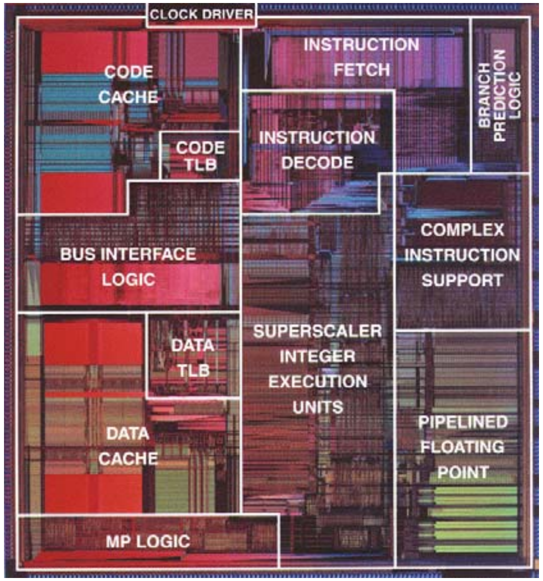
Over-provisioned Multicore Systems

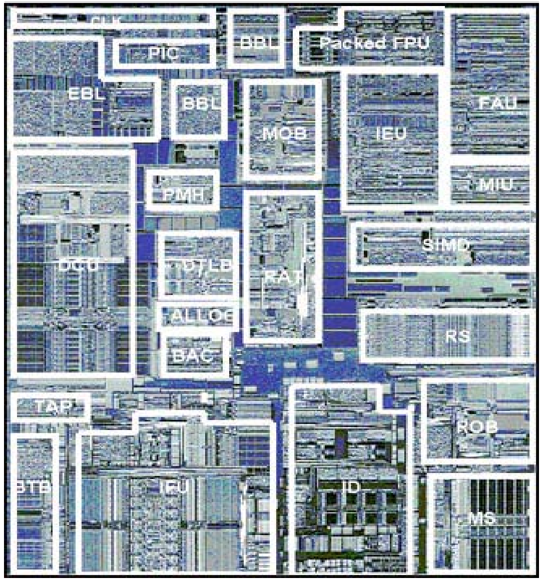
Koushik Chakraborty
Computer Sciences Department

THE UNIVERSITY of WISCONSIN
MADISON

Technology Scaling: Classical View



Generation 1



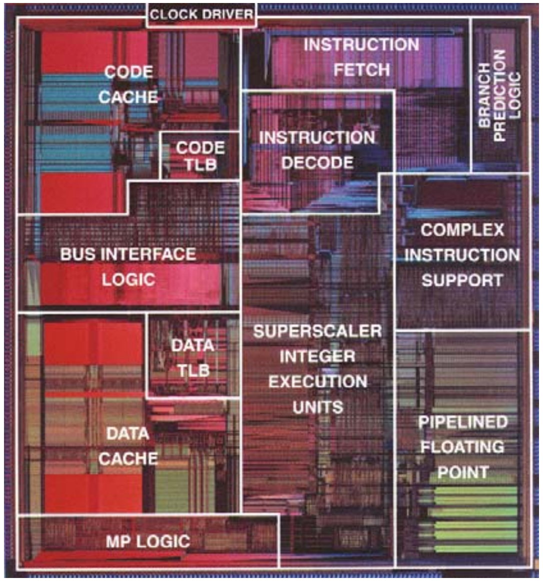
Generation 2

Generation N 3

- Classical Scaling: successive generations
  - Device size is halved: same area offers **2X** resources
- Opportunity: performance and concurrency

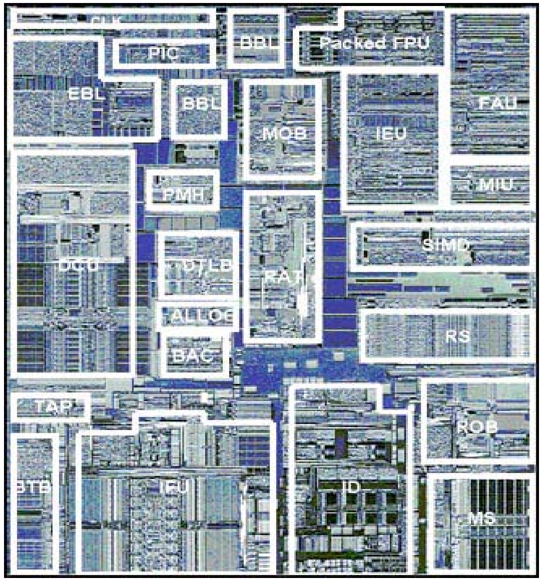
Technology Scaling: Power

18 W



Generation 1

27 W



Generation 2

610 W

Generation N

- **Scaling Challenge: Power**
  - Power improvement lags capacity improvement

Why power is a problem?

AMD
Athlon 1400
Socket 462
VIA KT133A Chipset

* Warning: Do not attempt this on your system

Power and Cooling

- **Power Supply and Cooling**
  - Hard limit on cost-effective cooling solution
  - Difficult to supply (large) power in small enclosure
  - Cost components are substantial
- **Limited room for increasing processor power consumption**
  - Constant Thermal Design Power (TDP)
  - Performance and energy efficiency must improve

Thesis Contributions

- ❑ **Simultaneously Active Fraction**
  - – Model for power constraint
  - – Application in multicore design
- ❑ **Over-provisioned Multicore Systems (OPMS)**
  - – Over-provisioning core resources
  - – Design consideration and implementation
- ❑ **Computation Spreading**
  - – Classic application for an OPMS
  - – Selectively employs on-chip processing cores to reduce power consumption but improve compute efficiency

Outline

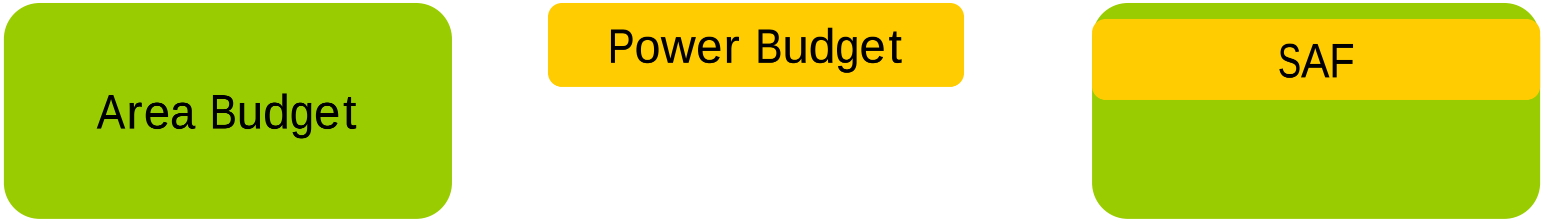
- ❑ Motivation
- ❑ **Simultaneously Active Fraction**
  - Area perspective of power constraint
  - SAF Trends
  - Application of SAF in Multicore Design
- ❑ Over-provisioned Multicore System
- ❑ Computation Spreading
- ❑ Results
- ❑ Conclusion

Area and Power constraints

- ❑ **Area constraint**
  - – Aggregation of on-chip device area
  - – Statically satisfied
    - – Each technology generation defines the minimum device area
- ❑ **Power constraint**
  - – Aggregation of individual device power
  - – *Dynamically* satisfied
    - – Devices operate at a wide range of power levels
    - – Different subsets of devices can account for chip power
  - – Hard to accurately estimate chip power at an early design phase

Power constraint: An Area Perspective

- ❑ **Current systems are power limited**
  - – A shift from area limited designs of the past
    - – Many architectural intuitions deal well with area
  - – Connecting theme: managing resources



Simultaneously Active Fraction (SAF): Fractional area consuming target power

Transformation: Power constraint to Area constraint

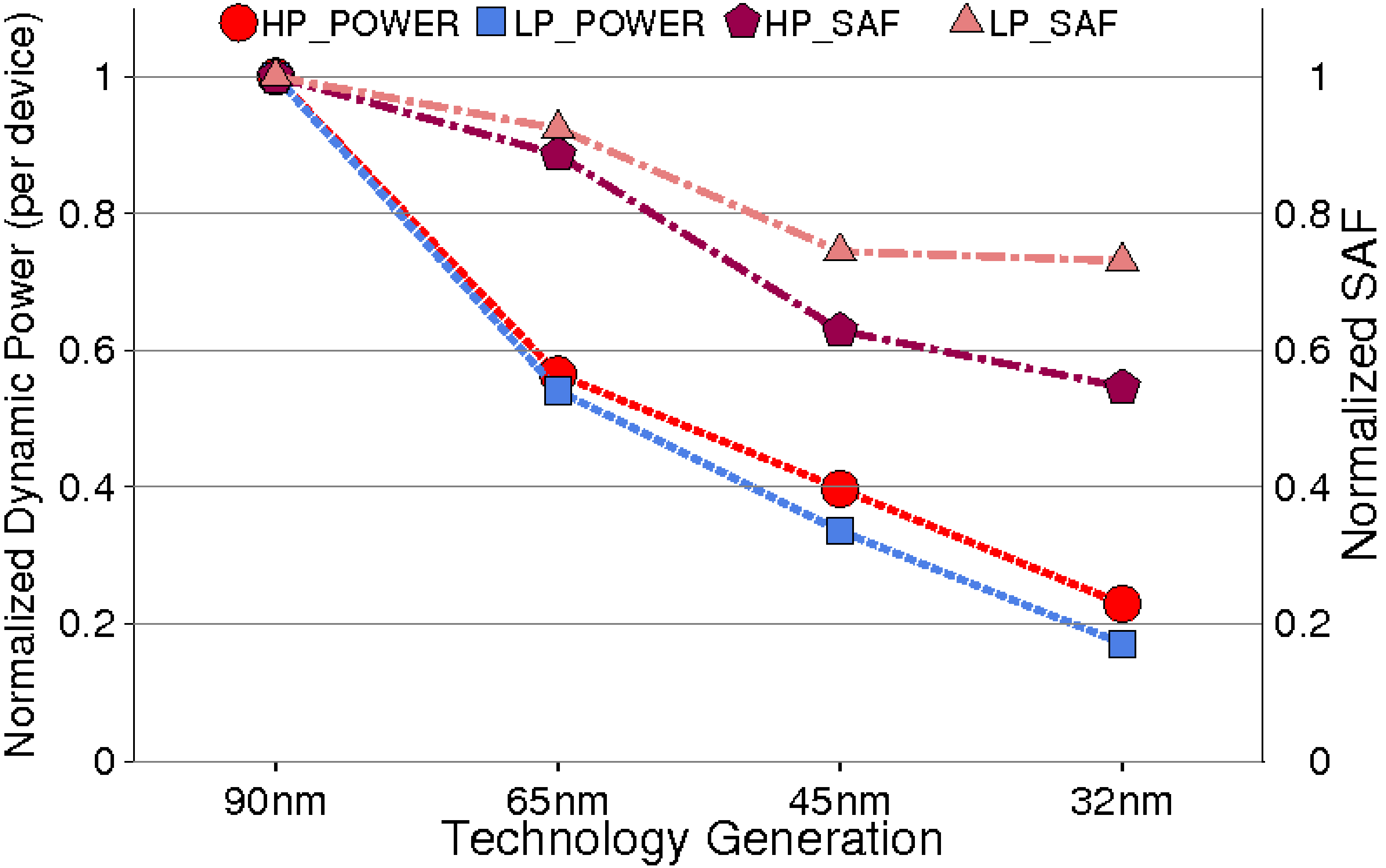
SAF: First order model

SAF = Power / (*Individual Device Power * N_D*)

- ❑ N_D: Number of devices
  - – 2X increase from Device scaling
- ❑ **Power: remains constant**
- ❑ Individual Device Power
  - – Dynamic power from switching
    - – Key parameters: **voltage**, capacitance, frequency
    - – Small improvement due to limited voltage scaling
  - – Static power from leakage
    - – Manufacturing process and circuit design style

SAF will shrink with technology scaling

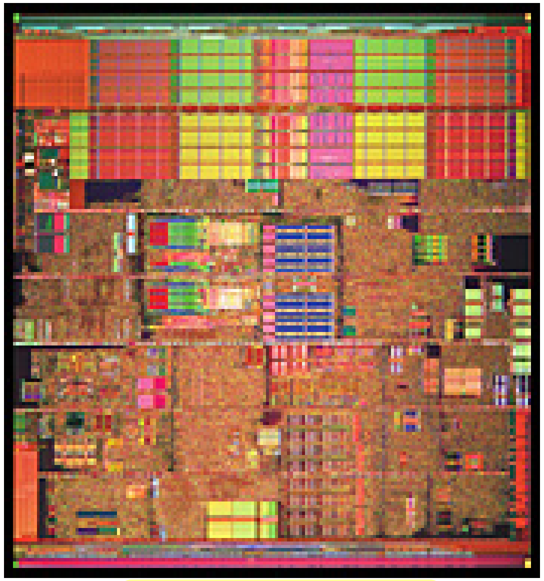
SAF Trend



* Assuming only dynamic power

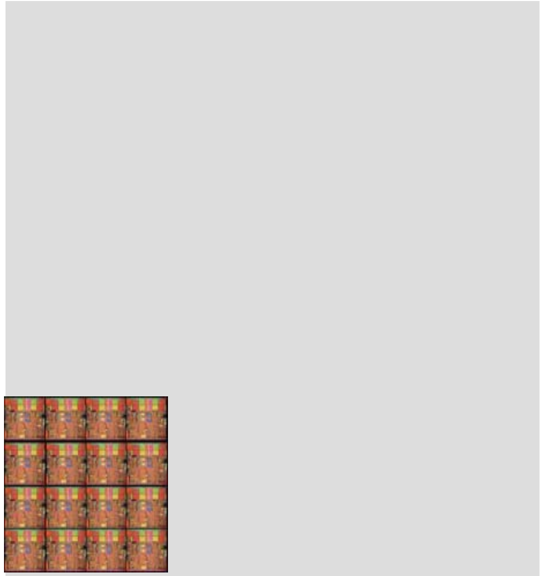
Technology Scaling: SAF View

P



Generation 1

P



Generation X

- ❑ **Opportunity: SAF-aware multicore design**
- ❑ **Paradigm shift: Can’t use all resources simultaneously**
  - – what if 64 cores, but can use only 32 at a time?

Application of SAF

- ❑ **Impact of power constraint at the early design phase**
- ❑ **Dissertation illustrates two examples**
  - – Hill-Marty model extension
  - – Multithreaded Workloads

Hill-Marty model: Multicore Speedup

- ❑ **Based on Amdahl’s Law**
  - – Workload: sequential and *infinitely* parallel phase
- ❑ **Resources: *n* unit cores**
  - – *r* cores can be combined, sequential performance: *perf(r)*
- ❑ **Multicore Configurations**
  - – Symmetric: all on-chip cores look alike
  - – Asymmetric: structurally distinct on-chip cores
  - – Dynamic: dynamic re-configuration (e.g., combine *r* unit cores dynamically to boost sequential performance)
- ❑ **Speedup: Dynamic > Asymmetric > Symmetric**

What if Multicores are only power limited?

Power constraint in Hill-Marty Model

- **De-couple Area and Power constraint**
  - Modeling Power constraint using SAF α, where $(0 < \alpha \leq 1)$
  - Number active cores limited by $\alpha * n$
  - Dynamically allocate power budget among on-chip cores
- **Symmetric Multicore**

$$Speedup = \frac{1}{\frac{1-f}{perf(r)} + \frac{f}{perf(r)*(n/r)}}$$

$$SAFSpeedup = \frac{1}{\frac{1-f}{perf(r)} + \frac{f}{perf(r)*(\alpha*n/r)}}$$

- **Dynamic Multicore**

$$Speedup = \frac{1}{\frac{1-f}{perf(n)} + \frac{f}{n}}$$

$$SAFSpeedup = \frac{1}{\frac{1-f}{perf(\alpha*n)} + \frac{f}{\alpha*n}}$$

Asymmetric Multicore

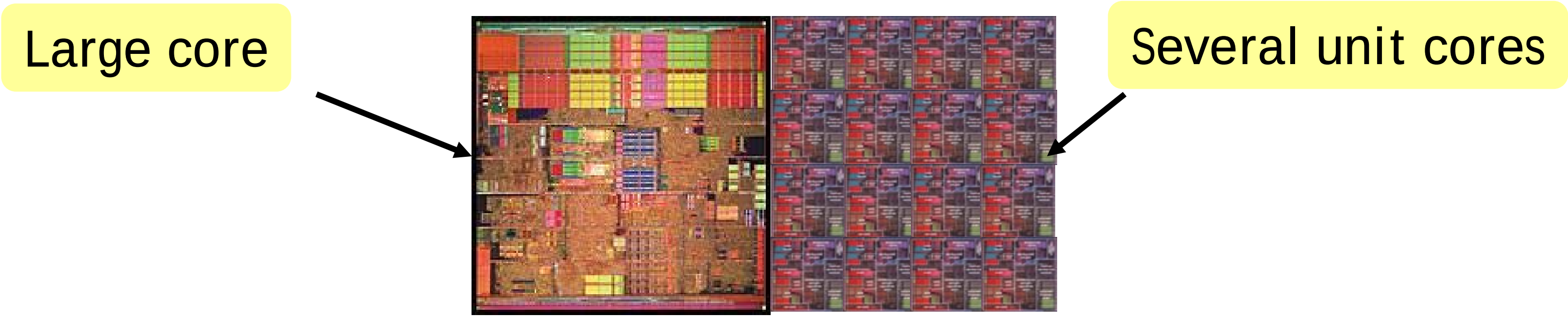
- ❑ **Distinct cores: where to assign computation?**
  - – Best fit computation assignment
- ❑ **Hill-Marty Model**
  - – Sequential phase: large core composed of *r* unit cores
  - – Parallel phase: all the on-chip cores

$$Speedup = \frac{1}{\frac{1-f}{perf(r)} + \frac{f}{perf(r)+n-r}}$$

- ❑ **SAF aware: available cores > allowable active cores**
  - – Core resources can be **over-provisioned**

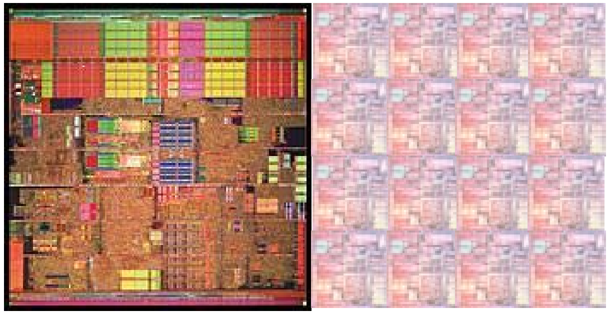
Asymmetric Multicore

- ❑ **Sequential and parallel phase exploit different cores**
  - – Best case: $\alpha * n + r \leq n$

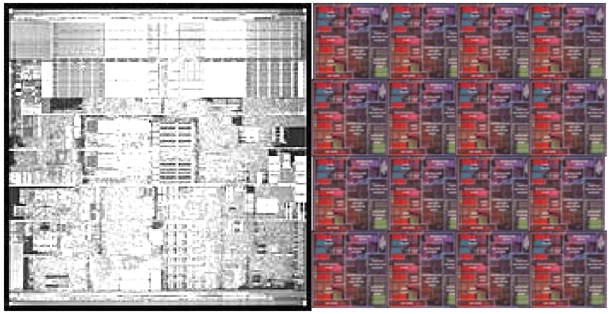


Large core

Several unit cores



Sequential Phase



Parallel Phase

SAF-aware Asymmetric Multicore

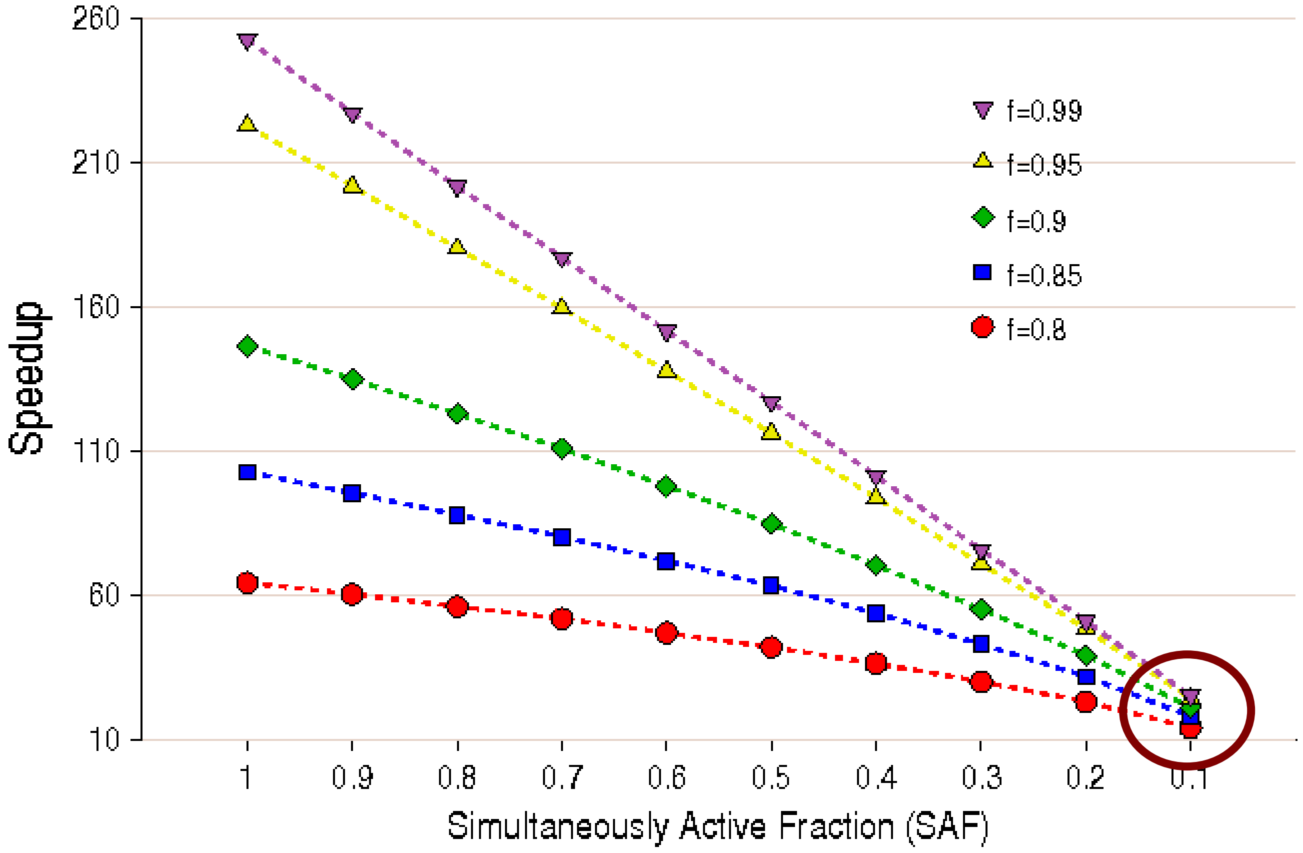
- **Sequential and parallel phase exploit different cores**
  - Best case: $\alpha * n + r \leq n$
  - Otherwise, during parallel phase choose between
    - Using sequential core + few unit cores
    - Only use unit cores

$$SAFSpeedup = \begin{cases} \dfrac{1}{\dfrac{1-f}{perf(r)} + \dfrac{f}{\alpha * n}}, \text{if } \alpha * n + r \leq n \\ \dfrac{1}{\dfrac{1-f}{perf(r)} + \dfrac{f}{\max(perf(r) + \alpha * n - r, n - r)}}, \text{if } \alpha * n + r > n \end{cases}$$

Parameters

- ❑ **SAF: α**
  - – Speedups shown for $\alpha = 0.1 \text{ to } 1.0$
- ❑ *n:* **256**
- ❑ ***r*: graph shows speedup for optimal *r***
  - – Restriction: $r \leq \alpha * n$
- ❑ ***f*: degree of parallelism**
  - – Graph shows speedup for five different *f*

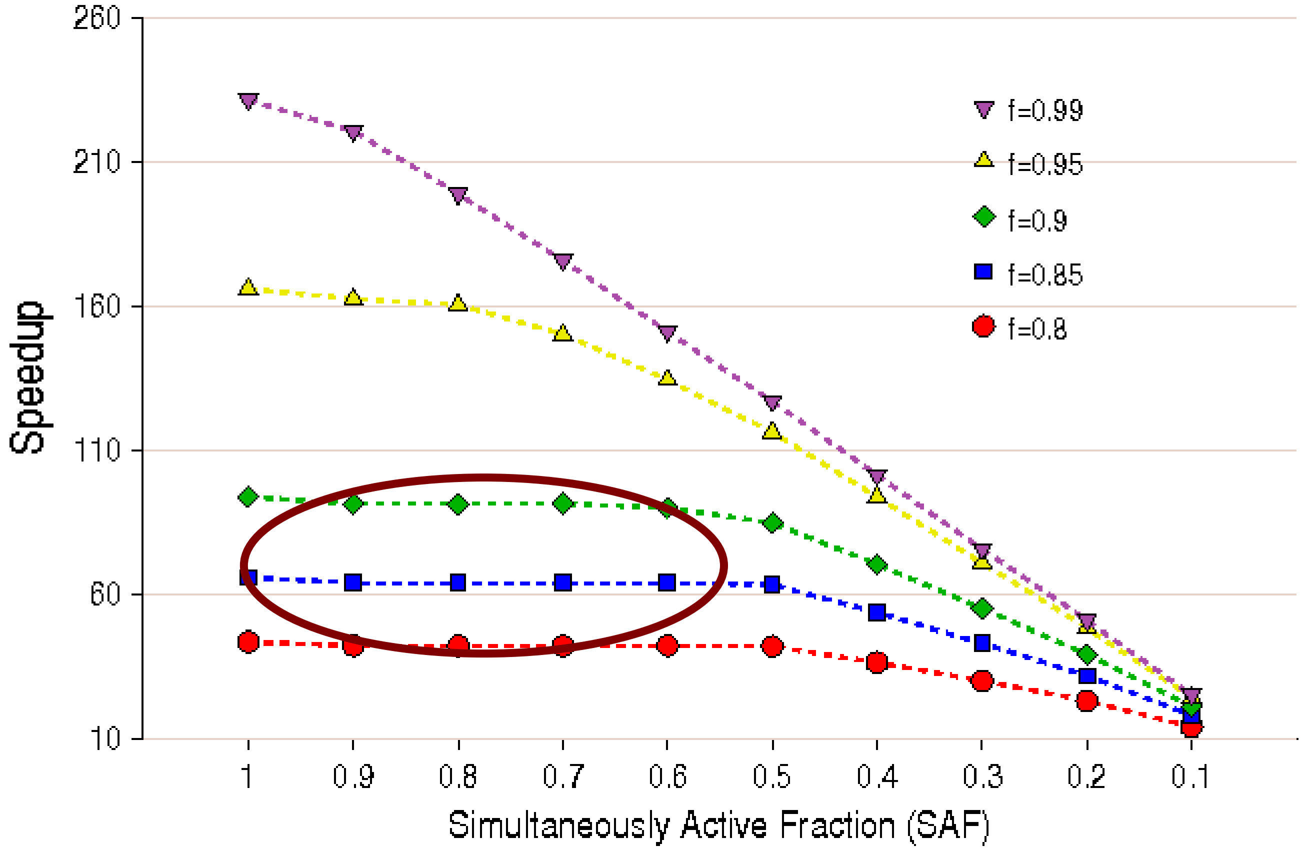
SAF Speedup: Dynamic Multicore



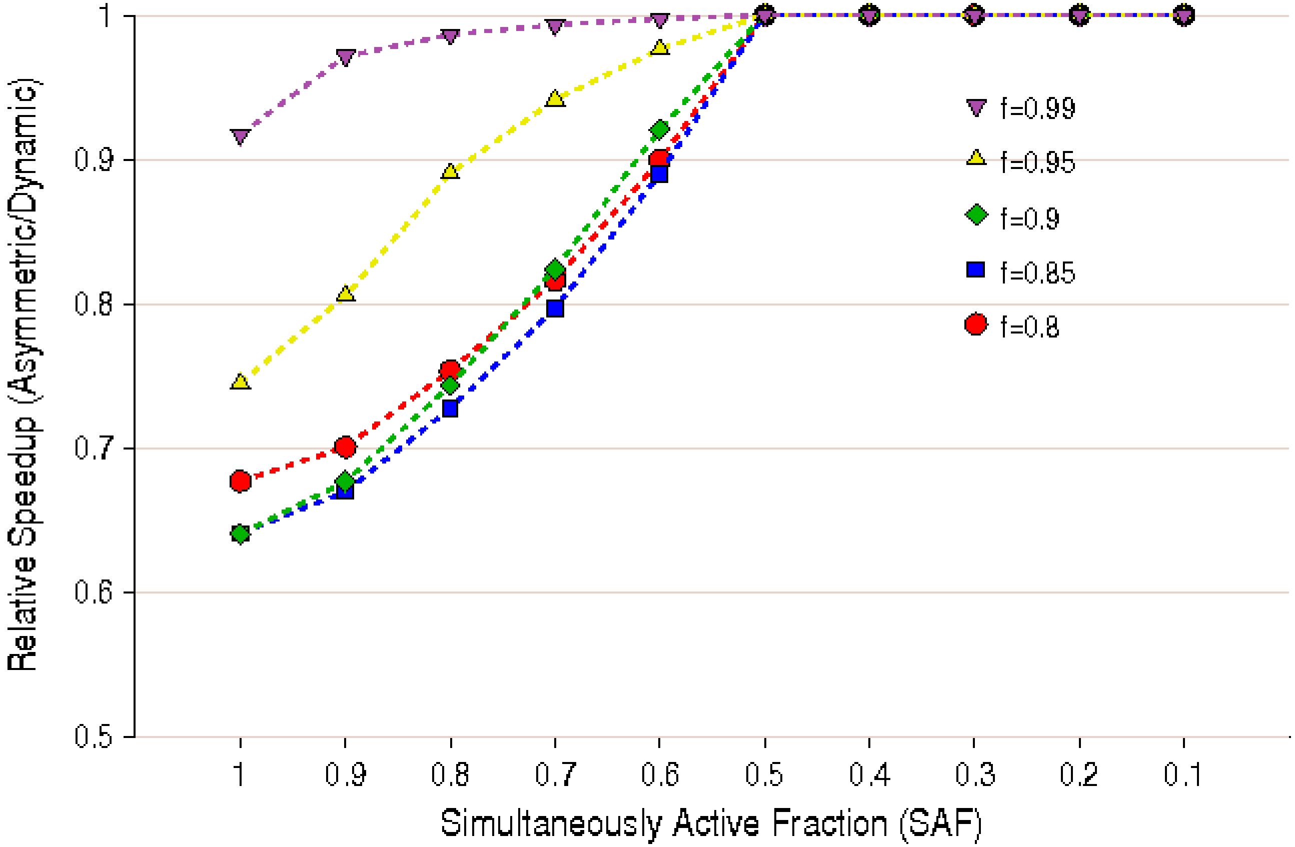
Diminishing performance gap at lower SAF

SAF Speedup: Asymmetric Multicore

Performance stability with diminishing SAF



Asymmetric versus Dynamic



At SAF=1/2 and lower, core assignments become logically equivalent

SAF Summary

- ❑ **SAF: abstract model of power constraint**
  - – SAF expected to shrink with technology scaling
- ❑ **SAF Application: Hill-Marty model extension**
  - – At higher power constraints, power rivals available parallelism as a major performance bottleneck
  - – Asymmetric multicore speedup equals dynamic multicore at higher power constraint
    - – Over-provisioning core resources is the key

Outline

- ❑ Motivation
- ❑ Simultaneously Active Fraction
- ❑ **Over-provisioned Multicore Systems**
  - SAF-aware Multicore design paradigm
  - Fundamental Characteristics
- ❑ Computation Spreading
- ❑ Results
- ❑ Conclusion

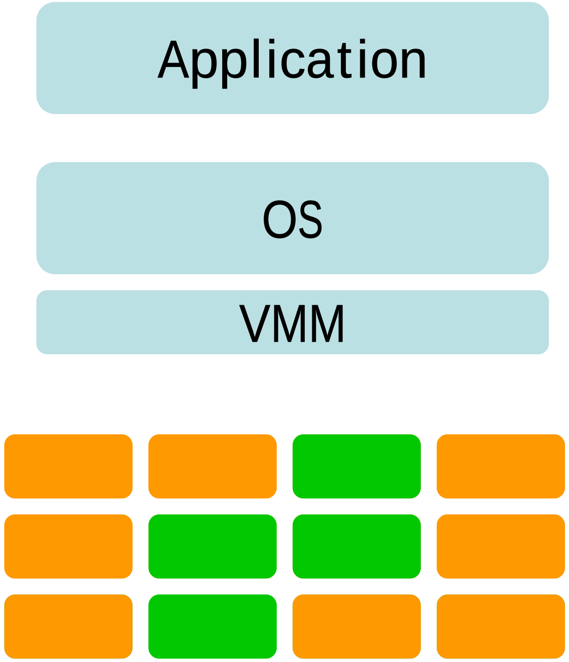
Power Management: SAF reduction

- ❑ **Utilizing more resources requires SAF reduction**
- ❑ **Current Approaches**
  - Clock Gating: save power from unused circuit component
    - Dynamic, but fine grain
  - L2/L3 Caches: low SAF by design
    - Coarse grain, but static
    - Performance does not scale with size
- ❑ **New approaches for SAF reduction**

Dynamic coarse-grain SAF reduction

Over-provisioned Multicore Systems

- Consider SAF Reduction at core granularity
  – ALU:Uni-processor = core:Multicore



OPMS Design Principles

– By design, total cores exceed power budget

– SAF-aware: avoid concurrent computation on all cores

– Flexible computation assignment on cores

  – VMM maintains software transparency

🟩 Inactive 🟧 Active

Technology Invariants in OPMS design

- ❑ **Processing Cores**
  - – Area cost is marginal compared to the cost of powering them up simultaneously
- ❑ **On-chip Communication**
  - – Superior bandwidth between on-chip cores

OPMS: Design Considerations

- **Interfacing with System Software**
  - Constantly varying pool of computation resources
  - Dissertation implements a lightweight VMM component
    - Virtualizes processor resources *only*
    - Software transparent *Computation Transfer (CT)* between on-chip cores
- **Managing Inactive Cores**
- **Flexible Computation Assignment**

Managing Inactive Cores: Cost/Benefit

- ❑ **Benefit**
  - – Retain predictive state
    - – Speed up computation
  - – Reduce thermal load on each core
    - – Avoid *hotspots*
- ❑ **Cost**
  - – Static Power
- ❑ **Remedy: Use circuit techniques**
  - – Sleep transistors based on MTCMOS removes leakage
    - – Design issues: length of inactive periods (> ~100 cycles [Borkar 2003]), no state-retention
  - – Retain state in low leakage drowsy mode [Flautner 2002]

Flexible Computation Assignment

- **Opportunity: More available cores than active**
- **Distribute computation to enhance benefit from predictive structures**
  - Improve execution time and reduce energy consumption
- **Classic Application: Computation Spreading**

Outline

- ❑ Motivation
- ❑ Simultaneously Active Fraction
- ❑ Over-provisioned Multicore Systems
- ❑ **Computation Spreading**
  - Multithreaded Server Application
  - General Case and specific application
  - Implementation
- ❑ Results
- ❑ Conclusion

CSP: Overview

- ❑ **Multithreaded Server Application**
  - – Extensive code reuse among on-chip processor cores
  - – Poor utilization of private resources
- ❑ **Computation Spreading (CSP)**
  - – Collocate similar computation fragments from different threads on the same core
  - – Distribute dissimilar computation fragments from same thread onto different cores

▶

CSP: Design Considerations

- **Dynamic Specialization**
  - Mutually exclusive code fragments
- **Preserving Data Locality**
  - Different computation fragments may share data
- **Fragment Size**
  - Amortizing computation transfer cost
- **Core Contention**
  - Different fragments may be assigned to the same core

Implementation

- **OS and User computation**
  - Satisfies all fragment selection objectives ▶
  - Server apps spend significant time in OS mode
- **Core provision**
  - Provision some cores for running user code, rest for OS code
  - VMM perform CT on mode transfer
  - OPMS mitigates core contention
- **Assignment Policies**
  - Thread Assignment Policy (TAP)
    - Maintain VCPU to core mapping
  - Syscall Assignment Policy (SAP)
    - Maintain system call to core mapping for OS computation

Outline

- □ Motivation
- □ Simultaneously Active Fraction
- □ Over-provisioned Multicore System
- □ Computation Spreading
- □ **Results**
- □ Conclusion

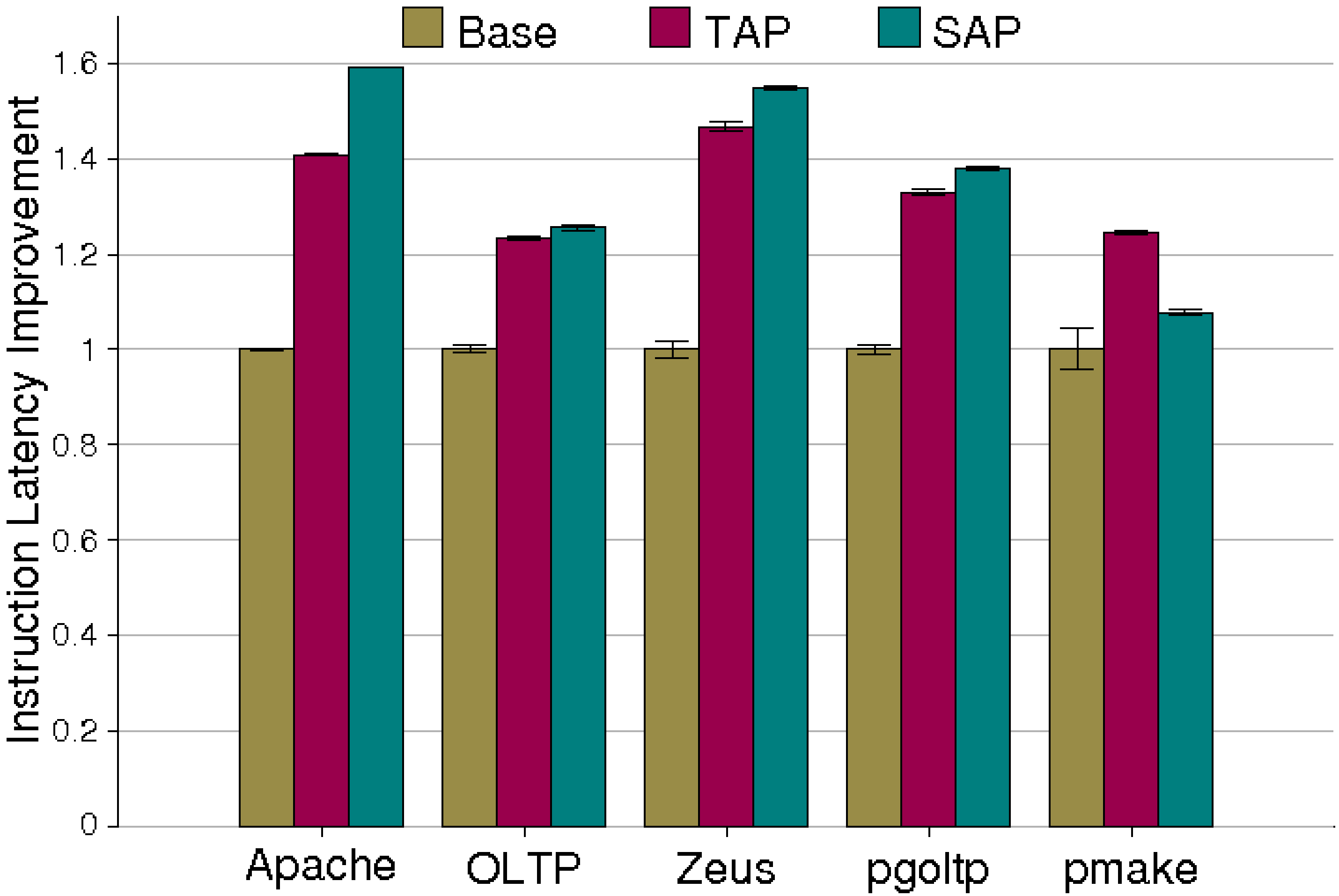
Methodology

- ❑ **SIMICS based full system simulation**
- ❑ **Energy estimation: Wattch and HotSPOT**
  - – Thermal model used to calibrate power estimation
  - – 32nm technology generation, 0.9V, 3.0GHz
- ❑ **Unmodified Application running on Solaris 9**
- ❑ **Out-of-order cores**
- ❑ **Performance Comparison**
  - – Baseline System: 8 cores, 16MB shared L2
  - – OPMS: 12 cores, 12MB shared L2
  - – Invariants: Power and Area

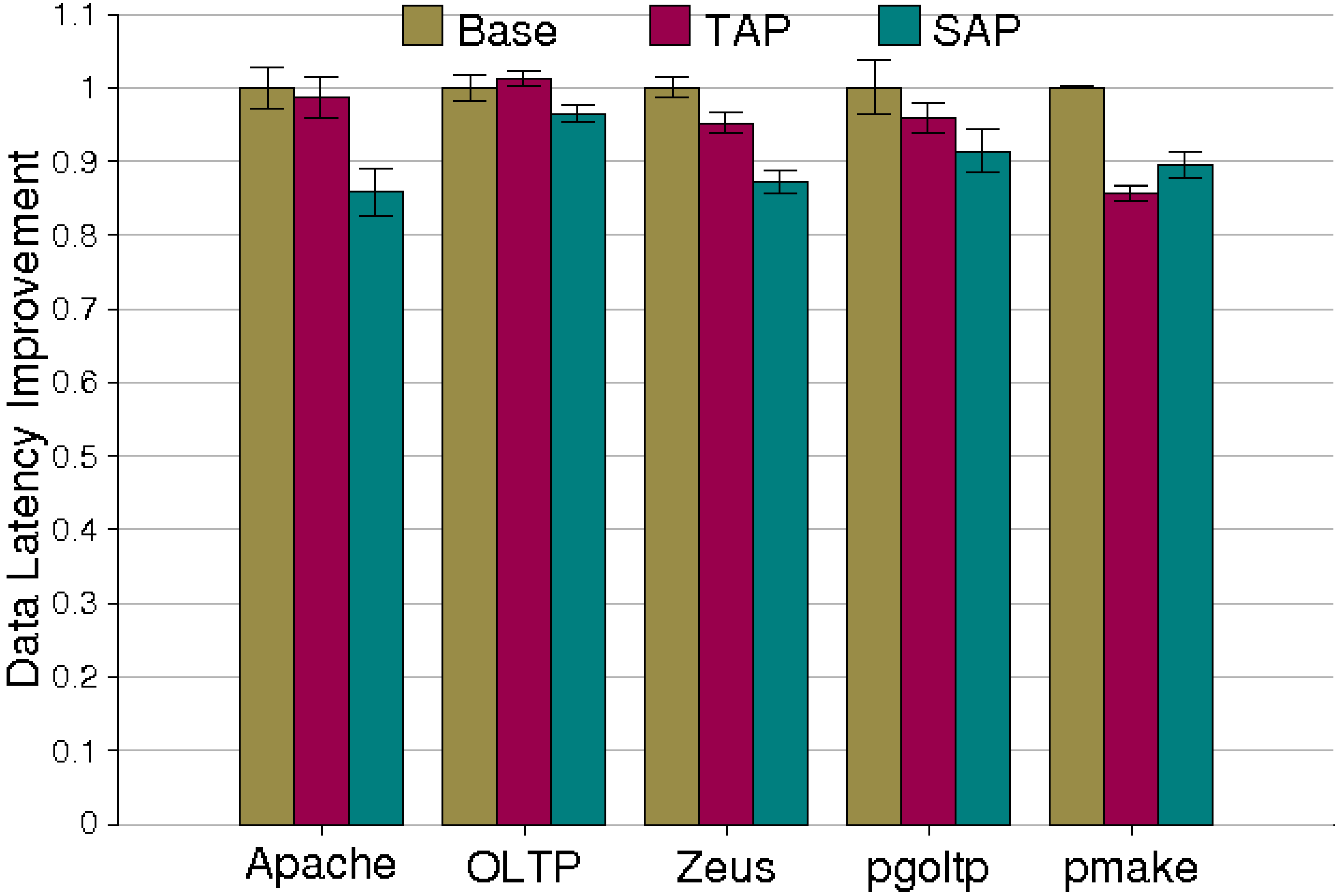
Results

- ❑ **Locality Impact**
  - – Memory references: instruction and data
  - – Performance impact
- ❑ **Energy Efficiency**
  - – Core utilization, energy savings, energy-delay
- ❑ **Sensitivity Analysis**
  - – 12-core system fully utilized at all times

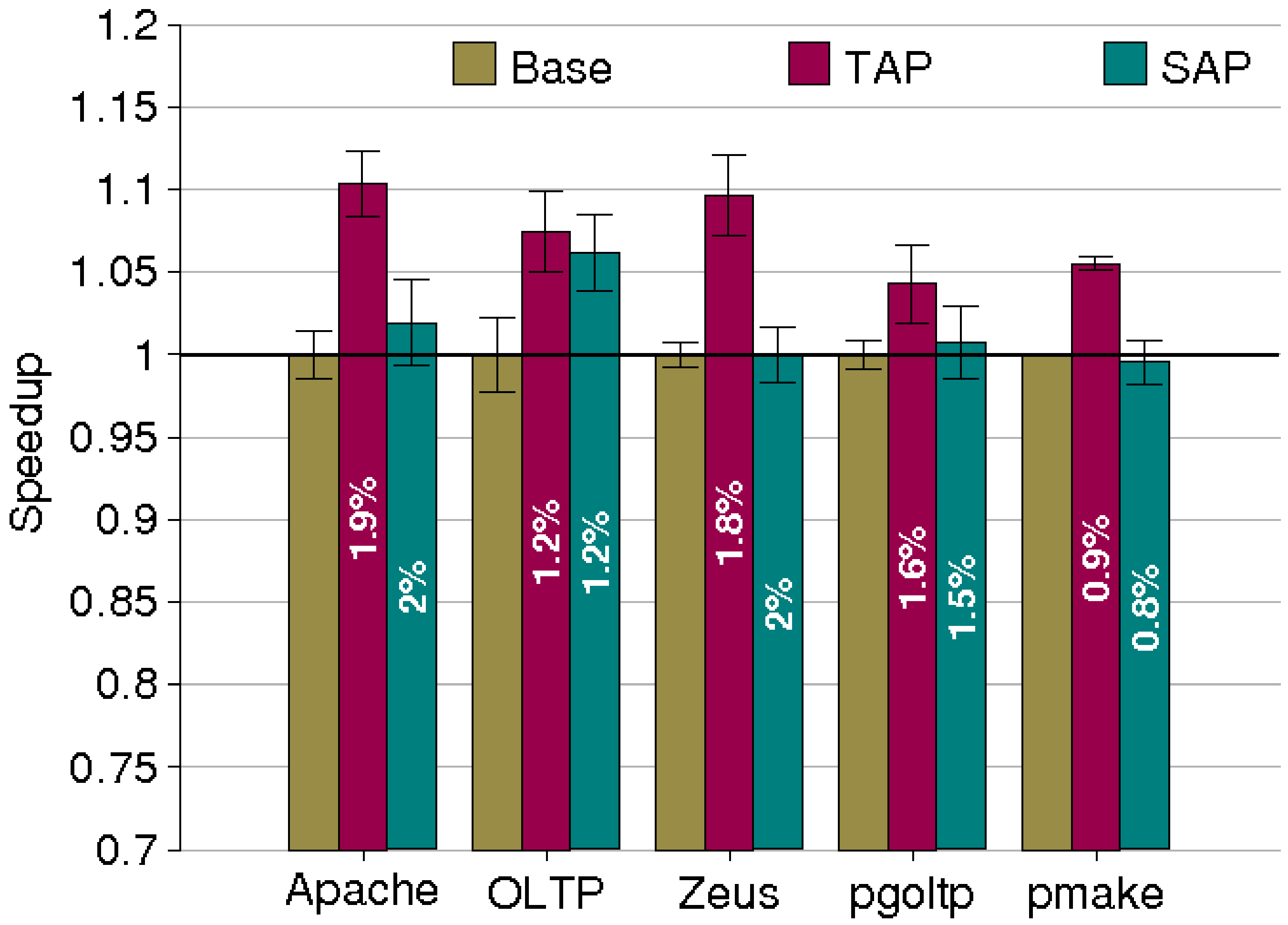
Instruction Latency Improvement



Data Latency Improvement



Performance

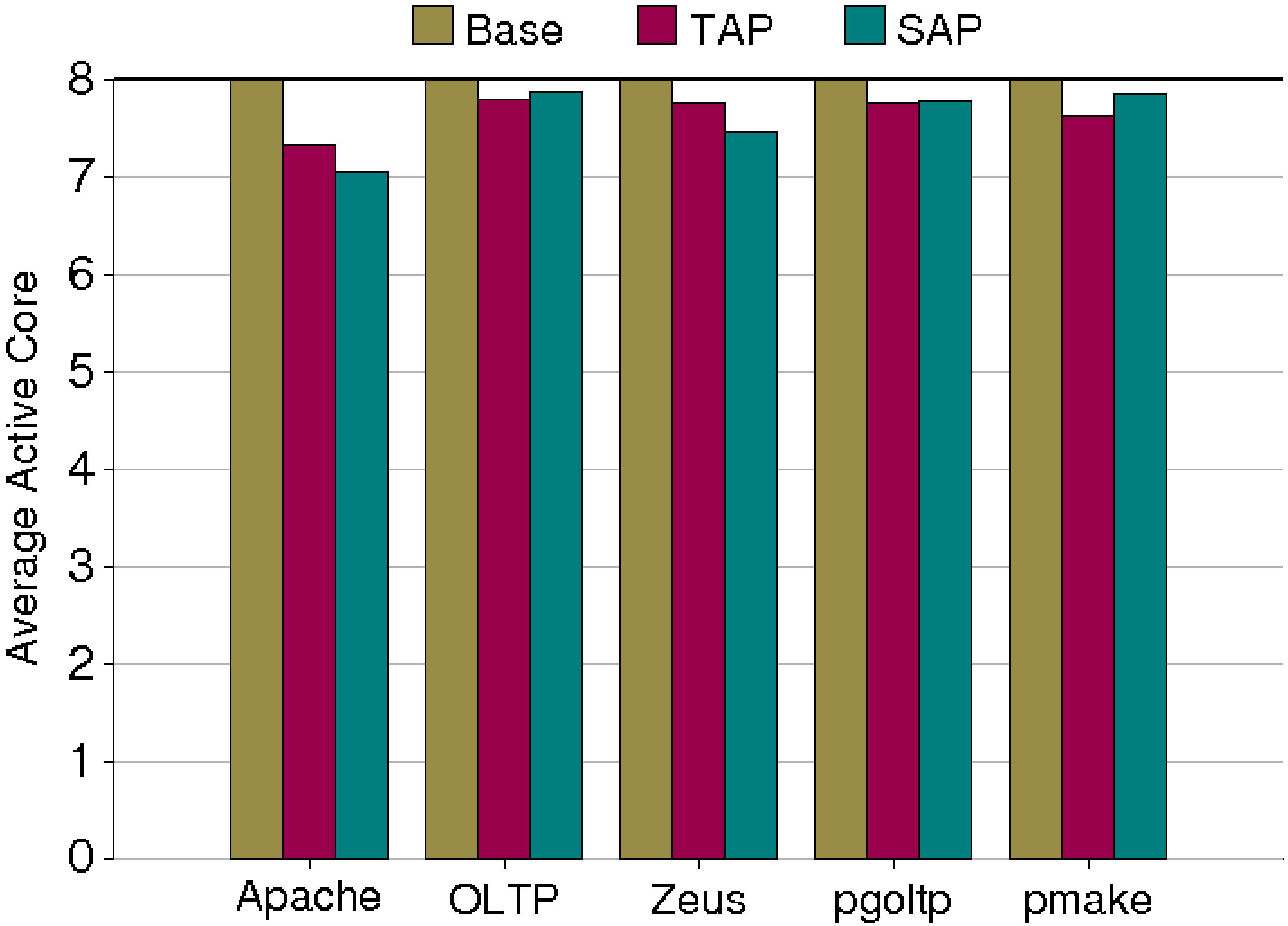


CT Overhead: percentage runtime spent in performing CT

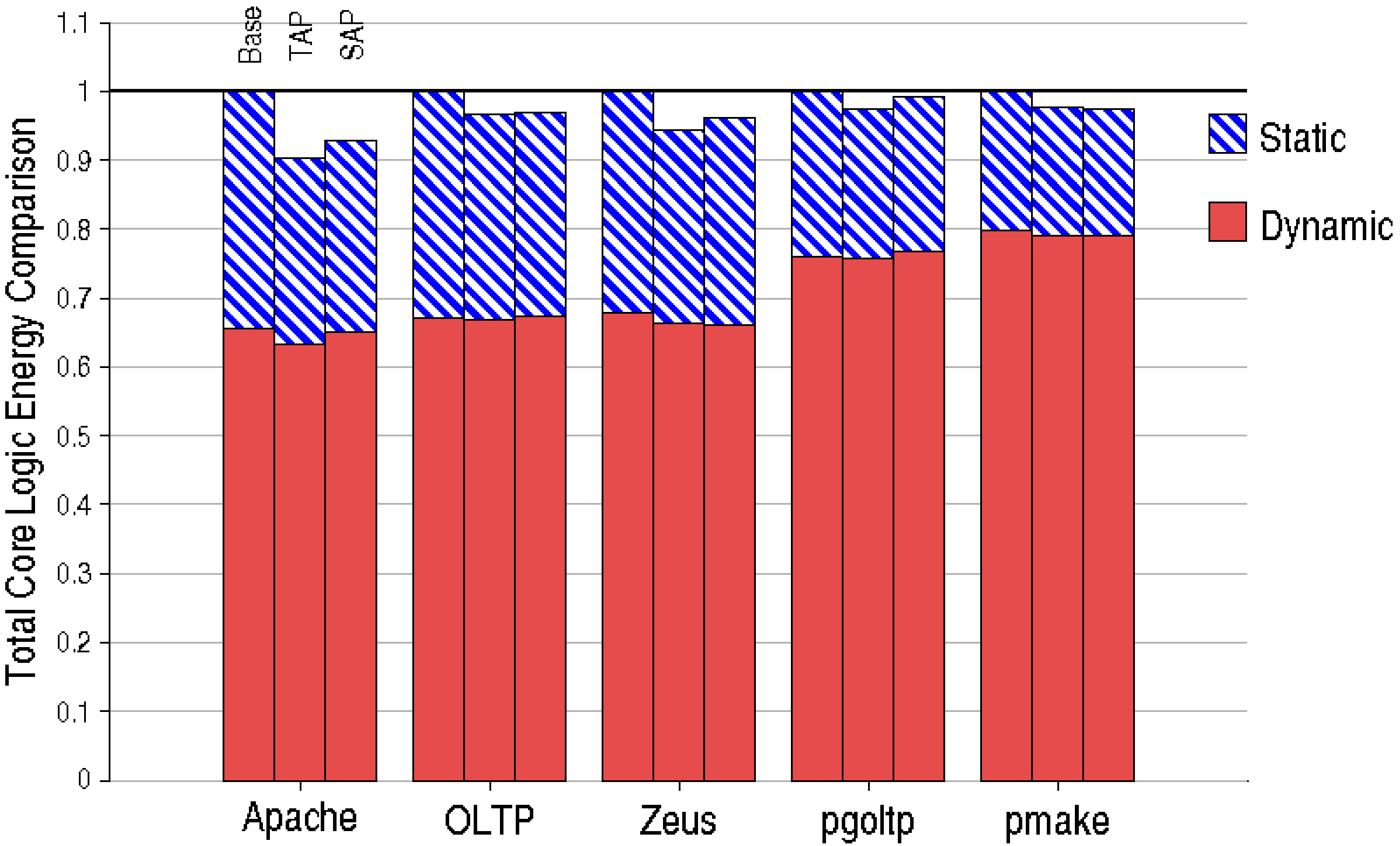
Energy Efficiency

- **OPMS employs 12 cores instead of 8**
  - Cores engaged in computation largely determine SAF/power
  - Partial reduction in active cores can allow several inactive cores to subsist within the same power envelope
- **Impact of better compute efficiency**
  - Runtime reduction will save leakage energy
  - Lesser access in shared L2 saves active energy
  - Energy-delay improvements from savings in energy *and* delay

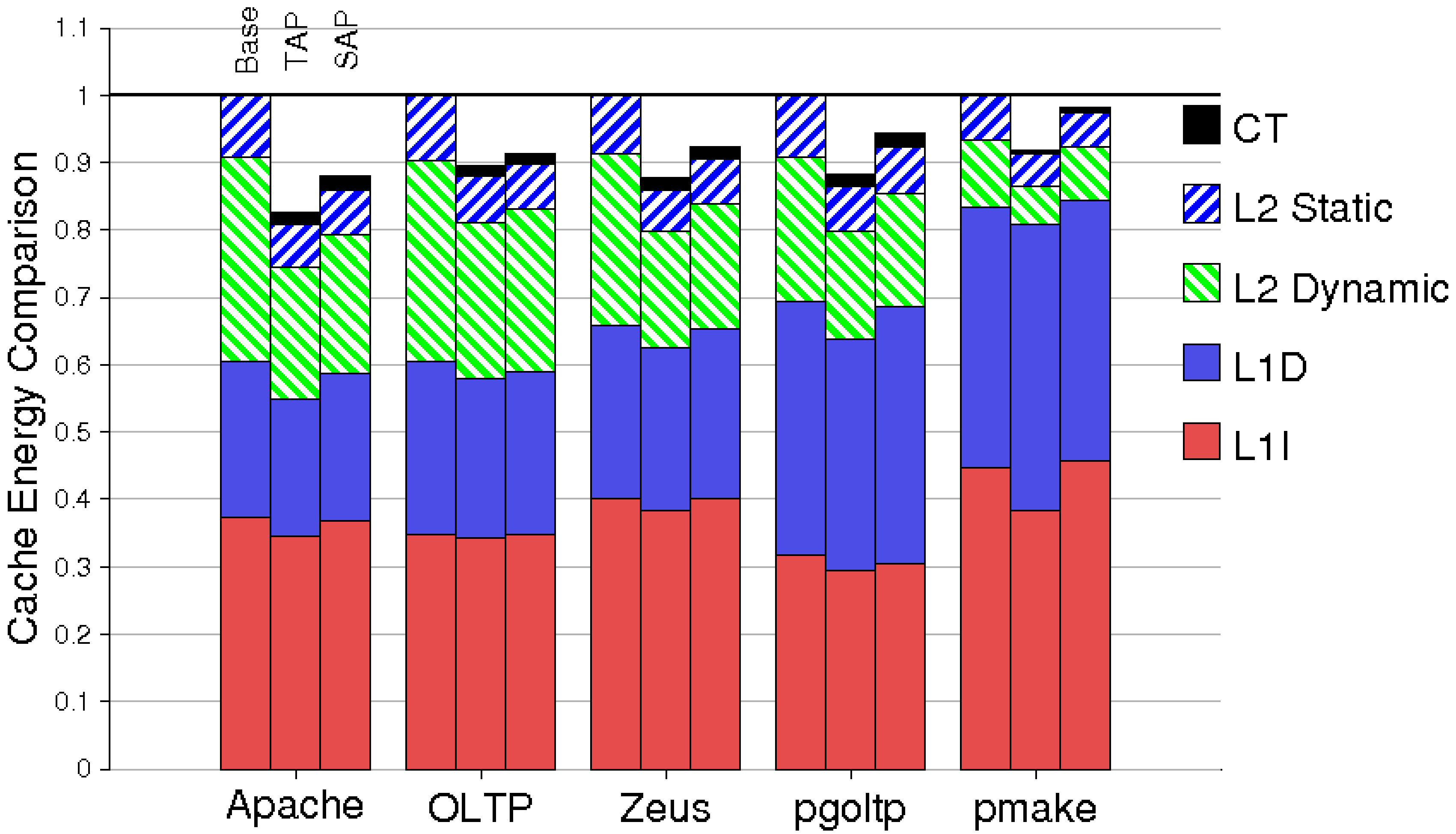
Active Cores



Total Core Logic Energy Comparison



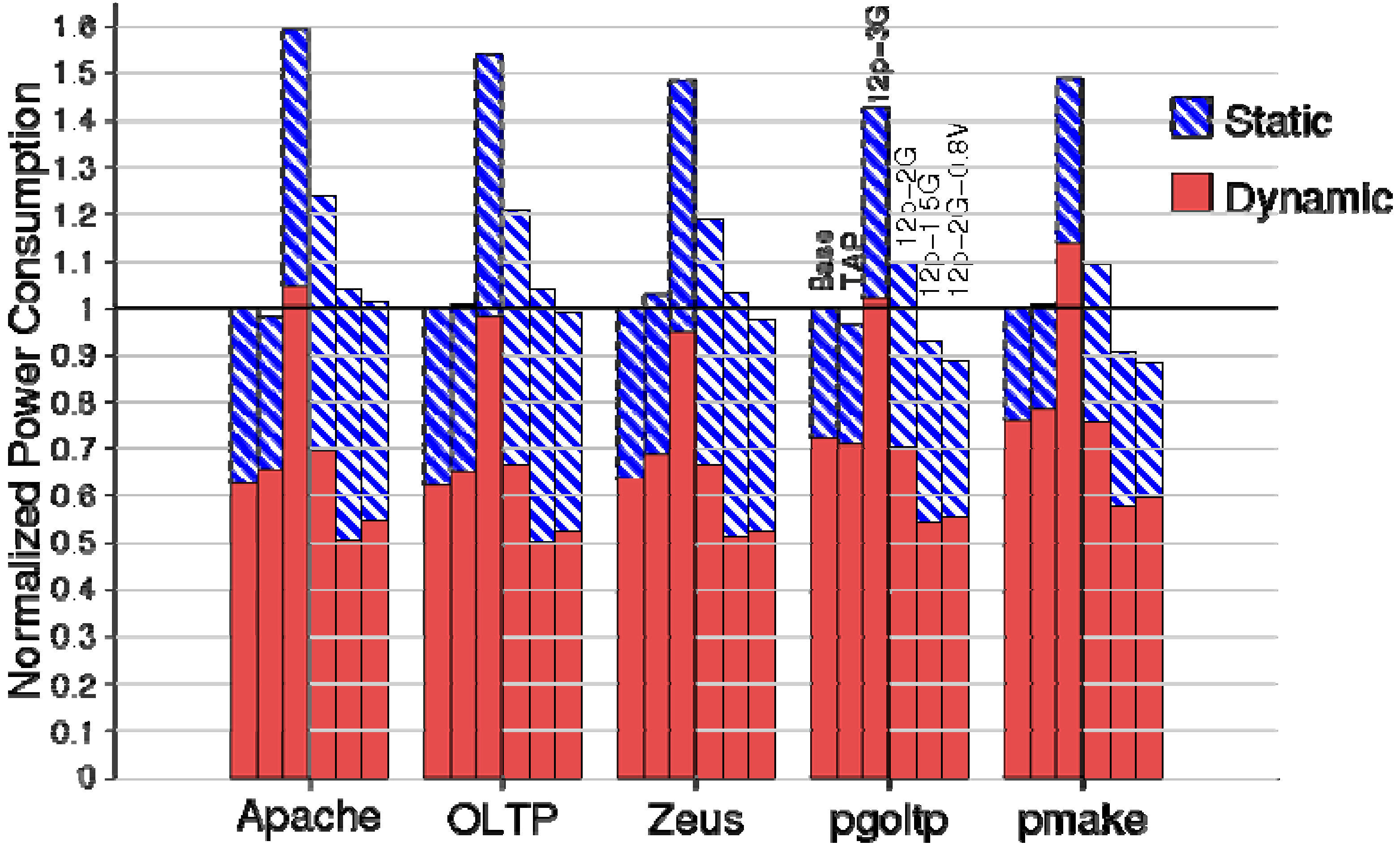
Cache Energy



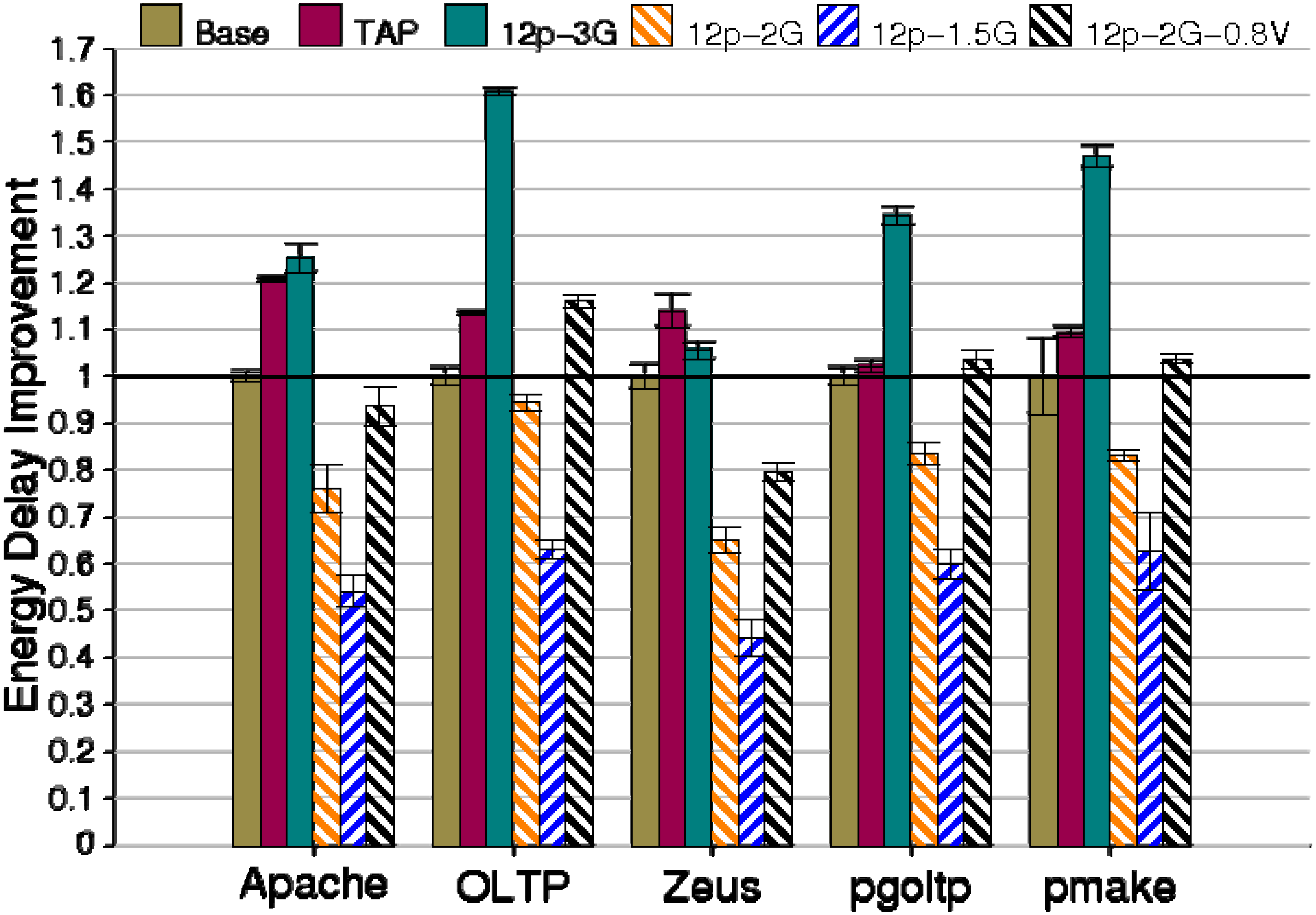
Comparative Study with 12-core

- ❑ **Improvements in performance and energy efficiency in OPMS**
  - – But, OPMS employs different micro-arch (12 cores)
  - – What if the same micro-arch exploits more threads?
- ❑ **Exploiting app. concurrency on 12-core system**
  - – Will exceed the baseline power budget
- ❑ **Apply frequency scaling to reduce power**
  - – Voltage scaling is unlikely at this design point, but results will show its impact
- ❑ **Methodological challenge from differing system configs**
  - – Longer simulation runs to alleviate transient effects

Power Comparison



Energy Delay Improvement



Outline

- □ Motivation
- □ Simultaneously Active Fraction
- □ Over-provisioned Multicore System
- □ Computation Spreading
- □ Results
- □ **Conclusion**
  - Related and Summary

Related Work

- ❑ **OPMS**
  - – Power Reduction [several]
    - – Dynamic voltage frequency scaling
  - – Activity Migration
    - – Heat and Run [Powell 2004], AM [Barr 2003]
- ❑ **Computation Spreading**
  - – Software re-design: staged execution
    - – Cohort Scheduling [Larus and Parkes 01], STEPS [Ailamaki 04], SEDA [Welsh 01], LARD [Pai 98]
  - – OS and User Interference [several]
    - – Structural separation to avoid interference

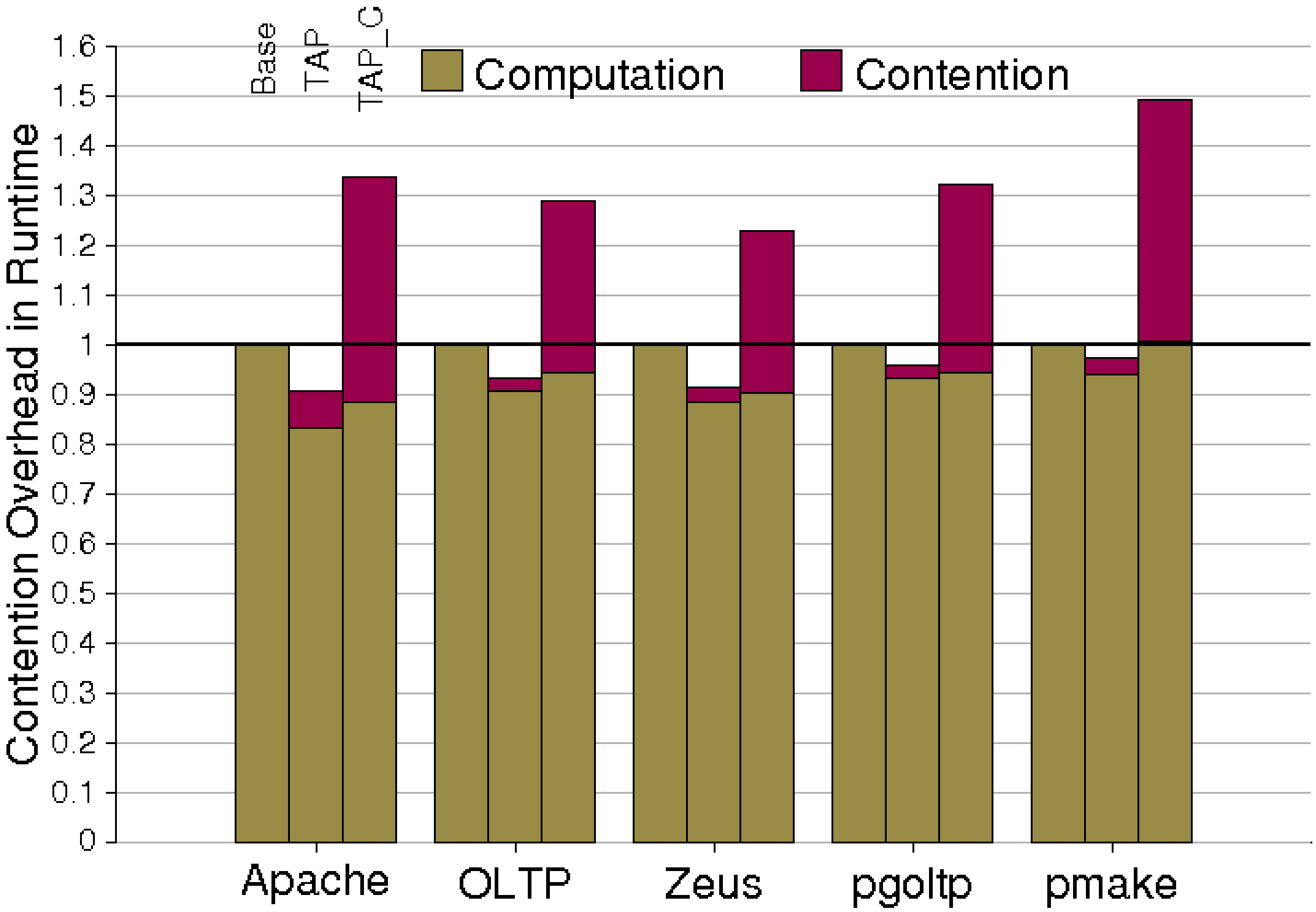
Summary of Contributions

- ❑ **Simultaneously Active Fraction**
  - – Models first order impact of power constraint in architectural design
  - – Technology trends indicate diminishing SAF in future chips
  - – Demonstrates reasoning with SAF in multicore designs
- ❑ **Over-provisioned Multicore Systems**
  - – SAF-aware paradigm of multicore designs
  - – Versatile framework enabling flexible computation assignments
- ❑ **Computation Spreading**
  - – Dynamic specialization of on-chip cores in an OPMS
  - – Energy-efficiency and performance without demanding more power

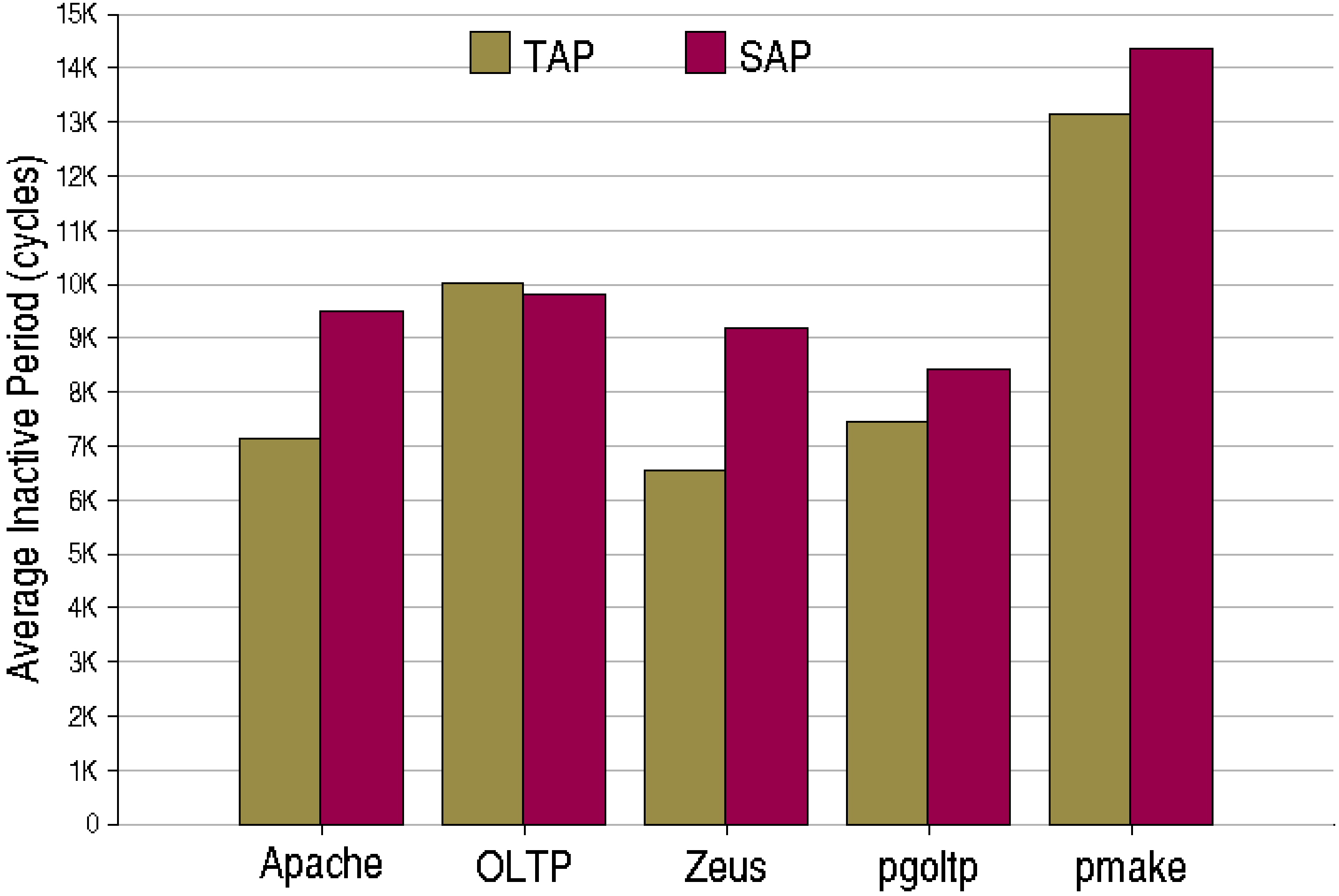
Thank You!

http://www.cs.wisc.edu/~kchak

Contention Overhead

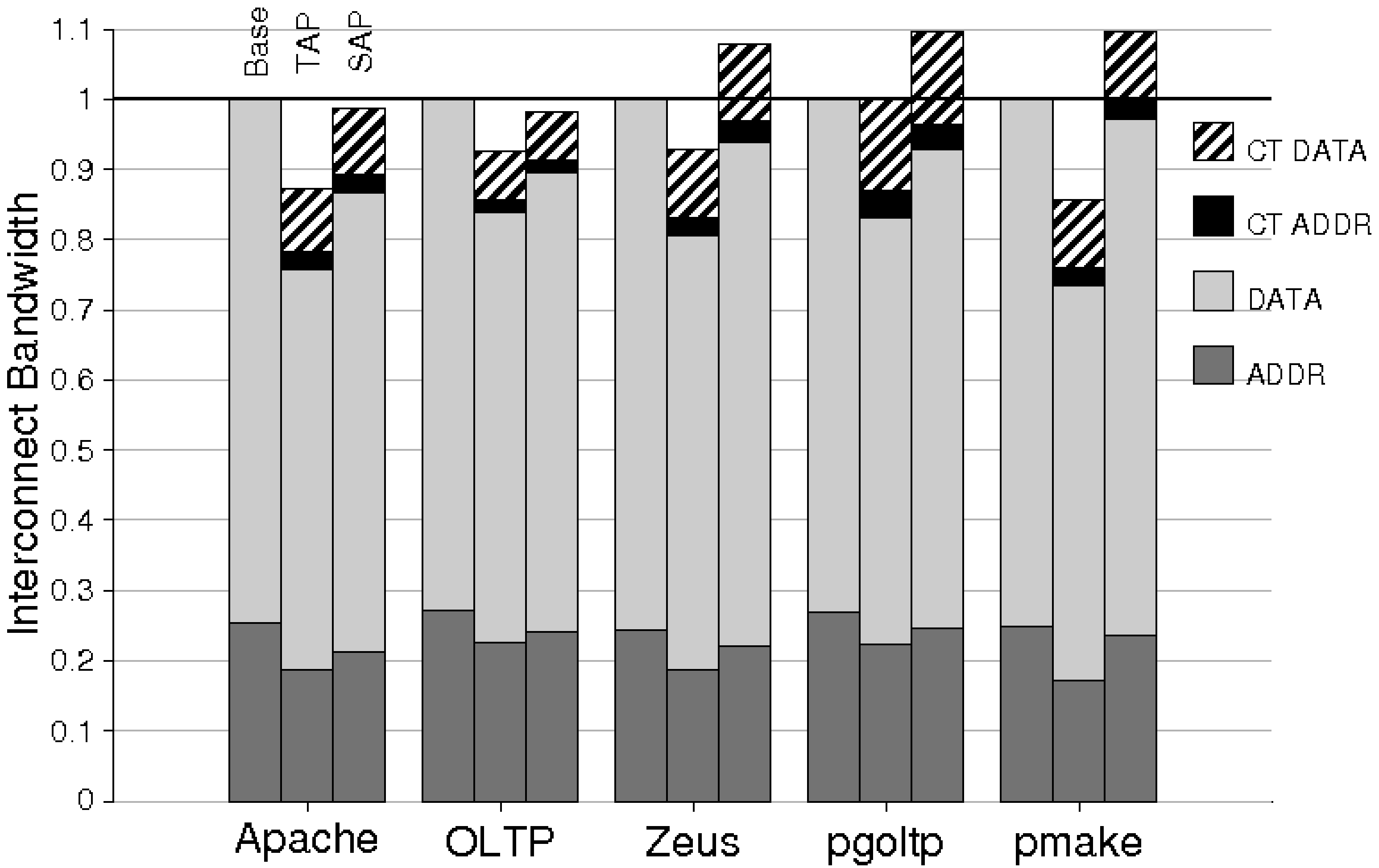


Inactive Periods

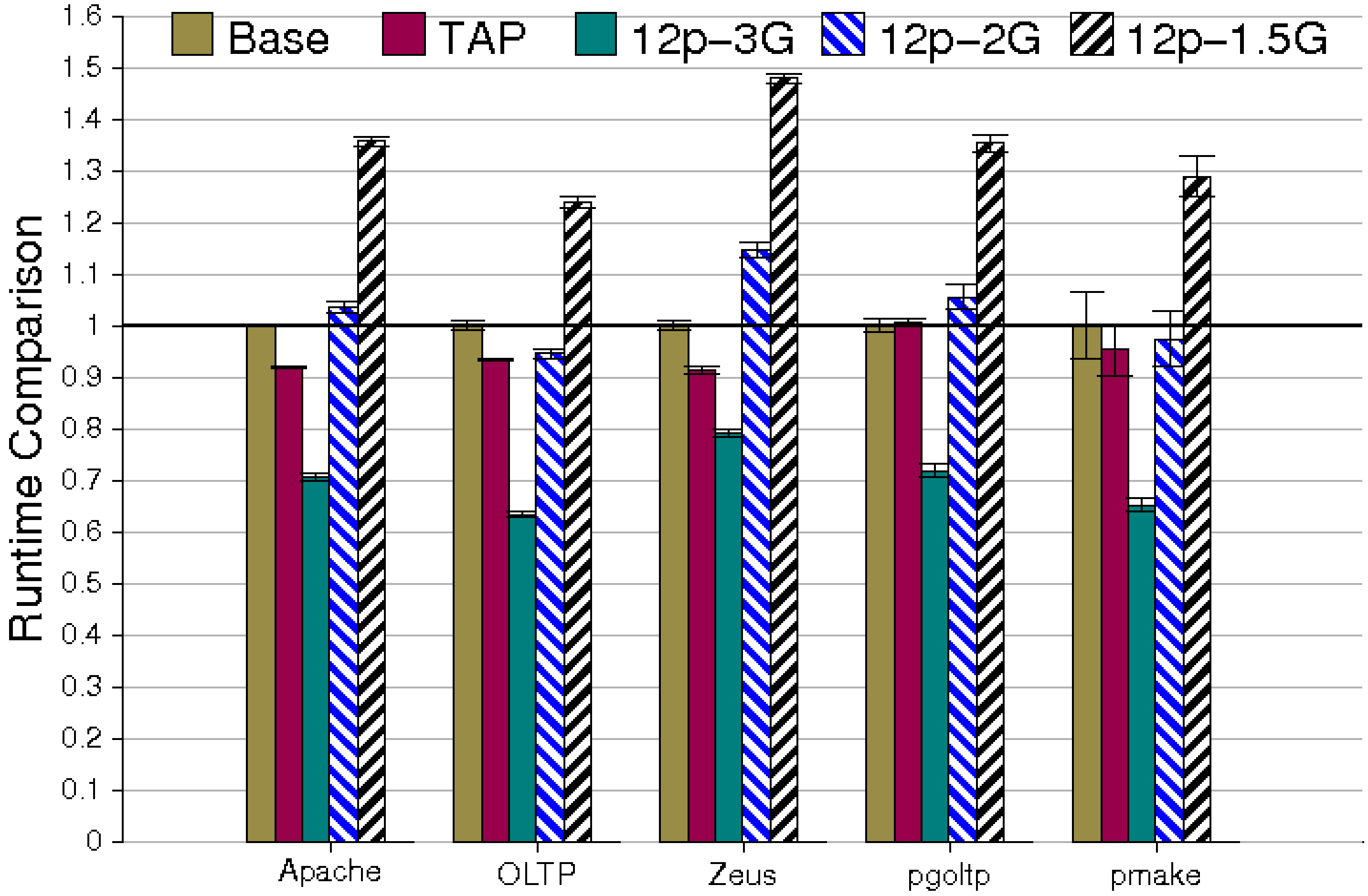


Long inactive periods allow very efficient leakage reduction

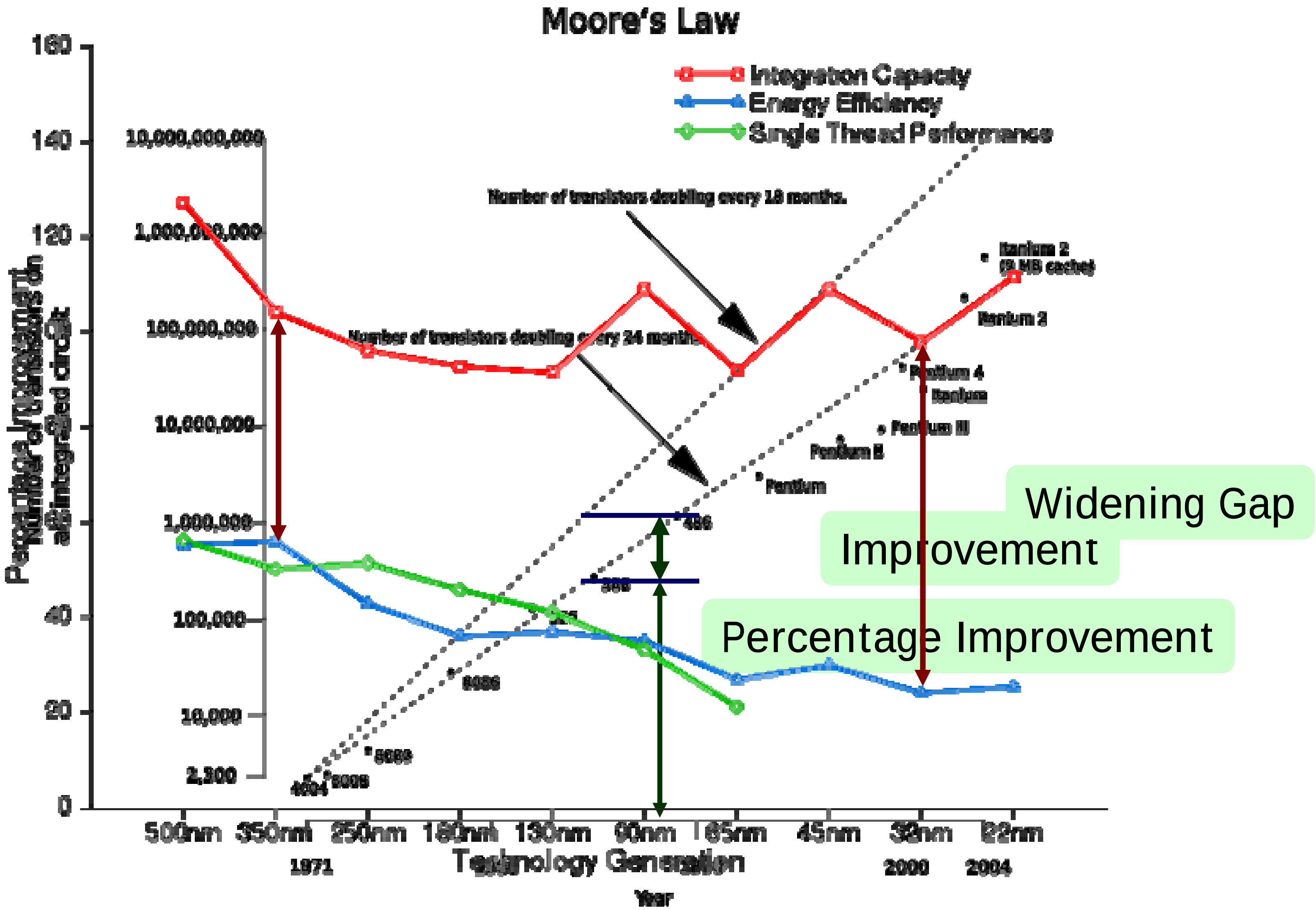
Interconnect Bandwidth



Runtime


Base
TAP
12p–3G
12p–2G
12p–1.5G
Runtime Comparison
1.6
1.5
1.4
1.3
1.2
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Apache
OLTP
Zeus
pgoltp
pmake

Multicore Evolution



Widening Gap

Improvement

Percentage Improvement

SAF

SAF

Current Multicore

Core Logic contributes 75% of power

– 30% area: 7 ? ower

5-10 years

Technology Trend:

– Improvement in power lagging improvement in *effective area*

Today (Tulsa: Intel Xeon)

SAF will shrink with technology scaling

OPMS: The Next Step Ahead

Conventional Multicore aims simultaneous computation on all cores

OPMS Design Principles

– By Design, total cores exceed power budget

– Forgo concurrent computation on all cores

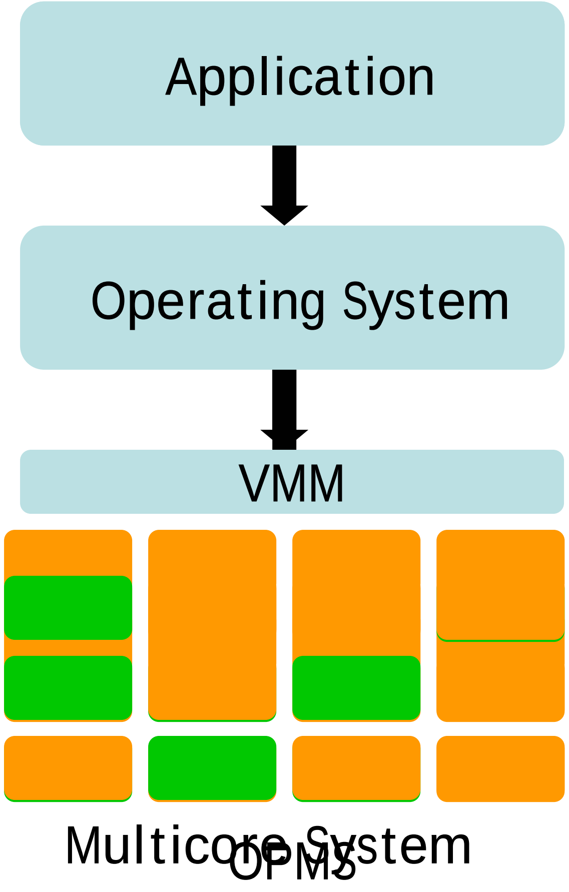
– Flexible computation assignment on cores

  – VMM maintains software transparency

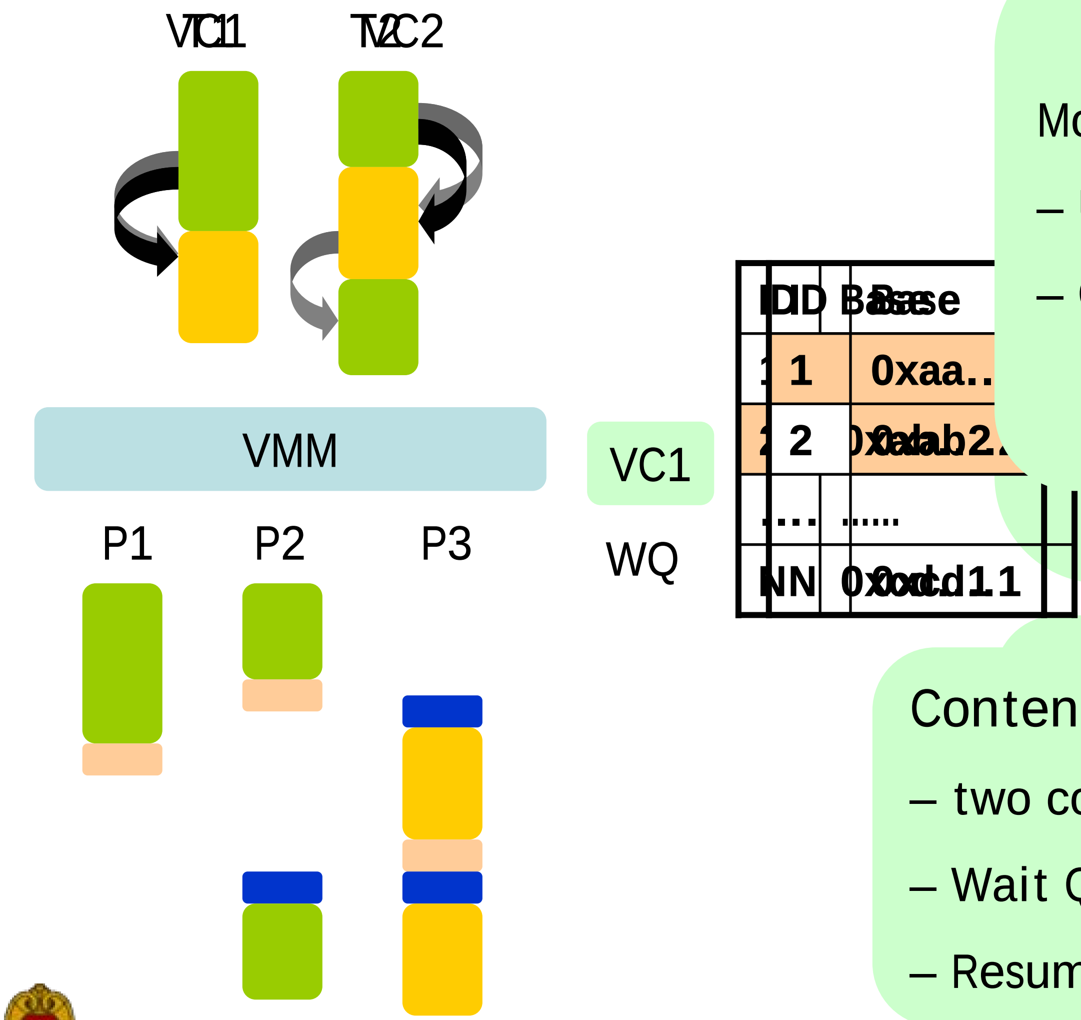
OPMS Design: What, How and Why ?

🟩 Inactive

🟧 Active



Implementation



– OS assigns two threads T1. T2

Mode transfers:

– User to OS: system calls, traps

– OS to User: returns

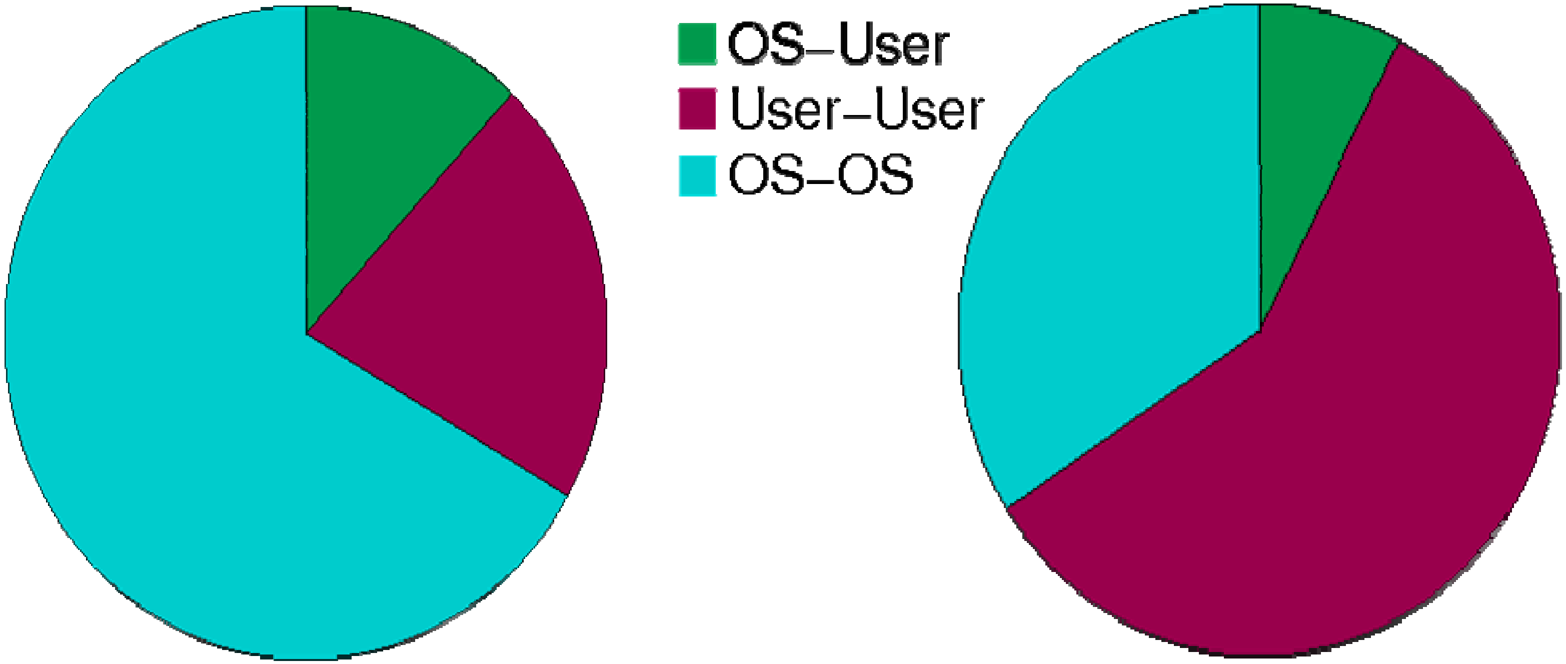
Contention

– two computation assigned to same core

– Wait Queue is populated

– Resumes computation when available

OS-User Data Communication



Apache

OLTP

OS-User Communication is limited

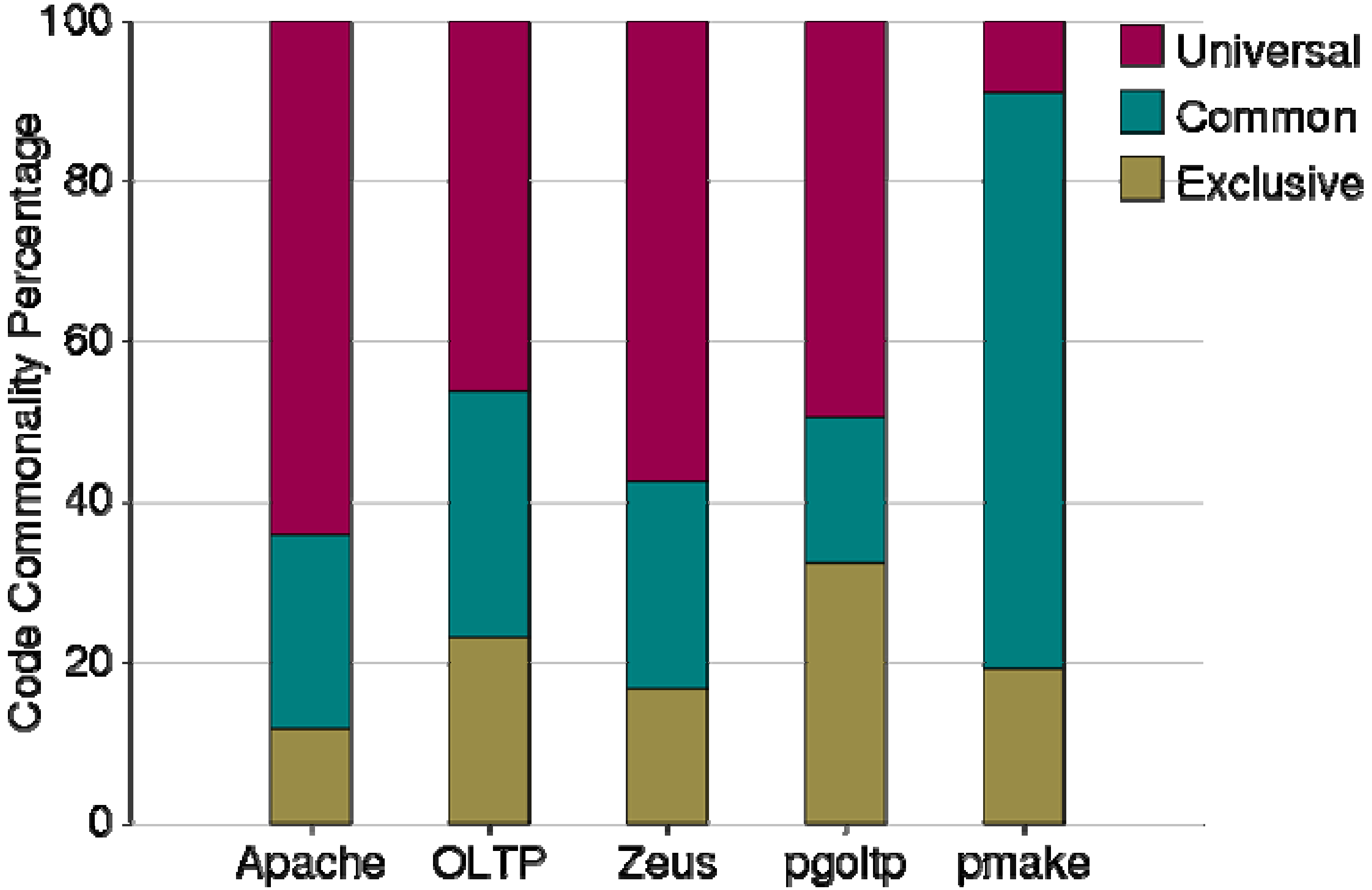
CSP: Longer

Multithreaded Server Application

- ❑ **Important class of multicore applications**
- ❑ **Memory stalls are #1 performance bottleneck**
  - Memory stall = instruction stall + data stall
  - Substantial instruction stalls from large code footprint
- ❑ **Software architecture**
  - Each server thread services one client request
  - Individual thread assigned to individual core

Extensive code reuse

Multicore Code Reuse

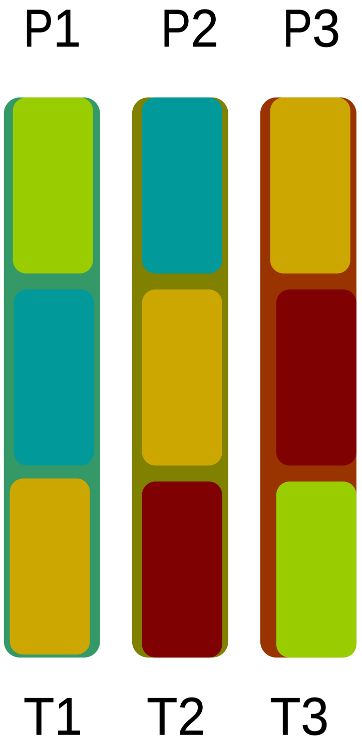


Exploiting Code Reuse

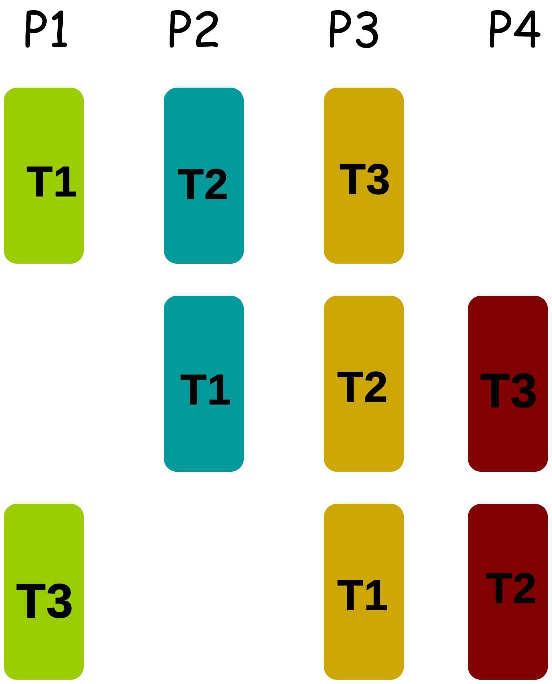
- ❑ **Lack of instruction stream specialization**
  - Redundancy in predictive state and poor capacity utilization
  - Destructive interference
- ❑ **No synergy among multiple cores**
  - Lost opportunity for co-operation
- ❑ **Computation Spreading (CSP)**
  - Collocate similar computation fragments from multiple threads
  - Distribute dissimilar computation fragments from a single thread

Example

Conventional Multicore



CSP



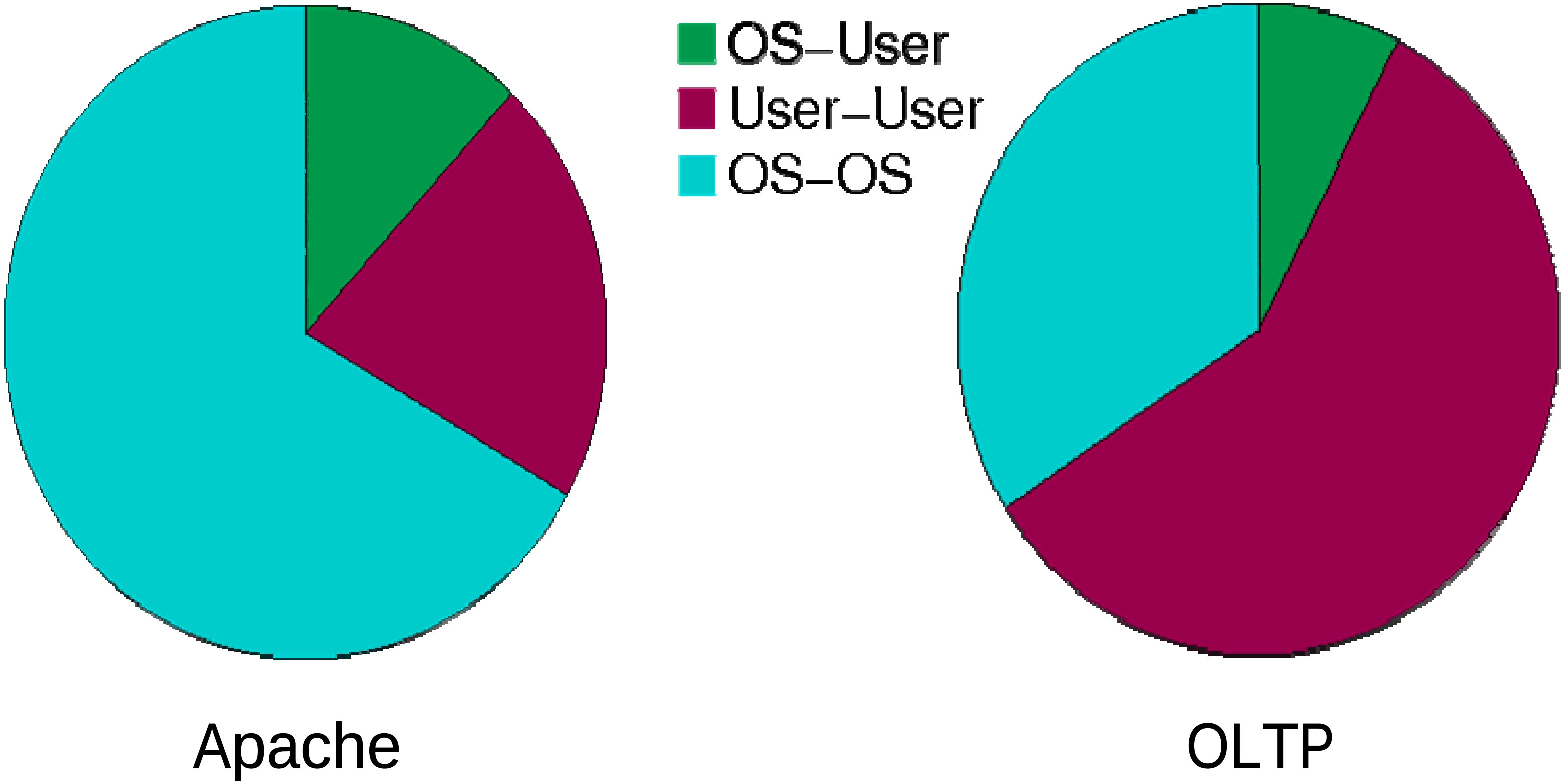
CSP: Design Considerations

- ❑ **Dynamic Specialization**
  - – Mutually exclusive code fragments
- ❑ **Preserving Data Locality**
  - – Different computation fragments may share data
- ❑ **Fragment Size**
  - – Amortizing computation transfer cost
- ❑ **Core Contention**
  - – Different fragments may be assigned to the same core

OS and User Computations

- ❑ Coarse grain computation fragments
- ❑ Exercise mutually exclusive code
- ❑ Limited data communication

OS-User Data Communication

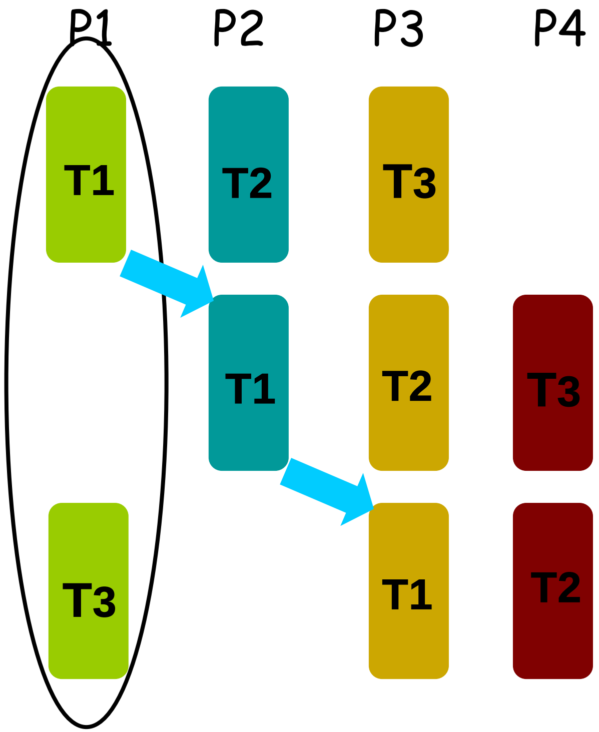


OS-User Communication is limited

Implementation

- ❑ **OS and User computation**
  - Satisfies all fragment selection objectives
  - Server apps spend significant time in OS mode
- ❑ **Core provision**
  - Provision some cores for running user code, rest for OS code
  - VMM perform CT on mode transfer
- ❑ **Assignment Policies**
  - Thread Assignment Policy (TAP)
    - Maintain VCPU to core mapping
  - Syscall Assignment Policy (SAP)
    - Maintain system call to core mapping for OS computation

CSP: Key Aspects



Dynamic Specialization and Heterogeneity

Moving Computation to Data

Inactive cores maintain predictive state

– Computation fragments are localized

Selecting Fragments: Key objectives

– Heterogeneity derived from structurally identical cores

– Mutually exclusive code fragment

Computation Transfer (CT)

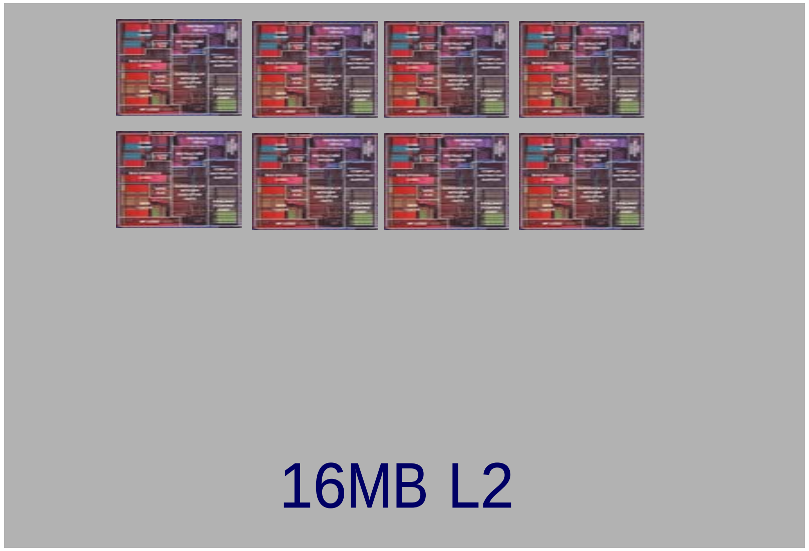
– Relatively coarse to amortize CT

– Independent data footprint as much as possible

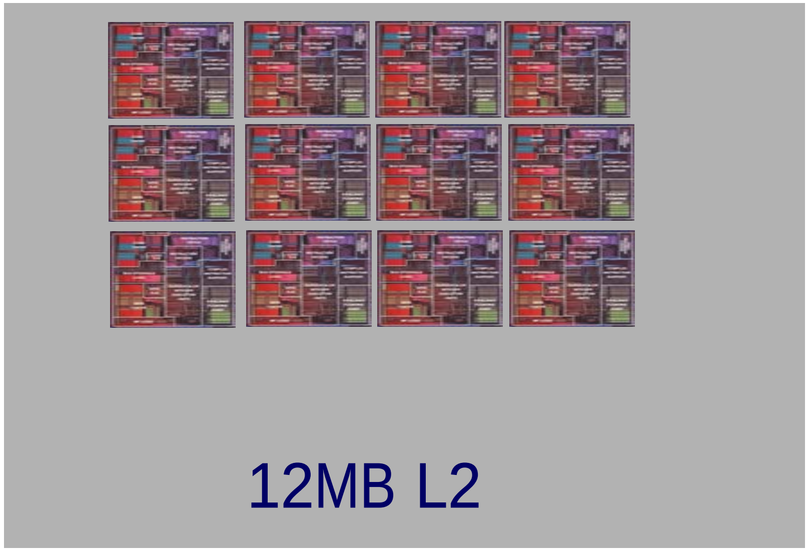
– Enabling mechanism: transfer necessary state between on-chip cores

Performance Comparison

- ❑ **Invariants: Area and power budget**



Baseline *



OPMS *

- ❑ OPMS Schemes: Core Hopping (CHP) and Computation Spreading (CSP)
- ❑ Full System simulation using SIMICS
  - – Unmodified server apps running on Solaris

* Figures do no represent floorplan

Branch Prediction Improvements

Future Work

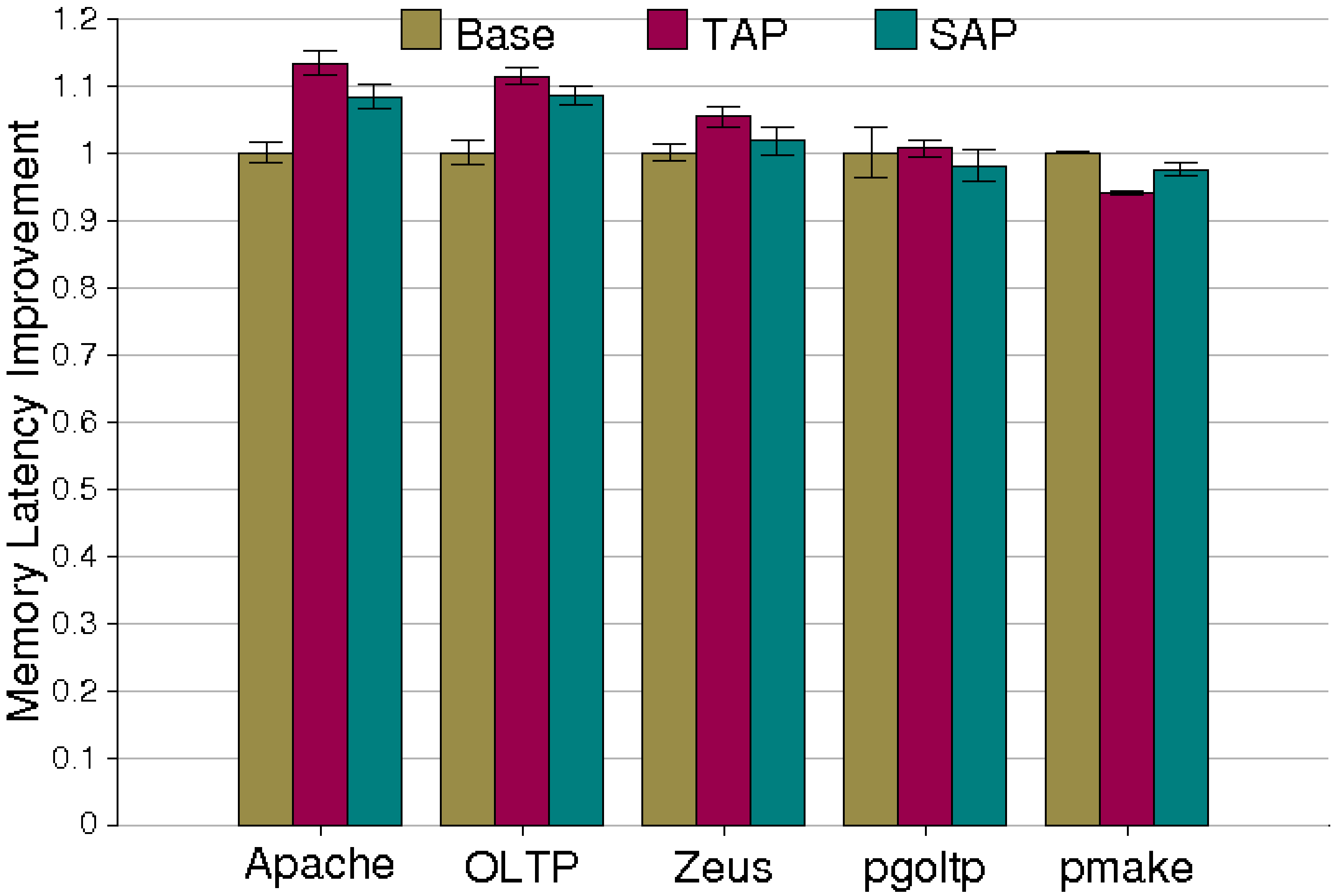
❑ **Managing Heterogeneity**

- Need for energy efficiency push towards specialization
- Both static and dynamic heterogeneity will co-exist
- How can we engage application developers and compilers?
  - Abstract model and interface

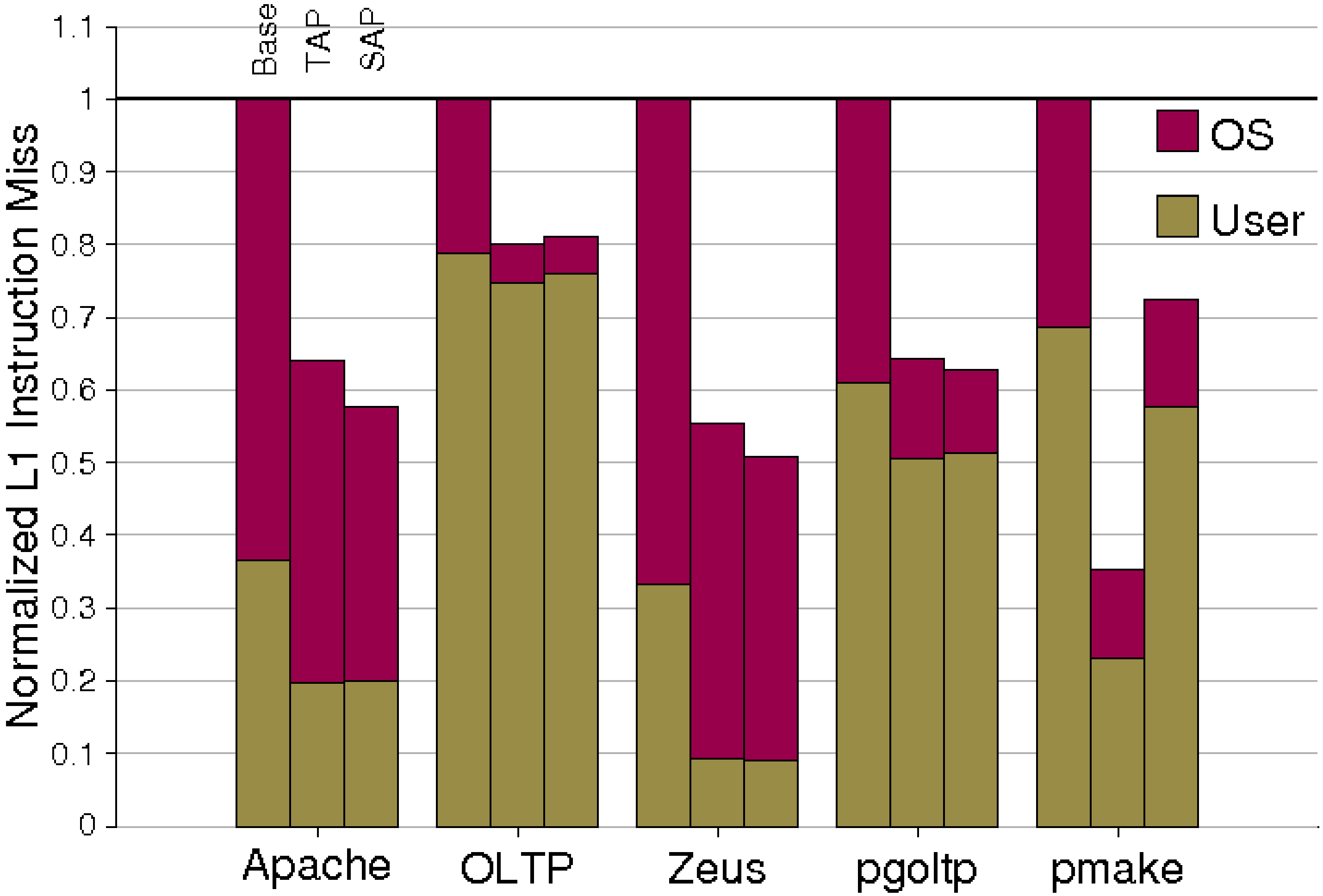
❑ **Bridging general purpose and mobile architecture**

- Mobile: sophisticated software with diverse requirements
- Holistic approach breaks the separation of s/w and h/w
  - Managing complexity will become infeasible
- Requires abstraction for developing complex software

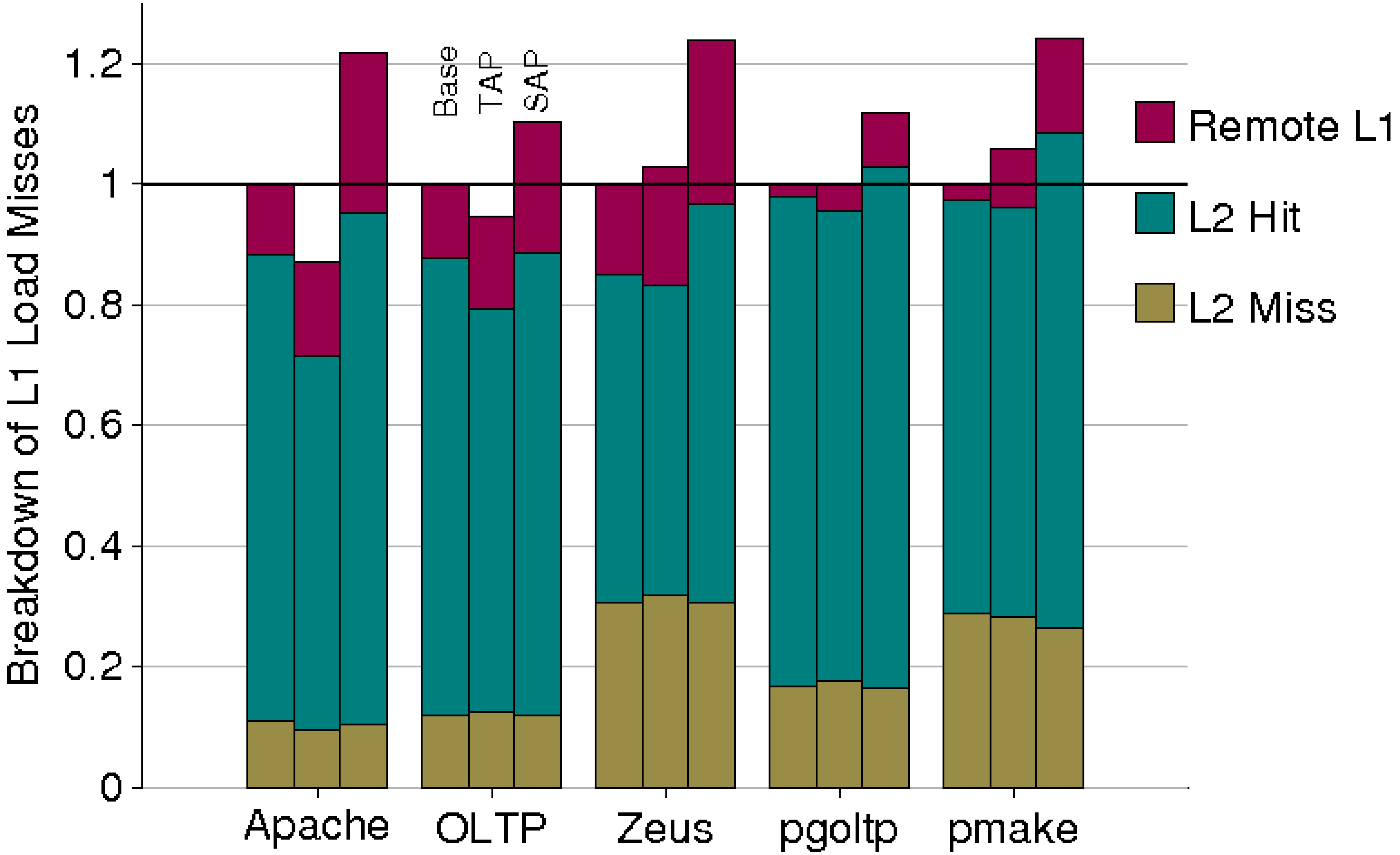
Memory Latency Improvements



L1 Instruction Miss Comparison

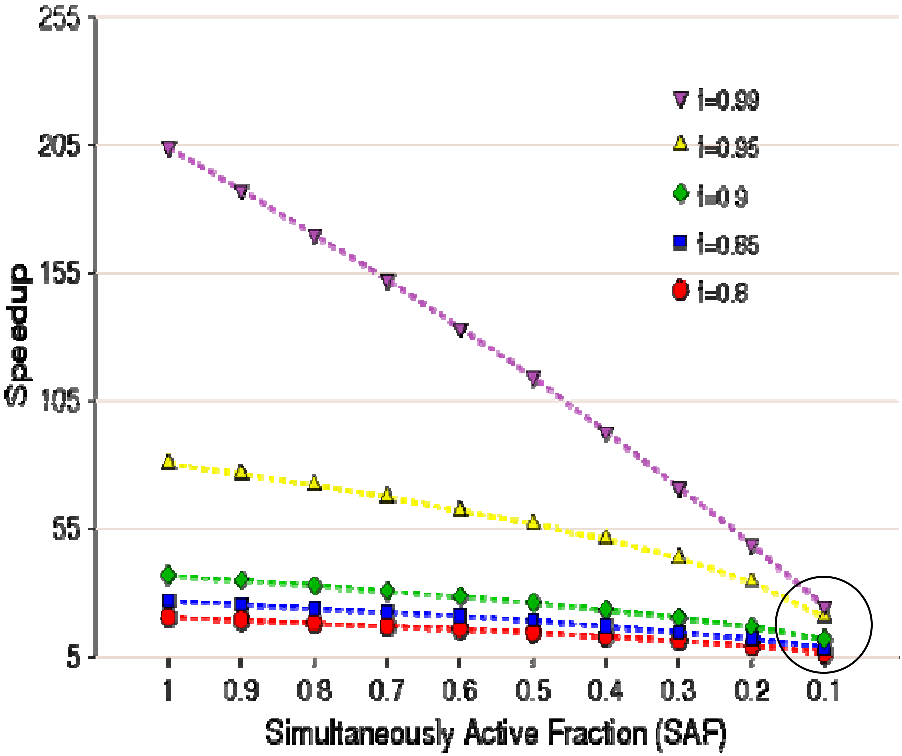


L1 Load Miss Breakdown

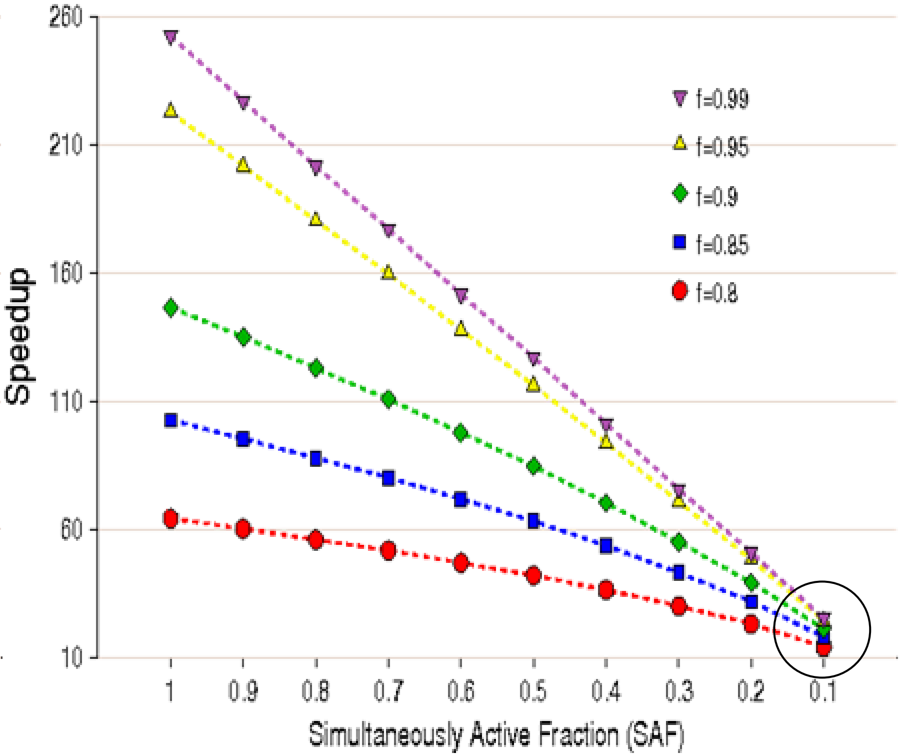


SAF Speedup: Symmetric and Dynamic

Diminishing perfomance gap at higher power constraint



Symmetric Multicore



Dynamic Multicore