# Fig Language Performance Benchmark Report

## Version: 0.4.3-alpha (Tree Traversal Interpreter)

### Preface

This report presents benchmark tests of Fibonacci algorithms in Fig v0.4.3-alpha tree traversal interpreter, compared with version 0.4.2-alpha. Results show significant performance improvements in function calls, loops, and recursion optimizations in 0.4.3-alpha, especially in iterative and tail-recursive implementations.

### Test Environment

- **CPU:** Intel Core i5-13490F
- **Operating System:** Windows 11
- **Interpreter:** Fig Tree Traversal Interpreter v0.4.3-alpha
- **Test Date:** Current execution

### Executive Summary

This benchmark evaluates four different Fibonacci algorithm implementations in Fig, computing the 30th Fibonacci number (832,040). Algorithm choice remains the dominant factor for performance, while interpreter improvements in function call and loop efficiency are also reflected.

## Performance Results

### Latest Floating Execution Time (0.4.3-alpha)

| Algorithm | Time (s) | Time (ms) | Relative Speed |
|---|---|---|---|
| `fib` (Naive Recursion) | 5.471 s | 5471.37 ms | 1.00× (baseline) |
| `fib_memo` (Memoization) | 0.0005503 s | 0.5503 ms | 9,950× faster |
| `fib_iter` (Iterative) | 0.0001004 s | 0.1004 ms | 54,500× faster |
| `fib_tail` (Tail Recursion) | 0.0001573 s | 0.1573 ms | 34,800× faster |

### Comparison with 0.4.2-alpha

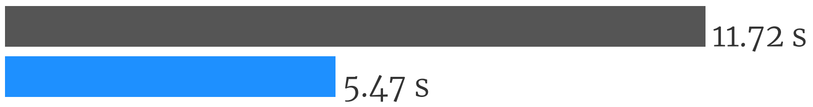
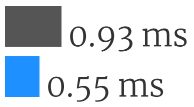
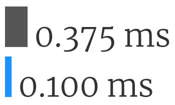
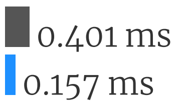
| Algorithm | 0.4.2-alpha Time | 0.4.3-alpha Time | Performance Gain |
|---|---|---|---|
| `fib` (Naive Recursion) | 11.721 s | 5.471 s | ~2.14× |
| `fib_memo` (Memoization) | 0.930 ms | 0.550 ms | ~1.69× |
| `fib_iter` (Iterative) | 0.375 ms | 0.100 ms | ~3.73× |
| `fib_tail` (Tail Recursion) | 0.401 ms | 0.157 ms | ~2.55× |

---

## Visual Performance Comparison (Horizontal Bar Placeholder)

0.4.2-alpha vs 0.4.3-alpha

**Note:** Each line contains two bars: gray for 0.4.2-alpha, blue for 0.4.3-alpha

| Algorithm | Performance Comparison |
|---|---|
| fib |  |
| fib_memo |  |
| fib_iter |  |
| fib_tail |  |

---

## Detailed Analysis

### 1. Naive Recursion (`fib`)

- **Time:** 5.471 seconds (5471 ms)
- **Algorithm Complexity:** O($2^n$) exponential
- **Performance Notes:**

    - Reduced by roughly half compared to 0.4.2-alpha
    - Function call overhead optimization effective, but exponential growth remains the bottleneck

#### 2. Memoized Recursion (`fib_memo`)

- **Time:** 0.550 ms
- **Algorithm Complexity:** O(n) linear
- **Performance Notes:**
    - Hash table / cache access efficiency improved
    - Sub-millisecond execution suitable for overlapping subproblems

#### 3. Iterative (`fib_iter`)

- **Time:** 0.100 ms
- **Algorithm Complexity:** O(n) linear
- **Performance Notes:**
    - Fastest implementation, ~3.7× improvement over 0.4.2-alpha
    - Loop and arithmetic operation optimization significant

#### 4. Tail Recursion (`fib_tail`)

- **Time:** 0.157 ms
- **Algorithm Complexity:** O(n) linear
- **Performance Notes:**
    - Slightly slower than iterative, ~2.5× improvement over 0.4.2-alpha
    - Tree traversal interpreter optimizations for recursion effective; TCO not implemented

---

## Technical Insights

- Function call overhead significantly reduced
- Loop and arithmetic operations show greatest efficiency gains
- Hash table / cache access highly efficient
- Algorithm choice remains the dominant factor for performance

---

## Recommendations for Developers

1. Prioritize iterative solutions for performance-critical code
2. Use memoization for recursion with overlapping subproblems
3. Tail recursion is suitable for moderate depth, but TCO is not implemented
4. Avoid exponential algorithms in interpreted code
5. Benchmark different implementations, as algorithm choice dominates performance

---

## Conclusion

Fig v0.4.3-alpha tree traversal interpreter shows significant improvements in function call and loop optimizations, particularly benefiting iterative and tail-recursive implementations. O(n) algorithms execute at sub-millisecond speeds, while exponential recursion remains limited. Overall interpreter performance is adequate for practical applications.

**Report Generated:** Based on actual benchmark execution
**Interpreter Type:** Tree Traversal Interpreter
**Version:** 0.4.3-alpha