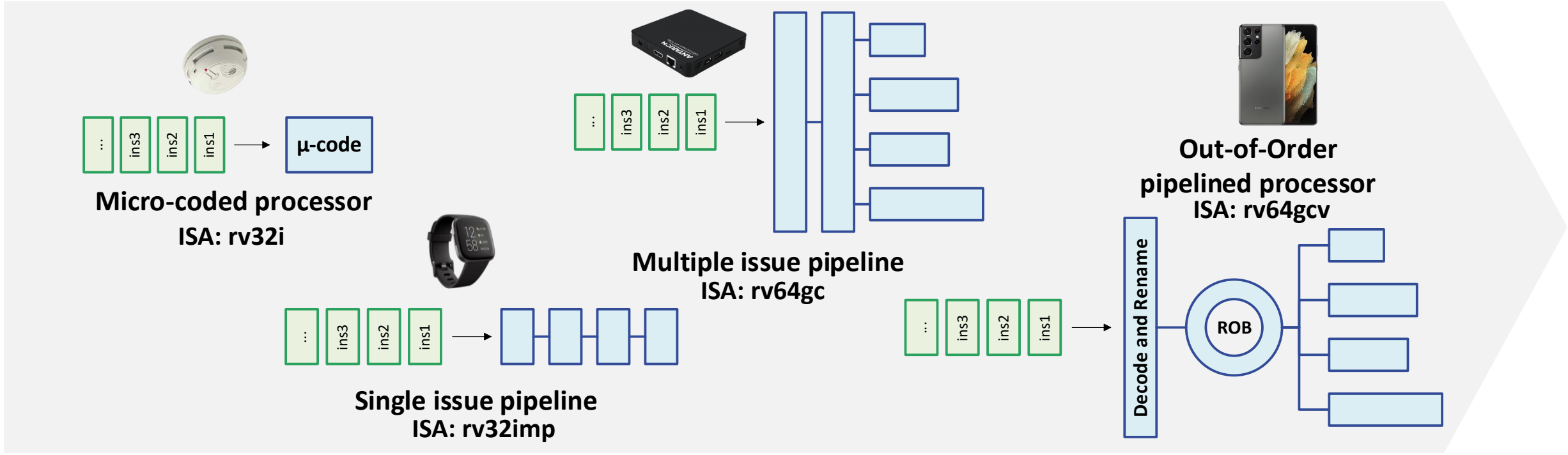
Design Exploration of RISC-V Soft-Cores through Speculative High-Level Synthesis

Jean-Michel Gorius, Dylan Léothaud, Simon Rokicki, Steven Derrien
Univ Rennes, Inria, CNRS, IRISA

RISC-V Summit Europe - Munich
June 2024

Processor Design Landscape



Low energy **High performance**

This work

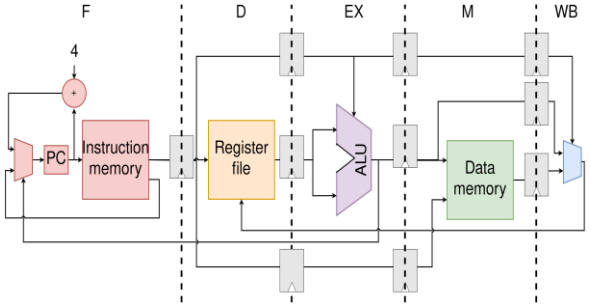
Rely on High-Level Synthesis to automatically synthesize pipelined processors from an Instruction Set Simulator

Customizing Instruction Set Processors

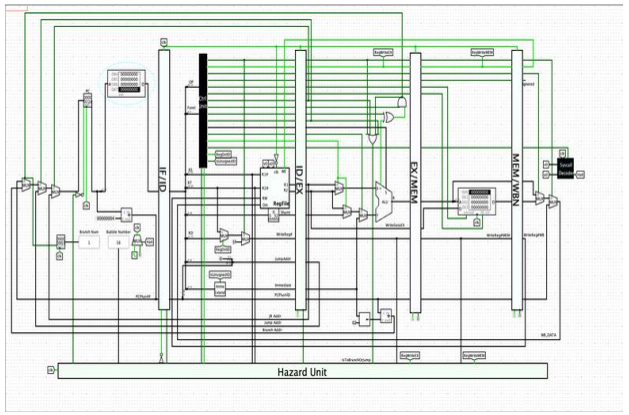
Instruction set

Example instrucmtion	Instruction name	Meaning
add x1,x2,x3	Add	Regs[x1]←Regs[x2]+Regs[x3]
addi x1,x2,3	Add immediate unsigned	Regs[x1]←Regs[x2]+3
lui x1,42	Load upper immediate	Regs[x1]←0^{32}##42##0^{12}
sll x1,x2,5	Shift left logical	Regs[x1]←Regs[x2]<<5
slt x1,x2,x3	Set less than	if (Regs[x2]<Regs[x3]) Regs[x1]←1 else Regs[x1]←0

Micro-architecture



Circuit



ISS or DSL specification

```
while (1) {
    opcode,ra,rb,rc,imm = mem[pc];
    pc++;
    switch (opcode) {
        case LOAD: reg[rc] = mem[reg[ra] + imm]; break;
        case ADD: reg[rc] = reg[ra] + reg[rb]; break;
        case JUMP: pc = reg[ra]; break;
        case BNZ: pc = rb ? reg[ra] : pc; break;
            ...
    }
}
```

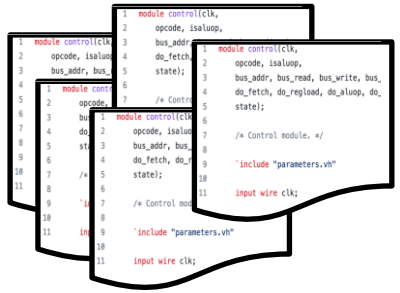
Proprietary DSLs

```
// Instruction-set grammar
opn my_core (arith_inst | ctrl_inst);

opn arith_inst (a:alu_inst,
  d: div_inst, l:load_store_inst);


opn alu_inst (op:opcod, x:clu, y:clu,
  z:clu) {
  action {
    stage EX1:
      A = RA[x];
      B = RB[y];
      switch (op) {
      case add: C = add(A, B) @alu;
      case and: C = and(A, B) @alu;
      case or: C = or(A, B) @alu;
      ...
      }
    stage EX2:
      RA[z] = C @alu;
  }
  syntax: op " RA" y ", RB" x ", RA" z;
  image: "0"::op::x::y::z;
}
...
```

Verilog, Chisel

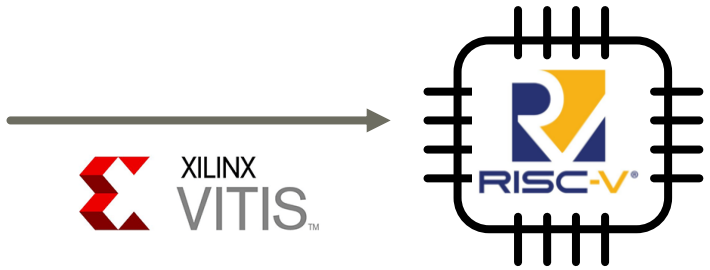


Synthesizing a CPU from an Instruction Set Simulator in C

```
while(1) {
  unsigned int ir = fetch(mem, pc);
  pc += 4;

  struct decode_info dc = decode(ir);
  unsigned int rs1 = x[dc.rs1];
  unsigned int rs2 = x[dc.rs2];

  switch(opcode(dc)) {
  case RISCV_ADD:
    x[dc.rd] = add(rs1, rs2);
    break;
  case RISCV_MUL:
    x[dc.rd] = mul(rs1, rs2);
    break;
  case RISCV_JAL:
    pc = pc + dc.simm_J;
    break;
  case RISCV_LD:
    if(dc.funct3 == RISCV_LD_LB)
        x[dc.rd] = mem[rs1 + dc.imm_S];
    else if(/* ... */)
      // ...
    break;
  // ...
  }
}
```

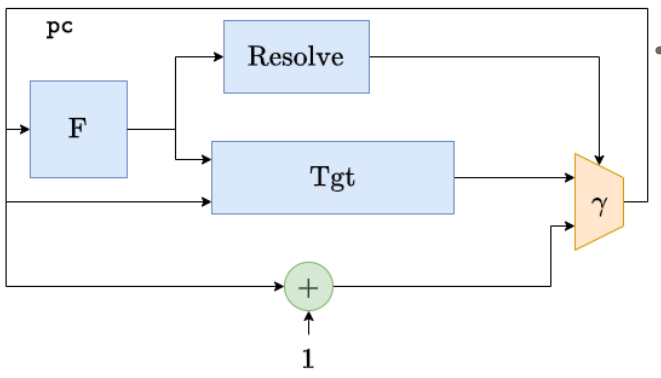


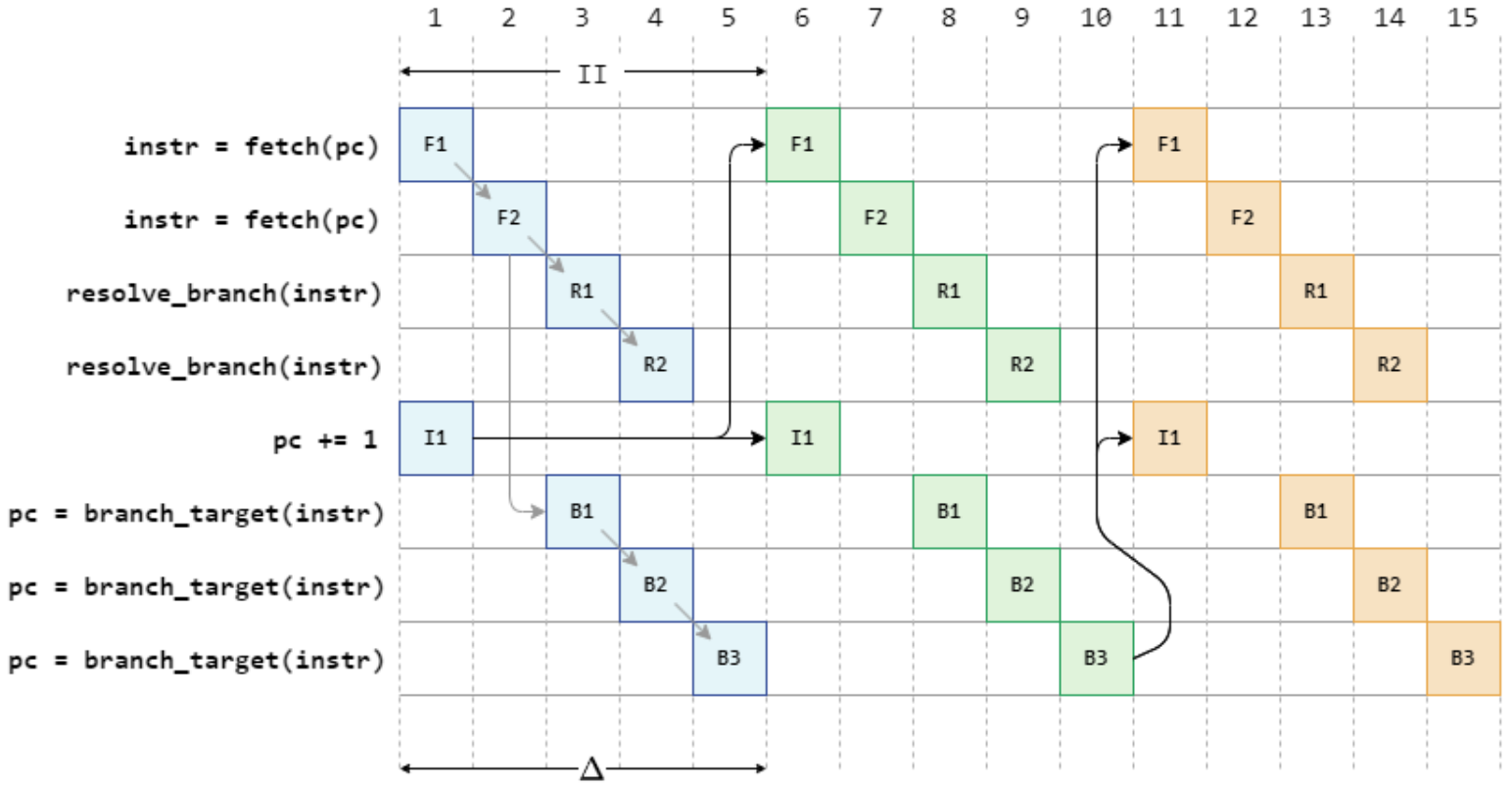
Main challenge

Current HLS can't generate **efficient** pipeline because of **static scheduling**

Synthesizing a CPU from an Instruction Set Simulator in C

```
while (1){
    unsigned int instr = fetch(pc);
    if (resolve_branch(instr)){
        pc = branch_target(pc, instr);
    else {
        pc += 1;
    }
}
```



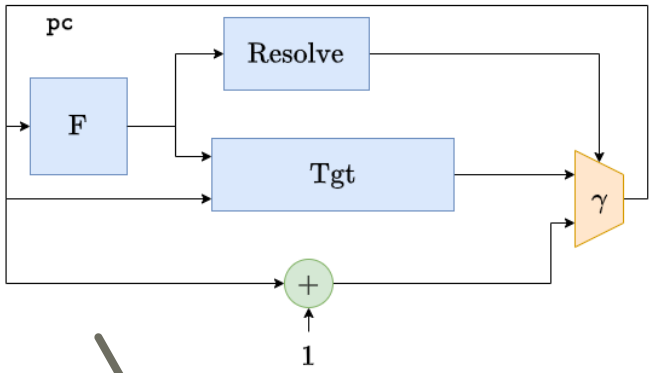


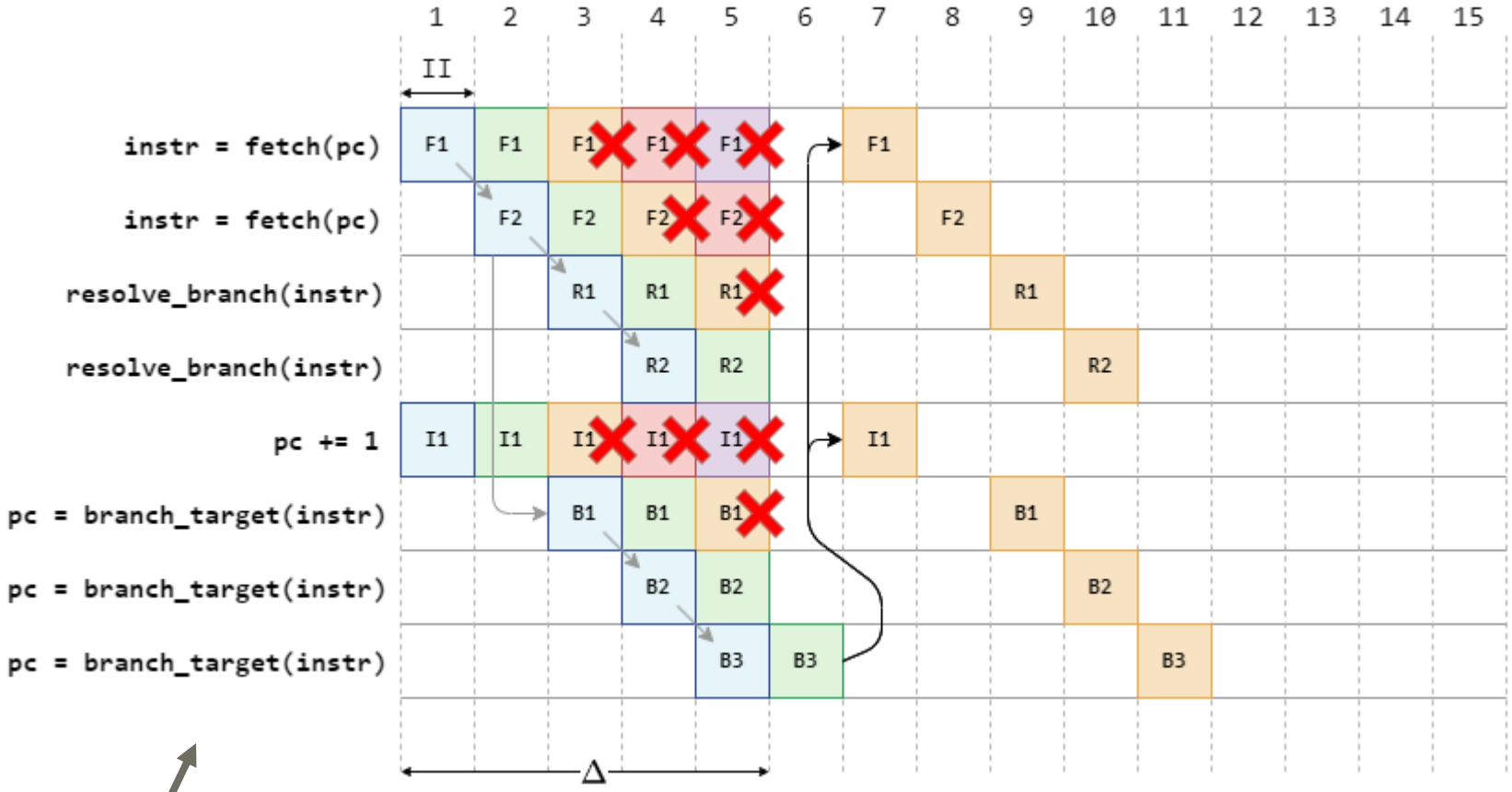
Key takeway

Static scheduling is pessimistic and can't generate efficient processor pipeline

Synthesizing a CPU from an Instruction Set Simulator in C

```
while (1){
    unsigned int instr = fetch(pc);
    if (resolve_branch(instr)){
        pc = branch_target(pc, instr);
    else {
        pc += 1;
    }
}
```





Speculative Loop Pipelining

Key takeway

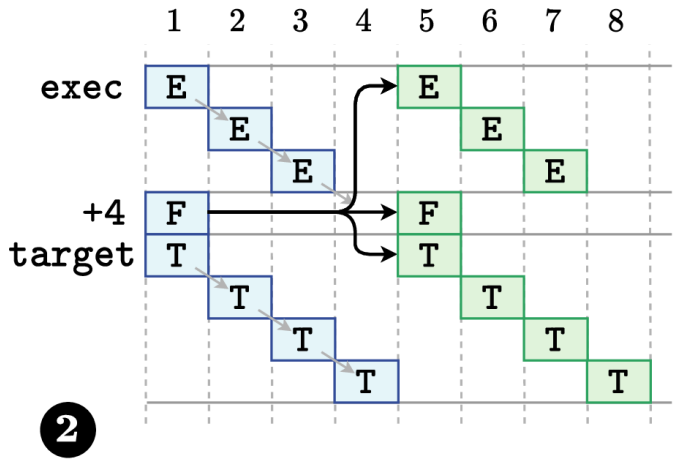
Speculative loop pipelining may lead to efficient instruction pipelines

State-of-the-art: Speculative High-Level Synthesis

Source-to-source transformations enabling Speculative HLS

```
while(1) {
  ir =  mem[pc];
  taken = exec(ir) {
  if (taken) {
      pc = target(pc);
  } else {
      pc = pc + 4;
  }
}
```

1



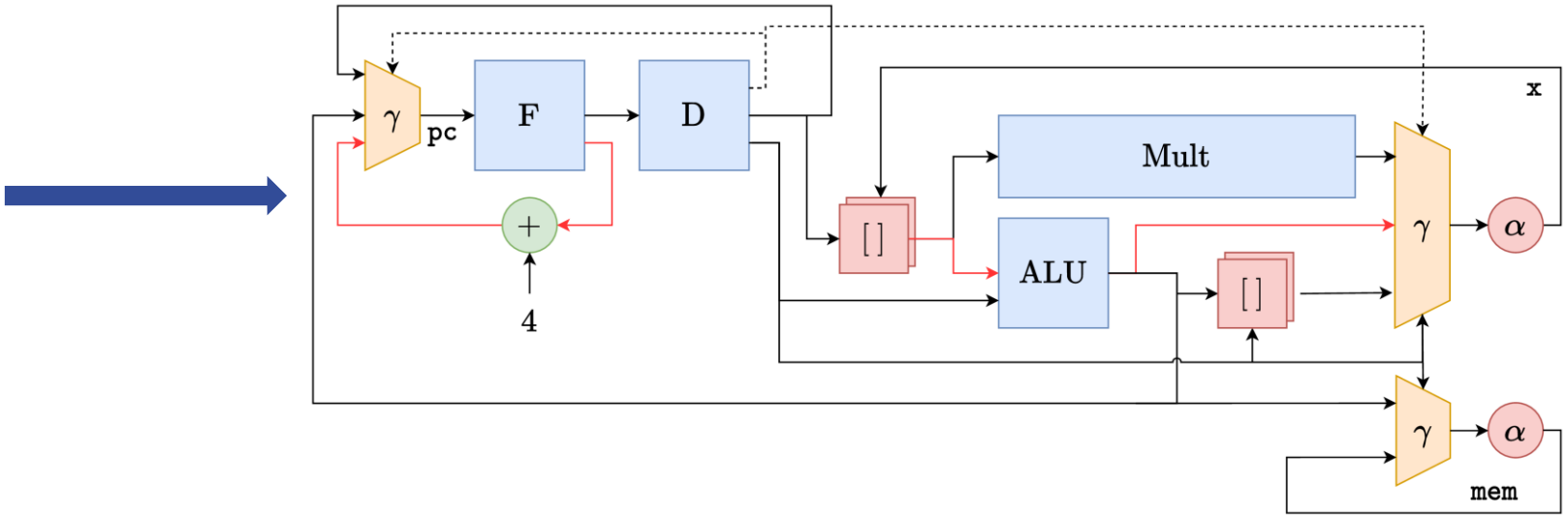
Static pipelining

Back to full scale RISC-V ISS

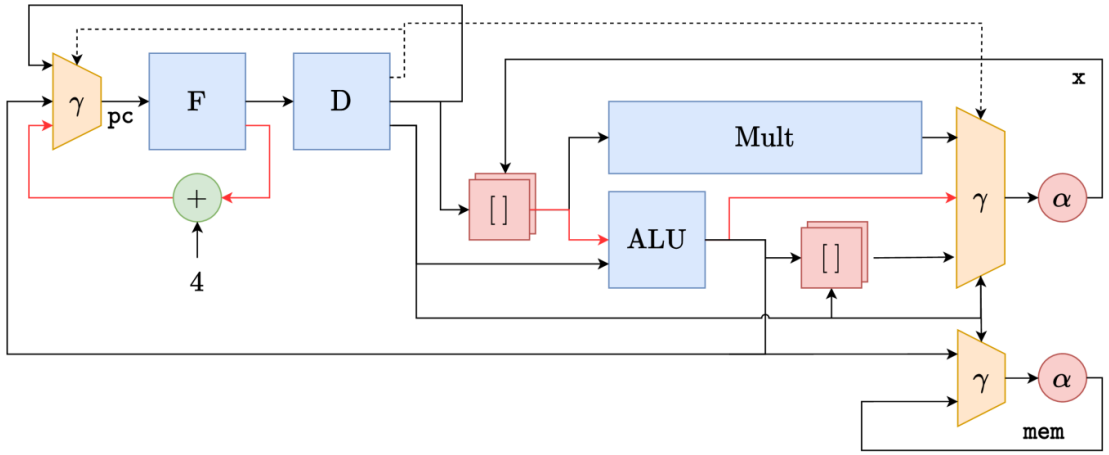
```
while(1) {
  unsigned int ir = fetch(mem, pc);
  pc += 4;

  struct decode_info dc = decode(ir);
  unsigned int rs1 = x[dc.rs1];
  unsigned int rs2 = x[dc.rs2];

  switch(opcode(dc)) {
  case RISCV_ADD:
    x[dc.rd] = add(rs1, rs2);
    break;
  case RISCV_MUL:
    x[dc.rd] = mul(rs1, rs2);
    break;
  case RISCV_JAL:
    pc = pc + dc.simm_J;
    break;
  case RISCV_LD:
    if(dc.funct3 == RISCV_LD_LB)
        x[dc.rd] = mem[rs1 + dc.imm_S];
    else if(/* ... */)
      // ...
    break;
  // ...
  }
}
```

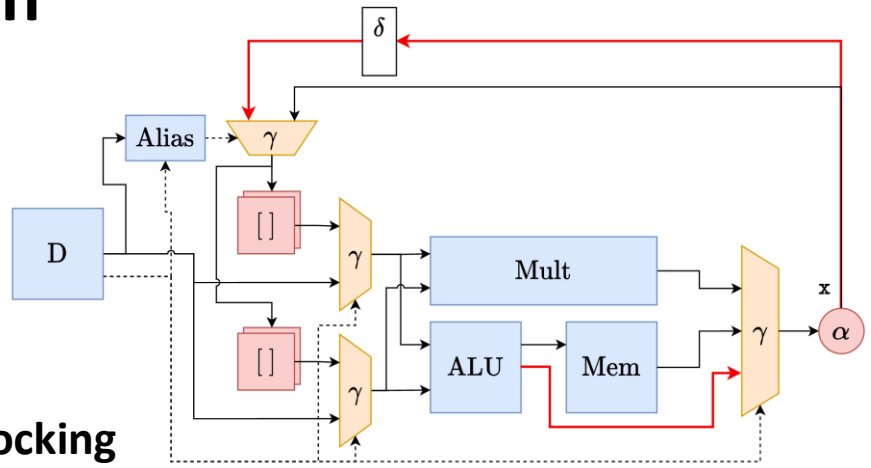


Automatic DSE Example: Memory Speculation

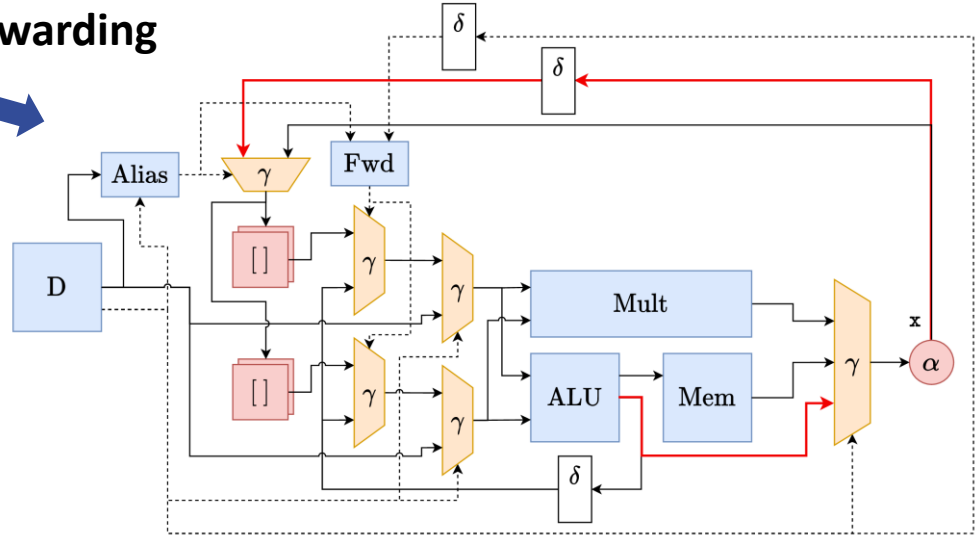


Gated-SSA CPU extracted from input program

Interlocking

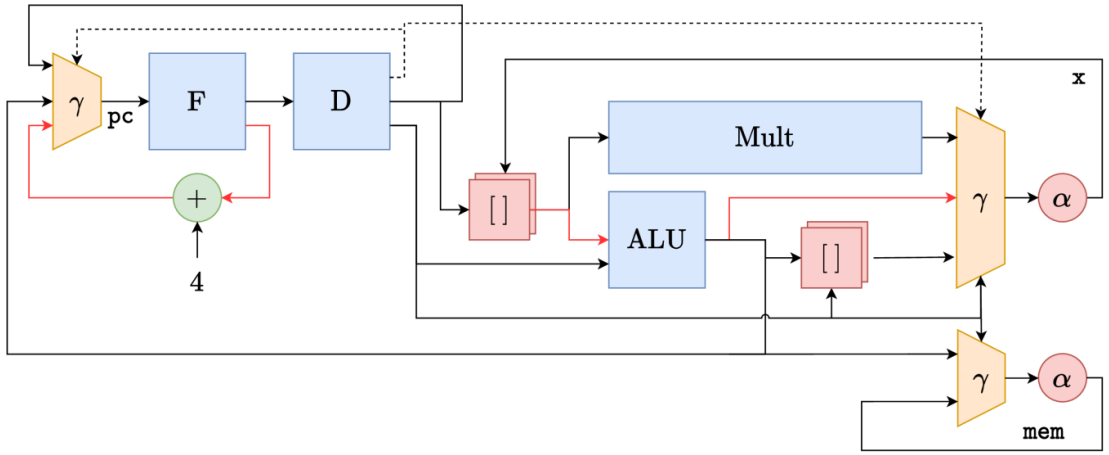


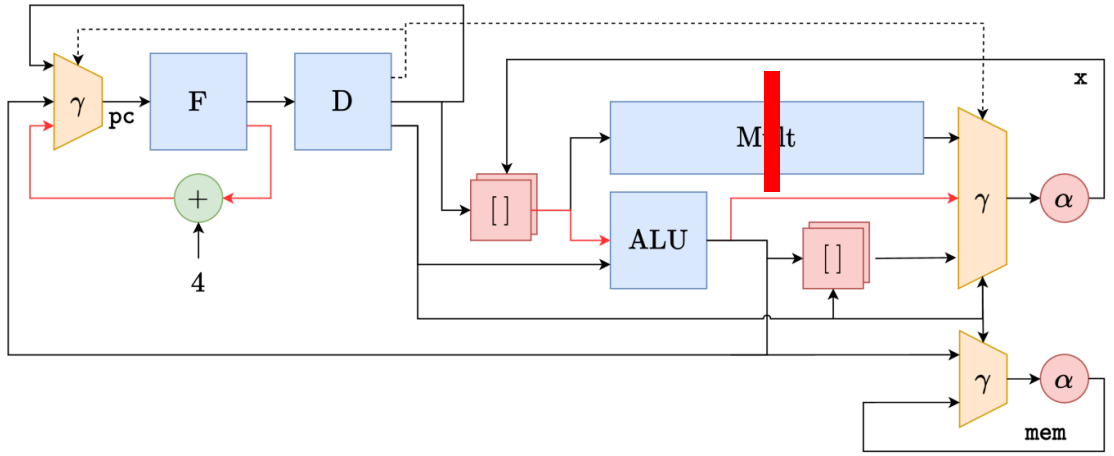
Forwarding

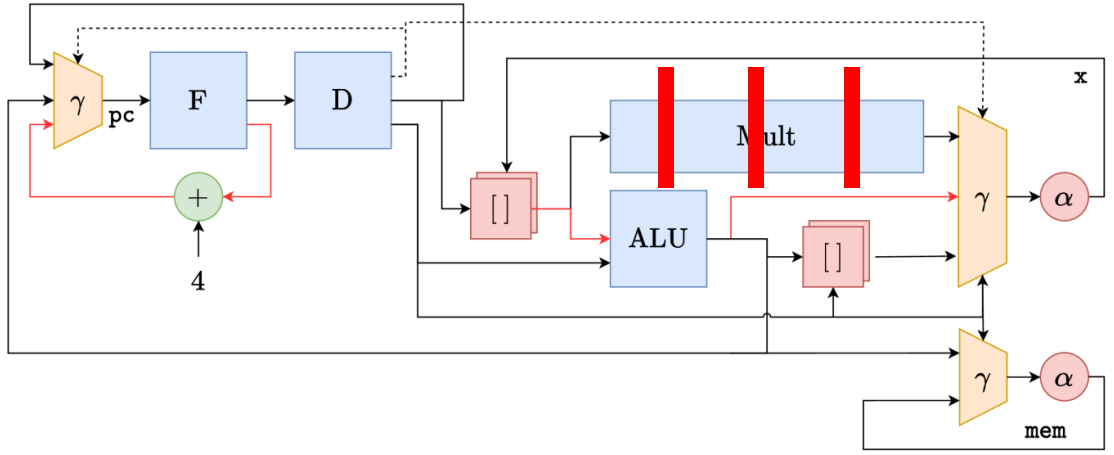


Jean-Michel Gorius, Simon Rokicki, Steven Derrien. A Unified Memory Dependency Framework for Speculative High-Level SynthesisCC 2024 - ACM SIGPLAN International Conference on Compiler Construction, Mar 2024, Edinburgh (Ecosse), United Kingdom.

Automatic DSE Example: Pipeline depth







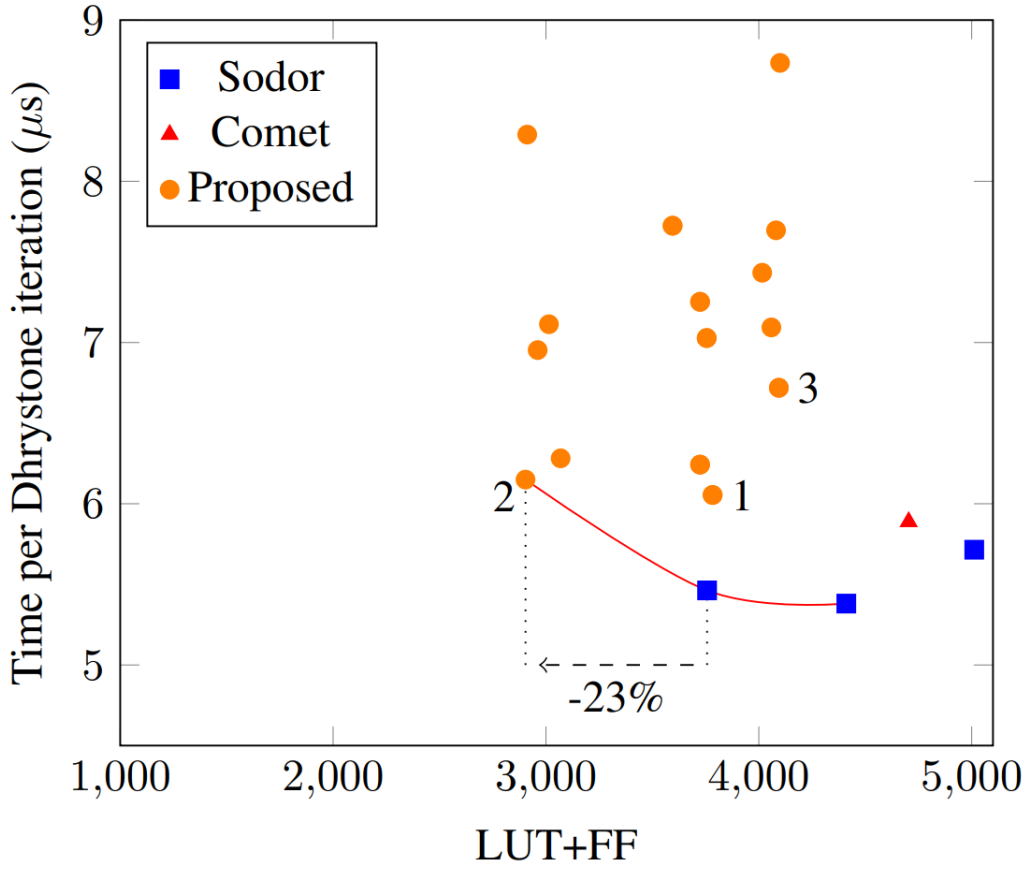
Gated-SSA CPU extracted from input program

Automatic Design Space Exploration

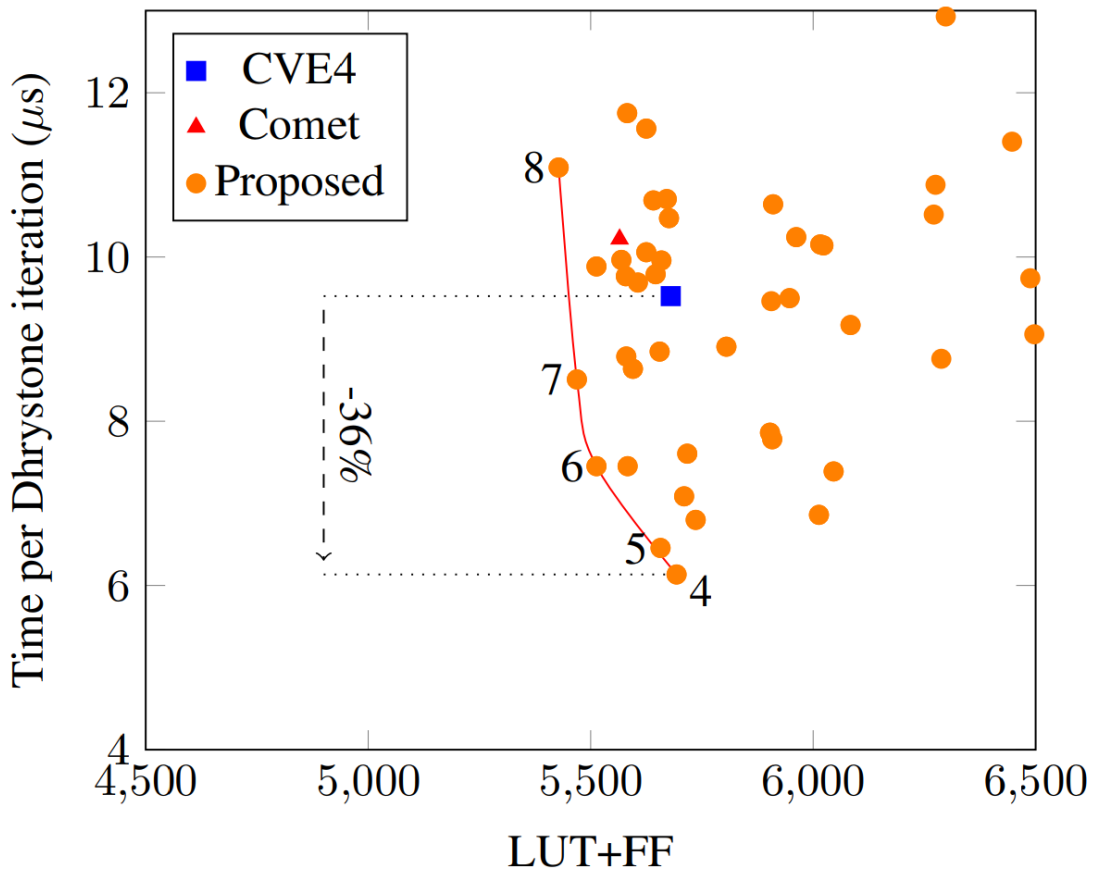
- Fully automated exploration of CPU designs
- Memory speculation, pipeline depth, non-pipelined multi-cycle operations

Results

RV32I



RV32IM



S. Rokicki, D. Pala, J. Paturel, and O. Sentieys, “What You Simulate if What You Synthesize: Design of a RISC-V Core from C++ Specifications,” in *RISC-V Workshop*, 2019
A. Traber, F. Zaruba, S. Stucki, A. Pullini, G. Haugou, E. Flamand, F. K. Gurkaynak, and L. Benini, “PULPino: A small single-core RISC-V SoC,” in *RISC-V Workshop*, 2016
https://github.com/usb-bar/riscv-sodor

Conclusion

Main results

- Fully automated exploration of CPU designs
- Memory speculation, pipeline depth, non-pipelined multi-cycle operations
- Competitive area and performance

Future work

- Unrolling main loop to synthesize superscalar
- How to integrate out-of-order execution

Contact

- Poster Wednesday (yesterday) at stand C-05
- Email: simon.rokicki@irisa.fr