ECE 752 - Advanced Computer Architecture I

THE UNIVERSITY of WISCONSIN MADISON

Aman Rakesh Chadha
(9068354597)

Abhishek Pandey
(9066383077)

Aditya Prakash
(9066452310)

1 Overview

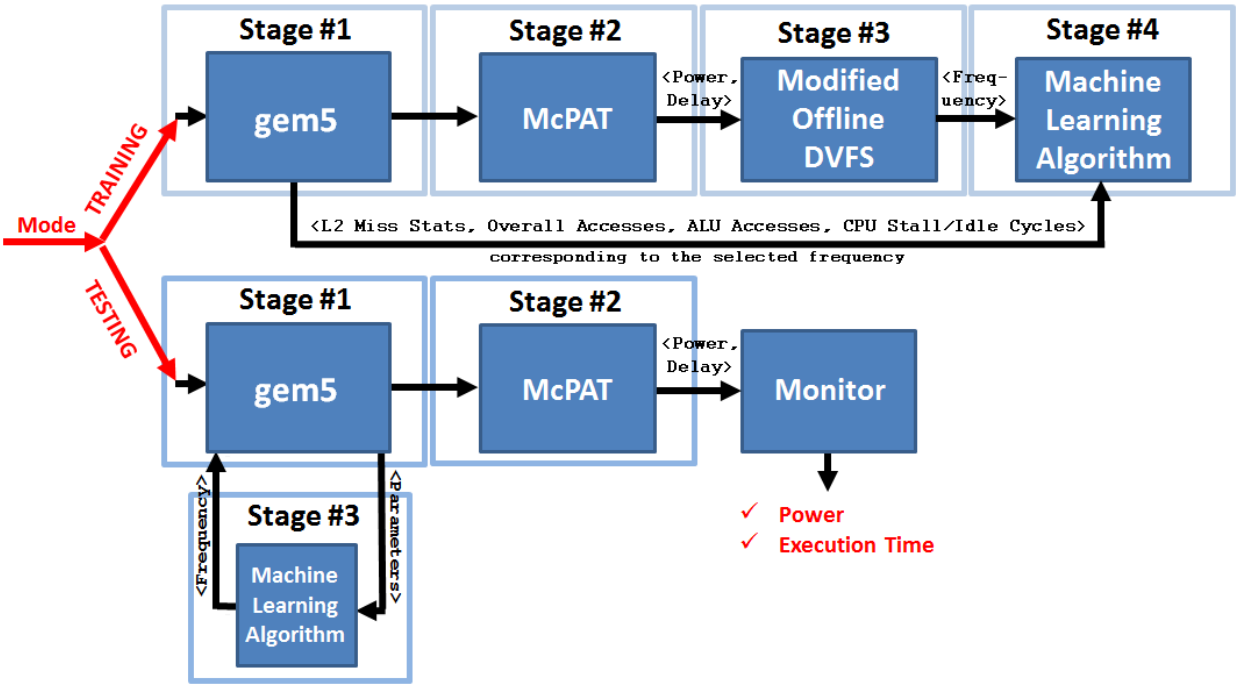

Figure 1: Schematic representation of all the modular components

a Abstract

Determining frequency scaling instants during runtime is an intricate task. In this report, we use a Machine Learning Algorithm to predict intervals where memory intensive tasks occur. A set of differentiating parameters which help to identify such opportunities are utilized. These parameters are used to train the Machine Learning Algorithm. Once trained, we capture real-time data and use the Machine Learning Algorithm to predict the optimum value of frequency based on profiled memory characteristics. This leads to substantial power savings in lieu of some increase in delay. The report summarizes our implementation details, experimental methodology, findings and results.

b Introduction

The project explores an intuitive idea of taking advantage of memory profile characteristics to build a system that 'adapts' to workload characteristics.

The MLA offers us a pathway to explore hidden information about the relationship between memory profile and the CPU frequency of operation. We apply the memory access profile using a Machine Learning Algorithm (MLA) to build rules mapping the specific memory access profile to the frequency. gem5 simulator is used in our study to simulate the hardware. It is integrated with McPAT power simulator to capture the power statistics for each defined interval. During the training period, we utilize an offline DVFS algorithm to minimize the overall power for a given delay constraint. During the testing period, the Machine Learning algorithm takes the previous interval's statistics from gem5 as input and predicts the frequency value for the next interval which can minimize the overall power usage and keep the delay within a given constraint.

c Related work

Dynamic Voltage and Frequency Scaling (DVFS) is a well-known technique to reduce energy in digital systems, but the effectiveness of DVFS is hampered by slow voltage transitions that occur

on the order of tens of microseconds. In addition, the recent trend towards chip-multiprocessors (CMP) executing multi-threaded workloads with heterogeneous behavior motivates the need for per-core DVFS control mechanisms. Voltage regulators that are integrated onto the same chip as the microprocessor core provide the benefit of both nanosecond-scale voltage switching and per-core voltage control. In [1], we see that these characteristics provide significant energy-saving opportunities compared to traditional off-chip regulators.

In [2], dynamically selecting among a set of DPM policies with a machine learning algorithm is carried out. The work leverages the fact that different policies outperform each other under different workloads and devices. The proposed algorithm adapts to changes in workloads and guarantees quick convergence to the best performing policy for each workload. Experiments were performed with a policy set representing state of the art DPM policies on a hard disk drive and a WLAN card. The results show that our algorithm adapts really well with changing device and workload characteristics and achieves an overall performance comparable to the best performing policy at any point of time.

2 System Architecture

Building Blocks

The system consists of:

1. *Phase 1:* gem5 + McPAT
2. *Phase 2:* ILP + Offline DVFS
3. *Phase 3:* Machine Learning

The following sections describe these 3 phases in detail.

2.1 Phase 1: gem5 + McPAT

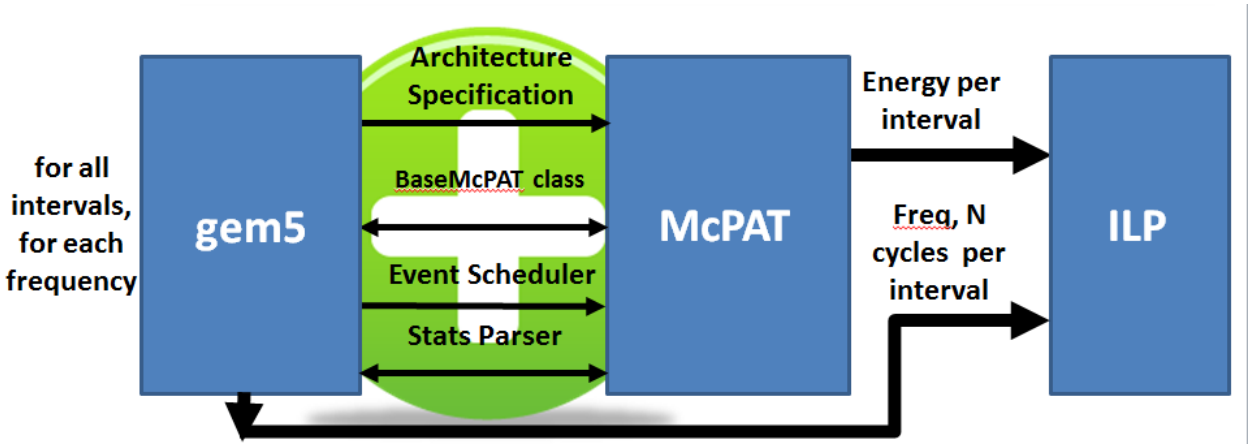

Figure 2: gem5 interface with McPAT

Integration

We integrated the McPAT source code into gem5 such that we can obtain energy statistics for each interval. When the simulation begins, the architectural specifications of the simulated machine are passed to the McPAT interface as an XML file. This specification includes attributes like number of cores, cache sizes, number of floating point, integer ALU, instruction issue width, rob width and so on. We incorporated hooks into the McPAT source which are shown in Figure 3. A new class called BaseMcPAT was created which inherits class SimObject so that we can make use of the Event Scheduler defined in SimObject class. We schedule a recurring mcpat event every N cycles. In this event, we feed interval stats like CPU idle cycles, L2, L1 miss, delay statistics, number of committed instructions (both integer and FP) and so on. These statistics are fed to McPAT and energy is computed for that interval. Power is computed from the current CPU frequency and number of cycles N. These power and delay statistics are passed to the offline DVFS algorithm during the training phase. The same benchmark is run for multiple frequencies, the delay and power stats are parsed into offline DVFS algorithm which chooses the optimum frequency for each interval for a given constraint in order to minimize the overall power.

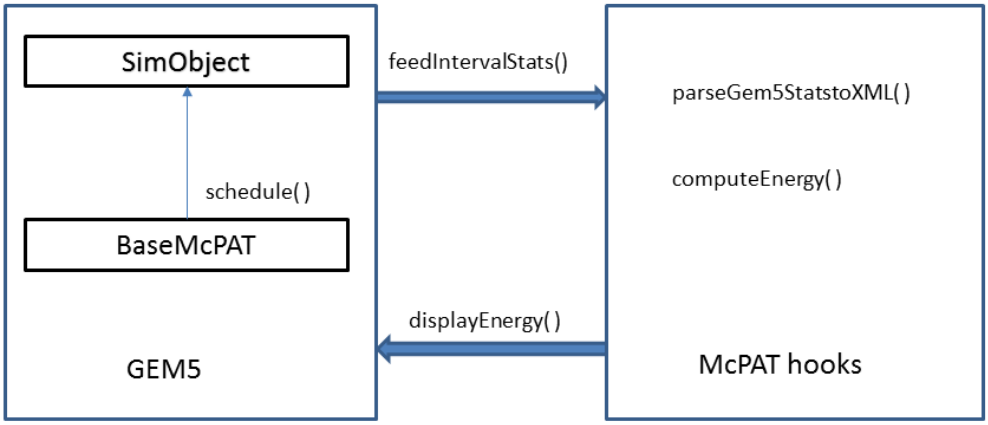

Figure 3: Essential subroutines in McPAT interface with gem5

2.2 Phase 2: ILP + Offline DVFS

Figure 4: Offline DVFS components

a Original DVFS Algorithm[1]

We adopt the offline DVFS algorithm proposed by Kim et. al. in "System Level Analysis of Fast, Per-Core DVFS using On-Chip Switching Regulators" (HPCA'08).

The goal of any DVFS algorithm is to *minimize energy consumption* of the application within certain *performance constraints*.

This can be done by exploiting the slack due to asynchronous memory events. *Scaling down the frequency* of the processor *slows down CPU-bound operations*, but does not affect the time taken by *memory-bound operations*.

We *exploit the presence* of such memory-bound intervals to reduce the *voltage* and *frequency* of the processor. Dynamic frequency scaling *reduces the number of instructions* a processor can issue in a given amount of time, thus reducing performance. Hence, it is generally used when the workload is *not CPU-bound*, i.e., when it is memory bound (or otherwise *does not depend on the CPU clock frequency*).

The DVFS control problem is formulated as an Integer Linear Programming (ILP) optimization problem, which seeks to reduce the total power consumption of the processor within specific performance constraints (δ).

[1] The overheads associated with switching between different frequency settings are not considered due to time limitations.

b Choice of optimizer algorithm

We used *Gurobi Optimizer 5.5* to solve this ILP optimization problem.

The Gurobi Optimizer is a commercial optimization solver for linear programming (LP), quadratic programming (QP), quadratically constrained programming (QCP), mixed integer linear programming (MILP), mixed-integer quadratic programming (MIQP), and mixed-integer quadratically constrained programming (MIQCP).

Gurobi was designed from the ground up to exploit modern architectures and multi-core processors, using advanced implementations of the latest optimization algorithms. Gurobi wins in public benchmark tests vs. solvers such as CPLEX and XPRESS (Solver). Specifically, in benchmark tests Gurobi has the fastest times among linear programming solvers, the fastest times among mixed-integer programming solvers, the fastest times for finding feasible solutions, the fastest times for detecting infeasibility, the fastest times for solving mixed-integer (QC)QP problems.

c ILP Formulation and Solving procedure

The application runtime is divided into *N intervals*. A total of $L = 4$ frequency (V/F) levels are considered. For each runtime interval i and frequency j, the power consumption, P_{ij}, is calculated. The delay for each interval and V/F level, D_{ij} is also calculated. Heuristics for the delay of individual intervals are obtained by calculating the relative memory-boundedness of each interval through cache miss behavior. The set of P_{ij} and D_{ij} values represent the *input set*.

The below equations specify the ILP formulation of the offline DVFS algorithm:

$$\min\left(\sum_{i=1}^{N}\sum_{j=1}^{L} P_{ij}x_{ij}\right)$$

$$\left(\sum_{i=1}^{N}\sum_{j=1}^{L} D_{ij}x_{ij}\right) < \delta$$

$$\sum_{i=1}^{N}\sum_{j=1}^{L} x_{ij} = N$$

. . . where x_{ij} is a binary variable having values 1 or 0.

The selection of the value of the binary variable on the basis of these constraints, leads to the *output set*.

We consider an out-of-order (O3/OoO) processor with the capability of switching between four frequencies switching between *1GHz, 1.3GHz, 1.6GHz and 2GHz*.

We consider performance constraints of 10% and 15%. In order to keep the runtime overheads of the ILP algorithm tractable, we divide the simulation trace into smaller windows of 30,000 cycles each; finding optimal DVFS assignments within the windows, but not necessarily across the entire trace. Given below are the code snippets for the minimization and constraint equation:

```
for i in Iset:
   for j in Jset:
      x[i,j] = m.addVar(vtype=GRB.BINARY,obj=P[i][j],name='x_%s_%s' % (i, j))

m.addConstr(
```

```
        quicksum(DIJ[i,j] * x[i,j] for i in Iset for j in Jset) <= 73230.03,
            'capacity' )
```

d **Modifications to the adopted algorithm**

Instead of placing a constraint on all intervals as a whole by using the following equation: $\sum_{i=1}^{N}\sum_{j=1}^{L} x_{ij} = N$, we place a constraint on each interval by using $\sum_{j=1}^{L} x_{ij} = N \quad \forall \; i = 1 \; to \; N$.

This ensures selection of a new binary variable x_{ij} for each interval. Using the original equation could lead to cases when two or more values of x_{ij} be selected in a single interval. Below is the code snippet.

```
for i in Iset:
  m.addConstr(
      quicksum(x[i,j] for j in Jset) == 1, eqn2_%s_%s' %(i,j))
```

2.3 Phase 2: Machine Learning

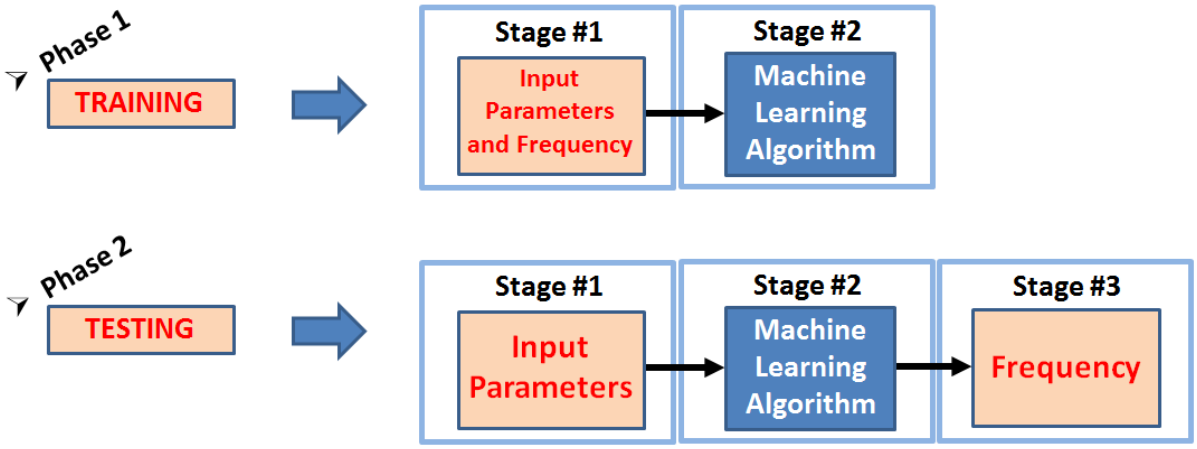

Figure 5: Block Diagram describing the Machine Learning Components

a **Algorithm choice**

We use a multi-class Support Vector Machine (SVM) to play the role of an MLA. These are basically supervised learning models with associated learning algorithms that analyze data and recognize patterns, used for classification and regression analysis. The basic SVM takes a set of input data and predicts, for each given input, which of two possible classes forms the output, making it a non-probabilistic binary linear classifier. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.

SVMs are the state-of-the-art classifier in many applications and have become ubiquitous thanks to the wealth of open-source libraries implementing them.

An MLA is thus used to build rules that map *memory profile* to an optimistic configuration of *frequency* (Frequency Scaling) based on the evaluation of performance.

b **SVM Internals**

Case I: The linearly separable case (maximum margin)

In SVM, the class of an input vector x can be decided by evaluating the sign of $y(x)$.

$$y(x) = w^T \phi(x) + b$$

If $y(x) > 0$ we assign x to class +1 and if $y(x) < 0$, we assign it to class -1. Here $\phi(x)$ is a feature-space transformation, which can map x to a space of higher, possibly infinite, dimensions.
Given a data set comprising N input vectors and their corresponding labels $t_1, \ldots, t_n$, where $t_n \in \{-1, +1\}$, we would like to find w and b such that it explains the training data: $y(x_n) \geq 1$ when $t_n = +1$ and $y(x_n) \leq 1$ when $t_n = -1$. This can be rewritten in a single constraint:

$$t_n(w^T \phi(x) + b) \geq 1, \; n = 1, \ldots, N$$

In addition, w and b are chosen so that the distance between the decision boundary $w^T \phi(x) + b = 0$ (a line in the 2-d case, a plane in the 3-d case, a hyperplane in the n-d case) and the closest points to it is maximized. This distance is called the margin, hence the name *maximum margin* classifier. Geometrically, the margin is found to be $2 / \|w\|^2$ and so the maximum margin problem can be equivalently expressed as the minimization problem:

$\arg\min_{w,b} \frac{1}{2}\|w\|^2$ subject to constraint 1.

In the figure below, the decision line is the plain line and the margin is the gap between the two dotted lines. Only 3 support vectors (green dots) out of 180 training examples are necessary.

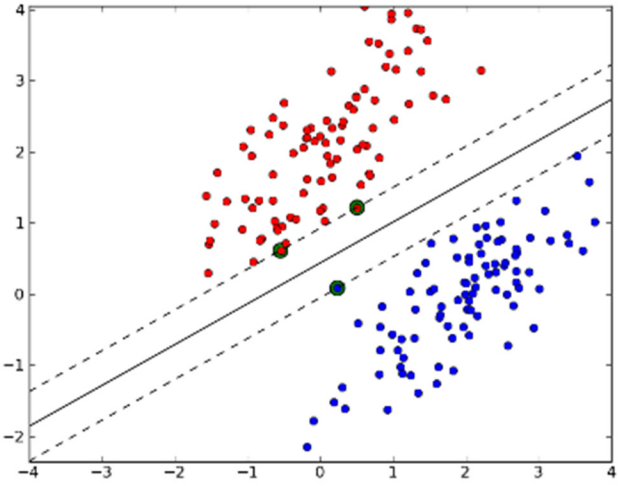

Figure 6: SVM based Classification between 2 groups

Case II: The non-linearly separable case

Since this is a constrained optimization problem, we can introduce Lagrange multipliers $a_n \geq 0$ (one per training example), differentiate the Lagrangian function with respect to *w* and *b* and inject the solution back in the Lagrangian function, so that it depends only on the Lagrange multipliers. Doing so, we find that the maximum margin equivalently emerges from the maximization of:

$$\tilde{L}(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(\mathrm{x}_n, \mathrm{x}_m)$$

subject to the constraints, $a_n \geq 0, \; n = 1, \; \ldots, N$

$$\sum_{n=1}^{N} a_n t_n = 0$$

This is the so-called *dual representation* and is a quadratic programming (QP) problem.

$k(\mathrm{x}_n, \mathrm{x}_m) = \phi(\mathrm{x}_n)^{\mathrm{T}} \phi(\mathrm{x}_m)$ is called the kernel function.

Similarly to the objective function, *y(x)* can also be re-expressed solely in terms of the Lagrange multipliers.

$$y(x) = \sum_{n=1}^{N} a_n t_n k(\mathrm{x}, \mathrm{x}_n)$$

The important thing to notice here is that we've gone from a sum over *M* dimensions (the dot product in equation (7.1)) to a sum over *N* points. This may seem like a disadvantage as the number of training examples *N* is usually bigger than the number of dimensions *M*. However, this is very useful and is called the kernel trick: this allows to use SVM, originally a linear classifier, to solve a non-linear problem by mapping the original non-linear observations into a higher-dimensional space, but without explicitly computing $\phi(x)$.

In many situations, only a small proportion of the Lagrange multipliers a_n will be non-zero, therefore we only need to store the corresponding training examples x_n. These are called the *support vectors* and this is why SVMs are sometimes called sparse kernel machines.

That being said, in the linear case, i.e. when $\phi(x) = \mathrm{x}$, it is faster to directly compute *y(x)* from equation (7.1). *w* and *b* can be computed in terms of the Lagrange multipliers.

The figure below shows an example of a Gaussian kernel with parameter sigma=5.0. Perfect prediction is achieved on the held-out 20 data points.

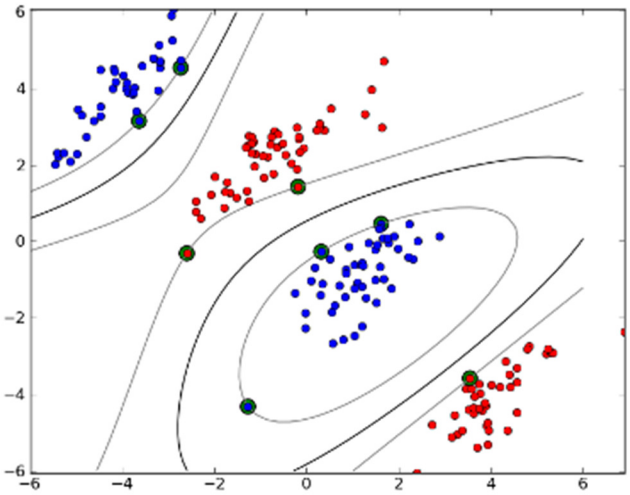

Figure 7: An example of a Gaussian kernel with parameter sigma=5.0

b **Training**

The MLA generates a model on the basis of the input parameters and frequency.
The format of training and testing data file is:

<label> <index1>:<value1> <index2>:<value2> . . .
.
.
An example of one such set of values corresponding to a particular time interval is as follows:

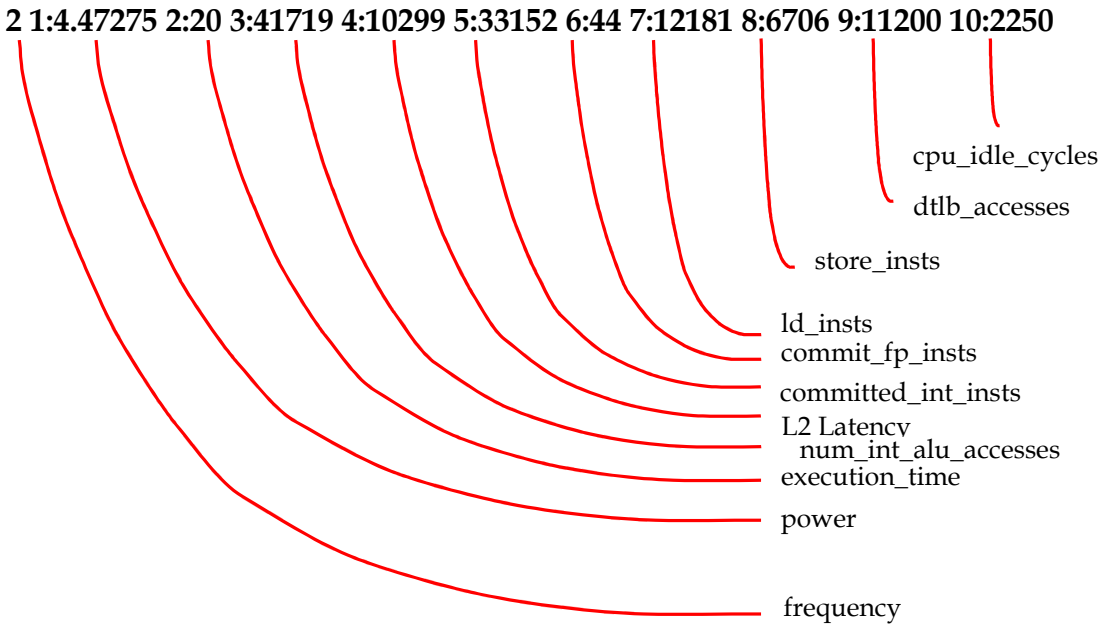

Figure 8: Explanation of all input parameters to the machine learning algorithm

Each line contains an instance and is ended by a '\n' character. For classification, <label> is an integer indicating the class label (multi-class is supported). For regression, <label> is the target

value which can be any real number. The pair <index>:<value> gives a feature (attribute) value: <index> is an integer starting from 1 and <value> is a real number.

Labels in the testing file are only used to calculate accuracy or errors. If they are unknown, just fill the first column with any numbers.

We developed a custom script to obtain values from the gem5+McPAT integrated model and write them in the above format.

The MLA generates a 'model' (with feature vectors) at the output and writes it to a file.

c **Prediction/Testing**

Using the 'model' generated in the training phase, prediction is performed for values/parameters in the same format as above.

d **The nexus between gem5+McPAT and Machine Learning**

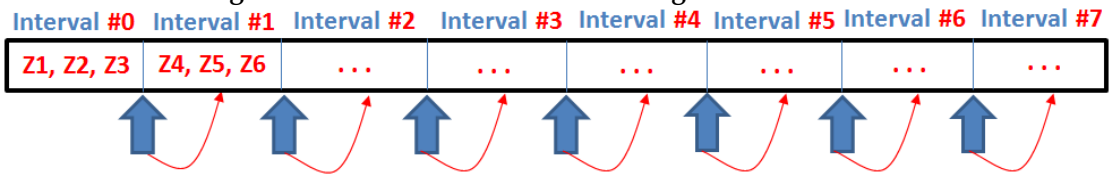

Figure 9: gem5+McPAT with Machine Learning

The parameters obtained at the end of an interval are passed on to the MLA and the resulting frequency output is used in the next interval.

2.4 Implementation Details

a **Training phase**

For training, the flow of data is as follows:

1. An integrated model of gem5 and McPAT is used.
2. McPAT generates Power and Delay values which are passed on to the Modified DVFS (MDVFS) Algorithm.
3. The MDVFS algorithm, in turn, selects a particular set of Power and Delay values and accordingly decides the corresponding frequency.
4. This frequency value is passed on to the MLA.
5. gem5 passes on all the parameters <L2 Miss Stats, Overall Accesses, ALU Accesses, CPU Stall/Idle Cycles> to the Machine Learning Algorithm.
6. With inputs as the set of parameters and the frequency, the MLA is trained.
7. The MLA generates a 'model' (with feature vectors) which is used during prediction/testing.

b **Testing phase**[2]

For testing, the flow of data is as follows:

[2] It should be noted that the *offline DVFS algorithm* is used only during the training phase to determine frequency values corresponding to power and delay parameters. It is not a part of system flow during the testing phase.

1. gem5 passes on the set of parameters to the MLA which outputs a frequency value based on the trained model.
2. gem5 also feeds McPAT which generates Power and Delay values which are monitored.

3 Benchmarking Methodology and Results

a Test-bed specifications

Parameter	Specification
ISA	ALPHA
Execution Mode	Out of Order Execution CPU
Base Frequency	2 GHz
L1 I-Cache and D-Cache Size	32 kB
L2 Cache Size	256 kB
Number of cores	1
Other parameters at their default values	

b Memory/CPU Boundedness of Benchmarks

Amongst SPEC2000 benchmarks, an application is considered to be memory-bound if the stall time reported is 30% or higher for a 512kB 8-way set-associative L2 cache configuration [3].
Thus, the following benchmarks are categorized as Memory-bound:

1. 168.wupwise
2. 171.swim
3. 173.applu
4. 175.vpr
5. 179.art
6. 181.mcf

c Parameters chosen

We chose the following parameters to profile memory characteristics of a workload.
Basically, we aimed at identifying the memory-boundedness of the workload in a particular interval of execution.

Parameter	
L2 Miss Stats	**Functional Unit Busy Rate**
# of FPU Accesses	**Miss Latency for L2**
# of ALU Accesses	**CPU Idle Cycles**

d Results

We trained the MLA using the results from *mcf* which are shown as the base-line normalization reference (ILP) in the figure below. We test-ran the system to predict using GCC and OCEAN as benchmarks. Comparison of all results is shown in the following two figures.

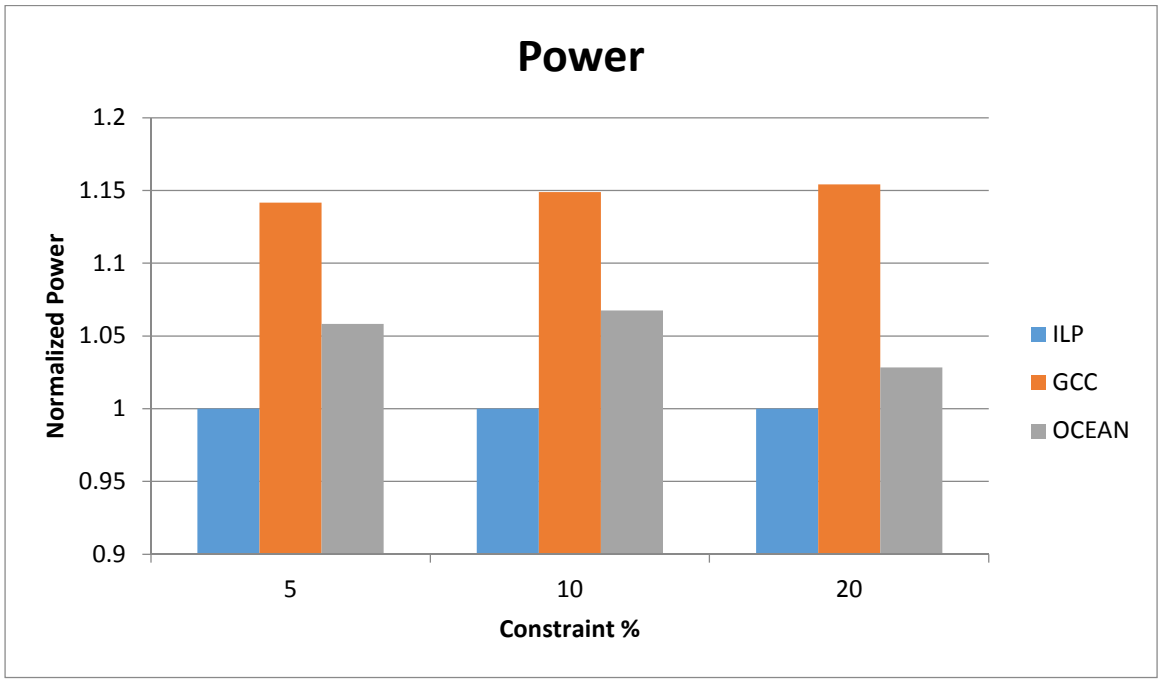

Figure 10: Plot of power v/s constraint for ILP, GCC and OCEAN

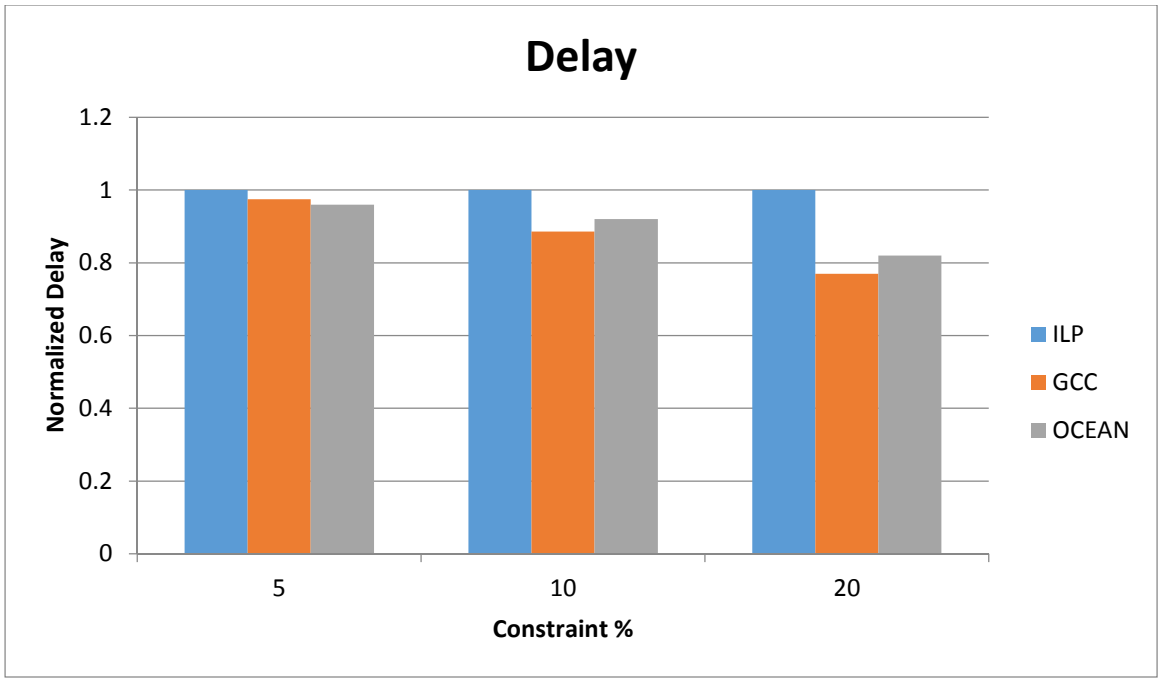

Figure 11: Plot of delay v/s constraint for ILP, GCC and OCEAN

e **Comparison of GCC running with and without DFS**
Constraints: 5%, 10%, 20%

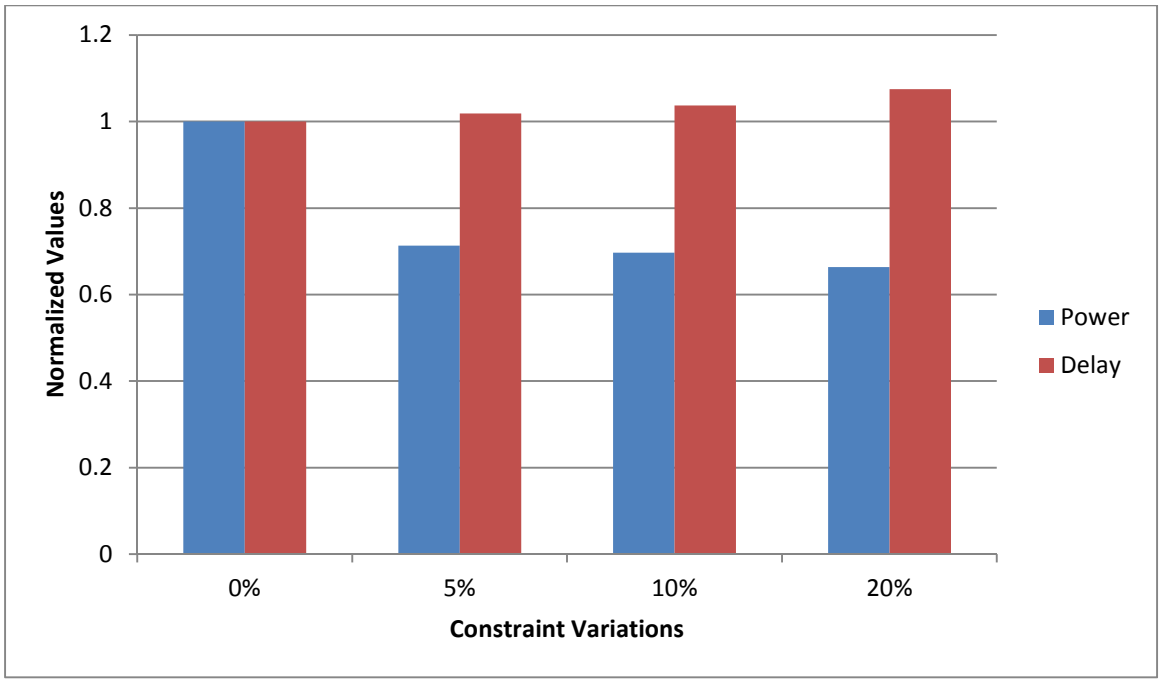

Figure 12: Variation in Power and Delay by changing constraints for GCC

f **Comparison of OCEAN running with and without DFS**
Constraints: 5%, 10%, 20%

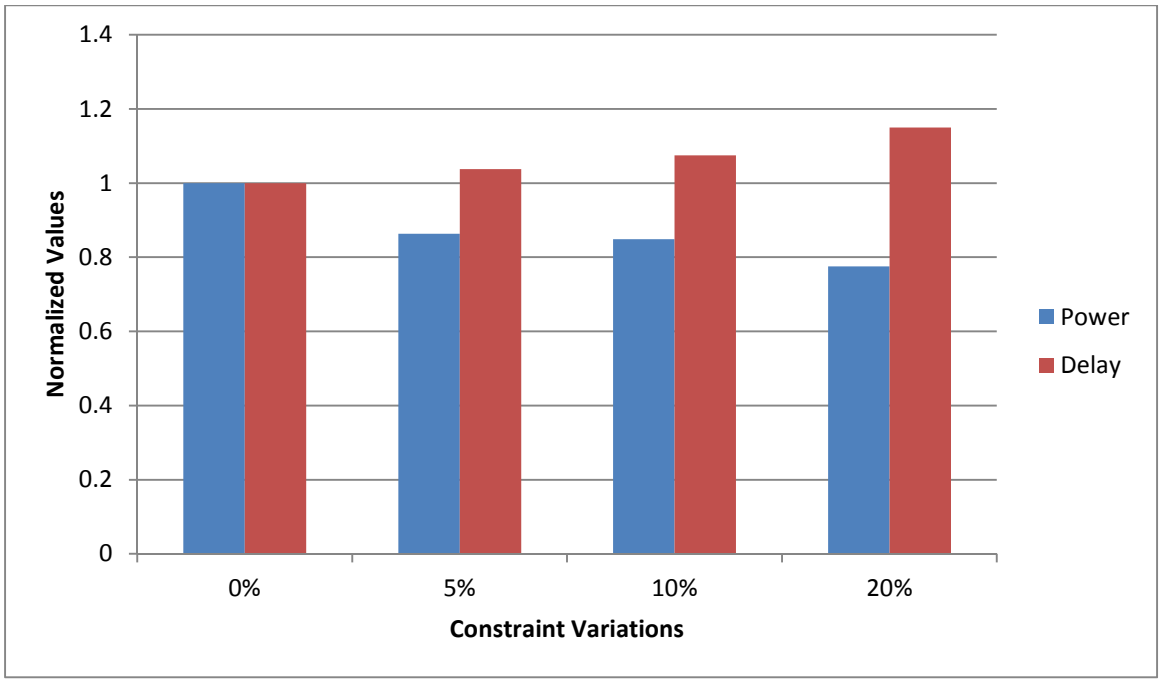

Figure 13: Variation in Power and Delay by changing constraints for OCEAN

6 Conclusion & Future Work

In this report, we described our machine learning based implementation of Dynamic frequency scaling using 4 frequencies. We observed some power savings at the cost of increased delay by 5% to 20%. Although we discussed the general idea above, some specific problems are also existing, such as the intended training and testing time, the period sample length, and so on. Other MLAs can be evaluated and a comparative analysis of all algorithms can be presented. Additional memory/CPU parameters can be explored which can help to better train the MLA. Dynamic Voltage Scaling (DVS) is another mechanism that can be used in tandem with Frequency Scaling (DFS) to further the aim of power management.

7 References

[1] Wonyoung Kim, Meeta S. Gupta, Gu-Yeon Wei and David Brooks, “System Level Analysis of Fast, Per-Core DVFS using On-Chip Switching Regulators,” 14th IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 123-134, 2008.

[2] Gaurav Dhiman and Tajana Simunic Rosing, “Dynamic Power Management Using Machine Learning,” IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2006.

[3] Arkaprava Basu, Nevin Kirman, Meyrem Kirman, Mainak Chaudhuri and Jose F. Martinez, "Scavenger: A New Last Level Cache Architecture with Global Block Priority," 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 421-432, 2007

[4]Chih-Chung Chang et al. “LIBSVM: A library for support vector machines.” ACM Transactions on Intelligent Systems and Technology (TIST)

[5]Binkert et al. “The gem5 simulator”, ACM SIGARCH Computer Architecture News