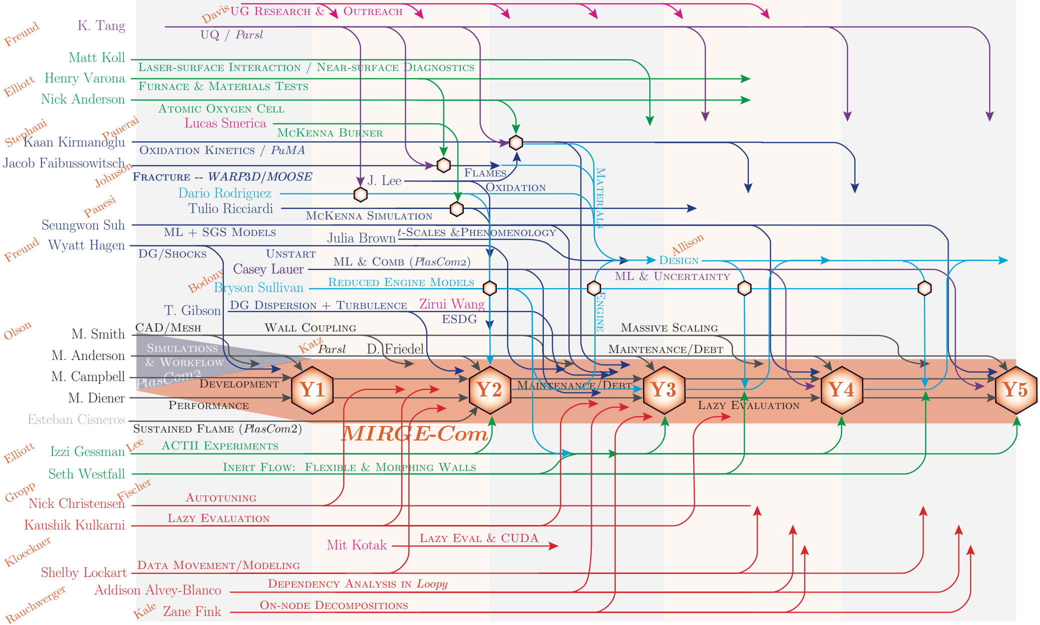


Davis
UG Research & Outreach
Freund
K. Tang
UQ / Parsl
Matt Koll
Laser-surface Interaction / Near-surface Diagnostics
Elliott
Henry Varona
Furnace & Materials Tests
Nick Anderson
Atomic Oxygen Cell
Lucas Smerica
McKenna Burner
Stephani
Panerai
Kaan Kirmanoglu
Oxidation Kinetics / PuMA
Jacob Faibussowitsch
Johnson
Fracture -- WARP3D/MOOSE
J. Lee
Flames
Oxidation
Materials
Dario Rodriguez
Tulio Ricciardi
McKenna Simulation
Panesi
Seungwon Suh
ML + SGS Models
Julia Brown
t-Scales &Phenomenology
Freund
Wyatt Hagen
DG/Shocks
Unstart
Allison
Design
Casey Lauer
ML & Comb (PlasCom2)
ML & Uncertainty
Bodony
Bryson Sullivan
Reduced Engine Models
Engine
T. Gibson
DG Dispersion + Turbulence
Zirui Wang
ESDG
Olson
M. Smith
CAD/Mesh
Wall Coupling
Massive Scaling
Simulations & Workflow
Katz
Parsl
D. Friedel
M. Anderson
Maintenance/Debt
M. Campbell
PlasCom2
Development
Y1
Y2
Maintenance/Debt
Y3
Y4
Y5
M. Diener
Performance
Lazy Evaluation
Esteban Cisneros
Sustained Flame (PlasCom2)
MIRGE-Com
Elliott
Izzi Gessman
Lee
ACTII Experiments
Seth Westfall
Inert Flow: Flexible & Morphing Walls
Gropp
Fischer
Nick Christensen
Autotuning
Kaushik Kulkarni
Lazy Evaluation
Mit Kotak
Lazy Eval & CUDA
Kloeckner
Shelby Lockart
Data Movement/Modeling
Addison Alvey-Blanco
Dependency Analysis in Loopy
Rauchwerger
Kale
Zane Fink
On-node Decompositions

Task Graph Parallelism on GPUs via CUDAGraphs

Mit Kotak, Kaushik Kulkarni

University of Illinois at Urbana-Champaign

October 27, 2022

Motivation

MIRGE-Com: Overview

- DG-FEM solver using array-valued data flow graphs.
- For our experiments, we use an unfused version of the operator.

Problem

- Target the concurrency across the launched GPU kernels.
  - Would lead to better device usage at lower problem sizes.
  - Hypothesize: Cost savings from:
    - Lower launch overhead
    - Exploiting overlap

Approach

- Develop a new `ArrayContext`.
- Map *Numpy*-like operations to graph-based IR (*Pytato*).
- Generate *CUDAGraph* source code by mapping *Pytato* IR onto *PyCUDA*.

Code Transformation

```
actx = PytatoCUDAGraphArrayContext()
def f():
    return actx.zeros(100, dtype="float") + 1
f_compiled = actx.compile(f)
f_compiled()
```

Figure: `Arraycontext` Program

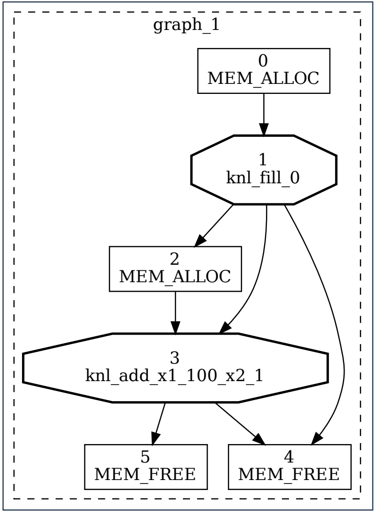


Figure: Generated *CUDAGraph*

Code Transformation

```
actx = PytatoCUDAGraphArrayContext()
def f():
    return actx.zeros(100, dtype="float") + 1
f_compiled = actx.compile(f)
f_compiled()
```

Figure: `Arraycontext` Program

```
import pycuda.driver as _pt_drv
import numpy as np
from pycuda.compiler import SourceModule as _pt_SourceModule
from pycuda import gpuarray as _pt_gpuarray
from functools import cache
_pt_mod_1 = _pt_SourceModule('#define bIdx(N) ((int) blockIdx.N)\n#define tIdx(N) ((int) threadIdx.N)\n\
      nextern "C" __global__ void __launch_bounds__(32) knl_fill_0(double *__restrict__ out)\n{\n  if (99 +
       -1 * tIdx(x) + -32 * bIdx(x) >= 0)\n  {\n    int const ibatch = 0;\n\n    out[tIdx(x) + 32 * bIdx(x)
      ] = 0.0;\n  }\n}')
_pt_mod_2 = _pt_SourceModule('#define bIdx(N) ((int) blockIdx.N)\n#define tIdx(N) ((int) threadIdx.N)\n\
      nextern "C" __global__ void __launch_bounds__(32) knl_add_x1_100_x2_1(double *__restrict__ out,
      double const *__restrict__ _in0)\n{\n  if (99 + -1 * tIdx(x) + -32 * bIdx(x) >= 0)\n  {\n    int
      const ibatch = 0;\n\n    out[tIdx(x) + 32 * bIdx(x)] = _in0[tIdx(x) + 32 * bIdx(x)] + 11;\n  }\n}')

@cache
def exec_graph_builder():
    _pt_g = _pt_drv.Graph()
    _pt_buffer_acc = {}
    _pt_node_acc = {}
    (_pt_memalloc_0, _pt_array_0) = _pt_g.add_memalloc_node(size=800, dependencies=[])
    _pt_kernel_0 = _pt_g.add_kernel_node(_pt_array_0, func=_pt_mod_1.get_function('knl_fill_0'), block=(32,
      1, 1), grid=(4, 1, 1), dependencies=[_pt_memalloc_0])
    _pt_buffer_acc['_pt_array_0'] = _pt_array_0
    _pt_node_acc['_pt_kernel_0'] = _pt_kernel_0
    (_pt_memalloc, _pt_array) = _pt_g.add_memalloc_node(size=800, dependencies=[_pt_kernel_0])
    _pt_kernel = _pt_g.add_kernel_node(_pt_array, _pt_array_0, func=_pt_mod_2.get_function('
      knl_add_x1_100_x2_1'), block=(32, 1, 1), grid=(4, 1, 1), dependencies=[_pt_memalloc, _pt_kernel_0])
    _pt_buffer_acc['_pt_array'] = _pt_array
    _pt_buffer_acc['_pt_array_0'] = _pt_array_0
    _pt_node_acc['_pt_kernel'] = _pt_kernel
    _pt_g.add_memfree_node(_pt_array_0, [_pt_kernel, _pt_kernel_0])
    _pt_g.add_memfree_node(_pt_array, [_pt_kernel])
    return (_pt_g.get_exec_graph(), _pt_g, _pt_node_acc, _pt_buffer_acc)

def f(allocator=_pt_drv.mem_alloc):
    _pt_result = _pt_gpuarray.GPUArray((100,), dtype='float64', allocator=allocator)
    (_pt_exec_g, _pt_g, _pt_node_acc, _pt_buffer_acc) = exec_graph_builder()
    _pt_exec_g.set_kernel_node_params(_pt_buffer_acc['_pt_array_0'], kernel_node=_pt_node_acc['_pt_kernel_0
      '])
    _pt_exec_g.set_kernel_node_params(_pt_result.gpudata, _pt_buffer_acc['_pt_array_0'], kernel_node=
      _pt_node_acc['_pt_kernel'])
    _pt_exec_g.launch()
    _pt_tmp = {'_pt_out_': _pt_result}
    return _pt_tmp
```

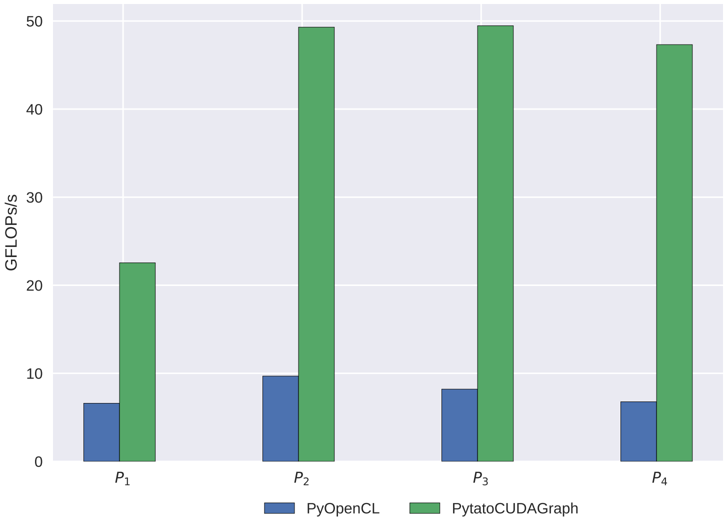
Graph dependencies

Figure: Generated *PyCUDA* code

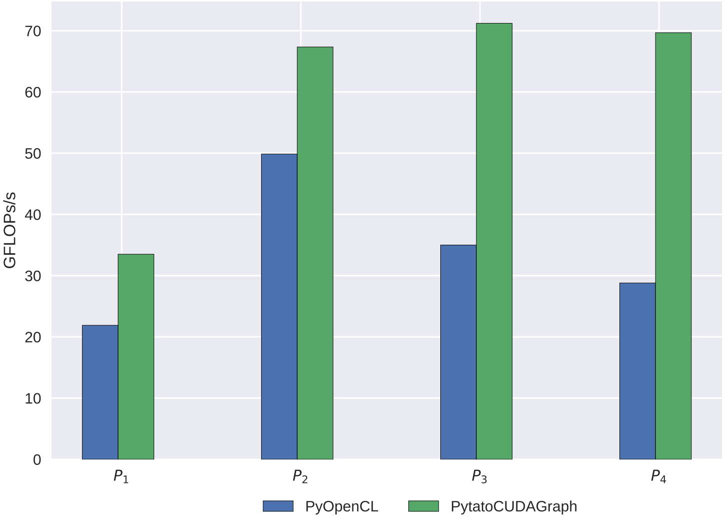
Experimental Setup

- Nvidia Titan V
  - Peak Double prec. FlOps: 6144 GFlOps/s
  - Peak bandwidth: 652.8 GB/s
- 3D Wave and Euler Operators
- $p \in \{1, 2, 3, 4\}$
- #Tetrahedrons in mesh: 10K (for lower orders)- 27K (for higher orders)
- OpenCL Implementation: PoCL-CUDA (3.0)
- Benchmarks, run instructions at github.com/mitkotak/dg_benchmarks

Wave Operator



Euler Operator



Key Takeaways

- `PytatoCUDAGraphArraycontext` abstraction can compile real-world DG-FEM operators.
  - as a drop-in replacement for array program backend.
- Observed a speedup of 3-6x for Wave and 1.5-3x for Euler.

Open Questions

- Is CUDAGraph+FusionActx profitable for MIRGE-Com?
  - Develop a performance model for *CUDAGraphs*.
  - Model peak memory usage for *CUDAGraphs*.

Future Work

- Upstream the work: Integrate with *PyCUDA*, *Pytato*, `Arraycontext`.