EURO 4SEE

GPU Assisted Brute Force Cryptanalysis of GPRS, GSM, RFID, and TETRA

Cihangir Tezcan, PhD
Graduate School of Informatics, METU, Ankara

ncc@ulakbim.gov.tr

Lesson 6: GPU Implementation Techniques for Ciphers

GPU Implementation Techniques for Ciphers

- There are generally three ways to implement symmetric key encryption algorithms on GPUs and depending on the design of the cipher, some methods provide better performance.
- These are naive/straightforward, table-based, and bitsliced implementations:
- For a captured plaintext and corresponding ciphertext, an exhaustive search attack encrypts the plaintext with every key that is possible and check whether the desired ciphertext is obtained.
- For a block cipher, it is enough to capture a single plaintext block and corresponding ciphertext block.
- The same attack can be performed on the ciphertext block by performing decryption instead of encryption.

Lesson 6: GPU Implementation Techniques for Ciphers

GPU Implementation Techniques for Ciphers

Exhaustive search attack implementation of a symmetric key encryption algorithm on a GPU can be categorized as follows:

1. ***Straightforward***
2. ***Table-based***
3. ***Bitsliced***
    1. ***Thread-level bitslicing***
    2. ***Variable-level bitslicing***

Lesson 6: GPU Implementation Techniques for Ciphers

1. Straightforward Implementation Technique

- In straightforward implementations every operation of the encryption algorithm is implemented as they are.
- But it is generally not possible to parallelize an encryption algorithm to use all of the GPU cores since modern GPUs have thousands of cores.
- Thus, using each thread to perform a different encryption operation is preferred in this technique.
- In a brute force attack, each thread can try a different key in a subspace of the keyspace that is assigned to it.
- Similarly, for a block cipher mode of operation that offers parallelism like counter mode, each thread can encrypt a different block of the plaintext.
- Such an approach does not require communication between threads and thus avoids many memory operations.
- Since straightforward implementation requires every operation of the cipher to be performed, the main bottleneck comes from the number of instructions that are performed.

Lesson 6: GPU Implementation Techniques for Ciphers

2. Table-Based Implementation Technique

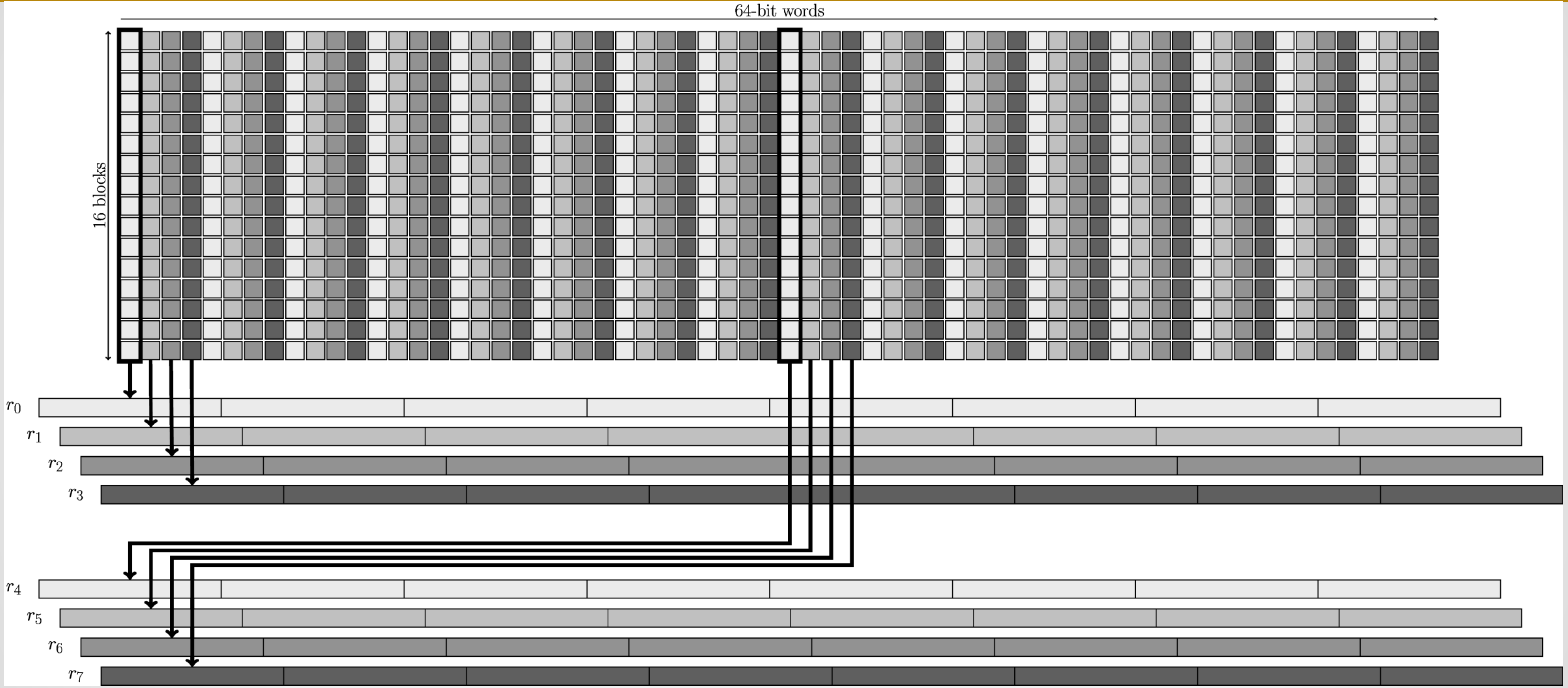
- Intermediate layers of some ciphers can be partitioned into disjoint subspaces so that every output for every possible input of these subspaces can be precomputed and stored in a table.
- This way performing those layers can be reduced to table look-up operations for the inputs of these subspaces and combining the values obtained from the tables.
- Such tables are generally too large to be stored in GPU registers and storing them in the global memory of the GPU causes a lot of delays since reading and writing to the global memory is slower.
- Thus, storing the tables in the shared memory provides the best performance.
- Shared memory bank conflicts are generally the main bottleneck in this approach.
- Moreover, storing very large tables in the shared memory might reduce the occupancy of the GPU or might even not be possible due to the device limits.

Lesson 6: GPU Implementation Techniques for Ciphers

3. Bitsliced Implementation Technique

- Programming languages and processors generally work on words that are multiples of 8 bits and bit operations can become more costly compared to hardware.
- Thus, hardware-oriented ciphers might use bit-level operations intensively so that the straightforward implementation might become taxing in software.
- Instead of working on bits of large words, we can store each bit in a different register so that we can avoid operations that try to access and use bits of a word.

Lesson 6: GPU Implementation Techniques for Ciphers



Bit-slice packing image by Jérémy Jean

Lesson 6: GPU Implementation Techniques for Ciphers

3. Bitsliced Implementation Technique

Bitslicing technique can be implemented in two different approaches on a GPU:

1. **Thread-level bitslicing:**
   1. In this approach, each bit of a word is assigned to a single thread in a GPU block.
   2. The optimizations in this approach focus on the choice of the memory types for transferring values between threads and the timing of memory operations.
   3. Since threads need to communicate to each other in this approach, the main bottleneck comes from the delays introduced by memory read and write operations.

Lesson 6: GPU Implementation Techniques for Ciphers

3. Bitsliced Implementation Technique

Bitslicing technique can be implemented in two different approaches on a GPU:

2. **Variable-level bitslicing:**
    1. In this approach, every thread stores each bit of a word in a register.
    2. The data sizes cause the main bottleneck in this approach because we need to use a register for every bit of stored value.
    3. For example, we need to use ***n*** registers for an ***n***-bit round key, ***n***-bit block of a block cipher, or ***n***-bit LFSR of a stream cipher.
    4. Thus, we can perform ***m*** encryptions in parallel at the thread-level when we use ***m***-bit registers to store bits of ***m*** different inputs.
    5. However, we lose the full occupancy of a modern GPU when we use more than 64 32-bit registers.

Lesson 6: GPU Implementation Techniques for Ciphers

GPU Optimization Techniques for Encryption Algorithms

- Note that straightforward, table-based, or bitsliced implementations are just implementation techniques
- We are going to consider many things to achieve the best optimization

Lesson 6: GPU Implementation Techniques for Ciphers

GPU Assisted Brute-Force Cryptanalysis of Ciphers

- In a brute-force attack, the attacker captures a single plaintext and its corresponding ciphertext.
- Attacker encrypts the plaintext with every possible key ($\boldsymbol{2^{k}}$ possibilities) to see if the resulting ciphertext matches the captured one.
- Thus, the key length $\boldsymbol{k}$ and the number of key trials that we can perform in a second determines how much it would take to capture the key.

Next lecture

CUDA Optimization of KASUMI

EURO 4SEE

Thanks!

Co-funded by
the European Union

EuroHPC
Joint Undertaking

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101191697. The JU receives support from the Digital Europe Programme and Germany, Türkiye, Republic of North Macedonia, Montenegro, Serbia, Bosnia and Herzegovina.