Brute Force Attack on Block Ciphers Using OpenCL

Wookeun Jung, Gangwon Jo, and Jaejin Lee

Center for Manycore Programming

School of Computer Science and Engineering, Seoul National University, Seoul, 151-744, Korea

http://aces.snu.ac.kr

Introduction

- **Block cipher**
  - Block cipher is a widely used concept in cryptography
  - Used for file encryption in general
  - Divide a plaintext into multiple blocks
  - Encrypt the blocks using the same key
  - Data Encryption Standard (DES), 1975
  - Advanced Encryption Standard (AES), 2001
- **Brute Force Attack on Block cihper**
  - The goal is to find the key used for encryption
  - Exhaustive Search for the possible key
  - Straightforward, but time-consuming
  - Embarrassingly parallel workload
- **Accelerated brute force attack using OpenCL**
  - Distribute possible keys to OpenCL work items
  - Two target block ciphers
    - DES, AES
  - Three target accelerators
    - AMD GPU, Nvidia GPU, Intel Xeon Phi Coprocessor
  - **68.39X speedup on average** to sequential version
- **Comparison with other framework**
  - CUDA vs. OpenCL
  - MPI + Vector Intrinsic vs. OpenCL
- **Suggestion on OpenCL language feature**
  - Scatter/Gather extension

Target Block Ciphers

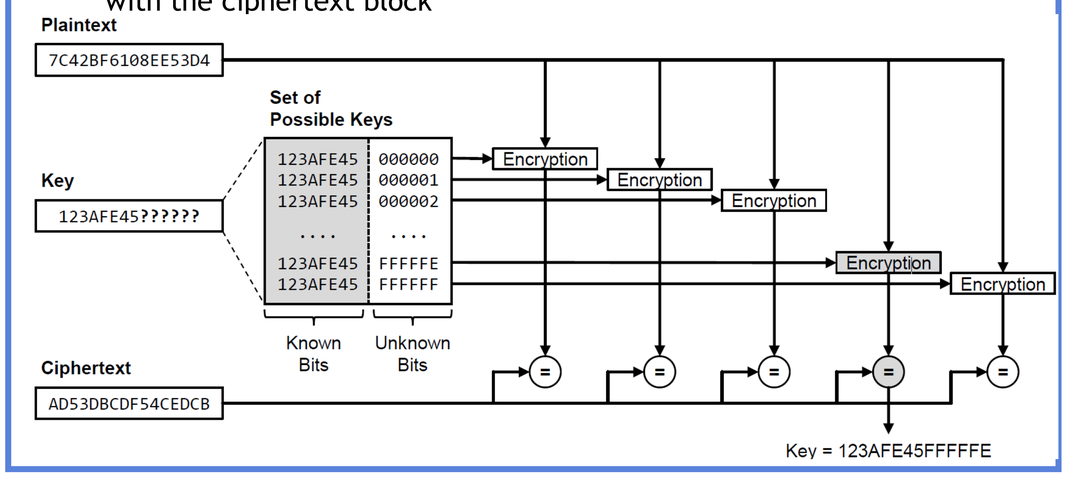
- **Data Encryption Standard**
  - Proposed in 1975
  - 56bit Key, 64bit block size
- **Advanced Encryption Standard**
  - Proposed in 2001
  - Larger block & Longer key length than DES
- **Bottleneck is substitution phases**
  - Called **S-Box access**
  - Random index array reference

	DES	AES
Key Length	56 bit	128 bit
Block Size	64 bit	128 bit
Bottleneck	S-Box access	

Specification of target block ciphers

Brute Force Attack on Block Cipher

- **Assumption on the attack**
  - A pair of plaintext / ciphertext block is assumed to be given
  - The key used for the encryption is unknown
  - The goal is to find the key used for the encryption
  - To make the problem size be in a reasonable scale, we assume that the subset of the input key is known
- **Approach**
  - Encrypt the plaintext block using every possible keys and compare it with the ciphertext block



OpenCL Implementation

- **Baseline implementation**
  - Port sequential brute force attack C code to OpenCL
  - Possible keys to check are evenly distributed to work-items
  - Each work-item encrypts the plaintext block using assigned keys sequentially one by one
  - If encrypted block is same as ciphertext, notify it to the host code through global memory
- **Optimization**
  - Work group size selection
    - For each accelerator, the best performing work group size is selected
    - Tuned by hand
  - Local memory utilization
    - The bottleneck is S-Box access latency
    - Put S-Box on OpenCL local memory to reduce the latency
      - Gains speedup on GPUs
      - Does not gain speedup on Intel Xeon Phi

```
//plaintext, ciphertext is given

for( i = 0; i < N; i++ )
{
  key = get_possible_key(i);
  temp = AES(plaintext, key);
  if( isEqual( temp, ciphertext ) )
    break;
    //i'th possible key is the answer
}
```

Sequential brute force attack code

```
__kernel void key_search(
  __constant unsigned char * plaintext,
  __constant unsigned char * ciphertext,
  __global int* result)
{
  const uint gid = get_global_id(0);
  const uint gsz = get_global_size(0);
  const uint start = (N * gid) / gsz;
  const uint end = (N * (gid + 1)) / gsz;
  for( i = start; i < end; i++ )
  {
    key = get_possible_key(i);
    temp = AES(plaintext, key);
    if( isEqual( temp, ciphertext ) )
      result[gid] =  i;
  }
}
```

OpenCL kernel code

Implementation in Other Programming Models

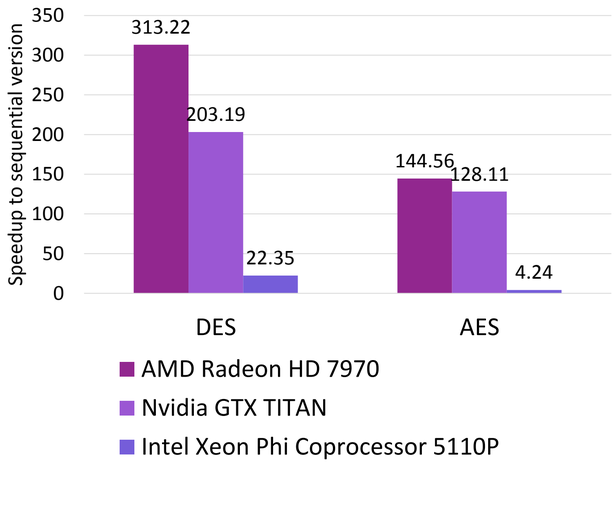
- **CUDA**
  - For Nvidia GPU
  - Implementation is very similar to OpenCL
  - Optimizations for OpenCL can be applied to CUDA in a similar manner
    - ex) Local memory -> Shared memory
- **MPI + Vector**
  - For Intel Xeon Phi Coprocessor
  - Distribute possible keys to MPI processes, and to SIMD lanes
  - Vectorize using Intel Compiler intrinsic
  - Use **Scatter/Gather instruction**
    - Key for AES/DES vectorization
      - S-Box reference is vectorized using gather instruction
    - OpenCL vector type does not support Scatter/Gather instruction

Experiment

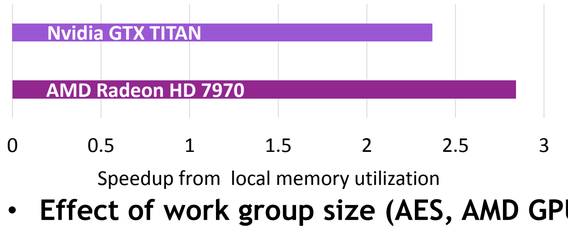
- **Experimental environment**

	Number of Cores	Core Clock Frequency
Intel CPU (E5-2650)	8 (only 1 core is used)	2.0 GHz
AMD GPU (HD Radeon 7970)	2048	925 MHz
Nvidia GPU (GeForce GTX TITAN)	2880	837 MHz
Intel Xeon Phi (Xeon Phi Coprocessor 5110P)	60	1053 MHz

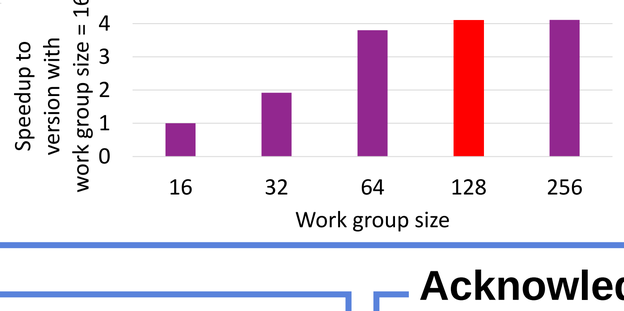
- **Optimized OpenCL Performance**



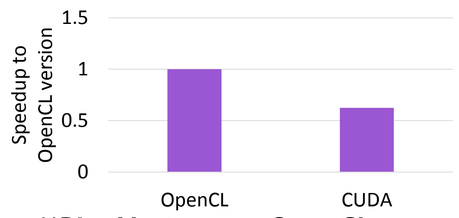
- **Effect of local memory utilization (AES)**



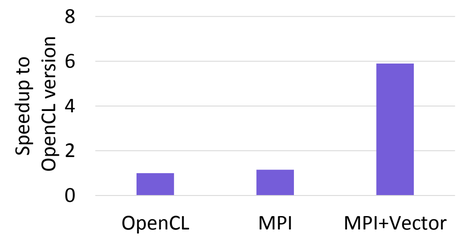
- **Effect of work group size (AES, AMD GPU)**



- **CUDA vs. OpenCL (AES, Nvidia GPU)**



- **MPI + Vector vs. OpenCL (AES, Intel Xeon Phi)**



Conclusion

- **Our OpenCL implementations achieved**
  - **69.39X** speedup to sequential version on average
    - **185.29X** speedup using GPU
    - **9.73X** speedup using Intel Xeon Phi
- **Our experimental result shows**
  - Importance of local memory utilization
- **OpenCL support for Intel Xeon Phi need to be improved**
  - Explicit approach
    - OpenCL built-in support for Scatter/Gather instruction
  - Implicit approach
    - Advanced auto-vectorization support in OpenCL Compiler

Acknowledgement

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No.2013R1A3A2003664). ICT at Seoul National University provided research facilities for this study.