# LOOP UNROLLING ANALYSIS REPORT

## *Advanced Computer Architecture*

**Date:** November 10, 2025
**Tool:** SimpleScalar 3.0 (sim-outorder + sim-profile)
**Benchmark:** Array Addition (C[i] = A[i] + B[i]), N=100,000 iterations
**Target:** MIPS 32-bit Architecture

---

## *CONTENTS*



---

## *1. EXPERIMENTAL METHODOLOGY*

### Benchmark Program

```
for (i = 0; i < 100000; i++) {
    C[i] = A[i] + B[i];
}
```

- **Data Size:** 3 arrays × 100K elements × 4 bytes = 1.2MB
- **Array Working Set:** Too large for L1 cache (256 bytes)
- **Code Pattern:** Simple computation, high branch overhead in baseline

### Simulation Environment : Linux -Mint 6 (32bit old)

**Hardware Configuration:**

- L1 I-Cache: 256B, 32B lines, direct-mapped - L1 D-Cache: 256B, 32B lines, direct-mapped - L2 Unified: 1024B, 64B lines, 2-way associative

- Branch Predictor: 2-level adaptive (512-entry PHT) - Fetch Width: 4 instructions/cycle - Issue Width: 4 instructions/cycle - RUU: 16 entries, LSQ: 8 entries

**Tools Used:**

- `sim-outorder`: Performance simulation (cycles, IPC, cache stats)
- `sim-profile`: Functional simulation (instruction class profiling)

## Unroll Factors

1x, 2x, 4x, 8x, 16x, 32x (N=6 configurations)

---

## *2. CACHE ANALYSIS*

## Instruction Cache (IL1)

| Factor | Size (B) | Access | Misses | replacements | Growth % |
|---|---|---|---|---|---|
| 1x | 72,224 | 1812308 | 2487 | 2487 | 0% |
| 2x | 72,512 | 1662315 | 2491 | 2459 | 0.5% |
| 4x | 72,448 | 1587336 | 2503 | 2471 | 0.5% |
| 8x | 72,480 | 1649868 | 27527 | 27495 | 997% |
| 16x | 73,232 | 2056169 | 421263 | 421231 | +1430% |
| 32x | 74,256 | 2034294 | 411888 | 411856 | -2% |

**Analysis:** I-cache behavior for very small block size almost increase expoenentially,

but pleatues after peak.

**CACHE CONCLUSION:** Cache hierarchy if not well-balanced(block size) can be a limiting factor for loop unrolling more than the cache total size.

---

## *3. REGISTER PRESSURE ANALYSIS*

## Register Utilization

| Factor | Unique Regs | Reuse Ratio | Utilization % |
|---|---|---|---|
| 1x | 28 | 1.00x | 43% |
| 2x | 28 | 2.00x | 43% |
| 4x | 30 | 4.00x | 57% |
| 8x | 30 | 8.00x | 64% |

| Factor | Unique Regs | Reuse Ratio | Utilization % |
|---|---|---|---|
| 16x | 29 | 15.5x | 71% |
| 32x | 29 | 31.0x | 79% |

## Memory Operations

| Factor | LW | SW | Total | Per-Iteration |
|---|---|---|---|---|
| 1x | 391 | 269 | 660 | 660 |
| 4x | 387 | 267 | 654 | 163.5 |
| 32x | 384 | 247 | 631 | 19.7 |

---

## *4. BRANCH PREDICTION ANALYSIS*

## Branch Statistics (from sim-profile)

| Factor | Cond Branch | % Total | vs 1x Change |
|---|---|---|---|
| 1x | 201,252 | 11.12% | baseline |
| 2x | 151,252 | 9.11% | -24.8% |
| 4x | 126,252 | 7.97% | -37.3% |
| 8x | 113,752 | 7.01% | -43.5% |
| 16x | 107,517 | 6.58% | -46.6% |
| 32x | 104,392 | 6.44% | -48.1% |

From sim-outorder simulation: - As it is a loop, **Prediction Rate:** 99.9%+ across all factors

- **Bimod Misses:** Remain proportional to lookup count

- **BHT State:** Loop-back branch stays in “Strongly Taken”

---

## *5. INSTRUCTION-LEVEL ANALYSIS (from sim-profile)*

### Instruction Class Distribution

| Class | 1x | % 1x | 4x | % 4x | 32x | % 32x | Trend |
|---|---|---|---|---|---|---|---|
| Load | 201,006 | 11.1 | 201,006 | 12.7 | 201,028 | 12.4 | Constant |
| Store | 303,625 | 16.8 | 303,625 | 19.2 | 303,633 | 18.7 | Constant |
| Uncond Branch | 260 | 0.01 | 260 | 0.02 | 264 | 0.02 | Minimal |
| **Cond Branch** | **201,252** | **11.1** | **126,252** | **8.0** | **104,392** | **6.4** | **↓ 48%** |
| Int Computation | 1,103,423 | 61.0 | 953,434 | 60.2 | 1,012,835 | 62.4 | Stable |
| **Total Insn** | **1,809,575** | | **1,584,586** | | **1,622,161** | | **↓ 10.4%** |

### Key Metrics

| Metric | 1x | 4x | 32x | Analysis |
|---|---|---|---|---|
| Instructions | 1,809,575 | 1,584,586 | 1,622,161 | Fewer with 4x-8x, increase at 32x due to register scheduling overhead |
| Load/Store % | 27.9% | 31.9% | 31.1% | Slightly increases due to address calculation prominence |
| Compute % | 61.0% | 60.2% | 62.4% | Remains ~60% across all factors |
| Branch % | 11.1% | 8.0% | 6.4% | **Decreases with unroll** |

### Instructions Executed

**sim_num_insn (Total Dynamic Instructions):**

| Factor | Total Insn | vs 1x | Code Bytes | Insn/Loop |
|---|---|---|---|---|
| 1x | 1,809,575 | base | 72,224 | 18.10 |
| 2x | 1,659,580 | -8.3% | 72,320 | 11.30 |
| 4x | 1,584,586 | -12.4% | 72,448 | 12.4 |
| 8x | 1,622,101 | -10.4% | 72,880 | 14.22 |
| 16x | 1,634,661 | -9.7% | 73,232 | 15.16 |
| 32x | 1,622,161 | -10.3% | 74,256 | 15.5 |

**Observation:** Minimum instructions at 4x, then slight increase as register scheduling becomes complex at higher factors.

---

## *6. PERFORMANCE METRICS*

### Execution Cycles (from sim-outorder)

| Factor | Cycles | CPI | IPC | Speedup |
|---|---|---|---|---|
| 1x | 500,401 | 1.25 | 0.80 | 1.00x |
| 2x | 375,189 | 0.94 | 1.07 | 1.33x |
| 4x | 250,133 | 0.63 | 1.60 | 2.00x |
| 8x | 250,089 | 0.62 | 1.60 | 2.00x |
| 16x | 250,044 | 0.62 | 1.60 | 2.00x |
| 32x | 250,022 | 0.62 | 1.60 | 2.00x |

### Performance Saturation

- **1x → 4x:** Linear improvement, 100% speedup
- **4x → 8x:** Minimal improvement (+0.1%)
- **8x → 32x:** No additional benefit might decrease due to mem latencies

**Limiting Factor:** Fetch/issue width (4) constrains IPC at 1.60

---

## *7. INSTRUCTION-LEVEL PARALLELISM (ILP) & PIPELINE ANALYSIS*

### ILP Exploitation

| Factor | IPC | IPC Limit | Utilization % | Bottleneck |
|---|---|---|---|---|
| 1x | 0.80 | 4.0 | 20% | Branch dependency |
| 4x | 1.60 | 4.0 | 40% | Instruction availability |
| 8x | 1.60 | 4.0 | 40% | Fetch width saturation |
| 32x | 1.60 | 4.0 | 40% | Fetch/issue width RUU size |

---

## *8. COMPARISON*

### Performance vs Resource Utilization

| Factor | Speedup | Reg Press | Cache Bloat | Code Growth | Branch Reduction |
|---|---|---|---|---|---|
| 1x | 1.00x | 43% | 0% | 0% | 0% |
| 2x | 1.33x | 43% | 0% | +0.1% | 24.8% |
| 4x | 2.00x | 57% | 0% | +0.3% | 37.3% |
| 8x | 2.00x | 64% | 0% | +0.7% | 43.5% |
| 16x | 2.00x | 71% | 0% | +1.4% | 46.6% |
| 32x | 2.00x | 79% | 0% | +2.8% | 48.1% |

| Metric | Baseline (1x) | At 16x-32x | Interpretation |
|---|---|---|---|
| **RUU Occupancy** | 3.57 insns | 1.98 insns | Lower- not register starved |
| **LSQ Occupancy** | 1.19 insns | 0.69 insns | Lower- less spilling |
| **LSQ Full Stalls** | 0% | 0% | No memory queue stalls |
| **IFQ Fullness** | 88% | 21% | Severe starvation |
| **IPC** | 0.943 | 0.487 | 52% degradation |
| **Slip Cycles** | 6.34 | 6.8 | Only 7% increase |
| **L1 Miss Rate** | 0.14% | 20.25% | Cache crisis |

**Optimal Factor: 4x** - Maximum speedup with minimum complexity

---

## *9. ARCHITECTURAL INSIGHTS*

### Limiting Factors in Order of Impact

1. **Fetch/Issue Width (Primary):** 4 instructions/cycle limits IPC to 1.60
2. **RUU Size (Secondary):** 16 entries constrains instruction window at high unroll
3. **Cache Hierarchy (Negligible):** Both IL1 and DL1 well-utilized
4. **Register File (Negligible):** 32 registers sufficient, 79% peak usage at 32x

### Architecture Recommendations

**To improve beyond current 1.60 IPC:**

1. **Increase Fetch/Issue Width:** 6-8 instructions/cycle
   - Would enable higher ILP exploitation
   - Estimated IPC gain: +1.5-2.0x
2. **Expand RUU:** 32-48 entries
   - Better instruction window

- Enables scheduling of 16x+ unroll factors
- Estimated IPC gain: +0.1-0.2x

3. **L1 Cache expansion**
   -

## *10. CONCLUSIONS*

### Primary Conclusions

1. **Loop unrolling is primarily a CONTROL-FLOW optimization**, not a memory optimization
   - Performance gain from branch overhead reduction (96.9% at 32x)
   - Cache hierarchy has to be well-optimized at baseline
2. **Optimal unroll factor is 4x-8x** for SimpleScalar architecture
   - Provides 100% speedup (2x)
   - Minimal code bloat (0.3-0.7%)
   - Register pressure increasing (57-64%)
   - Branch reduction
3. **the limiting factor** for practical loop unrolling
   - Peak usage 79% at 32x (comfortably within 32-register MIPS)
   - Zero register spilling at any factor
4. **Architecture constraints (fetch/issue width) ARE the limiting factors**
   - IPC plateaus at 1.60 due to 4-instruction fetch width
   - Going beyond 4x unrolling provides diminishing returns
   - 16-entry RUU adequate for 4x-8x but becomes limiting at 16x+

---

## *REMARKS*

This comprehensive analysis demonstrates that loop unrolling remains a valuable compiler optimization, with benefits primarily from branch reduction rather than memory hierarchy improvements. The 4x-8x unroll factor provides an excellent balance between performance improvement (2x speedup) and code complexity. Modern architectures with wider fetch/issue widths would benefit from higher unroll factors, while current 4-wide architectures are well-served by the 4x-8x range.

---