GPGPU での暗号アルゴリズムの実装と評価

村上 智祐† 笠原 竜大† 杉浦 寛†† 大釜 正裕†† 羅 鏡栄†† 齋藤 孝道†
† 明治大学 †† 明治大学大学院

1 はじめに

ネットワーク上で通信するデータの中には個人情報など秘匿性の高い情報が含まれており，それらを盗聴や改ざんから保護するために，ハッシュ処理や暗号処理を利用する必要がある．しかし，これらの処理は処理コストが高く，通信のボトルネックとなる場合がある．これらの処理を高速化する方法として，GPU(Graphics Processing Unit)[1] を利用し，CPU での処理をオフロードする方法がある．GPU は複数のプロセッサを搭載しており，それらで並列にハッシュ処理や暗号処理を実行することで，より高い性能を発揮することが期待できる．

本論文では，ハッシュ処理及び暗号処理の高速化を目的として，GPU にハッシュアルゴリズムとして SHA1, MD5 を，また共通鍵暗号化方式のブロック暗号アルゴリズムとして AES を実装し，それらの並列化を試みた．それら実装についてパフォーマンス計測を行い，その結果について考察する．

2 GPU

2.1 概要

GPU は，画像処理を高速に行うことを目的として開発されたマイクロプロセッサである．GPU 上に搭載された複数のプロセッサが同時に並列して演算を行う．従来 CPU 上で処理されていた 3 次元から 2 次元への座標変換，レンダリング，テクスチャの張り込みなどの処理を GPU 上で処理することで，処理時間の短縮や，CPU の負荷軽減などが可能となる．

2.2 アーキテクチャ

ここでは，本論文で利用した NVIDIA 社の GPU である GeForce GTX280(以降，GTX280) のアーキテクチャについて説明する．図 1 に GTX280 のアーキテクチャの概要を示す．

GTX280 は 30 台の Streaming Multiprocessor(以降，SM) と Thread Execution Manager，GlobalMemory などから構成されている．各 SM は，Streaming Processor(以降，SP) を 8 個と，共有メモリ，Instruction Unit(以降，IU) などから構成される．Thread Execution Manager は，CPU から受け取った命令を実行する際に，各 SM に同じ命令を一括して送信し，全ての SM で同時に同じ命令を実行させる．GlobalMemory は SM 外の VRAM 上に搭載されており，全ての SM が共有しているメモリである．

2.3 GPGPU と CUDA

2.3.1 GPGPU

GPGPU(General Purpose computing on GPUs) とは，GPU を画像処理以外の汎用的な数値演算に用いる技術の総称である．GPU 向けのプログラム開発では，C 言語や C++を利用することができる．

2.3.2 CUDA

CUDA(Compute Unified Device Architecture) とは，NVIDIA 社が提供する GPU 向けの C 言語による総合開発環境であり，GPU のハードウェアを直接操作する CUDA ドライバと，GPU の使用をサポートするランタイムライブラリ，GPU で数値計算を行うためのライブラリから構成されている．これらを用いることで，GPU 側のメモリ領域の確保や，スレッド並列実行を行うことができる．また CUDA を利用するためのコンパイラとして，nvcc が用意されている．

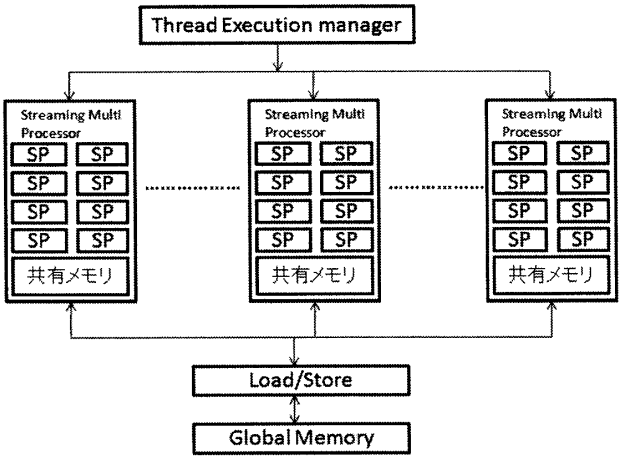

図 1: GeForce GTX280 のアーキテクチャ

2.3.3 CUDA プログラミング

CUDA では GPU で動作するプログラムのことをカーネルと呼び，CPU からカーネルを関数のように呼び出して実行する．

カーネルは関数の前にカーネル用関数を示す__global__という関数識別子の宣言で定義される．また，ブロック，スレッドの数は <<<…>>> にて指定する．

```
//カーネルの定義
__global__ void vecAdd(float* A, float* B, float* C)
{ //GPU 上での処理 }
int main()
{
//カーネルの呼び出し
vecAdd<<< ブロック, スレッド >>>(A, B, C);
}
```

また，スレッドとは SM 内の SP で実行される単位であり，1 つのスレッドで 1 つのカーネルを実行する．ブロックとは GPU 内の各 SM に割り当てられる処理単位であり，複数のスレッドをまとめたものである．1 つのブロックで生成できるスレッド数は最大 512 個であるが，一度に並列実行できる最大スレッド数は 32 個である．ただし 1 つのスレッドでの処理量が大きいと，並列に実行できるスレッド数は減少する．

3 実装内容

本論文では，SHA1，MD5，AES の C 言語コードを GPU で実行できるように改変した．以降，これらを実装コードと呼ぶ．実装において SHA1，MD5 の場合，CPU が複数の平文データを用意し，AES の場合には平文データと鍵を用意する．ブロック数，スレッド数はプログラムに応じて，それらの値を指定する．ただし，ブロック数，スレッド数はそれぞれ 65535 個，512 個以内にする必要がある．また，GPU 上のメモリは Global Memory を指すこととする．図 2 は，実装コードの処理の流れを示したものである．以下に示

† Tomosuke MURAKAMI, Ryuuta KASAHARA, Takamichi SAITO
†† Kan SUGIURA, Masahiro OHGAMA, Keang-Weng LO
{tomo47,kasahara,kan_s,ohgama,oscar.weng,saito}@cs.meiji.ac.jp
Meiji University(†), Graduate School of Meiji University(††)
1-1-1 Higashimita, Tama-ku, Kawasaki-shi, Kanagawa, 214-8571, Japan(†)(††)

す説明文に割り当てた番号は，図 2 の各処理に割り当てたそれと対応している．

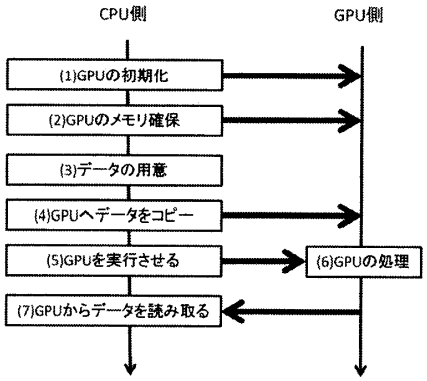

図 2: 処理の流れ

(1) CPU 側が，GPU の初期化を行う．
(2) CPU 側が，GPU 上のメモリ領域の確保を行う．
(3) CPU 側が，GPU に処理させるデータを用意する．
(4) CPU 側が，用意したデータを GPU へコピーする．
(5) CPU 側が，スレッド数とブロック数を指定して GPU を実行させる．
(6) GPU が処理を行う．CPU 側は GPU 処理の終了を待つ．
(7) CPU 側が GPU 上のメモリから計算結果を読み取る．

ここで，SHA1 及び MD5 の実装では CPU は複数の平文を 1 つの配列にまとめて GPU 上のメモリにコピーし，GPU の各スレッドがスレッドに割り当てられた平文のハッシュ処理を行う．また，AES の実装では，GPU の各スレッドが，AES の各ブロックのラウンド処理を行うように実装した．

4 評価

4.1 計測環境

実装コードの評価を行うにあたって，Core2Quad と GTX280 を搭載したマシンを用いた．その上で，実装コードと OpenSSL[2][3] speed コマンドを用いて計測を行った．ここで実装コードは，GTX280 を用いての計測となるが，OpenSSL は GTX280 には対応していないため Core2Quad のみを用いた計測となる．評価に用いたマシンの Core2Quad は 3GHz で動作し，2GByte のメモリを搭載している．また OS として FedoraCore8（Kernel 2.6.23）が稼働しており，実装に用いた CUDA のバージョンは 2.1，OpenSSL のバージョンは 0.9.8b である．

4.2 計測項目

実装コードの評価のために，16Byte，64Byte，256Byte，1024Byte，8192Byte の平文を用意し，ハッシュ処理では複数の平文を，暗号処理では 1 つの平文を GPU で処理した場合のスループットを計測した．また，計測には MD5，SHA1，AES の ECB モードを用いた．計測は実装コード，OpenSSL speed コマンドをそれぞれ 10 回行い，その平均値を結果とした．

(1) 実装コードのスループットの計測

実装コードによるハッシュ処理，暗号処理のスループットの計測を行った．ハッシュ処理の計測結果を表 1，暗号処理の計測結果を表 2 に示す．表 1 のデータサイズは 1 つのスレッドで処理する平文のデータサイズであり，全てのスレッドで処理されるデータサイズの合計はブロック数×スレッド数×データサイズから求められる．例えば，表 1 のデータサイズが 16Byte の場合では，合計データサイズは $8192 \times 256 \times 16$ である．表 2 のデータサイズは，スレッド全てで処理される平文の合計データサイズである．

(2) OpenSSL speed コマンドのスループットの計測

OpenSSL speed コマンドによるハッシュ処理，暗号処理の処理時間の計測を行った．Core2Quad 上で実行した場合の計測結果を表 3 に示す．

表 1: ハッシュ処理のスループット

データサイズ (Byte)	16	64	256	1024	8192
ブロック数	8192	4096	8192	16	16
スレッド数	256	256	32	128	128
SHA1(Gbps)	5.60	8.82	10.12	15.46	11.52
MD5(Gbps)	6.89	9.05	10.50	16.60	11.49

表 2: 暗号処理のスループット

データサイ (Byte)	16	64	256	1024	8192
ブロック数	1	1	9	22	30
スレッド数	2	5	2	3	20
AES-ECB(Gbps)	0.02	0.008	0.034	0.139	0.869

表 3: OpenSSL speed コマンドのスループット

データサイズ (Byte)	16	64	256	1024	8192
SHA1(Gbps)	0.82	2.58	6.28	9.88	11.83
MD5(Gbps)	0.82	2.75	7.32	12.45	15.81
AES-ECB(Gbps)	4.69	5.12	5.29	5.30	5.35

5 考察

ここでは，実装コードと OpenSSL speed コマンドにおけるハッシュ処理，暗号処理のスループットを比較することで，そのパフォーマンスについて考察する．

(1) ハッシュ処理

実装コードでは，SHA1，MD5 共にブロック数が 16，スレッド数が 128，データサイズが 1024Byte の場合が最もスループットが高くなった．OpenSSL speed コマンドではデータサイズが 8192Byte の場合が最もスループットが高くなった．データサイズが 8192Byte の場合では，OpenSSL speed コマンドの方がスループットが高くなっている．しかし，データサイズが 16～1024Byte までは OpenSSL speed コマンドよりも実装コードの方がスループットが高い．データサイズが 8192Byte の場合では，1 つのスレッドでの処理量が大きくなり，一度に並列に実行できるスレッドの数が減るため，スループットが低くなると考えられる．

(2) 暗号処理

AES-ECB モードはブロック数が 30，スレッド数が 20，データサイズが 8192Byte の場合が最もスループットが高速となった．OpenSSL speed コマンドではデータサイズが 8192Byte の場合が最もスループットが高速となった．全てのデータサイズにおいて，実装コードよりも OpenSSL speed コマンドのスループットの方が高かった．実装コードでは，GPU の Global Memory へアクセスする際に，各スレッドのアクセスする番地が連続で並んでいる場合に限り，最大 16 スレッド分のメモリアクセスを 1 回にまとめることができる．しかし，AES のラウンド処理では各スレッドのアクセスする番地は連続ではなく，複数のスレッドのメモリアクセスを 1 回にまとめることができないためスループットが低くなると考えられる．

6 まとめ

本論文では，GPU 上でのハッシュ処理，暗号処理の並列化を試み，そのパフォーマンス計測と評価を行った．今後の課題としては，GPU に適したコードへの改良と，実システムでの利用が挙げられる．

参考文献

[1] NVIDIA Compute Unified Device Architecture Programming Guide Version2.0
[2] John Viega,Matt Messier,Pravir Chandra 共著 齋藤孝道 監訳 "OpenSSL 暗号・PKI・SSL/TLS ライブラリの詳細" オーム社
[3] http://www.openssl.org/