Pointwise Bottleneck Analysis of `compute_tst`

Michael Crabb

May 2025

This report presents a pointwise bottleneck analysis of the `compute_tst` function under four configurations: no splitting and splits with strides of 2, 4, and 8. Cycle counts per four iterations were measured using FE (fetch/decode), DE (decode/dispatch), EX (execution), and WB (write-back) stages.

No Splitting

Without any splitting, the pipeline stages consume the following cycles per four iterations:

Stage	Cycles
FE	3.00
DE	2.75
EX	1.20
WB	0.00

Split with Stride 2

A split stride of 2 increases dispatch overhead, making DE the primary limiter:

Stage	Cycles
FE	3.00
DE	3.75
EX	1.20
WB	0.00

Split with Stride 4

Increasing the stride to 4 doubles execution latency, so EX joins DE as a bottleneck:

Stage	Cycles
FE	3.00
DE	3.75
EX	2.00
WB	0.00

Split with Stride 8

At stride 8, the pattern holds: DE and EX remain the dominant stages, showing saturation:

Stage	Cycles
FE	3.00
DE	3.75
EX	2.00
WB	0.00

Overall, introducing pointwise splits shifts the primary bottleneck from fetch/decode (FE/DE) in the unsplit case to decode/dispatch (DE) at stride 2, and to a combined decode/dispatch and execution (DE/EX) bottleneck at larger strides. Instruction fetch remains constant across all cases, and write-back pressure remains negligible.