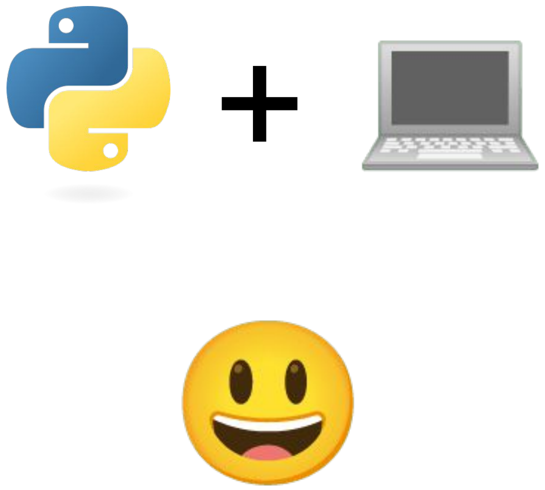
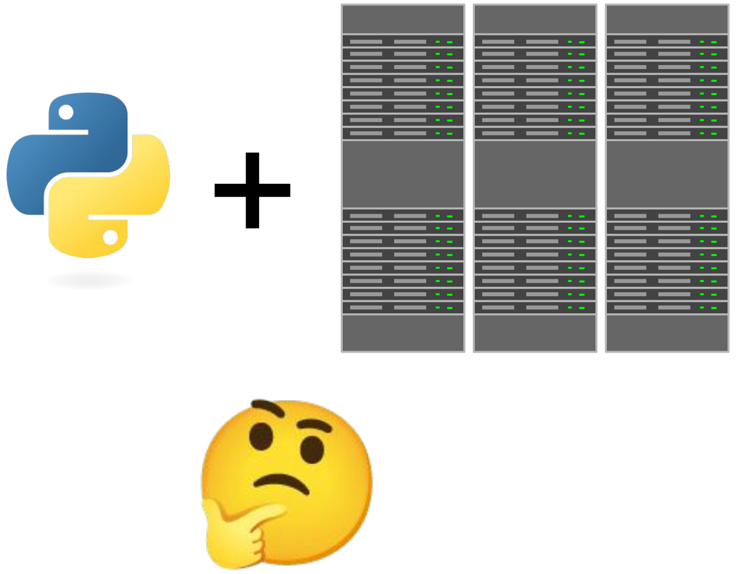
Distributed Python at NERSC

Daniel Margala
danielmargala@lbl.gov

NERSC Data Day
2024-02-21

Distributed Python at NERSC?





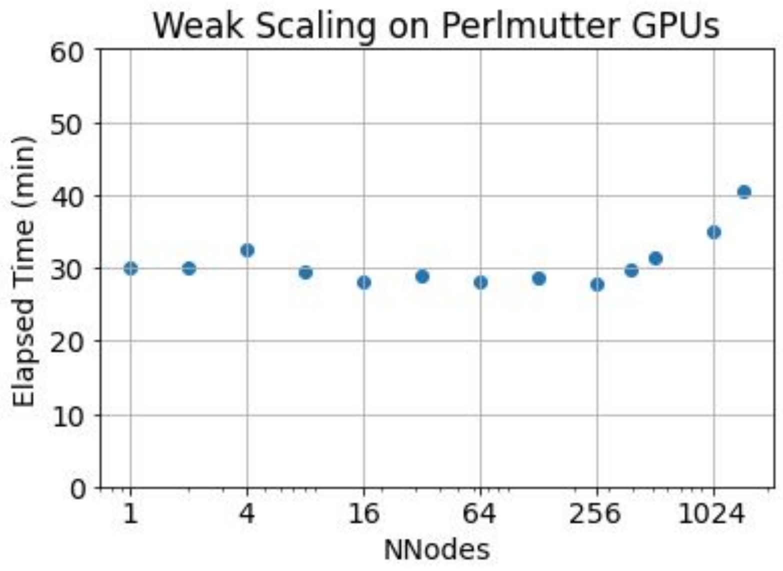
Distributed Python at NERSC

- MPI (Message Passing Interface) is the predominant distributed memory programming model in HPC

- Hybrid approach “MPI+X” is commonly used with MPI for scaling out across nodes and “X” for parallelization within a node.

- Python applications can follow this pattern…
  but there are other options too! 🎉

mpi4py + numpy/cupy



Distributed Python at NERSC*

*not a complete list

mpi4py

HPC 💪

dask

❤'s pydata stack

DragonHPC

cuNumeric (legate)

Developers are interested in working with users to integrate these into their work.

TensorFlow

RAY

PyTorch

HOROVOD

^ not discussed in this talk

mpi4py

- mpi4py provides MPI bindings for Python applications
- MPI defines a standard set of functions that facilitate communication between processes:
  - point-to-point
  - collectives
  - non-blocking
  - one-sided
  - and more…
- References:
  - https://mpi4py.readthedocs.io/en/stable
  - https://docs.nersc.gov/development/languages/python/parallel-python/#mpi4py

mpi4py at NERSC

setup:

Install with PrgEnv compiler wrappers to link with cray-mpich

```
module load conda
conda create -p $ENV_PATH python numpy
conda activate $ENV_PATH
MPICC="cc -shared" pip install --force \
 --no-cache-dir --no-binary=mpi4py mpi4py
```

run:

```
module load conda
conda activate $ENV_PATH
srun -n 4 python example.py
```

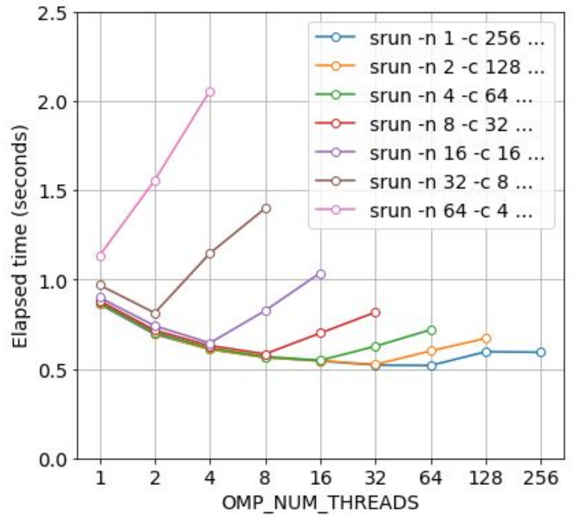
code:

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
    data = np.arange(100, dtype='i')
else:
    data = np.empty(100, dtype='i')
comm.Bcast(data, root=0)
for i in range(100):
    assert data[i] == i
```

MPI + X: Python edition

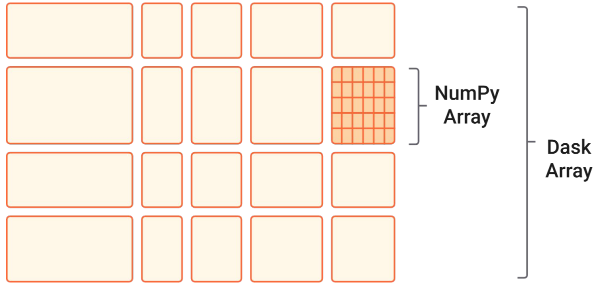
- mp4py + X examples:
  - multiprocessing
    - use MPI to run independent workloads on different nodes
  - numpy
    - numpy BLAS backends such as OpenBLAS or MKL may use multiple threads
  - cupy
    - CuPy for GPU-accelerated NumPy / SciPy
  - mpi4py
    - MPI for parallelization within a node as well
    - e.g. 1 rank per core or GPU, will vary by application

example exploration of perf tradeoff of MPI tasks vs threads



dask

- dask is a Python library for parallel and distributed computing
- APIs:
  - Array: subset of NumPy ndarray API
  - DataFrame: parallelized pandas
  - Futures: extends Python’s concurrent.futures
  - Bag: map, filter, fold, groupby, …
- References:
  - https://docs.dask.org/en/stable/
  - https://examples.dask.org/
  - https://docs.nersc.gov/analytics/dask/
  - https://gitlab.com/NERSC/nersc-notebooks/-/tree/main/perlmutter/dask



dask at NERSC

Launch scheduler and workers:

```
module load conda
conda activate daskenv

scheduler_file=$SCRATCH/scheduler_file.json
rm -f $scheduler_file

# launch scheduler
dask-scheduler --scheduler-file $scheduler_file \
      --interface hsn0 &

dask_pid=$!
sleep 5
until [ -f $scheduler_file ]
do
     sleep 5
done

# launch workers
srun dask-worker --scheduler-file $scheduler_file \
    --interface hsn0 --nworkers 1
```

Connect a client to scheduler:

```
import os
import dask
from dask.distributed import Client

scheduler_file = os.path.join(
      os.environ["SCRATCH"],
      "scheduler_file.json"
)

client = Client(scheduler_file=scheduler_file)
```

Full example is here:
https://gitlab.com/NERSC/nersc-notebooks/-/tree/main/perlmutter/dask
Checkout upcoming dask training:
TBD

cuNumeric (legate)

- cuNumeric aims to provide a distributed and accelerated drop-in replacement for the NumPy API
  - implicit data distribution and parallelization through Legate
  - similar APIs on top of Legate are in the works (legate.pandas, legate.sparse, …)
  - profiling support with NVIDIA Nsight Systems
  - dataflow diagrams for debugging
  -
- "One program for any scale machine"
- References:
  - https://developer.nvidia.com/blog/accelerating-python-applications-with-cunumeric-and-legate/
  - https://github.com/nv-legate/legate.core
  - https://github.com/nv-legate/cunumeric

cuNumeric at NERSC (setup)

```
export ACCOUNT=m1234
export PREFIX=$SCRATCH/legate
mkdir -p $PREFIX
cd $PREFIX

git clone https://github.com/nv-legate/quickstart.git
git clone https://github.com/nv-legate/legate.core.git
git clone https://github.com/nv-legate/cunumeric.git

# build on an interactive gpu node
salloc -A $ACCOUNT -C gpu -N 1 -q interactive -t 30

# Create conda environment w/ dependencies
module load conda
export CONDA_PKGS_DIRS=$(mktemp -d)
export CONDA_PREFIX=$PREFIX/env
conda env create -p $PREFIX \
     -f environment-test-linux-py3.11-cuda12.2.2.yaml
conda activate $CONDA_PREFIX
```

```
# Install Legate packages
module load cray-pmi
module unload cray-libsci
conda uninstall pkg-config

cd legate.core
../quickstart/build.sh
cd ..
cd cunumeric
../quickstart/build.sh
```

The quickstart repo has useful helper scripts and configurations for various HPC platforms including perlmutter
https://github.com/nv-legate/quickstart

cuNumeric at NERSC (run)

Many more examples in the cuNumeric repo:
https://github.com/nv-legate/cunumeric

Multi-node launch using quickstart helper:

```
export ACCOUNT=m1234
export PREFIX=$SCRATCH/legate
export CONDA_PREFIX=$PREFIX/env

cd $PREFIX/cunumeric

module load conda
conda activate $CONDA_PREFIX

../quickstart/run.sh 2 examples/stencil.py
```

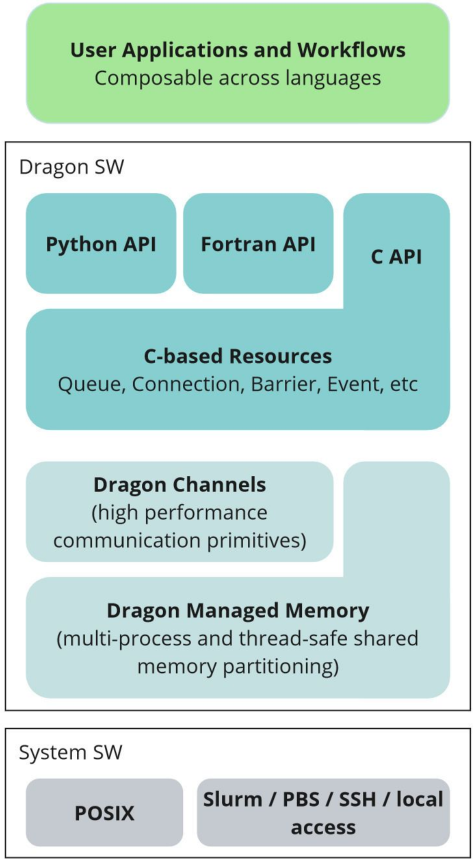
library	elapsed time (ms)	speedup rel. numpy
numpy	424112.53	1x
cupy	4885.96	87x
cunumeric	430.90	984x

multi-gpu + multi-node

```
import cunumeric as np

def initialize(N):
    grid = np.zeros((N + 2, N + 2))
    grid[:, 0] = -273.15
    grid[:, -1] = -273.15
    grid[-1, :] = -273.15
    grid[0, :] = 40.0
    return grid

def run_stencil(N, I):
    grid = initialize(N)

    center = grid[1:-1, 1:-1]
    north = grid[0:-2, 1:-1]
    east = grid[1:-1, 2:]
    west = grid[1:-1, 0:-2]
    south = grid[2:, 1:-1]

    for i in range(I):
        average = center + north + east + west + south
        work = 0.2 * average
        center[:] = work

run_stencil(20000, 100)
```

DragonHPC

- Dragon is a distributed run-time for HPC applications and workflows
  - Python multiprocessing program across nodes
  - Interface and adapters for workflows
  - Distributed key-value store
  - Telemetry / introspection
  - Scalable data loaders
- References:
  - http://dragonhpc.org/
  - https://github.com/DragonHPC/dragon


User Applications and Workflows
Composable across languages
Dragon SW
Python API
Fortran API
C API
C-based Resources
Queue, Connection, Barrier, Event, etc
Dragon Channels
(high performance communication primitives)
Dragon Managed Memory
(multi-process and thread-safe shared memory partitioning)
System SW
POSIX
Slurm / PBS / SSH / local access

DragonHPC at NERSC

Many more examples in the DragonHPC repo:
https://github.com/DragonHPC/dragon

setup*:

```
mkdir -p $SCRATCH/dragonhpc

git clone https://github.com/DragonHPC/dragon.git
cd dragon

# edit "hack/clean_build":
#     remove "-c src/constraints.txt"

# edit "src/lmod/dragon-dev.lua":
#     change rome -> milan
#     change cray-python -> cray-python/3.9

# build on an interactive cpu node
salloc -C cpu -N 1 -t 30 -q interactive -A m1234

# create venv and build from source
source hack/clean_build
```

*A prebuilt wheel is also available from the github repo release page

run:

```
salloc -C cpu -N 2 -t 30 -q interactive -A m1234

source hack/setup

dragon example.py
```

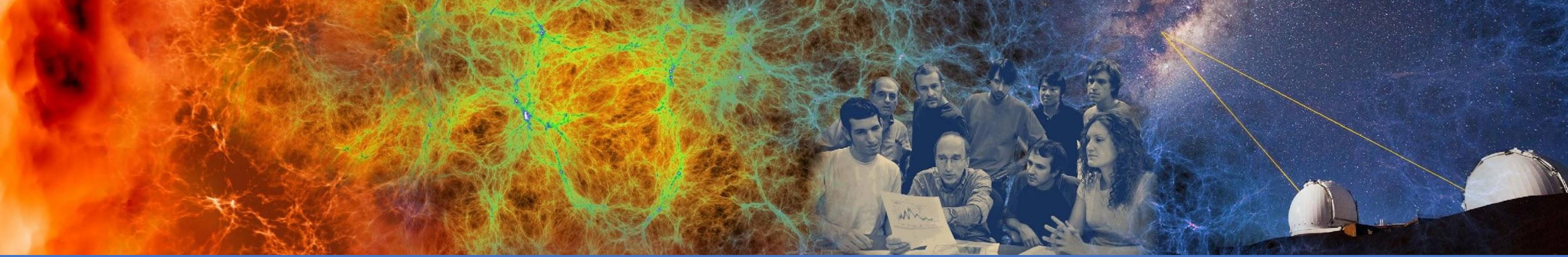
code:

```
import dragon
import multiprocessing as mp
...
if __name__ == "__main__":
    mp.set_start_method("dragon")
    cpu_count = mp.cpu_count()
    with mp.Pool(cpu_count) as pool:
        result = pool.map(...)
...
```

Summary

Framework	🙂	🤔
mpi4py	• HPC Stalwart • High speed network	• Learning curve for Python users not familiar with HPC
dask	• Interoperable with PyData ecosystem • Many APIs • Fun dashboard	• May seem a bit clunky on HPC • Does not leverage high speed network (no libfabric support currently)
cunumeric (legate)	• One program for any scale machine • Built-in support for GPUs • High speed network • Developers interested in engaging with users	• NumPy API coverage is WIP • May be challenging to compose with non-legate libraries • May be challenging to debug issues / performance
dragon	• Distributed multiprocessing • Workflow adapters and lower level core • Developers interested in engaging with users	• Not sure about support for high speed network (?) • May be challenging to debug issues / performance

Please reach out if you have questions about distributed Python:
danielmargala@lbl.gov



Thank you

danielmargala@lbl.gov

HPC Python housekeeping

- python startup can be file system intensive
  - python needs to access lots of tiny files. typically, way more than a traditional HPC program compiled to a binary executable
  - best practice is to **use a container** or **/global/common/software/<project>** for software environments (esp. Python!)
- composing multiple methods parallelism can lead to "oversubscription".
  - For example, NumPy BLAS backend and multiprocessing.cpu_count() both default to MAX_THREADS.
- distributed Python frameworks will need to work with (or around) the scheduler (Slurm).