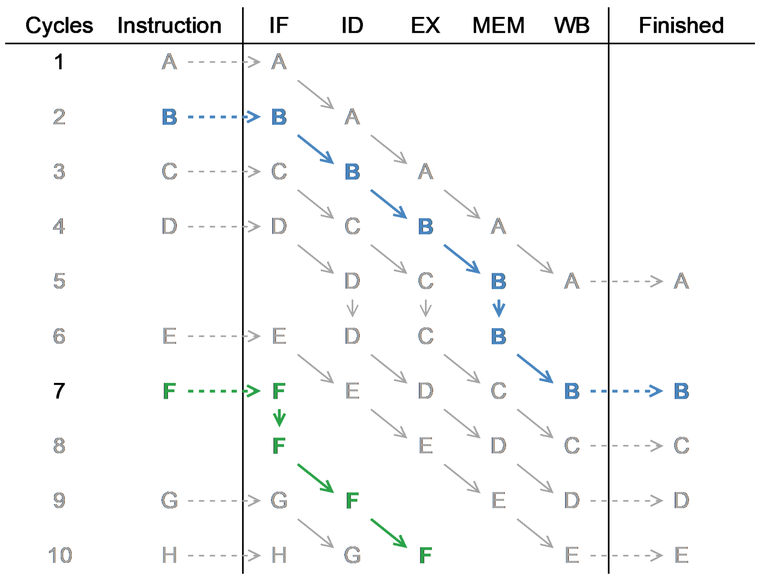
# Detection and Mitigation of Load Stalls on AArch64

**Jonas R. Schönauer, David Bremner, Kenneth B. Kent**
Faculty of Computer Science, University of New Brunswick
{jonas.schoenauer,bremner,ken}@unb.ca

**Julian Wang**
IBM Canada
zlwang@ca.ibm.com

## Load Stalls

- Processors use pipelines to increase instruction throughput
- Loading data from memory is costly
- When the processor waits for data it stalls




Example: Stall

- Two different stall types
  - Backend Stall
  - Frontend Stall

## AArch64

- Simple Instruction Set
- Few instructions can lead to frontend or backend stalls

## Motivation

- Detect impact of Load Stalls on OpenJ9 runtime
- Previously shown: Load Stalls have a high impact on x86
- Automize data gathering process

## Used Technologies

- AArch64 Assembly, Bash, C/C++, Java, Python
- Git, Perf
- Windows & Linux

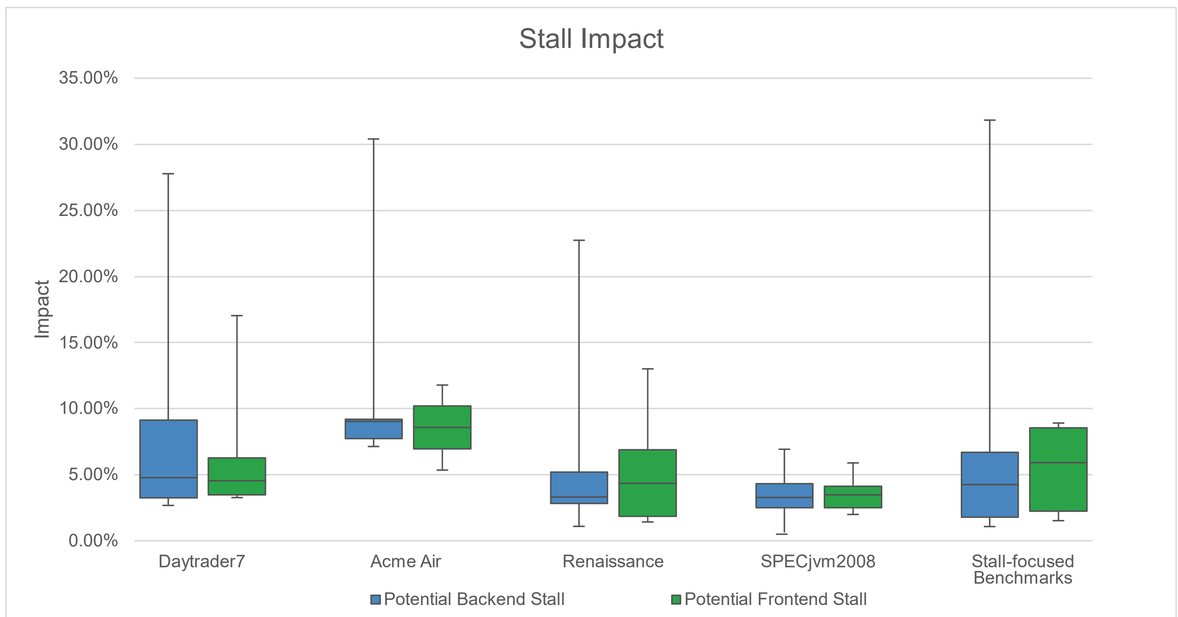
## Automated Framework

- Procedure is the same for all benchmarks
- Flexible
- Minimize manual processes

```
symbols = SymbolList()
  for i = 0; i < 5; ++i
    symbols.Add(RunBenchmark())
hotSymbols = symbols.Filter()
instructions = RunBenchmark(hotSymbols)
hotInstructions = instructions.Filter()
  for instruction in hotInstructions
    Emit(instruction, Classify(instruction))
```

Benchmarking Procedure

## Preliminary Results




- Impact of Backend Stalls seems to be higher
- However, the averages are relatively close
- Impact comparison might not be suitable

CAS–Atlantic