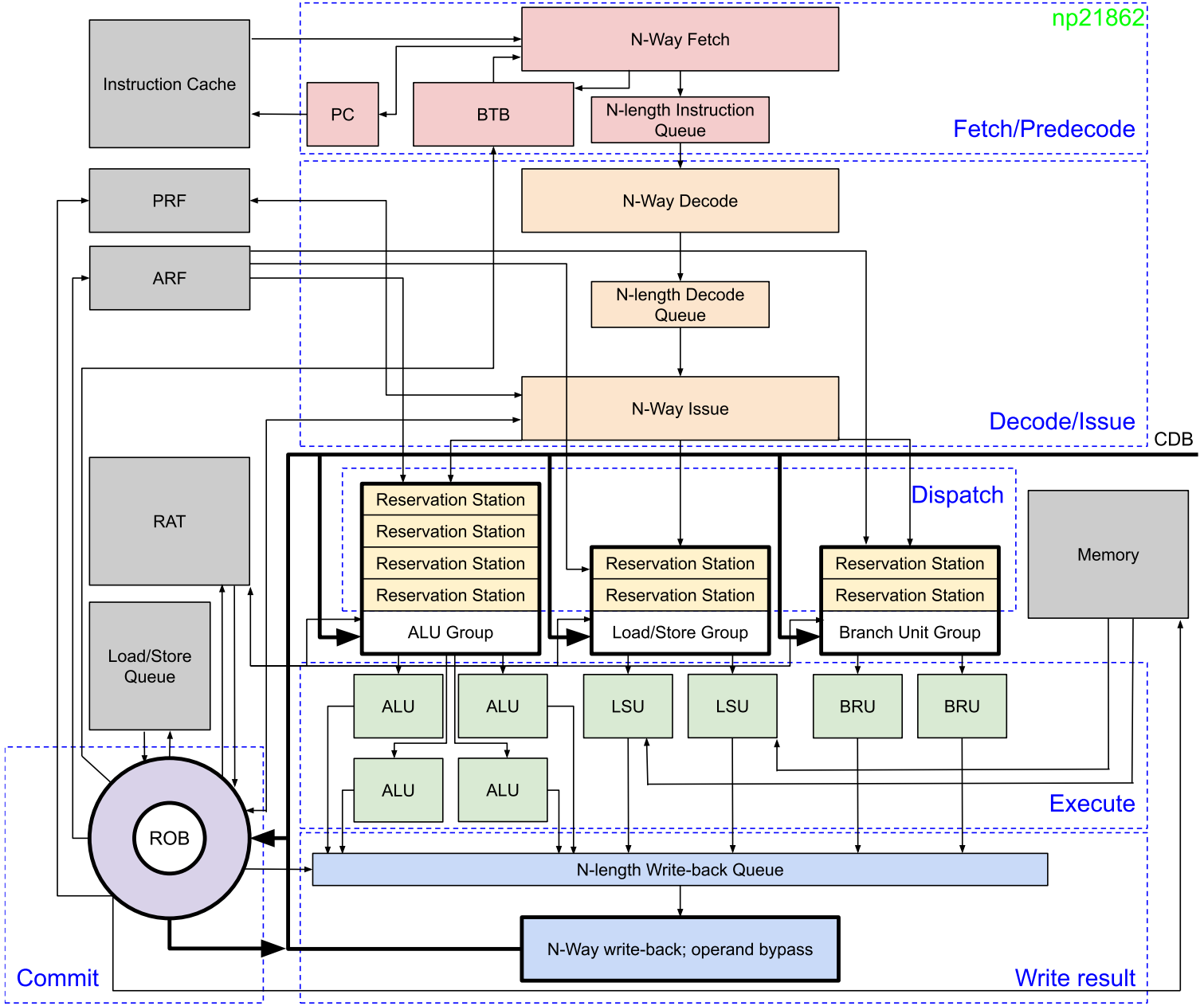
np21862@bristol.ac.uk

Architecture Diagram

Shown: an example configuration of my simulator with four ALU reservation stations, four ALUs, two load/store reservation stations, two load store units, two branch unit reservation stations, and two branch units.



University of BRISTOL

np21862@bristol.ac.uk

Features

Pipeline:

- Six-stage pipeline (fetch/predecode, decode/issue, dispatch, execute, write-result, commit)
- Out of Order execution, re-order buffer, register alias table
- Grouped reservation stations grouped by execution unit types
- Tomasulos algorithm and register renaming with the ROB
- Execution unit operand forwarding, OoO memory reads
- Aligned/unaligned instruction fetch
- Nested speculative execution with RAT history
- Dynamic branch prediction using a BTB/BTAC (1 or 2 bit predictor)
- Multi-cycle instruction latency (configurable cycle count for each instruction)
- Indexed load and store instructions
- Fixed taken & not taken and static branch backward taken & not taken predictors
- Store-to-load forwarding or store-load conflict-based flushing
- <= 8 ALUs, <= 4 LSUs, <= 4 branch units
- Any number (>= 1) of reservation stations in each group

Registers and Memory (default settings):

- 256B working memory
- 32 architectural registers corresponding to those in RISC-V
- 64 physical registers
- 64 ROB entries
- 32 BTB entries
- Six different branch predictor modes

Superscalar:

- N-way fetch/decode/issue/writeback/commit per cycle
- Configurable number of ALUs, Load/Store Units, Branch Units, reservation stations

Flushing:

- All units can flush from a signaled ROB entry

Configurability & Usage:

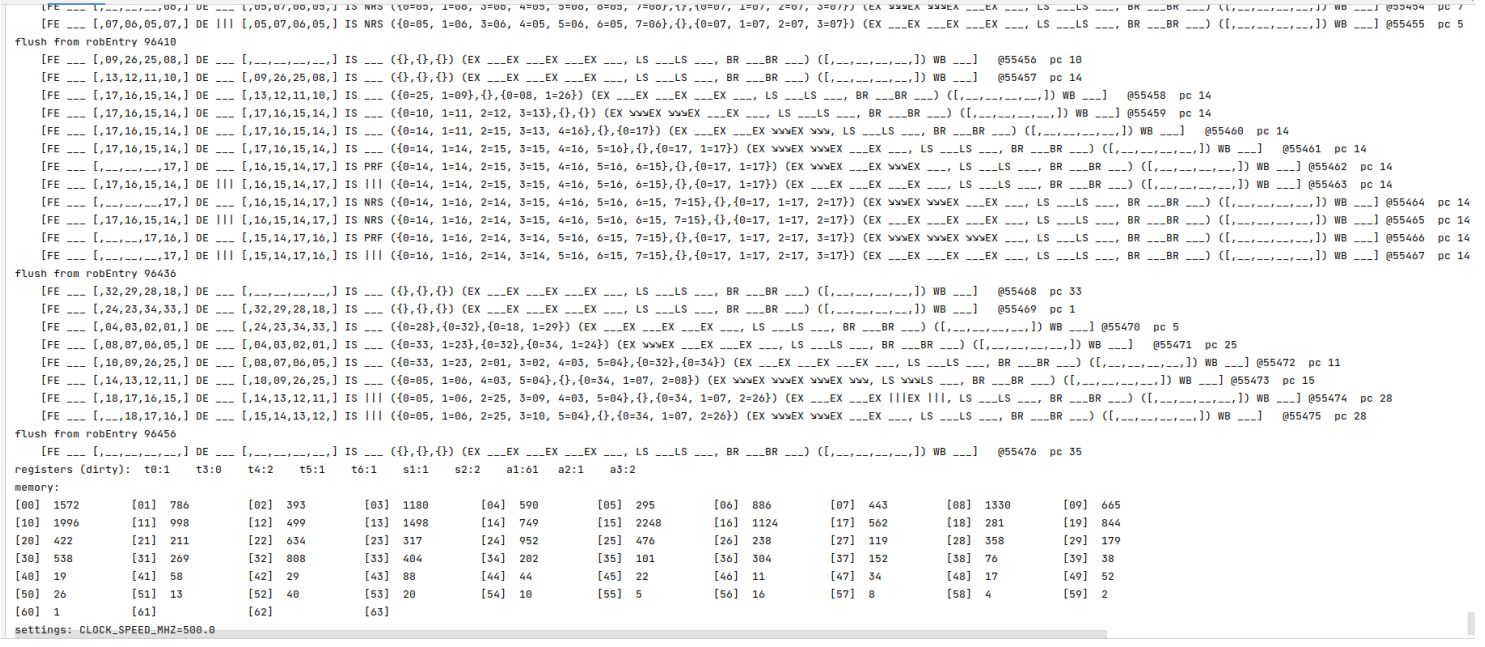
- Command-line arguments for superscalar width, predictor type, BTB size, ALU count, load/store unit count, branch unit count, ALU reservation station count, LSU reservation station count, branch unit reservation station count, ROB size, PRF size, and an aligned-fetch flag.
- Console pipeline view **(see next slide)**

np21862@bristol.ac.uk

Pipeline View and Benchmark Programs

Benchmark	Information (cycle count is for single-fetch OoO)
bubble_sort	Sort 40 element list of integers in-place in memory using bubble sort (~17k cycles)
collatz	Calculate the collatz sequence from 3144 of length 61 using mod and quot routines (~98k cycles)
fact	Computes factorial of 14
mat3_mul	Stores two 3x3 matrices, then calculates and stores their product
pi	Computes the first few digits of pi from a grid of points and counts those < distance 1 from the origin (~300k cycles)
tn	'Taken not-taken' does not take branch A and takes branch B (repeats 640 times)
ttn	'Taken taken not-taken' takes branch A, then does not, then takes branch B (repeats)
ttttn	'Taken x4 not-taken' takes branch A thrice, does not take it, then takes branch B (repeats)
unrolled_maths	A loop of mostly independent simple maths unrolled (~26k cycles)

Shown: console pipeline view of *collatz* finishing and printing register and memory content below. Run with aligned fetch, and four-way superscalar, and with a small PRF. Units are shown as a two-letter code and pipeline buffers sit between them. Instructions are shown in the pipeline by their id. Cycles is shown at the end of each like proceeding @. Reservation stations are shown as {} maps and in groups before the EU groups.



University of BRISTOL

np21862@bristol.ac.uk

Superscalar width and EU Count for IPC

Hypothesis: Up to twice the Instructions Per Cycle of some benchmark programs as we go from one to eight-way superscalar. We expect a plateau as true data dependencies begin to limit instruction parallelism.

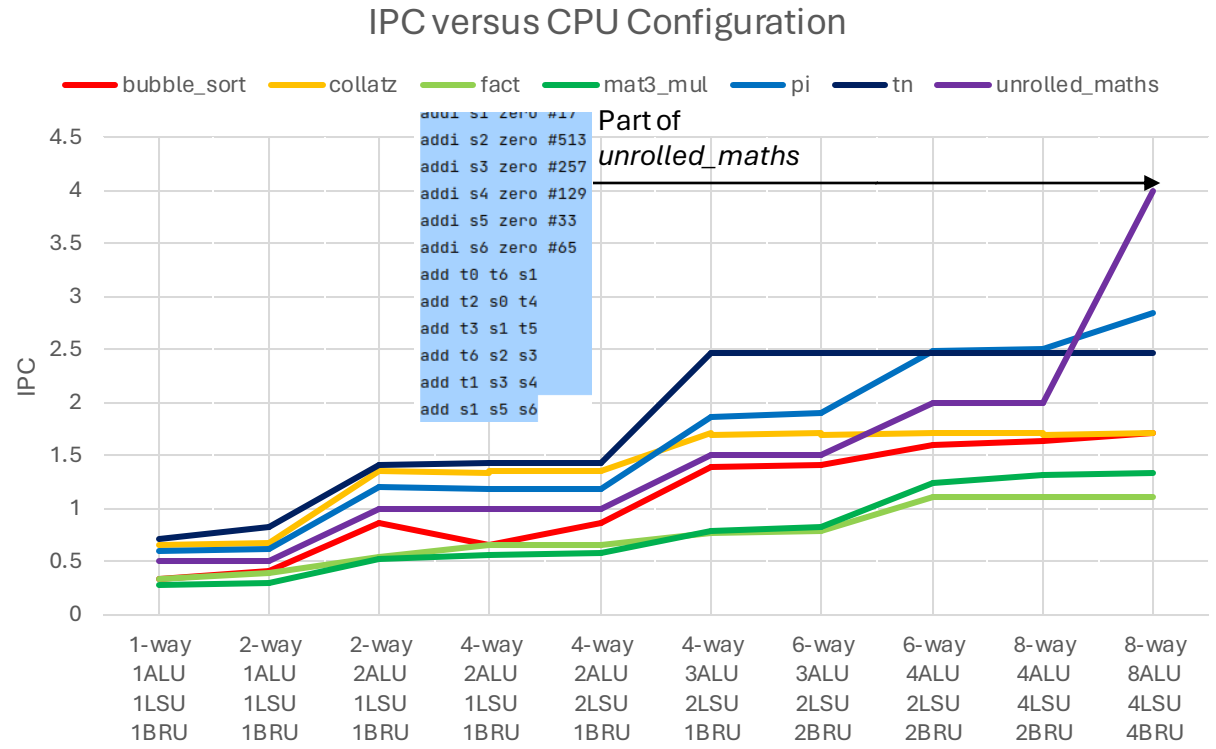
Experiment: All tests run with 64 rob entries, 8, 4, 4 reservation stations for ALUs, LSUs and branch units respectively. Performance metric of programs expressed in terms of Instructions Per Cycle, which implies programs will execute in a shorter time.

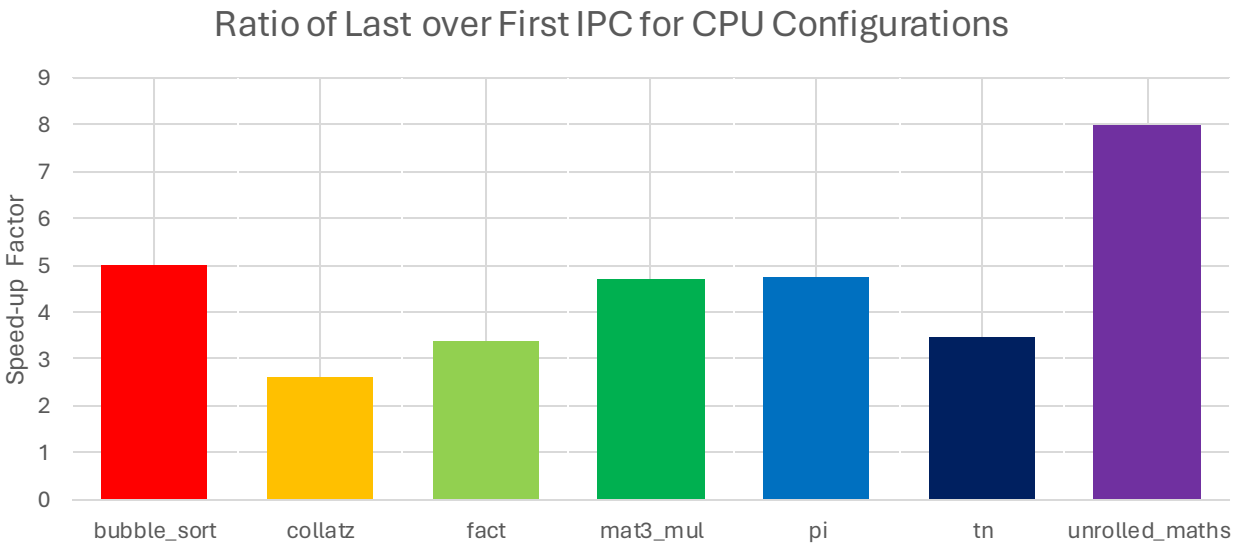
The bottom graphs shows the factor of speed-up we see from the first to the last measurement.

Result: We can see at least 4.5x IPC increase for *bubble_sort, mat3_mul, pi,* and *unrolled_maths* compared with a one-way scalar pipeline (as high as ~8x speed-up for *unrolled_maths*). The largest performance increases arise when increasing the number of ALUs in the processor, from 1 to 2 ALUs, from 2 to 4 and 4 to 8. After 4 ALUs and 6-way superscalar, for all but two programs' IPC plateaus likely due to limited instruction-level parallelism, implying data dependencies.

unrolled_maths and *pi* increased in IPC approx. proportionally to the number of ALUs up to 4. They contain almost no memory operations (and *unrolled_maths* contains no branches). *pi* cant keep up with *unrolled_maths* for 8 ALUs due to true data dependencies.

Bubble sort exhibits performance dips when it is allowed to speculate incorrectly for longer. Overall we accept the hypothesis.





University of BRISTOL

np21862@bristol.ac.uk

Reservation Stations versus Instruction Misprediction

Hypotheses: 1) There will be at least three times the mispredicted instruction percentage for our benchmarks between lowest and higher # reservation stations, 2) programs will all see increased IPC regardless.

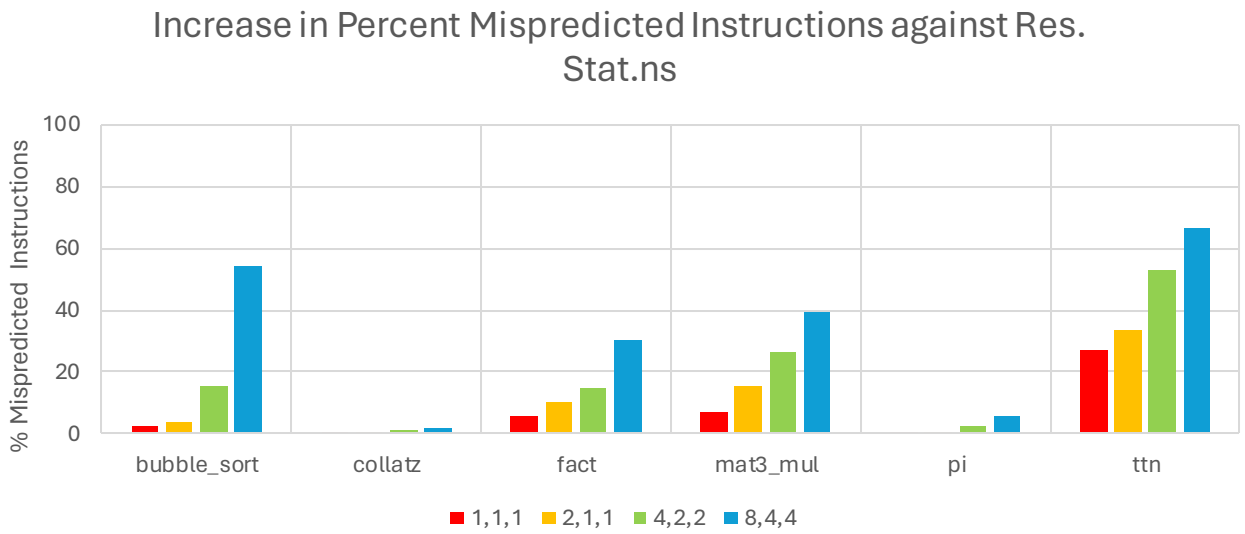
Experiment: Percentage of mispredicted instructions is calculated as:

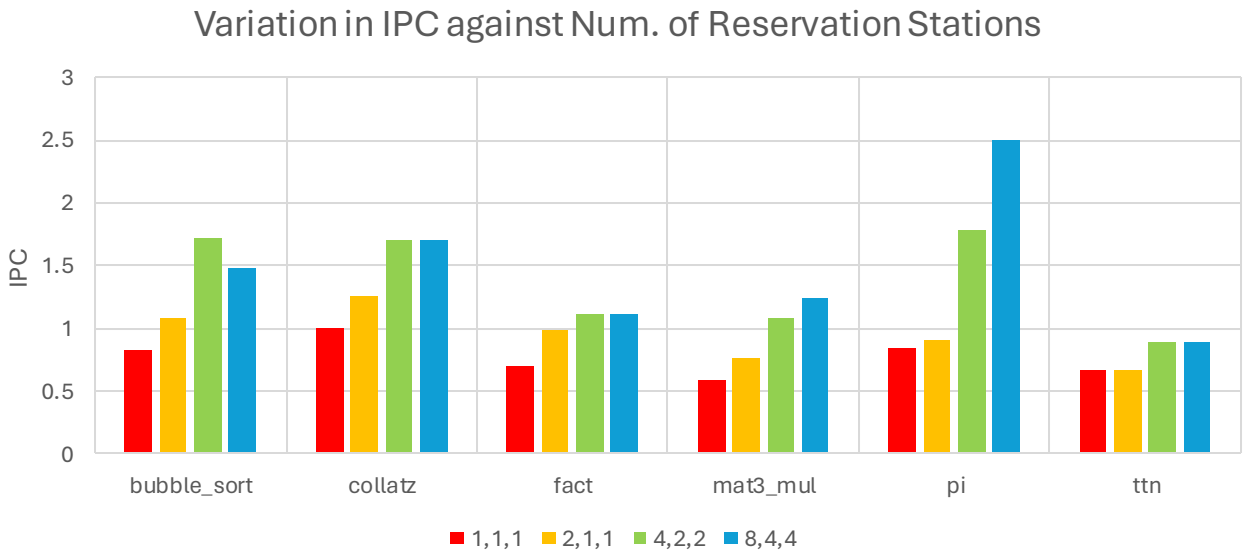
$$\frac{\#\ \text{Instructions Flushed from the ROB}}{\#\ \text{Committed Instructions} + \#\ \text{Instructions Flushed from the ROB}}$$

Test run with 64 rob entries, 4 ALUs, 4 LSUs, 2 branch units, 8-way superscalar, two-bit predictor, four times PRF as ARF. *Graphs display 3 numbers corresponding to the number of reservation stations in each group* ***(ALU, LSU, BRU)*****.**

Result: We see a monotonic increase in program IPC for all programs when we increase the number of reservation stations, except for *bubble_sort* and *collatz*. Confirming our hypothesis, we see more than three times the mispredicted instruction percentage for all programs except *ttn*. Overall we accept the hypotheses, noting the exceptions.

bubble_sort loses IPC when going to 8 ALU-reservation-stations due to the wasted time and resources processing the increased number of incorrectly speculated instructions. Note percent of mispredicted instructions seems to scale approximately linearly with our exponential increase in reservation stations, indicating they allow for more instruction-level parallelism.





University of BRISTOL

np21862@bristol.ac.uk

Branch Predictor Speed

Hypothesis: using more sophisticated branch predictors will show an increased IPC indicative of accurate speculation.

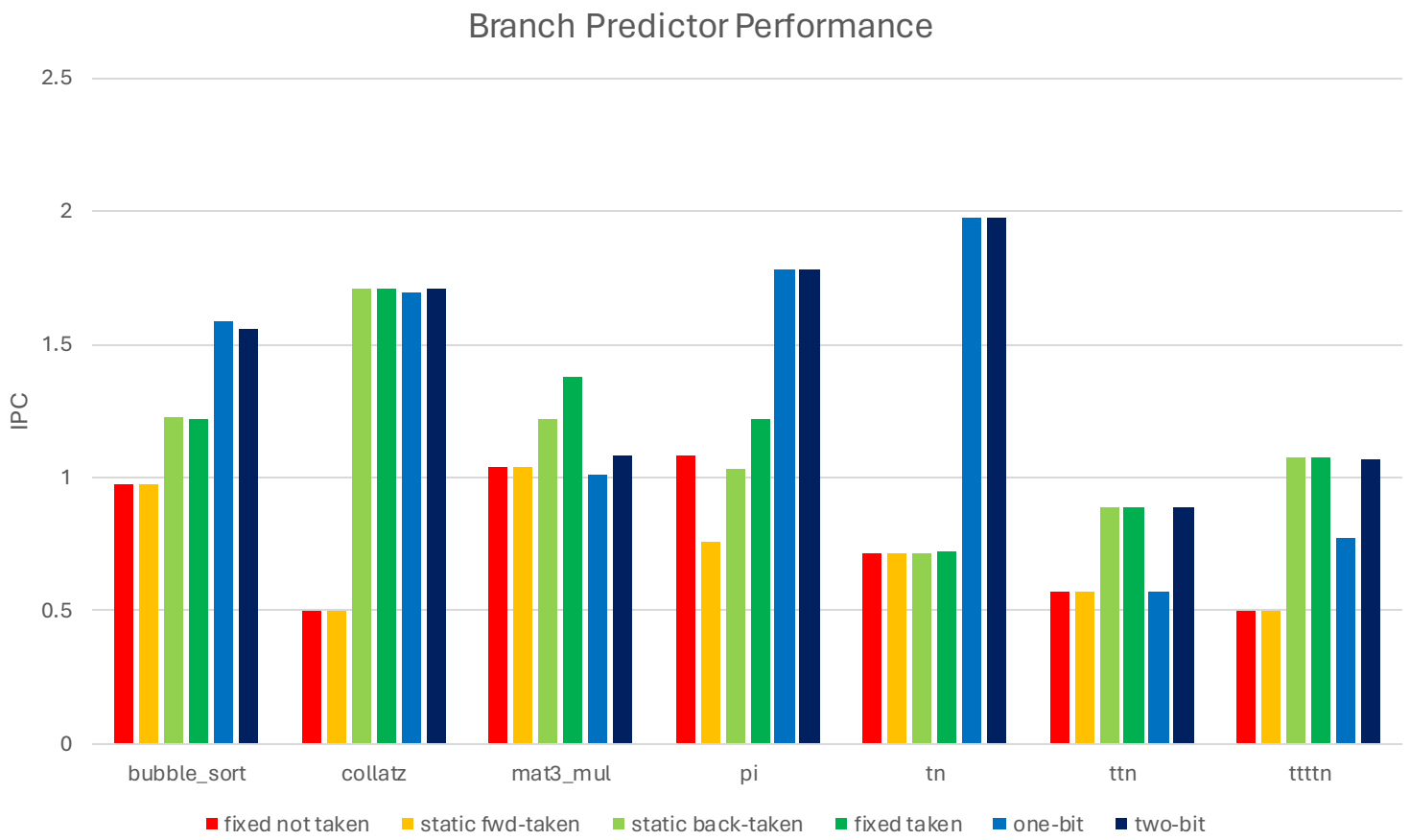
Experiment: Run 8-way superscalar with 4 ALUs, 4 LSUs, 2 Branch Units. 4,2,2 reservation stations for ALUs, LSUs and branch units resp. Showing the effect of fixed, static and one/two-predictors. The latter two use a BTB/BTAC to which also updates a state-machine for each recorded branch.

Result: Static forward-taken backward-not-taken performed the worst of all. This makes sense because I write assembly in the opposite way.

Fixed not-taken does poorly, but better than the previous in the case of *pi* because it will at least correctly predict forward branches generally aren't taken.

One and two-bit generally show the best IPC. Two-bit pulls ahead of one-bit for *ttn* and *ttttn* because it quickly produces a strongly-taken state in the retaken branch's BTB entry.

In all but one case the one- or the two-bit do no worse than the others on which we accept the hypothesis.



University of BRISTOL

np21862@bristol.ac.uk

The effect of PRF Size on IPC

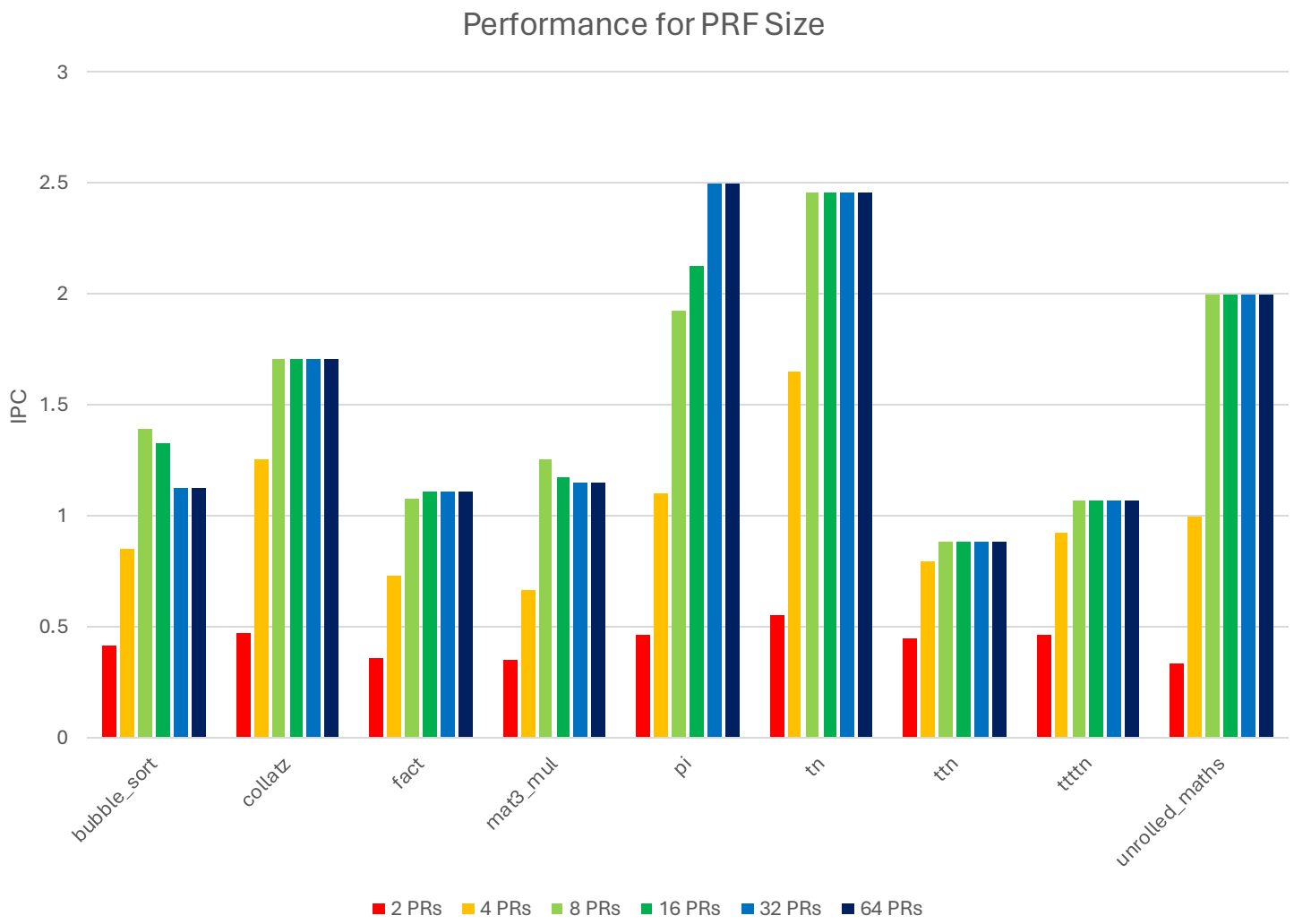
Hypothesis: Limiting the size of the physical register file will cause the issue stage to block often as renaming cannot occur until physical registers are available. This will create stalls and we'll see reduced IPC by at least two-times.

Experiment: The simulator was run 8-way superscalar and sufficient headroom for high ILP. PR count is increased exponentially to investigate whether any programs require a large PRF.

Result: All programs show reduced IPC at two and four physical registers. For two, only one instruction is allowed to flow through at a time as there must be at least two free physical registers to place a register into a reservation station.

All programs drastically increase in performance from 2 to 8 physical registers. Most increase and plateu at 16. Notable outliers include *bubble_sort* and *mat3_mul* where less blocking causes more incorrectly speculated instructions and lower IPC.

pi is able to make use of the extra 16 physical registers gained in the jump from 16 to 32. To get maximum ILP, in-flight instructions have more diverse destination registers than other benchmarks. The graph clearly supports our hypothesis.



University of BRISTOL