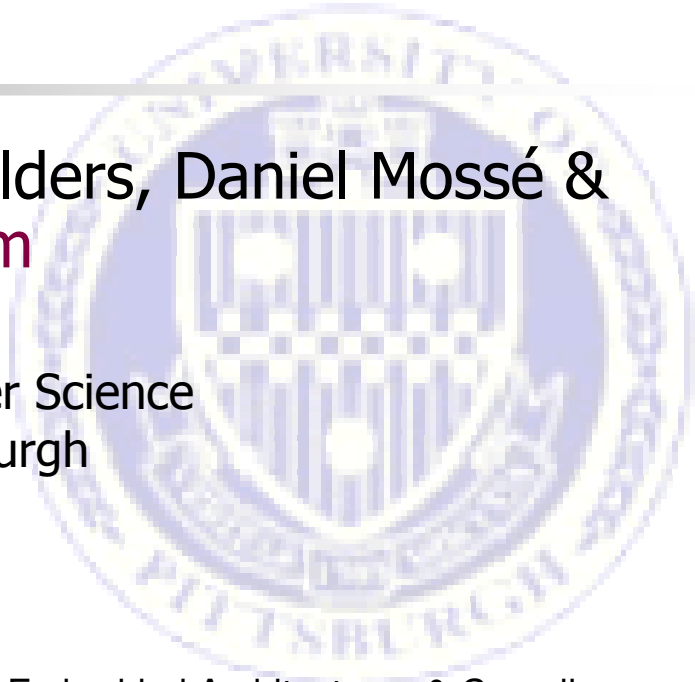
Integrated CPU and Cache Power Management in Multiple Clock Domain Processors

Nevine AbouGhazaleh, Bruce Childers, Daniel Mossé &
Rami Melhem

Department of Computer Science
University of Pittsburgh



HiPEAC 2008: International Conference on High Performance Embedded Architectures & Compilers

Introduction

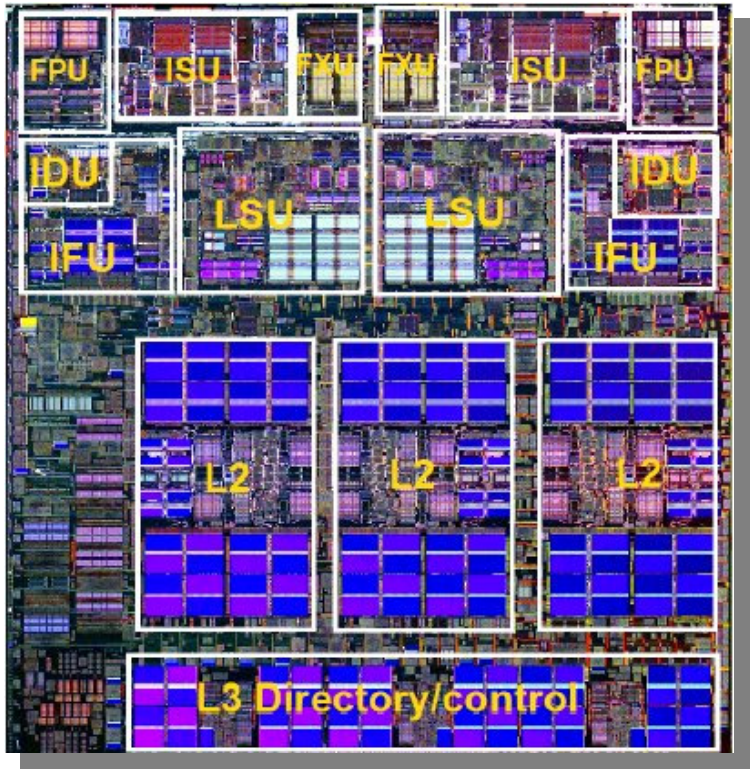
- Energy is a first class resource
- Dynamic Voltage scaling (DVS) in processors
  - Reduce processor's voltage and frequency *linearly* to reduce energy *quadratically*.
  - DVS is in widespread use today
    - Mainstream processors have DVS

Multiple Clock Domains Chips

- Trend: Large chips
- Problem
  - Hard to synchronize
- Solution:
  - Multiple clock domains (MCD): Globally Asynchronous, Locally synchronous (GALS) [DAC'99]

- MCD chips:
  - Each domain has separate clock and power supply
  - Allow finer granularity of power control

IBM Power 5



IFU: Instruction Fetch Unit, **ISU**: Instruction issue unit, **IDU:** instruction decode unit, **LSU:** Load Store Unit, **FPU:** Floating point Unit

Related work

DVS in MCD chips

- Offline:
  - Profile based [Magklis et al.- ISCA'03]
    - Insertion of reconfiguration instructions into applications
- Online:
  - Attack-Decay [Semeraro et al. - MICRO'02]
    - Vary domain's voltage according to domain's workload
  - Formal solution [Wu et al. - ASPLOS'04]
    - Model each domain as queuing system, and solve
  - Adaptive [Wu et al. - HPCA'05]
    - Self-tuned reaction times adaptive to workload changes

Problem

- Set the voltage and frequency for each domain to reduce the *Combined* energy consumption with little impact on delay

- Previous solutions:

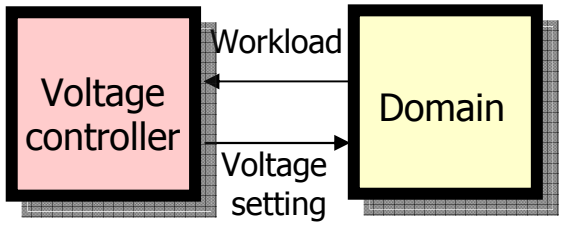
  Online DVS policies in each domain based on local workloads of each domain

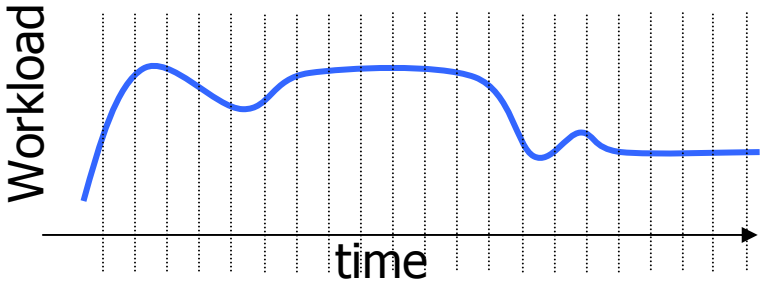
- Our solution:

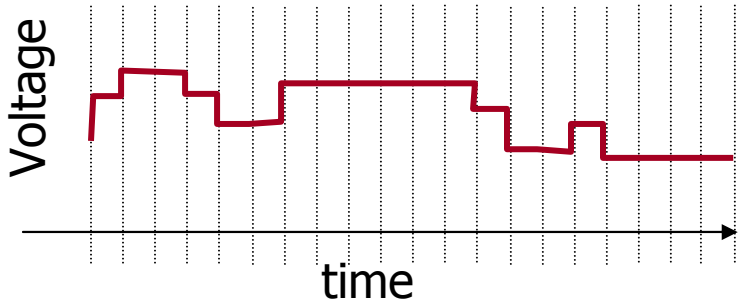
  Online *Integrated* Dynamic Voltage Scaling in CPU-core and L2 cache.

Online DVS : Basic Idea

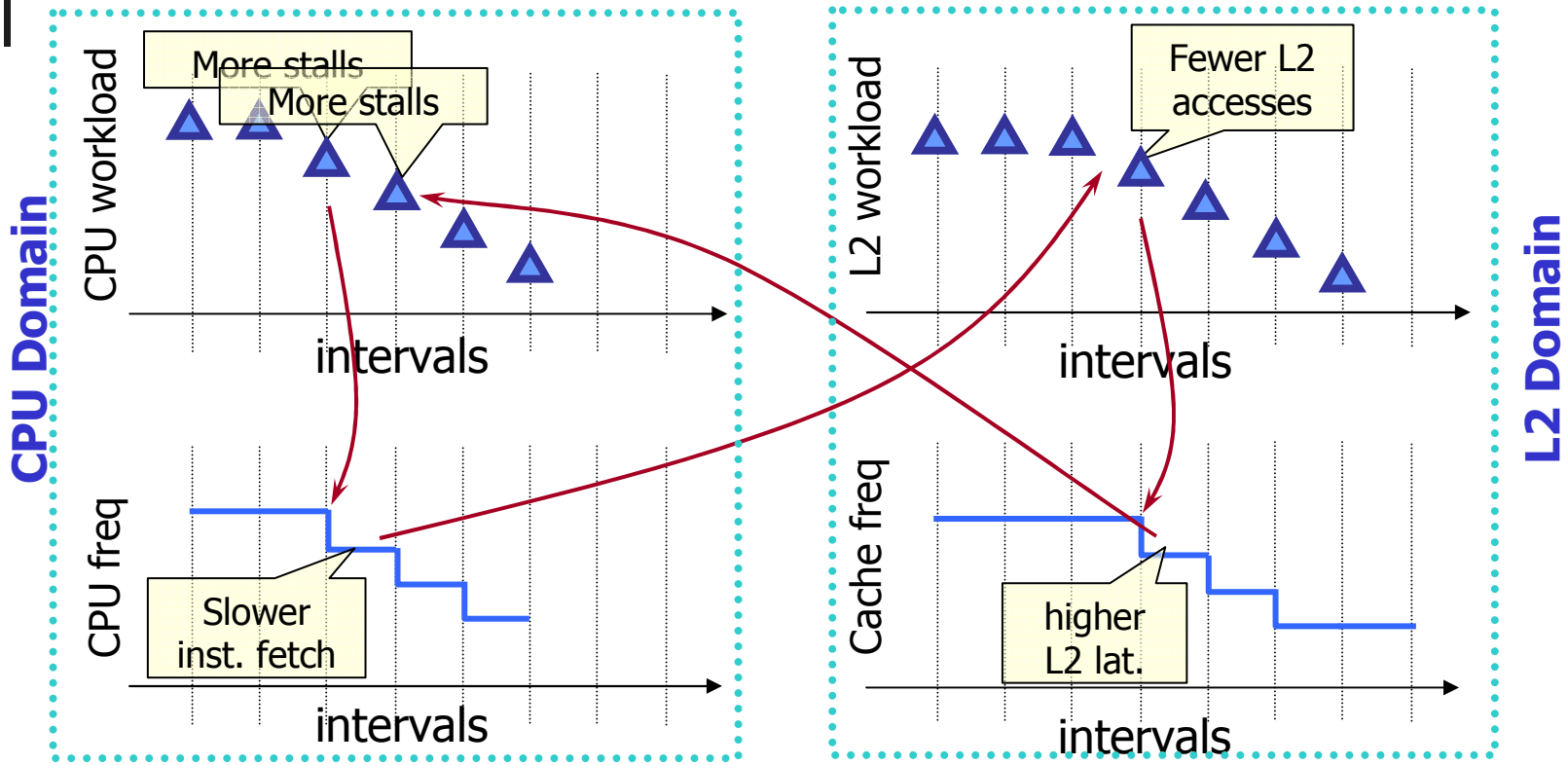
- Set the frequency of a domain based workload
  - speed α workload
- Workload measured by performance counters
  - # instruction for CPU-core
  - L2 access for L2-cache
- Control loop!
  - periodically measure workload to set speed







Positive Feedback Intervals



Freq. of positive feedbacks

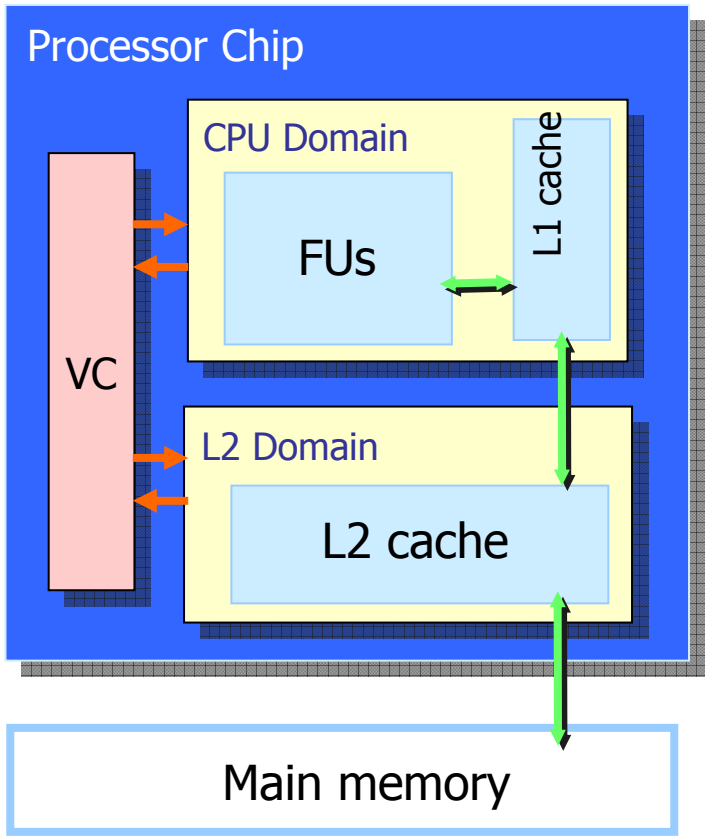
ammp	apsi	art	bzip	equake	gcc	gzip	lucas
20.5%	18.7%	4.03 %	8.18%	0.02%	20.97%	4.59%	12.7%

mcf	mesa	mgrid	parser	swim	twolf	vortex	vpr
0.85%	6.21%	5.9%	9.06%	0.2%	25.79%	6.50%	16.12%

Integrated DVS policy

Consider two domain chips
CPU-core and L2 cache

VC: Voltage Controller
set *configuration* using global info (observed activity) → configure using DVS



Online-IDVS Policy

- Rules for setting voltages
- Break positive feedback
  - in rules 1 & 5

rule #	Event to monitor		Action by proposed plcy	
	IPC	L2access	V_c	$V_\$$
1	⇑	⇑	⇓	⇑
2	⇑	⇓	⇑	⇓
3	⇑	-	⇑	-
4	⇓	⇑	⇓	⇑
5	⇓	⇓	-	⇓
6	⇓	-	⇓	-
7	-	⇑	-	⇑
8	-	⇓	-	⇓
9	-	-	-	-

Positive feedback scenarios:

Workloads *increase* in both domains

Rule #	IPC	L2 access	V_c	$V_\$$
1	⇑	⇑	⇓	⇑

- Indicate a start of *memory bound* program phase
- Preemptively reduce core speed
  - avoid overloading L2 cache domain with excess traffic.
- Increasing core speed will exacerbate load in both domains.
- Decrease core speed rather than keeping it unchanged to save core energy
  - likely longer core stalls due to expected higher L2 traffic.

Positive feedback scenarios:

Workloads *decrease* in both domains

Rule #	IPC	L2 access	V_c	$V_\$$
5	⇓	⇓	-	⇓

- Decrease in IPC is *not* due to higher L2 traffic.
- Longer core stalls are a result of local core activity
  - Increasing or decreasing the core speed may not eliminate the source of these stalls.
  - Change core speed risk unnecessarily increasing execution time or energy consumption.
- Maintain the core speed unchanged
  - break the positive feedback scenario without hurting delay or energy.

Experimental Results

- Setup:
  - Simulator: Simplescalar with MCD extension
  - Models:
    - two domains (CPU, L2)
    - five domains (Fetch U, Int. U, FP U, Reorder buffer and L2)
      - Fetch + L2 are correlated, other are independent.
    - Simple and high performance processors
  - Compare against:
    - no-power management and
    - Isolated DVS in each domain [Semeraro et al.'03]
  - Metric: energy-delay product

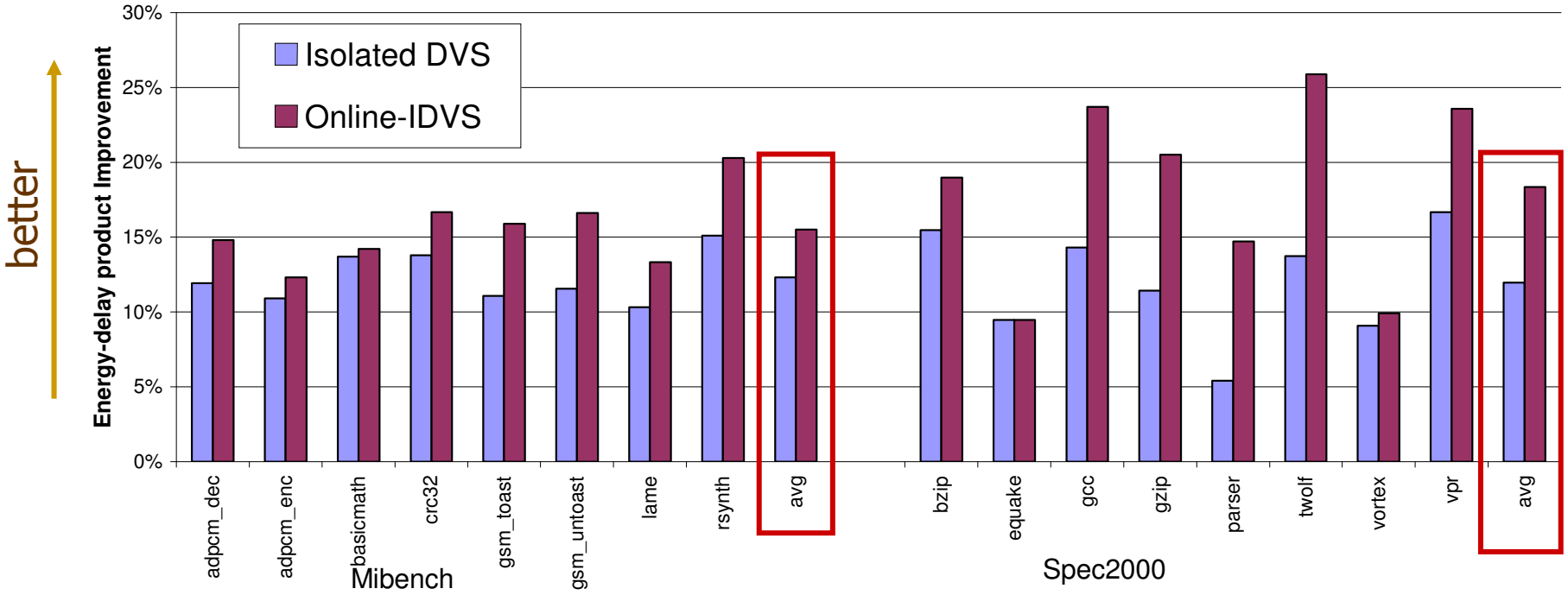
Simulation Configuration

Simple embedded (Config. A) and High performance (Config. B) processors

Parameter	Config A	Config B
Dec./Iss. Width	1/1	4/6
dL1 cache	64KB, 2-way	64KB, 2-way
iL1 cache	64KB, 2-way	64KB, 2-way
L2 Cache	1MB DM	1MB DM
L1 lat.	2 cycles	2 cycles
L2 lat.	12 cycles	12 cycles
Int ALUs	2+1 mult/div	4+1 mult/div
FP ALUs	1+1 mult/div	2+1 mult/div
INT Issue Queue	4 entries	20 entries
FP Issue Queue	4 entries	15 entries
LS Queue	8	64
Reorder Buffer	40	80

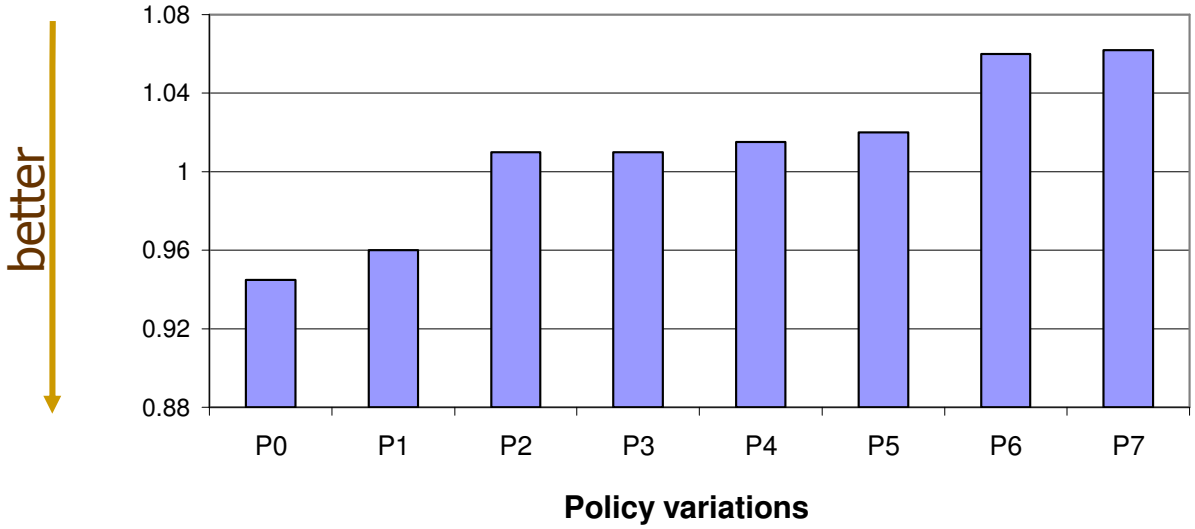
Results

Normalized to no-DVS policy



Online-IDVS improves E.D up to 26% over no-DVS and up to 12% over isolated DVS policy.

Policy Variations

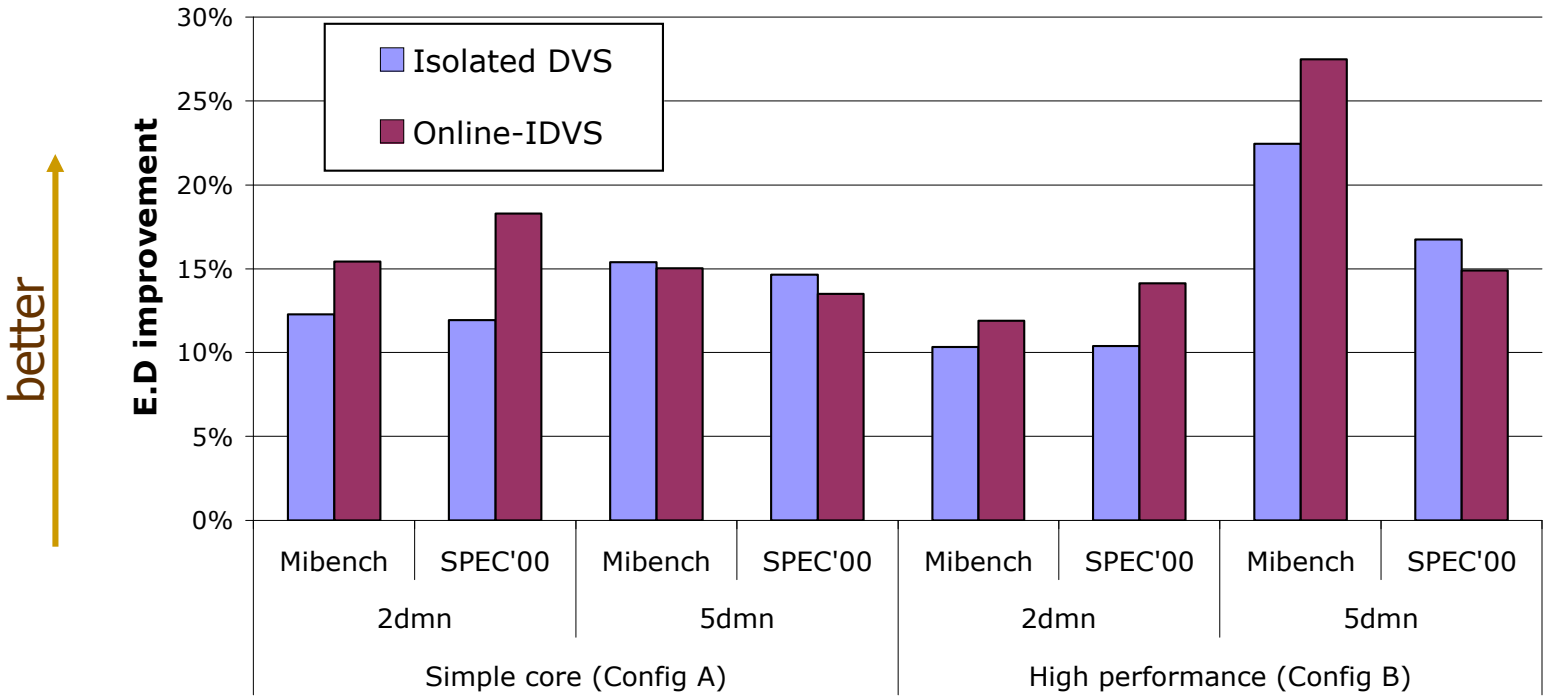


rule #	P0		P1		P2		P3		P4		P5		P6		P7	
	V_c	$V_\$$	V_c	$V_\$$	V_c	$V_\$$	V_c	$V_\$$	V_c	$V_\$$	V_c	$V_\$$	V_c	$V_\$$	V_c	$V_\$$
1	⇓	⇑	⇓	–	⇓	–	⇓	–	–	⇑	–	–	–	⇑	–	⇑
5	–	⇓	–	⇓	–	–	⇑	–	–	⇓	⇑	–	⇑	–	–	–

Resolving positive feedback loops using P0 is most effective.

Sensitivity Analysis

Varying number of domains and Processor Configurations



Online-IDVS is more effective (higher E.D improvement) in high performance proc.

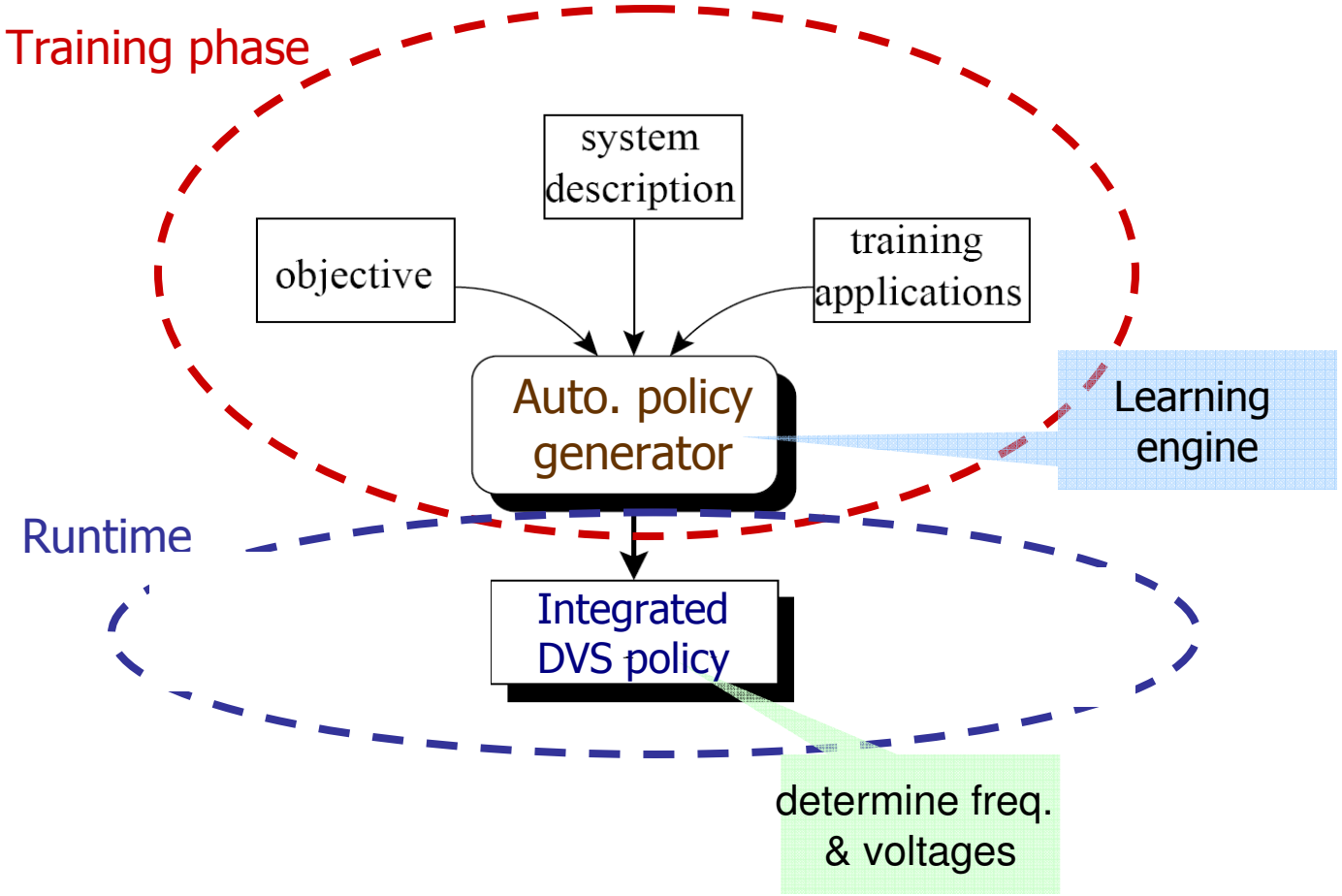
Relative improvement over isolated DVS is higher in simple core with two domains.

IDVS for multiple domains

- It is hard to reason about the interaction between more than two domains.
- Hence, designing correlated policies for more than two domains is not intuitive
- Statistical Machine Learning techniques can discover the correlation and generate IDVS policies

An off-line solution:

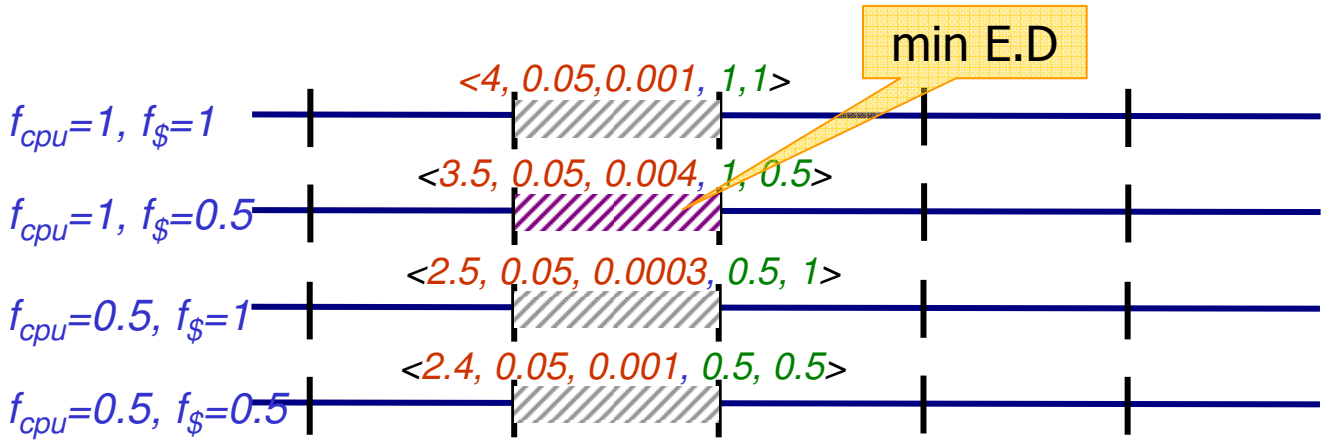
Machine learning based IDVS



An off-line solution:

ML-IDVS : Offline learning

- Divide training programs into “code sections”
- Simulate the execution of a large number of code sections for all combination of frequencies.
- Characterize each “code section” by a parameter tuple (CPI, L2PI, MPI, execution frequencies)
- Record the best operating frequency for each class of “code sections” (best WRT some metric – ex. ED).

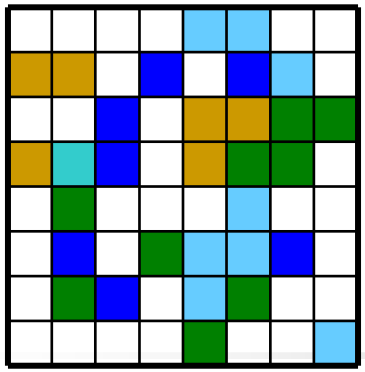


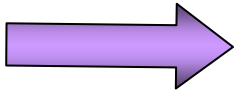
An off-line solution:

ML-IDVS : Offline learning

- Store the best frequencies for each class of "code sections" a table
- At run-time, use monitors to classify the current code section and use the table to setup the frequencies of the next section.
- May apply ML learning to translate the "very large" table into a small number of rules.

State Table indexed by
<f_{CPU}, f_{L2}, CPI, L2PI, MPI>
Stores best frequency combinations





Machine learning to generate policy rules

If (L2PI >0.32) and (CPI<2) then f_{L2}= 1GHz
else f_{L2} = 0.5GHz
If (MPI >0.002) and (CPI<3) then f_{cpu} = 0.5GHz
else f_{cpu} = 1GHz

ML-IDVS v.s. Online-IDVS

	ML-IDVS	Online-IDVS
E.D improvement	Higher - due to offline profiling	
For more domains	Time-consuming rule generation	Less intuitive domain interactions
Diverse set of applications in system	higher training overhead	Same policy
Optimizations	Work with multiple objectives	

Conclusion

- Using existing hardware, we can increase the energy saving by considering the interaction between domains when applying DVS policies in systems with MCD
- Integrated DVS is more effective in simple cores.
- When the number of domains increases, we need a more systematic way to discover the interaction between the domains
- Can be applied to Chip multiprocessors