Projektarbeit des Studienganges "Ingenieurinformatik" im Rahmen des Softwarepraktikums

Beschleunigung von AES-Verschlüsselungen unter Zuhilfenahme einer GPU

Simon Waloschek, Benedikt Krüger, Ibrahim Alptekin, Daniel Nickchen

16. November 2010

Inhaltsverzeichnis

1. Einführung und Grundlagen

1.1. Einleitung

Moderne Verschlüsselungsalgorithmen sind im Allgemeinen sehr rechenintensiv und werden oft als Bestandteil des Betriebssystems ausgeführt. Da gewöhnliche CPUs für diese Art von Operationen nicht ausgelegt sind, wird das gesamte System durch die entstehende Auslastung gebremst. Es liegt also nahe, eine geeignetere Plattform für die Berechnung von Verschlüsselungen zu nutzen.

Im Rahmen dieses Praktikums wird daher evaluiert, inwiefern sich moderne Grafikkarten bzw. FPGAs zur optimierteren Ausführung nutzen lassen. Die Verschlüsselungen sollen transparent in den Linux-Kernel eingebunden und systemweit zur Verfügung gestellt werden.

Im Folgenden wird mithilfe des CUDA-Frameworks von NVIDIA der AES-Algorithmus auf einer GPU vom Typ "GTS 8800" aus dem Hause NVIDIA implementiert und durch eine Reihe von Tests auf eine eventuelle Verbesserung der Datendurchsatzrate hin überprüft.

1.2. Grundlagen

1.2.1. AES im Kurzüberblick

Der Advanced Encryption Standard (AES) ist ein symmetrisches Kryptosystem. Es wurde von Joan Daemen und Vincent Rijmen im Rahmen eines international ausgeschriebenen Wettbewerbes des National Institute of Standards and Technology (NIST) entwickelt. Als Nachfolger von DES und 3DES, gilt AES seit 2000 als De-facto Verschlüsselungsstandard, welcher Dank seiner starken Verschlüsselung selbst höchsten Sicherheitsansprüchen genügt.

Bei AES handelt es sich um ein Blockverschlüsselungssystem, auch Blockchiffre genannt, also ein Verschlüsselungsverfahren, bei dem der Klartext in eine Folge gleichgroßer Blöcke zerlegt wird. Diese Blöcke werden anschließend unabhängig voneinander mit einem aus einem Schlüsselwort berechneten Blockschlüssel chiffriert. Somit werden auch Chiffretextblöcke mit einer festen Länge erzeugt und letzendlich zum endgültigen Chiffretext aneinandergereiht.

AES schränkt die Blocklänge auf 128 Bit ein. Die Schlüssellänge kann jedoch zwischen 128, 192 und 256 Bit gewählt werden, weshalb zwischen den drei AES-Varianten AES-128, AES-192 und AES-256 unterschieden wird. AES bietet ein sehr hohes Maß an Sicherheit und ist in den USA sogar für staatliche Dokumente mit höchster Geheimhaltungsstufe zugelassen. Der Algorithmus ist frei verfügbar und darf ohne Lizenzgebühren eingesetzt sowie in Software bzw. Hardware implementiert werden.

1.2.2. CUDA Framework

Das „Compute Unified Device Architecture Software Developer Kit“ (CUDA SDK) wurde von NVIDA am 15. Februar 2007 erstmals der Öffentlichkeit vorgestellt. Intention dieses SDKs ist, die Programmierung aktueller Grafikkarten unter einer einheitlichen und standardisierten Schnittstelle zu ermöglichen.

Die Architektur moderner GPUs ist aufgrund ihrer Geschichte als reine Berechnungseinheit für Bildschirmausgaben für den Zweck ausgelegt, Operationen parallel auszuführen. Als Co-Prozessor können GPUs somit Dank der CUDA-API dazu genutzt werden, bestimmte Programmteile signifikat schneller abzuarbeiten.

CUDA basiert auf einer optimierten Variante von C (“C for CUDA”) und ist damit weitestgehend plattformunabhängig. So ist es möglich, entsprechend programmierte CUDA-Anwendungen unter Windows, Linux und Mac OS auszuführen - eine kompatible Grafikkarte vorausgesetzt.

2. GPU Architektur

Zum besseren Verständnis der AES-Implementierung, ist ein Überblick über die Speicher- und Prozessverwaltungsarchitektur von Nöten. Die Nachfolgenden Erläuterung beziehen sich zwar speziell auf die verwendete Grafikkarte, sind jedoch in großen Teilen auf verwandte Modelle übertragbar.

Im Folgenden ist die Architektur der GPU abgebildet. Hierbei ist zu beachten, dass die Anzahl der Multiprozessoren auf der GPU, sowie die Anzahl der Streaming-Prozessoren (SP) je nach Modell unterschiedlich sind.

2.1. Prozessverwaltung

2.1.1. Multithreaded Instruction Unit (MT IU)

Die MT IU verwaltet die Ausführung von Threads auf dem Multiprocessor. Hierbei werden einzelne Threads zu einem Block, mehrere Blöcke zu einem Grid zusammengefasst.

2.1.2. Streaming Multiprocessor

Auf jeden Streaming Multiprocessor wird genau ein Block abgebildet. Die in dem Block befindlichen Threads werden -soweit möglich- parallel abgearbeitet. Hierbei hat jeder Thread in dem Block eine eindeutige ID, auf welche auch in dem Thread zugegriffen werden kann.

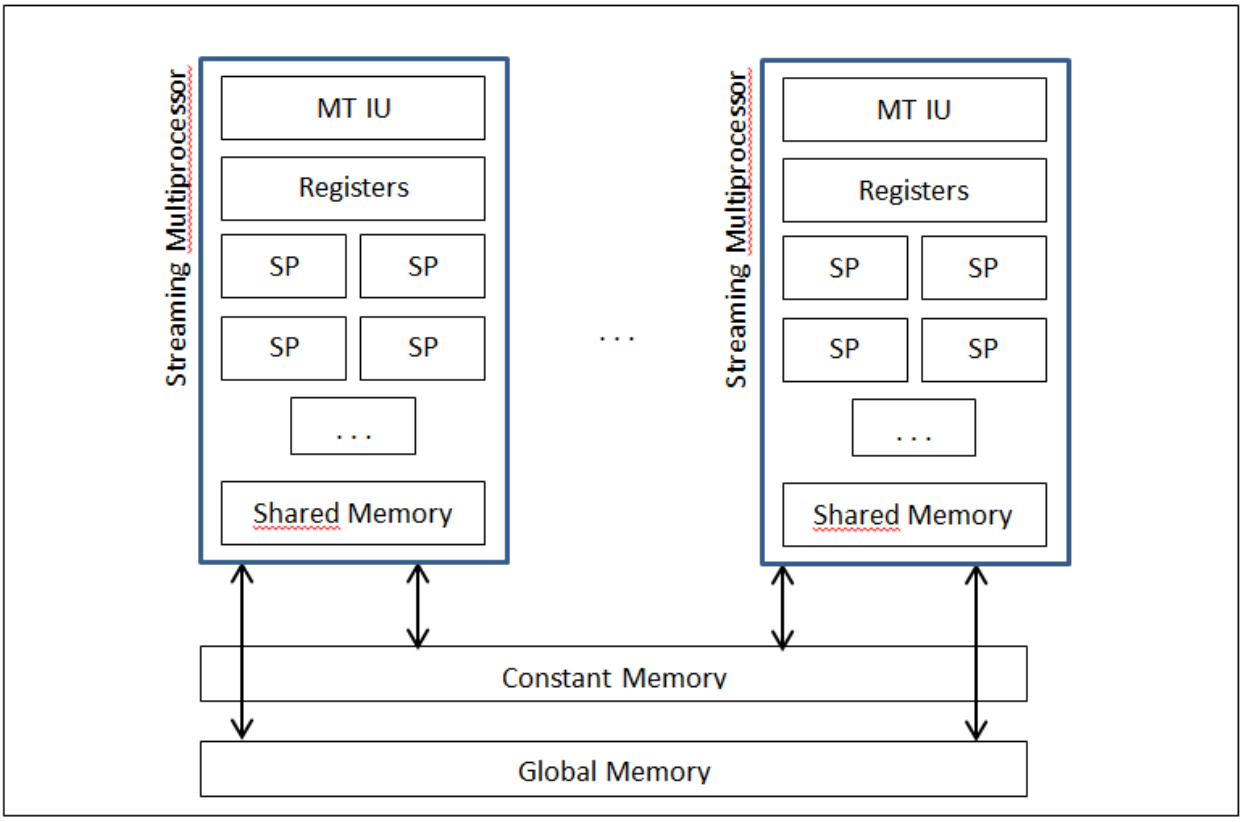

Abbildung 1: GPU-Architektur

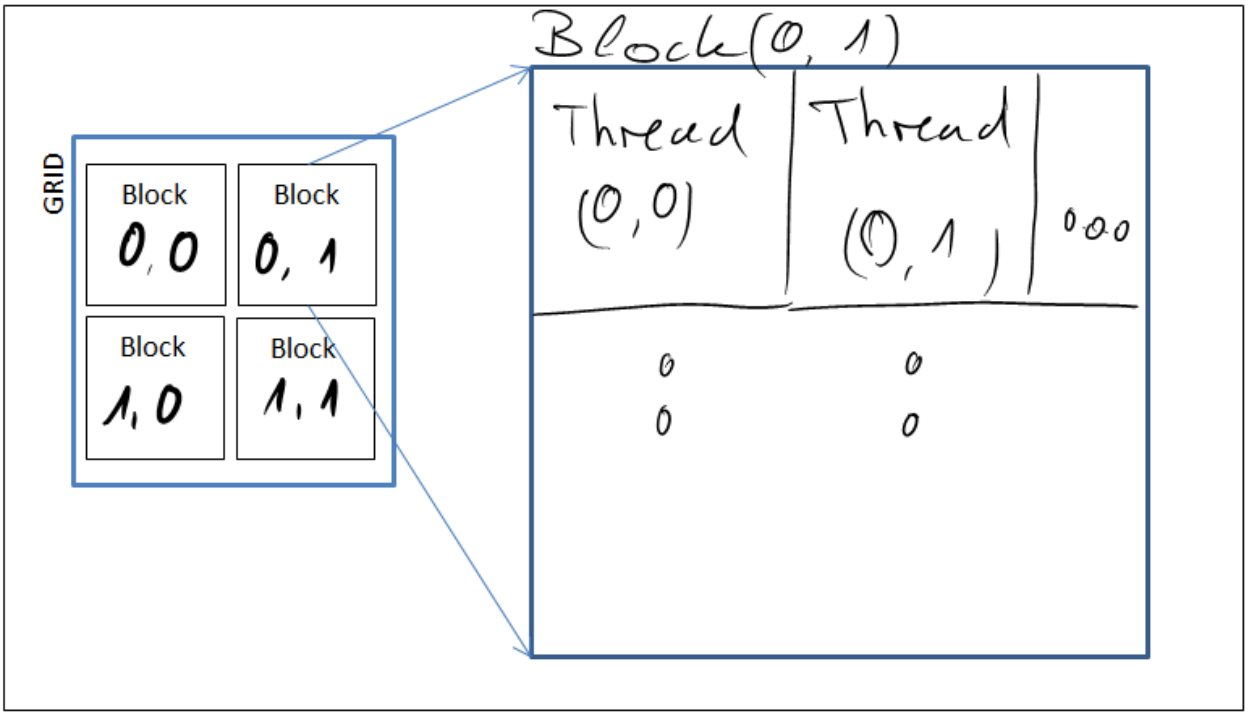

Abbildung 2: Streaming Processor

2.1.3. Streaming Processor (SP)

Jeder SP führt genau einen Tread aus.

2.2. Speicherhierarchie

Moderne Grafikarten sind über die PCI-E Schnittstelle an die CPU angebunden. Über diesen Bus werden die Daten und Prozesse an die berechnenden Einheiten der GPU übertragen.

2.2.1. Global Memory

Der Global Memory (RAM) ist der Größte Bereich und sowohl von CPU als auch von der GPU schreib- und lesbar. Dieser Speicher ermöglicht den Austausch von Daten zwischen GPU und CPU. Dieser Speicher hat die größte Kapazität, ist jedoch der langsamste der Speicherhierarchie bezüglich der GPU.

2.2.2. Constant Memory

Der Constant Memory ist physikalisch auf 64KB beschränkt.

2.2.3. Register

Threads eines Bocks teilen sich gemeinsame Register.

Shared Memory

Für jeden Streaming-Multiprozessor-Prozessor ist ein Shared Memory vorgesehen, welches der schnellste -mit 16 KB jedoch auch der kleinste- Speicher in der Hierarchie der GPU ist. Der Multiprozessor teilt diesen verfügbaren Speicher und seinen Streamingprozessoren auf. Dieser Speicher kann nur von der GPU gelesen und geschrieben werden.

3. AES-Implementierung

3.1. Algorithmus im Detail

Im diesem Kapitel wird der Ablauf des AES Algorithmus vorgestellt. Er besteht aus mehreren Phasen, auf die im Folgenden genauer eingegangen wird.

Jeder Block wird zunächst in eine zweidimensionale Tabelle mit vier Zeilen und 4 Spalten geschrieben, deren Zellen ein Byte groß sind. Jeder Block wird nun nacheinander bestimmten Transformationen unterzogen. Anstatt jeden Block einmal mit dem Schlüssel zu verschlüsseln, wendet AES verschiedene Teile des erweiterten Originalschlüssels nacheinander auf den Klartext-Block an. Die Anzahl r dieser Runden variiert und ist von der Schlüssellänge k abhängig:

Schlüssellänge k	Runden r
128	10
192	12
256	14

Tabelle 1: Rundenlängen

Der Ablauf jeder einzelnen Block-Verschlüsselung entspricht folgendem Schema:

3.1.1. Schlüsselexpansion

Der Benutzerschlüssel wird in r+1 Teilschlüssel aufgeteilt, die sogenannten Rundenschlüssel. Diese müssen dieselbe Länge wie die Blöcke haben, was bedeutet, dass der Benutzerschlüssel zunächst auf die Länge 128*(r+1) expandiert werden muss. Aus diesem werden die Rundenschlüssel erzeugt und ebenfalls in Tabellen mit 4 Spalten und 4 Zeilen gespeichert. Für die Erzeugung der ersten Spalte des jeweils nächsten Rundenschlüssels wird

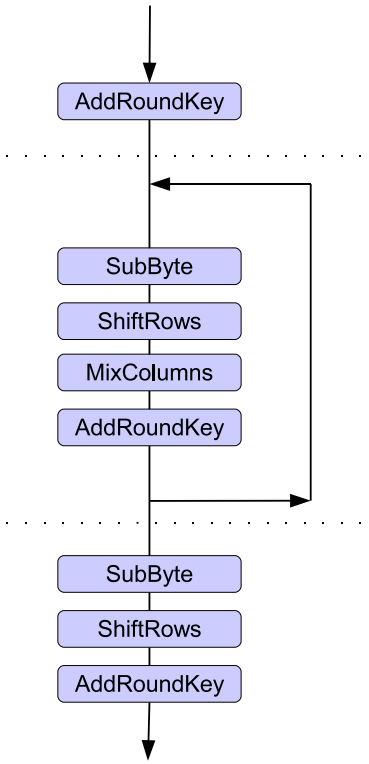

Abbildung 3: Ablaufschema

zunächst die letzte Spalte des vorherigen Schlüssels, am Anfang also des Benutzerschlüssels, genommen und um eine Zeile nach oben rotiert. Die oberste Zelle wird unten wieder eingefügt. Nun erfolgt eine Substitution der vier Werte mit Hilfe der Substitutionsbox. Sie ist meist als Array aufgebaut und gibt an, wie jedes Byte durch einen anderen Wert zu ersetzen ist.

Die Konstruktion der S-Box unterliegt Designkriterien, die die Anfälligkeit für die Methoden der linearen und der differentiellen Kryptoanalyse sowie für algebraische Attacken minimieren sollen. Mit Hilfe der S-Box wird jedes Byte des Blocks durch ein Äquivalent ersetzt und die Daten somit monoalphabetisch verschlüsselt. Diese neuerzeugte Spalte wird mit der drei Spalten zurückliegenden Spalte und mit der ersten Spalte der sogenannten Rcon-Tabelle XOR-verknüpft.

Die Rcon-Tabelle ist ebenfalls in Form eines Arrays mit 4 Zeilen und einer Spalte für jeden Rundenschlüssel aufgebaut. Sie enthält konstante Werte die auf einem mathematischen System beruhen. Die aus der XOR-Verknüpfung erzeugten Werte ergeben die erste Spalte des nächsten Rundenschlüssels. Die restlichen drei Spalten ergeben sich jeweils aus einer XOR-Verknüpfung der davor liegenden und der zu dieser drei zurückliegenden Spalte. Für alle folgenden Rundenschlüssel läuft die Berechnung analog ab. Abbildung 3

Abbildung 4: Berechnung der Rundenschlüssel

veranschaulicht dieses Vorgehen.

3.1.2. Vorrunde

Bei AddRoundKey erfolgt eine XOR-Verknüpfung zwischen dem zu verschlüsselnden Block und dem ersten Rundenschlüssel (siehe Abb.4). Nur an dieser Stelle ist der Algorithmus vom Benutzerschlüssel abhängig.

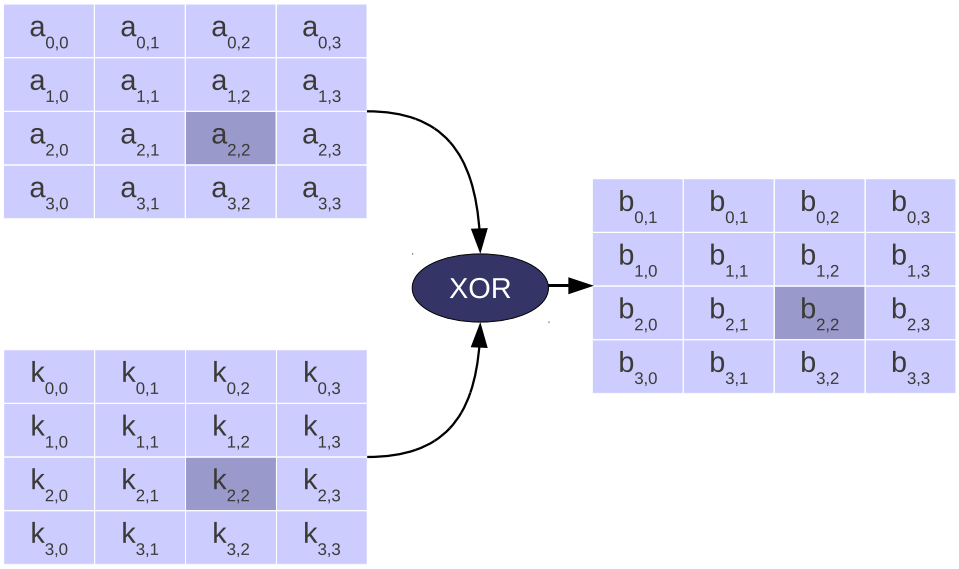

Abbildung 5: AddRoundKey

3.1.3. Verschlüsselungsrunden

In den folgenden Verschlüsselungsrunden wird zunächst eine Substitution durchgeführt.

Dazu wird die Substitutionsbox verwendet. Im darauffolgenden Schritt, genannt ShiftRow, werden die Zeilen um eine bestimmte Anzahl von Spalten nach links verschoben und links hinausgeschobene Zellen rechts wieder angefügt. Die erste Zeile bleibt konstant, die zweite wird um eine, die dritte um zwei und die vierte um drei Spalten verschoben. Abbildung 5 veranschaulicht dieses Vorgehen.

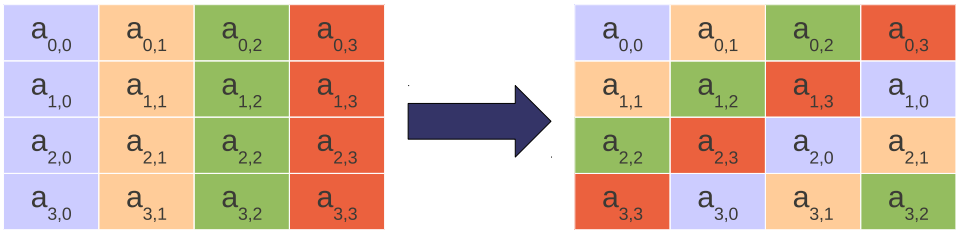

Abbildung 6: ShiftRow

Im MixColumn-Schritt wird zunächst jede Zelle mit einer Konstanten multipliziert und dann die Spalten mit den Ergebnissen der Multiplikation XOR verknüpft. Durch eine geschickte Analyse dieser Operation, vereinfacht sich die Rechnung zu einer simplen Matrixmultiplikation.

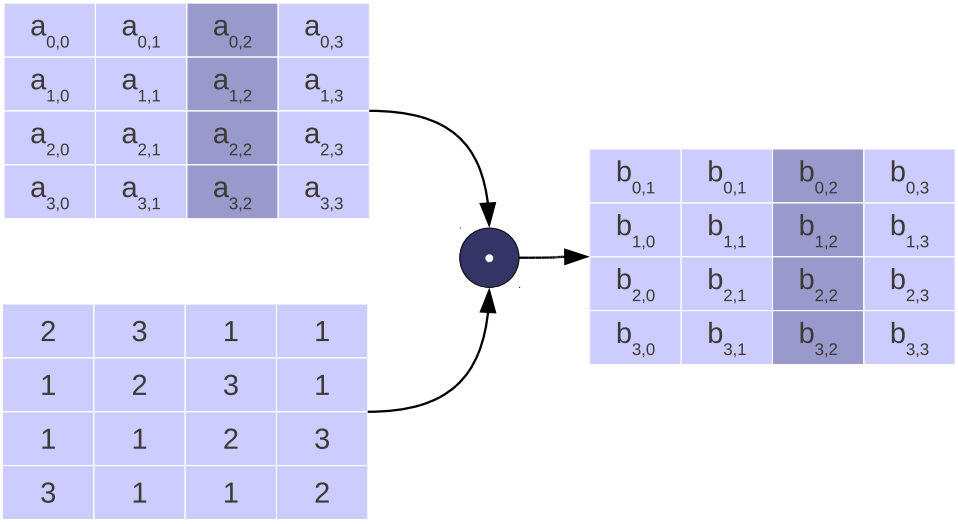

Abbildung 7: MixColumn

Zum Ende jeder Verschlüsselungsrunde wird noch einmal AddRoundKey ausgeführt. Abgeschlossen wird die Verschlüsselung mit der Schlussrunde, sie verläuft identisch zu den übrigen Verschlüsselungsrunden mit dem Unterschied, dass keine MixColumn-Funktion ausgeführt wird.

3.1.4. Entschlüsselung

Bei der Entschlüsselung von Daten wird entsprechend rückwärts vorgegangen. Die Daten werden zunächst wieder in zweidimensionale Tabellen gelesen und die Rundenschlüssel generiert. Allerdings wird nun mit der Schlussrunde angefangen und alle Funktionen in jeder Runde in der umgekehrten Reihenfolge aufgerufen. Durch die vielen XOR-Verknüpfungen

unterscheiden sich die meisten Funktionen zum Entschlüsseln nicht von denen zum Verschlüsseln. Jedoch muss eine andere S-Box genutzt werden (die sich aus der originalen S-Box berechnen lässt) und die Zeilenverschiebungen erfolgen in die andere Richtung.

3.2. C++ Implementierung

Ausgangspunkt der weiteren Bearbeitung ist eine unfertige C++ Implementierung des Algorithmus von Paulo S. L. M. Barreto. Der Autor hat bereits erste Ansätze zur Datenverschlüsselung auf einer GPU programmiert, hat das Projekt jedoch aus unbekannten Gründen eingestellt, sodass noch einige Ergänzungen und Änderungen vorgenommen werden müssen.

Die Implementierung ist Dank der Ausnutzung komplexerer Galois-Feld-Operationen vergleichsweise schnell. Hierbei werden die Berechnungen durch Nutzung vorkalkulierter Tabellen auf Kosten der Speichereffizienz beschleunigt. Der Aufwand der Verschlüsselung verringert sich somit auf eine Reihe (schneller) Binäroperationen, welche sich auf der GPU vorteilhaft parallel ausführen lassen können.

Zusätzlich ist anzumerken, dass zugunsten der Parallelität der sogenannte "Electronic Code Book Mode" (ECB) verwendet wird, also jeder Block mit dem gleichen expandierten Schlüssel chiffriert wird.

3.2.1. Schnittstelle zur Außenwelt

Um von außen auf die Verschlüsselungsfunktionen zugreifen zu können, enthält die genutzte AES-Klasse neben dem Konstruktur "öffentliche" Funktionen für die Schlüsselexpansionen sowie für die Ver- bzw. Entschlüsselung.

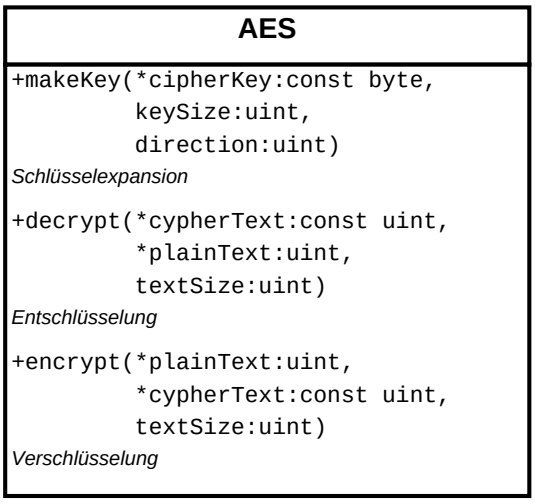

Abbildung 8: Klassendiagramm mit public Funktionen

3.3. CUDA-spezifische Veränderungen

3.3.1. Prozessaufteilung

3.3.2. Speichernutzung

4. Tests und Benchmarks

4.1. Testumgebung

4.2. Ergebnisse

5. Ausblick

A. Quelltext

```
1   /**
2    * AES.cpp
3    *
4    * The Advanced Encryption Standard (AES, aka AES) block cipher,
5    * designed by J. Daemen and V. Rijmen.
6    *
7    * @author Paulo S. L. M. Barreto, Simon Waloschek, Benedikt Krueger
8    *
9    * This software is hereby placed in the public domain.
10   *
11   * THIS SOFTWARE IS PROVIDED BY THE AUTHORS ''AS IS'' AND ANY EXPRESS
12   * OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
13   * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
14   * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE
15   * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
16   * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
17   * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
18   * BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
19   * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
20   * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
21   * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
22   */
23  #include <assert.h>
24  #include <string.h>
25  #include <stdlib.h>
26
27  #ifdef BENCHMARK
28  #include <stdio.h>
29  #include <time.h>
30  #endif
31
32  #include "AES.h"
33  #include "AES.tab"
34
35  #define FULL_UNROLL
36
37  #ifdef _MSC_VER
38  #define SWAP(x) (_lrotl(x, 8) & 0x00ff00ff | _lrotr(x, 8) & 0xff00ff00)
39  #define GETWORD(p) SWAP(*((uint *)(p)))
40  #define PUTWORD(ct, st) (*((uint *)(ct)) = SWAP((st)))
41  #else
42  #define GETWORD(pt) (((uint)(pt)[0] << 24) ^ ((uint)(pt)[1] << 16) ^ ((uint)(pt)[2] <<  8) ^ ((uint)(pt)[3]))
43  #define PUTWORD(ct, st) ((ct)[0] = (byte)((st) >> 24), (ct)[1] = (byte)((st) >> 16), (ct)[2] = (byte)((st) >>
        8), (ct)[3] = (byte)(st), (st))
44  #endif
45
46  /////////////////////////////////////////////////////////////////////
47  // Construction/Destruction
48  /////////////////////////////////////////////////////////////////////
49
50  AES::AES() {
```

```
51      cudaMalloc((void**)&ce_sched, sizeof(e_sched));
52      cudaMalloc((void**)&cd_sched, sizeof(d_sched));
53  }
54
55  AES::~AES() {
56      Nr = 0;
57      memset(e_sched, 0, sizeof(e_sched));
58      memset(d_sched, 0, sizeof(d_sched));
59
60      cudaFree(ce_sched);
61      cudaFree(cd_sched);
62  }
63
64  ////////////////////////////////////////////////////////////////////////
65  // Support methods
66  ////////////////////////////////////////////////////////////////////////
67
68  void AES::ExpandKey(const byte *cipherKey, uint keyBits) {
69      uint *rek = e_sched;
70      uint i = 0;
71      uint temp;
72      rek[0] = GETWORD(cipherKey     );
73      rek[1] = GETWORD(cipherKey +  4);
74      rek[2] = GETWORD(cipherKey +  8);
75      rek[3] = GETWORD(cipherKey + 12);
76      if (keyBits == 128) {
77          for (;;) {
78              temp  = rek[3];
79              rek[4] = rek[0] ^
80                  (Te4[(temp >> 16) & 0xff] & 0xff000000) ^
81                  (Te4[(temp >>  8) & 0xff] & 0x00ff0000) ^
82                  (Te4[(temp      ) & 0xff] & 0x0000ff00) ^
83                  (Te4[(temp >> 24)       ] & 0x000000ff) ^
84                  rcon[i];
85              rek[5] = rek[1] ^ rek[4];
86              rek[6] = rek[2] ^ rek[5];
87              rek[7] = rek[3] ^ rek[6];
88              if (++i == 10) {
89                  Nr = 10;
90                  return;
91              }
92              rek += 4;
93          }
94      }
95      rek[4] = GETWORD(cipherKey + 16);
96      rek[5] = GETWORD(cipherKey + 20);
97      if (keyBits == 192) {
98          for (;;) {
99              temp = rek[ 5];
100             rek[ 6] = rek[ 0] ^
101                 (Te4[(temp >> 16) & 0xff] & 0xff000000) ^
102                 (Te4[(temp >>  8) & 0xff] & 0x00ff0000) ^
103                 (Te4[(temp      ) & 0xff] & 0x0000ff00) ^
104                 (Te4[(temp >> 24)       ] & 0x000000ff) ^
105                 rcon[i];
106             rek[ 7] = rek[ 1] ^ rek[ 6];
107             rek[ 8] = rek[ 2] ^ rek[ 7];
108             rek[ 9] = rek[ 3] ^ rek[ 8];
109             if (++i == 8) {
110                 Nr = 12;
111                 return;
112             }
113             rek[10] = rek[ 4] ^ rek[ 9];
114             rek[11] = rek[ 5] ^ rek[10];
115             rek += 6;
116         }
117     }
118     rek[6] = GETWORD(cipherKey + 24);
119     rek[7] = GETWORD(cipherKey + 28);
120     if (keyBits == 256) {
121         for (;;) {
122             temp = rek[ 7];
123             rek[ 8] = rek[ 0] ^
124                 (Te4[(temp >> 16) & 0xff] & 0xff000000) ^
125                 (Te4[(temp >>  8) & 0xff] & 0x00ff0000) ^
126                 (Te4[(temp      ) & 0xff] & 0x0000ff00) ^
127                 (Te4[(temp >> 24)       ] & 0x000000ff) ^
128                 rcon[i];
129             rek[ 9] = rek[ 1] ^ rek[ 8];
130             rek[10] = rek[ 2] ^ rek[ 9];
131             rek[11] = rek[ 3] ^ rek[10];
132             if (++i == 7) {
133                 Nr = 14;
134                 return;
```

```
135             }
136             temp = rek[11];
137             rek[12] = rek[ 4] ^
138                 (Te4[(temp >> 24)       ] & 0xff000000) ^
139                 (Te4[(temp >> 16) & 0xff] & 0x00ff0000) ^
140                 (Te4[(temp >>  8) & 0xff] & 0x0000ff00) ^
141                 (Te4[(temp      ) & 0xff] & 0x000000ff);
142             rek[13] = rek[ 5] ^ rek[12];
143             rek[14] = rek[ 6] ^ rek[13];
144             rek[15] = rek[ 7] ^ rek[14];
145             rek += 8;
146         }
147     }
148     Nr = 0; // this should never happen
149 }
150
151 void AES::InvertKey() {
152     uint *rek = e_sched;
153     uint *rdk = d_sched;
154     assert(Nr == 10 || Nr == 12 || Nr == 14);
155     rek += 4*Nr;
156     /* apply the inverse MixColumn transform to all round keys but the first and the last: */
157     memcpy(rdk, rek, 16);
158     rdk += 4;
159     rek -= 4;
160     for (uint r = 1; r < Nr; r++) {
161         rdk[0] =
162             Td0[Te4[(rek[0] >> 24)       ] & 0xff] ^
163             Td1[Te4[(rek[0] >> 16) & 0xff] & 0xff] ^
164             Td2[Te4[(rek[0] >>  8) & 0xff] & 0xff] ^
165             Td3[Te4[(rek[0]      ) & 0xff] & 0xff];
166         rdk[1] =
167             Td0[Te4[(rek[1] >> 24)       ] & 0xff] ^
168             Td1[Te4[(rek[1] >> 16) & 0xff] & 0xff] ^
169             Td2[Te4[(rek[1] >>  8) & 0xff] & 0xff] ^
170             Td3[Te4[(rek[1]      ) & 0xff] & 0xff];
171         rdk[2] =
172             Td0[Te4[(rek[2] >> 24)       ] & 0xff] ^
173             Td1[Te4[(rek[2] >> 16) & 0xff] & 0xff] ^
174             Td2[Te4[(rek[2] >>  8) & 0xff] & 0xff] ^
175             Td3[Te4[(rek[2]      ) & 0xff] & 0xff];
176         rdk[3] =
177             Td0[Te4[(rek[3] >> 24)       ] & 0xff] ^
178             Td1[Te4[(rek[3] >> 16) & 0xff] & 0xff] ^
179             Td2[Te4[(rek[3] >>  8) & 0xff] & 0xff] ^
180             Td3[Te4[(rek[3]      ) & 0xff] & 0xff];
181         rdk += 4;
182         rek -= 4;
183     }
184     memcpy(rdk, rek, 16);
185 }
186
187 ////////////////////////////////////////////////////////////////////////
188 // Public Interface
189 ////////////////////////////////////////////////////////////////////////
190
191 void AES::byte2int(const byte *b, uint *i) {
192     i[0] = GETWORD(b     );
193     i[1] = GETWORD(b +  4);
194     i[2] = GETWORD(b +  8);
195     i[3] = GETWORD(b + 12);
196 }
197
198 void AES::int2byte(const uint *i, byte *b) {
199     PUTWORD(b     , i[0]);
200     PUTWORD(b +  4, i[1]);
201     PUTWORD(b +  8, i[2]);
202     PUTWORD(b + 12, i[3]);
203 }
204
205 void AES::makeKey(const byte *cipherKey, uint keySize, uint dir) {
206     switch (keySize) {
207     case 16:
208     case 24:
209     case 32:
210         keySize <<= 3;
211         break;
212     case 128:
213     case 192:
214     case 256:
215         break;
216     default:
217         throw "Invalid␣AES␣key␣size";
218     }
```

```
219       assert(dir <= DIR_BOTH);
220       if (dir != DIR_NONE) {
221           ExpandKey(cipherKey, keySize);
222           cudaMemcpy(ce_sched, e_sched, sizeof(e_sched), cudaMemcpyHostToDevice);
223           if (dir & DIR_DECRYPT) {
224               InvertKey();
225               cudaMemcpy(cd_sched, d_sched, sizeof(e_sched), cudaMemcpyHostToDevice);
226           }
227       }
228   }
229
230   void AES::encrypt(const uint *pt, uint *ct, uint n = 1) {
231       uint *cpt, *cct;
232       uint size = (n << 2)*sizeof(uint);
233
234       cudaMalloc((void**)&cpt, size);
235       cudaMalloc((void**)&cct, size);
236       cudaMemcpy(cpt, pt, size, cudaMemcpyHostToDevice);
237
238       struct cudaDeviceProp prop;
239       cudaGetDeviceProperties(&prop, 0);
240
241       uint blocks, threads = 1;
242       if(n != 1) {
243           threads = (n < prop.maxThreadsPerBlock*2) ? n / 2 : prop.maxThreadsPerBlock;
244       }
245       blocks = n / threads;
246
247       dim3 dimBlock(threads);
248       dim3 dimGrid(blocks);
249
250       AES_encrypt<<<dimGrid, dimBlock, size>>>(cpt, cct, ce_sched, Nr);
251
252       cudaMemcpy(ct, cct, size, cudaMemcpyDeviceToHost);
253       cudaFree(cpt);
254       cudaFree(cct);
255   }
256
257   void AES::decrypt(const uint *ct, uint *pt, uint n = 1) {
258       uint *cpt, *cct;
259       uint size = (n << 2)*sizeof(uint);
260
261       cudaMalloc((void**)&cpt, size);
262       cudaMalloc((void**)&cct, size);
263       cudaMemcpy(cct, ct, size, cudaMemcpyHostToDevice);
264
265       struct cudaDeviceProp prop;
266       cudaGetDeviceProperties(&prop, 0);
267
268       uint blocks, threads = 1;
269       if(n != 1) {
270           threads = (n < prop.maxThreadsPerBlock*2) ? n / 2 : prop.maxThreadsPerBlock;
271       }
272       blocks = n / threads;
273
274       dim3 dimBlock(threads);
275       dim3 dimGrid(blocks);
276
277       AES_decrypt<<<dimGrid, dimBlock, size>>>(cct, cpt, cd_sched, Nr);
278
279       cudaMemcpy(pt, cpt, size, cudaMemcpyDeviceToHost);
280       cudaFree(cpt);
281       cudaFree(cct);
282   }
283
284   __global__ void AES_encrypt(const uint *pt, uint *ct, uint *rek, uint Nr) {
285       int x = blockIdx.x * blockDim.x + threadIdx.x;
286       int y = blockIdx.y * blockDim.y + threadIdx.y;
287       int i = x + y * gridDim.x * blockDim.x;
288       int offset = i << 2;
289
290       __shared__ __device__ uint s0, s1, s2, s3, t0, t1, t2, t3;
291
292       s0 = pt[offset + 0] ^ rek[0];
293       s1 = pt[offset + 1] ^ rek[1];
294       s2 = pt[offset + 2] ^ rek[2];
295       s3 = pt[offset + 3] ^ rek[3];
296
297       /* round 1: */
298       t0 = cTe0[s0 >> 24] ^ cTe1[(s1 >> 16) & 0xff] ^ cTe2[(s2 >>  8) & 0xff] ^ cTe3[s3 & 0xff] ^ rek[ 4];
299       t1 = cTe0[s1 >> 24] ^ cTe1[(s2 >> 16) & 0xff] ^ cTe2[(s3 >>  8) & 0xff] ^ cTe3[s0 & 0xff] ^ rek[ 5];
300       t2 = cTe0[s2 >> 24] ^ cTe1[(s3 >> 16) & 0xff] ^ cTe2[(s0 >>  8) & 0xff] ^ cTe3[s1 & 0xff] ^ rek[ 6];
301       t3 = cTe0[s3 >> 24] ^ cTe1[(s0 >> 16) & 0xff] ^ cTe2[(s1 >>  8) & 0xff] ^ cTe3[s2 & 0xff] ^ rek[ 7];
302       /* round 2: */
```

```
303       s0 = cTe0[t0 >> 24] ^ cTe1[(t1 >> 16) & 0xff] ^ cTe2[(t2 >>  8) & 0xff] ^ cTe3[t3 & 0xff] ^ rek[ 8];
304       s1 = cTe0[t1 >> 24] ^ cTe1[(t2 >> 16) & 0xff] ^ cTe2[(t3 >>  8) & 0xff] ^ cTe3[t0 & 0xff] ^ rek[ 9];
305       s2 = cTe0[t2 >> 24] ^ cTe1[(t3 >> 16) & 0xff] ^ cTe2[(t0 >>  8) & 0xff] ^ cTe3[t1 & 0xff] ^ rek[10];
306       s3 = cTe0[t3 >> 24] ^ cTe1[(t0 >> 16) & 0xff] ^ cTe2[(t1 >>  8) & 0xff] ^ cTe3[t2 & 0xff] ^ rek[11];
307       /* round 3: */
308       t0 = cTe0[s0 >> 24] ^ cTe1[(s1 >> 16) & 0xff] ^ cTe2[(s2 >>  8) & 0xff] ^ cTe3[s3 & 0xff] ^ rek[12];
309       t1 = cTe0[s1 >> 24] ^ cTe1[(s2 >> 16) & 0xff] ^ cTe2[(s3 >>  8) & 0xff] ^ cTe3[s0 & 0xff] ^ rek[13];
310       t2 = cTe0[s2 >> 24] ^ cTe1[(s3 >> 16) & 0xff] ^ cTe2[(s0 >>  8) & 0xff] ^ cTe3[s1 & 0xff] ^ rek[14];
311       t3 = cTe0[s3 >> 24] ^ cTe1[(s0 >> 16) & 0xff] ^ cTe2[(s1 >>  8) & 0xff] ^ cTe3[s2 & 0xff] ^ rek[15];
312       /* round 4: */
313       s0 = cTe0[t0 >> 24] ^ cTe1[(t1 >> 16) & 0xff] ^ cTe2[(t2 >>  8) & 0xff] ^ cTe3[t3 & 0xff] ^ rek[16];
314       s1 = cTe0[t1 >> 24] ^ cTe1[(t2 >> 16) & 0xff] ^ cTe2[(t3 >>  8) & 0xff] ^ cTe3[t0 & 0xff] ^ rek[17];
315       s2 = cTe0[t2 >> 24] ^ cTe1[(t3 >> 16) & 0xff] ^ cTe2[(t0 >>  8) & 0xff] ^ cTe3[t1 & 0xff] ^ rek[18];
316       s3 = cTe0[t3 >> 24] ^ cTe1[(t0 >> 16) & 0xff] ^ cTe2[(t1 >>  8) & 0xff] ^ cTe3[t2 & 0xff] ^ rek[19];
317       /* round 5: */
318       t0 = cTe0[s0 >> 24] ^ cTe1[(s1 >> 16) & 0xff] ^ cTe2[(s2 >>  8) & 0xff] ^ cTe3[s3 & 0xff] ^ rek[20];
319       t1 = cTe0[s1 >> 24] ^ cTe1[(s2 >> 16) & 0xff] ^ cTe2[(s3 >>  8) & 0xff] ^ cTe3[s0 & 0xff] ^ rek[21];
320       t2 = cTe0[s2 >> 24] ^ cTe1[(s3 >> 16) & 0xff] ^ cTe2[(s0 >>  8) & 0xff] ^ cTe3[s1 & 0xff] ^ rek[22];
321       t3 = cTe0[s3 >> 24] ^ cTe1[(s0 >> 16) & 0xff] ^ cTe2[(s1 >>  8) & 0xff] ^ cTe3[s2 & 0xff] ^ rek[23];
322       /* round 6: */
323       s0 = cTe0[t0 >> 24] ^ cTe1[(t1 >> 16) & 0xff] ^ cTe2[(t2 >>  8) & 0xff] ^ cTe3[t3 & 0xff] ^ rek[24];
324       s1 = cTe0[t1 >> 24] ^ cTe1[(t2 >> 16) & 0xff] ^ cTe2[(t3 >>  8) & 0xff] ^ cTe3[t0 & 0xff] ^ rek[25];
325       s2 = cTe0[t2 >> 24] ^ cTe1[(t3 >> 16) & 0xff] ^ cTe2[(t0 >>  8) & 0xff] ^ cTe3[t1 & 0xff] ^ rek[26];
326       s3 = cTe0[t3 >> 24] ^ cTe1[(t0 >> 16) & 0xff] ^ cTe2[(t1 >>  8) & 0xff] ^ cTe3[t2 & 0xff] ^ rek[27];
327       /* round 7: */
328       t0 = cTe0[s0 >> 24] ^ cTe1[(s1 >> 16) & 0xff] ^ cTe2[(s2 >>  8) & 0xff] ^ cTe3[s3 & 0xff] ^ rek[28];
329       t1 = cTe0[s1 >> 24] ^ cTe1[(s2 >> 16) & 0xff] ^ cTe2[(s3 >>  8) & 0xff] ^ cTe3[s0 & 0xff] ^ rek[29];
330       t2 = cTe0[s2 >> 24] ^ cTe1[(s3 >> 16) & 0xff] ^ cTe2[(s0 >>  8) & 0xff] ^ cTe3[s1 & 0xff] ^ rek[30];
331       t3 = cTe0[s3 >> 24] ^ cTe1[(s0 >> 16) & 0xff] ^ cTe2[(s1 >>  8) & 0xff] ^ cTe3[s2 & 0xff] ^ rek[31];
332       /* round 8: */
333       s0 = cTe0[t0 >> 24] ^ cTe1[(t1 >> 16) & 0xff] ^ cTe2[(t2 >>  8) & 0xff] ^ cTe3[t3 & 0xff] ^ rek[32];
334       s1 = cTe0[t1 >> 24] ^ cTe1[(t2 >> 16) & 0xff] ^ cTe2[(t3 >>  8) & 0xff] ^ cTe3[t0 & 0xff] ^ rek[33];
335       s2 = cTe0[t2 >> 24] ^ cTe1[(t3 >> 16) & 0xff] ^ cTe2[(t0 >>  8) & 0xff] ^ cTe3[t1 & 0xff] ^ rek[34];
336       s3 = cTe0[t3 >> 24] ^ cTe1[(t0 >> 16) & 0xff] ^ cTe2[(t1 >>  8) & 0xff] ^ cTe3[t2 & 0xff] ^ rek[35];
337       /* round 9: */
338       t0 = cTe0[s0 >> 24] ^ cTe1[(s1 >> 16) & 0xff] ^ cTe2[(s2 >>  8) & 0xff] ^ cTe3[s3 & 0xff] ^ rek[36];
339       t1 = cTe0[s1 >> 24] ^ cTe1[(s2 >> 16) & 0xff] ^ cTe2[(s3 >>  8) & 0xff] ^ cTe3[s0 & 0xff] ^ rek[37];
340       t2 = cTe0[s2 >> 24] ^ cTe1[(s3 >> 16) & 0xff] ^ cTe2[(s0 >>  8) & 0xff] ^ cTe3[s1 & 0xff] ^ rek[38];
341       t3 = cTe0[s3 >> 24] ^ cTe1[(s0 >> 16) & 0xff] ^ cTe2[(s1 >>  8) & 0xff] ^ cTe3[s2 & 0xff] ^ rek[39];
342       if (Nr > 10) {
343           /* round 10: */
344           s0 = cTe0[t0 >> 24] ^ cTe1[(t1 >> 16) & 0xff] ^ cTe2[(t2 >>  8) & 0xff] ^ cTe3[t3 & 0xff] ^ rek[40];
345           s1 = cTe0[t1 >> 24] ^ cTe1[(t2 >> 16) & 0xff] ^ cTe2[(t3 >>  8) & 0xff] ^ cTe3[t0 & 0xff] ^ rek[41];
346           s2 = cTe0[t2 >> 24] ^ cTe1[(t3 >> 16) & 0xff] ^ cTe2[(t0 >>  8) & 0xff] ^ cTe3[t1 & 0xff] ^ rek[42];
347           s3 = cTe0[t3 >> 24] ^ cTe1[(t0 >> 16) & 0xff] ^ cTe2[(t1 >>  8) & 0xff] ^ cTe3[t2 & 0xff] ^ rek[43];
348           /* round 11: */
349           t0 = cTe0[s0 >> 24] ^ cTe1[(s1 >> 16) & 0xff] ^ cTe2[(s2 >>  8) & 0xff] ^ cTe3[s3 & 0xff] ^ rek[44];
350           t1 = cTe0[s1 >> 24] ^ cTe1[(s2 >> 16) & 0xff] ^ cTe2[(s3 >>  8) & 0xff] ^ cTe3[s0 & 0xff] ^ rek[45];
351           t2 = cTe0[s2 >> 24] ^ cTe1[(s3 >> 16) & 0xff] ^ cTe2[(s0 >>  8) & 0xff] ^ cTe3[s1 & 0xff] ^ rek[46];
352           t3 = cTe0[s3 >> 24] ^ cTe1[(s0 >> 16) & 0xff] ^ cTe2[(s1 >>  8) & 0xff] ^ cTe3[s2 & 0xff] ^ rek[47];
353           if (Nr > 12) {
354               /* round 12: */
355               s0 = cTe0[t0 >> 24] ^ cTe1[(t1 >> 16) & 0xff] ^ cTe2[(t2 >>  8) & 0xff] ^ cTe3[t3 & 0xff] ^ rek
                      [48];
356               s1 = cTe0[t1 >> 24] ^ cTe1[(t2 >> 16) & 0xff] ^ cTe2[(t3 >>  8) & 0xff] ^ cTe3[t0 & 0xff] ^ rek
                      [49];
357               s2 = cTe0[t2 >> 24] ^ cTe1[(t3 >> 16) & 0xff] ^ cTe2[(t0 >>  8) & 0xff] ^ cTe3[t1 & 0xff] ^ rek
                      [50];
358               s3 = cTe0[t3 >> 24] ^ cTe1[(t0 >> 16) & 0xff] ^ cTe2[(t1 >>  8) & 0xff] ^ cTe3[t2 & 0xff] ^ rek
                      [51];
359               /* round 13: */
360               t0 = cTe0[s0 >> 24] ^ cTe1[(s1 >> 16) & 0xff] ^ cTe2[(s2 >>  8) & 0xff] ^ cTe3[s3 & 0xff] ^ rek
                      [52];
361               t1 = cTe0[s1 >> 24] ^ cTe1[(s2 >> 16) & 0xff] ^ cTe2[(s3 >>  8) & 0xff] ^ cTe3[s0 & 0xff] ^ rek
                      [53];
362               t2 = cTe0[s2 >> 24] ^ cTe1[(s3 >> 16) & 0xff] ^ cTe2[(s0 >>  8) & 0xff] ^ cTe3[s1 & 0xff] ^ rek
                      [54];
363               t3 = cTe0[s3 >> 24] ^ cTe1[(s0 >> 16) & 0xff] ^ cTe2[(s1 >>  8) & 0xff] ^ cTe3[s2 & 0xff] ^ rek
                      [55];
364           }
365       }
366       rek += Nr << 2;
367
368       ct[offset + 0] =
369           (cTe4[(t0 >> 24)       ] & 0xff000000) ^
370           (cTe4[(t1 >> 16) & 0xff] & 0x00ff0000) ^
371           (cTe4[(t2 >>  8) & 0xff] & 0x0000ff00) ^
372           (cTe4[(t3      ) & 0xff] & 0x000000ff) ^
373           rek[0];
374       ct[offset + 1] =
375           (cTe4[(t1 >> 24)       ] & 0xff000000) ^
376           (cTe4[(t2 >> 16) & 0xff] & 0x00ff0000) ^
377           (cTe4[(t3 >>  8) & 0xff] & 0x0000ff00) ^
378           (cTe4[(t0      ) & 0xff] & 0x000000ff) ^
```

```
379           rek[1];
380       ct[offset + 2] =
381           (cTe4[(t2 >> 24)       ] & 0xff000000) ^
382           (cTe4[(t3 >> 16) & 0xff] & 0x00ff0000) ^
383           (cTe4[(t0 >>  8) & 0xff] & 0x0000ff00) ^
384           (cTe4[(t1      ) & 0xff] & 0x000000ff) ^
385           rek[2];
386       ct[offset + 3] =
387           (cTe4[(t3 >> 24)       ] & 0xff000000) ^
388           (cTe4[(t0 >> 16) & 0xff] & 0x00ff0000) ^
389           (cTe4[(t1 >>  8) & 0xff] & 0x0000ff00) ^
390           (cTe4[(t2      ) & 0xff] & 0x000000ff) ^
391           rek[3];
392   }
393
394   __global__ void AES_decrypt(const uint *ct, uint *pt, uint *rdk, uint Nr) {
395       int x = blockIdx.x * blockDim.x + threadIdx.x;
396       int y = blockIdx.y * blockDim.y + threadIdx.y;
397       int i = x + y * gridDim.x * blockDim.x;
398       int offset = i << 2;
399
400       __shared__ __device__ uint s0, s1, s2, s3, t0, t1, t2, t3;
401
402       s0 = ct[offset + 0] ^ rdk[0];
403       s1 = ct[offset + 1] ^ rdk[1];
404       s2 = ct[offset + 2] ^ rdk[2];
405       s3 = ct[offset + 3] ^ rdk[3];
406
407       /* round 1: */
408       t0 = cTd0[s0 >> 24] ^ cTd1[(s3 >> 16) & 0xff] ^ cTd2[(s2 >>  8) & 0xff] ^ cTd3[s1 & 0xff] ^ rdk[ 4];
409       t1 = cTd0[s1 >> 24] ^ cTd1[(s0 >> 16) & 0xff] ^ cTd2[(s3 >>  8) & 0xff] ^ cTd3[s2 & 0xff] ^ rdk[ 5];
410       t2 = cTd0[s2 >> 24] ^ cTd1[(s1 >> 16) & 0xff] ^ cTd2[(s0 >>  8) & 0xff] ^ cTd3[s3 & 0xff] ^ rdk[ 6];
411       t3 = cTd0[s3 >> 24] ^ cTd1[(s2 >> 16) & 0xff] ^ cTd2[(s1 >>  8) & 0xff] ^ cTd3[s0 & 0xff] ^ rdk[ 7];
412       /* round 2: */
413       s0 = cTd0[t0 >> 24] ^ cTd1[(t3 >> 16) & 0xff] ^ cTd2[(t2 >>  8) & 0xff] ^ cTd3[t1 & 0xff] ^ rdk[ 8];
414       s1 = cTd0[t1 >> 24] ^ cTd1[(t0 >> 16) & 0xff] ^ cTd2[(t3 >>  8) & 0xff] ^ cTd3[t2 & 0xff] ^ rdk[ 9];
415       s2 = cTd0[t2 >> 24] ^ cTd1[(t1 >> 16) & 0xff] ^ cTd2[(t0 >>  8) & 0xff] ^ cTd3[t3 & 0xff] ^ rdk[10];
416       s3 = cTd0[t3 >> 24] ^ cTd1[(t2 >> 16) & 0xff] ^ cTd2[(t1 >>  8) & 0xff] ^ cTd3[t0 & 0xff] ^ rdk[11];
417       /* round 3: */
418       t0 = cTd0[s0 >> 24] ^ cTd1[(s3 >> 16) & 0xff] ^ cTd2[(s2 >>  8) & 0xff] ^ cTd3[s1 & 0xff] ^ rdk[12];
419       t1 = cTd0[s1 >> 24] ^ cTd1[(s0 >> 16) & 0xff] ^ cTd2[(s3 >>  8) & 0xff] ^ cTd3[s2 & 0xff] ^ rdk[13];
420       t2 = cTd0[s2 >> 24] ^ cTd1[(s1 >> 16) & 0xff] ^ cTd2[(s0 >>  8) & 0xff] ^ cTd3[s3 & 0xff] ^ rdk[14];
421       t3 = cTd0[s3 >> 24] ^ cTd1[(s2 >> 16) & 0xff] ^ cTd2[(s1 >>  8) & 0xff] ^ cTd3[s0 & 0xff] ^ rdk[15];
422       /* round 4: */
423       s0 = cTd0[t0 >> 24] ^ cTd1[(t3 >> 16) & 0xff] ^ cTd2[(t2 >>  8) & 0xff] ^ cTd3[t1 & 0xff] ^ rdk[16];
424       s1 = cTd0[t1 >> 24] ^ cTd1[(t0 >> 16) & 0xff] ^ cTd2[(t3 >>  8) & 0xff] ^ cTd3[t2 & 0xff] ^ rdk[17];
425       s2 = cTd0[t2 >> 24] ^ cTd1[(t1 >> 16) & 0xff] ^ cTd2[(t0 >>  8) & 0xff] ^ cTd3[t3 & 0xff] ^ rdk[18];
426       s3 = cTd0[t3 >> 24] ^ cTd1[(t2 >> 16) & 0xff] ^ cTd2[(t1 >>  8) & 0xff] ^ cTd3[t0 & 0xff] ^ rdk[19];
427       /* round 5: */
428       t0 = cTd0[s0 >> 24] ^ cTd1[(s3 >> 16) & 0xff] ^ cTd2[(s2 >>  8) & 0xff] ^ cTd3[s1 & 0xff] ^ rdk[20];
429       t1 = cTd0[s1 >> 24] ^ cTd1[(s0 >> 16) & 0xff] ^ cTd2[(s3 >>  8) & 0xff] ^ cTd3[s2 & 0xff] ^ rdk[21];
430       t2 = cTd0[s2 >> 24] ^ cTd1[(s1 >> 16) & 0xff] ^ cTd2[(s0 >>  8) & 0xff] ^ cTd3[s3 & 0xff] ^ rdk[22];
431       t3 = cTd0[s3 >> 24] ^ cTd1[(s2 >> 16) & 0xff] ^ cTd2[(s1 >>  8) & 0xff] ^ cTd3[s0 & 0xff] ^ rdk[23];
432       /* round 6: */
433       s0 = cTd0[t0 >> 24] ^ cTd1[(t3 >> 16) & 0xff] ^ cTd2[(t2 >>  8) & 0xff] ^ cTd3[t1 & 0xff] ^ rdk[24];
434       s1 = cTd0[t1 >> 24] ^ cTd1[(t0 >> 16) & 0xff] ^ cTd2[(t3 >>  8) & 0xff] ^ cTd3[t2 & 0xff] ^ rdk[25];
435       s2 = cTd0[t2 >> 24] ^ cTd1[(t1 >> 16) & 0xff] ^ cTd2[(t0 >>  8) & 0xff] ^ cTd3[t3 & 0xff] ^ rdk[26];
436       s3 = cTd0[t3 >> 24] ^ cTd1[(t2 >> 16) & 0xff] ^ cTd2[(t1 >>  8) & 0xff] ^ cTd3[t0 & 0xff] ^ rdk[27];
437       /* round 7: */
438       t0 = cTd0[s0 >> 24] ^ cTd1[(s3 >> 16) & 0xff] ^ cTd2[(s2 >>  8) & 0xff] ^ cTd3[s1 & 0xff] ^ rdk[28];
439       t1 = cTd0[s1 >> 24] ^ cTd1[(s0 >> 16) & 0xff] ^ cTd2[(s3 >>  8) & 0xff] ^ cTd3[s2 & 0xff] ^ rdk[29];
440       t2 = cTd0[s2 >> 24] ^ cTd1[(s1 >> 16) & 0xff] ^ cTd2[(s0 >>  8) & 0xff] ^ cTd3[s3 & 0xff] ^ rdk[30];
441       t3 = cTd0[s3 >> 24] ^ cTd1[(s2 >> 16) & 0xff] ^ cTd2[(s1 >>  8) & 0xff] ^ cTd3[s0 & 0xff] ^ rdk[31];
442       /* round 8: */
443       s0 = cTd0[t0 >> 24] ^ cTd1[(t3 >> 16) & 0xff] ^ cTd2[(t2 >>  8) & 0xff] ^ cTd3[t1 & 0xff] ^ rdk[32];
444       s1 = cTd0[t1 >> 24] ^ cTd1[(t0 >> 16) & 0xff] ^ cTd2[(t3 >>  8) & 0xff] ^ cTd3[t2 & 0xff] ^ rdk[33];
445       s2 = cTd0[t2 >> 24] ^ cTd1[(t1 >> 16) & 0xff] ^ cTd2[(t0 >>  8) & 0xff] ^ cTd3[t3 & 0xff] ^ rdk[34];
446       s3 = cTd0[t3 >> 24] ^ cTd1[(t2 >> 16) & 0xff] ^ cTd2[(t1 >>  8) & 0xff] ^ cTd3[t0 & 0xff] ^ rdk[35];
447       /* round 9: */
448       t0 = cTd0[s0 >> 24] ^ cTd1[(s3 >> 16) & 0xff] ^ cTd2[(s2 >>  8) & 0xff] ^ cTd3[s1 & 0xff] ^ rdk[36];
449       t1 = cTd0[s1 >> 24] ^ cTd1[(s0 >> 16) & 0xff] ^ cTd2[(s3 >>  8) & 0xff] ^ cTd3[s2 & 0xff] ^ rdk[37];
450       t2 = cTd0[s2 >> 24] ^ cTd1[(s1 >> 16) & 0xff] ^ cTd2[(s0 >>  8) & 0xff] ^ cTd3[s3 & 0xff] ^ rdk[38];
451       t3 = cTd0[s3 >> 24] ^ cTd1[(s2 >> 16) & 0xff] ^ cTd2[(s1 >>  8) & 0xff] ^ cTd3[s0 & 0xff] ^ rdk[39];
452       if (Nr > 10) {
453           /* round 10: */
454           s0 = cTd0[t0 >> 24] ^ cTd1[(t3 >> 16) & 0xff] ^ cTd2[(t2 >>  8) & 0xff] ^ cTd3[t1 & 0xff] ^ rdk[40];
455           s1 = cTd0[t1 >> 24] ^ cTd1[(t0 >> 16) & 0xff] ^ cTd2[(t3 >>  8) & 0xff] ^ cTd3[t2 & 0xff] ^ rdk[41];
456           s2 = cTd0[t2 >> 24] ^ cTd1[(t1 >> 16) & 0xff] ^ cTd2[(t0 >>  8) & 0xff] ^ cTd3[t3 & 0xff] ^ rdk[42];
457           s3 = cTd0[t3 >> 24] ^ cTd1[(t2 >> 16) & 0xff] ^ cTd2[(t1 >>  8) & 0xff] ^ cTd3[t0 & 0xff] ^ rdk[43];
458           /* round 11: */
459           t0 = cTd0[s0 >> 24] ^ cTd1[(s3 >> 16) & 0xff] ^ cTd2[(s2 >>  8) & 0xff] ^ cTd3[s1 & 0xff] ^ rdk[44];
460           t1 = cTd0[s1 >> 24] ^ cTd1[(s0 >> 16) & 0xff] ^ cTd2[(s3 >>  8) & 0xff] ^ cTd3[s2 & 0xff] ^ rdk[45];
461           t2 = cTd0[s2 >> 24] ^ cTd1[(s1 >> 16) & 0xff] ^ cTd2[(s0 >>  8) & 0xff] ^ cTd3[s3 & 0xff] ^ rdk[46];
462           t3 = cTd0[s3 >> 24] ^ cTd1[(s2 >> 16) & 0xff] ^ cTd2[(s1 >>  8) & 0xff] ^ cTd3[s0 & 0xff] ^ rdk[47];
```

```
463           if (Nr > 12) {
464               /* round 12: */
465               s0 = cTd0[t0 >> 24] ^ cTd1[(t3 >> 16) & 0xff] ^ cTd2[(t2 >>  8) & 0xff] ^ cTd3[t1 & 0xff] ^ rdk
                      [48];
466               s1 = cTd0[t1 >> 24] ^ cTd1[(t0 >> 16) & 0xff] ^ cTd2[(t3 >>  8) & 0xff] ^ cTd3[t2 & 0xff] ^ rdk
                      [49];
467               s2 = cTd0[t2 >> 24] ^ cTd1[(t1 >> 16) & 0xff] ^ cTd2[(t0 >>  8) & 0xff] ^ cTd3[t3 & 0xff] ^ rdk
                      [50];
468               s3 = cTd0[t3 >> 24] ^ cTd1[(t2 >> 16) & 0xff] ^ cTd2[(t1 >>  8) & 0xff] ^ cTd3[t0 & 0xff] ^ rdk
                      [51];
469               /* round 13: */
470               t0 = cTd0[s0 >> 24] ^ cTd1[(s3 >> 16) & 0xff] ^ cTd2[(s2 >>  8) & 0xff] ^ cTd3[s1 & 0xff] ^ rdk
                      [52];
471               t1 = cTd0[s1 >> 24] ^ cTd1[(s0 >> 16) & 0xff] ^ cTd2[(s3 >>  8) & 0xff] ^ cTd3[s2 & 0xff] ^ rdk
                      [53];
472               t2 = cTd0[s2 >> 24] ^ cTd1[(s1 >> 16) & 0xff] ^ cTd2[(s0 >>  8) & 0xff] ^ cTd3[s3 & 0xff] ^ rdk
                      [54];
473               t3 = cTd0[s3 >> 24] ^ cTd1[(s2 >> 16) & 0xff] ^ cTd2[(s1 >>  8) & 0xff] ^ cTd3[s0 & 0xff] ^ rdk
                      [55];
474           }
475       }
476       rdk += Nr << 2;
477
478       pt[offset + 0] =
479           (cTd4[(t0 >> 24)        ] & 0xff000000) ^
480           (cTd4[(t3 >> 16) & 0xff] & 0x00ff0000) ^
481           (cTd4[(t2 >>  8) & 0xff] & 0x0000ff00) ^
482           (cTd4[(t1       ) & 0xff] & 0x000000ff) ^
483           rdk[0];
484       pt[offset + 1] =
485           (cTd4[(t1 >> 24)        ] & 0xff000000) ^
486           (cTd4[(t0 >> 16) & 0xff] & 0x00ff0000) ^
487           (cTd4[(t3 >>  8) & 0xff] & 0x0000ff00) ^
488           (cTd4[(t2       ) & 0xff] & 0x000000ff) ^
489           rdk[1];
490       pt[offset + 2] =
491           (cTd4[(t2 >> 24)        ] & 0xff000000) ^
492           (cTd4[(t1 >> 16) & 0xff] & 0x00ff0000) ^
493           (cTd4[(t0 >>  8) & 0xff] & 0x0000ff00) ^
494           (cTd4[(t3       ) & 0xff] & 0x000000ff) ^
495           rdk[2];
496       pt[offset + 3] =
497           (cTd4[(t3 >> 24)        ] & 0xff000000) ^
498           (cTd4[(t2 >> 16) & 0xff] & 0x00ff0000) ^
499           (cTd4[(t1 >>  8) & 0xff] & 0x0000ff00) ^
500           (cTd4[(t0       ) & 0xff] & 0x000000ff) ^
501           rdk[3];
502   }
```