Dynamic Power Management using Machine Learning
Motivation:

Problem: Determining when to scale down the frequency at runtime is an intricate task.

Proposed Solution: Use Machine learning algorithm to predict intervals where memory intensive tasks occur.

Phase 1: gem5 + McPAT

Phase 2: ILP + Offline DVFS

Phase 3: Machine Learning
System: Overview

- gem5 + McPAT Integrated Model
- ILP + Offline DVFS
- MLA

Monitor

- Power
- Execution Time

Mode

TRAINING

TESTING
## Testbed System Specifications

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISA</td>
<td>ALPHA</td>
</tr>
<tr>
<td>Execution Mode</td>
<td>Out of Order Execution CPU</td>
</tr>
<tr>
<td>Base Frequency</td>
<td>2 GHz</td>
</tr>
<tr>
<td>L1 I-Cache and D-Cache Size</td>
<td>32 kB</td>
</tr>
<tr>
<td>L2 Cache Size</td>
<td>256 kB</td>
</tr>
<tr>
<td>Number of cores</td>
<td>1</td>
</tr>
<tr>
<td><strong>Other parameters at their default values</strong></td>
<td></td>
</tr>
</tbody>
</table>
**Phase 1: gem5 + McPAT**

Integration

- **gem5**
  - Architecture Specification
  - BaseMcPAT class
  - Event Scheduler
  - Stats Parser

- **McPAT**
  - Energy per interval
  - Freq, N cycles per interval

- **ILP**

  for all intervals, for each frequency
Phase 1: gem5 + McPAT

- **SimObject**
  - `schedule()`

- **BaseMcPAT**

- **GEM5**

- **McPAT hooks**
  - `feedIntervalStats()`
  - `parseGem5StatstoXML()`
  - `computeEnergy()`
  - `displayEnergy()`
Phase 1: gem5 + McPAT

differentiating 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>L2 Miss Stats</td>
<td>Functional Unit Busy Rate</td>
</tr>
<tr>
<td># of FPU Accesses</td>
<td>Miss Latency for L2</td>
</tr>
<tr>
<td># of ALU Accesses</td>
<td>CPU Idle Cycles</td>
</tr>
</tbody>
</table>
**Phase 2: ILP + Offline DVFS**

- Power value for all intervals and for all frequencies
- Delay value for all intervals for all frequencies
- Performance constraint = $\delta$
- Execution time = $1/(\text{Performance})$

- $P_{ij}$ where $i$ is the interval number and $j$ is the frequency number
- Hence,
  - $P_{11}$ means interval #1 and frequency number #1 ( = 800 MHz)
  - $P_{12}$ means interval #1 and frequency number #2 ( = 1 GHz)

Considering 2 intervals $I_1$ and $I_2$:

$I_1$: $P_{11}, P_{12}, P_{13}, P_{14}$
$I_2$: $P_{21}, P_{22}, P_{23}, P_{24}$

$I_1$: $D_{11}, D_{12}, D_{13}, D_{14}$
$I_2$: $D_{21}, D_{22}, D_{23}, D_{24}$

- Select $P_{11}$ for $I_1$.
  - $x_{11} = 1$. \(\therefore x_{12} = x_{13} = x_{14} = 0\).
- Select $P_{23}$ for $I_2$.
  - $x_{23} = 1$. \(\therefore x_{21} = x_{22} = x_{24} = 0\).
  - ... where $x_{ij}$ is a binary variable.
Phase 2: ILP + Offline DVFS

- 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 DVFS control problem is formulated as an Integer Linear Programming (ILP) optimization problem
- We seek to reduce the total power consumption of the processor within specific performance constraints ($\delta$)
- The application runtime is divided into $N$ intervals
- A total of $L = 2$ frequency 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
- The equations below formulate the ILP, which is solved using Gurobi Optimizer

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

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

$$\sum_{j=1}^{L} x_{ij} = N \quad \forall \quad i = 1 \text{ to } N$$

... where $x_{ij}$ is a binary variable having values 1 or 0.
Phase 3: Machine Learning

Stage #1: Input Parameters and Frequency

Stage #2: Machine Learning Algorithm

Stage #3: Frequency

Interval #0: \( z_{10}, z_{20}, z_{30} \)  
Interval #1: \( z_{11}, z_{21}, z_{31} \)  
Interval #2:  
Interval #3:  
Interval #4:  
Interval #5:  
Interval #6:  
Interval #7:  

\( z_{ij} \) where \( i \) is the parameter number (there are \( n \) such parameters) and \( j \) is the interval.
Phase 3: Machine Learning
learning and classification

1. FEATURE EXTRACTION
2. CLASSIFICATION
3. FREQUENCY OUTPUT

Input Space → Feature Space

\[ f_1 = \]
\[ f_2 = \]
\[ f_3 = \]
\[ f_4 = \]
Contributions from our end...

- **ILP Formulation and Solving**
  - Python implementation for solving the *ILP formulation* using Gurobi Optimizer

- **Gem5 modifications**
  - McPAT integration
  - *Dynamic Frequency Scaling* in gem5
  - Integrating the *SVM Model* with gem5

- **Developed scripts**
  - Parsing the *output of gem5* in a format acceptable to the *Machine Learning Algorithm*
  - C implementation to fetch execution statistics corresponding to the down-scaled frequency obtained after performing *Dynamic Frequency Scaling*
  - In progress is a single script to *automate the entire process*
Experimental Results

Normalized Power

- Benchmarks: ocean, mcf, gcc
- Constraints: 0% (No constraint), 10%, 20%

Graph showing normalized power for different benchmarks and constraints.
Experimental Results

Normalized Delay

Benchmarks:
- ocean
- mcf
- gcc

Constraints:
- No constraint
- 10% Constraint
- 20% Constraint
Future Work

✓ Avenues for future optimization:
  ✓ Finding the optimum interval size and number of intervals for training
✓ Other MLAs can be evaluated and a comparative analysis of all algorithms can be presented
✓ Several steps of frequencies can be introduced
✓ System performance can be evaluated with the permutation-combination of training and testing with some more benchmarks
✓ Additional memory/CPU parameters can be explored which can help to better train the MLA
✓ Memory profile of each benchmark can be analyzed to perform selective training for the MLA
Thank You . . .

. . . questions are welcome