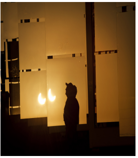
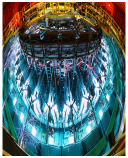
6/27/15

SAND2015-5175C

Phase Detection with Hidden Markov Models for DVFS on Many-Core Processors

Joshua Dennis Booth
Computer Science Research Institute,
Sandia National Laboratories

Sandia National Laboratories

Exceptional service in the national interest

Acknowledgement

Coauthors

- Jagadish Kotra, jbk5155@cse.psu.eu
  Ph.D. Candidate CSE, Penn State
- Hui Zhao, hzz105@cse.psu.edu
  Ph.D. CSE, Penn State
- Mahmut Kandemir, kandemir@cse.psu.edu
  Professor CSE, Penn State
- Padma Raghavan, raghavan@cse.psu.edu
  Distinguished Professor CSE, Penn State

Funding Support

- NSF Grants 1213052, 1205618, 1439021, 096389, 1017882
- Grant support from Intel

Disclaimer

The views presented here are the author's only.
Sandia National Lab takes no responsibility for this work or results.

Outline

Motivation, Current Problem

- Current DVFS Techniques
- Phases and Hidden Markov Models

Proposed Approach

- Phased-Based DVFS, *PVFS*
- Initialization
- Training
- Applying

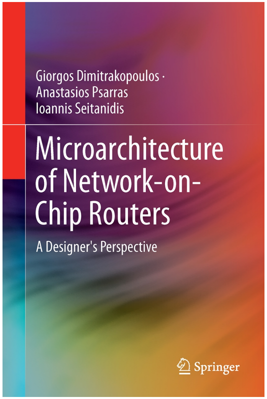
Experimental Results

- Setup
- Observed HMMs
- Major Results

Outline

Future of Many-Core Processors

Future Intel Many-Core, 60+ cores, 300W+


Giorgos Dimitrakopoulos · Anastasios Psarras Ioannis Seitanidis

Microarchitecture of Network-on-Chip Routers

A Designer's Perspective

Springer

NoC, SoC etc

GPU, 2496+ Cuda cores, 240W+

- Coding styles and techniques vary greatly with device
- The execution pattern (time, hardware components used, power) vary for same code.

DVFS Techniques

Goals

- Detecting if code has area with poor performance
- Identifying code segments for optimize
  Characterizing interaction with hardware leading to performance, power, and scalability.
- Value of time investment

Current Techniques

- Time-Driven
  - Samples taken at regular interval
  - Can be applied to a multitude of device components
  - Can it scale ?
- Compiler-Driven
  - Critical path analysis
  - Hard problem, Integer Linear Programming
  - What about applications that very greatly with input?

Can We Do Better?

Phases

Phase

- All goals can reworded in terms of *Phase*
- *Phase:* a segment of an application characterized with an unique performance and power
- Even short application have multiple phases
- Long running HPC application having many!
- Phase detection is difficult

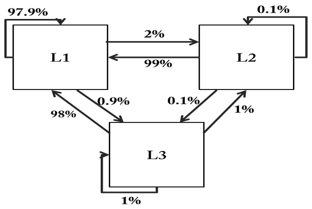
Statistical Models

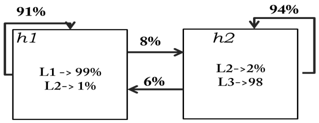
- Have been shown to be able to identify Phase
- Many different statistical models and levels of depth that can taken
- Need a model that can discover interaction of measurable observations

Hidden Markov Models for Phase Detection

Hidden Markov Models (HMMs)

- A form of Dynamic Bayesian Network
- Model time at *t* from *t*-1
- Have been used in many areas such as bioinformatics, robotics, etc
- Model contains both observable and unobservable states
- Hidden states combine unobservable interactions
- Flexibility of user defined observable states
- Smoothing, disregarding outliers
- Weighting, can weigh phases based on number of occurrences





Outline

Motivation, Current Problem
- Current DVFS Techniques
- Phases and Hidden Markov Models

Proposed Approach
- Phased-Based DVFS, *PVFS*
- Initialization
- Training
- Applying

Experimental Results
- Setup
- Observed HMMs
- Major Results

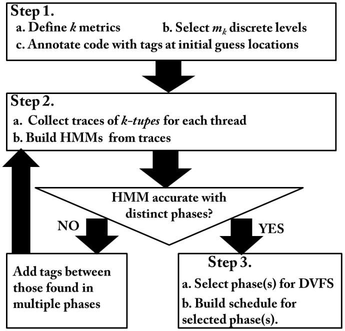
Phase-Based DVFS *PVFS*

Steps to Apply PVFS

- Parametrization
- Training
- Applying

Parametrization

- *System Metric*, a simple measurement of system
- *Free Parameter Set*, a collection of system metrics to describe perfomance and power
- Discretization, a distribution that matches the system metric and application



Step 1.
a. Define k metrics
b. Select m_k discrete levels
c. Annotate code with tags at initial guess locations
Step 2.
a. Collect traces of *k-tupes* for each thread
b. Build HMMs from traces
HMM accurate with distinct phases?
NO
YES
Add tags between those found in multiple phases
Step 3.
a. Select phase(s) for DVFS
b. Build schedule for selected phase(s).

Phase-Based DVFS *PVFS*

Training

- Tag used to collect observation
- Tags placement can be automated
- Initial placement
  - Beginning/End of function
  - Beginning/End of loops
  - Beginning/End of large blocks
- Update tag placement
  - Use of Viterbi Path to decide if need additional tags
  - Hidden state found with low probability and contains multiple tags with diverse performance/power
  - When a tag id is semi-uniformaly distributed between hidden states

Evaluation of Model

- Probability the sequence of the Viterbi Path mates a sample trace
- No new tags needed

Algorithm 1 Initial placement of tags in C code.

```
1: for all Source Files do
2:     Find beginning of function based on regex
3:     Place tag at location
4:     Scan through function for LOOP using regex
5:     Scan for end of LOOP by matching "{" and "}"
6:     if LOOP length is greater than 4 then
7:         Place tag at beginning location of LOOP
8:         Place tag at end location of LOOP
9:     end if
10:    Scan for end of function by matching "{" and "}"
11: end for
```

Algorithm 2 Update tags in C code.

```
1: Let T be the set of critical tags
2: Add tags to T from hidden states with probability $< \tau$ and
   number of tags $> \gamma$
3: for all Tags (t) in the HMM do
4:     for all Hidden State (h) do
5:         if $100/|H| - \epsilon \leq$ Probability(t) $\in$ h $\leq 100/|H| + \epsilon$ then
6:             Add t to T
7:         end if
8:     end for
9: end for
10: for all t in T do
11:     Scan for function call, loop, or control statement from t
12:     Add tag at found location
13: end for
```

Phase-Based DVFS *PVFS*

Applying

- Hidden states (Phases) with power performance and high power
- Tags can be used to signal DVFS

Discussion

- Hardware
  - Multiple levels of controller - HMM for each control
  - Heterogeneous processors - Tag based on device and HMM for each
  - GPU - Breaking execution into warps, thread blocks, etc
- Complexity
  - Let T be the number of time steps
  - Let S be the number of hidden states
  - Traditional Baum-Welsh, $O(TS^2)$
  - Long codes will increase the training time in a linear fashion
  - S tends to be small, will be on discretization of system metrics

Outline

Experimental Setup

Simulation

- Chip simulation, Sniper
- Memory Simulation, McPAT
- NoC simulation, DSENT

Comparison Methods

- Energy Limit (EL)
  $EL(app,p) = T(app,p,baseline) \times min(P(app,p,baseline))$
- Energy Delay Product (EDP)
  - Time-Driven DVFS, Swaminathen et al. [a]
  - Change at every epoch
  - Fixed epoch size
  - Tuned tolerance parameter γ

Simulated chip description.

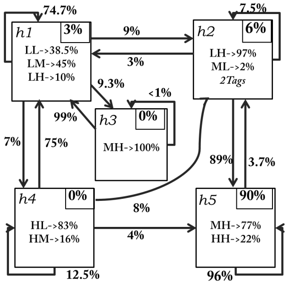
Core	22nm Ivybridge
L1 cache	32KB, 8-way associative, Access Latency: 4 cycles
L2 Cache	256KB, 8-way associative, Access Latency: 8 cycles
Coherence	MESI
L3 Cache	2MB/core, Shared, 16-way, Access Latency: 30 cycles
NoC	2 cycles latency; 64 bits/cycle 16 cores (4x4); 32 cores (8x4)
DVFS Latency	2 usecs (Both core and NoC)
DRAM	45 nsecs Latency; DDR3-1600; per-controller b/w: 7.6 GB/sec; 4-8 cores use 1 controller; 16-32 cores use 4 controllers

[a] Improving energy efficiency of multi-threaded applications using heterogeneous CMOS-TFET

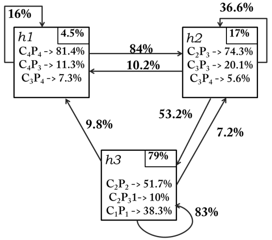
Experimental Setup
Benchmark Suite

- `Freqmine`, *Frequent itemset mining*, PARSEC
    - Tree and sorting algorithms. Common in data mining.
- `Dedup`, *Data deduplication compression*, PARSEC
    - Compression algorithm using pthreads.
- `Mgrid`, *Multigrid solver*, SPECOMP
    - V-cycle, has variable levels of parallelism through V-cycle.
- `Wupwise`, *Quantum chromodynamics*, SPECOMP
    - Lattice based quark propagation.
- `XSBench(MC)`, *Monte-Carlo transport*, CORAL
    - Monte Carlo neutronics, read/write intensive. Low performance.
- `LCALS`, *Collection of loops*, CORAL
    - Loops that represents float-point execution patterns from multiple applications.
- `CoMD`, *Molecular dynamics*, MANTEVO
    - Execution pattern varies as molecules between cells. Different patterns in and between cells.
- `HPCCG`, *Irregular conjugate gradients*, MANTEVO
    - Test system sparse computation. Gives a comparison bound to LINPACK

HMM of `Freqmine`



+



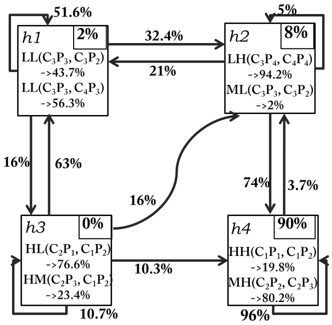
HMM with Core

- 5 hidden states
- *h*1 and *h*2 to reduce power
- Tags correspond to irregular date access and thread barriers
- Reduced by .8v

HMM with Network

- Power due to network contention is a small fraction of the power
- Due to this model DVFS would be applied to *h*2
- Tags do not coordinate to those in the core model

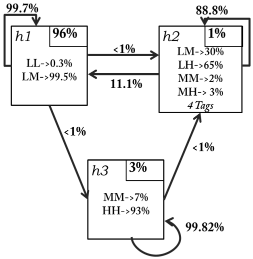
HMM of `Freqmine`

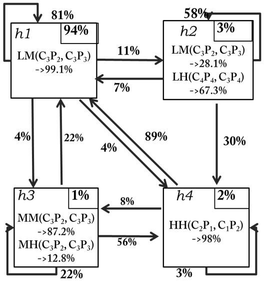


HMM of Core and Network

- DVFS on *h*2
- These tags are slighlty different than those of both models
- Reduce to .8V
- Reduce energy by 8.78%

HMM of LCALS





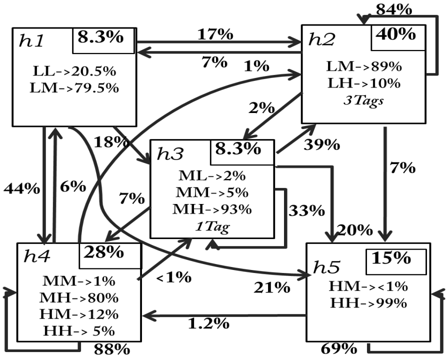
HMM of Core

- 15 Tags to capture the 7 unique loops
- 4 Tags to apply DVFS, irregular long strided access
- A reduction of 7.1%

HMM of Core and Network

- A network only model will only have one phase, similar network contention
- Both together, 16 Tags, 2 tags for DVFS
- A reduction of 7.9%

HMM of CoMD

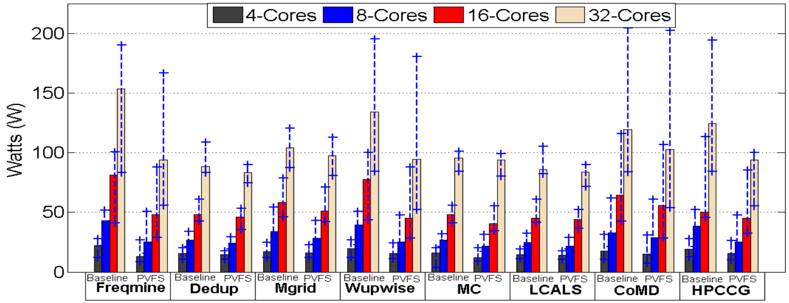


HMM of Core and Network

- Most complicated model of test suite
- 14 tags are found and used
- 2 Phases to apply DVFS
- Reducing 9.8% energy

Raw Effect on Dynamic Power

Dynamic Power with and without PVFS



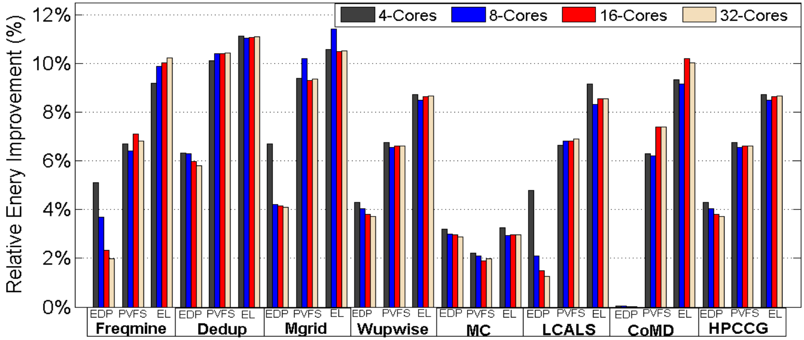
- Baseline without DVFS and PVFS dynamic power
- Reduction of maximum and average power for all applications
- Reduction of 97 Watts for maximum power and 62 Watts on average on 32 cores
- Leakage Power is fixed, energy will go up with time penalty

	4-Core	8-Core	16-Core	32-Core
Leakage Power	1.82W	3.15W	7.53W	15.27W

Leakage power of simulated processors in Watts (W).

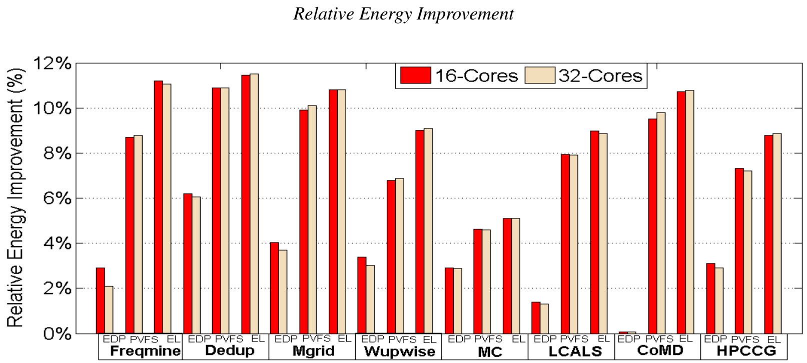
Core Energy Improvement

Relative Energy Improvement (REI)



- $REI(app,p,m) = \frac{E(app,p,baseline) - E(app,p,m)}{E(app,p,baseline)}$
- With only 32 core DVFS, can achieve over 6% on 4 applications
- EL provides guide for the best possible energy use of an application
- PVFS in worst case achieves 64% of EL and 94% of EL for `Dedup`

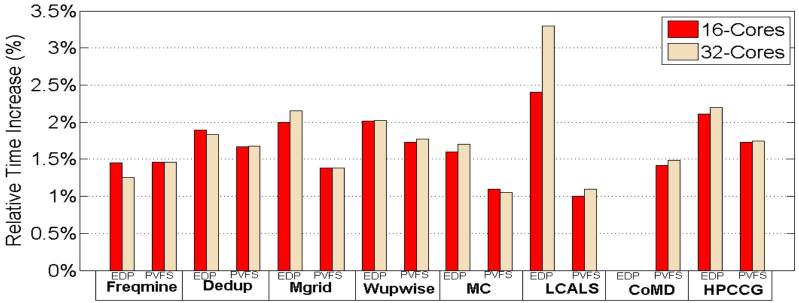
Core+NoC Energy Improvement



- Larger saving using DVFS on NoC, No Lose is hard.
- Worst case achieves 75% of EL
- Best case achieves 95% of EL

Time Impact

Relative time increase $RTI(app, p, m)$ *chip and network*



- $RTI(app, p, m) = \frac{T(app,p,m) - T(app,p,baseline)}{T(app,p,baseline)}$
- Every DVFS method has some impact on performance
- EDP has slightly higher time increase

Conclusion

- Hidden Markov Models provides a generic framework for phase detection
- This framework is flexible enough to fit any future manycore system
- DVFS with Phase (PVFS) can achieve a large percent of the energy limit
- PVFS can model energy use of multiple controllers