GPUを用いた暗号処理の性能予測フレームワークに関する研究

防衛大学校理工学研究科後期課程

電子情報工学系専攻・情報知能メディア学教育研究分野

西川 尚紀

平成 ２６年 ３月

論 文 の 内 容 の 要 旨

申請者　西川 尚紀

論 文 題 目

GPU を用いた暗号処理の性能予測フレームワークに関する研究

情報システムが社会へ浸透し，シンクライアントやクラウドサービスのように，ネットワークを介してサーバへ接続するリモートコンピューティング環境が注目されるようになってきた．この環境では，サーバにはあらゆる端末のデータを一括保管するため，ストレージの容量は膨大となる．暗号化によるデータ防護は，情報漏洩に対するセキュリティ対策の基本手段であるため，膨大なデータを高速に暗号化する必要性が生じてきている．一方，暗号アルゴリズムの処理の効率性は，コンピュータ内のプロセッサの影響を受ける．プロセッサのアーキテクチャは，2000 年頃まではシングルコアで設計されてきた．しかし，消費電力に起因する発熱の問題から，そのアーキテクチャ設計は 2002 年以降，主にマルチコアやメニ―コアに移行しており，今後はこのような並列プロセッサにおける暗号の設計及び実装評価が重要になると考えられる．特に現在では，マルチコア CPU だけでなく，専用ハードウェア並みの演算能力を持った Graphics Processing Unit（GPU）や Intel Xeon Phi が，暗号処理を効率化できる並列プロセッサとして注目されてきている．

しかし，並列プロセッサに対する暗号の設計及び実装評価は，上記のような様々な種類のプロセッサを対象としなければならず，その負担は大きい．その上，現在使用されている暗号は多種多様である．そのため，本研究では，並列プロセッサ向けの暗号設計用ツールがあれば，暗号設計の負担を軽減できる．そこで本研究では，暗号アルゴリズムを選択して実行可能な並列プロセッサとして注目されている GPU 向けに，大容量データの処理に適するブロック暗号を効率的に設計するためのフレームワーク (Performance Prediction Framework: PPF) を開発することを目的とした．PPF には，ブロック暗号の逐次処理コードの GPU 実装性能を，設計段階で予測できるという特徴を持たせる．また本研究では，並列プロセッサに共通したプラットフォームである OpenCL を採用した．このことにより PPF は，GPU ベンダに依存したフレームワークにならず，将来的にはマルチコア CPU や Intel Xeon Phi への拡張も可能となる．

第 2 章では，OpenCL プラットフォームの仕様とそれに準拠した並列プロセッサ及び GPU について述べる．この章で述べた GPU アーキテクチャの特徴は，第 6 章で述べる

PPF の設計に反映される．

第 3 章では，PPF の設計及び実装評価で用いるブロック暗号のアルゴリズムについて述べる．PPF は，公開鍵暗号等での利用も視野に入れて設計しているが，本研究での PPF の評価はブロック暗号に対してのみ行う．

本研究で提案する PPF では，これから並列プロセッサ向けに暗号アルゴリズムを設計しようとする設計者が容易に利用できるように，設計の簡単化を目指す．そこで，第 4 章では，AES を題材として，複数のブロック暗号に共通した GPU への実装方針を導出する．この実装方針は，第 6 章で述べる PPF の設計の簡単化に反映される．

第 5 章では，第 4 章で導出した実装方針を用いて，ブロック暗号の代表的な構造である SPN 構造，Feistel 構造，SPN と Feistel のハイブリッド構造の暗号アルゴリズムを実装し，異なる世代のアーキテクチャで開発された NVIDIA Geforce GTX 580 と Geforce GTX 280 という 2 種の GPU での処理速度を測定する．具体的な暗号アルゴリズムには，SPN 構造に AES，Feistel 構造に Camellia，SPN と Feistel のハイブリッド構造に SC2000 をそれぞれ選択した．この値は，第 6 章で必要となる GPU を用いた暗号処理性能の実測値となる．なお，Camellia と SC2000 はそれぞれ AES と異なる構造をとるブロック暗号であるため，この測定結果から，併せて第 4 章で導出した実装方針の有効性も確認できる．

第 6 章では，PPF の開発と評価について述べる．PPF の評価では，第 5 章で測定した完成版のブロック暗号に加え，ブロック暗号の設計段階の実装性能を予測し，暗号設計者の作業量を軽減できることを示す．第 5 章と同じ種類のブロック暗号と GPU を対象として PPF の評価を行ったところ，これらの暗号化処理時間を測定した結果は，PPF から見積もられる結果と概ね一致した．また，仕様書を基にシミュレートした設計段階の Camellia に対しても，GPU 実装による暗号化処理時間を予測できることを確認し，効果的な開発を補助できることを確認できた．

本研究で実施した PPF の評価は，Fermi 及び GT200 アーキテクチャの NVIDIA GPU のみで評価した．今後は，Kepler アーキテクチャの NVIDIA GPU に加え，AMD 社の GPU や Intel Iris Graphics 等のモバイル型 GPU にも評価を行う必要がある．また，マルチコア CPU や Intel Xeon Phi などの並列プロセッサに対しても評価を行い，PPF の

適用可能範囲を拡張していく必要がある．本研究では，暗号アルゴリズムとして，共通鍵暗号を対象としたが，公開鍵暗号等も実利用されている．今後は，このような暗号にも PPF を対応させていく必要がある．

目　　次

頁

第 1 章

序　論

1.1　高速な暗号処理の要求

1.1.1　リモートコンピューティング環境の普及

コンピュータによる情報システムが広く市民生活や企業活動に浸透している情報化社会が，1990 年代に到来した．デジタルデータには，(1) 大きな容量を小型の可搬記憶媒体に保存可能，(2) 共有や交換が容易，といった利便性がある．そのため，デジタルデータの一般化に伴い，様々な端末やそのデータを保存するための記憶媒体が普及し，インフラとしてインターネットが浸透することとなった．

一方，情報システムの規模が拡大するにつれ，大量の端末を導入する際の手間や管理コストの問題が表面化してきた．また，個人情報の保護や情報漏洩対策も大きな問題となってきた．こうした理由から，シンクライアント [1] やクラウドサービス [2] のように，ネットワークを介してサーバへ接続するリモートコンピューティング環境が注目されており，その市場規模は年々増加している．例えば，我が国の総務省が 2010 年に発行したスマート・クラウド研究会報告書 [3] によれば，クラウドサービス市場は**図 1.1** に示すように，2015 年度には約 2 兆 3,700 億円の規模に達するものと見込まれている．リモートコンピューティング環境では，使用したデータ量や時間に応じた料金を支払えばよく，端末側でサーバやソフトウェアを所有する必要がない．そのため，情報システムを利用するコストを低く抑えることができる．また，データは，基本的にサーバ側に保存されるため，端末が盗難や紛失に遭っても，情報流出を食い止めることができる．

リモートコンピューティング環境には，サーバからデータをダウンロードして

端末側で処理する方法や，全ての処理をサーバ側で実行してしまう方法がある．いずれの方法でも，多数の端末のデータをサーバに一括保管するので，サーバで管理するデータの容量は著しく増大することとなる．なお，このような社会のニーズは，半導体の技術向上によるストレージ容量の増加に支えられている．半導体集積回路技術に関する国際会議である ISSCC 2013 で報告された，2001～2013 年の不揮発性メモリにおけるストレージ容量の推移は**図 1.2** に示す通りである [4]．いずれの種類のメモリでも，その容量は年々増加しており，この傾向は今後も継続すると見積もられている．

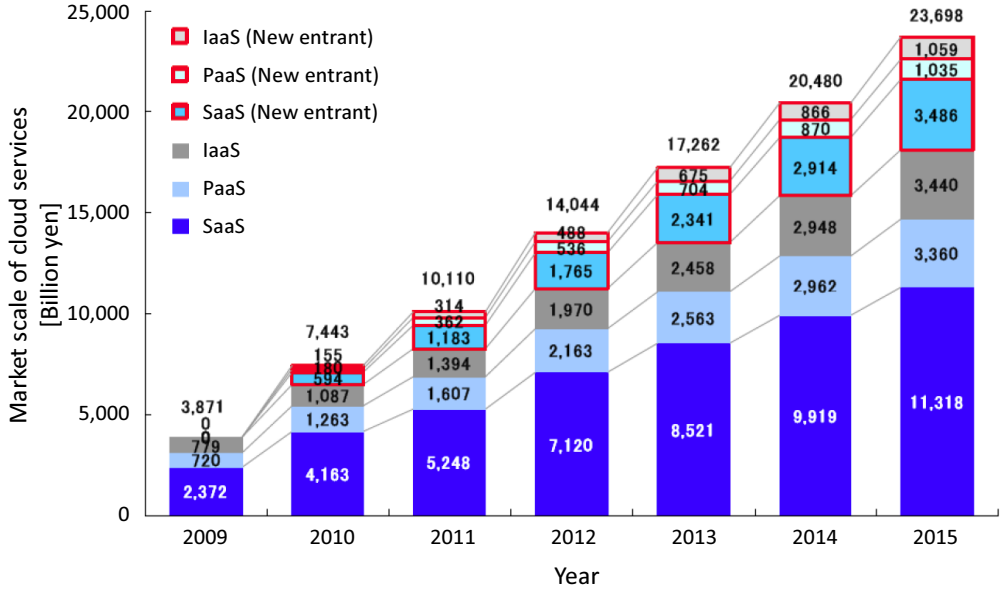


図 1.1　2015 年までに予想されているクラウドサービスの市場規模 [注 1)]（出典：総務省，“スマート・クラウド報告書”[3]）

[注 1)] 凡例中の IaaS，PaaS，SaaS は，それぞれ Infrastructure as a Service，Platform as a Service，Software as a Service の略であり，クラウドサービスの利用形態の種類を表している．また，“New entrant” は，従来内製されていたシステムの一部が外注化されて新規にクラウド化されると想定した台数を表している．

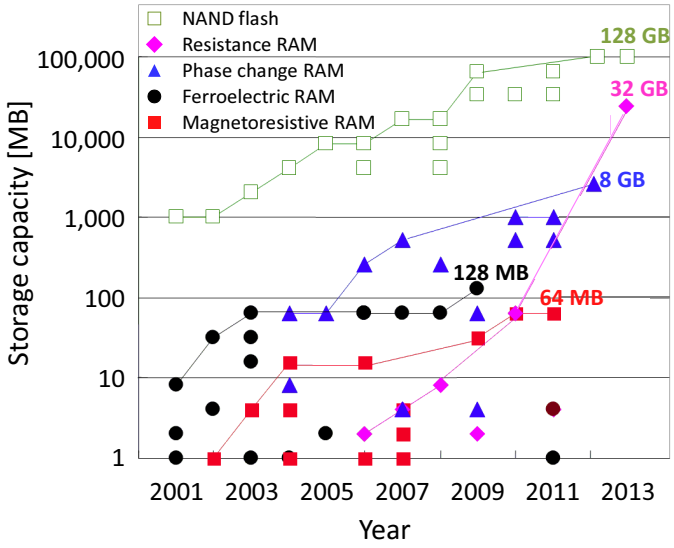


図 1.2　不揮発性メモリにおけるストレージ容量の推移（出典：ISSCC 2013, “ISSCC 2013 Press Kit”[4]）

1.1.2　高速な暗号化処理の必要性

日本ネットワークセキュリティ協会による調査報告書 [5],[6] を基に作成した，個人情報漏洩事案における 1 人当たりの平均想定損害賠償額の推移を**図 1.3** に示す．また，情報漏洩 1 件あたりの想定損害賠償額も，2011 年の時点で 1 億 2,810 万円にまで達しており [6]，情報漏洩が発生した場合の影響が致命的となってきていることが分かる．特に，リモートコンピューティング環境では，サーバで管理するデータの容量が膨大となることから，強固なセキュリティ対策が必要不可欠である．

暗号化によるデータ防護は，情報漏洩に対するセキュリティ対策の基本手段である．リモートコンピューティング環境では，多数のユーザがサーバを共有するので，データは暗号化して保管することとなる．この環境では，あるデータを共有するグループに新しいユーザが追加されたときは，その新規ユーザも復号できるように鍵を付け替えてデータを暗号化し直す必要がある．また，この鍵の付け替えがセキュリティホールとなる可能性を考慮し，あるユーザの鍵による暗号文

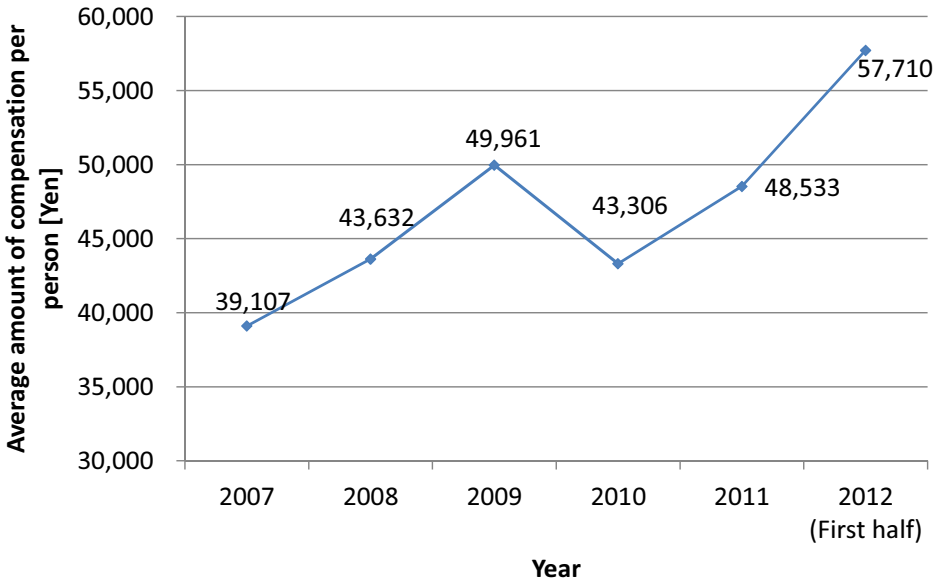


図 1.3　個人情報漏洩事案における 1 人当たりの平均想定損害賠償額の推移（日本ネットワークセキュリティ協会による調査報告書 [5],[6] を基に作成）

を他のユーザの鍵で復号できるように再暗号化するプロキシ再暗号化技術も注目されている [7]．こうした理由から，リモートコンピューティング環境では，大量のデータを高速に暗号化する必要性が発生している．

1.1.3　暗号アルゴリズムに対する処理速度の要求

これまで暗号アルゴリズムでは，安全性がその設計の主たる要件とされてきた．しかし現在では，**第 1.1.1 節**で述べたように，暗号化が必要なデータ容量が増大しているため，暗号設計は安全性に加え，処理速度を含めた実装性も重視されてきている [8]．現代暗号のアルゴリズムは，共通鍵暗号方式と公開鍵暗号方式に大別される．データの暗号処理では，公開鍵暗号方式は共通鍵の交換に使用し，データの暗号化処理は共通鍵暗号方式で行うことが多い．共通鍵暗号方式は，ストリーム暗号とブロック暗号に大別される．ストリーム暗号は，平文を bit または Byte 単位で暗号化するのに対し，ブロック暗号は平文を固定長のブロックに分割して暗号化する．そのため，ブロック暗号は大容量データの処理に向いており，

ストレージの暗号化に広く利用されている．このような特徴から，ブロック暗号は今後も高いニーズがある．例えば現在，小規模なデバイスでも実行可能な軽量暗号の研究が盛んになってきているが，同暗号の国際標準規格である ISO/IEC 29192[9] には，CLEFIA[10] 及び PRESENT[11] というブロック暗号が新たに採択されている．しかし，ブロック暗号のアルゴリズムは，一般にコストの高い計算処理であり，同暗号に対する高速化の要求は高くなっている．

ブロック暗号である AES[12] を高速に処理する方法として，Intel 社と AMD 社は，AES をハードウェアに実装し，x86 命令の拡張セットとして CPU に直接組み込んだ AES-NI[13] を開発している．ただし，AES-NI には，高速処理できる暗号アルゴリズムが AES のみに限定される，決定的な短所がある．情報処理推進機構（Information-technology Promotion Agency, Japan: IPA）[14] が報告しているように，現在使用されている暗号アルゴリズムは多種多様である [15] ため，AES-NI では機能的に不足している状況にある．

1.2　並列プロセッサでの暗号処理

あらゆる暗号アルゴリズムを選択して実行可能とするには，暗号アルゴリズムをソフトウェアとして実行する方法が有効である．このため，暗号アルゴリズムの処理の効率性は，コンピュータ内のプロセッサの影響を受ける．従って本節では，プロセッサのアーキテクチャの変遷を整理する．

プロセッサのアーキテクチャは，2000 年頃まではシングルコアで設計されてきた．シングルコアのアーキテクチャの性能向上は，動作周波数の増大による命令実行速度の向上や命令レベル並列性を高める等の工夫に依存してきた．しかし，消費電力に起因する発熱の問題から，そのアーキテクチャ設計は 2004 年以降，シングルコアからマルチコアへ変更を迫られるようになった [16]．その結果，現在では，プロセッサのアーキテクチャは，主にマルチコアに移行している．

マルチコアプロセッサには，コア数を増やした分だけ並列度が上がり，処理速

度を向上できるという利点がある．そのため，コードを並列化すれば，概ねコア数の増加に沿った性能向上が期待できる．ただし，シングルコアプロセッサで動作するソフトウェアとの互換性を維持するため，各コアにはシングルコアプロセッサと同じ命令セットを用意しなければならなかった．このため，コアの回路面積が比較的大きくなり，プロセッサに封入可能なコア数の上限を圧迫して，シングルコアプロセッサ時代のような動作周波数の増大等に依存した性能向上が望めなくなってきた．現在では，マルチコアプロセッサに封入されるコア数は，2〜6 個程度が一般的となっている．

しかし，マルチコアプロセッサの性能向上が鈍化しても，処理速度への要求が留まることはない．マルチコアプロセッサとは異なるアプローチとして，制御プロセッサとして用いる CPU に並列処理に特化したアクセラレータを加え，全体の計算負荷を分散することで性能の向上を実現するヘテロジニアスコンピューティングが試みられてきた．現在利用できるアクセラレータとしては，Graphics Processing Unit（以下，GPU），Intel Xeon Phi[17]，Cell Broadband Engine[18]，Digital Signal Processor 等がある．このうち，特に GPU は，他のアクセラレータと比較して安価に入手可能であり，CPU と比べて性能の伸びが著しいため，注目を浴びている．Intel Xeon Phi は，Intel 社が 2012 年に開発した，x86 命令セット互換のプロセッサを搭載したアクセラレータであり，今後，データセンタやワークステーション等への利用が期待されている．GPU と Intel Xeon Phi には，数百〜数千個，60 個程度ものコアがそれぞれ内蔵されている．従って，マルチコアプロセッサと区別する目的から，これらはメニーコアプロセッサと呼ばれている．本論文では，以降，マルチコアプロセッサとメニーコアプロセッサをまとめて，並列プロセッサと呼称する．

1.3 並列プロセッサ向けの暗号設計用ツールの必要性

一方，我が国の暗号評価プロジェクトであるCRYPTREC[19]は，2000〜2003年の間に暗号アルゴリズムの安全性や実装性を評価し[20]，2003年に電子政府推奨暗号リスト[21]を策定した．ただし，ソフトウェア実装評価には，Intel Pentium III[22]，UltraSparc IIi[23]，Alpha 21664[24]，Z80[25]という4種のシングルコアプロセッサが用いられていた．しかし現在では，プロセッサのアーキテクチャは，**第1.2節**に示した通りマルチコアに移行しているため，今後は並列プロセッサにおける暗号の設計及び実装評価が重要になってくると考えられる．実際，CPU等のマルチコアプロセッサを用いて並列化することにより，暗号化処理を効率化できることが先行研究[26–30]により報告されている．

また現在では，増大するデータ容量に対応するため，専用ハードウェア並みの演算能力を持ったメニーコアアクセラレータであるGPUやIntel Xeon Phiが注目されてきている．従って，暗号の設計及び実装評価は，上記のような様々な種類の並列プロセッサを対象としなければならない段階にある．その上，IPAが報告した通り，現在使用されている暗号アルゴリズムは多種多様となっている．設計した暗号の処理速度は設計者自身の環境で実測して検証する必要がある[8]ため，並列プロセッサ向けに暗号を設計する負担は大きい．そのため，並列プロセッサ向けの暗号設計ツールがあれば，暗号の設計及び実装評価の作業量を軽減できる．

1.4 研究目的と研究内容

並列プロセッサ向けの暗号設計用ツールとして，本研究では，暗号アルゴリズムを選択して実行可能な並列プロセッサとして注目されているGPU向けに，効率的にブロック暗号を設計するためのフレームワーク（Performance Prediction Framework: PPF）を開発することを目的とした．PPFの開発方針の重要なポイントは，ブロック暗号の逐次処理コードのGPU実装性能を，設計段階で予測で

きることにある．また，PPF が広く利用されるためには，広範囲に渡って多様な GPU を扱えるように設計しなければならない．このことにより PPF は，GPU ベンダに依存したフレームワークにならず，将来的にはマルチコア CPU や Intel Xeon Phi への拡張も可能とする必要がある．更に，PPF の普及のためには，暗号設計者にとって単純で，理解が容易であることが望ましい．

1.5 本論文の構成

以下に，本論文の構成を示す．

並列プロセッサへ広範囲に適用できる PPF を開発するため，その設計には，並列プロセッサに共通したプラットフォームである OpenCL[31] を用いることとした．そのため，まず，**第 2 章**では，OpenCL プラットフォームの仕様とそれに準拠した並列プロセッサ及び GPU について述べる．高精度かつ利用が容易な性能予測フレームワークを構築するため，この章では OpenCL の仕様に加え，GPU アーキテクチャに固有の特徴を整理する．

第 3 章では，PPF の設計及び実装評価で用いるブロック暗号のアルゴリズムについて述べる．PPF は，公開鍵暗号での利用も視野に入れて設計しているが，本研究での PPF の評価はブロック暗号に対してのみ行った．これは，**第 1.1.2 節**で述べたように，大容量データの暗号化処理にはブロック暗号が多く利用されており，同暗号の高速化の要求が高いためである．

これまでに発表されている GPU アプリケーションの性能予測モデル [32–37] は複雑で，利用が容易とは言い難い．一方，本研究で提案する PPF では，これから並列プロセッサ向けに暗号アルゴリズムを設計しようする設計者が容易に利用できるように，設計の簡単化を目指す．そこで，**第 4 章**では，AES[12] を題材にして，複数のブロック暗号に共通した GPU への実装方針を導出する．この実装方針は，**第 6 章**で述べる PPF の設計の簡単化に反映される．

第 5 章では，**第 4 章**で導出した実装方針で 3 種類のブロック暗号（AES，Camellia[38],

SC2000[39]）を実装し，NVIDIA 社の Geforce GTX 580 と Geforce GTX 280 という 2 種の GPU での処理速度を測定し，**第 6 章**で必要となる GPU を用いた暗号処理性能の実測値を取得する．なお，Camellia と SC2000 は AES とは異なる 2 種のブロック暗号であるため，この測定結果から，併せて**第 4 章**で導出した実装方針の有効性も確認できる．

第 6 章では，PPF の設計と評価について述べる．PPF の評価では，**第 5 章**で測定したブロック暗号の処理時間に対して予測を行う．なお，**第 5 章**と**第 6 章**ではそれぞれ暗号化処理の処理速度，処理時間と，異なる測定単位が使用されているが，これは両章で得られた結果に対する考察を容易にするためである．また，PPF が，GPU へのブロック暗号の実装性能をそのアルゴリズムの設計段階で予測できることを示し，効率的に設計を実施できることを実証する．

本研究の概要を図示すると，**図 1.4** のようになる．

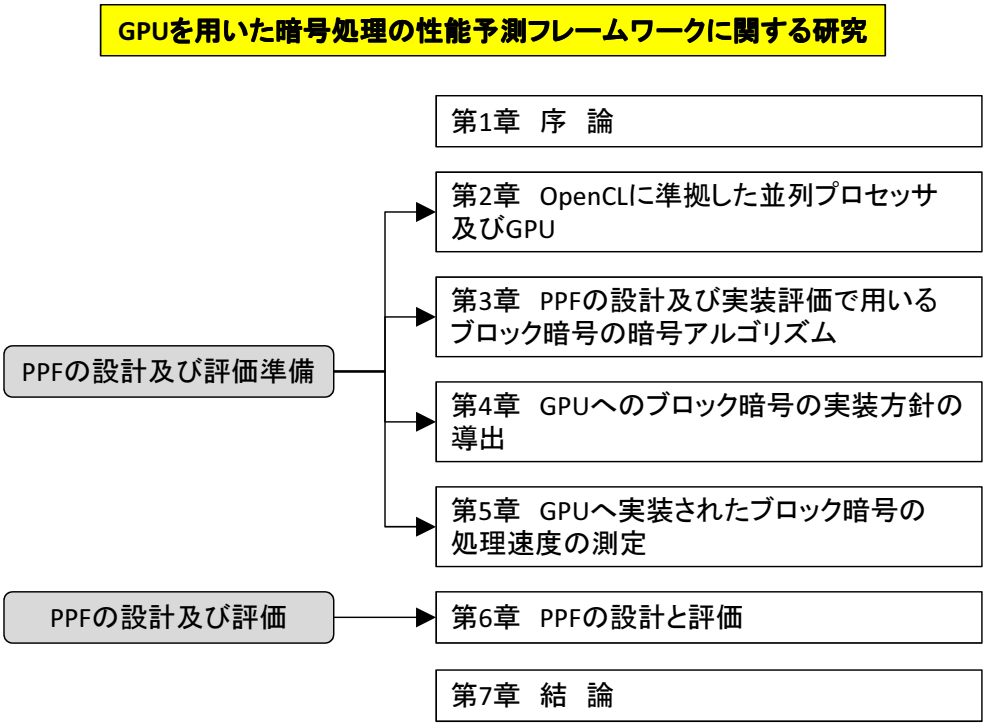


図 1.4　本研究の概要

第2章

OpenCLに準拠した並列プロセッサ及びGPU

2.1 まえがき

本研究で提案するPPFが広く利用されるためには，多様な並列プロセッサを扱えるようにしなければならない．そのためPPFは，複数の並列プロセッサに共通したプラットフォームであるOpenCL[31]を用いて開発した．本章では，OpenCLの仕様，及びそれに準拠した並列プロセッサ及びGPUについて述べる．

2.2 OpenCL プラットフォーム

2.2.1 仕様

OpenCLプラットフォームは，**図 2.1** に示すように，1つのホスト及びそれに接続された1つ以上のOpenCLデバイスから構成される．ホストは，アプリケーションを実行するコンピュータ本体を表す．OpenCLでは，ホストで実行するホストプログラムとOpenCLデバイスで実行する関数であるカーネルをそれぞれ別個に記述する必要がある．ホストプログラムとカーネルの記述言語は，それぞれC/C++，OpenCL C言語である．OpenCL C言語は，従来のC言語の拡張である．ホストプログラムは，アプリケーション全体の流れを制御する役割を担当し，OpenCL APIを用いてOpenCLデバイスにカーネルを発行する．なお，ホスト，OpenCLデバイスという分け方は概念的なものであり，実際のハードウェアでは，ホストに属しているマルチコアCPUとOpenCLデバイスに相当するマ

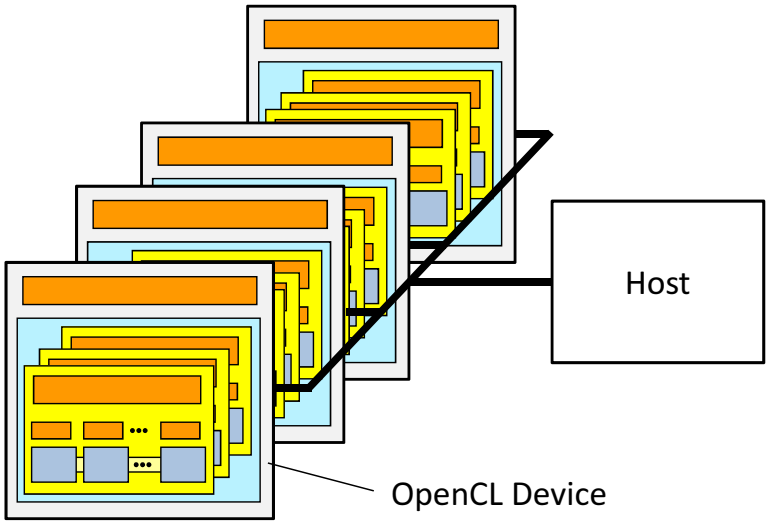


図 2.1　OpenCL プラットフォーム

ルチコア CPU が同じ場合もある．つまり，1 つ以上の OpenCL デバイスが存在していれば OpenCL プラットフォームを構成することができるので，マルチコア CPU が 1 つのみの構成でも OpenCL アプリケーションの実行は可能となっている．

2.2.2　OpenCL に準拠した並列プロセッサのアーキテクチャとプログラミングモデル

多くのマルチコア CPU，Intel Xeon Phi，Cell Broadband Engine，Digital Signal Processor 等の並列プロセッサは OpenCL に準拠しており，OpenCL デバイスとして扱われる．このような並列プロセッサのアーキテクチャは，**図 2.2** に示すように，コンピュートデバイスとグローバルメモリから成る．コンピュートデバイスは複数個のコンピュートユニット (以下，CU) から構成され，各 CU は複数個のプロセッシングエレメント (以下，PE) をから構成されている．即ち，コンピュートデバイスは，プロセッサコアの階層構造から成る．また，各 CU には，PE の他にローカルメモリとプライベートメモリが含まれる．グローバルメモリは，全ての CU に接続されているメモリ領域である．また，グローバルメモリに

は，定数を保持するコンスタントメモリが置かれる．

OpenCL では，このような多数のコアを用いて，スレッドレベルでの並列処理を行う．OpenCL では，並列処理を実行する単位であるスレッドを，ワークアイテムと呼称する．ワークアイテムは，複数個をまとめたワークグループとして管理される．また，ワークグループは，CU へ均等にスケジューリングされ，ワークグループ内のワークアイテムが PE で実行される．このようにワークアイテムは，上述したプロセッサコアの階層構造から成るアーキテクチャに対応して，階層的に管理されている．ホストプログラムを記述する際は，カーネルを実行する際のパラメータであるワークグループ数とワークグループ当たりのワークアイテム数を指定する．CU において，あるワークグループの処理が終了すると，別のワークグループがその CU にスケジューリングされる．従ってデバイスの処理効率を高めるには，ワークグループ数に CU 数の倍数を指定することが望ましい．また，同様の理由から，ワークグループ当たりのワークアイテム数には，CU 当たりの PE 数の倍数を指定することが望ましい．なお，カーネルで起動可能なワークグループ当たりのワークアイテム数の最大値は，並列プロセッサ毎に異なる．

ワークアイテムは，上述した 4 種類のメモリ領域（グローバルメモリ，コンスタントメモリ，ローカルメモリ，プライベートメモリ）にアクセスしながらカーネル内の処理を実行していく．これらのメモリ領域には，以下のような特徴がある．

- グローバルメモリ： 全てのワークグループに属するワークアイテムから読み書きが可能
- コンスタントメモリ： 全てのワークグループに属するワークアイテムから読み出しのみが可能
- ローカルメモリ： CU にスケジューリングされたワークグループ内のワークアイテムから読み書きが可能であり，かつワークグループ内のワークアイテム間で共有して利用

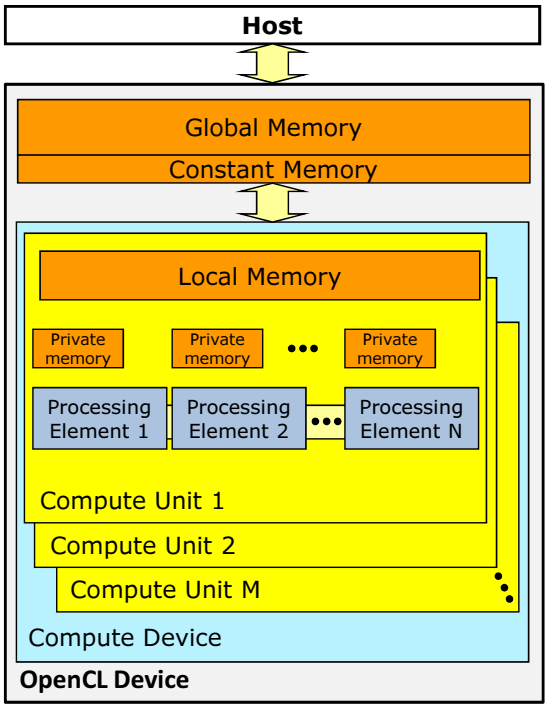


図 2.2　OpenCL に準拠した並列プロセッサのアーキテクチャ

- プライベートメモリ：ワークアイテムにより読み書きが可能であり，かつワークアイテムが占有．このため，このメモリ領域に割り当てられた変数は，他のワークアイテムから参照が不可．ワークアイテム相互でのデータには，ローカルメモリを仲介

なお，カーネルの実行に必要なデータは，ホスト側のメモリから直接 OpenCL デバイスのローカルメモリまたはレジスタに転送することはできない．カーネル実行に必要なデータは，グローバルメモリまたはコンスタントメモリに転送する必要がある．

2.3　GPU

現在，多くのベンダにより提供されている GPU のアーキテクチャは，OpenCL に準拠しており，数百から数千個ものプロセッサコアの階層構造で構成されてい

る．このような多数のプロセッサコアを高効率で稼働させるためには，高いデータ供給スループットが必要になる．そのため，**第 2.2.2 節**で述べた GPU 内の 4 種のメモリは，容量とアクセスレイテンシに概ねトレードオフの関係を伴う形で用意されている．従って，アプリケーションの特性に応じて，メモリ領域の利用を最適化することが重要となる．GPU を OpenCL デバイスとして利用した場合，グローバルメモリは GPU のビデオメモリに置かれる．ローカルメモリは，容量は小さいがレイテンシの小さい，CU 内のローカルなメモリに割り当てられる．コンスタントメモリは，グローバルメモリ内のキャッシュされる領域に割り当てられる．プライベートメモリは，CU 内のレジスタに割り当てられる．

更に，NVIDIA 社や AMD 社等の主要なベンダにより提供されている GPU は，多数のワークアイテムによる並列処理で，その演算能力を引き出せるように設計されている．その具体的な特徴は，以下の 4 点にまとめられる．

(i) 複数個のワークアイテムをまとめたワークアイテムバッチ[注 1)]（以下，バッチ）単位での命令発行

(ii) パイプライン化された PE

(iii) 複数個のワークアイテムからメモリアクセス要求を並列に受け付けられるように設計されたメモリシステム

(iv) PE パイプラインのストール時に別のバッチへ命令実行を切り替えて，あるバッチの算術論理演算のレイテンシを別のバッチのメモリアクセス命令に隠蔽するアウトオブオーダースケジューリング機能

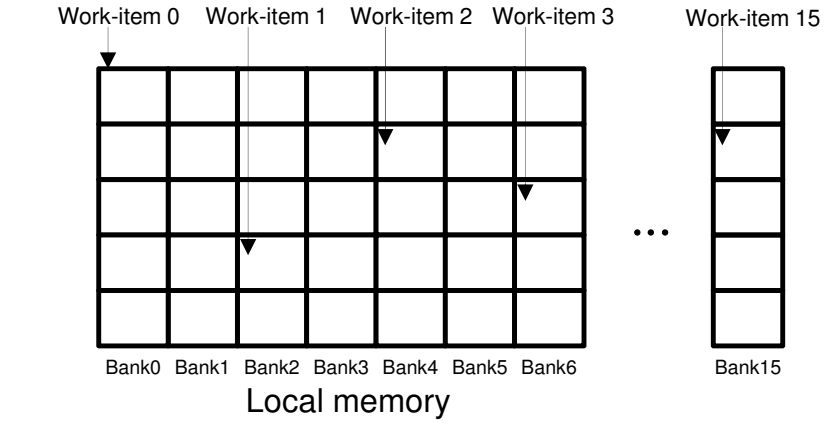
特徴 (i) と (ii) に示したように，命令はバッチ単位で発行され PE で処理される．このため，ワークグループ当たりのワークアイテム数は，**第 2.2.2 節**で述べた条件に加え，バッチサイズであるワークアイテム数の倍数で指定すると，CU 内の

[注 1)] NVIDIA GPU 専用のプログラミング環境 CUDA[40] では，ワークアイテムバッチはワープという用語として知られている．

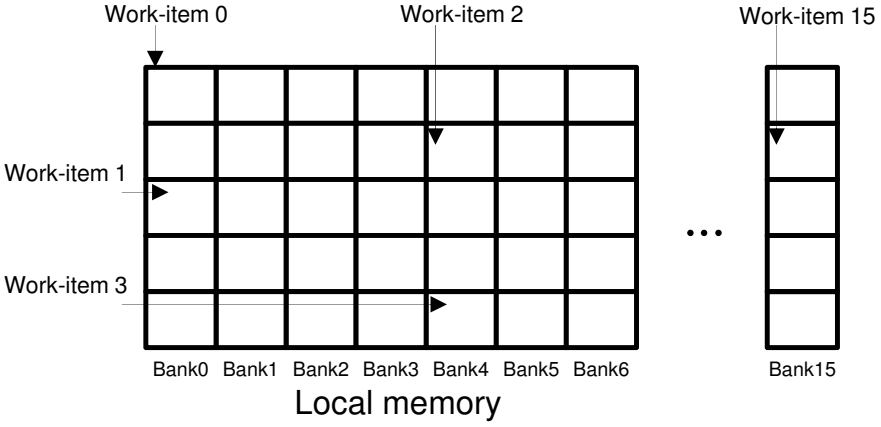
PE の処理効率を上げることできる．例えば，NVIDIA GPU により提供されている GPU では，バッチサイズは 32 ワークアイテムである [41] ため，同 GPU を利用する場合は，ワークグループ当たりのワークアイテム数は 32 の倍数の指定が望ましい．更に，命令のレイテンシはバッチ数で決定されるが，一定数までのバッチ数に対してはレイテンシは固定である．ある閾値を超えたバッチ数に対しては，PE のパイプラインがビジー状態となるため，命令のレイテンシはバッチ数に比例して増加する．ただし，バッチ当たりで見れば，レイテンシは一定値となる特徴[注 2)]がある．また，多数のワークアイテムを起動可能なように，各 CU には数万本の 32-bit レジスタが用意されている [40],[42]．そのため，算術論理演算やメモリアクセス命令に 32-bit 命令を使用することで，レジスタ長を最大限に利用できる．

特徴 (iii) の一例として，**図 2.3** に示すように，複数のワークアイテムからメモリアクセスを並列に受け付けられるように設計されたマルチバンク型のローカルメモリがある．GPU におけるローカルメモリのバンク幅は，通常，レジスタ幅と同様の 32-bit である [40],[42]．複数のワークアイテムが，同図 (a) のようにそれぞれ異なるバンクへアクセスする場合は，バンクコンフリクトは生じない．複数のワークアイテムが同図 (b) のように同じバンクへアクセスするとバンクコンフリクトが生じ，メモリバンド幅の実効値は低下する．同図 (b) の例では，2 つのワークアイテムが同じバンクにアクセスしているので，2 回のアクセスにシリアライズされる．ただし，アクセスするアドレス値が同図 (c) のように完全に同じであれば，ブロードキャストアクセスとなる．このときバンクコンフリクトは発生しない．また，GPU アーキテクチャには，NVIDIA 社の Fermi[43] のように，メモリバンド幅を向上させるために PE とグローバルメモリの間にキャッシュメモリが挿入されているものがある．

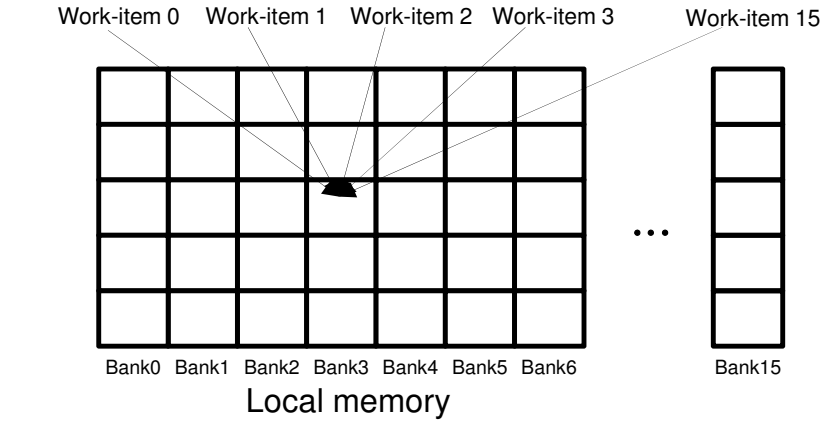
[注 2)] 本論文では，**第 6 章**で実測した結果を示す．



(a) 並列アクセス時



(b) バンクコンフリクト発生時



(c) ブロードキャストアクセス時

図 2.3　GPU におけるローカルメモリへのアクセス

2.4 第 2 章のまとめ

本章では，OpenCL プラットフォームの仕様，それに準拠した並列プロセッサ及び GPU の特徴についてまとめた．**第 4 章**では，本章で述べた GPU アーキテクチャの特徴に沿って，ブロック暗号に共通した GPU への実装方針を導出する．また，GPU アーキテクチャは，多数のワークアイテムによる並列処理でその演算性能を引き出せるように設計されている．これらの特徴は，**第 6 章**において PPF の設計に組み込む．

第 3 章

PPF の設計及び実装評価で用いるブロック暗号のアルゴリズム

3.1 まえがき

ブロック暗号は，入力データを固定長のブロック単位でまとめて暗号化していくため，大容量データの処理に適している．そのため，1970 年代に DES[44] の原型となる Lucifer[45] が登場して以降，ブロック暗号は広く社会へ浸透していくことになった．このような背景から現在では，128-bit ブロック暗号 Camellia[38] のように，8-bit から 64-bit アーキテクチャに至るプラットフォームでの実装を想定したブロック暗号が設計されている．

ブロック暗号のブロック長は，64-bit と 128-bit が代表的である．64-bit ブロック暗号は 1990 年代中頃までは主力として開発されてきた．その代表的なアルゴリズムとして，DES や MISTY1[46] がある．一方，128-bit ブロック暗号は 2000 年前後に開発されるようになった暗号である．128-bit ブロック暗号では，暗号解読法に対する安全性が 64-bit ブロック暗号よりも大幅に向上し，処理速度も向上した．そのため現在では，CRYPTREC はブロック暗号の利用においては，128-bit ブロック暗号を推奨している [21]．実際の利用実績でも，128-bit ブロック暗号の方が様々な製品やシステムで多く利用されているという調査結果 [15] が得られている．また，ブロック暗号の構造は，SPN 構造と Feistel 構造が代表的である．SPN 構造では，換字処理と転置処理を繰り返し適用することで入力された平文ブロックを撹拌する．Feistel 構造では，入力された平文ブロックを左右 2

個のサブブロックに分割して撹拌する．また，限定的ではあるが，SPN と Feistel のハイブリッド構造も一部のブロック暗号で利用されている．

以上の理由から，本研究で実施する PPF の設計及び実装評価には，これら 3 種類の構造に対応した 128-bit ブロック暗号を利用し，その例として AES，Camellia，SC2000 を選択することとした．AES と Camellia はそれぞれ SPN と Feistel 構造であり，SC2000 は SPN と Feistel のハイブリッド構造の 128-bit ブロック暗号である．AES と Camellia は，平成 25 年に改訂された CRYPTREC 暗号リストでは，電子政府推奨暗号リストに採用された暗号アルゴリズムである．また，欧州連合である EU の制定した暗号規格 NESSIE[47] でも同様に推奨暗号として採用されている．SC2000 は，CRYPTREC の推奨候補暗号リストに採用されている．また，並列プロセッサへの暗号実装はソフトウェアで行われる．多くの暗号アルゴリズムでは演算処理による撹拌がベースとなっているが，ソフトウェア実装ではこの演算処理部分の入出力関係をテーブル化する最適化が一般的である [8]．そして，更なる高速化を図るため，実装対象のプロセッサに用意されたメモリ容量の上限の下で，より多くの演算処理のテーブル化を行う．テーブルサイズは大きくなるが，演算量が減り，暗号化処理を高速化できる．PPF は，並列プロセッサ向けに開発するフレームワークであるため，その設計及び実装評価には，ソフトウェア実装に最適化されたテーブルベースによるブロック暗号のアルゴリズムを用いることとした．それぞれの暗号アルゴリズムの詳細は以下に述べる．

3.1.1 暗号利用モード

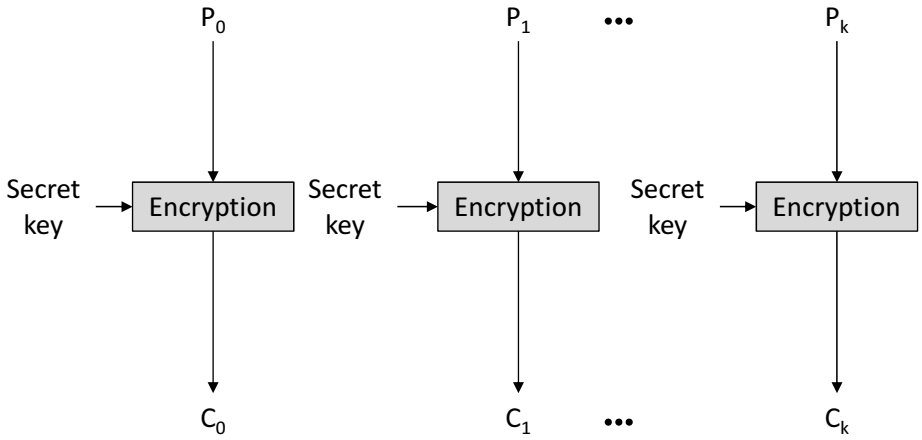
ブロック暗号には，ブロック長よりも長い平文を暗号化する場合に，平文ブロックの処理順序等を規定した暗号利用モードが複数存在する．そのうち，並列処理可能な暗号利用モードとしては，Electronic Code Book（ECB）や，CounTeR（CTR），Xor-encrypt-xor Tweakable code book mode with cipher text Strealing（XTS）[48] がある．これらの暗号利用モードの概要を**図 3.1** に示す．ECB は，1 つの秘密鍵を全ての平文ブロックに作用させるモードである．CTR は，カウンタ

と 1 つの秘密鍵から生成した鍵ストリームを平文ブロックと XOR するモードであり，鍵ストリームは ECB と同様の手順で生成される．XTS は，IEEE 1619-2007 standard[49] で規定されたディスク暗号化のためのモードである．各平文データユニット内の平文ブロックを，ディスクのセクタ番号と 2 つの 128-bit の秘密鍵を使用して，ECB と同様の手順の処理を 2 回実行することで暗号化を行う．このように，並列処理可能なこれら 3 種の暗号化モードは，ECB と同様の手順で暗号化を行っているため，ブロック暗号の GPU 実装による性能の向上に関する議論は，ECB モードによって総合的に評価できる．従って，本研究では，並列処理の暗号利用モードとして，ECB のみ扱うこととした．

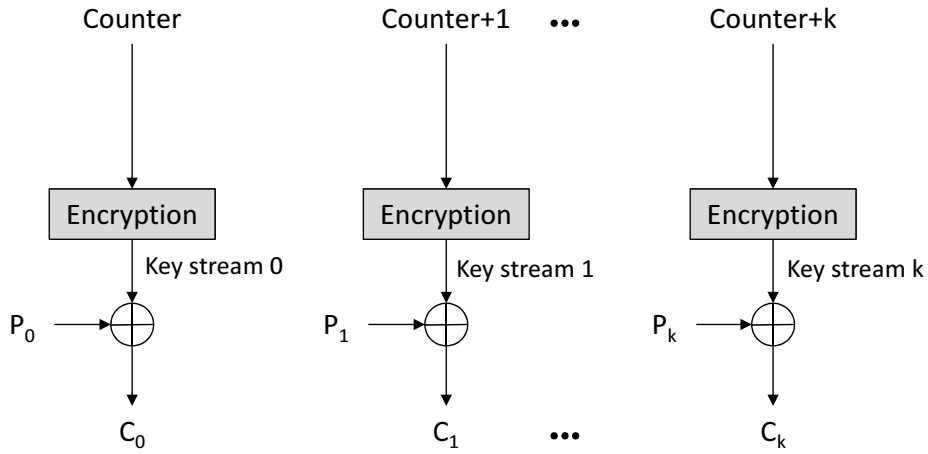
3.1.2 AES

AES[12] は，SPN 構造から成る 128-bit ブロック暗号である．秘密鍵の鍵長は，128-bit，192-bit，256-bit から選択でき，これらの秘密鍵から生成されるラウンド鍵の長さはそれぞれ 10 × 128-bit，12 × 128-bit，14 × 128-bit である．128-bit 鍵，192-bit 鍵，256-bit 鍵モードにおけるデータ撹拌部のラウンド数は，それぞれ 10 段，12 段，14 段である．各ラウンドでは，SubBytes，ShiftRows，MixColumns，AddRoundKey という 4 つの変換が行われる．最終ラウンドは，他のラウンドとは僅かに異なり，MixColumns を含まない．なお，SubBytes は，8-bit 単位で独立に変換を行う非線形置換処理である．ShiftRows は，各 8-bit の位置を一定のルールに従って入れ替える処理である．MixColumns は，4 つの連続する 8-bit を 1 つの 32-bit ブロックと見立て，その各 32-bit ブロックに一定の算術論理演算を行う処理である．AddRoundKey は，4 × 32-bit から成るラウンド鍵を XOR する処理である．

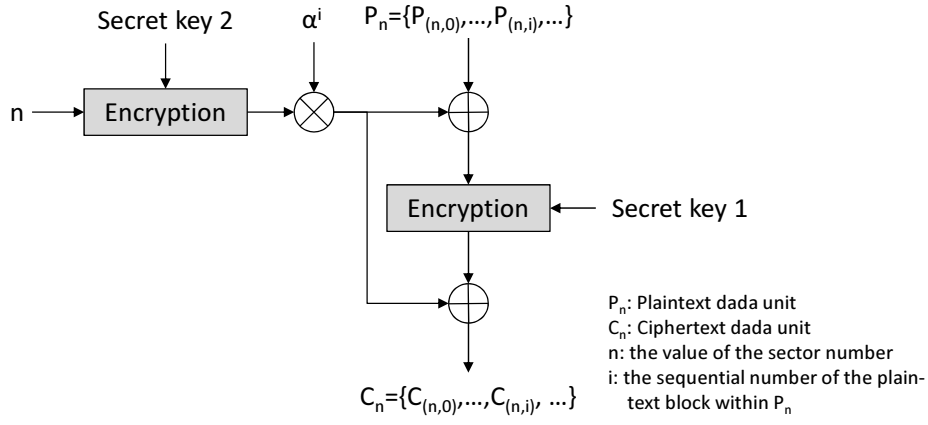
AES の開発者は，AES を高速に処理できる最適化されたアルゴリズムも提案している [50]．このアルゴリズムでは，SubBytes から MixColumns までの変換中，その 8-bit 入力値と 32-bit 出力値が完全な 1 対 1 関係となっていることを利用し，この入出力関係を換字テーブルとして表す．この新たな換字テーブルは T-Box と



(a) ECB モード



(b) CTR モード



(c) XTS モード

図 3.1　3 種の並列処理可能な暗号利用モードの概要

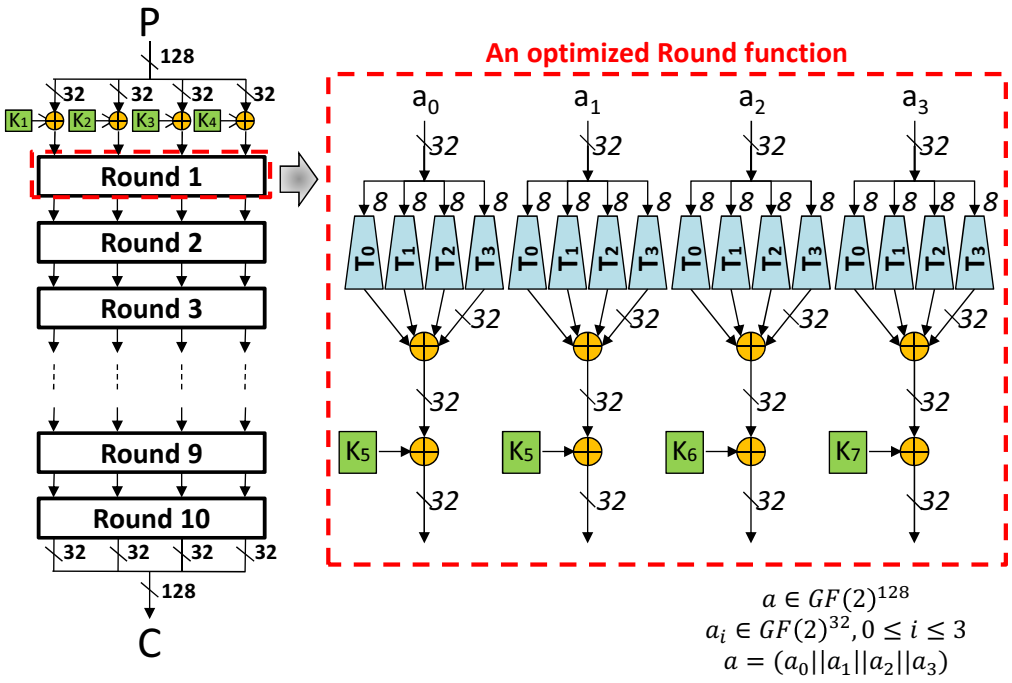


図 3.2　最適化された AES（データ撹拌部）

呼ばれる．T-Box を用いて最適化された AES のデータ撹拌部の概要を**図 3.2** に示す．この AES では，各ラウンドは T-Box（図では T_0〜T_3）と XOR 命令（図では $\oplus$）を用いた変換で構成される．a を各ラウンドの 128-bit 入力とし，これを 4 つに分割した 32-bit をそれぞれ a_0, a_1, a_2, a_3 とすると，各ラウンドの 128-bit 出力 e は

$$e_j = T_0[a_{0,j}] \oplus T_1[a_{1,j+1}] \oplus T_2[a_{2,j+2}] \oplus T_3[a_{3,j+3}] \oplus k_j, \quad (0 \leq j < 4) \quad (3.1)$$

のように表される．ただし，T_0，T_1，T_2，T_3 は換字テーブル，e_j は e を 4 つに分割した各 32-bit を表す．また，k_j は 128-bit から成る各ラウンド鍵の j 番目の列 32-bit を表す．

3.1.3　Camellia

Camellia[38] は，Feistel 構造から成る 128-bit ブロック暗号である．秘密鍵の鍵長は 128-bit，192-bit，256-bit から選択できる．鍵スケジュールにより生成

されるラウンド鍵の長さは，128-bit の秘密鍵では 26 × 64-bit であり，192-bit と 256-bit の秘密鍵ではいずれも 34 × 64-bit である．データ撹拌部のラウンド数は，128-bit 鍵モードでは 18 段であり，192-bit 鍵と 256-bit 鍵モードではいずれも 24 段である．各ラウンドでは，F 関数が 1 回実行される．F 関数には，暗号化中の値の一部である 64-bit とラウンド鍵 64-bit が入力される．F 関数では，入力された 64-bit は，まずラウンド鍵と XOR される．次に，換字テーブルによる換字変換の S 関数と XOR による P 関数により撹拌されて，64-bit が出力される．FL 及び FL^{-1} 関数は，32-bit 単位の AND，OR，XOR，左巡回シフトで構成されており，6 ラウンド毎に挿入される．

Camellia の仕様書 [38] には，ソフトウェア実装での最適化手法が記載されている．同手法に基づいて最適化された Camellia のデータ撹拌部の概要を**図 3.3** に示す．Camellia では，F 関数がデータ撹拌部の大半を占めるため，F 関数の最適化は重要である．F 関数では，演算処理による P 関数の入力値と出力値には 1 対 1 関係があるため，P 関数の入出力関係は換字テーブルで表せることができる．更に，この換字テーブルは S 関数の換字テーブルと結合することもできる．結合後の換字テーブルは，SP テーブルと呼ばれる．このような最適化を施すことにより，F 関数は 8-bit 入力 32-bit 出力の SP テーブルによる換字を 8 回実行するアルゴリズムとして，**図 3.3** のように表すことができる．

3.1.4 SC2000

SC2000[39] は，Feistel と SPN のハイブリッド構造から成る 128-bit ブロック暗号である．秘密鍵の鍵長は，128-bit，192-bit，256-bit から選択できる．鍵スケジュールにより生成されるラウンド鍵の長さは，128-bit の秘密鍵では 56 × 32-bit であり，192-bit と 256-bit の秘密鍵ではいずれも 64 × 32-bit である．データ撹拌部のラウンド数は，128-bit 鍵モードでは 7 段であり，192-bit 鍵と 256-bit 鍵モードではいずれも 8 段である．各ラウンドは，I，B，I，R，R という 5 段階の関数による変換処理から成る．I 関数では，暗号化中の値である 128-bit の入力値が，

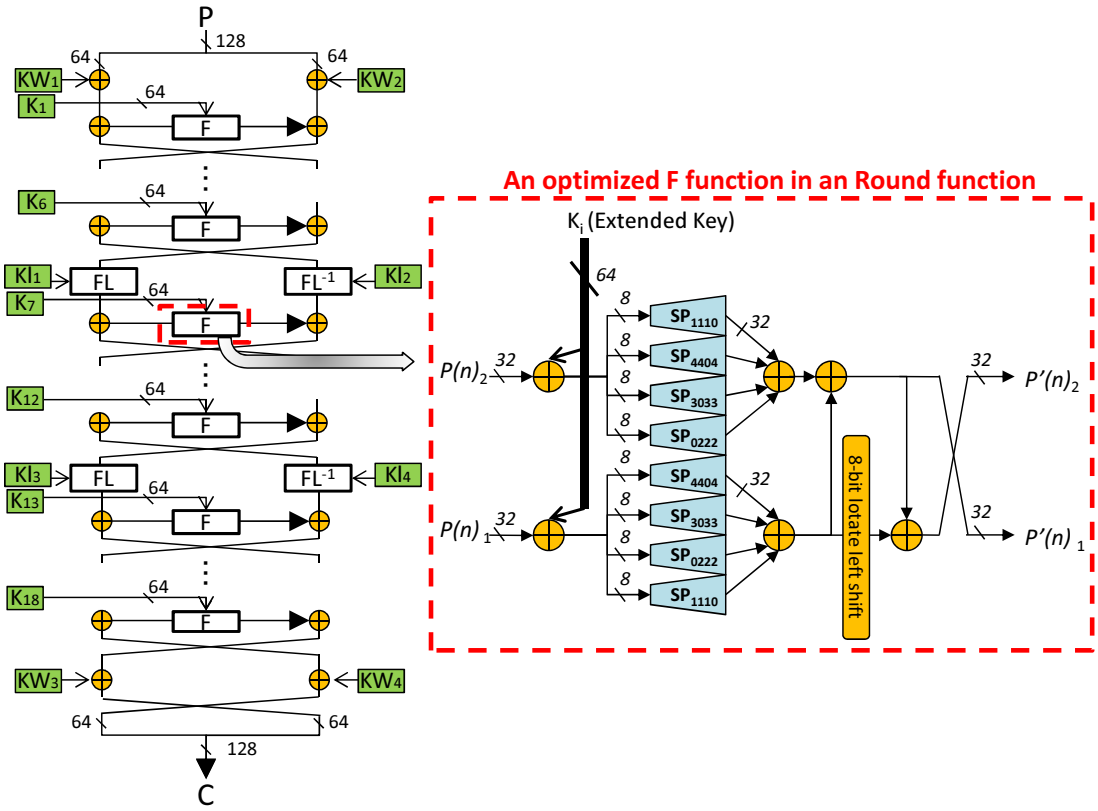


図 3.3　最適化された Camellia（データ撹拌部）

ラウンド鍵と XOR される．B 関数では，入力された 4 × 32-bit を 32 × 4-bit と見なして 32 回の S_4 テーブルで変換し，再度 4 × 32-bit として出力する．R 関数は，S 関数，M 関数，L 関数から成る．S 関数では，入力された 32-bit を，(6-bit, 5-bit，5-bit，5-bit，5-bit，6-bit）または（11-bit，10-bit，11-bit）のように任意のサイズの断片に分割し，S_6 及び S_5 テーブル，または S_{11} 及び S_{10} テーブルで換字変換する．M 関数では，入力は 32-bit × 32-bit から成る M 行列と $GF(2)$ 上で乗算される．L 関数では，2 × 32-bit 入力が定数値による AND や XOR 命令で撹拌される．最終ラウンドは他のラウンドとは僅かに異なり，2 回の R 関数を含まない．

SC2000 の仕様書 [39] には，ソフトウェア実装での最適化手法が記載されている．同手法に基づいて最適化された SC2000 のデータ撹拌部の概要を**図 3.4** に示

す．SC2000 では，B 関数と，R 関数内の S 関数及び M 関数が最適化できる．B 関数では，S_4 テーブルへの 32 回のアクセスが論理演算に変換でき，S_4 テーブルによる処理よりも高速に処理できる．また，演算処理から成る M 関数はその入出力が 1 対 1 関係であるため，換字テーブルに変換することが可能であり，更に S 関数における換字テーブルと結合できる．結合後のテーブルは，SM テーブルと呼ばれる．SC2000 の仕様書 [39] によれば，このようにして結合された SM 関数へ入力される 32-bit の構成法として，(6-bit, 5-bit, 5-bit, 5-bit, 5-bit, 6-bit)，(6-bit, 10-bit, 10-bit, 6-bit)，(11-bit, 10-bit, 11-bit)，(16-bit, 16-bit) が選択できる．換字テーブルもこの要領に従って構成される．これら 4 種類のテーブル構成法とそれらの換字テーブルのアクセス回数及びサイズとの関係を**表 3.1** に示す．合成するテーブルの数が少ないテーブル構成法を用いると，サイズは大きくなる反面，アクセス回数を少なくすることができる．高い処理速度を引き出すには，実装対象の並列プロセッサのアーキテクチャを確認し，そのメモリ容量に応じたテーブル構成法のアルゴリズムを実装することが重要である．

表 3.1　SM 関数で選択可能なテーブル構成法と F 関数内での換字テーブルへのアクセス回数，総テーブルサイズ

Table constitution method	# table accesses per SM function	Total table size
(6, 5, 5, 5, 5, 6)	6	1 KByte
(6, 10, 10, 6)	4	8.5 KByte
(11, 10, 11)	3	20 KByte
(16, 16)	2	512 KByte

3.2　第 3 章のまとめ

注 1) 代表として，(11-bit，10-bit，11-bit) のテーブル構成法によるアルゴリズムを示している．

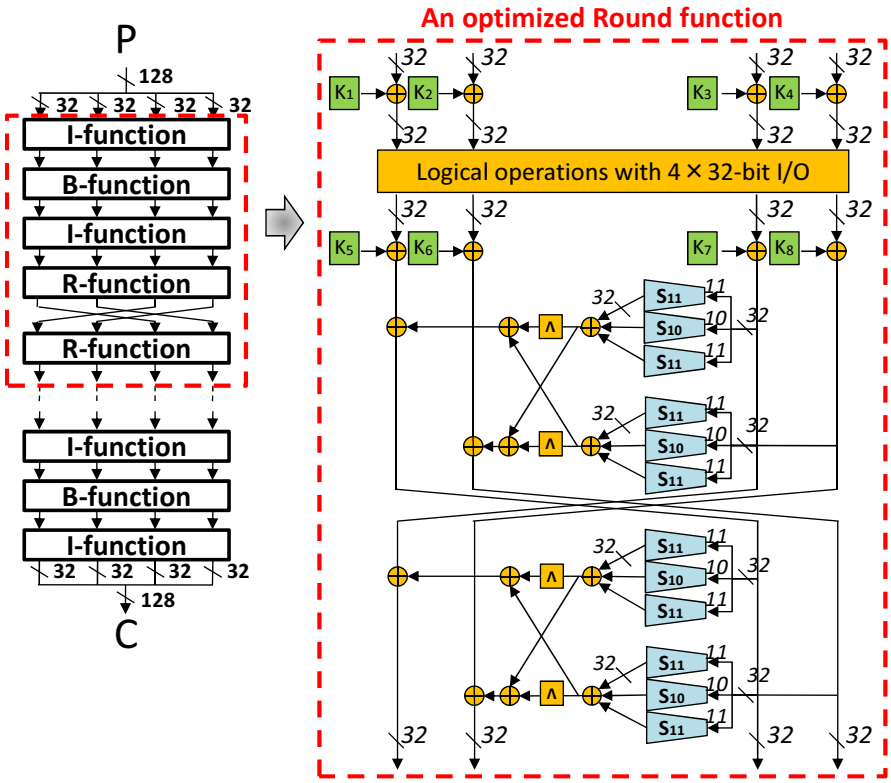


図 3.4　最適化された SC2000（データ撹拌部）[注 1)]

本章では，PPF の設計及び実装評価に用いる，CRYPTREC 推奨の 128-bit ブロック暗号についてまとめた．ブロック暗号は，一般に，算術論理演算による換字で平文ブロックを撹拌する．更に，多くのブロック暗号では，撹拌中に入力と出力の関係が 1 対 1 関係となる部分では，その入出力関係を換字テーブルで表すことで最適化できる．特に，GPU には暗号化処理に利用できる多様な種類のメモリがあるため，換字テーブルを積極的に用いた暗号化アルゴリズムの最適化は，GPU から高い処理速度を引き出すために有効である．

また，並列処理可能な暗号利用モードとして ECB，CBC，XTS が知られているが，暗号化はいずれのモードでも ECB と同様の手順で行われる．従って，GPU を用いたブロック暗号による性能の向上は，ECB モードで総合的に議論できる．

第 4 章

GPU へのブロック暗号の実装方針の導出

4.1 まえがき

実利用されているブロック暗号は，IPA の調査結果 [15] の通り多種多様である．このため，一見すると，各々のブロック暗号ごとに GPU への実装方針を導出する必要性があると考えられる．しかし，それぞれのブロック暗号で GPU への実装方針が異なると，GPU へのブロック暗号の実装性能を予測する PPF の設計及び利用が複雑になってしまう恐れがある．そのため本章では，暗号設計者にとって理解が容易で利用の簡単な PPF の設計に資するため，複数のブロック暗号に共通した GPU への実装方針を導出する．

4.2 導出の指針

GPU を用いた暗号化処理の流れを**図 4.1** に示す．**第 3.1.1 節**で示した理由から，GPU を用いた暗号化処理の並列実装では，暗号利用モードに ECB を用いている．このため，ラウンド鍵は秘密鍵から 1 種類のみを生成すればよく，GPU の並列処理機能を利用する必要がない．従ってラウンド鍵は，同図 ① のように，ホスト側で生成することとした．なお，平文のサイズが十分大きいとき，GPU での暗号化時間が長くなり，ホスト側でのラウンド鍵の生成時間は無視できるほど小さくなる．① 終了後，平文，換字テーブル，ラウンド鍵はホスト側のメモリ空間に格納されている．そのため，ラウンド鍵を生成した後，平文を ② のように GPU のグローバルメモリに転送する．ラウンド鍵と換字テーブルは暗号化中に

不変のデータなので，読み出しのみ可能な領域であるコンスタントメモリに転送する．これらデータの転送完了後，③のように，GPU でブロック暗号による暗号化処理が実行される．暗号化完了後，暗号文が④のようにホスト側のメモリへリードバックされる．なお，本章で導出するのは，③の部分の最適な並列処理の実装方針である．

一般に，並列プロセッサでは，内蔵された複数のコアを利用した並列処理によって，その処理速度を向上させることができる．しかし，並列化には通常，同期等のオーバーヘッドを伴うため，過度な並列化は却って処理速度を低下させる恐れがある．本研究では，各プロセッサコアにディスパッチするタスクサイズを計算粒度と呼称する．一般に並列処理では，最適な計算粒度は並列化するアプリケーションや実装するアーキテクチャに依存する．そのため，GPU へのブロック暗号の実装方針を導出するためには，計算粒度が暗号化の処理速度に与える影響を定量的に考察し，最適な計算粒度を導出しなければならない．

また，**第 2.3 節**で述べたように，GPU には性質の異なる多様なメモリが用意されている．そのため，ラウンド鍵，換字テーブル，平文といったデータを，暗号化処理中，どのメモリへ配置するかという最適化戦略は，暗号処理速度に大きく影響してくる．また，GPU におけるローカルメモリはマルチバンク型である．計算粒度が異なるとワークアイテムのタスクサイズが異なるので，各ワークアイテムがアクセスするメモリアドレスも異なってくる．そのため，ローカルメモリを利用する最適化では，計算粒度によってバンクコンフリクトの発生回数が変化してくる．以上の理由から，GPU へのブロック暗号の実装方針を導出するためには，計算粒度とデータのメモリ配置戦略を総合的に考察する必要がある．

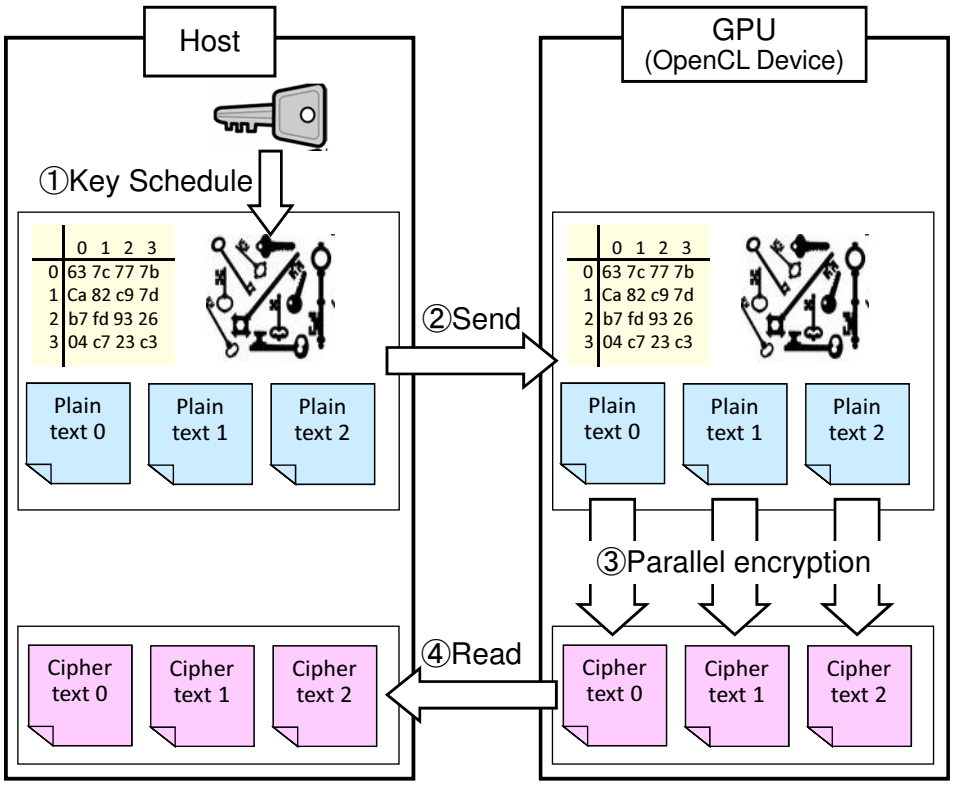


図 4.1　GPU を用いたブロック暗号による暗号化の実行の流れ

4.3　計算粒度

本章では，実装方針の導出は，取り得る実装を実際の GPU へ実装し，その処理速度を比較，分析する方法をとる．ただし，あらゆるブロック暗号に対し，取り得る全ての計算粒度を考察すると，各々の暗号における計算粒度のプログラムを全て実装する必要があり，作業量が膨大となるため現実的ではない．そのため，計算粒度が処理速度に与える影響の考察においては，ブロック暗号が SPN と Feistel 構造に大別されることを利用し，導出に利用する暗号アルゴリズムを絞ることにした．

SPN 構造と Feistel 構造の比較を**図 4.2** に示す．ブロック暗号では，平文は固定長のブロックに分割されてそれぞれ暗号化されるため，**第 3.1.1 節**で述べたよ

うな並列処理可能な暗号利用モードを使用することで，そのブロック長ごとの並列性の抽出が可能である．この並列性に加え，SPN では，平文ブロックが複数個のサブブロックに分割された後，それぞれのサブブロックが換字処理と転置処理で撹拌されていくので，アルゴリズム内部に更なる並列性を持つ．このため，SPN 構造の並列処理において考察すべき計算粒度には，ブロック長に加え，そのブロック内のサブブロック長という複数個の選択肢が存在する．一方，Feistel 構造では，左右 2 個のサブブロックに分割されて暗号化されていくが，アルゴリズム内部に並列性を有していない[注 1)]．これは例えば，Feistel 構造の 1 段目の処理では，左側のサブブロックは，右側のサブブロックを入力とした F 関数の出力結果を受けて，2 段目以降を処理しなければならないからである．SPN と Feistel のハイブリッド構造も，並列性に関しては Feistel 構造と同様の性質を持つ．このため，Feistel 構造及び SPN と Feistel のハイブリッド構造の並列処理において選択可能な計算粒度は，ブロック長の 1 種類のみである．ブロック暗号において，代表的な 3 種の構造のアルゴリズムが有する並列性を**表 4.1** にまとめる．

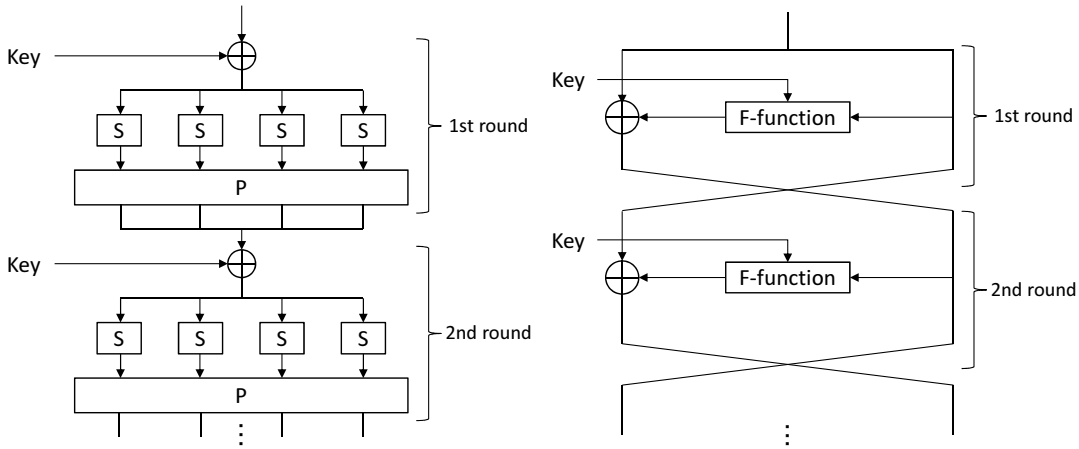
このように，SPN 構造は，他の 2 種の構造よりも考察すべき計算粒度の種類が多く，かつ他の 2 種類の構造で取り得る計算粒度が含まれている．従って，実装方針の導出に利用するブロック暗号として，SPN 構造のアルゴリズムを選択することとした．これにより，計算粒度が処理速度に与える影響について，他の 2 種類の構造が取り得る計算粒度を含め，かつより詳細な分析を行うことができる．また，SPN 構造のブロック暗号には AES が該当するため，計算粒度が処理速度に与える影響は，AES を題材として考察することとした．

ソフトウェア実装に最適化された AES の各ラウンドは，**第 3.1.2 節**で示したように，換字テーブルへのアクセスとラウンド鍵との XOR による 32-bit 単位の

[注 1)] Feistel 構造を拡張した構造として，分割するサブブロック数を 2 よりも大きくした一般化 Feistel 構造 [51] も提案されている．しかし，並列性に関しては，Feistel 構造と同様の性質を持つので，Feistel 構造に含めて扱う．

変換処理を並列に 4 個並べた構造となっているため，その内部に 32-bit×4 個の並列性を有する．そのため，128-bit ブロック暗号である AES を並列化する上では，各ワークアイテムが並列に処理するタスクサイズの大小によって，次の 4 種類の計算粒度を考えることができる．

- 16 Bytes/Thread
- 8 Bytes/Thread
- 4 Bytes/Thread
- 1 Byte/Thread



(a) SPN 構造　　(b) Feistel 構造

図 4.2　SPN 構造と Feistel 構造の比較

表 4.1　代表的な 3 種類のブロック暗号の構造が有する並列性

Data randomization structure	Parallelism	Example
SPN	Plaintext block length or some patterns of sub-blocks within a plaintext block	AES
Feistel	Plaintext block length only	Camellia
Hybrid of SPN and Feistel	Plaintext block length only	SC2000

4.3.1 16 Bytes/Thread

16 Bytes/Thread の計算粒度による並列処理の概要を**図 4.3** に示す．この計算粒度では，各ワークアイテムが 128-bit（16-Byte）の平文ブロックを独立かつ並列に処理する．この粒度では，128-bit の平文ブロックから抽出される並列度のみを利用しており，ワークアイテム間の同期やデータ共有を必要としないため，他の計算粒度よりも並列化に伴うオーバーヘッドは低い．

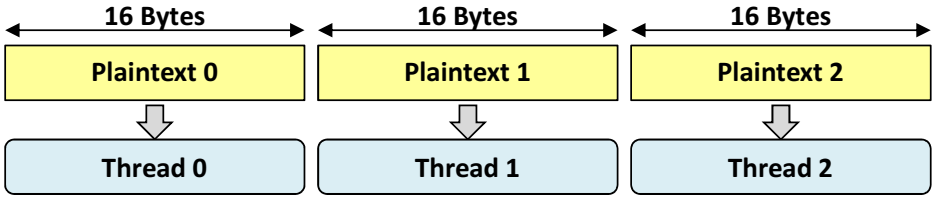


図 4.3 16 Bytes/Thread の計算粒度による並列処理

4.3.2 8 Bytes/Thread 及び 4 Bytes/Thread

8 Bytes/Thread と 4 Bytes/Thread の計算粒度による並列処理の概要を，**図 4.4** と**図 4.5** にそれぞれ示す．8 Bytes/Thread の計算粒度では，2 つのワークアイテムが 1 つの平文ブロックを協調して暗号化する．一方，4 Bytes/Thread の計算粒度では，4 つのワークアイテムが 1 つの平文ブロックを協調して暗号化する．これらの計算粒度は，暗号アルゴリズムに内在する並列性を利用し，16 Bytes/Thread よりも高い並列度を引き出すことができる長所を持つ．ただし，複数のワークアイテムが 1 つの平文ブロックを協調して暗号化するため，暗号化途中の中間値はワークアイテム間で共有する必要がある．ワークアイテム間でのデータ共有は，**第 2.2.2 節**で述べたように，ローカルメモリを仲介して行う．また，ワークアイテム間でデータ共有を行うには，ローカルメモリへのデータ格納後にワークアイテム間で同期をとる必要がある．このように，これらの計算粒度には，16 Bytes/Thread より並列化に伴うオーバーヘッドが大きくなる短所がある．

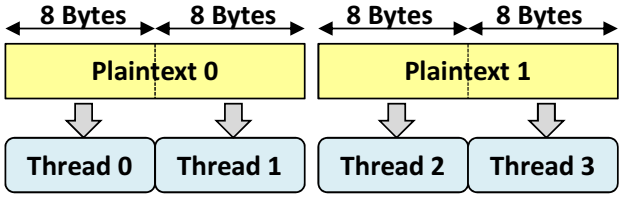


図 4.4　8 Bytes/Thread の計算粒度による並列処理

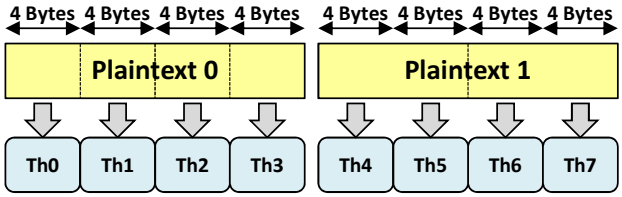


図 4.5　4 Bytes/Thread の計算粒度による並列処理

4.3.3　1 Byte/Thread

1 Byte/Thread は，16 ワークアイテムが協調して 1 つの平文ブロックを暗号化する計算粒度である．この計算粒度では，8 Bytes/Thread や 4 Bytes/Thread よりも高い並列度を引き出すことができる長所を持つ．ただし，NVIDIA 社や AMD 社の GPU は 32-bit プロセッサとして設計されているため，レジスタ[注 2)]長を十分に活用することができない短所がある．

4.4　暗号化処理に必要なデータのメモリ配置戦略

GPU へのブロック暗号の並列実装では，メモリへ配置するデータの候補として，

- 平文ブロックの暗号化処理中の値（以下，中間値）

[注 2)] **第 2.3 章**で説明した通り，OpenCL プラットフォームにおける GPU アーキテクチャでは，プライベートメモリとレジスタは同義である．ただし，コンピュータアーキテクチャではレジスタという用語が一般的なので，本章では混乱を避けるため，レジスタという用語で説明する．

- 換字テーブル
- ラウンド鍵

という 3 種類が考えられる．これらのデータをどのメモリへ配置するかという最適化戦略では，各計算粒度の並列実装において取り得るメモリ配置方法をデータの性質に沿って決定する．また，**第 4.3 節**で述べたように計算粒度の考察では暗号アルゴリズムを AES に固定したため，メモリ配置の考察も AES に固定して行う．

4.4.1　ラウンド鍵と換字テーブル

換字テーブルとラウンド鍵は，全ワークアイテム間で共有される，暗号化処理中に不変のデータである．一方，コンスタントメモリとローカルメモリはそれぞれ読み出し専用，読み書き可能であり，かつ，いずれもグローバルメモリよりも小さいレイテンシでアクセスできるメモリである．また，ラウンド鍵と換字テーブルは，レジスタに配置するには容量が大きい．このような特徴を考慮すると，これらのデータの配置先の候補は，コンスタントメモリまたはローカルメモリとなる．ただし，コンスタントメモリは，ワークアイテムによるアクセスがキャッシュヒットする場合に低いレイテンシとなるメモリである．また，ローカルメモリは，各ワークアイテムのアクセスするアドレスが異なるバンクに属していれば並列に処理できるが，そうでない場合はバンクコンフリクトの問題が付きまとう．

4.4.2　平文ブロックの暗号化処理中の値

GPU で暗号化処理が開始される時点では，平文はグローバルメモリに格納されている．16 Bytes/Thread の計算粒度では，ワークアイテムは互いの中間値を共有する必要がない．そのため，それらの値はレジスタに格納したまま暗号化処理を完了できる．しかし，その他の計算粒度では，複数のワークアイテムが 1 つの平文ブロックを協調して暗号化するため，ローカルメモリに格納する必要がある．ローカルメモリはマルチバンク型のメモリであり，複数のワークアイテムが

同じバンクにアクセスする状況では，バンクコンフリクトが発生してしまう．そのため，メモリ空間内に中間値を配置する方法として，Array of Structure（以下，AoS) と Structure of Array（以下，SoA）という 2 通りを考えることができる [52]．AoS は平文ブロックの中間値をそのままの並びで配置し，SoA は平文ブロックの中間値のうち同じ要素を 1 つの配列としてまとめて配置する方法である．AoS と SoA では，中間値の各 32-bit を格納するバンク番号は異なってくる．一例として，8 Bytes/Thread の計算粒度で並列実装されたワークアイテムが，AoS と SoA という並び方で格納された中間値のデータへアクセスしている状況を，**図 4.6** と**図 4.7** にそれぞれ示す．AoS と SoA による並び方によって，ワークアイテムがアクセスするバンク番号が異なり，バンクコンフリクトの発生状況に違いが生じていることが分かる．以上の理由から，各データのメモリ配置先の候補は**表 4.2** のようにまとめられる．

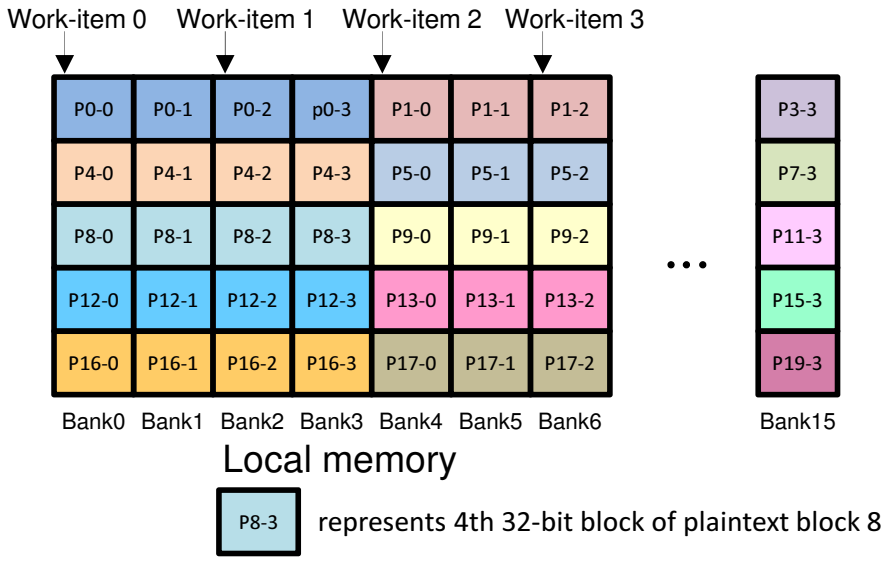


図 4.6　AoS でローカルメモリ内に配置された中間値に対する，8 Bytes/Thread の計算粒度で並列実装されたワークアイテムのアクセス状況

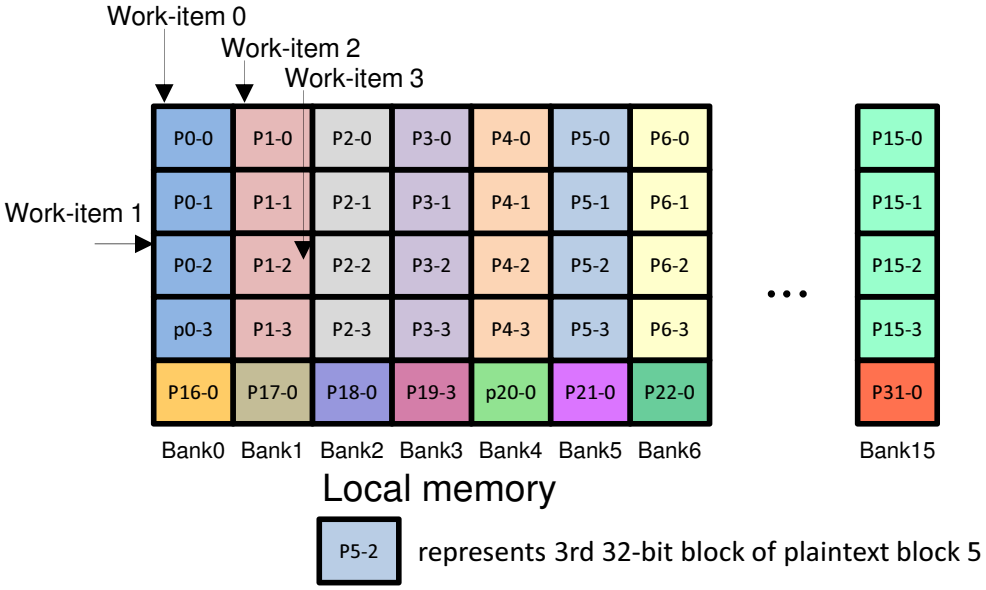


図 4.7　SoA でローカルメモリ内に配置された中間値に対する，8 Bytes/Thread の計算粒度で並列実装されたワークアイテムのアクセス状況

表 4.2　レイテンシの低減を目的とした各データの配置先の候補

	Constant memory (Read only)	Local memory (Read and write)	Register (Read and write)
Substitution tables (Invariant)	○	○	×
Round keys (Invariant)	○	○	×
Intermediate value (Variant)	×	○ (AoS or SoA)	○ (Case of 16 Bytes/Thread)

4.5 考察

4.5.1 各並列実装における処理速度の測定環境

考察に用いるGPUの選択は，導出した実装方針の適用範囲に汎用性を持たせるため重要である．OpenCLプラットフォームは，NVIDIA社のGPUコンピューティング基盤であるCUDAに基づく形でその仕様が策定され，2008年12月に公開された[53]．CUDAに対応したNVIDIA GPUのアーキテクチャのうち，最もプリミティブなものは，OpenCLとほぼ同時期に登場したGT200[54]である．そして，以降のNVIDIA GPUは，このアーキテクチャの特徴を継承する形で進化してきた．こうした歴史的背景から，**第2.2.2節**で述べたOpenCLプラットフォームの特徴は，GT200のアーキテクチャと類似している．従って本節の考察には，GT200アーキテクチャの利用が相応しいと考えた．このため，測定に用いるGPUには，NVIDIA GT200アーキテクチャであるGeforce GTX 280を選択することとした．処理速度の測定環境とGeforce GTX 280の仕様をそれぞれ**表4.4**と**表4.5**に示す．

ワークグループ当たりのワークアイテム数とワークグループ数は，OpenCLカーネルの最適化に重要なパラメータである．**第2.2.2節**と**第2.3節**でそれぞれ述べたように，ワークグループ数はGPUのCU数の倍数を指定し，ワークグループ数当たりのワークアイテム数はバッチサイズであるワークアイテム数の倍数を指定することが望ましい．本章で用いたGPUであるGeforce GTX 280は30個のCUを内蔵するので，ワークグループ数においては30の倍数でチューニングを試みた．ワークグループ数を順に増加させたところ，それぞれの並列実装の処理速度は90で最も高い値に達し，以降のワークグループ数では収束したので，同パラメータは90に設定することとした．また，**第2.3節**で述べたように，Geforce GTX 280を含めたNVIDIA GPUでは，バッチサイズは32ワークアイテムとなっている[41]．そのため，ワークグループ当たりのワークアイテム数においては32の倍数でチューニングを試みた．その結果，それぞれの並列実装の処理速度は，

ワークグループ当たりのワークアイテム数が 512（16 バッチ）のときに最も大きい値となったので，同パラメータは 512 に設定することとした．

各並列実装のピーク性能を比較するため，平文のサイズは処理速度が収束する 360 MB に固定し，平文に設定する値はランダム値とした．また，前述の通り，暗号利用モードには ECB を用いた．

表 4.3　実装方針導出のために分析を必要とするパラメータ

Granularity		16 Bytes/Thread, 8 Bytes/Thread, 4 Bytes/Thread, 1 Byte/Thread
Memory allocation	Substitution tables	Constant memory, Local memory
	Round keys	Constant memory, Local memory
	Intermediate value	Local memory (AoS or SoA), register

表 4.4　測定環境

CPU	Intel Core i7-2600K（4 cores，Hyper-thread technology available）
Motherboard	ASUS P8Z68-V
Memory capacity	32 GB
OS	CentOS 6.0（Kernel version 2.6.32-71）
Host compiler	gcc version 4.4.4（option -O3）
OpenCL compiler	Bundled with nvcc version 4.2

表 4.5　Geforce GTX 280 の仕様

Name	Geforce GTX 280
Architecture	GT200
Core clock frequency	1.296 GHz
# CUs	30
# PEs per CU	8
Local memory capacity per CU	16 KB
# local memory banks	16
Maximum # work-item batches per CU	16

4.5.2　処理速度の測定結果と分析

各計算粒度の並列実装による処理速度について，16 Bytes/Thread の測定結果を**表 4.6** に，8 Bytes/Thread，4 Bytes/Thread，1 Byte/Thread の測定結果を**表 4.7** にそれぞれ示す．処理速度の単位は，単位時間に暗号化できるビット数であるスループットで表すこととする．

表 4.6　16 Bytes/Thread の計算粒度における AES の各並列実装の処理速度比較

Implementation ID	(16Bs-I)	(16Bs-II)	(16Bs-III)	(16Bs-IV)	(16Bs-V)
Memory allocation of substitution tables	Constant	Local	Local	Local	Local
Memory allocation of round keys	Constant	Constant	Local	Local	Local
Memory allocation of intermediate value	register	register	register	Local	Local
Memory allocation scheme of intermediate value	–	–	–	SoA	AoS
Throughput	4.0 Gbps	27.3 Gbps	27.9 Gbps	23.7 Gbps	17.2 Gbps

表 4.7　8 Bytes/Thread，4 Bytes/Thread，1 Byte/Thread の計算粒度における AES の各並列実装の処理速度比較．Bs/Th と B/Th は，それぞれ Bytes/Thread と Byte/Thread を表す．

Implementation ID	(8Bs-I)	(8Bs-II)	(8Bs-III)	(4Bs-I)	(4Bs-II)	(4Bs-III)	(1B-I)
Granularity	8 Bs/Th	8 Bs/Th	8 Bs/Th	4 Bs/Th	4 Bs/Th	4 Bs/Th	1 B/Th
Memory allocation of substitution tables	Local memory						
Memory allocation of round keys	Constant	Local	Constant	Constant	Local	Constant	Local
Memory allocation of intermediate value	Local memory						
Allocation scheme of the intermediate value	AoS	AoS	SoA	AoS	AoS	SoA	AoS
Throughput	21.3 Gbps	18.9 Gbps	18.5 Gbps	20.1 Gbps	19.8 Gbps	13.6 Gbps	2.3 Gbps

(1)　計算粒度が処理速度に与える影響

表 4.6 と**表 4.7** から，各計算粒度の最大値を抽出し，比較した結果を**図 4.8** に示す.

16 Bytes/Thread の計算粒度による処理速度は，27.9 Gbps と全計算粒度の中で最も高い値となった．この計算粒度では，ワークアイテムは平文ブロックを独立かつ並列に暗号化する．そのため，平文ブロックの中間値を共有する必要がない．平文ブロックの中間値は，暗号化完了まで，メモリ領域の中で最もレイテンシが低いレジスタに格納し続けることができる.

一方，8 Bytes/Thread と 4 Bytes/Thread の計算粒度における処理速度の最大値は，それぞれ 21.3 Gbps，20.1 Gbps となり，16 Bytes/Thread と比較して 30 %程度低い値となった．この計算粒度では，平文ブロックの中間値は，ワークアイテム間での共有のためローカルメモリに格納する必要がある．複数のワークアイテム間で値を共有するためには，その値をローカルメモリへ格納した後に，ワークアイテム間の同期をとる必要がある．また，ローカルメモリへのアクセスでは，バンクコンフリクト発生の問題がつきまとう．これらの理由から，8 Bytes/Thread と 4 Bytes/Thread の計算粒度における処理速度が，16 Bytes/Thread よりも低下したと考えられる.

なお，1 Bytes/Thread では，その処理速度は 2.3 Gbps と非常に低い値となった．この計算粒度でも 8 Bytes/Thread や 4 Bytes/Thread と同様，暗号中の値はワークアイテム間の共有のため，ローカルメモリに格納場所する必要がある．更に，各ワークアイテムは 1-bit 単位の処理を実行するため，GPU のレジスタ長である 32-bit を有効に活用できない．これらの理由から，処理速度が大幅に低下したと考えられる.

(2)　換字テーブルのメモリ配置戦略が処理速度に与える影響

本節 (1) で述べたように，最も高い処理速度は 16 Bytes/Thread の計算粒度による並列実装で得られた．このため，3 種のデータ（換字テーブル，ラウンド鍵,

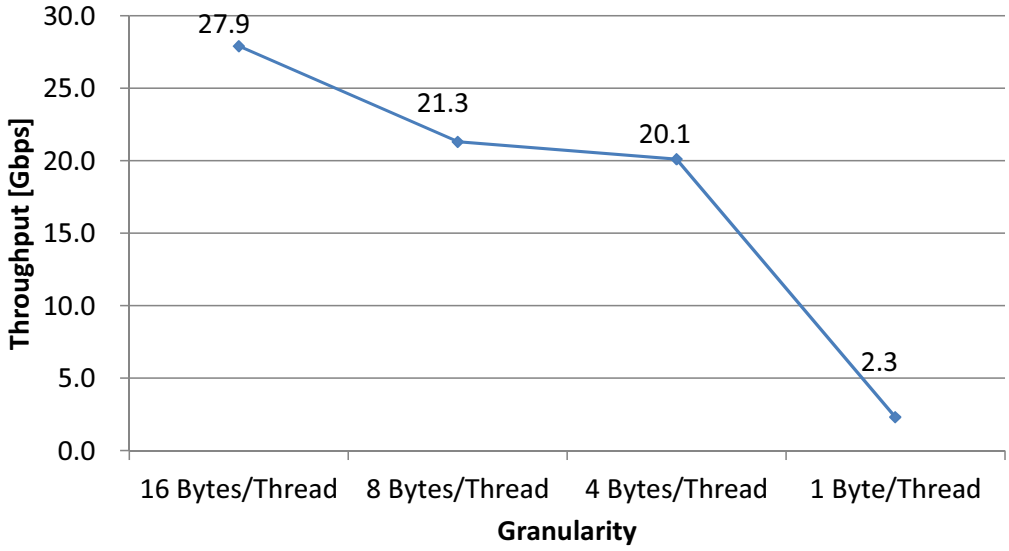


図 4.8　各計算粒度による並列実装の処理速度比較

平文の中間値）のメモリ配置戦略が暗号化処理速度に与える影響は，この計算粒度を中心に考察する．

まず，換字テーブルのメモリ配置戦略が，暗号化処理速度に与える影響を考察する．**表 4.6** に示した 16 Bytes/Thread の計算粒度による並列実装 (16Bs-I) と (16Bs-II) では，換字テーブルの配置先のみが異なっている．そのため，これら 2 種の並列実装の比較により，16 Bytes/Thread の並列実装では，換字テーブルの配置先はローカルメモリとコンスタントメモリのどちらが適しているかを分析できる．換字テーブルをローカルメモリに配置した並列実装 (16Bs-II) では，処理速度は 27.3 Gbps であったのに対し，コンスタントメモリに配置した (16Bs-I) では，処理速度は 4.0 Gbps と極端に低い値になった．従って，この計算粒度では，換字テーブルはローカルメモリへ配置すべきという結論に至る．この速度差の理由として，換字テーブルへのアクセスパターンが，コンスタントメモリに適していないことが考えられる．つまり，コンスタントメモリは，各ワークアイテムによるアクセスがキャッシュヒットする場合に低いレイテンシとなる．また，換字テーブルは平文ブロックを撹拌するためのデータであるため，そのアクセスパ

ターンはランダムである．そのため，換字テーブルをコンスタントメモリに配置する実装では，ワークアイテムによるキャッシュヒットが十分に期待できず，アクセスレイテンシは比較的大きくなった．なお，換字テーブルをローカルメモリに配置する実装では，テーブルへのアクセスパターンがランダムであるため，バンクコンフリクトの発生は回避できない．それでも，キャッシュヒットの期待できないコンスタントメモリと比較すると，最適化にはローカルメモリの利用が有効であることが分かる．

(3)　ラウンド鍵のメモリ配置戦略が処理速度に与える影響

次に，ラウンド鍵のメモリ配置戦略が，処理速度に与える影響を考察する．**表 4.6** に示した 16 Bytes/Thread の計算粒度による並列実装 (16Bs-II)，(16Bs-III) では，ラウンド鍵の配置先のみが異なっている．そのため，これら 2 種の並列実装の比較により，16 Bytes/Thread の並列実装では，ラウンド鍵の配置先はローカルメモリとコンスタントメモリのどちらが適しているかを分析できる．ラウンド鍵をローカルメモリに配置した並列実装 (16Bs-III) では，処理速度は 27.9 Gbps であったのに対し，コンスタントメモリに配置した (16Bs-II) では，処理速度は 27.3 Gbps であった．従って，この計算粒度では，ラウンド鍵もローカルメモリへ配置すべきという結論に至る．ただし，その速度差は僅か約 2 %であり，この理由は，ラウンド鍵へのアクセスパターンが規則的であるため，(16Bs-II) の並列実装ではコンスタントメモリから十分なキャッシュヒットが得られたことにあると考えられる．

一方，8 Bytes/Thread と 4 Bytes/Thread の計算粒度ではいずれも，ラウンド鍵をコンスタントメモリに配置した並列実装の処理速度は，ローカルメモリに配置した並列実装よりも高い値となった．これはラウンド鍵をローカルメモリに配置した方が高い処理速度となる 16 Bytes/Thread とは正反対の結果である．このため，プロファイラ [55] を用いてその原因を調べたところ，この理由は，8 Bytes/Thread 及び 4 Bytes/Thread の並列実装では，16 Bytes/Thread よ

りもバンクコンフリクトが多く発生したためであることが分かった．これは，16 Bytes/Thread の計算粒度では，並列実行されている各ワークアイテムの処理は全く同一であるため，ラウンド鍵へのアクセス時のアドレスが完全に同じとなるからである．つまり，このとき，**図 2.3**(c) に示したようなブロードキャストアクセスとなり，バンクコンフリクトが発生しない．一方，例えば 8 Bytes/Thread のように，1 つの平文ブロックを並列処理する計算粒度では，ワークアイテムはラウンド鍵にストライドアクセスし，アクセスするバンクでコンフリクトが発生する．このため，バンクコンフリクトの発生を回避できず，16 Bytes/Thread の並列実装よりも処理速度は低下してしまう．

(4)　平文ブロックの中間値のメモリ配置戦略が処理速度に与える影響

最後に，平文ブロックの中間値のメモリ配置戦略が処理速度に与える影響を考察する．**表 4.6** に示した 16 Bytes/Thread の計算粒度による並列実装 (16Bs-III), (16Bs-IV)，(16Bs-V) では，ラウンド鍵のメモリ配置先が異なっている．中間値をレジスタに格納した並列実装 (16Bs-III) では，処理速度は 27.9 Gbps であったのに対し，中間値をローカルメモリに配置した (16Bs-IV) 及び (16Bs-V) ではそれぞれ 23.7 Gbps 及び 17.2 Gbps であった．従って，この計算粒度では，中間値はレジスタに配置すべきということなる．

一方，8 Bytes/Thread，4 Bytes/Thread，1 Byte/Thread の実装では，中間値はローカルメモリへの配置が必須となる．ただし，ローカルメモリのアクセスはバンクコンフリクトが発生するため，これらの計算粒度では，平文ブロックの中間値を AoS と SoA で配置する方法を考察した．なお，本節 (1) において，1 Bytes/Thread の並列実装による処理速度は，他の計算粒度より大きく劣ることが明らかとなったため，1 Byte/Thread については AoS と SoA の配置方法の考察を省略した．8 Bytes/Thread の計算粒度を用い，中間値を AoS でローカルメモリに配置した並列実装 (8Bs-I) では，処理速度は 21.3 Gbps であったのに対し，SoA で配置した (8Bs-III) では 18.5 Gbps であった．また，4 Bytes/Thread の計算粒

度を用い，中間値を AoS でローカルメモリに配置した並列実装 (4Bs-I) では，処理速度は 20.1 Gbps であったのに対し，SoA で配置した (4Bs-III) では 13.6 Gbps であった．プロファイラ [55] を用いてこれらの処理速度の差の原因を調べたところ，並列実装 (8Bs-I) と (8Bs-III) では，バンクコンフリクトの発生回数はほぼ同じ値であり，並列実装 (4Bs-I) では，バンクコンフリクトの発生回数は並列実装 (4Bs-III) よりも少なくなっていた．ただし，上述した通り，SoA による並列実装である (8Bs-III) と (4Bs-III) の処理速度は，それぞれ，AoS による並列実装 (8Bs-I) と (4Bs-I) よりも低い値となっている．SoA による並列実装では，中間値に対するアクセス時にワークアイテムをストライドさせるためのアドレス計算が必要であり，AoS よりもコード量が増加する [56] することを考慮すると，処理速度の低下には，SoA におけるアドレス計算が，バンクコンフリクトよりも大きく影響したものと考えられる．

4.6 関連研究との比較

[57] では，GPU への AES の実装方針を，計算粒度とメモリ配置という本章と似たアプローチで考察している．計算粒度については，粗粒度として 16 Bytes/Thread，細粒度として 4 Bytes/Thread を扱い，換字テーブルの配置についてはコンスタントメモリとローカルメモリを扱っている．一方，本研究では，計算粒度と各データのメモリ配置戦略にそれぞれ更なるパラメータを加え，かつ計算粒度とメモリ配置戦略が処理速度に与える影響を総合的に考察した．計算粒度として 8 Bytes/Thread と 1 Bytes/Thread も扱い，メモリ配置先についてはラウンド鍵と平文ブロックも扱った．更に，平文ブロックを複数のワークアイテムで協調して並列処理する計算粒度では，SoA と AoS という 2 種のローカルメモリへの配置方法による処理速度への影響も考察した．

4.7 第 4 章のまとめ

本章では，PPF の設計の簡単化に資する複数のブロック暗号に共通した GPU への実装方針を導出するため，計算粒度とメモリ配置戦略が処理速度に与える影響を，AES を題材として総合的に分析した．各ワークアイテムが平文ブロックを独立かつ並列に処理する計算粒度である，16 Bytes/Thread による並列実装は，他の計算粒度（8 Bytes/Thread，4 Bytes/Thread，1 Bytes/Thread）と比較して，高い処理速度を得られる傾向にあった．この理由は，16 Bytes/Thread では，ワークアイテム間で同期を取る必要がなく，また中間値をレジスタに直接保持し続けることができるためである．また，分析結果から，計算粒度とメモリ戦略の組み合わせが，処理速度に大きく影響を及ぼすことが分かった．特に，8 Bytes/Thread，4 Bytes/Thread，1 Bytes/Thread の計算粒度による並列実装では，ワークアイテム間の同期のために平文ブロックの中間値の格納場所にローカルメモリを利用する必要があるため，処理速度は 16 Bytes/Thread よりも低下しまうこととなった．

以上の分析結果から，GPU へのブロック暗号の実装において最も高い処理速度を得るためには，ワークアイテムによる並列処理の計算粒度をブロック長とし，ラウンド鍵と換字テーブルはローカルメモリに配置し，暗号化処理の中間値をレジスタに配置する実装方針が最適であることを導出できた．

第 5 章

GPU へ実装されたブロック暗号の処理速度の測定

5.1 まえがき

第 3 章で述べたように，PPF の設計及び実装評価には，SPN 構造，Feistel 構造，SPN と Feistel のハイブリッド構造という 3 種の代表的なブロック暗号の構造を設定している．また，これらの構造における暗号アルゴリズムの一例として，それぞれ AES，Camellia，SC2000 を用いている．そのため本章では，GPU へ実装されたこれらのブロック暗号に対して，処理速度の測定を行う．この測定値は，後述する**第 6 章**では，GPU へ実装されたブロック暗号の処理性能の実測値として扱われる．

5.2 測定対象としているブロック暗号のアルゴリズム

テーブルサイズが大きくなるようなテーブル構成法を用いることにより，平文ブロックの暗号化に要する換字テーブルへのアクセス回数を減らし，暗号化処理を高速化することができる．ただし，導出した GPU へのブロック暗号の実装方針から，換字テーブルとラウンド鍵はローカルメモリへ格納される．そのため，ラウンド鍵と換字テーブルの合計サイズは，ローカルメモリの容量に収める必要がある．ソフトウェア実装に最適化された AES と Camellia では，換字テーブルとラウンド鍵の合計サイズは 16 KB 以下となるので，これらのデータは全てローカルメモリに配置できる．一方，SC2000 では，換字テーブルの構成法は，

第 3.1.4 節で述べた数種類の候補から選択できる．Geforce GTX 580 と Geforce GTX 280 では，ローカルメモリの容量はそれぞれ 48 KB，16 KB である．そのため，Geforce GTX 580 と Geforce GTX 280 における SC2000 の実装ではそれぞれ，20 KB のテーブルサイズとなる（11-bit, 10-bit, 11-bit）と 8.5 KB のテーブルサイズとなる（6-bit, 10-bit, 10-bit, 6-bit）のテーブル構成法を用いた．本章では，**第 4 章**で示した実装方針で，上記のような 3 種類のブロック暗号のアルゴリズムをコーディングし，それらプログラムの処理速度を測定した．

5.3 測定環境

処理速度の測定には，**表 4.4** の環境を再び用いた．処理速度の測定に用いた GPU には，GPU へのブロック暗号の実装方針を導出する過程で用いた GT200 アーキテクチャの Geforce GTX 280 に加え，Fermi アーキテクチャである Geforce GTX 580 を選択した．Geforce GTX 580 の仕様を**表 5.1** に示す．Fermi は，GT200 の後継となるアーキテクチャであり，**第 2.3 節**で述べた GPU アーキテクチャの特徴は継承しつつ，CU に内蔵されたローカルメモリの容量の増加，CU とグローバルメモリ間のキャッシュの追加，CU 当たりの PE 数の増加，1 クロックサイクル当たり 2 つのバッチに対する命令の並列発行等の機能強化が図られている [43]．**第 4 章**で導出した実装方針を，世代の異なるアーキテクチャの GPU 実装及び評価することにより，その有効性を確認する．これら 2 種の GPU は，**表 4.4** の測定環境へ相互に装着し，単一 GPU としての処理速度を測定した．

なお，高い暗号化処理速度を引き出すためには，**第 2.2 節**で述べたように，ワークグループ数は CU の倍数とし，ワークグループ当たりのワークアイテム数は CU 当たりの PE 数の倍数とするのが望ましい．そのため，Geforce GTX 580 におけるワークグループ数とワークグループ当たりのワークアイテム数については，**第 4.5.1 節**で述べた Geforce GTX 280 と同様の要領でチューニングを試みた結果，それぞれ 64，1024（32 バッチ）に設定することとした．Geforce GTX 280 にお

けるワークグループ数とワークグループ当たりのワークアイテム数については，**第 4.5.1 節**と同様の理由から，それぞれ 90，512（16 バッチ）に設定することとした．

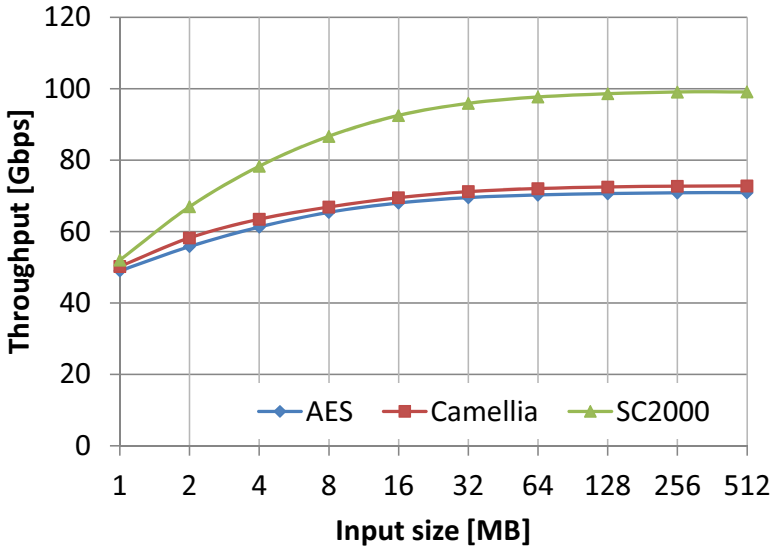
表 5.1　測定に追加する Geforce GTX 580 の仕様

Name	Geforce GTX 580
Architecture	Fermi
Core clock frequency	1.544 GHz
# CUs	16
# PEs per CU	32
Local memory capacity per CU	48 KB
# local memory banks	32
Maximum # work-item batches per CU	32

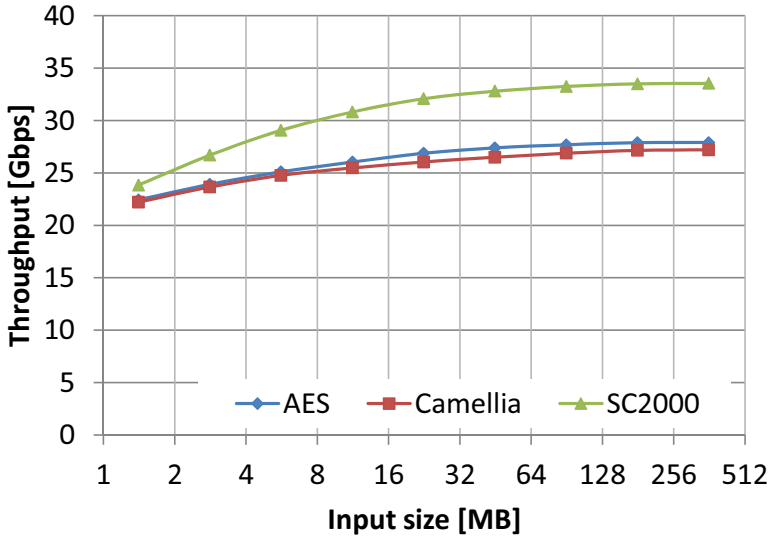
5.4　GPU を用いた暗号化処理速度の測定結果

平文サイズの変化に応じた 3 種類の 128-bit ブロック暗号（AES，Camellia，SC2000）の処理速度を**図 5.1** に示す．同図 (a) と (b) で処理速度の測定対象とした平文のサイズが一致していないのは，Geforce GTX 580 と Geforce GTX 280 では最適なワークグループ数とワークグループ当たりのワークアイテム数が異なるため，最も高い処理速度を引き出せる平文サイズもそれぞれの GPU で異なるからである．また，処理速度は，収束したときに評価すべき値である．同図によれば，処理速度は平文サイズの増加に従って増加し，256 MB 付近で収束した．従って，GPU の処理速度を十分に引き出すには，256 MB 以上の平文が必要であることが分かる．また，これらのグラフにおける 3 種類の暗号の処理速度の最大値を**図 5.2** にまとめる．同図 (a)，(b) における AES，Camellia，SC2000 の結果を比較すると，いずれの GPU でも SC2000 が最も高い値となっていた．なお，SC2000 の処理速度は，Geforce GTX 280 では AES と Camellia の約 1.2 倍の処理

速度であるのに対し，Geforce GTX 580 では約 1.4 倍の処理速度となった．これは，それぞれの GPU のローカルメモリ容量を最大限に利用できるテーブル構成法のアルゴリズムをそれぞれ使用しており，テーブルアクセスの回数が異なるためである．



(a) Geforce GTX 580



(b) Geforce GTX 280

図 5.1　GPU へ実測した 3 種のブロック暗号の暗号化処理速度の計測結果

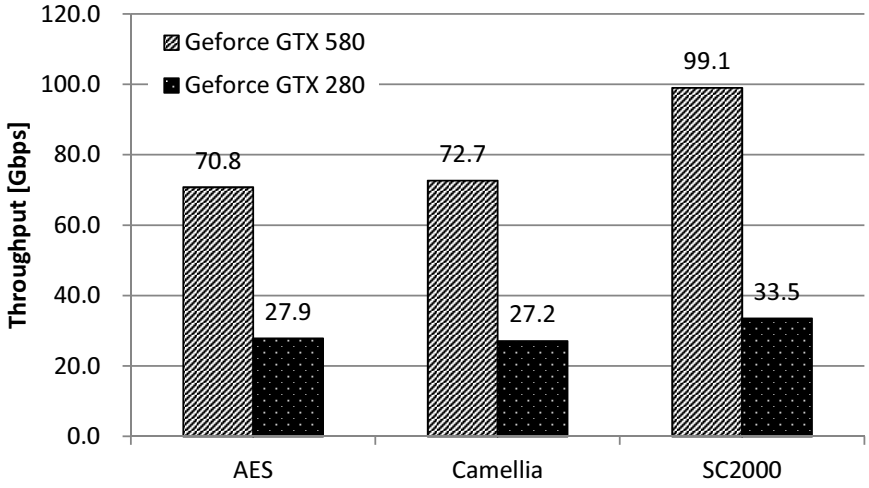


図 5.2　GPU へ実装した 3 種のブロック暗号の暗号化処理速度の最大値比較

5.5　GPU を用いた暗号化処理速度の比較

GPU を利用する目的は，CPU の計算負荷を分散して，全体の処理を高速化することにある．そのため，GPU を用いた暗号化処理が，どの程度マルチコア CPU より高速化されているかを確認しておく必要がある．また，これまでに，AES をハードウェアで高速実行する拡張命令セットである AES-NI[13] が開発されており，一部の CPU では既に利用できる．このため本節では，マルチコア CPU 及び一部の CPU で利用できる AES-NI 命令で暗号化処理速度を測定して比較した結果を述べる．

5.5.1　マルチコア CPU との比較

測定に用いた**表 4.4** の環境には，4 コアの CPU である Intel Core i7-2600K が搭載されている．また，この CPU には，1 つのコアで複数のスレッドを多重実行することにより各コアのレジスタやパイプラインの空きリソースを有効利用する技術であるハイパースレッディングテクノロジー [58] が導入されている．そのため，マルチコア CPU による暗号化処理では，1 スレッド，4 スレッド，8 スレッドという 3 種の並列実装による AES，Camellia，SC2000 の暗号化処理速度を計

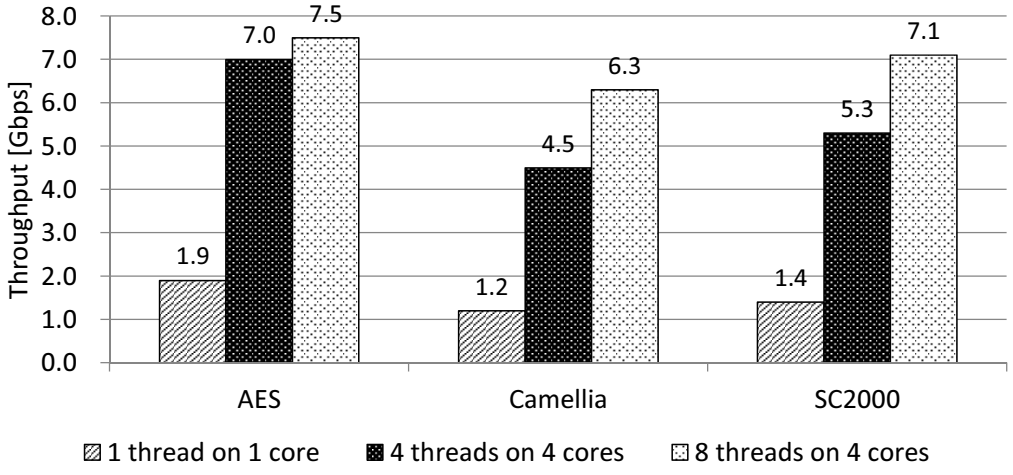


図 5.3　Intel Core i7-2600K CPU における暗号化処理速度

測した．秘密鍵の鍵長は 128-bit とし，平文サイズは GPU での測定時と同じ 256 MB とした．用いた暗号利用モードは ECB である．なお，並列実装には，マルチコアプロセッサにおける並列処理の API 仕様である OpenMP[59] を用いた．実装には，各スレッドが 128-bit ブロック暗号のブロック長である 16 Bytes ごとに並列処理する計算粒度を用いた．

計測結果を**図 5.3** に示す．マルチコア CPU を用いた並列実装による AES，Camellia，SC2000 の暗号化処理速度は，いずれも 8 スレッドでの並列処理で最も高い値が得られ，このときの処理速度はそれぞれ 7.5 Gbps，6.3 Gbps，7.1 Gbps であった．なお，4 の倍数かつ 8 以上のスレッド数で並列処理した場合の処理速度は，8 スレッドの計測時と概ね同じ値に収束することを確認している．一方，**図 5.2** に示した Geforce GTX 580 における AES，Camellia，SC2000 の処理速度は，これらマルチコア CPU の処理速度のそれぞれ 9.4 倍，11.5 倍，14.0 倍の値となっていた．この結果から，GPU を利用することにより暗号化処理速度を大きく向上できることが分かる．**第 4 章**では，GPU へのブロック暗号の実装方針は AES を題材として導出していたが，本節の比較結果から，導出した実装方針が他のブロック暗号に対しても有効であることを，併せて確認できた．

5.5.2 AES-NI 命令との比較

マルチコア CPU による暗号化処理速度の測定に用いた**表 4.4** の環境では，CPU やマザーボードの仕様が AES-NI 命令に対応しており，同命令も利用することができる．そのため，AES-NI 命令による AES 暗号化処理速度も併せて計測した．なお，暗号通信プロトコルのオープンソースの実装として OpenSSL[60] が知られている．この OpenSSL は既に AES-NI 命令に対応しており，ライブラリには同命令のコードが含まれている．そのため，AES-NI による暗号化処理速度の計測については，OpenSSL のライブラリから同命令の関数部分を抽出して実装したコードを対象とすることにした．また，AES-NI 命令は，OpenMP と併用することも可能である [61]．従って，並列実装には OpenMP を用い，各スレッドに AES-NI 命令を実行させた．秘密鍵の鍵長はここでも同様に 128-bit とし，平文サイズは GPU での測定時と同じ 256 MB とした．用いた暗号利用モードは ECB である．

計測結果を**図 5.4** に示す．同図には比較のため，Geforce GTX 580 と Geforce GTX 280 を使用した場合の暗号化処理速度を併せて示している．4 スレッドで AES-NI を並列実行したときに最も高い処理速度が得られ，このときのスループットは 44.2 Gbps であった．この値は，**図 5.2** で示した Geforce GTX 580 と Geforce GTX 280 の暗号化処理速度のそれぞれ 1.6 倍，0.6 倍であり，GPU に匹敵する処理速度を得られていることが分かる．しかし，これまで述べてきたように，実利用されている暗号は AES だけではない．GPU を利用した暗号化処理では，AES-NI 命令と同等のレベルの処理速度向上が，複数のブロック暗号に対して得られる．

5.5.3 GPU への 128-bit ブロック暗号の実装に関する研究との比較

GPU への 128-bit ブロック暗号の実装に関する既存成果を**表 5.2** にまとめる．同表に示した暗号化処理速度は，全て CPU と GPU 間のデータ転送を無視した値を示している．

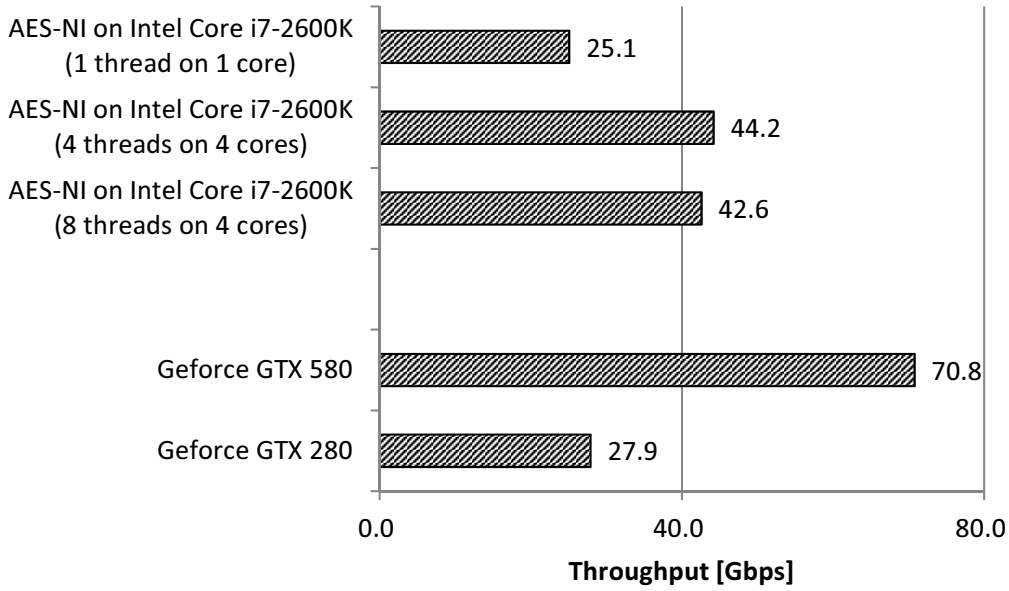


図 5.4　AES-NI と GPU による AES-128 暗号化処理速度の比較

(1)　GPU へ実装されたブロック暗号の処理速度

これまで，GPU への暗号実装に関する研究の多くでは，主に暗号化処理速度の評価が論じられてきた．最初の GPU への暗号実装として，2005 年に，グラフィクス処理用のプラットフォームである OpenGL[62] を用いて，AES を NVIDIA Geforce 3 Ti200 へ実装する研究が試みられた [63]．しかし，その処理速度は 1.5 Mbps に過ぎず，CPU よりも劣る結果となった．[63] の著者は，この原因を，当時の GPU アーキテクチャがグラフィクス処理に特化しており，暗号処理に必要な論理演算等がグラフィクス API で疑似的に実現されているためと結論付けた．現在では，算術論理演算のサポートも含めた形で GPU アーキテクチャが進化し，暗号化処理の処理速度は数十 Gbps のオーダーまで大幅に向上した [57],[64],[65]．特に，本研究で計測した，Geforce GTX 580 における暗号化処理の処理速度は約 50 Gbps にまで達し，既存成果 [65] からは 1.6〜1.9 倍の性能向上となっている．なお，GPU を用いたブロック暗号の実装に関する研究の対象はそのほとんどが AES であり，特に SC2000 の実装研究は報告されていない．一方，本研究で

は，AES の他，2 種のブロック暗号の GPU への実装処理速度の計測結果を示している．

(2)　GPU へ実装されたブロック暗号の考察

GPU へのブロック暗号の実装と処理速度の関係を分析した先行研究には，計算粒度とメモリ配置が処理速度に与える影響を考察した [57] や，スレッド数が暗号化処理速度に与える影響等を考察した [66] がある．ただし，これらは AES の実装方針のみを考察対象としたものであり，複数のブロック暗号を意図した研究ではない．複数のブロック暗号に共通した実装方針を開発するには，本研究のように，実装方針の導出から複数のブロック暗号の実装評価に至るまでの過程を，体系的に実施する必要がある．

表 5.2　GPU を用いた暗号処理の関連研究

Algorithm	Reseachers	Year	Language	GPU	Throughput [Gbps]	Research purposes		Paper
						Throughput evaluation	Implementation method	
AES (128-bit key)	Cook et al.	2005	OpenGL	Geforce 3 Ti200	0.002	○	○	[63]
	Harrison et al.	2007	DirectX	Geforce 7900 GT	0.9	○	×	[67]
	Manavski et al.	2007	CUDA	Geforce 8800 GTX	8.3	○	×	[68]
	Liu et al.	2009	CUDA	Geforce 9800 GTX	4.2	×	○	[66]
	Biagio et al.	2009	CUDA	Geforce 8800 GT	12.5	○	○	[57]
	Oikawa et al.	2010	OpenCL	Geforce GTX 280	23.0	○	×	[64]
	Gilger et al.	2012	CUDA	Geforce GTX 295 注 1)	29.6	○	×	[65]
	Gilger et al.	2012	OpenCL	Geforce GTX 295 注 1)	25.4	○	×	[65]
	Nishikawa	**2012**	**CUDA**	**Geforce GTX 580**	**48.6**	○	○	–
Camellia (128-bit key)	Oikawa et al.	2010	OpenCL	Geforce GTX 280	23.0	○	×	[64]
	Gilger et al.	2012	CUDA	Geforce GTX 295 注 1)	26.3	○	×	[65]
	Gilger et al.	2012	OpenCL	Geforce GTX 295 注 1)	26.3	○	×	[65]
	Nishikawa	**2012**	**CUDA**	**Geforce GTX 580**	**50.7**	○	○	–

注 1) デュアルチップの GPU であるが，評価にはこのうち 1 チップのみを使用している．

5.6 第 5 章のまとめ

本章では，**第 4 章**で示した GPU へのブロック暗号の実装方針を用いて，SPN 構造，Feistel 構造，SPN と Feistel のハイブリッド構造という 3 種の代表的な構造から成るブロック暗号を GPU へ実装し，それらの処理速度を計測した結果について述べた．暗号アルゴリズムには，これらの構造の例として，SPN 構造に AES，Feistel 構造に Camellia，SPN と Feistel のハイブリッド構造に SC2000 を選択した．また，GPU には，異なる世代のアーキテクチャで開発された 2 種の GPU である Geforce GTX 580 と Geforce GTX 280 をそれぞれ交互に用いた．また，**第 4 章**では，GPU へのブロック暗号の実装方針は AES を題材として導出したが，本章では同方針を他のブロック暗号にも適用することにより，その有効性も併せて確認された．

第 6 章

PPF の設計と評価

6.1 まえがき

第 4 章及び**第 5 章**では，関連研究との比較を容易にするため，暗号化処理の並列実装性能の計測結果を処理速度で表していた．一方，本章では，暗号化処理の性能値を処理時間で表し，その単位は GPU のコアクロックのサイクル数とする．これは，本章で述べる PPF の設計が，暗号アルゴリズムで多用される算術論理命令等のレイテンシを事前に計測することを前提としているためである．つまり，これらの命令のレイテンシの計測結果は，通常，クロックサイクル数で表されているので，暗号化処理の性能値もクロックサイクル数で表すことで，見通し良く PPF を設計できる．

例えば，x [MB]$(= 8 \times 10^6 \times x$ [bit]$)$ の平文サイズにおける暗号化処理のスループット t [Gbps]$(= 10^9 \times t$ [bps]$)$ は，GPU のコアクロック周波数 f [GHz]$(= 10^9 \times f$ [Hz]$)$ を用いて，以下の式により暗号化処理時間 c [cycle] に換算できる．

$$\begin{aligned} c &= \frac{8 \times 10^6 \times x}{10^9 \times t} \times 10^9 \times f \\ &= \frac{8 \times 10^6 \times x \times f}{t} [\text{cycle}] \end{aligned} \tag{6.1}$$

6.2 ブロック暗号の GPU への実装方針

本節では**第 4 章**で述べたブロック暗号の GPU への実装方針をまとめ，PPF 設計のための要点を整理する．

6.2.1 実装方針のまとめ

暗号化カーネルの起動前に，ラウンド鍵はホスト側で秘密鍵から生成し，平文，換字テーブルとともにグローバルメモリに転送しておく．

暗号化カーネルが起動すると，ワークアイテムは暗号化を開始する前に，まず換字テーブルとラウンド鍵を CU 内のローカルメモリへ並列に展開する．ただし，この展開は，平文サイズに関わらず 1 回のみ行われる軽量な処理である．この展開の後，ワークアイテムは，各平文ブロックを並列に暗号化する．この暗号化は，ワークアイテムが平文ブロックをレジスタに保持したまま，換字テーブルやラウンド鍵へのアクセスや算術論理演算を実行する処理である．暗号化完了後，ワークアイテムは暗号文をグローバルメモリへストアする．この暗号化の完了後も，ワークアイテムは処理を終了せず，平文ブロックのポインタにストライド値を加算して，引き続き別の平文ブロックの暗号化を行う．なお，**第 2 節**で示した理由から，暗号化処理で実行される算術論理演算やメモリアクセス命令は主に 32-bit 命令である．

6.2.2 PPF 設計のための実装方針の要点

第 6.2.1 節に示した暗号化カーネルの内の処理の流れをまとめると，ブロック暗号のカーネルは，

- ローカルメモリへの換字テーブルとラウンド鍵の展開
- 各ワークアイテムによる暗号化処理のループによる反復実行

という 2 つの段階から成る．特に 2 つ目の段階である暗号化処理は，主に，

- 換字テーブルへのアクセス
- ラウンド鍵へのアクセス
- 算術論理演算
- 平文ブロックのロードまたは暗号文のストア

という 4 種の要素で構成される．

6.3 PPF の設計

以上の背景を基に PPF の設計を行う．また，PPF で使用する変数を**表 6.1** にまとめる．通常，暗号設計者は，設計した暗号アルゴリズムを実際にコーディングし，処理速度を実測することでそのアルゴリズムの優劣を判断する [8]．このコーディングは，暗号アルゴリズムをプログラムに翻訳する作業であるため，最も基本的な逐次処理で記述する．このため，暗号設計者の手元には，逐次処理で記述されたブロック暗号のコードがあるという状況を前提とし，以下，PPF の設計要領を説明していく．

提案する PPF のワークフローを**図 6.1** に示す．まず，ブロック暗号の逐次処理のコードを PPF に入力する．次に，入力されたブロック暗号の逐次処理のコード内にある暗号化関数を解析し，**第 6.2.2 節**に示した 4 種の要素の数（N_{tbl}，N_{key}，N_{inst}，N_{pt} または N_{ct}）を取得する．取得したこれらの値は，予測式に対する入力パラメータとなる．また，予測式の入力パラメータには，これら 4 種類の要素を GPU で処理した場合にかかるバッチ当たりのレイテンシ（$L_{local\_rand\_bt}$，$L_{local\_rgl\_bt}$，L_{inst_bt}，L_{glb_bt}）も必要となっている．これらの値は，事前に作成しておいたデータベースから取得する．データベースの生成要領については**第 6.4 節**で述べる．最終的に PPF は，予測式から，ブロック暗号を GPU へ実装したと想定した場合の処理時間の上限値及び下限値を出力することができる．

なお，ホスト側のメモリと GPU のグローバルメモリの間のデータ転送時間は，ホストと GPU 間のメモリバンド幅から容易に算出できる．従って，PPF の予測対象は，このデータ転送時間を含めず，実装方針で実装されたブロック暗号の暗号化カーネルに対する開始から終了までの処理時間とする．

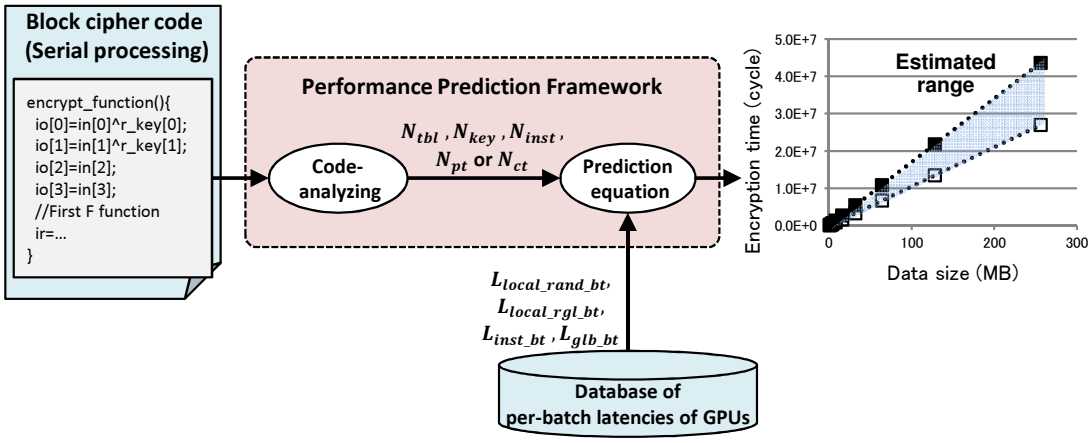


図 6.1　提案する PPF のワークフロー

6.3.1　コード解析

コード解析では，N_{tbl}，N_{key}，N_{inst}，N_{pt} または N_{ct} を取得するために，暗号の逐次処理のコードに現れる演算子を数え上げる方法を用いる．コード解析の説明のため，逐次処理コードの例として 128-bit ブロック暗号である Camellia を**図 6.2** に示す．**図 6.2** の 10〜18 行目は Camellia の F 関数を示しているが，この範囲に換字テーブルのポインタである `tbl_sp1110`，`tbl_sp0222`，`tbl_sp3033`，`tbl_4404` と，ラウンド鍵のポインタである `rkey` がそれぞれ 2 回ずつ現れるため，F 関数内では $N_{tbl} = 8$，$N_{key} = 2$ となる．また，AND，XOR，シフト演算，加算の演算子がそれぞれ 8，12，8，1 回現れるので，$N_{inst} = 29$ となる．N_{inst} の導出を簡単にするため，配列変数の使用で発生するアドレス演算は N_{inst} に含めないものとする．

6.3.2　予測式

PPF は，暗号化カーネルを呼び出し，GPU での暗号化処理が完了するまでの処理時間 L_{enc_total} を予測する．各ワークアイテムは複数の平文ブロックを一度に暗号化するため，大きなサイズの平文であっても暗号化カーネルは 1 回だけ呼び出されればよい．カーネルを呼び出すと，まずカーネルが起動し，次に換字テー

表 6.1　PPF で使用される変数

変数	説明
N_{tbl}	逐次処理コード内で換字テーブルにアクセスする回数
N_{key}	逐次処理コード内でラウンド鍵にアクセスする回数
N_{inst}	逐次処理コード内で算術論理演算を実行する回数
N_{pt}	逐次処理コード内でメモリからレジスタへ平文のサブブロックをロードする回数 （平文のブロック長に依存）
N_{ct}	逐次処理コード内でレジスタからメモリへ暗号文のサブブロックをストアする回数 （暗号文のブロック長に依存）
L_{ker}	カーネル起動にかかるレイテンシ
L_{ld}	ローカルメモリへの換字テーブルとラウンド鍵の展開にかかるレイテンシ
L_{enc}	暗号化カーネルで反復処理による暗号化にかかる時間
L_{enc_total}	暗号化カーネルの処理時間
$N_{batchsize}$	バッチ当たりのワークアイテム数
L_{enc_bt}	1 バッチによる暗号化の処理時間
N_{iter}	各ワークアイテムによる暗号化の反復回数
N_{batch}	ワークグループ中のワークアイテムバッチ数
N_{wg}	プログラマにより指定されたワークグループ数
$N_{wi\_per\_wg}$	プログラマにより指定されたワークグループ当たりのワークアイテム数
N_{cu}	GPU のコンピュートユニット数
$L_{local\_rand\_bt}$	ローカルメモリへのバッチ当たりのランダムアクセスレイテンシ （換字テーブルのアクセスにかかるレイテンシとなる）
$L_{local\_rgl\_bt}$	ローカルメモリへのバッチ当たりのレギュラーアクセスレイテンシ （ラウンド鍵のアクセスにかかるレイテンシとなる）
L_{inst_bt}	バッチ当たりの算術論理演算のレイテンシ （算術論理演算にかかるレイテンシとなる）
L_{glb_bt}	グローバルメモリへのバッチ当たりのアクセスレイテンシ （平文のサブブロックのロードまたは暗号文のサブブロックのストアのレイテンシとなる）

ブルとラウンド鍵がローカルメモリへ展開され，暗号化処理が実行されていくため，それぞれの段階にかかる処理時間を L_{ker}，L_{ld}，L_{enc} とすると，L_{enc_total} は，

$$L_{enc\_total} = L_{ker} + L_{ld} + L_{enc} \tag{6.2}$$

のように表される．

次に，L_{enc} の計算過程を示す．まずはじめに，暗号化カーネルの実行時にワー

```
 1 void camellia_encrypt128(...){
 2   uint io[4];
 3   uint il, ir, t0, t1;
 4   // Start Camellia encryption
 5   io[0]=plaintext[0]^rkey[0];
 6   io[1]=plaintext[1]^rkey[1];
 7   io[2]=plaintext[2];
 8   io[3]=plaintext[3];
 9   // First F function
10   ir=tbl_sp1110[io[1]&0xff]^tbl_sp0222[(io[1]>>24)&0xff]^tbl_sp3033[(io
    [1]>>16)&0xff]^tbl_sp4404[(io[1]>>8)&0xff];
11   il=tbl_sp1110[(io[0]>>24)&0xff]^tbl_sp0222[(io[0]>>16)&0xff]^tbl_sp3033[(io
    [0]>>8)&0xff]^tbl_sp4404[io[0]&0xff];
12   il^=rkey[2];
13   ir^=rkey[3];
14   ir^=il;
15   il=(il>>8)+(il<<24);
16   il^=ir;
17   io[2]^=ir;
18   io[3]^=il;
19   // Second to sixth F functions...
20   // FL and FLinv functions
21   t0=io[0]&rkey[14];
22   io[1]^=(t0<<1)+(t0>>31);
23   t1=io[1]|rkey[15];
24   io[0]^=t1;
25   il=io[3]|rkey[17];
26   io[2]^=il;
27   ir=io[2]&rkey[16];
28   io[3]^=(ir<<1)+(ir>>31);
29   // Seventh F function...
30   ...
31 }
```

図 6.2　Camellia の逐次処理コード

クグループ当たりのワークアイテム数 ($N_{wi\_per\_wg}$) とワークグループ数 (N_{wg}) を指定する．なお，バッチを構成するワークアイテム数 ($N_{batchsize}$) は各 GPU で固有

の値であるため，ワークグループ当たりのバッチ数 (N_{batch}) は $N_{batch} = \frac{N_{wi\_per\_wg}}{N_{batchsize}}$ と決定される．L_{enc} は N_{wg} 及び N_{batch} に加え，反復実行による暗号化処理内での 1 ループによるバッチ当たりの暗号化に要する処理時間 (L_{enc_bt})，各 GPU 内の CU 数 (N_{cu})，プログラム中の暗号化関数の反復回数 (N_{iter}) を用いて，

$$L_{enc} = L_{enc\_bt} \times N_{iter} \times N_{batch} \times \frac{N_{wg}}{N_{cu}} \tag{6.3}$$

と表される．ここで $\frac{N_{wg}}{N_{cu}}$ は，GPU のスケジューラによって CU へ均等に分配されるワークグループ数を表している．式 (6.3) の導出の根拠は，第 2 章に示した特徴 (i) 及び (ii) である．即ち，GPU アーキテクチャでは命令がバッチ単位で発行されるだけでなく，各 CU 内での一定数以上のバッチ起動により PE のパイプラインはビジーとなり，バッチ当たりの各命令のレイテンシが一定値になるためである [69]．なお，暗号化の反復実行時に平文ブロックのポインタにストライド値が加算されるが，その処理時間は全体と比較して無視できるので，L_{enc} の計算に含めない．

次に，L_{enc_bt} の計算過程を示す．GPU へ実装されたブロック暗号は，**第 6.2.2 節**で述べた 4 種の要素で構成されるため，L_{enc_bt} はこれらの要素におけるバッチ当たりのレイテンシ ($L_{local\_rand\_bt}$，$L_{local\_rgl\_bt}$，L_{inst_bt}，L_{glb_bt}) と回数 (N_{tbl}，N_{key}，N_{inst}，N_{pt} または N_{ct}) の積和を取ることによって算出できる．しかし，**第 2 節**で示した GPU の特徴 (iv) である，算術論理演算のレイテンシを隠蔽するアウトオブオーダースケジューリング機能は，暗号化の処理時間の予測を複雑にする．そのため L_{enc_bt} は，

- Scheduling-Aware (SA) サブモデル： 全ての算術論理演算のレイテンシが，メモリアクセス命令のレイテンシに完全に隠蔽されると仮定．暗号化処理時間の下限が得られる．

- Scheduling-Ignored (SI) サブモデル: 算術論理演算のレイテンシが，メモリアクセス命令のレイテンシに全く隠蔽されないと仮定．暗号化処理時間の

上限が得られる．

という 2 つのサブモデルで表す．従って，L_{enc_bt} は最終的に，

$$L_{enc\_bt} = \begin{cases} L_{glb\_bt} \times (N_{pt} + N_{ct}) + L_{local\_rand\_bt} \times N_{tbl} + L_{local\_rgl\_bt} \times N_{key} \\ \qquad\qquad \text{(SA サブモデル)} \\ L_{glb\_bt} \times (N_{pt} + N_{ct}) + L_{local\_rand\_bt} \times N_{tbl} + L_{local\_rgl\_bt} \times N_{key} + L_{inst\_bt} \times N_{inst} \\ \qquad\qquad \text{(SI サブモデル)} \end{cases} \tag{6.4}$$

と表される．このようにして求めた L_{end_bt} から，式 (6.2)，(6.3) を用いて暗号化カーネルの実行時間の予測値である L_{enc_total} を算出する．従って L_{enc_total} は，SI サブモデルと SA サブモデルから算出した値を上限値及び下限値とした範囲として得られる．なお，PPF では，式 (6.4) に示したように，予測処理時間の算出には，算術論理演算を含んだ SA サブモデルが用いられている．このため PPF は，例えば公開鍵暗号のように，算術論理演算のみから構成され，換字テーブルを用いない暗号アルゴリズムにも拡張性を有している．

6.4 GPU のデータベースの生成

式 (6.4) の計算過程では，GPU の 4 種のバッチ当たりのレイテンシ（$L_{local\_rand\_bt}$，$L_{local\_rgl\_bt}$，L_{glb_bt}，L_{inst_bt}）が必要となる．これらの値は，OpenCL で記述されたマイクロベンチマークを用いて事前に測定し，データベースを作成する．

6.4.1 マイクロベンチマークの設計

設計した各命令のマイクロベンチマークを**図 6.3** に示す．

(1) 算術論理演算用のマイクロベンチマーク

L_{inst_bt} を測定するマイクロベンチマークは，依存関係のある一連の算術論理演算を反復実行するカーネルとして設計した．なお，この反復処理では，条件分岐

```
1  // Kernel for instructions (XOR)
2  __kernel void inst_latency(__global uint out,uint p1,uint p2,int its){
3    uint lid=get_local_id(0);
4    uint a=p1,b=p2;
5    for(int i=0;i<its;i++) repeat256(a^=b;b^=a;)
6    out[lid]=a+b;
7  }
8  // Kernel for local memory latency (Random)
9  // "aes_sbox" is an array with S-Box of AES.
10 __kernel void local_latency(__global uint aes_s_box,__global uint out,int
   its){
11   uint lid=get_local_id(0);
12   __local l_data[256];
13   if(lid==0){for(i=0;i<256;i++) l_array[i]=aes_s_box[i];}
14   barrier(CLK_LOCAL_MEM_FENCE);
15   uint j=lid%256; // If j=0, the access pattern becomes regular.
16   for(int i=0;i<its;i++) repeat256(j=l_array[j];)
17   out[lid]=j;
18 }
19 // Kernel for global memory latency
20 // "data" is an array with precomputed data in global memory.
21 __kernel void global_latency(__global uint data,__global uint out,int its){
22   uint gid=get_global_id(0);
23   // In the case of 128-bit encryption
24   uint j=4*gid,k=4*gid+1,l=4*gid+2,m=4*gid+3;
25   for(int i=0;i<its;i++) repeat256(j=data[j];k=data[k];l=data[l];m=data[m];)
26   out[gid]=j+k+l+m;
27 }
```

図 6.3　各命令のマイクロベンチマーク

のオーバーヘッドを回避して純粋な算術論理演算のレイテンシのみを計測するため，for 文等のループ処理を使用せず，マクロとして展開している．XOR を例としたコードを**図 6.3** の 2〜7 行目に示す．他の算術論理演算の測定については 5 行目の演算子を置き換えるだけでよい．CU をビジーとしたときのバッチ当たりの算術論理演算のレイテンシを測定するため，ワークグループ数には 1 を設定し，

ワークグループ当たりのバッチ数は各 GPU で起動可能な最大数を設定した．

(2)　メモリアクセス用のマイクロベンチマーク

$L_{local\_rand\_bt}$，$L_{local\_rgl\_bt}$，L_{glb_bt} を測定するマイクロベンチマークは，ローカルメモリまたはグローバルメモリに配置した配列から依存関係のある一連のロード命令を反復実行するカーネルとして**図 6.3** の 10〜18 行目及び 21〜27 行目のように設計した．

ローカルメモリ用のマイクロベンチマーク

GPU へ実装されたブロック暗号のコードでは，各ワークアイテムはローカルメモリに格納された換字テーブルとラウンド鍵へ並列にアクセスする．それぞれのデータの性質からアクセスパターンは換字テーブルとラウンド鍵で異なり，それぞれランダムアクセス及びレギュラーアクセスとなる．ローカルメモリはマルチバンク型のメモリなので，バッチ当たりのレイテンシはワークアイテムによるアクセスパターンで大きく左右される．従って，バッチ当たりの換字テーブルとラウンド鍵のアクセスレイテンシである $L_{local\_rand\_bt}$ と $L_{local\_rgl\_bt}$ は区別して測定する必要がある．

$L_{local\_rand\_bt}$ を測定するには，重複した入出力関係ではなく，ランダムアクセスを満たしている必要がある．AES で採用される S-Box は，8-bit の 1 対 1 関数で，かつ乱数性を有することから今回の測定の要件を満たす．従って，$L_{local\_rand\_bt}$ を測定するコードでは，マイクロベンチマークの配列に AES の S-Box の値を利用した．なお，$L_{local\_rgl\_bt}$ の測定では，**図 6.3** の 15 行目にある `j=lid%256` を `j=0` に置き換えるだけでレギュラーアクセスが実現できる．$L_{local\_rand\_bt}$ と $L_{local\_rgl\_bt}$ の測定では，L_{inst_bt} と同じ理由から，ワークグループ数には 1 を設定し，ワークグループ当たりのバッチ数は各 GPU で起動可能な最大数を設定した．

グローバルメモリ用のマイクロベンチマーク

各 CU のレジスタ長は 32-bit なので，バッチは 32-bit ブロックへのアクセスを連続実行して，平文ブロックをグローバルメモリからロードする．従って L_{glb_bt}

を測定するマイクロベンチマークでは，平文ブロックサイズ × ワークアイテム数のストライド幅を持つグローバルメモリからのロード命令を連続かつ反復実行するカーネルとして，**図 6.3** の 21〜27 行目のように設計した．

また，グローバルメモリは全ての CU に接続されているため，L_{glb_bt} はメモリアクセス要求が全ての CU から同時に発生したときの値として測定する必要がある．そのため，L_{glb_bt} のマイクロベンチマークでは，$L_{local\_rand\_bt}$ や $L_{local\_rgl\_bt}$ と異なり，ワークグループ数は各 GPU の CU 数に設定し，ワークグループ当たりのバッチ数は各 GPU で起動可能な最大数を設定した．

6.4.2 データベースの測定値算出の条件

OpenCL では，命令レイテンシをカーネル内で直接測定できる CUDA 環境の `clock()` のような関数は用意されていない．そのため命令レイテンシには，`clGetEventProfilingInfo` API を用いてマイクロベンチマークと一連の命令を除いた空のカーネルの実行時間を取得し，その差分に GPU のコア周波数を掛けた値を採用した．なお，マイクロベンチマークは 2 回実行し 1 回目の結果を捨てることで，命令キャッシュのコールドミスを回避するように留意した．

データベースの生成には，**表 4.4** の測定環境を再び用いた．また，測定に用いる GPU にも，Geforce GTX 580 と Geforce GTX 280 を再び選択した．

6.4.3 データベースの生成結果

(1) 算術論理演算 (L_{inst_bt})

各 GPU の CU で保持できるバッチを最大数起動し（Geforce GTX 580 と GTX 280 ではそれぞれ 32 及び 16 バッチ），各 GPU の算術論理演算のスループットとバッチ当たりのレイテンシを測定した結果を**表 6.2** にまとめる．同表中において，ADD，SUB，MUL，MAD は，加算，減算，乗算，積和命令をそれぞれ表す．また，XOR，AND，OR，SHL，SHR は，排他的論理和，論理積，論理和，左シフト，右シフト命令をそれぞれ表す．**表 6.2** にはマイクロベンチマークの正当性を示すための補足データとしてスループットの測定結果も示してあるが，データ

ベースにはバッチ当たりのレイテンシのみが格納される．

測定結果から，算術論理演算のスループットはSHL及びSHR，MUL，MAD演算を除き，Geforce GTX 580とGeforce GTX 280でそれぞれ31.1 operation/cycle及び7.5 operation/cycleとなった．これらの値は各GPUのCU内のPE数とほぼ一致しており，ブロック暗号で多用されるADDやXOR等の算術論理演算は，各GPUでのPEパイプラインで効果的に処理されていることを確認できる．なお，**表 6.2**に示した演算のスループットは，CUDA言語でのマイクロベンチマークによる[69],[70]の既存研究の結果と概ね一致している．そのため，本研究で設計した**図 6.3**のマイクロベンチマークは，既存研究と同等の計測能力を有していることが分かる．

本論文では，PPFの予測式を簡略化するため，Geforce GTX 580とGeforce GTX 280におけるL_{inst_bt}は，**表 6.2**に示した結果からそれぞれ1.0 cycle/batch及び4.3 cycle/batchという一定値で扱うこととした．

表 6.2　各 GPU における算術論理演算のスループットとワークアイテムバッチ当たりのレイテンシの測定結果

GPU		Geforce GTX 580 (32 PEs per CU and 32 batches per CU available)		Geforce GTX 280 (8 PEs per CU and 16 batches per CU available)	
		Throughput (operation/cycle)	**Latency per batch (cycle/batch)**	Throughput (operation/cycle)	**Latency per batch (cycle/batch)**
Arithmetic	ADD, SUB	31.1	**1.0**	7.5	**4.3**
	MUL	15.9	**2.0**	1.7	**18.5**
	MAD	15.9	**2.0**	1.4	**22.6**
Logical	XOR, AND, OR	31.1	**1.0**	7.5	**4.3**
	SHL, SHR	15.6	**2.0**	3.9	**8.2**

(2)　メモリアクセス ($L_{local\_rand\_bt}$，$L_{local\_rgl\_bt}$，L_{glb_bt})

メモリアクセス命令に対するマイクロベンチマークによる測定を行い，生成された各 GPU のデータベースの内容を，前節の L_{inst_bt} の結果と併せて**表 6.3** に示す.

ランダムアクセスによって発生するバンクコンフリクトにより，いずれの GPU でも $L_{local\_rand\_bt}$ は $L_{local\_rgl\_bt}$ より大きくなった．また，$L_{local\_rand\_bt}$ 及び $L_{local\_rgl\_bt}$ では，バンク数に起因したメモリバンド幅の実効値の違いによって，Geforce GTX 580 と Geforce GTX 280 での測定結果に差が生じた.

Geforce GTX 580 における L_{glb_bt} は，PE とグローバルメモリの間に挿入された 128-bit ラインサイズのキャッシュメモリの効果により，Geforce GTX 280 と比較して大幅に小さくなった．このように，マイクロベンチマークによるバッチ当たりのグローバルメモリのレイテンシ測定では，GPU のキャッシュメモリの有無によって測定方法を使い分ける必要がないという利点がある.

表 6.3　生成された GPU のデータベース [cycle/batch]

GPU	$L_{local\_rand\_bt}$	$L_{local\_rgl\_bt}$	L_{glb_bt}	L_{inst_bt}
Geforce GTX 580	6.5	3.1	22.6	1.0
Geforce GTX 280	28.0	15.5	162.4	4.3

6.4.4　バッチ数を変化させた場合におけるバッチ当たりのレイテンシ

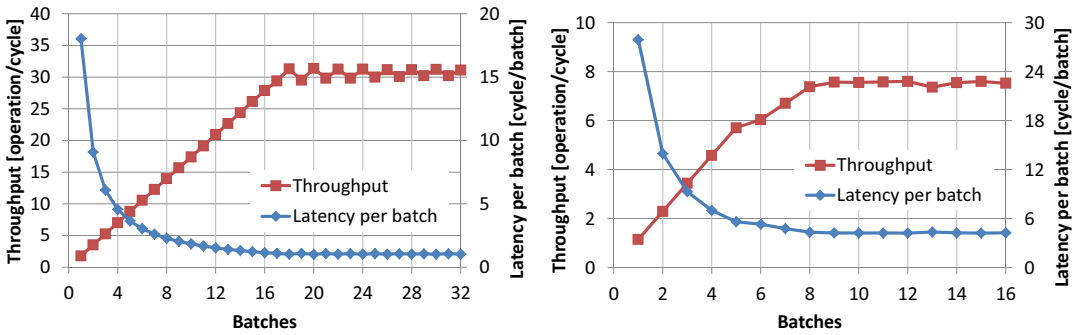
第 6.4.3 節で述べたデータベース生成の補足として，バッチ数を変化させた場合における，バッチ当たりのレイテンシの計測結果について述べる.

(1)　算術論理演算

算術論理演算における測定結果を，XOR 命令を例として**図 6.4** に示す．また，**第 6.4.3 節**と同様の理由から，スループットの計測結果も同図に併せて示す.

バッチ数を増加させていくと，バッチ当たりのレイテンシは，PE のパイプラインがビジーになるまで低下していく．そして，Geforce GTX 580 と Geforce GTX

280 では，バッチ数がそれぞれ 18 バッチと 8 バッチを超えると，バッチ当たりのレイテンシはそれぞれ 1.0 cycle/batch，4.3 cycle/batch と一定値になる．この結果は，**第 2.3 節**で前述した GPU アーキテクチャの特徴 (ii) の実証となっている．スループットも同様にバッチ数を増加させていくと，PE のパイプラインがビジーになるまで線形に増加する．そして，それぞれ上述したバッチ数でビジー状態となると，スループットはそれぞれ 31.1 operation/cycle と 7.5 operation/cycle で飽和する．これらの値は各 GPU における CU 内の PE 数（Geforce GTX 580; 32 PEs/CU，Geforce GTX 280; 8 PEs/CU）と概ね一致しており，XOR 命令が PE パイプラインを通じて理想的に並列処理されていることを確認できる．なお，Geforce GTX 580 では，バッチ数が $2N-1$（$N \geq 1$ を満たす整数）のときのスループットは，$2N$ のときより僅かに低い値となった．これは，Geforce GTX 580 をはじめとする Fermi アーキテクチャの GPU の CU では，命令が 2 つのバッチに対して並列に発行される [43] ためであると考えられる．



(a) Geforce GTX 580　　(b) Geforce GTX 280

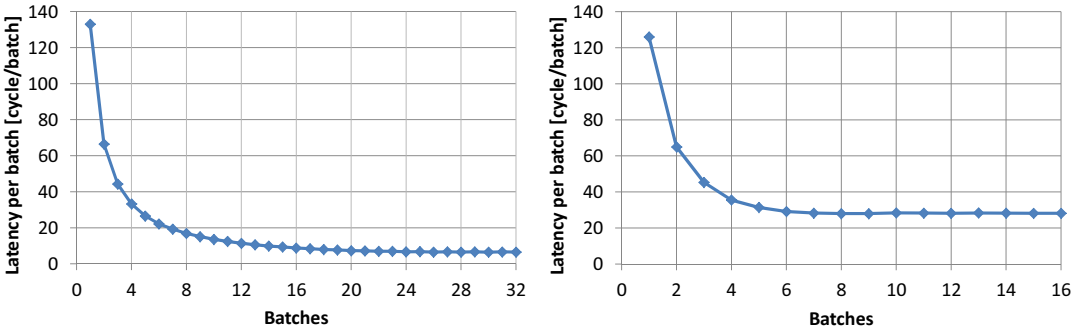
図 6.4　バッチ数を変化させた場合の XOR 命令のスループットとバッチ当たりのレイテンシ

(2)　メモリアクセス

まず，ローカルメモリに対するランダムアクセスとレギュラーアクセスにおける計測結果を，**図 6.5** と**図 6.6** にそれぞれ示す．バッチ数を増加させていくと，

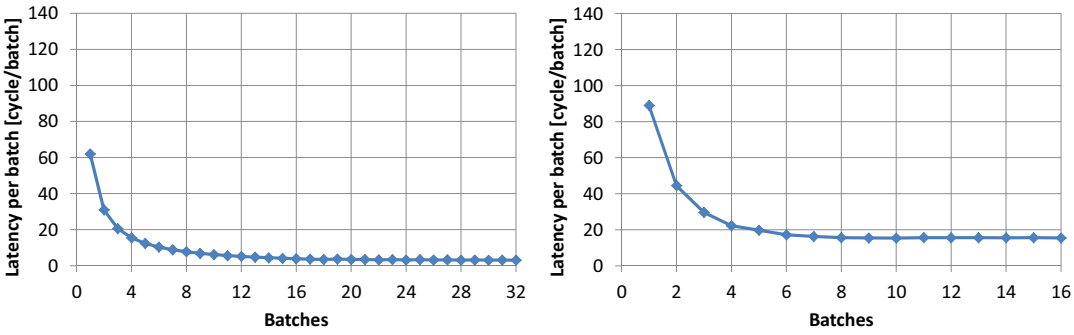
バッチ当たりのレイテンシは，最大バンド幅に達するまで低下していく．そして，Geforce GTX 580 と Geforce GTX 280 では，算術論理演算と同様に，バッチ数がそれぞれ 18 バッチと 8 バッチを超えると，バッチ当たりのレイテンシは概ね一定値となる．このとき，各アクセスパターンにおける，ローカルメモリの最大バンド幅に達したと考えられる．

次に，グローバルメモリに対するアクセスレイテンシの計測結果を**図 6.7** に示す．バッチ数を増加させていくと，ローカルメモリと同様に，バッチ当たりのレイテンシは低下していく．しかし，バッチ当たりのレイテンシが概ね一定値になるときのバッチ数は，Geforce GTX 580 と Geforce GTX 280 ではそれぞれ 8 及び 2 となっており，**図 6.5** と**図 6.6** に示した通りローカルメモリの場合よりも小さい．これは，グローバルメモリが，ローカルメモリと異なり，全ての CU に接続されていることにあると考えられる．つまり，グローバルメモリへのアクセスは，全ての CU から集中するため，ローカルメモリよりも少ないバッチ数で最大バンド幅に達したものと予想できる．



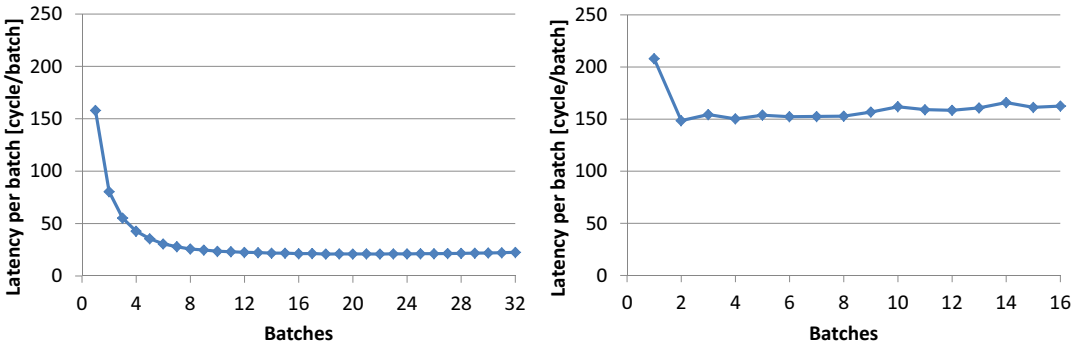
(a) Geforce GTX 580　　(b) Geforce GTX 280

図 6.5　バッチ数を変化させた場合のローカルメモリに対するランダムアクセスのバッチ当たりのレイテンシ



(a) Geforce GTX 580　　(b) Geforce GTX 280

図 6.6　バッチ数を変化させた場合のローカルメモリに対するレギュラーアクセスのバッチ当たりのレイテンシ



(a) Geforce GTX 580　　(b) Geforce GTX 280

図 6.7　バッチ数を変化させた場合のグローバルメモリに対するバッチ当たりのアクセスレイテンシ

6.5　PPF の評価

6.5.1　PPF の評価条件

これまで述べてきた通り，128-bit ブロック暗号の代表的な構造である SPN 構造，Feistel 構造，SPN と Feistel のハイブリッド構造のアルゴリズムとして，それぞれ AES，Camellia，SC2000 を，PPF の評価対象として選択した．秘密鍵の鍵長は 128-bit のみを対象とした．PPF の実行には，データベースを生成する際に使用した**表 4.4** の環境を再び用いた．また，PPF により出力される処理時間の

予測範囲の比較評価対象には，**図 5.2** に示した GPU 実装による暗号化処理速度の最大値を，**表 5.1** 及び**表 4.5** で示した Geforce GTX 580 と Geforce GTX 280 のクロックコア周波数で除して，クロックサイクル数に換算した値を用いる．

6.5.2 暗号の逐次処理コードの解析

ブロック暗号の GPU 実装では，その処理の高速化のためにソフトウェア実装に最適化されたアルゴリズムが用いられる．そのため PPF による予測評価は，AES，Camellia，SC2000 にはそれぞれ T-Box テーブル，SP テーブル，SM テーブルを用いたアルゴリズムによる GPU 実装を対象とした．

AES，Camellia，SC2000 に対してコード解析を行った結果を**表 6.4** にまとめる．3 種の暗号のブロック長はいずれも 128-bit であり，各ワークアイテムの各命令は 32-bit 命令なので，平文のサブブロックのロードまたは暗号文のサブブロックのストアの回数はいずれも 4 となっている．

なお SC2000 では，実装対象とする GPU のローカルメモリ容量に応じたテーブル構成法が選択できる [39]．**第 5.2 節**でも述べたように，Geforce GTX 580 と Geforce GTX 280 の各 CU のローカルメモリ容量はそれぞれ 48 KB 及び 16 KB であるため，SC2000 の Geforce GTX 580 と Geforce GTX 280 への実装は，20 KB のサイズとなる (11-bit，10-bit，11-bit) のテーブル構成法と，8.5 KB のサイズとなる (6-bit，10-bit，10-bit, 6-bit) のテーブル構成法によるアルゴリズムをそれぞれ用いている．合成する換字テーブルの数が少ないテーブル構成法を用いると，サイズは大きくなる反面，アクセス回数を少なくすることができる．**表 6.4** の SC2000 の各欄において，コンマによって区切られた左側と右側の数字は，それぞれ Geforce GTX 580 と Geforce GTX 280 へ実装したアルゴリズムに対する解析結果を示している．

表 6.4　予測式のパラメータとなる各ブロック暗号の要素数

Algorithm	Structure	Table (N_{tbl})	Key (N_{key})	Arithmetic & logical instructions (N_{inst})	Load of plaintext (N_{pt})	Store of ciphertext (N_{ct})
AES	SPN	160	44	436	4	4
Camellia	Feistel	144	48	642	4	4
SC2000 注 1)	Hybrid of SPN and Feistel	72, 96	56, 56	506, 590	4, 4	4, 4

注 1) 各欄における左側と右側の値は，それぞれ (11-bit，10-bit，11-bit) と (6-bit，10-bit，10-bit，6-bit) のテーブル構成法によるアルゴリズムで実装された逐次処理コードの解析結果を表す.

6.5.3　GPU へ実装される暗号の処理時間の予測

(1)　カーネル起動とデータ展開にかかるレイテンシの測定

式 (6.2) の計算には，**第 6.4 節**で述べた GPU のデータベースの他に，カーネル起動及びローカルメモリへのラウンド鍵と換字テーブルの展開にかかるレイテンシ（それぞれ L_{ker} と L_{ld}）が必要になる．ただし十分大きな平文を対象にした場合は，これらの値は無視できると考えられる．この予測を検証するため，L_{ker} についてはカーネルが起動して即終了し，L_{ld} については換字テーブルとラウンド鍵のデータをローカルメモリに展開して即終了するようなプログラムを記述し，レイテンシを測定した．

L_{ker} の測定結果は，Geforce GTX 580 と Geforce GTX 280 でそれぞれバッチ数に関わらず 0.002×10^7 cycle 及び 0.0007×10^7 cycle となった．一方 L_{ld} の測定結果は，Camellia で Geforce GTX 580 と Geforce GTX 280 でそれぞれ 0.003×10^7 cycle 及び 0.002×10^7 cycle となり，SC2000 では Geforce GTX 580 と Geforce GTX 280 でそれぞれ 0.004×10^7 cycle 及び 0.003×10^7 cycle となった．また，AES での L_{ld} の測定結果は，Camellia と同じ値となった．一方，本節 (2) に示すように，256 MB の平文サイズの暗号化には 1.0×10^7 cycle のオーダーの処理時間がかかる．このため，L_{ker} と L_{ld} が，全体の暗号化処理時間から無視できる程度であることを確認した．

(2)　処理時間の予測範囲の計算過程

GPU を用いた暗号化による処理時間の予測範囲算出を，Geforce GTX 580 における 256 MB の平文の Camellia 暗号化を例として以下に示す．

L_{enc_bt} の算出

式 (6.4) のパラメータとして必要な 4 種の要素におけるバッチ当たりのレイテンシと各暗号での使用回数は，それぞれ**表 6.3** と**表 6.4** で示した通りである．Geforce GTX 580 における Camellia の L_{enc_bt} はこれらの値を利用して，式 (6.4) により，

L_{enc_bt}

$$
\begin{aligned}
&= \begin{cases} 22.6\times(4+4)+6.5\times144+3.1\times48 \text{ (cycle)} \\ 22.6\times(4+4)+6.5\times144+3.1\times48+1.0\times642 \text{ (cycle)} \end{cases} \\
&= \begin{cases} 1265.6 \text{ (cycle) (SA サブモデル)} \\ 1907.6 \text{ (cycle) (SI サブモデル)} \end{cases}
\end{aligned} \tag{6.5}
$$

のように求められる．

L_{enc} の算出

次に，式 (6.5) より，ワークグループ当たりのワークアイテム数 ($N_{wi\_per\_wg}$) とワークグループ数 (N_{wg}) をそれぞれ 1024 と 64 と指定した場合の L_{enc} の計算過程を示す．NVIDIA GPU では $N_{batchsize} = 32$ であるから，$N_{batch} = \frac{N_{wi\_per\_wg}}{N_{batchsize}} = 32$ と求められる．また，$N_{cu} = 16$ である．各ワークアイテムは 128-bit (16 Byte) の平文ブロックを暗号化する．GPU 全体で起動されているワークアイテム数は $N_{wg}\times(N_{batchsize}\times N_{batch})$ なので，$16\times N_{wg}\times(N_{batchsize}\times N_{batch})$ バイトの平文が，暗号化カーネル内の反復処理による暗号化の 1 回のループで処理される．よって，256 MB の平文の暗号化に必要なループ回数 N_{iter} は，$\frac{256\times1024\times1024}{16\times N_{wg}\times(N_{batchsize}\times N_{batch})} = 256$ である．

最終的に，SA サブモデルに基づいて算出した L_{enc} は，式 (6.3) を用いて，$1265.6\times 256\times 32\times\frac{64}{16} = 4.147\times10^7$ cycle である．また，SI サブモデルに基づいた L_{enc} も，同様に式 (6.3) から 6.250×10^7 cycle と計算される．

L_{enc_total} の算出と実測値との比較

Geforce GTX 580 における L_{ker} と L_{ld} は，前述の通り，それぞれ 0.002×10^7 cycle と 0.003×10^7 cycle であったので，SA サブモデルに基づいた L_{enc_total} は式 (6.2) を用いて，$L_{enc\_total} = (0.002+0.003+4.147)\times10^7 = 4.152\times10^7$ cycle である．また，SI サブモデルに基づいた L_{enc_total} も，同様に式 (6.2) を用いて 6.255×10^7 cycle と計算される．

ワークグループ数とワークグループ当たりのワークアイテム数を Geforce GTX

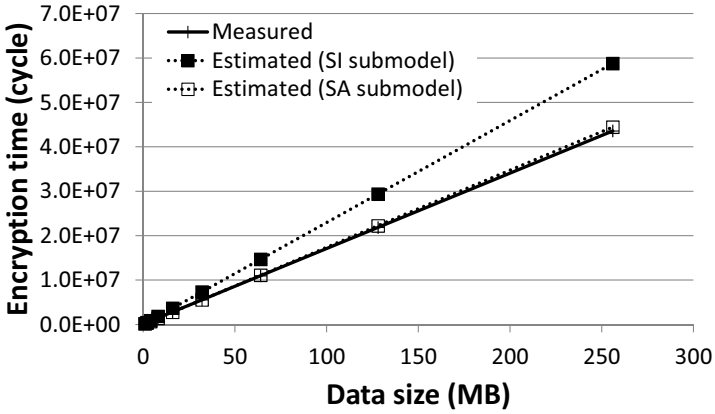
580 では 64 及び 1024，Geforce GTX 280 では 90 及び 512 と指定した場合，3 種のブロック暗号による暗号化処理時間の実測値と予測範囲の比較を，**図 6.8** と**図 6.9** にそれぞれ示す.

(3)　GPU 実装の暗号化に要する処理時間の予測結果に対する考察

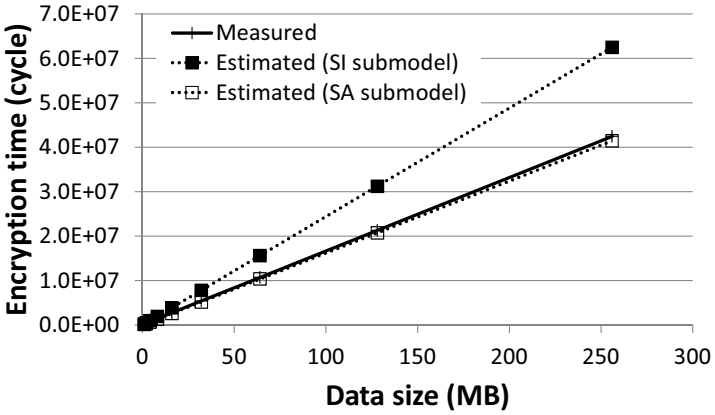
図 6.8 に示した結果から，Geforce GTX 580 の場合は実測値は予測範囲に収まり，PPF の有効性が確認できた．一方，**図 6.9** に示した Geforce GTX 280 の場合は，SA サブモデルで予測した範囲から僅かに外れている．これは，NVIDIA 社のドキュメント [71] で公開されているように，Geforce GTX 580 と Geforce GTX 280 ではそれぞれ Fermi 命令セットと GT200 命令セットという完全に異なる命令体系が使用されているためであると考えられる.

また，アウトオブオーダースケジューリングが十分に機能し，予測値が SA サブモデルに従うと想定したケースにおける，3 種のブロック暗号での暗号化に要する処理時間の予測誤差を**図 6.10** に示す.

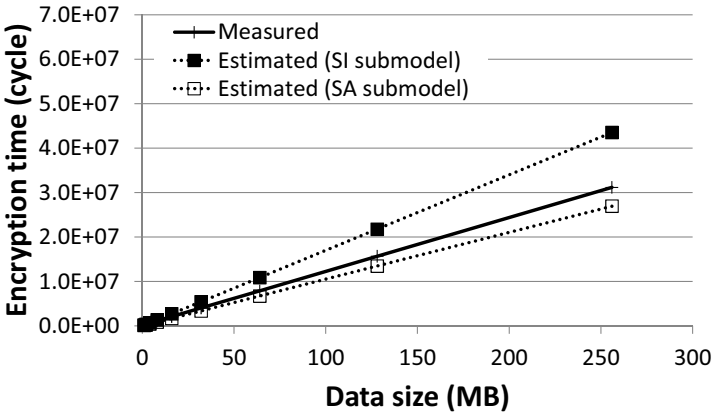
SI サブモデルから算出した予測値と実測値の差分は，メモリアクセス命令のレイテンシに隠蔽された算術論理演算の指標を示していると考えられる．Camellia の算術論理演算は 642 回と AES の 436 回よりも多いため，SI サブモデルから算出された Camellia の処理時間の予測値は，**図 6.8** と**図 6.9** で示したようにいずれの GPU でも AES より大きくなった．ところが暗号化に要する処理時間の実測値では SI サブモデルによる予測値とその優劣が逆転し，Camellia の実測値は AES と同じか僅かに低い程度であった．これは，Camellia では，メモリアクセスを伴う命令の回数（N_{tbl}，N_{key}，N_{pt} または N_{ct}）は AES と概ね同程度であるが算術論理演算の回数（N_{ct}）は AES よりも多いので，算術論理演算のレイテンシがメモリアクセスのレイテンシに大きく隠蔽されたためであると考えられる．一方 SC2000 では，Geforce GTX 580 と Geforce GTX 280 への実装の予測に用いたアルゴリズムはそれぞれ 72 回と 96 回のテーブルアクセスを含むが，Camellia の 144 回のテーブルアクセス回数よりは少ないため，算術論理演算を隠蔽するため



(a) AES

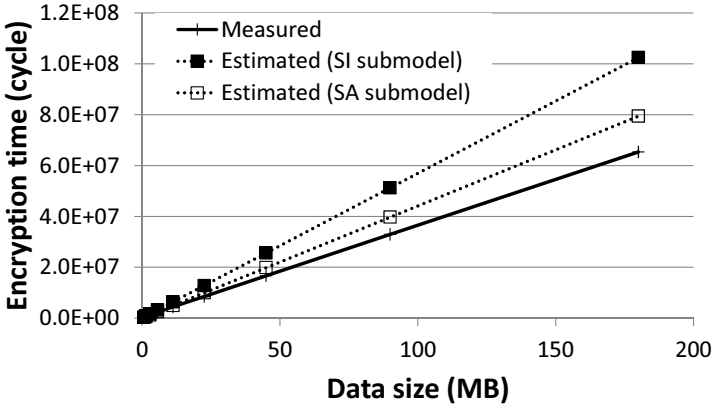


(b) Camellia

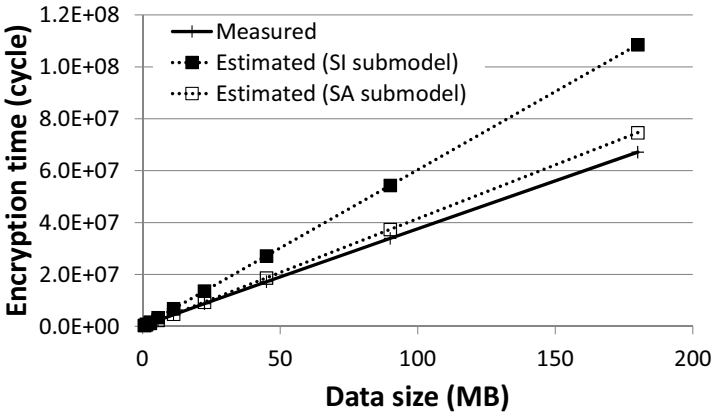


(c) SC2000

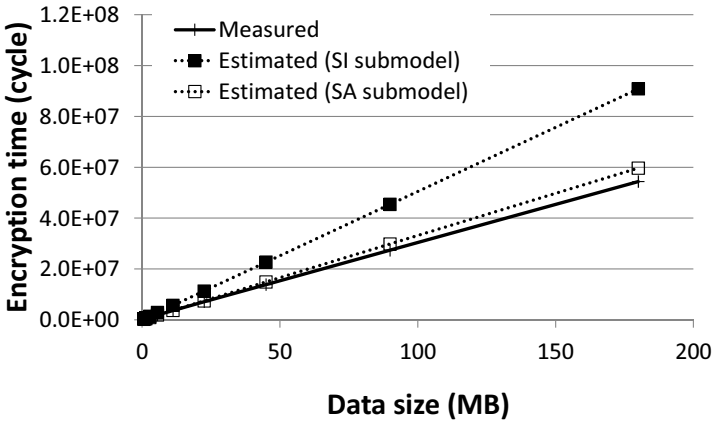
図 6.8 Geforce GTX 580 における暗号化処理時間の実測値と予測範囲の比較



(a) AES



(b) Camellia



(c) SC2000

図 6.9 Geforce GTX 280 における暗号化処理時間の実測値と予測範囲の比較

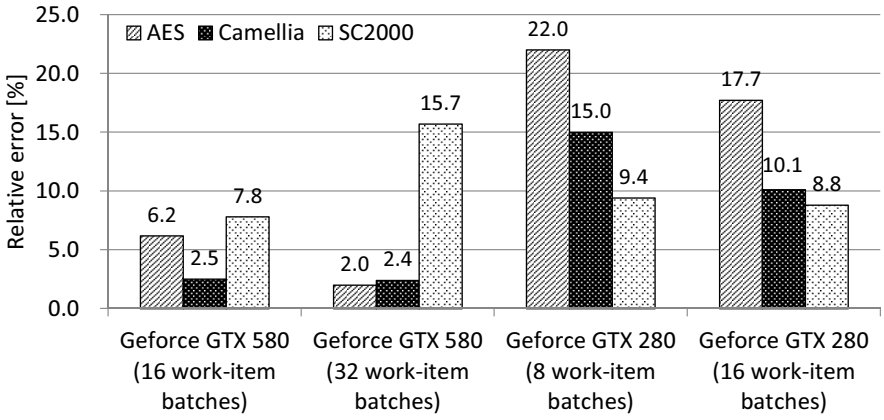


図 6.10　暗号化処理の実測値と SA サブモデルにより算出された予測値の相対誤差

のマージンは Camellia より小さくなると考えられる．実際，**図 6.8**(c) に示した Geforce GTX 580 における SC2000 の実装結果では，同図 (b) に示した Camellia の実装結果ほど算術論理演算は隠蔽されていない．以上の理由から，Camellia は，メモリアクセスを伴う命令と算術論理演算をバランスよく含んでおり，NVIDIA GPU アーキテクチャのアウトオブオーダースケジューリング機能による恩恵を受け易いアルゴリズムであることが分かる．

6.5.4　PPF を用いた設計段階の暗号による GPU 実装の処理時間の予測

(1)　設計段階での暗号

PPF の利点は，暗号の設計段階で GPU へ実装された場合の処理時間を予測可能なことにある．Camellia の仕様書 [38] は，最終的な暗号アルゴリズムに至るまでの設計の変遷を詳細に記述している．そこで本論文では，設計段階の暗号を GPU へ実装した場合の暗号化に要する処理時間を，Camellia の設計プロセスを例に予測する．

[38] には，Camellia の F 関数の設計が，128-bit ブロック暗号 E2[72] の「換字テーブル」→「XOR 及び巡回シフト」→「換字テーブル」という 2 段換字置換網から成る F 関数を踏襲したことが記述されている．そして，暗号化処理速度

を改善する過程で，最後の換字テーブルによる換字が省かれ,「換字テーブル」→「XOR 及び巡回シフト」という 1 段換字置換網へ変更されることになった.

また [38] には，Camellia の FL 及び FL^{-1} 関数の設計が，64-bit ブロック暗号 MISTY1[46] の FL 関数を踏襲したことが記述されている．そして，安全性向上のために，FL 及び FL^{-1} には巡回シフトが追加されることになった.

つまり，設計段階の Camellia では，F 関数の構造は 2 段換字置換網であり，また FL 及び FL^{-1} 関数では巡回シフトが省かれていたことになる．ここで，正規の Camellia の F 関数の最後に 1 段 8 並列の換字テーブルによる換字を追加し，復元された設計段階の F 関数を F' 関数と表す．F' 関数を**図 6.11** に示す．また，正規の FL 及び FL^{-1} 関数から巡回シフトを省き，復元された設計段階の FL 及び FL^{-1} 関数をそれぞれ FL' 及び $FL^{-1'}$ 関数と表す．設計段階の Camellia を GPU へ実装した場合の暗号化に要する処理時間の予測評価は，F' 関数または FL' 及び $FL^{-1'}$ 関数を用いて,

- **Pattern 1**： 「6 段の F' 関数」
- **Pattern 2**： 「6 段の正規 F 関数」→ 「FL' 及び $FL^{-1'}$ 関数」→ 「6 段の正規 F 関数」
- **Pattern 3**： 「6 段の正規 F 関数」→ 「正規の FL 及び FL^{-1} 関数」→「6 段の正規 F 関数」→ 「正規の FL 及び FL^{-1} 関数」→ 「6 段の F 関数」→ 「正規の FL 及び FL^{-1} 関数」→「6 段の正規 F 関数」→ 「正規の FL 及び FL^{-1} 関数」→ 「6 段の F 関数」

という 3 通りの流れによる暗号化を考察することとした.

(2) 設計段階の暗号に対する予測結果の考察

Pattern 1〜3 のアルゴリズムを用いて，Camellia の暗号設計をシミュレートし，暗号化処理時間の実測値と PPF を用いて算出した予測範囲の比較を**図 6.12** に示す.

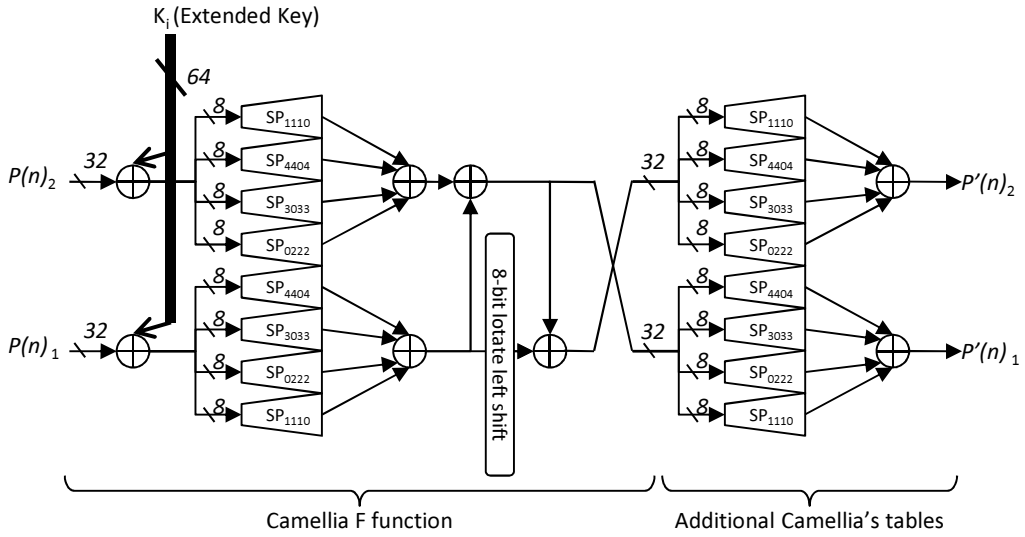


図 6.11　PPF の評価で使用する F' 関数

まず設計者は，初期段階において，F' 関数を用いた Pattern 1 のようなアルゴリズムを構築すると想定する．この場合の GPU 実装の実測値は，**図 6.12**(a) に示すように予測範囲に含まれ，PPF は実測値を予測できている．なお，Pattern 1 のアルゴリズムでは，$N_{tbl} = 72$，$N_{key} = 12$，$N_{pt} = N_{ct} = 4$ とメモリアクセスを多く含まない．そのため，算術論理演算の隠蔽マージンは小さくなり，GPU へ実装した場合の処理時間の実測値は，SI サブモデルと SA サブモデルから算出した予測値の概ね中間となった．

一般に，テーブルアクセス回数が多いと暗号としての安全性は向上するが，処理速度は低下する．そのためここでは，設計者が次に，F' 関数から 2 段目の換字テーブルを省いた F 関数を作成し，FL' 及び $FL^{-1\prime}$ 関数を挿入して Pattern 2 のようなアルゴリズムに改良したと想定する．この場合も，**図 6.12**(b) に示すように GPU 実装の実測値は予測範囲に含まれ，PPF が実測値を予測できていることを確認できる．なお Pattern 2 では $N_{tbl} = 96$，$N_{key} = 28$，$N_{pt} = N_{ct} = 4$ と，メモリアクセス回数が Pattern 1 よりも増加して算術論理演算の隠蔽マージンも大きくなり，暗号化処理時間の実測値は SI サブモデルから算出した予測値に近い

結果となった．

ラウンド数は，ブロック暗号の設計において，処理速度と安全性を決める重要なパラメータである [8]．ただし，ラウンド数と安全性にはトレードオフの関係がある．ラウンド数を減らすと処理速度は向上するが，安全性は低下する．そのため，設計者は，最小限の安全性確保に必要な値を下限とし，満足する処理速度を得られるようにラウンド数を決定する．従ってここでは，PPF がラウンド数の決定にも有効であることを示すため，設計者は，安全性を多めに確保した，正規の 128-bit 鍵の Camellia よりも過多なラウンドを実行する Pattern 3 のようなアルゴリズムを構築したと想定する．この場合も，PPF は**図 6.12**(c) に示すように，その GPU 実装の処理時間を予測できている．設計者は，この結果を基にラウンド数を調整し，最終的に**図 6.8**(b) のような処理時間のアルゴリズムとして Camellia を完成することができる．なお，Pattern 3 のようにメモリアクセスを伴う命令が一定数を超えると，算術論理演算が完全に隠蔽され，暗号化処理時間の実測値は SA サブモデルから算出した予測値と高い精度で一致するようになった．なお，ブロック暗号では，**第 3 章**で述べたように，秘密鍵の鍵長を数種類から選択できる．鍵長の違いは，ラウンド数の大小を左右する．上述した通り，PPF はラウンド数の異なるアルゴリズムに対しても，その GPU 実装の処理時間を予測できている．このため， PPF はブロック暗号において，秘密鍵の鍵長の違いにも対応できることが分かる．

6.6 既存研究との比較

GPU を利用したアプリケーションの性能を予測するモデルの研究がこれまでに数点報告されており，それらの概要を**表 6.5** にまとめる．これらのモデルには，主に以下の 3 点の問題がある．

- **問題 1** ベンダに依存した言語の利用
- **問題 2** ベンダに依存したツールの利用

・**問題 3** 対象とするアプリケーションを限定しないものの，複雑またはモデルの理解に特別な追加知識が必要

問題 1 については，モデルの適用範囲が，NVIDIA 社の GPU のみに限定されてしまうことになる．**表 6.5** に示したモデルは全て，NVIDIA 社の GPU の専用言語である CUDA[40] を用いて開発されている．一方，本研究における PPF の設計には，並列プロセッサの共通プラットフォームである OpenCL を採用しており，将来的にはマルチコア CPU や Intel Xeon Phi への拡張可能性を有している．

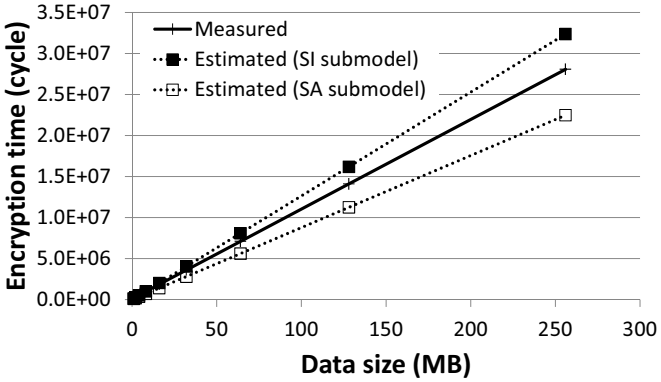
問題 2 についても，モデルの適用範囲が，特定のベンダの GPU のみに制限されてしまうことになる．また，モデル自体の利用方法に加え，モデルに組み込まれたツールの使用方法も把握する必要もある．更に，GPU に新しい製品が登場した場合は，そのサポートのため，ベンダによるツールのバージョンアップを待たなくてはならない．**表 6.5** に示した [32] 及び [33] では，NVIDIA 社の GPU アプリケーション開発用のツールを利用している．[32] では，モデルの計算に必要なパラメータを取得するため，NVIDIA 社の GPU に依存した仮想アセンブリ言語である，Parallel Thread Execution ISA[73] を利用している．[33] では，GPU で実行されるコードを取得するため，NVIDIA GPU 用の逆アセンブラである Decuda[74] を利用している．また，[33] では，モデルの計算に必要なパラメータを取得するため，NVIDIA GPU での処理をシミュレートする Barra[75] を利用している．一方，PPF では，性能予測値の算出のために必要なパラメータを，ベンダ依存のツールを用いず，ブロック暗号の逐次処理コードの暗号化関数部分の解析と事前に生成したデータベースから取得するように設計している．

問題 3 については，利用者によるモデルの利用を妨げてしまうことになる．[32] では，**第 2.3 節**で述べた，ワープ[注 1)] 間のアウトオブオーダースケジューリング機能を抽象化している．同機能には，バッチ数や GPU のメモリバンド幅のよう

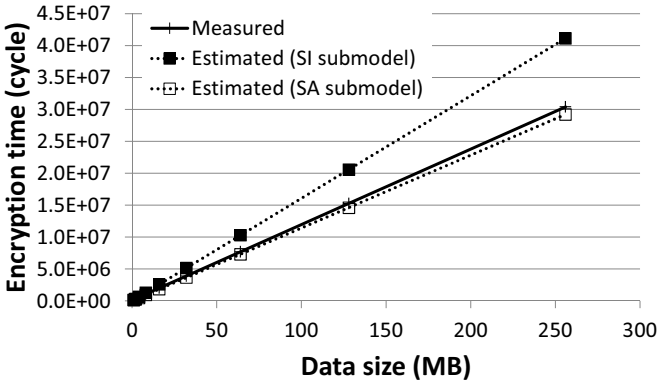
[注 1)] NVIDIA GPU 専用のプログラミング環境 CUDA[40] では，ワークアイテムバッチはワープという用語として知られている．

に，ソフトウェアとハードウェアの両面が絡む．そのため，同機能の抽象化は，モデルの設計を複雑にする．実際，[32] のモデルは，21 個ものパラメータと 27 本の式を使用して実現されている．[34] のモデルは，その内部に [32] のモデルを組み込んでおり，[32] と同様，複雑な設計となっている．また，[35] では，プログラム依存グラフをベースとした予測手法が採用されている．そのため，[35] のモデルを利用するには，GPU アーキテクチャの理解に加え，プログラム依存グラフの知識が必要である．一方，本研究では，GPU を利用したアプリケーションの性能を，計算命令のレイテンシがメモリアクセスに全く隠蔽されない場合と完全に隠蔽される場合のレイテンシをそれぞれ上限，下限とし，特別な知識を必要としない，シンプルな性能予測フレームワークを提案している．なお，本研究で用いたアウトオブオーダースケジューリング機能の簡潔な抽象化方法は，[36] のモデルでも同様に用いられている．

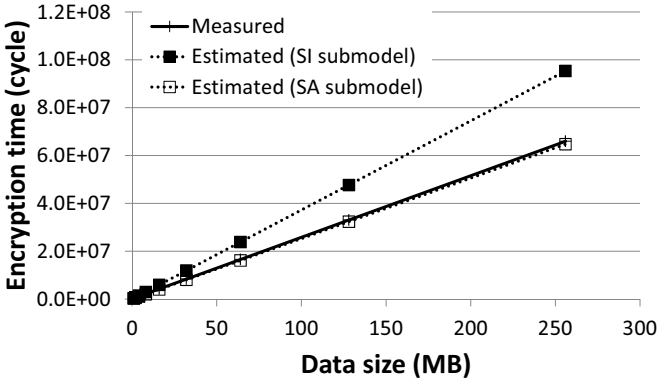
なお，本研究の PPF と同様に，アプリケーションに特化したモデルとして，粗行列ベクトル積に専用のモデルを開発した [37] がある．このモデルでは，マイクロベンチマークとして，計算対象の行列における部分行列の処理時間を計測した後，全体の行列計算の性能を予測するという手法を用いている．ただし，このモデルでは，複数個のワープを起動させた場合のアウトオブオーダースケジューリング機能のような，GPU アーキテクチャの特徴が無視されている．



(a) Pattern 1 ($N_{tbl}=72$, $N_{key}=12$, $N_{inst}=302$, $N_{pt}=4$, $N_{ct}=4$)



(b) Pattern 2 ($N_{tbl}=96$, $N_{key}=28$, $N_{inst}=364$, $N_{pt}=4$, $N_{ct}=4$)



(c) Pattern 3 ($N_{tbl}=240$, $N_{key}=76$, $N_{inst}=934$, $N_{pt}=4$, $N_{ct}=4$)

図 6.12　設計段階の Camellia による Geforce GTX 580 での暗号化処理時間の予測結果

表 6.5　GPU へ実装されたアプリケーションの性能を予測する研究

文献	言語	適用範囲	評価対象	誤差（使用 GPU）	モデルの特徴と問題点
[32]	CUDA	制限なし	Sepia，Linear，SVM，Mat.(naive)，Mat.(tiled)，Blacksholes	相乗平均で 13.3 %（Geforce GTX 280）	複雑（21 個のパラメータと 27 本の式を使用）
[33]	CUDA	制限なし	密行列積，三重対角行列ソルバ，疎行列ベクトル積	5〜15 %（Geforce GTX 8800）	ベンダ依存のツール [74][75] を使用
[34]	CUDA	制限なし	行列積，HotSpot，IspinEx，SpinFlap	相乗平均で 17 % (Tesla C1060)	複雑（[32] が組み込まれている）
[35]	CUDA	制限なし	密行列積	13〜48 %（Geforce GTX 280）	複雑（プログラム依存グラフを使用）
[36]	CUDA	制限なし	行列積，List ranking，ヒストグラム生成	18 %（Geforce GTX 280）	1 スレッドの処理にかかる処理時間から全体の処理時間を計算
[37]	CUDA	疎行列ベクトル積	32 種の疎行列ベクトル積	10 %以下（Geforce GTX 580）	疎行列ベクトル積のみに特化
本研究	**OpenCL**	**ブロック暗号**	**AES，Camellia，SC2000**	**概ね予測範囲内に含まれ，SA サブモデルに対しては 2.0〜15.7 % (Geforce GTX 580)**	**ベンダ依存の言語やツールを使用せず，ブロック暗号のみに特化して単純化**

6.7 第 6 章のまとめ

本章では，暗号アルゴリズムを選択して実行可能な並列プロセッサとして注目されている GPU 向けに，効率的にブロック暗号を設計するためのフレームワークである PPF を開発し，評価した結果を述べた．PPF の設計には，並列プロセッサへの拡張を考慮して，ベンダに依存したツール等を使用せず，並列計算アーキテクチャに共通のプラットフォームである OpenCL を用いることとした．また，設計には，**第 4 章**で導出した GPU へのブロック暗号の実装方針を用い，ブロック暗号に特化した簡潔なフレームワークを実現した．PPF では，第 1 段階として，逐次処理のブロック暗号のコード内にある暗号化関数を解析し，4 種の要素の数（逐次処理コード内で換字テーブルにアクセスする回数，ラウンド鍵にアクセスする回数，算術論理演算を実行する回数，メモリからレジスタへ平文のサブブロックをロードする回数または暗号文のサブブロックをメモリへストアする回数）を取得する．第 2 段階として，これらのパラメータを基に，予測式を用いて，ブロック暗号を GPU へ実装したと想定したと想定した場合の処理時間の上限値及び下限値を出力する．なお，予測式での計算時に必要となる，上述した 4 種の要素を GPU で処理した場合にかかるバッチ当たりのレイテンシは，マイクロベンチマークを用いて事前に生成しておいたデータベースから取得する．

PPF の評価には，ブロック暗号の代表的な構造である SPN，Feistel，SPN と Feistel のハイブリッド構造の暗号アルゴリズムを用いることとした．SPN，Feistel，SPN と Feistel のハイブリッド構造には，それぞれ 128-bit ブロック暗号である AES，Camellia，SC2000 を例として選択した．GPU には，異なる世代のアーキテクチャで開発された，Geforce GTX 580 と Geforce GTX 280 という 2 種の GPU を用いた．上記の 2 種の GPU を用いて，これら 3 種のブロック暗号を並列実行した処理時間の測定結果は，PPF から見積もられる結果と概ね一致した．最後に，設計段階の暗号を GPU へ実装した場合に要する処理時間を，Camellia の設計プロセスを例に評価した．この場合も，上記 2 種の GPU 実装による処理時間

が予測できることを確認し，GPU への暗号アルゴリズムの設計負担が軽減できることを確認できた.

第 7 章

結　論

7.1　本研究により得られた成果

データの防護には暗号化処理が基本手段であり，これまでに用途に応じた様々な暗号アルゴリズムが提案されてきた．このうち，大容量データの暗号化に向く特徴を持つブロック暗号が広く普及し，ストレージの暗号化等に用いられている．このような特徴から，CLEFIA や PRESENT のような新たなアルゴリズムが国際標準規格として採用されているように，ブロック暗号には今後も高い要求がある．その一方で，プロセッサのアーキテクチャはマルチコアやメニーコアが一般的となっているため，今後は並列プロセッサを対象として暗号アルゴリズムを設計する必要性が生じてきている．しかし，現在利用されている並列プロセッサは多種多様であるため，その設計負担は大きくなっている．そこで本研究では，並列プロセッサに対して，暗号アルゴリズムの設計及び実装評価を効率化するフレームワークである PPF を開発した．PPF の主な特徴は以下の通りである．

- 並列プロセッサへ広範囲に適用できるようにするため，並列プロセッサに共通のフレームワークである OpenCL を用いて設計を行った．これにより PPF は，GPU ベンダに依存したフレームワークとならず，将来的にはマルチコア CPU や Intel Xeon Phi 等の並列プロセッサへの拡張も可能となっている．

- PPF の普及のためには，暗号設計者にとって単純で，理解が用意であることが望ましい．そのため，ブロック暗号に共通する GPU への実装方針を導出し，同方針を PPF の設計に組み込んだ．その結果，PPF は，20 個のパ

ラメータと 3 つの式で記述された簡潔なフレームワークとして設計されている．

PPF の評価には，Geforce GTX 580 と Geforce GTX 280 という 2 種の GPU を用いた．評価結果を以下にまとめる．

- ブロック暗号の代表的な構造である，SPN 構造，Feistel 構造，SPN と Feistel のハイブリッド構造の暗号アルゴリズムに対して，それぞれ PPF の評価を行ったところ，これらの暗号化処理時間を測定した結果は，PPF から見積もられる結果と概ね一致した．
- 仕様書を基にシミュレートした設計段階の Camellia に対しても，GPU 実装による暗号化処理時間を予測できることを確認し，効果的な開発を補助できることを確認できた．

7.2 今後の課題

今後の課題を以下にまとめる．

- 本研究では，PPF の評価を Fermi 及び GT200 アーキテクチャの NVIDIA GPU のみで評価した．今後は，Kepler アーキテクチャ[76] の NVIDIA GPU に加え，AMD 社の GPU[77] や，Intel Iris Graphics[78]，ARM Mali[79]，Qualcomm Adreno[80] といったモバイル型の GPU にも評価を行う必要がある．また，マルチコア CPU や Intel Xeon Phi などの並列プロセッサに対しても，PPF の適用可能範囲を拡張していく必要がある．
- 本研究では，予測対象とする暗号アルゴリズムとして，ストレージの暗号化等の用途に広く利用されている共通鍵ブロック暗号を選択した．共通鍵ブロック暗号の代表的な構造を採用するアルゴリズムとして，SPN 構造には AES，Feistel 構造には Camellia，SPN と Feistel のハイブリッド構造には SC2000 をそれぞれ用いた．今後は，一般型 Feistel 構造を採用したブロッ

ク暗号に対しても，PPF を評価していく必要がある．また，暗号方式としては，共通鍵暗号以外にも公開鍵暗号が利用されている．そのため，今後は，これらの暗号にも PPF を対応させていく必要がある．

謝　　辞

この論文は，防衛大学校理工学研究科後期課程における研究成果として執筆されたものです．研究の遂行および論文の執筆にあたり，防衛大学校情報工学科 コンピュータ工学研究室 黒川 恭一 教授，田中 秀磨 准教授，岩井 啓輔 講師 より，研究の全般の方向性および方法論について御鞭撻と御助言を賜りました．厚く御礼申し上げます．また，本論文の審査にあたり，御多忙中にも関わらず，多くの御助言を賜りました 慶応大学理工学部 天野 英晴 教授，防衛大学校情報工学科 中村 康弘 教授 に深く感謝し，御礼申し上げます．

本論文はまた，防衛省 陸上幕僚監部 の実施する部外委託教育（防衛大学校理工学研究科留学・後期課程）の研修成果でもあります．3 年という長期に渡る研修期間を与えてくださった陸上幕僚監部の関係各位に対し，厚く御礼を申し上げます．

研究の遂行に関し，様々な面で支えていただいた コンピュータ工学研究室 の学生および OB の皆様（特に 南崎 大作 1 等海尉，菅野 哲太郎 1 等海尉，川村 和範 様，若林 邦爾 1 等海尉，櫻井 敦規 1 等空尉，生西 賢一郎 2 等陸尉, 岩元 勇樹 1 等陸尉, 戸塚 亙志 1 等海尉，山岡 香苗 1 等海尉，水野 弘章 2 等陸尉，朴 駿漢 韓国海軍少佐, 渡部 匡 2 等海尉, Anh Tuan Nguyen ベトナム陸軍上尉, Enkhbold Chimedtseren モンゴル陸軍少尉）並びに防衛大学校入校期間中，関わりのあった全ての方々に対し，心からの感謝を申し上げます．

最後に，本研究活動に対して深い理解を示し，協力してくれた家族に心から感謝します．

参 考 文 献

[1] 技術評論社. KNOPPIX + α でシンクライアント環境を構築しよう. Software Design 2006 年 7 月号, 2006.

[2] 独立行政法人 情報処理推進機構. 情報セキュリティ白書 2012, 2012.

[3] 総務省 スマート・クラウド研究会. 暗号アルゴリズムの利用実績に関する調査報告書, 2010. http://www.soumu.go.jp/main_content/000066036.pdf.

[4] International Solid-State Circuits Conference 2013. ISSCC 2013 Press Kit, 2013. http://isscc.org/doc/2013/ISSCC_2013_PressKit.pdf.

[5] 日本ネットワークセキュリティ協会. 2012 年情報セキュリティインシデントに関する調査報告書〜個人情報漏えい編〜, 2011.

[6] 日本ネットワークセキュリティ協会. 2012 年情報セキュリティインシデントに関する調査報告書【上半期速報版】ver.1.1, 2012.

[7] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible Protocols and Atomic Proxy Cryptography. In *International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT'98)*, Vol. 1403 of *Lecture Notes in Computer Science*, pp. 127–144, 1998.

[8] 通信・放送機構. 共通鍵ブロック暗号の選択／設計／評価に関するドキュメント, 2000.

[9] International Organization for Standardization and International Electrotechnical Commission. ISO/IEC 29192-2:2012 Information technology – Security techniques – Lightweight cryptography – Part 2: Block ciphers, 2012. https://www.iso.org/obp/ui/#iso:std:iso-iec:29192:-2:ed-1:v1:en.

[10] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-Bit Blockcipher CLEFIA (Extended Abstract). In *Proceedings of 14th Fast Software Encryption*, Vol. 4593 of *Lecture Notes in Computer Science*, pp. 181–195, 2007.

[11] A. Bogdanov, L. R. Knudsen, G. Le, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In *Proceedings of 9th Workshop on Cryptographics Hardware and Embedded Systems*, Vol. 4727 of *Lecture Notes in Computer Science*, pp. 450–466, 2007.

[12] National Institute of Standards and Technology. Advanced Encryption Standard, Federal Information Processing Standards Publication 197, 2001. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

[13] Intel Corp. Securing the Enterprise with Intel AES-NI, 2010. `http://www.intel.com/content/dam/doc/white-paper/enterprise-security-aes-ni-white-paper.pdf`.

[14] 独立行政法人 情報処理推進機構. `https://www.ipa.go.jp/index.html`.

[15] 独立行政法人 情報処理推進機構. 暗号アルゴリズムの利用実績に関する調査報告書, 2012. `http://www.ipa.go.jp/files/000014263.pdf`.

[16] 株式会社フィックスターズ. 改訂新版 OpenCL 入門 1.2 対応マルチコア CPU・GPU のための並列プログラミング. インプレスジャパン, 2012.

[17] Intel Corp. Intel Xeon Phi Coprocessor Developer's Quick Start Guide, 2012. `http://software.intel.com/sites/default/files/article/335818/intel-xeon-phi-coprocessor-quick-start-developers-guide.pdf`.

[18] Michael Gschwind, Fred Gustavson, and Jan F. Prins. High Performance Computing with the Cell Broadband Engine. *Scientific Programming*, Vol. 17, No. 1-2, pp. 1–2, 2009.

[19] Cryptography Research and Evaluation Committees. `http://www.cryptrec.go.jp/english/index.html`.

[20] Cryptography Research and Evaluation Committees. CRYPTREC Report 2002 暗号技術評価報告書, 2002. `http://www.cryptrec.go.jp/english/index.html`.

[21] Cryptography Research and Evaluation Committees. 電子政府における調達のために参照すべき暗号のリスト（CRYPTREC 暗号リスト）, 2013. `http://www.cryptrec.go.jp/images/cryptrec_ciphers_list_2013.pdf`.

[22] Intel Corp. Intel Pentium III Processor Specification Update Revision 060, 2008. `http://download.intel.com/design/intarch/specupdt/24445358.pdf`.

[23] Sun Microsystems Inc. UltraSPARC-IIi User ' s Manual, 1997. `http://www.weblearn.hs-bremen.de/risse/RST/WS06/x86_SUN/Dokumentation/SPARC/UltraSPARC%20IIi%20User's%20Manual%20.pdf`.

[24] Compaq Computer Corp. Alpha 21264 Microprocessor Hardware Reference Manual, 1999. `http://h18000.www1.hp.com/cpq-alphaserver/technology/literature/21264hrm.pdf`.

[25] ZiLOG Inc. Z80 Family CPU User Manual UM008004-1204, 2004. `http://www.z80.info/zip/z80cpu_um.pdf`.

[26] Osvaldo Gervasi, Diego Russo, and Flavio Vella. The AES Implantation Based on OpenCL for Multi/many Core Architecture. In *Proceedings of 2010*

International Conference on Computational Science and Its Applications, pp. 129–134, 2010.

[27] Xingliang Wang, Xiaochao Li, Mei Zou, and Jun Zhou. AES Finalists Implementation for GPU and Multi-Core CPU Based on OpenCL. In *Proceedings of the 2011 IEEE International Conference on Anti-Counterfeiting, Security and Identification*, pp. 38–42, 2011.

[28] 熊木武志, 黒川悠一朗, 藤野毅. 超小型組込みボードを用いた暗号処理の並列化に関する研究. 電子情報通信学会技術研究報告, 第 110(473) 巻, pp. 279–284, 2011.

[29] 望月陽平, 吉田直之, 松本直樹, 村上佑馬, 熊木武志, 藤野毅. SIMD 型組込みプロセッサによる疑似乱数生成アルゴリズムの並列処理実装とその評価. 電子情報通信学会論文誌, Vol. J95-D, No. 3, pp. 376–386, 2012.

[30] 本多隼也, 望月陽平, 熊木武志, 藤野毅. ストリーム暗号 CryptMT のデータ並列処理による高速化手法及び SIMD 型組込みプロセッサによる実装と評価. 電子情報通信学会論文誌, Vol. J96-D, No. 3, pp. 495–505, 2013.

[31] Khronos Group. Open Compute Language. `http://www.khronos.org/registry/cl/specs/opencl-2.0.pdf`.

[32] Sunpyo Hong and Hyesoon Kim. An Analytical Model for a GPU Architecture with Memory-Level and Thread-Level Parallelism Awareness. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, pp. 152–163, 2009.

[33] Yao Zhang and John D. Owens. A Quantitative Performance Analysis Model for GPU Architectures. In *Proceedings of The 17th IEEE International Symposium on High-Performance Computing Architecture*, pp. 382–393, 2011.

[34] Jiayuan Meng, Vitali A. Morozov, Kalyan Kumaran, Venkatram Vishwanath, and Thomas D. Uram. GROPHECY: GPU Performance Projection from CPU Code Skeletons. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011.

[35] Sara S. Baghsorkhi, Matthieu Delahaye, Sanjay J. Patel, William D. Gropp, and Wen mei W. Hwu. An Adaptive Performance Modeling Tool for GPU Architectures. In *Proceedings of 15th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, pp. 105–114, 2010.

[36] Kishore Kothapalli, Rishabh Mukherjee, M. Suhail Rehman, Suryakant Patidar, P. J. Narayanan, and Kannan Srinathan. A Performance Prediction Model for the CUDA GPGPU Platform. In *Proceedings of International Conference of High Performance Computing*, pp. 463–472, 2009.

[37] Ping Guo and Liqiang Wang. Accurate CUDA Performance Modeling for Sparse Matrix-Vector Multiplication. In *Proceedings of The 2012 International Conference on High Performance Computing & Simulation*, pp. 496–502, 2012.

[38] 日本電信電話株式会社，三菱電機株式会社. 128 ビットブロック暗号 Camellia アルゴリズム仕様書（第 2.0 版）, 2001. `http://www.cryptrec.go.jp/cryptrec_03_spec_cypherlist_files/PDF/06_01jspec.pdf`.

[39] 富士通研究所，東京理科大学. 共通鍵ブロック暗号 SC2000 暗号技術仕様書, 2001. `https://www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/9_01jspec.pdf`.

[40] NVIDIA Corp. Compute Unified Device Architecture. `http://www.nvidia.com/object/cuda_home_new.html`.

[41] NVIDIA Corp. OpenCL Programming Guide for the CUDA Architecture, 2012. http://hpc.oit.uci.edu/nvidia-doc/sdk-cuda-doc/OpenCL/doc/OpenCL_Programming_Guide.pdf.

[42] AMD Corp. Southern Islands Series Instruction Set Architecture, 2012. http://developer.amd.com/tools/hc/AMDAPPSDK/assets/AMD_Southern_Islands_Instruction_Set_Architecture.pdf.

[43] NVIDIA Corp. Whitepaper for NVIDIA's Fermi Architecture, 2009. http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf.

[44] National Institute of Standards and Technology. Data Encryption Standard, Federal Information Processing Standards Publication 46-2, 1993. http://www.itl.nist.gov/fipspubs/fip46-2.htm.

[45] Arthur Sorkin. Lucifer, a Cryptographic Algorithm. *Cryptologia*, Vol. 8, No. 1, pp. 22–41, 1984.

[46] 三菱電機株式会社. 暗号技術仕様書 MISTY1, 2002. http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/5_02jspec.pdf.

[47] New European Schemes for Signatures, Integrity and Encryption. https://www.cosic.esat.kuleuven.be/nessie/.

[48] The IEEE Security in Storage Working Group. XTS block cipher-based mode (XEX-based tweaked-codebook mode with ciphertext stealing). http://siswg.net/.

[49] IEEE P1619. Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices. IEEE-Std 1619-2007.

[50] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES – The Advanced Encryption Standard*. Springer Verlag, 2002.

[51] Yuliang Zheng, Tsutomu Matsumoto, and Hideki Imai. On the Construction of Block Ciphers Provably Secure and Not Relying on Any Unproved Hypothesis. In *16th Proceedings of International Cryptology Conference*, Vol. 435 of *Lecture Notes in Computer Science*, pp. 461–480, 1990.

[52] James Abel, Kumar Balasubramanian, Mike Bargeron, Tom Craver, and Mike Phlipot. Applications Tuning for Streaming SIMD Extensions. Intel Technology Journal, Q2 1999. `https://noggin.intel.com/content/applications-tuning-for-streaming-simd-extensions`.

[53] Chris Lamb. OpenCL for NVIDIA GPUs, 2009. `http://www.hotchips.org/wp-content/uploads/hc_archives/hc21/1_sun/HC21.23.2.OpenCLTutorial-Epub/HC21.23.250.Lamb-NVIDIA-OpenCL--for-NVIDIA-GPUs.pdf`.

[54] NVIDIA Corp. NVIDIA GeForce GTX 200 GPU Architectural Overview, 2008. `http://www.nvidia.com/docs/IO/55506/GeForce_GTX_200_GPU_Technical_Brief.pdf`.

[55] NVIDIA Corp. Profiler User's Guide, 2012. `http://docs.nvidia.com/cuda/profiler-users-guide/`.

[56] Intel Corp. A Guide to Vectorization with Intel C++ Compilers, 2012. `http://download-software.intel.com/sites/default/files/8c/a9/CompilerAutovectorizationGuide.pdf`.

[57] Andrea Di Biagio, Alessandro Barenghi, Giovanni Agosta, and Gerardo Pelosi. Design of a Parallel AES for Graphics Hardware using the CUDA

Framework. In *Proceedings of 23th IEEE International Parallel & Distributed Processing Symposium*, pp. 1–8, 2009.

[58] Intel Corp. Intel Hyper-Threading Technology. http://www.intel.com/content/www/us/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html.

[59] OpenMP Architecture Review Board. OpenMP Application Program Interface Version 4.0. http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf.

[60] The OpenSSL Project. OpenSSL: The Open Source toolkit for SSL/TLS. http://www.openssl.org.

[61] Intel Corp. Boosting OpenSSL AES Encryption with Intel IPP, 2012. http://software.intel.com/fr-fr/articles/boosting-openssl-aes-encryption-with-intel-ipp.

[62] Khronos Group. The OpenGL Graphics System: A Specification Version 4.4, 2013. http://www.opengl.org/registry/doc/glspec44.core.pdf.

[63] Debra L. Cook, John Ioannidis, Angelos D. Keromytis, and Jake Luck. CryptoGraphics: Secret Key Cryptography Using Graphics Cards. In *The Cryptographers' Track at the RSA Conference 2005*, Vol. 3376 of *Lecture Notes in Computer Science*, pp. 334–350, 2005.

[64] 及川一樹, 王家宏, 児玉英一郎, 高田豊雄. OpenCL を用いた暗号アルゴリズムの実装と評価. SCIS 2009 論文集 3C4-2, 2010.

[65] Johannes Gilger, Johannes Barnickel, and Ulrike Meyer. GPU-Acceleration of Block Ciphers in the OpenSSL Cryptographic Library. In *Proceedings*

of the 15th International Conference on Information Security, Vol. 7483 of *Lecture Notes in Computer Science*, pp. 338–353, 2012.

[66] Gu Liu, Hong An, Wenting Han, Guang Xu, Ping Yao, Mu Xu, Xiurui Hao, and Yaobin Wang. A Program Behavior Study of Block Cryptography Algorithms on GPGPU. In *The 4th International Conference on Frontier of Computer Science and Technology*, pp. 33–39, 2009.

[67] Owen Harrison and John Waldron. AES Encryption Implementation and Analysis on Commodity Graphics Processing Units. In *Proceedings of 9th Workshop on Cryptographic Hardware and Embedded Systems*, Vol. 4727 of *Lecture Notes in Computer Science*, pp. 209–226, 2007.

[68] Svetlin A. Manavski. CUDA Compatible GPU as an Efficient Hardware Accelerator for AES Cryptography. In *Proceedings of 2007 IEEE International Conference on Signal Processing and Communication*, pp. 65–66.

[69] Henry Wong, Misel-Myrto Papadopoulou, Maryam Sadooghi-Alvandi, and Andreas Moshovos. Demystifying GPU Microarchitecture through Microbenchmarking. In *Proceedings of 2010 IEEE International Symposium on Performance Analysis of Systems & Software*, pp. 235–246, 2010.

[70] Andreas Resios. GPU Performance Prediction using Parameterized Models, 2009. Master thesis of Utrecht University.

[71] NVIDIA Corp. cuobjdump, 2013. `http://docs.nvidia.com/cuda/cuda-binary-utilities/index.html`.

[72] Masayuki Kanda, Shiho Moriai, Kazumaro Aoki, Hiroki Ueda, Youichi Takashima, Kazuo Ohta, and Tsutomu Matsumoto. E2-A New 128-Bit Block

Cipher. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. 83, No. 1, pp. 48–59, 2000.

[73] NVIDIA Corp. Parallel Thread Execution ISA v3.2, 2013. http://docs.nvidia.com/cuda/pdf/ptx_isa_3.2.pdf.

[74] Wladimir J. van der Laan. Decuda and Cudasm, the cubin utilities package, 2009. https://github.com/laanwj/decuda.

[75] Sylvain Collange, Marc Daumas, David Defour, and David Parello. Barra: A Parallel Functional Simulator for GPGPU. In *Proceedings of The 18th Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp. 351–360, 2010.

[76] NVIDIA Corp. Whitepaper for NVIDIA's Kepler Architecture, 2012. http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf.

[77] AMD Corp. AMD Accelerated Parallel Processing OpenCL Programming Guide rev. 2.7, 2013. http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2013/07/AMD_Accelerated_Parallel_Processing_OpenCL_Programming_Guide-rev-2.7.pdf.

[78] Intel Corp. Intel HD and Iris Graphics: Welcome to the Visual Age and Capabilities of the 4th Generation Intel Core Processors, 2013. https://intel.activeevents.com/sf13/connect/fileDownload/session/FAECD540A244E206B347FE3A9A7EB7EF/SF13_ARCS001_100.pdf.

[79] ARM Holdings plc. Mali-T600 Series GPU OpenCL Version 1.1.0, 2013. http://infocenter.arm.com/help/topic/com.arm.doc.dui0538e/

DUI0538E_mali_t600_opencl_dg.pdf.

[80] Qualcomm Inc. Mobile Gaming & Graphics Optimization Tools and Resources, 2013. https://developer.qualcomm.com/mobile-development/maximize-hardware/mobile-gaming-graphics-adreno/mobile-gaming-graphics-optimization-tools.

研 究 業 績

1. 学術論文

(1) Keisuke Iwai, Naoki Nishikawa, Takakazu Kurokawa. Acceleration of AES Encryption on CUDA GPU, *International Journal of Networking and Computing*, Vol. 2, No. 1, pp. 131-145, 2012.

(2) Naoki Nishikawa, Keisuke Iwai, Takakazu Kurokawa. High-Performance Symmetric Block Ciphers on Multicore CPU and GPUs, *International Journal of Networking and Computing*, Vol. 2, No. 2, pp. 251-268, 2012.

(3) Naoki Nishikawa, Keisuke Iwai, Takakazu Kurokawa. Power Efficiency Evaluation of Block Ciphers on GPU-Integrated Multicore Processor, In *Proceedings of 12th International Conference on Algorithms and Architectures for Parallel Processing*. Vol. 7439 of *Lecture Notes in Computer Science*, pp. 347-361, 2012.

(4) Naoki Nishikawa, Keisuke Iwai, Hidema Tanaka, Takakazu Kurokawa. Performance Prediction Model for Block Ciphers on GPU Architectures, In *Proceedings of The 7th International Conference on Network and System Security*, Vol. 7873 of *Lecture Notes in Computer Science*, pp. 405-423, 2013.

(5) Keisuke Iwai, Naoki Nishikawa, Takakazu Kurokawa. HiCrypt: A Specialized Translator for Symmetric Block Cipher and GPGPU, *IEICE Transaction on Information and Systems*, Vol.E96-D No.12, pp. 2575-2586, 2013.

(6) Naoki Nishikawa, Keisuke Iwai, Hidema Tanaka, Takakazu Kurokawa. Throughput and Power Efficiency Evaluation of Block Ciphers on Kepler and GCN GPUs using Micro-Benchmark Analysis, *IEICE Transaction on Information and Systems*, In press

(7) 西川 尚紀, 岩井 啓輔, 田中 秀磨, 黒川 恭一. GPU を用いた次世代暗号設計のための性能予測フレームワーク, 電子情報通信学会論文誌, 査読中

2. 国際会議（査読あり）

(1) Keisuke Iwai, Takakazu Kurokawa, Naoki Nishikawa. AES Encryption Implementation on CUDA GPU and its Analysis, In *Proceedings of The 2010 First International Conference on Networking and Computing*, pp. 209-214, 2010.

(2) Naoki Nishikawa, Keisuke Iwai, Takakazu Kurokawa. High-Performance Symmetric Block Ciphers on CUDA, In *Proceedings of The 2011 Second International Conference on Networking and Computing*, pp. 221-227, 2011.

(3) Naoki Nishikawa, Keisuke Iwai, Takakazu Kurokawa. Implementation of Symmetric Block Ciphers using GPGPU, In *Proceedings of 7th International Conference on Information Warfare and Security*, pp. 222-232, 2012.

(4) Keisuke Iwai, Naoki Nishikawa, Takakazu Kurokawa. HiCrypt: C to CUDA Translator for Symmetric Block Ciphers, In *Proceedings of The 2012 Third International Conference on Networking and Computing*, pp. 41-48, 2012.

(5) Naoki Nishikawa, Keisuke Iwai, Hidema Tanaka, Takakazu Kurokawa. Throughput and Power Efficiency Evaluation of Block Ciphers on Kepler and GCN GPUs, In *Proceedings of The First International Symposium on Computing and Networking - Across Practical Development and Theoretical Research*, pp. 366-372, 2013.

3. 国内学会等（口頭発表）

(1) 西川尚紀, 岩井啓輔, 黒川恭一, GPU の汎用計算環境 CUDA による暗号アルゴリズムに対するキークラックの高速化, 電子情報通信学会技術研究報告, Vol. 109 No. 168 (CPSY2009-18), pp. 49-54, 2009.

(2) 西川尚紀, 岩井啓輔, 黒川恭一, GPGPU を用いた暗号攻撃, 情報科学技術フォーラム, Vol. FIT 2009, 講演論文集 第 4 分冊, pp. 149-150, 2009.

(3) 西川尚紀, 岩井啓輔, 黒川恭一, CUDA による AES 実装のための計算粒度最適化手法, 電子情報通信学会技術研究報告, Vol. 109, No. 395 (CPSY2009-68), pp. 107-112, 2010.

(4) 西川尚紀, 岩井啓輔, 黒川恭一, CUDA 環境における共通鍵ブロック暗号の高速実装, 電子情報通信学会技術研究報告, Vol. 111, No. 163 (CPSY2011-13), pp. 25-30, 2011.

(5) 西川尚紀, 岩井啓輔, 黒川恭一, CUDA 実装された共通鍵ブロック暗号のための性能予測モデルの検討, 電子情報通信学会技術研究報告, Vol. 111, No. 398 (CPSY2011-75), pp. 123-128, 2012.

(6) 西川尚紀, 岩井啓輔, 黒川恭一, GPGPU 実装されたブロック暗号のための性能モデルの提案, 電子情報通信学会技術研究報告, Vol. 112, No. 173 (CPSY2012-16), pp. 43-48, 2012.

4. 受賞歴

(1) 情報処理学会 SACSIS 併設企画 GPU Challenge 2009 自由課題部門奨励賞

(2) 情報処理学会第 143 回 SLDM 研究会優秀発表学生賞

(3) 情報処理学会 2010 年度コンピュータサイエンス領域奨励賞

(4) 情報処理学会第 154 回 SLDM 研究会優秀発表学生賞