Cell/B.E. での OpenSSL 暗号処理のオフロードの実装と評価

杉浦 寛† 大釜 正裕† 羅 鏡栄† 齋藤 孝道‡

†明治大学大学院理工学研究科 ‡明治大学理工学部
214-8571 神奈川県川崎市多摩区東三田 1-1-1

{kan_s, ohgama, oscar_weng, saito}@cs.meiji.ac.jp

あらまし 従来のシングルコアプロセッサと比較して，高い処理性能を期待できるマルチコアプロセッサが普及してきている．特に，ヘテロジニアス・マルチコアプロセッサは，一つのプロセッサ上に1基ないし複数の汎用コア，特定の演算に特化したコアを複数搭載し，それに適した処理を行わせることにより，高い処理性能を得ることができる．この特性を活かし，演算用コアに暗号処理の一部をオフロードすることによって暗号処理全体のパフォーマンスを向上させることが期待できる．本論文では，ヘテロジニアス・マルチコアプロセッサの一つである Cell Broadband Engineを用いて，その演算用コアにOpenSSLの暗号処理の一部をオフロードする実装を行い，計測と評価を行う．

An Implementation and Evaluation of Offloading to OpenSSL Encryption Process on Cell/B.E.

Kan SUGIURA† Masahiro OHGAMA† Keang-Weng LO†
Takamichi SAITO‡

†Meiji University Graduate School ‡Meiji Universicty
1-1-1, Higashimita, Tama-ku, Kawasaki-shi, Kanagawa 214-8571, Japan

{kan_s, ohgama, oscar_weng, saito}@cs.meiji.ac.jp

Abstract A multicore processor is expected to have higher processing performance than a single one. Especially, a heterogineus multicore processor consists of any number of mainframe core, for a general operation, and processing ones, for a specific operation. When we exploit a feature of it in the encryption process, a performance of entire processing can be improved to make off-loading a part of process into processing cores. In this paper, we implement to offload OpenSSL encryption process to processing cores, and evaluate it on Cell Broadband Engine.

1 はじめに

一つのプロセッサ上に複数のコアを搭載する技術の発達により，マルチコアプロセッサが普及してきた．マルチコアプロセッサは，複数のコアがそれぞれ他のコアに依存せずに並列に動作し，一つのプロセッサとして動作することによって，プロセッサ全体としての性能向上を図っている．

マルチコアプロセッサのうち，ヘテロジニアス・マルチコアプロセッサは，一般的に，OSなどの制御を受け持つ汎用コアと，画像処理や暗号処理などに特化した演算用コアを持つ．そのため，既存のソフトウェアをそのまま動作させても高速化を望むことはできないが，演算用コアを有効に活用することにより，様々なケース

の処理の高速化が期待できる．そこで，ソフトウェアの開発者が，それぞれのコアの特性を活かした実装を行い，プロセッサの性能を引き出すことが重要となる．

本論文では，OpenSSLをマルチコアプロセッサ上で効率良く動作させることを目的とし，Sony，東芝，IBMの3社によって開発されたヘテロジニアス・マルチコアプロセッサであるCell Broadband Engine (以下Cell/B.E.と呼ぶ) [1][2]を用いて，暗号処理のオフロード [3]を行う．Cell/B.E.は，汎用コアと，高速な処理を実現するための演算用コアをそれぞれ1基以上搭載しており，OpenSSLは汎用コア上では動作できるが，演算用コア上では動作できない．そこで，演算用コアをモジュールとして利用し，OpenSSLの暗号処理の一部をオフロードするよう実装し，暗号処理の高速化を試みた．そのパフォーマンス計測を行い，結果について考察する．

2 Cell Broadband Engine

ここでは，開発環境として使用したPS3 (PLAYSTATION3) に搭載されているCell/B.E.について述べる．以降，本論文では，Cell/B.E.とはPS3に搭載されているCell/B.E.のことを示す．

2.1 Cell/B.E.のアーキテクチャ

Cell/B.E.は，汎用コアとしてPPE (PowerPC Processor Element) を1基，演算用コアとしてSPE (Synergistic Processor Element) を8基，搭載している．ただし，8基のSPEのうち，1基は使用できないように設定[1]されており，1基はハイパーバイザ[2]専用に確保されているので，これら2基は使用することができない．そのため，実質的に使用できるSPEは6基である．また，PPEとSPEはともに3.2GHzで動作しており，PS3自体は256MbyteのRAMを搭載している．Cell/B.E.の主な構成要素 [4]を図1に示す．

- PPE

  PPEとは，プロセッサコア本体であるPPU (PowerPC Processor Unit) に512Kbyteのキャッシュを付属させたものである．PPUは，64bitのPowerPCアーキテクチャを基にした汎用のプロセッサコアであり，既存のPowerPCと互換性がある．

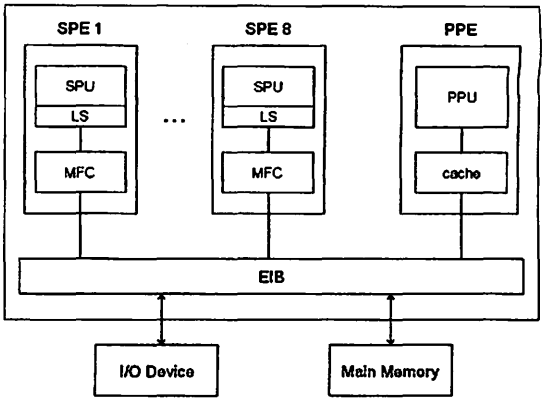

図 1: Cell/B.E.のハードウェアアーキテクチャ

- SPE

  SPEとは，プロセッサコア本体であるSPU (Synergistic Processor Unit) に，後述するLS，MFCを付属させたものである．

  a) SPU

  SPUとは，Cell/B.E.を構成する要素として開発された，演算処理に特化した32bitのプロセッサコアであり，128bitレジスタを複数搭載している．

  b) LS

  LS (Local Storage) とは，各SPEが持つ256Kbyteの専用メモリで，SPEが直接アクセスすることができるのは，このLSだけである．PPEや他のSPEからは，後述のMFCを介することによりアクセスすることができる．

  c) MFC

  MFC (Memory Flow Controller) とは，LSとメインメモリや他のSPEのLSとの間でデータのやりとりを行うためのインタフェースである．SPEがデータ転送を行う際には，MFCが転送命令を発行し，EIBを介して，データ参照先にアクセスを行う．

- EIB

  EIB (Element Interconnect Bus) とは，Cell/B.E.におけるバスで，帯域は204.8 Gbyte/sである．各プロセッサコアは，こ

[1]生産性向上のため，歩留まりを7基としているため

[2]Cell/B.E.は，複数のOSを管理するために，最下層でハイパーバイザと呼ばれるミドルウェアが動作している．

のEIBを介して接続されており，I/Oデバイスやメインメモリもこのバスで接続されている．

2.2 Cell/B.E. 上でのコードの実行

Cell/B.E.は，PPEとSPEとでコードを実行する環境が異なるので，PPE上で実行するコード (以降，PPEコードと呼ぶ) とSPE上で実行するコード (以降，SPEコードと呼ぶ) は別々に用意し，同期をとるためには，コア間で通信を行う必要がある．以下で，PPEとSPEの動作概要について示す．

A) PPE

PPEでは，PowerPCアーキテクチャで動作するコードをそのまま実行することができる．SPEコードを同時に実行する場合には，専用の関数を使用し，PPEコードから実行を指示する．ただし，SPEコードを実行する関数はブロッキング関数なので，並列処理を行うには，スレッドないしプロセスを生成し，そこからSPEコードの実行を指示する．

B) SPE

SPEは，メインメモリに直接アクセスすることができないので，PPEコードがSPE内のLSへとSPEコードを転送し，そこからSPEコードを実行する．SPEコードがメインメモリに格納されているデータを使用する場合や，処理結果をメインメモリに格納する場合は，MFCを介してLSとメインメモリ間でデータのやりとりを行う．

2.3 MFCを用いたデータ転送

MFCを用いたデータ転送として，DMA転送とメールボックスがある．ここでは，これらの機能について概要を示す．

- DMA転送

DMA転送は，LSとメインメモリとの間，及び他のSPEのLSとの間でのデータ転送である．MFCは，DMA転送を行うためのDMAコントローラーを備えており，このDMAコントローラーがDMA転送命令を発行することにより，データ転送を行うことができる．1回のDMA転送で転送できる最大のデータサイズは16Kbyteである．また，DMA転送命令は，1度に複数発行することができ，それらは並列に動作する．

- メールボックス

メールボックスは，あるSPEとPPEないし他のSPEとの通信手段である．各SPEのMFCに，SPEへのデータを受信し，最大4個までのデータを蓄積できるSPU Inbound MailboxとSPEからのデータを送信し，最大1個のデータを蓄積できるSPU Outbound Mailboxが備えられている．入出力はFIFO (First in First Out) キュー構造となっている．また，キューのサイズは32bitとなっており，DMA転送と比較して関数呼び出しのオーバーヘッドが小さいため，状態やパラメータの通知に適している．

3 アプリケーションからのSPEの利用

3.1 概要

Cell/B.E.でのアプリケーションからSPEへのオフロードは，SPEを特定の処理を専用に行うモジュール (以降，SPEモジュールと呼ぶ) として用意し，汎用処理を行うPPEから，特定の処理をSPEモジュールに任せることで実現できる (図2)．ここで，SPEモジュールは，SPEを管理するプロセス (以降，SPE管理プロセスと呼ぶ) と暗号処理を行うSPEモジュール本体から構成される．実際には，SPE管理プロセスをあらかじめ起動しておき，SPEモジュールを利用するアプリケーションからのリクエストを受け付けることで，オフロードを実現する．

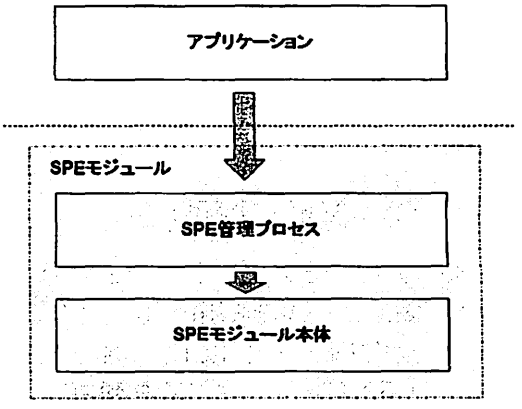

図 2: SPE利用の概要図

しかしながら，一般に，アプリケーション側のコードで，SPEへ処理をオフロードすることは現実的ではなく，OCF (OpenBSD Cryptographic Framework)[5] といったミドルウェアがそれを代替する．

ここでは，OCFを用いて，OpenSSL[6][7]から暗号処理専用モジュールを透過的に利用する方法について示す．OpenSSLの暗号処理の一部を暗号処理専用モジュールにオフロードするため，OpenSSLとOCFを組み合わせた利用基盤を構築する．以下，OpenSSLとOCF，それらからの暗号処理専用モジュールの利用について示す．

3.2 OpenSSLについて

OpenSSLはSSLやTLSだけでなく，公開鍵暗号化方式，共通鍵暗号化方式やハッシュ関数などを容易なインタフェースで利用可能としたAPIライブラリを含むツールキットである．

3.3 OCFについて

OCFは，様々なハードウェアアクセラレータへの透過的なアクセスを提供するAPIとデバイスドライバから構成されたカーネルモジュールである．OCFでは，OpenSSL，OpenSSHやOpenSwanを利用したアプリケーションから，ハードウェアアクセラレータが提供する暗号処理機能を利用するための共通のインタフェースを提供する．

3.4 暗号処理専用モジュール利用の概要

ここでは，OpenSSLとOCFを組み合わせて用いた場合の，暗号処理専用モジュールの利用方法と暗号処理の流れを示す．まず，暗号処理専用モジュールを利用するために必要なアプリケーション側の設定について説明し，次に，OpenSSLとOCF間およびOCF内の処理について説明する．

<u>アプリケーション側の設定:</u>

OpenSSLを用いたアプリケーションでは，暗号処理専用モジュールを利用するために，ENGINE型のオブジェクト（以下，ENGINEオブジェクト）とENGINE API [8]を利用し，ENGINEのセットアップをする必要がある．この設定を行った上で，EVP APIを利用することで，自動的にOpenSSLが暗号処理専用モジュールを利用して暗号処理を行う．ただし，暗号処理専用モジュールが提供しない暗号処理もOCFでエミュレート（カーネル内の暗号APIを利用）することができる．

<u>OpenSSLとOCF間の処理:</u>

OpenSSLからは，キャラクタ型のデバイスノードである `/dev/crypto` (メジャー番号: 10, マイナー番号: 70) を通して，OCFの制御を行う．デバイスノードを利用することで，通常のファイル操作と同様に，`open()`, `read()`, `ioctl()` といった標準的なシステムコールの利用が可能である．

OCFを介して暗号処理専用モジュールを利用する場合は，`ioctl()`を使用し，/dev/cryptoに対して，暗号処理のシステムコールを発行する．これには，主に，`CIOCGSESSION`, `CIOCCRYPT`と`CIOCFSESSION`がある．ただし，これらのリクエストは，あらかじめOpenSSLで定義されている．OpenSSLは，`CIOCGSESSION`を発行することで，OCF間でセッション（以降，OCFセッションと呼ぶ）を確立する．このとき，OpenSSLは，暗号アルゴリズム，ブロック暗号の利用モード，暗号化鍵とを指定する．これらの情報を，OCFセッション情報と呼び，OCFセッションとは，その情報を保持している状態をいう．OCFセッションが確立すると，`CIOCCRYPT`を発行し，セッション情報を元に，暗号処理を行う．そして，暗号処理が終了すると，`CIOCFSESSION`を発行し，OpenSSLとOCF間のセッションを解放する．

4 実装

4.1 Cell/B.E.開発環境

Cell/B.E.での開発を行うにあたって，FedoraCore6 (Kernel2.6.21)と，Cell/B.E.のカーネルパッチ，Cell/B.E.開発者向けのツールキットであるCellSDK 2.1[9]を使用した．実装に用いたコンパイラは，PPU及びSPUをターゲットとしたppu-gcc4.1.1及びspu-gcc4.1.1である．

4.2 オフロード実装について

ここでは，本論文で提案するCell/B.E.のPPE上でのOpenSSLの暗号処理の一部をSPEにオ

フロードする実装 (以降，SPE-Offloadedと呼ぶ) について示す．SPEモジュールを暗号処理専用モジュールとして利用する[3]ため，OpenSSL側で，暗号処理に用いる平文，IV，鍵 (以降，暗号処理データと呼ぶ) を，共有メモリへ格納するように変更した．ここでの共有メモリは，OpenSSLとSPE管理プロセスとのデータ共有のために用いられることに注意されたい．さらには，オリジナルのOpenSSLでのCIOCCRYPT発行の代わりとしてSPE管理プロセスへBSDソケットによる暗号処理リクエストの発行，処理結果を受け取るためのコード変更を行った．その上で，OpenSSLとOCFを組み合わせて利用基盤を構築し，OpenSSLからSPEモジュールへの処理のおオフロードを行った．ただし，利用基盤の構築には，OpenSSL-0.9.8g，OCFの実装であるocf-linux-20071215を用いた．

以下に示す説明文に割り当てた番号は，図3の各番号と対応している．

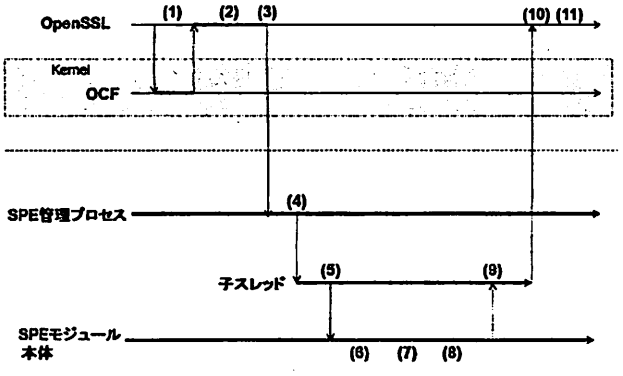

図 3: SPEへの暗号処理のオフロード

(1) OpenSSLとOCF間でOCFセッションを確立する．
(2) 暗号処理に用いるデータを共有メモリへ格納する．
(3) OpenSSLからの暗号処理リクエストとして，SPE管理プロセスへOCFセッション情報，共有メモリのアドレスをBSDソケットにより送信する．
(4) SPEと同期をとるためのスレッドを生成する．
(5) スレッドからSPEへ暗号処理開始，及びセッション情報と共有メモリのアドレスを，メールボックスにより通知する．
(6) SPEは，スレッドから通知を受けると，メインメモリからLSへ暗号処理に用いるデータをDMA転送する．
(7) DMA転送が終了すると，暗号処理を行う．
(8) 暗号処理が終了すると，LSからメインメモリへ暗号処理結果をDMA転送する．
(9) SPEからスレッドへ暗号処理終了を，メールボックスにより通知する．
(10) スレッドは，OpenSSLへ処理結果を返し，スレッドの処理を終了する．
(11) OpenSSLとOCF間のOCFセッションを解放する．

5 評価

ここでは，SPE-Offloadedの評価のため，SPEへ暗号処理のオフロードを行っていないオリジナルのOpenSSLとOCFの組み合わせについても利用基盤を構築 (以降，Original環境と呼ぶ) し，併せてパフォーマンス計測を行う．

5.1 評価項目

SPE-Offloadedの性能評価として，N個のプロセスを起動した時点から，それぞれのプロセスで暗号処理に用いるデータの転送，暗号処理，暗号処理結果のデータ転送を行い，全てのプロセスの処理が終わるまでの時間を`gettimeofday`関数を用いて計測した．ただし，各プロセスが一度に暗号化する平文のデータサイズは256byte，1024byteとした．評価に用いる暗号化方式としてAES[4]のCBCモードを使用した．比較対象であるOriginal環境でも，同様に計測を行った．各データサイズの場合においてそれぞれの処理時間を10回計測し，その平均値を結果とした．

データサイズが256byteの場合:

一度に暗号処理するデータサイズが256byteの場合の計測結果を表1に示す．ただし，表中のOriginalとOffloadは，Original環境，SPE-Offloadedをそれぞれ示している．

表 1: 256byteの場合の処理の総時間

プロセス数 (N)	1	2	3	4	5	6
Original (msec)	0.51	5.16	7.58	9.84	12.83	15.08
Offload (msec)	3.63	9.42	16.99	24.75	29.9	45.83

[3]以降，SPEモジュールを暗号処理専用モジュールとして扱う．

[4]http://xyssl.org/のコードを用いて，SPEに実装した．

計測結果より，いずれの場合も SPE-Offloaded が Original 環境よりもパフォーマンスが低くなることが分かる．

データサイズが 1024byte の場合:

一度に暗号処理するデータサイズが 1024byte の場合の計測結果を表 2 に示す．

表 2: 1024byte の場合の処理の総時間

プロセス数 (N)	1	2	3	4	5	6
Original (msec)	0.59	5.27	6.86	11.16	12.93	16.42
Offload (msec)	3.62	9.72	16.26	25.5	37.86	40.36

計測結果より，データサイズが 256byte の場合と同様にパフォーマンスが低くなることが分かる．

6 考察

ここでは，Original 環境と，SPE-Offloaded の処理時間を比較し，その結果について考察する．

全ての場合において，SPE-Offloaded のほうが，Original 環境よりもパフォーマンスが低くなった．また，データサイズによる SPE-Offloaded と Original 環境のパフォーマンスの差は，ほとんど見られなかった．これらから，暗号処理以外の部分で，遅延が発生していると考えられる．その要因として，プロセス間の通信や SPE との同期処理が挙げられる．

プロセス間の通信について注目すると，OpenSSL からの暗号処理リクエストの発行の際に，Original 環境では，カーネル内へのシステムコールを呼び出しているが，SPE-Offloaded では，OpenSSL と SPE モジュールとの間で BSD ソケットによる通信を行っているため，オーバーヘッドとなっていることが考えられる．SPE との同期について注目すると，複数の SPE 上での暗号処理の終了を待つため，PPE 上の SPE 管理プロセスで，while を使ってそれぞれ待っている．そのため，PPE のリソースが浪費されていることが考えられる．

これらを解決する方法として，BSD ソケットを行わず，SPE 管理プロセスをカーネル空間で実行することや，複数の SPE に対して効率良くスケジューリングを行う [10] ことが挙げられる．

7 まとめと今後の課題

本論文では，オリジナルの OpenSSL と，OpenSSL の暗号処理を SPE にオフロードした場合の処理時間を計測し，評価した．これにより，OpenSSL をマルチコアプロセッサ上で効率良く動作させるための方法と，その課題について示した．

今後として，SSL や IPSec などのセキュリティ用の実システムで利用するために，カーネルモジュールとしての SPE の活用，スケジューラの検討，並列化によるパフォーマンスの向上という課題がある．

謝辞

本研究の成果の一部は，科学研究費補助金(課題番号 19700070) の助成を受けたものである．

参考文献

[1] http://cell.scei.co.jp/index_j.html

[2] http://www-128.ibm.com/developerworks/power/cell/docs_documentation.html

[3] 杉浦寛，大釜正裕，黒羽秀一，齋藤孝道: CBE を用いた AES-CTR モードの並列化実装，暗号と情報セキュリティシンポジウム 2008 概要集，p.216，2008.

[4] http://cell.scei.co.jp/pdf/CBE_Architecture_v101_j.pdf

[5] A. D. Keromytis, J. L. Wright, and T. de Raadt, "The Design of the OpenBSD Cryptographic Framework", In Proceedings of the USENIX Annual Technical Conference, June 2003.

[6] http://www.openssl.org/

[7] John Viega, Matt Messier, Pravir Chandra 共著 齋藤孝道 監訳 "OpenSSL 暗号・PKI/SSL/TLS ライブラリの詳細" オーム社.

[8] http://www.openssl.org/docs/crypto/engine.html

[9] http://www.bsc.es/projects/deepcomputing/ linuxoncell/

[10] 黒羽秀一，齋藤孝道: 暗号モジュールの効率的な利用のためのスケジューラの検討，暗号と情報セキュリティシンポジウム 2008 概要集，p.39，2008.