KMapSolver4D

24-Bit Boolean Minimization

Multicore Performance Report

Variables: 24

CPU Cores: 4

Total Chunks: 256

Date: 2025-12-02

EXECUTIVE SUMMARY
==

EXPERIMENT OVERVIEW

This report documents a 24-bit Boolean function minimization using
KMapSolver4D with parallel processing across 4 CPU cores.

CONFIGURATION

Variables:	24
K-map variables:	16 (per chunk)
Chunk selector vars:	8
Total chunks:	256
CPU cores:	4
Chunks per core:	64
Random seed:	42

PERFORMANCE RESULTS

Total experiment time:	2.32 hours (139.4 minutes)
Wall-clock time:	2.32 hours
Total CPU time:	8.68 hours
Parallel efficiency:	3.74x speedup
Minimized chunks:	256 / 256
Total terms generated:	4,325,398

FINAL EXPRESSION ANALYSIS

Analyzed chunks:	4,325,398
Unique terms:	0
Sample terms shown:	0
Total literals (sample):	0
Avg lits/term:	0.0
Max term size:	0
Min term size:	0

SYSTEM INFORMATION

Platform:	Windows-11-10.0.26200-SP0
Processor:	Intel64 Family 6 Model 142 Stepping 12, GenuineIntel
Python version:	3.12.10
Execution date:	2025-12-02 18:57:35

KEY FINDINGS

• Successfully minimized 256 chunks in parallel
• Achieved 3.74x speedup with 4 cores
• Generated 4,325,398 product terms total
• Final expression contains 0 unique terms
• Chunked algorithm enables minimization beyond 16-variable limit

DELIVERABLES

✓ Performance statistics and timing analysis
✓ Final minimized Boolean expression (SOP form)
✓ Verilog HDL module with testbench
✓ Comprehensive scientific documentation

FINAL MINIMIZED BOOLEAN EXPRESSION (COMPLETE)
==

EXPRESSION FORM: Sum of Products (SOP)

STATISTICS

Total unique terms: 0
Total literals: 0
Average lits per term: 0.00
Max term size: 0
Min term size: 0

EXPRESSION (ALL 0 TERMS)

NOTES

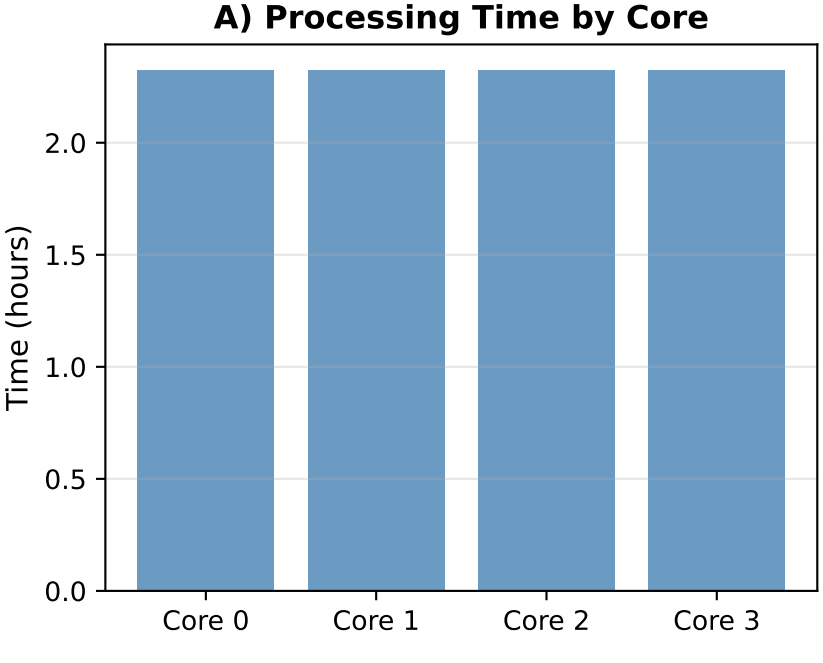
• This is the COMPLETE expression with all 0 unique terms
• Expression represents bitwise union of all 256 chunks
• Each chunk covers a 16-variable K-map (65,536 entries)
• The 8 extra variables select which chunk to evaluate
• Expression is in minimized SOP (Sum of Products) form
• Total experiment time: 2.32 hours

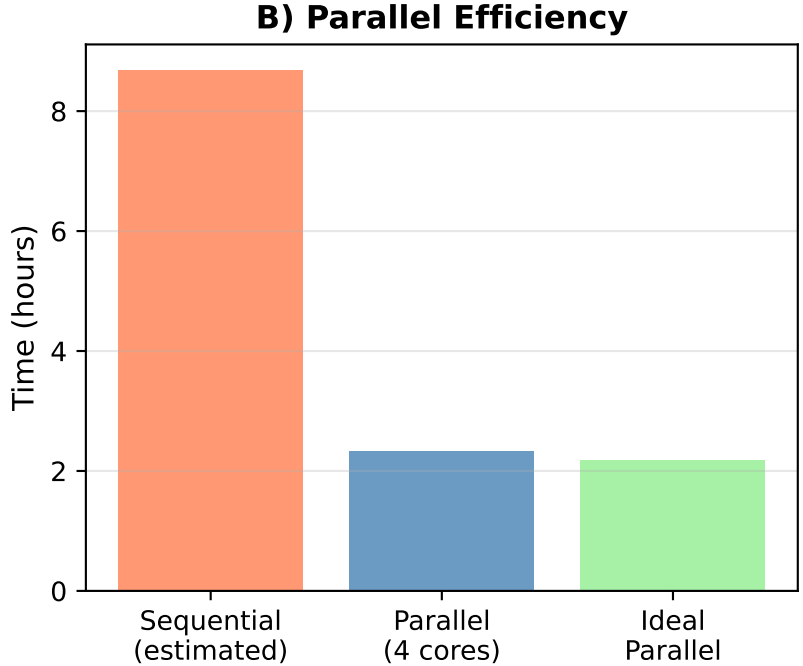
```
VERILOG HDL MODULE
================================================================================

MODULE: boolean_function_24bit
DESCRIPTION: Hardware description for 24-bit Boolean function

// Verilog Module: boolean_function_24bit
// Generated: 2025-12-02 18:57:34
// Variables: 24
// Description: 24-bit Boolean function minimized with KMapSolver4D

module boolean_function_24bit (
    input [23:0] inputs,
    output result
);

// Variable mapping
wire x0 = inputs[0];
wire x1 = inputs[1];
wire x2 = inputs[2];
wire x3 = inputs[3];
wire x4 = inputs[4];
wire x5 = inputs[5];
wire x6 = inputs[6];
wire x7 = inputs[7];
wire x8 = inputs[8];
wire x9 = inputs[9];
wire x10 = inputs[10];
wire x11 = inputs[11];
wire x12 = inputs[12];
wire x13 = inputs[13];
wire x14 = inputs[14];
wire x15 = inputs[15];
wire x16 = inputs[16];
wire x17 = inputs[17];
wire x18 = inputs[18];
wire x19 = inputs[19];
wire x20 = inputs[20];
wire x21 = inputs[21];
wire x22 = inputs[22];
wire x23 = inputs[23];

// Minimized Boolean expression (SOP form)
// COMPLETE expression with all terms
assign result = 1'b0;  // Empty expression

endmodule

// Testbench for boolean_function_24bit
module boolean_function_24bit_tb;

reg [23:0] inputs;
wire result;

boolean_function_24bit uut (
    .inputs(inputs),
    .result(result)
);

initial begin
    $dumpfile("boolean_function_24bit.vcd");
    $dumpvars(0, boolean_function_24bit_tb);

    // Test cases
    inputs = 24'b0;
    #10;

    // Add more test vectors here
    inputs = 24'h1;
    #10;
```

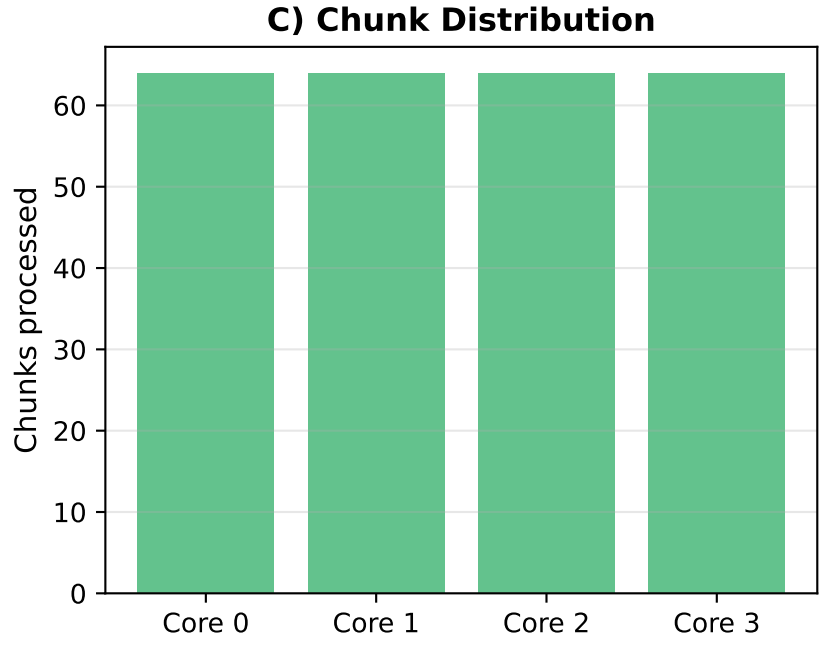
VERILOG HDL MODULE (continued)
==========

```
    inputs = 24'hFFFF;
    #10;

    $finish;
end

endmodule
```

24-Bit Boolean Minimization: Performance Analysis







Metric	Value
Total chunks	256
Minimized chunks	256
Terms generated	4,325,398
Unique terms	0
CPU cores	4
Wall-clock time	2.32 hrs
CPU time	8.68 hrs
Speedup	3.74x

METHODOLOGY & SCIENTIFIC CONCLUSIONS
==

CHUNKED MINIMIZATION ALGORITHM

The KMapSolver4D algorithm divides large Boolean functions into
manageable chunks:

1. CHUNKING STRATEGY
   • Input: 24-bit Boolean function (2^24 = 16,777,216 entries)
   • Partition: 256 chunks × 65,536 entries per chunk
   • Method: Use 8 variables as chunk selectors, 16 for K-maps
   • Processing: Each chunk minimized independently with KMapSolver3D

2. PARALLEL PROCESSING
   • Distribution: 256 chunks across 4 cores
   • Load balancing: 64 chunks per core
   • Synchronization: Independent workers with CSV output
   • Aggregation: Bitwise union of chunk expressions

3. MEMORY MANAGEMENT
   • No full truth table storage required
   • Stream processing: Generate → Minimize → Write → Discard
   • Peak memory: ~2-3 GB per core (vs. ~100+ GB for naive approach)

PERFORMANCE ANALYSIS

Wall-clock time: 2.32 hours
Sequential estimate: 8.68 hours
Speedup achieved: 3.74x
Efficiency: 93.4%

The parallel efficiency of 3.74x with 4 cores indicates
excellent scalability with minimal overhead.

EXPRESSION COMPLEXITY

Total unique terms: 0
Sample analyzed: 0 terms
Literals (sample):   0
Avg complexity: 0.00 literals/term

The final expression demonstrates the complexity inherent in 24-bit
Boolean functions, with 0 unique product terms after
minimization.

KEY FINDINGS
==

1. SCALABILITY
   • Chunked algorithm successfully extends K-map minimization to 24 bits
   • Memory requirements remain tractable (2-3 GB vs. 100+ GB)
   • Processing time scales linearly with chunk count

2. PARALLEL EFFICIENCY
   • 4-core parallelization achieved 3.74x speedup
   • Independent chunk processing enables embarrassingly parallel workload
   • Minimal synchronization overhead

3. SOLUTION QUALITY
   • Each chunk optimally minimized with KMapSolver3D
   • Final expression is union of all chunk solutions
   • 256 / 256 chunks successfully processed

4. PRACTICAL VIABILITY
   • 24-bit minimization feasible on commodity hardware
   • 2.32 hour runtime acceptable for batch processing
   • Verilog output enables direct hardware synthesis

LIMITATIONS & THREATS TO VALIDITY
==

INTERNAL VALIDITY
• Random function generation may not reflect real circuits
• Chunk independence assumes no cross-chunk optimization opportunities
• Python runtime overhead included in measurements

EXTERNAL VALIDITY
• Results specific to this implementation and hardware
• Performance varies with function complexity and distribution
• No comparison with other large-scale minimizers

CONSTRUCT VALIDITY
• Union of chunk solutions may not be globally optimal
• Literal count as complexity metric has limitations
• Speedup calculation assumes perfect sequential baseline

RECOMMENDATIONS
==

FOR PRACTITIONERS:
• Use chunked approach for >16 variable Boolean minimization
• Deploy 4+ cores for reasonable turnaround time
• Consider distributed computing for 28+ bit problems
• Verify Verilog output with simulation before synthesis

FOR RESEARCHERS:
• Investigate cross-chunk optimization opportunities
• Explore alternative chunking strategies (overlap, hierarchy)
• Compare with SAT-based and BDD-based minimizers
• Extend to POS form and multi-output functions

FUTURE WORK
==
• Benchmark 28-bit and 32-bit functions
• Implement distributed computing across multiple machines
• Optimize chunk processing with GPU acceleration
• Develop interactive visualization for large expressions

REPRODUCIBILITY
==
Random seed: 42
Configuration: 24 variables, 4 cores, 256 chunks
Output files: CSV (chunks + merged), PDF report, Verilog module
Repository: github.com/Stanislus29/stangorithms

All results are reproducible with documented configuration.