# Apollo: Lightweight Models for Dynamically Tuning Data-Dependent Code

**David Beckingsale, Olga Pearce, and Todd Gamblin**

Lawrence Livermore National Laboratory, Livermore, CA, USA

## Performance variability is data-dependent

- The performance of numerical physics kernels in scientific applications depends strongly on input dataset, data storage, access pattern, and work to thread mappings.
- We studied three applications: LULESH and CleverLeaf are hydrodynamics mini-applications, and ARES is a large-scale production code used for munitions modeling and inertial confinement fusion simulations.
- We see up to three orders of magnitude in the time taken to execute each kernel, depending on the input data and the way the kernel is executed.

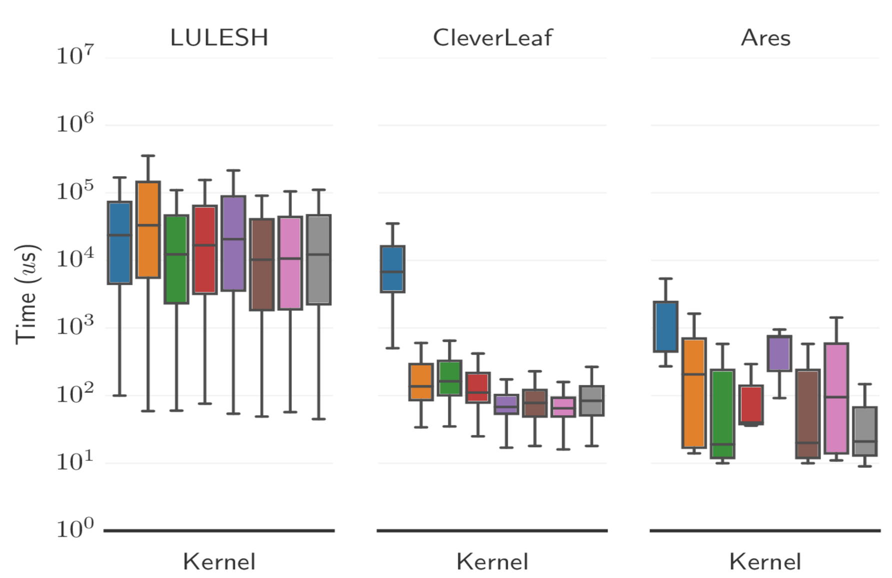



**Figure 1:** Performance variability per-kernel in LULESH, CleverLeaf and ARES

## RAJA provides a way to map applications kernels to different programming models

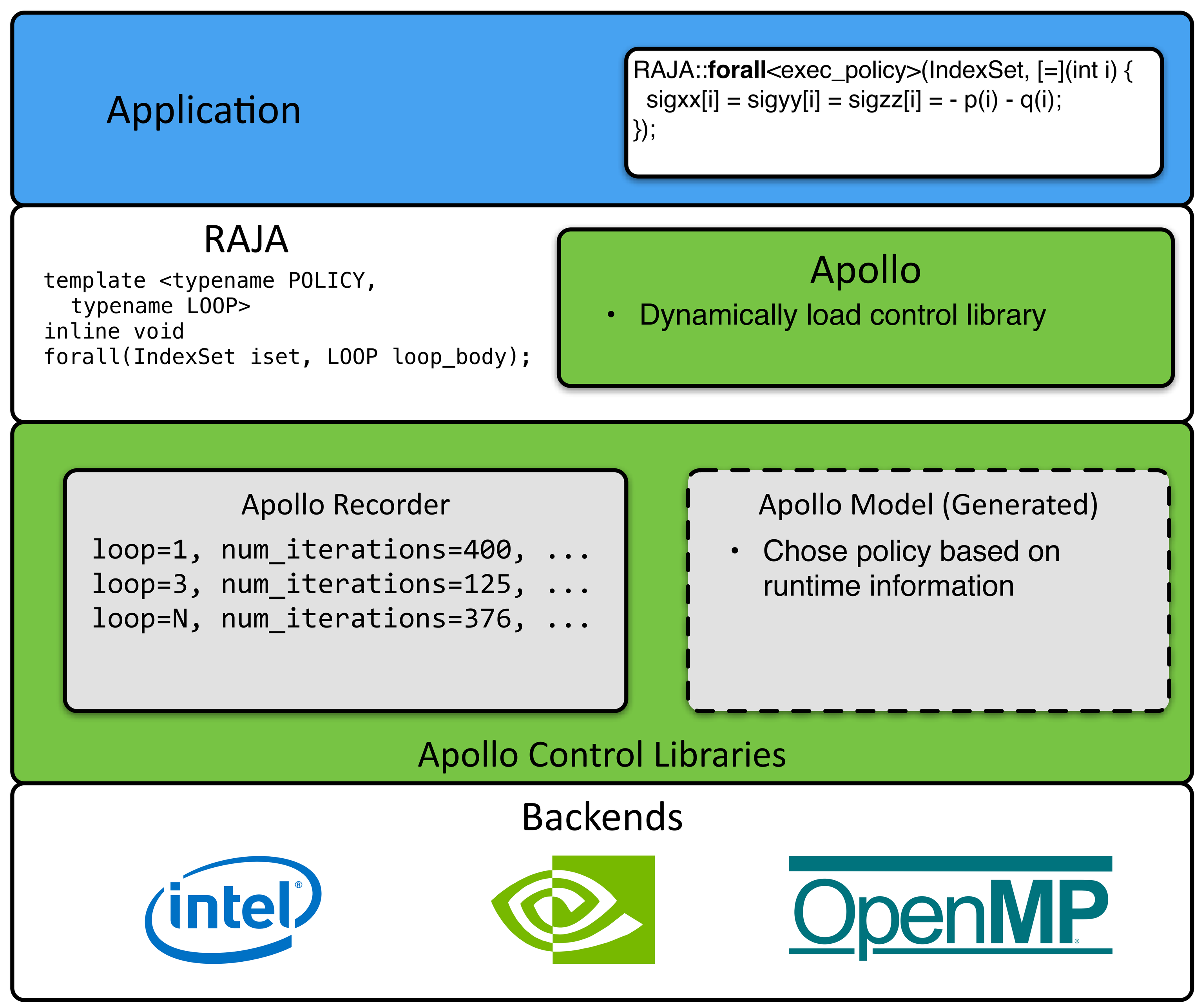
- Frameworks like RAJA (http://github.com/LLNL/RAJA) allow developers to **map loops to different programming models** independent of scientific code.

```
forall< EXECUTION_POLICY >(0, N, [=] (Index_type i){
     y[ i ] += a * x[ i ];
}
```

- Execution policies let us chose where to execute each kernel, but the programming framework can't tell us which is fastest!

## Apollo builds decision-models to tune parameters

- Using supervised learning in an offline training phase, we **build a classifier that directly predicts the fastest parameter value** for each kernel invocation.
- The application is run multiple times with various input problems and execution policies to generate a training data set.
- Apollo's Python framework process this data and learns a decision tree model.
- We then generate a **C++ decision model that can be evaluated at runtime** to dynamically tune the execution policies an application is using.




## Apollo selects the fastest parameter up to 98% of the time

- We build one model per-application, based on training data generated from three input problems and 5 problem sizes.
- Using 10-fold cross-validation, the mean accuracy of these models is up to 0.98

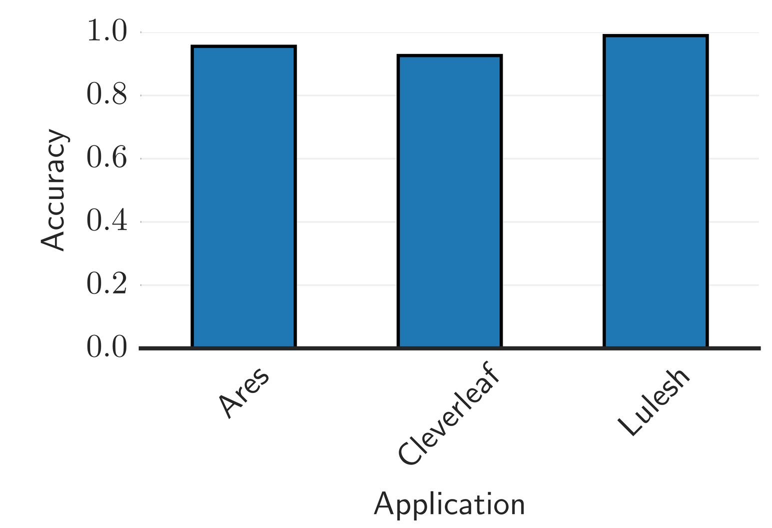
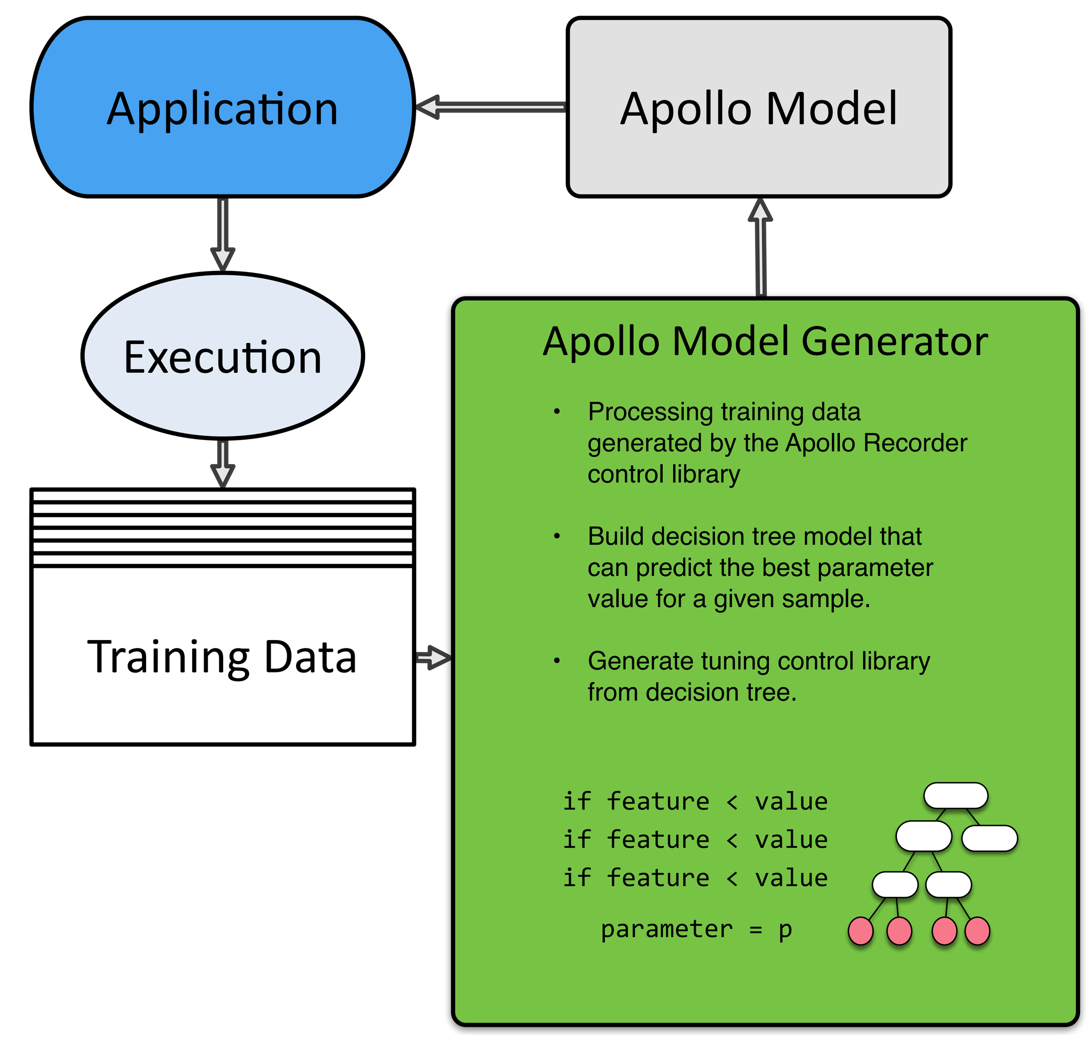



**Figure 3:** Predictive accuracy of each Apollo models.




## Apollo models are dynamically loadable

- Apollo loads the compiled C++ decision model at application startup.
- Execution policies are template parameters that control which programming model backend is selected. Apollo instantiates each policy type to allow dynamic backend selection.
- The decision model sits between RAJA and the execution policy back ends, dynamically selecting an execution policy type based on the features of the kernel that is about to be executed.

## Dynamically tuning policies at runtime provides speedups of up to 4.8x

- These speedups are problem-dependent, since the fastest policy choices dependent on the kernel invocation parameters.
- The data sizes encountered in the Triple Point problem in CleverLeaf benefit most, with a speedup of 4.8x.

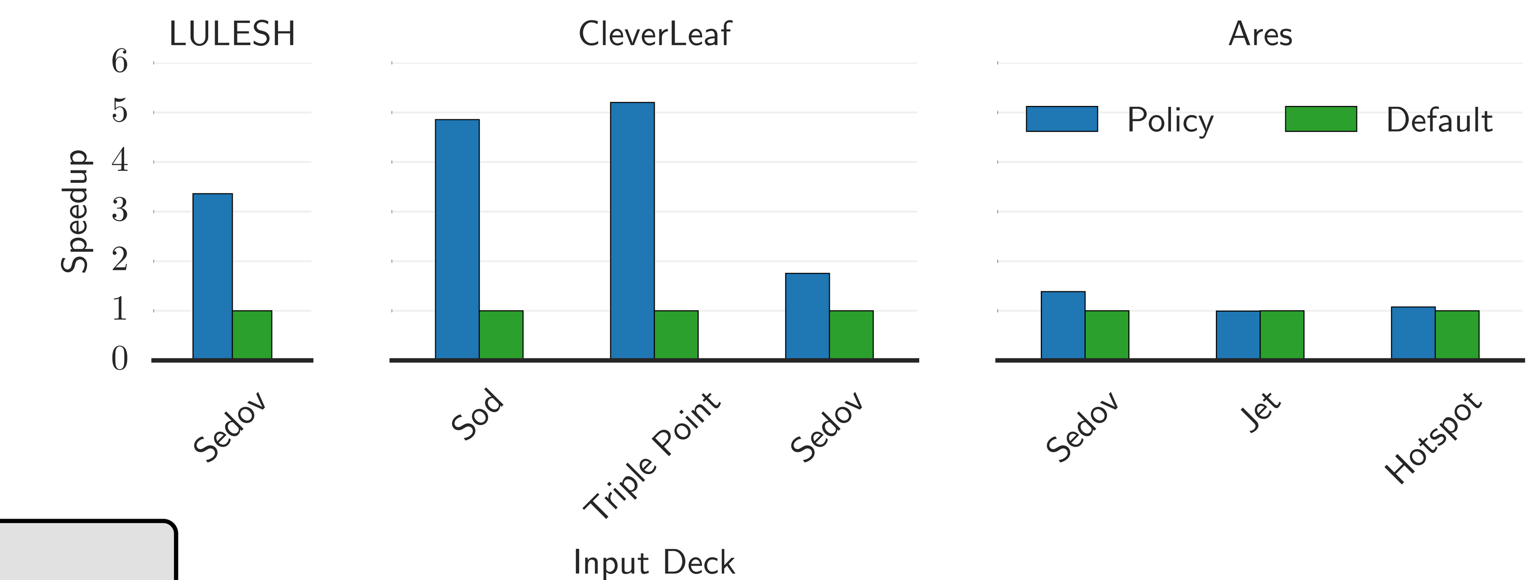



**Figure 5:** Online speedups for multiple input problems in each application.

## MPI ranks make independent tuning decisions

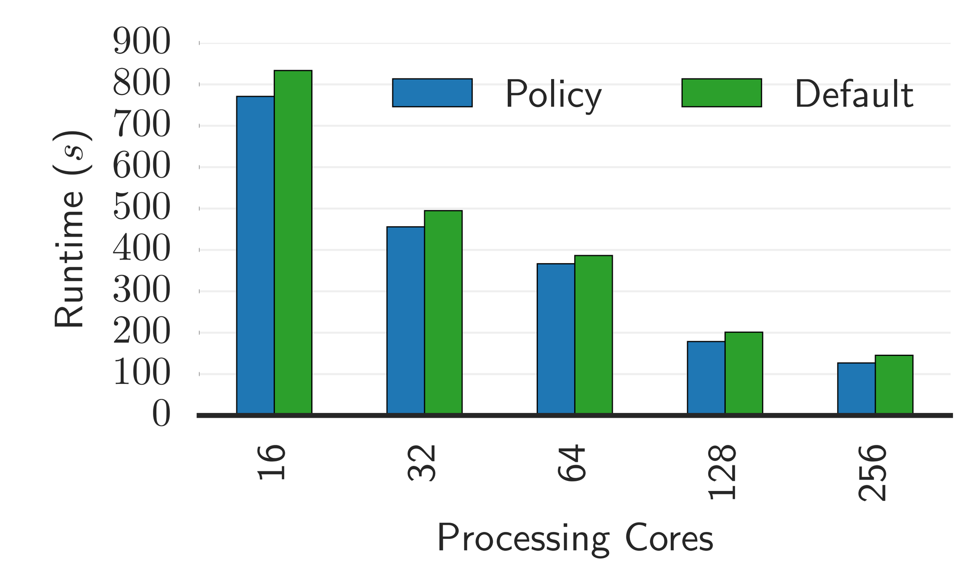



**Figure 6:** Parallel runtimes for the Hotspot problem in ARES. We see speedups on up to 256 processor cores.

## Auto-tuning with Apollo frees application developers from manually choosing parameters

- Existing auto-tuners rely on costly search procedures and over fit for specific inputs, whilst Apollo can learn how to select the fastest parameters based on application features.
- We are working to build general models that will allow us to predict the fastest parameters across both applications and hardware platforms, allowing us to learn models from a vast body of training data, then apply them to a wide range of application runs.
- We will also extend our work to support heterogeneous platforms, using Apollo to predict *where* to run kernels, in addition to other parameters.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

LLNL-POST-697252