

Positional Adaptation of Processors: Application to Energy Reduction

Michael Huang
University of Rochester

UNIVERSITY OF
ROCHESTER
Electrical & Computer Engineering

Jose Renau, Josep Torrellas
University of Illinois



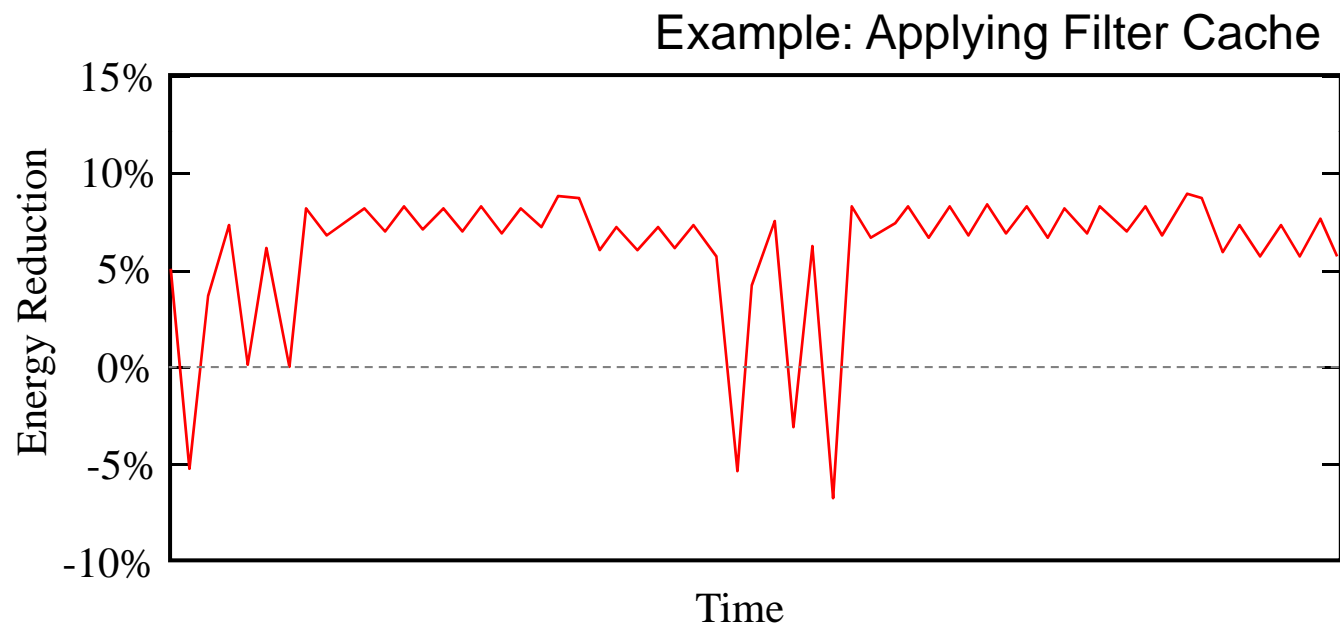
Motivation

- Adaptivity is increasingly important
 - Wider variety of applications
 - More complex applications
 - ⇒ Behavior (demand) changes within/across apps
- ...and more common
 - Different architectural components will be adaptable



Motivation (cont)

- Controlling adaptation has to be precise



Contribution: Positional Adaptation

- Adaptation tied to code position (address)
 - Testing the effects of adaptations
 - Applying adaptations
- Advantages
 - More accurate in assessing effects
 - More effective adaptation: global scheduling

⇒ More effective than conventional temporal solution



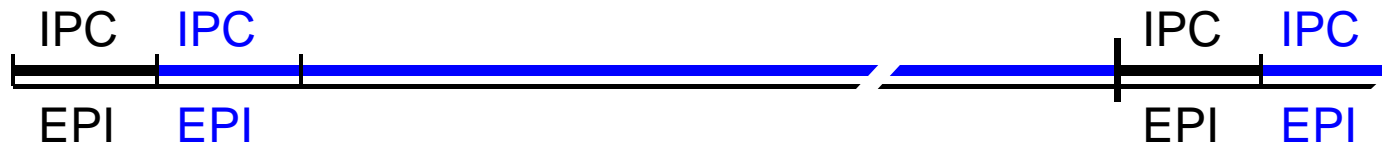
Outline

- Mechanisms of positional adaptation
- Different implementations
- Evaluation/Discussion
- Related work
- Conclusions



Conventional Adaptation: *Temporal*

- A simple algorithm – DEETM'
 - Huang *et. al.* [MICRO'00]

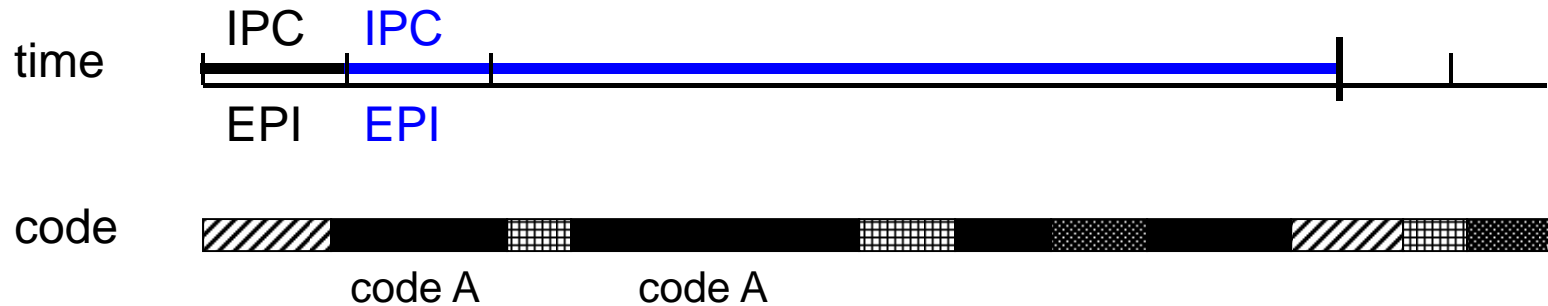


- Improvement: variable interval
 - Balasubramonian [MICRO'00]: Phase change detection
 - Dhodapkar [ISCA'02]: Unnecessary tuning inhibition

⇒ Testing and application (of adaptation) tied to time



Problems with Temporal Adaptation

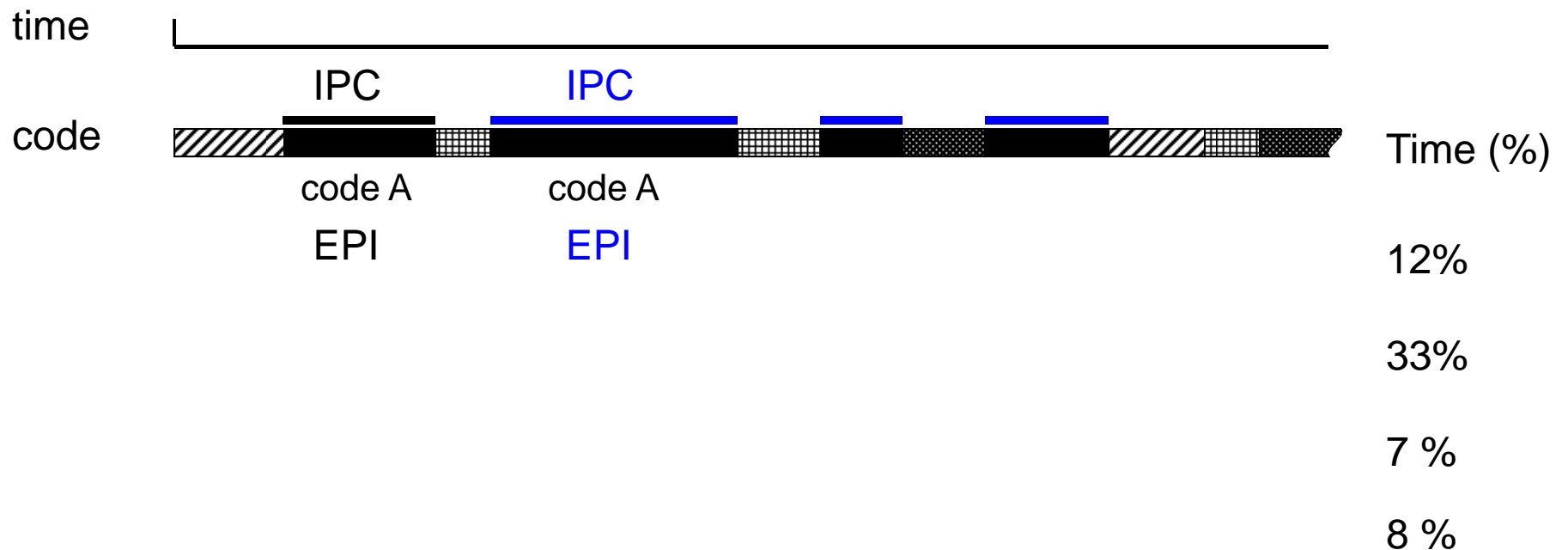


- Testing on different code sections
- Applying chosen adaptation on yet a different section



Proposed Adaptation: *Positional*

- Testing and application tied to code address
 - Less variation: more accurate testing
 - Enables *global* adaptation control



Implementing Positional Adaptation

- When to adapt: instrumentation (e.g. major subroutines)
- What to adapt: decision

WHAT

Decision Making
Static Dynamic

WHEN


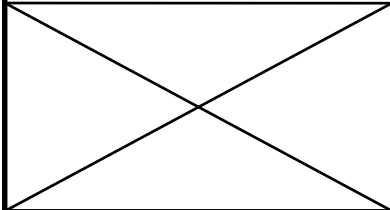
Instrumentation Static	SISD	SIDD
Dynamic		DIDD



SISD: Deciding *When* & *What* Statically

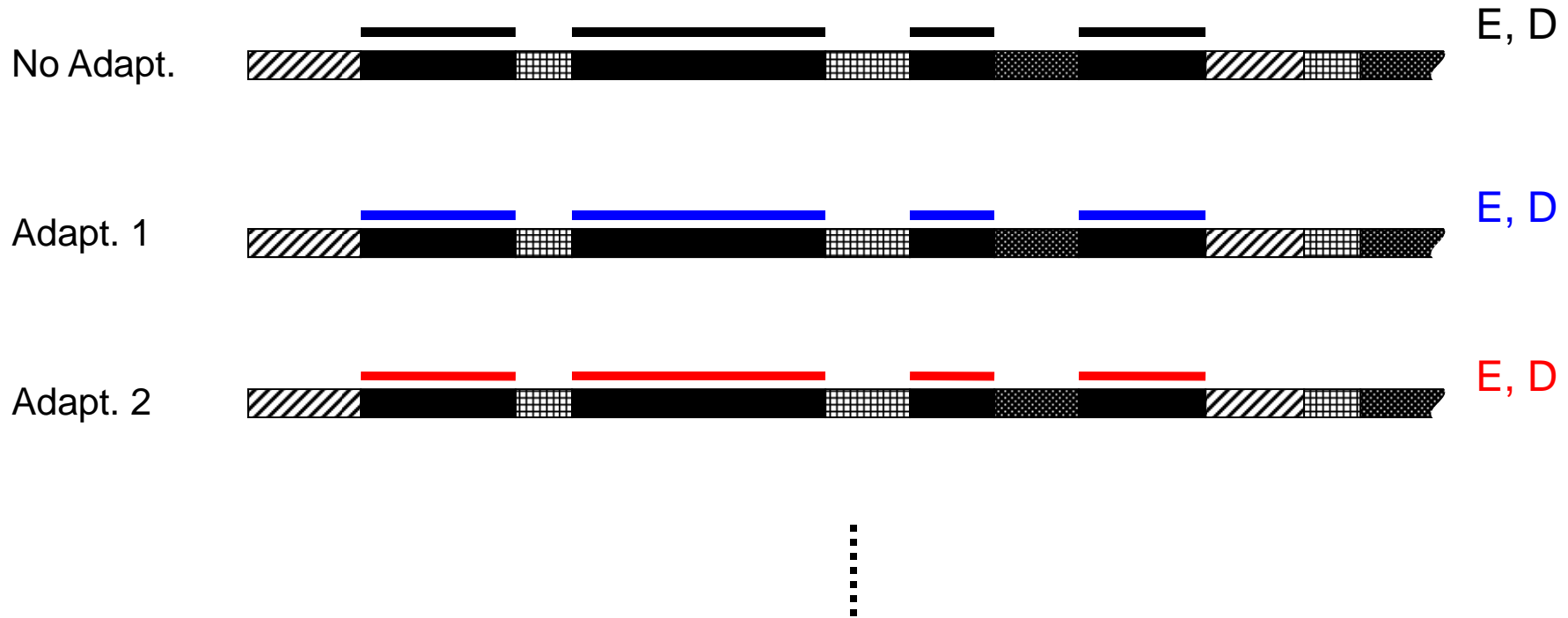
WHAT

Decision Making

		Static	Dynamic
WHEN	Instrumentation	 SISD	SIDD
	Dynamic		DIDD



SISD: Deciding *When & What* Statically



SISD: Deciding *When* & *What* Statically

Off-line profiling of ΔD , ΔE

- Per adaptation technique
 - Per subroutine
- } a pair

$$\Delta E = E_{\text{orig}} - E_{\text{activated}}$$

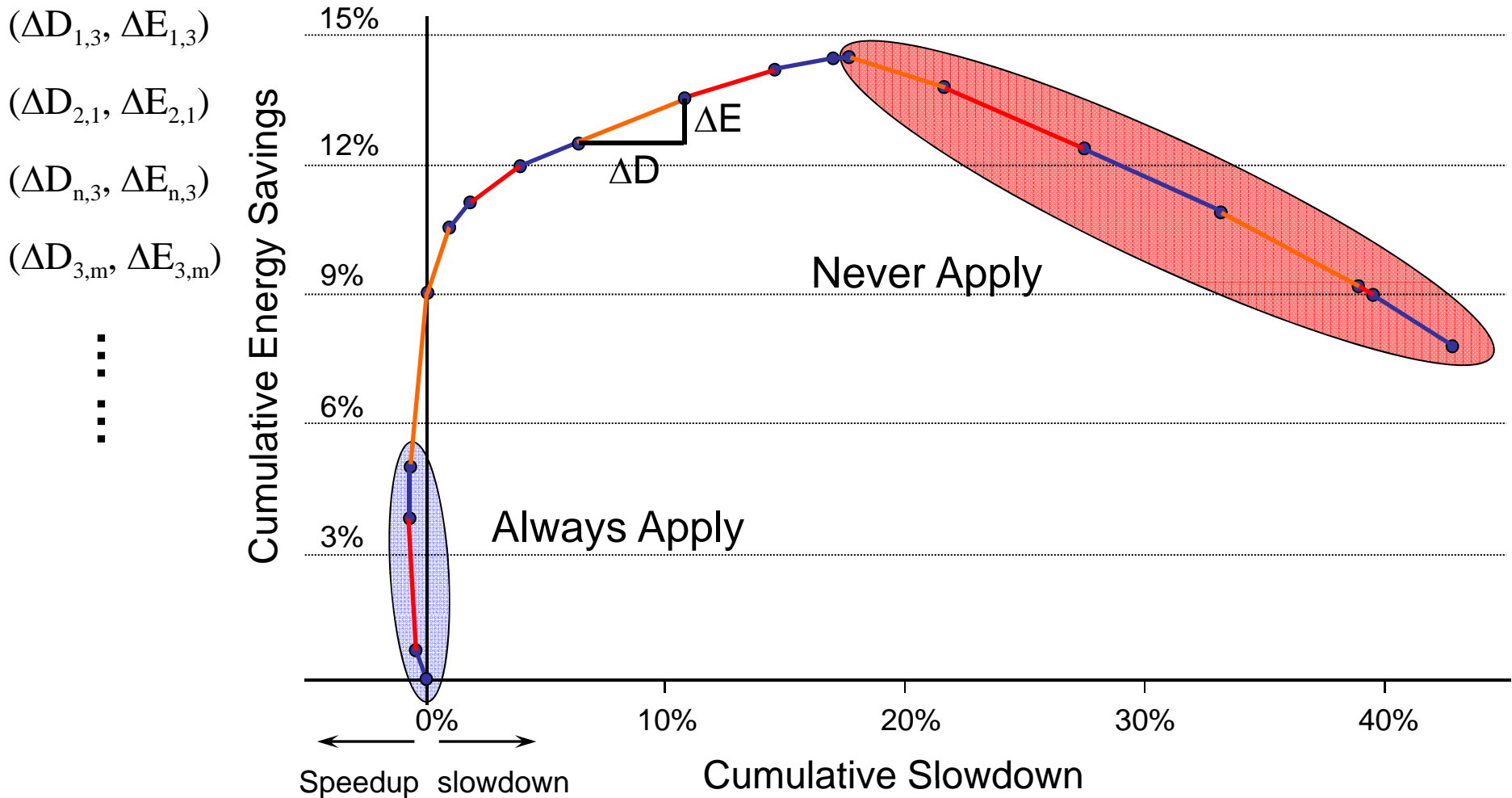
$$\Delta D = D_{\text{activated}} - D_{\text{orig}}$$

i \ j	Adapt ₁	Adapt ₂		Adapt _m
sub ₁	($\Delta D_{i,j}$, $\Delta E_{i,j}$)	($\Delta D_{1,3}$, $\Delta E_{1,3}$)	
sub ₂	($\Delta D_{2,1}$, $\Delta E_{2,1}$)			
				($\Delta D_{3,m}$, $\Delta E_{3,m}$)
sub _n		($\Delta D_{n,2}$, $\Delta E_{n,2}$)		

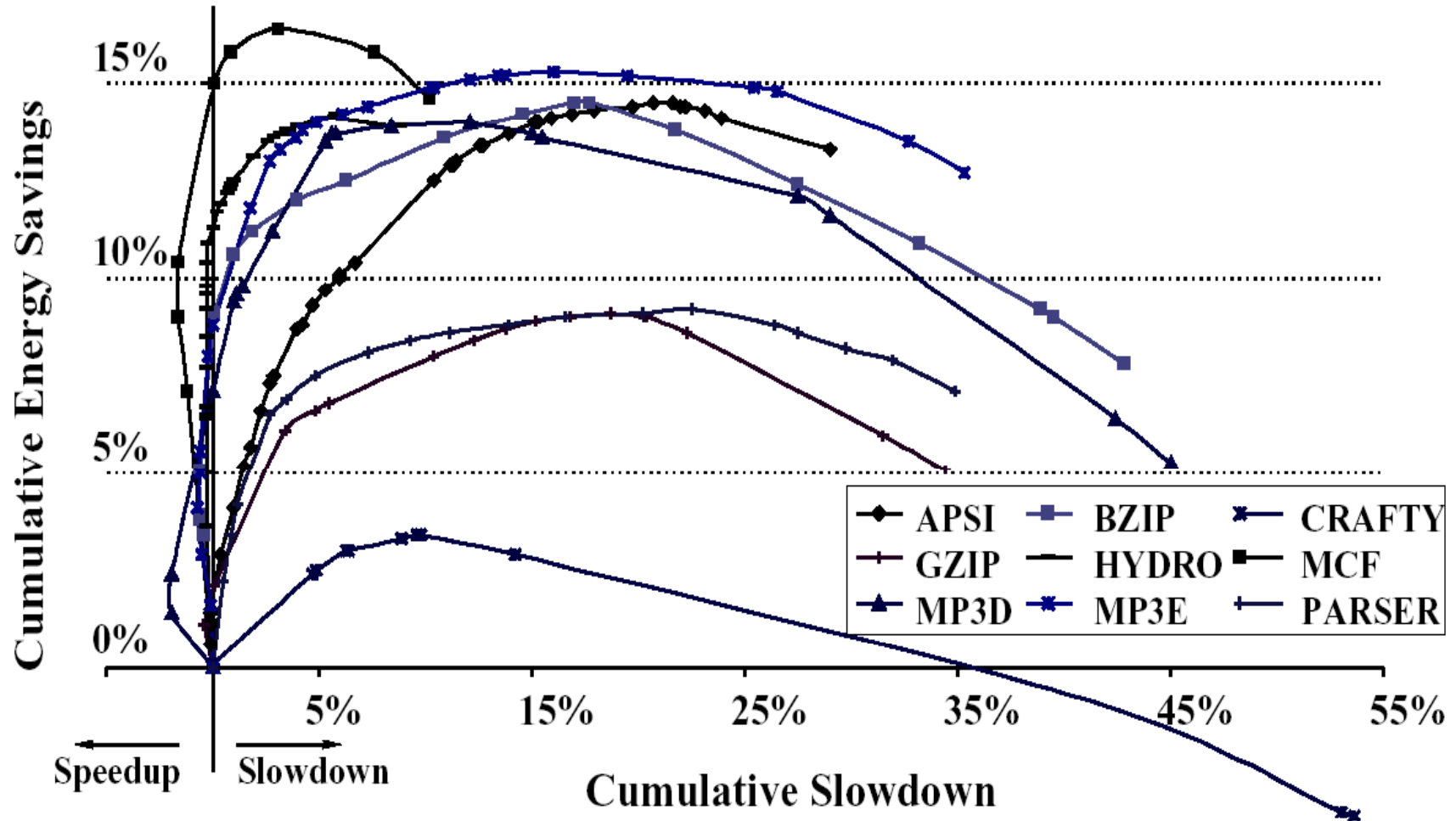
⋮



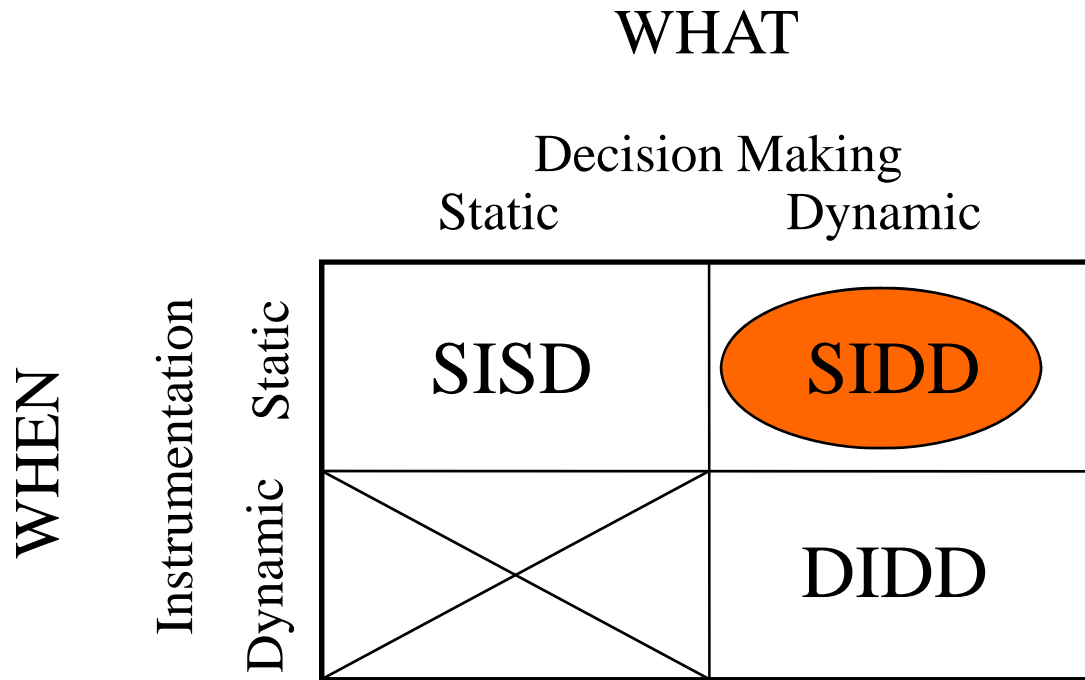
E-D Tradeoff Differs Greatly across Pairs



Trends for Applications



SIDD: Deciding *What* Dynamically



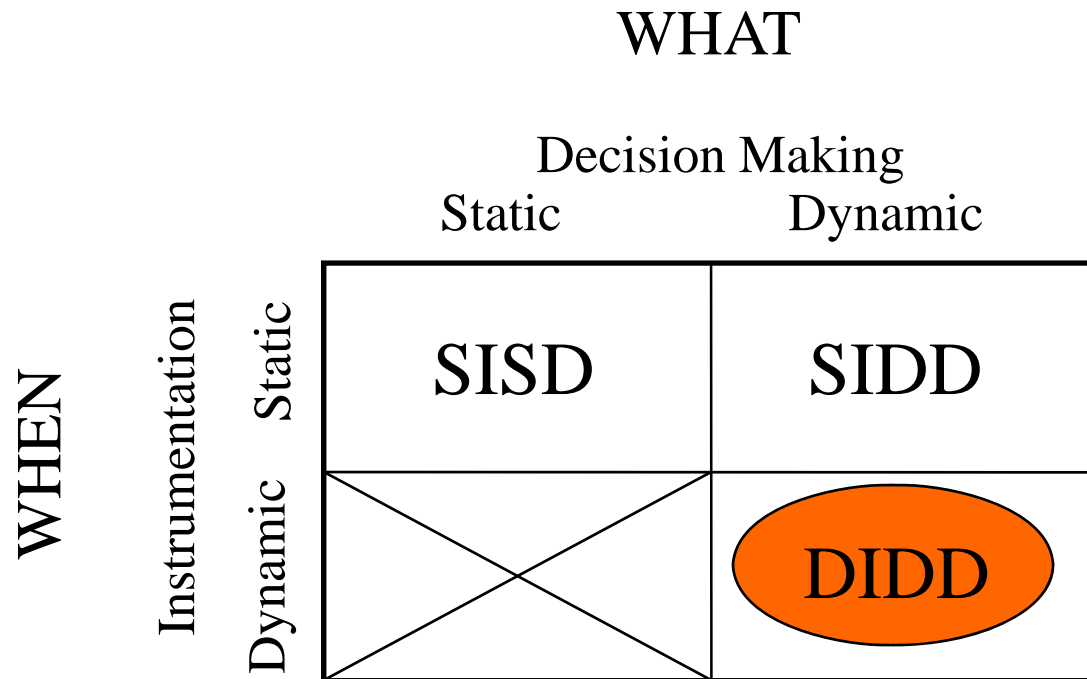
SIDD: Deciding *What* Dynamically

- Measure ΔD , ΔE for the first few invocations

$i \backslash j$	Adapt ₁	Adapt ₂		Adapt _m
sub ₁	$(\Delta D_{i,j}, \Delta E_{i,j})$		
sub ₂				
sub _n				



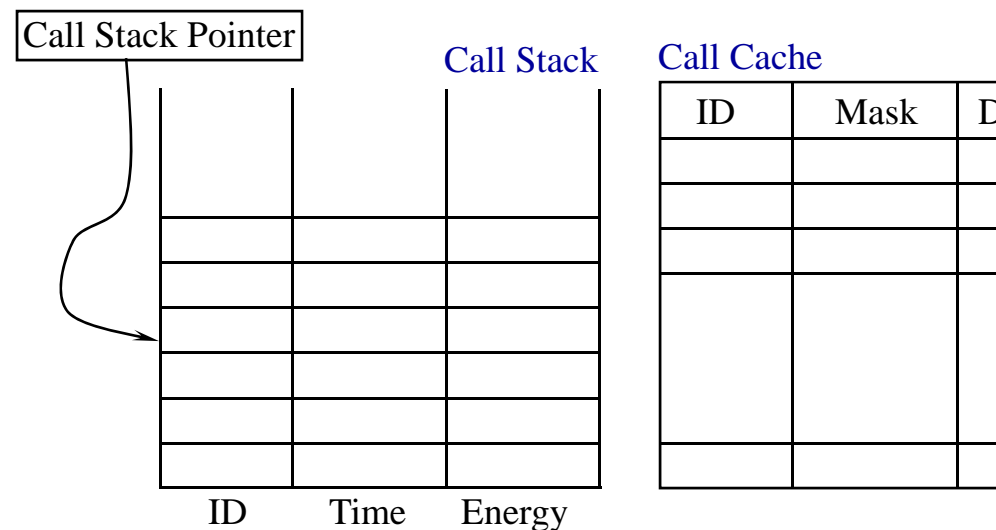
DIDD: Deciding *When* & *What* Dynamically



DIDD: Deciding *When* & *What* Dynamically

Use hardware to

- Identify major subroutines
- Facilitate efficient profiling



Tradeoffs for Different Positional Schemes

Schemes	Pros	Cons	Target market
SISD	<ul style="list-style-type: none">▪ Global info	<ul style="list-style-type: none">▪ Profiling	<ul style="list-style-type: none">▪ Embedded system▪ Specialized server
SIDD	<ul style="list-style-type: none">▪ Less profiling	<ul style="list-style-type: none">▪ Limited global info▪ Runtime overhead	<ul style="list-style-type: none">▪ General purpose
DIDD	<ul style="list-style-type: none">▪ No profiling	<ul style="list-style-type: none">▪ As SIDD + Hardware support	<ul style="list-style-type: none">▪ Unavailable offline profile: e.g. dynamically generated binary



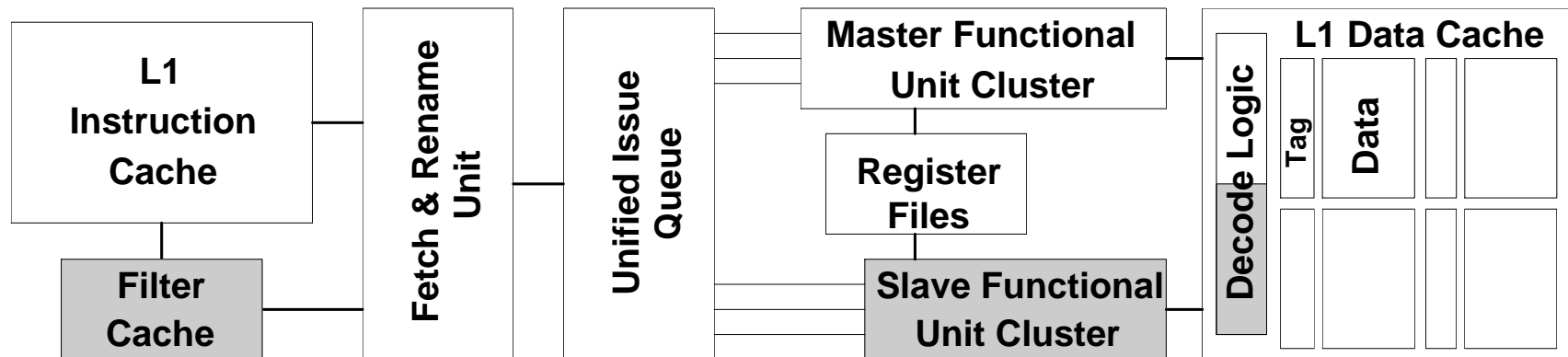
Experimental Setup

- Processor: 6-issue OOO
- Mem System: 32KB L1 + 512KB L2 + 2 Rambus
- Variety of applications
 - SPEC-Int: bzip2, crafty, gzip, mcf, parser
 - SPEC-FP: apsi, hydro2d
 - Multimedia: mp3enc, mp3dec
- Energy largely based on Wattch [Brooks'00]



Adaptive Low-Power Techniques

- Adaptations (framework not limited to these):
 - Instruction filter cache
 - D-L1 phased cache
 - Disable slave FU cluster
- Save energy at a performance cost

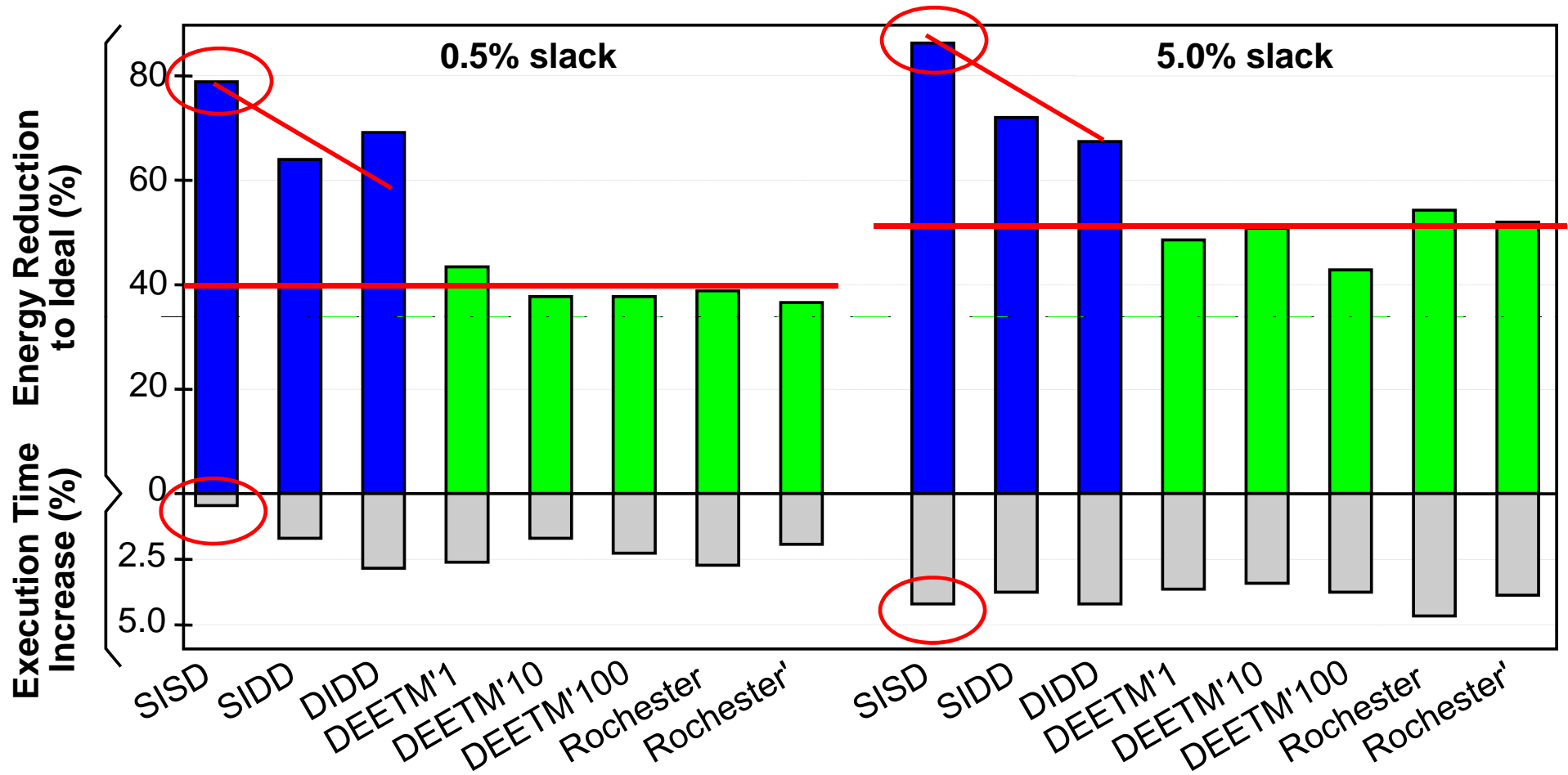


Schemes Compared

- Ideal: best our low-power techniques can do
- Positional
 - SISD
 - SIDD
 - DIDD
- Temporal
 - DEETM' (interval: 1,10, and 100 μ s): [Huang *et. al.*]
 - Rochester: [Balasubramonian *et. al.*]
 - Rochester': enhancement from [Dhodapkar *et. al.*]



Positional Schemes More Effective



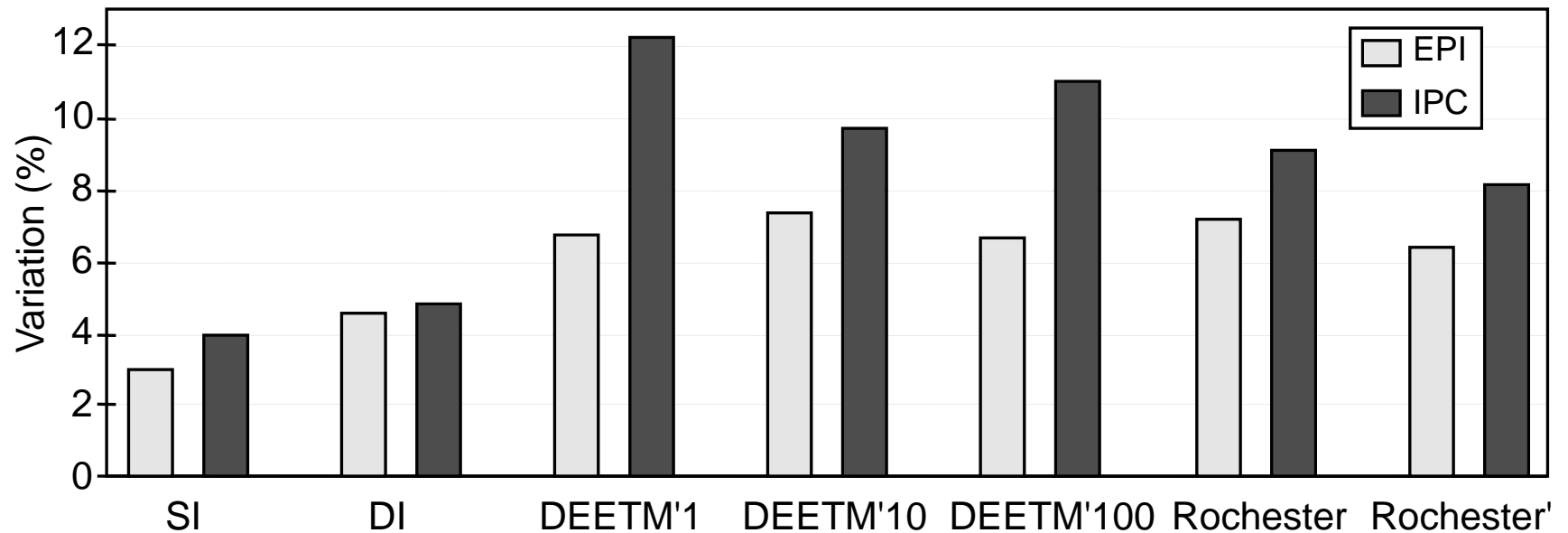
Key Observations

- Positional schemes are more effective than temporal ones
 - Especially under small slack
- SISD (static *when & what*) is close to ideal
 - ⇒ Code behavior in major subroutines appears largely homogenous (to our low-power techniques)



Accuracy in the Testing Period

- Behavior fluctuation across intervals w/o adaptations



⇒ Lower fluctuation for positional: more accurate testing



Related Work

- Temporal schemes (many)
 - [Balasubramonian'00] [Bahar'01] [Folegnani'01] ...
- Schemes with some positional flavor
 - Adaptation for multimedia [Hughes'01][Sasanka'02]
 - Working set signature [Dhodapkar'02]
 - Hot spot-based reconfiguration [Iyer'01]



Conclusions and Future Work

- Positional more effective than temporal schemes
 - Different invocations of same code usually:
 - Have similar behavior
 - React similarly to the same adaptation
- Code behavior in subroutines appears largely homogenous (to our low-power techniques)
- Positional adaptation not limited to low-power:
 - Adaptation for high-performance



Positional Adaptation of Processors: Application to Energy Reduction

Michael Huang
University of Rochester



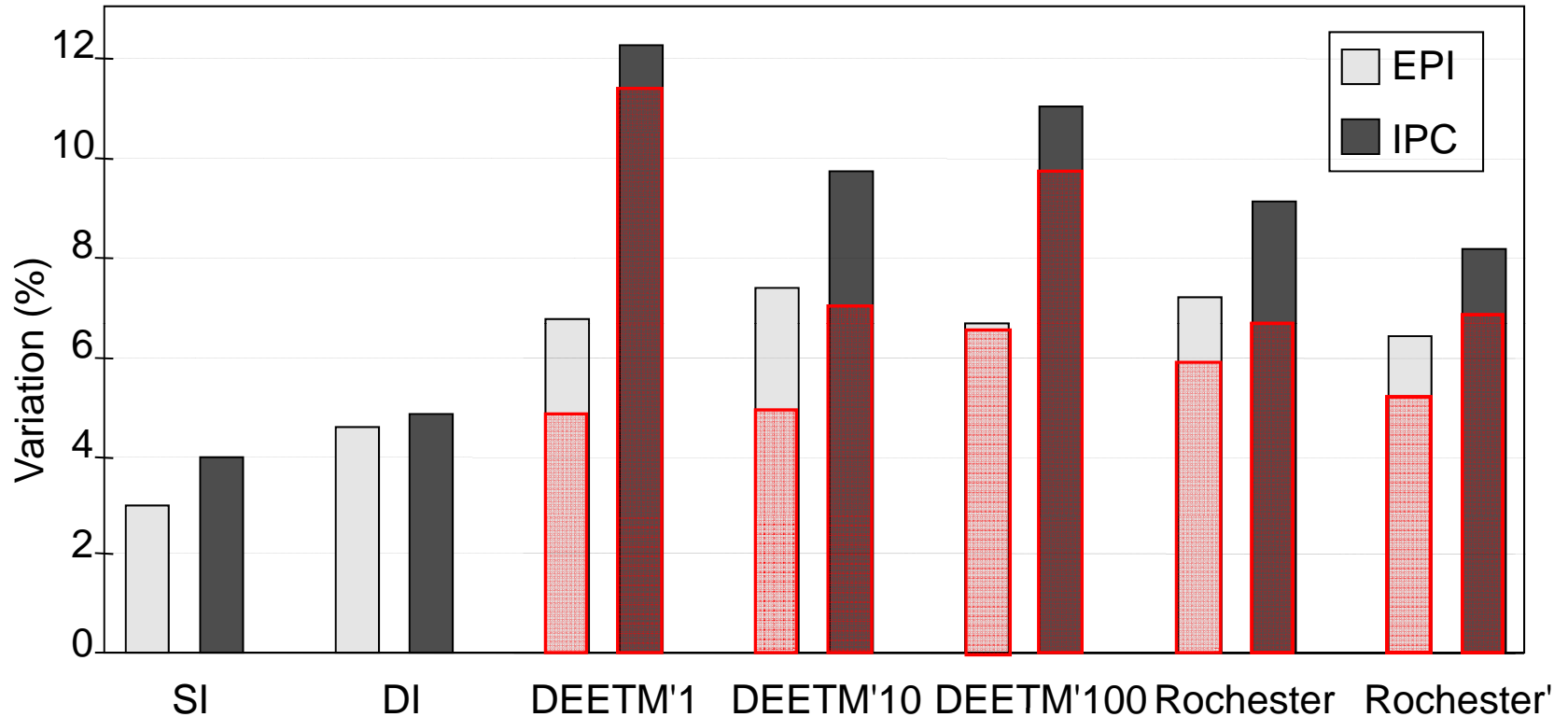
<http://www.ece.rochester.edu/~mihuang>

Jose Renau, Josep Torrellas
University of Illinois



<http://iacoma.cs.uiuc.edu>

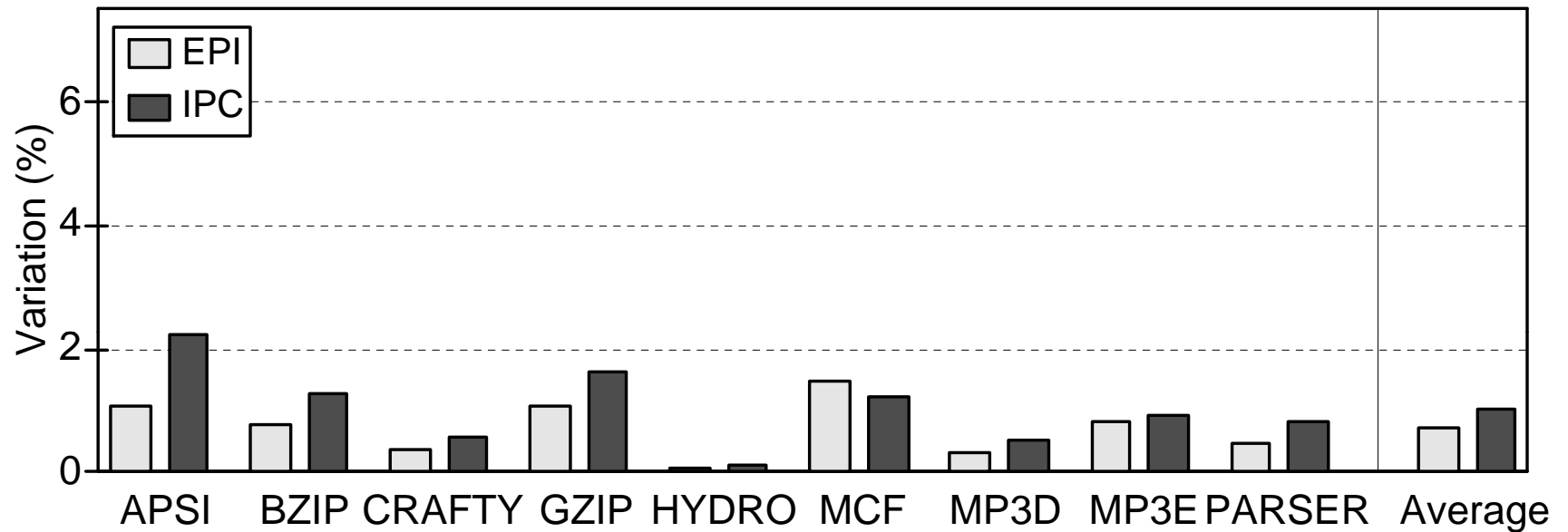
Reduced *ref* vs. *ref*



High variation not due to reduced input set



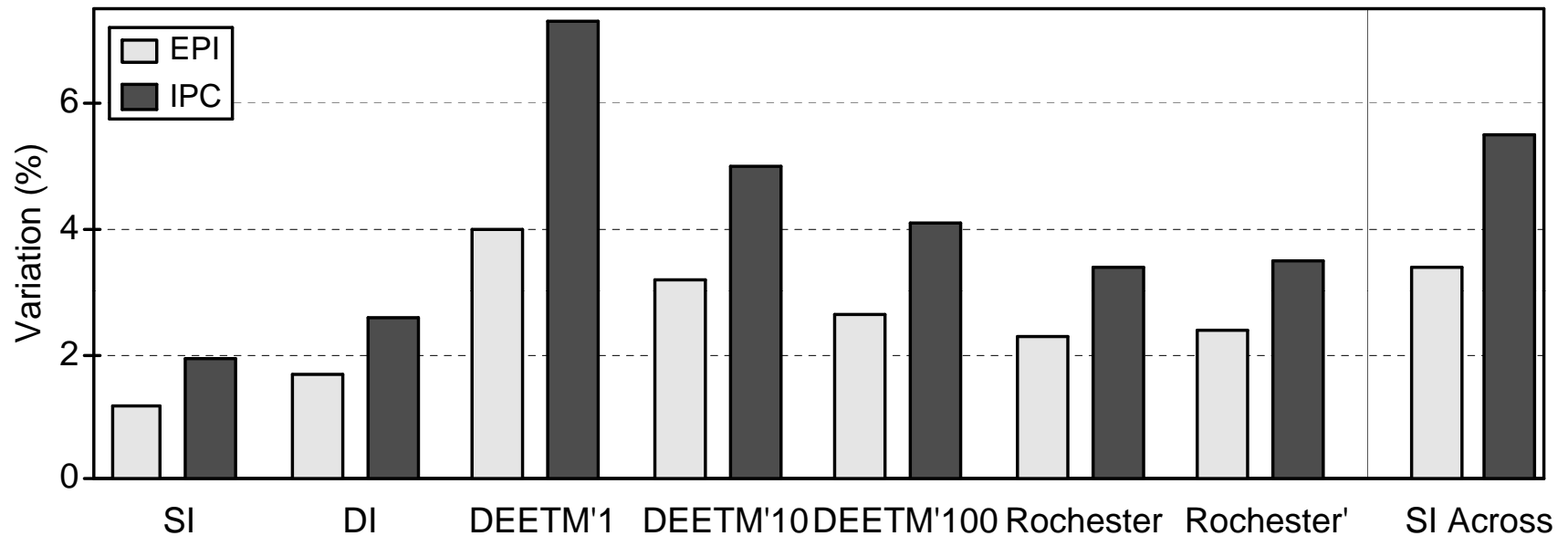
Influence of Input Sets Small



Input-induced variation less than invocation to invocation variation



Impact Variation in the Steady-State



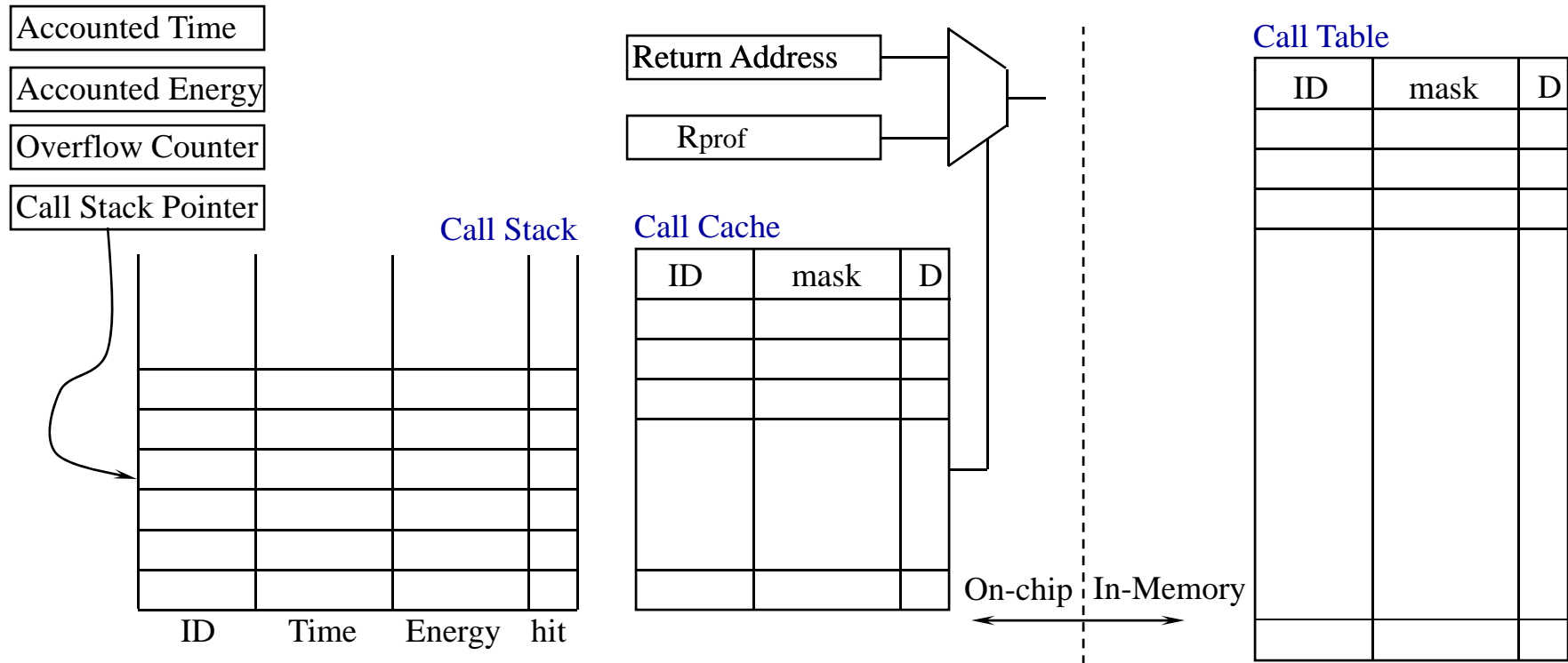
- Subroutine: homogenous among invocations, heterogeneous across subroutines
- More sophisticated temporal schemes better identify behavior change



Architectural Support for DIDD

Architecture support to

- Identify top subroutines
- Facilitate efficient profiling



Phased Cache [Hasegawa'95]

- Sequentially access tag and data
 - Longer latency, lower E consumption

