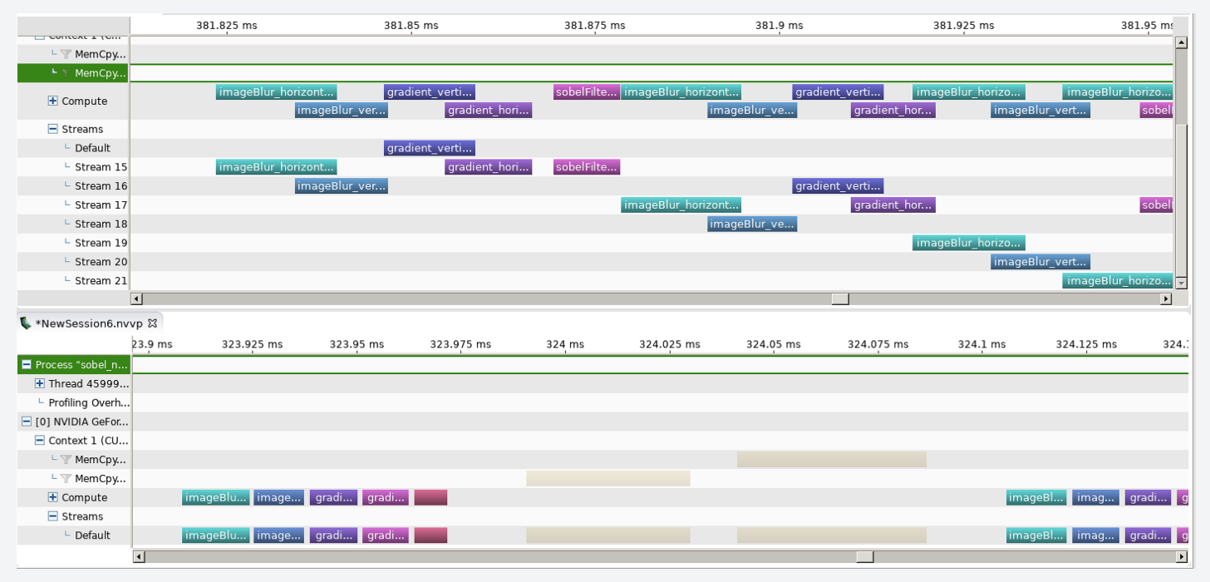


Figure: Kernel Execution timeline with (top) and without (bottom) *CUDAGraph* API Image size 300 x 246

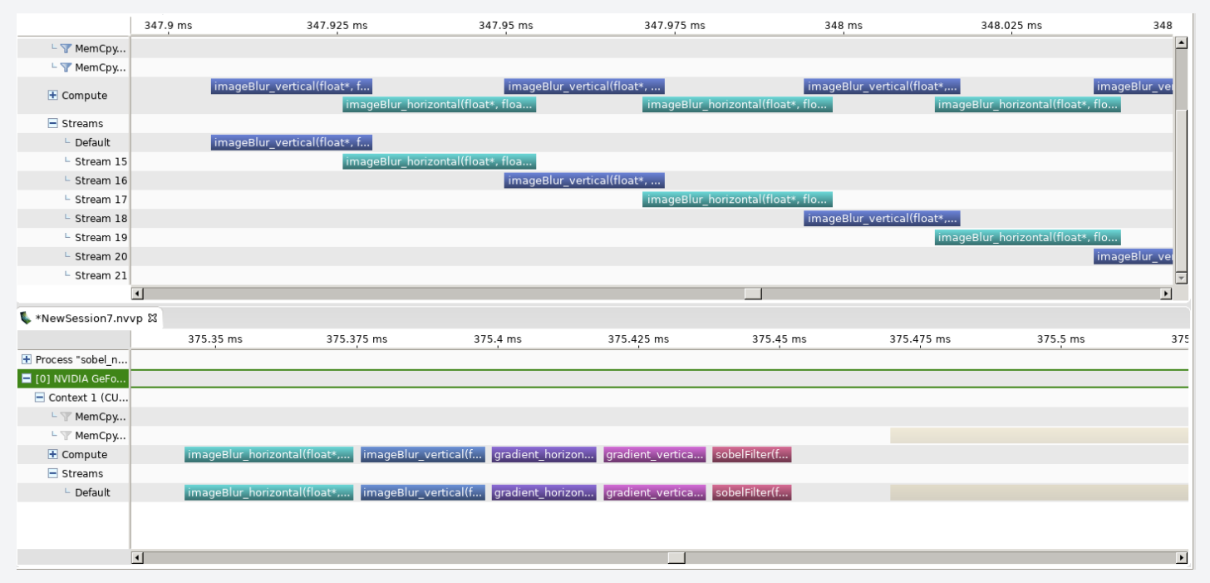


Figure: Kernel Execution timeline with (top) and without (bottom) *CUDAGraph* API Image size 600 x 600

Results, run instructions at `https://github.com/mitkotak/sobel`

Future Work

- Upstream work: Integrate with *PyCUDA* and *Arraycontext*
- Evalulate this approach on large scale scientific simulations

Acknowledgements

This material is based in part upon work supported by the Department of Energy, National Nuclear Security Administration, under Award Number DE-NA0003963

References

- CUDA Programming: A Developer's Guide to Parallel Computing with GPUs (https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html)
- Chapter 27 - GPU Scripting and Code Generation with PyCUDA (https://doi.org/10.1016/B978-0-12-385963-1.00027-7)
- Theano: A Python framework for fast computation of mathematical expressions (https://doi.org/10.48550/arXiv.1605.02688)

mitak2@illinois.edu