UNIVERSITÀ DEGLI STUDI DI MILANO

Dipartimento di Informatica

CORSO DI LAUREA MAGISTRALE IN INFORMATICA

PROGETTO DI GPU COMPUTING

Parallelizzazione dell'algoritmo Advanced Encryption Standard (AES) su architettura CUDA

A cura di
Andrea Ceccarelli
Tommaso Celata

Docente
Giuliano Grossi

Anno Accademico 2015/2016

Indice

1 Introduzione **2**

2 Advanced Encryption Standard (AES) **3**

2.1 Caratteristiche 3

2.1.1 La chiave di sessione 3

2.1.2 Lo stato 3

2.2 Descrizione dell'algoritmo 4

2.2.1 Descrizione ad alto livello 4

2.2.2 AddRoundKey 5

2.2.3 SubBytes 5

2.2.4 ShiftRows 6

3 Architettura CUDA **7**

4 Parallelizzazione AES in CUDA **8**

5 Risultati **9**

6 Considerazioni **10**

Capitolo 1

Introduzione

Il progetto da noi realizzato si pone l'obiettivo di verificare i vantaggi che l'architettura CUDA può portare nella parallelizzazione di algoritmi che presentano una sostanziosa parte seriale che può essere eseguita parallelamente. Per tale motivo si è scelto l'algoritmo Advanced Encryption Standard **(AES)** che cifra stati di dimensione fissa senza concatenare i risultati tra loro dandoci la possibilità di ottenere un buon livello di parallelizzazione.

In una prima fase si è implementato l'algoritmo in linguaggio c verificandone la corretta esecuzione tramite vettori di test trovati online, poi si è modificato il codice per adattarsi e sfruttare l'architettura **CUDA**. Una prima implementazione non è stata sufficiente per ottenere dei vantaggi rispetto alla versione c, questo dovuto al fatto che venivano lanciati **troppi kernel** che creando un collo di bottiglia rendevano inutile il vantaggio portato dalla parallelizzazione. Andando avanti con le varie versioni si è diminuito sostanzialmente il numero di kernel lanciati arrivando ad ottenere buoni risultati.

Spiegheremo quindi le caratteristiche delle varie versioni implementate concentrandoci sull'argomento **parallelizzazione** e su quali vantaggi (o svantaggi) si sono ottenuti da una versione all'altra.

Capitolo 2

Advanced Encryption Standard (AES)

2.1 Caratteristiche

Sviluppato dai due crittografi belgi Joan Daemen e Vincent Rijmen l'Advanced Encryption Standard (AES), conosciuto anche come Rijndael, di cui più propriamente è una specifica implementazione, è un algoritmo di cifratura a blocchi utilizzato come standard dal governo degli Stati Uniti d'America.

Data la sua sicurezza e le sue specifiche pubbliche si presume che in un prossimo futuro venga utilizzato in tutto il mondo come è successo al suo predecessore, il Data Encryption Standard (DES) che ha perso poi efficacia per vulnerabilità intrinseche. AES è stato adottato dalla National Institute of Standards and Technology (NIST) e dalla US FIPS PUB nel novembre del 2001 dopo 5 anni di studi, standardizzazioni e selezione finale tra i vari algoritmi proposti.

2.1.1 La chiave di sessione

AES usa un **key schedule** per espandere una chiave primaria corta in un certo numero di chiavi di **ciclo** differenti.

2.1.2 Lo stato

AES opera utilizzando matrici di 4x4 byte chiamate **stati**. Quando l'algoritmo ha blocchi di 128 bit in input, la matrice di stato ha 4 righe e 4 colonne; se il numero di blocchi in input diventa di 32 bit più lungo, viene aggiunta una

colonna allo stato, e così via fino a 256 bit. In pratica, si divide il numero di bit del blocco in input per 32 e il quoziente specifica il numero di colonne.

2.2 Descrizione dell'algoritmo

L'algoritmo sfrutta diverse funzioni che modificano i valori della matrice di stato mantenendone però la dimensione. Le singole funzioni vengono poi ripetute in un certo ordine per costruire l'algoritmo vero e proprio.

2.2.1 Descrizione ad alto livello

KeyExpansions—round keys are derived from the cipher key using Rijndael's key schedule. AES requires a separate 128-bit round key block for each round plus one more. InitialRound AddRoundKey—each byte of the state is combined with a block of the round key using bitwise xor. Rounds SubBytes—a non-linear substitution step where each byte is replaced with another according to a lookup table. ShiftRows—a transposition step where the last three rows of the state are shifted cyclically a certain number of steps. MixColumns—a mixing operation which operates on the columns of the state, combining the four bytes in each column. AddRoundKey Final Round (no MixColumns) SubBytes ShiftRows AddRoundKey.

1. **KeyExpansions** La chiave di cifratura viene espansa dal key schedule per generare una chiave più grande contenente tutte le chiavi di ciclo.
2. **Round iniziale**
   (a) *AddRoundKey* Ogni byte dello stato viene combinato con il byte corrispondente della chiave di ciclo tramite uno XOR
3. **Rounds**
   (a) *SubBytes* Ogni byte viene sostituito con un altro secondo delle tabelle.
   (b) *ShiftRows* Una trasposizione dove le ultime tre righe dello stato sono shiftate a sinistra un certo numero di volte.
   (c) *MixColumns* Opera sulle colonne combinandone i byte.
   (d) *AddRoundKey*
4. **Round Finale (senza MixColumns)**
   (a) *SubBytes*
   (b) *ShiftRows*
   (c) *AddRoundKey*

2.2.2 AddRoundKey

Nella fase AddRoundKey, la chiave di sessione viene combinata con lo stato. Per ogni round viene derivata una sottochiave dalla chiave originaria usando il key schedule; ogni sottochiave è della stessa dimensione dello stato. La sottochiave è aggiunta allo stato combinando ogni byte di questo con il corrispondente byte della chiave con uno **XOR**.

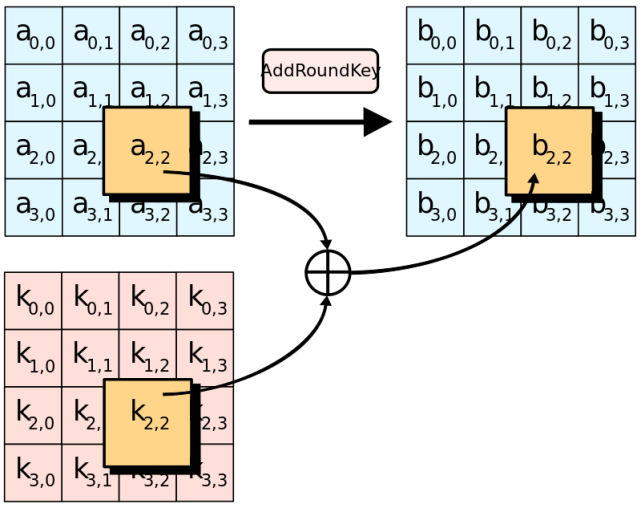

Figura 2.1: Nel passaggio AddRoundKeys ogni byte della matrice viene combinato con la sua sottochiave tramite un'operazione di XOR.

2.2.3 SubBytes

In SubBytes ogni byte $a_{i,j}$ della matrice di stato è sostituito con il corrispondente byte di una matrice chiamata **S-box**.

$$b_{i,j} = S(a_{i,j})$$

Questa operazione garantisce la non-linearità della cifratura. La S-box utilizzata è derivata da una funzione inversa nel campo finito GF(2^8), conosciuta per avere delle ottime proprietà di non linearità. Per evitare un potenziale attacco basato sulle proprietà algebriche la S-box è costruita combinando la funzione inversa con una trasformazione affine invertibile. La S-box è stata scelta con cura per non possedere né punti fissi né punti fissi opposti.

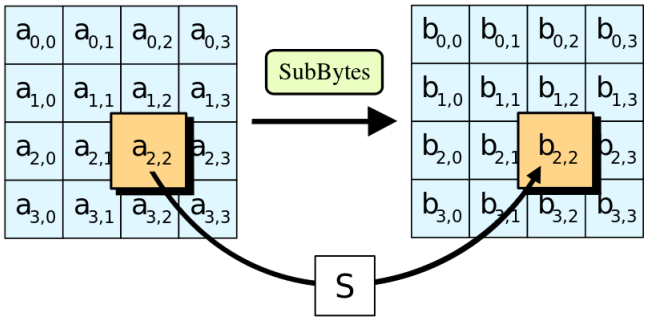

Figura 2.2: Nel passaggio SubBytes, ogni byte della matrice è sostituito con i dati contenuti nella trasformazione S; $b_{ij} = S(a_{ij})$.

2.2.4 ShiftRows

ShiftRows opera sulle righe dello stato shiftando ciclicamente i byte di ogni riga di un certo offset dipendente dal numero di riga lasciando la prima riga invariata. Ogni byte della seconda riga è ciclicamente spostato a sinistra di una posizione. Similmente i byte della terza riga sono shiftati ciclicamente a sinistra di due posizioni e quelli della quarta di tre.

In questo modo l'ultima colonna dei dati in ingresso andrà a formare la diagonale della matrice in uscita. (Rijndael utilizza un disegno leggermente diverso per via delle matrici di lunghezza non fissa.)

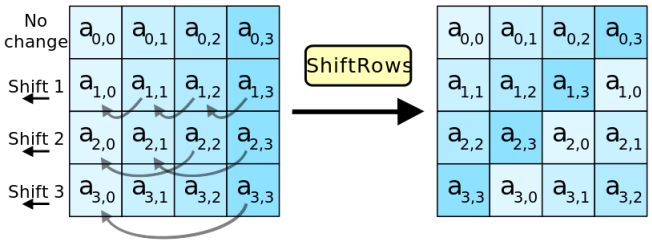

Figura 2.3: Nel passaggio ShiftRows, i byte di ogni riga vengono spostati verso sinistra dell'ordine della riga. Vedi figura per i singoli spostamenti.

2.2.5 MixColumns

Il passaggio MixColumns prende i quattro byte di ogni colonna e li combina utilizzando una trasformazione lineare invertibile. Utilizzati in congiunzione, ShiftRows e MixColumns provvedono a far rispettare il criterio di confusione e diffusione nell'algoritmo (teoria di Shannon). Ogni colonna è trattata come un polinomio in $GF(2^8)$ e viene moltiplicata modulo $x^4 + 1$ per un polinomio fisso $c(x) = 3x^3 + x^2 + x + 2$.

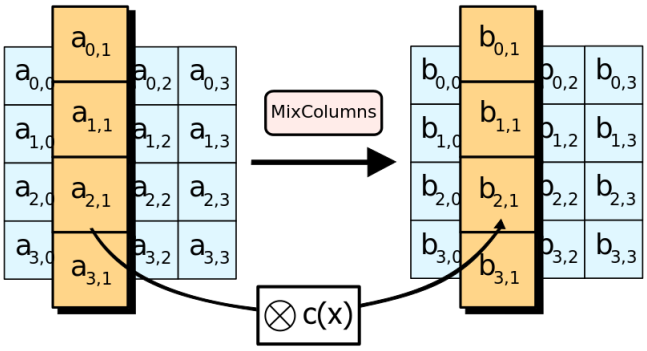

Figura 2.4: Nel passaggio MixColumns ogni colonna di byte viene moltiplicata per un polinomio fisso $c(x)$.

Capitolo 3

Architettura CUDA

Capitolo 4

Parallelizzazione AES in CUDA

Capitolo 5

Risultati

Capitolo 6

Considerazioni

Bibliografia

[1] Kenneth Price , Rainer M. Storn , Jouni A. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, 2005

[2] Janez Brest, Mirjam Sepesy Maucec, *Self-adaptive Differential Evolution Algorithm using Population size Reduction and Three Strategies*. Springer Novembre 2011, Volume 15, Issue 11, pp 2157-2174

[3] Ivan Gerace, Francesca Martinelli, Patrizia Pucci, *Tecniche di Regolarizzazione in Elaborazione di Immagini*. Giornate di Algebra Lineare e Applicazioni, 2009.

[4] Swatagam Das, Ponnuthurai Nagaratnam Suganthan, *Differential Evolution: A Survey of the State-of-the-Art*. IEEE Transactions on Evolutionary Computation, vol. 15, no. 1, pp. 4-31, 2011.

[5] P.C. Hansen, J.G. Nagy, D.P. O'Leary, *Deblurring Images. Matrices, Spectra and Filtering*. SIAM Publisher, Philadelphia, 2006