ECE382N.23: Embedded System Design and Modeling

Lecture 8 – ML-Based Predictive Modeling

Andreas Gerstlauer
Electrical and Computer Engineering
University of Texas at Austin
gerstl@ece.utexas.edu

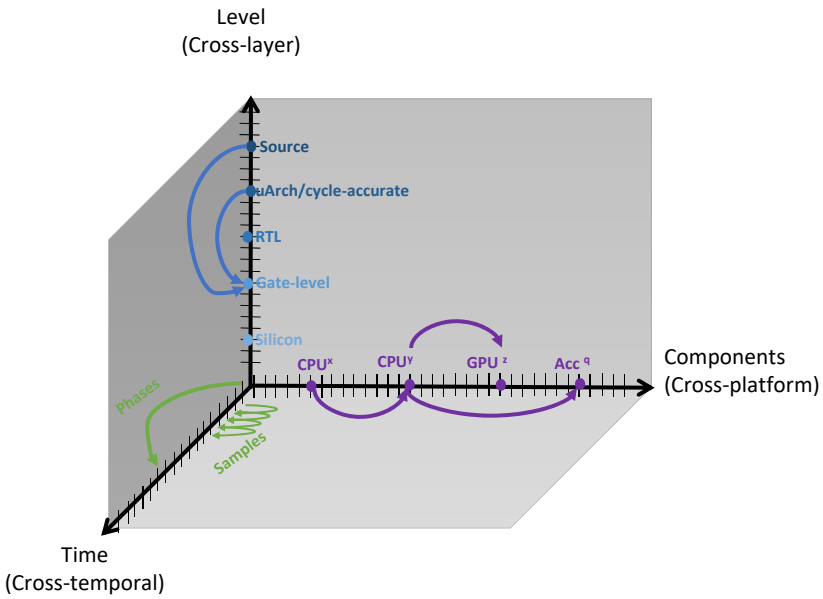
Lecture 8: Outline

- **Overview**
  - Predictive modeling dimensions
- **Cross-layer prediction**
  - Micro-architecture & source level power prediction
- **Cross-platform prediction**
  - CPU to CPU performance and power prediction
- **Cross-temporal prediction**
  - Short-term & long-term workload forecasting

Machine Learning for Modeling

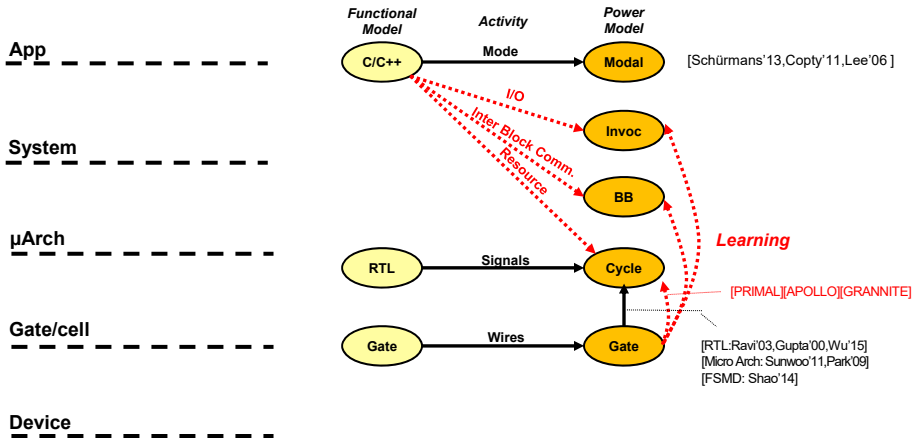
- **Learn rather than construct models**
  - Learn an abstract model from detailed observations
- **Predict rather than simulate**
  - Replace detailed simulations with predictions
- ➢ **Supervised learning**
  - Interpolate/extrapolate (complex) behavior
  - From (a few) training samples
  - Exploit inherent correlations
  - ➢ Domain-specific feature selection & learning formulations
    - ➢ Can not afford deep learning from raw data

Predictive System Modeling



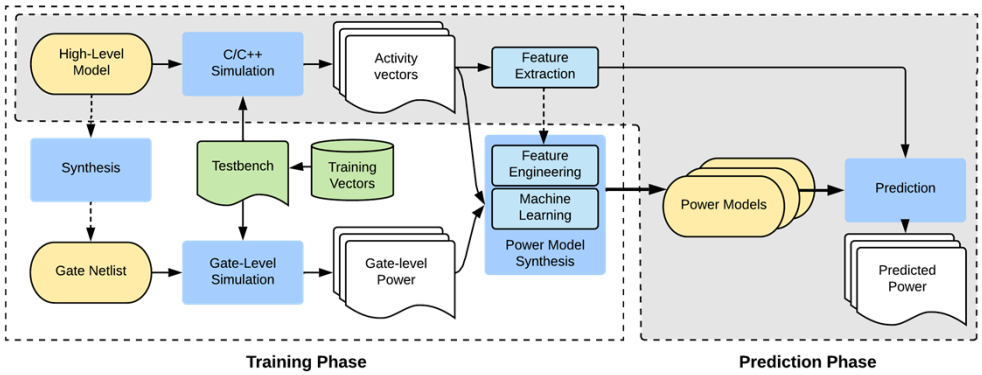
Cross-Layer Prediction

- **Power modeling example**



Cross-Layer Prediction

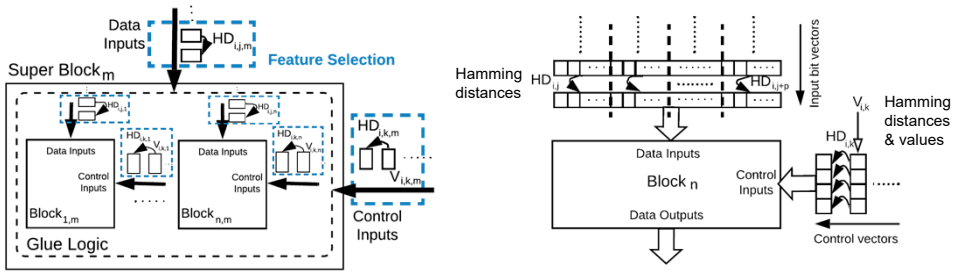
- **Power modeling example**
  - Supervised learning setting



- **Micro-architecture level prediction**
- **Source level prediction**

Micro-Architecture Level Prediction

- **Predict gate-level power from high-level CPU simulations**
  - Activity features from micro-architectural simulations
    - Block-level I/O for key data vs. control signals
    - Hamming distances & actual values
  - Hierarchically compose micro-architecture block models
    - Blocks -> Pipeline stages -> Core
    - Include glue logic model at the whole core level
  - ➢ Predict cycle- and block-level power up to complete CPU

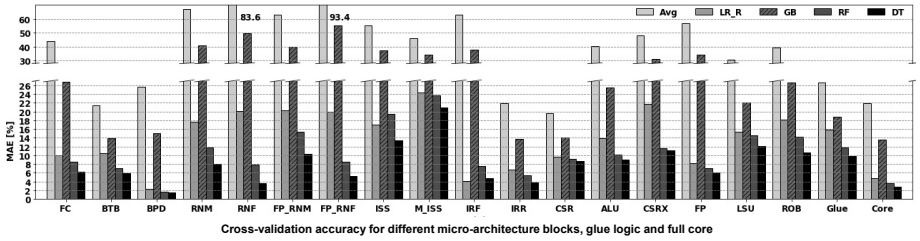


Source: A. K. Ananda Kumar,et al., "Machine Learning-Based Microarchitecture-Level Power Modeling of CPUs," IEEE TC, 2022.

Learning Setup

- **Apply advanced machine learning regressors**
  - Linear (LR), gradient boosting (GB)
  - Random forest (RF), decision tree (DT)

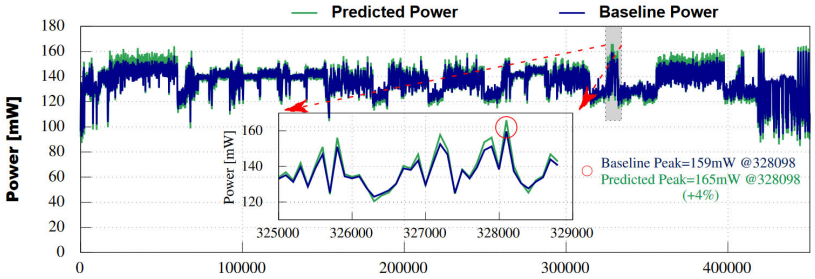
➢ **In-order and out-of-order RISC-V cores (RI5CY & BOOM)**
  - Training & cross-validation using micro-benchmarks
  - Testing using full applications (CoreMark & FFT)
  - Cycle-by-cycle power error vs. gate-level simulations



Cross-validation accuracy for different micro-architecture blocks, glue logic and full core

Micro-Architecture Prediction Results

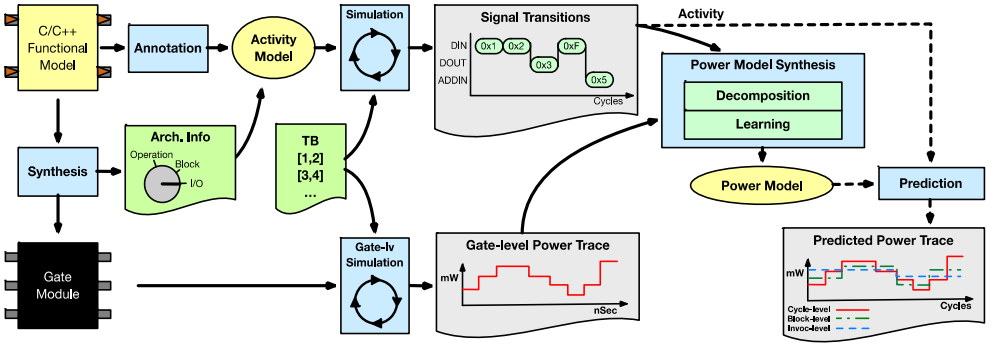
- **Cycle-by-cycle / avg. power with 3.6% / less than 1% error**
  - MAE decreases exponentially with increasing averaging



- **Very fast learning rate and low prediction overhead**
  - Models trained in less than 20k samples
  - Models predict with a throughput of 4Mcycles/s

Source-Level Prediction

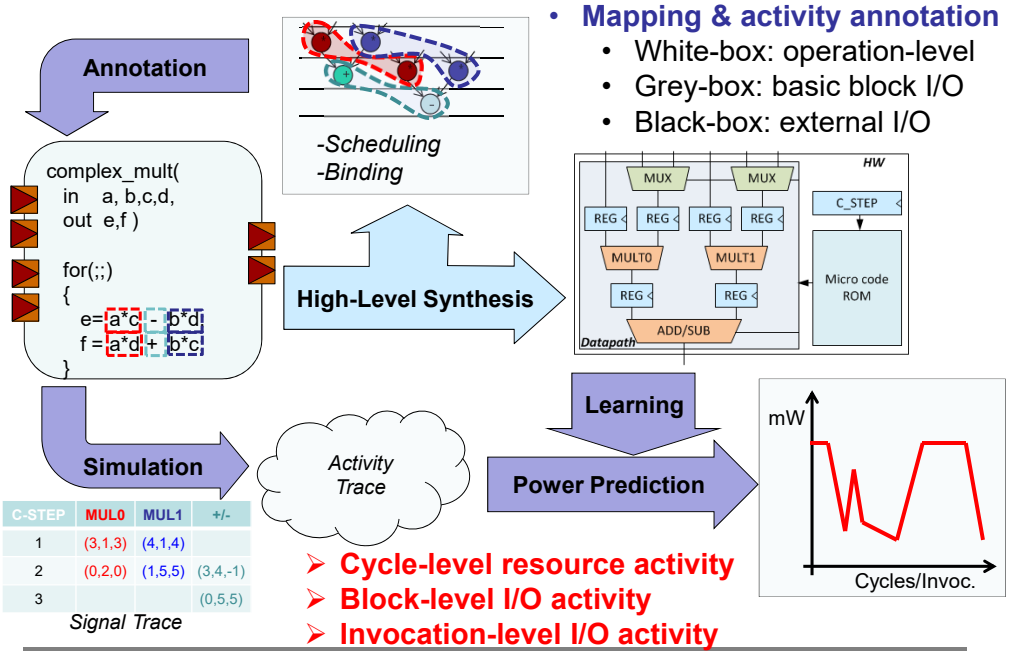
- **Custom hardware accelerator power modeling**
  - White / grey / black box hardware models
  - Operation / block / I/O activity from C/C++ simulation
  - Predict gate-level trace at cycle / block / invocation granularity



➢ **Data-dependent, Fast**

Source: D. Lee, A. Gerstlauer, "Learning-Based, Fine-Grain Power Modeling of System-Level Hardware IPs," ACM TODAES, 23(3), 2018.

Source-Level Prediction Flow

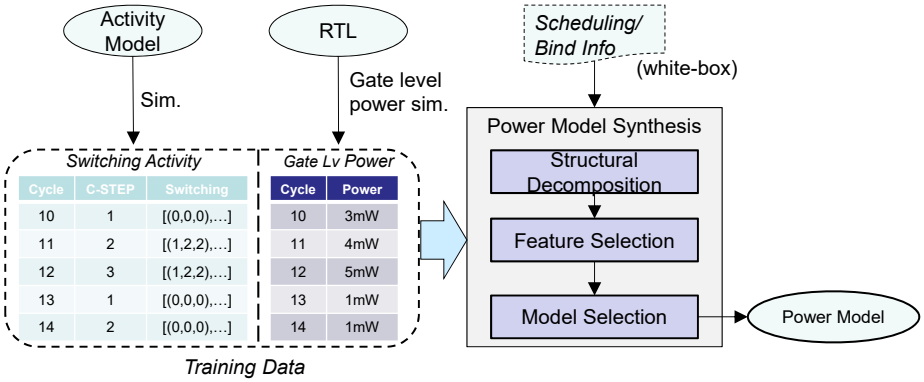


- **Mapping & activity annotation**
  - White-box: operation-level
  - Grey-box: basic block I/O
  - Black-box: external I/O

- **Cycle-level resource activity**
- **Block-level I/O activity**
- **Invocation-level I/O activity**

Power Model Synthesis

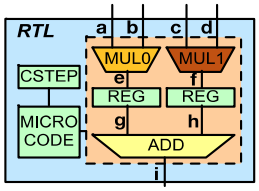
- **Learning- vs. library-based power model**
  - More accurate than library-based models (glue logic, etc.)
  - One-time training overhead using gate-level simulation



- **Apply state-of-the-art machine learning techniques**
  - Structural model decomposition & feature selection

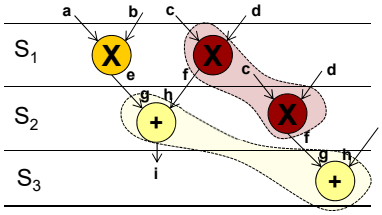
Power Model Decomposition

- **Single power model**



- White-box: cycle-level, signal activity
  $P(t)=f_{cycle}(HD_n(t)), n=\mathbf{a}...\mathbf{i}$
- Black-box: invocation-level, I/O history
  $P(t)=f_{invoc}(HD_n(t), HD_n(t\text{-}1), HD_n(t\text{-}2)), n=\mathbf{a}...\mathbf{d}$

- **Decomposed model**

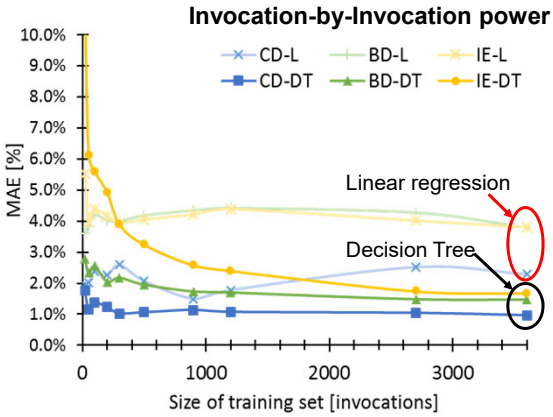


- White-box: decomposed model
  $P_{S1}(t)=f_1(HD_{\mathbf{a}}(t), \ldots, HD_{\mathbf{f}}(t))$
  $P_{S2}(t)=f_2(HD_{\mathbf{c}}(t),HD_{\mathbf{d}}(t),HD_{\mathbf{f}}(t),HD_{\mathbf{g}}(t),HD_{\mathbf{h}}(t),HD_{\mathbf{i}}(t))$
  $P_{S3}(t)=f_3(HD_{\mathbf{g}}(t),HD_{\mathbf{h}}(t),HD_{\mathbf{i}}(t))$
- Black-box: ensemble model
  $P(t)=\text{avg}(f_{S(i)}(HD_n(t), HD_n(t\text{-}1), HD_n(t\text{-}2))), n=\mathbf{a}...\mathbf{d}$

  - Only capture signals (features) utilized in each state
  - Apply feature selection to further remove correlated signals

Learning Formulation

- **Dedicated, domain-specific learning formulations**
  - Structural model decomposition & feature selection
- **Advanced, non-linear regression models**
  - *Not* deep learning (small training size)

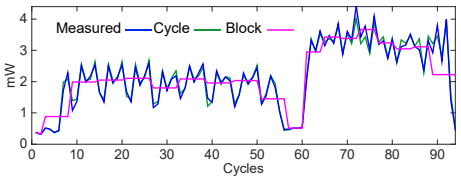


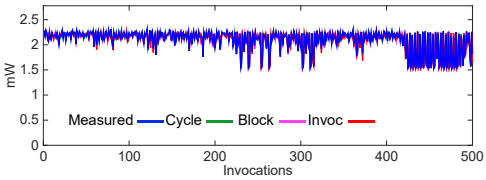
- L : Linear regression
- DT: Decision Tree regression

- CD : Cycle decomposed model
- BD : Block decomposed model
- IE : Invocation ensemble model

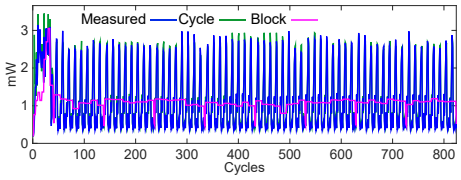
Source-Level Prediction Results

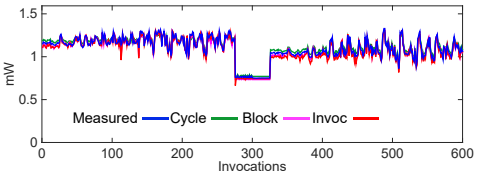
- **Pipelined 2D-DCT**
  - Cycle-by-cycle trace
  - Invocation-by-invocation trace





- **Pipelined HDR weight comp.**
  - Cycle-by-cycle trace
  - Invocation-by-invocation trace

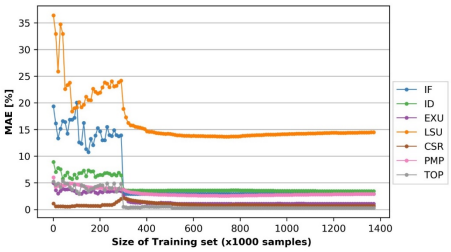




➢ **> 90-97% accuracy @ 1-10Mcycles/s speed**
  - 2,000-10,000x faster than gate-level, 100x-500x faster than RTL

Cross-Layer Prediction Questions

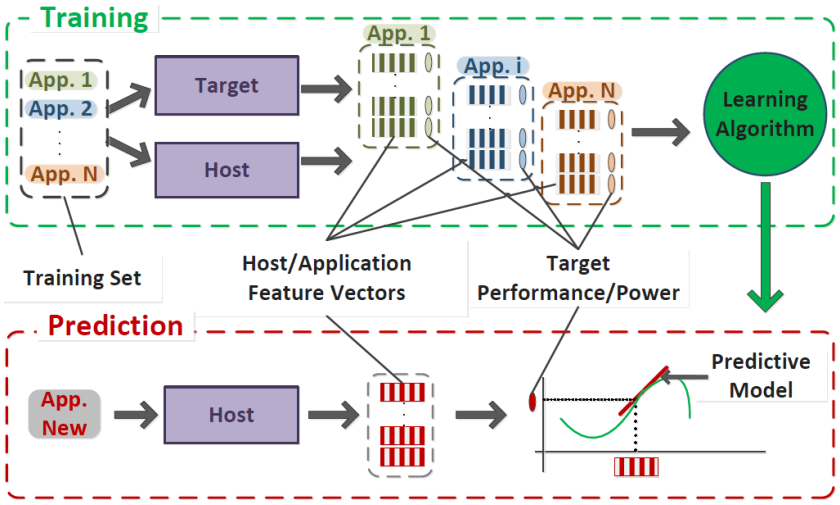
- **Training time and training set**
  - Time to collect training samples and train model less than time to simulate
  - ➢ Deep learning not an option!



- **Need for accurate reference model**
  - Modeling effort
  - Unless real hardware is available
    - May defeat the purpose of modeling

Cross-Platform Prediction

- **Power & performance prediction**



- CPU->CPU, GPU->GPU, CPU->FPGA

CPU-to-CPU Prediction

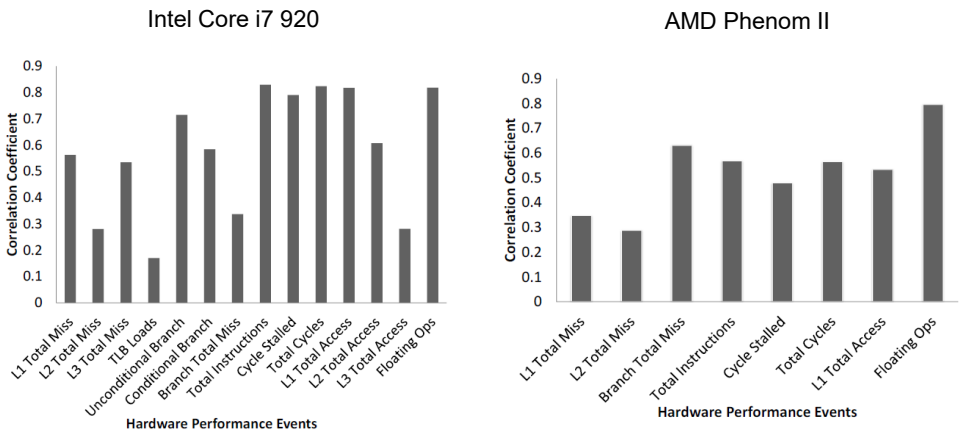
- **Predict on target CPU while running on host CPU**
  - Using hardware counters on host as features
  - Predict target performance and power
  - At program phase level

➢ **Instrumentation-based**
  - Compiler-based instrumentation at basic block granularity
  - Collect features and train/call model every *N* basic blocks

➢ **Sampling-based**
  - Source-oblivious at binary level using timer interrupts
  - Sample alignment during training

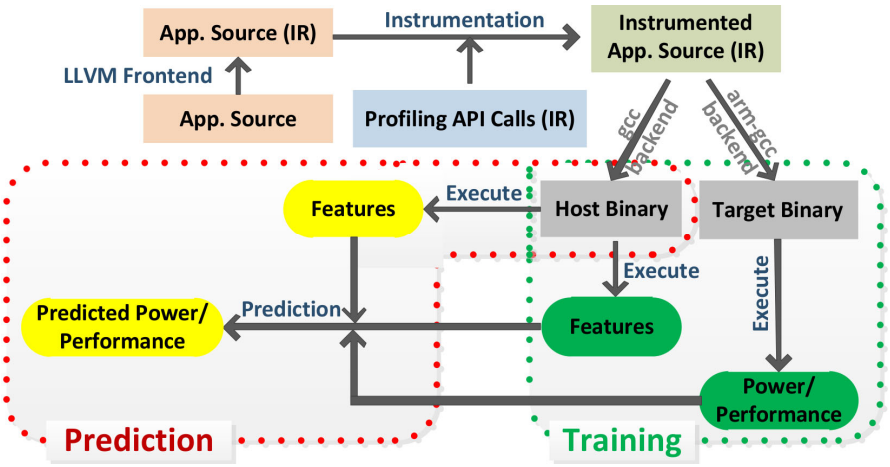
Hardware Counters as Features

- **Correlation of host counter events with target timing**
  - ARM target [GEM5] vs. counters on host [PAPI]



Instrumentation-Based Prediction

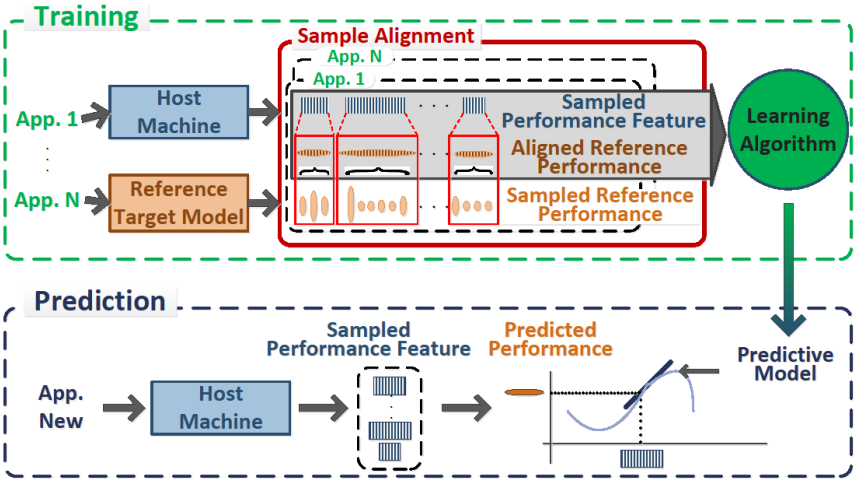
- **Every *N* number of basic blocks (BBs)**
  - Collect host counters and target metrics during training
  - Collect host counters and call model during prediction



Source: X. Zheng, L. John, A. Gerstlauer, "LACross: Learning-Based Analytical Cross-Platform Performance and Power Prediction," IJPP, 45(6), 2017.

Sampling-Based Prediction

- **Source-oblivious, binary-level prediction**
  - Transparent background prediction on timer interrupt
  - Arbitrary library, OS and system code



Source: X. Zheng, et al., "Sampling-Based Binary-Level Cross-Platform Performance Estimation," DATE, 2017.

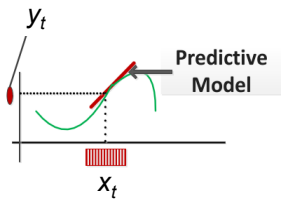
Learning Formulation

- **Given training set (x_i, y_i)**
  - $x_i \in \mathbb{R}^d$: d-dimensional counter feature vector from host
  - $y_i \in \mathbb{R}$: reference performance/power on target
- **Want to find function $F(x_i) \approx y_i$**
  - Fundamentally non-linear

➢ **Locally linear approximation $F_t(x_t)$ at input x_t**

$$F_t(x_t) = \theta_t^T x_t$$

- Around neighborhood of x_t
- LASSO regression to solve for θ_t



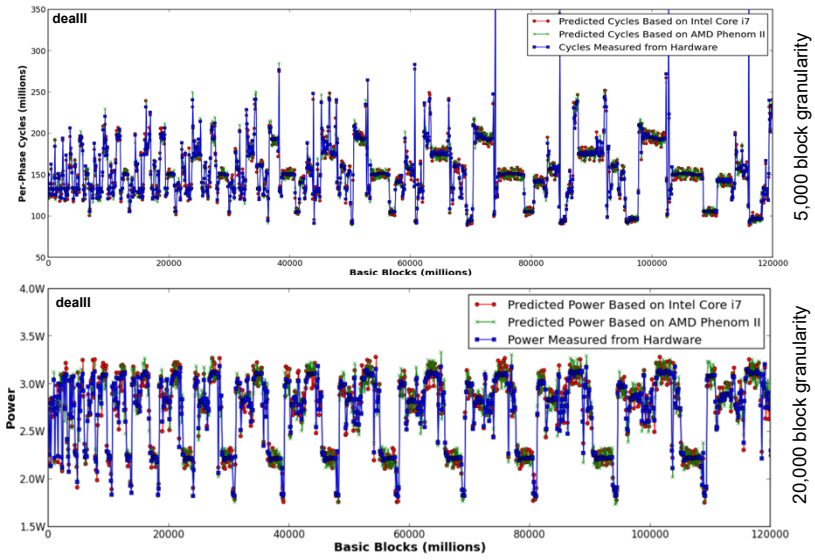
CPU-to-CPU Prediction Setup

- **Platforms**
  - Target: Samsung ARM A9/A15 Exynos
  - Host: Intel Core i7 / AMD Phenom II
- **Host counters**
  - Instrumentation-based: 14 / 8 counters
  - Sampling-based: 6 counters
- **Learning formulation**
  - Phase-level localized LASSO regression
- **Training set**
  - 157-284 programs of ACM-ICPC competition
- **Test set**
  - 7 programs from MiBench and 8 programs from SD-VBS
  - 19 programs from SPEC CPU 2006
  - 13 Java & Python benchmarks from DaCapo/PyBench

Host Counters
Instructions
Cycles
Total Cache Misses
Total Cache References
Total Branches
Total Branch Misses

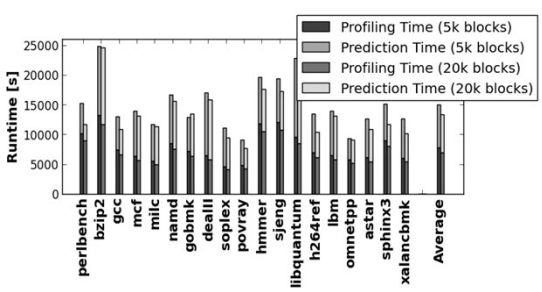
Instrumentation-Based Prediction Results

- **Performance & power prediction**
  - 90-95% per-phase accuracy @ 500-600 MIPS throughput

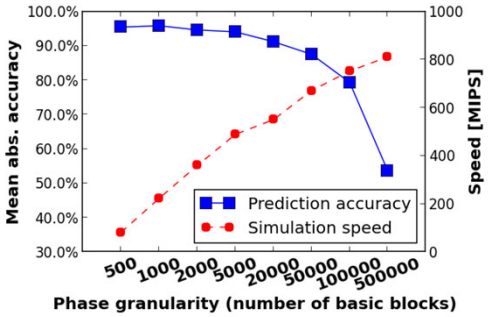


Instrumentation-Based Speed & Accuracy

- **Accuracy & speed vs. phase granularity**
  - Finer granularity requires more prediction overhead
  - But: more & better training data w/ finer granularity
    - Phase similarity: number of unique phases *decreases* linearly
  - Runtime also limited by hardware counter support on host
    - Multiple runs needed to collect all counters

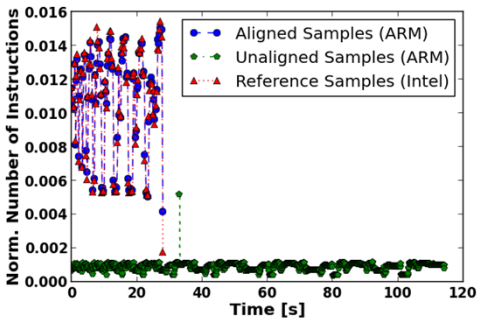


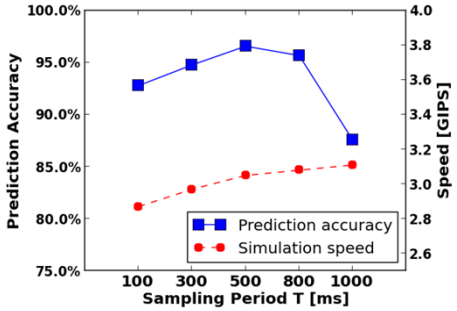
[SPEC 2006]



Sampling-Based Results

- **Speed & accuracy increase with coarser host sampling *T***
  - Better alignment, until lack of training data (*T* > 500ms)

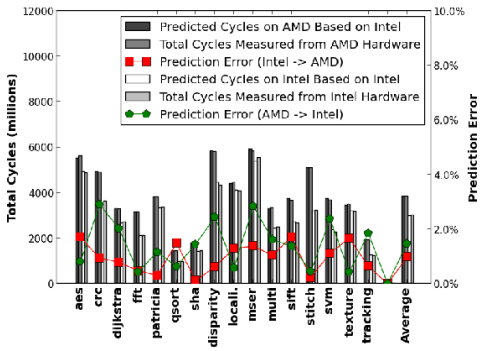


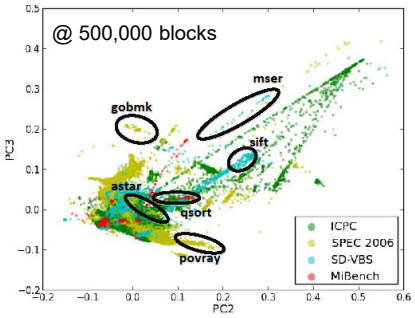


➢ **96% accuracy @ 3 GIPS (*T* = 500 ms)**
  - No instrumentation overhead (6x faster)
    - Fewer counters, coarser granularity, but requires more training
  - 2x faster than running native on ARM target

Cross-Platform Prediction Questions

- **Host/target pairs**
  - ARM from x86, x86-to-x86
  - From simple to complex?
- **Prediction features**
  - Which counters?
  - Other information?
- **Training set**
  - Larger granularity requires larger training set
  - Optimal training set?
  - ➢ Generate synthetic training set (Genesys) [SAMOS'16]



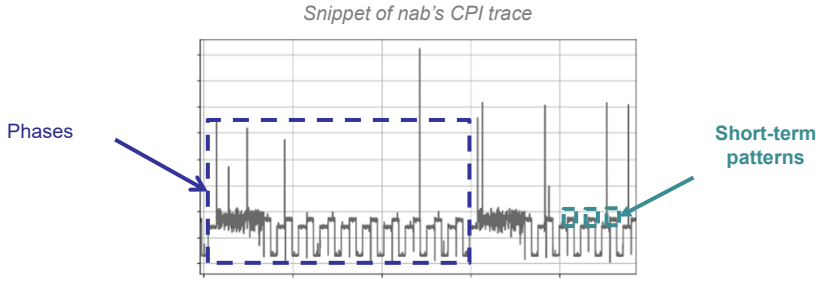


Other Cross-Platform Approaches

- **GPU performance models (Intel/UC Riverside, P. Brisk)**
  - GPU-to-GPU prediction using performance counters
  - Commercial GPUs to predict pre-silicon hardware
- **FPGA high-level synthesis models (UC Riverside, P. Brisk)**
  - Predict FPGA performance of code regions of interest
  - Running on host CPU, using hardware counters
- **Heterogeneous ISA models for OSs (UCSD, D. Tullsen)**
  - Predict performance on different CPU cores
  - Use prediction to make OS scheduling decisions
- **CPU benchmark performance models (Harvard, D. Brooks)**
  - Predict benchmark performance from CPU specifications

Cross-Temporal Prediction

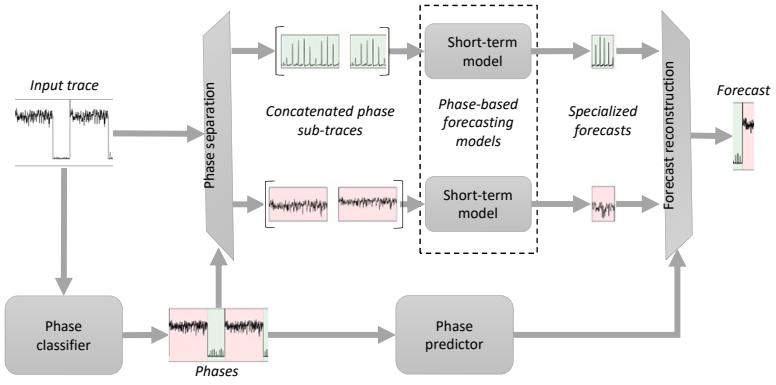
- **Dynamic program behavior prediction**
  - Programs go through phases and repetitive patterns
  - Learn patterns
  - Predict future short- and long-term behavior



➢ Proactive vs. reactive runtime optimizations
  – Voltage/frequency scaling, cache reconfiguration, etc.

Cross-Temporal Prediction

- **Predict future workload & system performance/load**



- Short-term phase-aware forecasting
- Long-term phase classification & prediction
- Combined long- and short-term prediction

Source: E. S. Alcorta, A. Gerstlauer, "Learning-Based Phase-Aware Multi-Core CPU Workload Forecasting," ACM TODAES, 2022.

Short-Term Forecasting

- **Basic workload forecasting formulation**

$$(\hat{y}_{t+1}, \ldots, \hat{y}_{t+k}) = m_w(U_{t-h+1}, \ldots, U_t)$$

  - U_t: observation of a vector of hardware counters at time t
  - y_t: counter variable of interest, e.g., CPI, at time t
  - $\hat{y}_{t+i}$: prediction of y_{t+i} made at time t
  - m_w: model function with given model parameters w
  - h: input history size
  - k: forecast horizon

➢ **Time series forecasting problem**

Time Series Forecasting Models

- **Support Vector Machines (SVM)**
  - Minimizes an error bound instead of residuals
  - Commonly used with non-linear transformations
- **Long-Short Term Memory (LSTM)**
  - Recurrent neural network
  - Popular for handling time-dependent data
- **Dynamic Linear Model (DLM)***
  - Dynamically regressive, handles non-stationary data
  - State-space model representation similar to Kalman filter
- **Matrix Profile (MP)***
  - Finds a subsequence in time series history closest to the most recent window
  - Predicts history repeats exactly the same

* Not used for CPU workload forecasting in the past

Experimental Setup

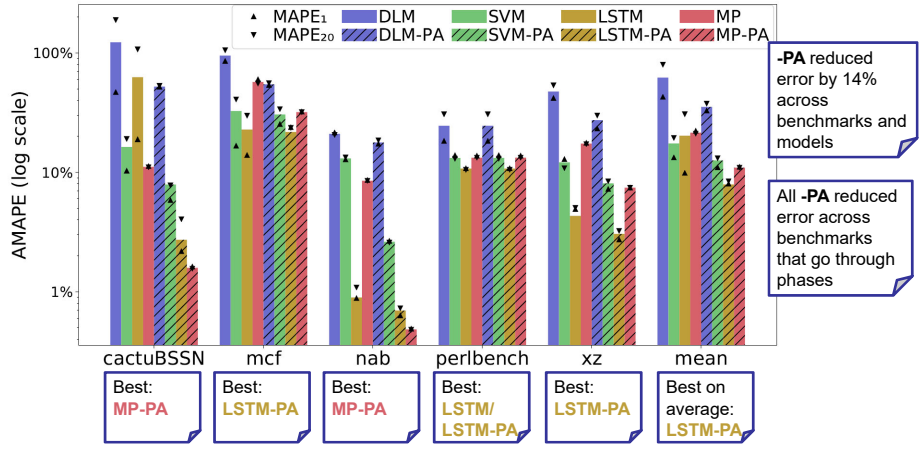
- **Data collection**
  - SPEC CPU 2017
  - Platform: Intel Xeon
  - Performance monitoring counters
  - Period: 10 ms
  - Variable of interest: CPI
  - Train-test split: 70%-30%
    - Validation: 50%-20% split of training set

Benchmark	Samples	No. of phases	Avg. ph. length	Phase behavior
cactuBSSN	202,179	5	167	abrupt transitions
mcf	52,673	5	599	hard to predict
nab	170,251	5	231	uniform pattern
perlbench	16,462	1	-	single phase
xz	126,669	4	7,037	long phases

- **Models**
  - SVM: scikit-learn, DLM: PyDLM, LSTM: Keras, MP: PySCAMP
  - Phase-aware (PA) variants using Oracle phase prediction

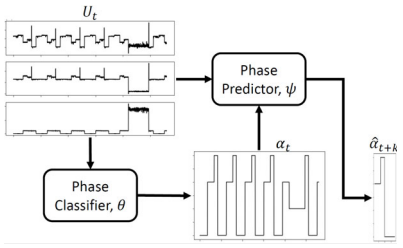
Workload characteristics	Performance monitoring counters
Memory boundedness	L2 accesses, L2 hits, L3 misses
Control flow	Total and mis-predicted branches
Operation mix	Retired FP operations
Other resources	Stall cycles, micro-operations count

Short-Term Forecasting Results



Long-Term Phase Classification & Prediction

- **Phase classification**
  - Periods of execution w/ similar behaviors
  - ➢ Unsupervised clustering

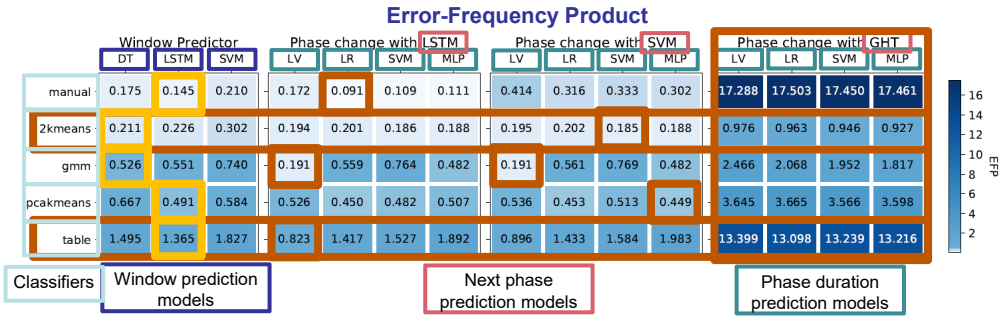


- **Phase prediction**
  - Learns phase patterns
  - ➢ Window-based time series forecasting
  - ➢ Phase-change prediction (phase duration & next phase)

➢ **Best combinations of phase classifier and predictor**

- Adopt existing classifiers to use hardware counters
- Explore advanced ML models not studied for phase prediction before

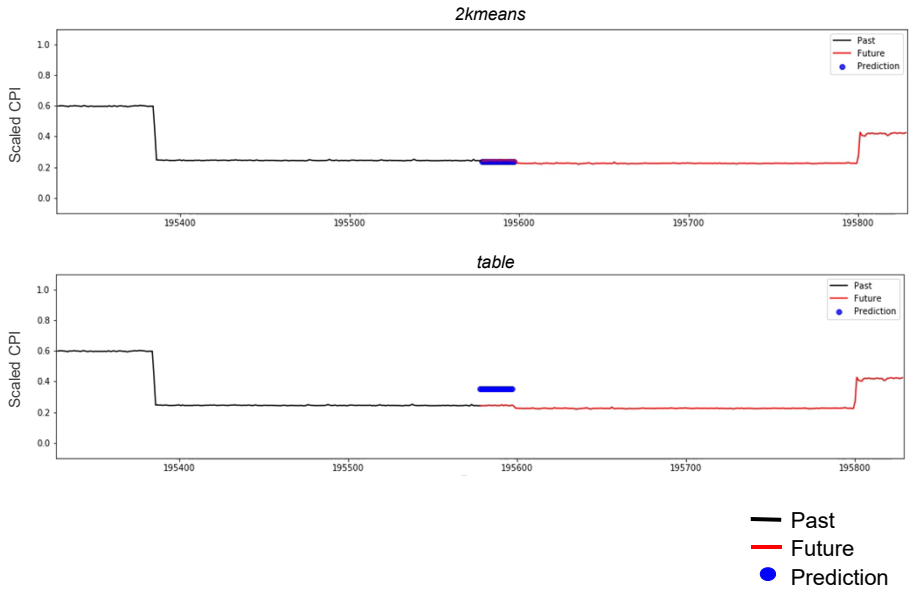
Phase Classification & Prediction Results



- Discarding *manual*, *2kmeans* is the best classifier for 13 out of 15 predictors
- Simple *table* based classifier tends to have worst EFP values
- Phase change predictor always better than a window predictor across all classifiers
- Global history table (GHT) next phase prediction shows the highest EFP values

➢ Discarding *manual*, best combination is *2kmeans* with SVM phase change models

Prediction Demo with *nab* Workload



Lecture 8: Summary

- **ML for modeling**
  - Learn, not model
  - Predict, not simulate
- **Long history of learning-based modeling approaches**
  - Various forms of regression
  - Most problems are not linear
- **Advanced machine-learning to capture complex relations**
  - Cross-layer
  - Cross-platform
  - Cross-temporal