Discovering Optimal Execution Policies in KRIPKE using RAJA

William K. Killian[1,2], Adam J. Kunen[2] (Advisor), Ian Karlin[2] (Advisor), John Cavazos[1] (Advisor) [1] University of Delaware [2] Lawrence Livermore National Laboratory

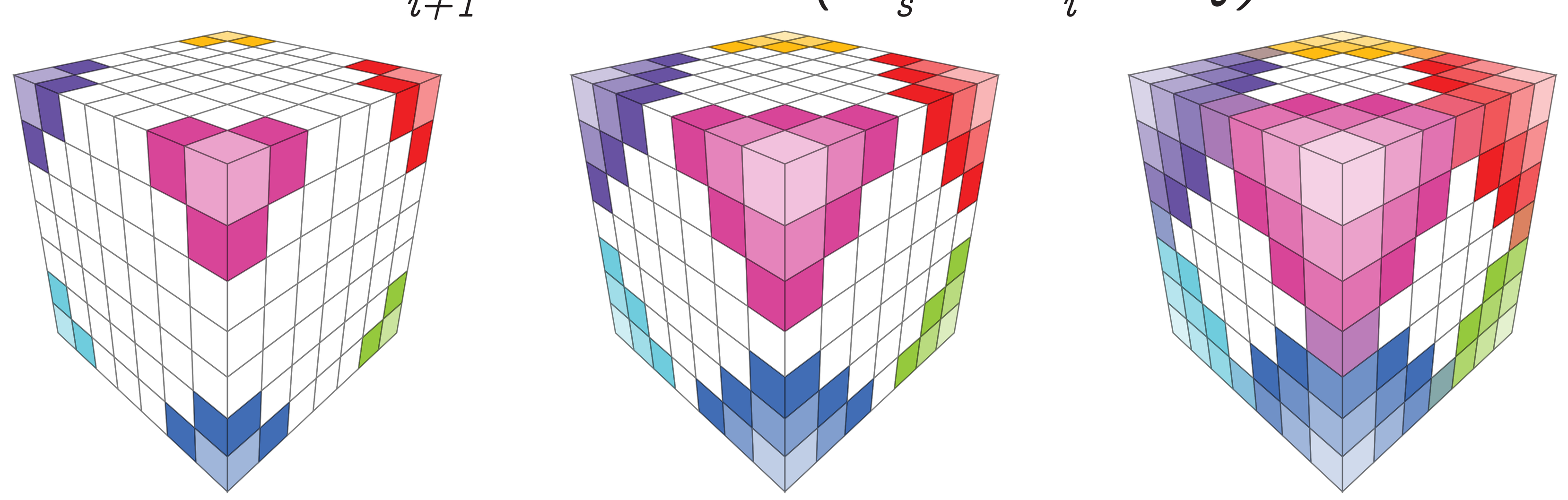
Motivation

- Legacy physics applications need updating to run well on newer architectures but are not always designed for architecture flexibility
- With architectures changing frequently (multicore, many-core, GPU), applications need to be adaptable to many different architectures.
- Adaptive, flexible programming layers are necessary to intelligently search large optimization spaces

KRIPKE

- KRIPKE is a proxy application for Sn particle transport developed at LLNL
- Highly dimensional: composed of directions, groups, zones, and moments
- Many possible nestings of data and execution. Difficult to find the best
- Solves the linear Bolzmann equation using sweeps over a 3D domain space
- **Goal: find optimal execution policies for common configurations of KRIPKE**

$$\Psi_{i+1} = H^{-1}\, L^{+}\, (\Sigma_s\, L\ \Psi_i + Q)$$



Sweep (t=1) Sweep (t=2) Sweep (t=3)

Time sequence of the sweep kernel (H^{-1}) moving through the mesh. Multiple sweeps can occur at the same time. Grid contention occurs when a location has equal manhattan distance from two or more sources (corners).

RAJA Performance Portability Layer

- Provides C++ abstractions to enable architecture portability
- Predefined execution policies exist for SIMD, OpenMP, and CUDA
- Nested and advanced loop transformations (tiling, reordering) are available
- **Goal: use RAJA to drive optimization search space exploration for KRIPKE**

Example RAJA Execution Policy to apply

```
NestedPolicy<
 ExecList<
  seq_exec, seq_exec,
  omp_for_nowait_exec, simd_exec>,
 OMP_Parallel<
  Tile<
   TileList<
    tile_none, tile_none,
    tile_none, tile_fixed<512>>,
   Permute<PERM_JIKL>
  >
 >
>
```

Basic loop implementation

```
for d in range(0,dom<IDirection>(id)):
  for nm in range(0,dom<IMoment>(id)):
    for g in range(0,dom<IGroup>(id)):
      for z in range(0,dom<IZone>(id)):
```

Nested Policy applied to loop

```
#pragma omp parallel
for z2 in range(0,dom<IZone>(id),512):
  for d in range(0,dom<IDirection>(id)):
    for nm in range(0,dom<IMoment>(id)):
      #pragma omp for nowait
      for g in range(0,dom<IGroup>(id)):
        for z in range(z2,z2+512):
```

Policy Description and Generation

Policy Search Space

- Four execution policies: sequential, SIMD, OpenMP, collapsed OpenMP
- Five tiling policies: no tiling and fixed tiles of sizes 8, 32, 128, and 512
- Considered only loop valid nests, tiles must fit in L3 cache, no nested thread parallelism, OpenMP clauses only with OpenMP loop nests
- Policies are generated for each independent loop nest
- Five different loop nests:
  1. LTimes [L] -- 4-nested loop with 850K versions
  2. LPlusTimes [L^{+}] -- 4-nested loop with 850K versions
  3. Scattering [Σ_s] -- 4-nested loop with 850K versions
  4. Sweep [H^{-1}] -- 3-nested loop with 2.9K versions
  5. Source [Q] -- 2-nested loop with 0.45K versions

Optimization Space Exploration

- Assume kernel executions are independent of one another
- Too costly to run each execution policy for a larger Sn transport code.
- We propose two different strategies to explore the optimization space
- **Goal: find optimal execution policies of kernels without exhaustive execution**

Hill-climbing Strategy

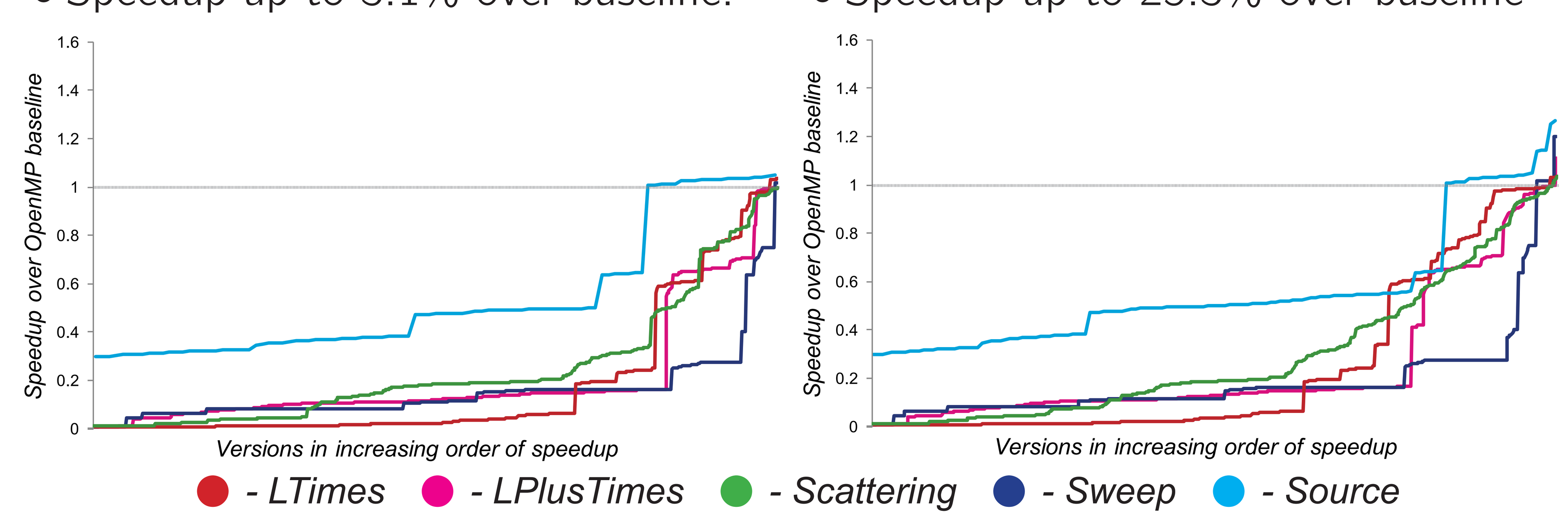
$V \leftarrow$ all versions of KRIPKE
$F \leftarrow$ all features of a loop nest
$count \leftarrow 0$
do while $count < threshold$
  $p \leftarrow$ rand (V)
  $best \leftarrow p$
  foreach $i, f \in$ shuffle (enumerate (F)) do
    foreach $option \in F_i$ do
      $p_i \leftarrow option$
      $count \leftarrow count + 1$
      if time $(p) <$ time $(best)$ $best \leftarrow p$ end
    end
  end

- Limited to 10% of total search space
- Speedup up to 3.1% over baseline.

Subspace Search Strategy

$V \leftarrow$ all versions of KRIPKE
$F \leftarrow$ all features of a loop nest
$count \leftarrow 0$
do while $count < threshold$
  $V' \leftarrow \{$rand $(V)\}$
  foreach $i, f \in$ shuffle (enumerate (F)) do
    foreach $option \in F_i$ do
      $V_{option} \leftarrow \{v_i \leftarrow option\ \forall\ v \in V'\}$
      $count \leftarrow count + |V'|$
      $V' \leftarrow V' \cup V_{option}$
    end
    remove all but top k from V'
  end

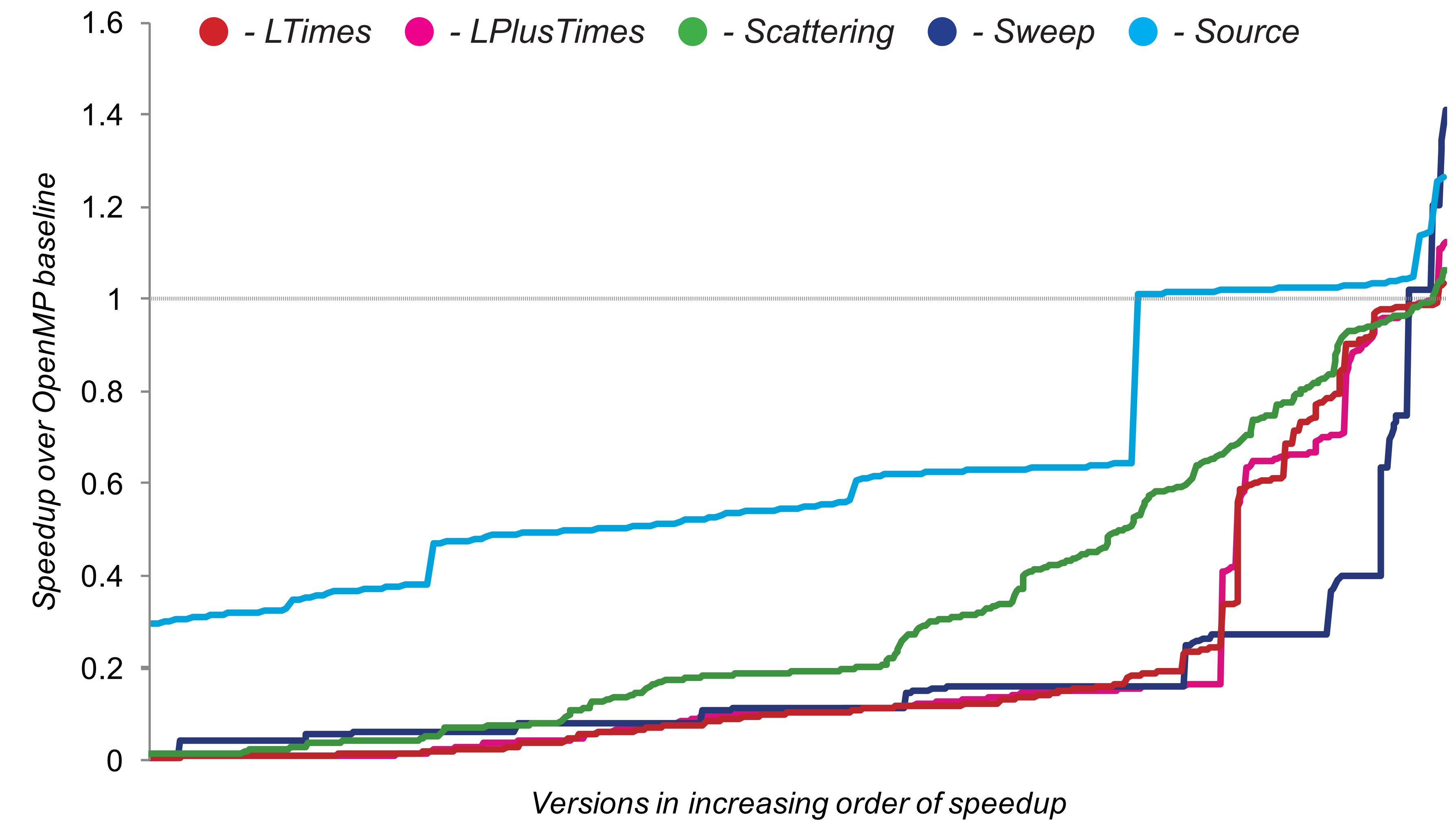
- Limited to 20% of total search space
- Speedup up to 25.3% over baseline



Explored versions are shown by increasing speedup over OpenMP baseline. Subspace search does better than hill-climbing because the strategy was more likely to cover more tiling policies and consider non-local search spaces.

Performance Analysis

Exhaustive Execution



- **Architecture:** dual-socket Intel Xeon E5-2670, 32GB DDR3 RAM
- **Compiler:** Clang 3.8.0 with OpenMP support (`-O3 -march=native`)

Comparison to Exhaustive Execution

- To evaluate our search strategies, we run all generated versions of KRIPKE.
- The best discovered policies improves over the basline performance of the entire KRIPKE proxy application by 19.5%.
- Hill-climbing achieves up to 95.6% of optimal performance while subspace search achieves up to 98.8% of optimal performance.

Conclusion and Future Work

- Used the RAJA performance portability layer to explore a large optimization space efficiently within the KRIPKE Sn transport proxy application
- Two different search space strategies can yield results up to 98.8% of optimal while only exploring 20% of the total search space.
- The best known execution time of KRIPKE improves by 19.5%.

Future Work

- Expand results to include GPU execution policies (NVIDIA Kepler/Pascal) and nested parallelism with many-core (Intel Knight's Landing) architectures
- Augment tiling policies to include multi-level tiling. This will be useful when targeting future architectures with complex memory hierarchies.
- Construct an accurate control-flow graph-based performance prediction model. The predictor replaces exaustive execution with only compilation.

Acknowledgments and Resources

[1] A. J. Kunen, T. S. Bailey, P. N. Brown, *KRIPKE - A Massively Parallel Transport Mini-App*, American Nuclear Society M&C, 2015 **[https://codesign.llnl.gov/kripke.php]**

[2] R. D. Hornung and J. A. Keasler, *The RAJA Portability Layer: Overview and Status*, *Tech Report*, LLNL-TR-661403, Sep. 2014. **[https://github.com/llnl/RAJA]**

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