Using GCC Analysis Techniques to Enable Parallel Memory Accesses in HLS

Johanna Rohde, Christian Hochberger

Date: 7.9.2017

TU Darmstadt

Computer Systems Group

Outline

- Background
  - SpartanMC Toolchain
  - GCC
  - Hardware Plugin
- Approach
  - Scalar Evolution Analysis
  - Memory Model
- Evaluation
- Conclusion

Background

SpartanMC SoC Kit

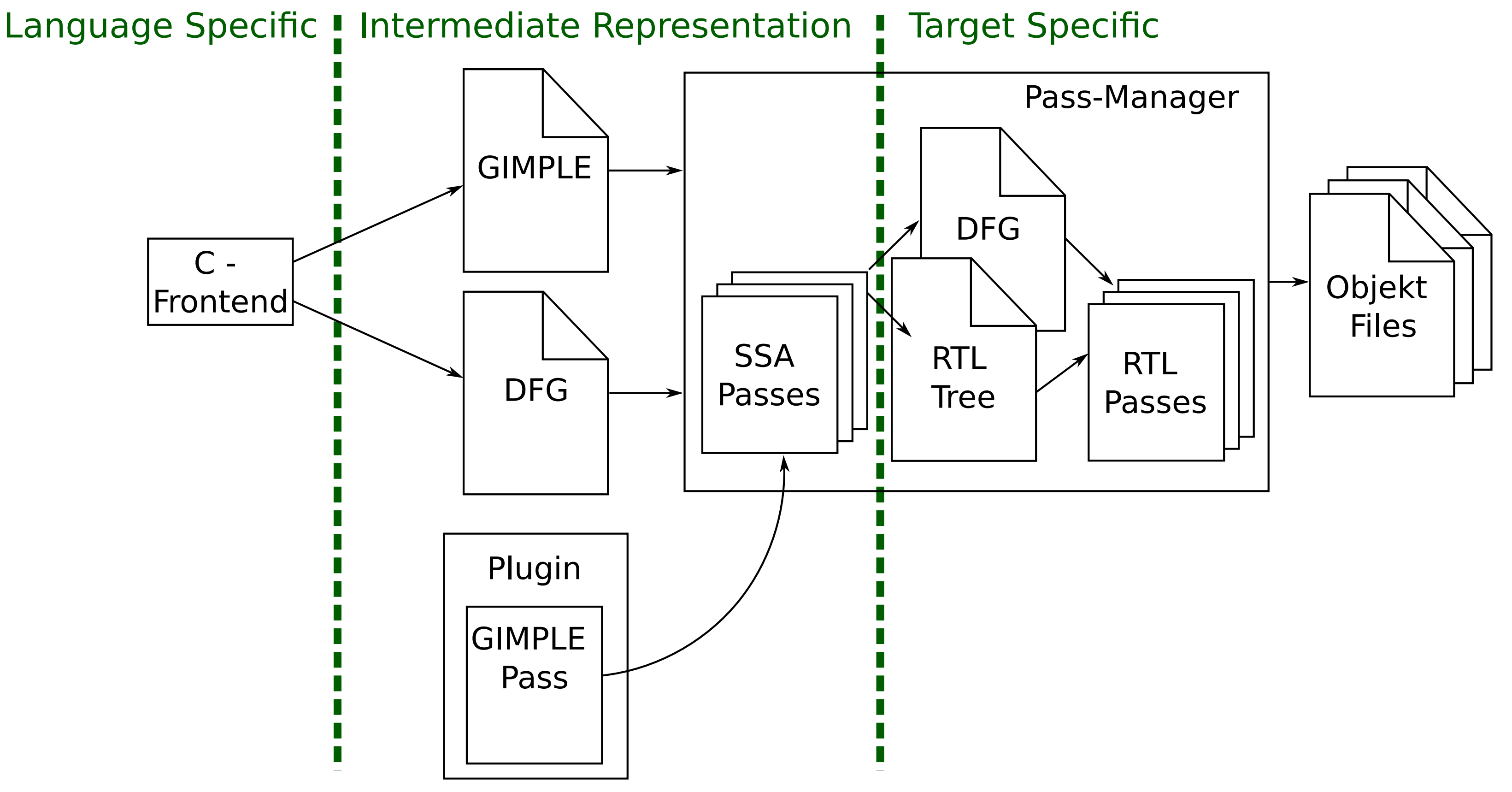
- 18 Bit Soft-Core
  - Tradeoff between 8 and 32 bit
  - FPGA use 18 bit for BRAMs and DSPs
  - 2 additional bit increase code density
- Graphical system configurator
- Peripheral components
- Simulator
- Supported FPGAs
  - Xilinx: Spartan 3/6, 7er Series, …
  - Altera: Cyclon 4
  - Lattice: ECP 3/5

Interested?
www.spartanmc.de

Background
Compiler Toolchain

- GCC
  - Support for 2-address-machines
  - Wide range of optimizations
    - e.g. Polyhedral framework
  - Widely used
  - Currently using GCC 7.1

Background
Compile-Flow

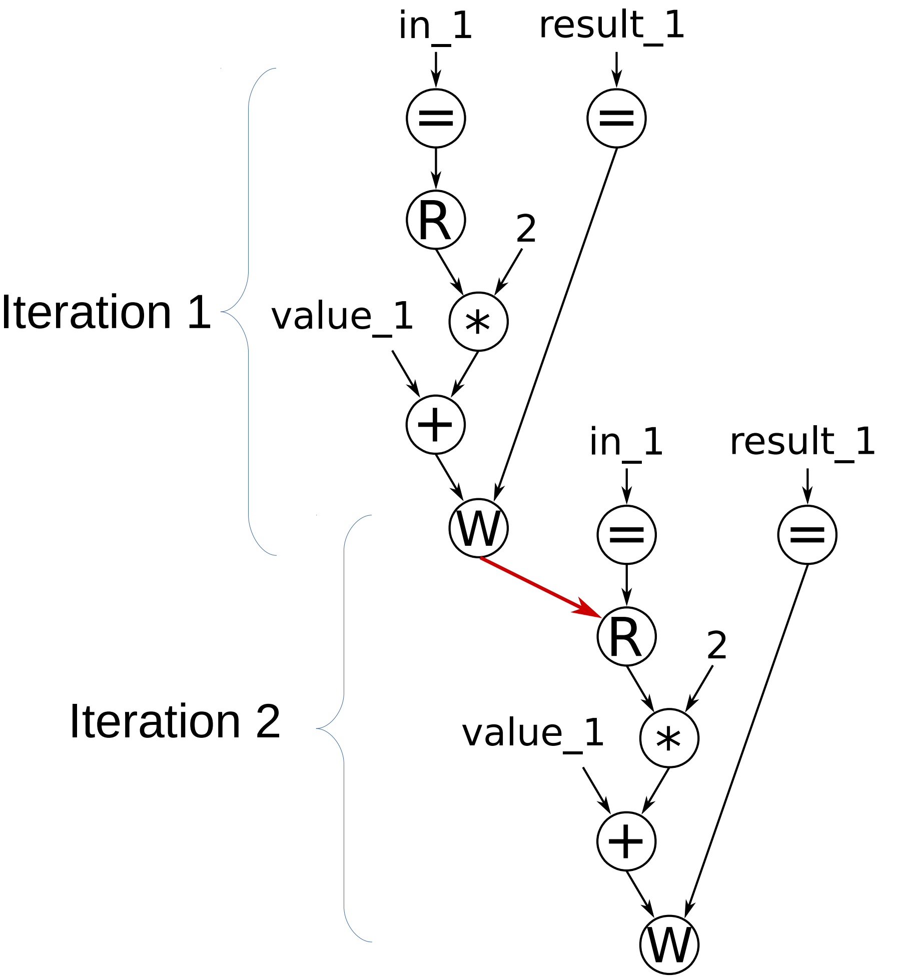


Background
Hardware Accelerator Plugin

- A specific SpartanMC design is generated for every application
- Automatically integrate hardware accelerators
  - Statical code analysis at compile time
  - Transparent to the user
  - No annotations or further information needed

Background
Optimizations

- Loop pipelining
  - Overlapping iterations
  - Problem: order of memory accesses
  - Increases initiation interval



Approach
Chain of Recurrences

- Express evolution of a variable
- Basic Recurrence (BR)

$$f=\{\varphi_0,\Theta,f_1\},\quad \Theta\in\{+,*\}$$

$\longrightarrow$

$$f(i)=\{\varphi_0,+,f_1\}(i)=\varphi_0+\sum_{j=0}^{i-1} f_1(j)$$

$$f(i)=\{\varphi_0,*,f_1\}(i)=\varphi_0+\prod_{j=0}^{i-1} f_1(j)$$

$$for\ 0\leq i<N$$

- Tree of Recurrence (TREC)

$$\Phi=\{\Phi_a,+,\Phi_b\}\text{ or }\Phi=c$$

Approach
Scalar Evolution Analysis

- Implemented in the GCC
- Calculates a TREC for every expression
- Example:

```
for(int i = 0; i < N; i++) {        \\loop 1
    for(int j = 0; j < M; j++{      \\loop 2
        A[i*M+j] = ….;
    }
}
```

→

$$i \rightarrow \{0,+,1\}_1$$

$$i*M \rightarrow \{0,+,1\}_1 * M = \{0,+,M\}_1$$

$$i*M+j \rightarrow \{\{0,+,M\}_1,+,1\}_2$$

Approach
Memory Access Model

- Anatomy of a (regular) memory access
  - `innermost_loop_behavior` struct
    - `tree base_address`
    - `tree offset` → always 0
    - `tree init`
    - `tree step`
  - `A = {(base + offset + init), +, step}`
- Pair
  - Dependency between
    - Read + Write
    - Write + Write

Approach
Memory Access Model

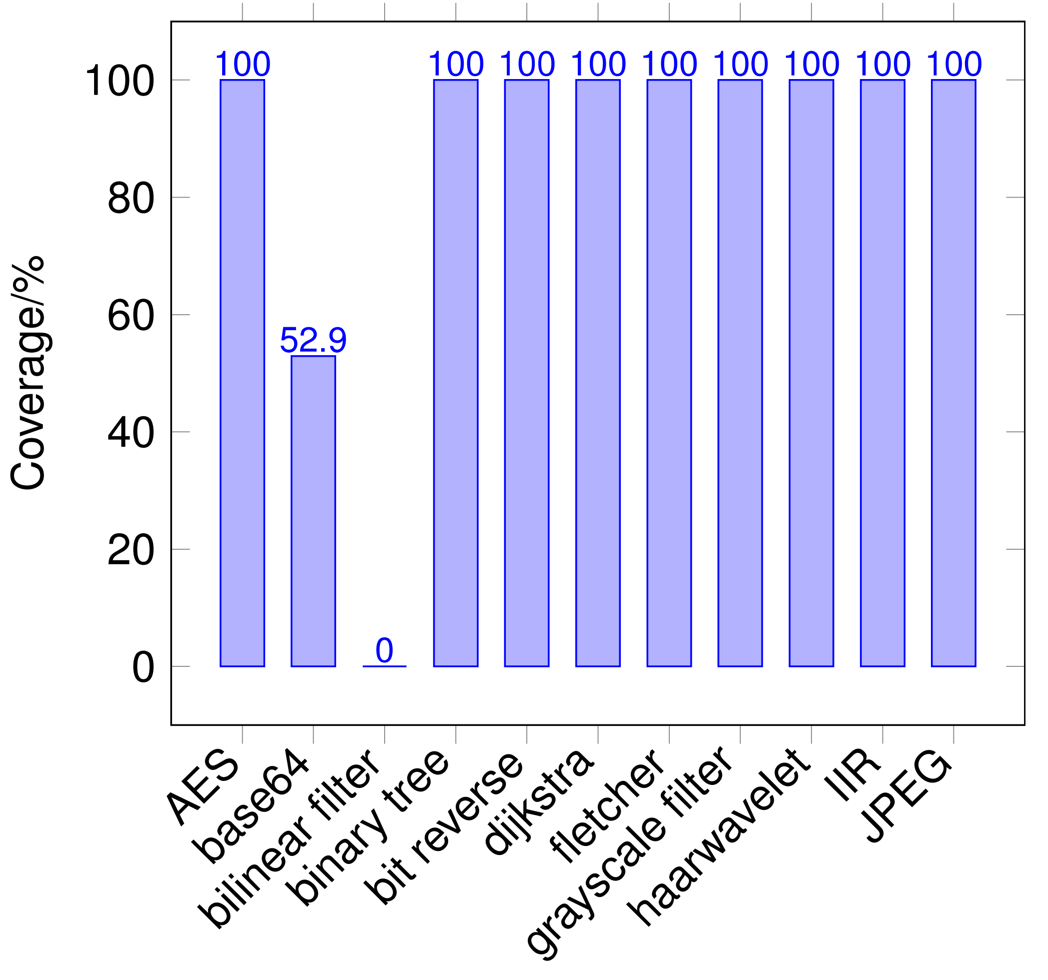
- Static Pairs
  - Analyzable at compile time
  - Base address equals
  - Step size equals
- Dynamic Pairs
  - Analyzable at runtime
  - Step size equals
  - $\Delta base + \Delta init + n * step = 0$
- Critical Pairs
  - Not analyzable

Evaluation
Benchmarks

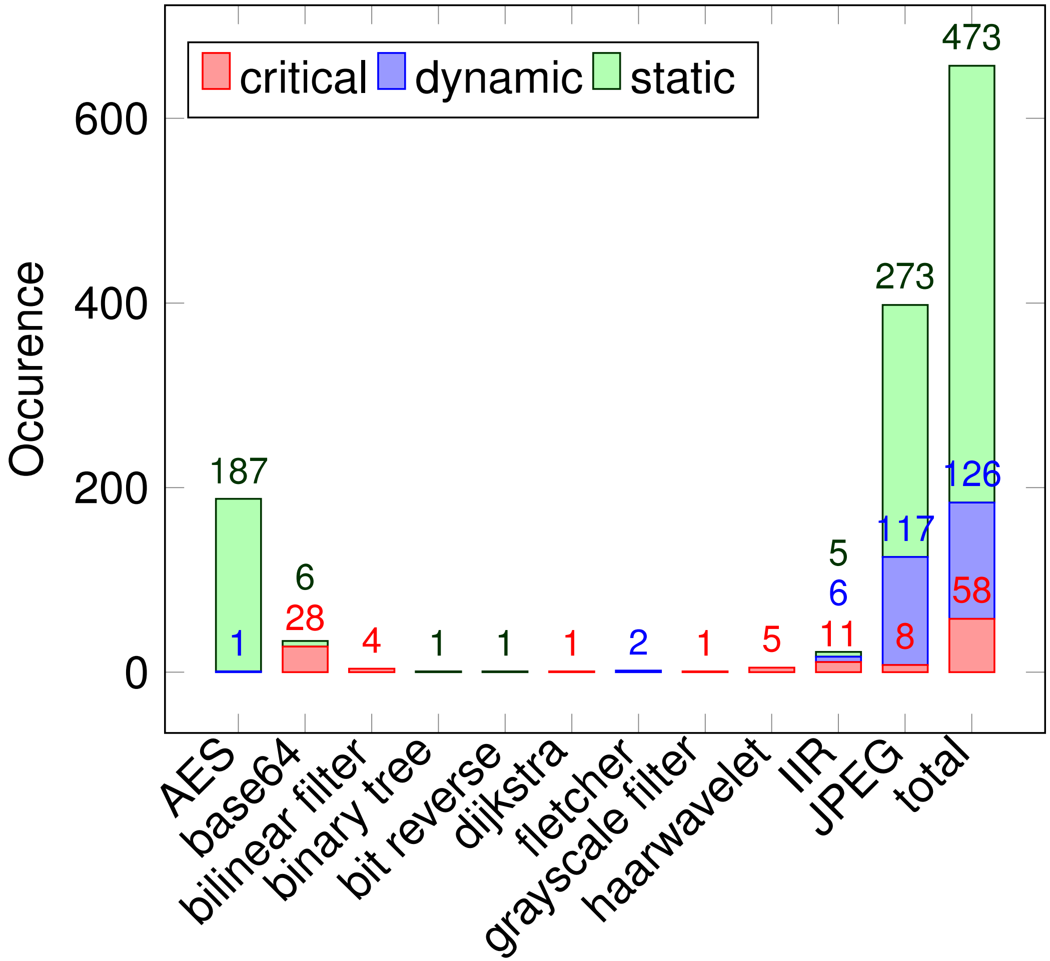
- 11 benchmarks with 23 loops
- Different areas of application
  - Cryptography
  - Image processing
  - Signal processing

AES	Base64	Bilinear Filter	Binary Tree	Bit Reverse	
Dijkstra	Fletcher	Grayscale Filter	Haar-wavelet	IIR	JPEG

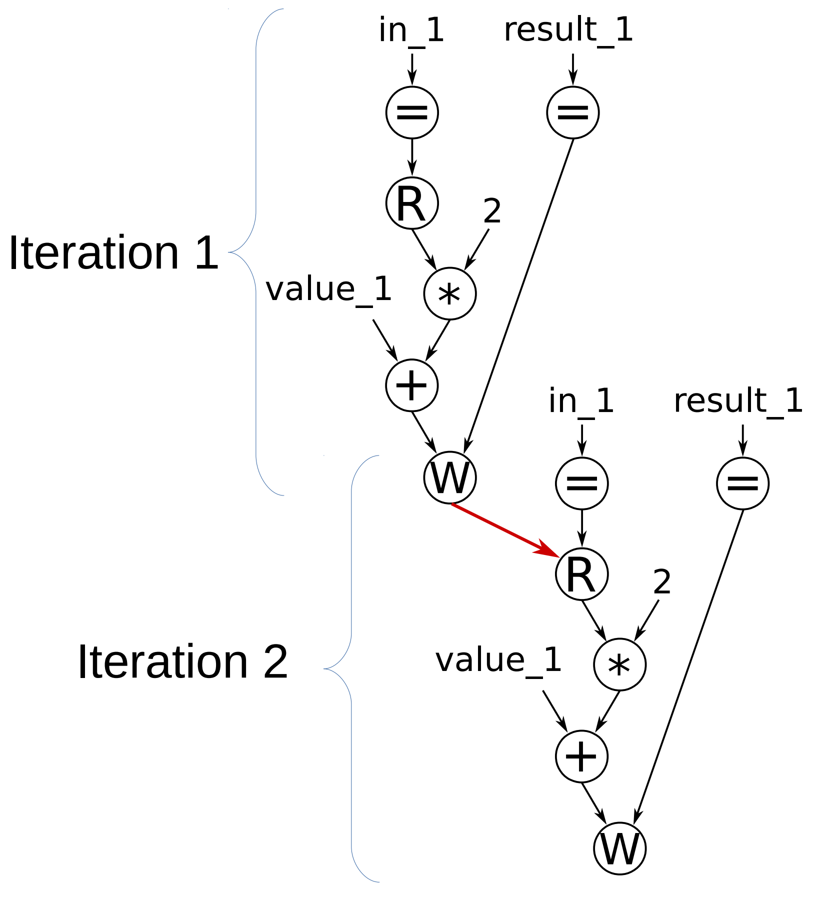
Evaluation
Coverage

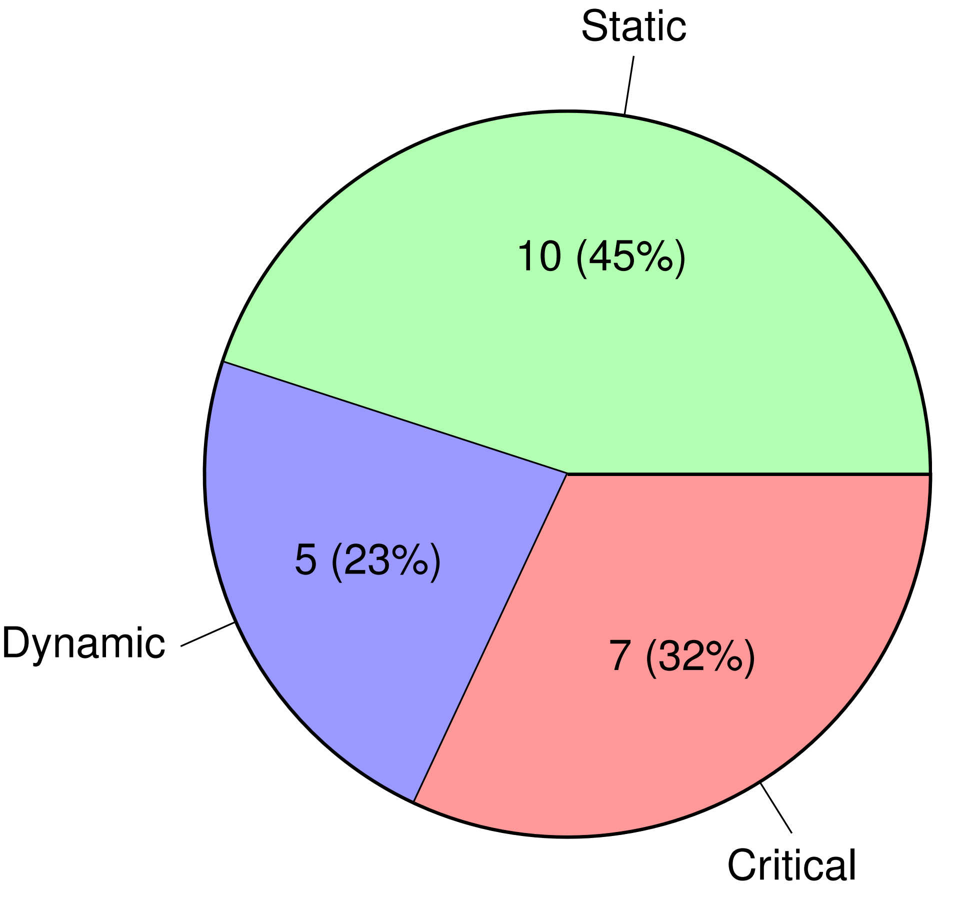


Evaluation
Classification

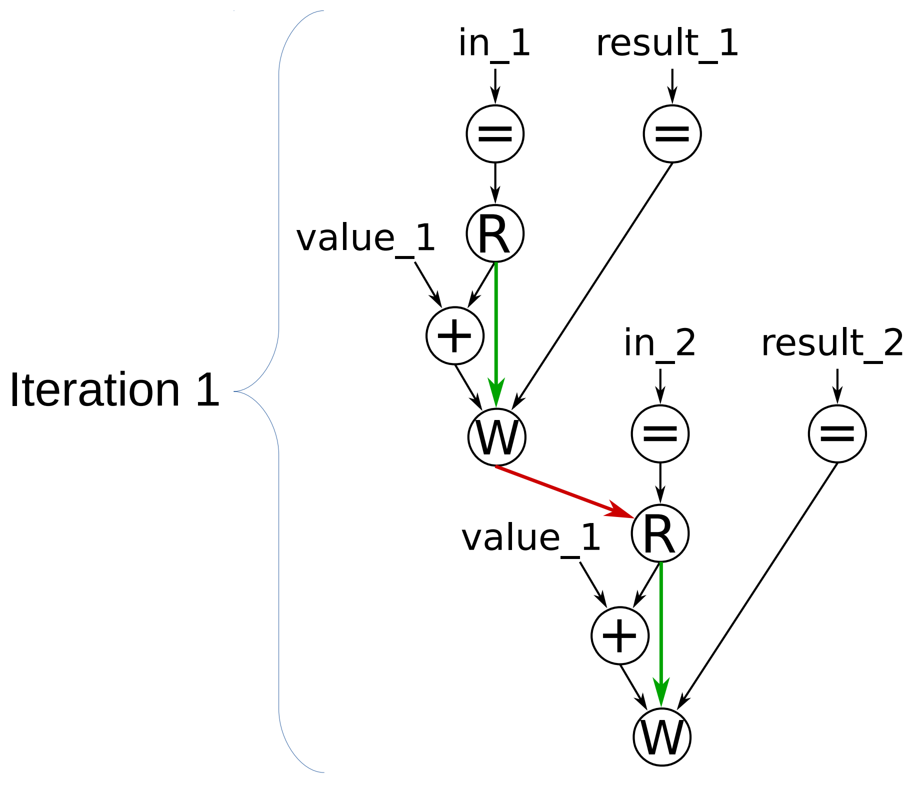


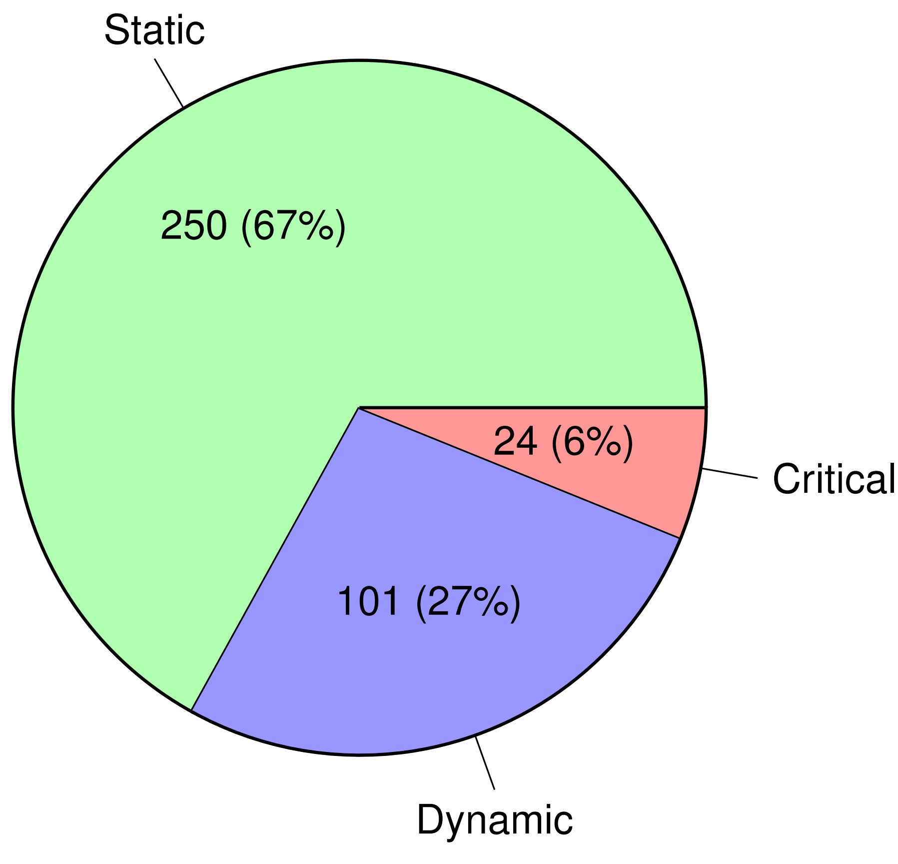
Evaluation
First-Last Dependencies





Evaluation
Memory Carried Dependencies





Evaluation
Step Size

- Condition for static/dynamic classification is equal step size
  - Distance is constant
- Special case: Distance is increasing
  - $A_{sml} = \{(base_{sml} + offset_{sml} + init_{sml}), +, step_{sml}\}$
  - $A_{lrg} = \{(base_{lrg} + offset_{lrg} + init_{lrg}), +, step_{lrg}\}$
  - Runtime condition:
    - $A_{lrg}(n) - A_{sml}(0) > 0$

Evaluation
Step Size

TECHNISCHE UNIVERSITÄT DARMSTADT

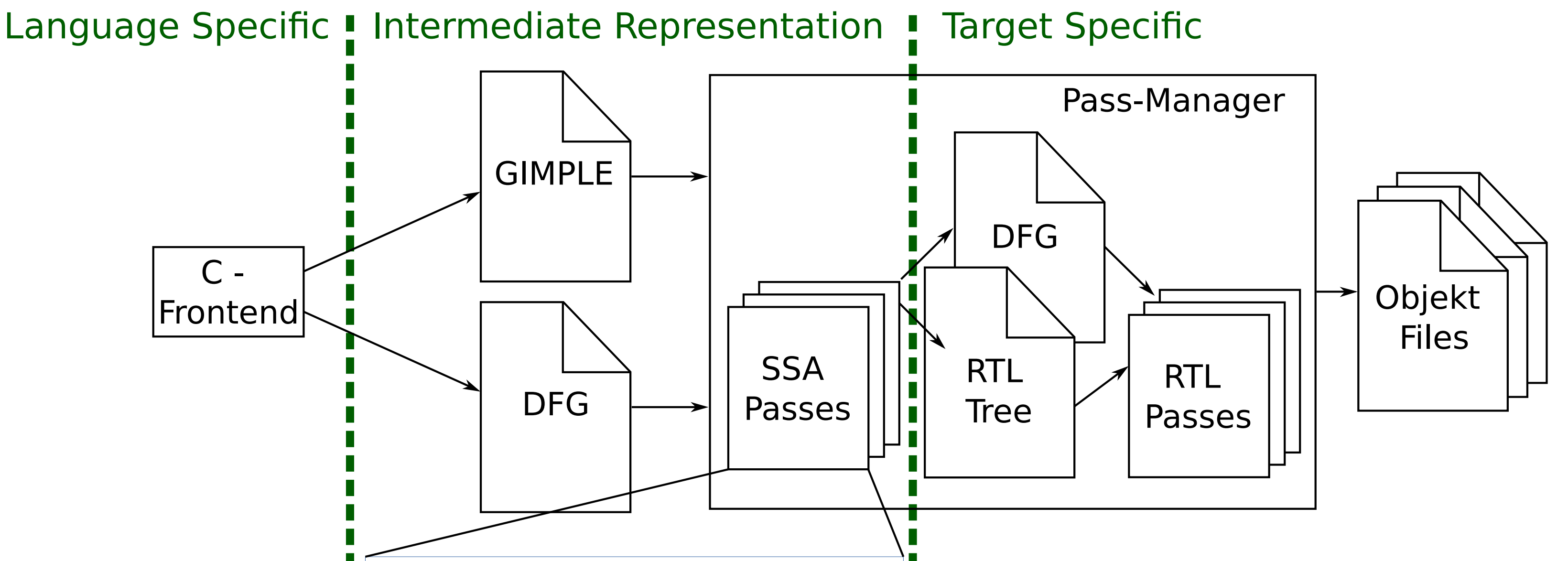
- General
  - 9% → 4% critical pairs
- First-Last Dependencies
  - 32% → 9% critical pairs
- Memory Carried Dependencies
  - 6% → 3% critical pairs

Conclusion

- Over 90% of all dependencies can be considered for decoupling
  - 91% of first-last dependencies
  - 97% of memory carried dependencies
- Analysis is already implemented
- Future work
  - Implementation of static/dynamic decoupling

Thank You!

Background
Compile-Flow



- Dead Code Elimination
- Forward Propagation
- Dead Store Elimination
- Redundancy Elimination
- Loop Optimization
- Constant Propagation
- ...

Benchmarks

TECHNISCHE UNIVERSITÄT DARMSTADT

- 17 benchmarks with 43 loops
- Different areas of application
  - Cryptography
  - Image processing
  - Signal processing

AES	Base64	Bilinear filter	Binary Tree	Bit Reverse	CRC
Dijkstra	Euclid	Fletcher	Grayscale Filter	Haar-wavelet	IDCT
IIR	JPEG	Mandel-brot	Matrix Mult	RSA	

Benchmarks

- 20 Loops excluded
  - No loop pipelining
  - No store operation
  - Single store operation

AES	Base64	Bilinear filter	Binary Tree	Bit Reverse	~~CRC~~
Dijkstra	~~Euclid~~	Fletcher	Grayscale Filter	Haar-wavelet	~~IDCT~~
IIR	JPEG	~~Mandel-brot~~	~~Matrix Mult~~	~~RSA~~	