P3.An Overview on Binary Translation

Nuno Paulino
CTM
INESC TEC

INESCTEC
TECNOLOGIA E CIÊNCIA
LABORATÓRIO ASSOCIADO
COORDENADO POR INESCPORTO

PEPCC

Overview

- Title: *Improving Performance and Energy Consumption in Embedded Systems via Binary Acceleration: A Survey (https://doi.org/10.1145/3369764)*
- Journal: *ACM Computing Surveys*
  - Q1 in Computer Science
  - Impact Factor: 6.13 (@2018)
  - Approx. 10-20 citations per paper
- 35 Page survey paper
  - Submitted: 11 Feb. 2019 → R1: 30 Mar. 2019 → R2: 3 Sep. 2019 → Accepted: 2 Oct. 2019!
- Project: *PEPCC (Power Efficiency and Performance for Embedded and HPC Systems with Custom CGRAs)*
- Keywords: *Surveys and overviews; Hardware Accelerators*
  - Binary acceleration, instruction traces, (automated) hardware synthesis

Specific focus: dev. of tools, toolflow, and methodologies, for HW/SW design

Rationale

- After single-row accelerator and MicroBlaze binary translation (session P1), we wanted
  - Multiple instruction set support
  - Detect / extract greater code blocks
    - Multiple-paths
    - Nested Loops
  - Explore CDFG transformations / optimizations
    - Splitting graphs
    - Extracting address generation patterns (session P7)
    - Node fusion
  - Explore different target architectures and scheduling techniques
- → Survey to characterize general panorama

50 Years of CMOS Processor Technology

- Dennard Scaling
  - Scale down
  - Voltage down
  - MHz up
  - Heat dissipation → constant
- Too small → current leakage!
- 2005 → End of Single-core scaling
- How far can Multi-Core go?
  - Dark Silicon
  - Amdahl's Law

15 Years of incremental improvements...

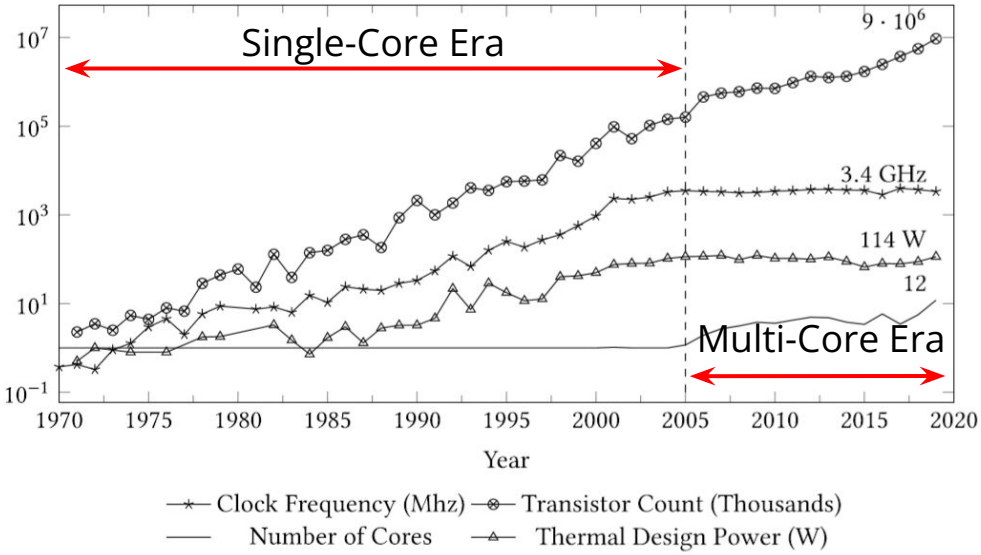


Fig. 1. Trends for desktop and server grade processors throughout the last 50 years, built from 950 data points from CPU DB [23] and Intel's and AMD's product pages

What's Left to Explore?

- Single-core workloads aren't nearly as optimized as they could be!
  - "*(...) a large amount of ILP that is not being exploited within a 128 to 512 instruction distance.*"
  - *"Compared to real machines as much as 929x more ILP is available."*
  - *"We found the upper bound on ILP averaged around 200 instructions/cycle (...)"*
    - *Fatehi et al., "ILP and TLP in shared memory applications: A limit study", 2014, 23rd International Conference on Parallel Architecture and Compilation Techniques (PACT)*, Edmonton, AB, 2014, pp. 113-125.
- This potential is not fully explored by typical techniques
  - e.g., VLIWs, Superscalar processors, threading, software pipelining, etc

Ergo, application-/workload-specific compute architectures!

How?

Binary Translation?

- It is a type of re-compilation:
  - Binary code from one ISA → same ISA + transformations OR another target ISA
- Transforming static code
  - Examples: ISA compatibility without re-compilation, instrumentation
- Transforming executing code (i.e. traces)
  - Examples: Java VM, Valgrind, Virtual Machines

This survey: translate binary to hardware descriptions/configurations

Further reading: *Wenzl et al., 2019. From Hack to Elaborate Technique – A Survey on Binary Rewriting. ACM Comput. Surv. 52, 3, Article 49 (June 2019), 37 pages.*

Binary Translation for Acceleration

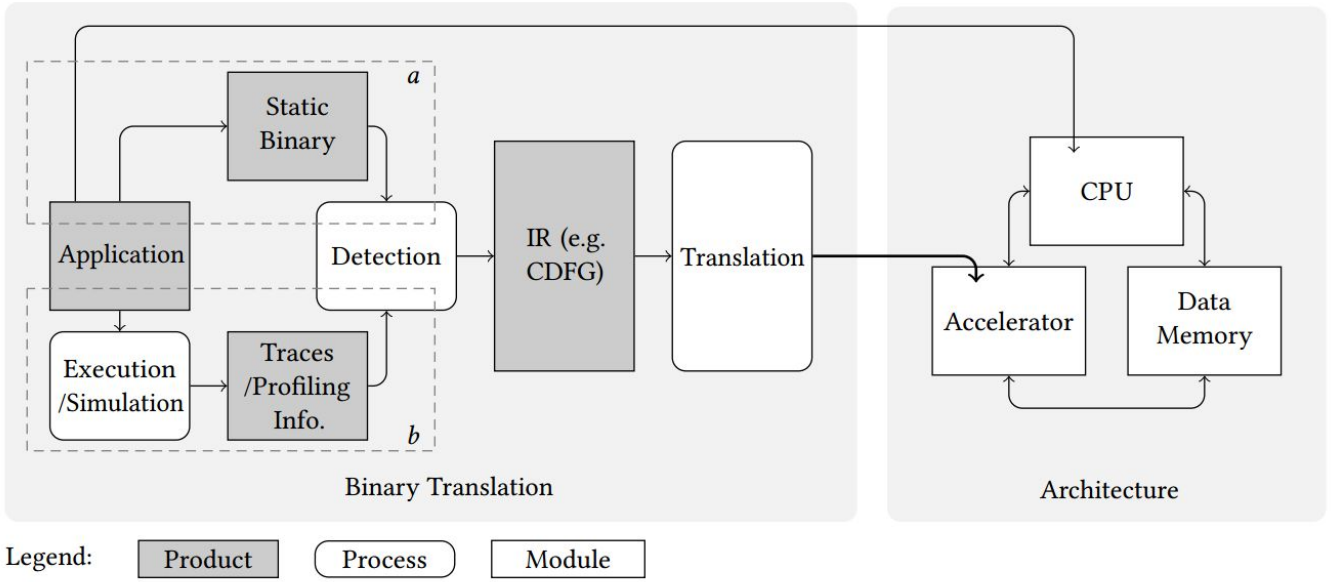


Fig. 2. High-level generic representation of binary translation flow into custom hardware (*a* - flows based on complied binary, *b* - flows based on binary traces)

Binary Segments

- Binary instruction lists
- Detected from
  - Static binary
  - Instruction traces
- 4 Major types
  - Frequent sequences
  - Basic Blocks
  - Acyclic blocks
  - Cyclic blocks

Transform sequence → Exploit ILP!

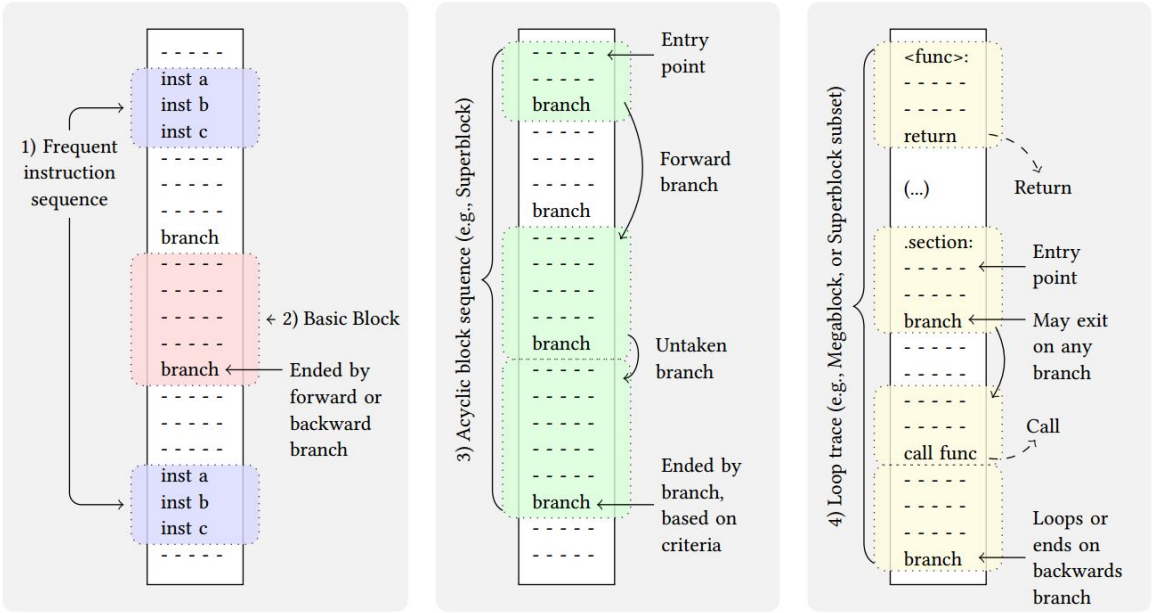


Fig. 3. Four types of binary segments than can be extracted from sequences of binary instructions

Binary Segments - Detection

- Online Detection
  - + More transparent
  - + Profile data
  - - Restricted
  - - Short segments
  - - Difficult translation

- Offline Detection
  - + Unrestricted
  - + More information
  - +/- Compiler integration
  - - Less transparent
  - - More tools required (?)

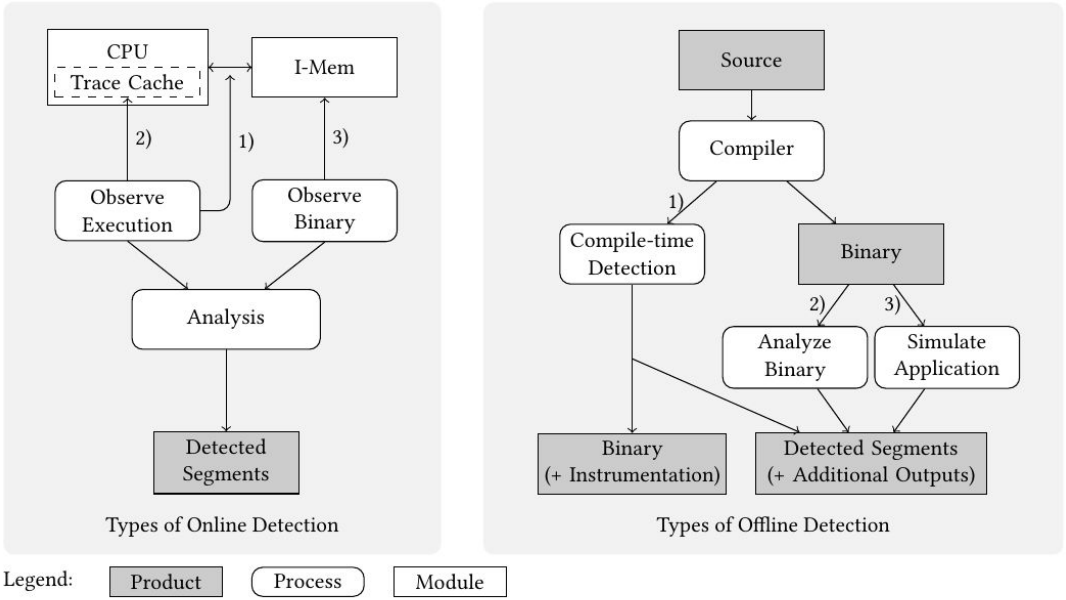


Fig. 4. Different methods for detection of binary segments

Binary Segments - Translation

- Extract ILP from Segments → CDFG
- Generate Accelerator Control
  - Assign operations to Accelerator Functional Units (FUs)
  - Generation of Custom Instructions
  - Scheduling
- Generate Accelerator Hardware
  - None: pre-designed
  - Template parameterization
  - Full HDL generation

+Specialization

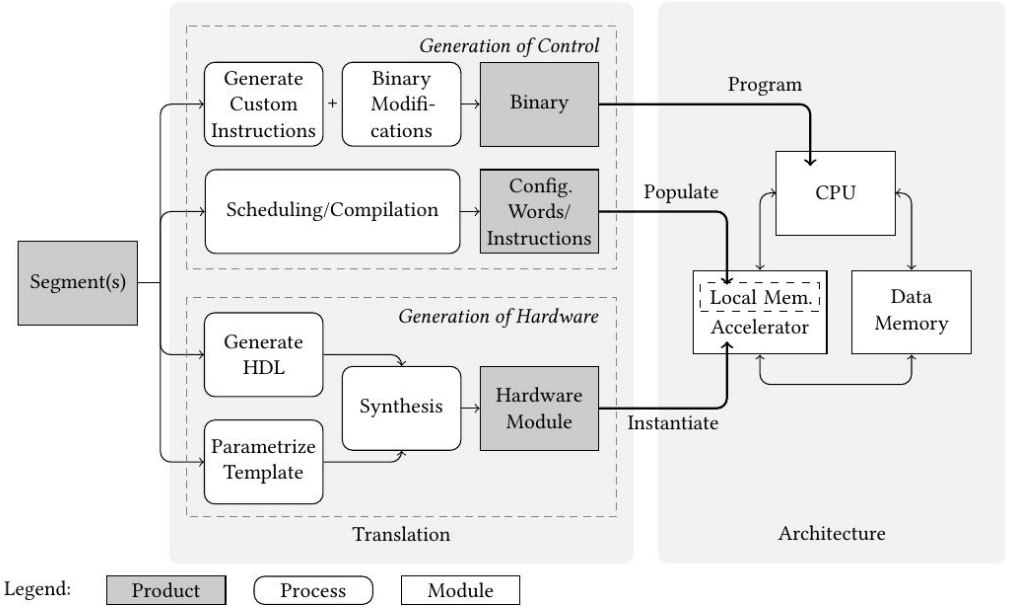


Fig. 5. Processes that may be involved in a translation step

Overview of State-of-the-Art Approaches

- What did I review?
  - +/- 30 papers that rely on some kind of translation of transformation of code to hardware
  - I created taxonomies to classify the binary translation, and accelerator architectures:

Features of Binary Translation Process

- Type of Segment
- Segment Detection
- Segment Translation
- Application Binary Modification
- Type of Acc. Architecture

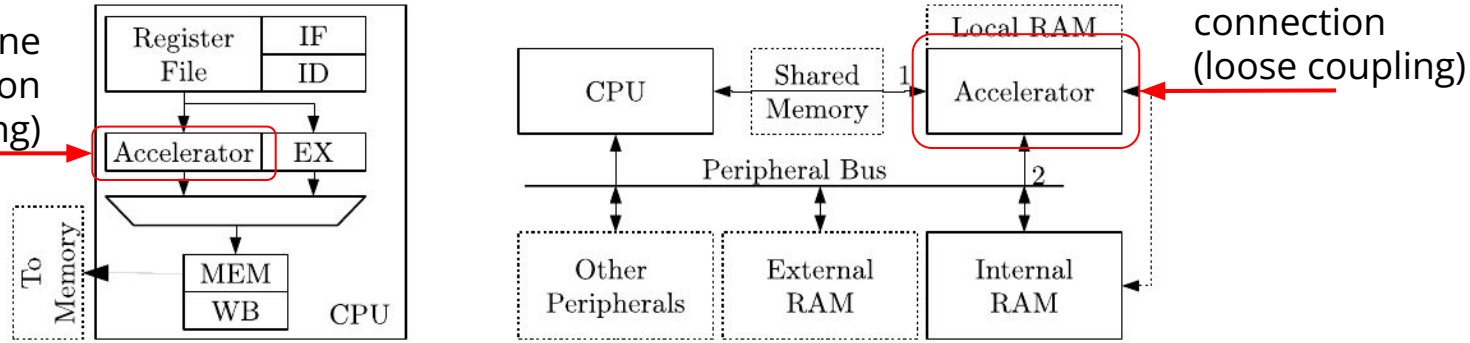
Features of Accelerator Architecture

- Acc./Host Interface
- Arrangement of Functional Units
- FU Interconnections
- Supported FU Operations
- Memory Access Capabilities
- Execution Model

Accelerator Architectures - System Level View

Pipeline integration (tight coupling)

Peripheral connection (loose coupling)



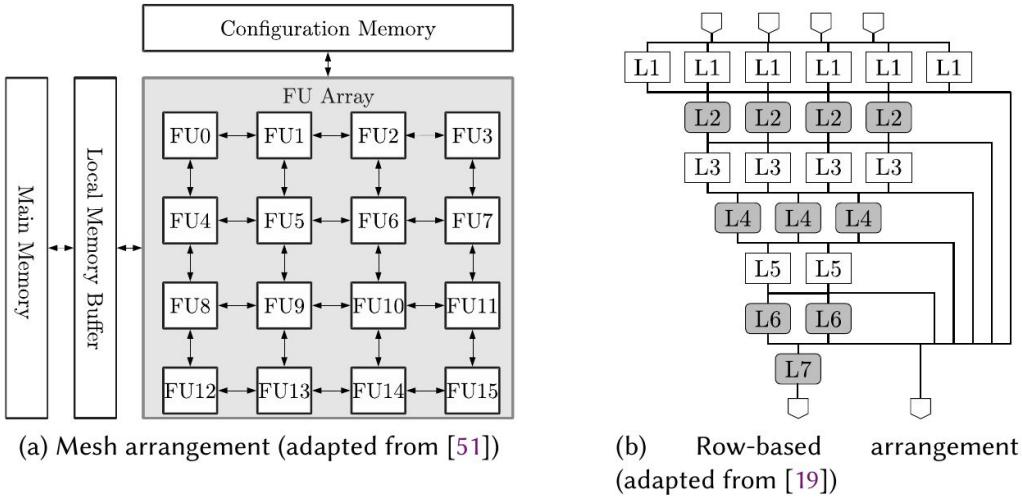
(a) Tight coupling into host processor pipeline

(b) Loose coupling via: 1) point-to- point connection; or 2) bus connection

Fig. 6. Different interfaces between host processor and accelerator (optional components in dotted lines)

Accelerator Architectures - Structure & Function

- Two design types:
  - Mesh-based
  - Row-based
- Mesh Designs
  - Apt for loop acceleration
  - Homogeneous
  - (More) Scalable
  - Peripheral-based
- Row Designs
  - Apt for instruction compression
  - Multi- or Single-Row
  - (More) Heterogeneous
  - Difficult to scale
  - Data directionality

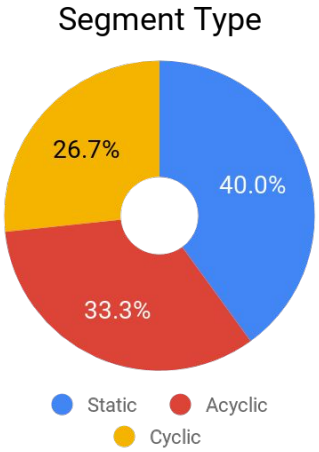


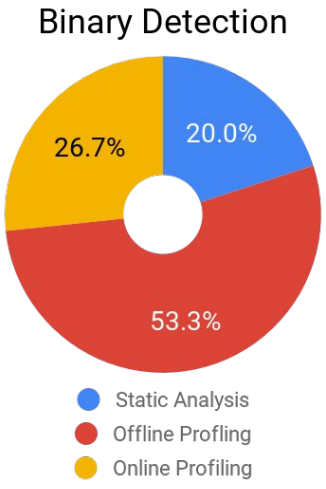
(a) Mesh arrangement (adapted from [51])

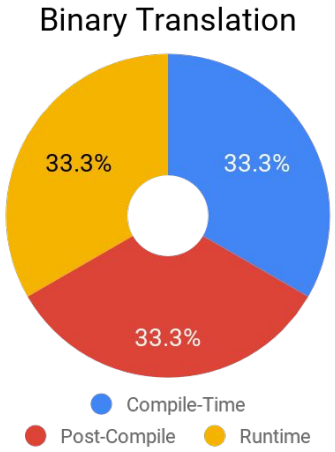
(b) Row-based arrangement (adapted from [19])

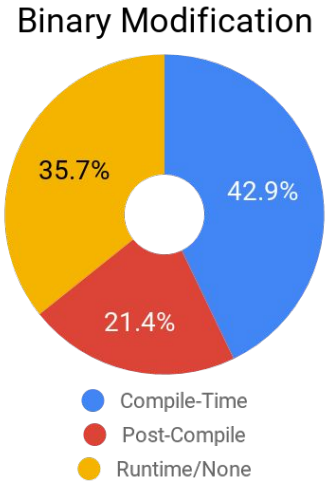
Fig. 7. Two *Functional Unit* arrangements and interconnections for accelerators

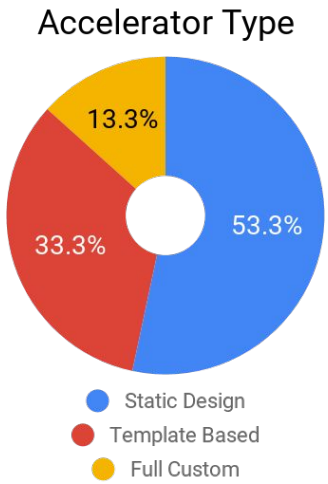
Summary - Overview of Binary Translation



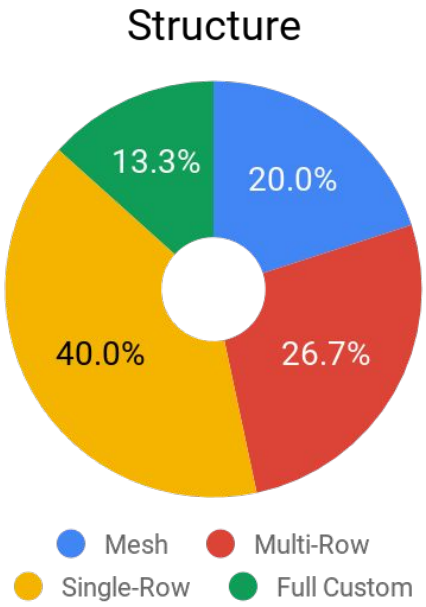


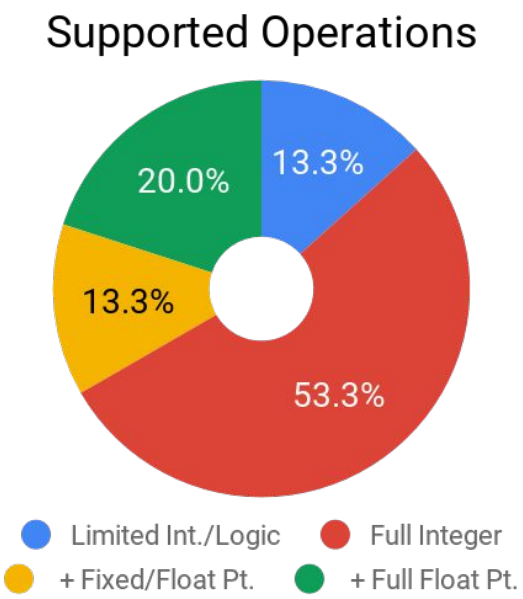


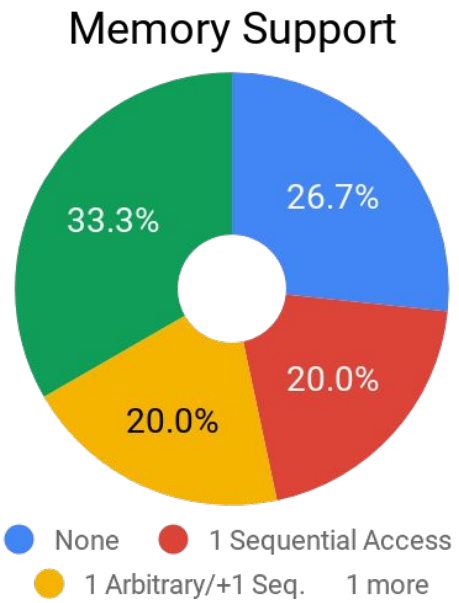




Summary - Overview of Accelerator Architectures







Summary - Performance and Power Improvements

Approach	1)	2)	3)	4)	5)	6)	7)	8)	9)	10)	11)	12)	13)	14)	15)
Speedup	3.2x	1.9x	2.6x	2.2x	2.2x	1.1x	5.6x	12.0x	2.5x	9.4x	1.1x	2.0x	7.1x	3.0x	3.3x
Power Reduction	2.9x	1.3x	2.2x	N/A	1.7x	1.5x	3.9x	12.0x	8.3x	N/A	N/A	N/A	1.6x	1.1x	11.0x

1) Warp; 2) ADEXOR; 3) DIM; 4) CCA; 5) DySE; 6) BERET; 7) CLA; 8) PLA; 9) PPA; 10) Paek et.al; 11) Chen et al.; 12) Ferreira et al.; 13) ASTRO; 14) Malazgirt et al.; 15) Rokicki et al.

Typical baselines/target systems:

- Single-thread, single-issue, bare-metal environments
- ARM, SPARC, Microblaze, MIPS based processors, VLIWs
  - i.e., RISC architectures → simpler binary segment detection and translation

Competing with higher-end multi-core devices is unlikely, but
improvements are promising for power-constrained embedded systems

Conclusion

- Pros:
  - Acceleration via analysis of instruction windows in compiled programs is demonstrated
  - Acceleration without use of new programming models, APIs, source modifications, etc
  - Potential for re-use of concept in emerging efforts heterogeneous hardware/software co-compilation
- Cons:
  - Application to real cases (beyond prototypes)
  - Support for multiple ISA; support for operating systems
  - Integration into end-system
  - Just-in-time translation (full transparency)
- We used this overview as a guide to start efforts on a Binary Translation Framework
- https://dl.acm.org/doi/10.1145/3369764
  - It's trending! 4 1,304