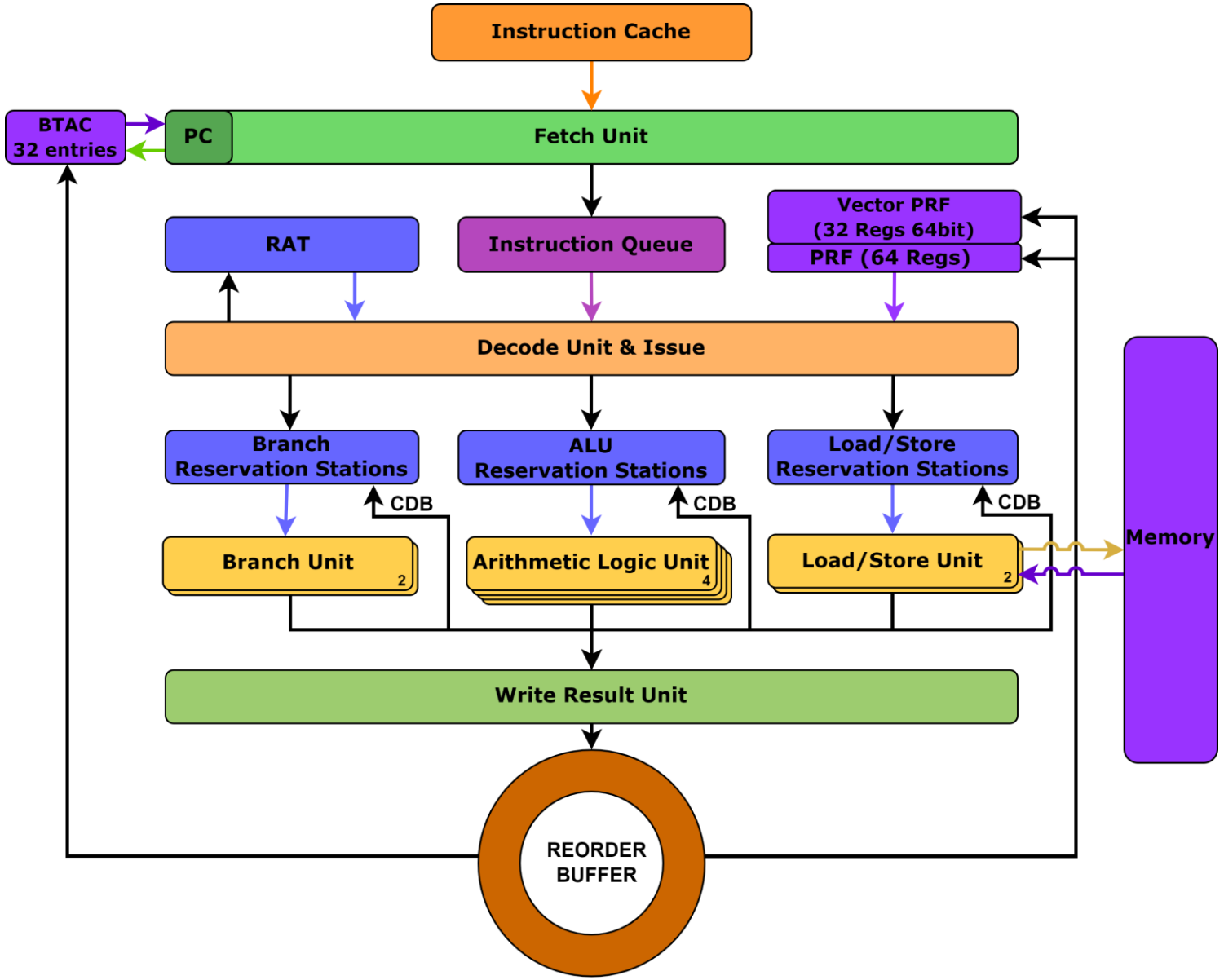
BENCHMARK PROGRAM	DESCRIPTION
Bubble Sort (BS)	Sorts 10 descending numbers into ascending order.
Factorial (Fact)	Finds 15! (15 factorial)
Fibonacci (Fib)	Finds the 40th Fibonacci number
Greatest Common Divisor (GCD)	Finds the greatest common divisor between 123215 and 235
Matrix Squaring SIMD (Mat Squaring SIMD)	Squaring a 3x3 matrix with vector instructions (no branches)
Matrix Squaring (Mat Squaring)	Squaring a 3x3 matrix with no vector instructions (no branches)

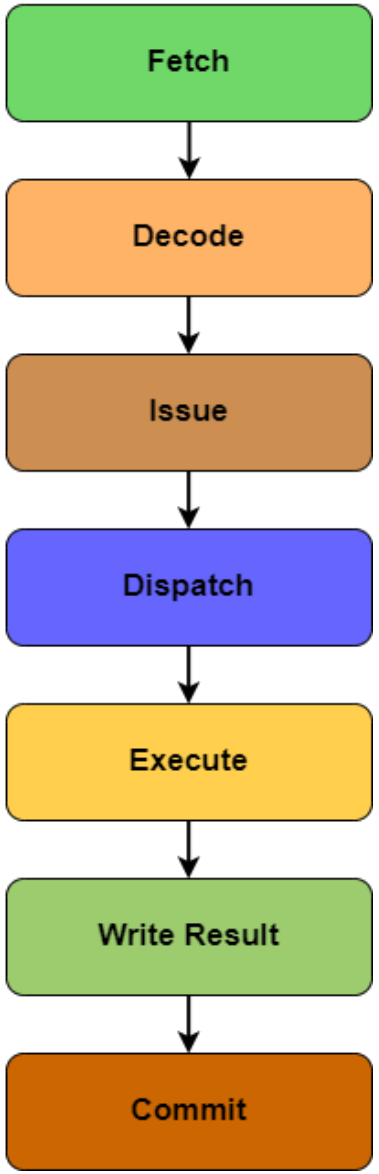
SUPER SCALAR PROCESSOR SIMULATOR

COMS30047
Leo Lai
um21226@bristol.ac.uk

Architecture Diagram



Pipeline Stages

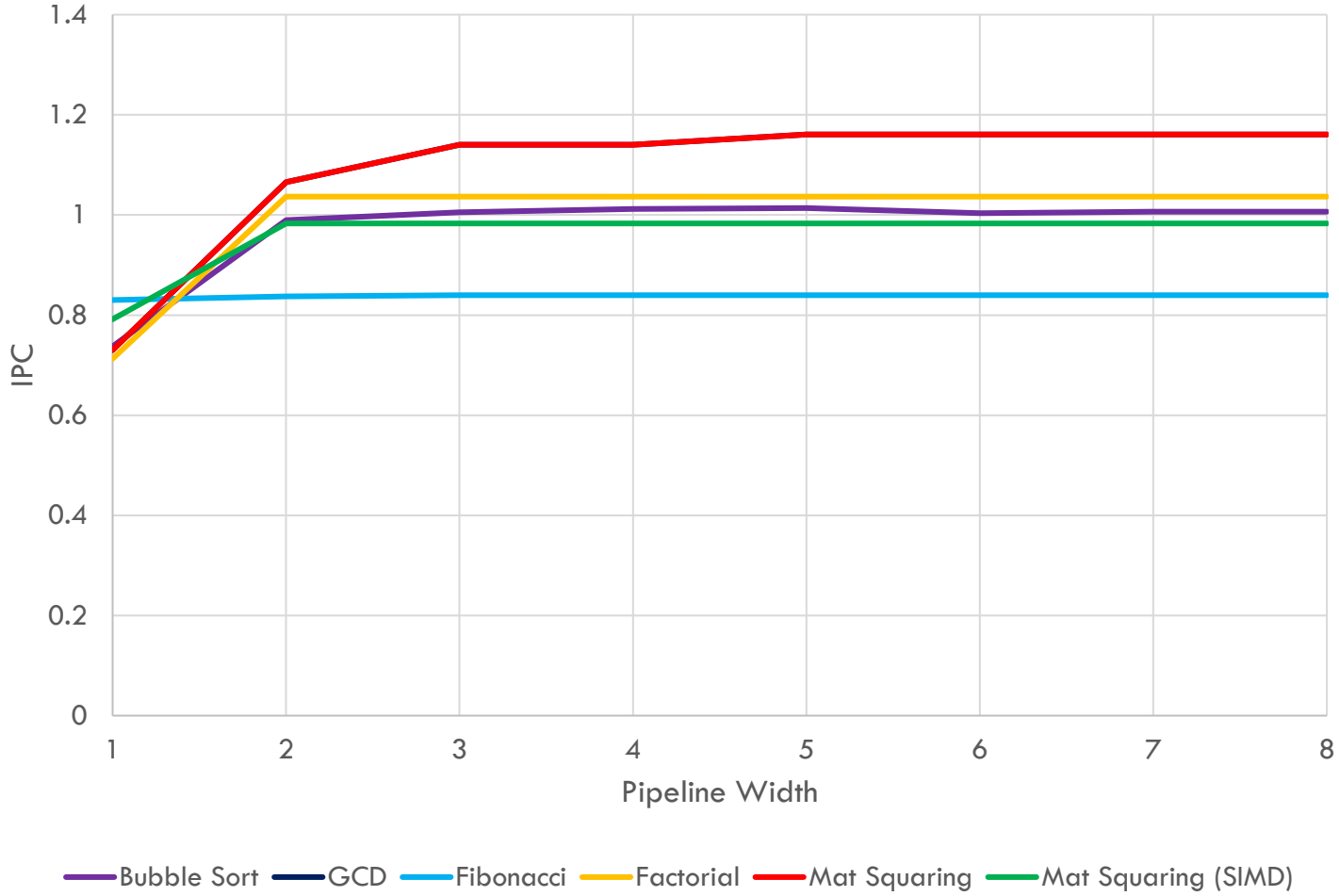


PROCESSOR FEATURES

- 7 stage pipeline (Fetch, Decode, Issue, Dispatch, Execute, Write-result, Commit)
- Execution Unit operand bypass
- Store to Load forwarding via store queue functionality integrated into rob
- Register renaming (32 architectural regs, 64 physical reg, 16 architectural vector regs, 32 (64bit) physical vector regs)
- Scoreboarding (keeps track of which operands are good to be ready straight away in issuing to RS)
- Grouped Reservation Stations
- Reservation Station bypassing
- Dynamic Scheduling + non-blocking issue
- Out of Order execution via Tomasulo + Reorder buffer (ROB)
- Out of Order Loads and Stores via memory disambiguation (check for RAW+WAW hazard in memory)
- Out of Order Vector instructions (SIMD) + chaining (store load forwarding for vector loads and stores)
- Speculation
- Multi-cycle instructions
- Super-scaling
- Branch Prediction : Dynamic 1bit and 2bit, Fixed and Static (always/never taken) + **BTAC**
- Indexed addressing mode loads and stores
- Variable number of execution units
- Variable rob size

EXPERIMENT 1

Hypothesis: The number of instructions per cycle increases as pipeline width increases but will plateau due to dependencies and hazards.



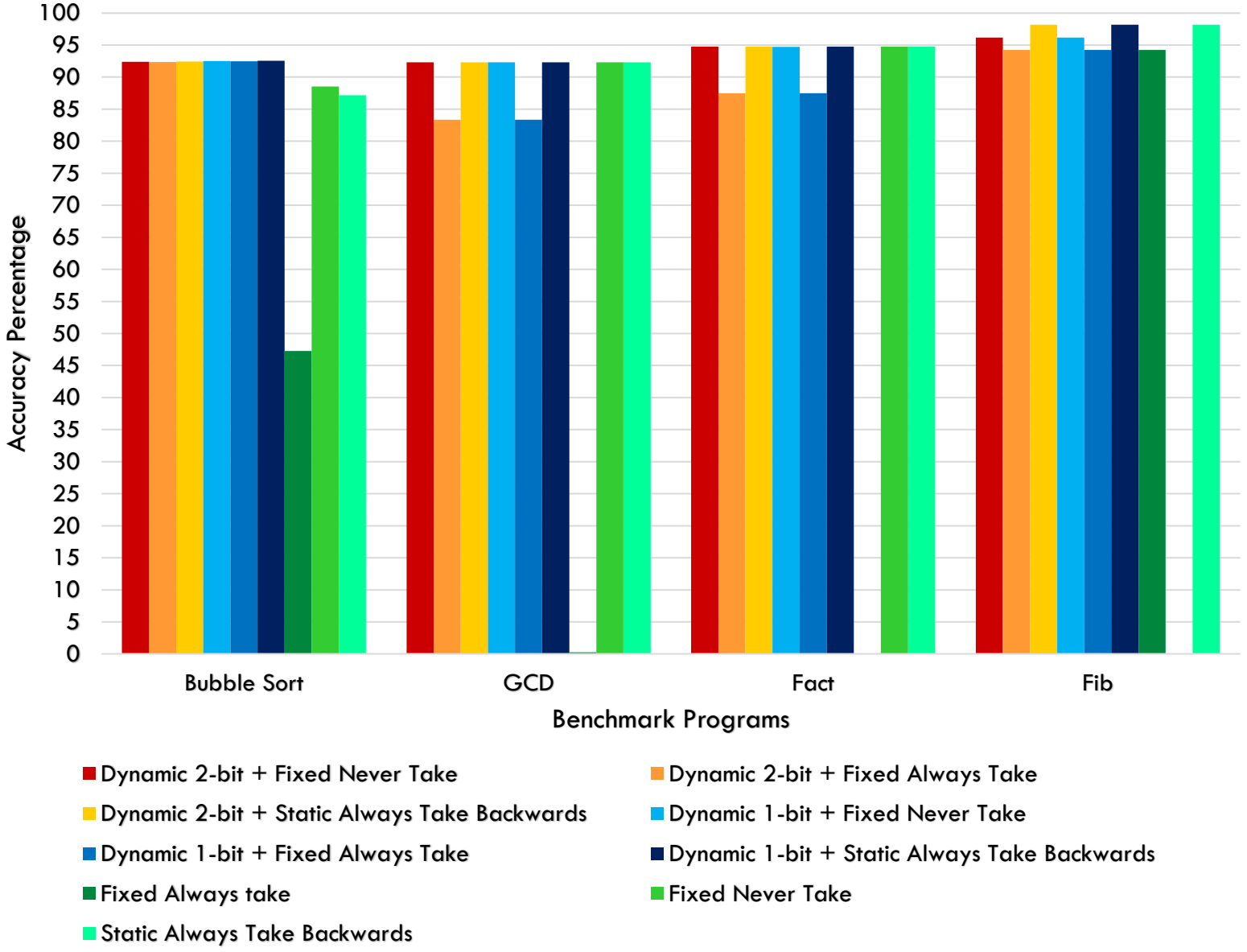
Experiment:
We run each program at different pipeline widths with the cpu default set up of:
4 ALU, 2 LSU, 2 branch units + branch prediction.

Results:
For all the benchmark programs aside from the matrix squaring ones, they approached the highest IPC at around a width of 2 or 3. Only bubble sort has a slight improvement after width 3.

This indicates that the CPU isn't able to make use of the increased pipeline width possibly due to dependencies. The Fibonacci program seems to reach plateau immediately, but it does have a minute increase between 1 and 2 widths.

EXPERIMENT 2

Hypothesis: Dynamic branch prediction (1-bit or 2-bit) will always have a strictly higher accuracy than fixed or static branch predictors (e.g. always take, never take, always take backwards branch).



DEFAULT CPU SETUP:
4 ALU 2 LSU 2 Branch Units
Out of Order, RS bypassing, pipelined 2-way superscalar

EXPERIMENT:
Run benchmark programs with varying branch prediction schemes.

For dynamic schemes we use a fixed/static prediction to supplement the prediction when not available in the BTB/BTAC

RESULTS:
The hypothesis was incorrect, dynamic branch prediction methods are not strictly always more accurate than fixed or static predictions.

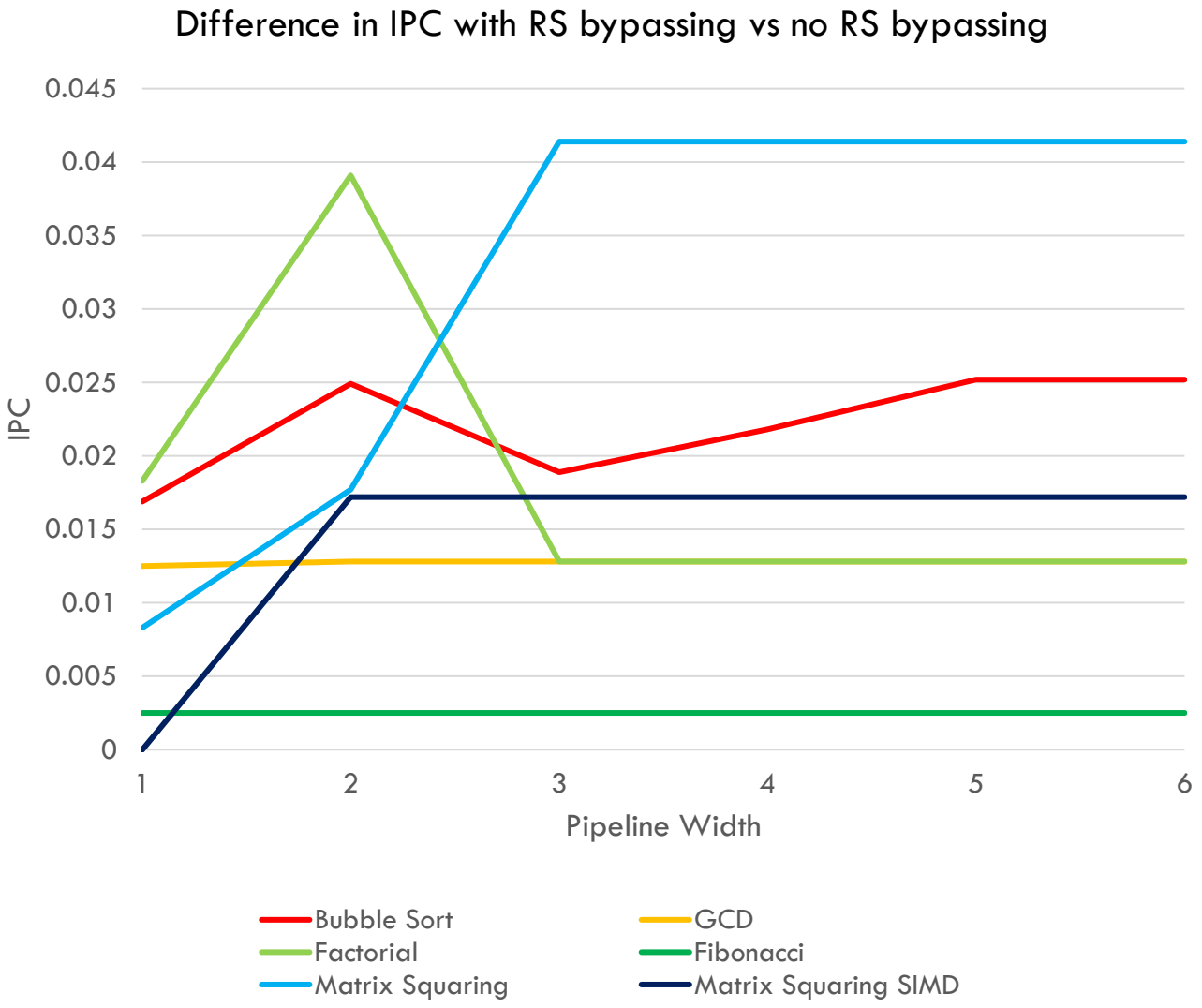
From the accuracies seen, dynamic programs will at most match one of the fixed/static schemes. However, in general dynamic 1-bit and 2-bit branch prediction schemes perform more consistently over these programs presented.

The worst performing branch predictor based on these results is, “Fixed Never Take” where branches are never taken. The best scheme averaging across these results are Dynamic 2-bit/1-bit + Static Always Take Backward in which where it consistently matches the highest accuracies.

NOTE:
These results depend highly on the style in which the assembly is written. For the programs presented, the “Fixed Never Take” scheme isn’t very good, most likely due to many of the branches being used for looping which are taken the majority of time.

EXPERIMENT 3

Hypothesis: Having Reservation station bypass will improve IPC but this improvement will decrease with increased pipeline width due to reservation stations being consistently more full.



Default CPU setup:
4 ALU 2 LSU 2 Branch Units
Out of Order, RS bypassing, pipelined, branch prediction

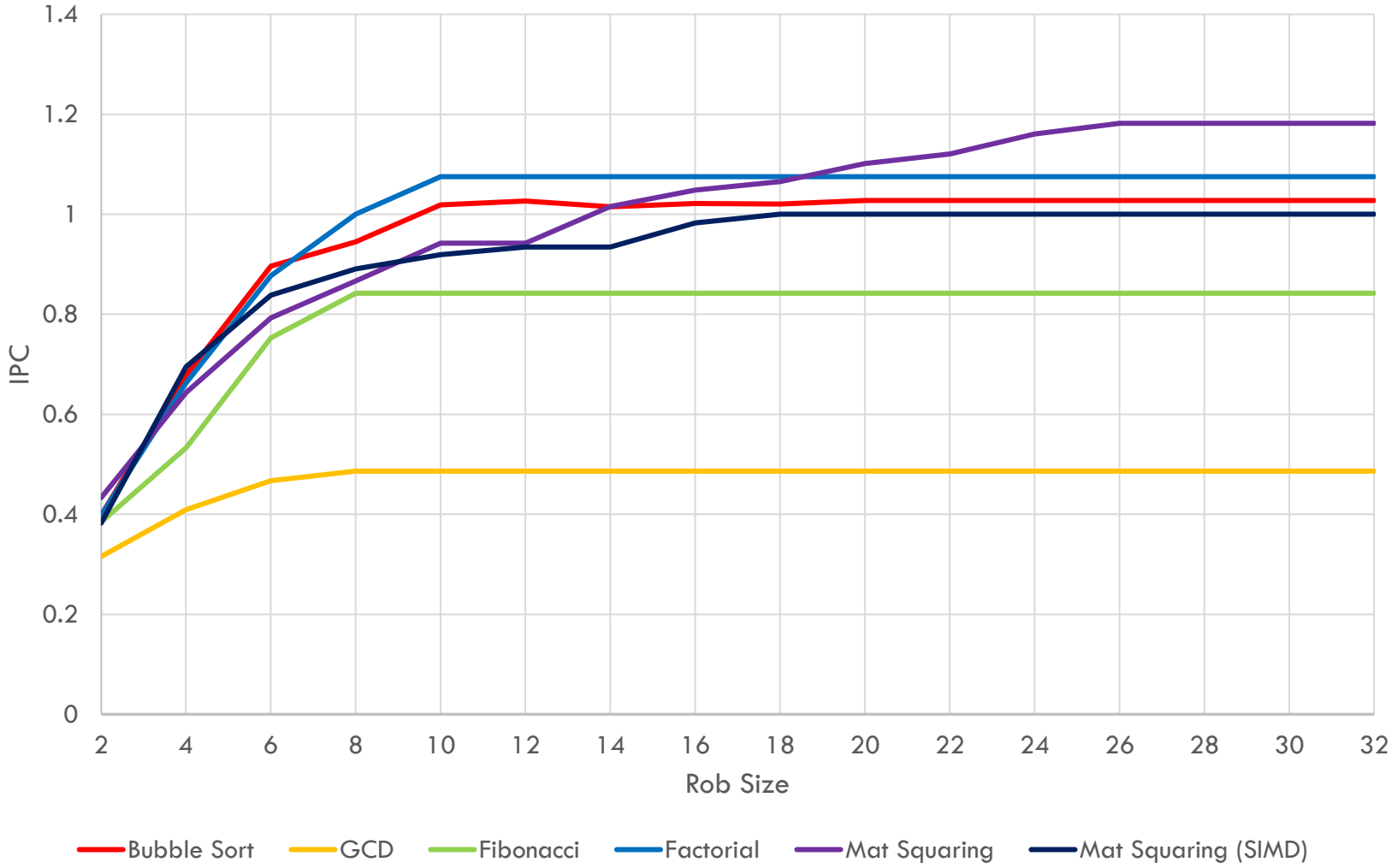
EXPERIMENT:
We run each benchmark program at widths starting at 1 through to 4 with and without reservation station bypassing. Then we take the ***difference*** of IPC between with and without RS bypassing to observe the difference RS bypassing makes.

RESULTS:
The first part of the hypothesis is correct in that reservation station bypassing does indeed improve the IPC as the graph shows only positive values. However, the hypothesis in which IPC improvement will decrease with increased pipeline width is false. Most likely due to each program having various types of dependencies that occur at different frequencies thus preventing dispatching and keeping reservation stations full.

NOTE:
This is most likely due to reservation stations consistently being full regardless of pipeline width. Reservation station bypassing may be more beneficial for programs with more branching and thus more flushing due to reservation stations being empty allowing for bypassing. Bubble sort (35 flushes). Also matrix squaring has no branches but the greatest benefit from RS bypassing. This could be due to the benefit from early entry into EUs being preserved.

EXPERIMENT 4

Hypothesis: Performance increases with ROB size but plateaus at a certain size due to being restricted by reservation station size and the number of execution units.



Experiment:
We ran each program with a default setup of 4 ALU 2 LSU 2 Branch Units + branch prediction + RS bypassing with varying rob sizes.

In total we have 24 reservation stations, 3 per unit. There could be 24 instructions held in reservation stations and 8 instructions being executed, therefore a rob larger than 32 will not see any improvement in IPC.

Results:
The programs do not show an any improvements after 26 rob entries. Additionally, most programs plateau before 26, most likely due to dependencies preventing out of order. “mat squaring” tends to have less dependencies as after each element of the matrix is loaded all further calculations can go out of order.