Energy Efficient Watchpoint Systems

MAD Workshop 15.11.2013

Vasileios Karakostas, Sasa Tomic, Osman Unsal,
Mario Nemirovsky, Adrian Cristal
Barcelona Supercomputing Center

Background on Watchpoints

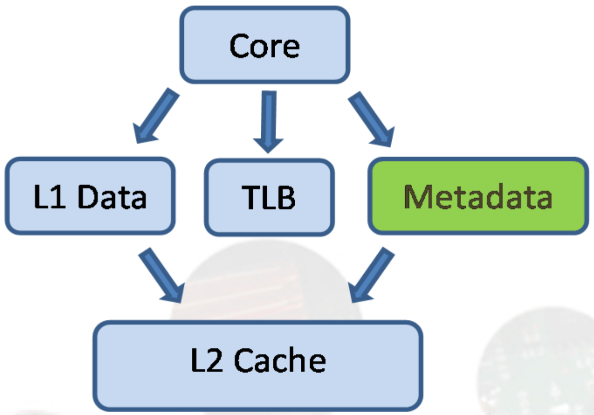
- What is a watchpoint?
  - Memory **range**
  - Access **rights**
  - **User-level** exception handler
- Why we need them?
  - Catch memory **bugs** [HPCA 2007]
  - **Multi-module** software engineering [ISCA 2010]
  - Accelerate **analysis** tools [ASPLOS 2012]
- Why hardware support?
  - **Always-on** execution in **production** runs

Motivation

- Hardware support
  - Cache & check watchpoint access rights
  - Metadata-cache $\rightarrow$ where and when?

Motivation

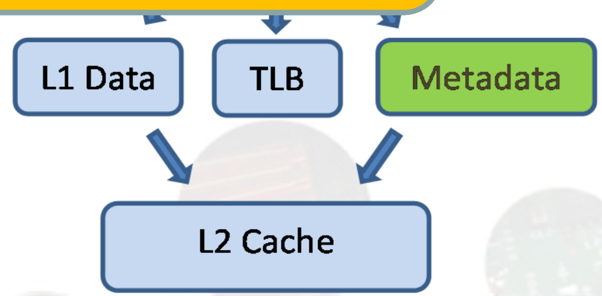
- Hardware support
  - Cache & check watchpoint access rights
  - Metadata-cache $\rightarrow$ where and when?
- High-performance
  - MemTracker [HPCA 2007]
  - On every memory access



Motivation

- Hardware support
  - Cache & check watchpoint access rights
  - Metadata-cache → where and when?
- High-performance
  - MemTracker [HPCA 2007]
  - y access

Introduces dynamic energy overhead

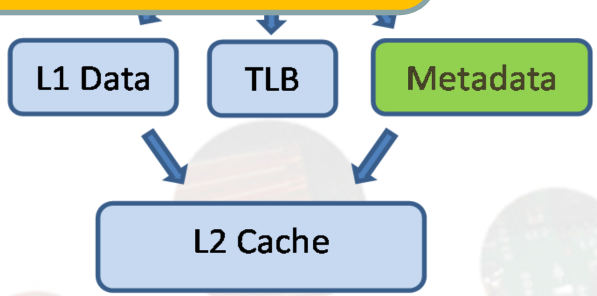


Motivation

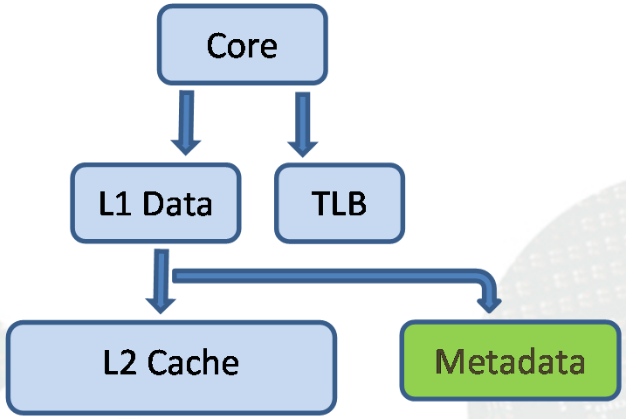
- Hardware support
  - Cache & check watchpoint access rights
  - Metadata-cache → where and when?

- High-performance
  - MemTracker [HPCA 2007]
  - y access

Introduces dynamic energy overhead



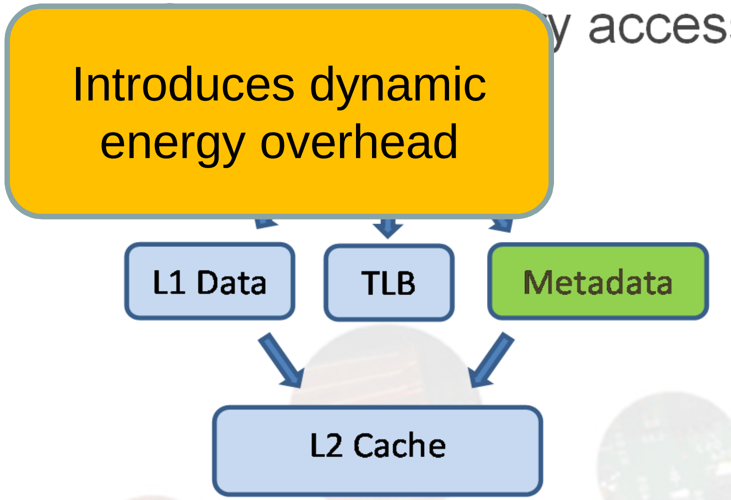
- Energy efficiency
  - Sentry [ISCA 2010]
  - On L1 miss path



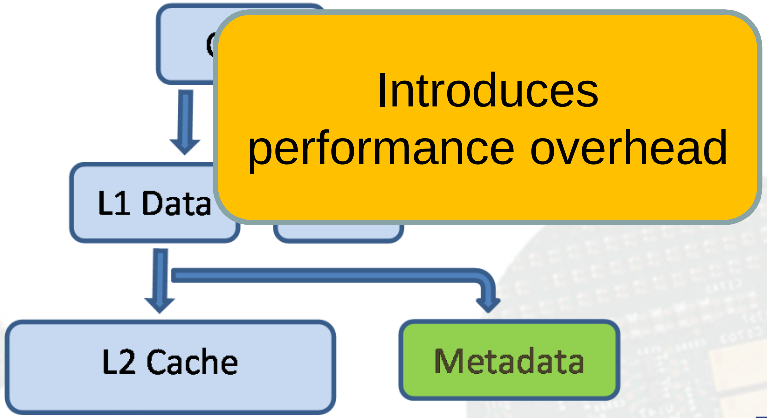
Motivation

- Hardware support
  - Cache & check watchpoint access rights
  - Metadata-cache → where and when?

- High-performance
  - MemTracker [HPCA 2007]
  - y access



- Energy efficiency
  - Sentry [ISCA 2010]
  - On L1 miss path

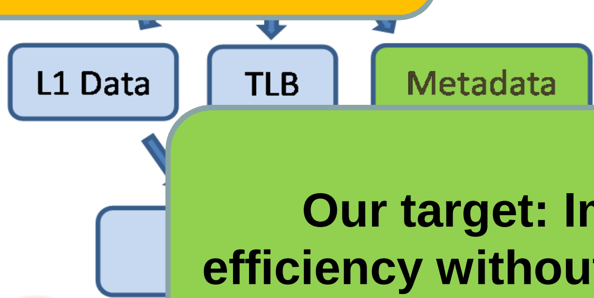


Motivation

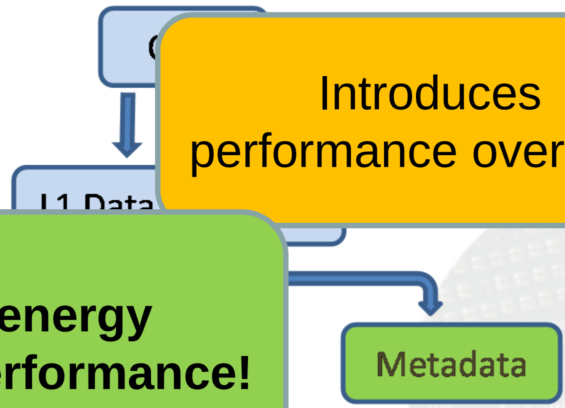
- Hardware support
  - Cache & check watchpoint access rights
  - Metadata-cache → where and when?

- High-performance
  - MemTracker [HPCA 2007]
  - y access

Introduces dynamic energy overhead



- Energy efficiency
  - Sentry [ISCA 2010]
  - On L1 miss path

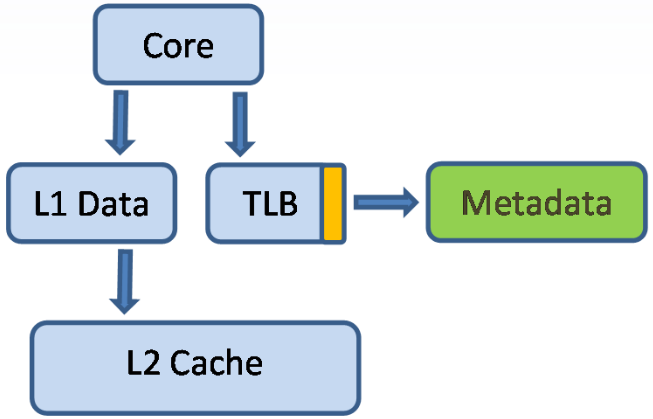


Introduces performance overhead

Our target: Improve the energy efficiency without hurting performance!

Reducing the metadata checks

- WatchPoint Filtering (WPF)
  - Reuse a bit in the TLB entry: WP-bit
  - Do/Don't access metadata cache
  - Page level filtering
  - Non-intrusive mechanism



TAG		DATA		
ASID	**VPN**	**PPN**	**VM Prot**	**WP**
0x7	0x683	0x60758	r/w	0
0x7	0x623	0x60342	r/w	1
. . .	. . .	. . .	. . .	. . .
0x7	0x725	0x6884C	r/w	0

WatchPoint Filtering - example

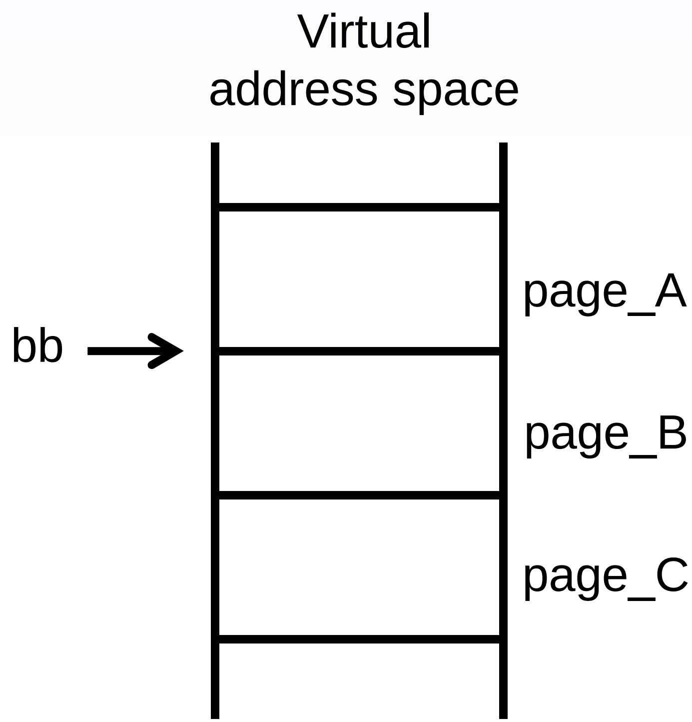
- Use case: heap checker
  - Buffer overflow
  - Dangling pointers

```
char *bb = malloc (4096 * sizeof(char));

for (int i=0; i<4096; i++) {
    // process each element
    bb[i] = … ;
}
```

WatchPoint Filtering - example

- Use case: heap checker
  - Buffer overflow
  - Dangling pointers

```
char *bb = malloc (4096 * sizeof(char));

for (int i=0; i<4096; i++) {
    // process each element
    bb[i] = … ;
}
```

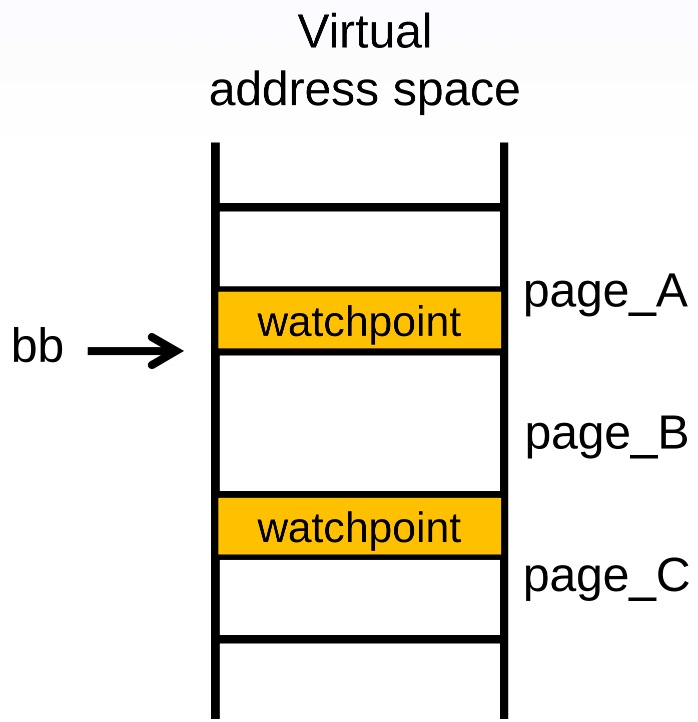


VPN	PPN	WP
page_A	. . .	0
page_B	. . .	0
page_C	. . .	0

WatchPoint Filtering - example

- Use case: heap checker
  - Buffer overflow
  - Dangling pointers

```
char *bb = malloc (4096 * sizeof(char));

for (int i=0; i<4096; i++) {
    // process each element
    bb[i] = … ;
}
```

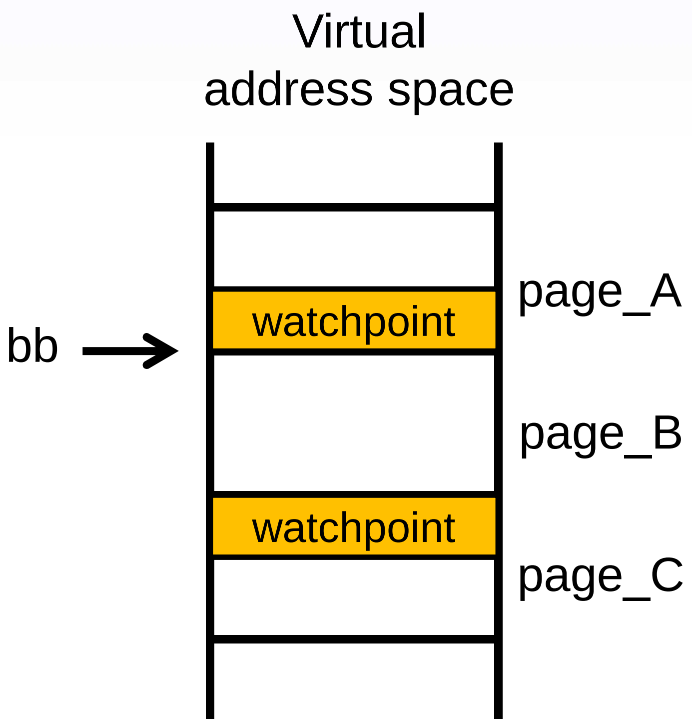


VPN	PPN	WP
page_A	. . .	0
page_B	. . .	0
page_C	. . .	0

WatchPoint Filtering - example

- Use case: heap checker
  - Buffer overflow
  - Dangling pointers

```
char *bb = malloc (4096 * sizeof(char));

for (int i=0; i<4096; i++) {
     // process each element
     bb[i] = … ;
}
```



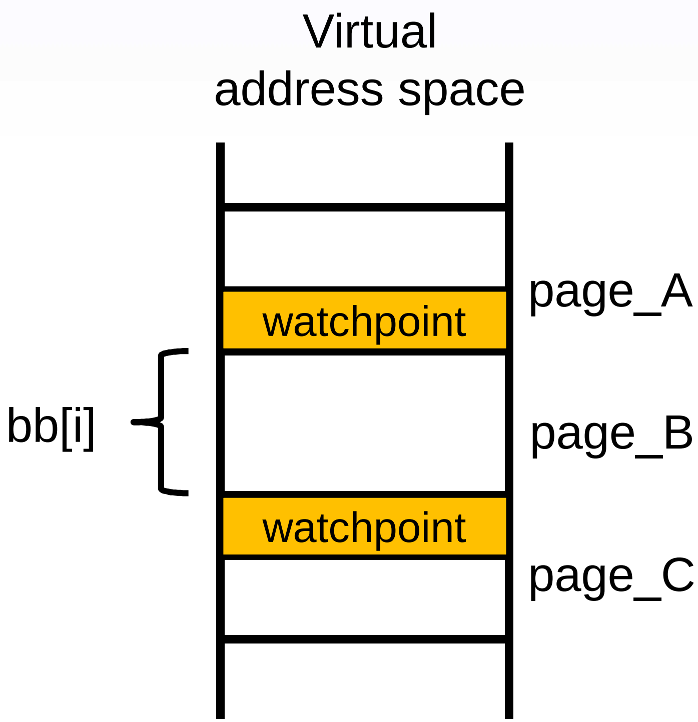
VPN	PPN	WP
page_A	. . .	1
page_B	. . .	0
page_C	. . .	1

Update the **WP** bits!

WatchPoint Filtering - example

- Use case: heap checker
  - Buffer overflow
  - Dangling pointers

```
char *bb = malloc (4096 * sizeof(char));

for (int i=0; i<4096; i++) {
    // process each element
    bb[i] = … ;
}
```



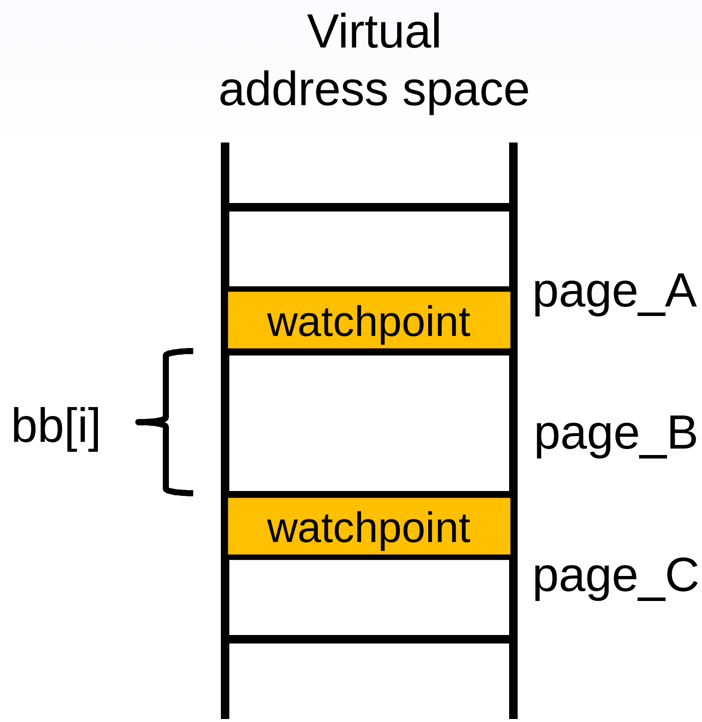
bb[i] →

VPN	PPN	WP
page_A	. . .	1
page_B	. . .	**0**
page_C	. . .	1

WatchPoint Filtering - example

- Use case: heap checker
  - Buffer overflow
  - Dangling pointers

```
char *bb = malloc (4096 * sizeof(char));

for (int i=0; i<4096; i++) {
    // process each element
    bb[i] = … ;
}
```



bb[i] →

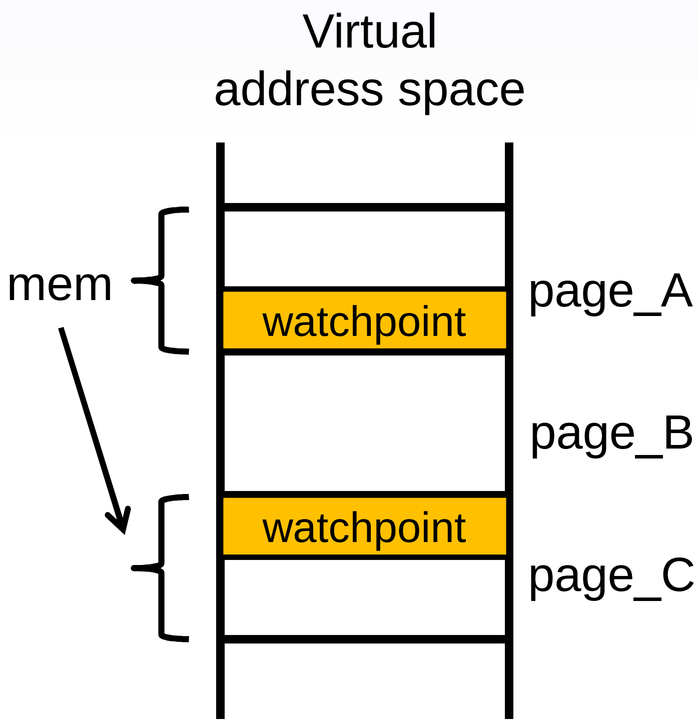
VPN	PPN	WP
page_A	. . .	1
page_B	. . .	**0**
page_C	. . .	1

→ Do **not** access the metadata cache!

WatchPoint Filtering - example

- Use case: heap checker
  - Buffer overflow
  - Dangling pointers

```
char *bb = malloc (4096 * sizeof(char));

for (int i=0; i<4096; i++) {
    // process each element
    bb[i] = … ;
}
```



mem

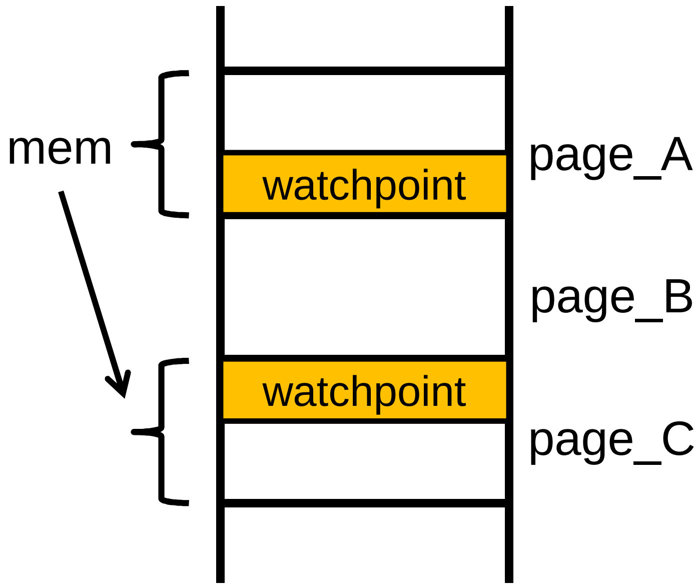
VPN	PPN	WP
page_A	. . .	**1**
page_B	. . .	0
page_C	. . .	**1**

WatchPoint Filtering - example

- Use case: heap checker
  - Buffer overflow
  - Dangling pointers

```
char *bb = malloc (4096 * sizeof(char));

for (int i=0; i<4096; i++) {
    // process each element
    bb[i] = … ;
}
```

Virtual
address space



mem

VPN	PPN	WP
page_A	. . .	**1**
page_B	. . .	0
page_C	. . .	**1**

Access the
metadata cache!

WatchPoint Filtering

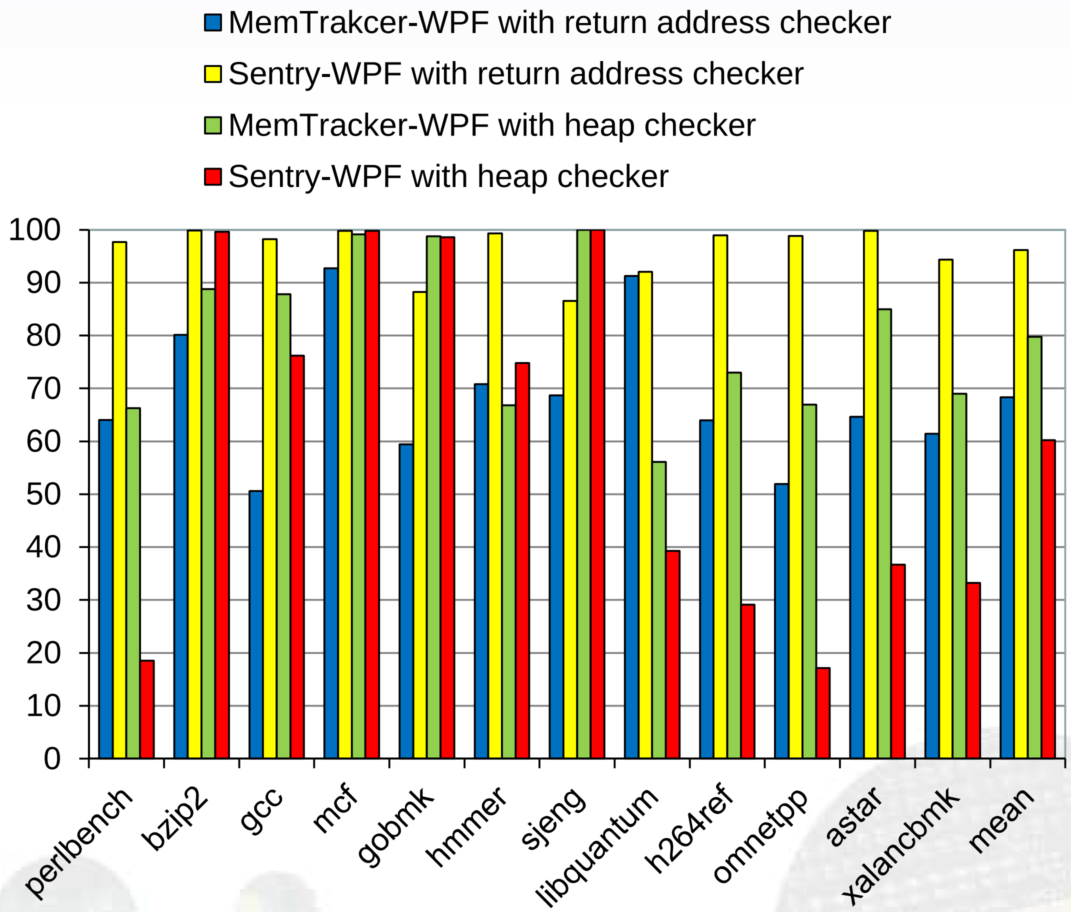
- Advantages
  - **Reduced dynamic energy** of the metadata cache
  - **Higher** metadata **hit-ratio**
  - Reduced execution overhead
    - Less **transitions** to the software backend
    - Less **cache interference**
- Implementation details in DAC 2012 paper
  - Who **controls** the WP-bit?
  - **When** is WP-bit updated?
  - **How** it works in **CMPs**?

How it works in CMPs

- TLB shootdown to keep copies of WP bit coherent
- Shootdown algorithm takes place only when the check of the watchpoints is enforced (WP-bit goes from 0 to 1)
- So that no watchpoint checks are missed

Evaluation Results

- Pin-based simulator
- Cacti 6.5
- Use cases
  - Heap checker
  - Return-address
- Benchmarks
  - specint2006
  - splash2

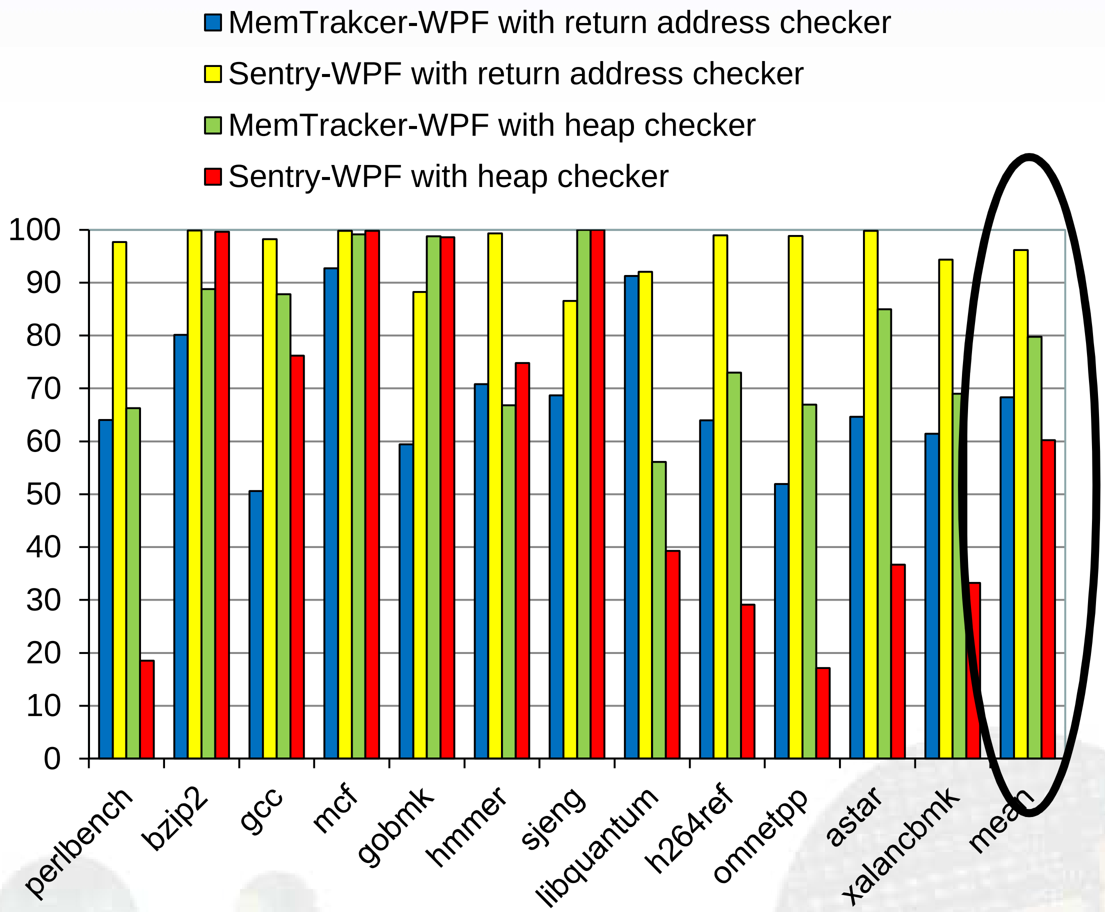


Percentage of reduced accesses in the metadata-cache

Evaluation Results

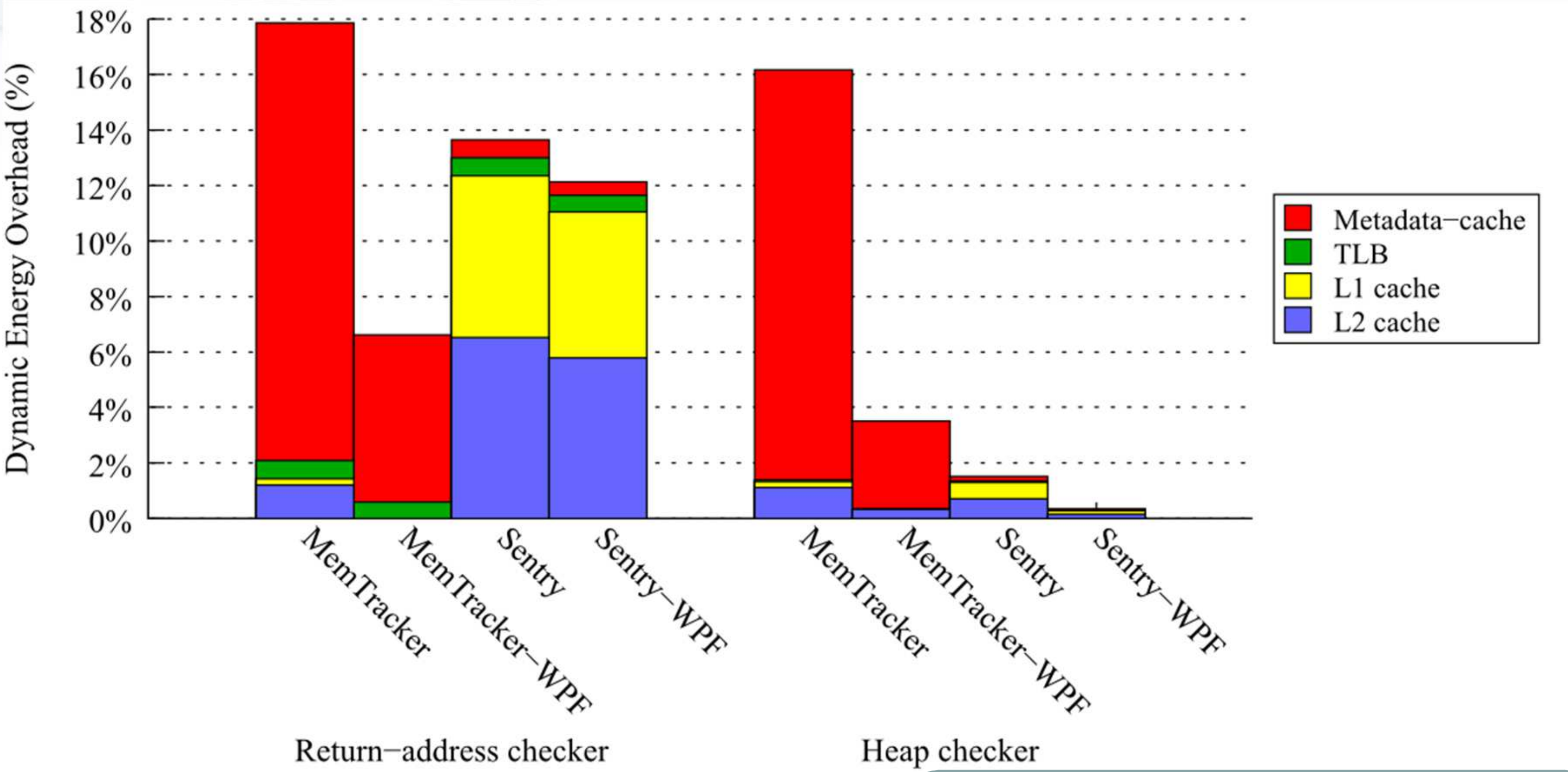
- Pin-based simulator
- Cacti 6.5
- Use cases
  - Heap checker
  - Return-address
- Benchmarks
  - specint2006
  - splash2

WPF **eliminates 79%** of **metadata checks** for MemTracker and **87%** for Sentry.



Percentage of reduced accesses in the metadata-cache

Evaluation Results

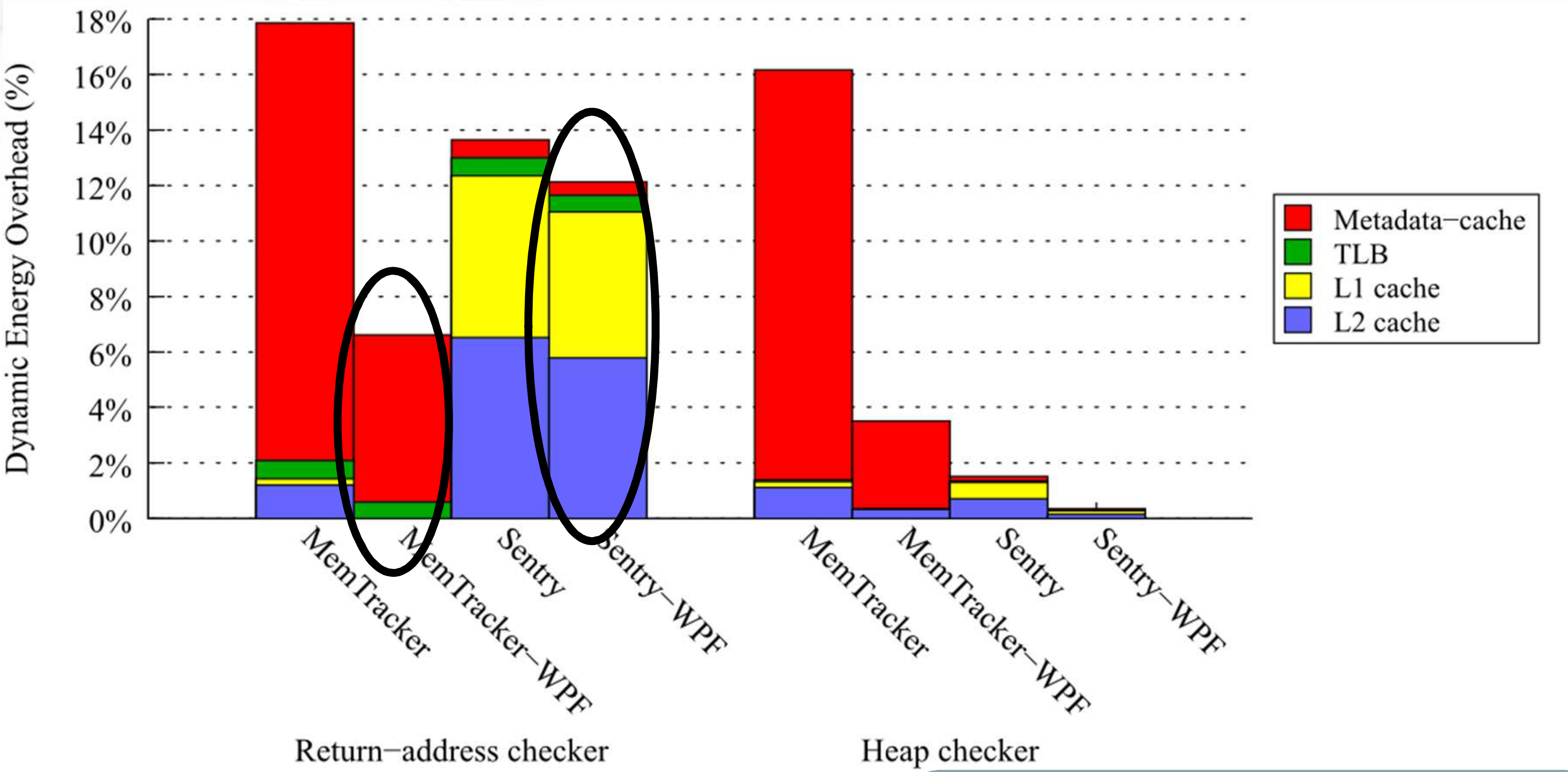


Breakdown of dynamic energy ove

WPF reduces energy overhead

- **71%** for MemTracker
- **44%** for Sentry

Evaluation Results



rgy ove

MemTracker-WPF vs. **Sentry-WPF**
- **more energy efficient:** 6.6% vs. 13.7%
- **less energy efficient:** 3.5% vs. 0.4%
- **Depends on the use case**!

WPF reduces energy overhead
- **71%** for MemTracker
- **44%** for Sentry

Conclusions

- Watchpoint systems may introduce
  - dynamic energy overheads
  - performance overheads
  - depending on the use case
- We propose the WatchPoint Filtering mechanism
  - Simple yet effective
  - Non-intrusive mechanism
- We demonstrate that WPF
  - eliminates 83% of the watchpoint checks
  - reduces 57% of the dynamic energy overhead
  - without penalties in performance execution

Related Work

- BSC-Microsoft Research Center (http://www.bscmsrc.eu)
  - FPGA tracing-visualization

  TMbox: A Flexible and Reconfigurable 16-core Hybrid Transactional Memory System FCCM11

  A low-overhead profiling and visualization framework for Hybrid Transactional Memory FCCM12
  - TM-aware debugger

  Debugging Programs that use Atomic Blocks and Transactional Memory PPoPP10

  Discovering and Understanding Performance Bottlenecks in Transactional Applications PACT210
  - TM-aware race detection
  - Dataflow – Shared Memory PM verification

- Intel BSC Exascale Lab (http://www.bsc.es/intel-bsc-exascale-lab)
  - Adopting task-based PM & dataflow runtime for exascale reliability

FP7 projects

- ParaDIME (www.paradime-project.eu)
  - Limits of energy saving; using actor based message passing PM
  - Speculate by going below safe Vdd
  - Approximate processing

- Rethink-big
  - Roadmap for HW and Networking support for Big Data

Thank you!

Questions?

osman.unsal@bsc.es