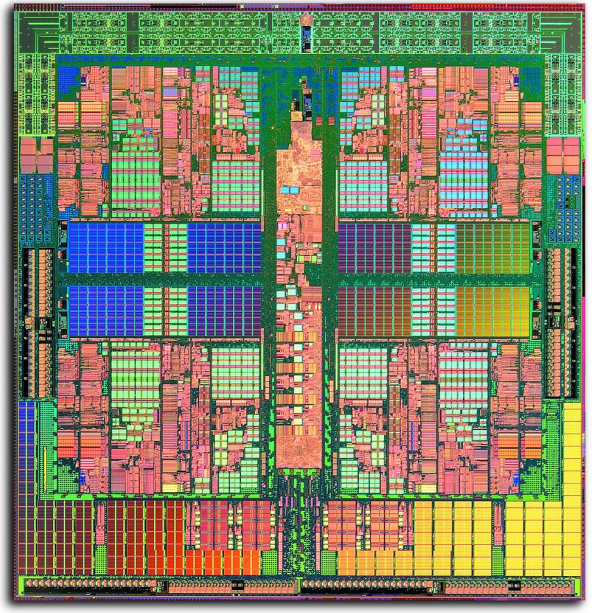
R3-DLA (Reduce, Reuse, Recycle): A More Efficient Approach to Decoupled Look-Ahead Architecture

Sushant Kodguli, Michael Huang (University of Rochester)
HPCA 2019
Presented By: Nick from CoffeeBeforeArch

Overview

Overview

- Always a demand for compute
  - Single-thread performance is still critical
  - Not everything is paralizable
- Use idle resources to actively improve performance
  - Decoupled look-ahead architecture (DLA)
- Expand on a baseline DLA to increase single thread performance by ~40%



Background

Background

- Dennard Scaling is over
  - Transistor and frequency scaling is no longer feasible
- Still far from ideal ILP
  - Branch misprediction
  - Cache misses
- Number of transistors is still increasing
  - Typically used to exploit parallelism
  - Instead, use for more advanced prediction

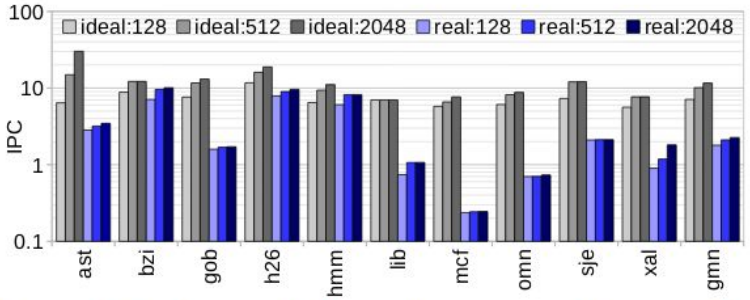


Figure 1. Implicit parallelism of integer applications. The three bars on the left indicate the amount of available parallelism (instructions per cycle) in the application when inspected with a moving window of 128, 512, or 2048 instructions. The three bars on the right repeat the same experiments only this time under a realistic branch misprediction and cache miss situation that further constrains the available parallelism to be exploited. Note that the vertical axis is in log scale.

Baseline Architecture

Baseline Architecture

- Generates a skeleton of the original program to execute in parallel
- Pass hints back to the main thread
- Feedback in order to prevent veering of the right path
  - Re-initialization

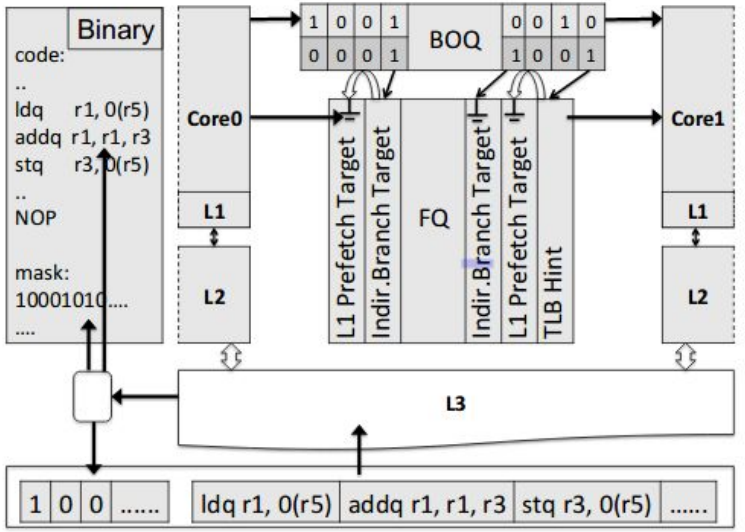


Figure 2. Architectural support for baseline DLA.

Design Extensions

Design Extensions

- T1:
  - FSM for prefetching loop-based strided accesses
- Value Reuse:
  - Pass register values between cores
- Fetch Buffer:
  - Extended fetch down predicted path
- Re-Cycle Controller:
  - Find a better instruction profile

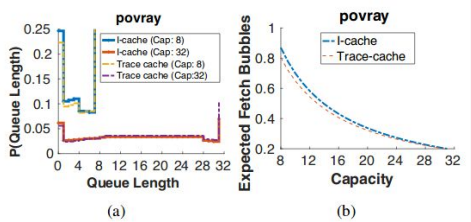
state	loop PC	inst. PC	eff. addr.	stride	cur. time	pref. distance

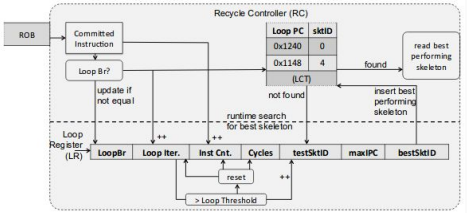
Figure 3. T1 prefetch register fields.

	skeleton mask		
i1	1	mul	r8,r11,r5
i2	1	add	r6,r21,r4
i3	0	mul	r5,r12,r11
i4	1	add	r4,r5,r2
i5	1	sub	r10,r11,r10
i6	1	ret	

can skip value prediction validation

Figure 4. Example of skipping value prediction validation.





Design Extensions

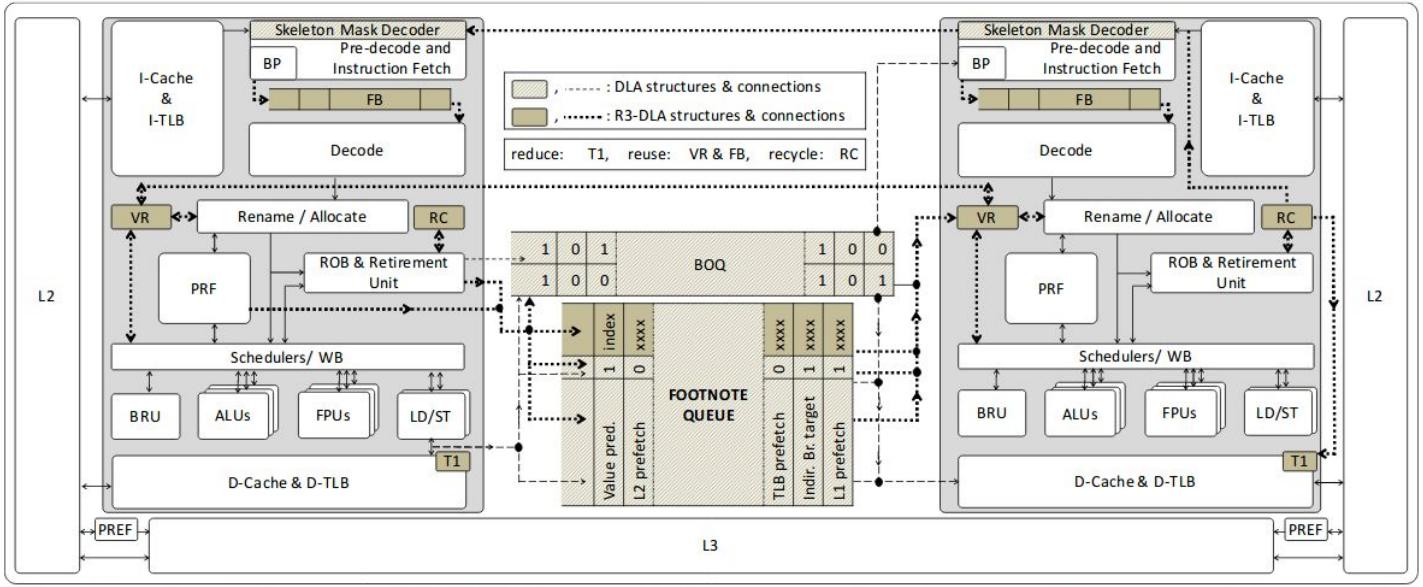


Figure 8. Architectural support for R3-DLA. The gray colored rectangle on the left represents the lead core and the one on the right represents the main core. Structures included by DLA are patterned and the connections are indicated with the dashed lines. Structures included by R3-DLA are shaded and the connections are indicated with the dotted lines.

Results

Results

- Turbo-Boosting Mechanism
  - Assume parallelism would yield better results if possible

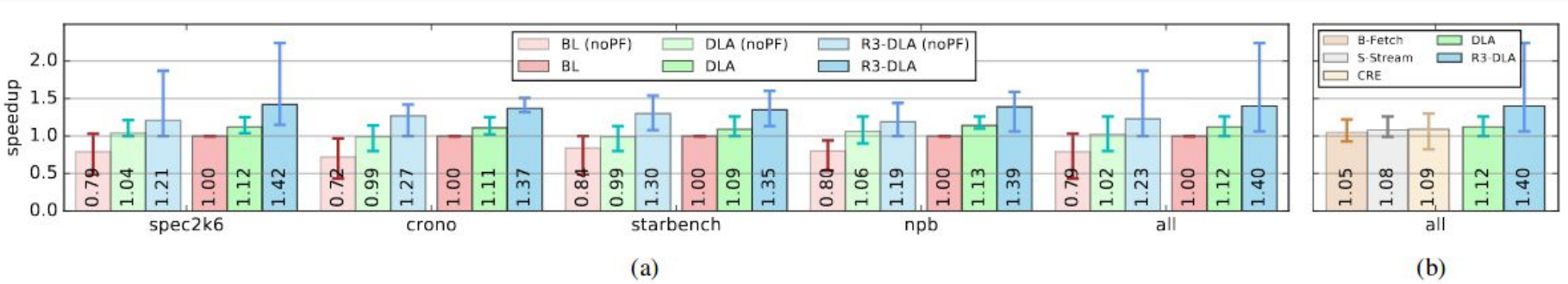


Figure 9. Performance gain over an aggressive baseline with BOP at L2. NoPF shows the normalized performance of a baseline configuration with no prefetcher.

Results

- Improvement on power over DLA
  - Still wide variation in results

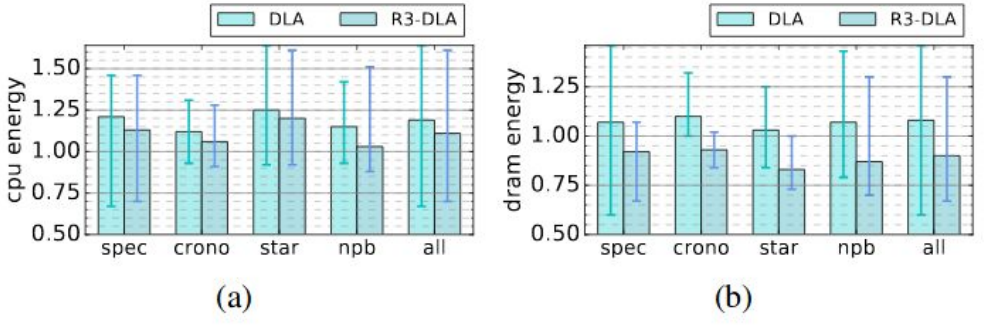


Figure 10. Comparison of energy normalized to baseline spent in (a) cpu (b) dram by DLA and R3-DLA.

Results

- Still far from SMT throughout
  - Much better than a single “wide core”

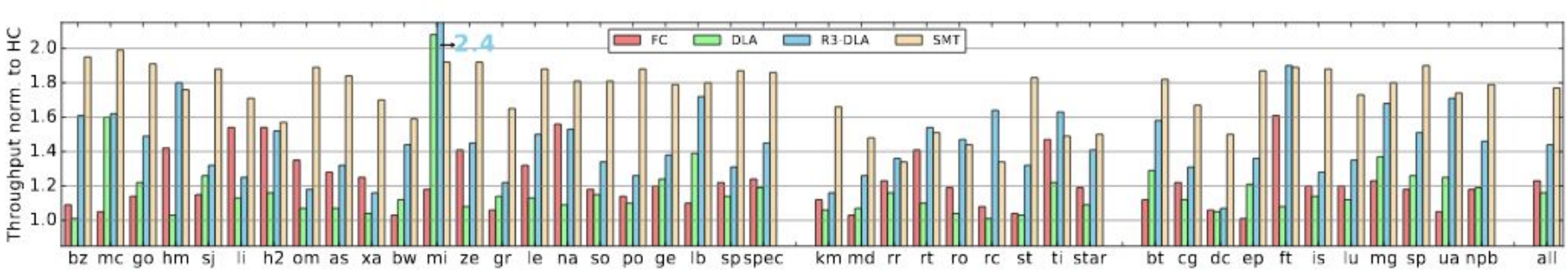


Figure 11. Comparison of normalized throughput obtained by R3-DLA using a single wider core over a half-core (HC).

Results

- Changing the skeleton matters!
  - Different configurations better at different points

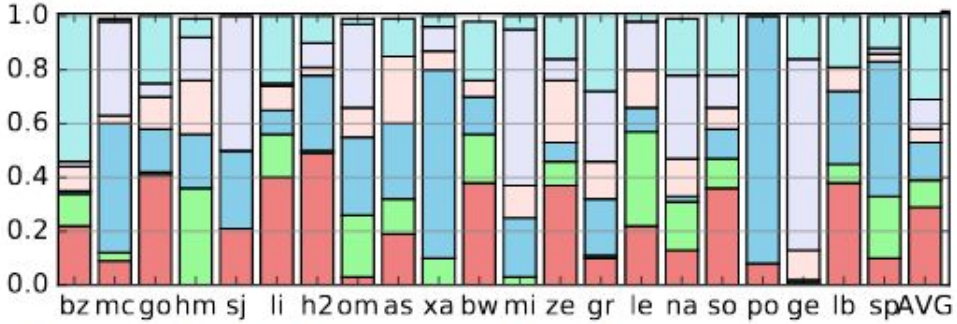


Figure 15. The distribution of skeleton versions chosen during online tuning.

Conclusion

Conclusion

- Single thread performance is still important!
- Idle resources elsewhere can improve performance on one core
- Combining techniques can be more powerful than the effects in isolation