PerfVec: Generalizable Representation Learning for Performance and Energy Modeling of Computer Architectures

Lingda Li

Brookhaven National Laboratory

ModSim 2025

Seattle, WA

Computer Architecture ModSim

- Essential to computer architecture research and engineering
  - New design evaluation, design space exploration, resource scheduling, …
- Goals: speed, accuracy, and generalizability

Methodology	Speed	Accuracy	Generalizability
Analytical Modeling	Fast	Low	High
		High	Low
Discrete Event Simulation	Slow	Variable	High
Emulation	Medium	Variable	Medium
Machine Learning (ML)-based Modeling	Fast	High	Low
ML-based Simulation	Medium	Variable	Medium
PerfVec (this work)	Fast?	High?	**High?**

Goal: explore better tradeoff using ML, especially on generalizability

Brookhaven National Laboratory

Li, Flynn, Hoisie. Learning Generalizable Program and Architecture Representations for Performance Modeling. SC 2024.

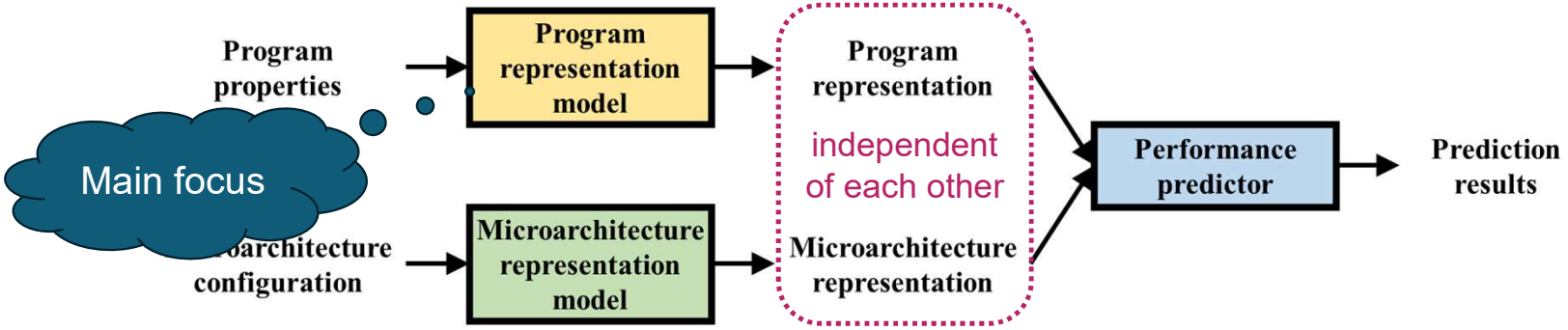
Generalizable Modeling

- Performance/energy is determined by software and hardware.
- A generic model should separate the impact of software (program) and hardware (microarchitecture).
  - When one changes, no need to re-model the other
- Not trivial to achieve such separation
  - Complex interplay between program and microarchitecture
  - Previous attempts are not general (e.g., several analytical models, matrix completion).

PerfVec **learns** the separation automatically.

Learning Separation

Key idea 1: have two independent ML models to capture the impacts of program and microarchitecture, respectively



- The same program representation is used to predict its performance on any microarchitecture, and vice versa.

Learning Program Representation

Input	Level of Detail	Limitation
Profiling info (e.g., performance counters)	Low	• Often microarchitecture dependent • Lack of low level details
Static program info (e.g., control flow graph)	Medium	• Cannot capture dynamic execution (e.g., input impact) • Huge graphs to learn from
Instruction execution trace	High	• Huge amounts of instructions to learn from

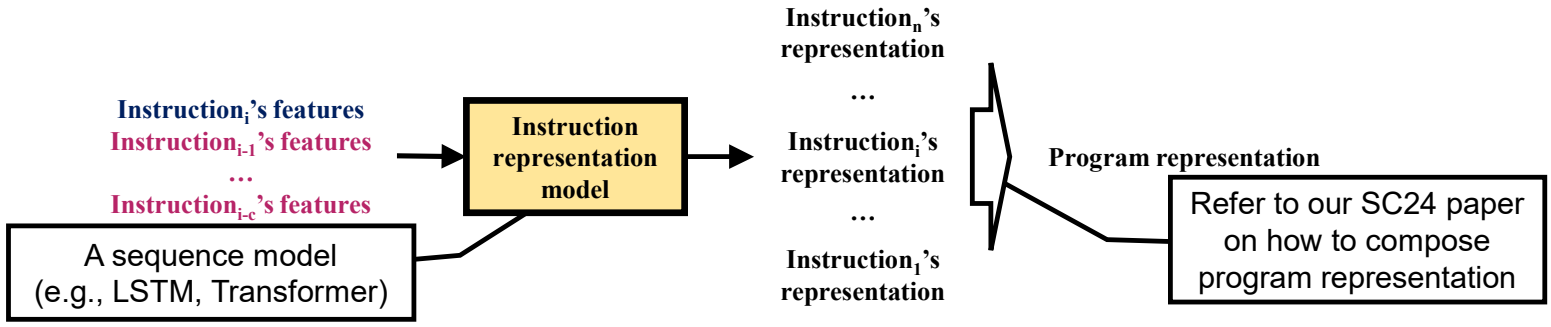
- Challenge: programs typical execute at least billions of instructions, and no ML model can deal with such long sequences.

Key idea 2: 1) learn the representations of individual instructions and 2) compose a program representation from those of all its executed instructions (divide-and-conquer)

Learning Instruction Representation

- Performance determination factors of an instruction

Factor	Microarchitecture Independent Feature
Own properties	Static properties (e.g., operation type); dynamic behavior (e.g., branch direction); reuse distance; branch entropy
Co-running instructions	Co-running instructions' properties

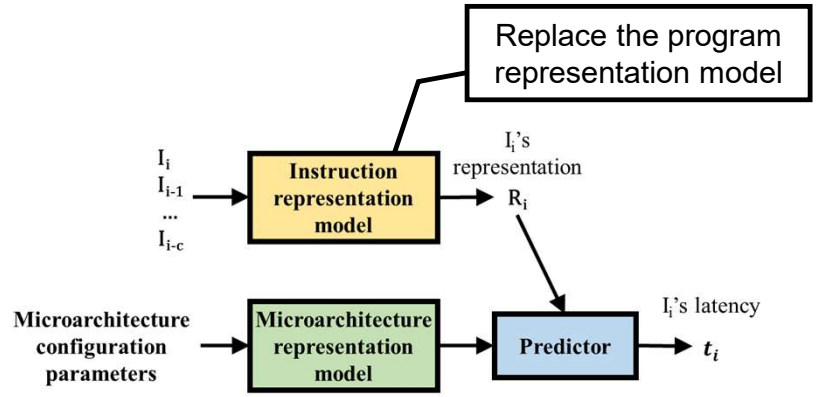


Generalizable across programs

Program traces are different combinations of instructions from the same set.

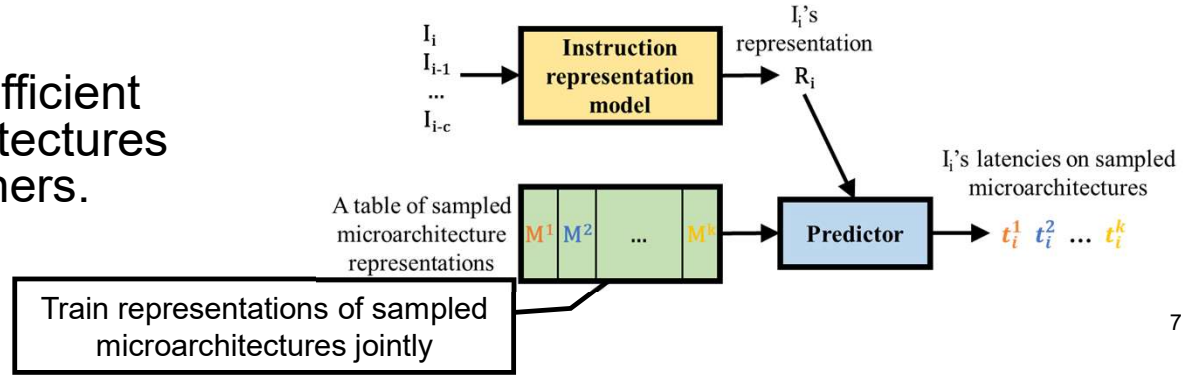
PerfVec Training

- Naïve solution: train all 3 models end-to-end
- Challenge
  - Irregular and alterable microarchitectural space
  - Difficult to train a universal microarchitecture representation model



Key idea 3: train the instruction representation model to predict instruction latencies on randomly selected microarchitectures for generalizability

- Hypothesis: training with a sufficient number of diverse microarchitectures enables generalizability to others.



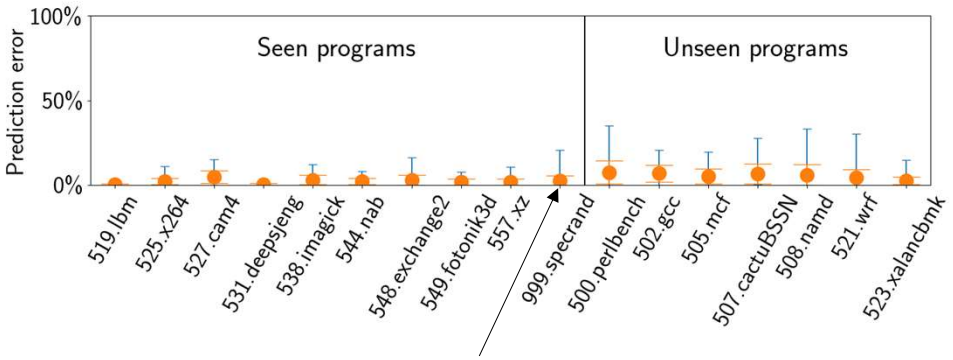
Brookhaven National Laboratory

Data Acquisition & Model Architecture

- Instruction traces from gem5 simulation
  - Easy to obtain instruction level latency
  - Easy to configure microarchitectures
- Programs
  - 17 SPEC CPU2017 benchmarks
  - 10 for training, 7 for testing
- Microarchitectures
  - Randomly generated gem5 configurations
    - In-order/out-of-order cores, caches, memory, etc.
  - 77 for training, 10 for testing
- Model architecture
  - 2-layer LSTM by default
  - See our SC24 paper for other architectures

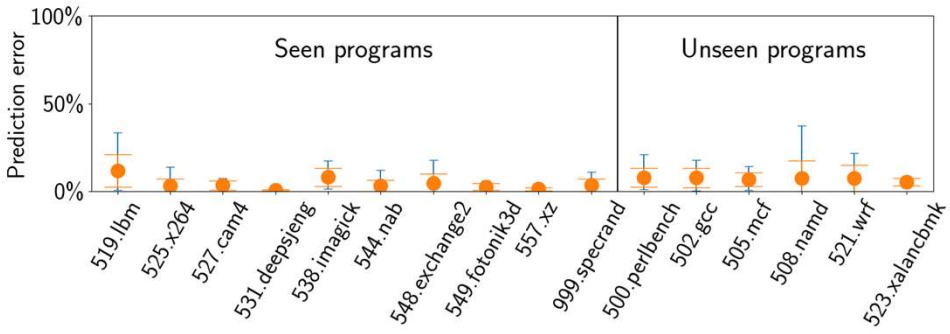
Generalizability Evaluation

Unseen programs



Prediction error range against gem5 simulation across all microarchitectures

Unseen microarchitectures



The trained model generalizes well to unseen programs and microarchitectures.

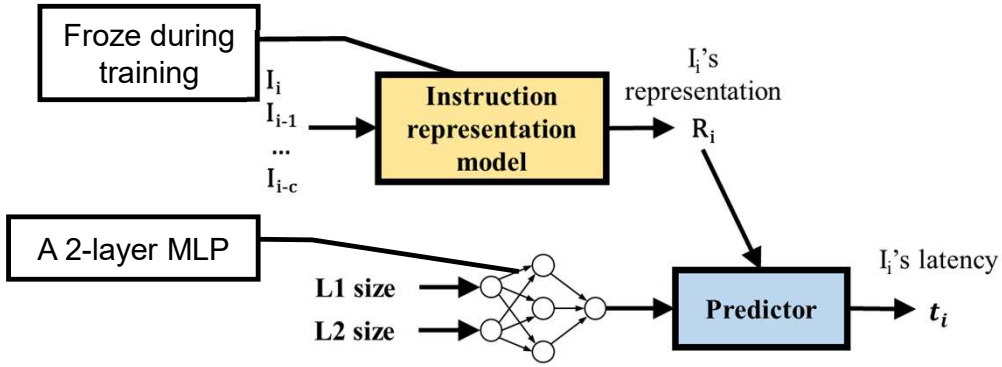
Brookhaven National Laboratory

PerfVec Use Cases

- PerfVec can be used in many tasks that require ModSim.
- A case study: design space exploration (DSE)
  - Find the optimal design(s) given one/multiple objective function(s)
- DSE example: L1 and L2 cache size exploration
  - Objective function: execution_time * (1000 + 10*L1_size + L2_size)
    - Similar to latency area product
  - Select the best cache sizes for 17 SPEC CPU2017 benchmarks

DSE Procedure

- Train a microarchitecture representation model



- Training data: gem5 simulation traces of 3 benchmarks on selected configurations
- Predict the performance of all benchmarks
  - Use the trained microarchitecture representation model and existing program representations

DSE Results

- Complete gem5 simulation: 600 hours
- Previous ML-based DSE methods
  - Use selected simulation results to train program-specific models
  - Need to simulate many configurations for each program
- PerfVec
  - Incur significantly less overhead with comparable quality
  - 11 hours = 5 (data collection) + 6 (training)

Lower is better

Method	Overhead	Quality
ASPLOS06	150	4.4%
MICRO07	84	4.7%
DAC16	170	3.6%
PerfVec	11	3.6%

Overhead: the time to construct models (hours)
Quality: how close the selected design is to the optimal design.

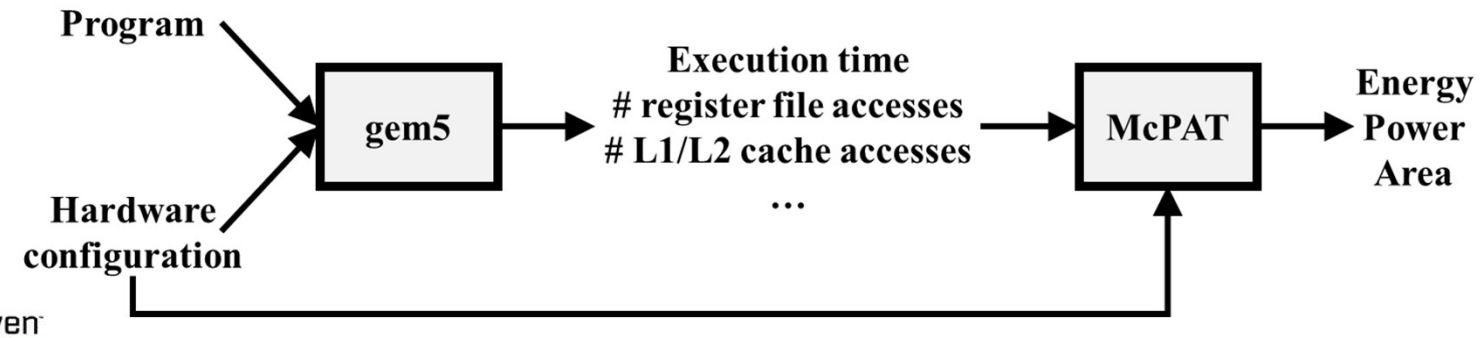
1. Ïpek et al. Efficiently exploring architectural design spaces via predictive modeling. ASPLOS 2006.
2. Dubach et al. Microarchitectural design space exploration using an architecture-centric approach. MICRO 2007.
3. Li et al. Efficient design space exploration via statistical sampling and adaboost learning. DAC 2016.

Analytical Energy Modeling

- Architecture-level analytical energy models (e.g., McPAT; Watch)

static energy — dynamic energy

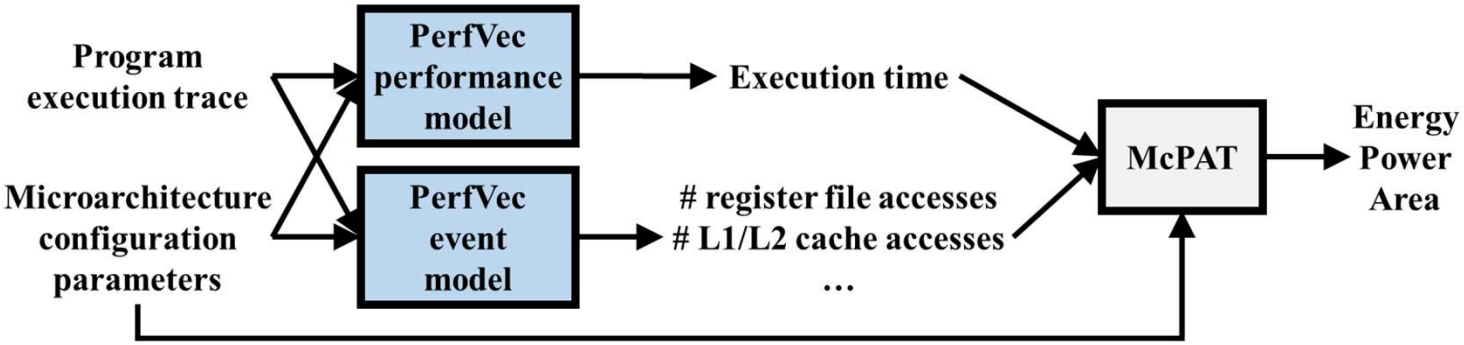
- Energy = static_power * execution_time + Σ(event_count * energy/event)
  - Events: FP multiplications, register file/ROB accesses, L1/L2 cache accesses, …
  - Static power and energy per event are architecture specific.
- Rely on slow simulation to generate execution time and event counts



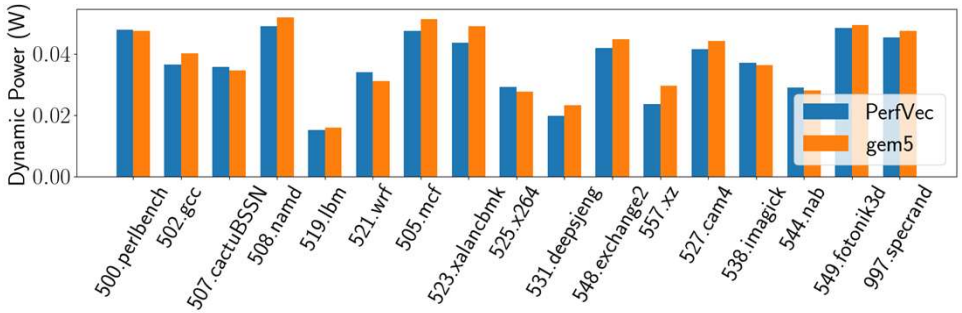
Brookhaven National Laboratory

Extend PerfVec for Energy Modeling

- Train a PerfVec event model to predict instruction event counts
  - Similar to the latency prediction models



- Average dynamic power prediction error: 8.1%
- Orders of magnitude faster



PerfVec Summary

- High generalizability
  - Learn independent program and microarchitecture representations
- Good accuracy
  - Learn program representations from instruction execution traces
- Fast speed
  - Simple combination of program and microarchitecture representations
- Many potential applications
  - Design space exploration, performance and power analysis, …
- Code: https://github.com/PerfVec/PerfVec
- Rapid fire: Kuan-Chieh and Sairam on design space exploration