WearCore: A Core for Wearable Workloads

Sanyam Mehta* and Josep Torrellas
University of Illinois at Urbana Champaign

PACT, September 2016

*Sanyam is currently with Cray Inc.

Smartwatches

- There will be 485 million annual wearable shipments by 2018 [ABI Research]

- Smartwatches require to be tethered to the phone to access the network
  - ✓ Smartwatches are not "self-sufficient"

Making smartwatches self-sufficient is key to improving their utility to users

The First Steps to Self-Sufficiency

- Advances in the industry
  - ✓ **OS:** Google's Android Wear now enables Wi-fi connectivity in the watch to connect to a distant phone
  - ✓ **Networking:** Samsung's latest Gear S smartwatch can house its own SIM card and has built-in 3G connectivity
    - ➢ Can make/receive calls and send/receive messages
  - ✓ **Apps:** Samsung, Apple, Google have their own Personal Assistants to interface with the watch via speech

Is that all to self-sufficiency? What are we missing?

The Missing Piece …
A General-Purpose Core

- Why not use a core as in smartphones?
  - ✓ Discrepancy in terms of applications run on a phone and a watch
    - ➢ Different usage scenario: simple short queries, small display size
  - ✓ Wide discrepancy in energy budget
    - ➢ Smartwatches have a much lower battery life (Gear S has one-eighth the battery life of Galaxy S6)

- Currently:
  - ✓ Samsung: out-of-order dual core
  - ✓ LG: in-order quad core
  - ✓ Apple: out-of-order single core

Need to first identify important applications,
and then build energy-efficient hardware

Overview

- WearBench
- Key Insights
- Our proposed solution: WearCore
- Implementation
- Results and Discussion
- Conclusion

Overview

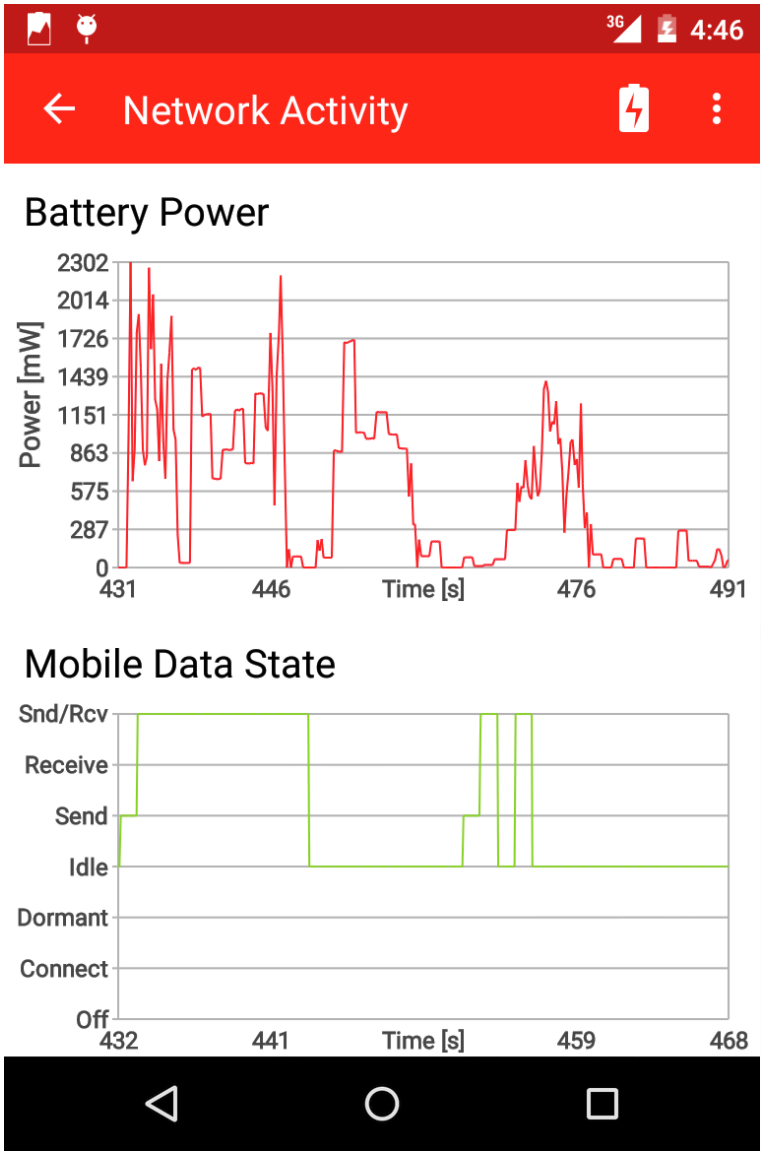
- WearBench
- Key Insights
- Our proposed solution: WearCore
- Implementation
- Results and Discussion
- Conclusion

WearBench: A Suite of Target Applications for a Smartwatch

- Automatic Speech Recognition (ASR)
  - ✓ Critical to smartwatch users given the difficulty in typing on the small screen
- Image Processing
  - ✓ Relevant given the thin camera lens on a smartwatch
- Computer Vision
  - ✓ Face recognition to log into the phone and other apps
- Audio Playback
  - ✓ Important to support this efficiently on a smartwatch
- Video Rendering
  - ✓ Useful as smartwatches are equipped with 3G service

ASR: Issuing a "Google Now" Command

Power profile and mobile data state of Google Nexus3 phone using the Qualcomm Trepn profiler when executing a command through Google Now



- A smartphone moves to high power Snd/Rcv state when using 3G service: 12 sec, 1200mW avg
- A DNN-based offline speech recognizer can execute the same query in <1 sec, 1000mW peak: 14.4x less energy

It is important to support ASR and other apps in WearBench in offline mode and run them efficiently

Overview

- WearBench
- Key Insights
- Our proposed solution: WearCore
- Implementation
- Results and Discussion
- Conclusion

Observations

- Seven of ten applications in WearBench are inherently parallel and can utilize all the cores on the chip.
- The average miss rate incurred by WearBench is more than 4x higher than in BBench (a suite of Mobile Workloads). Apps in WearBench load data from memory and then process it.
- On average, 25% of the operations utilize the SIMD unit on the core. In contrast, BBench does not utilize the SIMD unit at all.
  - ✓ Apps in WearBench can be vectorized
- Combination of parallelism + vectorization:
  - ✓ Pressure on the memory system

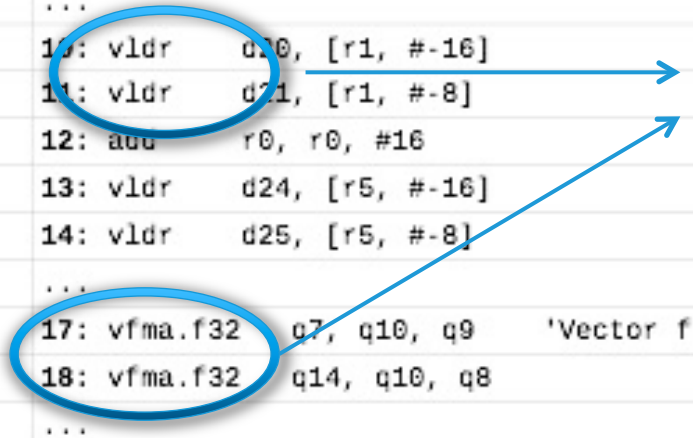
Insights

- Future smartwatches must employ multiple cores to meet the aggressive QoS needed for certain applications.
- The cores must be simple to satisfy the low area and power budget of smartwatches.
- Given strict QoS requirements, an in-order core will not do
- We propose to use an out-of-order core that is simple (e.g., two outstanding cache misses only).

Will simple out-of-order execution be enough?

More Insights

% Samples		Disassembly			
In-order	Out-of-order				
		.L40			
		...			
3.33%	10.34%	5: vldr	d18, [r2, #-16]	'Vector load'	
7.58%	3.45%	6: vldr	d19, [r2, #-8]		
0.91%	-	7: add	r1, r1, #16	'Integer add'	
	-	...			
1.82%	8.62%	10: vldr	d20, [r1, #-16]		
5.15%	3.45%	11: vldr	d21, [r1, #-8]		
0.91%	-	12: add	r0, r0, #16		
0.91%	1.72%	13: vldr	d24, [r5, #-16]		
4.24%	13.79%	14: vldr	d25, [r5, #-8]		
		...			
1.21%	-	17: vfma.f32	q7, q10, q9	'Vector fp multiply-add'	
5.76%	-	18: vfma.f32	q14, q10, q8		
		...			
6.36%		27: bne	.L40	'End of loop'	



- Prefetching to L2: OoO core stalls even with L2-resident data
- Need to prefetch the data to L1 cache to prevent stalls

WearCore needs a light-weight prefetcher to prefetch the data to the L1 cache to hide stalls and approach an aggressive OoO core

Overview

- WearBench
- Key Insights
- Our proposed solution: WearCore
- Implementation
- Results and Discussion
- Conclusion

Software Assisted Hardware Prefetcher (SAHP)

WearCore uses a unique Software-Assisted Hardware Prefetcher (SAHP) to prefetch data to L1 cache.

Features of SAHP:

- SAHP is triggered by a software prefetch instruction that specifies the base address and the total number of lines to be prefetched
  - ✓ The hardware learns about the streams from the software
  - ✓ Negligible instruction overhead
- SAHP hardware schedules the prefetches to maintain the appropriate prefetch distance
  - ✓ Absolves the user from this involved decision-making
  - ✓ Effective when done in hardware

SAHP Implementation: Main Ideas

- A prefetch binds an MSHR to a data stream until all the data requested is prefetched, or the hardware recognizes useless prefetch requests
- As a line is prefetched, a counter in the MSHR is incremented. When a prefetched line is read, the counter is decremented
  - ✓ Tracked using a prefetch bit in each cache line
- When counter reaches 16, prefetching stops. When counter is reduced to 8, prefetching resumes
  - ✓ Goals: hide latency and prevent L1 pollution
- When all the data requested is prefetched, the MSHR is freed

Overview

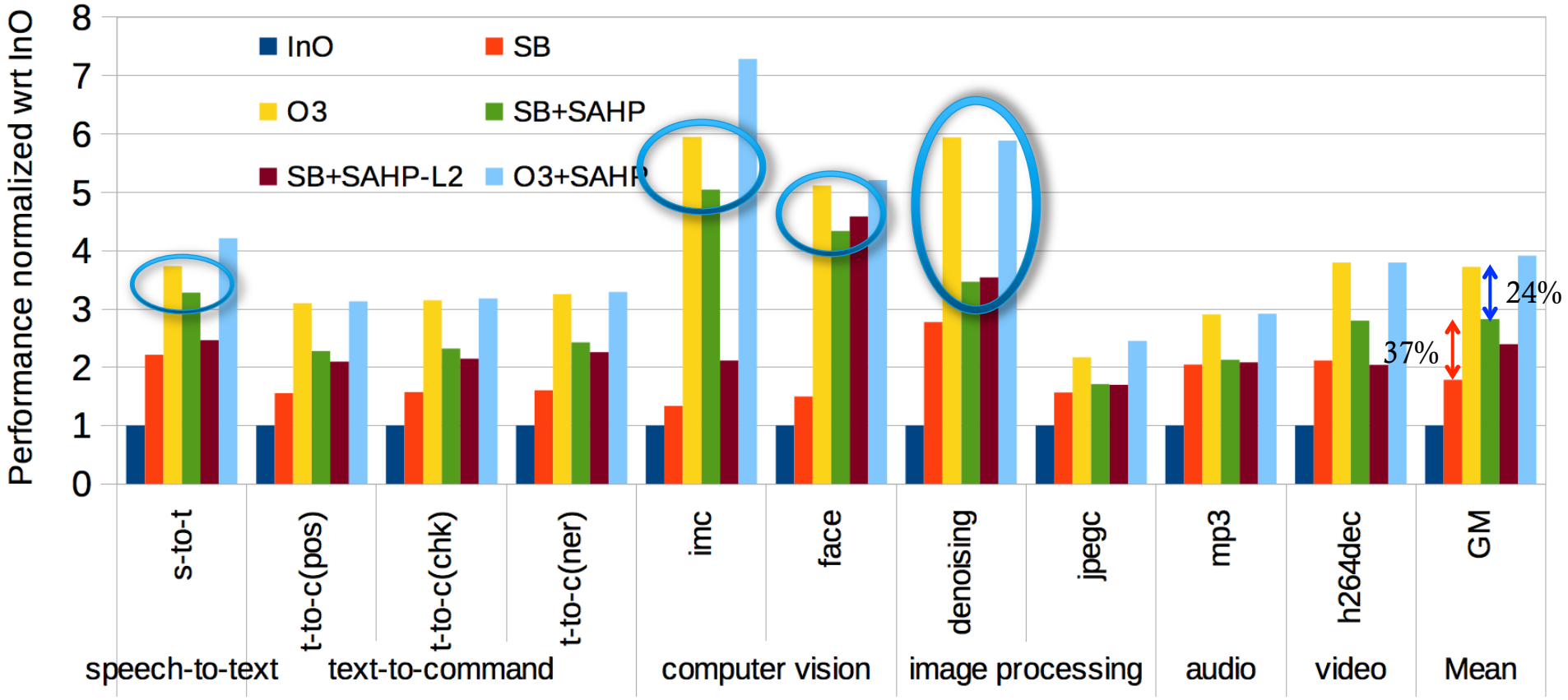
- WearBench
- Key Insights
- Our proposed solution: WearCore
- Implementation
- Results and Discussion
- Conclusion

Hardware Configurations

- InO: baseline in-order core without any prefetching (Cortex-A7)
- SB: Our simple out-or-order core
  - ✓ 2-issue; only 2 outstanding misses; no prefetching
- O3: Full out-of-order core (Cortex-A15)
  - ✓ 2-issue; hardware prefetching into L2
- SB+SAHP: WearCore
- SA+SAHP-L2: WearCore with no L2 (to save power)
- O3+SAHP: O3 with SAHP prefetcher

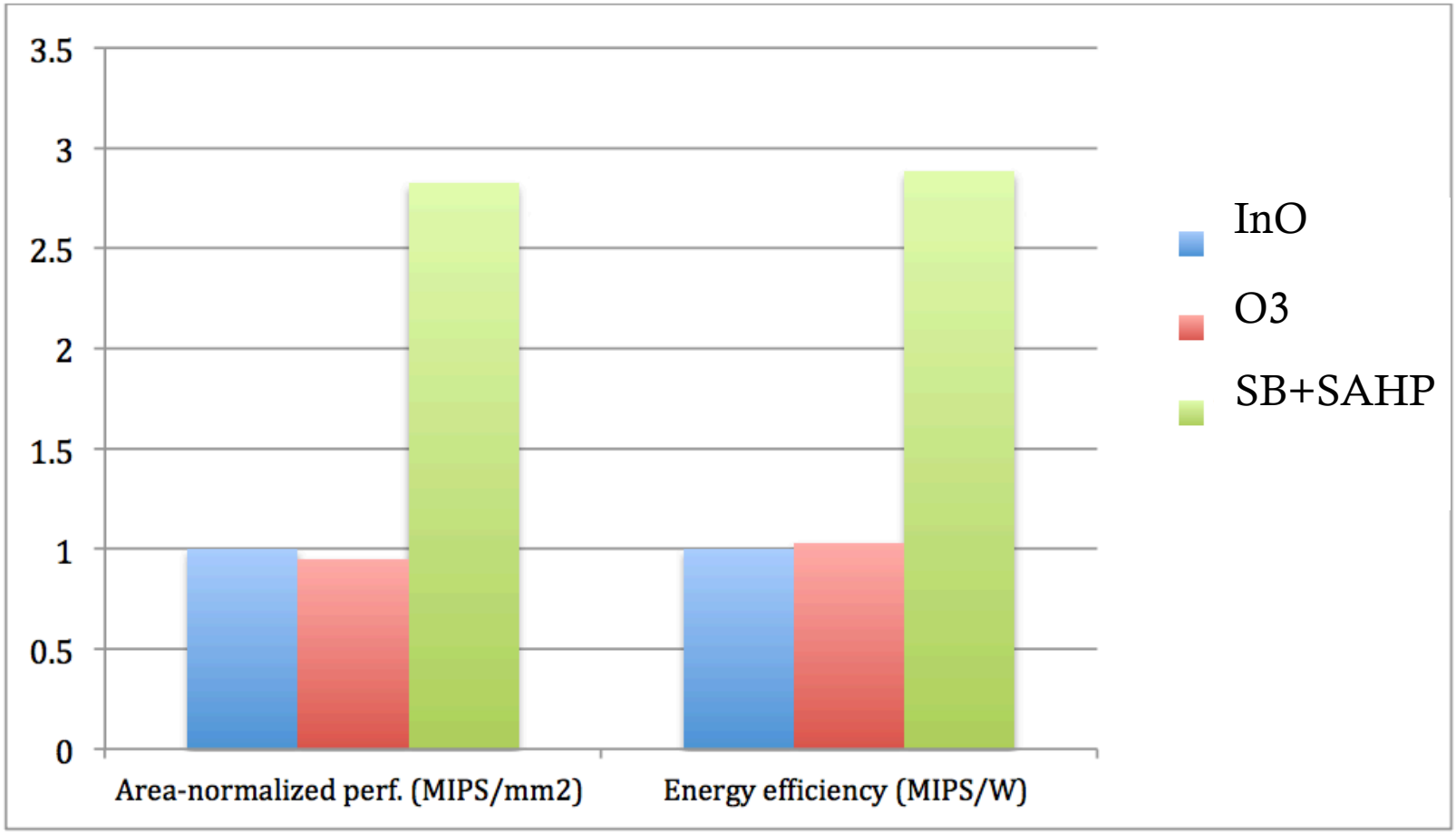
- All designs have 4 cores
- O3 has twice the L2 (1MB)

Performance



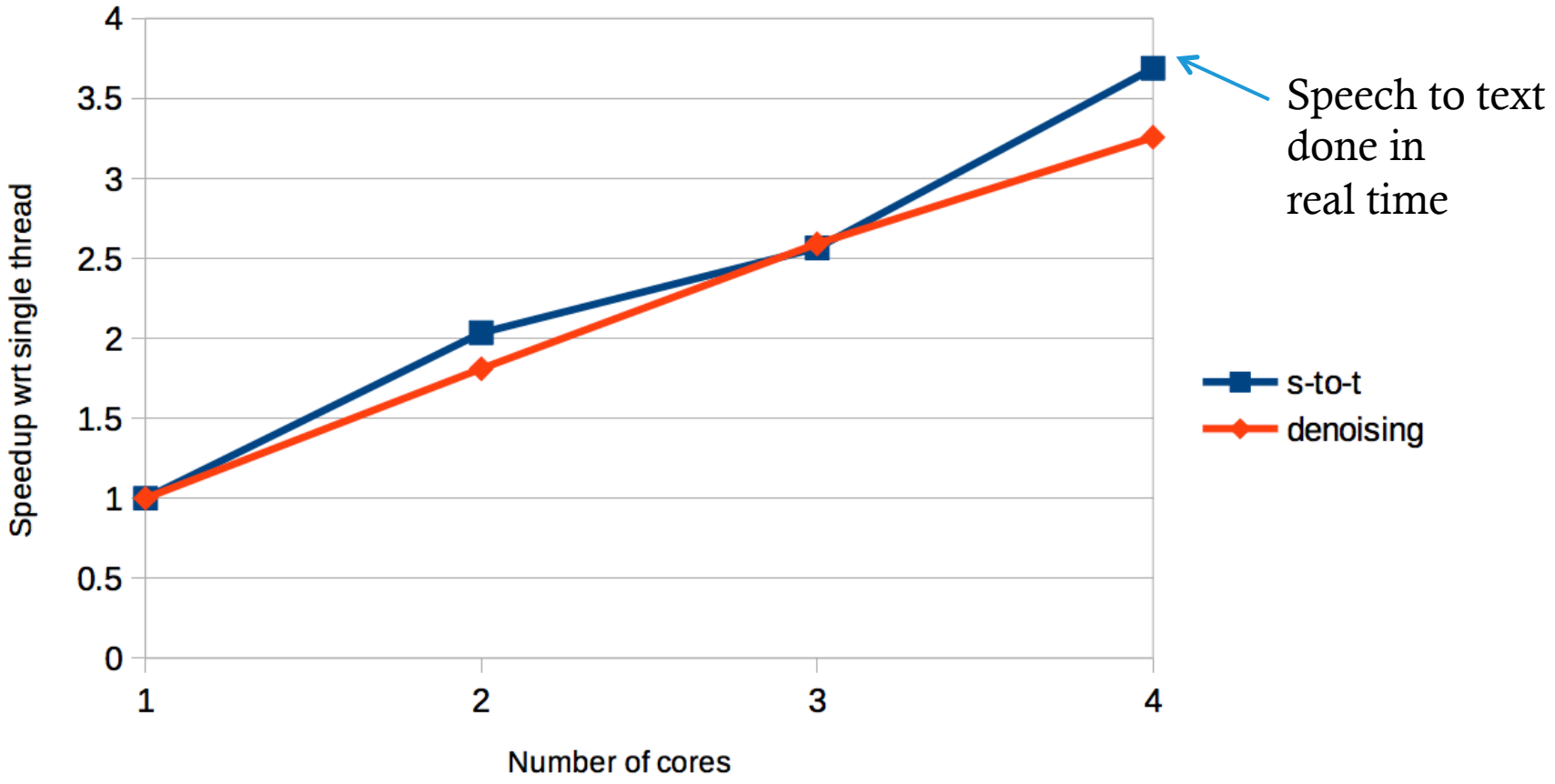
Performance comparison of different cores on applications in WearBench

Efficiency



Area-normalized performance and energy-efficiency

WearCore Parallel Performance



Linear scaling with the number of cores proves useful to achieve QoS

Discussion

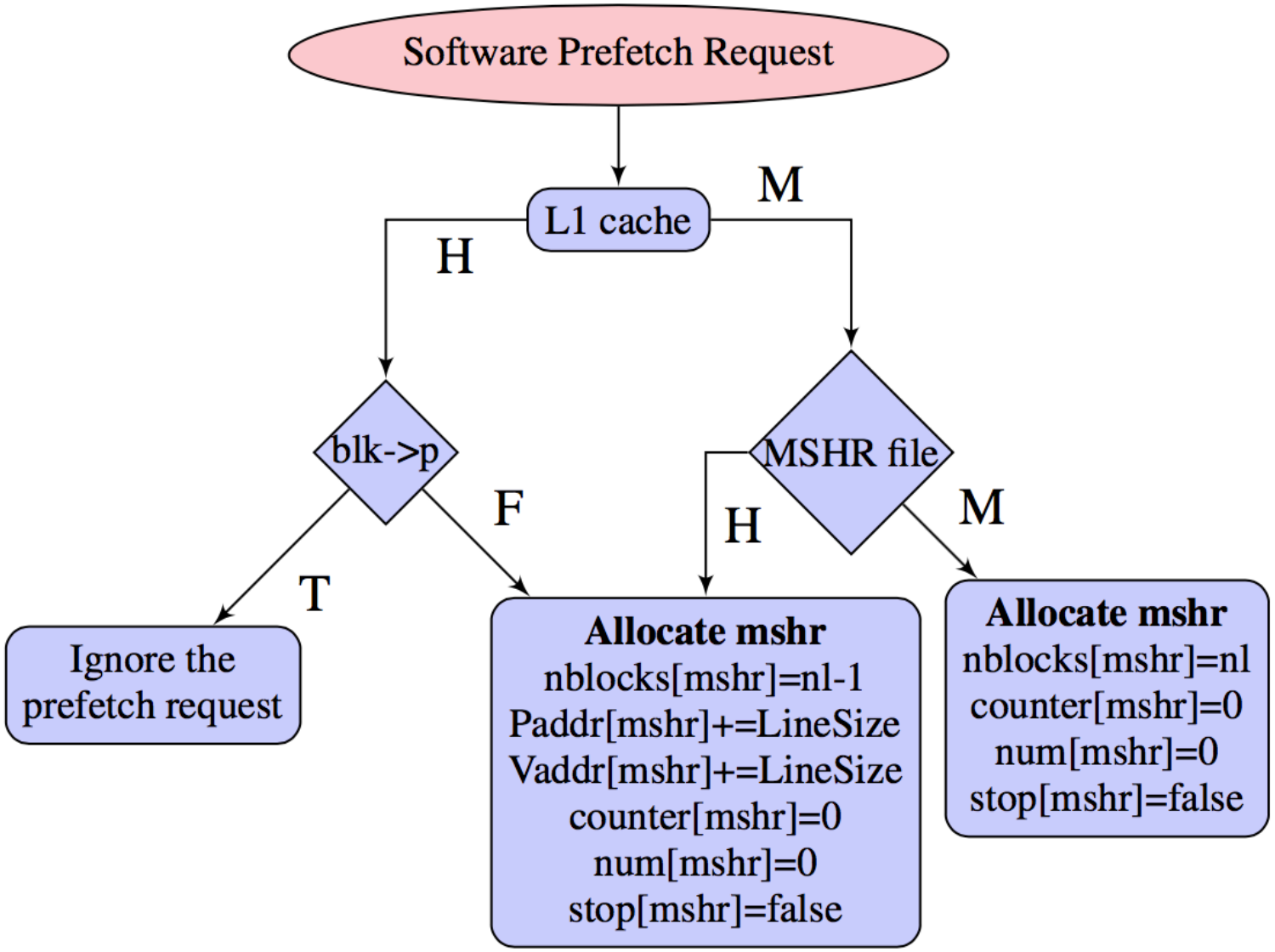
- DSP for WearBench?
  - ✓ Multiprocessing not supported
    - ➢ Parallelism is critical to meet QoS in WearBench
- An accelerator for DNN?
  - ✓ Risky to employ specialization in an emerging domain
    - ➢ Eg: Google move to RNN, Samsung's latest gesture recognition
- Who will insert the SAHP prefetches?
  - ✓ Library developers, who currently tune the code for SIMD

Conclusion

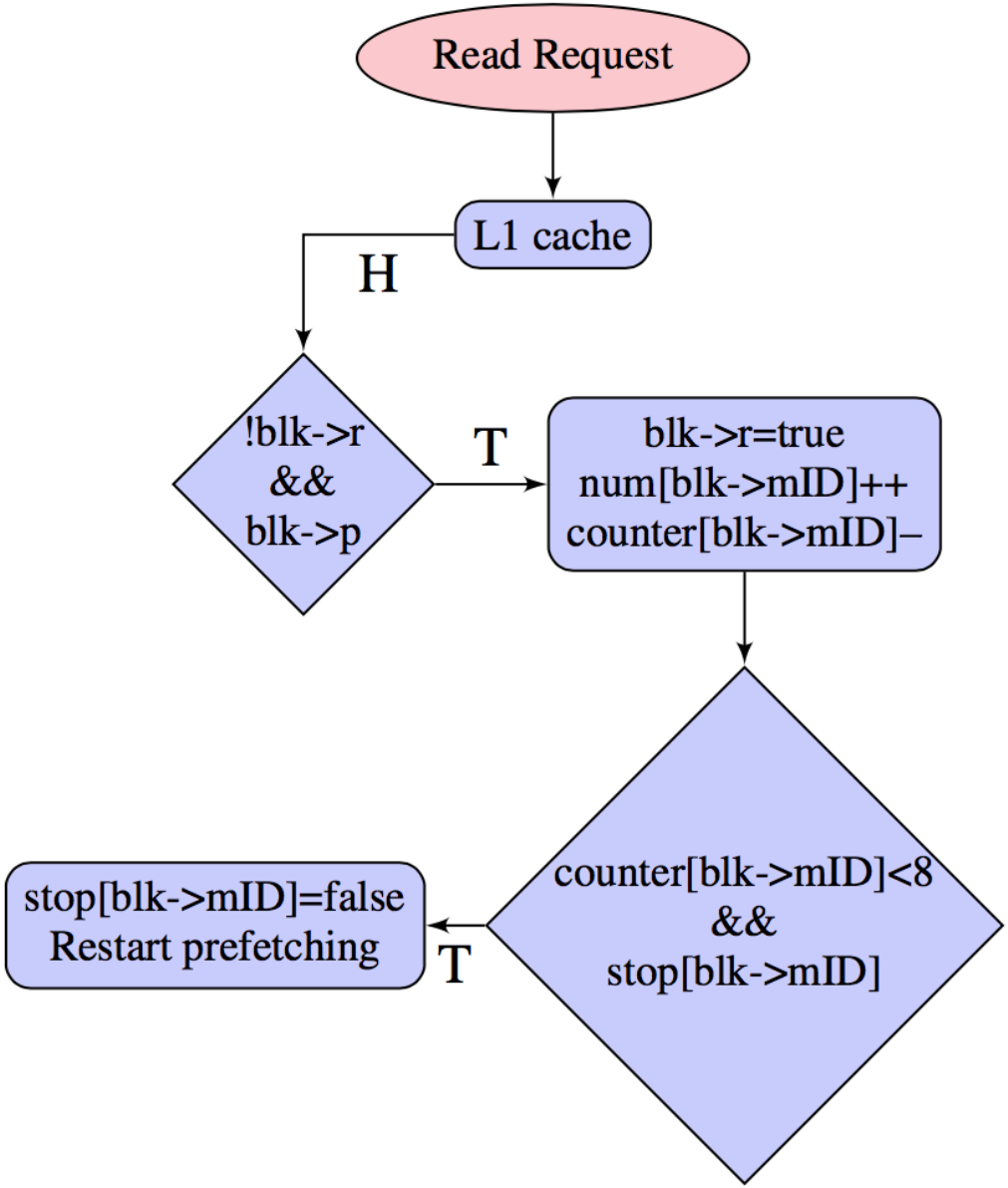
- WearCore: A core for efficiently executing smartwatch workloads
- Built on the insight that the important workloads for a smartwatch are parallel, and make extensive use of vectorization.
- Augments a simple ooo quad-core processor with a light-weight software-assisted hardware prefetcher.
- Prefetcher adds minimal overhead in area and power to the core, but adds 37% performance.
- WearCore achieves significant improvement in performance per area and energy efficiency over both an in-order and an out-of-order core.

THANK YOU!

Implementation (1/3)



Implementation (2/3)



Implementation (3/3)