Predictive Modeling in a Polyhedral Optimization Space

Eunjung "EJ" Park[1], Louis-Noël Pouchet[2], John Cavazos[1], Albert Cohen[3], and P. Sadayappan[2]

[1] University of Delaware
[2] The Ohio State University
[3] ALCHEMY group, INRIA Saclay - Île-de-France

April 5th, 2011
IEEE/ACM Symposium on Code Generation and Optimization
Chamonix, France

INRIA

THE OHIO STATE UNIVERSITY

Do you want good performance?

- Optimizing loops is crucial!

- Difficulty on finding good loop optimizations
  - Complex interplay between hardware resources
  - Conflicts between optimization strategies
  - Large compiler optimization space

- Any Solutions?
  - Using iterative compilation
  - Using polyhedral framework

Why polyhedral framework?

- Advantages over standard compiler framework
  - Good expressiveness
  - Complex optimization sequences
  - Loop tiling of imperfectly nested loop

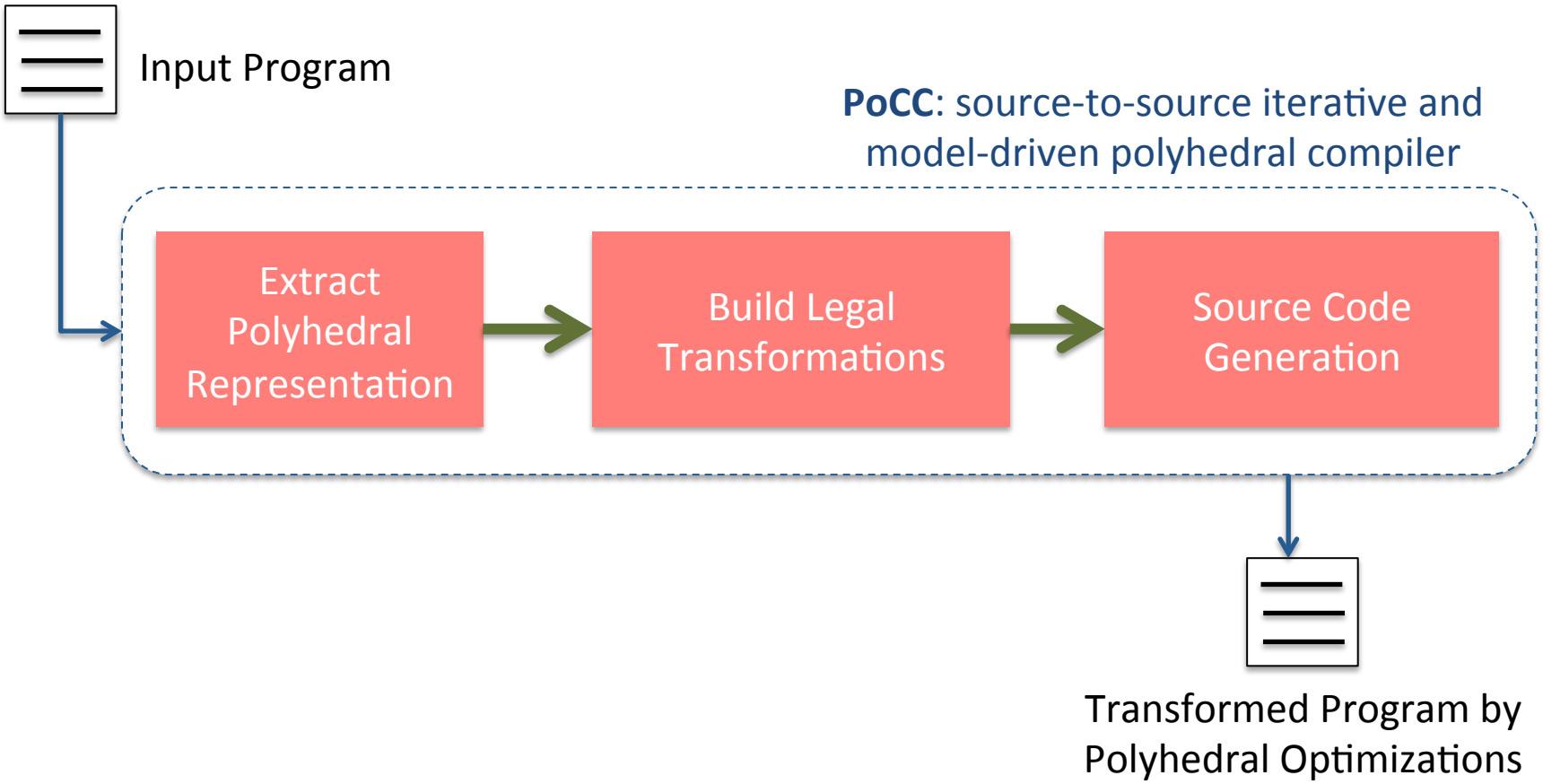
- Very large optimization space

Our Solution
Let's take advantages of **Polyhedral Framework** and **Iterative Compilation!**

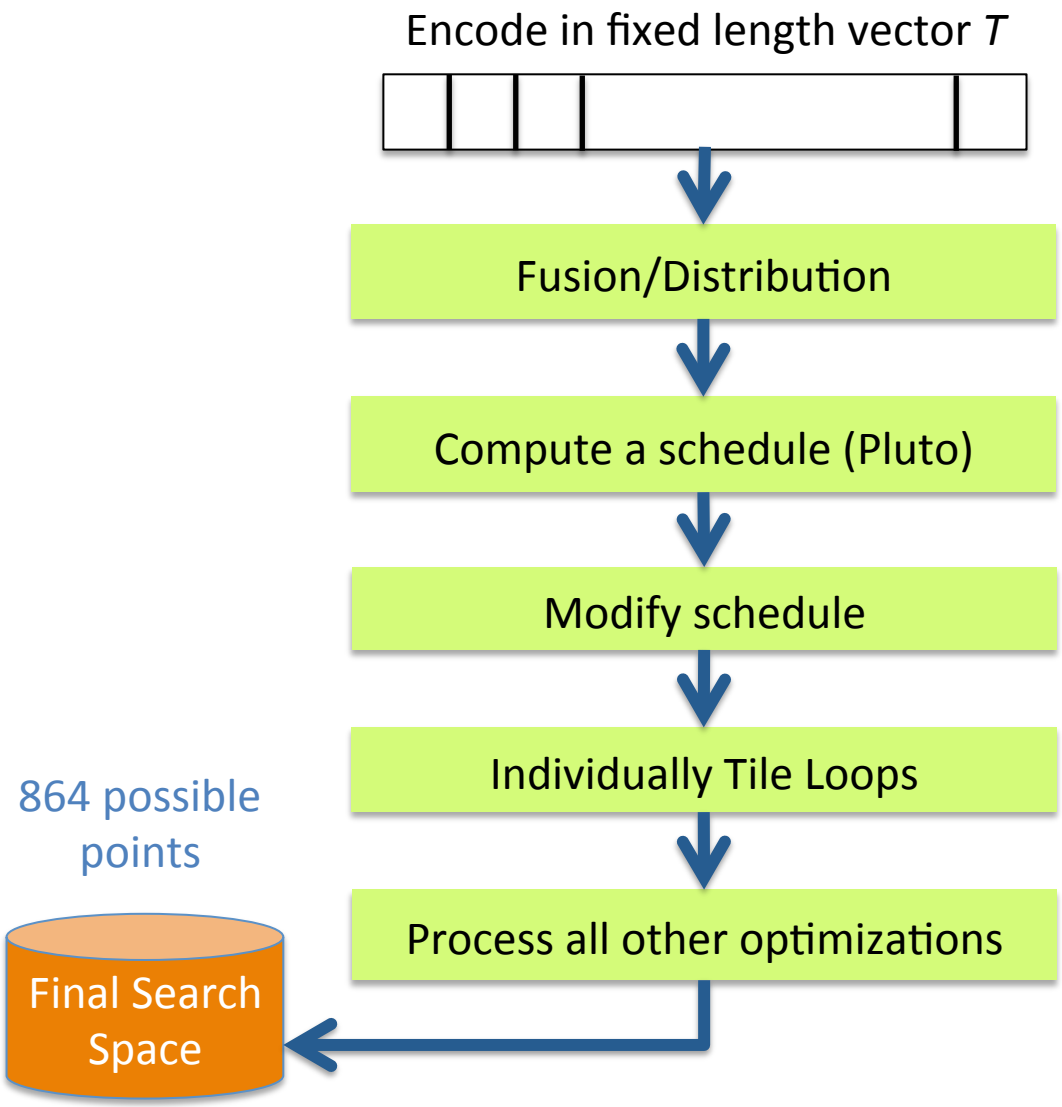
Contributions of this paper

- Power of three approaches
  - Expressiveness from polyhedral framework
  - Performance prediction by machine learning
  - Iterative compilation
- Build prediction model for polyhedral optimization primitives
- Reduce number of evaluations, achieve good performance

Polyhedral Compiler Framework



Polyhedral Optimization Space

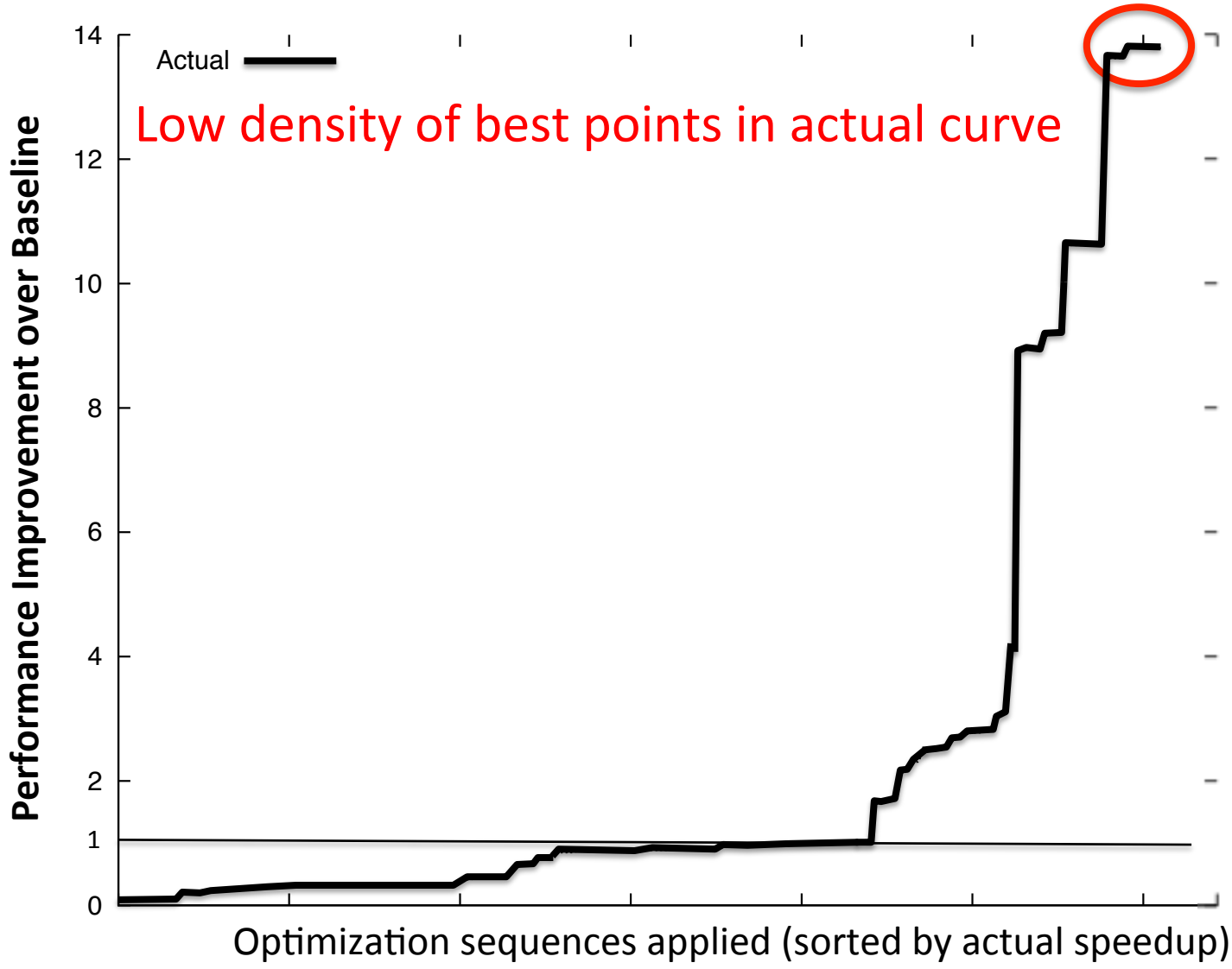


864 possible points

Optimizations Handled in each step

- Fusion/Distribution/Code Motion
- Auto Parallelization, Tileability
- Vectorization
- Tiling
- OpenMP Pragma, Unrolling

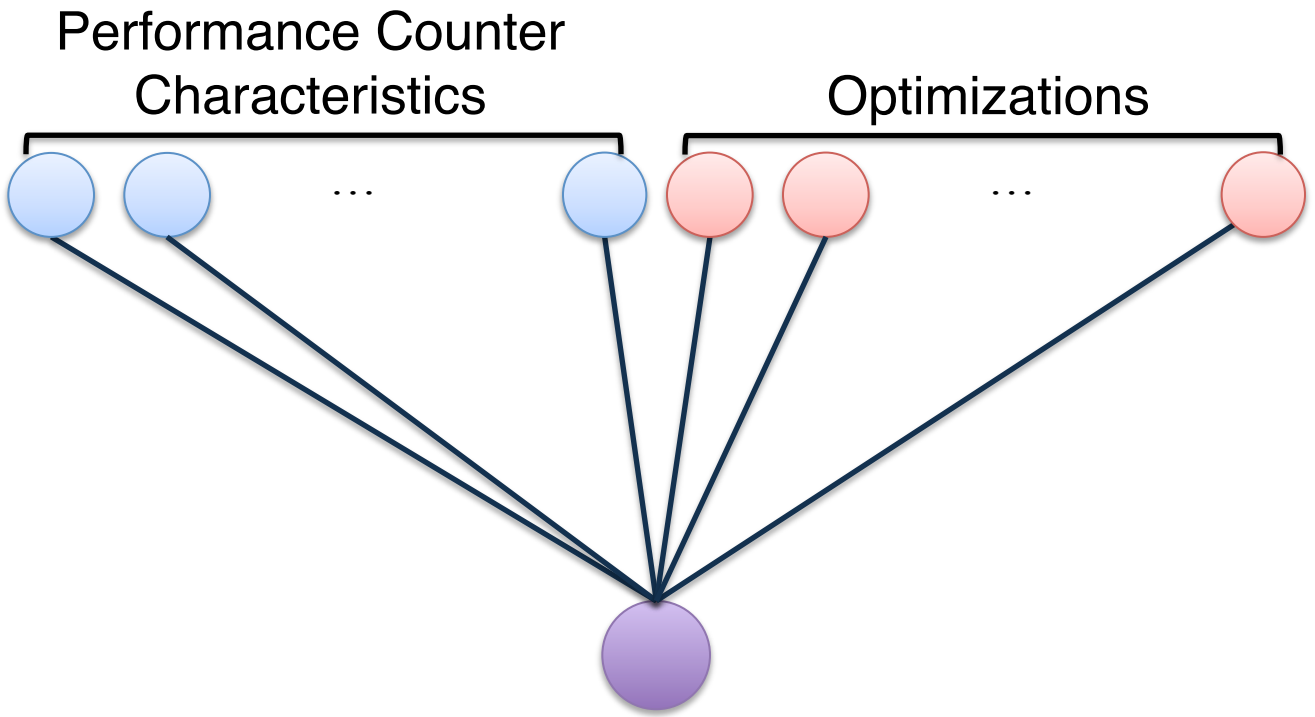
Finding Best Speedup is Hard



Prediction Model

- Model Description
- Building Model
  - Model Construction
  - Model Usage

Speedup Prediction Model



Building Model

- Leave-one-out cross validation

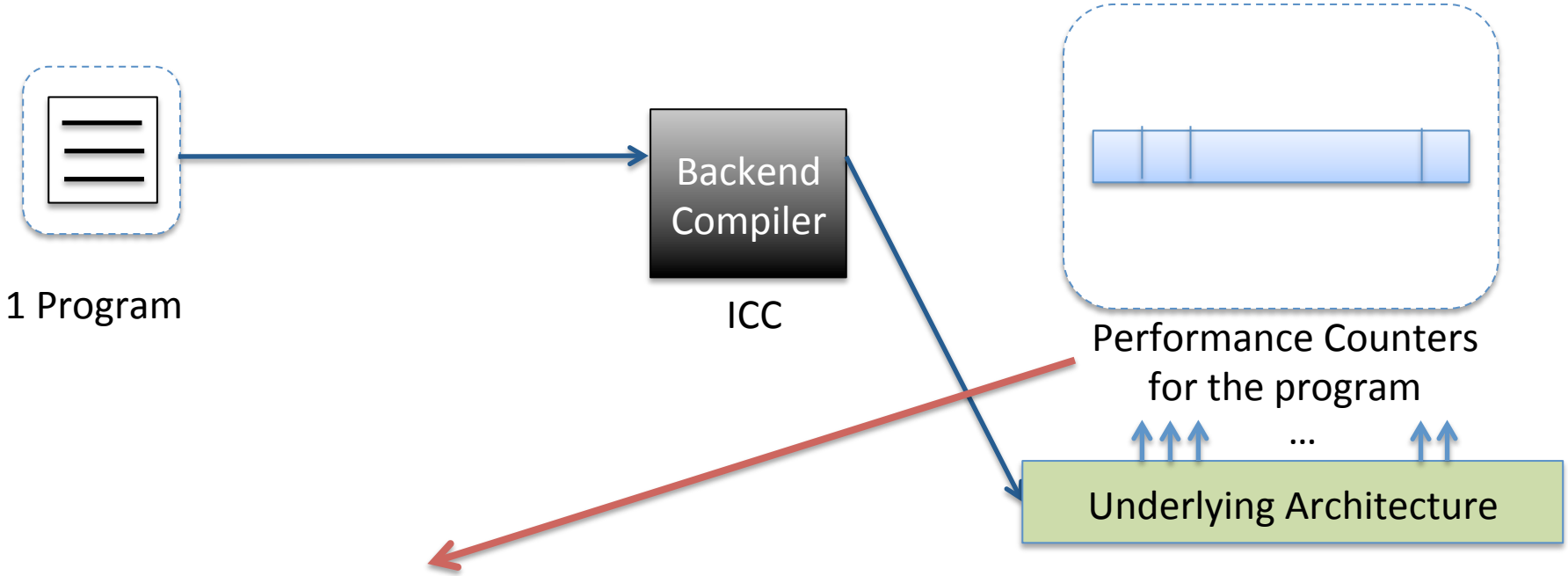
```
for i-th program where i=1 to N
do

    Train a model on N-1 programs excluding i-th program
    Test the model on the i-th program left out

done
```

Model Construction

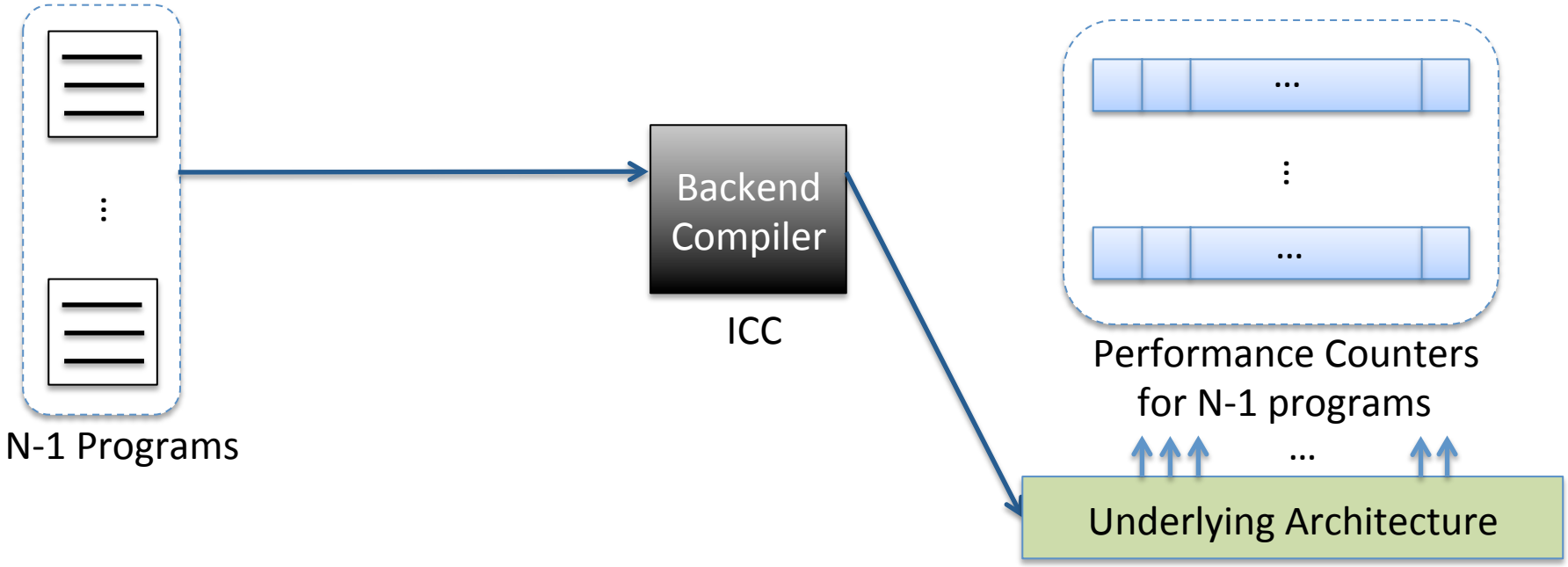
- Collect dynamic behavior for a program



Performance Counters include total number of
- accesses and misses in all levels of cache and TLB
- stall cycles
- vector instructions
- issued instructions

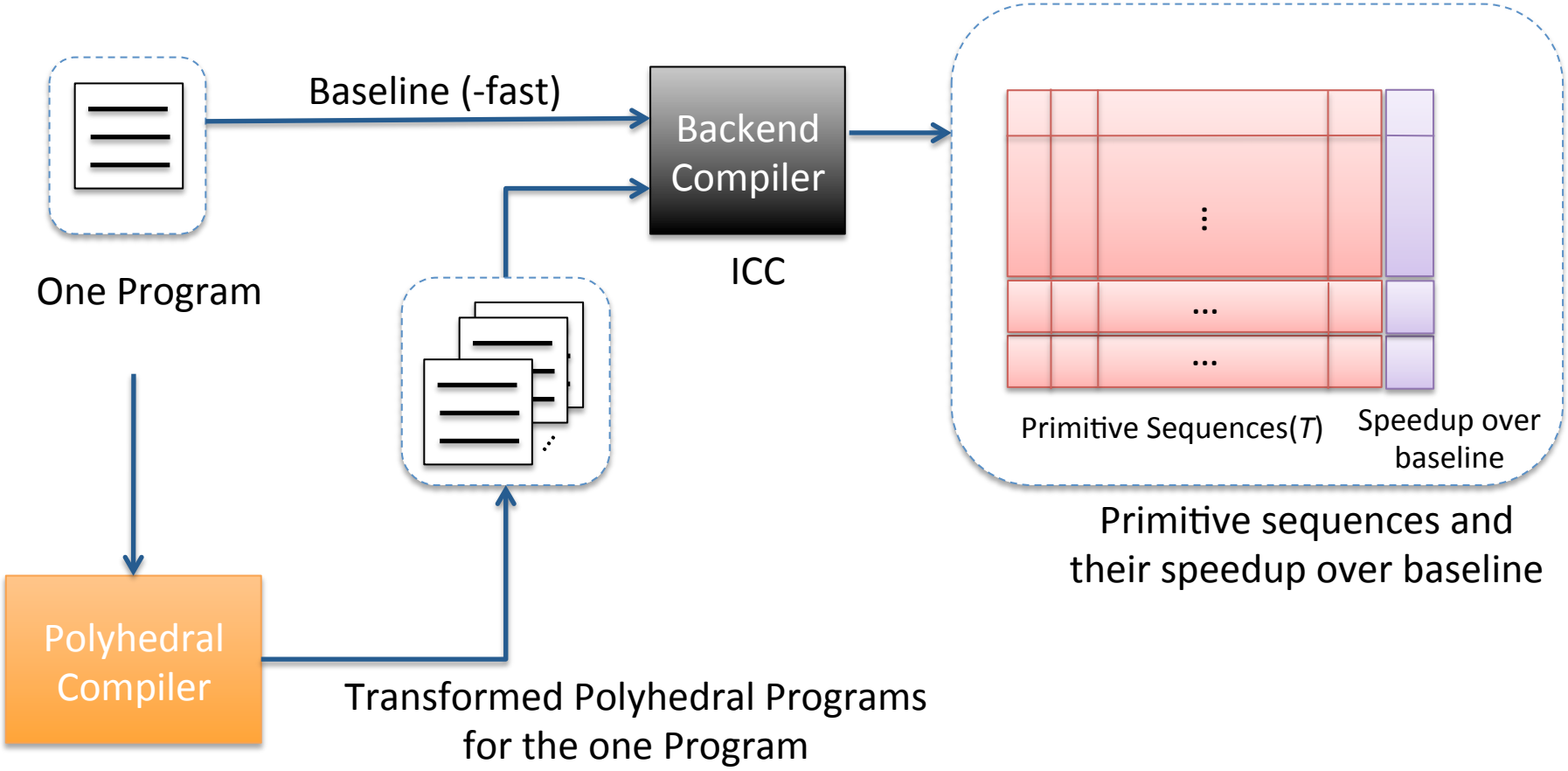
Model Construction

- Now do this for N-1 programs



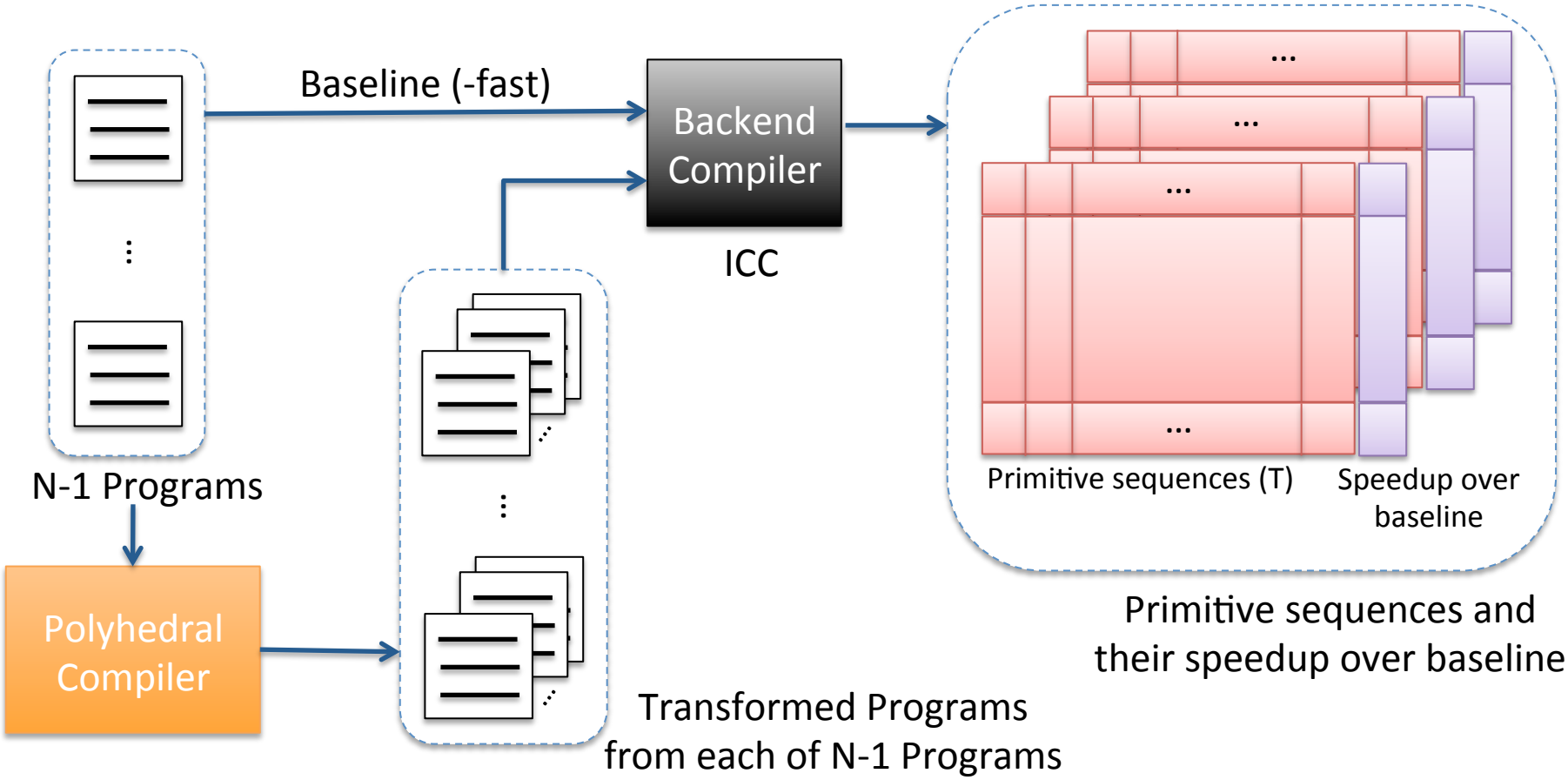
Model Construction

- Run transformed programs, and get speedups

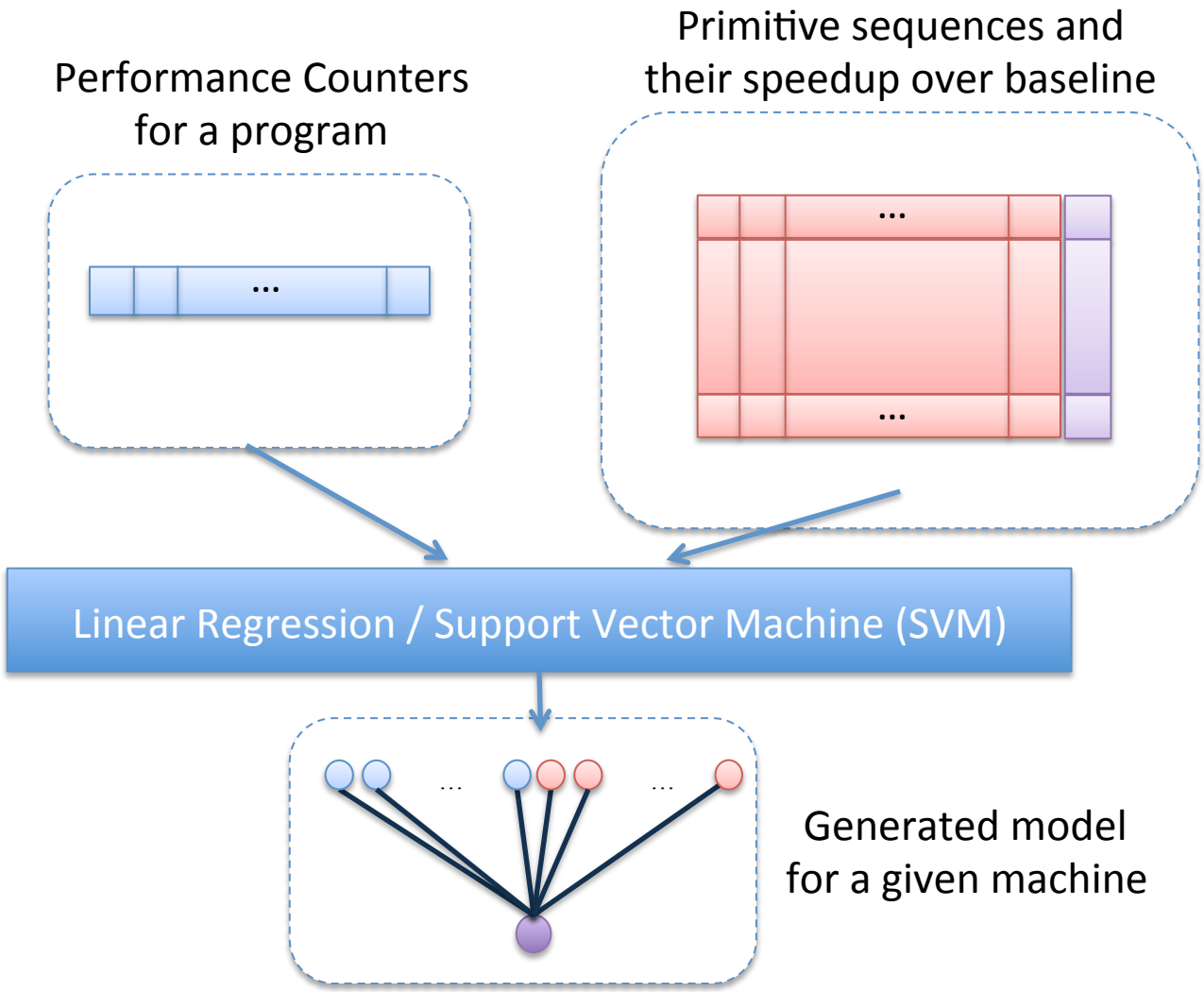


Model Construction

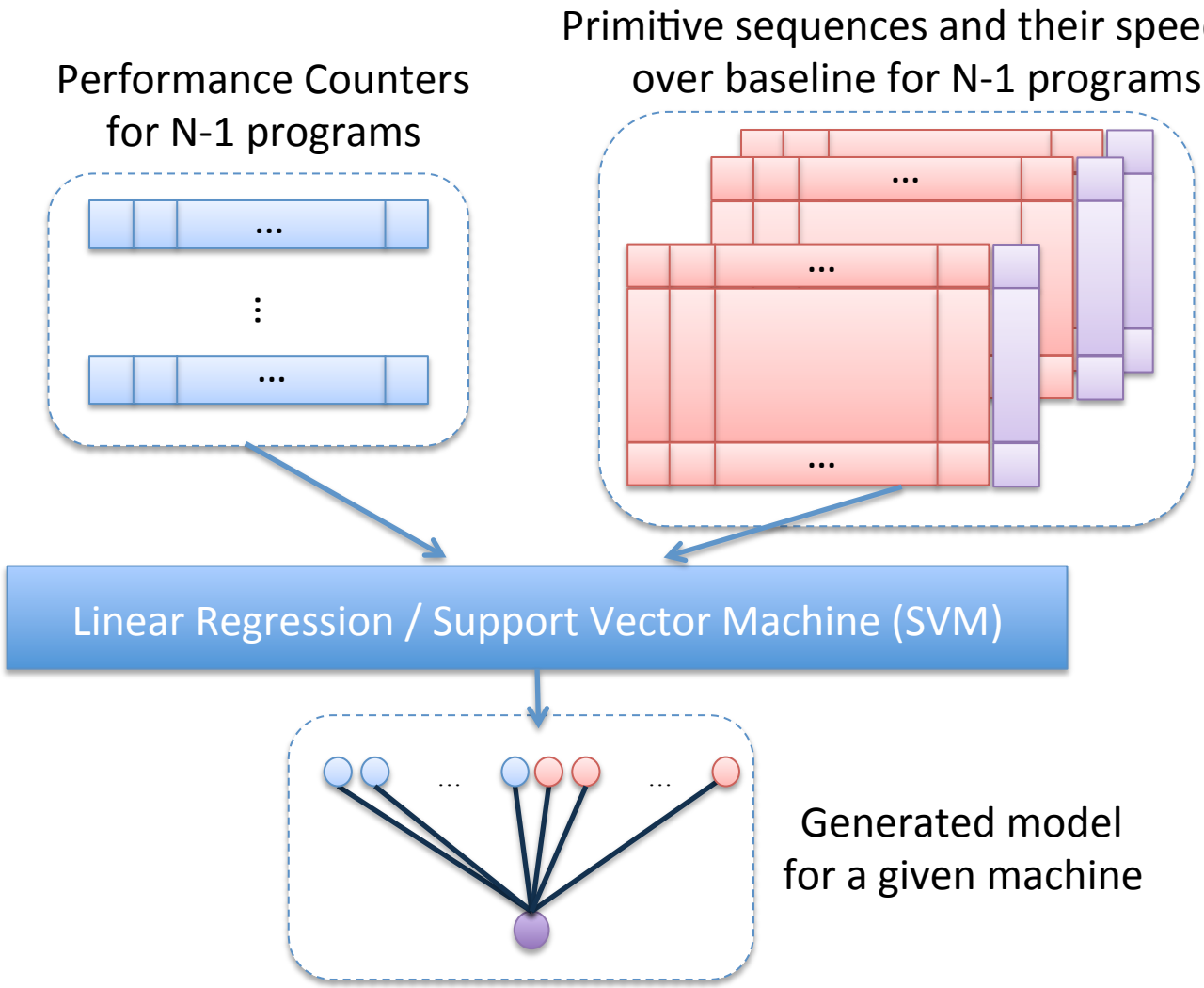
- Now do this for N-1 programs



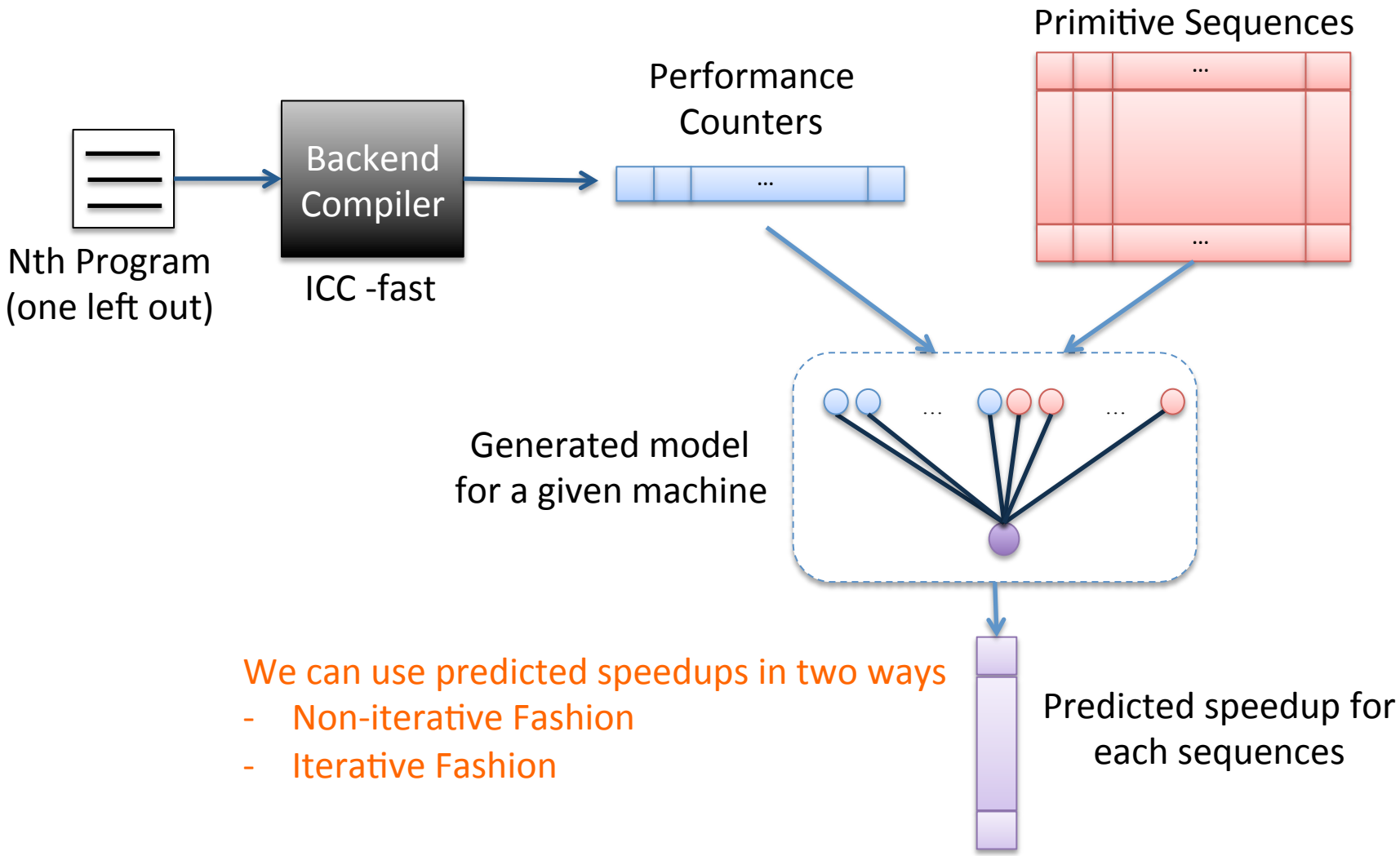
Model Construction



Model Construction



Using Model


Primitive Sequences
Performance Counters
Backend Compiler
Nth Program (one left out)
ICC -fast
Generated model for a given machine
We can use predicted speedups in two ways
- Non-iterative Fashion
- Iterative Fashion
Predicted speedup for each sequences

Experimental Configuration

- Hardware configuration
  - Nehalem
    - Intel Xeon E5620 2.4GHz, 2 sockets 4 cores, 16 H/W threads, L3 12MB
  - R900/Dunnington
    - Intel Xeon E7450 2.4GHz, 4 sockets 6 cores, 24 H/W threads, L3 12MB

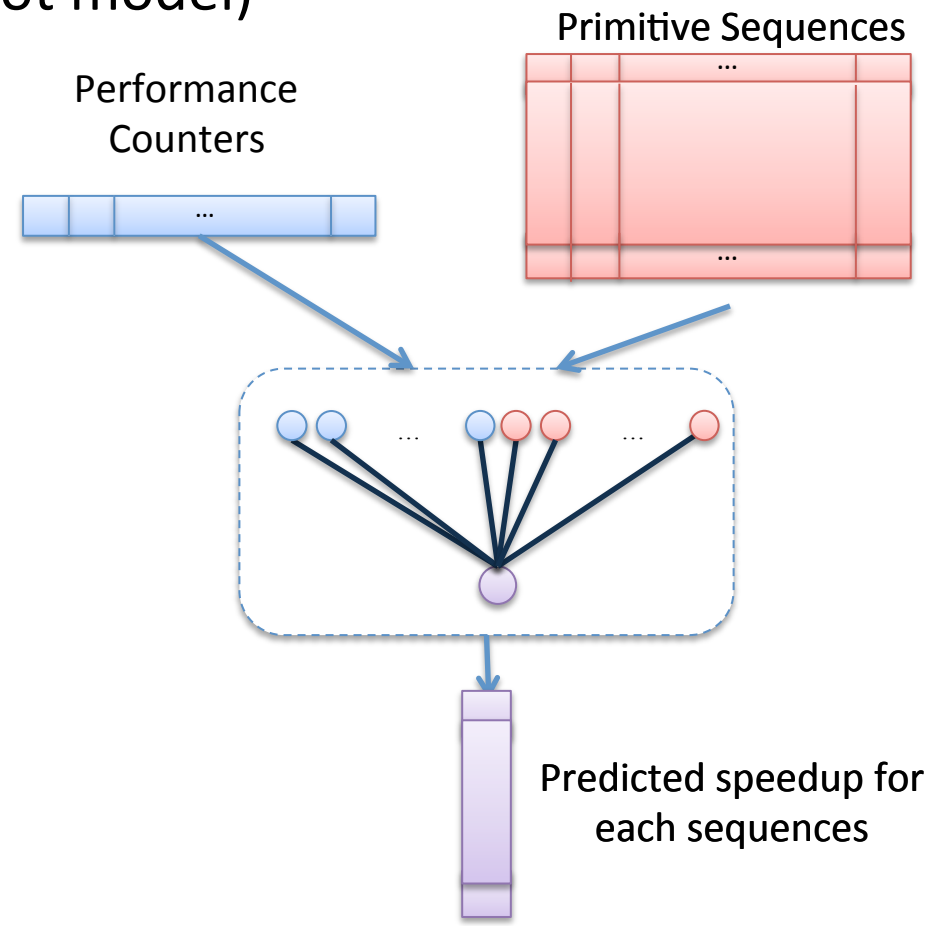
- Software Configuration
  - Backend compiler: ICC
  - Baseline: ICC with –fast, single threaded (no auto-par)
  - Machine learning framework: Weka v3.6.2
    - Linear Regression/SVM (SMOReg)
  - Benchmark: PolyBench 2.0 (28 different kernels and applications)

Experimental Analysis

- Performance of our model
- Our model versus random

Experiment 1: Performance of our model

- Shot = Evaluating optimization sequence
- Non-iterative fashion (1-shot model)
- Iterative fashion
  - 2-shot model
  - 5-shot model



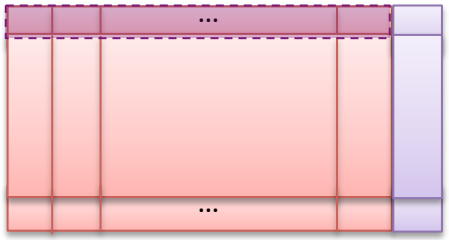
Experiment 1: Performance of our model

- Shot = Evaluating optimization sequence
- Non-iterative fashion (1-shot model)

- Iterative fashion
  - 2-shot model
  - 5-shot model

Sorted by Predicted Speedup

top1



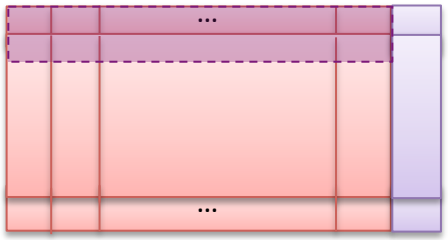
Experiment 1: Performance of our model

- Shot = Evaluating optimization sequence
- Non-iterative fashion (1-shot model)

- Iterative fashion
  - 2-shot model
  - 5-shot model

Sorted by Predicted Speedup

top2



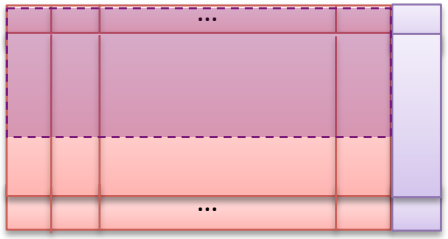
Experiment 1: Performance of our model

- Shot = Evaluating optimization sequence
- Non-iterative fashion (1-shot model)

- Iterative fashion
  - 2-shot model
  - 5-shot model

Sorted by Predicted Speedup

top5



Experiment 1: Performance of our model

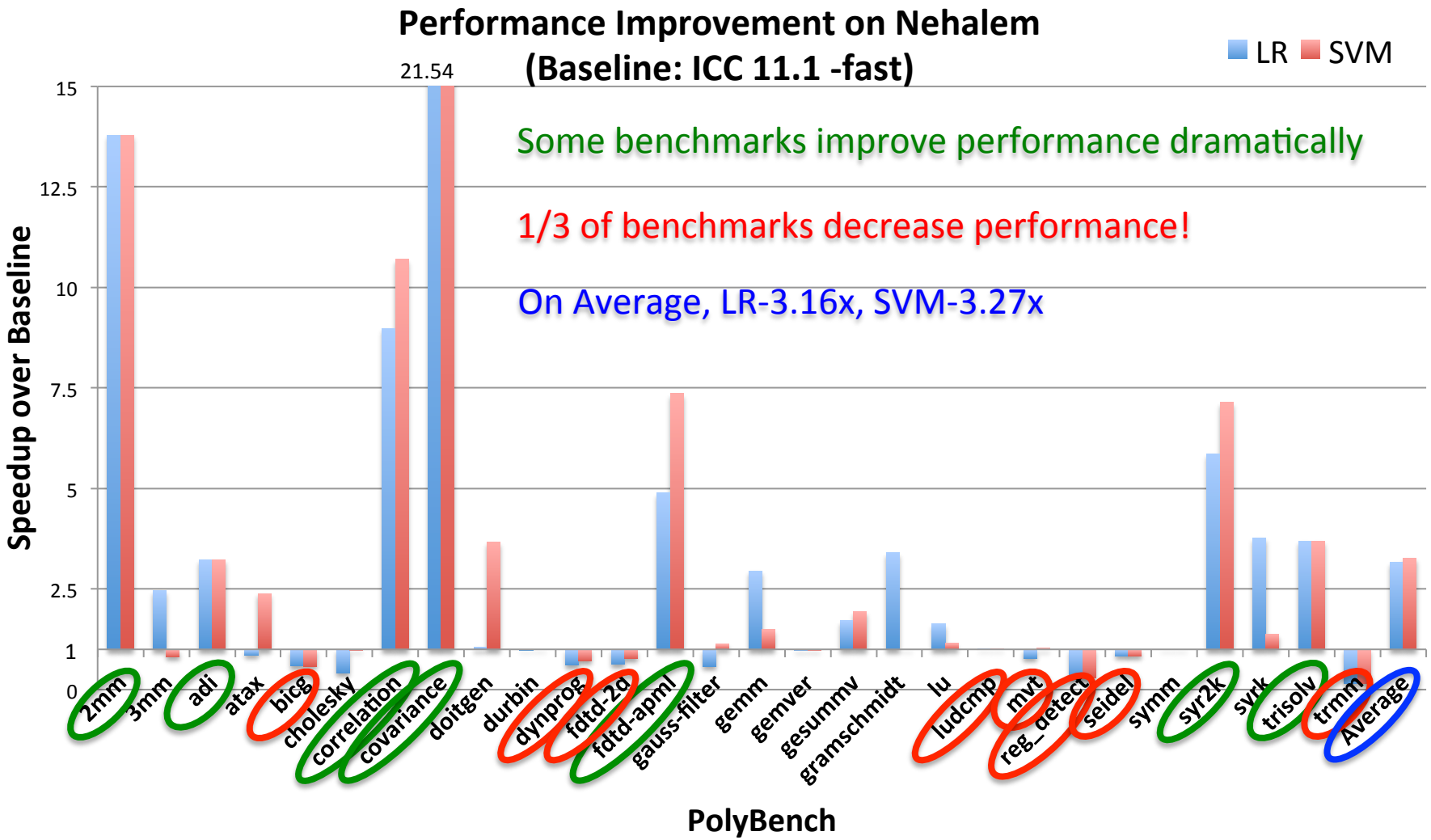
- Summary
  - With 5 iterations, we reached more than 80% of the space optimal!

	Nehalem		R900/Dunnington	
	LR	SVM	LR	SVM
1-shot	3.16x	**3.27x**	**4.91x**	3.77x
2-shot	3.16x	**3.43x**	**4.92x**	4.91x
5-shot	3.50x	**4.68x**	5.82x	**6.60x**

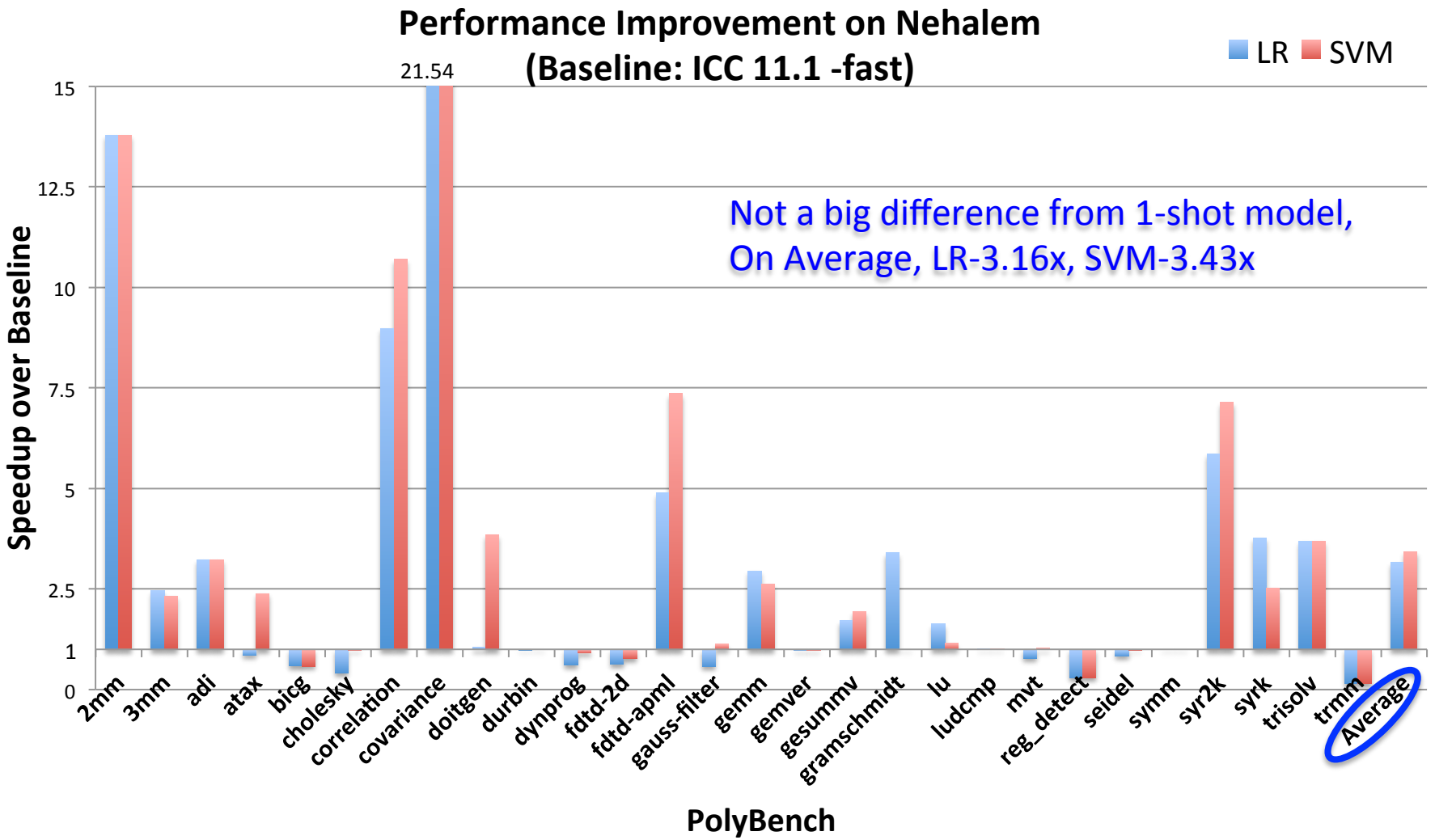
- Space Best vs. Poly vs. ICC

	Nehalem	R900/Dunnington
SpaceBest	6.10x	8.04x
Poly	2.13x	4.48x
ICC	1.98x	3.38x

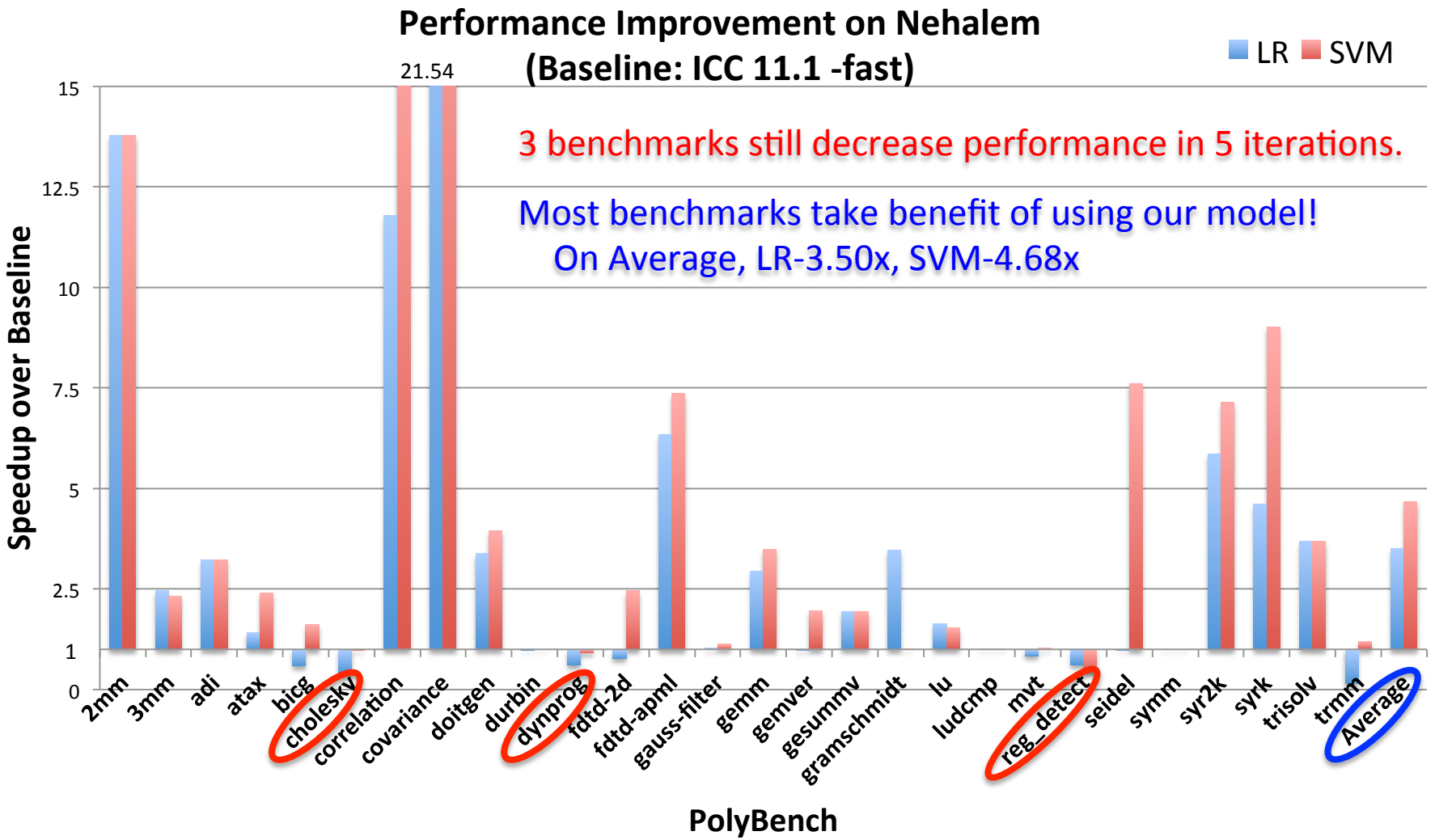
Experiment 1: Nehalem 1-Shot



Experiment 1: Nehalem 2-Shot



Experiment 1: Nehalem 5-shot



Experiment 2: Our Model vs. random

- Random search discovers performance improvement

	Nehalem		R900/Dunnington	
	Random	Our Model	Random	Our Model
1-shot	1.61x	**3.27x**	2.14x	**4.91x**
2-shot	2.24x	**3.43x**	3.19x	**4.92x**
5-shot	3.32x	**4.68x**	3.99x	**6.60x**

- However, our model performs significantly better!

Conclusion

- Power of three approaches
  - Expressiveness from polyhedral framework
  - Performance prediction by machine learning
  - Iterative compilation
- Build prediction model for polyhedral optimization primitives
- Reduce number of evaluations, achieve good performance
  - More than 80% of search space optimal performance in 5 iterations
- Machine-independent algorithm but machine-dependent result (trained specifically on the target machine)
  - Performance portability is achieved

Predictive Modeling in a Polyhedral Optimization Space

Thank You!

INRIA

THE OHIO STATE UNIVERSITY