Karlsruhe Institute of Technology

Reducing Response Time with Preheated Caches

Mathias Gottschlag, Frank Bellosa | August 22, 2016

Operating Systems Group, Department of Informatics

Problem

- More transistors = more cores?
  - Limited power density limits usable silicon area
  - *Dark Silicon,* only a fraction of the chip is usable

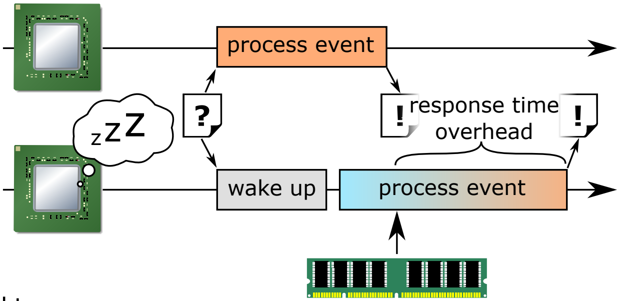
- Aggressive power management is beneficial for performance
  - “Saved” power can be later reinvested to increase performance

- Example: *Computational Sprinting*
  - Temporarily exceed the power budget
  - Remove power to inactive cores and caches (power gating)

Problem

- Negative effects of power gating in a server system:
  - Wakeup latency (see Zhu et al., “anticipatory CPU wakeups”)
  - Lost state (e.g., cold caches)

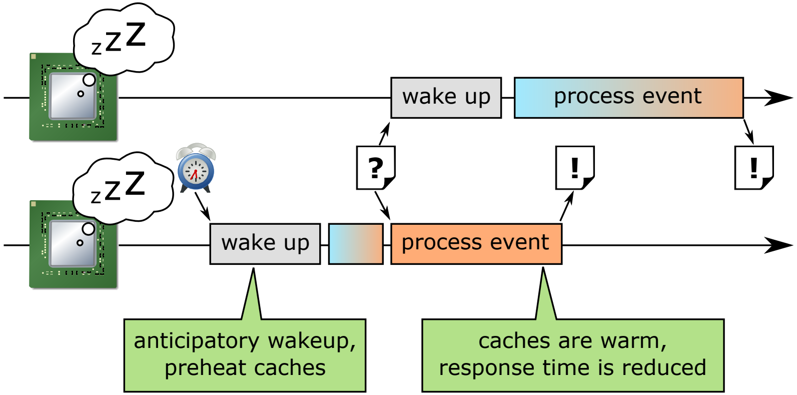
➜ Significantly increased response times



- We need to...
  - ... hide the wakeup latency
  - ... reduce the number of cache misses

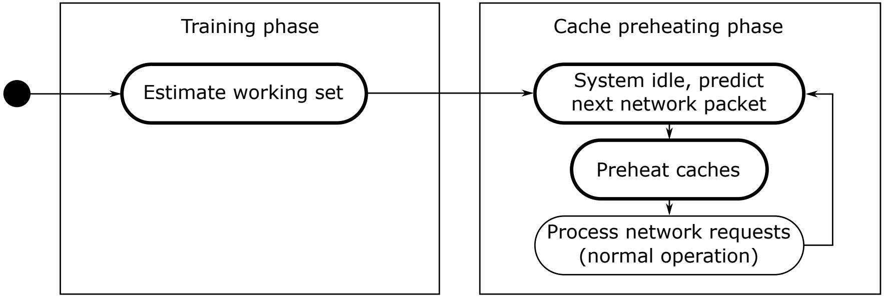
Approach

Preheat the caches before the network event occurs:


zzZ
wake up
process event
?
!
!
zzZ
wake up
process event
anticipatory wakeup, preheat caches
caches are warm, response time is reduced

Approach

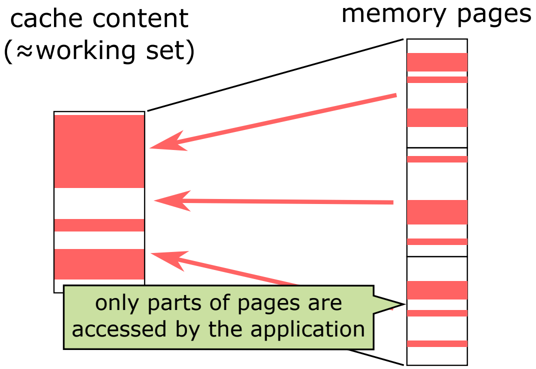
1. Find out what data is accessed after a network event
   - Fine-grained working set estimation based on cache state analysis
2. Predict future network events
   - Predictable network architecture announces future packets
3. Preheat the caches before a packet is received
   - Load the predicted working set from memory


Training phase
Estimate working set
Cache preheating phase
System idle, predict next network packet
Preheat caches
Process network requests (normal operation)

Working Set Estimation

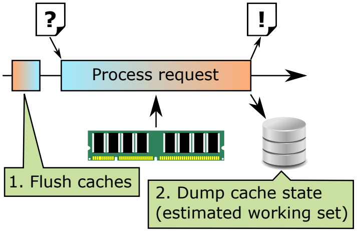
Requirement: **cache line granularity** on **commodity hardware**

- Size should be minimized (cache capacity, loading time)
- Page granularity not sufficient



Working Set Estimation

- ARM: facilities to read/write cache memories (`RAMINDEX`)
- Dump the L2 tag bits after processing a network request



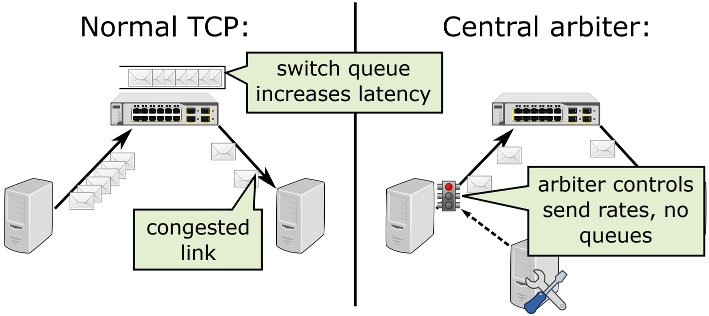
- Alternative on x86: Use PEBS to record all cache misses?
- Exclude variable parts of the working set (e.g., network buffers):
  - Repeat cache state analysis and remove infrequent entries

Approach

1. Find out what data is accessed after a network event
   - Fine-grained working set estimation based on cache state analysis
2. Predict future network events
   - Predictable network architecture announces future packets
3. Preheat the caches before a packet is received
   - Load the predicted working set from memory

Network Request Prediction

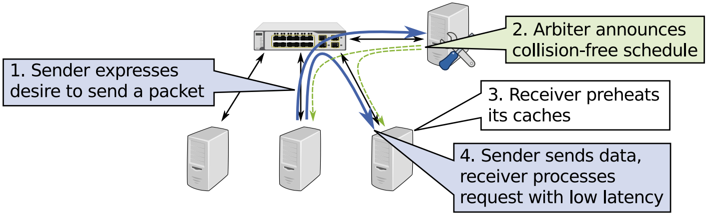
- Problem: Preheat caches **before** network request arrives
  - TCP highly nondeterministic
- Basis of our solution: Predictable network architectures (example: Fastpass, SIGCOMM'14)
  - Central arbiter implements congestion control
  - Shorter queues in switches ⇒ lower latency


Normal TCP:
Central arbiter:
switch queue increases latency
congested link
arbiter controls send rates, no queues

Network Request Prediction

- Extension to these network architectures:
  Arbiter sends the schedule to the "receiver" systems

➜ Acts as an announcement of incoming network packets!


1. Sender expresses desire to send a packet
2. Arbiter announces collision-free schedule
3. Receiver preheats its caches
4. Sender sends data, receiver processes request with low latency

- Low-latency network, modifications must not increase latency

➜ announcement only ≈60 µs in advance!

Approach

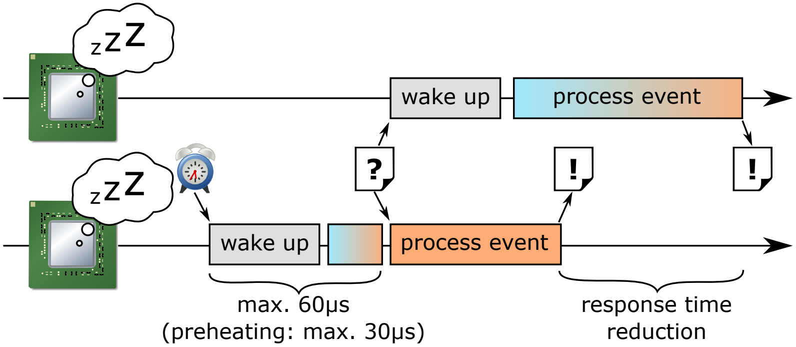
1. Find out what data is accessed after a network event
   - Fine-grained working set estimation based on cache state analysis
2. Predict future network events
   - Predictable network architecture announces future packets
3. Preheat the caches before a packet is received
   - Load the predicted working set from memory

Cache Preheating

- Load working set into cache after CPU wakes up
- Challenge: Only ≈30 µs available
  - Announcement 60 µs in advance
  - Current CPUs cause 30 µs wakeup latency
- Efficient memory bandwidth utilization mandatory
  - Sort all accesses by address ⇒ improved access pattern
  - compress address list with RLE ⇒ reduced preheater cache footprint

➜ Mostly memory bandwidth limited

Evaluation – Metrics

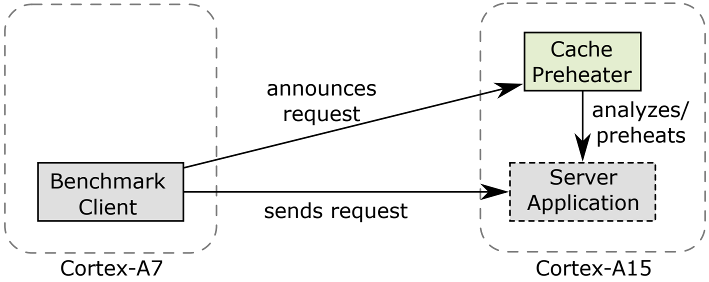


Metrics:

- Processing time
- Cache misses, CPI reduction (fewer stall cycles?)
- Time required to preheat the caches
- Response time ≈ processing time + preheating time - 30 µs

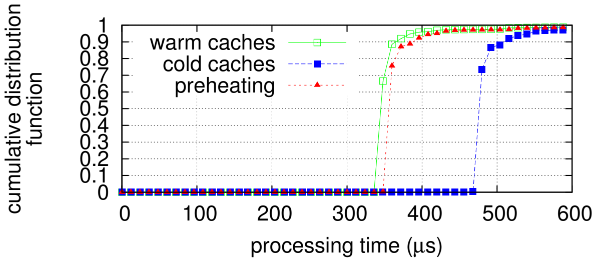
Evaluation – Setup

- Benchmarks: DBT-2 (MariaDB), memcached, nginx
- System: Odroid XU3, Samsung Exynos 5422
  - ARM big.LITTLE (typical use: smartphones)
  - 4x Cortex A7, 4x Cortex A15 (two separate L2 caches)
  - USB ethernet
- Limited prototype: All network functionality emulated



Evaluation – Results

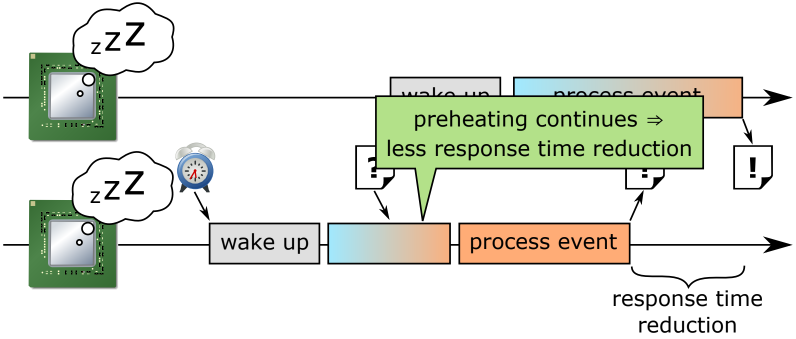
- Example: nginx web server



- Average over all benchmarks
  - 80% processing time overhead reduction
  - 66% cache miss reduction
  - 20% CPI reduction
  - Time required to preheat the caches: 50 µs to 150 µs (goal: 30 µs)

Discussion

- Prototype needs up to 5x too much time to preheat the caches
  But: Still provides a **net response time reduction**
  - 80% less processing time overhead, 44% less response time overhead



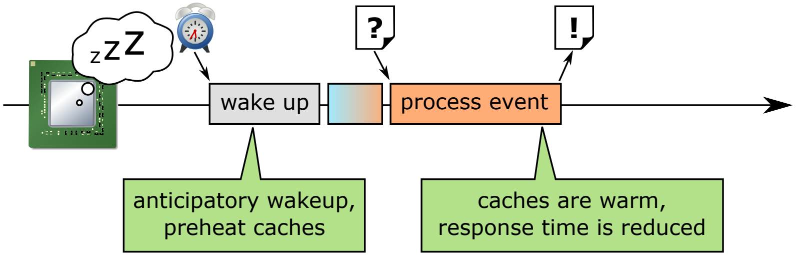
- Modern server CPUs provide 7x more memory bandwidth

➜ Better response time reduction expected

Summary

- Dark silicon is one of the big problems of the manycore era
- Aggressive power management beneficial for performance
- Requires efficient management of volatile state (CPU state, caches, ...)

➜ Proposal: Preheating of cache state to reduce response times


zzZ
?
!
wake up
process event
anticipatory wakeup, preheat caches
caches are warm, response time is reduced

References (excerpt)

- Perry, J., Ousterhout, A., Balakrishnan, H., Shah, D., Fugal, H.: Fastpass: A Centralized “Zero-Queue” Datacenter Network (SIGCOMM’14)
- Raghavan, A., Luo, Y., Chandawalla, A., Papaefthymiou, M., Pipe, K.P., Wenisch, T.F., Martin, M.M.K.: Computational Sprinting (HPCA’12)
- Schöne, R., Molka, D., Werner, M.: Wake-up latencies for processor idle states on current x86 processors (CSRD, Springer, May 2015)
- Zhu, Q., Zhu, M., Wu, B., Shen, X., Shen, K., Wang, Z.: Software Engagement with Sleeping CPUs (HotOS’15)