Workload Prediction for Adaptive Power Scaling Using Deep Learning

The Harvard community has made this article openly available. Please share how this access benefits you. Your story matters

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Published Version</td>
<td>doi:10.1109/ICICDT.2014.6838580</td>
</tr>
<tr>
<td>Citable link</td>
<td><a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:12415311">http://nrs.harvard.edu/urn-3:HUL.InstRepos:12415311</a></td>
</tr>
<tr>
<td>Terms of Use</td>
<td>This article was downloaded from Harvard University’s DASH repository, and is made available under the terms and conditions applicable to Open Access Policy Articles, as set forth at <a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos">http://nrs.harvard.edu/urn-3:HUL.InstRepos</a>: dashboard.current.terms-of-use#OAP</td>
</tr>
</tbody>
</table>
Workload Prediction for Adaptive Power Scaling Using Deep Learning

Stephen J. Tarsa, Amit P. Kumar, and H.T. Kung

Harvard University, Cambridge, MA

Intel Corporation, Microarchitectures Research Lab, Santa Clara, CA

Abstract—We apply hierarchical sparse coding, a form of deep learning, to model user-driven workloads based on on-chip hardware performance counters. We then predict periods of low instruction throughput, during which frequency and voltage can be scaled to reclaim power. Using a multi-layer coding structure, our method progressively codes counter values in terms of a few prominent features learned from data, and passes them to a Support Vector Machine (SVM) classifier where they act as signatures for predicting future workload states. We show that prediction accuracy and look-ahead range improve significantly over linear regression modeling, giving more time to adjust power management settings. Our method relies on learning and feature extraction algorithms that can discover and exploit hidden statistical invariances specific to workloads. We argue that, in addition to achieving superior prediction performance, our method is fast enough for practical use. To our knowledge, we are the first to use deep learning at the instruction level for workload prediction and on-chip power adaptation.

I. INTRODUCTION

Mechanisms like dynamic voltage and frequency scaling (DVFS) enable adaptive power management, and promise to improve operating efficiency by tailoring device parameters to workloads at runtime. Such adaptation requires anticipating future circuit states, and online workload modeling is one predictive approach that minimizes static or steady-state assumptions about workloads. This flexibility is important since performance characteristics like power consumption vary widely by user and application mix [1] [2].

In this paper, we use statistical relationships learned from hardware performance counters to predict periods of low instruction throughput, during which voltage and frequency can be scaled to reclaim power. We target predictions that are long-range and low-latency, meaning that look-ahead time is maximized to allow for chip adjustment, and predictive models update quickly when workloads change.

The most popular method for workload prediction is regression [3] [4] [5], which fits a polynomial to counter measurements and extrapolates future states. However, we show that regression accuracy degrades at long ranges, as future states are unlikely to be a simple extrapolation of prior measurements. Moreover, to capture fine-grained behaviors, regression coefficients must continually be updated even if the high-level workload is the same.

Instead, we implement modeling and prediction using hierarchical sparse coding and Support Vector Machine (SVM) classification, as depicted in Figure 1. This approach first codes counter measurements in terms of a few atoms selected from a dictionary of patterns, or features, learned from training data. These feature vectors are then concatenated and coded again to capture feature interrelationships over a larger spatial scale. An SVM classifies the resulting sparse vectors with a common label when they precede an event of interest.

The process of sparse coding cuts away noise from measurement data, and improves SVM classification accuracy when data is subject to non-Gaussian variations. In addition, hierarchical models can capture statistical patterns embedded in large state spaces from a modest number of training examples. We show that training time can be reduced even more by bootstrapping feature dictionaries using a canonical set of Layer 1 features, which are common across workloads. As a result, when workloads change, only a small number of training samples are required to update our predictor.

We adopt hierarchical sparse coding to capture complicated signature patterns appearing over time, and show that this improves prediction range over regression and heuristic techniques. This is because data vectors that have been hierarchically sparse coded are classified in a transformed domain: the feature space. By performing statistical inference on feature vectors, we exploit workload-specific invariant patterns that are typically “hidden,” or not immediately observable in the raw data domain. In this paper, we use clustering on training data to extract this hidden structure. This form of deep learning has yielded major application gains in fields like computer vision, speech recognition, and machine translation (e.g. [6] [7]). To our knowledge, we are the first to apply sparse hierarchical models to chip performance data for adaptive optimization.

Fig. 1. We sparse-code vectors of counter values in two layers to extract patterns over time, and use SVM classification to identify events of interest.

II. TARGET SCENARIO AND DATA COLLECTION

A. Device, Workloads, and Event Prediction

We collect performance counter data using gem5, an architecture-level simulator with full system support, including frame buffer rendering and an interactive shell [8]. Snapshots of counter values are taken every 500µs from a simulated 1.0 GHz ARM v7a chip.

This material is based on research sponsored in part by the Intel Corporation © 2014 IEEE - ICICDT '14
These regions.

Potential way of short-circuiting ineffective predictions within
rent patterns is limited; detecting rapid phase changes is one
have few prediction opportunities, since the number of recur-
will show in Section IV B, short unstable workload regions
training to update our statistical models. Furthermore, as we
apply a threshold to one or more hardware counters to identify
optimization-driven adjustments can be sufficiently amortized.

Hierarchical sparse coding, we will find predictive signatures
these dips, which make up 7.3% of the surfing phase.

Fig. 2. A plot of committed instructions per cycle for BBENCH shows
that nearly 20% of the web surfing phase exhibits low instruction-throughput,
and is a target for DVFS. These intervals arise due to data I/O, and also
intermittently during computations, due to architecture-level interactions
chip components. We use deep learning to find predictive signatures for this
latter class of dips, which make up 7.3% of the surfing phase.

Our primary workload is the BBENCH benchmark [9],
running atop Android Gingerbread. Figure 2 plots committed
instructions/cycle for three phases of activity: OS boot, web
browser startup, and a web surfing phase. During surfing, web
sites are loaded from off-chip using Android’s built-in browser,
and Javascript code simulates user link clicks. We see that
instruction throughput drops below 25% during nearly 20%
of the surfing phase, making these intervals a good target
for DVFS. Here, low instruction throughput arises under two
sets of circumstances: first, while waiting on web page I/O,
and second, during computation-dominated intervals due to the
architecture-level interactions of chip components. This latter
class of intermittent dips make up 7.3% of the overall work-
load, and are difficult to predict with heuristics. By applying
hierarchical sparse coding, we will find predictive signatures for
these dips, and reclaim power by voltage/frequency scaling.

To generalize gains from our approach, we also report
prediction performance for the ASIMBENCH/Moby bench-
mark suite [10]. This set of workloads includes examples of a
game, audio and video playback, and document manipulation
applications running under Android ICS. Though these work-
loads capture additional application behaviors, they lack the
simulated user interactions of BBENCH that cause pertinent
statistical patterns to recur over time. We therefore present
these limited results with the caveat that incorporating user
interaction is a necessary next-step to assess power savings.

Our workload prediction scenario is related to the well-
studied problem of workload phase detection. Phase detection
is often motivated by the desire to identify large stable regions
of a workload, which ensure that overheads resulting from
optimization-driven adjustments can be sufficiently amortized.
Detection techniques like working set signatures, basic block
vectors, and conditional branch counters (see [11] for a review)
apply a threshold to one or more hardware counters to identify
deviations relative to a long-term average. We will show
that more sophisticated statistical techniques are necessary to
implement long-range prediction.

However, phase detection can serve as a useful compli-
mentory technique. This lightweight method for detecting changes
in long-term workload characteristics can trigger additional
training to update our statistical models. Furthermore, as we
will show in Section IV B, short unstable workload regions
have few prediction opportunities, since the number of recur-
rent patterns is limited; detecting rapid phase changes is one
potential way of short-circuiting ineffective predictions within
these regions.

B. Performance Counter Configuration

Our sparse-coding predictor will use deep learning to
discover signatures that span multiple measurement windows
over time, and multiple counters across the chip. Counters
capture events like committed instructions, data table hits,
misses, flushes, etc. This approach contrasts with standard
regression modeling that focuses on one or two counters most
correlated with the metric of interest, e.g. in [4].

Typically, the choice of performance counters to include on
a chip is based on both the desired performance monitoring
data, as well as layout and design constraints. In this section,
we use gem5 to also characterize the statistical properties
of counter configurations. This allows us to choose a small
number of counters that still give good prediction performance.

First, we collect data from 120 simulated hardware counters
during the execution of our test workloads. This data represents
a superset of possible hardware configurations. Every 500µs,
deltas from previous values are recorded, and we use the
resulting data vectors to calculate a correlation matrix. We
then group together counters whose correlational magnitude
exceeds 0.98, since these counters are statistically interchangeable
due to their near-total correlation. By choosing one
counter from each group, we then have a minimal configuration
that provides comprehensive architecture-level statistics.

We identify 34 different groups from the 120 possible coun-
ters studied on our ARM v7a-chip. We find, for example, that
the number of integer register reads is interchangeable with
the number of committed integer operations, though these are
collected from different locations on the chip. Such statistics-
driven counter selection is useful to control the data-collection
overhead of our predictor without sacrificing accuracy.

Principal Component Analysis (PCA), a standard technique
for dimensionality reduction, was also applied. Even though
PCA found a lower dimensional basis for our data, that
representation relies on linear combinations of all 34 counters.
Therefore, this analysis does not allow us to reduce the
number of counters deployed on-chip, though it implies that
compressive techniques such as random linear combination are
worth investigating to reduce acquisition costs [12] [13].

III. Hierarchical Sparse Coding

Sparse coding [14] formalizes dictionary learning and
feature extraction by the following minimization:

$$\min \|X - DZ\|_2 \quad \text{s.t. } |z_i| \leq k \text{ for } i = 1...t$$ (1)

with X an n x t data matrix containing t snapshots of n
counters, D an n x m dictionary of m features, Z a sparse
m x t matrix of feature coefficients, and k a sparsity constraint.
We also put non-negativity constraints on D and Z to improve
coding stability for classification under data variation [7].

Sparse coding generalizes k-means, a method for finding
clusters in data and representing vectors by their associated
cluster centroid [15]. In sparse coding, when k = 1, cluster
centroids become dictionary atoms, and data-to-cluster assign-
ments correspond to Z’s coefficients. When k > 1, a data
vector is represented by a sparse, positive, linear combination
D by iteratively fixing Z and using rank-one approximation
to update D’s columns, and then fixing D to update Z by
Orthogonal Matching Pursuit (OMP) [16]. After training, OMP computes sparse representations relative to \( D \) for new data vectors.

We choose sparse coding over alternative learning methods such as Convolutional Neural Networks for several reasons: first, a sum of commonly occurring patterns is an appropriate intuitive model for performance counters on a chip with multiple concurrently operating functional circuits. Second, sparse coding yields good classification performance using a linear SVM when few labeled training examples are available [7]. And third, both K-SVD and OMP consist primarily of inner-product computations that can be built in hardware, e.g. [17].

We sparse code data hierarchically, as depicted in Figure 1. At Layer 1, canonical features present in 500\( \mu s \) measurement windows are extracted from raw counter data. At Layer 2, we concatenate sparse feature vectors from multiple measurement windows over time, and again cluster vectors to capture feature co-occurrences. Finally, the outputs of Layer 2 are fed to a linear SVM that implements prediction by assigning a common label to vectors preceding our target event. When detecting one among many events, multiple SVMs or decision trees can be used at this last step. As previously mentioned, we convert each snapshot of counter values into a delta from the previous measurement window. Furthermore, we normalize values to lie between 0 and 1, ensuring that learning using distance minimization treats approximation error in all counters equally.

Sparse hierarchical models are a primary driver of breakthroughs associated with deep learning for three major reasons. First, imposing sparsity on signals is a powerful denoising step that is critical when dealing with natural data variations. Second, by hierarchically learning features and their interrelationships, the model can express feature combinations not directly represented in training data. This means that a few training examples can be generalized for good statistical performance on a larger data set. Third, hierarchical models often include a non-linear pooling step that corrects for alignment variations by maximizing feature response over shifted measurement patches, similar to a convolutional filter. In this paper, we found little gain from pooling since \textit{gem5}'s timing is deterministic and repeatable, however we expect this tool be important when we expand to data measured from hardware.

IV. WORKLOAD PREDICTION

A. BBENCH Performance

We first present results comparing prediction performance between hierarchical sparse coding, linear regression, and static heuristics, for detecting sub-25\% instruction-throughput dips during BBENCH. We define prediction \textit{accuracy} as the portion of all 500 \( \mu s \) windows that are correctly labeled, based on whether their instruction throughput is above or below 25\%:

\[
\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total \# of Windows Classified}}
\]

We also care about false alarm statistics, especially since there is a recovery cost associated with false positive alarms. We therefore measure \textit{precision} and \textit{recall}. Precision is the number of correctly predicted dips over the total number of alarms:

\[
\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}
\]

B. ASIMBENCH/Moby Performance

In this section, we present prediction accuracy for workloads in the ASIMBENCH/Moby benchmark suite, which include a video game, audio and video playback, and viewers.
for PDF and Microsoft Office documents. For each workload, we report the percentage of computation time during which instruction throughput is below 25%, and the prediction accuracy using signatures spanning 2 windows.

Applying a phase detection filter as discussed in Section II, we see that some workloads can naturally be divided into distinct stable regions. For example, the Frozen Bubble video game has two phases: in the first, application data is loaded and game state initialized, and in the second, the game enters a regular frame-rendering loop. We therefore break out performance per phase, and report only on workload regions with at least 8,000 measurement snapshots. This conservatively ensures that models can be trained. Benchmark descriptions and overall prediction accuracy are shown in Table I.

Hierarchical sparse coding performs best when workloads have recurring patterns. In this set of benchmarks, media workloads that are cyclic and driven by a regular sampling rate have best prediction performance. For Frozen Bubble Phase 2, MXPlayer Phase 3, and ttop Phase 2, dip prediction is nearly perfect: 94.0%, 98.7%, and 97.6%, respectively.

For Frozen Bubble and ttop, an order-8 regression yields sub-60% accuracy, indicating that cyclic dips are not extrapolations of prior counter values sampled every 500µs. Though adapting measurement sampling rate to workloads may improve regression, our method needs no such workload-specific adjustment. Furthermore, adjustment may not be possible in cases like video streaming or games, since frame rates are variable, and affected by background computation and I/O.

We also find that short phases with a high degree of irregular variation lead to few useful long-range signatures. Fitting this description are the k9 Mail benchmark, and most applications’ initialization Phase 1’s. Here, phase detection for stable-region identification could be used to short-circuit applications’ initialization Phase 1’s. Here, phase detection variable, and affected by background computation and I/O.

V. DYNAMIC POWER FOR VOLTAGE SCALING

The false negative and false positive rates of a predictor impact power savings due to missed opportunity costs and recovery costs for incorrect scaling decisions. Given these, we model realized dynamic power during instruction throughput dips when our predictive frequency/voltage scaling is applied:

\[
P_{\text{dyn}} = Pr(\text{True Pos.}) \times (V_o)^2(f_o)(a_o) + Pr(\text{False Neg.}) \times (V_o)^2(f_o)(a_o) + Pr(\text{False Pos.}) \times (V_o)^2(f_o)(a_o + a_{plt})
\]

For simplicity, this model assumes constant capacitance and linear frequency/voltage scaling. As in [18], we proportionally scale power by the amount of realized switching activity in different scenarios. \(a_o = 0.75\) represents ordinary operation, based on the observed average activity during BBENCH. \(a_{rd}\) represents reduced activity during dips in instruction throughput. Finally \(a_{plt}\) represents additional penalty activity to compensate for incorrect scaling decisions.

When instruction throughput drops, gating is used to reduce switching activity. However, gating mechanisms are imperfect and design specific, so we use a gating efficiency term \(g\) to compare different scenarios. \(a_{rd}\) is therefore defined as \(a_{rd} = 1.0 - (0.75) + g\) for an instruction throughput drop of 75%.

The model consists of terms representing power consumption for correct predictions, false negatives, and false positives, respectively. When a dip is correctly predicted, voltage and frequency are reduced from \(V_o = 1.0\) to \(V_{rd} = 0.25\), and \(f_o = 1.0\) to \(f_{rd} = 0.25\). For false negatives, voltage and frequency remain at \(V_o = 1.0\) and \(f_o = 1.0\). For false positives, we penalize incorrect scaling with increased switching activity: \((a_o + a_{plt})\). This assumes that we detect higher-than-predicted activity, and execute additional recovery steps.

Figure 4 plots \(P_{\text{dyn}}\) per dip for our best sparse coding model, against look-ahead range. Gating efficiency establishes the do-nothing baseline power consumption during a dip, so we vary \(g = 0.15...0.66\) to capture savings for a range of chip designs. Power savings are also parameterized by the recovery cost \(a_{plt}\), which we vary from +10% to +40% switching activity. When the recovery cost is +25% activity, predictive voltage scaling successfully reduces power consumption with a 4 window heads up, or 2ms. If we can tolerate only a 1ms chip adjustment time, then this savings is a 50% gain over a \(g = 0.33\) gating-efficient design without voltage scaling.

VI. DICTIONARY TRAINING AND CONFIGURATION

For low-latency prediction, we must ensure that sparse coding dictionaries can be trained quickly under changing workloads. To this end, we first demonstrate the impact of hierarchy on training time. Figure 5 shows prediction accuracy as the number of training samples increases, comparing a 2 layer hierarchical model to an equivalent 1 layer model that codes concatenated measurement vectors directly.

Prediction accuracy for both models follows the same basic shape. With few training samples, predictors capture single-window effects, and converge to a local maximum. As data is added, variations arising under underlying feature interrelationships are observed. Initially, this added data complexity dilutes the training set, hurting prediction. However, when enough samples are acquired to fully capture second-order effects, prediction converges to a new, higher maximum. Here, we see that hierarchically treating feature interrelationships reduces training data requirements by roughly 3x.
Deep learning holds much promise for workload modeling and adaptive chip optimization. We believe this study is an important first step to begin exploring those opportunities.

VII. CONCLUSION & FUTURE WORK

Three major directions for future work exist. First, we must apply this methodology to more workloads, with user interaction as a first-class characteristic. Second, we will experimentally gather data from a hardware platform to verify that predictive performance holds when timing and measurement errors seep into data. We expect non-linear pooling, mentioned in Section III, to be important in this context. Finally, predictive DVFS is a first-cut use case, and we will look at other scenarios to which we can apply deep learning. One possibility is long-range workload prediction to schedule data prefetch.

REFERENCES