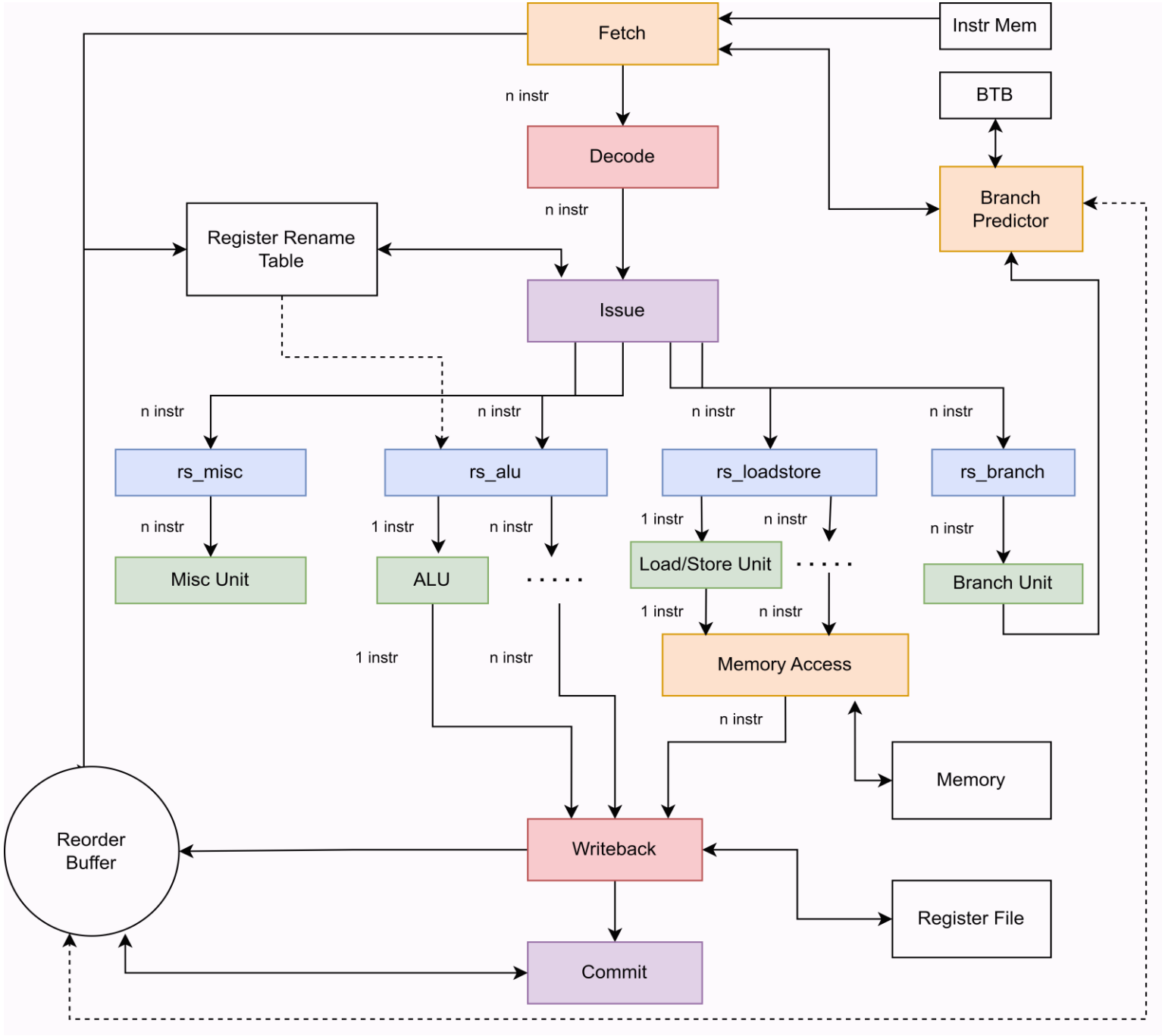
Superscalar Processor Simulator

Joseph Eastoe

Processor Architecture



Process Features

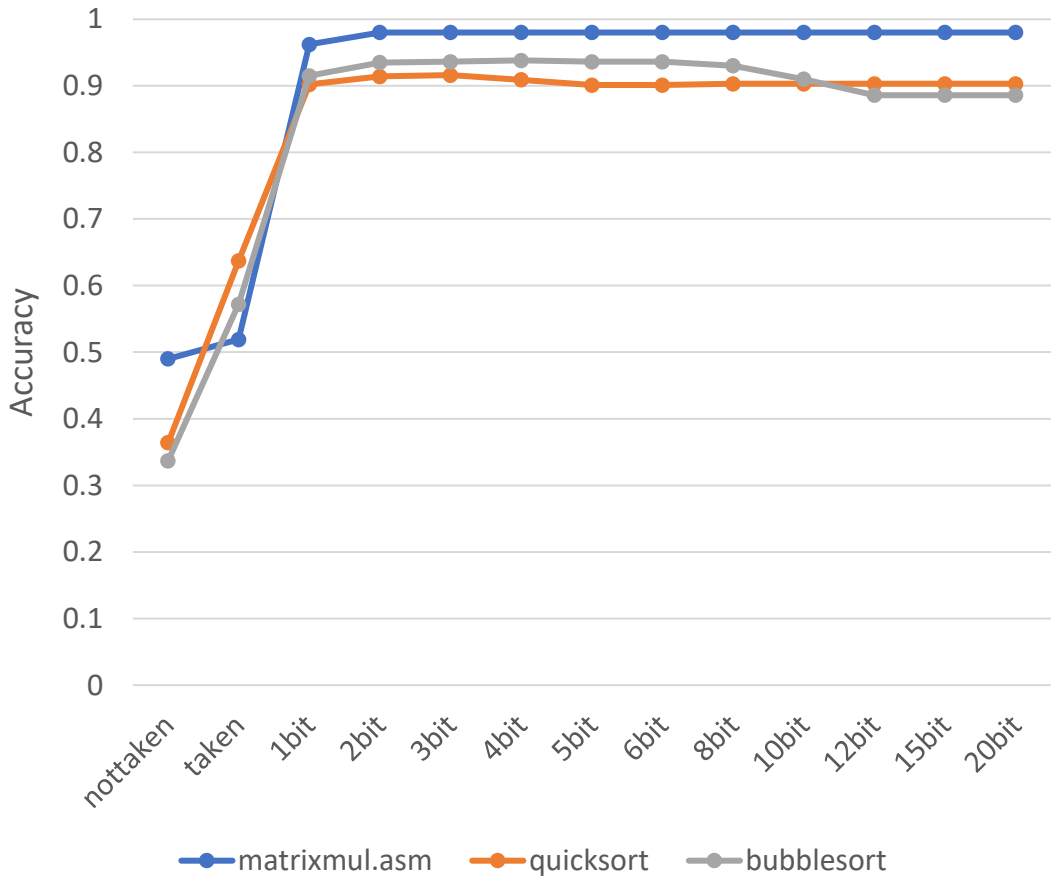
- N width superscalar
- 7-stage pipelined (fetch, decode, issue, execute, memory access and writeback)
- 4 reservation stations (rs_misc, rs_alu, rs_loadstore and rs_branch)
- Non-blocking issue
- Reorder buffer of n size
- Memory access queue and reorder buffer to handle memory dependencies
- Out of order execution
- Tomasulo's algorithm (using the reorder buffer) to handle WAR, WAW and RAW dependencies
- Vector supported instructions

- Execution units
  - N ALUs including vector support
  - N LoadStore units
  - N Branch units
- Configurable Latency for instructions
  - Branch 1 cycles
  - Arithmetic 2 cycles (excluding MUL which is 3)
  - Load/Store 3 cycles
  - Vector unit 3 – 10 units
- Branch prediction (not taken, taken, one bit, two bit and nbits)
- Speculative execution n levels

Hypothesis : 2-bit branch predictor will outperform static and n–bits predictors.

Experiment: I ran matrixmul.asm, quicksort.asm and bubblesort.asm using different branching predictors plotting the accuracy for each.

Results :

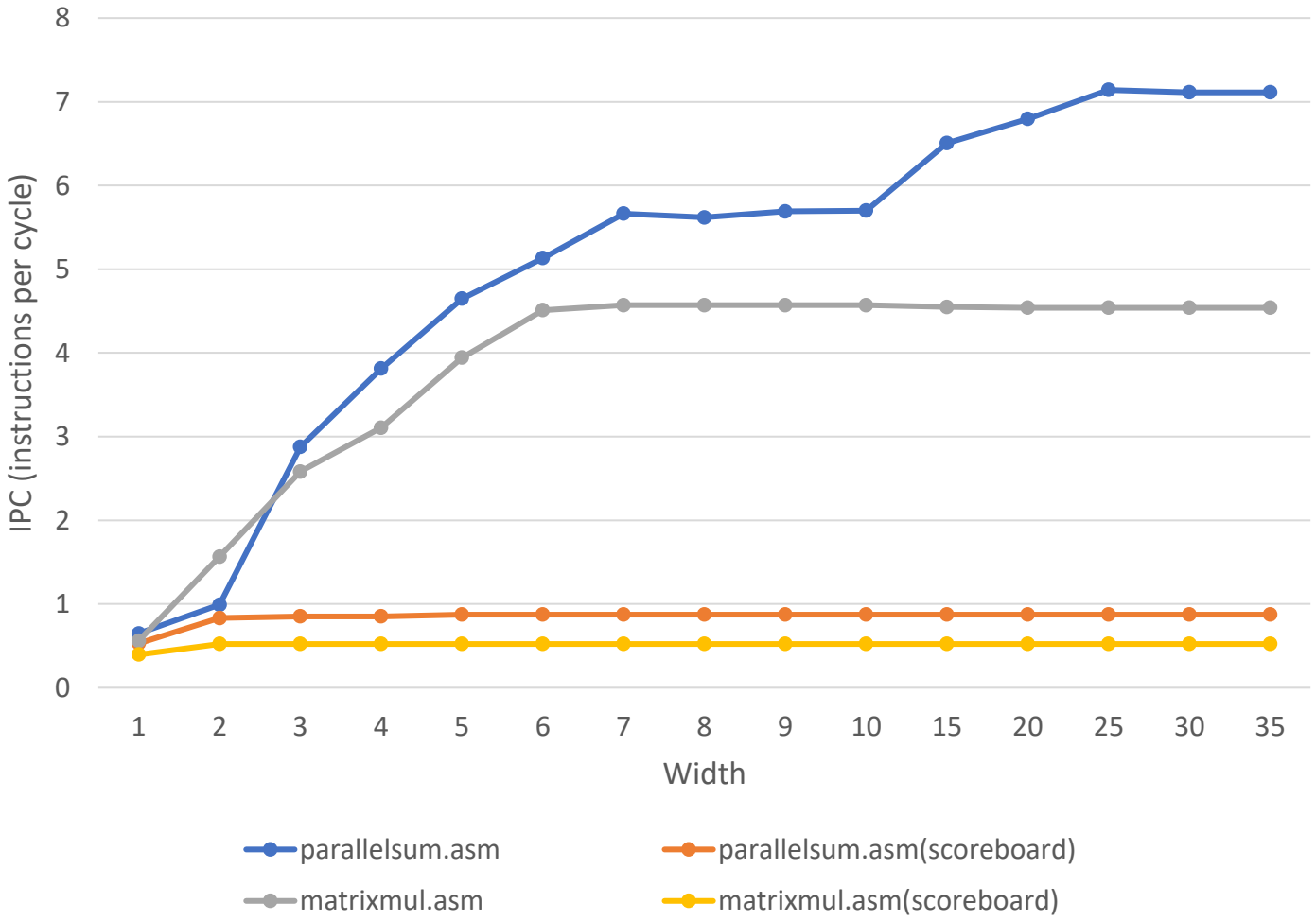


- Predictions for matrixmul where the most accurate, this makes sense as the branching in the other programs are less predictable due to branching depending on the input data.
- The 2-bit predictor outperforms all other predictors, except for the 3-bit predictor, which occasionally yields better results. However, the improvement gained from using the 3-bit predictor, does not justify the additional cost of storing the extra bit.
- Increasing the n-bit size beyond n > 2 does not lead to significant performance improvements, with a slight decrease for the sorting algorithms. This could be from miss predicting the swap function in the algorithms.

Hypothesis : Increasing the superscalar width will increase instructions per cycle with this effect being less prominent in high dependencies programs and when using a scoreboard.

Experiment: I ran parallelsum.asm (low dependencies, an unrolled loop summing up an array in memory), matrixmul.asm (high dependencies, including false). Varying the width of the processor (keeping the number of execution units fixed) and choosing register renaming or scoreboarding. I plotted the instructions per cycle for each.

Results :



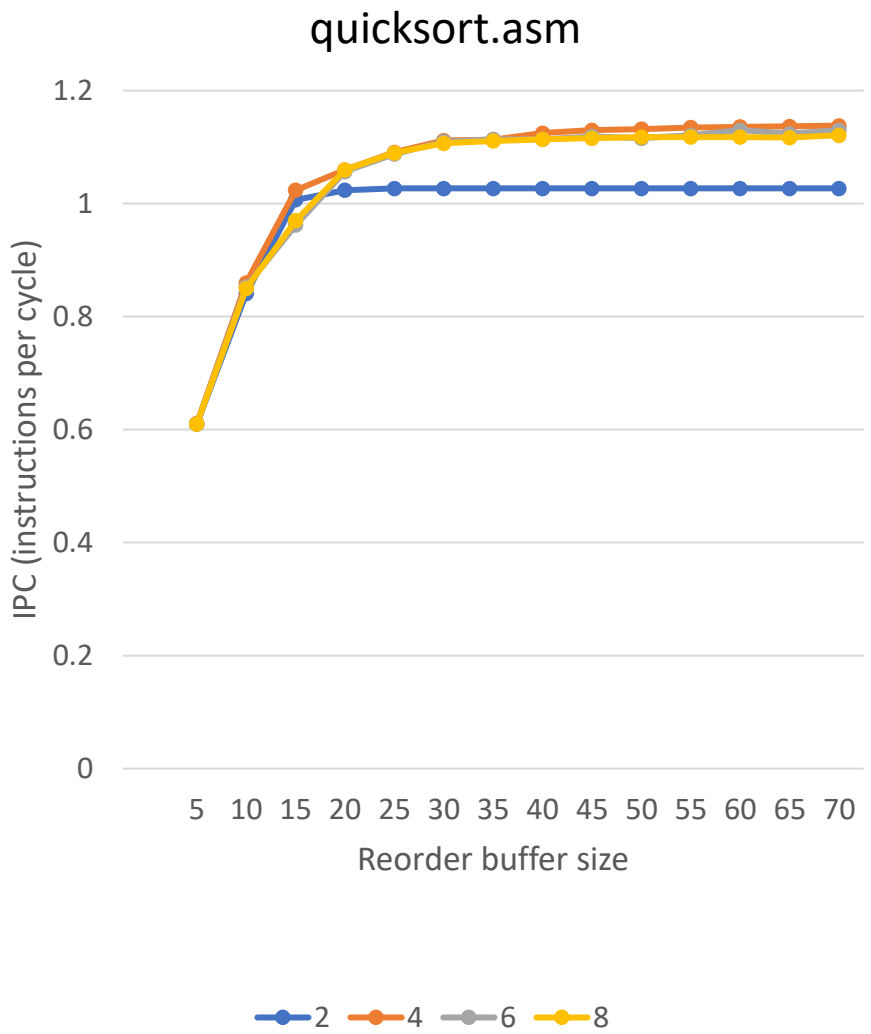
- Each instruction takes multiple cycles to complete excluding branch instructions.
- The hypothesis holds, however the increase in performs begins to plateau.
- This plateauing is caused by true dependences and the width catching up to the size of the reorder buffer, causing the reorder buffer to be full and the processor halting.
- The scoreboard performed worse, due to it not handling false dependences unlike register renaming.

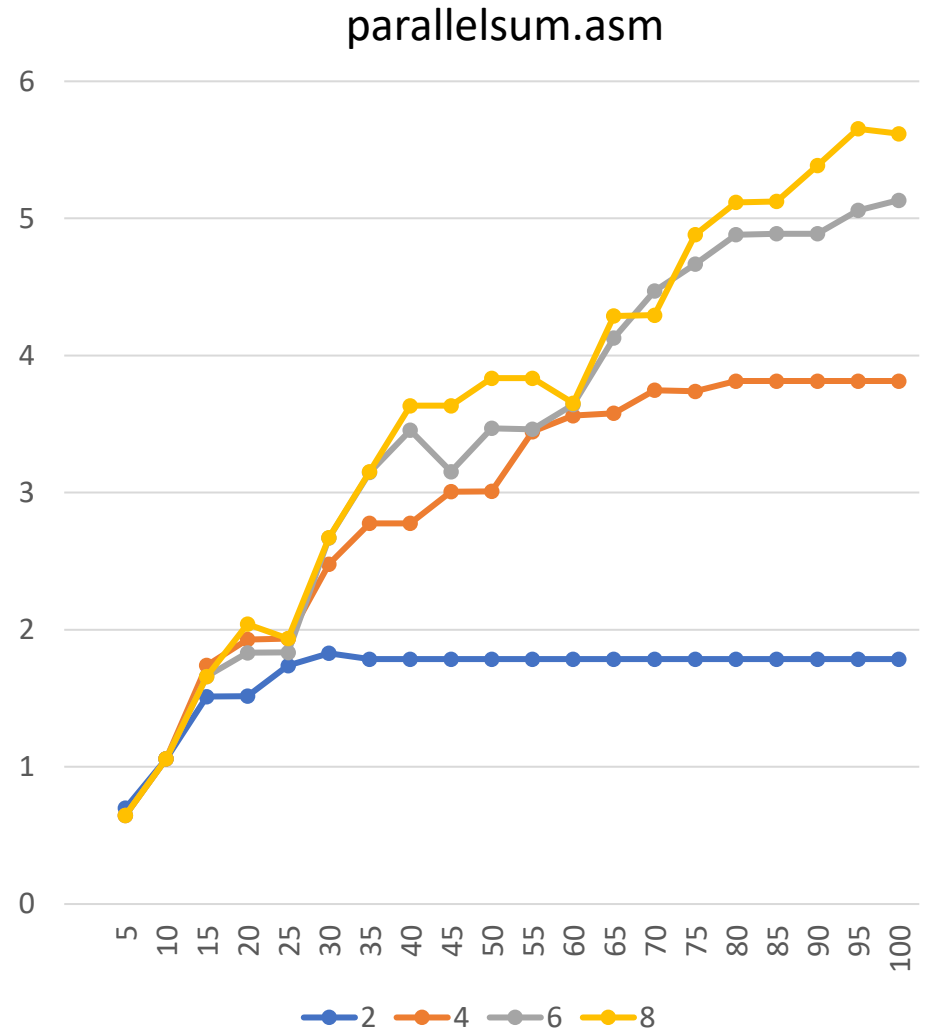
Hypothesis : Increasing reorder buffer size improves performs, however this plateaus when its size is more than 7 (number of stages in the pipeline) times the width of the superscalar processor.

Experiment: I ran quicksort.asm, changing the reorder buffer size, width (keeping the number of execution units fixed) and plotting IPC

Results :

- For quicksort, plateaus for each width start at similar sizes of the reorder buffer. This is due to the quicksort.asm having lots of true dependences which do not allow for increases in performances.
- In contrast, for parallelsum.asm which has a low number of true dependences, the plateaus start at different sizes depending on the width. Here the plateaus are caused by reorder buffer getting full up and the processor halting.

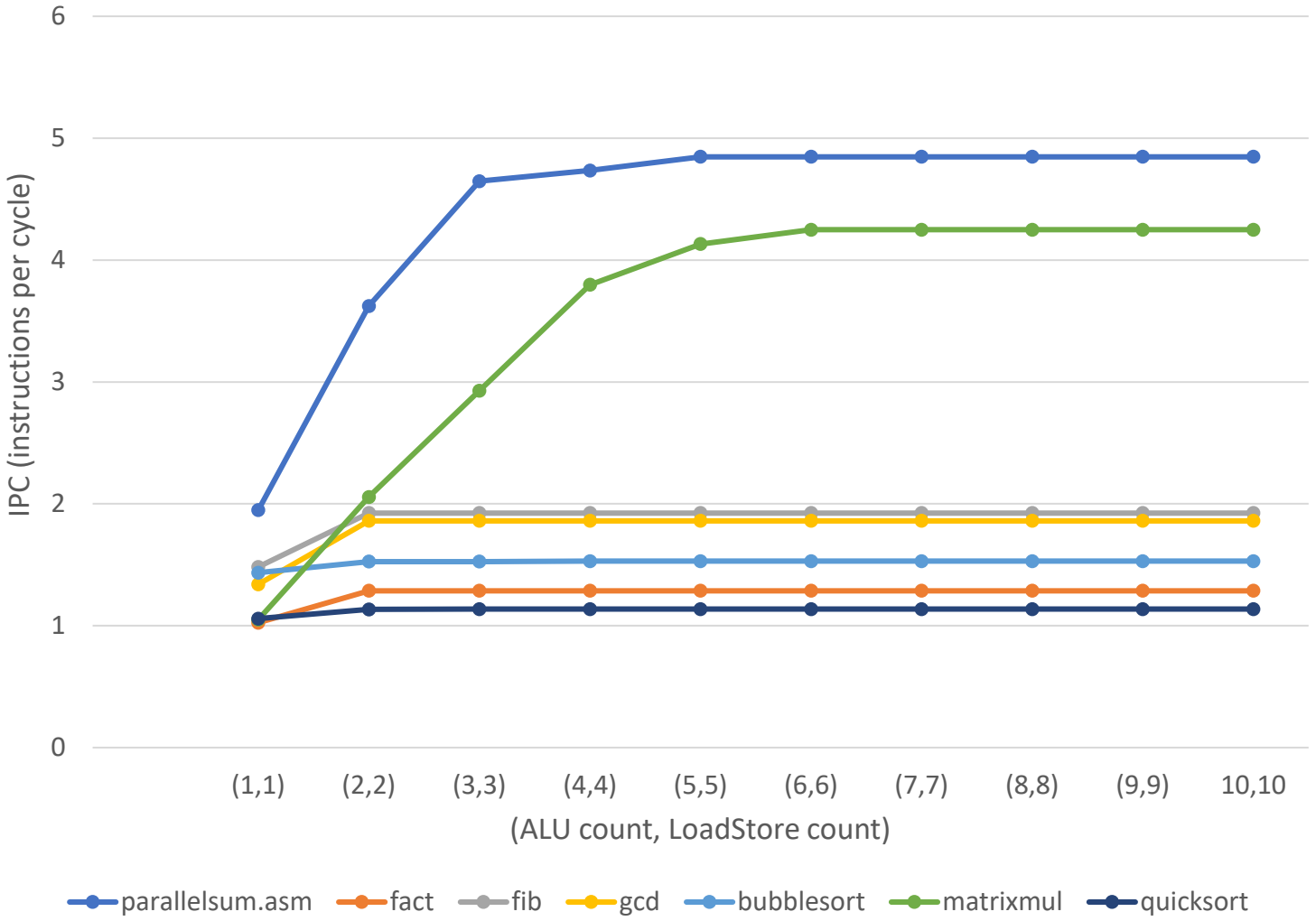




Hypothesis : Increasing the number of ALUs and LoadStore units improves performances.

Experiment: I ran each program varying number of ALUs and Loadstore units (keeping the width constant, setting it to 5 so its not a limiting factor) plotting IPC.

Results :



- Parallelsum.asm and matrixmul.asm took advantage of the extra ALUs and LoadStore units thus increase performances. They were able to do this better than the others because they have a lower number of true dependencies compare to the others.
- Eventually the true dependencies become a limiting factor thus causing plateaus.
- Another reason for plateaus is that, not enough instructions are being issued to the reservation stations; thus, some execution units are left free and not taking full advantage of the increased number. The issue rate depends on the width of the processor, parallelsum.asm and matrixmul.asm both plateau around (5,5) which makes sense as the width was set to 5.