Energy-Driven Hardware Adaptations for Multimedia Applications on General-Purpose Processors

Sarita Adve

with

Christopher J. Hughes, Rohit Jain, Praful Kaul,

Chanik Park, Ruchira Sasanka, Jayanth Srinivasan

Department of Computer Science

University of Illinois at Urbana-Champaign

http://www.cs.uiuc.edu/~sadve

Motivation and Goals

Multimedia and communication will be critical workloads

- Video, speech, images, wireless communication

Traditionally used ASICs, DSP processors, BUT

- Now general-purpose processors attractive
- Increasing application complexity $\Rightarrow$
  - Need for compilers, upgradeability, portability

How to build general-purpose architectures for new applications?

New Challenges

Stringent, dynamic, multidimensional resource constraints

Computation

Real-time

Energy

Battery life

Bandwidth

Wireless losses, congestion

New Challenges and Opportunities

$$\begin{tikzcd}
 & \text{Stringent, dynamic, multidimensional resource constraints} \arrow[dl] \arrow[d] \arrow[dr] & \\
\textbf{Computation} & \textbf{Energy} & \textbf{Bandwidth}
\end{tikzcd}$$

Computation

Real-time

Energy

Battery life

Bandwidth

Wireless losses, congestion

Real-time ⇒

Can slow processing to save energy

Soft correctness criteria ⇒

Can tradeoff output quality for resource usage

Resilience to losses and imprecise computation

Lots of parallelism in applications ⇒

Old and new architectural techniques to exploit parallelism

Key Themes in our Work

Dynamic system and flexible output quality ⇒

- Make hardware adaptive, flexible
  - Change configuration to optimize for current condition

Key Themes in our Work

Dynamic system and flexible output quality ⇒

- Exploit adaptation in all system layers
  - Hardware, network, operating system, application
  - *Collaborate* to optimize for current system conditions
- Integrated cross-layer adaptation control

With Jones, Kravets, Nahrstedt

Key Themes in our Work

Dynamic system and flexible output quality ⇒

- Exploit adaptation in all system layers
  - Hardware, network, operating system, application
  - *Collaborate* to optimize for current system conditions
- Integrated cross-layer adaptation control

With Jones, Kravets, Nahrstedt

Resilience to losses ⇒

- Aggressive speculation for performance and energy
- New models and techniques for fault tolerance

Key Themes in our Work

Dynamic system and flexible output quality $\Rightarrow$

- Exploit adaptation in all system layers
  - Hardware, network, operating system, application
  - *Collaborate* to optimize for current system conditions
- Integrated cross-layer adaptation control

With Jones, Kravets, Nahrstedt

Resilience to losses $\Rightarrow$

- Aggressive speculation for performance and energy
- New models and techniques for fault tolerance

Parallelism $\Rightarrow$

- Exploit past experience w/ instruction and thread parallelism

Key Themes in our Work

Dynamic system and flexible output quality ⇒

- Exploit adaptation in all system layers
  - Hardware, network, operating system, application
  - *Collaborate* to optimize for current system conditions
- Integrated cross-layer adaptation control

With Jones, Kravets, Nahrstedt

Resilience to losses ⇒

- Aggressive speculation for performance and energy
- New models and techniques for fault tolerance

Parallelism ⇒

- Exploit past experience w/ instruction and thread parallelism

Issues in Hardware Adaptation

Prediction of execution time, energy, bandwidth

Adaptation control algorithm

Architecture and design of adaptive hardware

Cross-layer integration

Issues in Hardware Adaptation

Prediction of execution time, energy, bandwidth [ISCA'01]

Adaptation control algorithm

Architecture and design of adaptive hardware

$\left.\begin{array}{l} \\ \\ \end{array}\right\}$ [MICRO'01], [under review]

Cross-layer integration

Outline

Predictability of Execution Time

Adaptive Hardware to Save Energy

Summary and Ongoing Work

Predictability - Motivation

Many multimedia applications real-time

- Must process *frame* of data within a deadline
- ⇒ Need predictability

Common conjecture for general-purpose processors (GPPs)

- *Complex features make GPPs unpredictable (???)*
- E.g., out-of-order issue, caches, speculation

Predictability - Motivation

Many multimedia applications real-time

- Must process *frame* of data within a deadline
- ⇒ Need predictability

Common conjecture for general-purpose processors (GPPs)

- *Complex features make GPPs unpredictable (???)*
- E.g., out-of-order issue, caches, speculation

We used variability at frame granularity to quantify predictability

Workload and Simulated Architecture

Workload

Speech:	GSMenc, GSMdec (low bit rate)
	G728enc, G728dec (high bit rate)
Video:	H263enc, H263dec (low bit rate)
	MPEGenc, MPEGdec (high bit rate)
Audio:	MP3dec

1GHz out-of-order processor simulated with RSIM

- 4-issue, 64 entry instruction window
- 64KB L1 data (2 cycles), 1MB L2 data (20 cycles)
- 102 cycles main memory
- 2 ALU, 2 FPU, 2 Address generation

Key Findings for Predictability

Complex features make GPPs unpredictable ???

Key Findings for Predictability

~~*Complex features make GPPs unpredictable (???)*~~

Some apps show high variability in execution time of a frame

- BUT architecture introduces little of it
- Most variability from algorithm, input

Key Findings for Predictability

~~*Complex features make GPPs unpredictable (???)*~~

Some apps show high variability in execution time of a frame

BUT architecture introduces little of it: IPC almost constant

Most variability from algorithm, input: Instruction count varies

$\Rightarrow$ *Amount of work changes, nature of work stays same*

Key Findings for Predictability

~~*Complex features make GPPs unpredictable (???)*~~

Some apps show high variability in execution time of a frame

BUT architecture introduces little of it: IPC almost constant

Most variability from algorithm, input: Instruction count varies

$\Rightarrow$ *Amount of work changes, nature of work stays same*

Other useful results

- Little time in memory stalls
- Instruction count changes slowly

Motivates algorithm to control when and what to adapt

Next….

Outline

Adaptive Hardware – Motivation and Goals

Many proposals for adaptive hardware to save energy

- **D**ynamic **v**oltage and frequency **s**caling (**DVS**)
- Architecture adaptation
  - Instruction window size, functional units, issue width, …

Two key questions

$$\left.\begin{array}{l}\Rightarrow \text{ When to adapt?} \\ \Rightarrow \text{ What to adapt?}\end{array}\right\} \text{ Adaptation control algorithm}$$

Our goal

- Adaptation control algorithm for multimedia applications
- Study DVS vs. architecture adaptation

Inter-Frame Control Algorithm – Key Ideas

Use results from study of execution time predictability

$\Rightarrow$ When to adapt?

Execution time variability at frame level

$\Rightarrow$ Adaptation at frame granularity

(2) What to adapt?

Predict time, energy of next frame for *all configurations*

Pick lowest energy configuration that can meet deadline

Execution Time Prediction for a Frame

$$\text{Execution cycles} = \frac{1}{\text{IPC}} \times \text{Instruction count}$$

(IPC = instructions per cycle)

Execution Time Prediction for a Frame

$$\text{Execution cycles} = \frac{1}{\text{IPC}} \times \text{Instruction count}$$

IPC constant $\Rightarrow$

Get by profiling initial frame

Memory time small $\Rightarrow$

Profile only one frequency

Execution Time Prediction for a Frame

$$\text{Execution cycles} = \frac{1}{\text{IPC}} \times \text{Instruction count}$$

IPC constant ⇒

Get by profiling initial frame

Memory time small ⇒

Profile only one frequency

DIC changes smoothly ⇒

Can use simple predictor

One prediction for all hardware

Execution Time Prediction for a Frame

$$\text{Execution cycles} = \frac{1}{\text{IPC}} \times \text{Instruction count}$$

IPC constant ⇒

Get by profiling initial frame

Memory time small ⇒

Profile only one frequency

DIC changes smoothly ⇒

Can use simple predictor

One prediction for all hardware

⇒ Frame execution time dynamically predictable

Dynamic predictor needed only for frame *instruction count*

Execution Time Prediction for a Frame

$$\text{Execution cycles} = \frac{1}{\text{IPC}} \times \text{Instruction count}$$

IPC constant ⇒

Get by profiling initial frame

Memory time small ⇒

Profile only one frequency

DIC changes smoothly ⇒

Can use simple predictor

One prediction for all hardware

⇒ Frame execution time dynamically predictable

Dynamic predictor needed only for frame *instruction count*

Energy prediction analogous

Inter-Frame Adaptation Control Algorithm

Profiling Phase

For each hardware, H

$Imax_H$ = Maximum instructions H can execute in deadline

EPI_H = **E**nergy **p**er **I**nstruction

↓

Adaptation Phase

Predict instruction count for next frame

Choose hardware with Imax ≥ prediction and least EPI

Inter-Frame Adaptation Control Algorithm

Profiling Phase

For each architecture, A, measure IPC_A at one voltage/freq

For each hardware, H, with architecture A

$Imax_H$ = Maximum instructions H can execute in deadline

$= Deadline \times Frequency_H \times IPC_A$

EPI_H = Energy per Instruction

Adaptation Phase

Predict instruction count for next frame

Choose hardware with Imax $\geq$ prediction and least EPI

Inter-Frame Adaptation Control Algorithm

Profiling Phase

For each arch, A, measure IPC_A and $Power_A$ at one V,f

For each hardware, H, with architecture A

$Imax_H$ = Deadline $\times$ $Frequency_H$ $\times$ IPC_A

EPI_H = Energy per Instruction $\propto$ $Power_A\ V_H^2/IPC_A$

Adaptation Phase

Predict instruction count for next frame

Choose hardware with Imax $\geq$ prediction and least EPI

Inter-Frame Adaptation Control Algorithm

Profiling Phase

For each arch, A, measure IPC_A and $Power_A$ at one V,f

For each hardware, H, with architecture A

$$Imax_H = Deadline \times Frequency_H \times IPC_A$$

$$EPI_H = \text{Energy per Instruction} \propto Power_A\, V_H^2 / IPC_A$$

Adaptation Phase

Predict instructions: Max of past 5 frames, leeway, hysteresis

Choose hardware with Imax $\geq$ prediction and least EPI

Inter-Frame Adaptation Control Algorithm

Profiling Phase

For each arch, A, measure IPC_A and $Power_A$ at one V,f

For each hardware, H, with architecture A

$Imax_H = Deadline \times Frequency_H \times IPC_A$

$EPI_H = \text{Energy per Instruction} \propto Power_A \, V_H^2 / IPC_A$

Order hardware by increasing EPI in EPI-Imax table

Adaptation Phase

Predict instructions: Max of past 5 frames, leeway, hysteresis

Choose hardware with Imax $\geq$ prediction and least EPI

= First hardware in EPI-Imax table with Imax $\geq$ prediction

Inter-Frame Adaptation Control Algorithm

Profiling Phase

For each architecture, A, measure IPC_A and $Power_A$ at one voltage/freq

For each hardware, H, with architecture A

$Imax_H$ = Deadline $\times$ $Frequency_H$ $\times$ IPC_A

EPI_H = Energy per Instruction $\propto$ $Power_A\ V_H^2/IPC_A$

Order hardware by increasing EPI in EPI-Imax table

$\downarrow$

Adaptation Phase

Predict instructions: Max of past 5 frames, leeway, hysteresis

Choose first h/w in EPI-Imax table with Imax $\geq$ prediction

Dynamic prediction only needed for frame instruction count

Independent of hardware configuration

Modifications for Continuous DVS

At least one processor has continuous DVS (CDVS)

$\Rightarrow$ EPI-Imax table too long

With CDVS, same architecture has least EPI for most cases

$$\downarrow$$

Architecture with least $\text{Power}_A / \text{IPC}_A^3$

- Find this architecture in profiling phase
- Pick only frequency in adaptation phase

$\Rightarrow$ No large EPI-Imax tables needed

Experimental Methodology

Workload same as for predictability study

RSIM + Wattch for time and energy simulations

- Aggressive clock gating

Experimental Methodology (Cont.)

Processors evaluated

- NoAdapt, Arch, CDVS, Arch+CDVS

Base hardware similar to predictability study except

- Processor ~ 2X as aggressive
  - 8-way, 128 entry inst. window, 6 ALU, 4 FPU, 2 addr gen.

Architecture adaptations

- Instruction window size: 128, 96, 64, 48, 32, 16
- Active ALUs: 6, 4, 2
- Active FPUs: 4, 2, 1 $\Rightarrow$ 54 configurations

DVS frequency: 100 MHz to 1GHz

How Good is the Inter-Frame Algorithm?

Missed deadlines

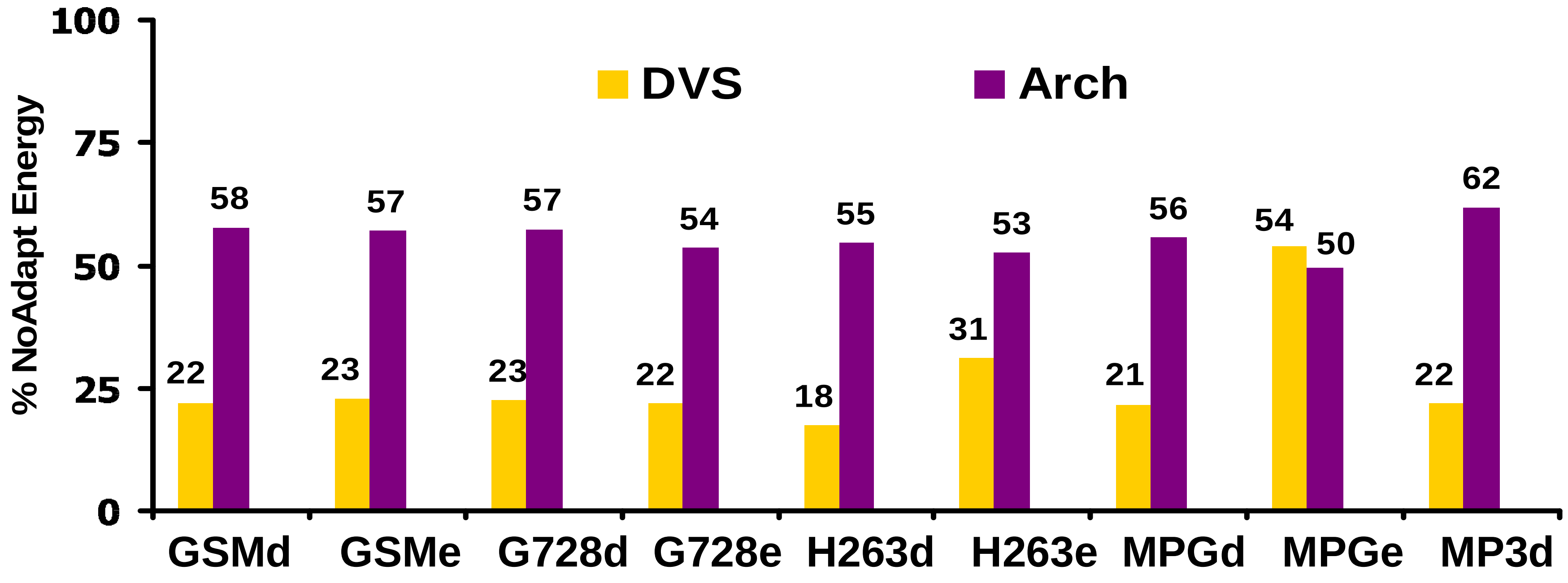
- For all deadlines and processors, very few deadlines missed
  - Average across all apps ≤ 0.8%
  - Maximum for a single app ≤ 3.6%

Slack removed

- Slack = Idle time between end of processing until deadline
- Most slack removed
- Remaining slack mostly from system limitations

Energy savings – *Next…*

Energy Savings From Inter-Frame Adaptation



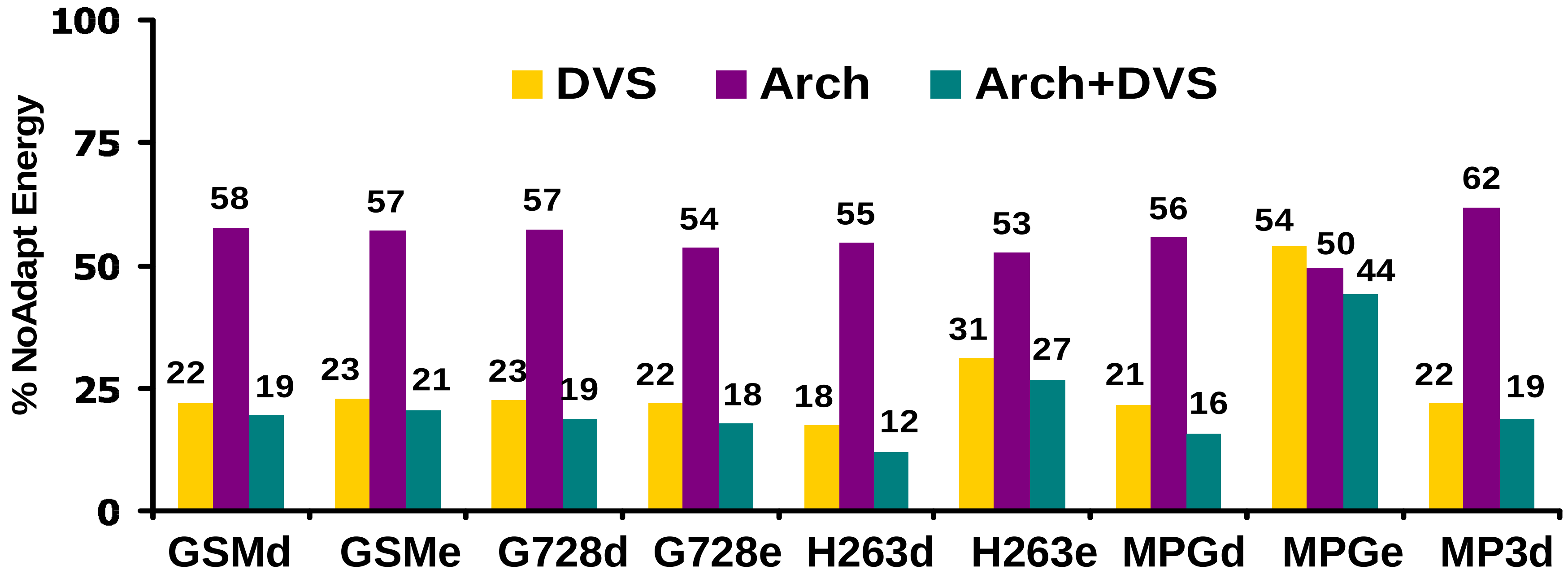
DVS very effective

46% to 82% savings, average 74%

Architecture adaptation effective, but much less than DVS

38% to 50% savings, average 44%

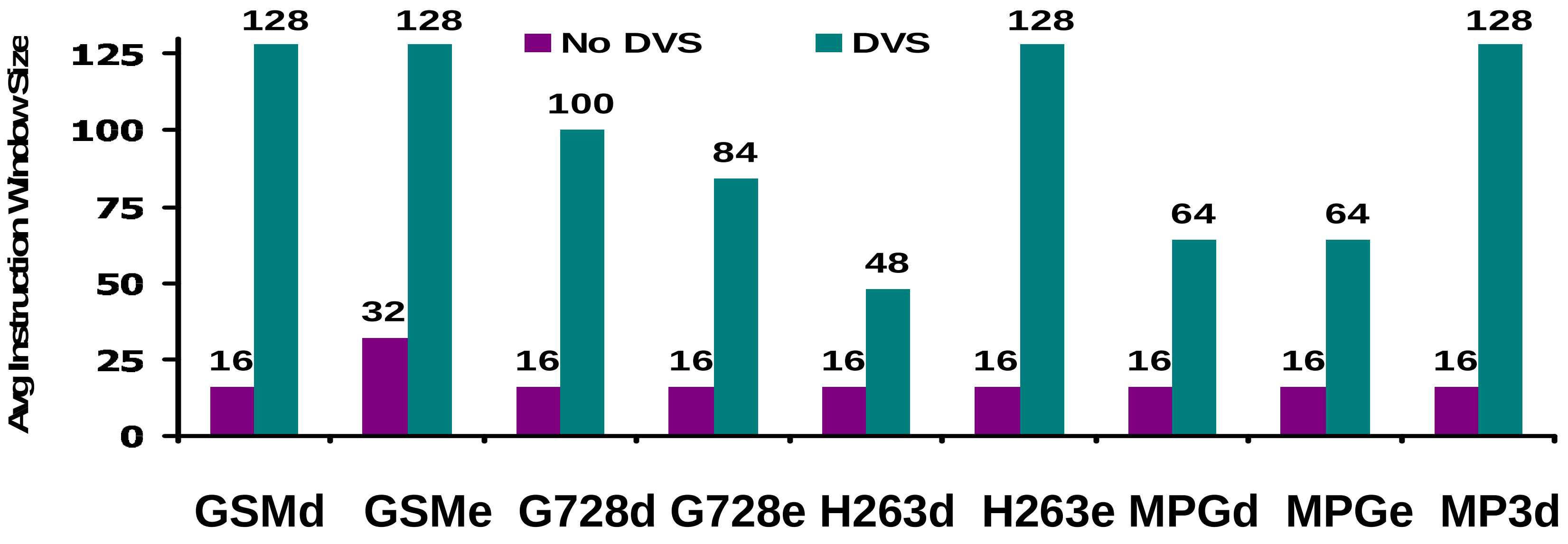
Inter-Frame DVS+Arch vs. DVS alone



DVS + Arch most energy efficient, but most benefit from DVS

Savings vs. DVS alone: 10% to 32%, average 18%

Instruction Window Utilization for Arch

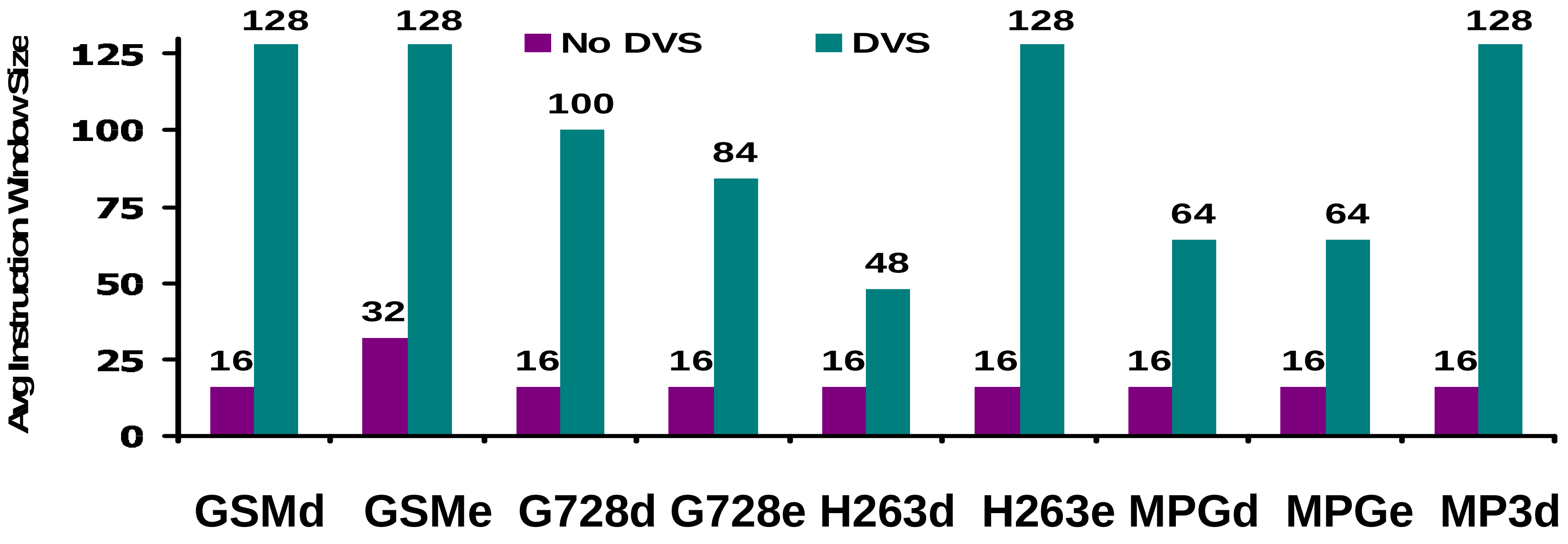


Most energy efficient architecture depends on presence of DVS

Without DVS, simple configurations (low IPC) chosen

With DVS, more aggressive configurations (high IPC) chosen

Instruction Window Utilization for Arch



Most energy efficient architecture depends on presence of DVS

- Without DVS, simple configurations (low IPC) chosen
- With DVS, more aggressive configurations (high IPC) chosen
  - High IPC allows running at low frequency

DVS + Architecture Adaptation

When is it effective to have architecture adaptation with DVS?

- Application has lower $\text{Power}_A / \text{IPC}_A^3$ for alternate hardware
- Application has slack at lowest frequency
- Optimal frequency not supported by D-DVS

DVS + Architecture Adaptation

When is it effective to have architecture adaptation with DVS?

- Application has lower $\text{Power}_A / \text{IPC}_A^3$ for alternate hardware
- Application has slack at lowest frequency
- Optimal frequency not supported by D-DVS

But so far assume same hardware for full frame

- What about intra-frame adaptation?

Outline

Intra-Frame Adaptation

IPC and resource usage often varies within a frame

$\Rightarrow$ Int***ra***-frame architecture adaptation

Adapt without slowing down (ideally)

(No intra-frame DVS due to high overhead)

Similar to architecture adaptation work for conventional apps

But previous work does not consider

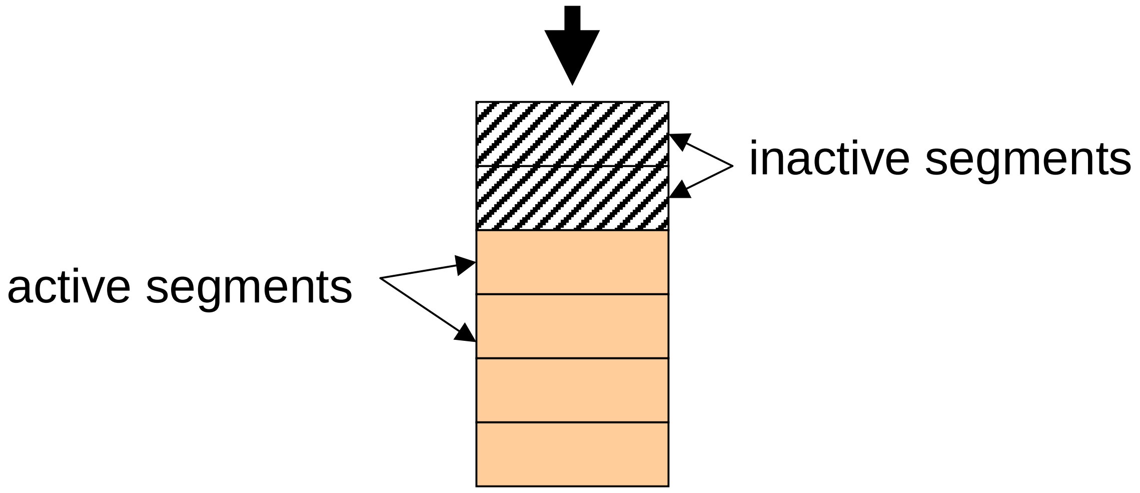
- Real-time multimedia applications
- Interaction with inter-frame adaptation

⇨

Instruction Window Adaptation – Key Idea

Divide instruction window into equal segments

- To save energy, deactivate active segments
- To avoid IPC loss, activate inactive segments



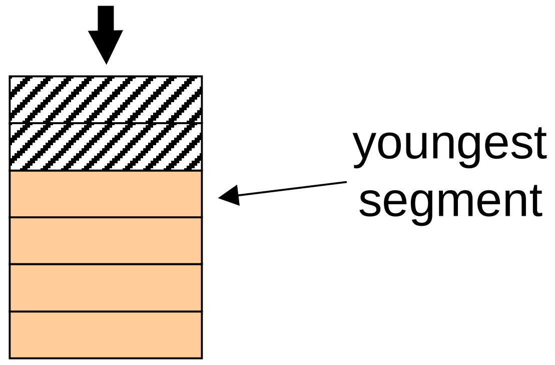
Control algorithm must determine

- When and how much to deactivate?
- When and how much to activate?

State-of-the-art Algorithm [Folegnani et al.]

Deactivates when # of issues from youngest segment are low

- + Deactivates segments that do not contribute to IPC



Activates periodically

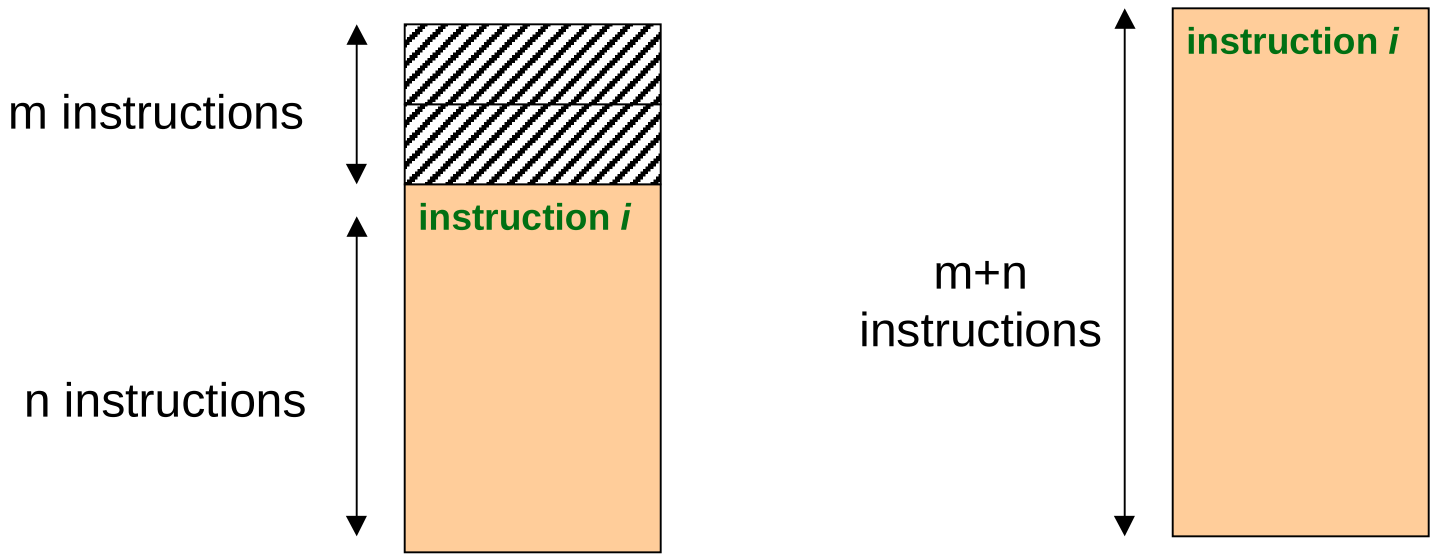
- – Can degrade energy due to unnecessary powering up
- – Can degrade IPC due to delays in powering up

Can we do better?

New Algorithm for Activation

Activate when reduced instruction window causes stalls

- Track stalls at retirement
- Instruction window reduces stalls by providing overlap



Instruction i lost m instructions worth of overlap

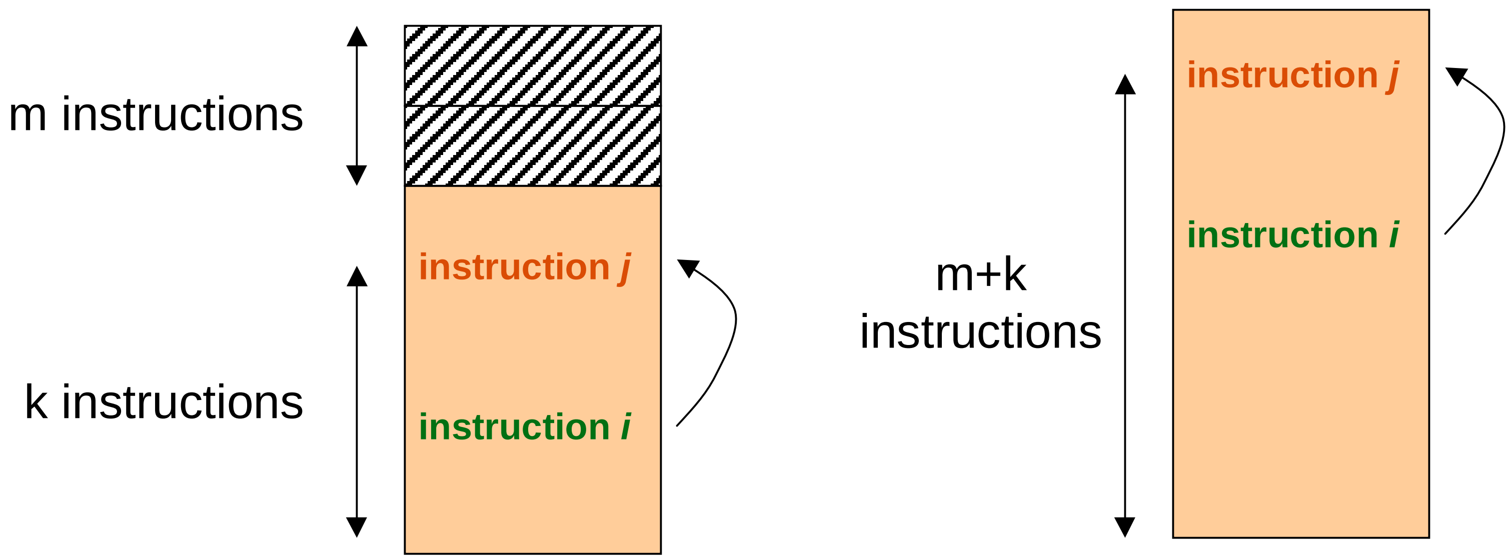
⇒ Tag instruction *i* with m

New Algorithm for Activation

Activate when reduced instruction window causes stalls

Track stalls at retirement

Instruction window reduces stalls by providing overlap



Instruction *j* lost m instructions worth of overlap

$\Rightarrow$ Pass tag m of instruction *i* to instruction *j*

New Algorithm for Activation (Cont.)

On entry

- Set tag for each instruction whose operands are available
- *Tag = # entries powered down*

On completion

- Reduce tag if instruction was stalled (at retirement)
- Pass tag to any instruction that consumes the result

On retirement

- If an instruction is tagged
  - Increment a counter by min(# of stall cycles, tag)
  - Increase the window size if counter exceeds a threshold

New vs. State-of-the-Art Activation Algorithm

State-of-the-art

– Can degrade energy due to unnecessary powering up

– Can degrade IPC due to delays in powering up

New algorithm

+ Activates only when there is IPC loss due to reduced window

– Overhead due to tags

Instruction Window Algorithms Studied

State-of-the-art [Folegnani et al.]

- Issue based deactivation
- Periodic activation

New Algorithm

- Issue based deactivation
- Stall based activation

Thresholds set for max average IPC degradation of 4%

Instruction Window Algorithms Studied

State-of-the-art [Folegnani et al.]

Issue based deactivation

Periodic activation

New Algorithm

Issue based deactivation

Stall based activation

Thresholds set for max average IPC degradation of 4%

New algorithm always same or slightly better

Use in rest of the talk

Functional Unit Adaptation

Can activate/deactivate each ALU or FPU

- To save energy, deactivate active units
- To avoid IPC loss, activate inactive units

Issue width proportional to # of active units

Control algorithm must determine

- When and how many to deactivate?
- When and how many to activate?

State-of-the-art Algorithm [Maro et al.]

Deactivates when utilization of a unit is low

Activates when utilization of remaining units is high

- Does not necessarily mean that more units are needed

Can we do better?

New Algorithm for Activation

Activate when # of structural hazards for a unit type is high

- + Only activates when more units are needed
- – Non-critical instructions can still activate

Functional Unit Adaptation Algorithms Studied

State-of-the-art [Maro et al.]

- Utilization based deactivation
- Utilization based activation

New Algorithm

- Utilization based deactivation
- Hazard based activation

Thresholds set for max average IPC degradation of 3%

Functional Unit Adaptation Algorithms Studied

State-of-the-art [Maro et al.]

- Utilization based deactivation
- Utilization based activation

New Algorithm

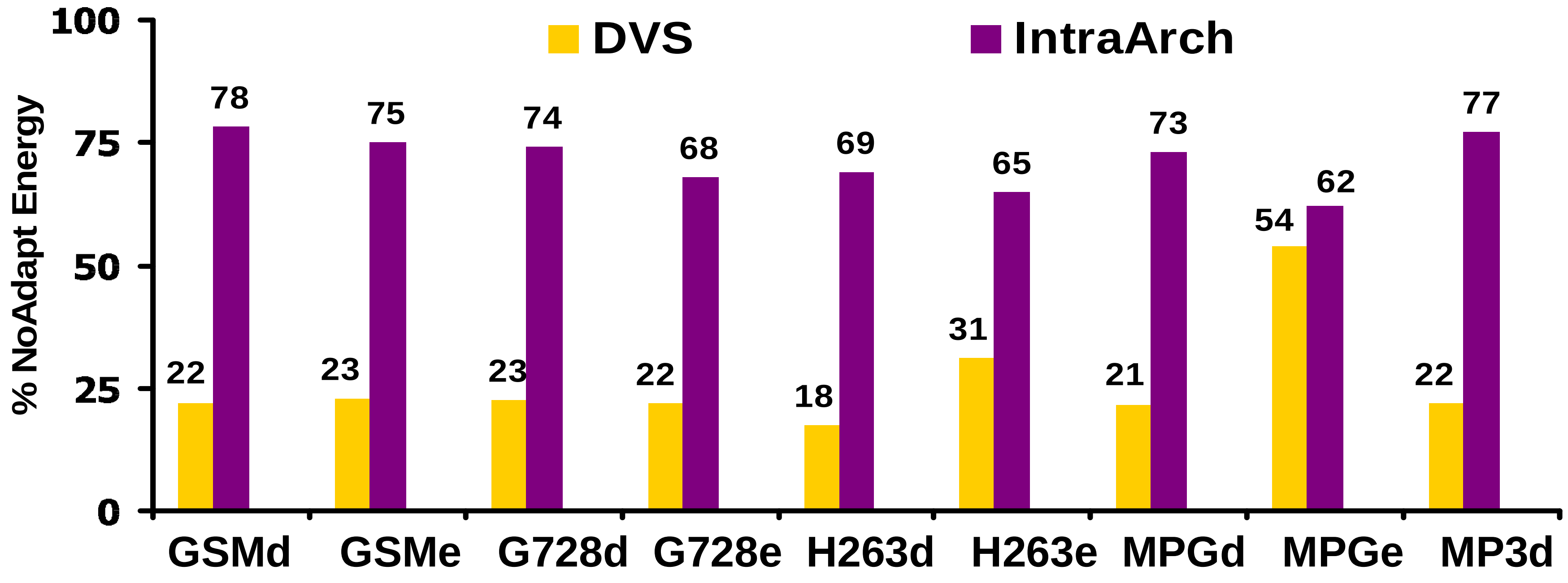
- Utilization based deactivation
- Hazard based activation

Thresholds set for max average IPC degradation of 3%

New algorithm always same or slightly better

Use in rest of the talk

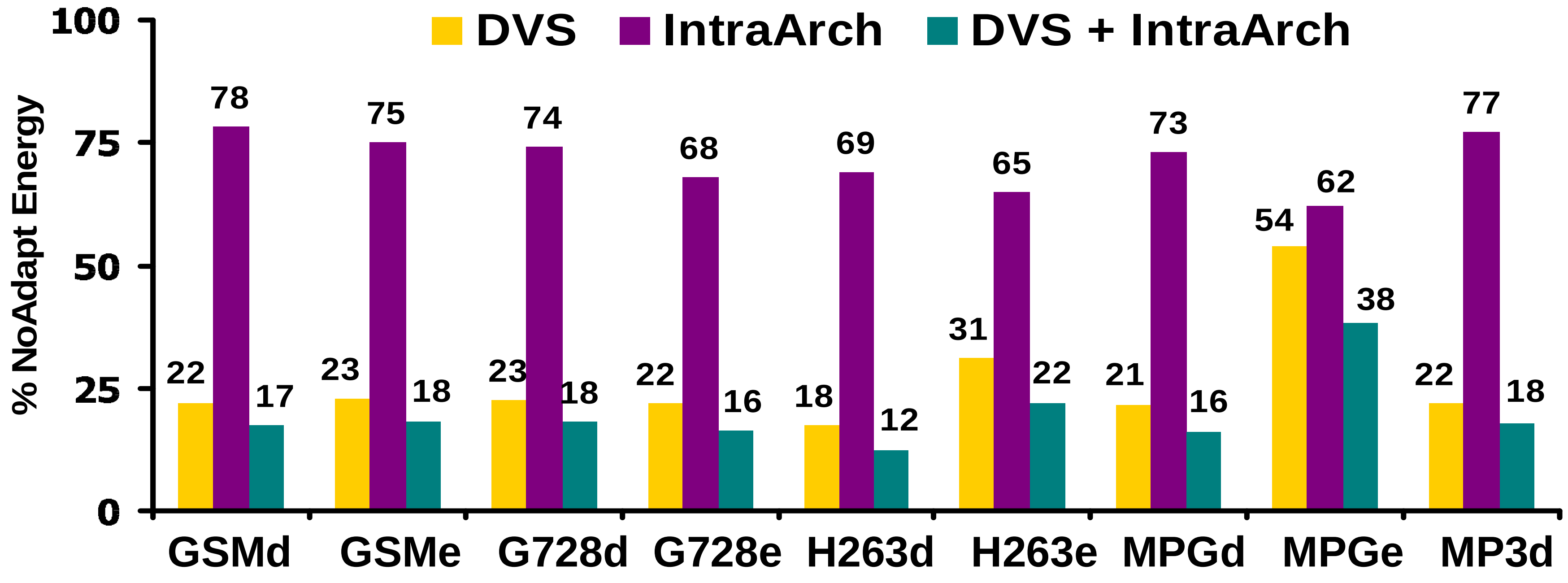
Savings with Intra-Frame Arch Adaptation



Intra-frame architecture adaptation effective, but less than DVS

22% to 38% savings, average 29%

Intra-Frame DVS+Arch vs. DVS alone



Intra-frame architecture adaptation effective, but less than DVS

- 22% to 38% savings, average 29%

DVS + IntraArch most energy efficient

- Savings vs. DVS alone: 18% to 30%, average 24%

Outline

Predictability of Execution Time

Adaptive Hardware to Save Energy

- Motivation and Goals
- Inter-Frame Adaptation
- Intra-Frame Architecture Adaptation
- Inter- vs. Intra- Frame and Combination

Summary and Ongoing Work

Inter vs. Intra Frame Adaptation

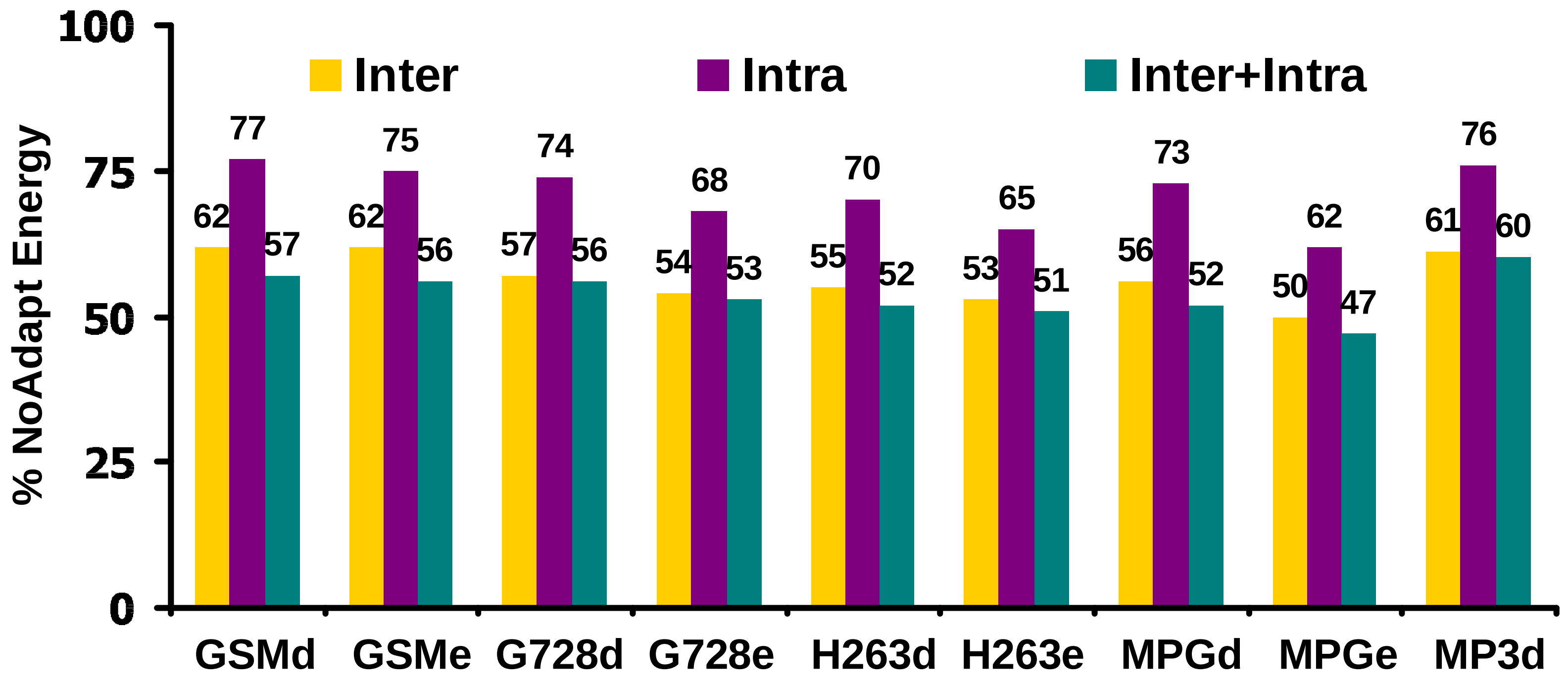
Properties	**Inter-frame**	**Intra-frame**
Basis of adaptation	Profiles mostly at start	Continuous monitoring
Features controlled	Global configuration	Individual features
Impact on exec time	May increase	No impact (ideally)
Best for which adaptations?	With high overhead	Those not applicable to full frame

Inter-Frame + Intra-Frame Adaptation

Same as inter-frame algorithm except

- Run profiling phase with intra-frame adaptations
- Run adaptation phase with intra-frame adaptations

Inter, Intra, Inter+Intra without DVS



Inter better than Intra for all applications

Inter can exploit slack

Inter + Intra always best, but most savings from Inter

Savings vs. Base: 40% to 53%, average 46%

Why is Inter Better than Intra with no DVS?

When applications have a lot of slack

- Inter increases execution time to save energy
  - Selects low IPC (low energy) architecture configurations
- Intra must maintain execution time
  - Selects more aggressive architecture configurations
- $\Rightarrow$ Inter better with lot of slack

When applications have little slack

- Inter cannot do much
- $\Rightarrow$ Intra better with little slack
- Similar to case with DVS described next

Conclusions for No DVS

Lot of slack ⇒ Inter better

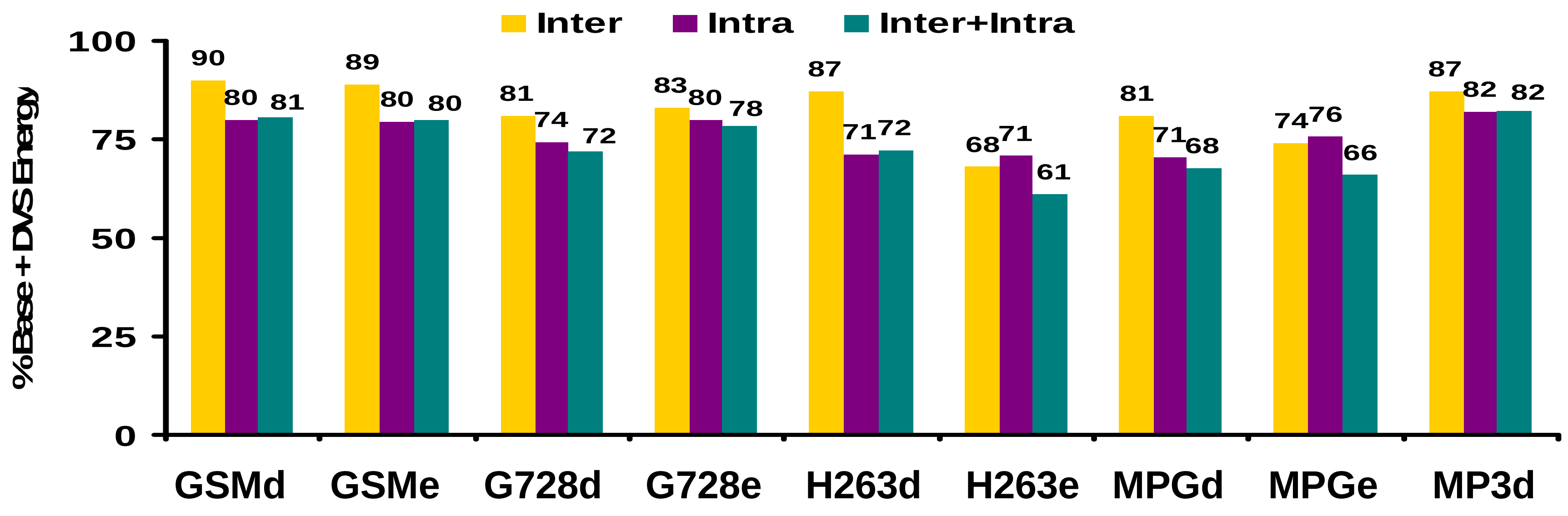
Little slack ⇒ Intra better

Inter + Intra best in all cases

Application slack unknown a priori

⇒ Inter+Intra best choice without DVS

Inter, Intra, Inter+Intra *with DVS*



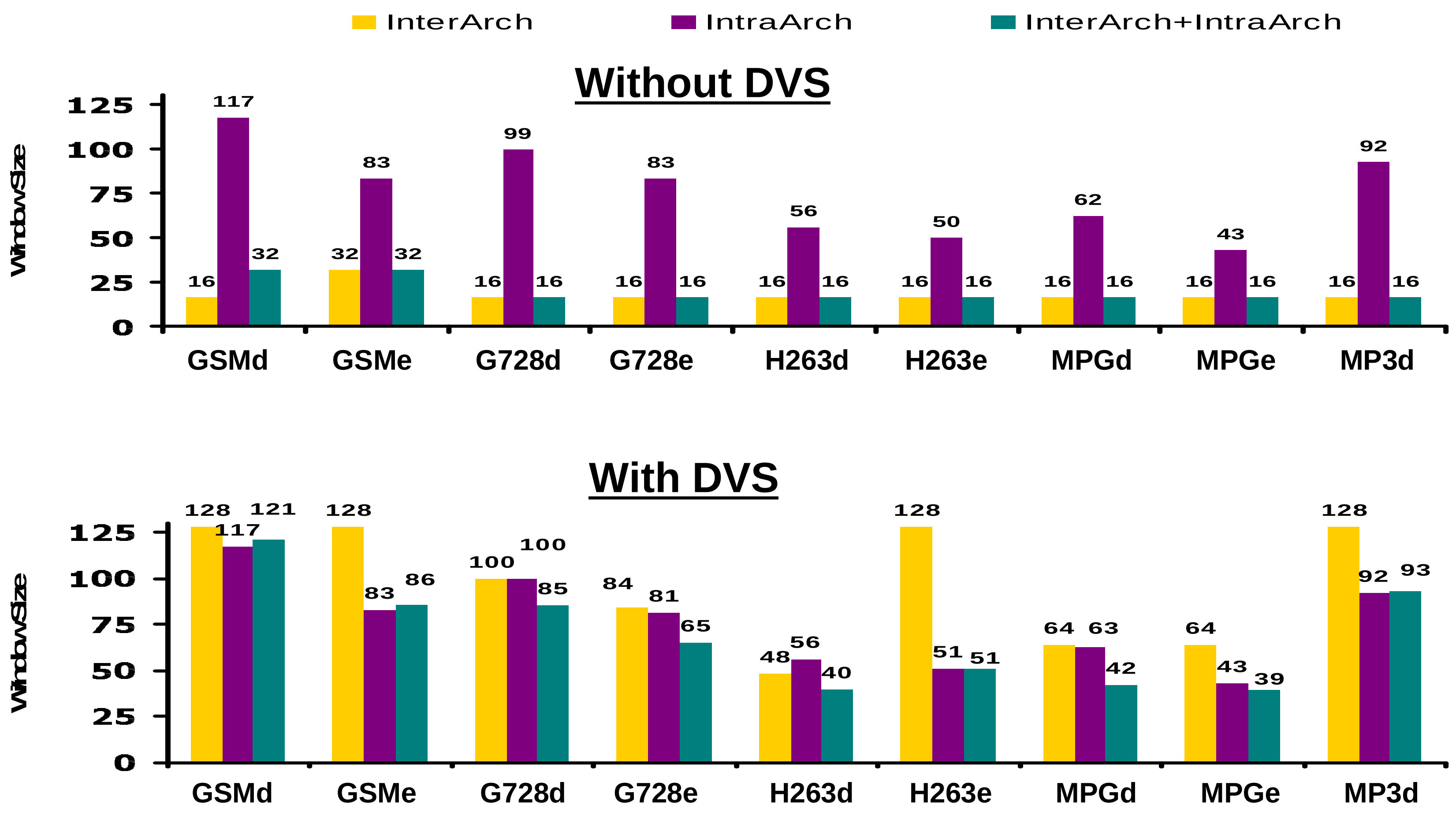
Intra better than Inter for most applications

- With DVS, Intra exploits slack as well as Inter!

Inter + Intra always best

- Intra gives most savings, but adding Inter costs little
- Savings vs. DVS alone: 18% to 39%, average 27%

Instruction Window Utilization w/o and w/ DVS



With DVS, more aggressive configurations chosen

Conclusions - Inter vs. Intra-Frame Adaptation

Inter+Intra frame architecture adaptation is best

- Without DVS
  - Lot of slack ⇒ Inter better
  - Little slack ⇒ Intra better
  - Inter + Intra best in all cases
- With DVS
  - Most of the savings come from Intra
  - But adding Inter costs little
- Average savings 46% without DVS, 26% with DVS

Summary and Conclusions (1 of 2)

Execution Time Predictability for Soft Real-Time

~~Conventional wisdom:~~

~~*Complex architecture features induce unpredictability (??)*~~

Variability from algorithm + input for *all* architectures

Findings motivate inter-frame adaptation control algorithm

Hardware Adaptation to Save Energy

Next slide…

Summary and Conclusions (2 of 2)

Hardware Adaptation to Save Energy

Inter-, Intra-, Inter+Intra- Frame adaptation control algorithms

- DVS + Architecture adaptation is best
- Inter-frame + Intra-frame architecture adaptation is best

Best architecture configuration depends on DVS

- No DVS $\Rightarrow$ simple architectures
- With DVS $\Rightarrow$ aggressive architectures

Design aggressive architectures at low frequency

Ongoing Work

- Hardware adaptation techniques
- Integration of hardware adaptation with other layers
- Adapting for thermal power
- Exploiting output quality flexibility and loss resilience
- Enhancing predictability of multithreading (SMT) for real-time

For more information

http://www.cs.uiuc.edu/~sadve

sadve@cs.uiuc.edu