```
================================================================================
                    CHESS ENGINE PROFILE ANALYSIS
================================================================================
Generated: 2025-12-07 17:04:39
Profile:   profile_results_20251207_170217

GLOSSARY
--------------------------------------------------------------------------------

+----------------------+------------------------------------------------------+
| Term                 | Description                                          |
+----------------------+------------------------------------------------------+
| Samples              | Number of times the profiler observed the CPU        |
|                      | executing this function (1 sample = 1 millisecond)   |
+----------------------+------------------------------------------------------+
| NPS                  | Nodes Per Second - positions evaluated per second    |
+----------------------+------------------------------------------------------+
| Nodes                | Total chess positions evaluated during search        |
+----------------------+------------------------------------------------------+
| Depth                | Maximum search depth reached in the game tree        |
+----------------------+------------------------------------------------------+
| Parallel Efficiency  | Ratio of useful work to total work (higher = better) |
+----------------------+------------------------------------------------------+
| Physical Footprint   | Actual memory in use by the process                  |
+----------------------+------------------------------------------------------+
| Peak Footprint       | Maximum memory used during execution                 |
+----------------------+------------------------------------------------------+
| Memory Leaks         | Allocated memory not freed (should be 0)             |
+----------------------+------------------------------------------------------+

EXECUTIVE SUMMARY
--------------------------------------------------------------------------------
  Search Performance:  6,972,806 nodes/second
  Search Depth:        20 plies
  Total Nodes:         289,099,521
  Execution Time:      41.46 seconds
  Thread Count:        8
  Search Efficiency:   75.4% of CPU time in search
  Memory Status:       No leaks detected
  Test Result:         PASS

MOST CPU-INTENSIVE FUNCTIONS
--------------------------------------------------------------------------------
Rank  Samples    Time %   Function                                 Source Locati
on
--------------------------------------------------------------------------------
1     89406      28.37  % negamax(chess::Board&, int, int, int, .. chess_engine_
parallelized.cpp:404
2     77932      24.73  % negamax(chess::Board&, int, int, int, .. chess_engine_
parallelized.cpp:401
3     11947      3.79   % negamax(chess::Board&, int, int, int, .. chess_engine_
parallelized.cpp:337
4     8779       2.79   % negamax(chess::Board&, int, int, int, .. chess_engine_
parallelized.cpp:276
5     7756       2.46   % negamax(chess::Board&, int, int, int, .. chess_engine_
parallelized.cpp:0
6     6655       2.11   % negamax(chess::Board&, int, int, int, .. chess_engine_
parallelized.cpp:289
7     5621       1.78   % operator new(unsigned long)              -
8     5399       1.71   % negamax(chess::Board&, int, int, int, .. chess_engine_
parallelized.cpp:407
9     4740       1.50   % szone_malloc_should_clear                -
```

```
10    4097       1.30   % free_tiny                                -
11    3935       1.25   % negamax(chess::Board&, int, int, int, .. chess_engine_
parallelized.cpp:370
12    3807       1.21   % order_moves(std::vector<chess::Move>&,.. chess_engine_
parallelized.cpp:243
13    3488       1.11   % chess::movegen::legalmoves<(chess::Col.. chess.hpp:380
1
14    3106       0.99   % tiny_malloc_should_clear                 -
15    2880       0.91   % negamax(chess::Board&, int, int, int, .. chess_engine_
parallelized.cpp:460

TIME BREAKDOWN BY CATEGORY
--------------------------------------------------------------------------------
Search (negamax)          [########################################]  73.2% (230
,717 samples)
Memory Allocation         [####------------------------------------]   8.5% (26,
653 samples)
Move Generation           [####------------------------------------]   7.4% (23,
326 samples)
Move Ordering             [#---------------------------------------]   3.4% (10,
744 samples)
Make Move                 [#---------------------------------------]   2.2% (6,8
98 samples)
Quiescence Search         [#---------------------------------------]   2.2% (6,8
08 samples)
Other                     [#---------------------------------------]   2.1% (6,6
99 samples)
Evaluation                [----------------------------------------]   0.9% (2,7
88 samples)
Static Exchange Eval      [----------------------------------------]   0.2% (531
 samples)

PARALLELIZATION ANALYSIS
--------------------------------------------------------------------------------
Detected threads: 8
Configured threads: 8

Time distribution:
  - Search algorithms: 75.4%
  - OpenMP overhead: 0.0%
  - Memory operations: 8.5%

  Parallel efficiency score: 89.9%

MEMORY ANALYSIS
--------------------------------------------------------------------------------
Physical footprint: 6849.0 KB
Peak footprint: 519.4 MB
Malloc nodes: 193
Malloc size: 15.0 KB
Memory leaks: 0
Leaked bytes: 0

[OK] No memory leaks detected.

Memory efficiency: 1.8839 bytes/node

PERFORMANCE METRICS
--------------------------------------------------------------------------------
Search depth reached: 20
Total nodes searched: 289,099,521
Nodes per second: 6,972,806
```

```
Search time (includes time for profiling and analysis): 41.46 seconds
Threads used: 8

Best move found: c4f7
Expected move: c4f7
Test result: PASS

NPS per thread (avg): 871,600

POTENTIAL IMPROVEMENTS
--------------------------------------------------------------------------------
1. Memory allocation takes >5% of time. Consider:
   - Pre-allocating move lists
   - Using thread-local storage for temporary data
   - Stack allocation for small objects


================================================================================
                              END OF REPORT
================================================================================
```