Efficient Design Space Exploration for Dynamic & Speculative High-Level Synthesis

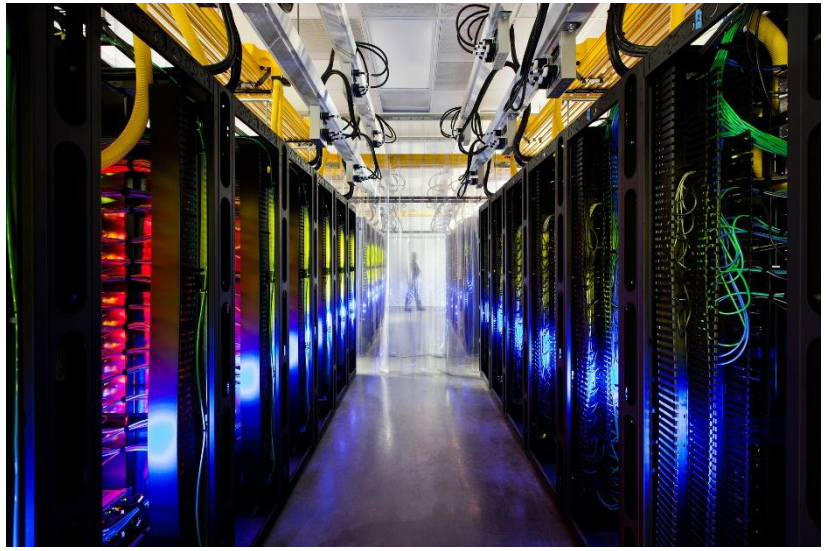
Dylan Leothaud, Jean-Michel Gorius, Simon Rokicki, Steven Derrien

Taran Team, Univ Rennes, IRISA, Inria

Compas 2024

Modern Hardware landscape

High performance processors

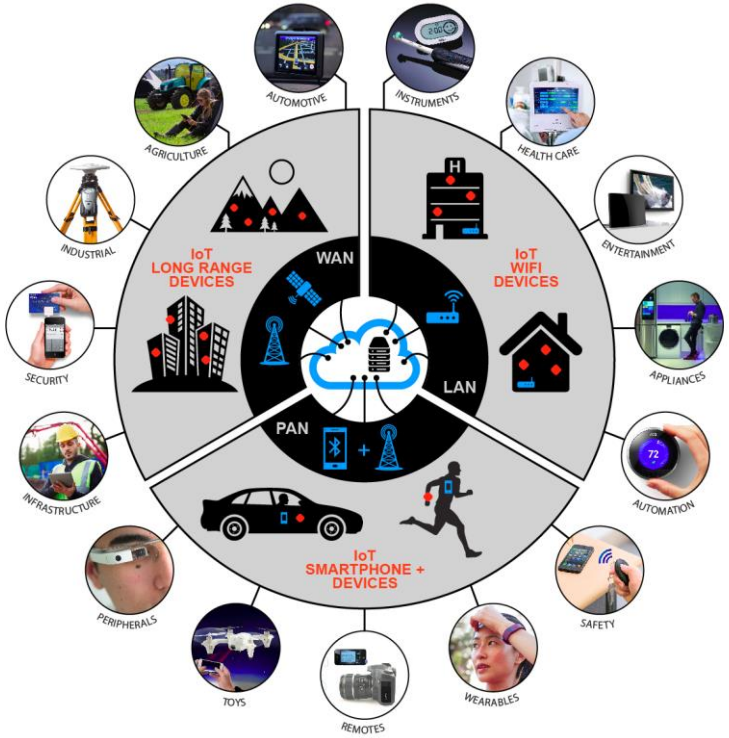


Hardware accelerators

ASIC

FPGA

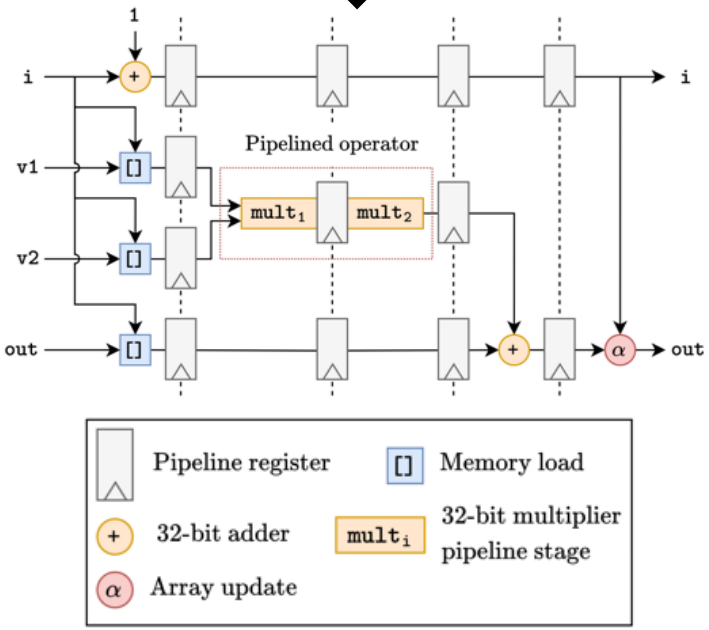
Embedded processors (Highly customized)



How to design all these circuits?

Manual hardware design using Verilog

```
void vec_mul(int *v1, int *v2, int *out,
             int n) {
  for (int i = 0; i < n; ++i)
    out[i] += v1[i] * v2[i];
}
```




```
module vec_mul (
        ap_clk,
        ap_rst,
        ap_start,
        ap_done,
        ap_idle,
        ap_ready,
        v1,
        v2,
        out_r_i,
        out_r_o,
        out_r_o_ap_vld,
        n
);

parameter    ap_ST_fsm_state1 = 3'd1;
parameter    ap_ST_fsm_state2 = 3'd2;
parameter    ap_ST_fsm_state3 = 3'd4;

input   ap_clk;
input   ap_rst;
input   ap_start;
output   ap_done;
output   ap_idle;
output   ap_ready;
input  [31:0] v1;
input  [31:0] v2;
input  [31:0] out_r_i;
output  [31:0] out_r_o;
output   out_r_o_ap_vld;
input  [31:0] n;

reg ap_done;
reg ap_idle;
reg ap_ready;
reg[31:0] out_r_o;

(* fsm_encoding = "none" *) reg   [2:0] ap_CS_fsm;
wire    ap_CS_fsm_state1;
wire   [31:0] mul_ln3_fu_57_p2;
reg   [31:0] mul_ln3_reg_63;
wire    ap_CS_fsm_state2;
wire    grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_ap_start;
wire    grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_ap_done;
wire    grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_ap_idle;
wire    grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_ap_ready;
wire   [31:0] grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_out_r_o;
wire    grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_out_r_o_ap_vld;
reg    grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_ap_start_reg;
wire    ap_CS_fsm_state3;
reg   [2:0] ap_NS_fsm;
reg    ap_ST_fsm_state1_blk;
wire    ap_ST_fsm_state2_blk;
reg    ap_ST_fsm_state3_blk;
wire    ap_ce_reg;

// power-on initialization
initial begin
#0 ap_CS_fsm = 3'd1;
#0 grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_ap_start_reg = 1'b0;
end

vec_mul_vec_mul_Pipeline_VITIS_LOOP_2_1 grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48(
    .ap_clk(ap_clk),
    .ap_rst(ap_rst),
    .ap_start(grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_ap_start),
    .ap_done(grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_ap_done),
    .ap_idle(grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_ap_idle),
    .ap_ready(grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_ap_ready),
    .n(n),
    .out_r_i(out_r_i),
    .out_r_o(grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_out_r_o),
    .out_r_o_ap_vld(grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_out_r_o_ap_vld),
    .mul_ln3(mul_ln3_reg_63)
);
```

```
vec_mul_mul_32s_32s_32_1_1 #(
    .ID( 1 ),
    .NUM_STAGE( 1 ),
    .din0_WIDTH( 32 ),
    .din1_WIDTH( 32 ),
    .dout_WIDTH( 32 ))
mul_32s_32s_32_1_1_U4(
    .din0(v2),
    .din1(v1),
    .dout(mul_ln3_fu_57_p2)
);

always @ (posedge ap_clk) begin
    if (ap_rst == 1'b1) begin
        ap_CS_fsm <= ap_ST_fsm_state1;
    end else begin
        ap_CS_fsm <= ap_NS_fsm;
    end
end

always @ (posedge ap_clk) begin
    if (ap_rst == 1'b1) begin
        grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_ap_start_reg <= 1'b0;
    end else begin
        if ((1'b1 == ap_CS_fsm_state2)) begin
            grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_ap_start_reg <= 1'b1;
        end else if ((grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_ap_ready == 1'b1))
↪   begin
            grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_ap_start_reg <= 1'b0;
        end
    end
end

always @ (posedge ap_clk) begin
    if ((1'b1 == ap_CS_fsm_state1)) begin
        mul_ln3_reg_63 <= mul_ln3_fu_57_p2;
    end
end

always @ (*) begin
    if ((ap_start == 1'b0)) begin
        ap_ST_fsm_state1_blk = 1'b1;
    end else begin
        ap_ST_fsm_state1_blk = 1'b0;
    end
end

assign ap_ST_fsm_state2_blk = 1'b0;

always @ (*) begin
    if ((grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_ap_done == 1'b0)) begin
        ap_ST_fsm_state3_blk = 1'b1;
    end else begin
        ap_ST_fsm_state3_blk = 1'b0;
    end
end

always @ (*) begin
    if (((grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_ap_done == 1'b1) & (1'b1 ==
↪   ap_CS_fsm_state3))) begin
        ap_done = 1'b1;
    end else begin
        ap_done = 1'b0;
    end
end
```

```
always @ (*) begin
    if (((ap_start == 1'b0) & (1'b1 == ap_CS_fsm_state1))) begin
        ap_idle = 1'b1;
    end else begin
        ap_idle = 1'b0;
    end
end

always @ (*) begin
    if (((grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_ap_done == 1'b1) & (1'b1 ==
↪   ap_CS_fsm_state3))) begin
        ap_ready = 1'b1;
    end else begin
        ap_ready = 1'b0;
    end
end

always @ (*) begin
    if (((grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_out_r_o_ap_vld == 1'b1) & (1'b1
↪   == ap_CS_fsm_state3))) begin
        out_r_o = grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_out_r_o;
    end else begin
        out_r_o = out_r_i;
    end
end

always @ (*) begin
    case (ap_CS_fsm)
        ap_ST_fsm_state1 : begin
            if (((ap_start == 1'b1) & (1'b1 == ap_CS_fsm_state1))) begin
                ap_NS_fsm = ap_ST_fsm_state2;
            end else begin
                ap_NS_fsm = ap_ST_fsm_state1;
            end
        end
        ap_ST_fsm_state2 : begin
            ap_NS_fsm = ap_ST_fsm_state3;
        end
        ap_ST_fsm_state3 : begin
            if (((grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_ap_done == 1'b1) &
↪   (1'b1 == ap_CS_fsm_state3))) begin
                ap_NS_fsm = ap_ST_fsm_state1;
            end else begin
                ap_NS_fsm = ap_ST_fsm_state3;
            end
        end
        default : begin
            ap_NS_fsm = 'bx;
        end
    endcase
end

assign ap_CS_fsm_state1 = ap_CS_fsm[32'd0];

assign ap_CS_fsm_state2 = ap_CS_fsm[32'd1];

assign ap_CS_fsm_state3 = ap_CS_fsm[32'd2];

assign grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_ap_start =
↪   grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_ap_start_reg;

assign out_r_o_ap_vld = grp_vec_mul_Pipeline_VITIS_LOOP_2_1_fu_48_out_r_o_ap_vld;

endmodule //vec_mul
```

+ 3 other modules

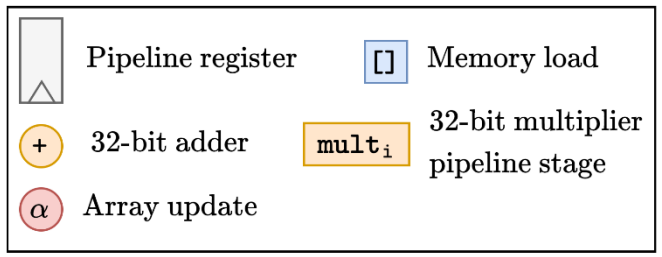
Designing various hardware using HDL is complex and time-consuming

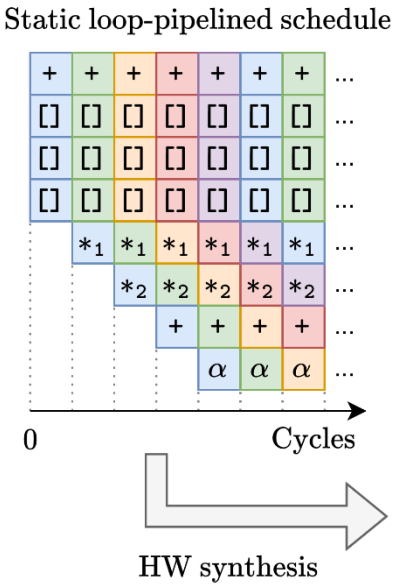
How to automatize hardware synthesis?

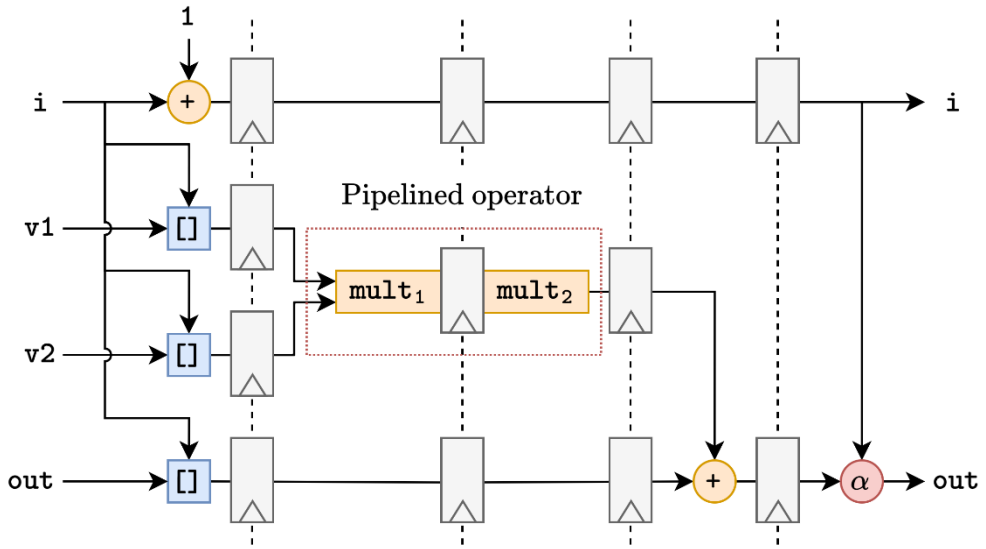
High-Level Synthesis (HLS) tools generate HDL from C++ description

```
void vec_mul(int *v1, int *v2, int *out,
             int n) {
  for (int i = 0; i < n; ++i)
    out[i] += v1[i] * v2[i];
}
```

HLS





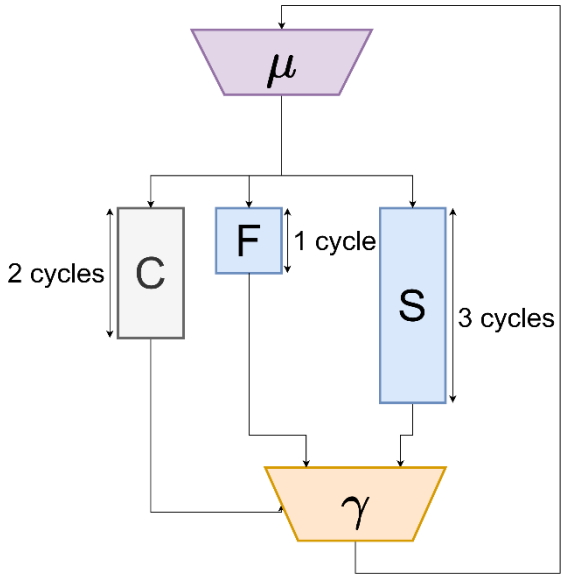


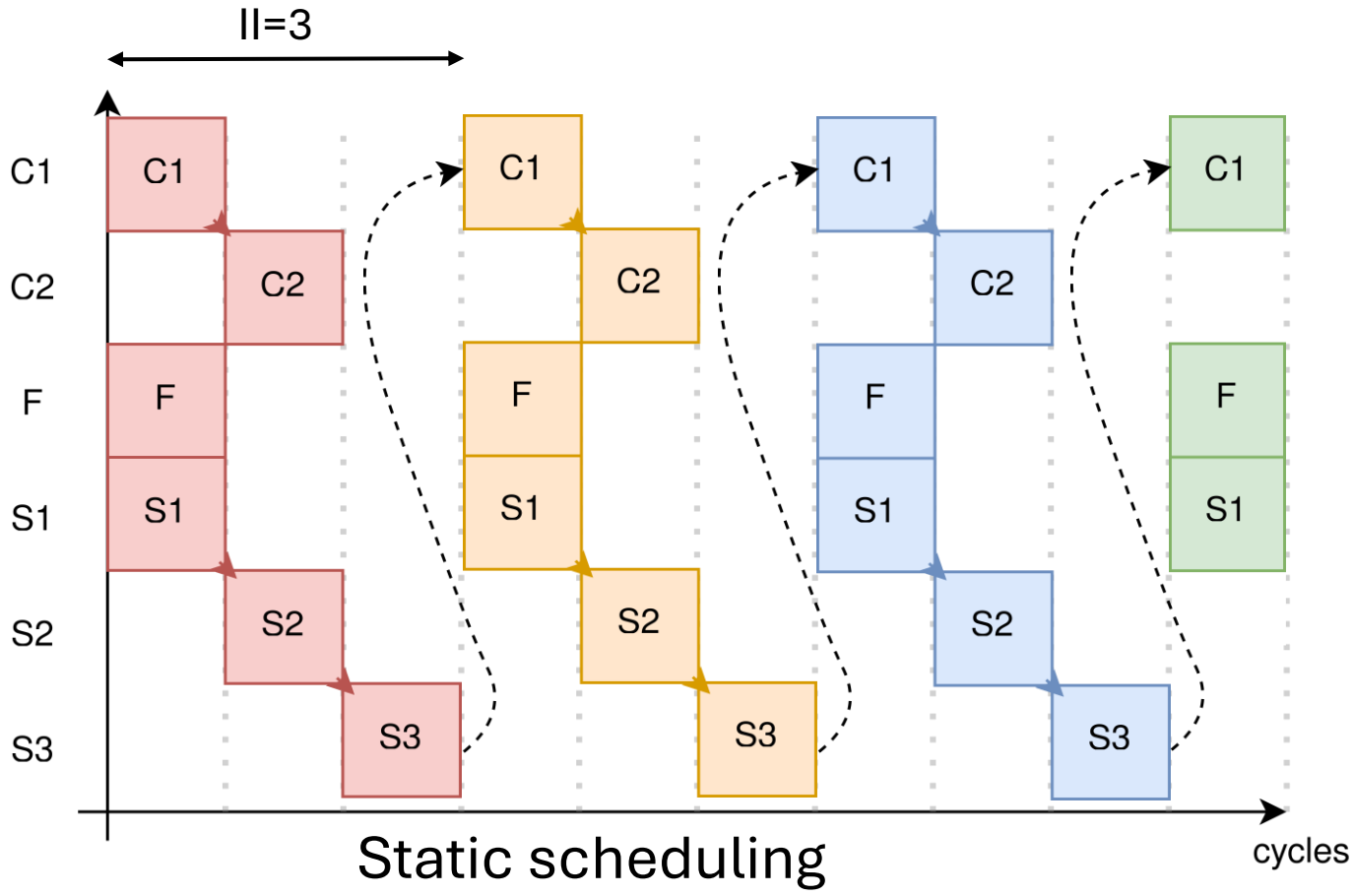
The key step for HLS is operation scheduling

An overview of scheduling techniques

Commercial HLS tools rely on static scheduling

```
while (1) {
  if (C(x)) // 2 cycles
    x = F(x); // 1 cycle
  else
    x = S(x); // 3 cycles
}
```

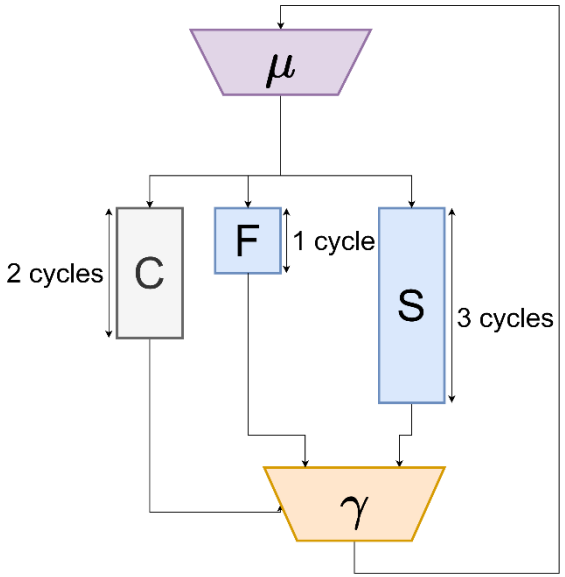


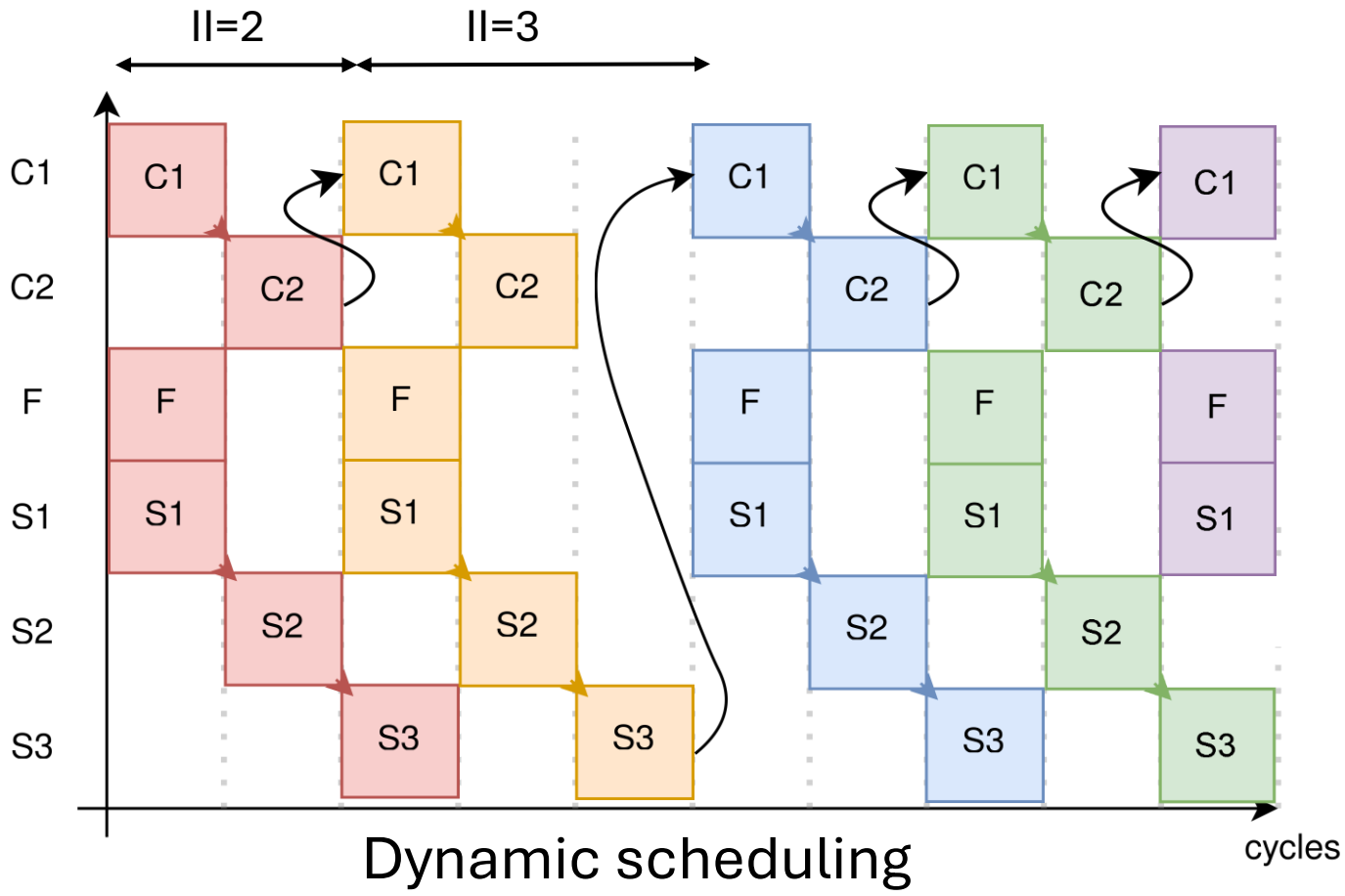


An overview of scheduling techniques

Commercial HLS tools rely on static scheduling

```
while (1) {
  if (C(x)) // 2 cycles
    x = F(x); // 1 cycle
  else
    x = S(x); // 3 cycles
}
```



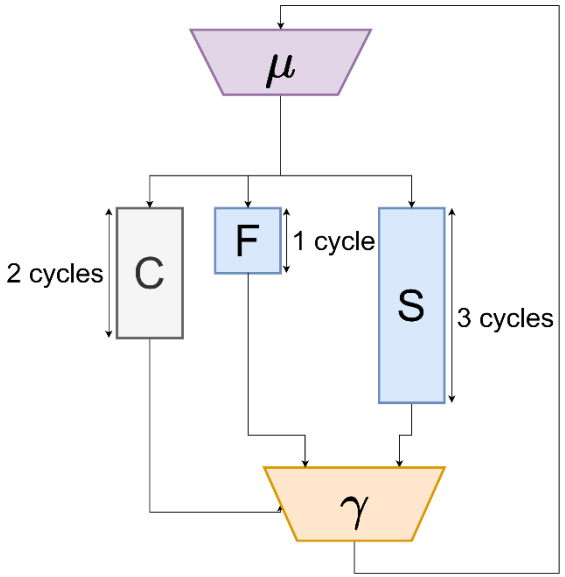


Dynamic scheduling

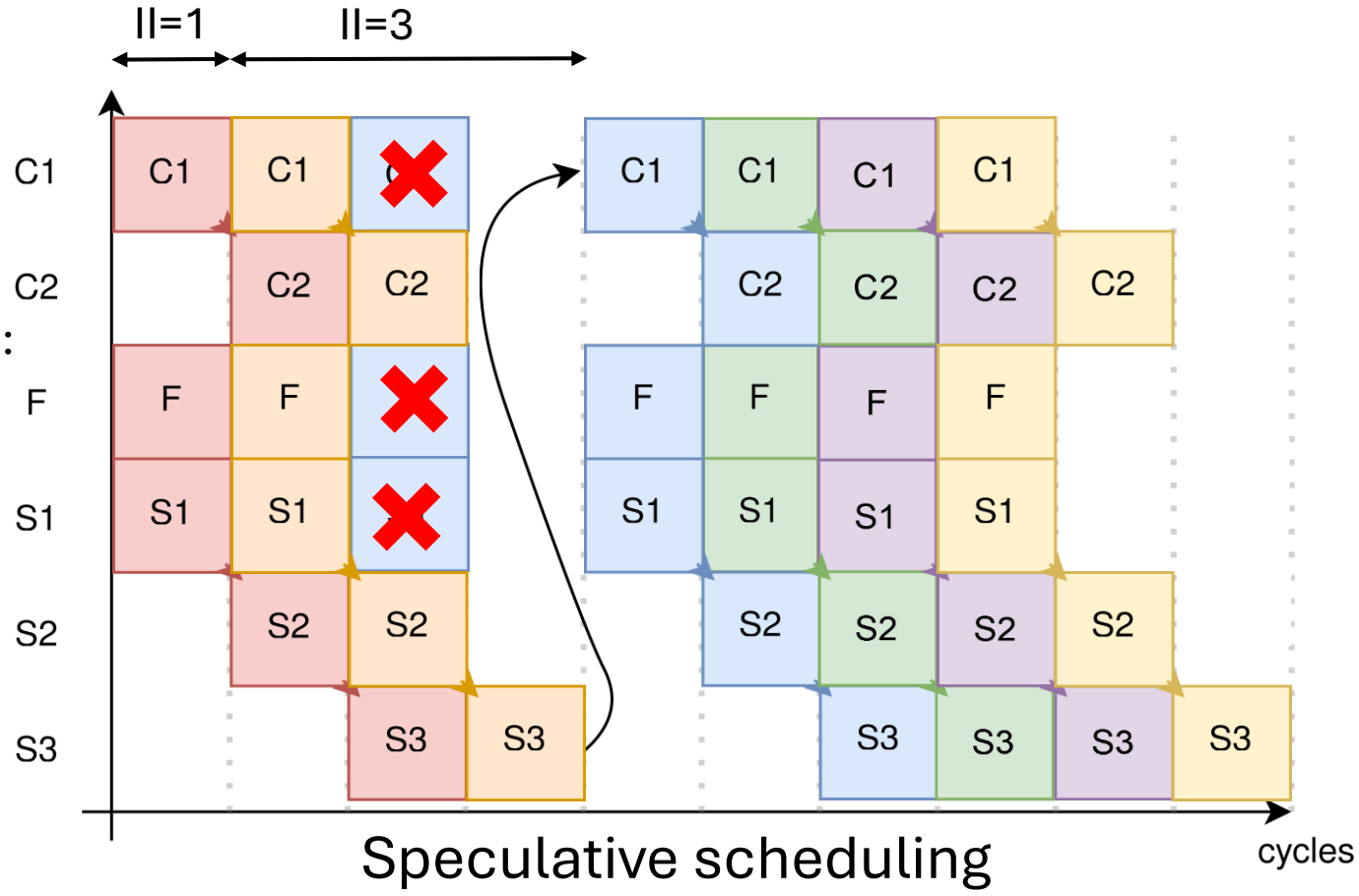
An overview of scheduling techniques

Commercial HLS tools rely on static scheduling

```
while (1) {
  if (C(x)) // 2 cycles
    x = F(x); // 1 cycle
  else
    x = S(x); // 3 cycles
}
```



Speculation **hypothesis**:
C always returns true

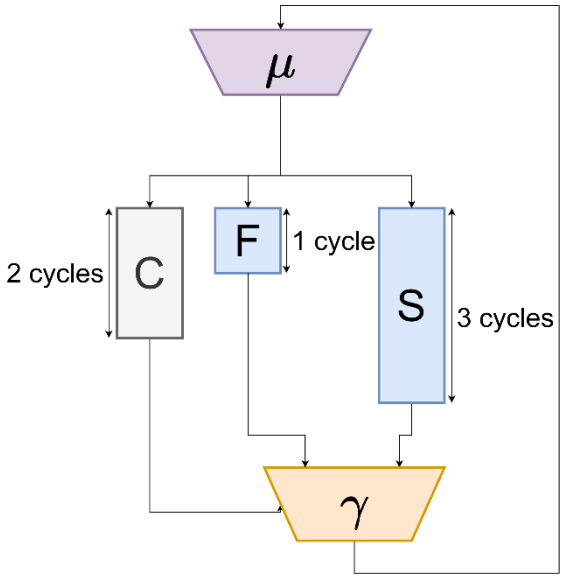


Speculative scheduling

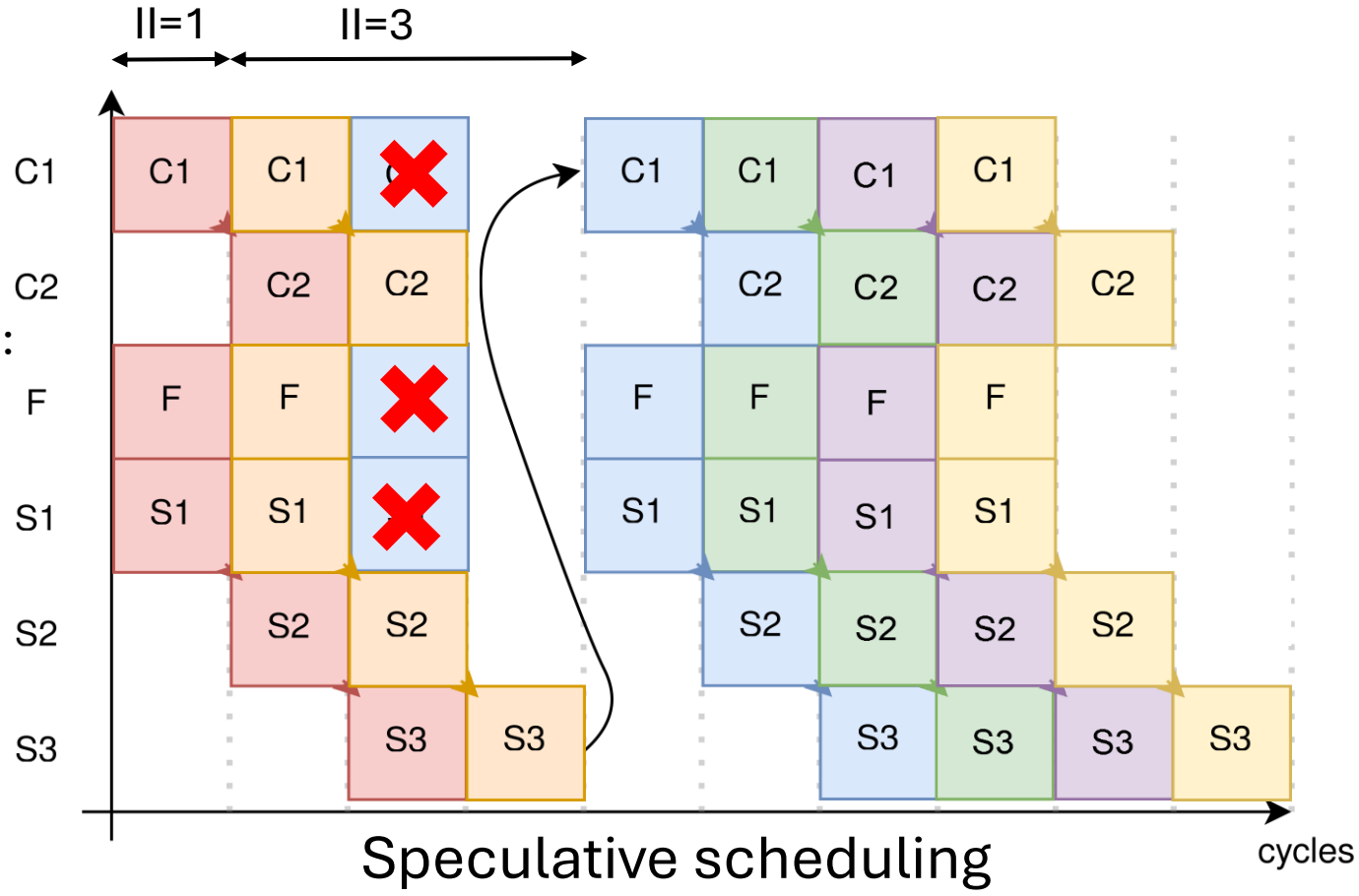
An overview of scheduling techniques

Commercial HLS tools rely on static scheduling

```
while (1) {
  if (C(x)) // 2 cycles
    x = F(x); // 1 cycle
  else
    x = S(x); // 3 cycles
}
```



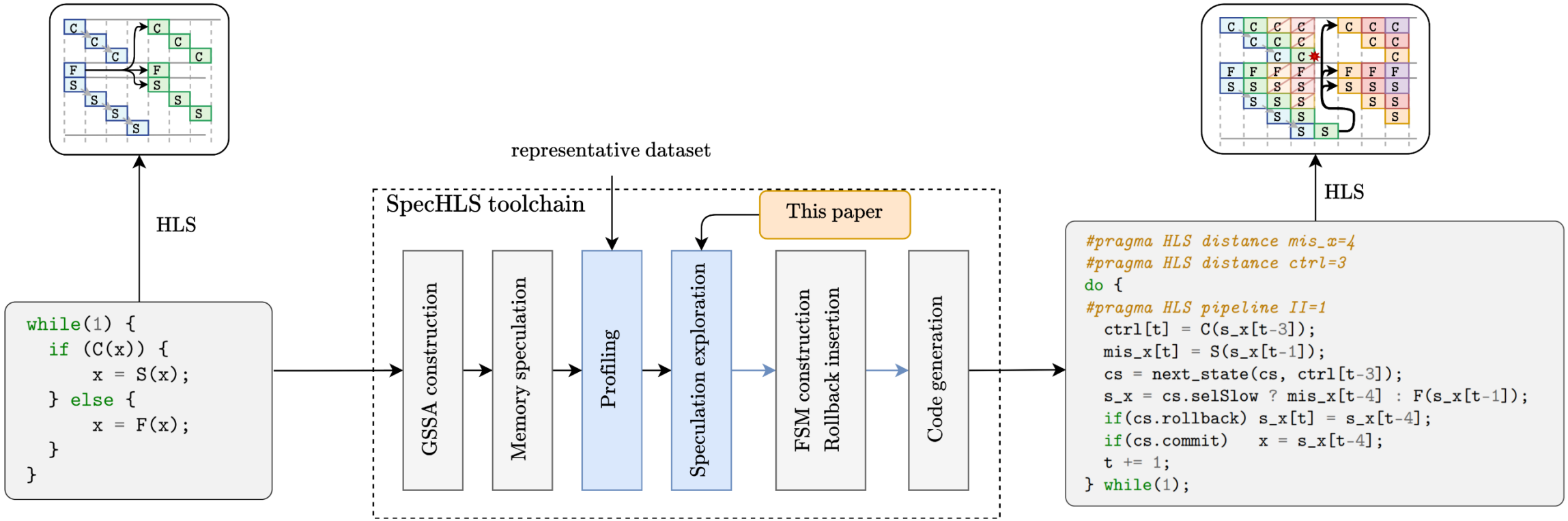
Speculation **hypothesis**:
C always returns true



Speculative schedules are more efficient than static schedules

The SpecHLS compilation flow

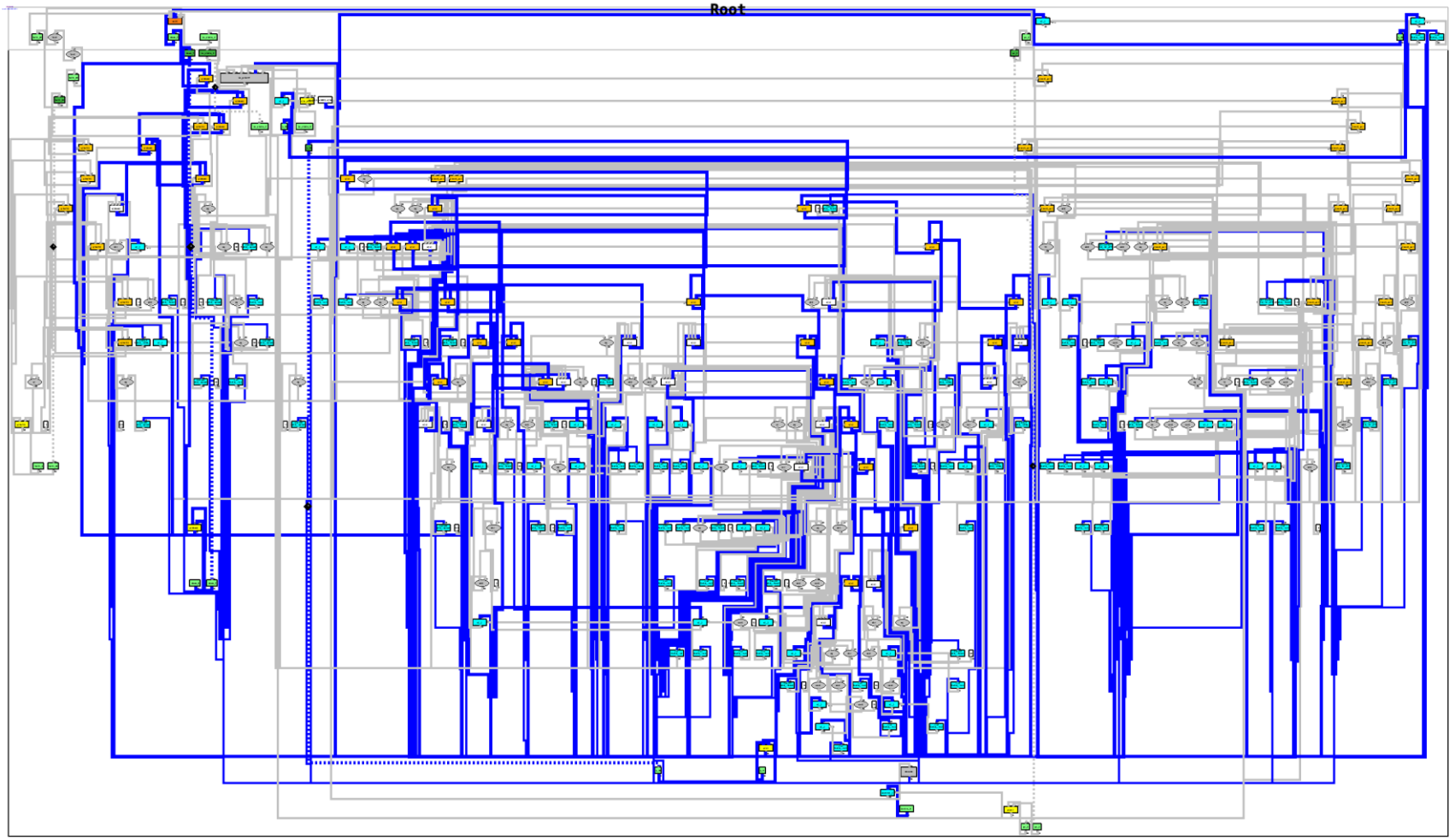
SpecHLS is a C-to-C compiler that allows the generation of speculative circuits using HLS tools



Where, and how, to speculate in synthesized circuits?

Gorius, J.-M., Rokicki, S., and Derrien, S. (2022). SpecHLS: Speculative Accelerator Design using High-Level Synthesis. IEEE MICRO.

Real life example: a small CPU



62 binary gamma nodes → 4.10^{28} configurations!

Problem statement

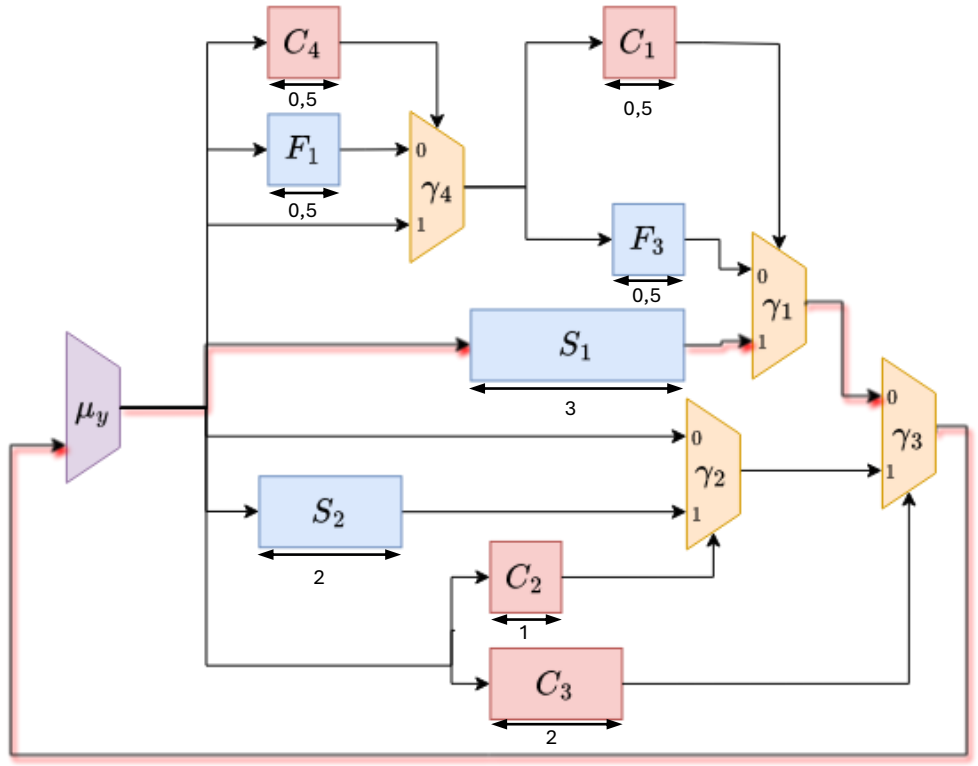
A valid configuration is a speculation hypothesis such that:

- The speculative schedule has II=1
- The configuration is minimal (i.e. is not a superset of a valid configuration)
- The estimated probability of mispeculation is lower than a threshold (ex: 90%)

Configurations in a simple example

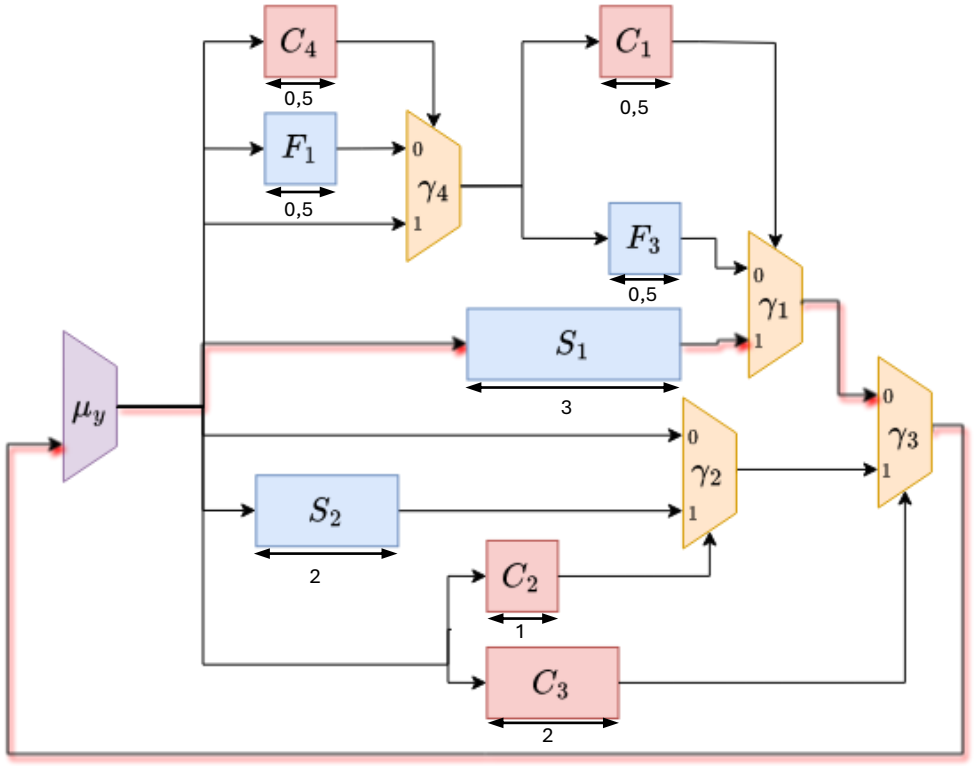
A valid configuration is a speculation hypothesis such that:
- The speculative schedule has II=1
- The configuration is minimal (i.e. is not a superset of a valid configuration)
- The estimated probability of mispeculation is lower than a threshold (ex: 90%)

```
while (1) {
  if (C3(x)) {
      x = S2(x);
  } else {
    if (C4(x))
      tmp = x;
    else
      tmp = F1(x);
    if (C1(tmp))
      x = S1(x);
    else
      x = F3(tmp);
  }
}
```



Configurations in a simple example

A valid configuration is a speculation hypothesis such that:
- The speculative schedule has II=1
- The configuration is minimal (i.e. is not a superset of a valid configuration)
- The estimated probability of mispeculation is lower than a threshold (ex: 90%)

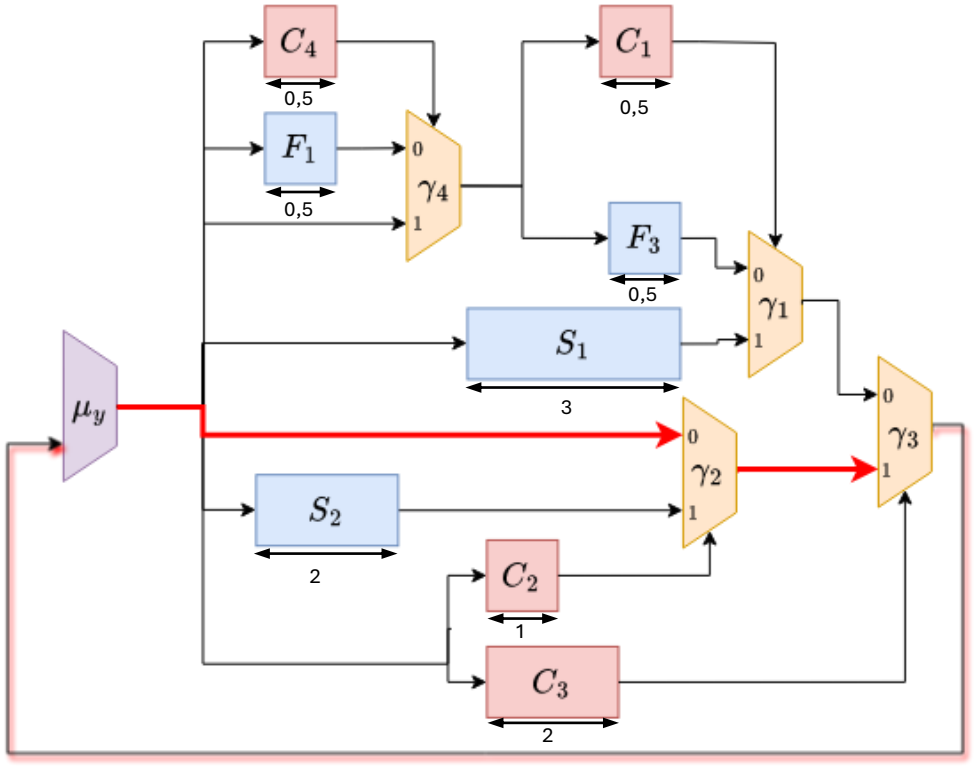


Configuration	II	Mispeculation probability
ø	3	0
...		

Configurations in a simple example

A valid configuration is a speculation hypothesis such that:
- The speculative schedule has II=1
- The configuration is minimal (i.e. is not a superset of a valid configuration)
- The estimated probability of mispeculation is lower than a threshold (ex: 90%)

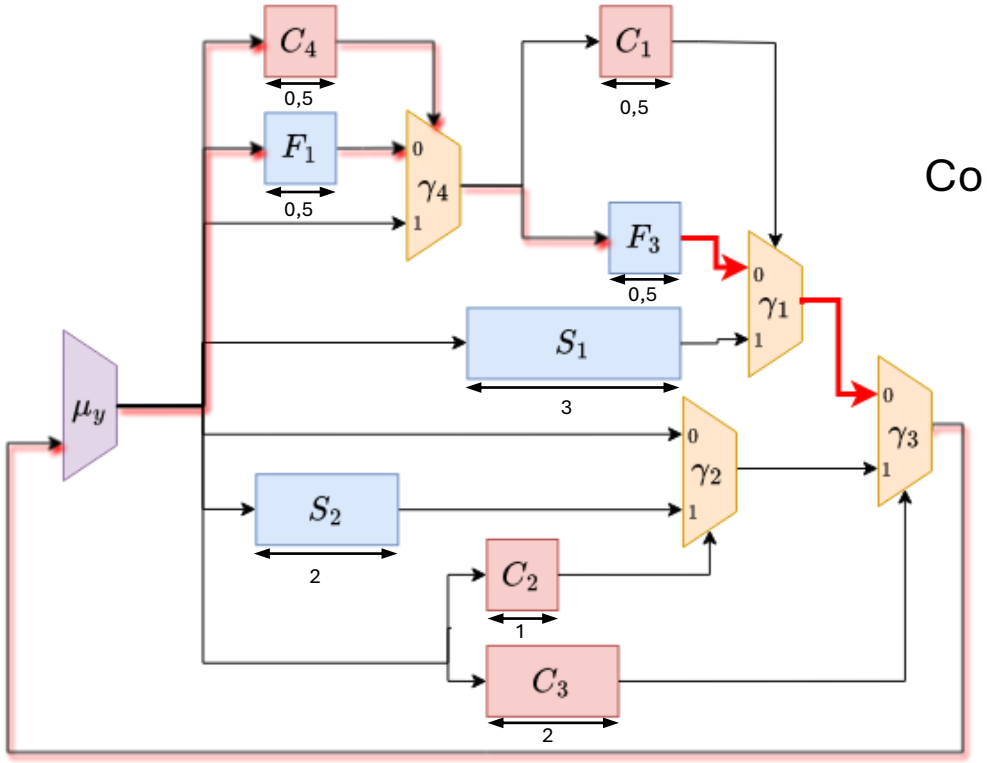
Not respected



Configuration	II	Mispeculation probability
ø	3	0
$\gamma_2 \rightarrow 0$; $\gamma_3 \rightarrow 1$	1	0.9
...		

Configurations in a simple example

A valid configuration is a speculation hypothesis such that:
- The speculative schedule has II=1
- The configuration is minimal (i.e. is not a superset of a valid configuration)
- The estimated probability of mispeculation is lower than a threshold (ex: 90%)



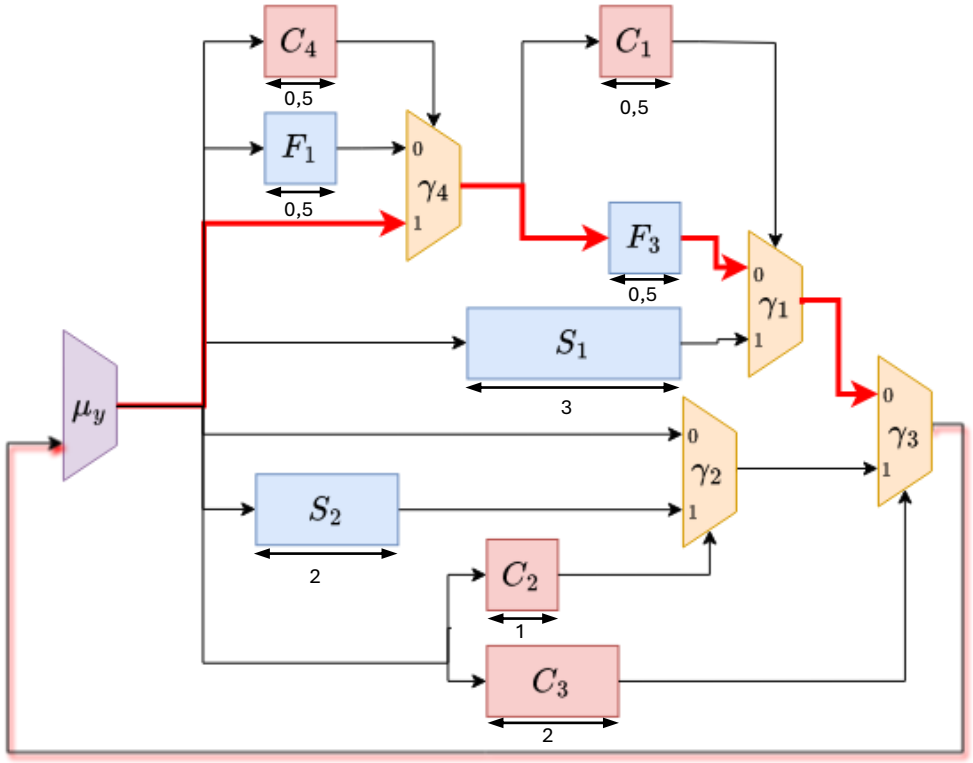
Configuration valid!

Configuration	II	Mispeculation probability
ø	3	0
$\gamma_2 \rightarrow 0$; $\gamma_3 \rightarrow 1$	1	0.9
$\gamma_1 \rightarrow 0$; $\gamma_3 \rightarrow 0$	1	0.6
...		

Configurations in a simple example

A valid configuration is a speculation hypothesis such that:
- The speculative schedule has II=1
- The configuration is minimal (i.e. is not a superset of a valid configuration)
- The estimated probability of mispeculation is lower than a threshold (ex: 90%)

Not respected

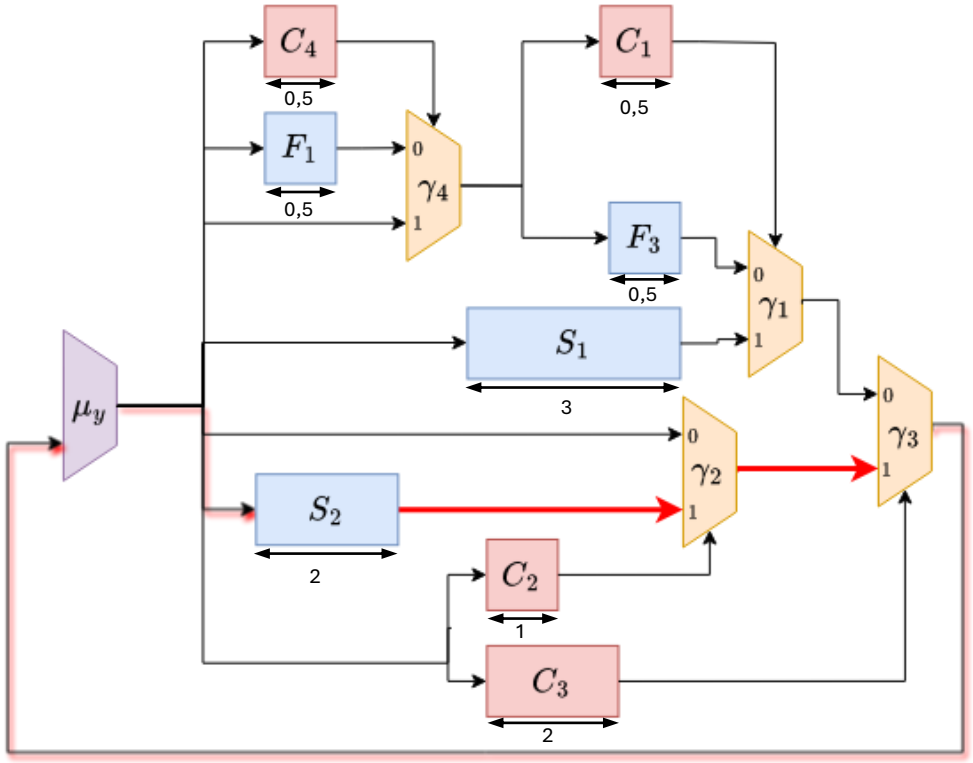


Configuration	II	Mispeculation probability
∅	3	0
$\gamma_2 \to 0; \gamma_3 \to 1$	1	0.9
$\gamma_1 \to 0; \gamma_3 \to 0$	1	0.6
$\gamma_1 \to 0; \gamma_3 \to 0; \gamma_4 \to 1$	1	0.7
...		

Configurations in a simple example

A valid configuration is a speculation hypothesis such that:
- The speculative schedule has II=1
- The configuration is minimal (i.e. is not a superset of a valid configuration)
- The estimated probability of mispeculation is lower than a threshold (ex: 90%)

Not respected

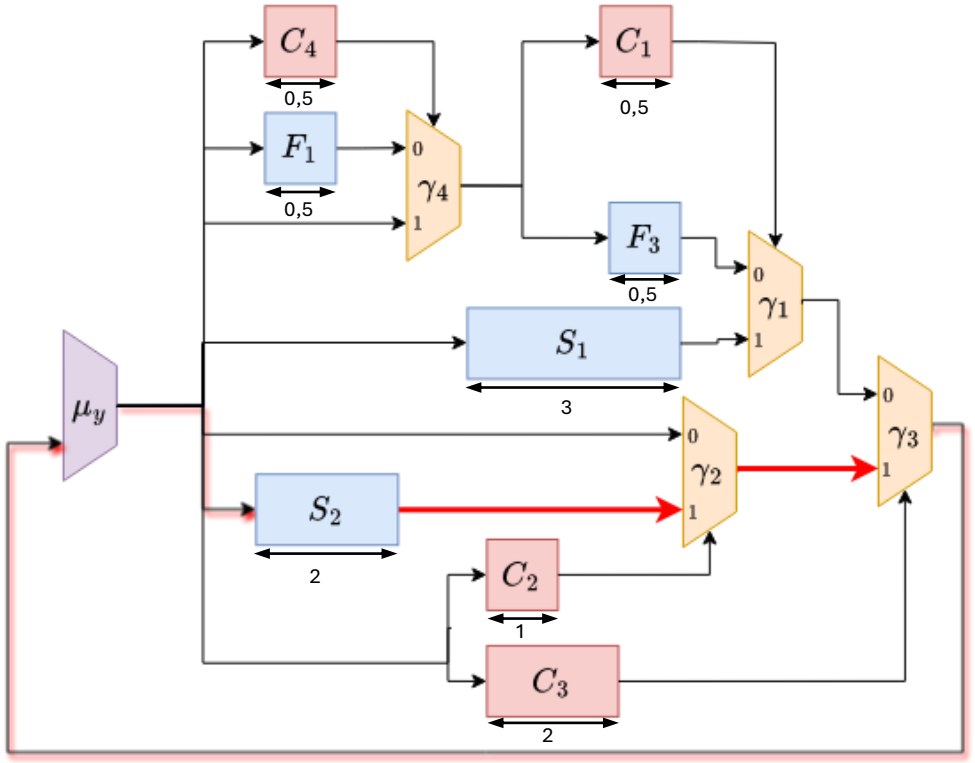


Configuration	II	Mispeculation probability
∅	3	0
$\gamma_2 \to 0$; $\gamma_3 \to 1$	1	0.9
$\gamma_1 \to 0$; $\gamma_3 \to 0$	1	0.6
$\gamma_1 \to 0$; $\gamma_3 \to 0$; $\gamma_4 \to 1$	1	0.7
$\gamma_2 \to 1$; $\gamma_3 \to 1$; …	2	…
…		

Configurations in a simple example

A valid configuration is a speculation hypothesis such that:
- The speculative schedule has II=1
- The configuration is minimal (i.e. is not a superset of a valid configuration)
- The estimated probability of mispeculation is lower than a threshold (ex: 90%)

Not respected



Configuration	II	Mispeculation probability
∅	3	0
$\gamma_2 \to 0$; $\gamma_3 \to 1$	1	0.9
$\gamma_1 \to 0$; $\gamma_3 \to 0$	1	0.6
$\gamma_1 \to 0$; $\gamma_3 \to 0$; $\gamma_4 \to 1$	1	0.7
$\gamma_2 \to 1$; $\gamma_3 \to 1$; ...	2	...
...		

How to find all valid speculation configurations?

Proposed approach: a branch and bound algorithm

We propose a branch and bound algorithm to find all valid configuration :

- Constructs configuration by extending explored ones
- Prune non-minimal configuration
- Prune configurations with a mispeculation probability too high
- Prune configurations that can’t be extended to get II=1

Leothaud, D., Gorius, J.-M., Rokicki, S., and Derrien, S. (2024). Efficient Design Space Exploration for Dynamic and Speculative High-Level Synthesis. FPL’24.

Non-minimal configuration pruning

We propose a branch and bound algorithm to find all valid configuration :

- Constructs configuration by extending explored ones
- **Prune non-minimal configuration**
- Prune configurations with a mispeculation probability too high
- Prune configurations that can't be extended to get II=1

Supersets of a valid configuration already explored do not need to be explored

Extending a configuration can only decrease the II
Collecting all valid minimal configurations allow to recreate all valid configurations

Leothaud, D., Gorius, J.-M., Rokicki, S., and Derrien, S. (2024). Efficient Design Space Exploration for Dynamic and Speculative High-Level Synthesis. FPL'24.

Probability pruning

We propose a branch and bound algorithm to find all valid configuration :

- Constructs configuration by extending explored ones
- Prune non-minimal configuration
- **Prune configurations with a mispeculation probability too high**
- Prune configurations that can't be extended to get II=1

Configurations with an estimated mispeculation probability higher than a threshold can be pruned

With a mispeculation probability too high, the ratio between area overhead and speedup is not deemed reasonable

Leothaud, D., Gorius, J.-M., Rokicki, S., and Derrien, S. (2024). Efficient Design Space Exploration for Dynamic and Speculative High-Level Synthesis. FPL'24.

Non-unit II configurations pruning

We propose a branch and bound algorithm to find all valid configuration :

- Constructs configuration by extending explored ones
- Prune non-minimal configuration
- Prune configurations with a mispeculation probability too high
- **Prune configurations that can't be extended to get II=1**

A static analysis may find that a configuration can't be extended to get II=1

Leothaud, D., Gorius, J.-M., Rokicki, S., and Derrien, S. (2024). Efficient Design Space Exploration for Dynamic and Speculative High-Level Synthesis. FPL'24.

Experimental results

Benchmark	Design space size			Runtime
	γ-nodes	Baseline	Heuristics	
FPU	21	30.9B	116k	1055s
SpMM	12	708k	60	6s
RISC-V CPU	16	752M	1.46k	263s
Superscalar	16	178M	1.19k	177s

The proposed approach decreases by several orders of magnitude the number of explored configurations

Leothaud, D., Gorius, J.-M., Rokicki, S., and Derrien, S. (2024). Efficient Design Space Exploration for Dynamic and Speculative High-Level Synthesis. FPL'24.

Conclusion

Speculation opens up new opportunities for High-Level Synthesis

The challenge is to discover **where**, and **how**, to apply speculation

Our approach reduces the exploration space size by several orders of magnitude

Gorius, J.-M., Rokicki, S., and Derrien, S. (2022). SpecHLS: Speculative Accelerator Design using High-Level Synthesis. IEEE MICRO.
Leothaud, D., Gorius, J.-M., Rokicki, S., and Derrien, S. (2024). Efficient Design Space Exploration for Dynamic and Speculative High-Level Synthesis. FPL'24.