GPU を用いたハッシュ関数 JH の実装

別所　佑樹 †　　桑門　秀典 †　　森井　昌克 †

† 神戸大学大学院工学研究科
〒 657-8501 兵庫県神戸市灘区六甲台町 1-1

y.betsushiyo@stu.kobe-u.ac.jp　{kuwakado, mmorii}@kobe-u.ac.jp

あらまし　GPU は単純で並列性の高い演算処理を行う場合には高い処理性能を出すことが可能である．暗号化関数は比較的単純な処理が多く，複雑な条件分岐もないため，GPU での処理に適している．実際，GPU を用いた暗号化関数の実装が報告されている．本稿では，統合開発環境 CUDA を利用し，GPU 上で演算を行うハッシュ関数 JH の実装を行った．リファレンス実装とビットスライス実装それぞれについて計算機実験を行い，ハッシュ演算に要する実行時間の比較を行う．それにより GPU でもビットスライス実装による高速化が効果的であることを示す．

Implementation of Hash Function JH on Graphics Processing Units

Yuki Bessho†　　Hidenori Kuwakado†　　Masakatu Morii†

†Graduate School of Engineering, Kobe University
1-1 Rokkodai-cho Nada-ku Kobe 657-8501, Japan

y.betsushiyo@stu.kobe-u.ac.jp　{kuwakado, mmorii}@kobe-u.ac.jp

Abstract　A graphic processing unit (GPU), which is originally a specialized processor that offloads 3D graphics rendering, has been used in performing computation in applications traditionally handled by the CPU recently. Since cryptographic functions use only basic operations, their computation is probably suitable for the GPU. Indeed, implementations of cryptographic functions on GPUs have been reported. In this paper, we implement the JH hash function, which is one of SHA-3 candidates, on GPUs and compare the reference implementation with the bit-slice one in terms of the speed of hashing. Our results show that the design of the JH hash function is suitable for the bit-slice implementation on not only the Intel processor but also GPUs.

1　はじめに

CPU(Central Processing Unit) は加速度的に性能が向上してきた．近年では一つのチップ上に複数個のコアを搭載したマルチコア CPU が普及しており，同時マルチスレッディング技術である Hyper-Threading Technology(HTT) を実装することで処理性能がさらに向上してきている．一方でグラフィックスの処理を担う GPU(Graphics Processing Unit) も同様に性能が向上してきている．GPU は単純な演算処理に特化しているため，一つ一つのプロセッサコアの性能は CPU よりも劣る．しかし，GPU は CPU よりも多数のプロセッサコアを搭載しており，単純で並列性の高い演算処理を行う場合

は同規模の CPU よりも高い処理性能を出すことが可能である．

近年，GPU を用いた汎用的な計算を行うための計算機環境が整いつつあり，GPU を利用した高速計算が着目されている．GPU を利用した様々なソフトウエアアプリケーションがあるが，暗号化関数は，単純な演算の繰り返しが多く，条件分岐もないため，GPU での処理に適したアプリケーションと考えられる．実際，GPU を用いた暗号化関数の実装が報告されている [1][2][3].

本稿では，統合開発環境 CUDA[4] を利用し，GPU 上で演算を行うハッシュ関数 JH[5] を実装した．GPU プログラム特有の CPU-GPU 間でのデータ転送に要する時間はハッシュ演算に要する時間に比べ，無視できる程度であることを確認した．また，ハッシュ関数 JH はビットスライス実装が可能である．一般にビットスライス実装は効果的な高速化手法として知られている．GPU 上で JH のビットスライス実装を行い，GPU プログラムでもビットスライス実装による高速化が有効であることを示す.

2 ハッシュ関数 JH

JH は NIST が公募している新しいハッシュ関数アルゴリズム SHA-3 候補の 1 つである．2^{64} ビット未満の長さの入力メッセージに対し，出力がそれぞれ 224 ビット，256 ビット，384 ビット，512 ビットである JH-224，JH-256，JH-384，JH-512 の 4 種類の仕様がある．

2.1 JH のアルゴリズム

入力メッセージは 512 ビットごとに処理されるため，512 ビットの整数倍となるようにパディング処理を行う．パディング処理後のメッセージは 512 ビットのメッセージブロック $M^{(i)}$ (i= 1, 2, ⋯, N) に分けられ，1024 ビットの初期値とともに圧縮関数に入力される．圧縮関数は 35 ラウンドからなるラウンド関数と Sbox で構成される．すべてのメッセージブロックの処理が終わり，得られた 1024 ビットのハッシュ値の最下位ビットから 224 ビット，256 ビット，384 ビット，512 ビットがそれぞれ JH-224，JH-256，JH-384，JH-512 のメッセージダイジェストである．次に JH で用いられる関数の一部を示す．なお，JH の詳細なアルゴリズムは参考文献 [5] に示されている．

2.1.1 Sbox

S_0 と S_1 は JH で用いられる Sbox である．仕様を表 1 に示す．4 ビットの入力に対し，表 1 を参照することで 4 ビットの出力が得られる．

2.1.2 Linear transform L

関数 L は 4 ビットワード A, B, C, D を用いて以下の式で表わされる．

$$\begin{aligned}(C, D) &= L(A, B) \\ &= (5\bullet A+2\bullet B, 2\bullet A+B).\end{aligned} \tag{1}$$

ここで，$\bullet$ は $GF(2^4)$ 上での乗算である．

2.1.3 Permutation P

置換関数 P は 1024 ビットの入力に対し，入力系列をスワップした 1024 ビットの系列を出力する．入力を $A=(a_0\|a_1\|\cdots\|a_{255})$，出力を $B=(b_0\|b_1\|\cdots\|b_{255})$，$v_i, w_i (0 \le i \le 255)$ を 4 ビット変数とすると，B=$P(A)$ は以下の式で表わされる．

1.
$$\begin{aligned} v_{4i+0} &= a_{4i+0} && (0 \le i \le 63), \\ v_{4i+1} &= a_{4i+1} && (0 \le i \le 63), \\ v_{4i+2} &= a_{4i+3} && (0 \le i \le 63), \\ v_{4i+3} &= a_{4i+2} && (0 \le i \le 63). \end{aligned} \tag{2}$$

2.
$$\begin{aligned} w_i &= v_{2i} && (0 \le i \le 127), \\ w_{i+128} &= v_{2i+1} && (0 \le i \le 127). \end{aligned} \tag{3}$$

3.
$$\begin{aligned} b_i &= w_i && (0 \le i \le 127), \\ b_{2i+0} &= w_{2i+1} && (64 \le i \le 127), \\ b_{2i+1} &= w_{2i+0} && (64 \le i \le 127). \end{aligned} \tag{4}$$

表 1: Sbox.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$S_0(x)$	9	0	4	11	13	12	3	15	1	10	2	6	7	5	8	14
$S_1(x)$	3	12	6	13	5	7	1	9	15	2	0	4	11	10	14	8

2.1.4 Round function R

ラウンド関数 R は Sbox，関数 L，置換関数 P から構成され，入力と出力はともに 1024 ビットである．a_i,b_i を 4 ビット変数とし，入力を $A=(a_0\|a_1\|\cdots\|a_{255})$，出力を $B=(b_0\|b_1\|\cdots\|b_{255})$ とする．また，各ラウンドで用いられる 256 ビットのラウンド定数 $C_r(0 \le r \le 34)$ を $C_r=(C_r^0\|C_r^1\|\cdots\|C_r^{255})$，$v_i, w_i(0\le i\le 255)$ を 4 ビット変数とすると，$B=R(A,C_r)$ は以下の式で表わされる．

1. for $i=0$ to 255,
   {
   if $C_r^i=0$, then $v_i=S_0(a_i)$, (5)
   if $C_r^i=1$, then $v_i=S_1(a_i)$. (6)
   }
2. $(w_{2i}, w_{2i+1})=L(v_{2i}, v_{2i+1}) \quad (0\le i\le 127)$. (7)
3. $(b_0, b_1, \cdots, b_{255})=P(w_0, w_1, \cdots, w_{255})$. (8)

2.2 JH のビットスライス実装

JH はビットスライス実装が可能である．ビットスライス実装を行うことで，前節で示したアルゴリズムの実装よりも少ない演算量で演算を行うことができる．これによりハッシュ演算が高速化される．

2.2.1 Sbox S^{bitsli}

表 1 に示される Sbox を用いた式 (5),(6) は，128 ビットの内部変数 $x_i(0\le i\le 3)$，定数 c，テンポラリ変数 t を用いて以下の式で書き換えることができる．これにより 256 回の表参照が 11 行の論理演算式で表現できる．

1. $x_3=\neg x_3$,
2. $x_0=x_0\oplus(c\wedge(\neg x_2))$,
3. $t=c\oplus(x_0\wedge x_1)$,
4. $x_0=x_0\oplus(x_2\wedge x_3)$,
5. $x_3=x_3\oplus((\neg x_1)\wedge x_2)$,
6. $x_1=x_1\oplus(x_0\wedge x_2)$,
7. $x_2=x_2\oplus(x_0\wedge(\neg x_3))$,
8. $x_0=x_0\oplus(x_1|x_3)$,
9. $x_3=x_3\oplus(x_1\wedge x_2)$,
10. $x_1=x_1\oplus(t\wedge x_0)$,
11. $x_2=x_2\oplus t$.

2.2.2 Linear transform L^{bitsli}

式 (1) で表わされる関数 L は，128 ビット変数 a_i,$b_i(0\le i\le 7)$ を用いて以下の式で書き換えることができる．

1. $b_4=a_4\oplus a_1$,
2. $b_5=a_5\oplus a_2$,
3. $b_6=a_6\oplus a_3\oplus a_0$,
4. $b_7=a_7\oplus a_0$,
5. $b_0=a_0\oplus b_5$,
6. $b_1=a_1\oplus b_6$,
7. $b_2=a_2\oplus b_7\oplus b_4$,
8. $b_3=a_3\oplus b_4$.

2.2.3 Round function R^{bitsli}

ラウンド関数 R は置換関数 P のビットスライス実装 ω，ビットスライス実装で用いられるラウンド定数 C'_r を用いて以下の式で書き換えることができる．ここで，入力は $A=(a_0\|a_1\|\cdots\|a_7)$，出力は $B=(b_0\|b_1\|\cdots\|b_7)$ とし，$a_i, b_i, v_i, u_i(0\le i\le 7)$ はそれぞれ 128 ビット変数である．また，$C'_{r,even}=({C'_r}^0\|{C'_r}^2\|\cdots\|{C'_r}^{254})$，$C'_{r,odd}=({C'_r}^1\|{C'_r}^3\|\cdots\|{C'_r}^{255})$ である．

1. $(v_0, v_2, v_4, v_6)=S^{bitsli}(a_0, a_2, a_4, a_6, C'_{r,even})$, (9)
   $(v_1, v_3, v_5, v_7)=S^{bitsli}(a_1, a_3, a_5, a_7, C'_{r,odd})$. (10)

2. $(u_0, u_2, u_4, u_6, u_1, u_3, u_5, u_7)$
$= L^{bitsli}(v_0, v_2, v_4, v_6, v_1, v_3, v_5, v_7)$. (11)
3. $b_{2i} = u_{2i} \quad (0 \leq i \leq 3)$, (12)
$b_1 = \omega(u_1, 2^{r \bmod 7})$, (13)
$b_3 = \omega(u_3, 2^{r \bmod 7})$, (14)
$b_5 = \omega(u_5, 2^{r \bmod 7})$, (15)
$b_7 = \omega(u_7, 2^{r \bmod 7})$. (16)

3 CUDA

CUDA とはNVIDIA 社が無料で提供する GPU 向け統合開発環境である．GeForce 8 世代以降の GPU で利用でき，専用のコンパイラやライブラリから構成される．

GPU 上で実行される関数をカーネルと呼ぶ．一度に実行されるカーネルは一つなので，異なる機能を持つカーネルを並列実行することはできない．しかし，同じカーネルであれば数千のデータに対して並列処理が可能である．以下に CUDA の基本的な処理の流れを示す．

Step 1 GPU 上にメモリを確保し，データをコピーする．

Step 2 CPU からカーネルを呼び出す．

Step 3 GPU 上でカーネルが実行される．

Step 4 演算結果を CPU のメモリにコピーする．

CPU プログラムとの相違点として，Step 1 と Step 4 の CPU-GPU 間のデータ転送が挙げられる．カーネルでの処理に必要なデータ容量が増加すると，それに伴いデータ転送に要する時間も増加する．

CPU-GPU 間におけるデータの転送やカーネルの実行などは CUDA 特有のプログラミングが必要だが，GPU 上で実行されるカーネルなどは C 言語のように記述することが可能である．

4 GPU を用いた JH の実装

2 章で述べた JH のリファレンス実装とビットスライス実装を行った．それぞれ CPU プログラムと GPU プログラムを作成し，計算機実験を行った．使用した計算機は CPU Intel Core i7 2.67GHz, メモリ 3GB, GPU GeForce GTX 285 と CPU Intel Atom 230 1.60GHz, メモリ 2GB, GPU ION Integrated，実験環境は CUDA2.3 である．

表 2: リファレンス実装の実行時間比較 (CPU : Core i7, GPU : GeForce GTX 285).

(a) JH-256

メッセージサイズ [KB]	GPU による実行時間 [ms]	CPU による実行時間 [ms]
1	513.65	1.50
8	3636.53	9.97
16	7247.24	19.74
32	14294.60	39.18
64	28539.31	78.13
128	57004.59	156.02
256	114066.08	311.77

(b) JH-512

メッセージサイズ [KB]	GPU による実行時間 [ms]	CPU による実行時間 [ms]
1	514.01	1.50
8	3642.48	10.03
16	7251.34	19.74
32	14314.36	39.27
64	28566.41	78.11
128	57131.87	155.93
256	114203.57	311.71

4.1 リファレンス実装による比較

CUDA 特有の CPU-GPU 間のデータ転送に要する時間がハッシュ演算全体の実行時間に占める割合を計算機実験により調査した．その結果，データ転送に要する時間は 1 [ms] 以下で全体の実行時間に比べ，無視できる程度であることがわかった．

表 2 に CPU Core i7, GPU GeForce GTX 285 を用いた場合の JH-256 と JH-512 のリファレンス実装の実行時間比較の結果を，表 3 に CPU

表 3: リファレンス実装の実行時間比較 (CPU : Atom 230, GPU : ION Integrated).

(a) JH-256

メッセージサイズ [KB]	GPU による実行時間 [ms]	CPU による実行時間 [ms]
1	1363.72	5.46
8	9620.03	38.44
16	19088.16	76.28
32	38024.47	151.90
64	75897.07	303.17
128	151651.46	605.70
256	303153.74	1210.93

(b) JH-512

メッセージサイズ [KB]	GPU による実行時間 [ms]	CPU による実行時間 [ms]
1	1363.41	5.46
8	9619.33	38.44
16	19087.13	76.25
32	38022.61	151.86
64	75893.66	303.18
128	151638.34	605.79
256	303123.71	1210.92

Atom 230, GPU ION Integrated を用いた場合の JH-256 と JH-512 のリファレンス実装の実行時間比較の結果を示す．なお，入力メッセージは 1〜256 [KB] とし，いずれも 100 回試行の平均実行時間を求めた．

JH-256 と JH-512 はハッシュアルゴリズムにほとんど違いがないため，実行時間はほとんど同じ結果となった．また，CPU Core i7 GPU と GeForce GTX 285 ではおよそ 360 倍，CPU Atom 230 と GPU ION Integrated では 250 倍の演算性能の差がみられた．

4.2 ビットスライス実装による比較

リファレンス実装の場合と同様に，CPU-GPU 間でのデータ転送に要する時間はハッシュ演算全体の実行時間に比べ，無視できる程度であった．表 4 に CPU Core i7, GPU GeForce GTX 285 を用いた場合の JH-256 と JH-512 のビットスライス実装の実行時間比較の結果を，表 5 に CPU Atom 230, GPU ION Integrated を用いた場合の JH-256 と JH-512 のビットスライス実装の実行時間比較の結果を示す．なお，入力メッセージは 1〜1024 [KB] とし，いずれも 100 回試行の平均実行時間を求めた．

表 4: ビットスライス実装の実行時間比較 (CPU : Core i7, GPU : GeForce GTX 285).

(a) JH-256

メッセージサイズ [KB]	GPU による実行時間 [ms]	CPU による実行時間 [ms]
1	20.07	0.02
8	148.10	0.11
16	294.93	0.22
32	588.61	0.33
64	1175.95	0.59
128	2350.63	1.09
256	4700.00	2.08
512	9398.70	5.16
1024	18796.13	8.21

(b) JH-512

メッセージサイズ [KB]	GPU による実行時間 [ms]	CPU による実行時間 [ms]
1	20.07	0.02
8	148.10	0.10
16	294.94	0.21
32	588.62	0.38
64	1175.99	0.61
128	2350.70	1.08
256	4700.14	2.05
512	9399.01	4.08
1024	18796.97	8.05

CPU Core i7 と GPU GeForce GTX 285 ではおよそ 1600 倍，CPU Atom 230 と GPU ION Integrated ではおよそ 2100 倍の演算性能の差がみられた．しかし，表 2, 3 で示したリファレンス実装と比較すると，ビットスライス実装ではリファレンス実装のおよそ 25 倍の演算性能が得られていることがわかる．よって，GPU プ

表 5: ビットスライス実装の実行時間比較 (CPU : Atom 230, GPU : ION Integrated).

(a) JH-256

メッセージサイズ [KB]	GPU による実行時間 [ms]	CPU による実行時間 [ms]
1	55.02	0.03
8	405.76	0.20
16	808.03	0.41
32	1612.56	0.79
64	3221.54	1.58
128	6439.53	3.16
256	12875.31	6.34
512	25747.55	12.67
1024	51491.42	25.33

(b) JH-512

メッセージサイズ [KB]	GPU による実行時間 [ms]	CPU による実行時間 [ms]
1	54.95	0.04
8	405.76	0.20
16	808.03	0.41
32	1612.55	0.80
64	3221.54	1.59
128	6439.55	3.16
256	12875.28	6.31
512	25747.57	12.67
1024	51491.36	25.32

ログラムでもビットスライス実装による高速化は有効である.

リファレンス実装よりも演算性能差が広がっている原因としては，CPU によるビットスライス実装ではストリーミング SIMD 拡張命令 2(SSE2 命令) を用いて実装していることが挙げられる．今回作成した GPU プログラムは最低限 GPU 上で動作するビットスライス実装を行ったにすぎず，さらなる高速化が期待できる．また，GPU プログラムの優位性の一つである並列演算を適用すれば，さらに高速化することができると考えられる.

5 結論

本稿では，GPU 上で演算を行うハッシュ関数 JH のリファレンス実装とビットスライス実装を行った．GPU プログラムに特有なデータ転送に要する時間は 1 [ms] 以下で，ハッシュ演算全体の実行時間に比べると無視できる程度であった．また，GPU プログラムにおいてもビットスライス実装は有効な高速化手法であることを示した.

今後の課題として，CUDA で用意されているベクタ型変数の導入，並列演算の適用などを行うことでさらなる高速化を図ることが挙げられる.

謝辞

本研究の一部は独立行政法人情報通信研究機構の受託研究として行った.

参考文献

[1] O.Harrison and J.Waldron, “Efficient acceleration of asymmetric cryptography on graphics hardware,” AFRICACRYPT 2009, Lecture Notes in Computer Science, vol.5580, pp.350-367, 2009.

[2] D.J.Bernstein, T.R.Chen, C.M.Cheng, T.Lange and B.Y.Yang,“ECM on graphics cards,” Cryptology ePrint Archive, Report 2008/480, 2008. http://eprint.iacr.org/.

[3] D.Cook and A.Keromytis, CryptoGraphics: Exploiting Graphics Cards For Security, Springer, 2006.

[4] Nvidia Corporation, “CUDA ZONE,” http://www.nvidia.co.jp/object/cuda_home_jp.html.

[5] H.Wu, “The hash function JH,” 2009. http://icsd.i2r.a-star.edu.sg/staff/hongjun/jh/jh.pdf.