PDF 2.0 CRACKING on CUDA

Hyun-Jun Kim, Si-Woo Eum, Hwa-Jeong Seo

Passward Cracking on GPU

- Modern graphics processors provide high processing power
- With the breakthrough performance of GPUs, research on cryptographic processing is actively underway.
- Password cracking is the operation of recovering passwords from data stored or transmitted on a computer system
  - Recover user's forgotten password
  - Gaining unauthorized access to the system
  - access digital evidence
- Encrypted files contain a hash value to verify the correctness of the password.
- The attacker attempts a brute force attack and compares the guessed password with the hash value inside the file.
- Hashcat and John the ripper are popular tools.

PDF(Portable Document Format)

- PDF (Portable Document Format) is a file format developed by Adobe in 1992.
- Standardized to ISO 32000 and used worldwide
- All PDF versions are all standard and backward compatible
- Mainly used PDF files may be subject to password cracking

- Latest PDF version cracking goal
- In this paper, PDF 2.0 version cracking algorithm is optimized for CUDA GPU.

PDF 2.0

- **PDF 1.7 Level 8 – PDF 2.0** target
  - Use multiple algorithms
  - AES, SHA-256, SHA-384, SHA-512

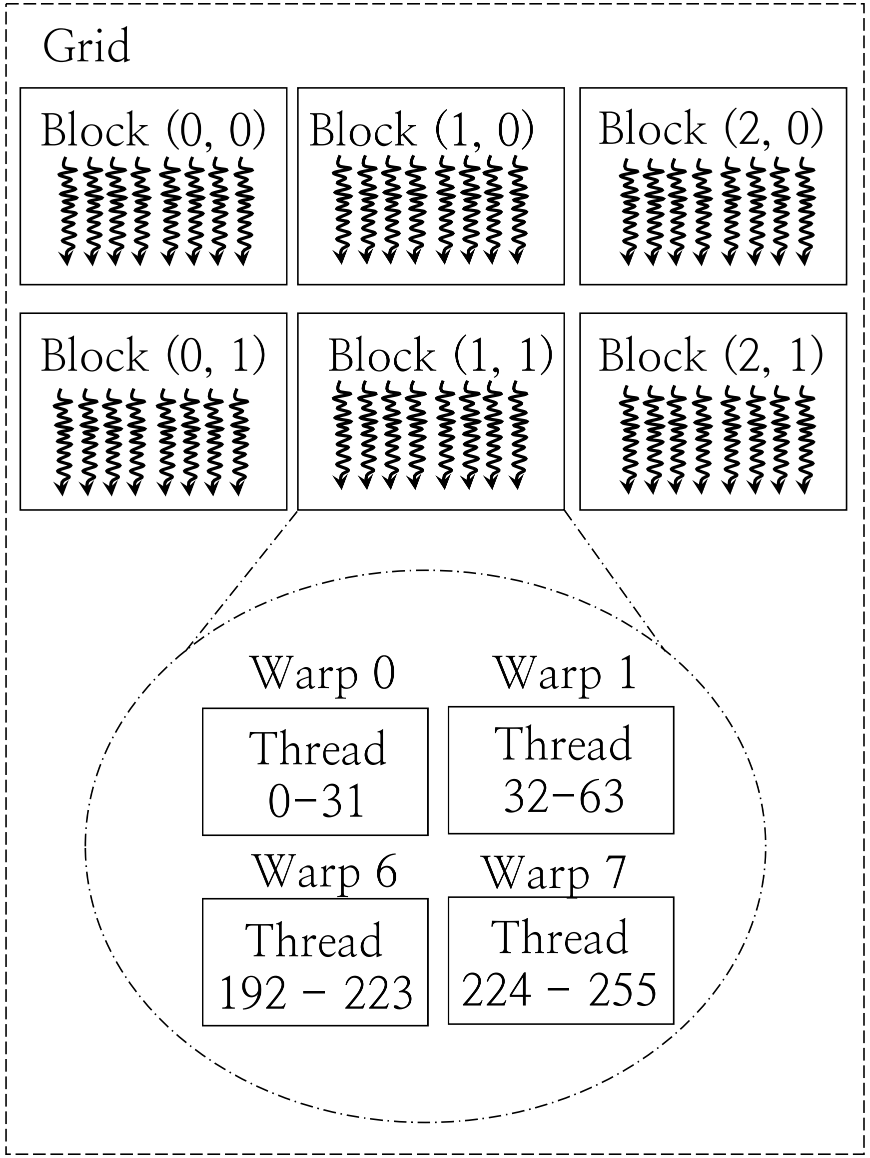
PDF Version	Acrobat Version	Password Algorithm
PDF 1.1 – 1.3	Acrobat 2 – 4	MD5:50 – RC4-40
PDF 1.4 – 1.7 R4	Acrobat 5 – 8	MD5:50 – RC4-128:20
PDF 1.7 R5	Acrobat 9	SHA256
PDF 1.7 Level 8 – PDF 2.0	Acrobat 10 - 11	AES, SHA-256, SHA-384, SHA-512

CUDA

- CUDA (Compute Unified Device Architecture) is a GPGPU technology that enables parallel processing algorithms performed on GPUs.
- CUDA GPU Configuration
    - Includes Function Kernel
    - Thread Group Block
    - Block Group Grid
    - 32 Thread Bundle Warp
    - Streaming Multi-processor (SM)
- Execute multiple passwords in parallel by trying to crack one password in one thread.
- Hashcat implementation reference

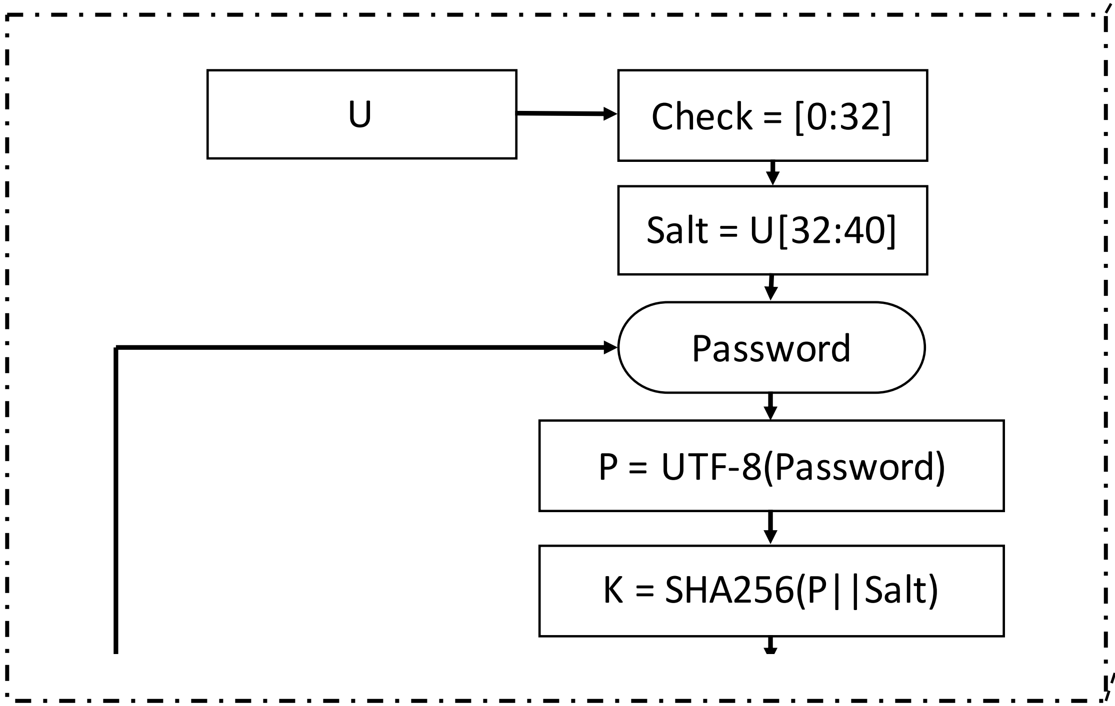
hashcat
advanced password recovery

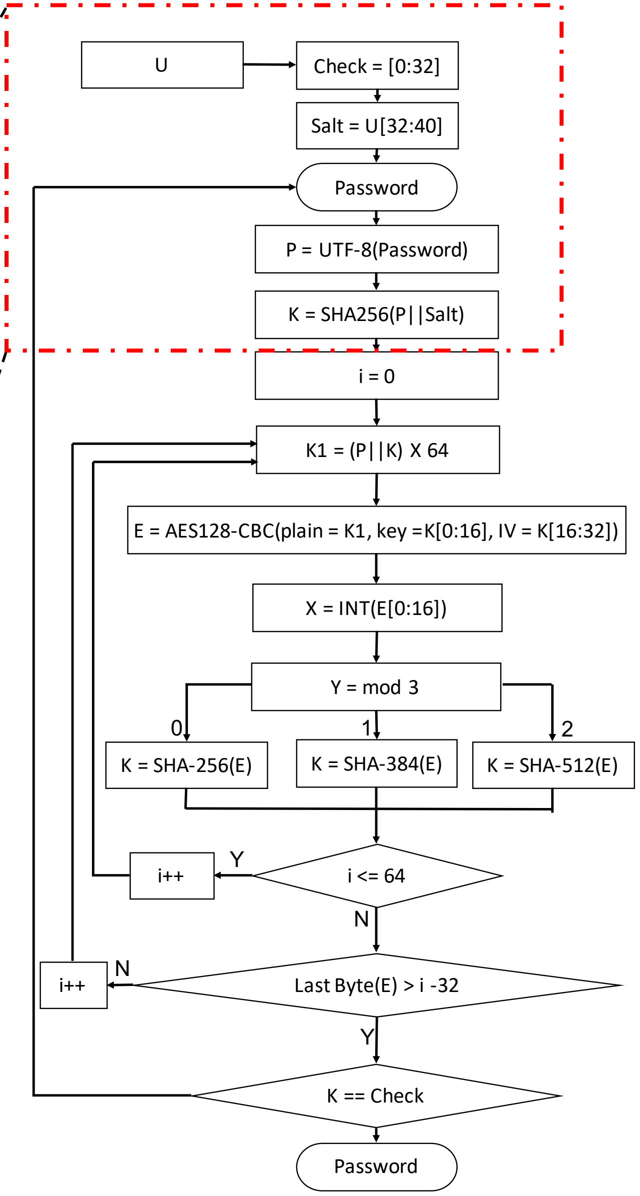
NVIDIA
CUDA



Password Cracking Algorithm

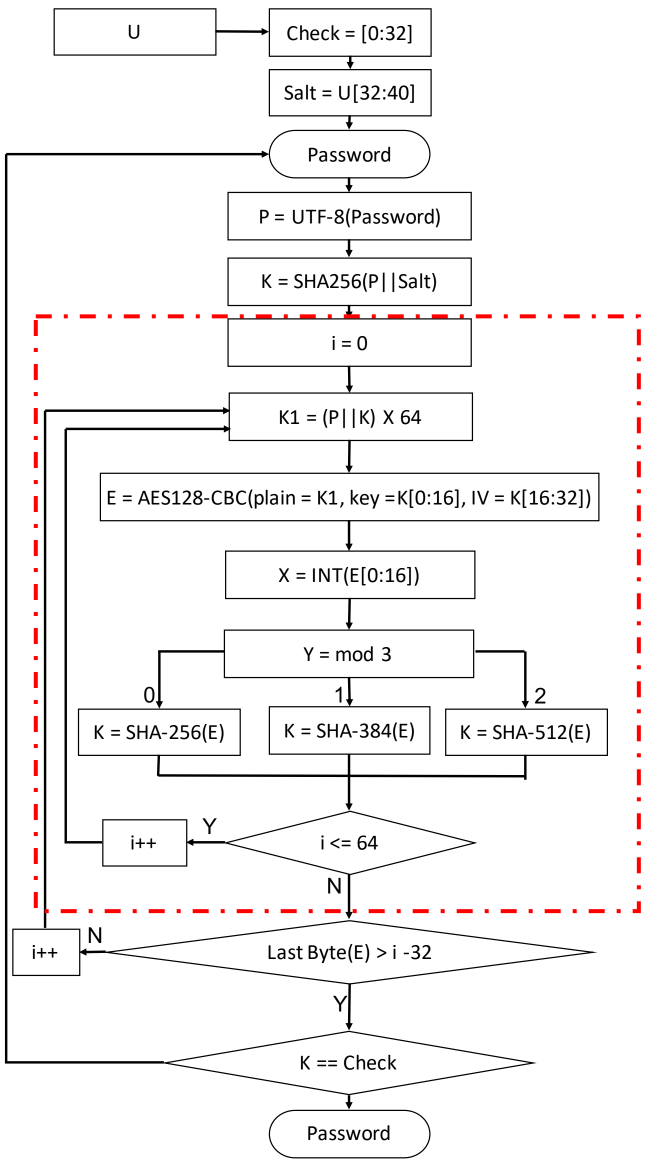
- Extract 48 bytes of U in the header of the PDF file
  - Check : 32 bytes of value U for password verification
  - Salt : Salt value used for hashing, 8 bytes of U
- P : Encode passwords as UTF-8
- K : Hash value obtained by hashing P||Salt with SHA256





Password Cracking Algorithm

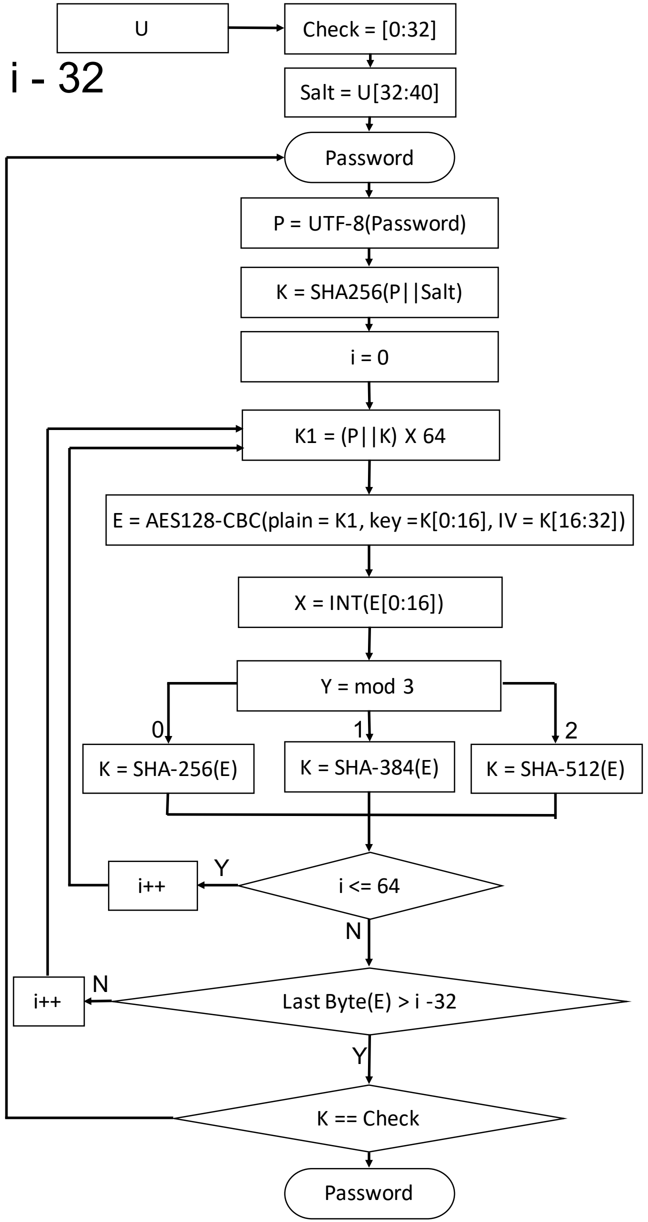
- Round 64 is repeated
- One of the SHA-2 algorithms and AES-128 CBC used in one round
- P||K is expanded to 64 and encrypted with AES-128 CBC
- 16 bytes of K are used as key values and the remaining 16 bytes are used as IV values.
- Next, the algorithm of SHA-2 works
- Modular operation of the first block of the ciphertext by 3.
- When 0, SHA-256, when 1, SHA-384, when 2, SHA-512 is selected.



Password Cracking Algorithm

- Rounds continue until the last byte of the last generated ciphertext E is greater than round i - 32
- Finally, compare with the validation value and return the password if they are the same
- If not, go back to the first step in the algorithm and enter another password.

- Characteristic
  - Change the execution time according to the entered password
  - Longer passwords increase the number of AES-128 operations.
  - Different SHA-2 algorithms are used depending on the ciphertext generated by AES128-CBC.
  - Perform additional rounds according to the last generated ciphertext E



PROPOSED

Shared memory : AES T table

- Memory space shared between threads within a block.
- Shared memory has higher bandwidth and lower latency than local and global memory
- Use of shared memory is suitable for Lookup-Table implementations using AES T tables
- Implementation without bank conflict resolution
- 23% performance improvement with shared memory compared to not using shared memory

```
__shared__ u32 s_te0[256];
__shared__ u32 s_te1[256];
__shared__ u32 s_te2[256];
__shared__ u32 s_te3[256];
__shared__ u32 s_te4[256];

for (u32 i = threadIdx.x; i < 256; i += blockDim.x)
   {
      s_te0[i] = te0[i];
      s_te1[i] = te1[i];
      s_te2[i] = te2[i];
      s_te3[i] = te3[i];
      s_te4[i] = te4[i];
   }

__syncthreads()
```

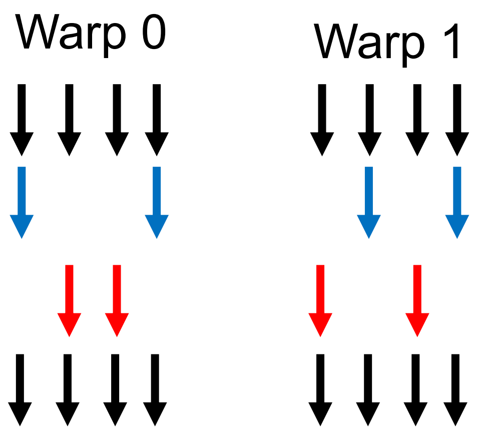
__syncthreads()

- syncthreads() is a barrier synchronization function
- Waits for all global and shared memory accesses performed by a thread to be visible to all threads in the block.
- The thread that called _syncthreads() is stopped at the calling location until all threads in the block have reached that location.
- Additional _syncthreads() calls are often required at the end of the loop if there are reads and writes from other threads to the shared memory location.
- The proposed technique is to add _syncthreads() after storing the T-table in shared memory.

__syncthreads()

- Add __syncthreads() to the iteration part of the cracking algorithm.
- Cannot execute two different instructions at the same time in warp (in SIMD registers)
- In the branch, the hardware calculates as shown
- Poor performance because it has to go through both A and B
- divergence occurs

```
if(c) {
        // A
}
else {
        // B
}
```



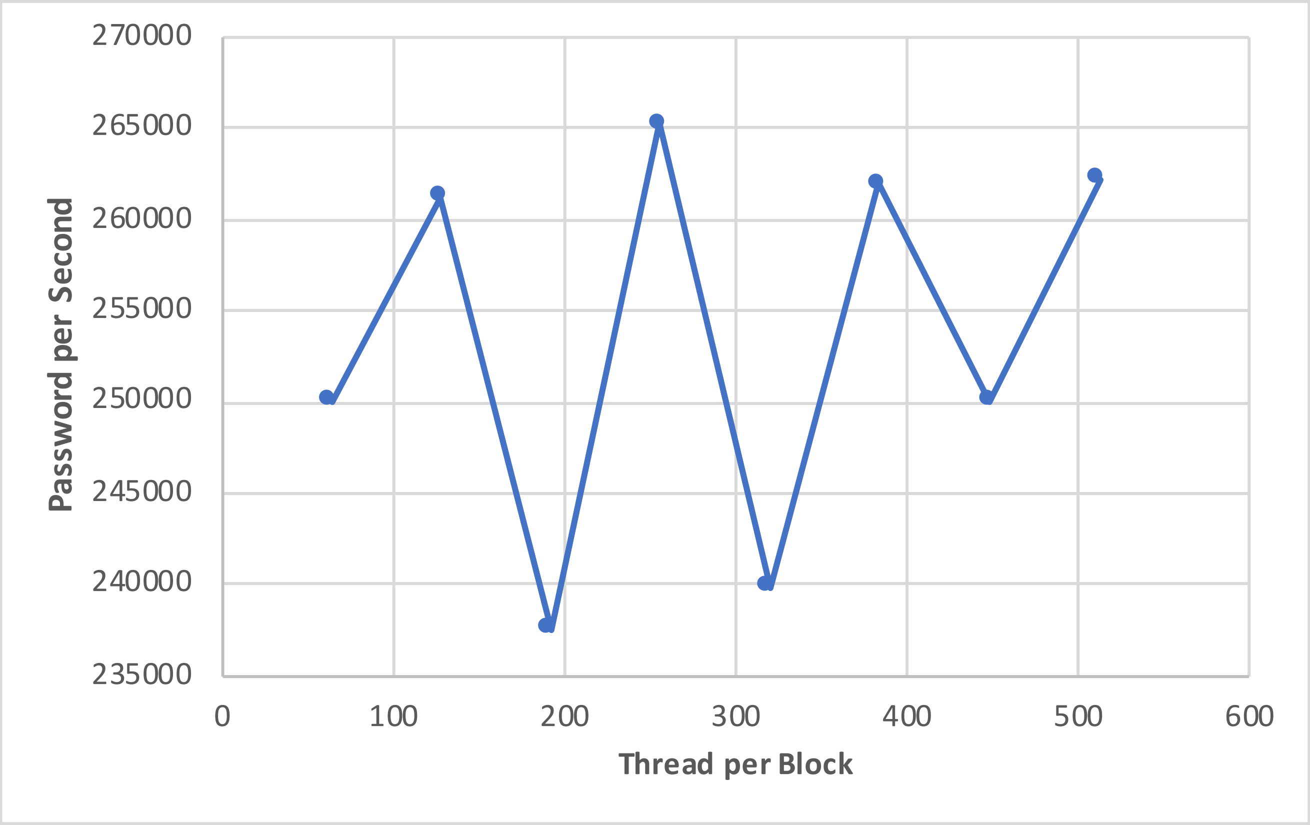
- The PDF 2.0 version of the cracking algorithm chooses the SHA-2 algorithm or performs additional iterations depending on the input and intermediate state values.
- Divergence occurs and consequently the performance of the program decreases.
- Cracking Algorithm Divergence Control Impossible
- Added _syncthreads() to the iteration section to clarify before and after branching as we need to do the same thing in Warp's threads before and after branching

Threads per block and blocks per grid

- Threads per block and blocks per grid affect performance
- Therefore, proper selection is necessary to achieve optimal performance.
- We check the performance change according to the number of threads per block and the number of blocks per grid.
- Measuring performance by applying the proposed technique in the RTX 3060 environment
- Since the execution time of the algorithm may vary depending on the input, the same password value was entered.

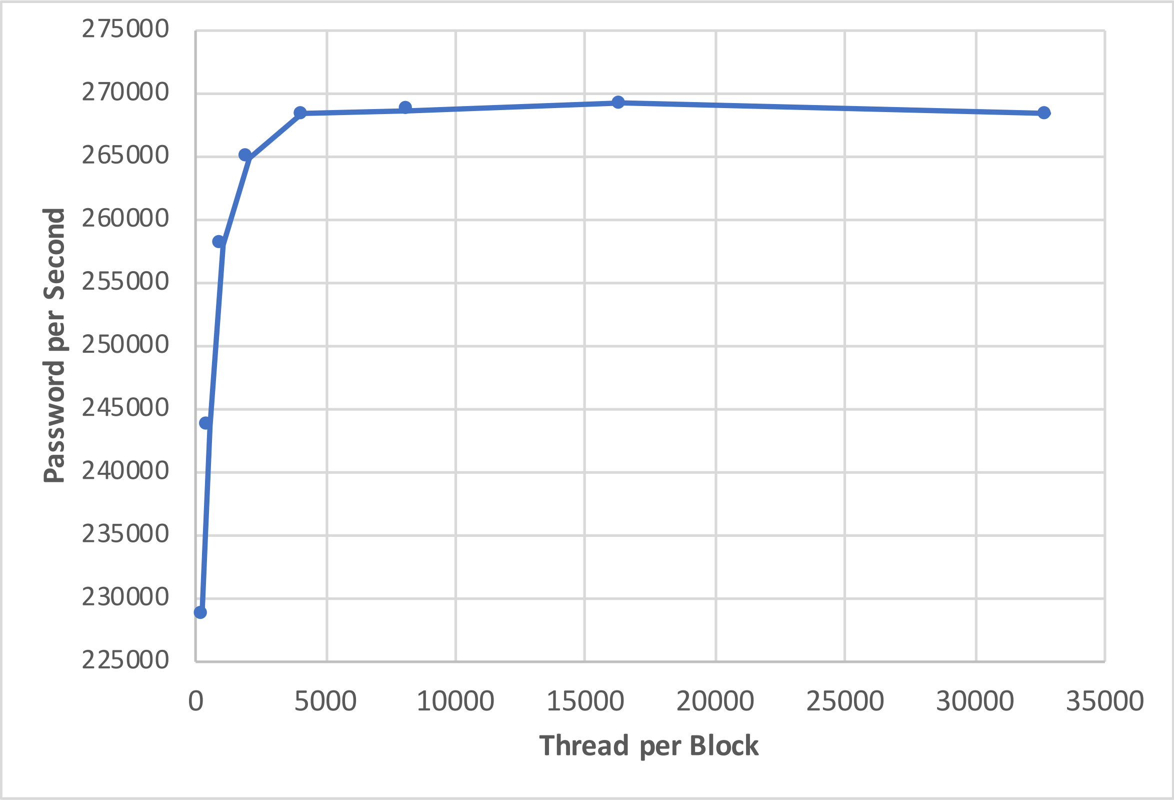
Comparison of the number of threads per block

- Measure the number of operations per second by fixing the block size and adding the thread size by 64
- Performance is erratic depending on thread size



Comparison of number of blocks per grid

- The number of operations per second was measured by fixing the thread size and increasing the block size by doubling from 256.
- It performed better with a larger block size and becomes constant as the value increases to some extent.



Experiment

- Comparison of performance gains when using shared memory and the main optimization technique, sysncthreads()
- As a result, the RTX 3060 has the best performance at 4096 blocks per grid and 512 threads per block.

TABLE I
THE NUMBER OF PASSWORD CRACKING PER SECOND AND PERFORMANCE DIFFERENCE ACCORDING TO OPTIMIZATION

Optimization	**Speed**	**Improvement**
None	51,241 p/s	0%
shared memory	63,030 p/s	23%
__sysncthreads()	66,377 p/s	28%
shared memory +__sysncthreads()	76,405 p/s	49%

CONCLUSION

- This paper optimizes the PDF 2.0 version decryption algorithm on CUDA GPU.
- It takes full advantage of GPU optimizations by applying the best implementation techniques using shared memory and sysncthreads().
- 49% better performance than before optimization on RTX 3060
- In future research, we implement performance comparison in more diverse environments and optimization of other algorithms.

Q & A