Integrated CPU and L2 Cache Voltage Scaling using Machine Learning

Nevine AbouGhazaleh, Alexandre Ferreira, Cosmin Rusu, Ruibin Xu, Frank Liberato, Bruce Childers, Daniel Mossé, Rami Melhem

Presenter: Minjun Wu

UMN CSCI 8980: Machine Learning in Computer Systems, Paper Presentation, 02/2019

University of Minnesota
Driven to Discover®

Power in 2007

New chip design: MCD

- Multiple Clock Domain

Scenario:

- Larger chip “size”, more transistor and circuits
- No single timing in chip anymore, domains

MCD: Fine-grained PM opportunity

Old design:

- one chip, entirely, has single frequency
- select from different “mode”

New design opportunity:

- different domain has different frequency
- can adjust with application’s requirement

=> Reduce power consumption for **inactive** domain

The target, this paper

- Provide a fine-grained power management by MCD
- The management is done by Supervised Learning

PACSL: a Power-Aware Compiler-based approach using Supervised Learning

- Using performance counters monitoring system
- Training to collect policies offline
- Apply policies for dynamic frequency adjustment

PACSL, overview

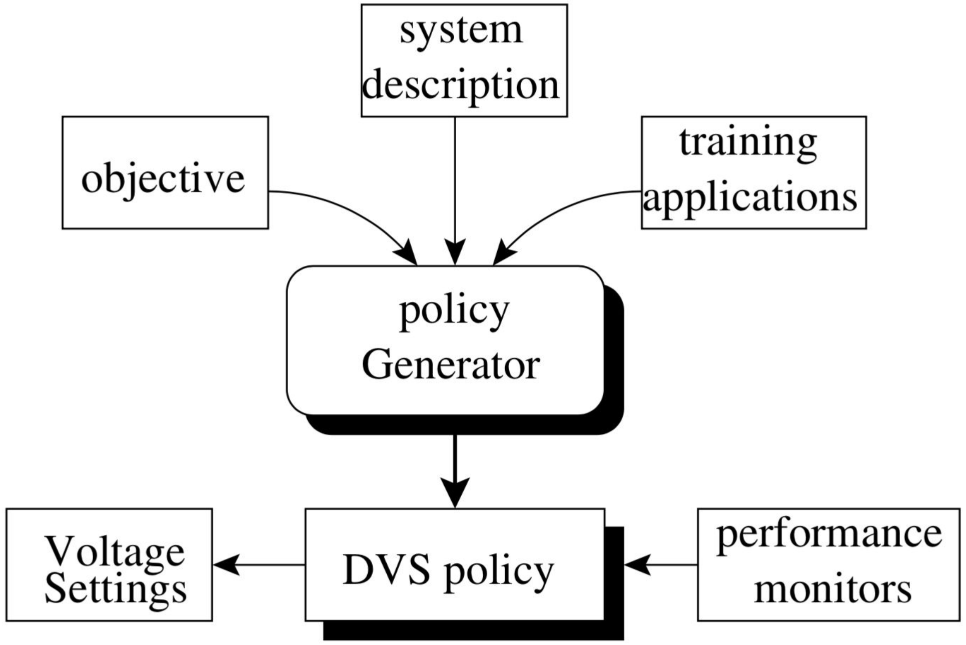

Figure 1. Information flow in PACSL

PACSL, overview

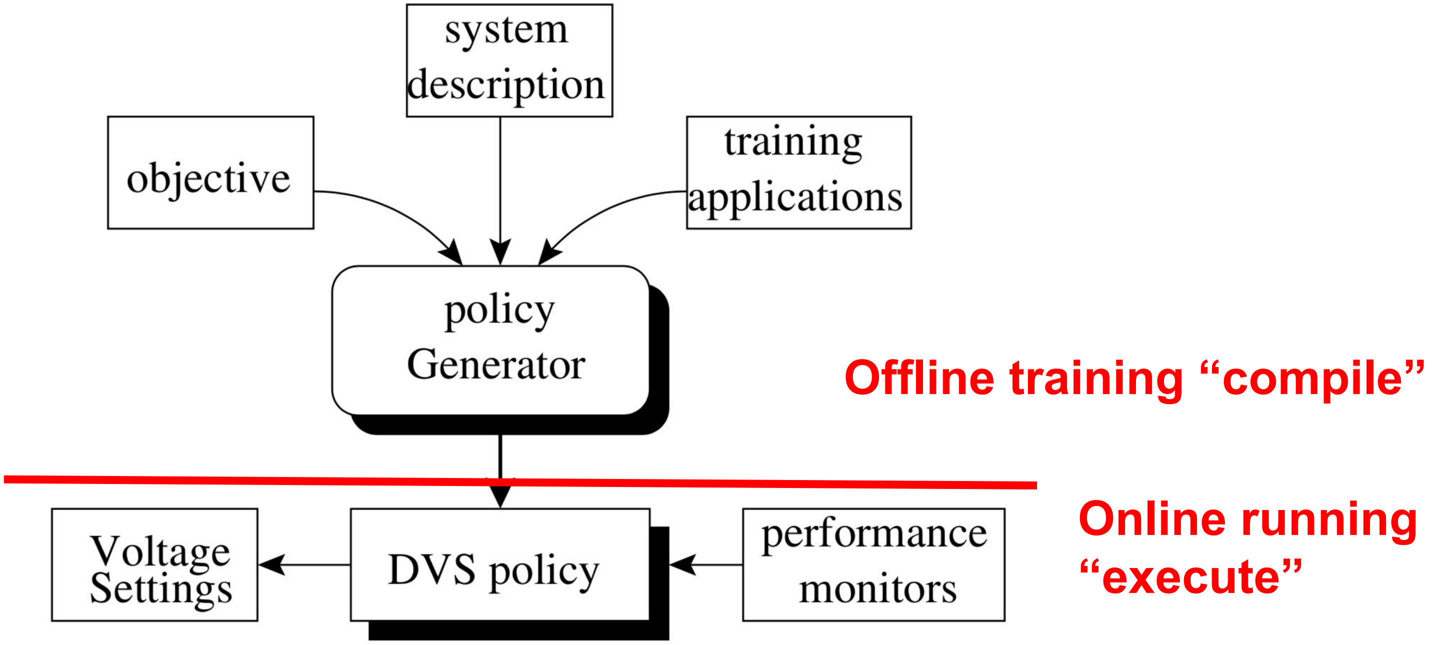

Figure 1. Information flow in PACSL

How to describe apps?

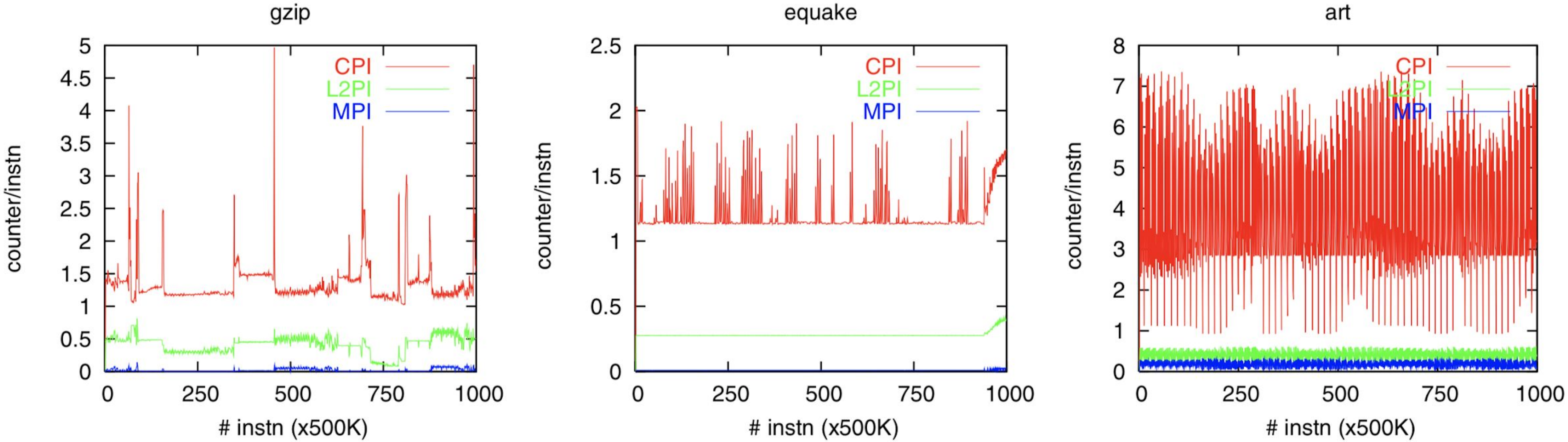

Figure 2. Variations in application phases throughout execution.

How to describe apps?

Hybrid (typical) | **CPU bound** | **Cache/Memory bound**

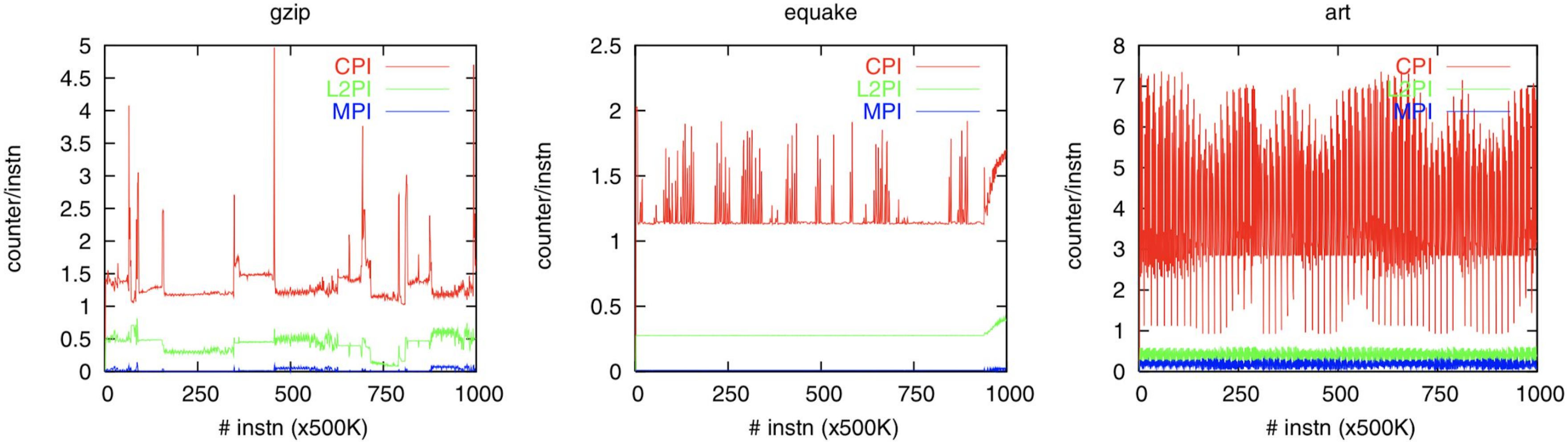

Figure 2. Variations in application phases throughout execution.

How to design this SL approach? [input]

Motivation: different application has different behavior:

- CPI: cycle per instruction
- L2PI: LLC access per instruction
- MPI: memory access per instruction

Different objective:

- Energy, Energy-Delay Product

System Configuration: LLC size, CPU etc.

How to design this SL approach? [output]

Policies:

- easy to apply at run time
- easy to understand

Propositional Rule:

“Under this condition, we should do that. ”

Design overview: more specific

- Two domains: CPU domain and LLC domain
- Offline stage:
    a. analysis training applications
    b. develop runtime policy (for diff objective)
- Runtime stage:
    a. periodically monitor activity
    b. determine best frequency based on policy

Design overview: more specific

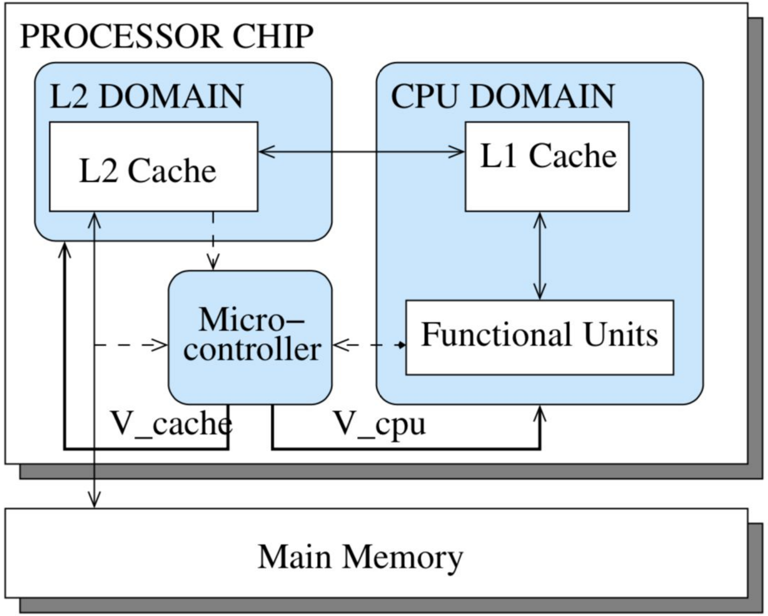

Figure 3. Example of an MCD processor design with integrated DVS control.

Offline stage: **a. analysis training applications**

Performance counter and frequency (“latency”):

- CPI, L2PI, MPI
- CPU domain frequency, L2C domain frequency

Some inputs are continuous, some are discrete:

- [c] CPI, L2PI, MPI, running program
- [d] CPU freq, L2C freq (choose from available set)

Offline stage: **a. analysis training applications**

Make continuous input discrete:

- CPI, L2PI, MPI: bins (same #entities each bin)
- running program: sampling

    K samples, each have “size” instructions

Now, the input data will be:

$$S_{kij} = \{CPI_{kij}, L2PI_{kij}, MPI_{kij}, M_{kij}\}$$

k: sample id, i: CPU freq, j: L2C freq, Mkij: objective (E or ED)

Offline stage: a. analysis training applications

Table 1. Eight training samples: CPI, L2PI and energy-delay product (ED) at all frequency combinations. O and 1 are the index of the CPI and L2PI bins.

discrete CPU/L2C freq

f_{cpu} / $f_{\$}$	0.5GHz / 0.5GHz			0.5GHz / 1GHz			1GHz / 0.5GHz			1GHz / 1GHz		
s	CPI	L2PI	ED	CPI	L2PI	ED	CPI	L2PI	ED	CPI	L2PI	ED
1	0	1	200	0	1	354	1	1	183	1	1	187
2	0	1	242	0	1	428	1	1	223	1	1	226
3	0	0	436	0	0	768	1	0	395	1	0	403
4	0	1	274	0	1	481	1	1	252	0	1	250
5	0	0	473	0	0	826	1	1	430	0	0	430
6	1	1	330	0	1	588	1	1	309	1	1	317
7	1	0	361	0	0	642	1	0	327	1	0	339
8	1	0	401	0	0	709	1	0	363	1	0	374

sample id

CPI bin 0 and 1

L2PI bin 0 and 1

Objective number

Offline stage: **a. analysis training applications**

How to describe the action?

- A action table! (ST, state table)
- By current status: CPI, L2PI, MPI; tell me what CPU/L2C frequency should I set in next stage?

Method:

Choose the **best** freq for **each** class of “code **sections**”

$$Acc[CPI_{kij}][L2PI_{kij}][MPI_{kij}][i][j][x][y] += M_{kxy}$$

best **Metrics** in each **<x,y>** of code section **<k>**

Offline stage: a. analysis training applications

Table 1. Eight training samples: CPI, L2PI and energy-delay product (ED) at all frequency combinations. O and 1 are the index of the CPI and L2PI bins.

f_{cpu} / $f_{\$}$	0.5GHz / 0.5GHz			0.5GHz / 1GHz			1GHz / 0.5GHz			1GHz / 1GHz		
s	CPI	L2PI	ED	CPI	L2PI	ED	CPI	L2PI	ED	CPI	L2PI	ED
1	0	1	200	0	1	354	1	1	$\boxed{183}$	1	1	187
2	0	1	242	0	1	428	1	1	$\boxed{223}$	1	1	226
3	0	0	436	0	0	768	1	0	$\boxed{395}$	1	0	403
4	0	1	274	0	1	481	1	1	252	0	1	$\boxed{250}$
5	0	0	473	0	0	826	1	1	$\boxed{430}$	0	0	430
6	1	1	330	0	1	588	1	1	$\boxed{309}$	1	1	317
7	1	0	361	0	0	642	1	0	$\boxed{327}$	1	0	339
8	1	0	401	0	0	709	1	0	$\boxed{363}$	1	0	374

Offline stage: **a. analysis training applications**

Method (cont'):

Use Accumulation to get the best one:

$ST[CPI_{kij}][L2PI_{kij}][MPI_{kij}][i][j]$

= min<x,y> of $Acc[CPI_{kij}][L2PI_{kij}][MPI_{kij}][i][j][x][y]$

(I show you how it works, but we will discuss it later)

Table 1. Eight training samples: CPI ... dex of the CPI and L2PI bins.

$$\boxed{Acc[CPI_{kij}][L2PI_{kij}][MPI_{kij}][i][j][x][y] += M_{kxy}}$$

f_{cpu} / $f_\$$	0.5GHz / 0.5GHz			0.5GHz / 1GHz			1GHz / 0.5GHz			1GHz / 1GHz		
s	CPI	L2PI	ED	CPI	L2PI	ED	CPI	L2PI	ED	CPI	L2PI	ED
1	0	1	200	0	1	354	1	1	183	1	1	187
2	0	1	242	0	1	428	1	1	223	1	1	226
3	0	0	436	0	0	768	1	0	395	1	0	403
4	0	1	274	0	1	481	1	1	252	0	1	250
5	0	0	473	0	0	826	1	1	430	0	0	430
6	1	1	330	0	1	588	1	1	309	1	1	317
7	1	0	361	0	0	642	1	0	327	1	0	339
8	1	0	401	0	0	709	1	0	363	1	0	374

i = 0.5	j = 0.5	<x, y>	<x, y>	<x, y>	<x, y>
CPI	L2PI	0.5, 0.5	0.5, 1	1, 0.5	1, 1
0	0	-	-	395+430	-
0	1	-	-	183+223	250
1	0	-	-	327+363	-
1	1	-	-	309	-

Table 2. Constructed *ST* from samples in Table 1.

		f_{cpu}=0.5GHz		f_{cpu}=1GHz	
CPI	L2PI	$f_\$$=0.5	$f_\$$=1	$f_\$$=0.5	$f_\$$=1
0	0	1/0.5	1/0.5	-	-
0	1	1/1	1/1	-	1/1
1	0	1/0.5	-	1/0.5	1/0.5
1	1	1/0.5	-	1/1	1/0.5

Offline stage: **b. develop runtime policy**

Problem for Table 2: not all states are covered

- Need to fill in the state-action and gen policy

They tried many ML method, then choose
“propositional rule”
For detail, they use “RIPPER” and “IREP algorithm”

Offline stage: **b. develop runtime policy**

"propositional rule":

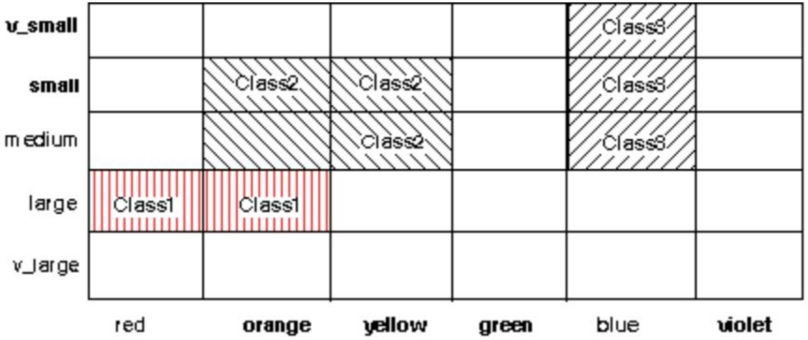

Figure 1: Discrimination on attributes and values

```
cover := {};
repeat
        select one positive example, e;
        construct the set of all conjunctive expressions
                that cover e and no negative example in E-;
        choose the `best' expression, x, from this set;
        add x as a new disjunctof the concept;
        remove all positive examples covered by x
until there are no positive examples left;
```

The `best' expression is usually some compromise between the desire to <u>cover as many positive examples as possible</u> and the desire to have <u>as compact and readable a representation as possible</u>.

ref: http://www.cse.unsw.edu.au/~billw/cs9414/notes/ml/06prop/06prop.html

Incremental reduced-error pruning

```
Initialize E to the instance set
Until E is empty do
   Split E into Grow and Prune in the ratio 2:1
   For each class C for which Grow contains an instance
     Use basic covering algorithm to create best perfect rule
        for C
     Calculate w(R): worth of rule on Prune
           and w(R-): worth of rule with final condition
                      omitted
     If w(R-) < w(R), prune rule and repeat previous step
   From the rules for the different classes, select the one
     that's worth most (i.e. with largest w(R))
   Print the rule
   Remove the instances covered by rule from E
Continue
```

(I think) like validation data: if not passed for validation, then repeat

ref: http://www.csee.usf.edu/~lohall/dm/ripper.pdf

Incremental reduced-error pruning Modified for RIPPER

- Order classes according to increasing prevalence
  $(C_1, \ldots, C_k)$
  find rule set to separate C_1 from other classes
  IREP(Pos=C_1,Neg=C_2,...,C_k)
  remove all instances learned by rule set
  find rule set to separate C_2 from $C_3, \ldots, C_k$
  ...
  C_k remains as default class

ref: http://www.csee.usf.edu/~lohall/dm/ripper.pdf

Offline stage: **b. develop runtime policy**

As result:

Table 3. Example of a policy to minimize energy-delay product.

#	Rule
1	if (L2PI $\geq$ 1) and (CPI $\leq$ 0) then $f_{\$}$=1GHz
2	else $f_{\$}$=0.5GHz
3	f_{cpu}=1GHz

Table 2. Constructed *ST* from samples in Table 1.

		f_{cpu}=0.5GHz		f_{cpu}=1GHz	
CPI	L2PI	$f_{\$}$=0.5	$f_{\$}$=1	$f_{\$}$=0.5	$f_{\$}$=1
0	0	1/0.5	1/0.5	-	-
0	1	1/1	1/1	-	1/1
1	0	1/0.5	-	1/0.5	1/0.5
1	1	1/0.5	-	1/1	1/0.5

Offline learning stage summary

- PACSL sample data in training app
- PACSL generate ST based on best Metrics
- PACSL generate simple rules based on SL

Before we go to evaluation part.. some design choices

Before evaluation

Training app selection:

- more coverage on ST (more CPI/L2PI/MPI variance)

Sample size, interval:

- smaller: fine-grained, more accurate and overhead

Evaluation

- based on Simulator with MCD extension (Simplescalar, Wattch)
- tools for propositional rules (JRip)
- break benchmark into training/testing set (exclusive)
- sample size: 500K instructions

Result:

MPI is not that significant, but huge reduction achieved

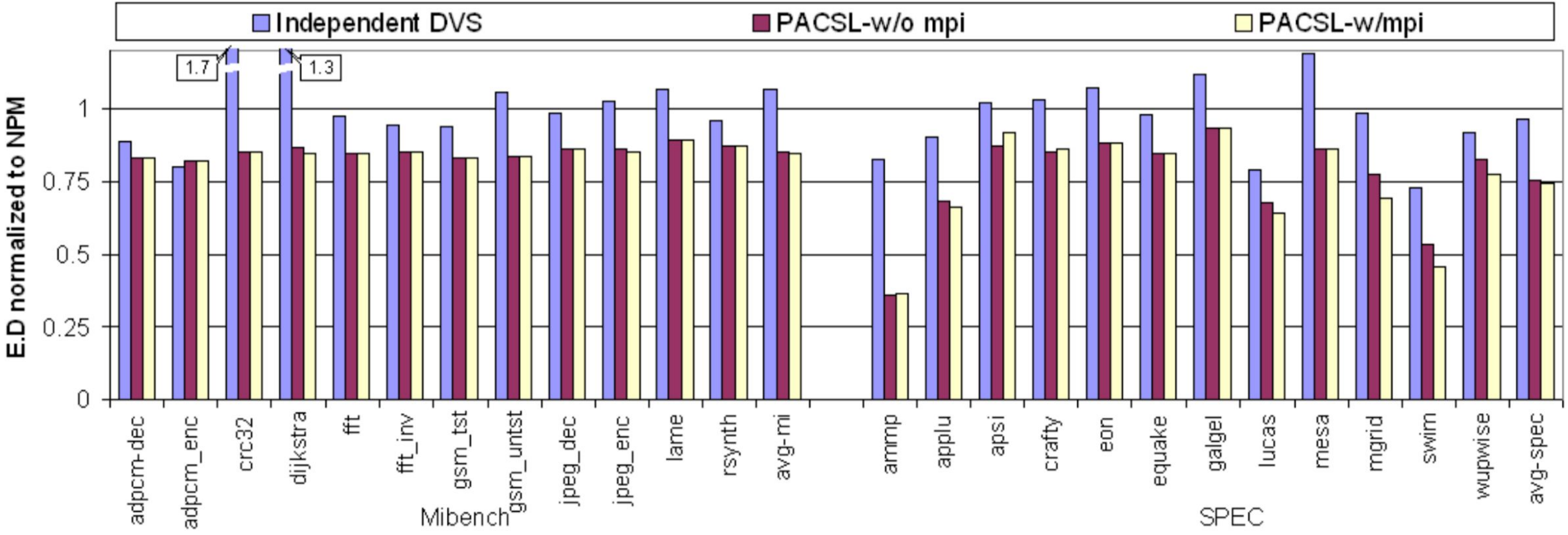

Figure 5. Energy-delay product for SPEC2000 and Mibench benchmarks when using Independent DVS versus PACSL.

Result:

different metrics: with delay bound, also demonstrate

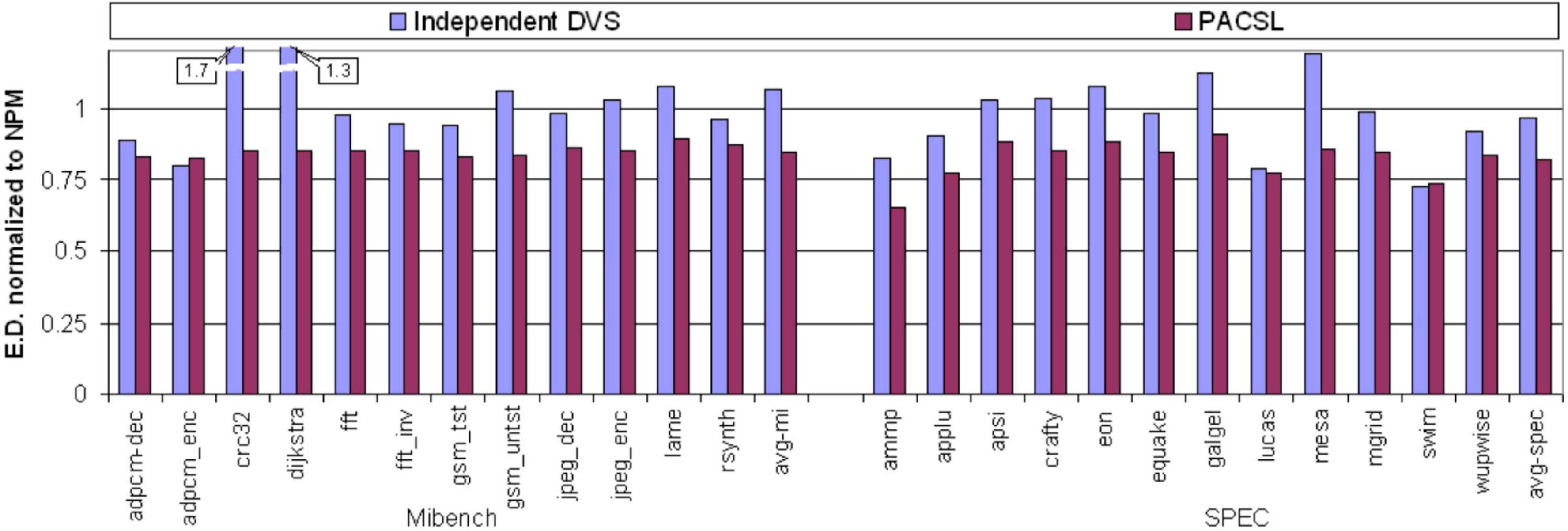

Figure 6. Energy-delay product when optimizing energy with delay bound.

Result:

different machine configuration: demonstrated

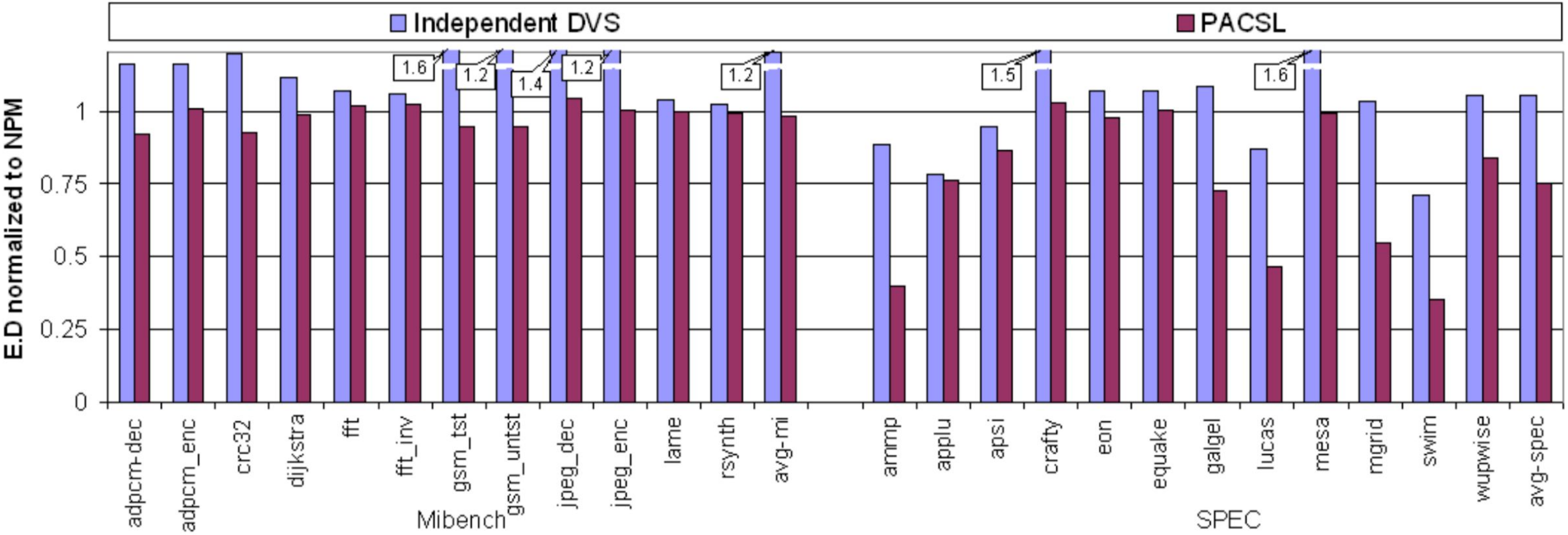

Figure 7. Energy-delay product for policies running on system with configuration Config B in Table 4.

Result:

longer interval will reduce the gap, less granularity

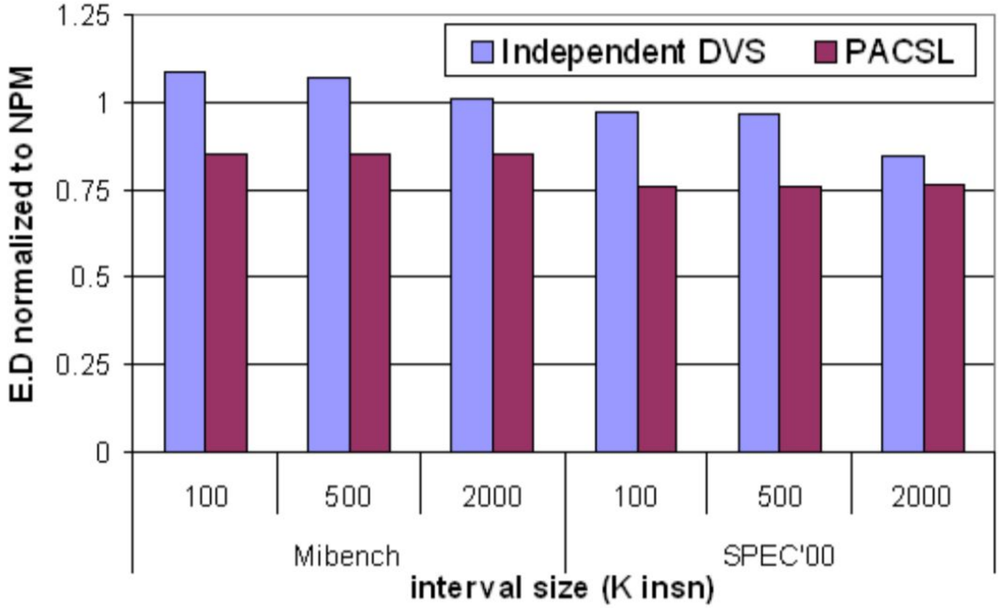

Figure 8. Average energy-delay product at different DVS control-interval sizes (using Config A).

Result:

complex app has more states, similar contribute less

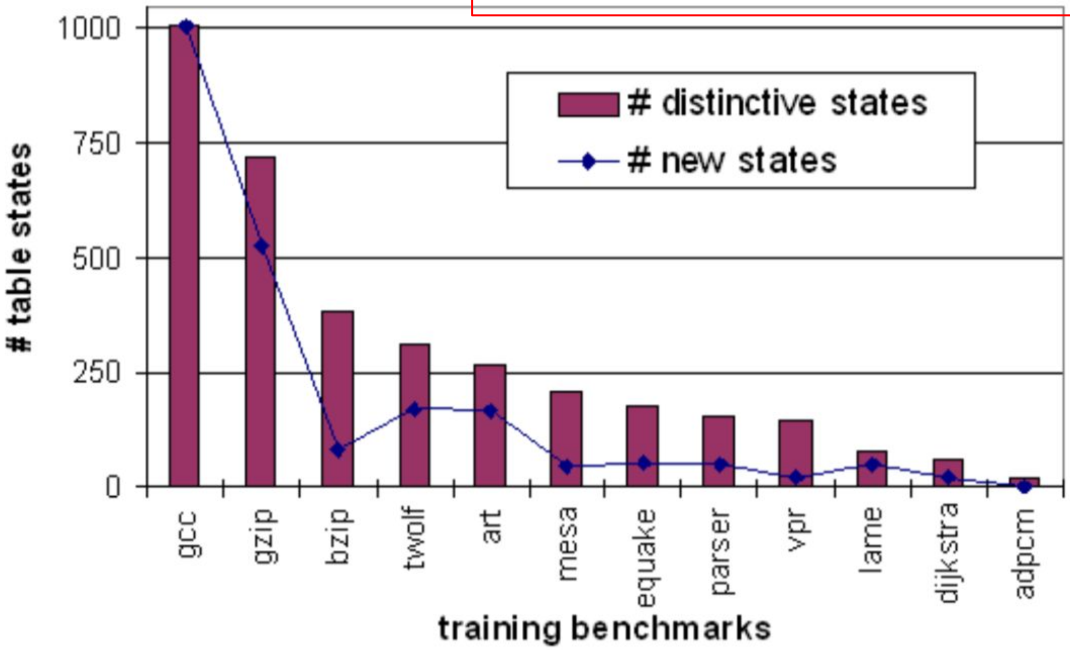

Figure 9. *ST* coverage.

Discussion, my opinion

Strength:

- Fine-grained new design provides opportunity for power optimization (the first ML work for MCD). Since the system is more and more complicated (more layers, controls), this opportunity increases.

- The ML method can capture the app requirement, generate policy from system behavior and apply to system. A good example showing “down to the ground” for ML in system design.

Discussion, my opinion

Weakness:

- Need to demonstrate current app state can be used to predict future state. I think this paper tries to cluster applications, and identify them at early stages. Then a proof for no “state intersection” is required (hard because program is not predictable).

- The ST generation is not clear enough, and it’s stateless (not like stochastic process, RNN). Is there any better way to describe the best metric like DP?

Thanks!

University of Minnesota

Driven to Discover®

Crookston Duluth Morris Rochester Twin Cities

Why frequency with power?

- “higher frequency, run faster, work more”
- $P_{cpu} = P_{dyn} + P_{sc} + P_{leak}$
- $P_{dyn} = CV^2 f$
- higher voltage will charge capacitor faster, then less latency (circuit design perspective)
- (Moore’s law is another thing)
- DVS: dynamic voltage scaling

What is DVS? relationship with MCD?

- Even though you can control both supply voltage and clock frequency, they are not independent.
- Less voltage will lead less frequency for longer delay
- adjust voltage and clock will lead different overhead. adjust voltage will be slower in “effective”.

Why not as low frequency as possible?

- Low frequency will decrease power consumption, but make execution time longer.

Why not online ML approach?

- They tried online ML approach, but the effectiveness is not as good as offline one. Also the runtime overhead is bigger.
- ref: https://cs.pitt.edu/PARTS/presentation/Hipeac_08.pdf

Many ML approach, why this one?

Why rules?

- they tested many, this one is the best.

why discrete?

- They didn't mention.

why accumulation? not average?

- I think it’s a mistake..