POLITECNICO
MILANO 1863

RISC CORE SPECIALIZATION FOR MOLECULAR DYNAMICS

30-06-2022

Emilio Ingenito
(emilio.ingenito@mail.polimi.it)

Professor: Marco Domenico Santambrogio

Supervisor: Francesco Peverelli

1. INTRODUCTION

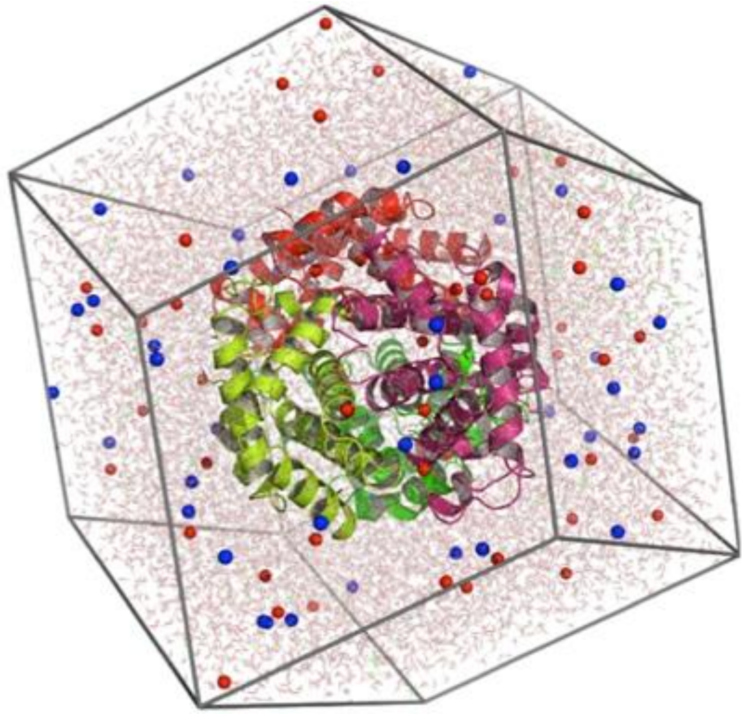
MOLECULAR DYNAMICS

↓

MD SIMULATORS

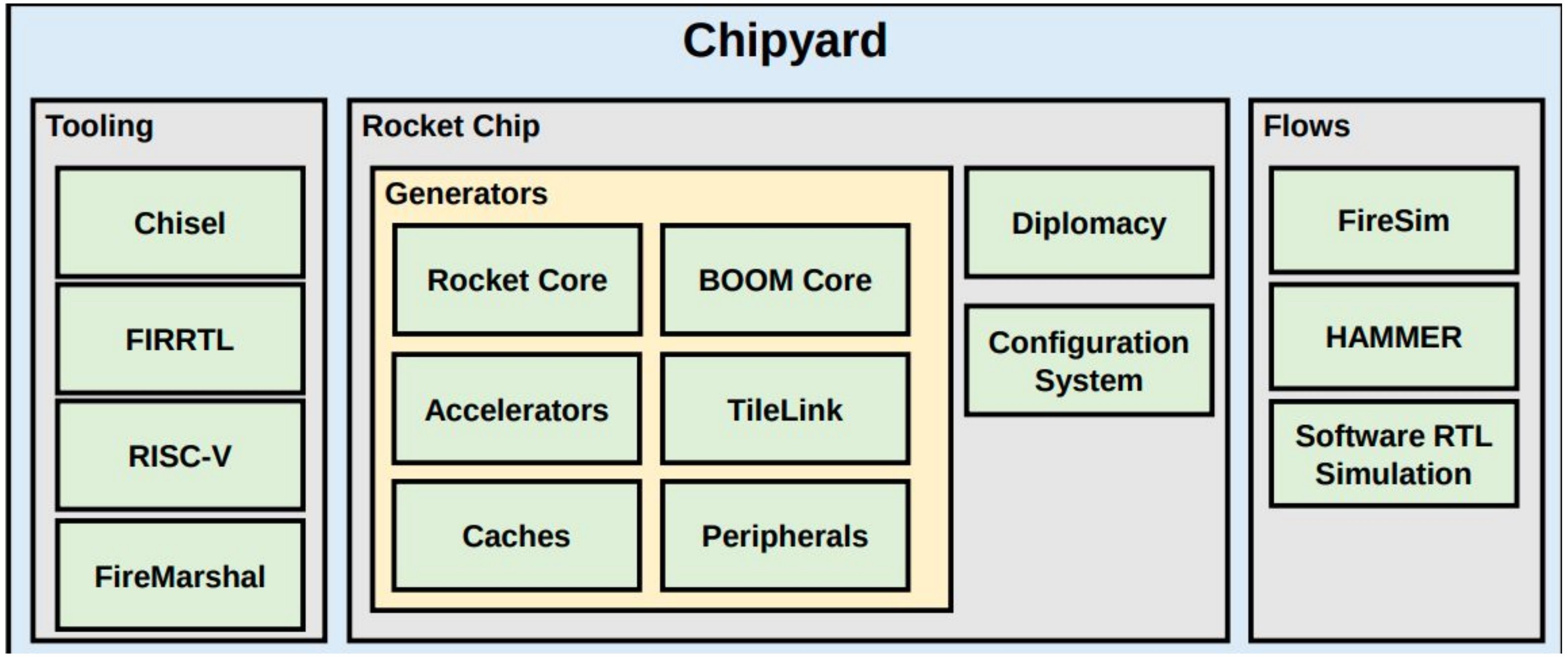
↓

miniMD
(https://github.com/Mantevo/miniMD)



1. INTRODUCTION

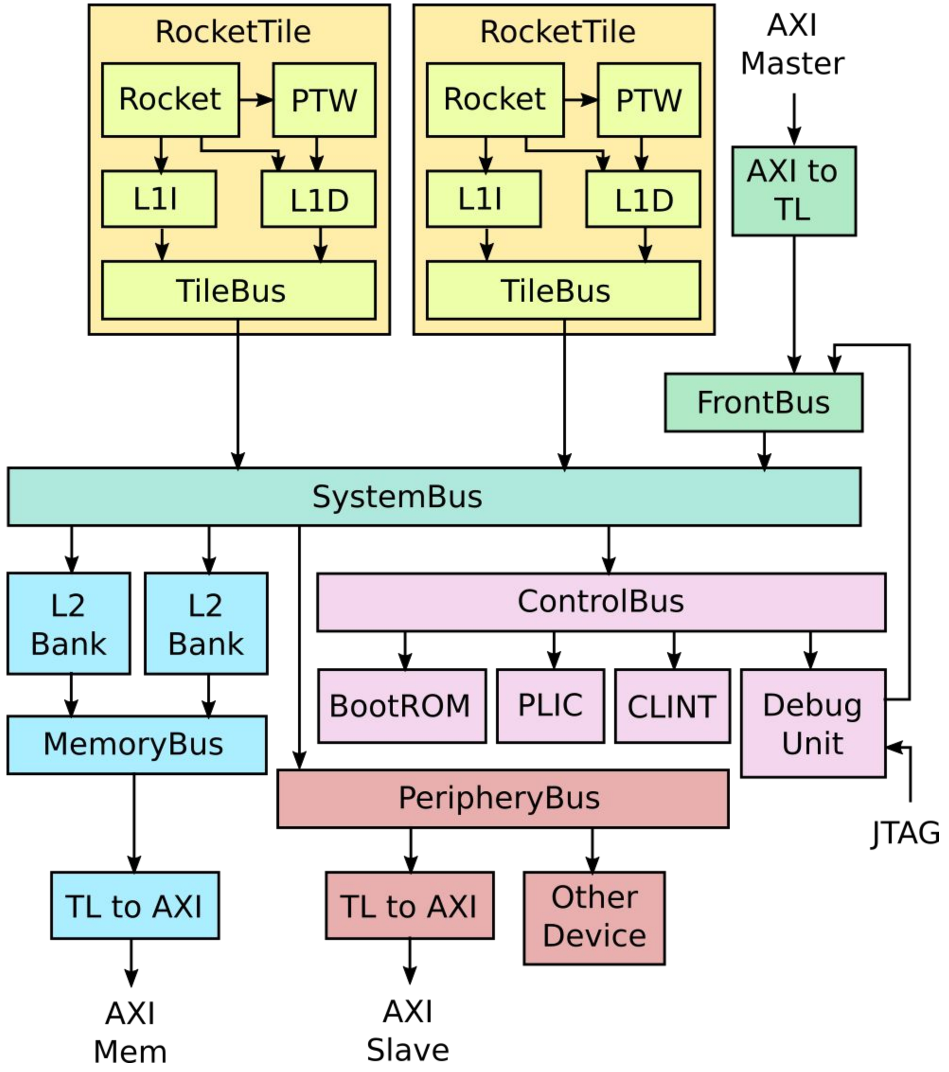
Chipyard: a framework for designing and developing a custom SoC



1. INTRODUCTION

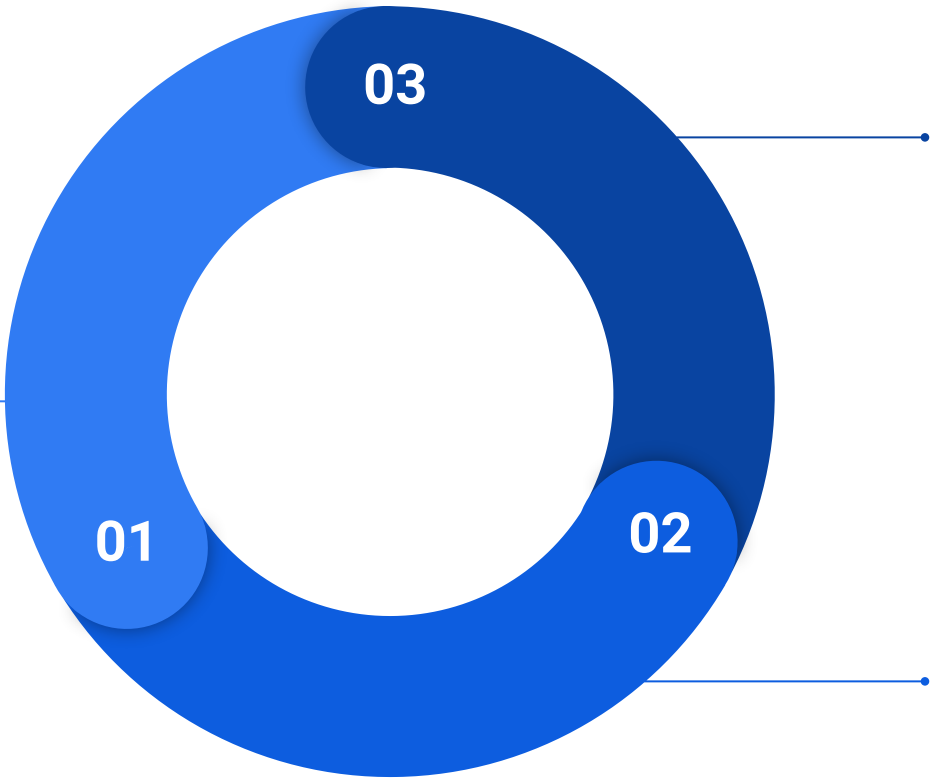
- **Verilator**: simulation of RISC V programs

- **RocketChip**: a SoC generator
  - RocketCore
  - BOOM



2. METHODOLOGIES

2. METHODOLOGY



First simulation*

We produced a *Makefile* to compile and produce an executable - which would them be used to simulate our project

Function isolation

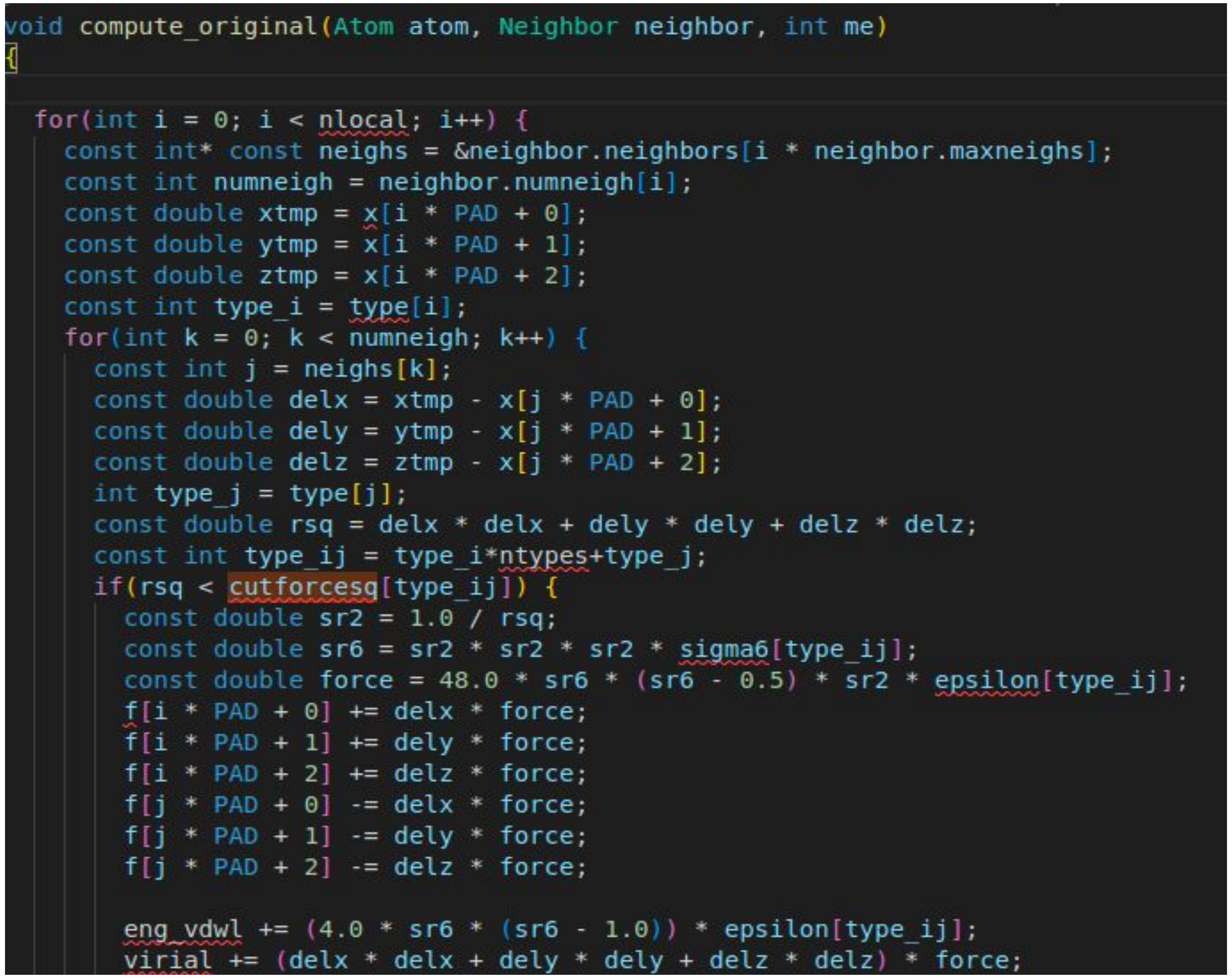
We extrapolated the core function of *miniMD* in order to analyze its main computations

Custom project set up

We created the proper environment around the function we previously extrapolated (import data structures, header files etc.)

*Verilator's limitations!

2. METHODOLOGY



```
void compute_original(Atom atom, Neighbor neighbor, int me)
{

  for(int i = 0; i < nlocal; i++) {
    const int* const neighs = &neighbor.neighbors[i * neighbor.maxneighs];
    const int numneigh = neighbor.numneigh[i];
    const double xtmp = x[i * PAD + 0];
    const double ytmp = x[i * PAD + 1];
    const double ztmp = x[i * PAD + 2];
    const int type_i = type[i];
    for(int k = 0; k < numneigh; k++) {
      const int j = neighs[k];
      const double delx = xtmp - x[j * PAD + 0];
      const double dely = ytmp - x[j * PAD + 1];
      const double delz = ztmp - x[j * PAD + 2];
      int type_j = type[j];
      const double rsq = delx * delx + dely * dely + delz * delz;
      const int type_ij = type_i*ntypes+type_j;
      if(rsq < cutforcesq[type_ij]) {
        const double sr2 = 1.0 / rsq;
        const double sr6 = sr2 * sr2 * sr2 * sigma6[type_ij];
        const double force = 48.0 * sr6 * (sr6 - 0.5) * sr2 * epsilon[type_ij];
        f[i * PAD + 0] += delx * force;
        f[i * PAD + 1] += dely * force;
        f[i * PAD + 2] += delz * force;
        f[j * PAD + 0] -= delx * force;
        f[j * PAD + 1] -= dely * force;
        f[j * PAD + 2] -= delz * force;

        eng_vdwl += (4.0 * sr6 * (sr6 - 1.0)) * epsilon[type_ij];
        virial += (delx * delx + dely * dely + delz * delz) * force;
```

3. EXPERIMENT CONFIGURATIONS

3. EXPERIMENT CONFIGURATIONS

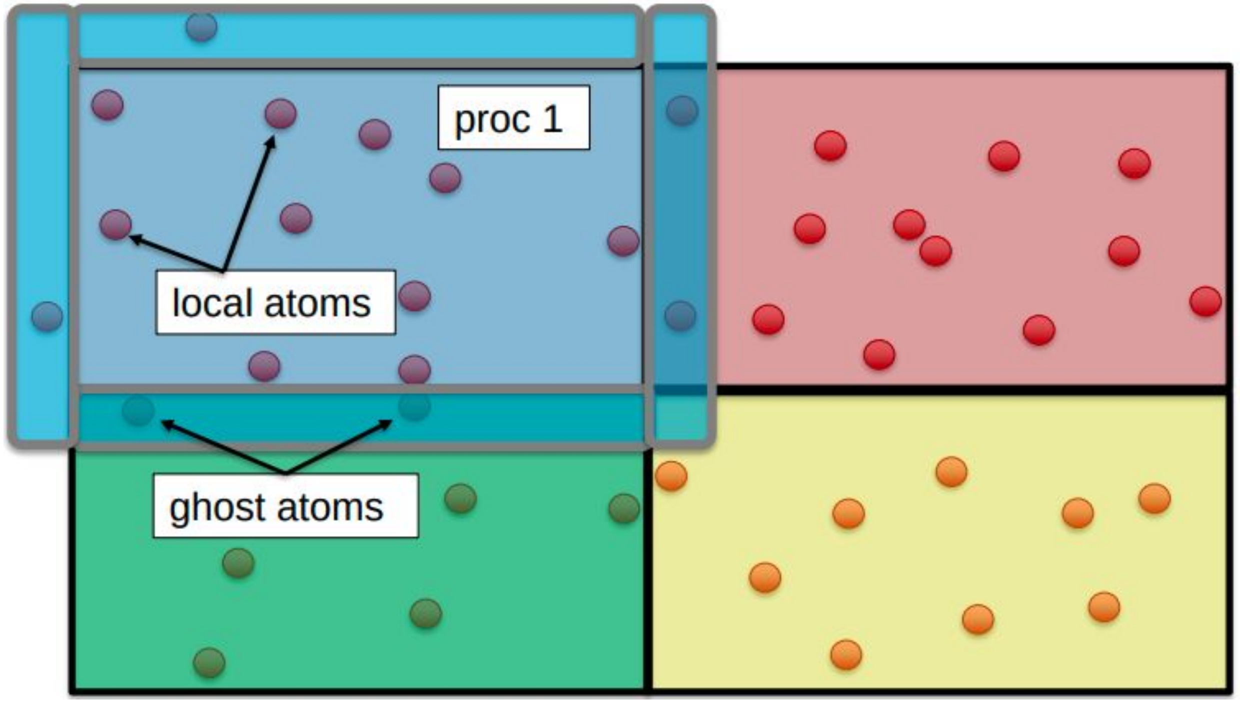


Figure 5: Visual representation of *local atoms* and *ghost atoms:* each square represents a different processor

Number of atoms	200
Number of local atoms	200
Number of ghosts	80
Types of atoms	4
Max number of atoms	364
Max number of neigh	20
Epsilon	1.0
Sigma	1.0
Cutforce	2.5
Virial	1.4822e-323

3. EXPERIMENT CONFIGURATIONS

ROCKET CORE

We first took into account this In-order core generator: we analyzed the performance and decided to move towards a different architecture.

BOOM

Berkeley out of order machine: an highly parameterizable RISC V core generator.
It offers several configurations, as:

- WithNSmallBooms
- WithNMediumBooms
- WithNLargeBooms
- WithNMegaBooms
- WithNGigaBooms

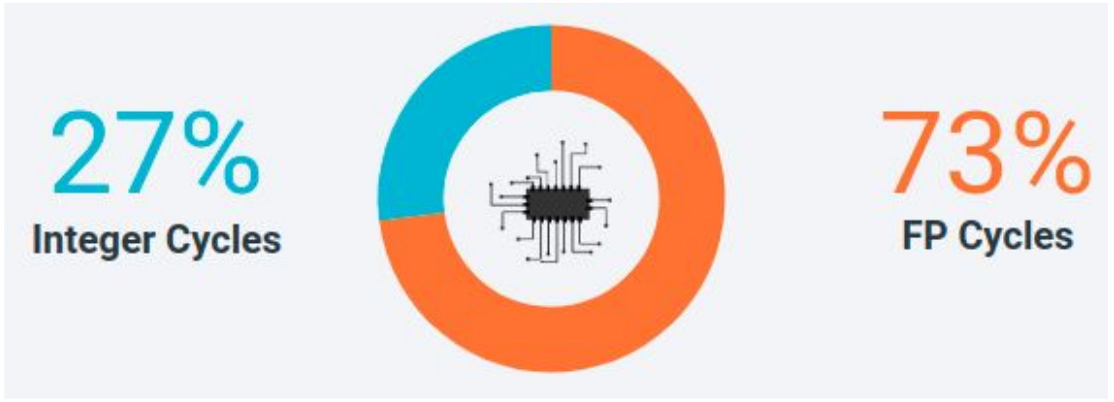
CUSTOM BOOM

After analyzing the core parameters of each configurations, we simulated and benchmarked different custom configs, in order to reduce the total amount of cc needed.

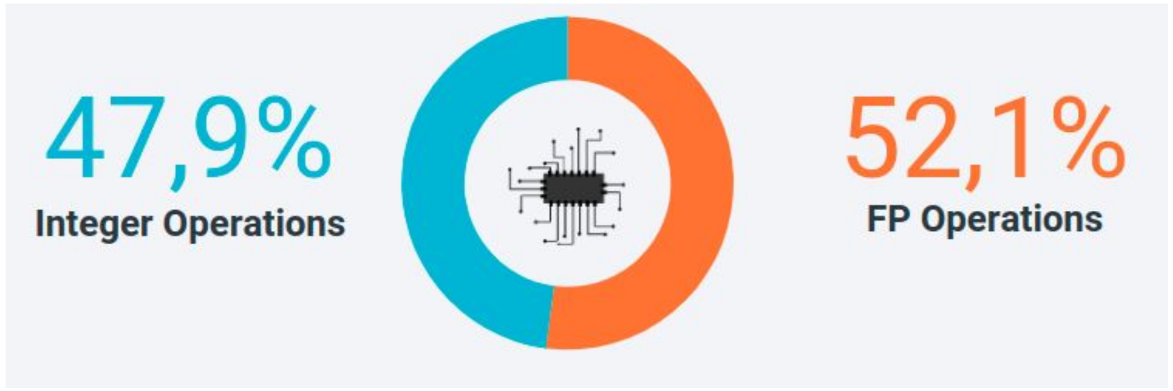
3. EXPERIMENT CONFIGURATIONS

ROCKET CORE: INSTRUCTION MIX

Clock cycles per instruction type (main loop)



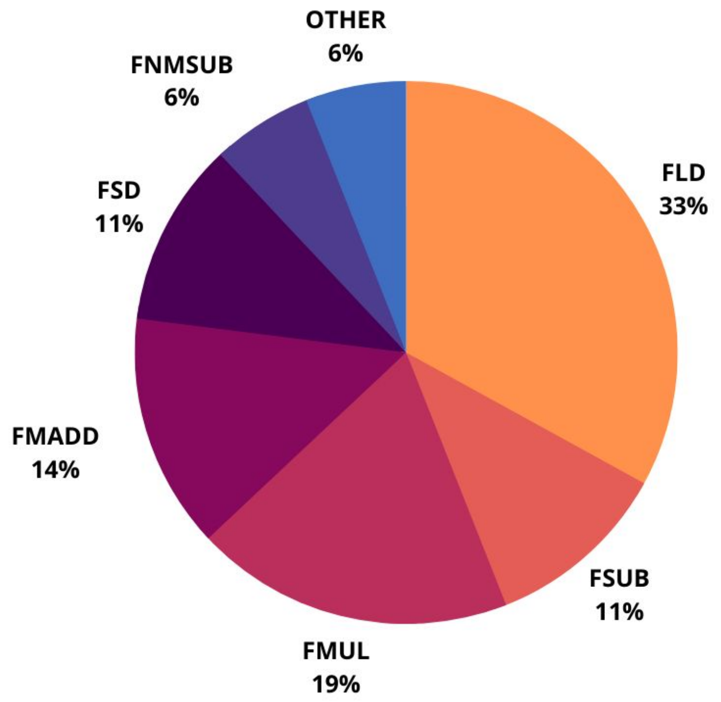
Percentage of integer and FP operations (main loop)



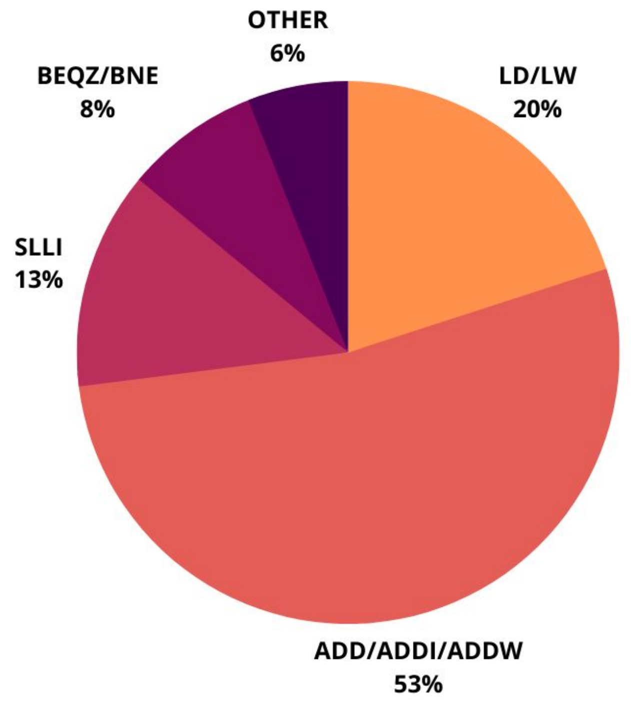
3. EXPERIMENT CONFIGURATIONS

ROCKET CORE: INSTRUCTIONS' FREQUENCY

Frequency of **FP** instructions



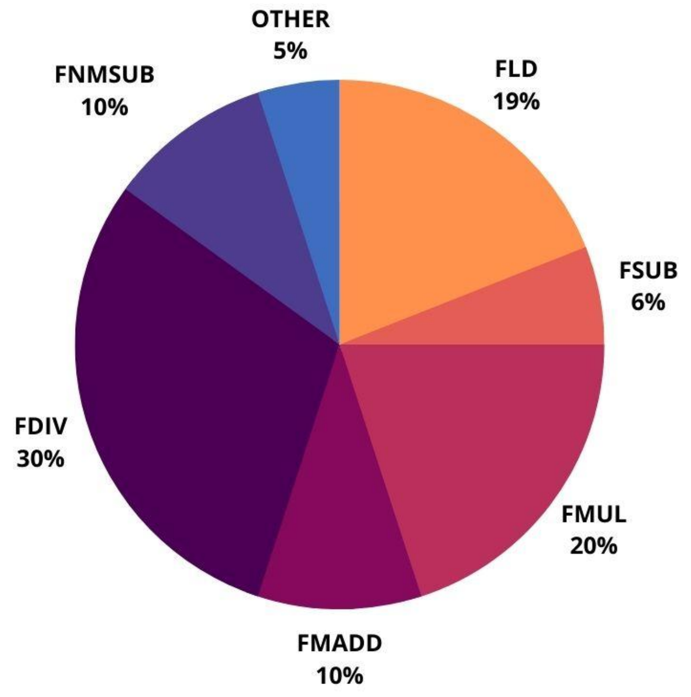
Frequency of **Integer** instructions



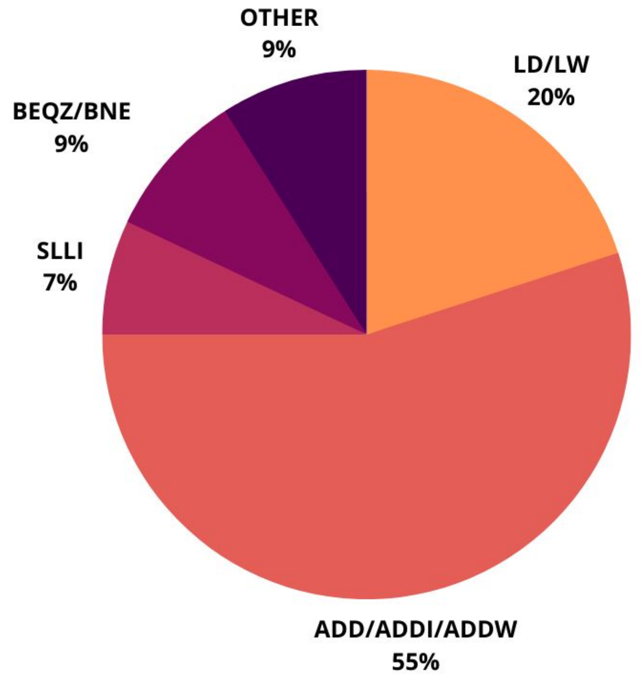
3. EXPERIMENT CONFIGURATIONS

ROCKET CORE: CLOCK CYCLES

Clock cycles per **FP** operation

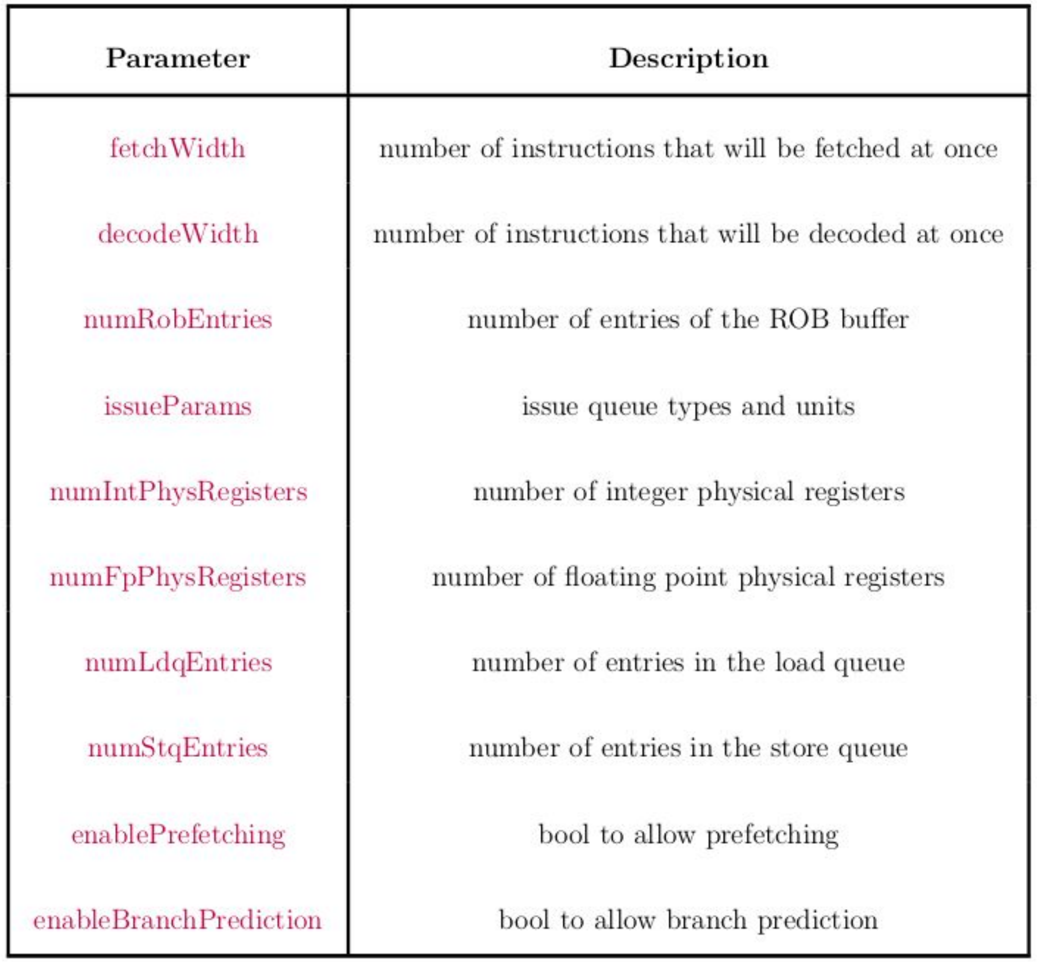


Clock cycles per **Integer** operation



3. EXPERIMENT CONFIGURATIONS

BOOM: PARAMETERS

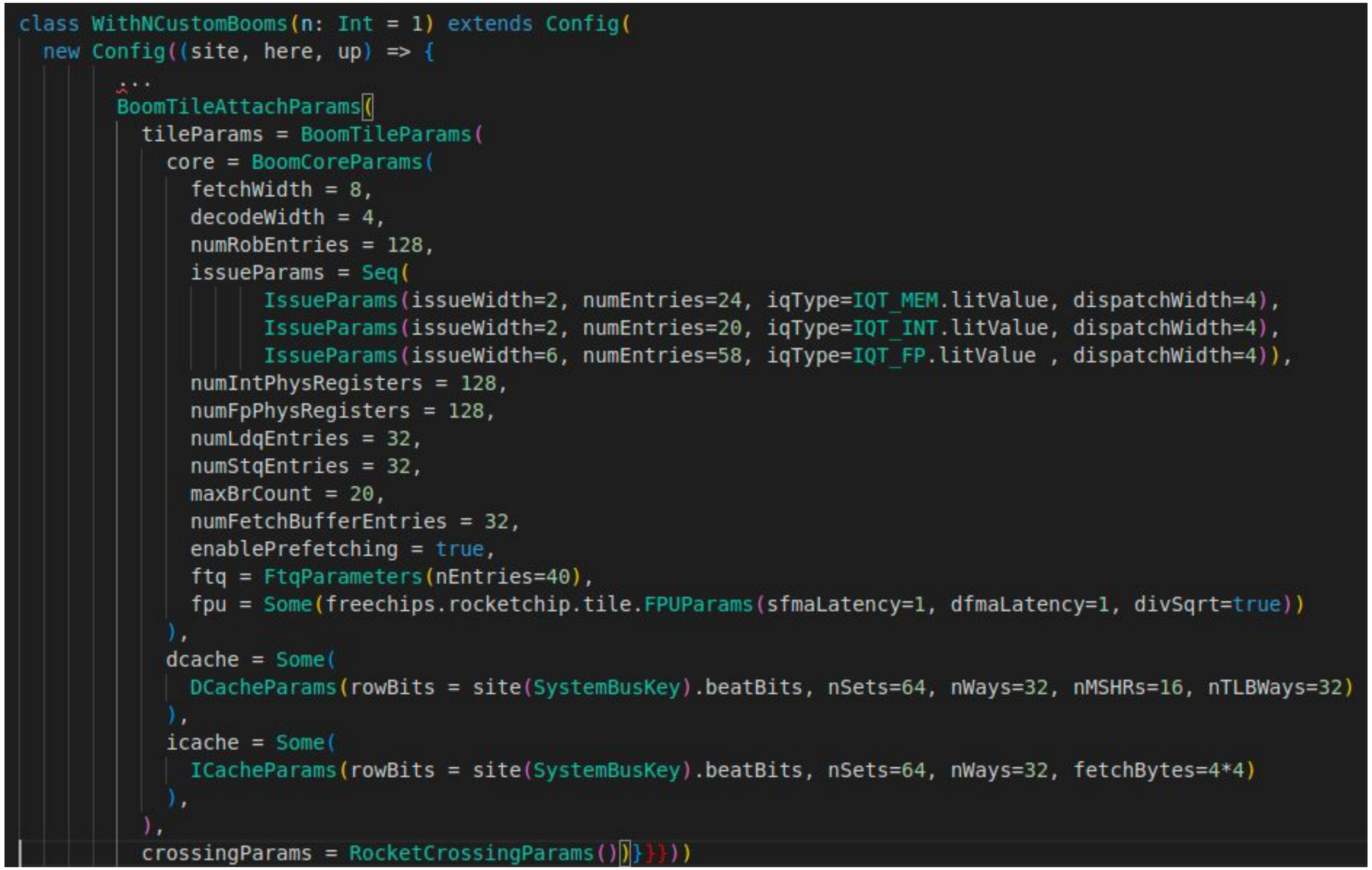


Parameter	Description
fetchWidth	number of instructions that will be fetched at once
decodeWidth	number of instructions that will be decoded at once
numRobEntries	number of entries of the ROB buffer
issueParams	issue queue types and units
numIntPhysRegisters	number of integer physical registers
numFpPhysRegisters	number of floating point physical registers
numLdqEntries	number of entries in the load queue
numStqEntries	number of entries in the store queue
enablePrefetching	bool to allow prefetching
enableBranchPrediction	bool to allow branch prediction

3. EXPERIMENT CONFIGURATIONS

CUSTOM BOOM

Final configuration



```
class WithNCustomBooms(n: Int = 1) extends Config(
  new Config((site, here, up) => {
        ...
        BoomTileAttachParams(
          tileParams = BoomTileParams(
            core = BoomCoreParams(
              fetchWidth = 8,
              decodeWidth = 4,
              numRobEntries = 128,
              issueParams = Seq(
                    IssueParams(issueWidth=2, numEntries=24, iqType=IQT_MEM.litValue, dispatchWidth=4),
                    IssueParams(issueWidth=2, numEntries=20, iqType=IQT_INT.litValue, dispatchWidth=4),
                    IssueParams(issueWidth=6, numEntries=58, iqType=IQT_FP.litValue , dispatchWidth=4)),
              numIntPhysRegisters = 128,
              numFpPhysRegisters = 128,
              numLdqEntries = 32,
              numStqEntries = 32,
              maxBrCount = 20,
              numFetchBufferEntries = 32,
              enablePrefetching = true,
              ftq = FtqParameters(nEntries=40),
              fpu = Some(freechips.rocketchip.tile.FPUParams(sfmaLatency=1, dfmaLatency=1, divSqrt=true))
            ),
            dcache = Some(
              DCacheParams(rowBits = site(SystemBusKey).beatBits, nSets=64, nWays=32, nMSHRs=16, nTLBWays=32)
            ),
            icache = Some(
              ICacheParams(rowBits = site(SystemBusKey).beatBits, nSets=64, nWays=32, fetchBytes=4*4)
            ),
          ),
          crossingParams = RocketCrossingParams())}}}))
```

4. EXPERIMENT EVALUATIONS

4. EXPERIMENT EVALUTATIONS

CLOCK CYCLES (WHOLE PROGRAM)

Architecture	Clock cycles (whole program)
RocketConfig	296'685
SmallBoomConfig	264'561
MediumBoomConfig	245'012
LargeBoomConfig	221'576
MegaBoomConfig	212'300
CustomBoomConfig	210'414

CLOCK CYCLES (MAIN LOOP)

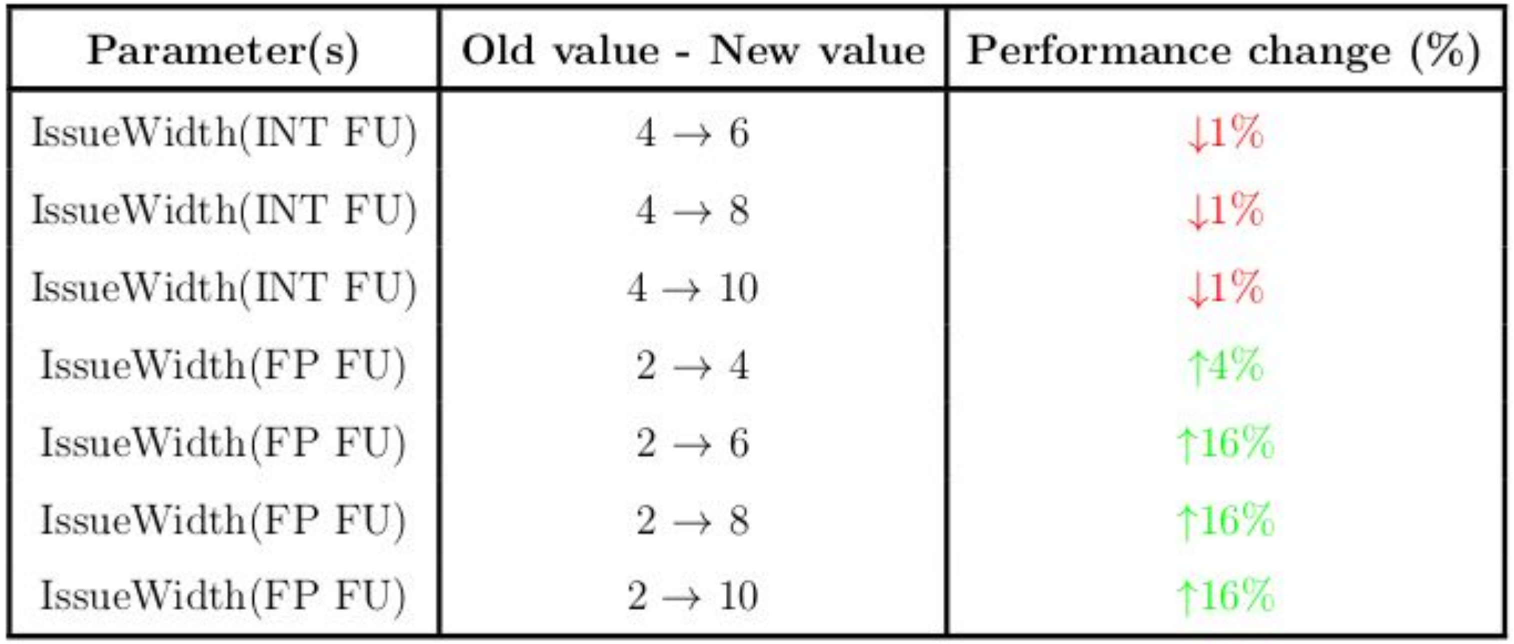
Architecture	Clock cycles (main loop)
RocketConfig	36'294
SmallBoomConfig	26'788
MediumBoomConfig	22'330
LargeBoomConfig	18'378
MegaBoomConfig	13'581
CustomBoomConfig	9'256

4. EXPERIMENT EVALUTATIONS

ISSUE WIDTHS

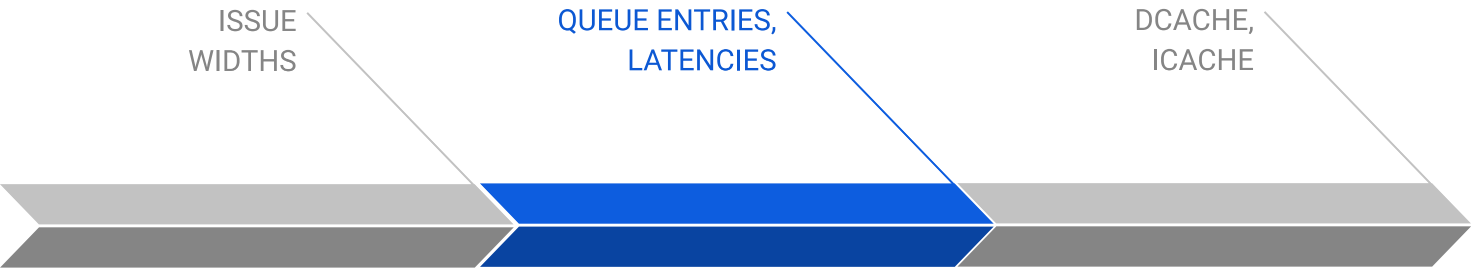
QUEUE ENTRIES, LATENCIES

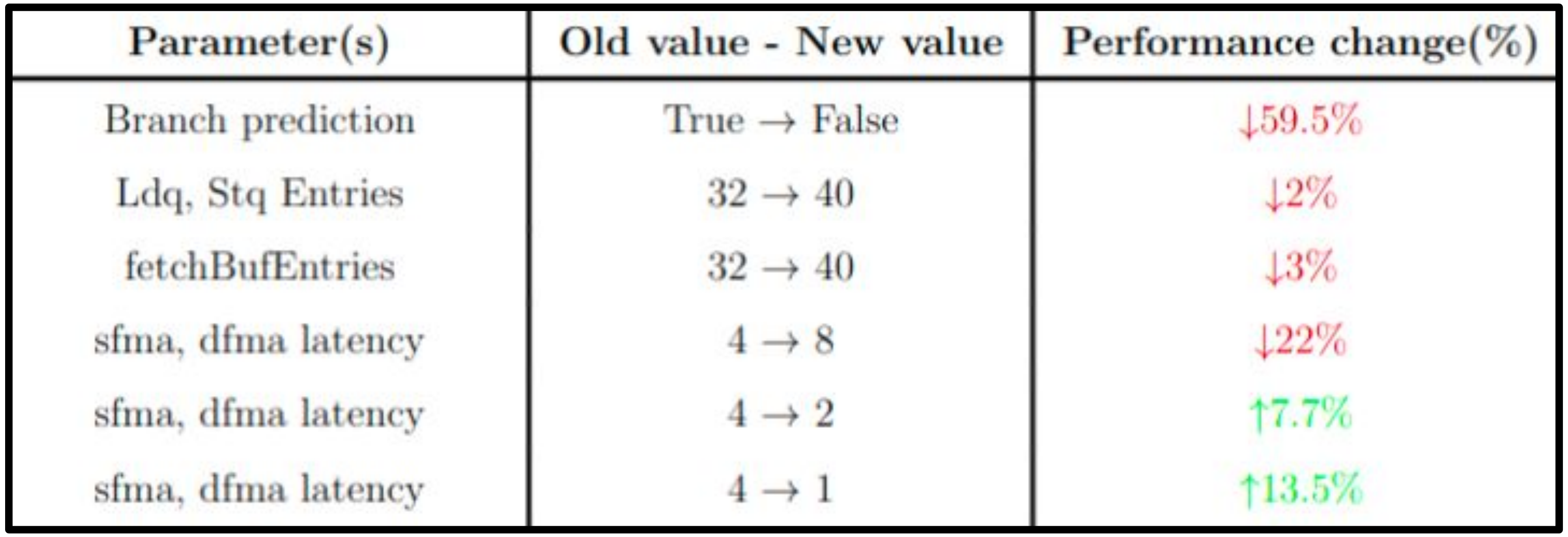
DCACHE, ICACHE



Parameter(s)	Old value - New value	Performance change (%)
IssueWidth(INT FU)	$4 \rightarrow 6$	↓1%
IssueWidth(INT FU)	$4 \rightarrow 8$	↓1%
IssueWidth(INT FU)	$4 \rightarrow 10$	↓1%
IssueWidth(FP FU)	$2 \rightarrow 4$	↑4%
IssueWidth(FP FU)	$2 \rightarrow 6$	↑16%
IssueWidth(FP FU)	$2 \rightarrow 8$	↑16%
IssueWidth(FP FU)	$2 \rightarrow 10$	↑16%

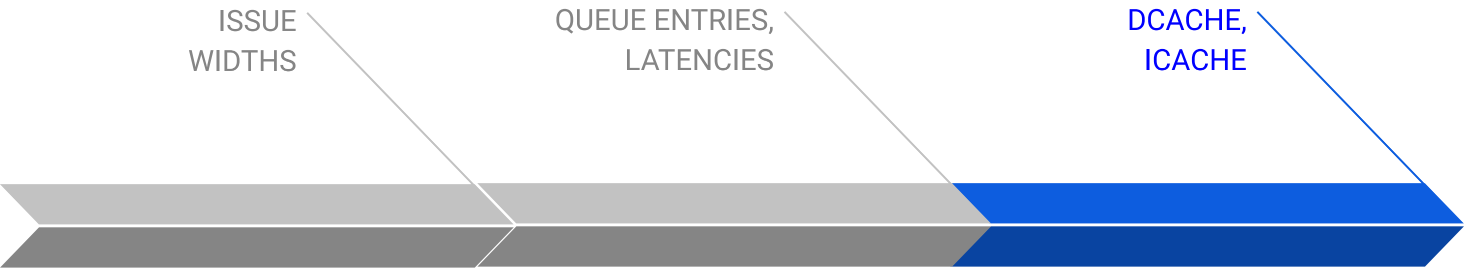
4. EXPERIMENT EVALUTATIONS

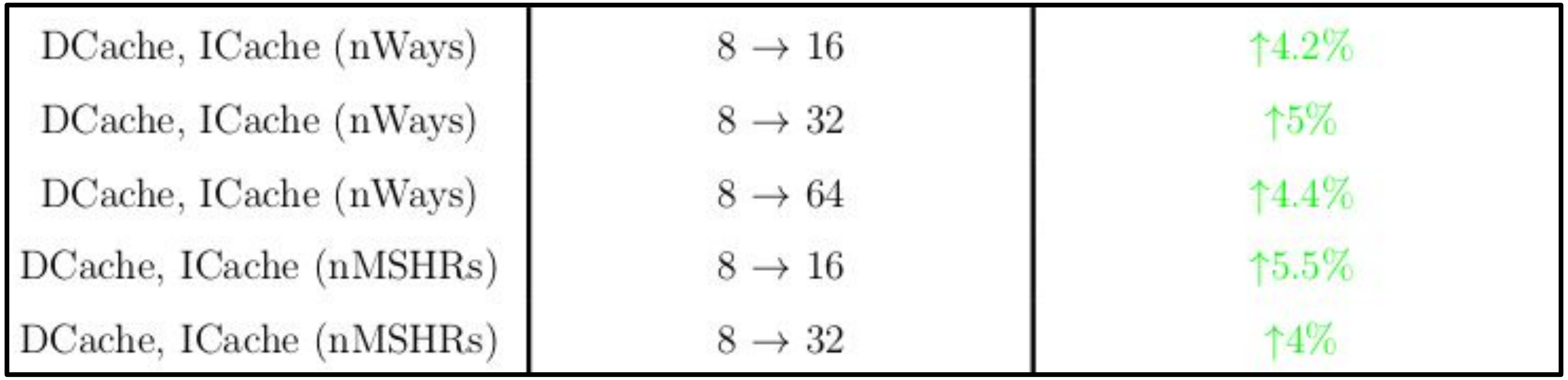




Parameter(s)	Old value - New value	Performance change(%)
Branch prediction	True $\rightarrow$ False	↓59.5%
Ldq, Stq Entries	32 $\rightarrow$ 40	↓2%
fetchBufEntries	32 $\rightarrow$ 40	↓3%
sfma, dfma latency	4 $\rightarrow$ 8	↓22%
sfma, dfma latency	4 $\rightarrow$ 2	↑7.7%
sfma, dfma latency	4 $\rightarrow$ 1	↑13.5%

4. EXPERIMENT EVALUTATIONS





DCache, ICache (nWays)	$8 \rightarrow 16$	↑4.2%
DCache, ICache (nWays)	$8 \rightarrow 32$	↑5%
DCache, ICache (nWays)	$8 \rightarrow 64$	↑4.4%
DCache, ICache (nMSHRs)	$8 \rightarrow 16$	↑5.5%
DCache, ICache (nMSHRs)	$8 \rightarrow 32$	↑4%

5. CONCLUSIONS

5. CONCLUSIONS

PROJECT'S TAKEAWAY

Main idea: test, simulate and benchmark the CORE computations of a crucial function through different architectures, in order to speed up the whole simulation.

FUTURE DEVELOPMENTS

Design a specialized FU, able to perform such computations in a custom, specialized and optimized way: this should be a specialized HW component.

THANK YOU!

Project's repo: github.com/emilioingenito/MDCoreSpecialization