Bottleneck Analysis of `reduction.c`

Michael Crabb

April 2025

1. Overview of the Loop

The loop in `compute_tst` performs:

- Load a float: `vmovss (%rdx), %xmm0`
- Load another float and add: `vaddss (%rax), %xmm0, %xmm0`
- Store the result: `vmovss %xmm0, (%rdx)`
- Update pointers: `addq $4, %rax`
- Compare to end pointer: `cmpq %rcx, %rax`
- Loop if necessary: `jne`

This structure represents a simple serial reduction over an array.

2. Instruction Bottleneck Highlights

Based on `reduction.c.mca`:

Instruction	**Latency**	**RThroughput**	**Notes**
`vmovss (%rdx), %xmm0`	8 cycles	0.50	Very high latency load
`vaddss (%rax), %xmm0, %xmm0`	10 cycles	0.50	High latency addition
`vmovss %xmm0, (%rdx)`	1 cycle	1.00	Store operation

Critical Observations:

- Loads and adds dominate cycle counts due to high latency.
- Floating-point operations are the primary cost.
- Memory access is moderately expensive but not the worst bottleneck.

3. Resource Bottleneck

From `reduction.c.mca` and `reduction.c.osaca`:

- Floating-point pipelines show high pressure, particularly ports 5 and 6.
- Load/store units are moderately utilized.

Port pressure summary from `osaca`:

Ports	Pressure
Floating Point Units (5/6)	1.20 (high)
Load Ports (8/9/10)	0.50 (moderate)

Critical Observations:

- Floating-point units are saturated.
- Memory subsystem is not fully saturated.
- Dispatch width is not the limiting factor; serial dependencies are.

4. Loop-Carried Dependencies

From `osaca`:

Instruction	Latency	Dependency Chain
`vmovss (%rdx), %xmm0`	8 cycles	[53, 58, 60]
`vaddss (%rax), %xmm0, %xmm0`	8 cycles	Same chain
`vmovss %xmm0, (%rdx)`	0 cycles	Store after FP calculation

Critical Observations:

- Each iteration depends on the result of the previous iteration.
- Instruction-Level Parallelism (ILP) is severely limited by data dependencies.

5. Summary

Category	Bottleneck?	Details
Floating-Point Arithmetic	Yes	High latency operations, FPU saturation
Memory Access	No	Moderate pressure
Loop-Carried Dependency	Yes	Serial reduction dependency
Dispatch/Ports	No	Ports used efficiently given constraints
uOp Pressure	No	Normal

6. Final Bottleneck Diagnosis

Floating-Point Execution and **Serial Dependency** are the primary bottlenecks.

Each addition operation depends directly on the previous result, which causes serialization of the floating-point operations. Memory accesses are moderately expensive but do not dominate the execution time. Dispatch width is adequate; the issue lies with the critical path created by the floating-point operations and their dependencies.