This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

SAND2022-6923C

Sandia National Laboratories

Machine Learning for CUDA+MPI Design Rules

Carl Pearson, Karen Devine, Aurya Javeed

Sandia National Labs

PDSEC 2022

This work is supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program through the FASTMath Institute.
This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231 using NERSC award ERCAP0019623.

FASTMATH

U.S. DEPARTMENT OF ENERGY

NNSA

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

SAND_XXX

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Automatic Discovery of Implementation Rules for Fast GPU + MPI Operations

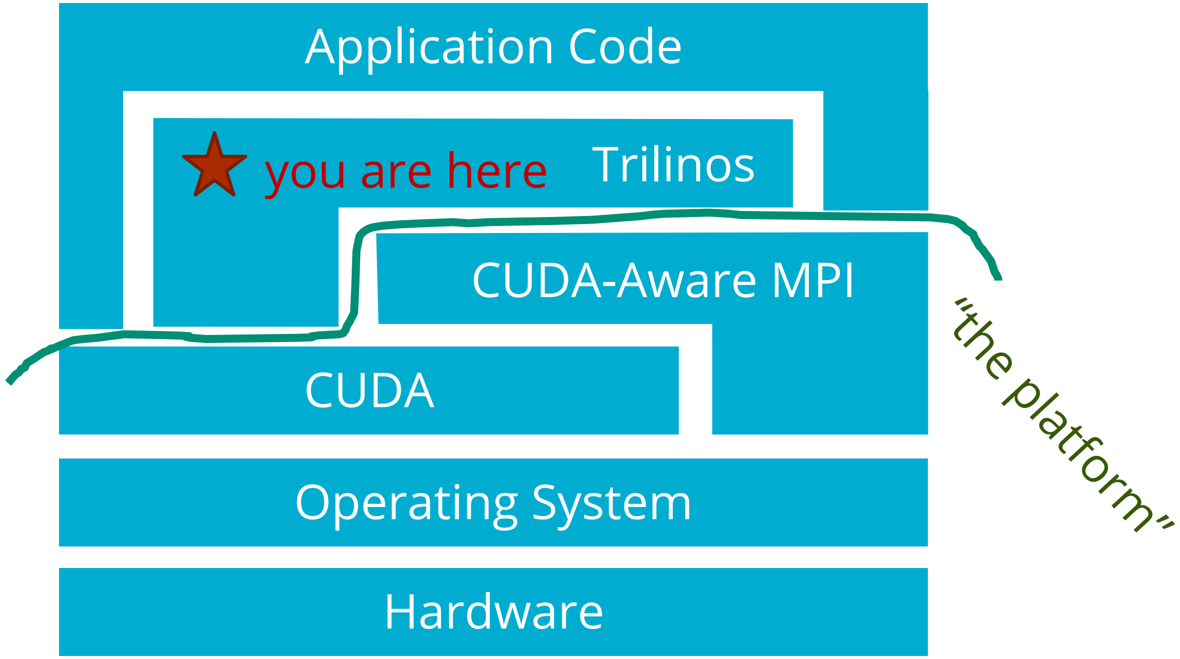
- Fast libraries for heterogeneous architectures
  - Mapping computation onto processors
  - Choosing communication strategy
  - Unpredictable performance interaction
- Prototype automatic tooling for discovering important design decisions
  - Reduced developer effort for performance on new systems
  - Maintain human provenance of library design
  - e.g. Modernize Tpetra MPI+GPU distributed linear algebra operations

Key Challenge	How it's Done
Large Design Space	• Express operation as a directed acyclic graph (DAG) of operations • Monte-Carlo Tree Search (MCTS) to identify and explore regions of interest
Extract performance insight	• Empirical benchmarking • Feature vector for each implementation • Decision tree training for design rules

Initial results pass “sniff test,” working on broader experiments and quantitative evaluation

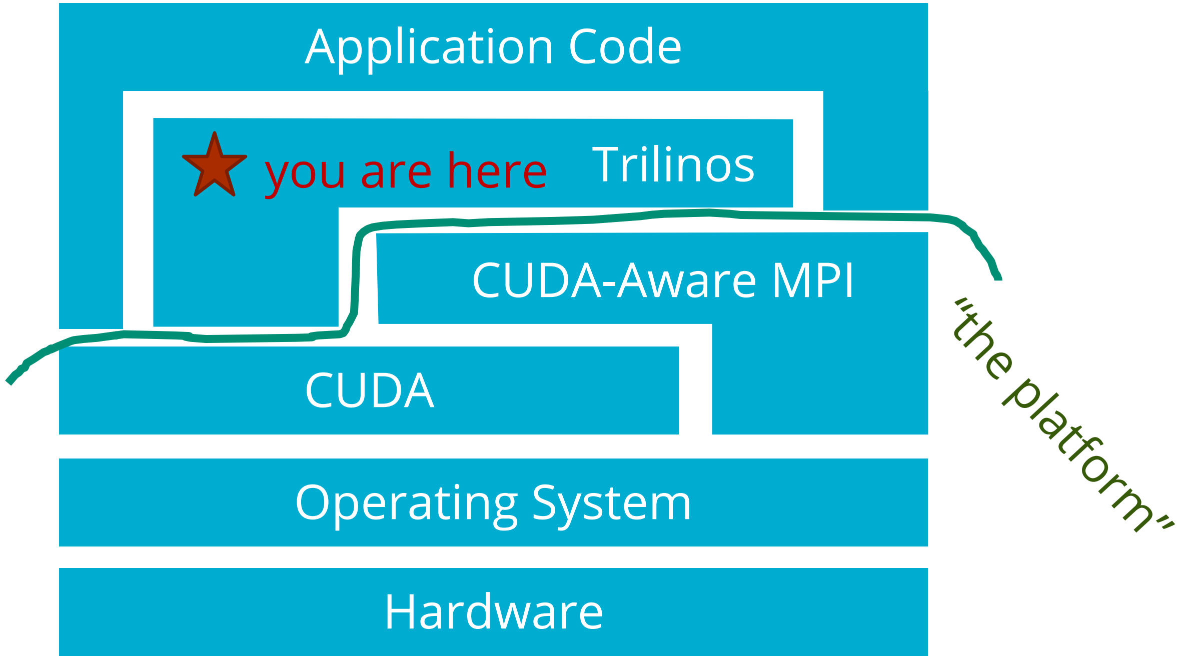
Libraries are built on existing lower-level primitives

- Our libraries (and applications) are combinations of existing library and vendor operations
  - and code to coordinate them
  - and code to implement custom behavior

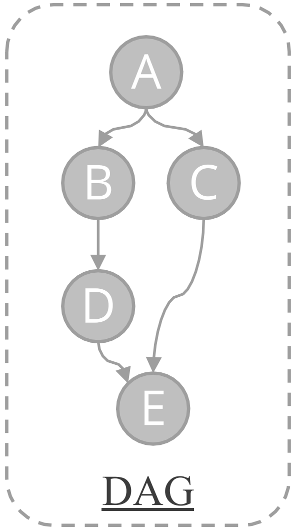


Libraries are built on existing lower-level primitives

- Our libraries (and applications) are combinations of existing library and vendor operations
  - and code to coordinate them
  - and code to implement custom behavior
- Performance changes at many layers for new platforms
  - new hardware,
  - new CUDA version,
  - new OS version,
  - etc.



Prototype Implementation in C++ and Python

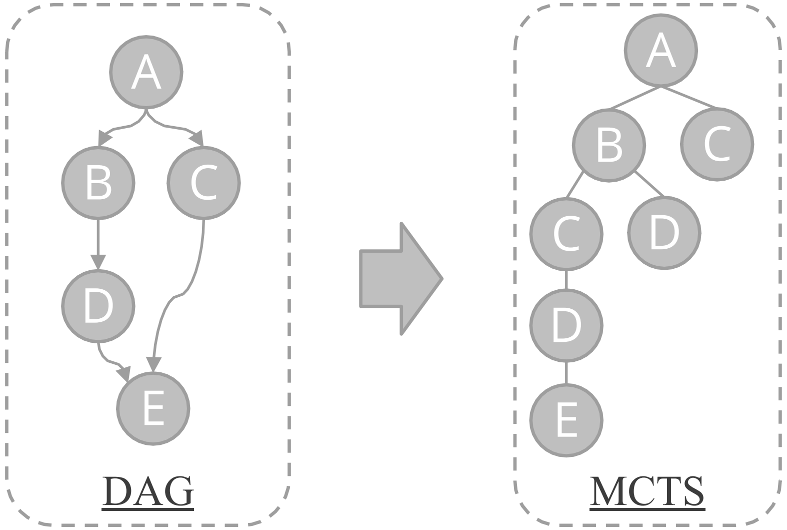


DAG of operations describes design space

C++ / CUDA / MPI

Python / scikit-learn

Prototype Implementation in C++ and Python



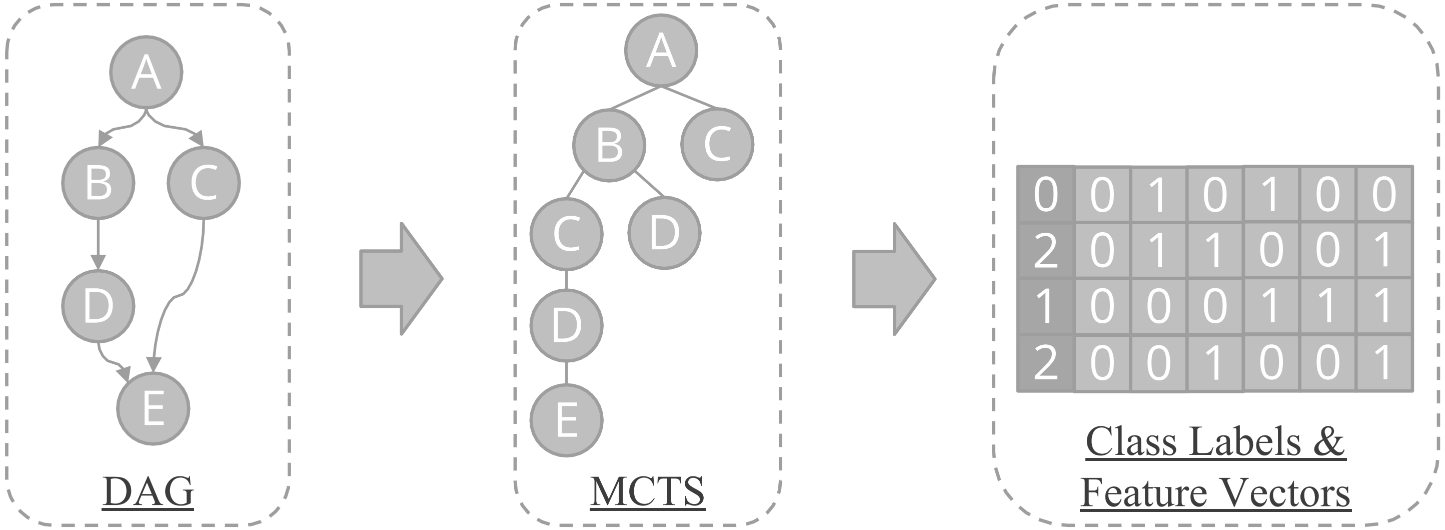
DAG of operations describes design space

MCTS searches order of operations and resource assignment

C++ / CUDA / MPI

Python / scikit-learn

Prototype Implementation in C++ and Python



0	0	1	0	1	0	0
2	0	1	1	0	0	1
1	0	0	0	1	1	1
2	0	0	1	0	0	1

DAG of operations describes design space

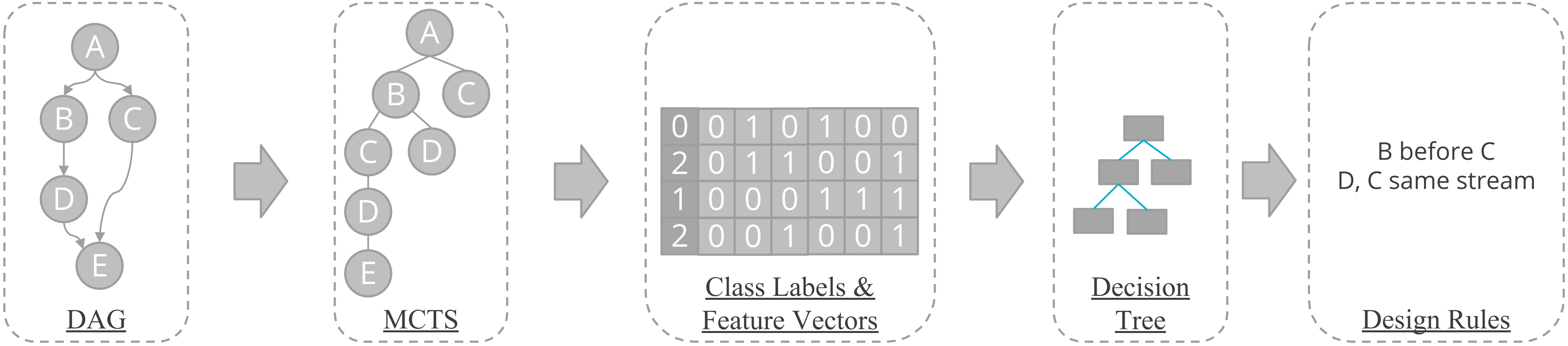
MCTS searches order of operations and resource assignment

Sequence-to-vector transformation for labels

C++ / CUDA / MPI

Python / scikit-learn

Prototype Implementation in C++ and Python



0	0	1	0	1	0	0
2	0	1	1	0	0	1
1	0	0	0	1	1	1
2	0	0	1	0	0	1

B before C
D, C same stream

DAG of operations describes design space

MCTS searches order of operations and resource assignment

Sequence-to-vector transformation for labels

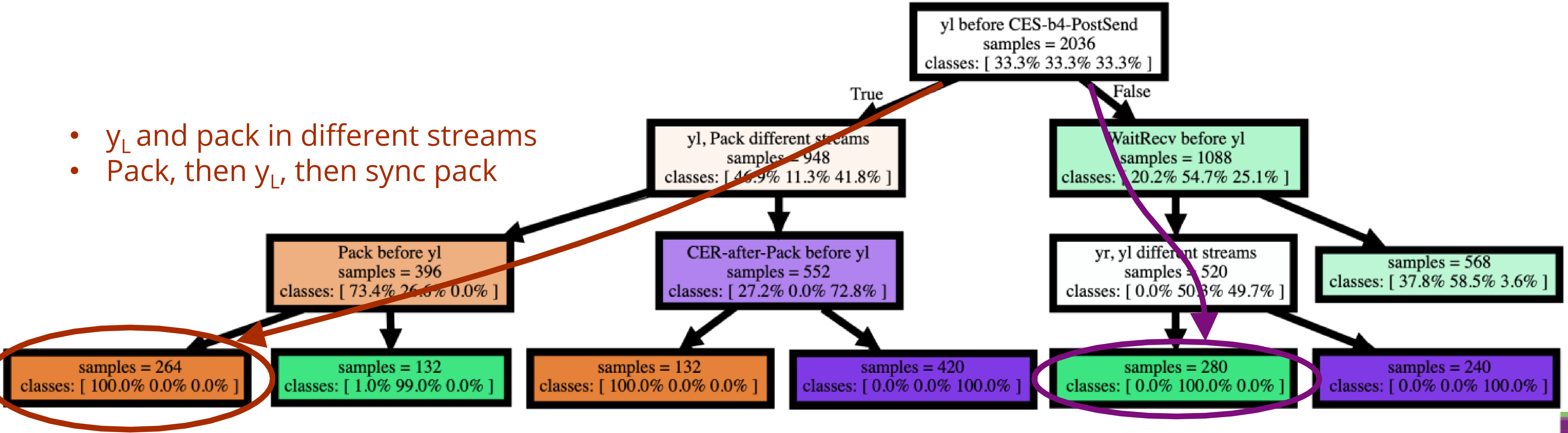
Decision tree training to identify discriminating design space features

C++ / CUDA / MPI

Python / scikit-learn

Decision Tree Training to Determine which Rules Discriminate between Classes

- y_L and pack in different streams
- Pack, then y_L, then sync pack



yl before CES-b4-PostSend
samples = 2036
classes: [33.3% 33.3% 33.3%]

True

False

yl, Pack different streams
samples = 948
classes: [46.9% 11.3% 41.8%]

WaitRecv before yl
samples = 1088
classes: [20.2% 54.7% 25.1%]

Pack before yl
samples = 396
classes: [73.4% 26.6% 0.0%]

CER-after-Pack before yl
samples = 552
classes: [27.2% 0.0% 72.8%]

yr, yl different streams
samples = 520
classes: [0.0% 50.3% 49.7%]

samples = 568
classes: [37.8% 58.5% 3.6%]

samples = 264
classes: [100.0% 0.0% 0.0%]

samples = 132
classes: [1.0% 99.0% 0.0%]

samples = 132
classes: [100.0% 0.0% 0.0%]

samples = 420
classes: [0.0% 0.0% 100.0%]

samples = 280
classes: [0.0% 100.0% 0.0%]

samples = 240
classes: [0.0% 0.0% 100.0%]

- sync pack before y_L
- WaitRecv before y_L
- $y_{L,}$ y_R in same stream

Each path through the tree is a set of design rules that define a performance class

Vision for this work

- Current
  - C++ MCTS implementation for MPI/CUDA codes with multiple streams
  - Prototype feature-vector and decision tree training using SciKit in Python
  - Available at github.com/sandialabs/tenzing
- Upcoming
  - Applying initial results to Tpetra distributed linear algebra package in Trilinos
- Future Explorations
  - Identify unexpected performance effects on target platforms (“performance bugs”)
  - What to do as communication / computation are more tightly integrated
- Summary
  - Represent CUDA+MPI operation as DAG
  - Automatically generate human-interpretable rules for library design
  - Maintain human provenance of implementation (no “black boxes”)