Advancing the RISC-V Performance Simulation Ecosystem with Data Prefetching

Author: Luís Crespo

Email: luis.miguel.crespo@tecnico.ulisboa.pt

Co-authors:

Nuno Neves

Pedro Tomás

Nuno Roma

inesc id lisboa 20 YEARS

DEFINING TECHNOLOGY

Overview

Motivation

- **The adoption of RISC-V for HPC**
  - Performance improvements for HPC
  - Performance Modelling (and Simulation) SIG
  - Olympia: performance model of an OoO Superscalar RISC-V Core

- **Olympia lacks support for:**
  - Data prefetching mechanisms, crucial technique for mitigating the “memory wall”
  - Flexible and full ISA trace generation (currently generated with an emulator restricted to RV64GC)

- **Contributions**:
  - **Parser to convert Spike-generated traces into Olympia’s format**
  - **Integration of data prefetching mechanisms into Olympia**
  - **Implementation of multiple prefetching algorithms within Olympia**

Background: Simulation

- **Functional Simulation**
  - Simulate only the outcome of instructions.
  - No cycle-level detail; fast and lightweight.

- **Timing Simulation**
  - Evaluate processor design choices.
  - Identify bottlenecks and estimate IPC

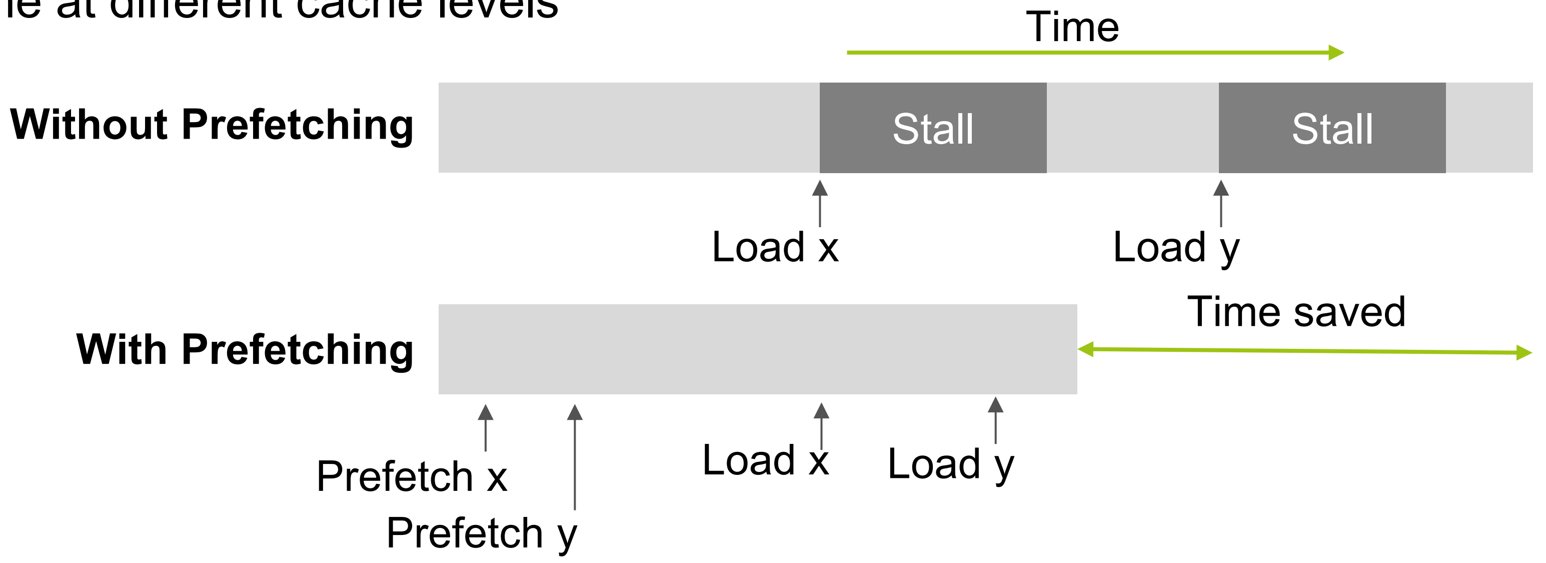
- **Spike**
  - Functional simulator
  - Golden reference RISC-V simulator

- **Olympia**
  - Trace-driven timing simulator
  - Reads Simulation Trace Format (STF) or JSON
  - Built on Sparta framework
  - Simulate OoO Superscalar RISC-V Cores
  - Designed for RISC-V performance modeling

Background: Data Prefetching

- Software Prefetching
  - Compiler or programmer inserts explicit prefetch instructions
- **Hardware Prefetching** (focus of this work)
  - Dedicated module that detects memory access patterns and issue prefetch requests

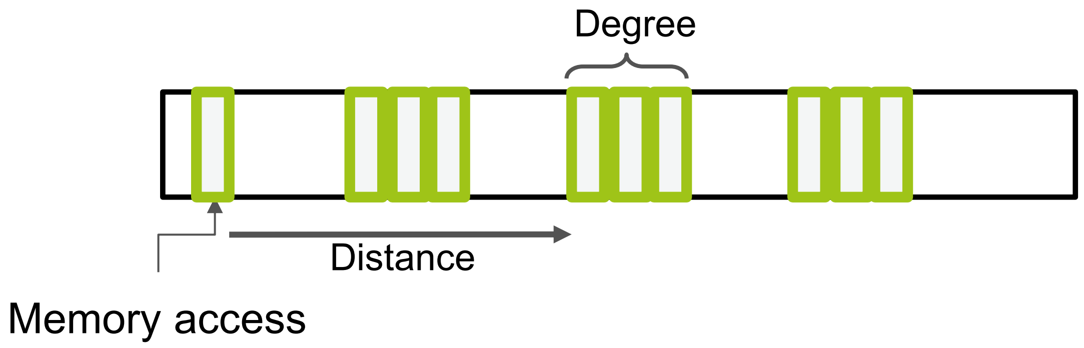
- Proactively load data into caches before the processor demands it
- Reduces stall cycles and improving throughput
- Can be done at different cache levels



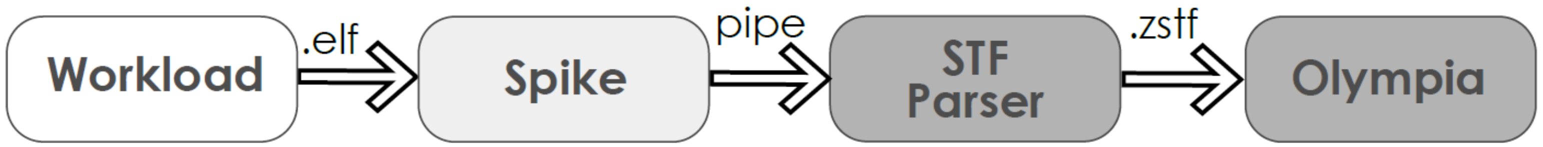
Background: Data Prefetching

- **Prefetching Evaluation Metrics**
  - Issued prefetches
  - Demand misses
  - Useful prefetches
  - Useless prefetches
  - Late prefetches
  - $\boldsymbol{Accuracy = \frac{Useful}{Issued}}$
  - $\boldsymbol{Coverage = \frac{Useful}{Useful + demand\ misses}}$

- **Aggressiveness parameters:**
  - **Distance** - how far ahead to prefetch
  - **Degree** - number of blocks fetched



Proposed Simulation Flow



Integrating Spike and Olympia

Trace Generation:

- Custom STF parser
- Converts Spike logs into STF
- Extract:
  - PC (next PC)
  - Instruction type (load/store)
  - Encoding
  - Memory address
  - Data size
  - Instruction width

START

Memory Address

Encoding

Data size

PC

```
0x10264 (0x00004033)
0x10268 (0x4629) x12 0x000000000000000a
0x1026a (0x0005a007) f0  0xffffffffc2ac75c3 mem 0x0000003ffffffb10 4
0x1026e (0x00052087) f1  0xffffffff00000000 mem 0x0000003ffffffae8 4
        (0x08a07043) f0  0xffffffffc3815852
0x10276 (0x00052027) mem 0x0000003ffffffae8 0xc3815852 4
0x1027a (0x0511) x10 0x0000003ffffffaec
0x1027c (0x167d) x12 0x0000000000000009
0x1027e (0x0591) x11 0x0000003ffffffb14
0x10280 (0xf66d)
0x1026a (0x0005a007) f0  0xffffffff4252999a mem 0x0000003ffffffb14 4
...
0x10282 (0x0010c033)
```

Spike log

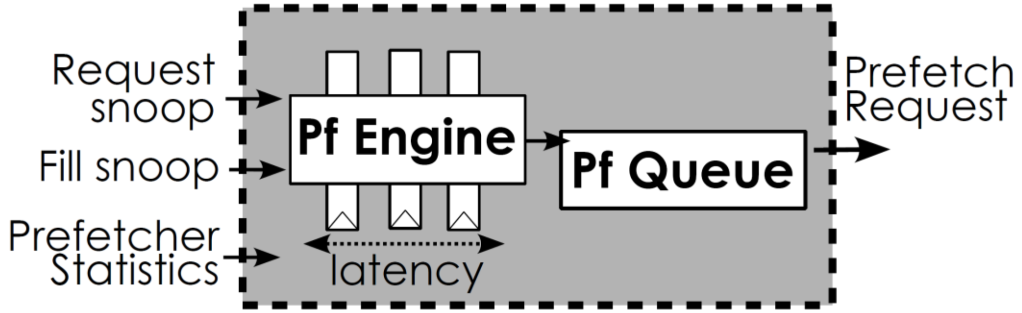
END

Prefetcher Module

- Implemented as a standalone module
  - Can be instantiated at multiple cache levels (L1D, L2)
  - Easily extendable to support other algorithms

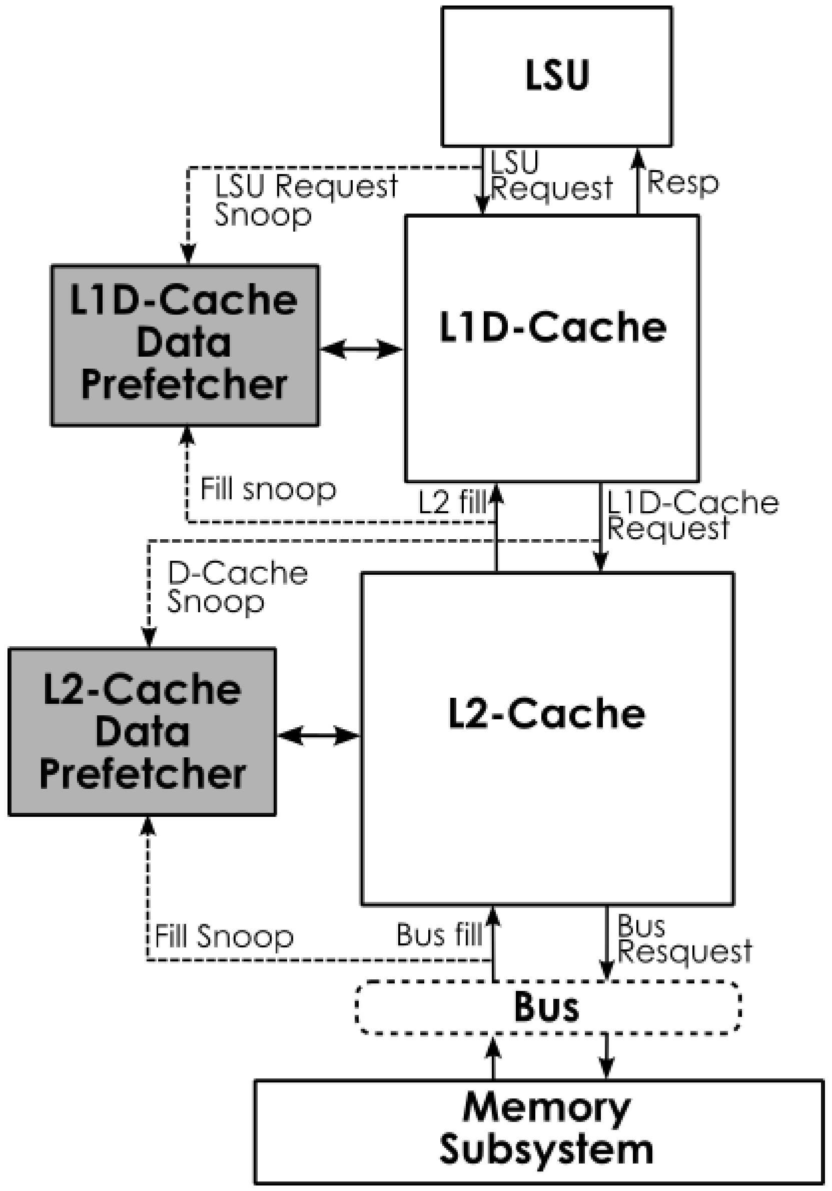
- Prefetch Engine:
  - Generates prefetches based on the selected algorithm
- Prefetch Queue:
  - Buffers pending requests
- Parameterizable
  - Prefetch Degree, Prefetch Distance, Prefetch Store addresses, etc

- Three prefetching algorithms were implemented:
  - Next-Line prefetcher
  - Stride prefetcher
  - Best-Offset prefetcher

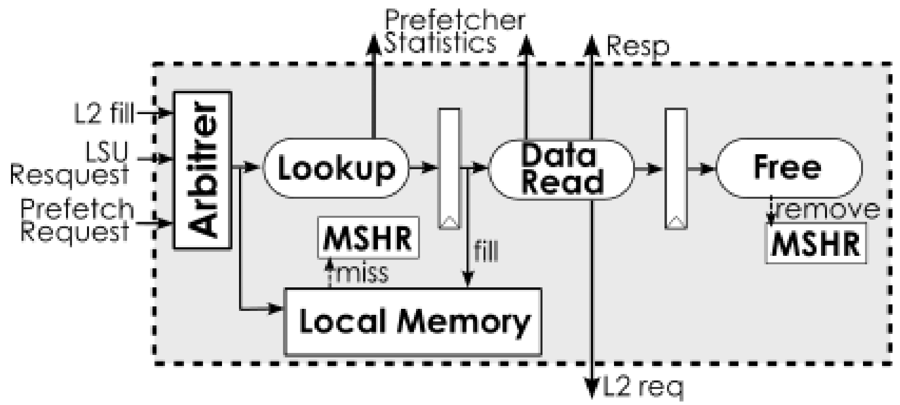


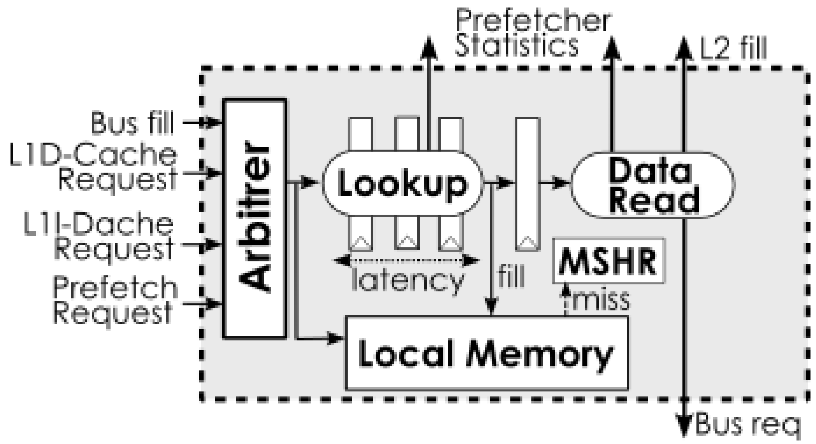
Prefetcher Integration

- One prefetcher module for each cache level
- By default, both the L1 and L2 prefetchers are disabled
- Activate a prefetcher in the architecture configuration file
- Each prefetcher snoops its cache requests and fill responses
- Squash unnecessary requests in the cache
  - Cache line is already present
  - Already requested (allocated entry in the MSHR)



Prefetcher Integration





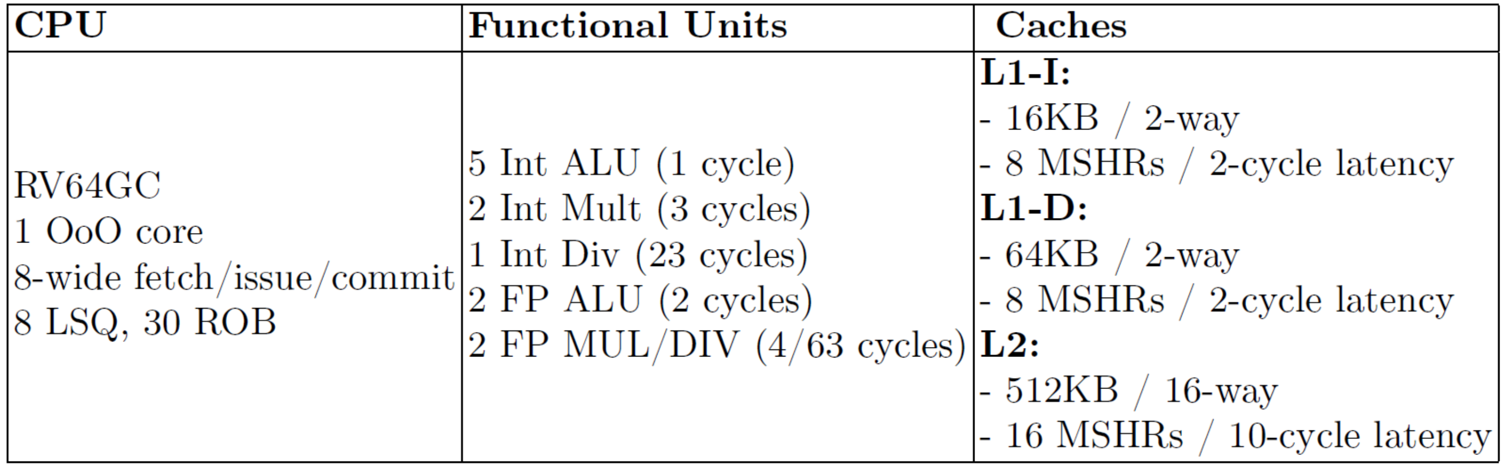
L1D-Cache:

- Three stage cache pipeline:
  - Arbitration and Lookup
  - Data Read
  - Deallocation
- Extended to non-blocking
- Notifies the prefetcher to send requests

L2-Cache:

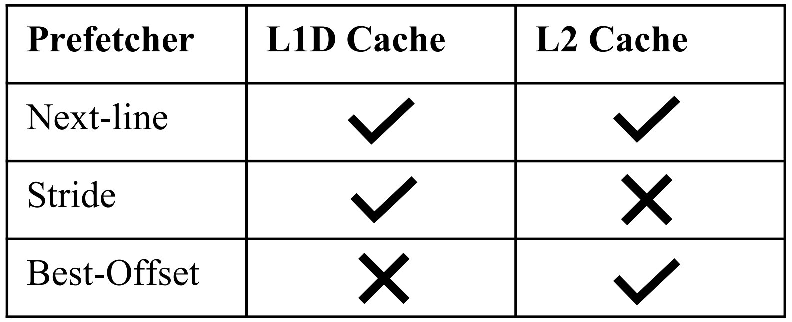
- Configurable size pipeline:
  - Arbitration
  - Lookup (several cycles)
  - Data Read
- Prefetchs arbitrated with other requests (lowest priority)

Evaluation: Setup



CPU	Functional Units	Caches
RV64GC 1 OoO core 8-wide fetch/issue/commit 8 LSQ, 30 ROB	5 Int ALU (1 cycle) 2 Int Mult (3 cycles) 1 Int Div (23 cycles) 2 FP ALU (2 cycles) 2 FP MUL/DIV (4/63 cycles)	**L1-I:** - 16KB / 2-way - 8 MSHRs / 2-cycle latency **L1-D:** - 64KB / 2-way - 8 MSHRs / 2-cycle latency **L2:** - 512KB / 16-way - 16 MSHRs / 10-cycle latency

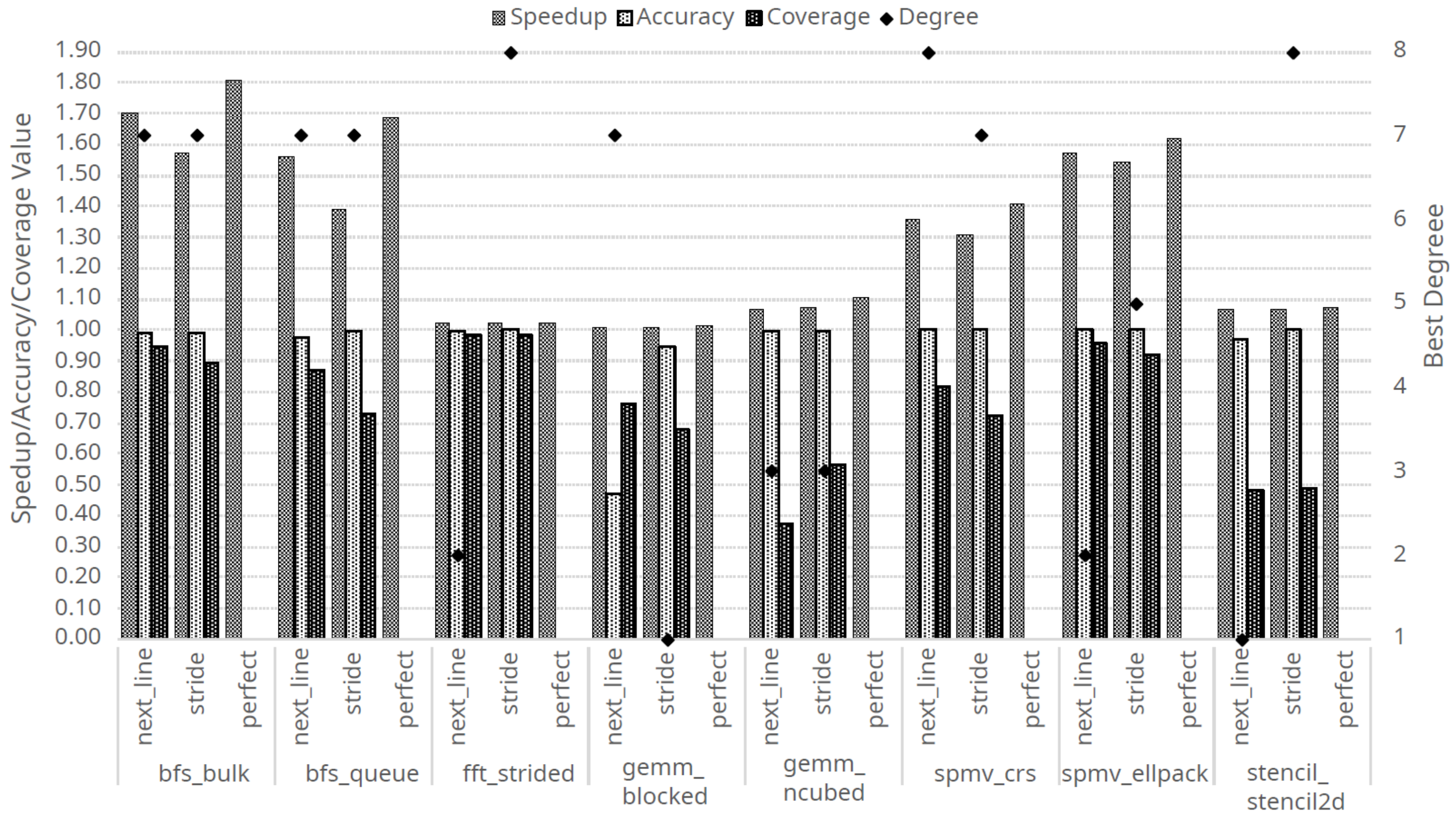
- CPU configuration available for Olympia (big_arch)
- 8 applications from the MachSuite
- Compiled using Clang with -O3
- Spike and proposed STF parser to generate traces
- Different prefetcher combination for each cache level



Prefetcher	**L1D Cache**	**L2 Cache**
Next-line	✓	✓
Stride	✓	✗
Best-Offset	✗	✓

Evaluation: L1D Cache

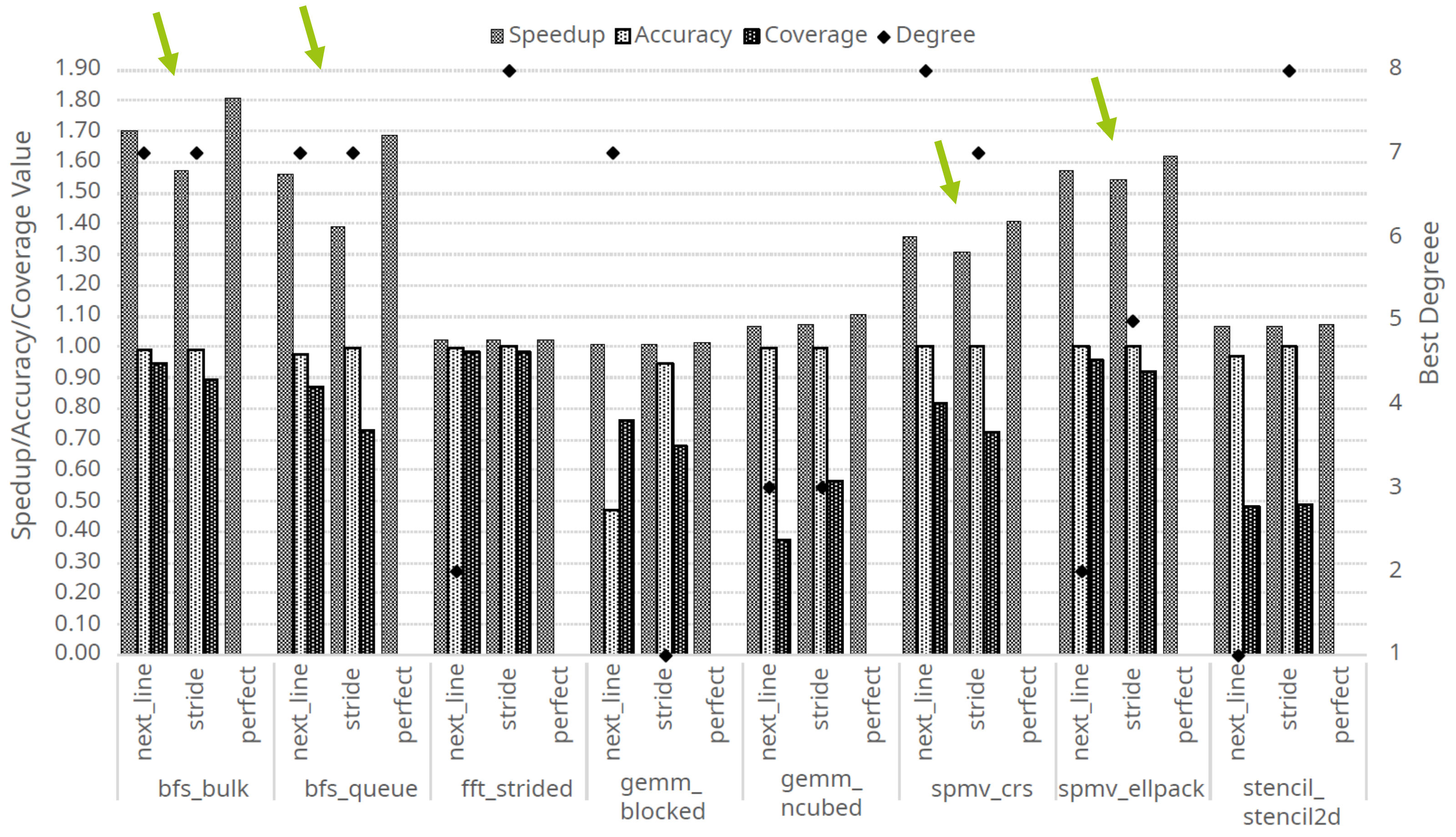
- Speedup (over the baseline)
- Accuracy, and Coverage



Evaluation: L1D Cache

- Speedup (over the baseline)
- Accuracy, and Coverage

- Highest speedups:
  - BFS (bulk & queue)
  - SPMV (ellpack & csr)
  - No spatial locality benefit

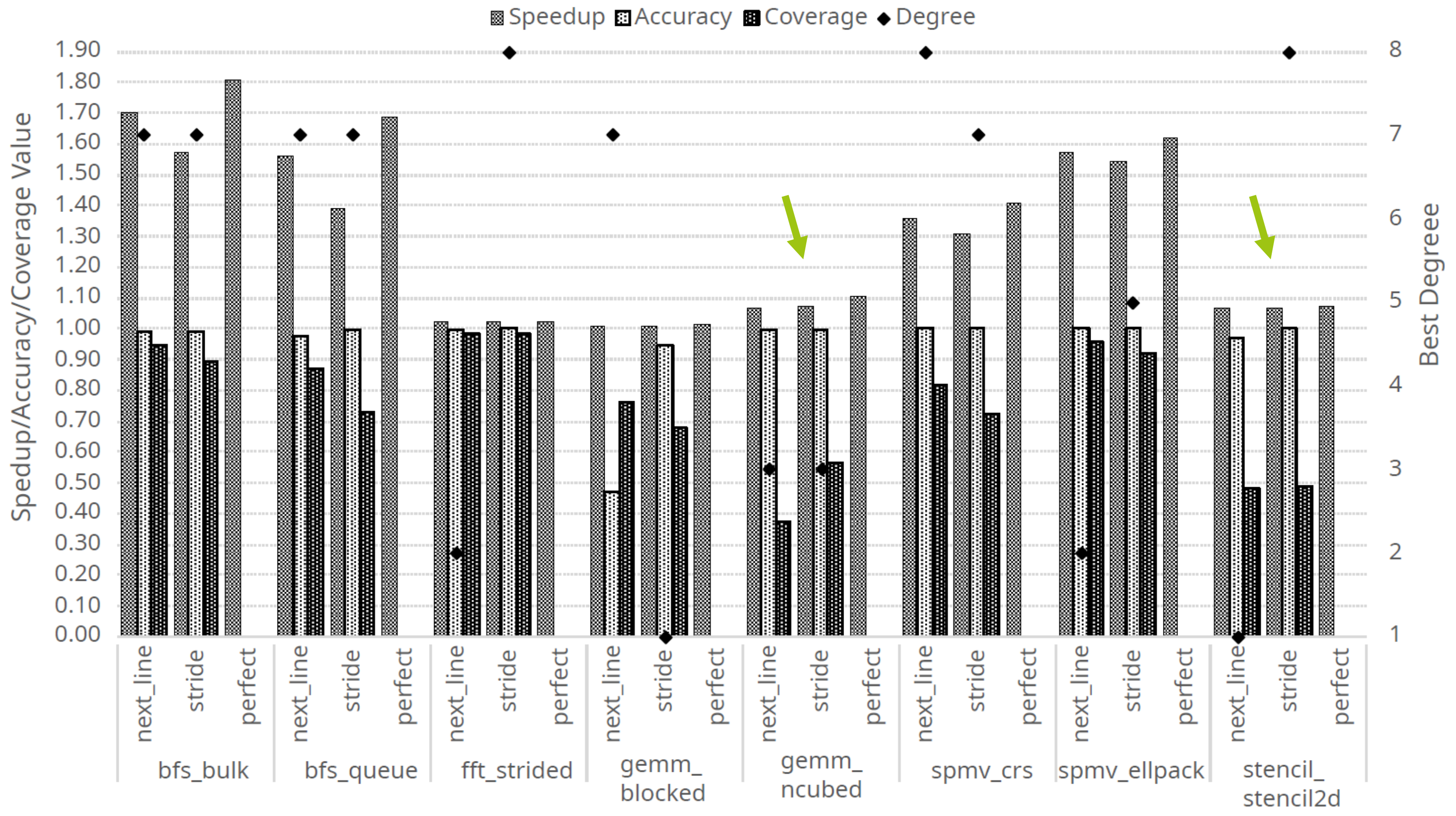


Evaluation: L1D Cache

- Speedup (over the baseline)
- Accuracy, and Coverage

- Highest speedups:
  - BFS (bulk & queue)
  - SPMV (ellpack & csr)
  - No spatial locality benefit

- High accuracy w/ low coverage:
  - GEMM (ncubed)
  - Stencil (2D)
  - Memory hierarchy is overloaded



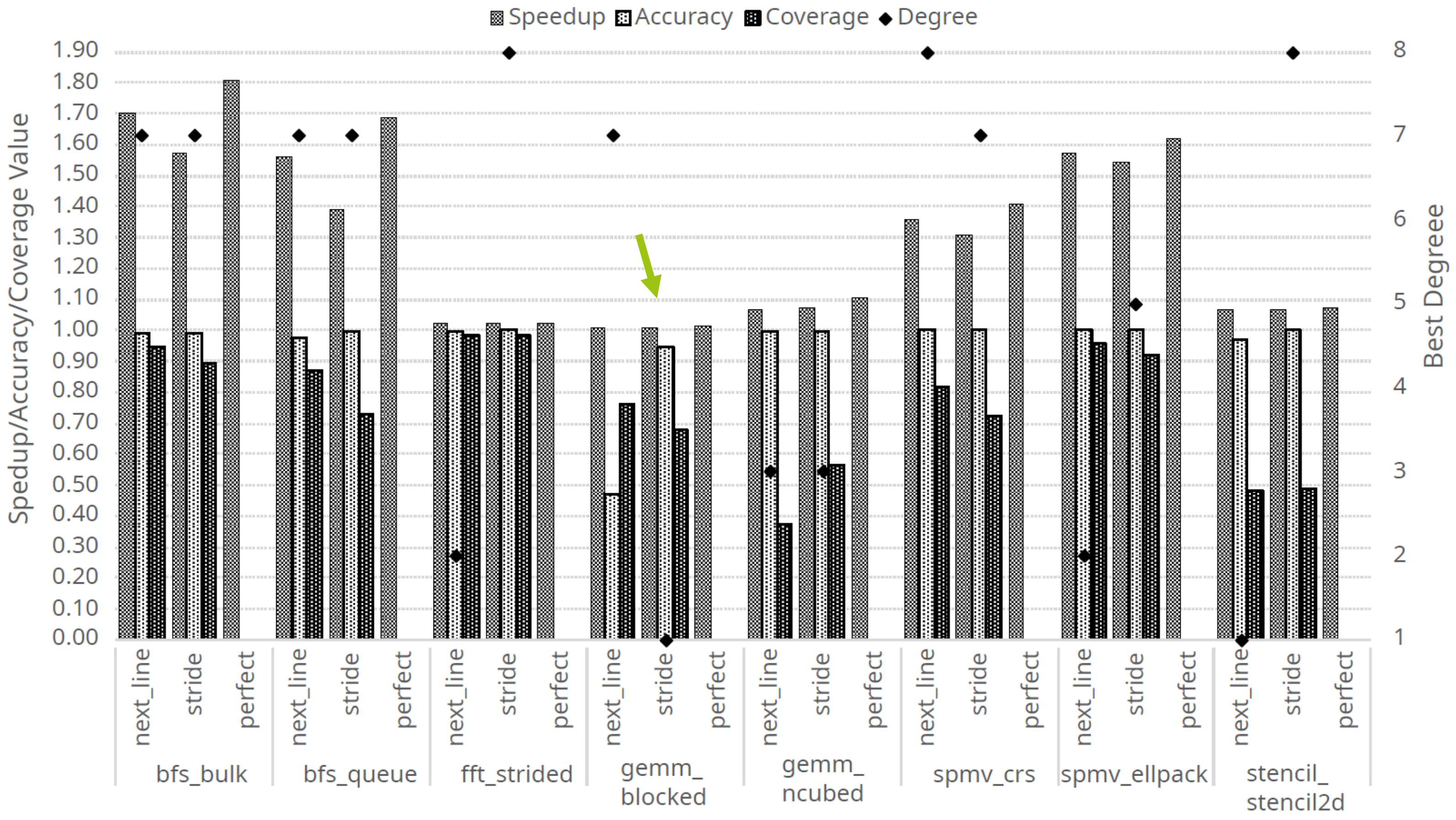
Evaluation: L1D Cache

- Speedup (over the baseline)
- Accuracy, and Coverage

- Highest speedups:
  - BFS (bulk & queue)
  - SPMV (ellpack & csr)
  - No spatial locality benefit

- High accuracy w/ low coverage:
  - GEMM (ncubed)
  - Stencil (2D)
  - Memory hierarchy is overloaded

- Stride vs Next-line in Gemm (blocked)

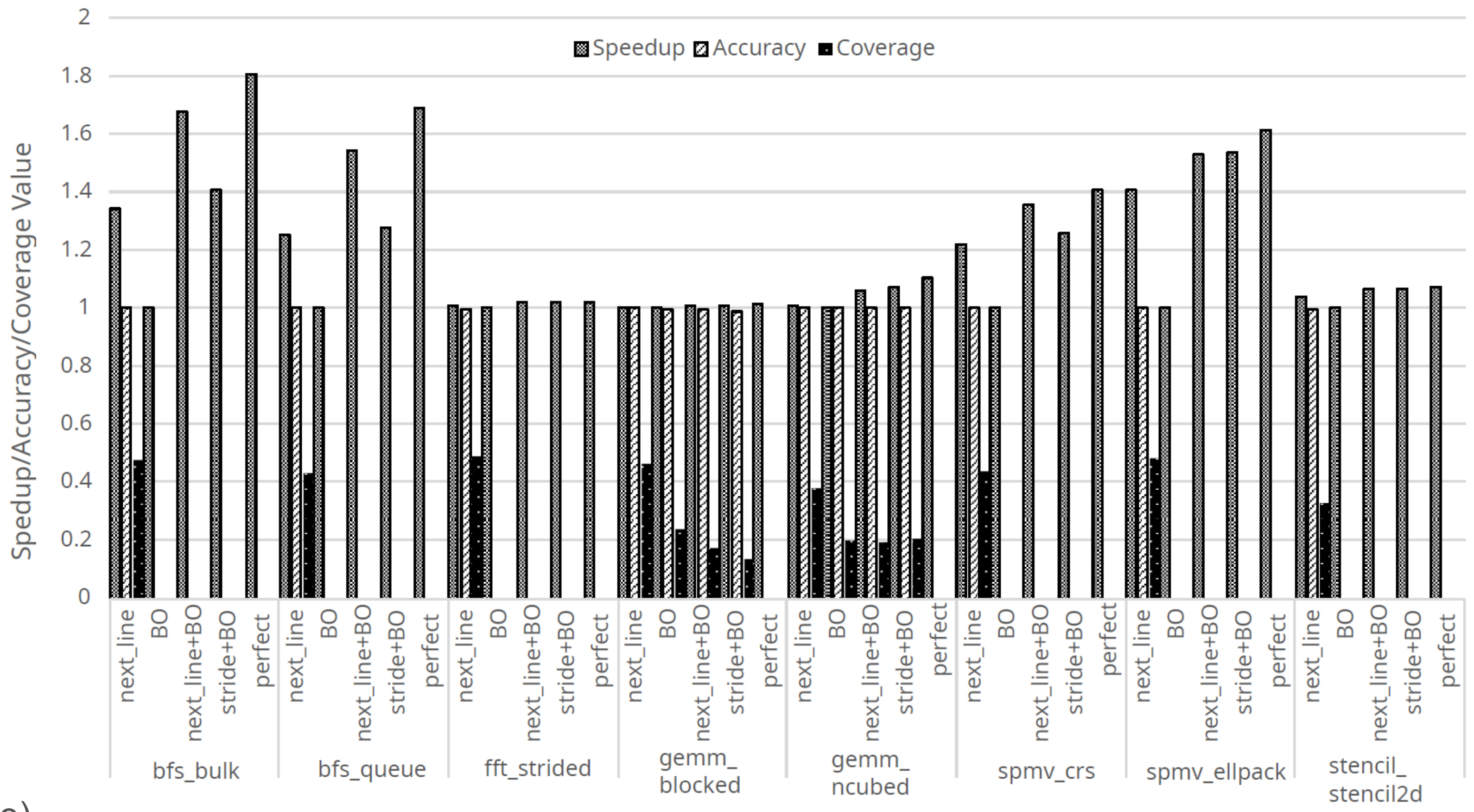


Evaluation: L2 Cache

- Same metrics as before
- Configurations:
  - L2 next-line
  - L2 best-offset
  - L2 best-offset + L1 next-line
  - L2 best-offset + L1 stride

- Next-line:
  - Insufficient to predict the patterns
  - Lower amount of cache requests

- Best-offset prefetcher:
  - Long training latency
  - Only works for GEMM (dataset size)



Conclusion

Key Limitations Identified:

- Limited Flexibility in Memory Hierarchy:
  - Aggressive prefetching exposes bottlenecks in cache and bus structures
  - Blocking bus model between L2 and memory stalls memory requests, causing pipeline bubbles
- Need for Store Queue:
  - Decoupling store operations from LSU can reduce cache pressure and improve efficiency
- Load-Store Unit (LSU) Bottleneck:
  - LSU generates redundant memory requests, increasing congestion
  - Immediate processing of store operations can delay/block loads and prefetches

Summary:

- Developed a new trace generation (via Spike)
- Developed a new extensible prefetcher module
- Integrated data prefetching in Olympia
- Validated performance gains with next-line, stride, and best-offset prefetchers

inesc id lisboa
20 YEARS
DEFINING TECHNOLOGY

Thank you!