Binary Acceleration in Embedded Systems: A Survey

Nuno Paulino
CTM
INESC TEC

INESCTEC
TECNOLOGIA E CIÊNCIA
LABORATÓRIO ASSOCIADO
COORDENADO POR
INESCPORTO

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

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...

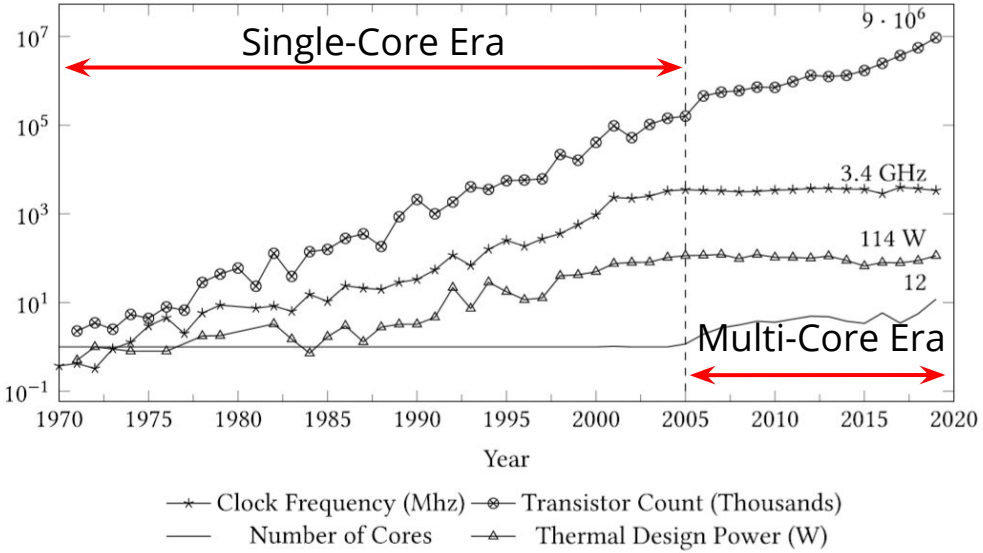


Fig. 1. Trends for desktop and server grade processors throughout the last 50 years, built from 950 data points from CPU DB [23] and Intel's and AMD's product pages

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

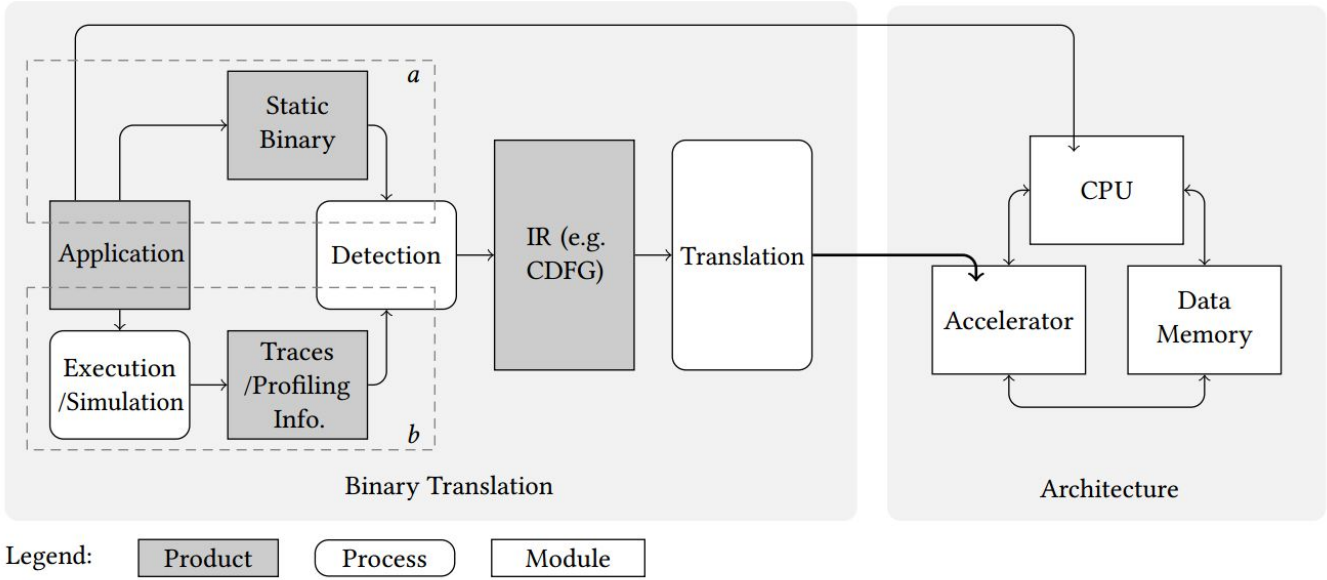


Fig. 2. High-level generic representation of binary translation flow into custom hardware (*a* - flows based on complied binary, *b* - flows based on binary traces)

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!

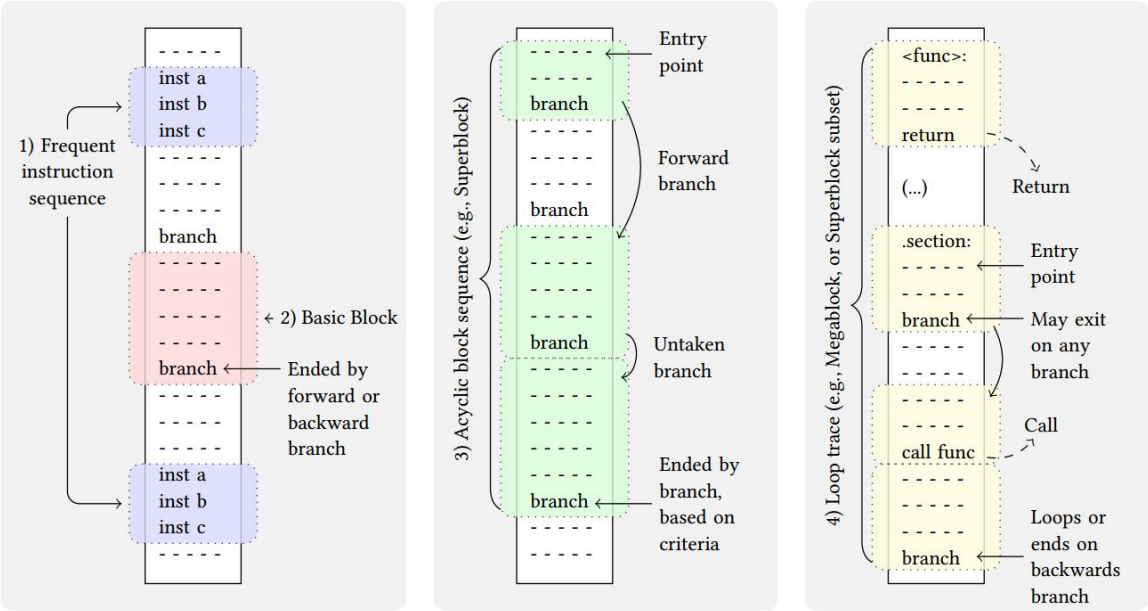


Fig. 3. Four types of binary segments than can be extracted from sequences of binary instructions

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 (?)

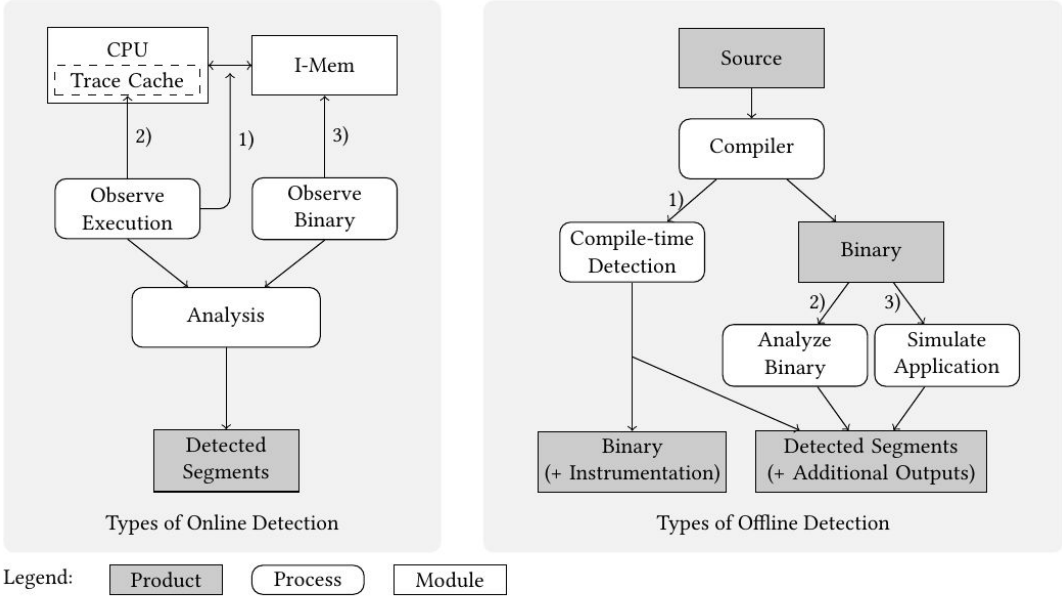


Fig. 4. Different methods for detection of binary segments

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

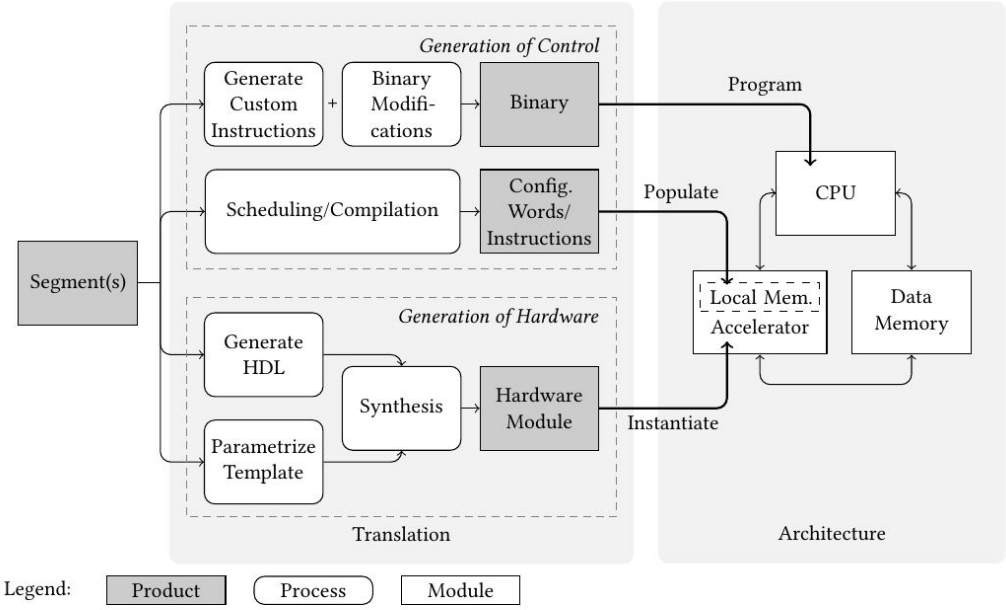
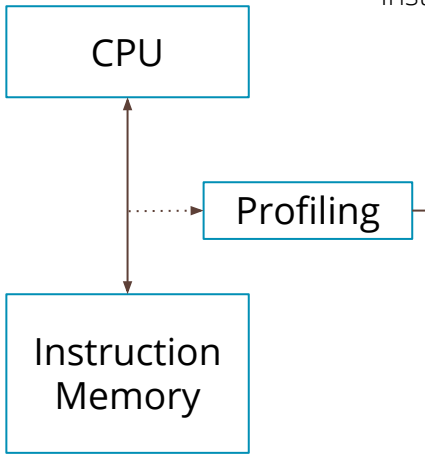


Fig. 5. Processes that may be involved in a translation step

Binary Segments - Example

- Example Megablock:

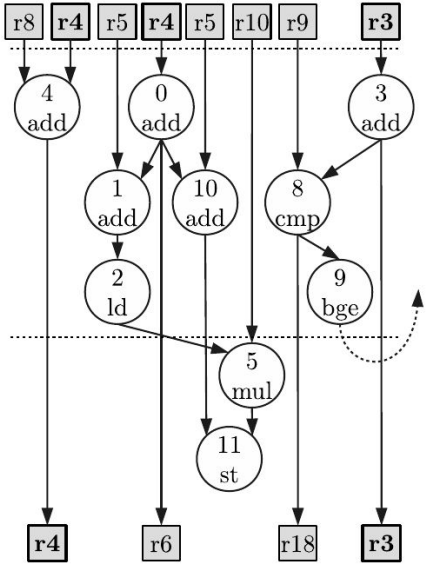


Instruction Stream

```
0x330 addk   r6, r4, r4
0x334 lhu    r7, r6, r5
0x338 addik  r3, r3, 1
0x33C addk   r4, r4, r8
0x340 mul    r7, r7, r10
0x344 addk   r7, r7, r12
0x348 bsrl   r7, r7, r11
0x34C cmpu   r18, r9, r3
0x350 bltid  r18, -32
0x354 sh     r7, r6, r5
```

Extract Instruction Parallelism

```
→ 0:add
→ 1:add + 2:load
→ 3:add
→ 4:add
→ 5:mul
→ 6:add
→ 7:srl
→ 8:i_cmpu
→ 9:branch (0x330)
→ 10:add + 11:store
```



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

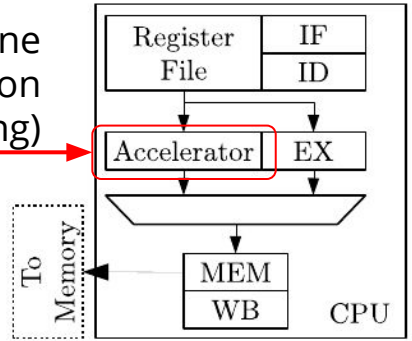
- Type of Segment
- Segment Detection
- Segment Translation
- Application Binary Modification
- Type of Acc. Architecture

Features of Accelerator Architecture

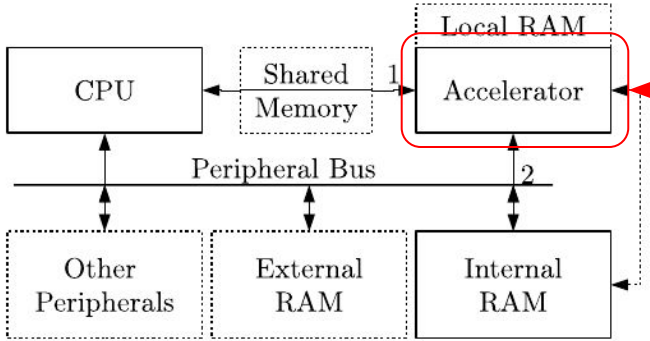
- Acc./Host Interface
- Arrangement of Functional Units
- FU Interconnections
- Supported FU Operations
- Memory Access Capabilities
- Execution Model

Accelerator Architectures - System Level View

Pipeline integration (tight coupling)



(a) Tight coupling into host processor pipeline



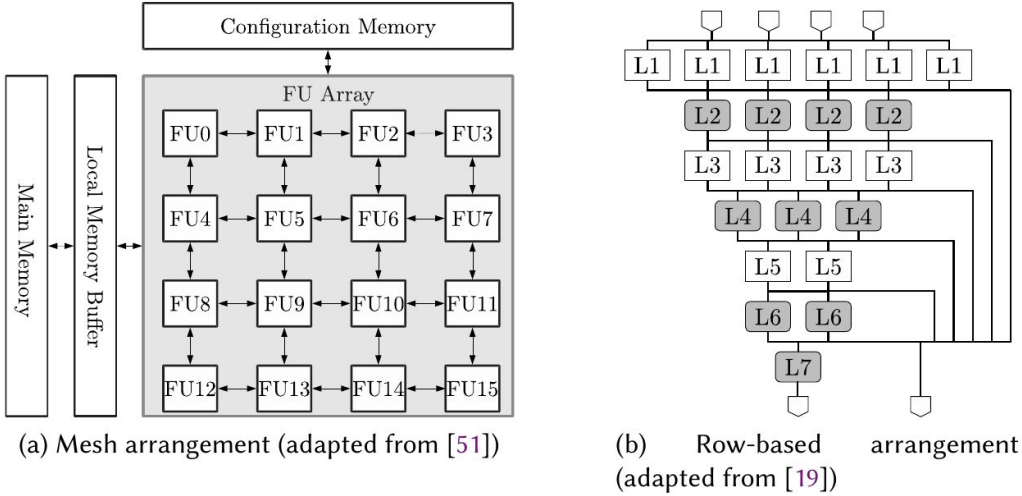
(b) Loose coupling via: 1) point-to- point connection; or 2) bus connection

Peripheral connection (loose coupling)

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



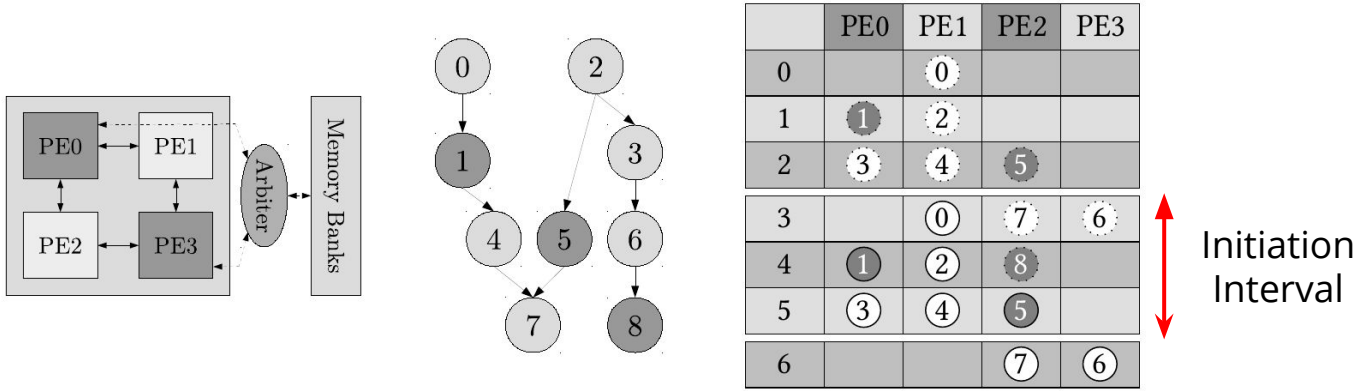
(a) Mesh arrangement (adapted from [51])

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

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

Accelerator Architectures - Execution Example

- Modulo-scheduling in a small mesh architecture

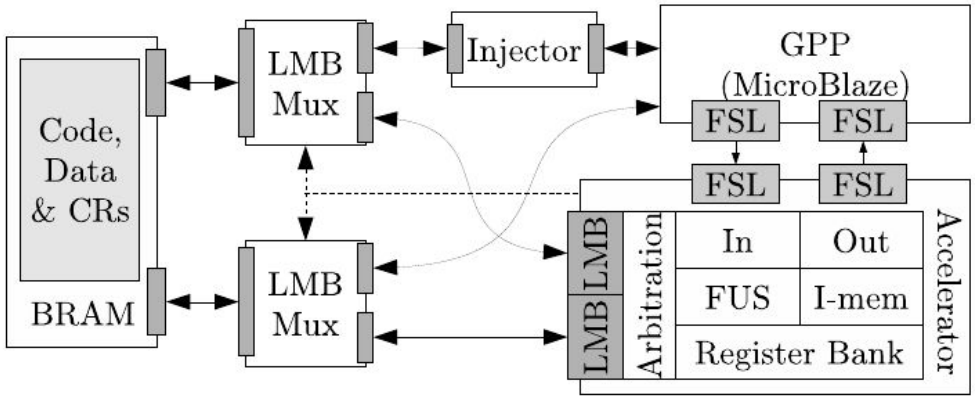


(a) target mesh architecture (b) target sub-graph (c) modulo reservation table

Fig. 8. Modulo scheduling performed on mesh arrays (adapted from [51]). The dark Processing Elements (PEs) in Figure 8a are capable of memory accesses, and dark nodes in Figure 8b are memory access operations. In Figure 8c, dashed nodes represent iteration *i-1*, and solid nodes represent iteration *i*.

My Work - Custom Loop Accelerator (CLA)

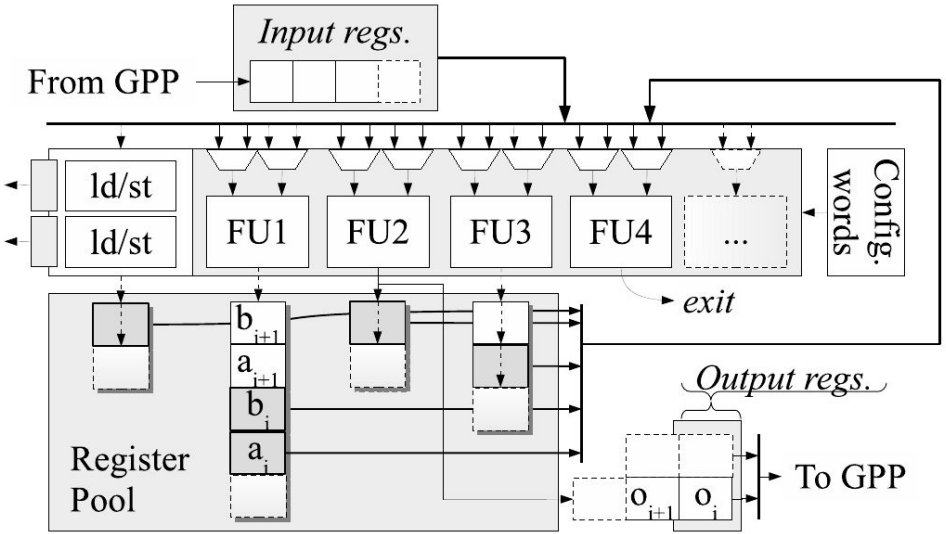
- Single-processor system
  - Bare-metal
  - FPGA Implementation
  - Soft-core 32 Bit RISC processor
- Accelerator
  - Automatically generated by offline profiling tools
  - Executes frequent loop paths
    - Exploits ILP
    - Exploits pipelining
- Experiments
  - MicroBlaze @100/150MHz
  - 5.6x Speedup (24 benchmarks)
- References: [16]



(b) System architecture

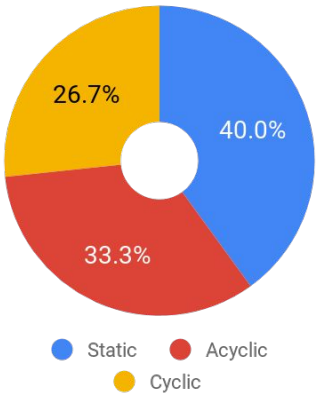
My Work - Custom Loop Accelerator (CLA)

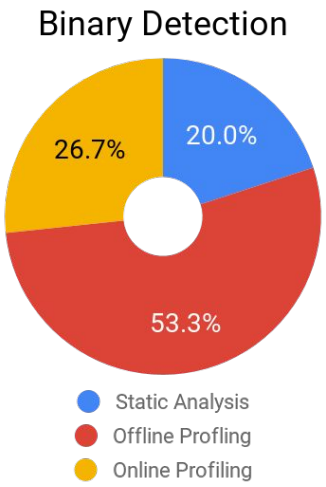
- Tailored for 1+ Loop traces
- Modulo-scheduled loops
  - Resource reutilization
- Customized row of single-function units
  - VLIW-like execution
  - Fully-pipelined, non blocking units
  - Tailored connectivity (min. required)
  - Minimum possible Initiation Intervals
  - Integer + Floating Point Arithmetic
  - Up to two arbitrary memory accesses
- Mixed Granularity Reconfiguration
  - Fast Context Switching
  - Dynamic Partial Reconfiguration

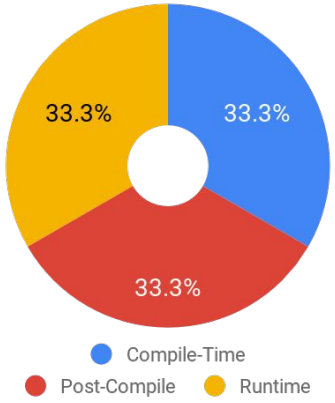


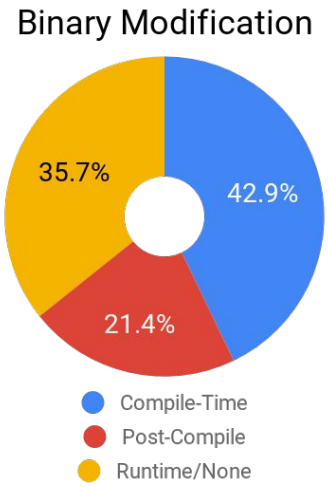
(a) CLA architecture

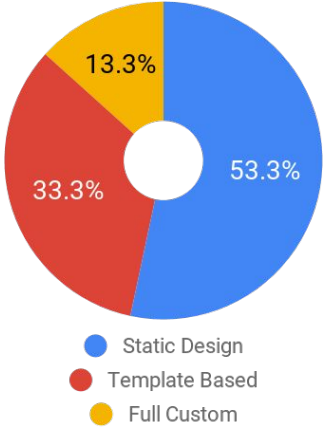
Summary - Overview of Binary Translation



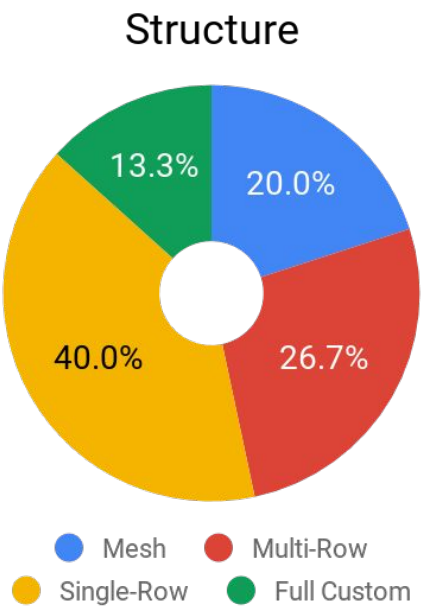


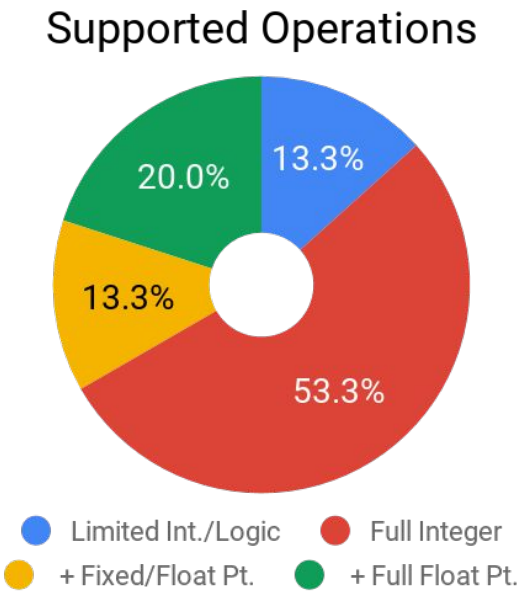


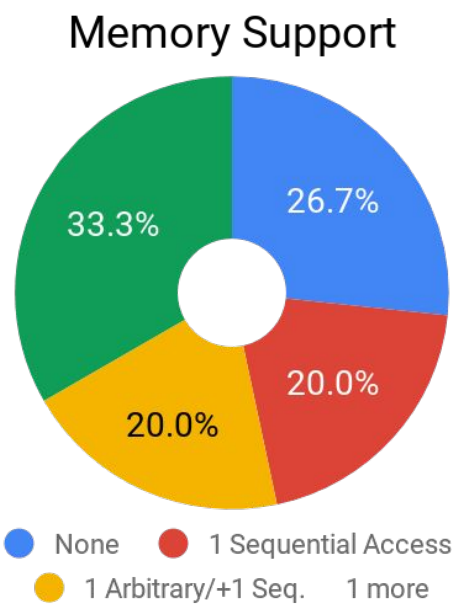




Summary - Overview of Accelerator Architectures







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

- Advantages
  - Acceleration of embedded applications by automatic specialization of hardware
  - Makes use of available resources if deployment platforms are FPGAs
  - Low effort acceleration of applications, abstracted from software development
- Open Issues?
  - Proper tools
  - Programming models
  - Support for multiple ISA
  - Integration into end-system
- Future trends?
  - Fully-fledged Multi-processor/Multi-core SoCs with FPGAs?
  - Desktop processors with reconfigurable hardware? e.g. Intel's Xeon+FPGA Family

References (Reviewed Approaches)

[1] Nathan Clark et al.. 2005. *An architecture framework for transparent instruction set customization in embedded processors*. In Proc. of the Intl. Symp. on Computer Architecture. 272–283.

[2] Kevin Fan et al. 2008. *Modulo Scheduling for Highly Customized Datapaths to Increase Hardware Reusability.* In Proc. of the IEEE/ACM Intl. Symp. on Code Generation and Optimization. 124–133.

[3] Kevin Fan et al. 2009. *Bridging the computation gap between programmable processors and hardwired accelerators.* In Proc. of the IEEE Intl. Symp. on High Performance Computer Architecture. 313–322.

[4] Hyunchul Park et al. 2009. *Polymorphic Pipeline Array: A flexible multicore accelerator with virtualized execution for mobile multimedia applications.* In Proc. of the IEEE/ACM Intl. Symp. on Microarch.. 370–380.

[5] Roman Lysecky and Frank Vahid. 2009. *Design and implementation of a MicroBlaze-based Warp Processor.* ACM Trans. on Embedded Computing Systems 8, 3 (April 2009), 22:1–22:22.

[6] Shantanu Gupta et al. 2011. *Bundled execution of recurring traces for energy-efficient general purpose processing.* In Proc. of the IEEE/ACM Intl. Symp. on Microarchitecture. 12–23.

[7] Jong Kyung Paek, Kiyoung Choi, and Jongeun Lee. 2011. *Binary acceleration using coarse-grained reconfigurable architecture.* SIGARCH Computer Architecture News 38, 4 (Jan. 2011), 33–39.

[8] Hamid Noori et al. 2012. *Improving performance and energy efficiency of embedded processors via post-fabrication instruction set customization.* The Journal of Supercomputing 60, 2 (May 2012), 196–222.

[9] Venkatraman Govindaraju et al. 2011. *Dynamically specialized datapaths for energy efficient computing.* In Proc. of the IEEE Intl. Symp. on High Performance Computer Architecture. 503–514.

[10] Antonio C. S. Beck et al. 2008. *Transparent Reconfigurable Acceleration for Heterogeneous Embedded Applications.* In Proc. of the Design, Automation and Test in Europe Conf. and Exhibition. 1208–1213.

[11] Liang Chen, Joseph Tarango, Tulika Mitra, and Philip Brisk. 2013. *A Just-in-Time customizable processor.* In Proc. of the IEEE/ACM Intl. Conf. on Computer-Aided Design. 524–531.

[12] Ricardo Ferreira et al. 2014. *A run-time modulo scheduling by using a binary translation mechanism.* In Proc. of the Intl. Conf. on Embedded Comp. Systems: Architectures, Modeling, and Simulation. 75–82.

[13] Mingjie Lin et al. 2015. *ASTRO: Synthesizing application-specific reconfigurable hardware traces to exploit memory-level parallelism.* Microprocessors and Microsystems 39, 7 (2015), 553–564.

[14] Gorker Alp Malazgirt, Arda Yurdakulm, and Small Niar. 2015. *Customizing VLIW processors from dynamically profiled execution traces.* Microprocessors and Microsystems 39, 8 (2015), 656–673.

[15] Simon Rokicki, Erven Rohou, and Steven Derrien. 2017. *Hardware-accelerated dynamic binary translation.* In Proc. of the Design, Automation and Test in Europe Conf. and Exhibition. 1062–1067.

[16] Nuno Paulino et al. 2017. *Generation of customized accelerators for loop pipelining of binary instruction traces.* IEEE Trans. on Very Large Scale Integration Systems 25, 1 (Jan. 2017), 21–34.

WARP Processor

- Custom reconf. fabric (1)
- Runtime binary profiling (2)
  - Hot Basic Blocks (HBBs)
- Runtime binary translation (3)
  - Disassembly
  - Binary Modification
- Experiments
  - 0.18um simulation
  - MicroBlaze @100MHz
  - 5.1x speedup (6 int. bench.)
- References: [5]

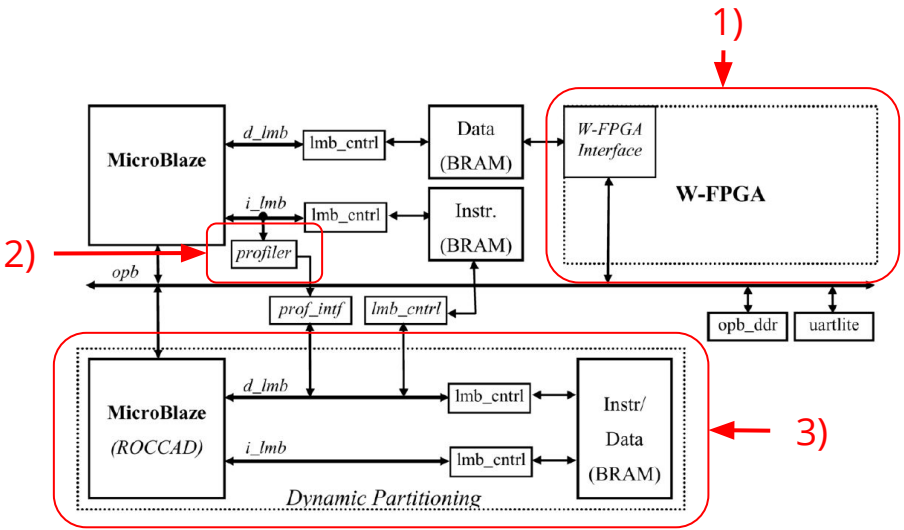


Fig. 9. The MicroBlaze-based Warp processor system. An additional processor performs runtime binary segment detection and translation [58].

AMBER/ADEXOR

- Rows of multi-function homogeneous units
  - Integrated into pipeline
- Offline binary profiling
  - Hot Basic Blocks (HBBs)
- Offline binary translation
  - Operation allocation +
  - Creation of custom instructions
- Experiments
  - 0.18um simulation
  - MIPS-based @300MHz
  - 1.2x speedup (14 int. bench.)
- References: [8]

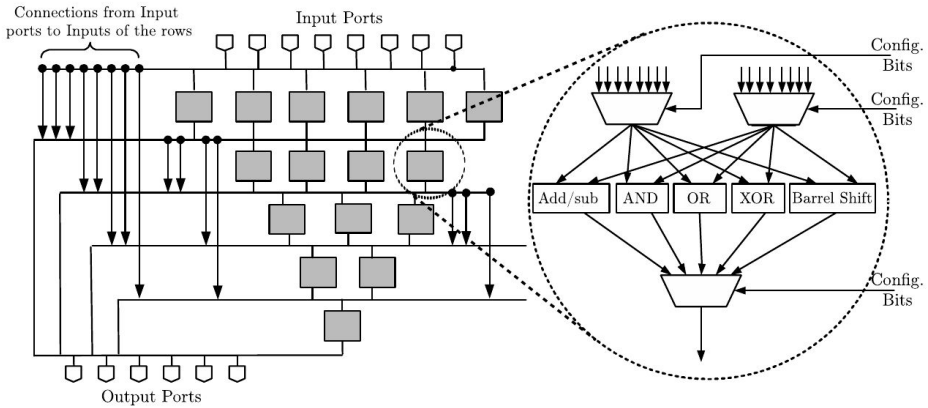


Fig. 10. FU array coupled to the processor pipeline in the AMBER/ADEXOR approach (adapted from [68]).

Dynamic Instruction Merging (DIM)

- Homogeneous rows of heterogeneous FUs
  - Integrated into the decode stage
- Online binary profiling
  - Hot Basic Blocks
- Online binary translation
  - ASAP scheduling into array
  - Generation of control bits
- Experiments
  - 0.18um simulation
  - MIPS-based processor @
  - 2.6x speedup (18 benchmarks)
- References: [10]

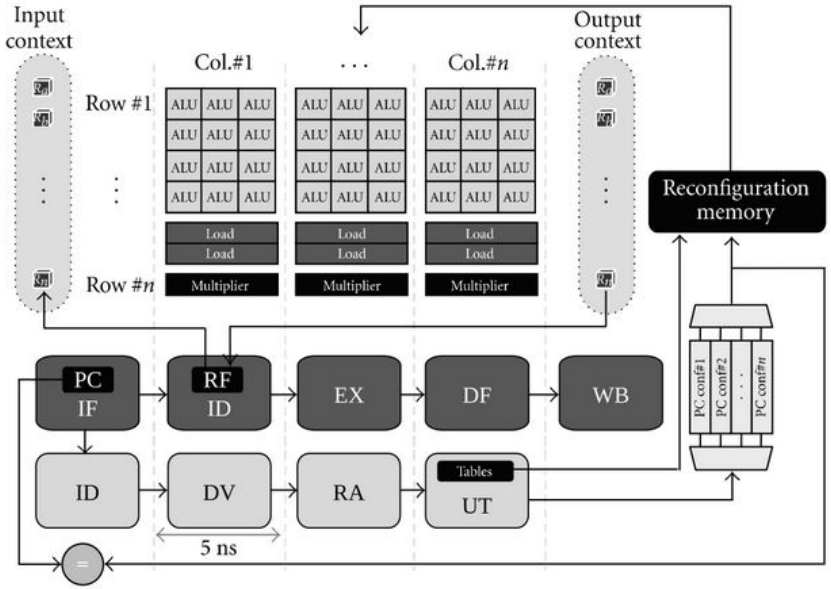


Fig. 11. DIM approach: runtime binary translation for a tightly integrated array (adapted from [80]).

Configurable Compute Accelerator (CCA)

- Rows of single-function heterogeneous units (1)
  - Integrated into pipeline
- On/Offline binary profiling (2)
  - Sequences of HBBs
- On/Offline binary translation (3)
  - Creation of CCA configs, replacement of original code
- Experiments
  - 4-issue ARM (simulation)
  - 1.3x speedup (11 int. bench.)
- References: [1]

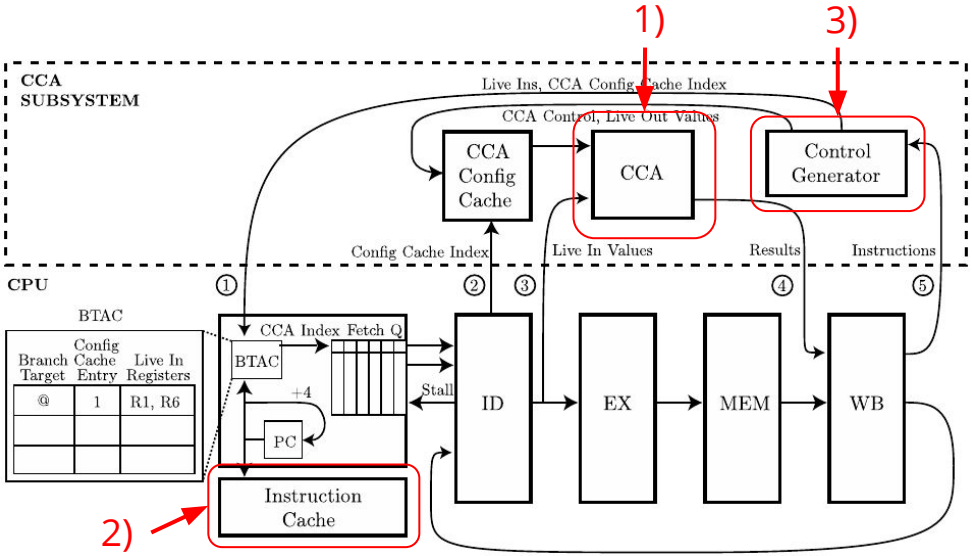


Fig. 12. The CCA array is coupled into the processor pipeline. Execution is shifted towards the CCA after generating configurations for offline delimited regions of code [17].

Dynamically Specialized Execution (DySE)

- Multi-directional mesh of heterogeneous units
  - Integrated into pipeline
- Offline binary profiling
  - Frequent BB loop paths
- Offline binary translation
  - Operation allocation +
  - Creation of custom instructions
- Experiments
  - 55nm simulation
  - SPARC Based processor
  - 2.2x speedup (23 benchmarks)
- References: [9]

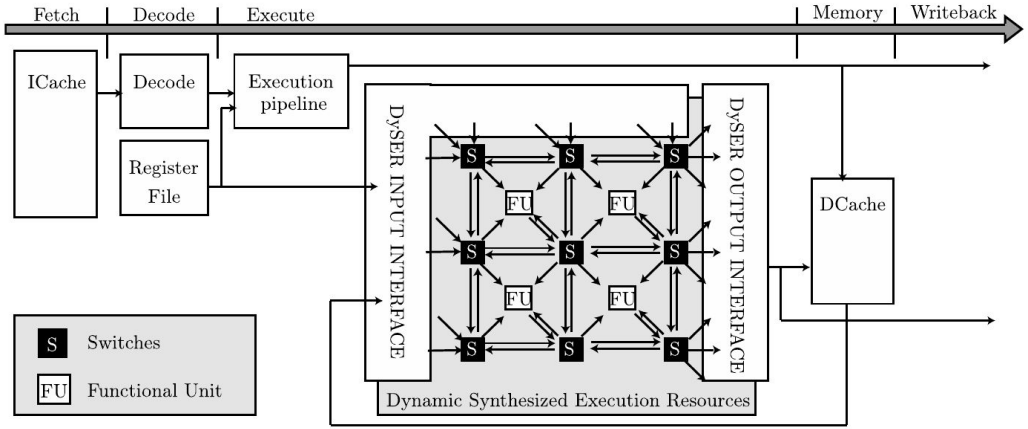


Fig. 13. DySER Architecture and Organization (adapted from [40])

Bundled Execution of Recurring Traces (BERET)

- Set of sub-graph accelerators
  - Integrated into pipeline
- Offline binary profiling
  - Compile-time superblock detection
- Offline binary translation
  - Static set of SEBs
  - Assign sub-graphs of the superlocks to specific SEBs
- Experiments
  - 65nm evaluation
  - ARM @800MHz
  - 1.1x speedup(12 benchmarks)
- References: [6]

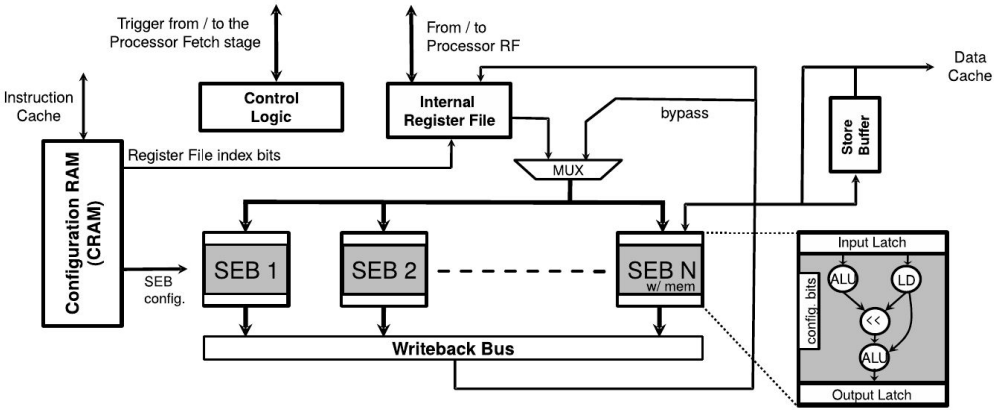
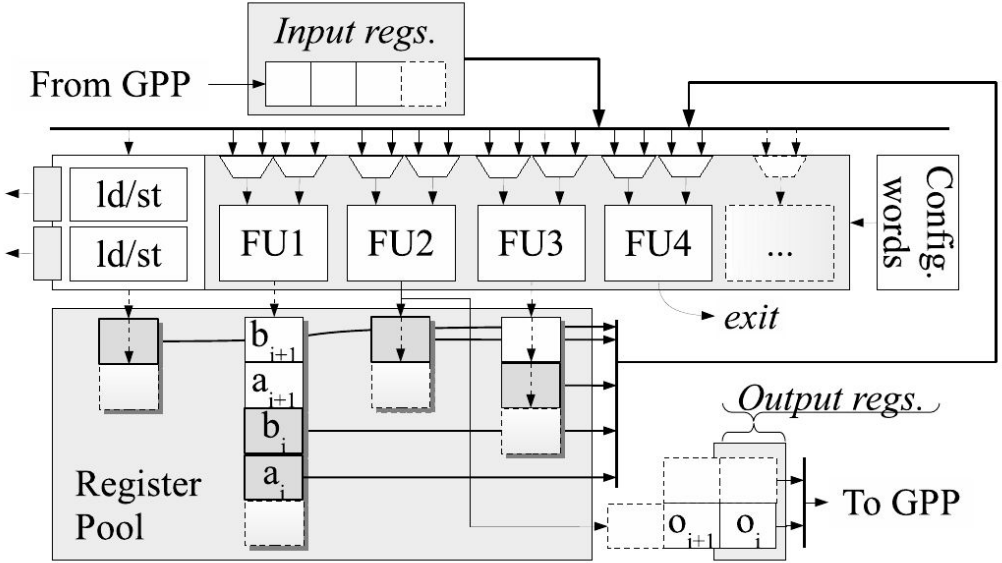


Fig. 14. *Sub-graph Execution Blocks* (SEBs) in BERET coupled to processor pipeline (adapted from [44]).

Custom Loop Accelerator (CLA)

- Single-Row of Single Function Units
  - Peripheral Connection
- Offline binary profiling
  - Frequent loop paths
- Offline binary translation
  - Customize row in function of modulo-scheduling loops
- Experiments
  - FPGA Implementation
  - MicroBlaze @100/150MHz
  - 5.6x Speedup (24 benchmarks)
- References: [16]



(a) CLA architecture