Fast Implementations of AES on Various Platforms

Joppe W. Bos¹  Dag Arne Osvik¹  Deian Stefan²

¹EPFL IC IIF LACAL, Station 14, CH-1015 Lausanne, Switzerland
{joppe.bos, dagarne.osvik}@epfl.ch

²Dept. of Electrical Engineering, The Cooper Union, NY 10003, New York, USA
stefan@cooper.edu
Outline

- Introduction
  - Motivation
  - Previous Work
  - Contributions

- The Advanced Encryption Standard

- Target Platforms
  - The 8-bit AVR Microcontroller
  - The Cell Broadband Engine Architecture
  - The NVIDIA Graphics Processing Unit

- Conclusions
Motivation

Advanced Encryption Standard

- Rijndael announced in 2001 as the AES.
- One of the most widely used cryptographic primitives.
  - IP Security, Secure Shell, Truecrypt
  - RFID and low-power authentication methods
  - Key tokens, RF-based Remote Access Control
- Many intensive efforts to speed up AES in both hard- and software.

Related work


Contributions

New software speed records for various architectures

- 8-bit AVR microcontrollers
  - compact, efficient single stream AES version
- Synergistic processing elements of the Cell broadband engine
  - widely available in the PS3 video game console
  - single instruction multiple data (SIMD) architecture
  - process 16 streams in parallel (bytesliced)
- NVIDIA graphics processing unit
  - first AES decryption implementation
  - single instruction multiple threads (SIMT) architecture
  - process thousand of streams in parallel (T-table based)
The Advanced Encryption Standard

- Fixed block length version of the Rijndael block cipher
- Key-iterated block cipher with 128-bit state and block length
- Support for 128-, 192-, and 256-bit keys
- Strong security properties $\rightarrow$ no attacks on full AES-128
- Very efficient in hardware and software
Algorithm consists of 5 steps:

1. **Key expansion:**
   
   \[ 128\text{-bit} \rightarrow N_r + 1 = 11 \text{ 128-bit round keys} \]

2. **State initialization:**
   
   \[ \text{initial state} \leftarrow \text{plaintext block} \oplus \text{128-bit key} \]

3. **Round transformation:**
   
   apply round function on state \( N_r - 1 \) times

4. **Final round transformation:**
   
   apply the modified round function

Core of AES, the round function, consists of the following steps:

- SubBytes, ShiftRows, MixColumns, and AddRoundKey.
Algorithm consists of 5 steps:

1. **Key expansion:**
   \[ 128\text{-bit} \rightarrow N_r + 1 = 11 \text{ 128-bit round keys} \]
2. **State initialization:**
   \[ \text{initial state} \leftarrow \text{plaintext block} \oplus 128\text{-bit key} \]
3. **Round transformation:**
   \[ \text{apply round function on state } N_r - 1 \text{ times} \]
4. **Final round transformation:**
   \[ \text{apply the modified round function} \]

Core of AES, the round function, consists of the following steps:

- SubBytes, ShiftRows, MixColumns, and AddRoundKey.

Decryption follows the same procedure
- round function steps are the inverse and run in reverse order
Round Function Steps

1. **SubBytes:**

   \[
   \begin{array}{cccc}
   a_{00} & a_{01} & a_{12} & a_{03} \\
   a_{10} & a_{11} & a_{12} & a_{13} \\
   a_{20} & a_{21} & a_{22} & a_{23} \\
   a_{30} & a_{31} & a_{32} & a_{33} \\
   \end{array}
   \rightarrow
   \begin{array}{cccc}
   S-box \\
   \end{array}
   \rightarrow
   \begin{array}{cccc}
   b_{00} & b_{01} & b_{12} & b_{03} \\
   b_{10} & b_{11} & b_{12} & b_{13} \\
   b_{20} & b_{21} & b_{22} & b_{23} \\
   b_{30} & b_{31} & b_{32} & b_{33} \\
   \end{array}
   \]
Round Function Steps

1. **SubBytes:**

   ![SubBytes Diagram]

   - Input: \( \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \)
   - Output: \( \begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \\ b_{30} & b_{31} & b_{32} & b_{33} \end{bmatrix} \)

2. **ShiftRows:**

   ![ShiftRows Diagram]

   - Input: \( \begin{bmatrix} a_{00} & a_{02} & a_{03} \\ a_{10} & a_{12} & a_{13} \\ a_{20} & a_{22} & a_{23} \\ a_{30} & a_{32} & a_{33} \end{bmatrix} \)
   - Output: \( \begin{bmatrix} a_{00} & a_{03} & a_{02} \\ a_{11} & a_{13} & a_{12} \\ a_{22} & a_{23} & a_{20} \\ a_{33} & a_{30} & a_{31} \end{bmatrix} \)
### Round Function Steps

1. **SubBytes:**

   ![SubBytes Diagram]

   - $S-box$

2. **ShiftRows:**

   ![ShiftRows Diagram]

3. **MixColumns:**

   ![MixColumns Diagram]
Round Function Steps

1. **SubBytes:**

   \[
   \begin{array}{cccc}
   a_{00} & a_{01} & a_{02} & a_{03} \\
   a_{10} & a_{11} & a_{12} & a_{13} \\
   a_{20} & a_{21} & a_{22} & a_{23} \\
   a_{30} & a_{31} & a_{32} & a_{33} \\
   \end{array}
   \rightarrow
   \begin{array}{cccc}
   b_{00} & b_{01} & b_{02} & b_{03} \\
   b_{10} & b_{11} & b_{12} & b_{13} \\
   b_{20} & b_{21} & b_{22} & b_{23} \\
   b_{30} & b_{31} & b_{32} & b_{33} \\
   \end{array}
   \]

2. **ShiftRows:**

   \[
   \begin{array}{cccc}
   a_{00} & a_{01} & a_{02} & a_{03} \\
   a_{10} & a_{11} & a_{12} & a_{13} \\
   a_{20} & a_{21} & a_{22} & a_{23} \\
   a_{30} & a_{31} & a_{32} & a_{33} \\
   \end{array}
   \rightarrow
   \begin{array}{cccc}
   a_{00} & a_{10} & a_{20} & a_{30} \\
   a_{01} & a_{11} & a_{21} & a_{31} \\
   a_{02} & a_{12} & a_{22} & a_{32} \\
   a_{03} & a_{13} & a_{23} & a_{33} \\
   \end{array}
   \]

3. **MixColumns:**

   \[
   \begin{array}{cccc}
   a_{00} & a_{01} & a_{02} & a_{03} \\
   a_{10} & a_{11} & a_{12} & a_{13} \\
   a_{20} & a_{21} & a_{22} & a_{23} \\
   a_{30} & a_{31} & a_{32} & a_{33} \\
   \end{array}
   \times
   \begin{array}{cccc}
   2 & 3 & 1 & 1 \\
   1 & 2 & 3 & 1 \\
   1 & 1 & 2 & 3 \\
   3 & 1 & 1 & 2 \\
   \end{array}
   \rightarrow
   \begin{array}{cccc}
   b_{00} & b_{01} & b_{02} & b_{03} \\
   b_{10} & b_{11} & b_{12} & b_{13} \\
   b_{20} & b_{21} & b_{22} & b_{23} \\
   b_{30} & b_{31} & b_{32} & b_{33} \\
   \end{array}
   \]

4. **AddRoundKey:**

   \[
   \begin{array}{cccc}
   a_{00} & a_{01} & a_{02} & a_{03} \\
   a_{10} & a_{11} & a_{12} & a_{13} \\
   a_{20} & a_{21} & a_{22} & a_{23} \\
   a_{30} & a_{31} & a_{32} & a_{33} \\
   \end{array}
   \oplus
   \begin{array}{cccc}
   k_{00} & a_{01} & k_{02} & k_{03} \\
   k_{10} & a_{11} & k_{12} & k_{13} \\
   k_{20} & a_{21} & k_{22} & k_{23} \\
   k_{30} & a_{31} & k_{32} & k_{33} \\
   \end{array}
   \rightarrow
   \begin{array}{cccc}
   b_{00} & b_{01} & b_{02} & b_{03} \\
   b_{10} & b_{11} & b_{12} & b_{13} \\
   b_{20} & b_{21} & b_{22} & b_{23} \\
   b_{30} & b_{31} & b_{32} & b_{33} \\
   \end{array}
   \]
Target Platforms

Cell B.E
NVIDIA GPUs
Atmel AVRs

© IBM Systems
Advanced Virtual RISC Architecture

- Modified Harvard architecture
- 32 · 8-bit registers
- 16-bit pointer registers
- Registers are addressable
- Mostly single-cycle execution
- $\frac{1}{2}$ KB to 384KB flash memory
- 0 to 32KB SRAM
- 0 to 4KB EEPROM
Cell Broadband Engine Architecture

- Use the Synergistic Processing Elements
  - runs at 3.2 GHz
  - 128-bit wide SIMD-architecture
  - two instructions per clock cycle (dual pipeline)
  - in-order processor
  - rich instruction set: i.e.
    all distinct binary operations
    \[ f : \{0, 1\}^2 \rightarrow \{0, 1\} \] are present.

- “Expensive” QS22 Blade Servers (2 × 8 SPEs)
- “Cheap” PS3 video game console (6 SPEs)
SPU Results Comparison

Throughput per PS3: **13.2** (encryption) and **10.8 Gbps** (decryption)

Work-in-progress, fill both pipelines

Current version: 1752 odd and 2764 even instructions for encryption.
NVIDIA Graphic Processing Units

- Contain 12-30 *simultaneous multiprocessors* (SMs):
  - 8 streaming processors (SPs)
  - 16KB 16-way banked *fast* shared memory
  - 8192/16384 32-bit registers
  - 8KB constant memory cache
  - 6KB-8KB texture cache
  - 2 special function units
  - instruction fetch and scheduling unit

- **GeForce 8800GTX:**
  - 16 SMs @ 1.35GHz

- **GTX 295:**
  - $2 \times 30$ SMs @ 1.24GHz
AES GPU Implementation

Combine SubBytes, ShiftRows, MixColumns using the standard “T-table” approach. Update each column \((0 \leq j \leq 3)\):

\[
[s_{j0}, s_{j1}, s_{j2}, s_{j3}]^T = T_0[a_{c_00}] \oplus T_1[a_{c_11}] \oplus T_2[a_{c_22}] \oplus T_3[a_{c_33}] \oplus k_j,
\]

where each \(T_i\) is 1KB and \(k_j\) is the \(j\)th column of the round key.
Combine SubBytes, ShiftRows, MixColumns using the standard “T-table” approach. Update each column \(0 \leq j \leq 3\):

\[
[s_{j0}, s_{j1}, s_{j2}, s_{j3}]^T = T_0[a_{c00}] \oplus T_1[a_{c11}] \oplus T_2[a_{c22}] \oplus T_3[a_{c33}] \oplus k_j,
\]

where each \(T_i\) is 1KB and \(k_j\) is the \(j\)th column of the round key.

Example \((j = 0)\):

```

\begin{array}{cccc}
  a_{00} & a_{01} & a_{02} & a_{03} \\
  a_{10} & a_{11} & a_{12} & a_{13} \\
  a_{20} & a_{21} & a_{22} & a_{23} \\
  a_{30} & a_{31} & a_{32} & a_{33} \\
\end{array}

\begin{array}{c}
  T_0 \\
  T_1 \\
  T_2 \\
  T_3 \\
\end{array}

\begin{array}{c}
  b_{00} \\
  b_{10} \\
  b_{20} \\
  b_{30} \\
\end{array}

\begin{array}{c}
  k_{00} \\
  k_{10} \\
  k_{20} \\
  k_{30} \\
\end{array}

\begin{array}{cccc}
  s_{00} & s_{01} & s_{02} & s_{03} \\
  s_{10} & s_{11} & s_{12} & s_{13} \\
  s_{20} & s_{21} & s_{22} & s_{23} \\
  s_{30} & s_{31} & s_{32} & s_{33} \\
\end{array}
```
AES GPU Implementation

- Combine SubBytes, ShiftRows, MixColumns using the standard “T-table” approach. Update each column \((0 \leq j \leq 3)\):

\[
[s_{j0}, s_{j1}, s_{j2}, s_{j3}]^T = T_0[a_{c00}] \oplus T_1[a_{c11}] \oplus T_2[a_{c22}] \oplus T_3[a_{c33}] \oplus k_j,
\]

where each \(T_i\) is 1KB and \(k_j\) is the \(j\)th column of the round key.

- Example \((j = 0)\):

- Optimization approach: launch thread blocks containing multiple independent groups of 16 \((1/2\text{-warp})\) streams.
Key expansion:

On-the-fly:

- allows thousands of independent streams
- speed dependent on $T$-access speed
- multi-block speed improvement: cache few round keys / stream in shared memory; 16-streams/group → no bank conflicts!
Key expansion:

1. On-the-fly:
   - allows thousands of independent streams
   - speed dependent on $T$-access speed
   - multi-block speed improvement: cache few round keys / stream in shared memory; 16-streams/group → no bank conflicts!

2. Texture memory:
   - keys alive between kernel launches: multi-block encryption is faster than on-the-fly!
   - thread count limited by texture cache size
Key expansion:

1. On-the-fly:
   - allows thousands of independent streams
   - speed dependent on $T$-access speed
   - multi-block speed improvement: cache few round keys / stream in shared memory; 16-streams/group → no bank conflicts!

2. Texture memory:
   - keys alive between kernel launches: multi-block encryption is faster than on-the-fly!
   - thread count limited by texture cache size

3. Shared memory:
   - 16 round key column reads with no bank conflicts → single kernel multi-block encryption is the fastest!
   - thread count limited by shared memory size
Placement of $T$-tables:

- **Constant memory:**
  - simple and very quick approach
  - unless encrypting same block with same key: almost all $T$-accesses are serialized
  - combine with any key scheduling algorithm

- **Shared memory:**
  - Collision-free approach
    - $T_i, i > 0$ are rotations of $T_0$: place 1KB $T_0$ in each bank.
  - All shared memory used by 1 thread block $\rightarrow$ very low device utilization.
  - Lazy approach: place $T$-tables in order.
    - On average: 6/16 collisions, so remaining reads are parallel.
    - Allows for multiple blocks/SM $\rightarrow$ higher device occupancy.
    - combine with on-the-fly key scheduling or key expansion in texture memory.

- **Texture memory:** ongoing work, but estimates are lower than lazy shared memory approach.
AES GPU Implementation (cont.)

- Placement of $T$-tables:
  1. Constant memory:
     - simple and very quick approach
     - unless encrypting same block with same key: almost all $T$-accesses are serialized
     - combine with any key scheduling algorithm
  2. Shared memory:
     - Collision-free approach
       - $T_i, i > 0$ are rotations of $T_0$: place 1KB $T_0$ in each bank.
AES GPU Implementation (cont.)

- Placement of $T$-tables:
  1. **Constant memory:**
     - **simple and very quick approach**
     - unless encrypting same block with same key: *almost all* $T$-accesses are *serialized*
     - combine with any key scheduling algorithm
  2. **Shared memory:**
     - Collision-free approach
     - $T_i, i > 0$ are rotations of $T_0$: place 1KB $T_0$ in each bank. *All shared memory used by 1 thread block $\rightarrow$ very low device utilization.*
 Placement of $T$-tables:

1. **Constant memory:**
   - simple and very quick approach
   - unless encrypting same block with same key: almost all $T$-accesses are serialized
   - combine with any key scheduling algorithm

2. **Shared memory:**
   - Collision-free approach
     - $T_i, i > 0$ are rotations of $T_0$: place 1KB $T_0$ in each bank. All shared memory used by 1 thread block $\rightarrow$ very low device utilization.
   - Lazy approach
     Place $T$-tables in order.
Placement of $T$-tables:

1. **Constant memory:**
   - Simple and very quick approach
   - Unless encrypting same block with same key: almost all $T$-accesses are serialized
   - Combine with any key scheduling algorithm

2. **Shared memory:**
   - Collision-free approach
     - $T_i, i > 0$ are rotations of $T_0$: place 1KB $T_0$ in each bank. All shared memory used by 1 thread block → very low device utilization.
   - Lazy approach
     - Place $T$-tables in order. On average: 6/16 collisions, so remaining reads are parallel. Allows for multiple blocks/SM → higher device occupancy.
AES GPU Implementation (cont.)

- Placement of $T$-tables:
  - **1** Constant memory:
    - Simple and very quick approach
    - Unless encrypting same block with same key: almost all $T$-accesses are serialized
    - Combine with any key scheduling algorithm
  - **2** Shared memory:
    - Collision-free approach
      - $T_i, i > 0$ are rotations of $T_0$: place 1KB $T_0$ in each bank. All shared memory used by 1 thread block $\rightarrow$ very low device utilization.
    - Lazy approach
      - Place $T$-tables in order. On average: 6/16 collisions, so remaining reads are parallel. Allows for multiple blocks/SM $\rightarrow$ higher device occupancy.
      - Combine with on-the-fly key scheduling or key expansion in texture memory.
AES GPU Implementation (cont.)

- Placement of $T$-tables:
  1. Constant memory:
     - simple and very quick approach
     - unless encrypting same block with same key: almost all $T$-accesses are serialized
     - combine with any key scheduling algorithm
  2. Shared memory:
     - Collision-free approach
       $T_i, i > 0$ are rotations of $T_0$: place 1KB $T_0$ in each bank. All shared memory used by 1 thread block $\rightarrow$ very low device utilization.
     - Lazy approach
       Place $T$-tables in order. On average: 6/16 collisions, so remaining reads are parallel. Allows for multiple blocks/SM $\rightarrow$ higher device occupancy.
     - combine with on-the-fly key scheduling or key expansion in texture memory.
  3. Texture memory: ongoing work, but estimates are lower than lazy shared memory approach.
Encryption: **59.6** and **14.6 Gbps** on the GTX 295 and 8800GTX, respectively.

Decryption: **52.4** and **14.3 Gbps** on the GTX 295 and 8800GTX, respectively.
Conclusions

AES-128 software speed records for encryption and decryption

- 8-bit AVR
  - 1.24× encryption
  - 1.10× decryption
  - smaller code size

- Cell Broadband Engine (SPE)
  - 1.06× encryption
  - 1.18× decryption

- NVIDIA GPU
  - 1.75× encryption
  - First decryption implementation

All numbers subject to further improvements
Conclusions

AES-128 software speed records for encryption and decryption

- 8-bit AVR
  - 1.24× encryption
  - 1.10× decryption
  - smaller code size

- Cell Broadband Engine (SPE)
  - 1.06× encryption
  - 1.18× decryption

- NVIDIA GPU
  - 1.75× encryption
  - First decryption implementation

- All numbers subject to further improvements

To be continued...