The University of Texas at Austin

# Parla and PyKokkos

**George Biros**, Martin Burtscher, Mattan Erez**, Milos Gligoric,** Keshav Pingali, Chris Rossbach,
Nader Al Awar, Jimmy Almgren-Bell, Hochan Lee, **Will Ruys**, Yineng Yan, Bozhi You

ODEN INSTITUTE

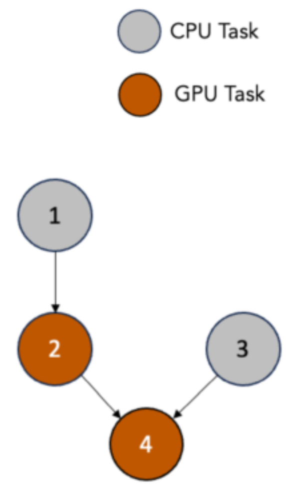
PECOS Predictive Engineering & Computational Science

https://pecos.oden.utexas.edu

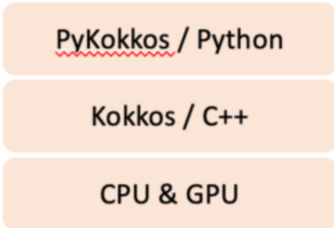
Director: Bob Moser

PSAAP III

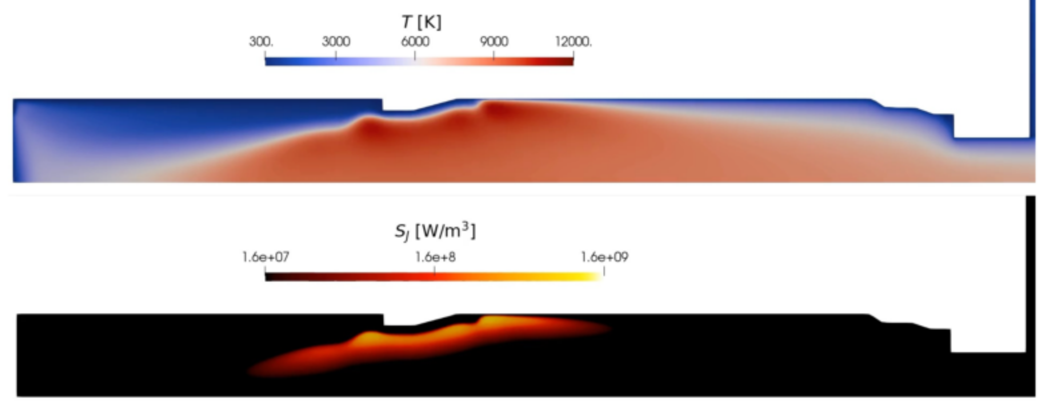
# Parla and PyKokkos for HPC in Python




- Parla: Single Node, multidevice task-based runtime
  - https://github.com/ut-parla
  - "Device": CPU cores, GPU, other accelerators
  - Kernels : Numba, Raw, PyKokkos, PyCuda
  - Single process multithreaded runtime
- PyKokkos: Kokkos API for single-device kernels
  - https://github.com/kokkos/pykokkos
  - Single device kernels / performance portable
  - JIT Python to C++
- Interoperability
  - CUDA, HIP, SYCL, OpenMP, MPI, Python, C++



# Torch/CS integration




**Python main**
**MFEM / PyMFEM**
Time stepper (operator split)
Multiphysics

Flow/Transport - MFEM

Boltzmann
Radiation

Parla
mpi4py

Maxwell - MFEM

**Computational kernels**
CPUs/GPUs

- EB-PIC: P2C/collisions/advection
- EB-PDE: collisions/advection
- Mesh-to-mesh projections
- RTE: Parallel sweeps

Python: PyKokkos / CuPY / NumPy

C++: Kokkos
Cython or PyBind11

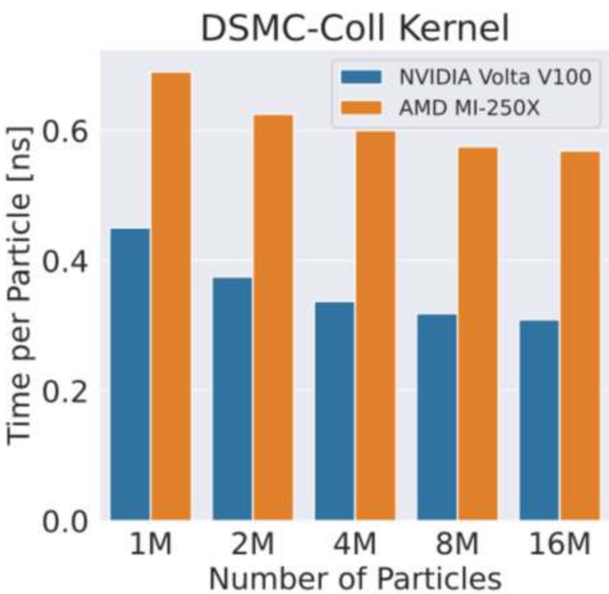
Vendor libraries
Kokkos kernels
VECs

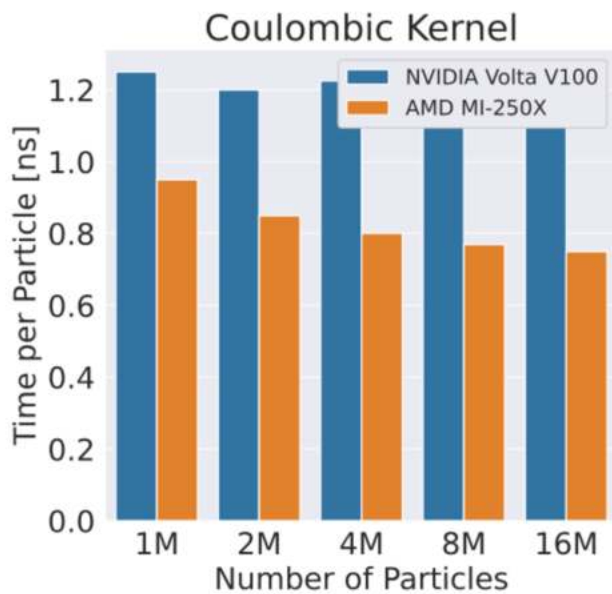
# PyKokkos Example

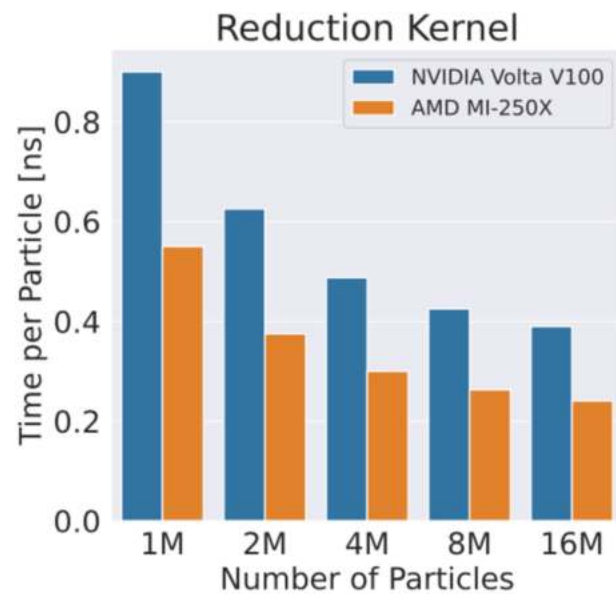
```python
import pykokkos as pk

@pk.workunit
def y_init(i, y_view):
    y_view[i] = 1

@pk.workunit
def matrix_init(j, cols, A_view):
    for i in range(cols):
        A_view[j * cols + i] = 1

@pk.workunit
def yAx(j, acc, cols, y_view, x_view, A_view):
    temp2: float = 0
    for i in range(cols):
        temp2 += A_view[j * cols + i] * x_view[i]
    acc += y_view[j] * temp2

def main():
    y = pk.View([N], pk.double)
    x = pk.View([M], pk.double)
    A = pk.View([N * M], pk.double)
    p = pk.RangePolicy(0, N)
    pk.parallel_for(p, y_init, y_view=y)
    pk.parallel_for(pk.RangePolicy(0, M), y_init, y_view=x)
    pk.parallel_for(p, matrix_init, cols=M, A_view=A)
    result = pk.parallel_reduce(p, yAx, cols=M, y_view=y, x_view=x, A_view=A)
```

# V100 vs MI-250X

**Complexity: linear on (N)**
405 FLOPs + 144 MOPS
Roofline: 0.2 ns / particle
Observed: 0.4 ns / particle

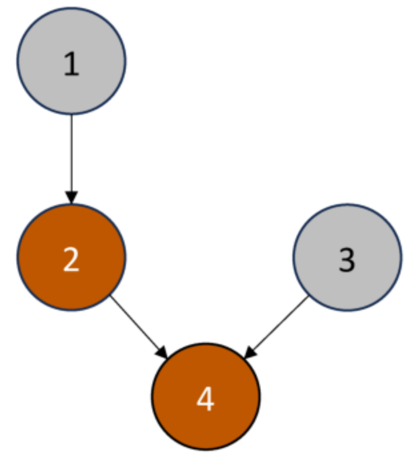


# Parla Overview

- Python library for task-parallel programming:
  - Heterogenous tasks
    - GPU + CPU devices
    - Specialized function variants
  - Data movement w/ Parla arrays
    - Prefetching / coherence / task mapping policy
  - Hardware queue-aware dependency resolution

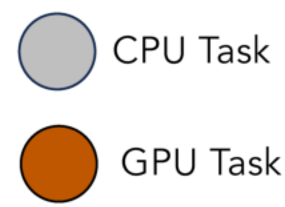



- A task is a unit of work with precedence & resource constraints

Constraints

Task Body

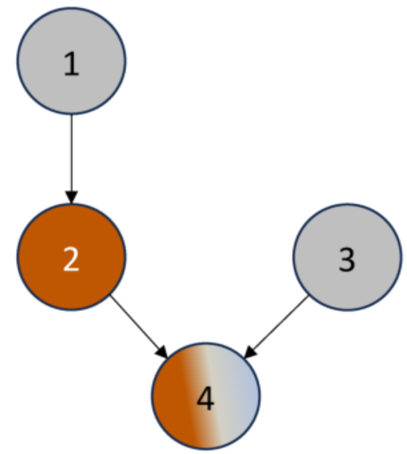
```
@spawn(T[4],
       dependencies=T[2:4],
       placement=[gpu])
def task():
    f()
```




- Mapping refers to choosing a specific device to execute on (e.g., GPU[0])

# Parla Overview

- Python library for task-parallel programming:
  - Heterogenous tasks
    - GPU + CPU devices
    - Specialized function variants
  - Data movement w/ Parla arrays
    - Prefetching / coherence / task mapping policy
  - Hardware queue-aware dependency resolution



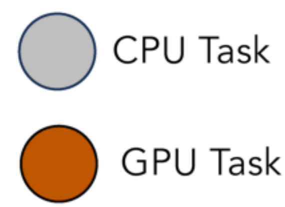

```python
 @specialize
def f():
   cpu_kernel()

@f.variant(gpu)
def f_gpu():
   gpu_kernel()
```

If CPU

If GPU

Runs on CPU or GPU

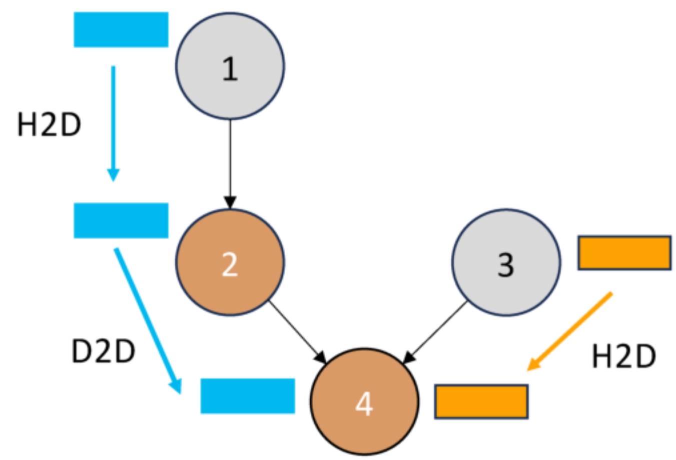
```python
@spawn(T[4],
       dependencies=T[2:4],
       placement=[gpu, cpu])
def task():
    f()
```



# Parla Overview

```
blue=parla.asarray(np.zeros(N))
orange=parla.asarray(np.zeros(M))
```

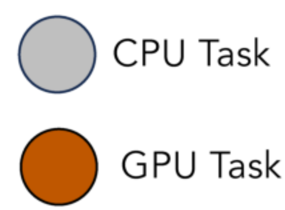
- Python library for task-parallel programming:
  - Heterogenous tasks
    - GPU + CPU devices
    - Specialized function variants
  - Data movement w/ Parla arrays
    - Prefetching / coherence / task mapping policy
  - Hardware queue-aware dependency resolution




Dataflow constraints

Moved to device before execution

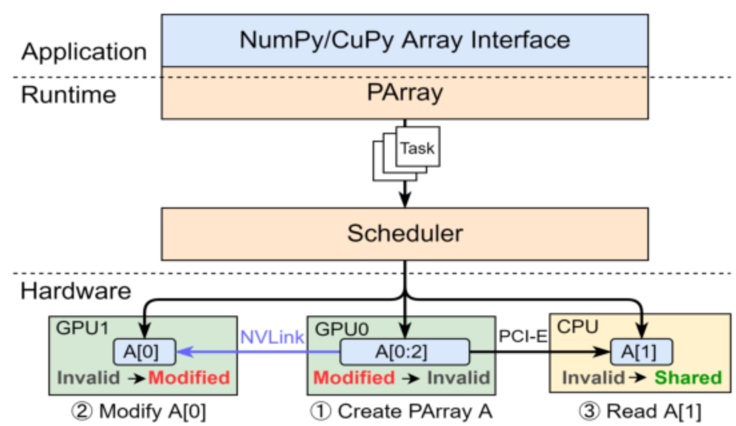
```
@spawn(T[4],
       dependencies=T[2:4],
       placement=[gpu],
       inout=[blue, orange]

)
def task():
    f(blue, orange)
```



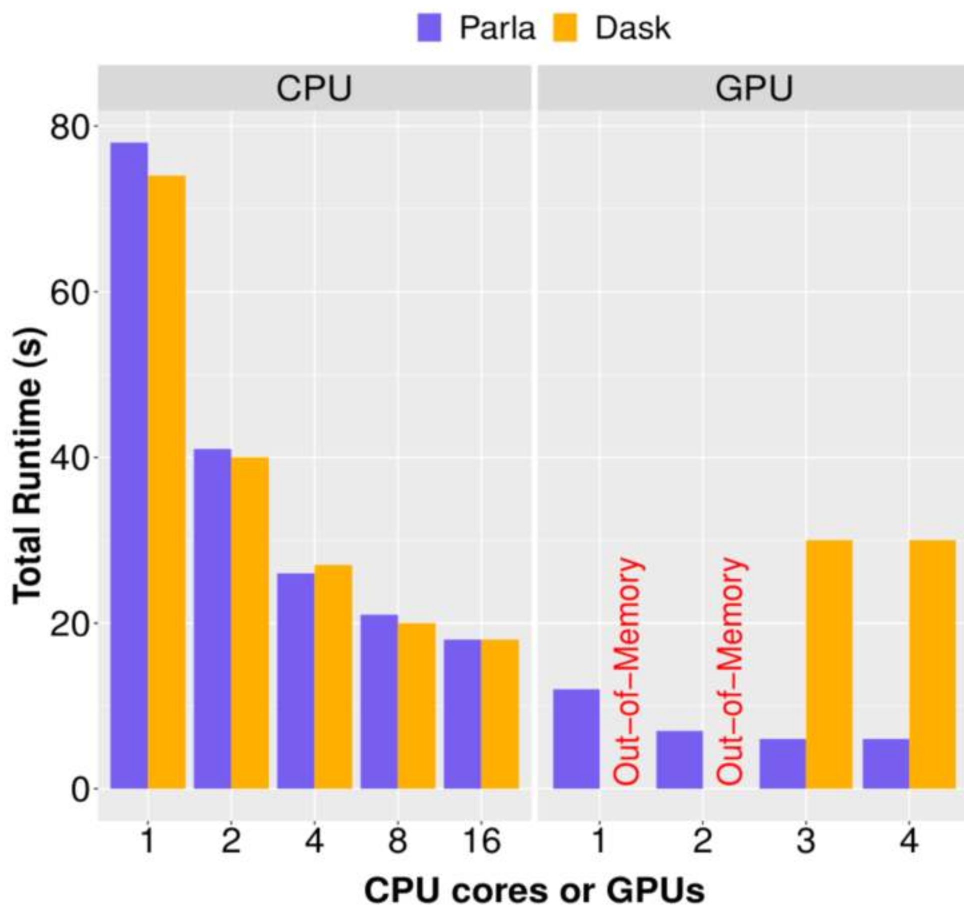
# Parla API example: PArrays

```
IND = TaskSpace("Independent Tasks")
for i in range(num_gpu):
  s = slice(i * block_size, (i + 1) * block_size)
  @spawn(IND[i], placement=gpu(i), input=[a[s], b[s]])
  def inner_product():
       partial_sums[i] =  a[s] @ b[s]

@spawn(reduce,
       dependencies=IND,
       placement=cpu)
def reduce():
  result = np.sum(partial_sums)
await result
```

- Auto data movement
  - PArrays to wrap ndarrays Specify ins and outs
  - Parla takes care of the rest
    - Copies and coherence,
    - Scheduled data movement
    - Locality-aware scheduling

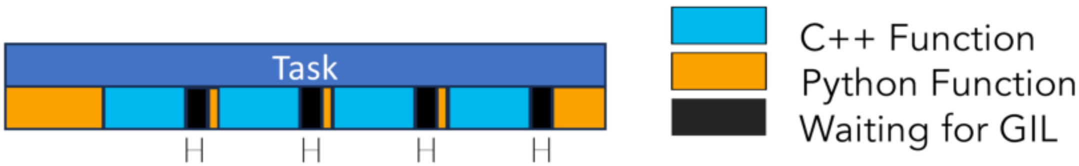


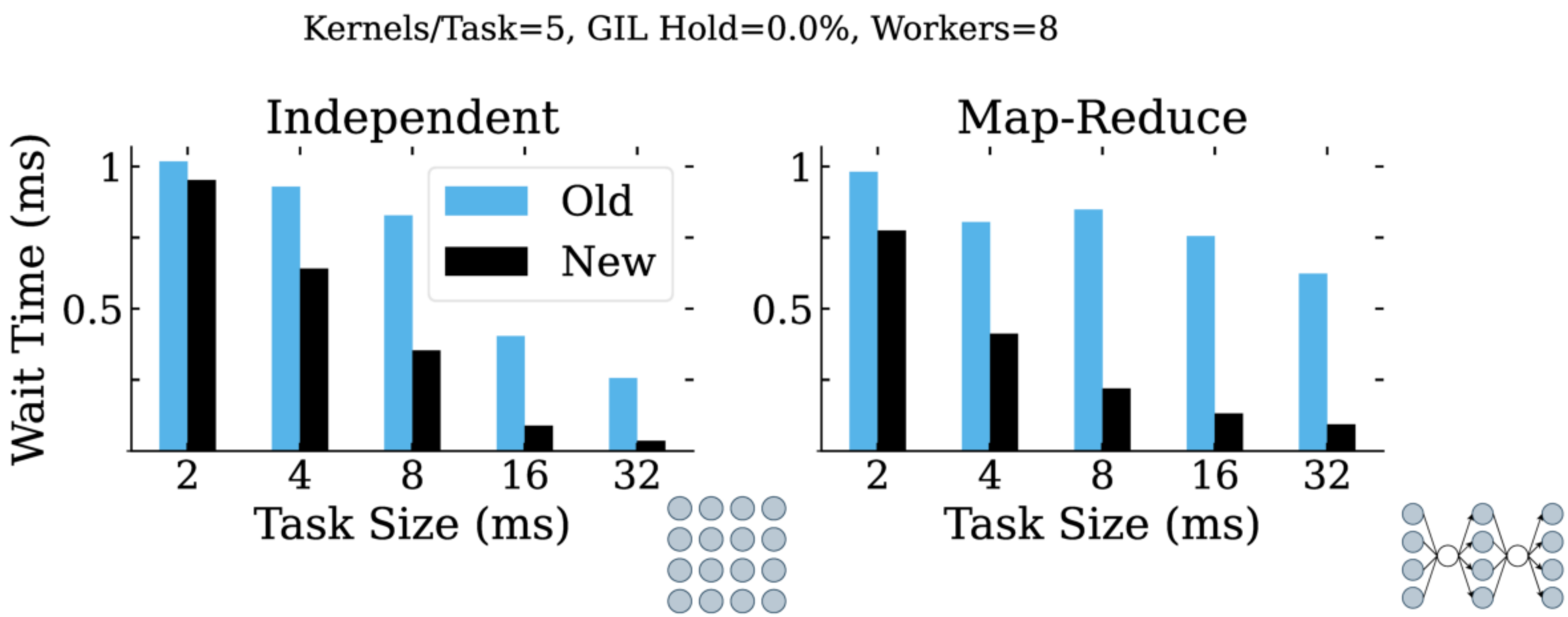
# Good performance and scalability (vs. Dask)



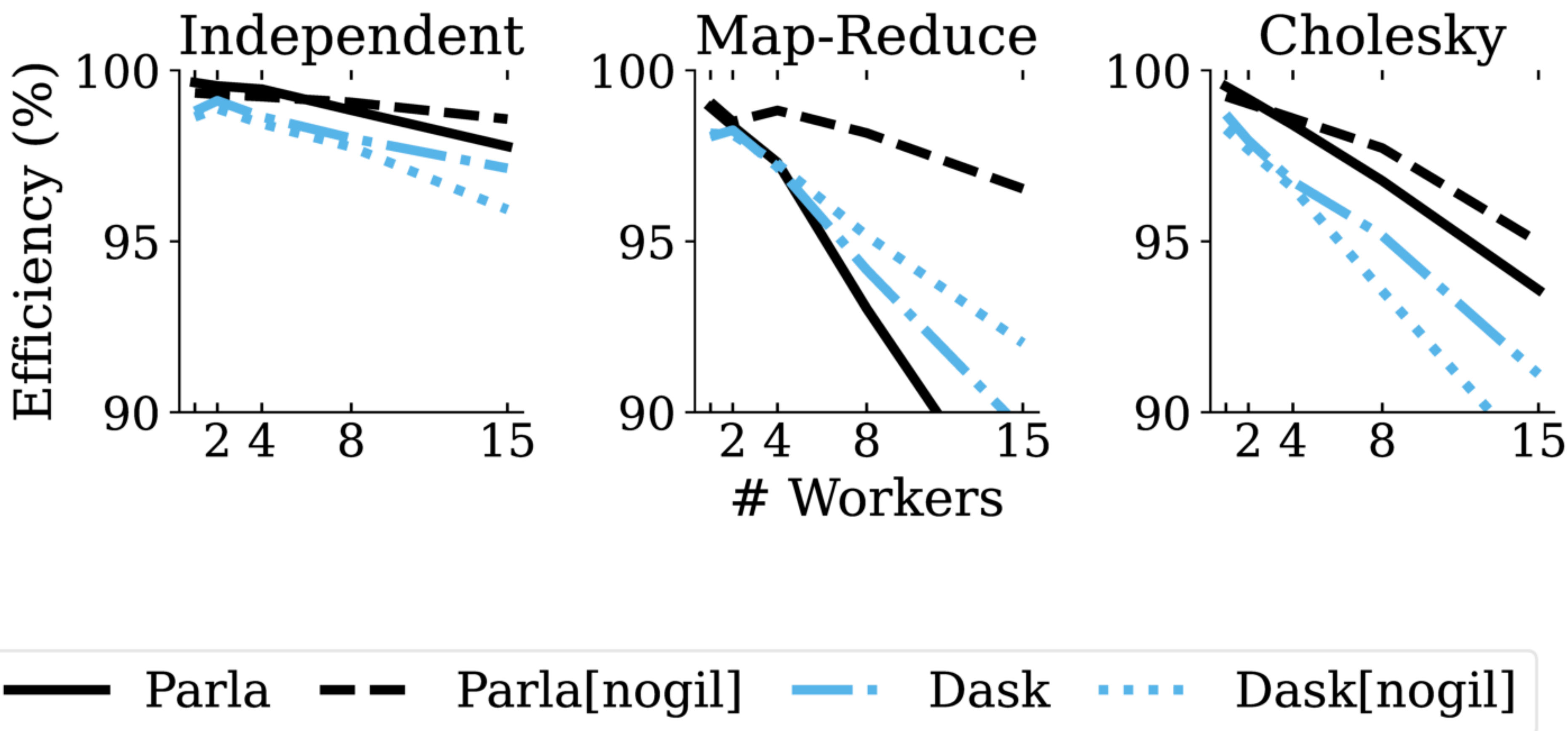
# Reduction in GIL-contention

New runtime has less time waiting for GIL

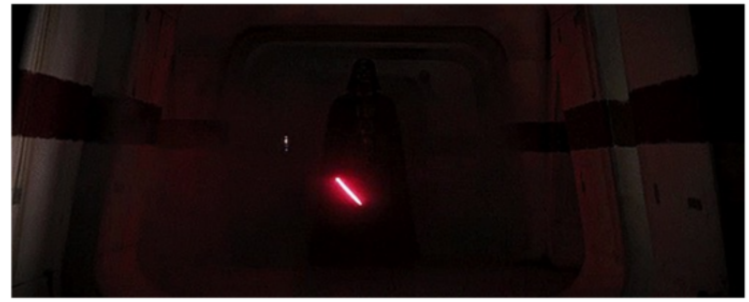


The new runtime shows an advantage even in a proposed "No GIL" Python (PEP703)



# Thank you



- PyKokkos: Nader Al Awar, Muhammad Hannan Naeem
- Parla: Will Ruys, Hochan Lee, Yineng Yan, Bozhi You

https://dl.acm.org/doi/abs/10.1145/3447818.3460376

https://ieeexplore.ieee.org/abstract/document/10046101