1 Implementation Overview

This assignment implements a parallelized Bitcoin proof-of-work miner using CUDA. The miner searches for a nonce value that, when hashed with the block header using double-SHA256, produces a hash below a target difficulty threshold.

Architecture

- **Host (CPU):** Block parsing, merkle root calculation, result formatting
- **Device (GPU):** Parallel nonce search using optimized SHA256

2 Parallelization and Optimization Techniques

2.1 SHA256 Midstate Pre-computation

Bitcoin block headers are 80 bytes long, but only the final 4 bytes (nonce) change. SHA256 operates on 64-byte chunks, meaning the first 64 bytes can be pre-hashed once on CPU. The resulting midstate is reused for every GPU thread.

```
sha256_compute_midstate(first64, midstate);
sha256_finalize_from_midstate(&ctx1, d_midstate, last16);
```

Benefits include an 80% reduction in hash work and massive reuse of CPU precomputation.

2.2 Circular W-Array in SHA256

The standard SHA256 algorithm uses a 64-word schedule array, which consumes large per-thread memory. Replacing it with a 16-entry circular buffer reduces local memory usage by 75%.

```
WORD w[16];
w[i & 15] = w_i_16 + s0 + w_i_7 + s1;
```

2.3 Specialized `sha256_32bytes()`

Double-SHA256's second hash always processes exactly 32 bytes. A custom version removes general-purpose overhead and branches.

```
block[32] = 0x80;
memset(block + 33, 0, 23);
block[62] = 1; block[63] = 0;
```

2.4 Register Caching

Repeated reads of global memory slow down SHA256. Caching values into registers eliminates repeated global memory loads.

2.5 Batched Atomic Operations

Instead of checking the global flag every iteration, threads check every 256 iterations, reducing contention massively.

```
if ((local_batch & 0xFF) == 0 && d_found) return;
```

2.6 Maximum Thread Utilization

Using 512 threads/block and 65535 blocks maximizes GPU parallelism without register spilling.

3 Experiments

Methodology

- Test platform: Tesla V100-SXM2-32GB
- Test case: public testcase 01
- Metric: Kernel execution time (1 run)

Config	Blocks	Threads	Total Threads	Time (s)
1	256	128	32768	15.36
2	512	128	65536	10.33
3	1024	256	262144	9.64
4	4096	256	1,048,576	5.07
5	16384	256	4,194,304	5.46
6	32768	256	8,388,608	6.03
7	32768	512	16,777,216	3.56
8	65535	512	33,553,920	3.58

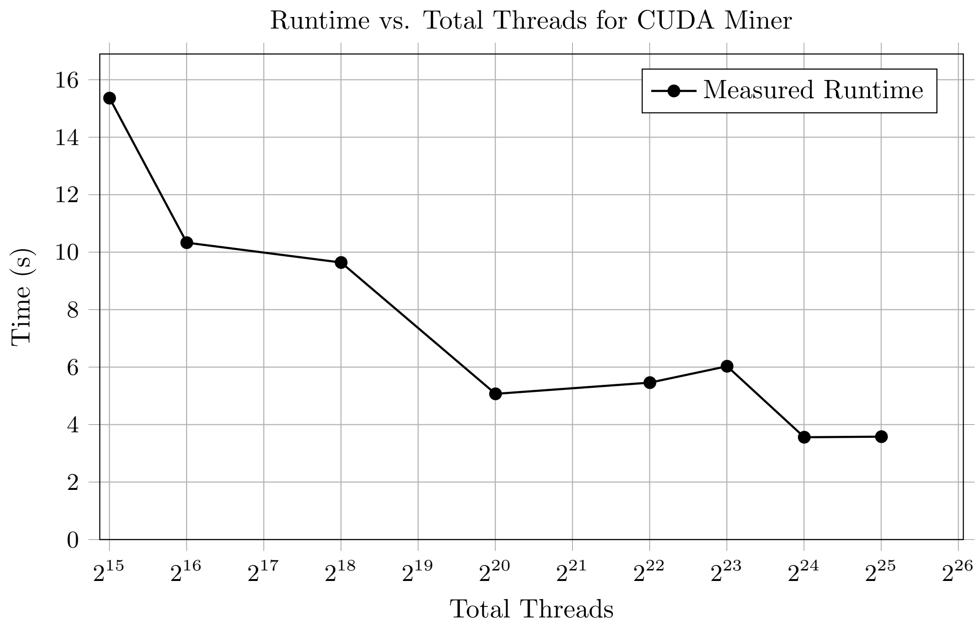

Figure 1: Runtime of CUDA Bitcoin Miner under different block/thread configurations. The x-axis is plotted in log scale to reveal scaling trends.

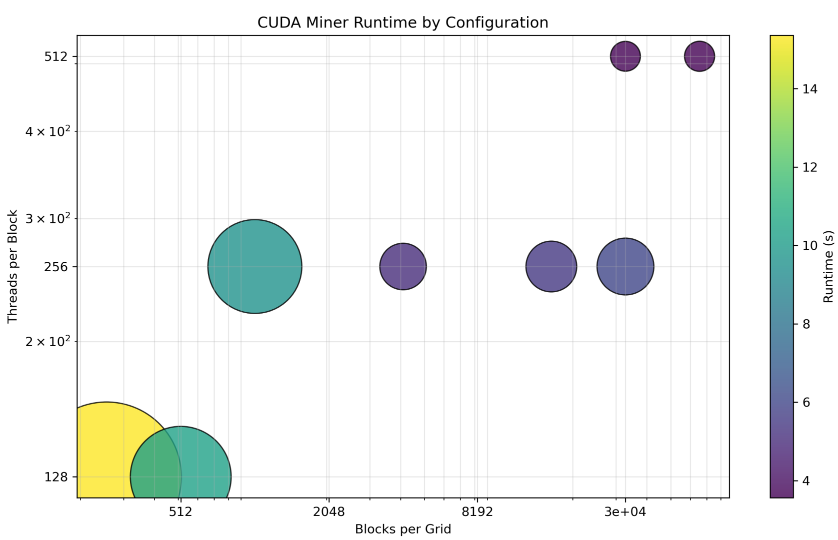

Figure 2: CUDA Miner Runtime for Different Configurations. Bubble size represents relative runtime, color corresponds to runtime in seconds. X-axis: Blocks per Grid, Y-axis: Threads per Block (both powers-of-2).

4 Other Details of CUDA programming

4.1 Constant Memory

```
__constant__ unsigned char d_target[32];
__constant__ WORD d_midstate[8];
```

4.2 Grid-Stride Loops

```
for (unsigned long long nonce = tid;
     nonce < max_nonce_space;
     nonce += total_threads) {
    ...
}
```

4.3 Atomic Compare-And-Swap

```
if (atomicCAS(&d_found, 0, 1) == 0) {
    d_solution_nonce = (unsigned int) nonce;
}
```