Exploring Heterogeneity within a Core for Improved Power Efficiency

Sudarshan Srinivasan, Nithesh Kurella

Israel Koren, Sandip Kundu

Outline

- Asymmetric Multicores
  - Asymmetric multicore processors (AMPs) consist of cores with the same instruction-set architecture
  - Different microarchitectural features, speed, and power consumption
    1. How closely can we match the core(s) to current computational needs?
    2. How quickly can we match the thread to the best core to run on?

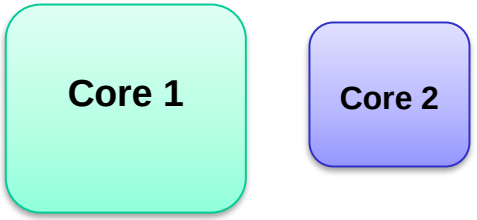
- Self-morphing – core adapts faster to application demands
  - Still need to architect core mode/type
  - Determine the rules for morphing as the computing needs change
  - How often?

- Experimental results
  - Quantitative evaluation of the benefits of the approach

Asymmetric Multicore Processors (AMPs)

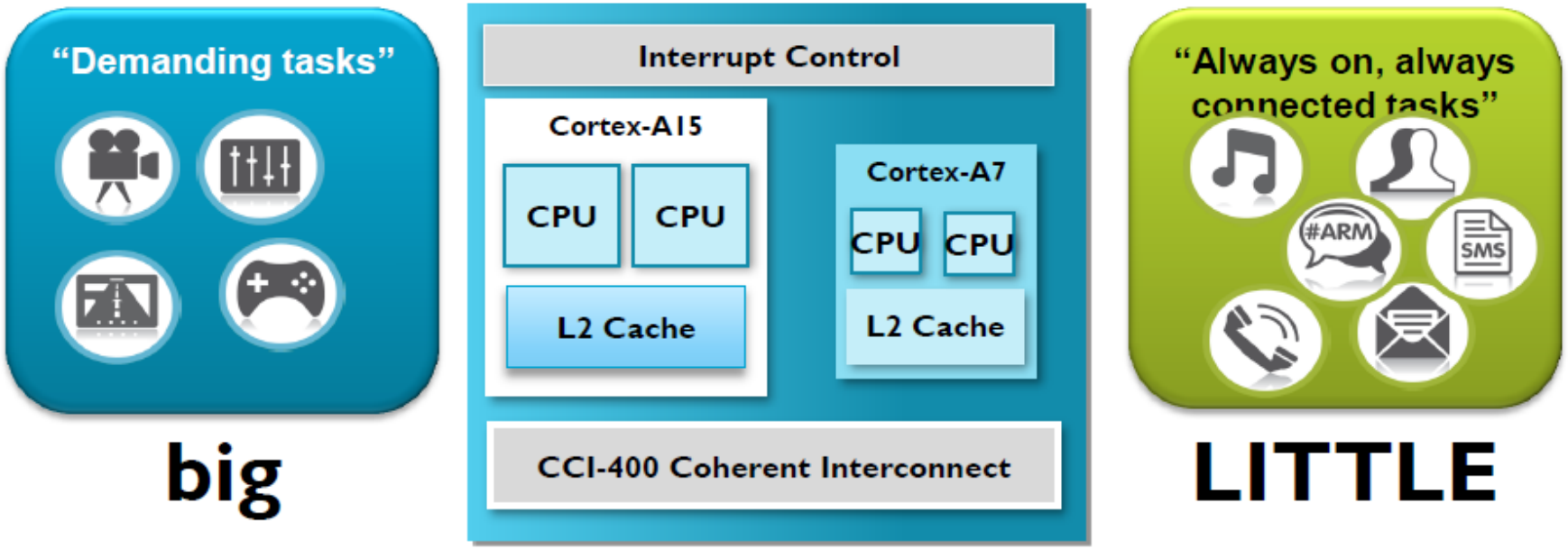
- Cores of different capabilities in the same chip
  - Such cores have different performance and power characteristics

- Typically consists of
  - Out-of-order (OOO) cores
    High performance
  - In-Order (InO) cores
    Low power



Asymmetric multicore

Commercial ARM Big/Little Architecture



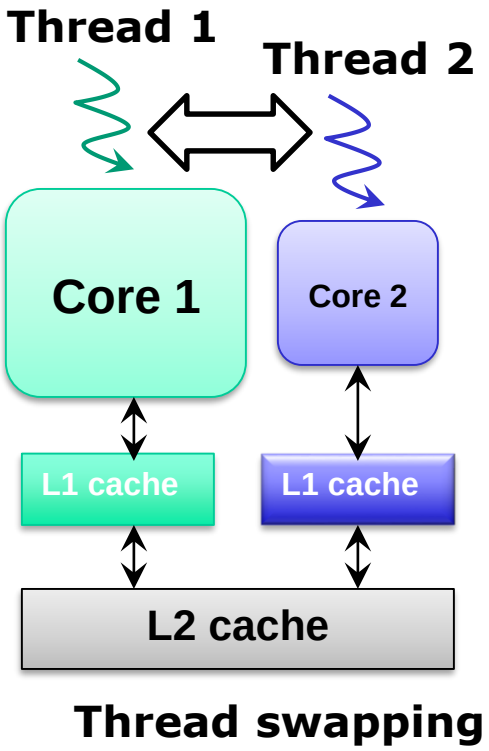
- Use the right processor for the right task

Source: John Goodacre, "Homogeneity of architecture in Heterogeneous world"

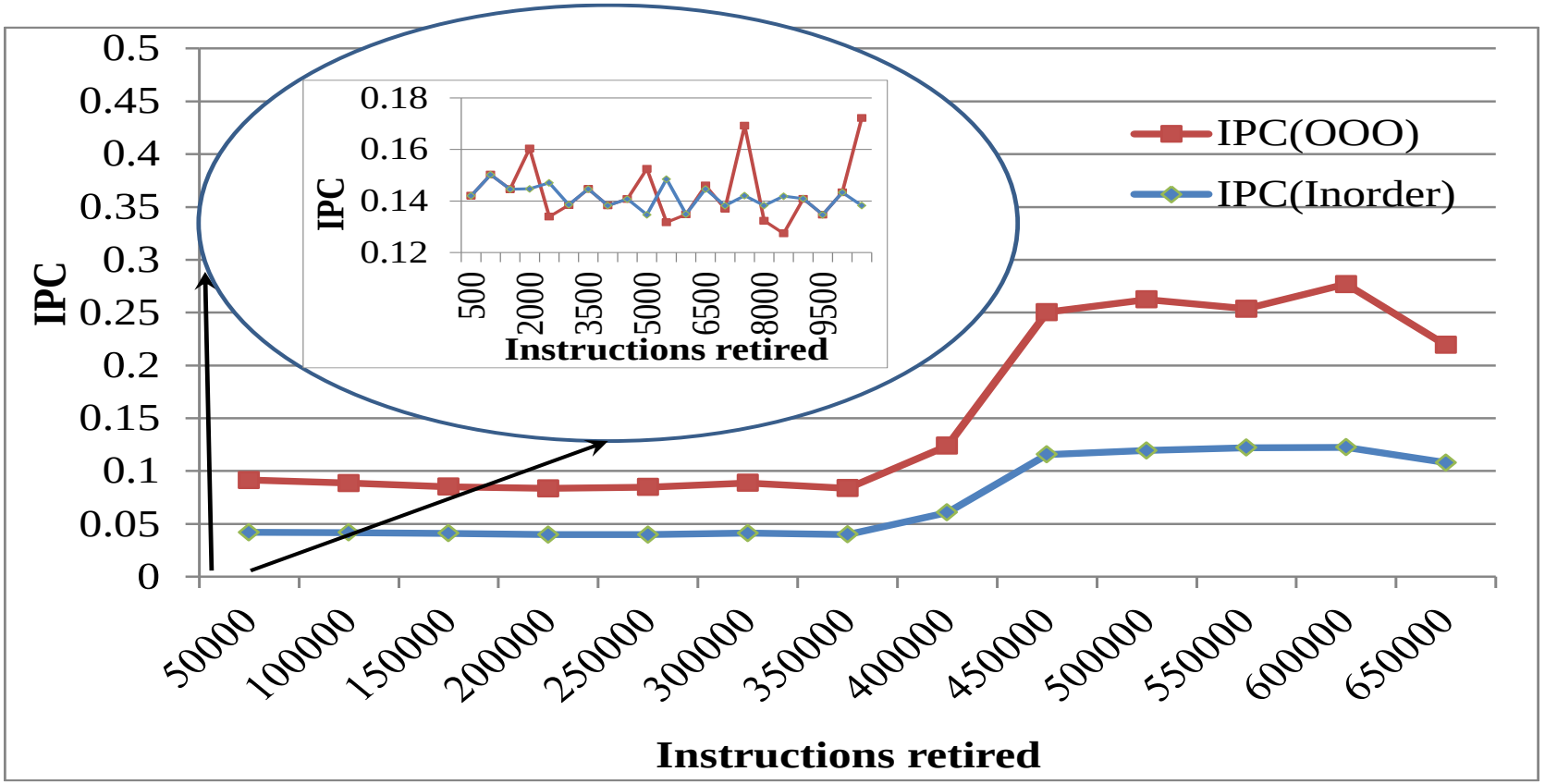
Limitations of current AMP Architectures

1. Limited architectural flexibility
   - Limited choices of core capabilities
   - Fixed number of large and small cores

2. Limited thread to core mapping flexibility
   - Applications have phases with different computational requirements
   - Swapping threads between cores can reduce the power consumed, but
   - Task migration has a high overhead (need to transfer thread state/data)
   - Thread migration/swap at granularity of millions of instructions (missed opportunities)



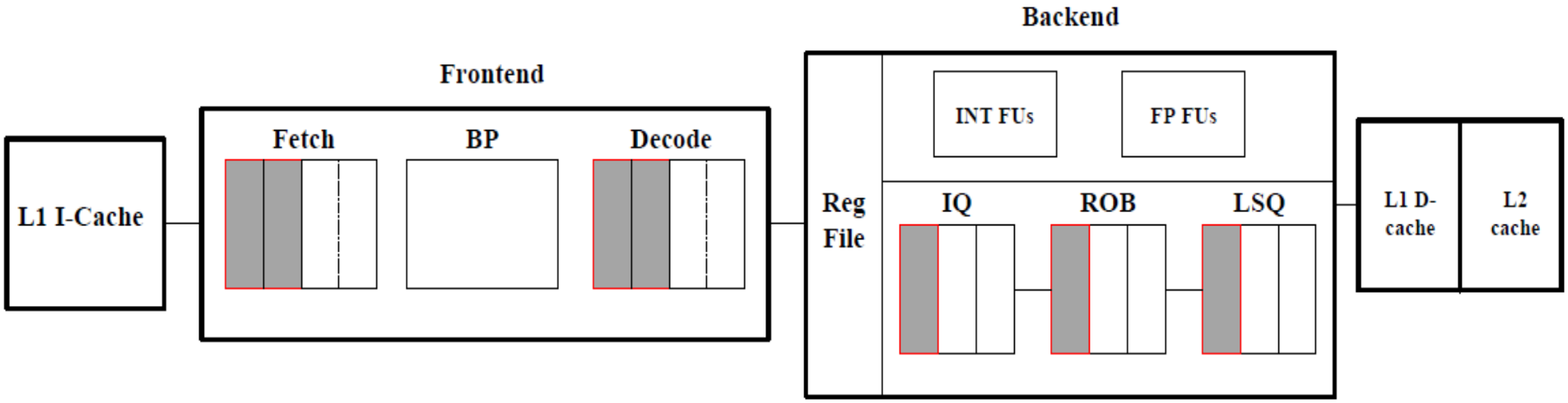
Can fine-grain task migration be beneficial?



Can we exploit Fine-Grain Changes?

- Take advantage of fine grain adaptation to improve power efficiency without high migration overhead

- **Self-morphing core**: morphs into multiple architecture types (core modes) with varying execution width and resource sizes.
- Significantly lower thread migration overhead:
  - Critical units (register file, caches and branch predictor) are used by all core modes



Morphable Architectures

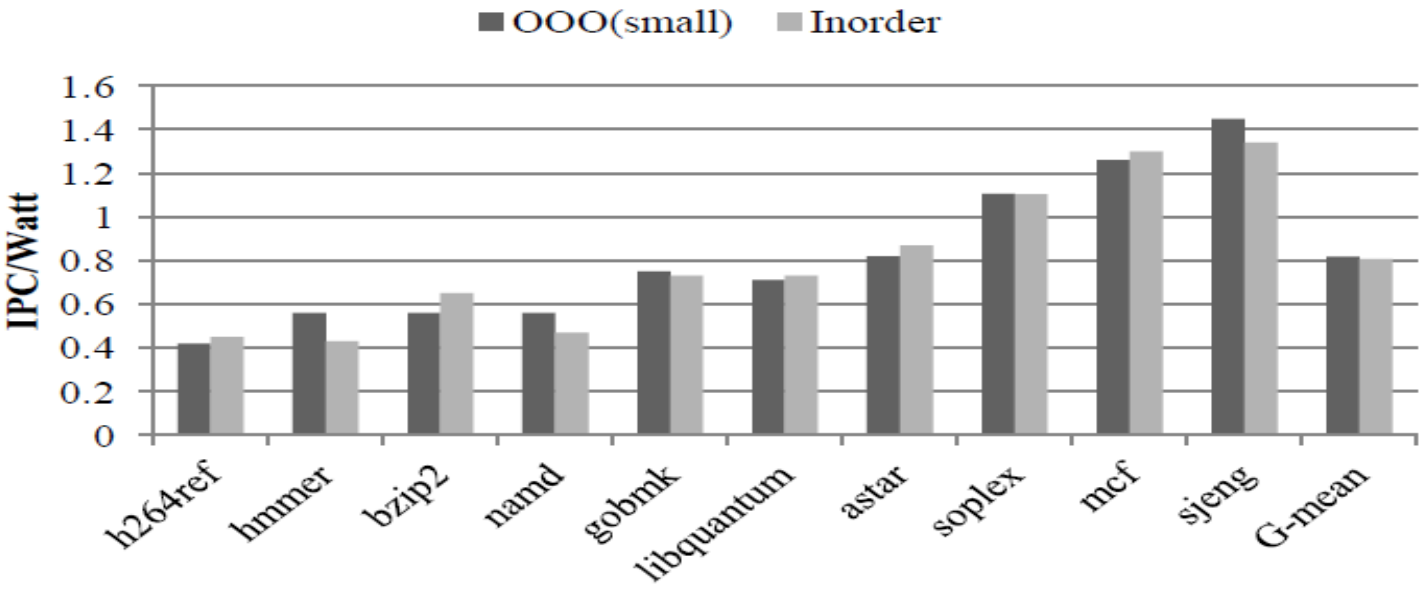
- A Morphable architecture where OOO core turns into InO was proposed by [Lukefahr *et al*., Micro 2012]
  - InO has much lower power consumption, but
  - Turning OOO core into InO in run time involves significant micro-architecture changes
  - These result higher design cost and verification

- Questions to be investigated:
  1. Is an InO mode necessary, as its inclusion complicates the design?
  2. Are two architecture modes (core types) sufficient to match the large variance in application needs?

Is InO mode necessary?

- InO core has smaller cache and array structures
  - Cache/Array leakage is no longer a problem as tri-gates cut leakage by 10X at 22nm
- Use instead a small OOO
  - Fetch, issue width of 1 and smaller ROB, LSQ and IQ
- For most benchmarks IPC/Watt of InO and small OOO are comparable

Simulation with MCPAT 22nm double gate models



Designing a Self-Morphing Core

- **Goal**: Design a core than can morph into various OOO modes with varying execution width and resource sizes

- **Questions:**
  - How many core modes should we have?
  - What should be the architectural parameters of these modes?
  - How fine-grained should mode switches be?
  - When to switch from one mode to another?
  - How much power savings can we get?

Core Design Space Exploration

- Find core types that would provide best performance/watt at fine-grain instruction granularities

Table 1: Core Design Parameters.

CoreParameter	Range of Values
Fetch, Issue Width	1,2,3,4,5
ROB size	8,16,32,64,96,128,192,256,384
Issue Queue size	12,24,36,48,64
LSQ Size	8,16,32,64,96,128,192
Clock Period	0.4ns-1ns(steps of 0.1ns)

- Initial design combinations had 2000 - pruned to 300

- Pruning accomplished by grouping processor structures which could achieve greater IPC/watt than performing independent structure resizing

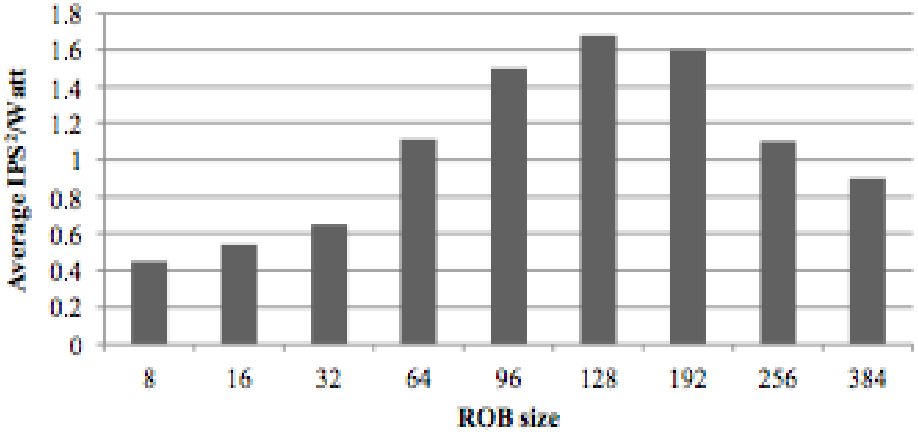
Number and Types of Cores

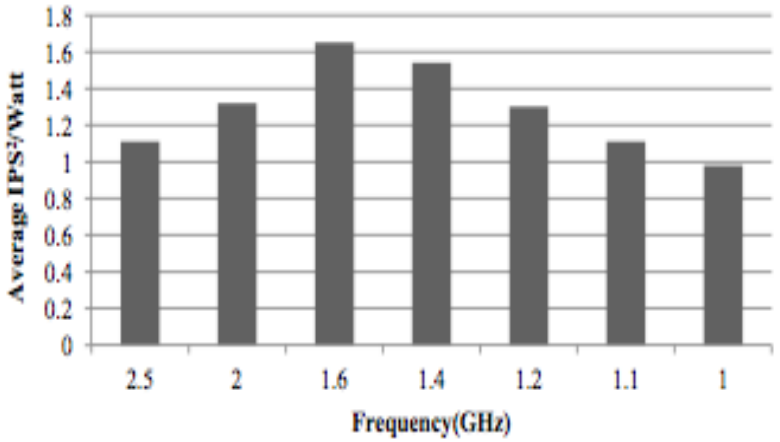
- **Objective**: achieve the highest possible IPS^2/Watt by allowing switching between core types at ~2K instruction granularity
  - IPS^2/Watt is used instead of IPS/Watt to emphasize performance
- Best core configuration selected from 300 candidates for each 2K retired instruction interval based on IPS^2/Watt
- IPS^2/Watt improvement threshold of 20% yields a set of 10 core types, resulting overall IPS^2/Watt improvement is small.
- Increasing the threshold to 40% reduced the number of core types to 4
- Fixed number of core types to 4

Core Types obtained

Power Unconstrained core parameters:

Core type	Freq(Ghz)	Buffer Sizes (IQ,LSQ,ROB)	Width (fetch, issue)	Average power(W)
Average(AC)	1.6	36,128,128	4,4	2.2
Narrow(NC)	2	24,64,64	2,2	1.7
Larger(LW)	1.4	48,128,256	4,4	2.4
Smaller(SW)	1.2	12,16,16	1,1	0.82





Frequency and ROB size analysis for IPS^2/watt for AC core

Power constrained core designs

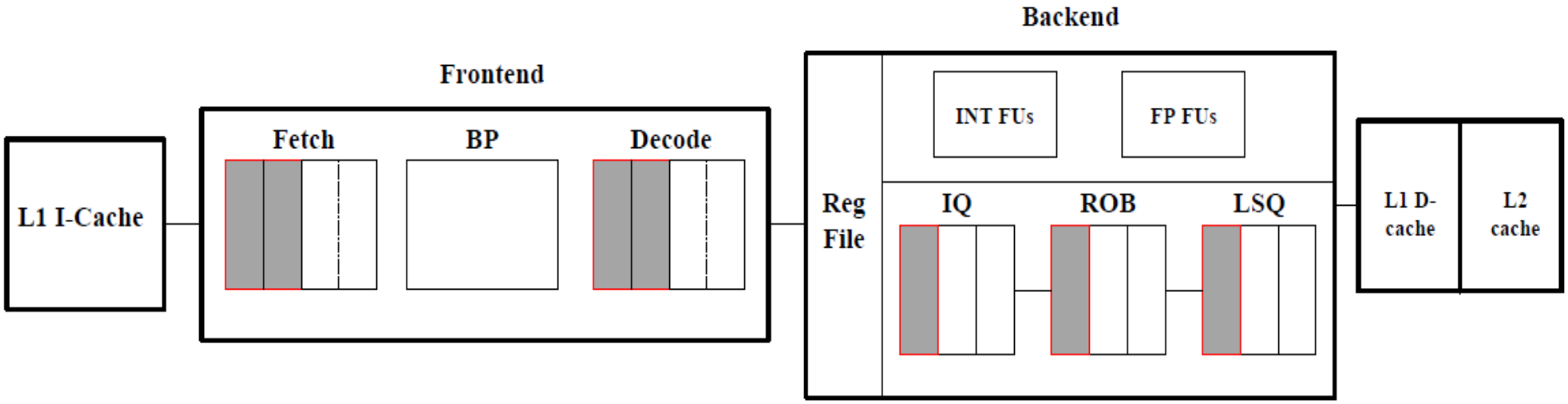
- Core types for a 2W peak power constraint:

Core type	Freq(Ghz)	Buffer Sizes (IQ,LSQ,ROB)	Width (fetch, issue)	Average power(W)
Average(AC)	1.4	36,128,96	3,3	1.6
Narrow(NC)	2	24,64,64	2,2	1.7
Larger(LW)	1.2	48,192,128	3,3	1.9
Smaller(SW)	1.2	12,16,16	1,1	0.82

- Core types for a 1.5W peak power constraint:

Core type	Freq(Ghz)	Buffer Sizes (IQ,LSQ,ROB)	Width (fetch, issue)	Average power(W)
Average(AC)	1.2	36,64,64	3,3	1.32
Larger(LW)	1	16,128,128	3,3	1.5
Smaller(SW)	1.2	12,16,16	1,1	0.82

Microarchitecture of Morphable Core



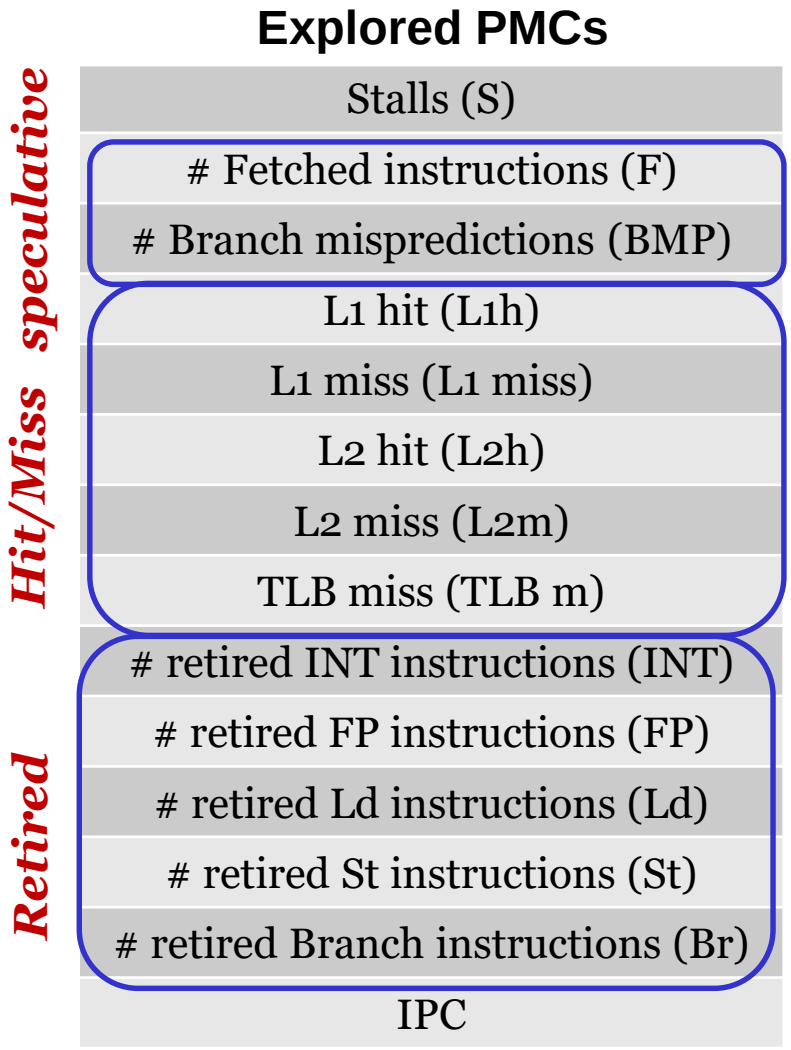
- IQ, ROB, LSQ are resized dynamically when morphing from one core type to another
  - ROB, LSQ and IQ are implemented as banked structures
  - Resizing involves turning on/off banks
- Reduce/increase fetch width, Power-off/on half the decoders

How to decide on a mode switch?

- Switching decision between modes is based on IPS^2/Watt
- To compute IPS^2/Watt , we need to estimate performance and power
  - Hardware performance counters (PMCs) are used to estimate performance and power at fine-grain granularity
- Need to estimate power and performance on the currently active mode as well as 3 other core modes

Power/IPC Prediction

1. Identify counters that impact performance & power
2. Choose representative workloads as "training set"
3. Identify smallest number and choice of counters
4. Regression analysis
   power(InO/OOO) = f(chosen counters)
5. Trained power/IPC expressions used online



Counter selection heuristic

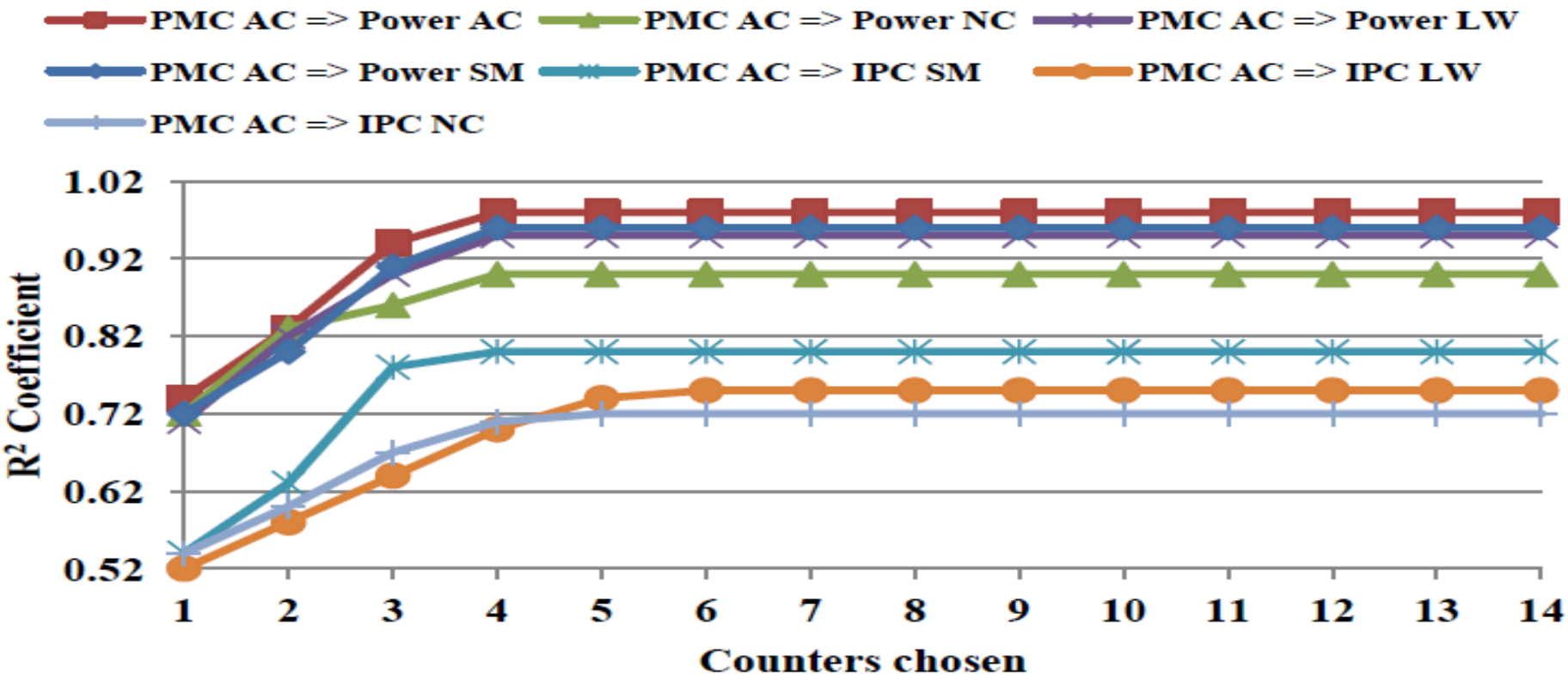
Input: PMCs & Power/IPC trace (of representative workloads)

Objective: Minimum no. of PMCs to fit power and IPC

Metric: R^2 coefficient of the fit (higher the better)

- Approach:
  - Search counter space (14) iteratively
  - Each iteration:
    - Choose a new counter that best fits IPC/Power trace along with counters chosen in the previous iterations
    - Note the R^2 coefficient value
  - Plot R^2 coefficient obtained for each iteration
  - Best set of counters around the region where R^2 coefficient saturates

Online Estimation using PMCs

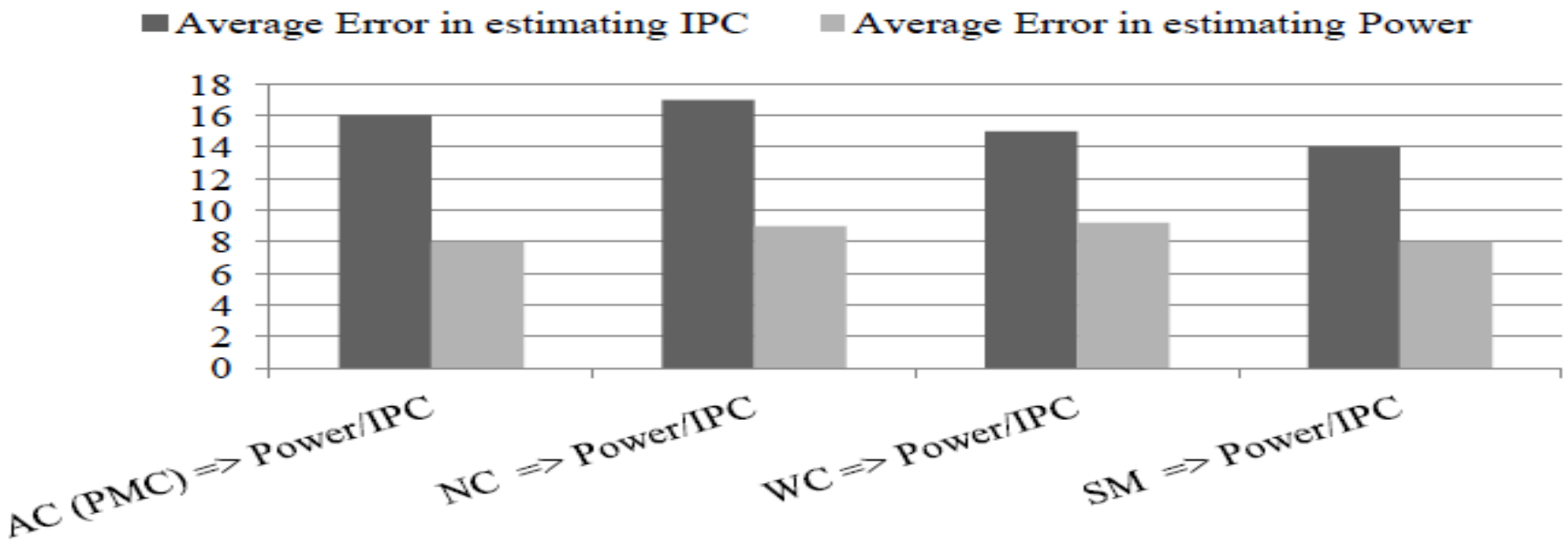


PMC AC => Power NC, denotes using the performance counters of the normal core to estimate the power on the narrow core.

Obtained Power and IPC expressions

Estimated Parameter	Expression
AC $\Rightarrow$ AC Power	$0.0141 \times L1h + 13.81 \times IPC + 0.0295 \times S - 0.0118 \times Bmp - 0.2989$
AC $\Rightarrow$ NC Power	$-1.3 \times Bmp - 0.85 \times L1m + 0.4112 \times Br + 2.3 \times 10^{-5} \times St + 0.46$
AC $\Rightarrow$ LW Power	$-0.347 \times L2m - 1.04 \times Ld - -0.569 \times Bmp - 1.4 \times 10^{-5} \times L1h + 0.1$
AC $\Rightarrow$ Power SM	$- 0.00616 \times L1m + 0.06671 \times IPC - 4.2 \times 10^{-4} \times Bmp + 0.2768$
AC $\Rightarrow$ IPC SM	$0.2 \times L1h + 0.9022 \times IPC + 0.0104 \times L - 0.0103 \times Bmp + 4.4669$
AC $\Rightarrow$ IPC LW	$0.0141 \times IPC - 1.81 \times L1h + 0.3195 \times S - 1.23 \times L1m - 0.2989$
AC $\Rightarrow$ IPC NC	$0.12 \times Br + 1.81 \times IPC + 0.395 \times S - 0.0118 \times L2h - 0.389$

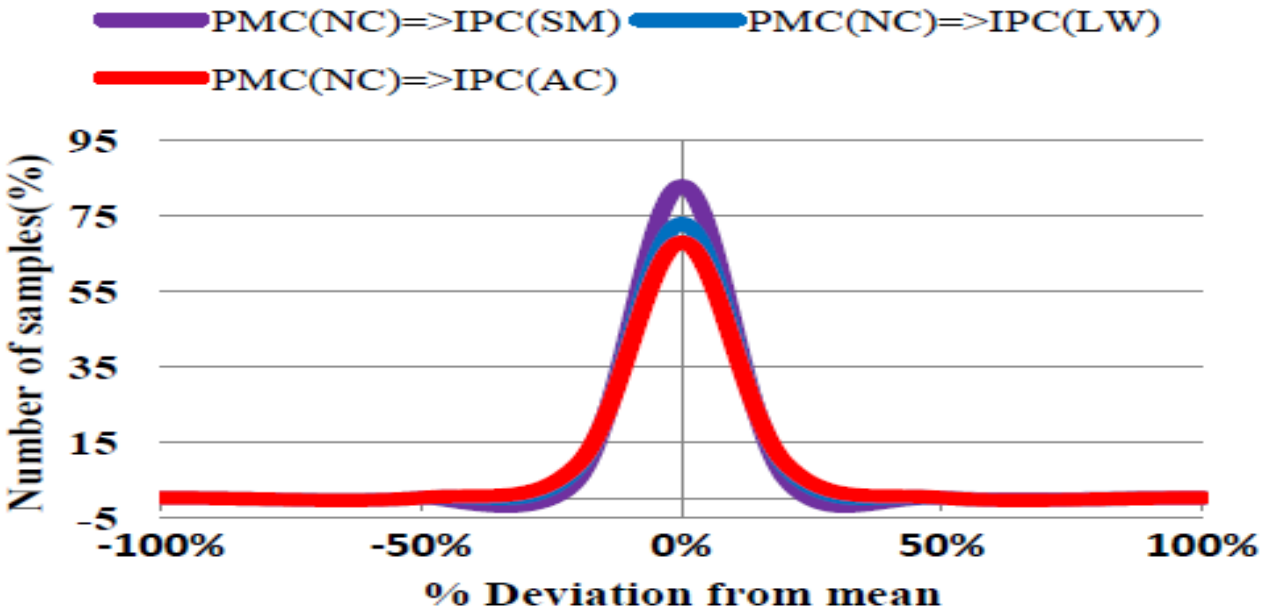
Average Error Estimation using PMCs



AC(PMC) => Power/IPC denotes the average error in estimating power and IPC for the 3 other core types using the PMCs of the average core (AC)

Maximum average % error of only 16 %; reasonably high accuracy

Error distribution



Distribution of error in estimating IPC in various core types using PMCs of narrow core (NC)

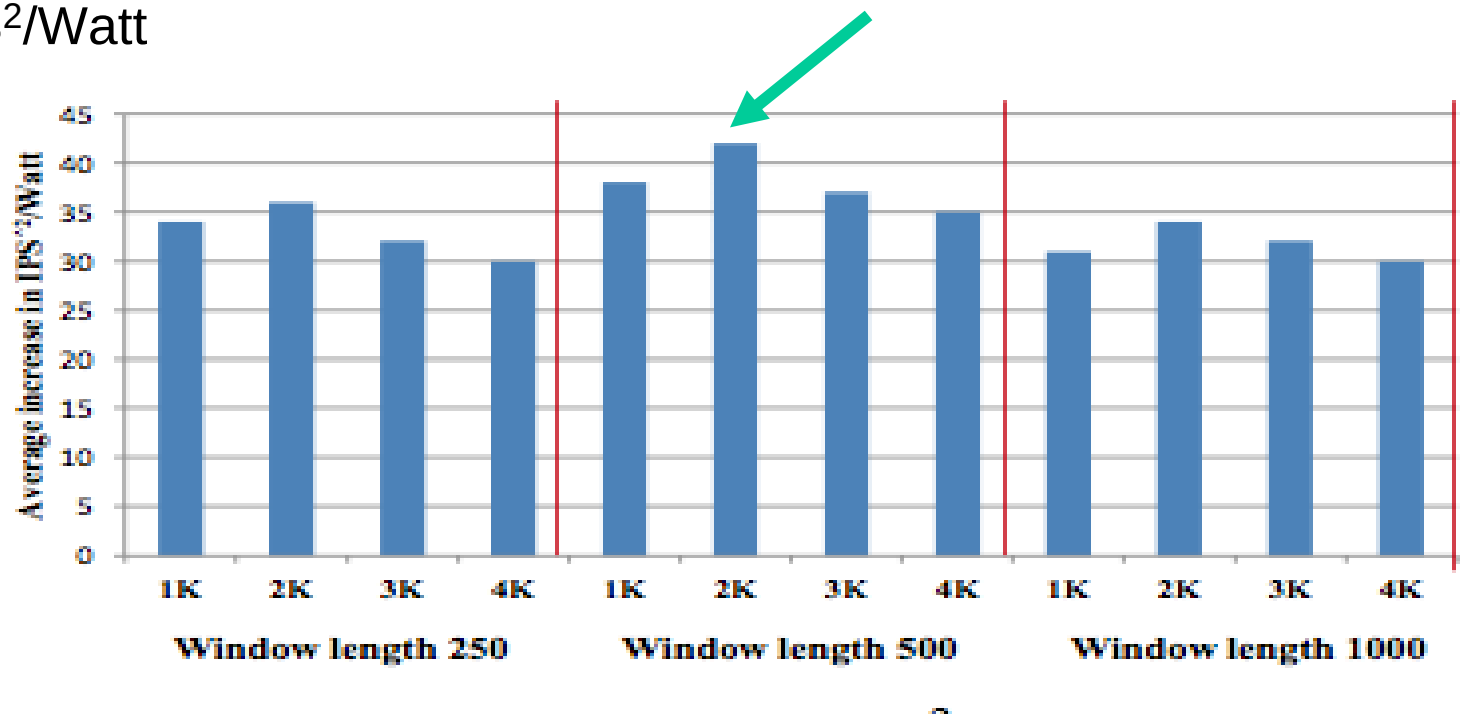
Deviation of errors from mean is low for most sample points with up to 80% between +/- 10% from the mean

Capturing Application Phase Behavior

- Power and performance are monitored every “window” of instructions (size to be determined)
- To prevent frequent morphing during a transient behavior, we wait for several windows (“history_depth”) and follow the most frequent recommendation
- Morphing decision is based on behavior during the last ***n*** retired instructions, where n=Window_Size x History_Depth
  - Too small ***n*** results in too frequent switching, causing high overhead
  - Too large ***n*** will result in much smaller benefits

Frequency of morphing decisions

- Morphing decision is based on behavior during the last ‘n’ retired instructions
- Window size and history depth combination that yields the max IPS^2/Watt



Decision to Reconfigure is taken at the end of every 2K instructions

Fine-Grain DVFS

- Morphing from one core mode to another involves frequency change
- We need low overhead mechanism for Voltage/Frequency scaling
- Traditional DVFS is applied at coarse grain instruction granularity (millions of cycles)
  - Due to high overhead involved in scaling voltage/frequency
- Use of On-Chip regulator reduce the time needed for scaling voltage to tens of nanoseconds or hundreds of processor cycles [Kim et al. HPCA 2008]
- Use fine-grain DVFS technique (upon mode switch) with overhead of 200 cycles using on chip regulator

Overhead for Morphing

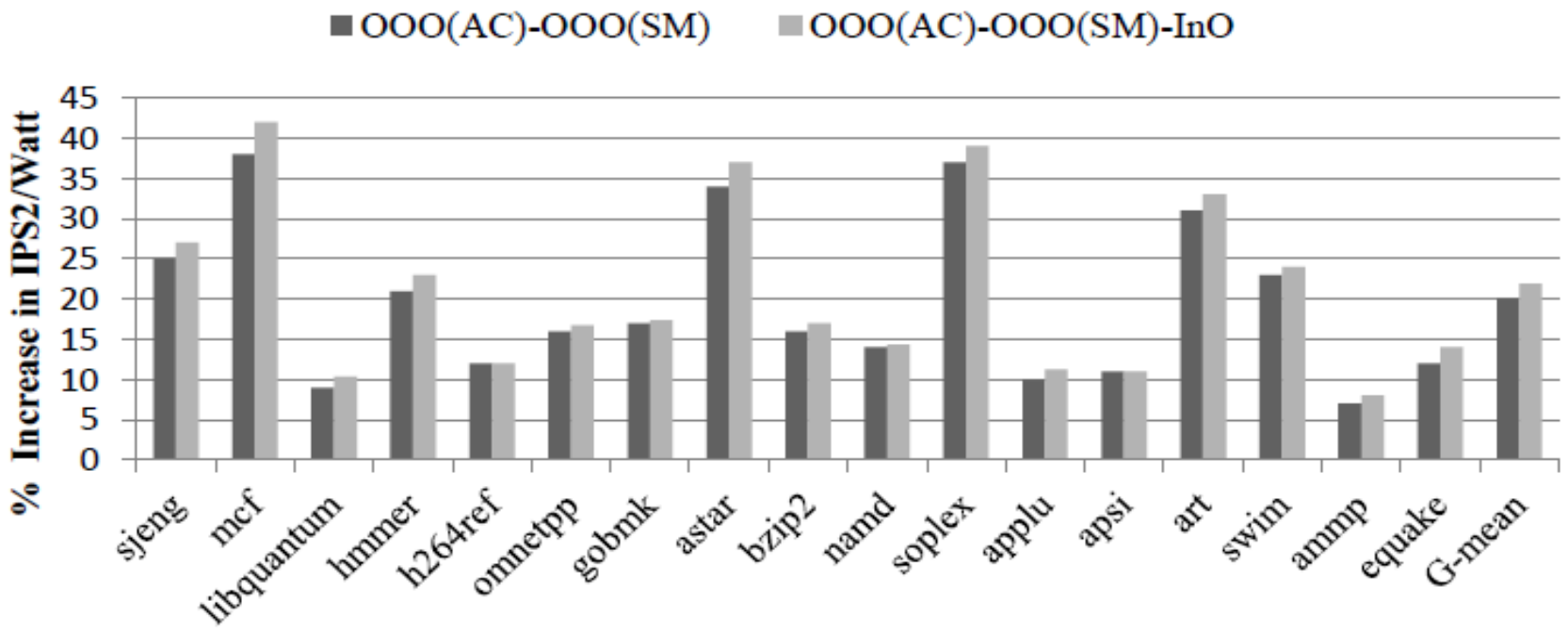
- In-core morphing retains processor state and cache content
  - register file, caches and branch predictor are shared
- Still additional cycles are needed upon reconfiguration
- Partial powering off/on fetch, decode units
  - Power gating banks of ROB, RAT and LSQ units (10 clock cycles to power off one bank)
  - The fine grain DVFS overhead is 200 clock cycles
  - Pipeline drain on every core mode switch
- On Average morphing overhead takes 500 cycles . Exact overhead can be determined only in run time depending on core we morph into

Result and Analysis

- Gem5 simulator; McPAT to measure power
- We evaluate our proposed scheme with SPEC2006 and SPEC2000 benchmarks suite
- Benchmarks were compiled using gcc for Alpha ISA with -O2 optimization
- Benchmarks ran for 2 billion with skipping the first 2 billion instructions

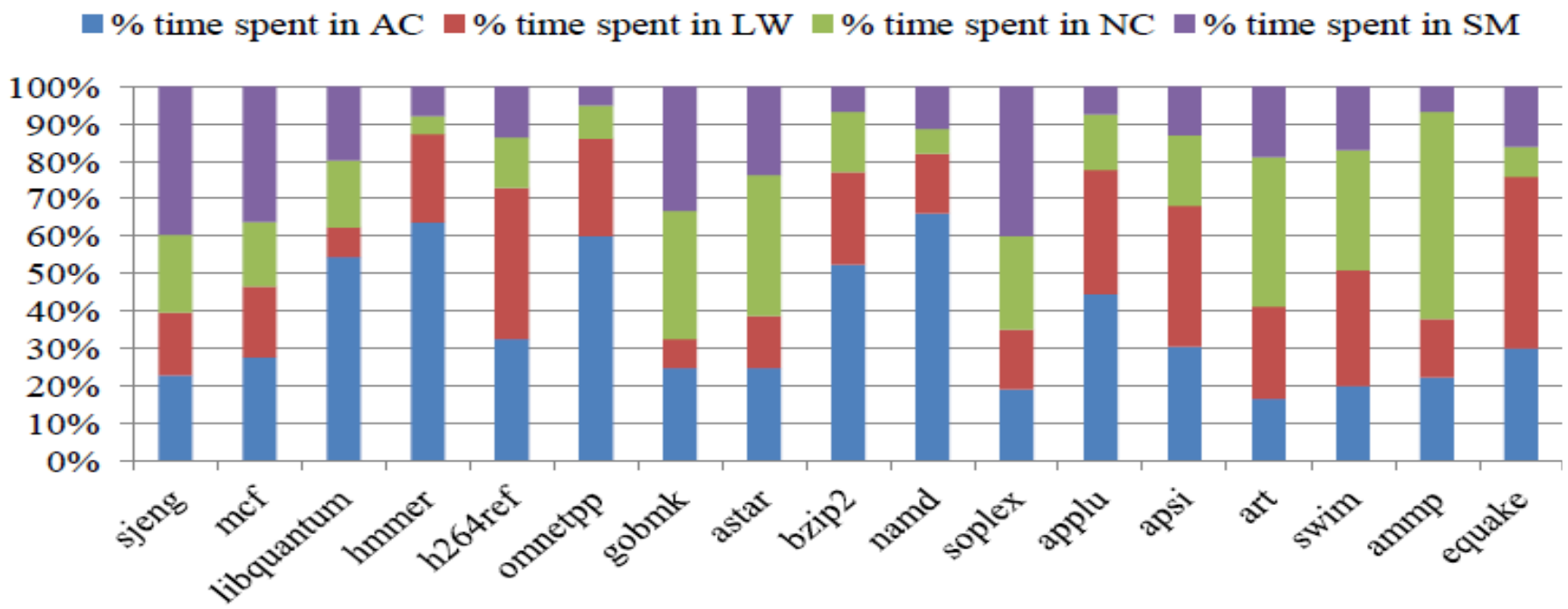
- When to switch Modes?
  - To avoid frequent switching the IPS^2/Watt computed on each core type should be at least greater than 5% than currently executing core mode.

Morphable architecture With/Without InO core



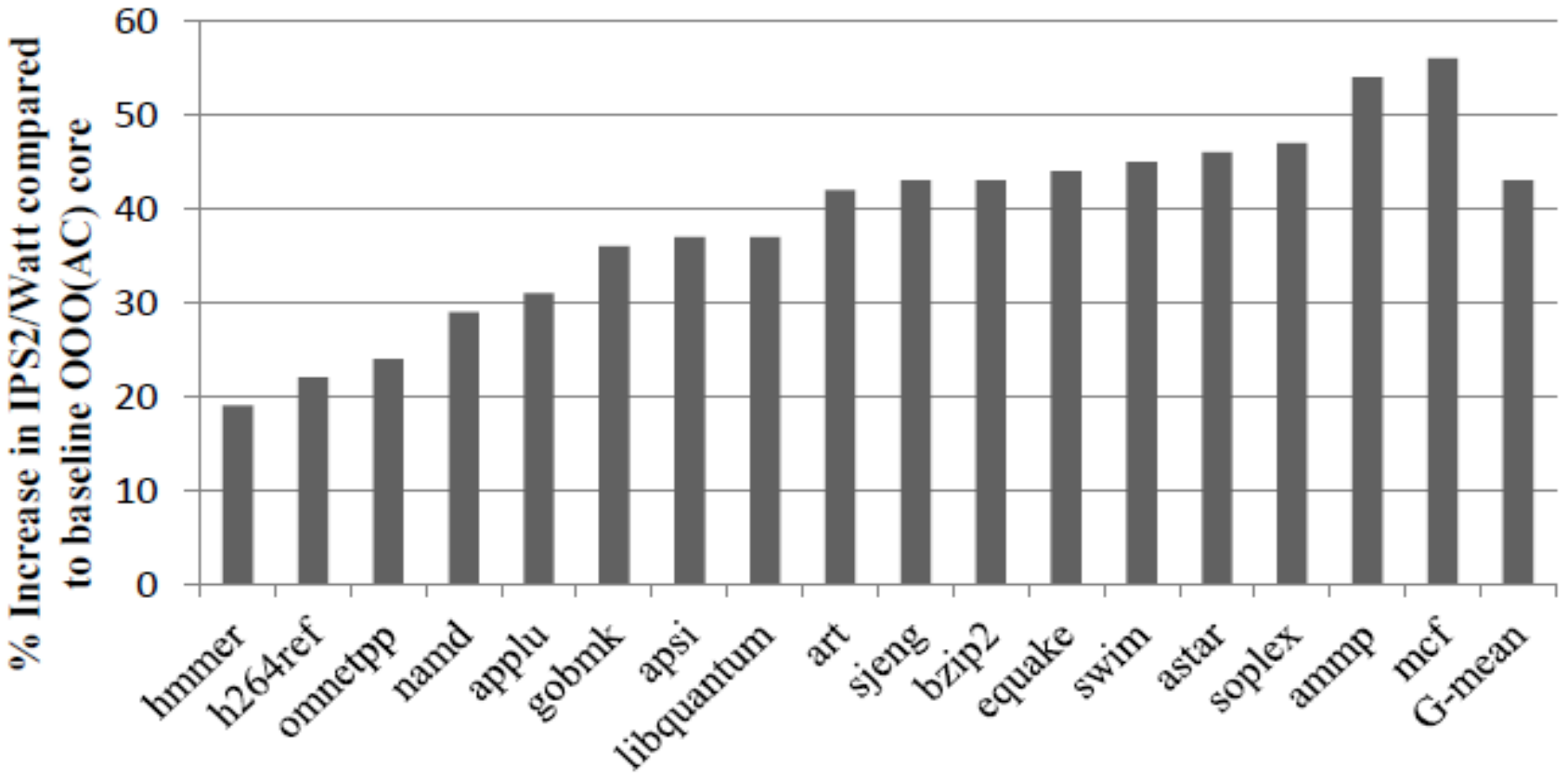
- 3-mode morphing scheme with the inclusion of InO can provide us only 2% additional *IPS*^{2}/Watt benefit compared to the 2-mode morphing between OOO(AC)-OOO(SM)

Time spent in cores types in self-morphing scheme



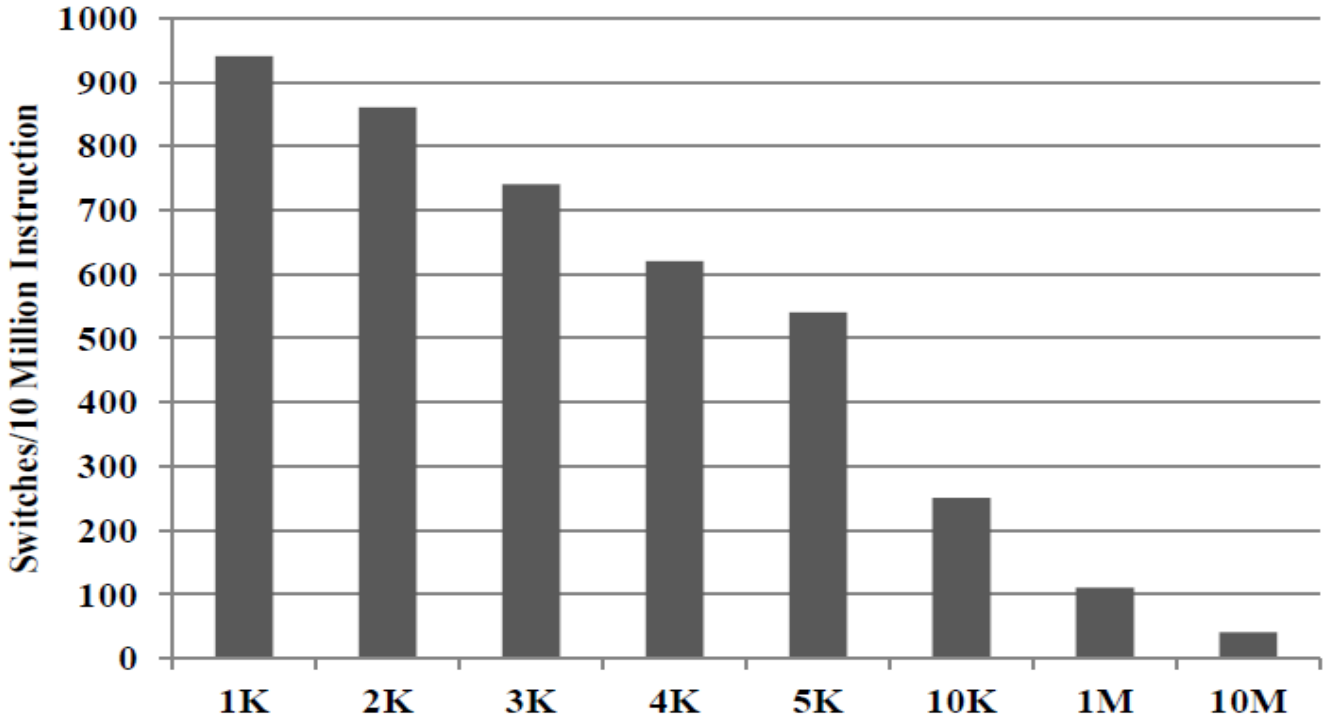
- Percentage occupancy in each of core types in the benchmark
- At fine grain granularity low performance phases of benchmark can be mapped to smaller(SM) core
- LW, NC core types resolve processor bottlenecks

Improvement in *IPS*^{2}/Watt of self-morphing scheme



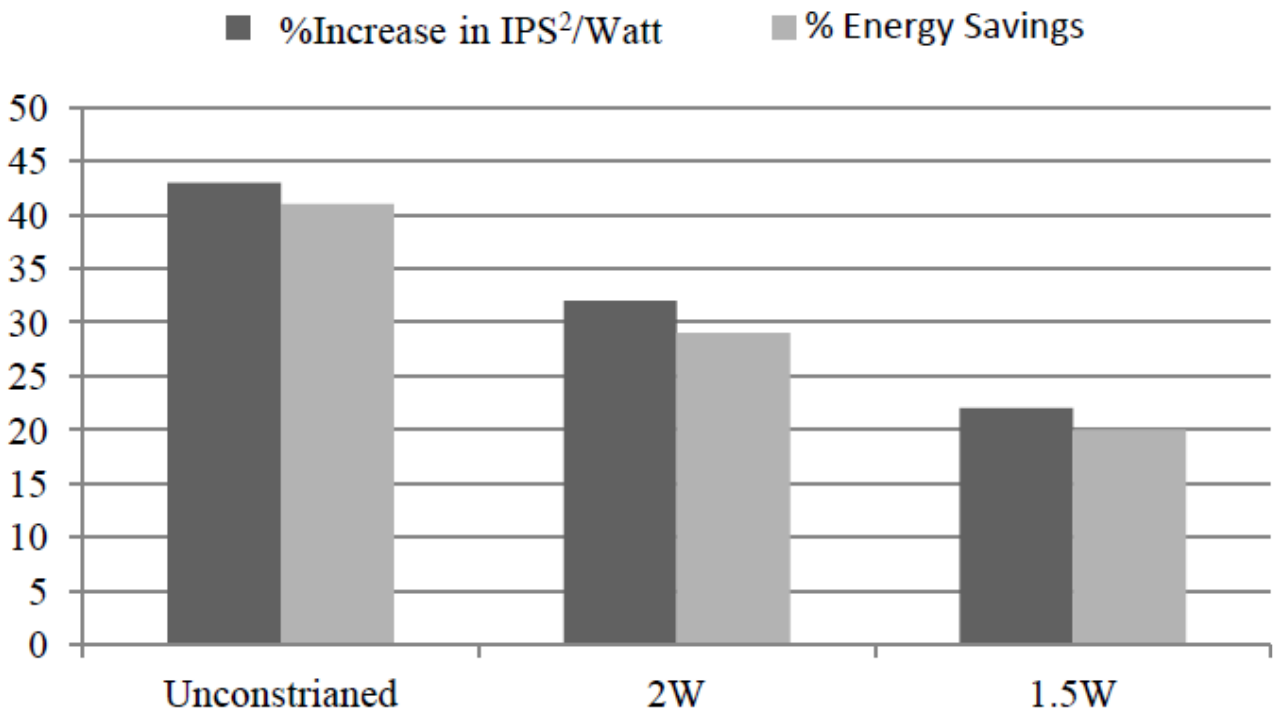
- Average IPS^2/Watt benefit of 43%

Number of switches in self-morphing scheme



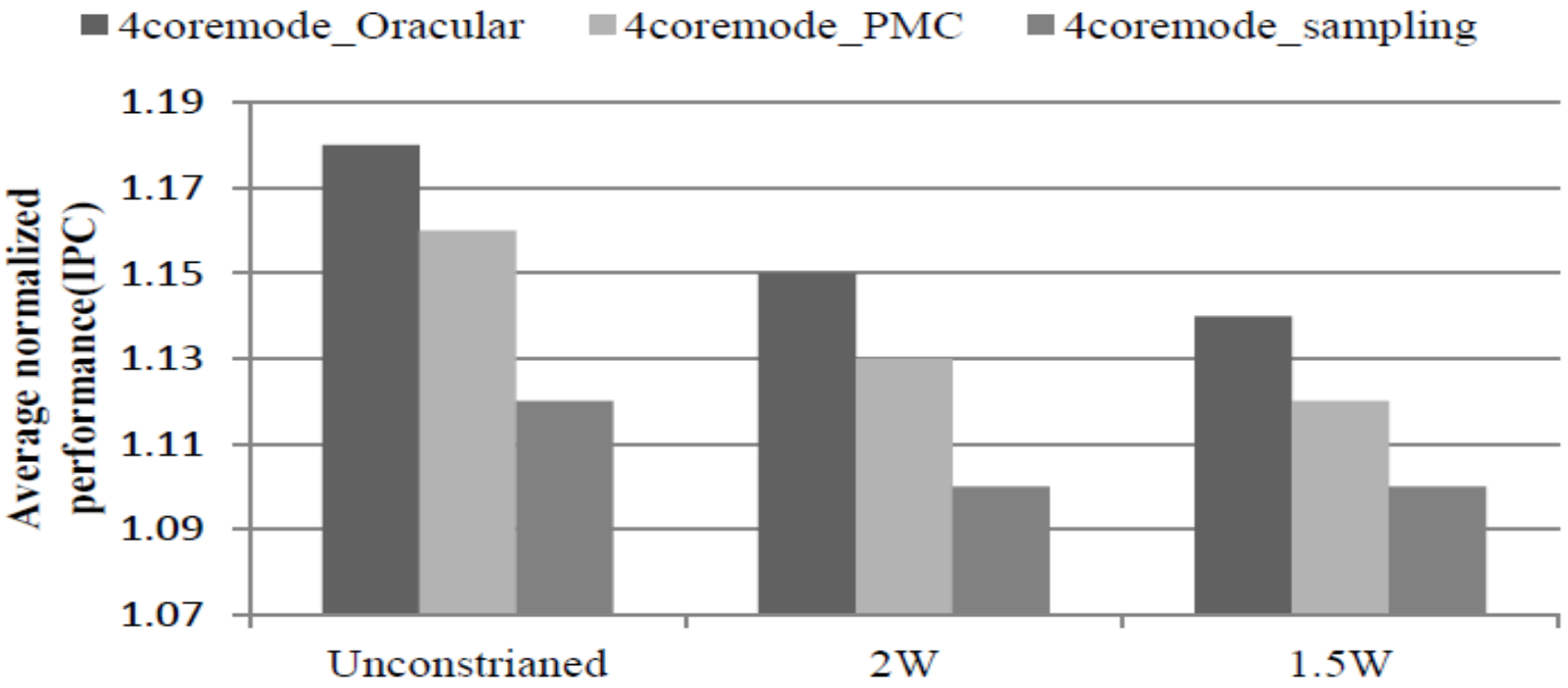
Selected 2K instruction interval the number of switches on average is 850 in 10M instructions, i.e., after every 2K instructions we have a probability of 17% to perform a mode switch

Comparing Power Constrained and Un- Constrained cores



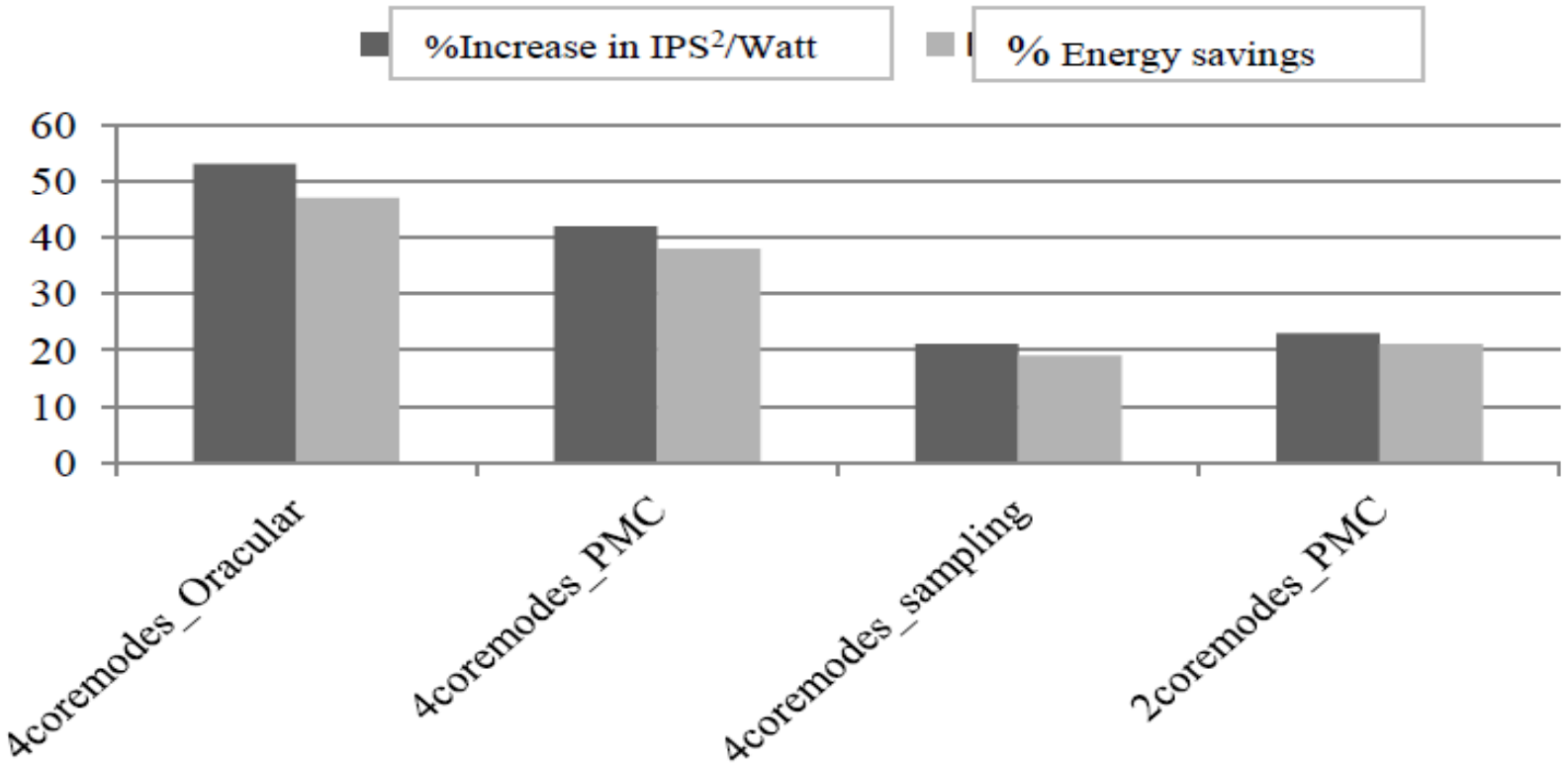
- For the unconstrained case, we obtained 38% energy saving compared to Average core type

IPC of switching schemes



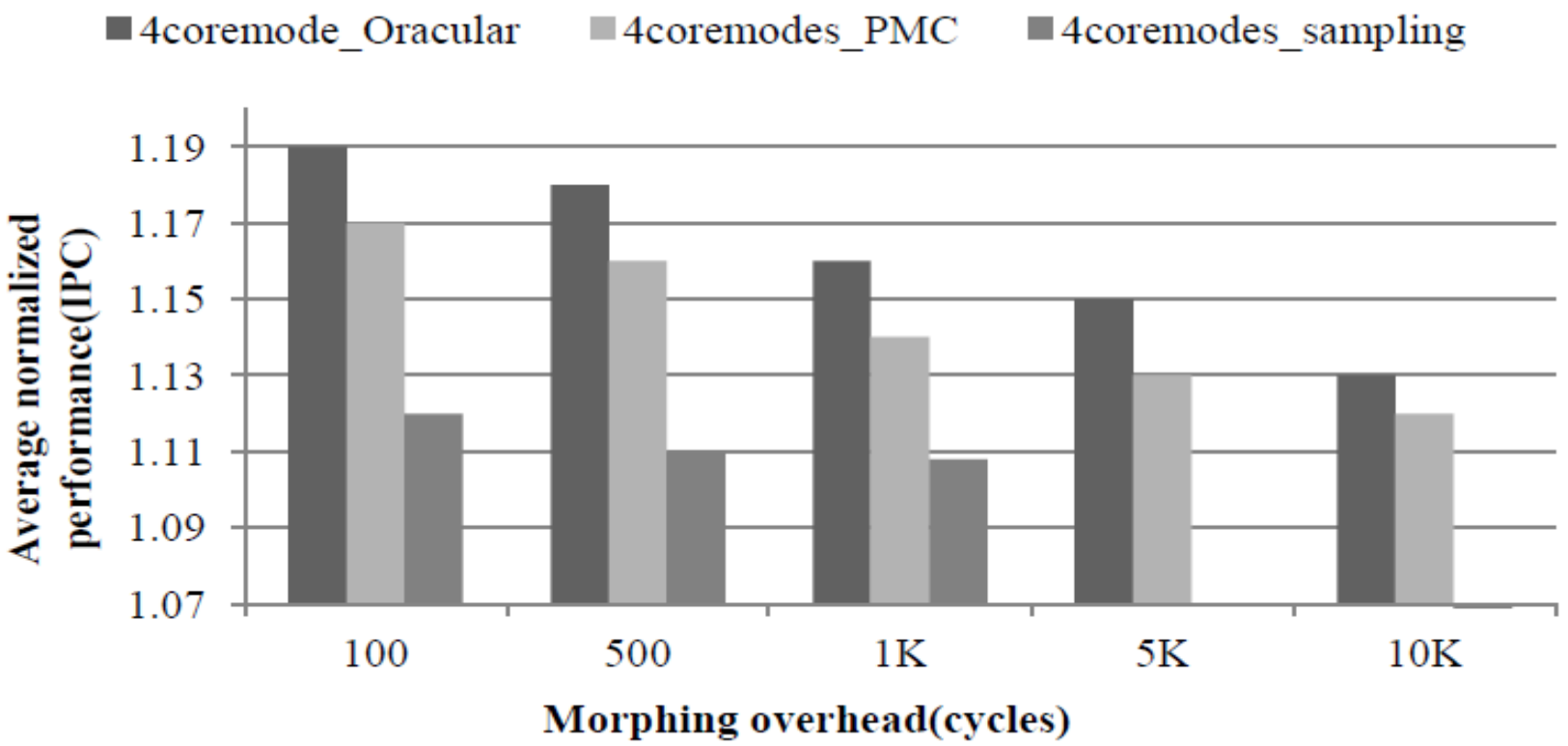
- Unconstrained case: 16% improvement in IPC using the PMC-based scheme compared to 12% achieved by the sampling-based scheme

IPS^2/Watt and Energy saving of morphing schemes



Our **4coremode_PMC** scheme provides 20% more energy savings compared to **2coremode_PMC**

Impact on IPC of morphing scheme overhead



When the morphing overhead increases from 500 to 5K cycles, the average performance of *4coremode_PMC* scheme drops by only 3%

Conclusion

- Thread migration/core-hopping is expensive
  - Performed at coarse grain
- In-core morphing preserves states, cache
  - Minimal overhead
  - Simple hardware
  - Allows frequent morphing - larger gain
- Using counter-based prediction mechanism to predict IPC and power at very fine grain granularity
- Results indicate average improvement in IPS^2/Watt of 43%