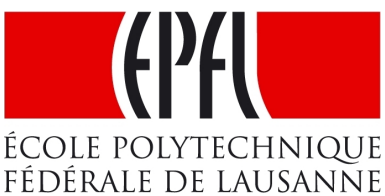
Fast Implementations of AES on Various Platforms

Joppe W. Bos[1] Dag Arne Osvik[1] Deian Stefan[2]

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

[2]Dept. 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

E. Käsper and P. Schwabe. *Faster and Timing-Attack Resistant AES-GCM.* CHES 2009.

P. Bulens, et al. *Implementation of the AES-128 on Virtex-5 FPGAs* AFRICACRYPT 2008.

O. Harrison and J. Waldron. *Practical Symmetric Key Cryptography on Modern Graphics Hardware.* USENIX Sec. Symp. 2008.

S. Rinne, et al. *Performance Analysis of Contemporary Light-Weight Block Ciphers on 8-bit Microcontrollers.* SPEED 2007.

K. Shimizu, et al. *Cell Broadband Engine Support for Privacy, Security, and Digital Rights Management Applications.* 2005.

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 $\longrightarrow$ no attacks on full AES-128
- Very efficient in hardware and software

AES-128 Block Cipher

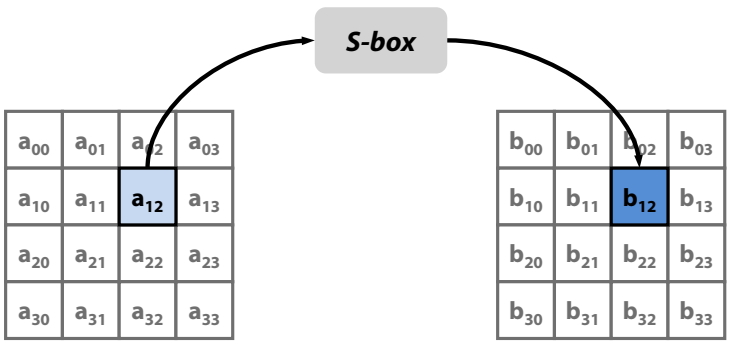
- Algorithm consists of 5 steps:
  1. Key expansion:
     128-bit $\rightarrow N_r + 1 = 11$ *128-bit round keys*
  2. State initialization:
     initial state $\leftarrow$ *plaintext block* $\oplus$ *128-bit key*
  3. Round transformation:
     apply round function on state $N_r - 1$ *times*
  4. Final round transformation:
     apply the modified round function
- Core of AES, the round function, consists of the following steps:
  - `SubBytes`, `ShiftRows`, `MixColumns`, and `AddRoundKey`.

AES-128 Block Cipher

- Algorithm consists of 5 steps:
  1. Key expansion:
     128-bit $\rightarrow N_r + 1 = 11$ *128-bit round keys*
  2. State initialization:
     initial state $\leftarrow$ *plaintext block* $\oplus$ *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`.
- Decryption follows the same procedure
  - round function steps are the inverse and run in reverse order

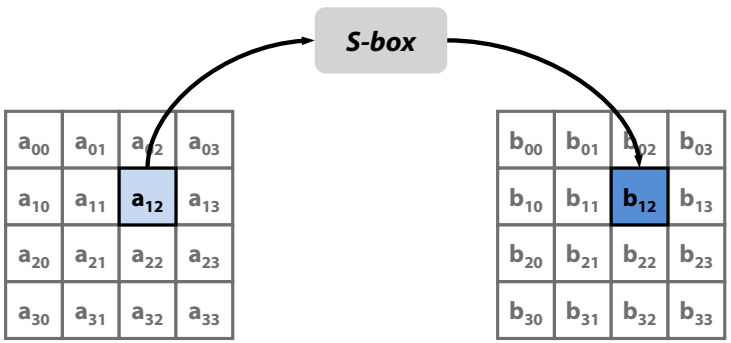
Round Function Steps

1. `SubBytes`:

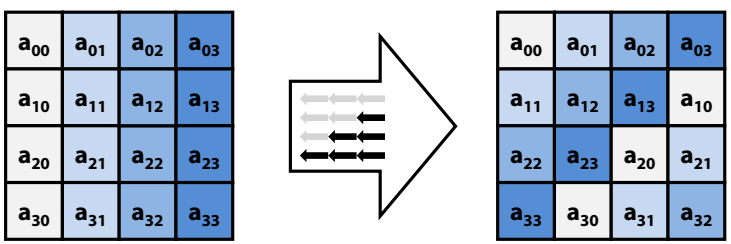


Round Function Steps

1. `SubBytes`:

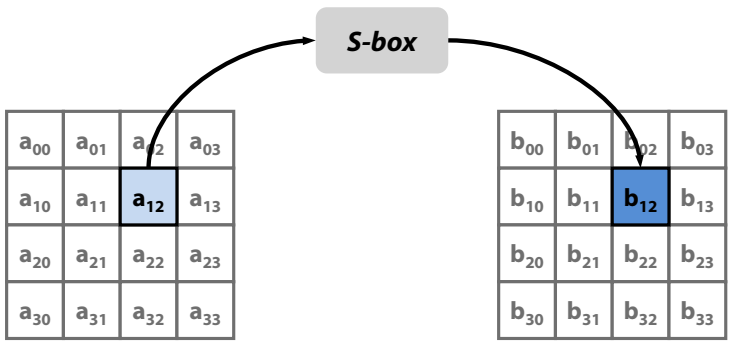


2. `ShiftRows`:

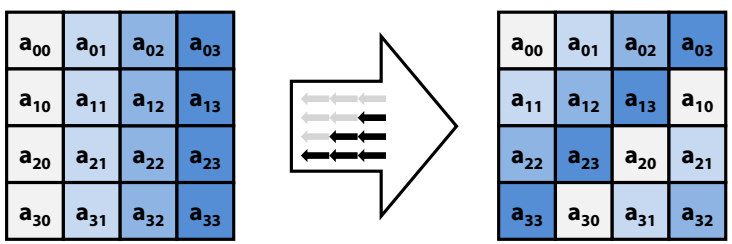


Round Function Steps

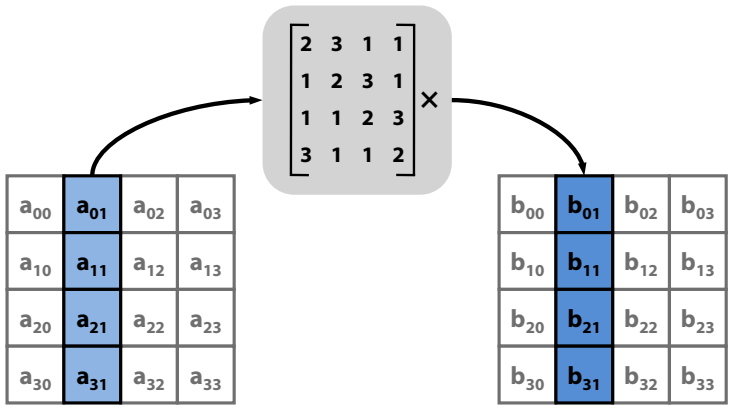
1. SubBytes:



2. ShiftRows:

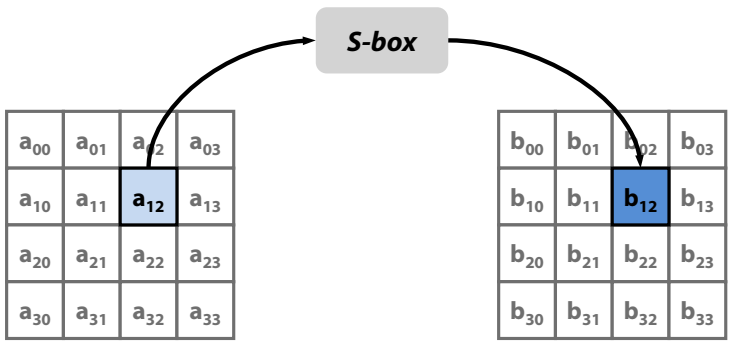


1. MixColumns:

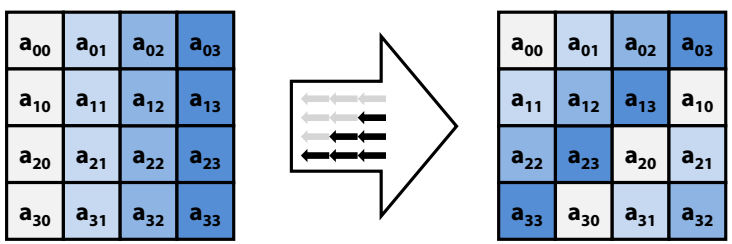


Round Function Steps

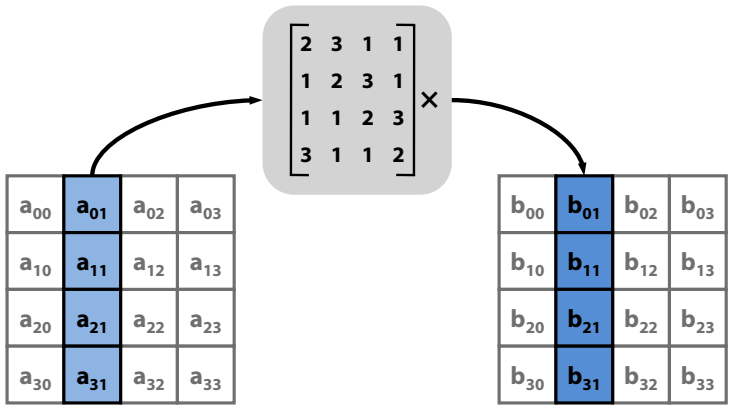
1. SubBytes:



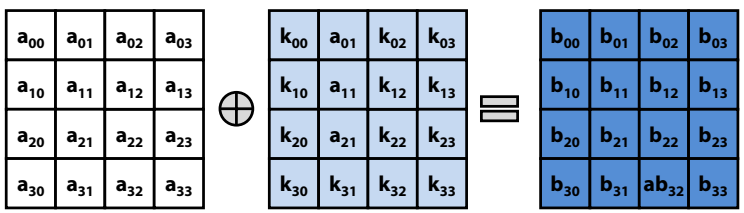
2. ShiftRows:



1. MixColumns:



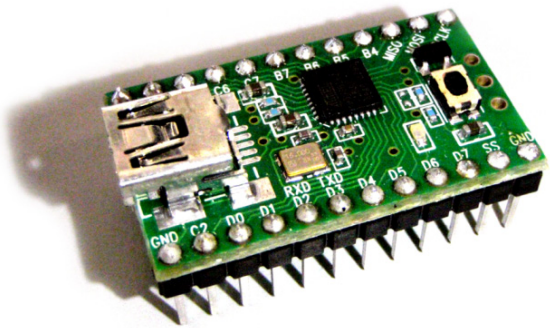
2. AddRoundKey:



Target Platforms

NVIDIA GPUs

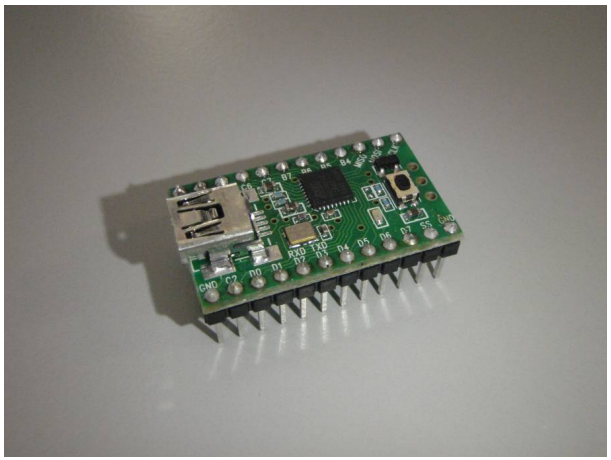
Cell B.E

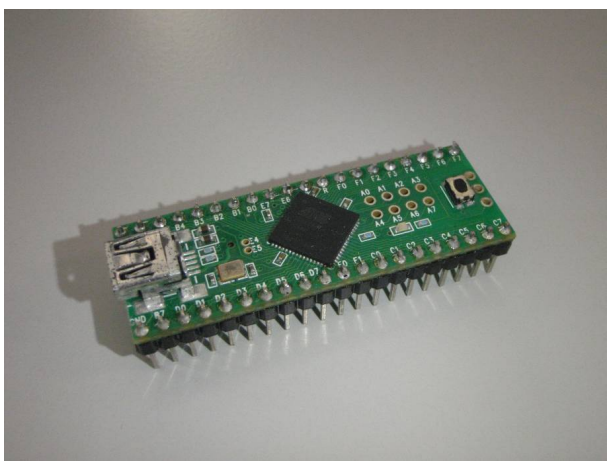


Atmel AVRs

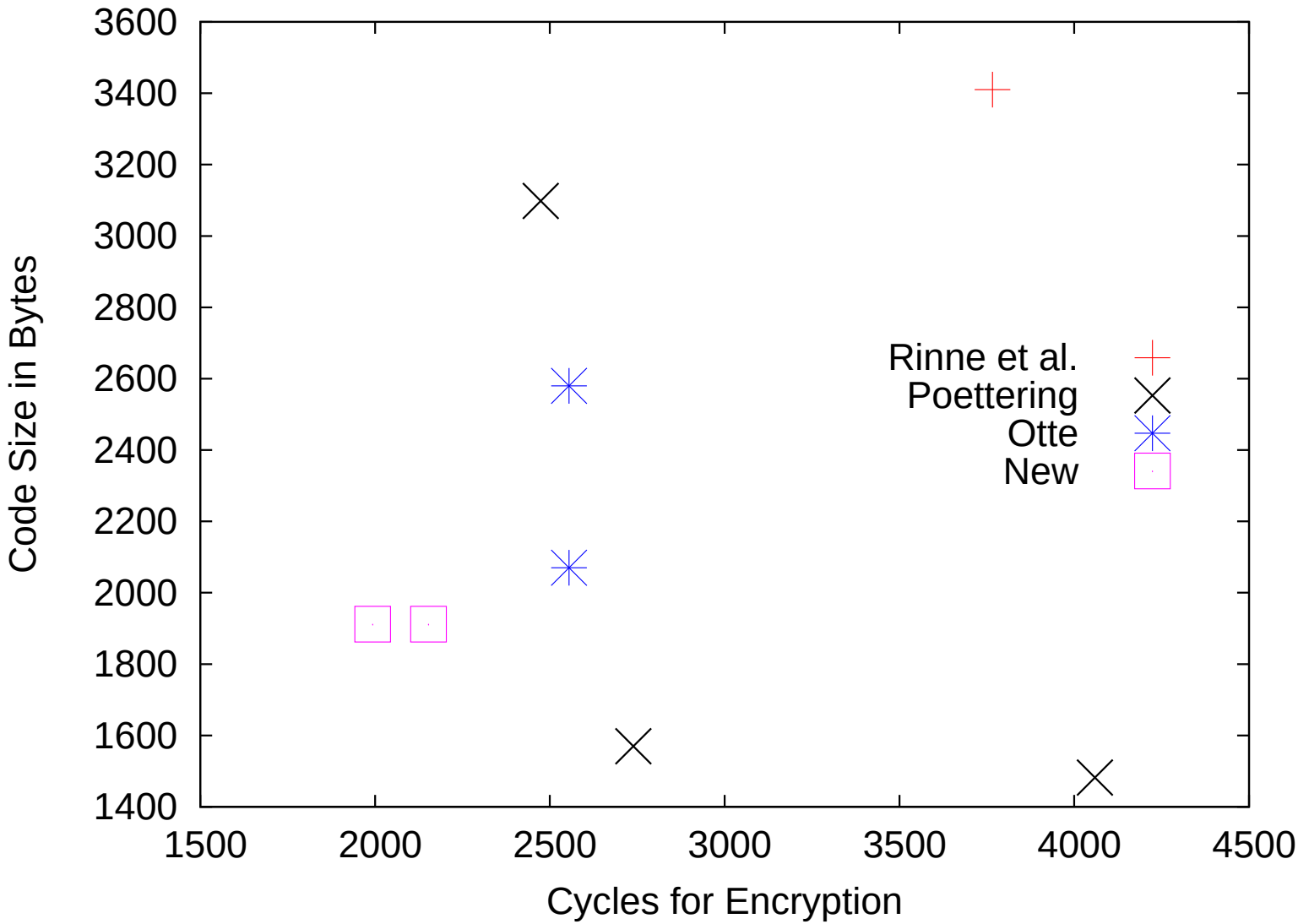
Advanced Virtual RISC Architecture

- 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

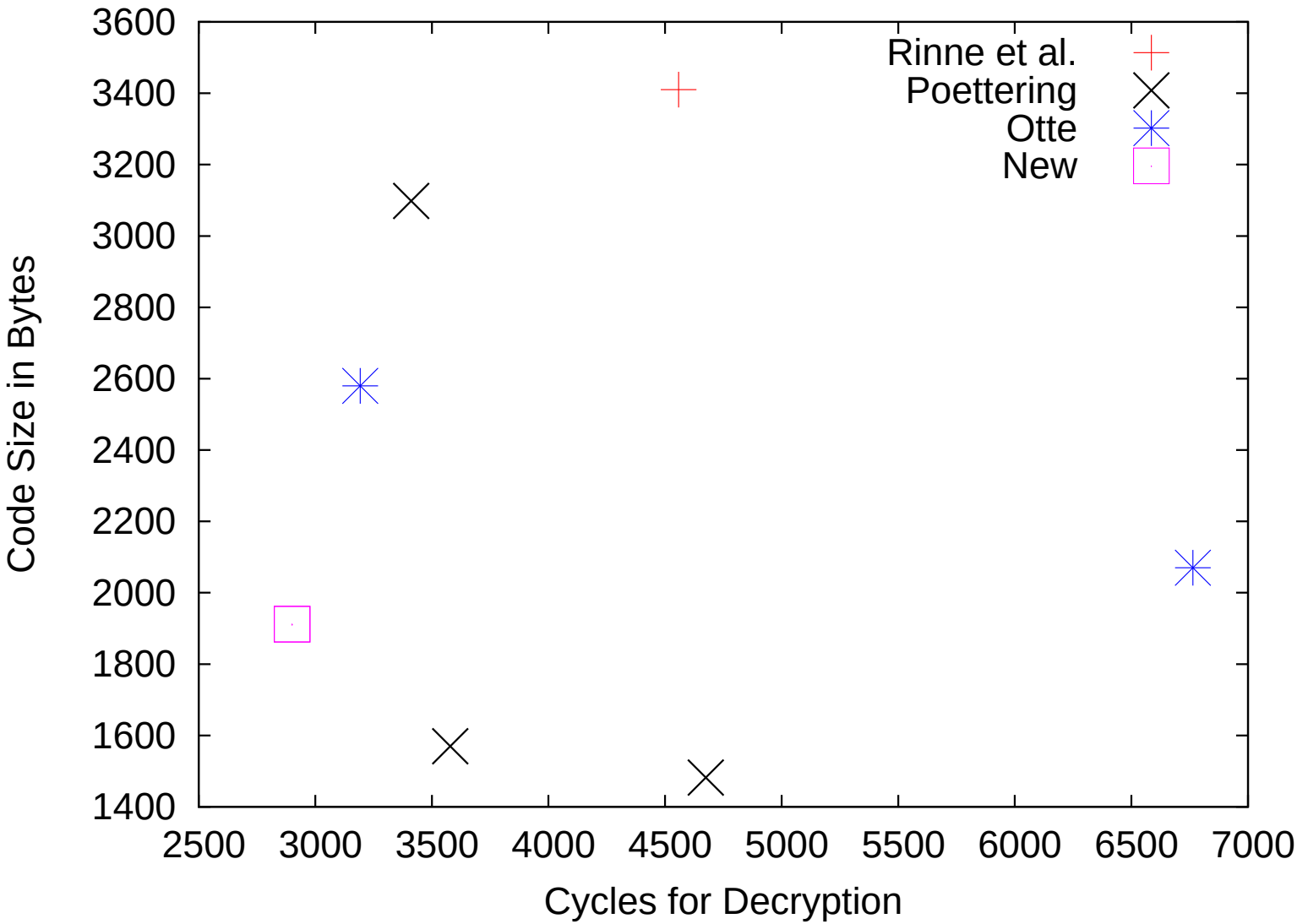




Encryption Comparison AVR



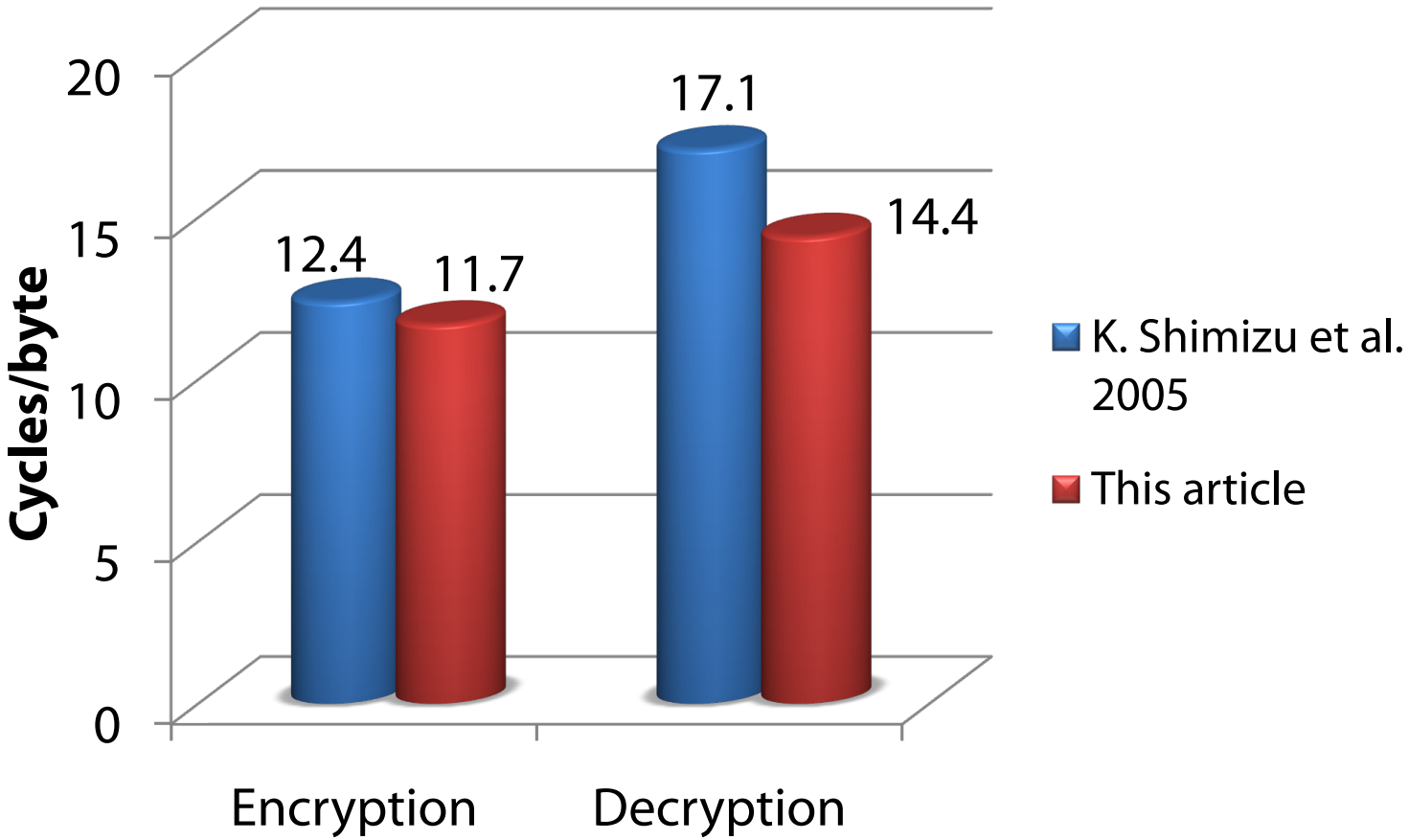
Decryption Comparison AVR



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
    $f : \{0,1\}^2 \to \{0,1\}$ are present.
- "Expensive" QS22 Blade Servers (2×8 SPEs)
- "Cheap" PS3 video game console (6 SPEs)

SPU Results Comparison



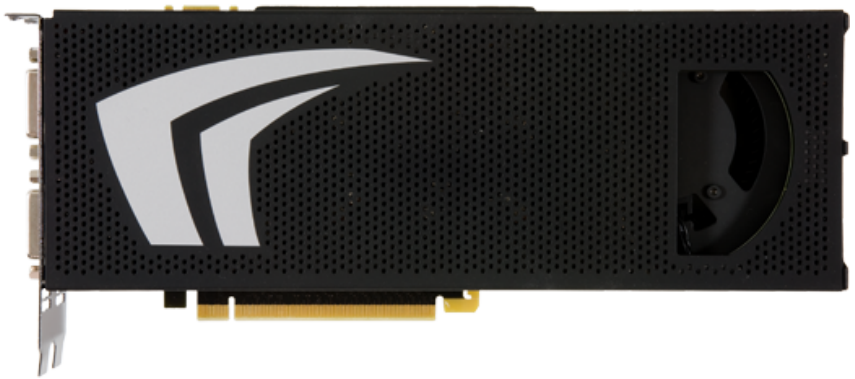
Throughput per PS3: **13.2** (encryption) and **10.8 Gbps** (decryption)

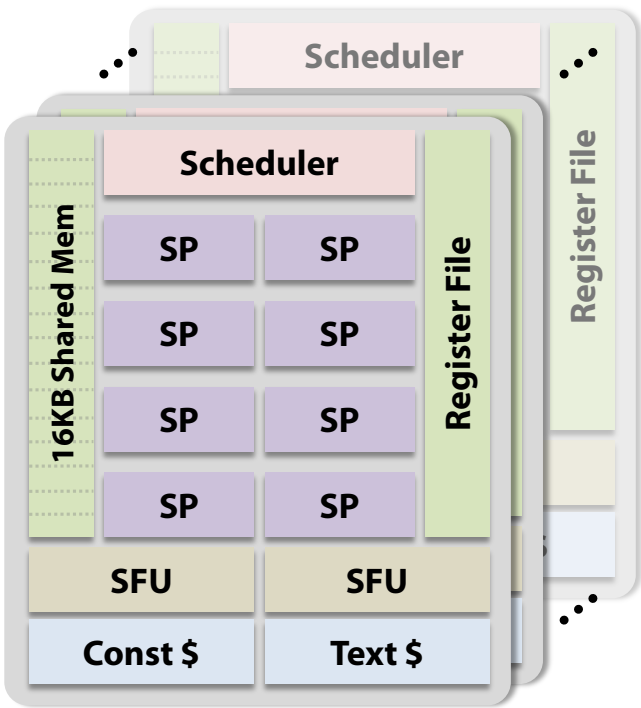
Work-in-progress, fill both pipelines

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:
  2×30 SMs @ 1.24GHz





AES GPU Implementation

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

$$[s_{j0}, s_{j1}, s_{j2}, s_{j3}]^{\mathrm{T}} = T_0[a_{c_0 0}] \oplus T_1[a_{c_1 1}] \oplus T_2[a_{c_2 2}] \oplus T_3[a_{c_3 3}] \oplus k_j,$$

where each T_i is $1KB$ and k_j is the jth column of the round key.

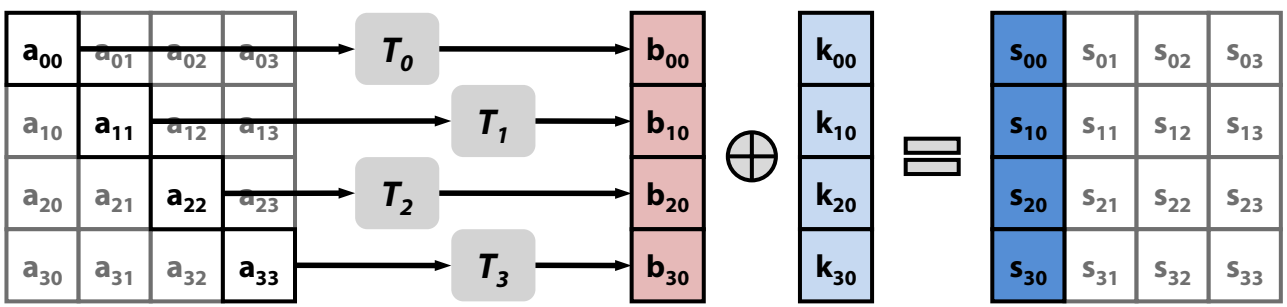
AES GPU Implementation

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

$$[s_{j0}, s_{j1}, s_{j2}, s_{j3}]^{\mathrm{T}} = T_0[a_{c_0 0}] \oplus T_1[a_{c_1 1}] \oplus T_2[a_{c_2 2}] \oplus T_3[a_{c_3 3}] \oplus k_j,$$

  where each T_i is $1KB$ and k_j is the jth column of the round key.

- Example ($j = 0$):



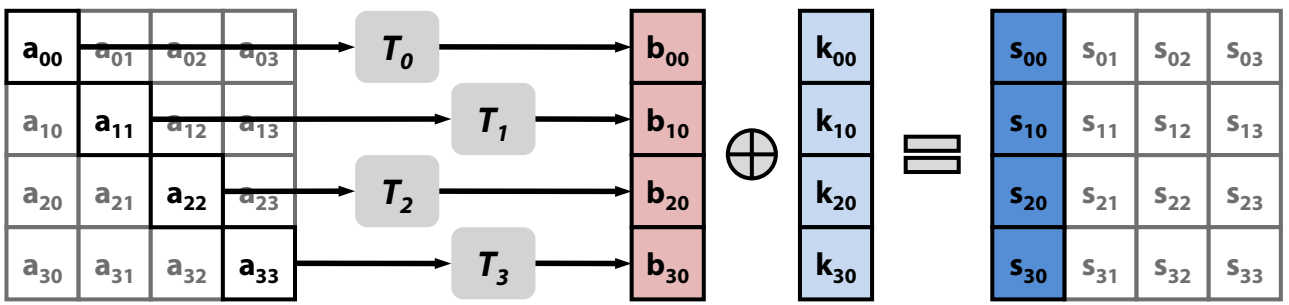
AES GPU Implementation

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

$$[s_{j0}, s_{j1}, s_{j2}, s_{j3}]^{\mathrm{T}} = T_0[a_{c_0 0}] \oplus T_1[a_{c_1 1}] \oplus T_2[a_{c_2 2}] \oplus T_3[a_{c_3 3}] \oplus k_j,$$

  where each T_i is $1KB$ and k_j is the jth column of the round key.

- Example ($j = 0$):



- Optimization approach: launch thread blocks containing multiple independent groups of 16 (1/2-warp) streams.

AES GPU Implementation (cont.)

- 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 $\longrightarrow$ no bank conflicts!

AES GPU Implementation (cont.)

- 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 $\longrightarrow$ 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

AES GPU Implementation (cont.)

- 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 $\longrightarrow$ 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 $\longrightarrow$ single kernel multi-block encryption is the fastest!
     - thread count limited by shared memory size

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

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 $\longrightarrow$ very low device utilization.

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 $\longrightarrow$ very low device utilization.
     - Lazy approach
       Place T-tables in order.

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 $\longrightarrow$ 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 $\longrightarrow$ higher device occupancy.

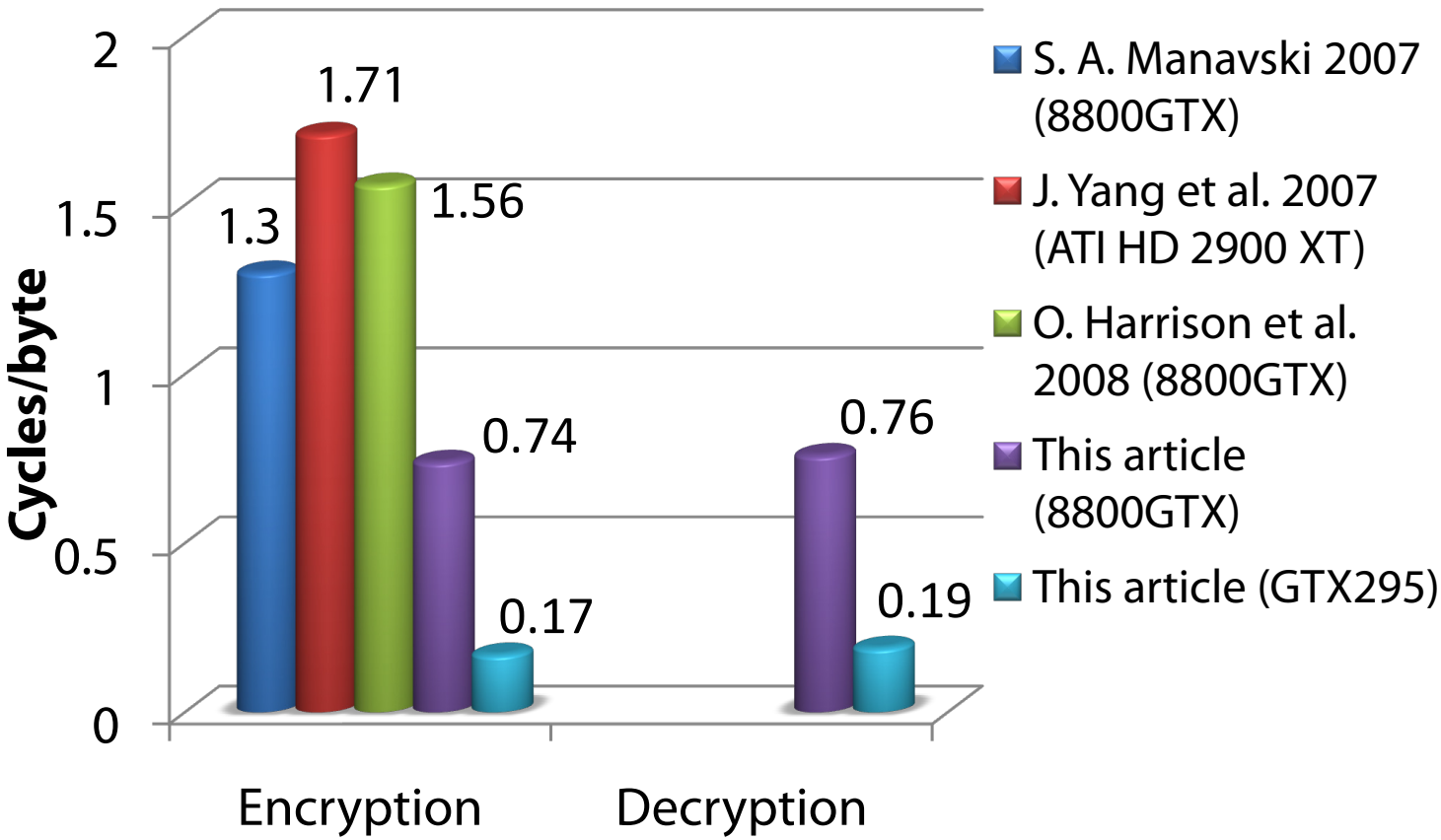
AES GPU Implementation (cont.)

- 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 $\longrightarrow$ 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 $\longrightarrow$ 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 $\longrightarrow$ 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 $\longrightarrow$ 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.

GPU Results Comparison



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...