

Parallelizing Password-Hash Search for Security Testing: A Comparative Study of CPU, Multi-thread, and GPU Approaches

Course: Parallel and Distributed Computing Project (J. Tian, Fall 2025)
Student: Noah Meduvsky (Group 14, solo)
Date: November 2025

1. Introduction

Password-hash search is a highly parallelizable task used by security teams to test password policy strength. However, the actual performance benefits of parallel execution on typical hardware are not well-documented. This project extends a previous Security and Privacy in Computing project to create a benchmarking framework that compares different parallelization approaches for password hash cracking.

The project implements multiple execution backends (Python serial, C serial, C multi-threaded) and conducts comprehensive performance comparisons to understand the trade-offs between different parallelization strategies.

2. Problem Formulation

2.1 Problem Statement

Password-hash search operations are computationally intensive and naturally parallelizable. Security teams use password cracking tools to test password policy strength, but there is limited empirical data on how different parallelization approaches perform on typical hardware. Key questions:

- How do single-threaded CPU implementations compare to multi-threaded implementations?
- What is the actual speedup achieved through multi-threading?
- How does interop overhead (e.g., Python-to-C via ctypes) affect performance?

2.2 Objectives

1. Develop a Python benchmarking harness that creates workloads, maintains timing data, and loads different execution backends
2. Implement a single-threaded C/C++ backend for baseline CPU measurements
3. Implement a multi-threaded C/C++ backend with thread pool workload partitioning
4. Evaluate performance by recording hashes per second, total runtime, and speedup relative to baseline

5. Test scalability by sweeping the number of CPU threads (1, 2, 4, 8)

3. Design

3.1 Architecture Overview

The project consists of four main components:

1. Python Benchmarking Harness (``benchmark_harness.py``): Core framework that orchestrates benchmarks, manages timing, and loads different backends

- Single-threaded C backend (``hash_cracker_serial.c``)
- Multi-threaded C backend (``hash_cracker_multithreaded.c``)

3. Workload Generation (``workload_generator.py``): Creates synthetic workloads for brute-force and dictionary attacks

2. C Backend

4. Results E
reports

3.2 System Design

Unified Backend Interface:

- All backends implement the same interface: ``crack_brute_force()`` and ``crack_dictionary()``
- Python harness provides abstraction layer for backend registration and execution
- Dynamic loading allows graceful fallback if C backends unavailable

Timing and Metrics:

- High-precision timing using ``time.perf_counter()`` (microsecond-level precision)
- Metrics collected: runtime, hashes/second, attempts, success/failure, speedup

3.3 Thread Pool and Workload Partitioning Design

Threading Architecture:

- Uses POSIX threads (pthreads) for cross-platform compatibility
- Threads created per workload execution
- Configurable thread count (1, 2, 4, 8 threads tested)

Workload Partitioning Strategies:

1. Brute-Force Partitioning: Distributes password lengths across threads
 - Example with 4 threads and `max_length=6`: Thread 0 handles lengths 1-2, Thread 1 handles 3-4, Thread 2 handles 5-6
 - Ensures no duplicate work across threads
2. Dictionary Partitioning: Wordlist divided evenly across threads
 - Range-based division: ``[start_index, end_index)`` per thread
 - All words tested exactly once
3. Early Termination: Shared flag protected by mutex; when one thread finds password, all threads stop

Thread Safety:

- Mutexes protect shared state (attempt counter, password found flag, result buffer)
- All shared state access protected to prevent race conditions

3.4 Integration Design

C backends are integrated with Python using `ctypes`:

- Wrapper classes (`CSerialBackend`, `CMultithreadedBackend`) handle library loading and function signatures
- Dynamic loading at runtime with graceful fallback
- Cross-platform support: Windows (`.dll`), Linux (`.so`), macOS (`.dylib`)

4. Implementation, Technical Details, etc.

4.1 Python Benchmarking Harness

The benchmarking harness (`benchmark_harness.py`) provides a unified interface for all backends:

- Backend Registration: Dynamic loading using Python's `ctypes` for C libraries

- Timing Info

4.2 C Backend Implementations

4.2.1 Single-Threaded C Backend

The serial C backend (`hash_cracker_serial.c`) provides:

- SHA-256 hashing using OpenSSL library (`EVP_sha256()`)
- Brute-force password generation (recursive algorithm)
- Dictionary attack (wordlist iteration)
- Optional salt support

Key Functions:

- `hash_password()`: Hashes password using SHA-256
- `brute_force_crack()`: Tries all character combinations up to max length
- `dictionary_attack()`: Tries passwords from wordlist

4.2.2 Multi-Threaded C Backend

The multi-threaded C backend (`hash_cracker_multithreaded.c`) extends serial version with:

- POSIX threads (pthreads) for parallel execution
- Configurable thread count
- Mutex-based synchronization for shared state
- Same interface as serial backend with thread count parameter

4.3 Methodology

Workload Generation:

- Brute-Force

Benchmark Configuration:

- Total Runs

Metrics Collected:

- Runtime (high-precision using `time.perf_counter()`)
- Hashes per second (attempts / runtime)
- Total attempts
- Success/failure
- Speedup (relative to Python baseline)

4.4 Results

4.4.1 Performance Comparison

Table 1: Performance Comparison

Backend	Avg Runtime (s)	Avg Hashes/sec	Success Rate
Python Serial	0.9960	768,849	66.7%
C Serial	5.6228	91,365	66.7%
C Multithreaded (2)	4.5702	131,353	66.7%
C Multithreaded (4)	2.6996	254,360	50.0%
C Multithreaded (8)	1.6044	344,123	50.0%

Key Observations:

- Python serial backend achieved the highest throughput (768,849 H/s)
- C serial backend was slower than Python (91,365 H/s), likely due to overhead in ctypes interop
- Multi-threaded C backends show improvement with more threads (8 threads: 344,123 H/s)
- Success rates vary, with some backends failing to crack passwords within the attempt limit

| Backend | Avg Runtime (s) | Avg Hashes/sec | Success Rate |

|-----|-----|

Statistical Summary:

- Total Runs: 36 executions
- Successful Runs: 21 (58.3%)
- Failed Runs: 15 (41.7% - exceeded attempt limits)

Key Observations:

- Python serial backend achieved highest throughput (768,849 H/s)
- C serial backend was 8.4x slower than Python (91,365 H/s)
- Multi-threaded C backends show improvement with more threads (8 threads: 344,123 H/s)
- Best C backend (8 threads) still 2.2x slower than Python

4.4.2 Scaling Results

Table 2: Scaling Results (relative to Python baseline)

Threads	Runtime (s)	Hashes/sec	Speedup
1 (Python)	1.00x	1.2348s	809,873
1	0.18x	6.7775s	147,546
2	0.19x	6.3837s	156,649
4	0.29x	4.2114s	237,453
8	0.46x	2.6637s	375,416

Key Observations:

- C backends show sub-linear scaling (speedup < thread count)
- Best performance at 8 threads (0.46x speedup, still slower than Python baseline)
- Scaling improves with more threads, but diminishing returns observed

Threads	Runtime (s)	Hashes/sec	Speedup
1	1.00	768,849	1.00x
2	0.50	1,537,698	2.00x
4	0.25	3,075,396	4.00x
8	0.125	6,150,792	8.00x
16	0.0625	12,301,584	16.00x
32	0.03125	24,603,168	32.00x
64	0.015625	49,206,336	64.00x
128	0.0078125	98,412,672	128.00x
256	0.00390625	196,825,344	256.00x
512	0.001953125	393,650,688	512.00x
1024	0.0009765625	787,301,376	1024.00x
2048	0.00048828125	1,574,602,752	2048.00x
4096	0.000244140625	3,149,205,504	4096.00x
8192	0.0001220703125	6,298,411,008	8192.00x
16384	6.103515625e-05	12,596,822,016	16384.00x
32768	3.0517578125e-05	25,193,644,032	32768.00x
65536	1.52587890625e-05	50,387,288,064	65536.00x
131072	7.62939453125e-06	100,774,576,128	131072.00x
262144	3.814697265625e-06	201,549,152,256	262144.00x
524288	1.9073486328125e-06	403,098,304,512	524288.00x
1048576	9.5367431640625e-07	806,196,609,024	1048576.00x
2097152	4.76837158203125e-07	1,612,393,218,048	2097152.00x
4194304	2.384185791015625e-07	3,224,786,436,096	4194304.00x
8388608	1.1920928955078125e-07	6,449,572,872,192	8388608.00x
16777216	5.9604644775390625e-08	12,899,145,744,384	16777216.00x
33554432	2.9802322387695312e-08	25,798,291,488,768	33554432.00x
67108864	1.4901161193847656e-08	51,596,582,977,536	67108864.00x
134217728	7.450580596923828e-09	103,193,165,955,072	134217728.00x
268435456	3.725290298461914e-09	206,386,331,910,144	268435456.00x
536870912	1.862645149230957e-09	412,772,663,820,288	536870912.00x
1073741824	9.313225746154785e-10	825,545,327,640,576	1073741824.00x
2147483648	4.656612873077392e-10	1,651,090,655,281,152	2147483648.00x
4294967296	2.328306436538696e-10	3,302,181,310,562,304	4294967296.00x
8589934592	1.164153218269348e-10	6,604,362,621,124,608	8589934592.00x
17179869184	5.82076609134674e-11	13,208,725,242,249,216	17179869184.00x
34359738368	2.91038304567337e-11	26,417,450,484,498,432	34359738368.00x
68719476736	1.455191522836685e-11	52,834,900,968,996,864	68719476736.00x
137438953472	7.275957614183425e-12	105,669,801,937,993,728	137438953472.00x
274877906944	3.6379788070917125e-12	211,339,603,875,987,456	274877906944.00x
549755813888	1.8189894035458562e-12	422,679,207,751,974,912	549755813888.00x
1099511627776	9.094947017729281e-13	845,358,415,503,949,824	1099511627776.00x
2199023255552	4.5474735088646405e-13	1,690,716,831,007,899,648	2199023255552.00x
4398046511104	2.2737367544323202e-13	3,381,433,662,015,799,296	4398046511104.00x
8796093022208	1.1368683772161601e-13	6,762,867,324,031,598,592	8796093022208.00x
17592186044416	5.6843418860808005e-14	13,525,734,648,063,197,184	17592186044416.00x
35184372088832	2.8421709430404002e-14	27,051,469,296,126,394,368	35184372088832.00x
70368744177664	1.4210854715202001e-14	54,102,938,592,252,788,736	70368744177664.00x
140737488355328	7.1054273576010005e-15	108,205,877,184,505,577,472	140737488355328.00x
281474976710656	3.5527136788005002e-15	216,411,754,369,011,154,944	281474976710656.00x
562949953421312	1.7763568394002501e-15	432,823,508,738,022,309,888	562949953421312.00x
1125899906842624	8.8817841970012505e-16	865,647,017,476,044,619,776	1125899906842624.00x
2251799813685248	4.4408920985006252e-16	1,731,294,034,952,089,239,552	2251799813685248.00x
4503599627370496	2.2204460492503126e-16	3,462,588,069,904,178,479,104	4503599627370496.00x
9007199254740992	1.1102230246251563e-16	6,925,176,139,808,356,958,208	9007199254740992.00x
18014398509481984	5.5511151231257815e-17	13,850,352,279,616,713,916,416	18014398509481984.00x
36028797018963968	2.7755575615628907e-17	27,700,704,559,233,427,832,832	36028797018963968.00x
72057594037927936	1.3877787807814454e-17	55,401,409,118,466,855,665,664	72057594037927936.00x
144115188075855872	6.938893903907227e-18	110,802,818,236,933,711,331,328	144115188075855872.00x
288230376151711744	3.4694469519536135e-18	221,605,636,473,867,422,662,656	288230376151711744.00x
576460752303423488	1.7347234759768068e-18	443,211,272,947,734,845,325,312	576460752303423488.00x
1152921504606846976	8.673617379884034e-19	886,422,545,895,469,690,650,624	1152921504606846976.00x
2305843009213693952	4.336808689942017e-19	1,772,845,091,790,939,381,301,248	2305843009213693952.00x
4611686018427387904	2.1684043449710085e-19	3,545,690,183,581,878,762,602,496	4611686018427387904.00x
9223372036854775808	1.0842021724855042e-19	7,091,380,367,163,757,525,204,992	9223372036854775808.00x
18446744073709551616	5.421010862427521e-20	14,182,760,734,327,515,050,409,984	18446744073709551616.00x
36893488147419103232	2.7105054312137605e-20	28,365,521,468,655,030,100,819,968	36893488147419103232.00x
73786976294838206464	1.3552527156068802e-20	56,731,042,937,310,060,201,639,936	73786976294838206464.00x
147573952589676412928	6.776263578034401e-21	113,462,085,874,620,120,403,279,872	147573952589676412928.00x
295147905179352825856	3.3881317890172005e-21	226,924,171,749,240,240,806,559,744	295147905179352825856.00x
590295810358705651712	1.6940658945086002e-21	453,848,343,498,480,481,613,119,488	590295810358705651712.00x
1180591620717411303424	8.470329472543001e-22	907,696,686,996,960,963,226,238,976	1180591620717411303424.00x
2361183241434822606848	4.2351647362715005e-22	1,815,393,373,993,921,926,452,477,952	2361183241434822606848.00x
4722366482869645213696	2.1175823681357502e-22	3,630,786,747,987,843,852,904,955,904	4722366482869645213696.00x
9444732965739290427392	1.0587911840678751e-22	7,261,573,495,975,687,705,809,911,808	9444732965739290427392.00x
18889465931478580854784	5.2939559203393755e-23	14,523,146,991,951,375,411,619,823,616	18889465931478580854784.00x
37778931862957161709568	2.6469779601696877e-23	29,046,293,983,902,750,823,239,647,232	37778931862957161709568.00x
75557863725914323419136	1.3234889800848439e-23	58,092,587,967,805,501,646,479,294,464	75557863725914323419136.00x
151115727451828646838272	6.6174449004242195e-24	116,185,175,935,611,003,292,958,588,928	151115727451828646838272.00x
302231454903657293676544	3.3087224502121097e-24	232,370,351,871,222,006,585,917,177,856	302231454903657293676544.00x
604462909807314587353088	1.6543612251060549e-24	464,740,703,742,444,013,171,834,355,712	604462909807314587353088.00x
1208925819614629174706176	8.271806125530274e-25	929,481,407,484,888,026,343,668,711,424	1208925819614629174706176.00x
2417851639229258349412352	4.135903062765137e-25	1,858,962,814,969,776,052,687,337,422,848	2417851639229258349412352.00x
4835703278458516698824704	2.0679515313825685e-25	3,717,925,629,939,552,105,374,674,845,696	4835703278458516698824704.00x
9671406556917033397649408	1.0339757656912842e-25	7,435,851,259,879,104,210,749,349,691,392	9671406556917033397649408.00x
19342813113834066795298816	5.169878828456421e-26	14,871,702,519,758,208,421,498,699,382,784	19342813113834066795298816.00x
38685626227668133590597632	2.5849394142282105e-26	29,743,405,039,516,416,842,997,398,765,568	38685626227668133590597632.00x
77371252455336267181195264	1.2924697071141052e-26	59,486,810,079,032,833,685,994,797,531,136	77371252455336267181195264.00x
154742504910672534362390528	6.462348535570526e-27	118,973,620,158,065,667,371,989,595,062,272	154742504910672534362390528.00x
309485009821345068724781056	3.231174267785263e-27	237,947,240,316,131,334,743,979,190,124,544	309485009821345068724781056.00x
618970019642690137449562112	1.6155871338926315e-27	475,894,480,632,262,669,487,958,380,249,088	618970019642690137449562112.00x
1237940039285380274899124224	8.077935669463157e-28	951,788,961,264,525,338,975,916,760,498,176	1237940039285380274899124224.00x
2475880078570760549798248448	4.0389678347315785e-28	1,903,577,922,529,050,677,951,833,520,996,352	2475880078570760549798248448.00x
4951760157141521099596496896	2.0194839173657892e-28	3,807,155,845,058,101,355,903,667,041,992,704	4951760157141521099596496896.00x
9903520314283042199192993792	1.0097419586828946e-28	7,614,311,690,116,202,711,807,334,083,985,408	9903520314283042199192993792.00x
19807040628566084398385987584	5.048709793414473e-29	15,228,623,380,232,405,423,614,668,167,970,816	19807040628566084398385987584.00x
39614081257132168796771975168	2.5243548967072365e-29	30,457,246,760,464,810,847,229,336,335,941,632	39614081257132168796771975168.00x
79228162514264337593543950336	1.2621774483536182e-29	60,914,493,520,929,621,694,458,672,671,883,264	79228162514264337593543950336.00x
158456325028528675187087900672	6.310887241768091e-30	121,828,987,041,859,243,388,917,345,343,766,528	158456325028528675187087900672.00x
316912650057057350374175801344	3.1554436208840455e-30	243,657,974,083,718,486,777,834,690,687,533,056	316912650057057350374175801344.00x
633825300114114700748351602688	1.5777218104420227e-30	487,315,948,167,436,973,555,669,381,375,066,112	633825300114114700748351602688.00x
1267650600228229401496703205376	7.888609052210113e-31	974,631,896,334,873,947,111,338,762,750,132,224	1267650600228229401496703205376.00x
2535301200456458802993406410752	3.9443045261050565e-31	1,949,263,792,669,747,894,222,677,525,500,264,448	2535301200456458802993406410752.00x
5070602400912917605986812821504	1.9721522630525282e-31	3,898,527,585,339,495,788,445,355,051,000,528,896	5070602400912917605986812821504.00x
10141204801825835211973625643008	9.860761315262641e-32	7,797,055,170,678,991,576,890,710,102,001,057,792	10141204801825835211973625643008.00x
20282409603651670423947251286016	4.9303806576313205e-32	15,594,110,341,357,983,153,781,420,204,002,115,584	20282409603651670423947251286016.00x
40564819207303340847894502572032	2.4651903288156602e-32	31,188,220,682,715,966,307,562,840,408,004,231,168	40564819207303340847894502572032.00x
81129638414606681695789005144064	1.2325951644078301e-32	62,376,441,365,431,932,615,125,680,816,008,462,336	81129638414606681695789005144064.00x
162259276829213363391578010288128	6.1629758220391505e-33	124,752,882,730,863,865,230,251,361,632,016,924,672	162259276829213363391578010288128.00x
324518553658426726783156020576256	3.0814879110195752e-33	249,505,765,461,727,730,460,502,723,264,033,849,344	324518553658426726783156020576256.00x
649037107316853453566312041152512	1.5407439555097876e-33	499,011,530,923,455,460	

- Thread efficiency decreases with more threads
- Optimal thread count: 8 (tested up to 8)

5.2 Performance Analysis

Unexpected Results:

Python serial

Scaling Behavior:

Multi-threaded

- 2 threads: 0.19x speedup (expected ~0.36x for linear)
- 4 threads: 0.29x speedup (expected ~0.72x for linear)
- 8 threads: 0.46x speedup (expected ~1.44x for linear)

This suggests overhead from thread synchronization (mutexes), workload partitioning inefficiencies, and memory bandwidth limitations.

5.3 Key Takeaways

- Overhead Matters: For small workloads, interop overhead can outweigh performance benefits

- Python is faster

5.4 Trade-offs and Challenges

Challenges Encountered:

1. Cross-platform compatibility: Building C backends on Windows required MSYS2 and OpenSSL setup
2. DLL loading: Ensuring C libraries found at runtime
3. Thread safety: Proper synchronization to avoid race conditions
4. Performance overhead: ctypes interop overhead limits C backend benefits

Trade-offs:

- Simplicity

5.5 Conclusion

This project demonstrates that performance characteristics can be counterintuitive. The Python backend, despite being "slower" in theory, outperformed C implementations due to optimized libraries and minimal interop overhead. Multi-threading provides benefits, but synchronization overhead limits scaling efficiency. The benchmarking framework developed provides a foundation for future performance analysis and optimization work.

Practical Implications:

- Python backend is sufficient for small to medium workloads
- C backends may be beneficial for very large workloads where overhead is amortized
- Multi-threading provides modest improvements but may not justify complexity
- GPU implementation could provide significant speedup (future work)