Performance Analysis of Page Table Implementations

Behrad Farahani Dylan Dingjan

July 3, 2025

1 Introduction

This report presents a performance analysis of two page table implementations: the Simple architecture and the AArch64 architecture. We evaluate their effectiveness in terms of memory usage, page fault handling, and Translation Lookaside Buffer (TLB) performance. The analysis includes single-process and multi-process scenarios with various configuration parameters.

2 Methodology

2.1 Test Environment

- Simple Architecture: 64 MiB pages, single-level page table
- AArch64 Architecture: 16 KiB pages, 4-level page table hierarchy
- TLB: Configurable size with LRU replacement policy
- Physical Memory: First-fit allocation with hole list management

2.2 Test Traces

We used memory traces generated by Valgrind's Lackey tool, including:

- `simple.txt`: Single-process trace with 136,252 memory accesses
- `test-multi-process.txt`: Multi-process trace with context switching

3 Results and Analysis

3.1 Architecture Comparison: Simple vs AArch64

Table 1: Simple vs AArch64 Performance Metrics

Metric	**Simple**	**AArch64**
Page Faults	3	53
Page Table Memory (bytes)	16,777,216	81,936
Page Table Memory (MiB)	16.0	0.078
Max Physical Pages	4	59
Page Size	64 MiB	16 KiB

3.1.1 Analysis

The Simple architecture shows significantly fewer page faults (3 vs 53) due to its large 64 MiB page size. Each page covers a vast address range, reducing the likelihood of page faults. However, this comes at the cost of:

1. **Memory Overhead**: Simple uses 16 MiB for page tables compared to just 78 KiB for AArch64, a roughly 205× difference.
2. **Internal Fragmentation**: With 64 MiB pages, even small allocations waste significant memory.
3. **Physical Memory Usage**: Despite fewer page faults, Simple allocates fewer physical pages (4 vs 59), indicating poor memory utilization.

The AArch64 architecture demonstrates superior memory efficiency through its hierarchical page table structure, allocating page table levels on demand and using smaller 16 KiB pages for fine-grained memory management.

3.2 TLB Effectiveness Analysis

3.2.1 Single-Process Performance

For the single-process trace, all tested TLB sizes achieved a 99.92% hit rate. This indicates:

- The working set of the test program fits within even a 16-entry TLB

Table 2: TLB Performance with Different Configurations

TLB Size	Lookups	Hits	Hit Rate (%)
16 entries	136,252	136,146	99.92
32 entries	136,252	136,146	99.92
64 entries	136,252	136,146	99.92
128 entries	136,252	136,146	99.92

- Strong locality of reference in the memory access pattern
- No benefit from increasing TLB size beyond 16 entries for this workload

The 106 misses (0.08%) correspond to compulsory misses when accessing new pages for the first time.

3.3 Multi-Process and ASID Effectiveness

Table 3: Multi-Process Scenario Performance

Metric	Value
Context Switches	2
Page Faults	4
TLB Lookups	13
TLB Hits	5
TLB Hit Rate	38.46%
TLB Flushes	1

3.3.1 Context Switch Impact

The multi-process trace shows a reduction in TLB hit rate (38.46% vs 99.92%) despite fewer total memory accesses. This demonstrates:

1. **TLB Pollution**: Process switches invalidate TLB entries, causing misses
2. **ASID Benefits**: With ASID support enabled, different processes can coexist in the TLB
3. **Working Set Conflicts**: Multiple processes compete for limited TLB entries

3.4 Physical Memory Management

The hole list implementation with first-fit allocation showed:

- **Efficient Merging**: Adjacent free blocks are coalesced to reduce fragmentation
- **Predictable Allocation**: First-fit provides consistent behavior
- **Low Overhead**: Hole list operations are O(n) where n is the number of holes

4 Recommendations

4.1 Architecture Selection

- Use AArch64 for general-purpose systems requiring efficient memory utilization
- Consider Simple architecture only for specialized workloads with very large contiguous allocations

4.2 TLB Configuration

- 16-32 entries sufficient for single-process workloads with good locality
- Increase to 64-128 entries for multi-process scenarios
- Enable ASID support to reduce context switch overhead

4.3 Future Improvements

1. Implement huge page support in AArch64 for large allocations
2. Add selective TLB flushing to preserve entries across context switches
3. Consider implementing a buddy allocator for better fragmentation management

5 Conclusion

The AArch64 implementation demonstrates superior memory efficiency compared to the Simple architecture, with a 205× reduction in page table overhead. While the Simple architecture shows fewer page faults, this advantage is outweighed by severe internal fragmentation and memory waste. TLB effectiveness is workload dependent. Single process workloads achieve very good hit rates with minimal TLB entries, while multi-process scenarios benefit from larger TLBs and ASID support. The hierarchical page table structure of AArch64, combined with demand paging and efficient TLB management, makes it the correct choice for general purpose operating systems where memory efficiency and scalability are paramount.