情報処理学会論文誌 Vol.53 No.2 815–824 (Feb. 2012)

Cell/B.E.における暗号処理の効率的なオフロード方式の提案と実装

齋藤 孝道[1,a)] 杉浦 寛[2]

受付日 2011年5月25日, 採録日 2011年11月7日

概要：プロセッサの性能向上を図る技術として，マルチコアプロセッサ化が採用されている．特に，ヘテロジニアスマルチコアプロセッサは，1つのプロセッサ上に1つないし複数の汎用系プロセッサと，特定の演算に特化した演算系プロセッサを搭載し，それらに適した処理を行わせることにより，高い処理性能を実現している．暗号処理においては，演算系プロセッサに暗号処理の一部をオフロードし，それらを並列に実行することによって，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1つである Cell Broadband Engine を用いて，演算系プロセッサを暗号処理専用のモジュールとして扱うための実装方式を提案し，OpenSSL を利用する複数のアプリケーションが暗号処理を効率良く実行するためのオフロード方式の実装と評価を行う．

キーワード：マルチコア，暗号処理オフロード，暗号ミドルウェア

Proposition and Implementation of an Efficient Offloading Method for Cryptographic on Cell/B.E.

Takamichi Saito[1,a)] Kan Sugiura[2]

Received: May 25, 2011, Accepted: November 7, 2011

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

Keywords: multi-core processor, cryptographic offloading, cryptographic midleware

1. はじめに

プロセッサの性能向上を図る技術として，マルチコア化が採用されている．その中でも，ヘテロジニアスマルチコアプロセッサは，オペレーティングシステムなどの汎用処理を受け持つ汎用系プロセッサと，画像処理などの演算処理に特化した演算系プロセッサを，それぞれ複数搭載している．これにより，高負荷な演算処理に関しては，演算系プロセッサを特定の処理専用のモジュールとして扱い，汎用系プロセッサでの処理の一部を演算系プロセッサにオフロードし並列に実行することで，システム全体のパフォーマンス向上が期待できる．そのような暗号モジュールを持つ CPU，たとえば，暗号処理用のプロセッサを1つだけ搭載する Intel IXP425 [9] では，IPSec，SSL（Secure Socket Layer）[10] や TLS（Transport Layer Security）[11] などにともなう暗号処理の一部を，暗号モジュールにオフロードすることにより，ネットワーク機器として，プロセッサ全体の高速化を期待することができる [20].

一般的な特徴として，暗号処理や暗号通信は，それ自体

[1] 明治大学理工学部
Department of Science and Engineering, Meiji University, Kawasaki, Kanagawa 214–8571, Japan
[2] 新日鉄ソリューションズ株式会社
NS Solutions Corporation, Chuo, Tokyo 104–0033, Japan
[a)] saito@cs.meiji.ac.jp

© 2012 Information Processing Society of Japan

が目的そのものであることばかりではなく，他の処理に付随するケースが多い．様々なケースが想定できるが，一例をあげると，インターネットにおけるショッピングなどのサービスを提供するWebサイトでは，Webサイトへの数多くのアクセスにおいて，商品在庫管理などのDBMSとの連携，いわゆるビジネスロジックに関する処理や，サイトデザインの動的な作成などがあるが，それらに加えて，セキュリティ確保のために暗号処理や暗号通信を行う．よって，汎用系プロセッサから，暗号処理や暗号通信の処理を，より高速なハードウェアでの処理に置き換えることにより，汎用系プロセッサを暗号処理以外の他の処理にあてることができる．すわなち，Webサイトのシステム構成や規模にもよるが，暗号処理のオフロードは，Webサイトの処理全体の高速化への基本的なアプローチである．

暗号処理や暗号通信向けのアプリケーションプログラムでは，暗号ライブラリであるOpenSSL [3], [4] を利用することが標準的で事例も多い．たとえば，Webサーバの実装の1つであるApacheやSSLハードウェアアクセラレータの実装においても利用されている．よって，暗号処理自体の高速化というよりは，OpenSSLを経由した際の処理の高速化が期待されている．しかしながら，OpenSSLは，セキュリティ対策のためのバージョンアップによる改修が行われることが多い．そのため，個別のハードウェアに対応させることを目的に，OpenSSL自体を独自に改修することは好ましくはない．

以上をふまえて，本論文では，ヘテロジニアスマルチコアプロセッサの1つであるCell Broadband Engine [1], [2]（以降，Cell/B.E.と呼ぶ）において，カーネルおよびOpenSSLを改修せずに，OpenSSLに対して透過的に暗号処理のオフロードを可能とすることを課題とした．ただし，暗号処理のオフロードにおいて，アプリケーションプログラムからOpenSSL ENGINE API（以降，単にENGINE APIと呼ぶ）経由でリクエストが同時に多数発生するケースも想定し，ENGINE APIからのリクエストをスケジュールすることもあわせて考慮する．特に，ディジタル家電におけるストリーム処理とは違い，本論文で想定する暗号処理の継続性，すなわち，1度にどれくらいのデータを暗号化・復号するのかは予測が困難であるため，その制約のもとで検討することとする．

上述の課題に対し，OpenSSLの標準機能であるENGINE APIから演算系プロセッサを利用するアプローチを採用した．そのため，提案システムにおいて，演算系プロセッサを暗号処理専用のモジュールとして扱い，複数のアプリケーションにおけるOpenSSLを用いた暗号処理の一部をオフロードするための共有ライブラリを新たに実装した．また，暗号処理のオフロードのリクエストが同時に多数発生するケースに対し，ENGINE APIからのリクエストをスケジュールするプロセススケジューラを，カーネルの改修をせずに実現することを目指した．特に，比較対象としている既存技術のMars [14] とはアプローチを変え，暗号処理を行うアプリケーションのプロセスだけをスケジュール対象とし，影響の範囲を限定したことが新たな特徴の1つである．

2. 関連研究

論文 [15] は，Cell/B.E.において，暗号処理をともなうアプリケーションプログラムの解析を試みる攻撃者から，暗号鍵といった機密データを守ることを目的とした開発基盤の提案である．これは，暗号処理の高速化を目的とした我々の提案の目的とは違っている．

論文 [16] の主な目的は，あるサーバから別途用意されたCell/B.E.を搭載するサーバへ，暗号処理とは限らないが処理をオフロードすることである．これは，我々の提案の適用範囲である，汎用系プロセッサから演算系プロセッサへのオフロードとは違っている．また，我々の提案では，その上位層に存在するOpenSSLから透過的に利用できることも目的の1つである．

論文 [17] は，Cell/B.E.上でのLinuxカーネルにおける汎用タスク管理法の提案である．我々の提案では暗号処理を扱うタスクに特化している一方で，暗号化・復号やそれにともなうデータ転送があり，扱う対象範囲が違っている．実際，当該論文では，空タスクの起動から終了までの時間のみを評価している．もちろん，当該論文の成果を本論文の提案と組み合わせることも可能であろうが，当該論文のアプローチを採用する場合，Linuxカーネル自体の改修をともなう点でも，我々の提案のそれとは違っている．

論文 [18] では，ディジタル家電におけるH.264などのコーデック処理を対象として，Cell/B.E.におけるスレッド実行環境に関するプログラミングモデルとスケジューリング技術についてを紹介している．我々の提案とは，処理の継続性が予測できない暗号処理を対象としている点で差異があるといえる．我々の提案ではスケジューラが対象とするプロセスを限定し，汎用スケジューラに手を加えず，OpenSSLという暗号ライブラリから透過的に利用できることを目指している点でも違う．

論文 [19] では，ネットワークプロセッサであるIntel IXP425を対象とし，暗号処理の効率的なオフロード方式の提案と評価を行っている．我々が今回対象としたCell/B.E.とは違い，Intel IXP425には暗号処理専用のモジュールが1つだけ内蔵されており，対象とするアーキテクチャが大きく違っている．オフロード処理もつねに一元的に扱える点でも違う．

マルチコアプロセッサでかつ複数のハードウェア暗号モジュールを持つものとして，Oracle社のSPARC Txシリーズの CPUがある．また，それらのCPU向けに，Oracle社が提供するSolaris Cryptographic Frameworkの `PKCS#11`

© 2012 Information Processing Society of Japan

という本論文と類似の目的を持つ技術 [21] があるが，ソースコードおよび内部の詳細な構造は，現在，非公開であるため，本論文での議論からは省く．

3. Cell Broadband Engine

ここでは，開発環境として使用した PLAYSTATION3（以降，PS3 と呼ぶ）に搭載されている Cell/B.E. について示す．

3.1 Cell/B.E. のアーキテクチャ

Cell/B.E. の主な構成要素 [5] を**図 1** に示す．Cell/B.E. は，汎用系プロセッサとして PPE（PowerPC Processor Element）を 1 つ，演算系プロセッサとして SPE（Synergistic Processor Element）を 8 つ搭載している．ただし，8 つの SPE のうち，1 つは使用できないように設定されており，また，もう 1 つはハイパーバイザ専用に確保されているので，それら 2 つは使用することができない．そのため，実質的にユーザが利用できる SPE は 6 つである．

また，Pham ら [6] や Williams ら [7] によると，Cell/B.E. の単精度の浮動小数点演算のピーク性能は 204.8 GFlops，メモリアクセス速度は 25.6 Gbytes/s というように，単一プロセッサとして非常に高いパフォーマンスを得られることが報告されている．

3.1.1 PPE

PPE は，プロセッサ本体である PPU（PowerPC Processor Unit）と PPSS（PowerPC Processor Storage Subsystem）を搭載した汎用系プロセッサである．この中で，メインメモリや外部デバイスへの入出力制御，SPE の制御を行っている．

PPU は，64 bit の PowerPC と同等の機能を有した汎用系プロセッサであり，既存の OS やアプリケーションを実行できる．PPSS は，PPU からメインメモリへのデータアクセスを制御するユニットであり，512 Kbytes の L2 キャッシュを搭載している．

3.1.2 SPE

SPE は，プロセッサ本体である SPU（Synergistic Processor Unit），256 Kbytes の専用メモリである LS（Local Storage）と，外部とのデータのやりとりを行うためのインタフェースである MFC（Memory Flow Controller）を搭載した演算系プロセッサである．

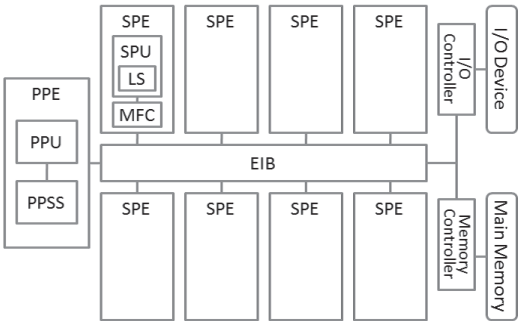

図 1 Cell/B.E. のハードウェアアーキテクチャ
Fig. 1 Architecture of Cell/B.E.

SPU は，演算処理に特化した 32 bit のプロセッサであり，演算能力向上のために高性能な浮動小数点ユニットや，128 bit のレジスタを 128 個搭載している．また，SPE が直接アクセスできるのは LS だけであり，SPE がメインメモリのデータにアクセスする場合や，PPE や他の SPE から LS にアクセスする場合には，MFC を介したデータ転送を行う．

3.1.3 EIB

EIB（Element Interconnect Bus）は，Cell/B.E. におけるバスであり，250 Gbytes/s 以上の帯域を持っている．各プロセッサは直接 EIB に接続されており，I/O デバイスやメインメモリなどは，それぞれ専用のコントローラを介して，EIB に接続されている．

3.2 アプリケーションからの SPE の利用

Cell/B.E. でのアプリケーション開発 [8] において，前述のとおり，PowerPC で動作する既存のアプリケーションを，PPE ではそのまま動作させることができるが，SPE では動作させることはできない．PPE で動作するアプリケーションから SPE を利用するためには，たとえば，SPE を演算処理専用のモジュールとして扱い，PPE の処理の一部を SPE で動作するプログラムで実行するようにアプリケーションを変更する必要がある．

3.3 SPE でのプログラム実行

SPE でプログラムを実行するには，PPE で動作するアプリケーションが専用のライブラリを用いて SPE へプログラムを転送し，実行を指示する．また，SPE はメインメモリに直接アクセスすることができないため，PPE と SPE との間でデータ共有や同期を行う場合には，MFC の提供するデータ転送機能である DMA 転送やメールボックスを利用する．加えて，SPE での処理を高速に実行するためには，SIMD 演算やレジスタの活用など，SPE のアーキテクチャに基づく実装を行う必要がある．ここで，SPE では 1 度に単一のプログラムしか実行できないため，複数のアプリケーションが同時に SPE を利用する場合には，SPE の動作を制御し，SPE とアプリケーションを対応させる仕組みが必要となる．

4. OpenSSL

ここでは，OpenSSL についての詳細と，OpenSSL からの暗号モジュールの利用について示す．

© 2012 Information Processing Society of Japan

4.1 OpenSSL の概要

OpenSSL は SSL/TLS だけでなく，共通鍵暗号化方式や公開鍵暗号化方式，証明書の発行といった PKI 関連の処理などを容易なインタフェースで利用可能とした API やライブラリを含むツールキットである．OpenSSL が提供するライブラリには libcrypto と libssl があり，前者は暗号技術を，後者は SSL/TLS 通信を提供するライブラリである．libcrypto は共通鍵暗号化方式として DES，3DES や AES など，公開鍵暗号化方式として RSA，DH（Deffie-Hellman）など，ハッシュ関数として SHA-1 や MD5 など，主要なアルゴリズムが利用可能であり，それぞれの方式に対して，共通のインタフェースでの利用を可能とする EVP API を提供している．

OpenSSL は，暗号処理を専用に行う（CPU 外部に接続する）ハードウェア暗号モジュールに一部対応しており，アプリケーションからそれらを利用するためのインタフェースとして ENGINE API [12] を提供している．ハードウェア暗号モジュールのベンダからライブラリもしくはドライバを提供されている場合，ENGINE API は，そのライブラリもしくはドライバを呼び出すことで，暗号処理をオフロードすることが可能である．ただし，Cell/B.E. における SPE には対応しておらず，独自に，ライブラリもしくはドライバを用意する必要がある．

また，OpenSSL は，アプリケーションからの関数呼び出しだけでなく，コマンドラインから実行できる様々なコマンドを提供している．その中でも，speed コマンドは，OpenSSL が利用可能な暗号アルゴリズムのスループットを計測するためのコマンドであり，実行システムごとの性能評価などに利用可能である．

4.2 OpenSSL での暗号処理

OpenSSL を用いた暗号処理には，主に EVP API と EVP 型のオブジェクト（以降，EVP オブジェクトと呼ぶ）を利用する．EVP API は，暗号処理の初期化処理，実行や，終了処理を行う関数などから構成される．EVP オブジェクトは，鍵長や処理サイズなどのパラメータや，暗号処理の初期化処理や実行をするための関数のポインタが登録され，アルゴリズムごとにそれぞれ専用のものが用意されている．EVP API を用いた暗号処理の実行例を**図 2** に示す．

4 行目で `EVP_EncryptInit` 関数を呼び出し，暗号処理の初期化処理を行う．第 2 引数には，アルゴリズムに対応した EVP オブジェクトを指定している．ここでは，AES の 128 bit，ECB モードでの暗号処理を示している．5 行目で `EVP_EncryptUpdate` 関数を呼び出し，暗号処理を実行する．引数には，平文が格納されている配列と暗号文の格納先の配列，さらにそれらの長さを保持する変数を指定している．6 行目では，`EVP_EncryptFinal` 関数を呼び出し，パディング処理を行う．8 行目では，`EVP_CIPHER_CTX_cleanup` 関数を呼び出すことで，暗号処理の終了処理を行う．

```
1  /* EVP Encrypt */
2  EVP_CIPHER_CTX ctx;
3
4  EVP_EncrytpInit(&ctx, EVP_aes_128_ecb(),key,iv);
5  EVP_EncryptUpdate(&ctx, out, &out_len, in, len);
6  EVP_EcnryptFinal(&ctx, out+out_len, &len);
7
8  EVP_CIPHER_CTX_cleanup(&ctx);
```

図 2 EVP API を用いた暗号処理の実行例
Fig. 2 Example of EVP API.

```
1  /* set up an engine */
2  ENGINE *e;
3  ENGINE_load_builtin_engines()
4  if(!(e=ENGINE_by_id("hwa"))){
5  /* error processing */
6  }
7  else if(!ENGINE_set_default(e,
8           ENGINE_METHOD_ALL)){
9  /* error processing */
10 }
```

図 3 暗号モジュールへのオフロードの手続き
Fig. 3 Procedures of offloading to cryptography module.

4.3 OpenSSL からのハードウェア暗号モジュールの利用

OpenSSL を用いたアプリケーションでは，EVP API を用いることで暗号処理を実行するが，（CPU 外部に接続する）ハードウェア暗号モジュールを利用して暗号処理を実行するには，EVP API に加えて，ENGINE 型のオブジェクト（以降，ENGINE オブジェクトと呼ぶ）と ENGINE API を利用する．ここで，ENGINE オブジェクトは，ハードウェア暗号モジュールの識別子や，ハードウェア暗号モジュールの初期化を行う関数などを登録する構造体であり，ENGINE API の提供する関数を用いることで，ハードウェア暗号モジュールごとに生成される．また，ENGINE オブジェクトに登録するパラメータや関数は，ハードウェア暗号モジュールに対応するライブラリに用意されている．

提案システムでは，この（CPU 外部に接続する）ハードウェア暗号モジュールを利用する機能を活用して，後述する（共有ライブラリである）暗号処理オフロードライブラリを呼び出し，OpenSSL からの暗号処理をオフロードする仕組みを採用した．具体的には，たとえば，暗号処理オフロードライブラリのファイル名を `libcell.so` とし，**図 3** の 4 行目の `"hwa"` を，`"cell"` とすることにより，暗号処理が SPE で実行されるように実装した．

4.4 暗号モジュール利用の手続き

OpenSSL から暗号モジュールへオフロードするための手続きを図 3 に示す．

3 行目で `ENGINE_load_builtin_engines` 関数を呼び出

© 2012 Information Processing Society of Japan

すと，この関数内で，OpenSSL が利用可能な暗号モジュールに対応する ENGINE オブジェクトを生成し，それらを ENGINE オブジェクト専用のリスト（以降，ENGINE リストと呼ぶ）に登録する．4 行目では，暗号モジュールの識別子を引数として指定して `ENGINE_by_id` 関数を呼び出し，引数に対応した ENGINE オブジェクトを ENGINE リストから取得する．ここでは，引数に指定した `"hwa"` に対応する ENGINE オブジェクトを取得している．7，8 行目では，`ENGINE_set_default` 関数を呼び出すことで，暗号モジュールが対応している暗号アルゴリズムを ENGINE オブジェクトに登録する．

5. 提案システム

5.1 概要

提案システムは，OpenSSL から SPE を利用するためのライブラリ（以降，暗号処理オフロードライブラリと呼ぶ）と，暗号処理を実行するプロセス（以降，暗号処理プロセスと呼ぶ）に対して SPE を割り当てるスケジューラ（以降，プロセススケジューラと呼ぶ）の 2 つから構成される（**図 4** 参照）．本論文では，これら 2 つを新たに実装した．以降で，これら 2 つについて説明する．

5.2 実装

5.2.1 暗号処理オフロードライブラリ

今回新たに実装した暗号処理オフロードライブラリは，PPE 側で主体となって動作する関数群（以降，PPE モジュールと呼ぶ）と，SPE 側で主体となって動作する関数群（以降，SPE モジュールと呼ぶ）により構成されている．PPE モジュールには，OpenSSL から SPE を呼び出すための ENGINE API をサポートする関数や，SPE を起動するための関数があり，SPE モジュールには，暗号処理の実行や，LS とメインメモリ間のデータ転送を行う関数がある．ここで，SPE モジュールでの暗号処理には，IBM の提供する SPE で暗号処理を実行するためのライブラリである libspucrypto [13] を利用している[*1]．

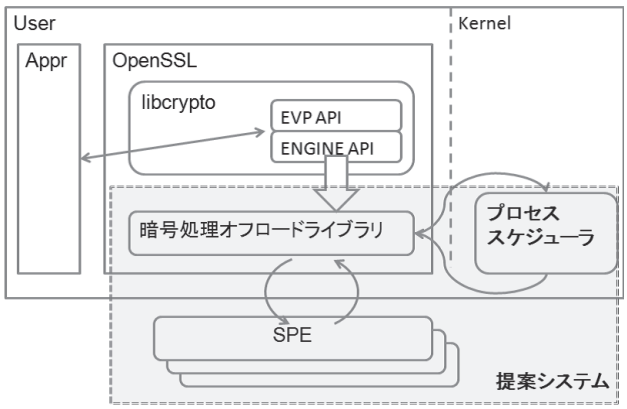

図 4 提案システムの全体像
Fig. 4 Overall of the proposed system.

暗号処理オフロードライブラリでは，ENGINE API を最初に利用した暗号処理プロセスが ENGINE オブジェクトを生成し，SPE をすべて起動する．SPE が動作している間は，ENGINE オブジェクトが暗号処理プロセスに保持されるため，以降，暗号処理プロセスは，ENGINE オブジェクトを取得するだけで，SPE を利用できる．

以下に暗号処理オフロードライブラリの動作概要を**図 5** の各番号と対応させながら示す．ただし，説明の簡略化のため，SPE は 1 つのみを利用するケースを表記している．また，プロセススケジューラの動作については後述する．

(1) 暗号処理プロセスが，ENGINE API を用いて，暗号モジュールとして SPE を利用するための ENGINE オブジェクトを生成し，PPE モジュールへ SPE の起動を指示する．PPE モジュールでは，SPE へプログラムをロードし，SPE を起動する．ただし，すでに (1) の処理が行われて SPE が起動している場合には，この処理は行わない．
(2) 暗号処理プロセスが暗号処理の初期化処理のために EVP API を呼び出す．EVP API では，ENGINE オブジェクトに登録されている関数を参照し，PPE モジュールへ処理を移す．
(3) PPE モジュールがプロセススケジューラを呼び出して，SPE を取得する．
(4) PPE モジュールは，SPE を取得すると，メールボックスにより SPE モジュールへ初期化処理を通知する．
(5) SPE モジュールでは，暗号処理に用いるラウンド鍵などを生成する．
(6) 暗号処理プロセスが暗号処理を実行するために EVP API を呼び出す．EVP API では，EVP オブジェクトに登録されている関数を参照し，PPE モジュールへ処理を移す．
(7) PPE モジュールが SPE モジュールへメールボックスにより暗号処理の実行を通知する．
(8) 暗号処理プロセスで指定した平文や，暗号鍵などのデータをメインメモリから LS へ DMA 転送し，SPE モジュールは暗号処理を実行する．
(9) SPE モジュールは，暗号処理が終了すると，処理結果を LS からメインメモリへ DMA 転送した後，PPE モジュールへメールボックスにより暗号処理の終了を通知する．
(10) PPE モジュールは，SPE モジュールからの暗号処理終了の通知を受け取ると，暗号処理プロセスへ暗号処理の結果を返す．
(11) 暗号処理プロセスが暗号処理の終了処理のために EVP API を呼び出す．EVP API では，EVP オブジェクト

[*1] OpenSSL をもとに，SPE のアーキテクチャに適した実装がなされており，SPE からの呼び出しにしか対応していない．

© 2012 Information Processing Society of Japan

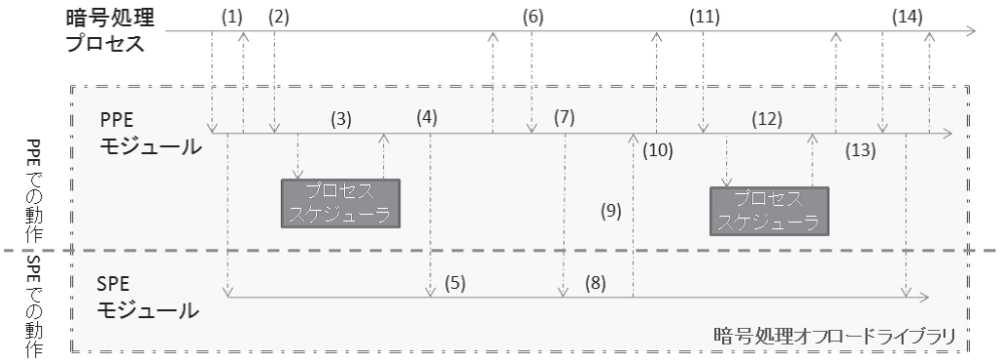

図 5 暗号処理オフロードライブラリの動作
Fig. 5 Working of the offload library.

に登録されている関数を参照し，PPE モジュールへ処理を移す．

(12) PPE モジュールがプロセススケジューラを呼び出して，SPE を解放する．

(13) 暗号処理プロセスへ処理結果を返す．

(14) 暗号処理プロセスでは暗号処理が終了すると，PPE モジュールを呼び出す．PPE モジュールでは SPE モジュールの実行ファイルを破棄し，SPE を終了する．ただし，他のプロセスが SPE モジュールを利用している場合は，この処理は行わない．

5.2.2 プロセススケジューラ

ここでは，カーネルモジュールとして，新たに実装し導入したプロセススケジューラについて示す．

プロセススケジューラの目的は，複数の暗号処理プロセスが，暗号処理オフロードライブラリ経由で SPE へ処理をオフロードする際，SPE をリソースとして取得することと，解放することを透過的に実現することである．ここで，当該プロセススケジューラは，暗号処理プロセスのみを対象としており，OS が管理する他のプロセスには影響がないことに注意されたい．

プロセススケジューラ呼び出しは，キャラクタ型デバイスノードとして用意した `/dev/cell_mod` を通して `ioctl` システムコールにより呼び出される．その際，システムコールの引数により，後述するとおり，SPE の取得または解放を区別し，それぞれの処理を行う．よって，`/dev/cell_mod` と，SPE の取得または解放の識別子を引数として，`ioctl` システムコールを呼び出す．

また，プロセススケジューラの内部においては，下記 2 つのデータ構造を用意した．

(a) SPE の状態管理を目的した要素数 6 の配列（以降，SPE 動作フラグと呼ぶ）：SPE に 1 対 1 で対応しており，これらの値により，SPE が使用中か否かをそれぞれ管理する．SPE 動作フラグの値には，定数 WORK と定数 IDLE があり，WORK は SPE が使用中であることを，IDLE は SPE が未使用であることを，それぞれ表す．

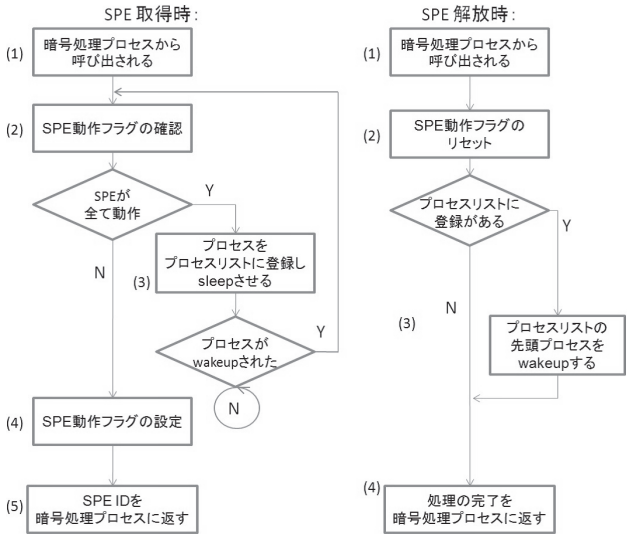

図 6 プロセススケジューラの動作
Fig. 6 Working of the process scheduler.

(b) 暗号処理プロセス管理を目的とした双方向線形リスト（以降，プロセスリストと呼ぶ）：これにより，SPE の利用を要求した暗号処理プロセスから順に，SPE の割当てを行う．後述のとおり，SPE 6 つがすべて利用中の場合，当該要求を出したプロセス ID に対応するリスト作成し，それをリストの最後尾につなげる．

以下では，プロセススケジューラの動作例を示す．ただし，ここでの説明は**図 6** に示すフローチャートと対応している．

SPE 取得時：

(1) プロセススケジューラが，SPE 取得目的で呼び出される．

(2) SPE 動作フラグを確認し，いずれかの SPE が使用可能である場合，(4) の処理に進む．

(3) 呼び出し元の暗号処理プロセスのプロセス ID をプロセスリストに登録（すでに登録されている場合は二重に登録しない）し，当該プロセスを sleep させる．暗号処理プロセスが wakeup されたら，(2) の処理に戻る．

© 2012 Information Processing Society of Japan

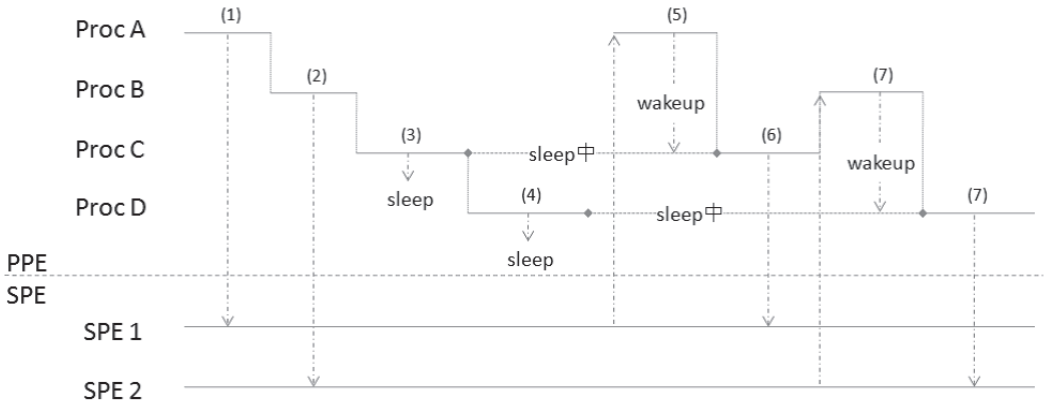

図 7 提案システムの動作例
Fig. 7 Working example of the proposed system.

(4) SPE 動作フラグが IDLE である SPE を特定し，当該 SPE ID を返り値として設定し，当該 SPE 動作フラグを WORK とする．
(5) セットされた SPE ID を暗号処理オフロードライブラリに返す．

SPE 解放時：

(1) プロセススケジューラが，SPE 解放目的で呼び出される．
(2) SPE ID により特定される SPE の動作フラグを IDLE とする．
(3) プロセスリストを確認し，sleep されているプロセスがある場合は，プロセスリストの先頭のプロセス ID に対応する暗号処理プロセスをプロセスリストから削除し，wakeup する．
(4) 処理の完了を暗号処理オフロードライブラリに返す．

5.3 提案システムの動作例

ここでは，提案システムの動作例として，暗号処理プロセスであるプロセス A，B，C，D が順に暗号処理を実行する場合について，**図 7** を用いて説明する．ただし，SPE は，説明の都合上，2 つのみ利用可能と仮定することとする．ここで，プロセス A，B，C，D は，図 7 の `Proc A`，`B`，`C`，`D` と，それぞれ対応する．また，以下の説明は図 7 の各番号と対応している．

(1) プロセス A が暗号処理を開始し，プロセススケジューラが呼び出される．プロセススケジューラは，プロセスリスト内の SPE 1 の SPE 動作フラグを WORK にし，SPE 1 の SPE ID をプロセス A に返す．これにより，プロセス A は，SPE 1 に暗号処理を実行させる．
(2) (1) と同様に，プロセス B が SPE 2 に暗号処理を実行させる．
(3) プロセス C が暗号処理を開始し，プロセススケジューラが呼び出される．プロセススケジューラは，すべての SPE が動作中のため，プロセス C のプロセス ID をプロセスリストに登録して，プロセス C を sleep させる．
(4) (3) と同様に，プロセス D を sleep させる．
(5) SPE 1 での処理が終了すると，プロセススケジューラが呼び出される．プロセススケジューラは，プロセスリスト内の SPE 1 の SPE 動作フラグを IDLE にする．さらに，プロセスリストの先頭に登録されているプロセス C のプロセス ID をプロセスリストから削除し，wakeup する．
(6) プロセス C によりプロセススケジューラが再度呼び出される．プロセススケジューラは，SPE 動作フラグが IDLE となっている SPE 1 の SPE ID をプロセス C に返す．プロセス C は，SPE 1 に暗号処理を実行させる．
(7) プロセス B とプロセス D の間でも (5)，(6) と同様の処理が行われ，プロセス D が，SPE 2 に暗号処理を実行させる．

6. 評価

6.1 評価環境

提案システムの評価を行うにあたり，Cell/B.E. 環境に対して比較対象とするため，Core2Duo 環境をハードウェアとして用意した．Cell/B.E. 環境として使用した PS3 は，3.2 GHz で動作する Cell/B.E.，256 Mbytes の RAM と，60 Gbytes の HDD を搭載する．開発環境として Fedora9（Kernel2.6.25-14）に Cell/B.E. 用のカーネルパッチをあてたものと，Cell/B.E. 開発者向けのツールキットである CellSDK 3.1 を使用した．コンパイラは，PPU および SPU をターゲットとした ppu-gcc4.1.1 および spu-gcc4.1.1 をそれぞれ使用した．Core2Duo 環境は，Core2Duo 3.16 GHz と 3 Gbytes の RAM を搭載したマシンである．開発環境として Fedora9（Kernel2.6.26.3）を使用した．また，OpenSSL のバージョンは，いずれの環境でも，0.9.8g である．

6.2 評価項目

提案システムの性能評価として，**表 1** に示す項目それぞ

© 2012 Information Processing Society of Japan

表 1 評価項目一覧
Table 1 Evaluation item list.

名称	詳細
提案システム	Cell/B.E. 環境で提案システムを用いて計測
Mars	Cell/B.E. 環境で Mars を用いて計測
PPE	Cell/B.E. 環境で PPE のみを用いて計測
Core2Duo	Core2Duo 環境で計測

表 2 AES-ECB モード，プロセス数 1 つの場合のスループット
Table 2 Throughput of a single process, AES-ECB.

データサイズ (bytes)	16	64	256	1,024	8,192
提案システム (Mbps)	9.71	38.53	152.39	490.68	971.2
Mars (Mbps)	8.34	16.11	56.29	178.08	315.82
PPE (Mbps)	295.9	337.17	353.6	357.97	358.2
Core2Duo (Mbps)	1,091.2	1,197.6	1,230.9	1,240.8	1,233.4

れについて，OpenSSL の speed コマンドを実行し，計測を行った．

表 1 中で，「提案システム」と「Mars」では，Cell/B.E. での SPE を用いた場合の性能評価を行っており，「PPE」と「Core2Duo」では，汎用プロセッサを用いた場合の性能評価を行っている．ここで，Mars [14] とは，既存のマルチコア環境で演算系プロセッサを効率良く利用するためのフレームワークであり，Cell/B.E. に対応している並列化ライブラリのソフトウェアである．Mars とともに OpenSSL と Mars を組み合わせたパフォーマンス評価のためのライブラリがリリースされており，これを提案システムと同じ環境下で実行する．

これらの項目について，暗号処理プロセスの数（以降，プロセス数と呼ぶ）が 1 つの場合と 6 つの場合で計測を行った．計測に用いた暗号アルゴリズムは，AES-128 bit の ECB モードと CBC モードであり，実行時には，プログラム実行の実時間を計測した．また，すべての計測は 5 回行い平均を結果とした．

6.3 評価

プロセス数が 1 つの場合の計測結果を**表 2** および**表 3** に，プロセス数が 6 つの場合の計測結果を**表 4** および**表 5** にそれぞれ示す．ここで，プロセス数が 6 つの場合は提案システムと Mars では 6 つの SPE で同時に演算する．また，Core2Duo では，プロセス数が 2 つの場合で，2 つのプロセスが同時に演算が可能であることに注意されたい．また，表中のデータサイズとは，1 度に暗号処理を行うデータサイズ[*2]を示しており，評価項目は，表 1 の名称に対応している．

計測結果より，ほぼすべての項目で，Mars の性能を上回った．よって，我々の提案方式の有用性が確認できたといえる．

[*2] ここで示すデータサイズを入力として，暗号処理を実行する．

表 3 AES-CBC モード，プロセス数 1 つの場合のスループット
Table 3 Throughput of a single process, AES-CBC.

データサイズ (bytes)	16	64	256	1,024	8,192
提案システム (Mbps)	9.94	39.47	68.62	285.72	534.11
Mars (Mbps)	8.37	14.03	56.15	195.28	298.16
PPE (Mbps)	184.79	209.09	216.89	218.86	219.23
Core2Duo (Mbps)	864.06	1,204.6	1,329.1	1,333.8	1,377.1

表 4 AES-ECB モード，プロセス数 6 つの場合のスループット
Table 4 Throughput of six processes, AES-ECB.

データサイズ (bytes)	16	64	256	1,024	8,192
提案システム (Mbps)	12.82	50.87	200.92	680.39	3,019.2
Mars (Mbps)	7.89	32.35	130.08	924.59	1,861.17
PPE (Mbps)	303.3	354.7	377.7	383.6	396.2
Core2Duo (Mbps)	2,231.2	2,451.4	2,509.8	2,545.7	2,593.8

表 5 AES-CBC モード，プロセス数 6 つの場合のスループット
Table 5 Throughput of six processes, AES-CBC.

データサイズ (bytes)	16	64	256	1,024	8,192
提案システム (Mbps)	12.62	50.06	197.45	655.75	2,282.47
Mars (Mbps)	8.17	33.0	130.87	848.43	1,761.78
PPE (Mbps)	231.3	273.81	285.13	286.92	294.2
Core2Duo (Mbps)	1,757.1	2,411.5	2,660.5	2,667.4	2,799.0

Mars および我々の提案システムでは，データサイズが 256 bytes もしくは 1,024 bytes 以下の場合で，PPE にスループットが劣った．我々の実装の場合，オフロード開始時，カーネル空間内に平文データを保存するメモリ領域をつど確保することやデータ転送があり，それらがオーバヘッドとなっていることなどが原因であると推測される．暗号処理に占める割合よりもその他の処理の割合が大きい場合，一般的なオフロードにおいても見受けられるケースであるが，オフロードは推奨されない．

次に，プロセス数とのパフォーマンスの関係について確認するため，提案システムにおいて，プロセス数を 1 つから 6 つまで変化させながら，データサイズごとに暗号処理（AES-ECB モード）のスループットを計測した（**図 8**）．図 8 の縦軸は暗号処理のスループットを，横軸はプロセス数を，それぞれ示している．図 8 に示すとおり，データサイズが 8,192 bytes の場合は線形的に値が向上していくが，データサイズが 1,024 bytes 以下の場合は，プロセス数が 2 つ以上に増加しても，スループットの向上がわずかだった．実際，表 2 および表 4 により確認してみると，8,192 bytes のとき，プロセス数が 6 のスループットはプロセス数が 1 のそれに比べて 310%の向上が見られたが，16 bytes のときには 30%程度の向上であった．オフロードするデータのサイズが大きく，プロセス数を 6 つ利用した場合，提案システムは効率が最も良いことが分かる．

文献 [22] によれば，Web サイトでやりとりされるデータの平均的なサイズは約 50 Kbytes である．また，暗号

© 2012 Information Processing Society of Japan

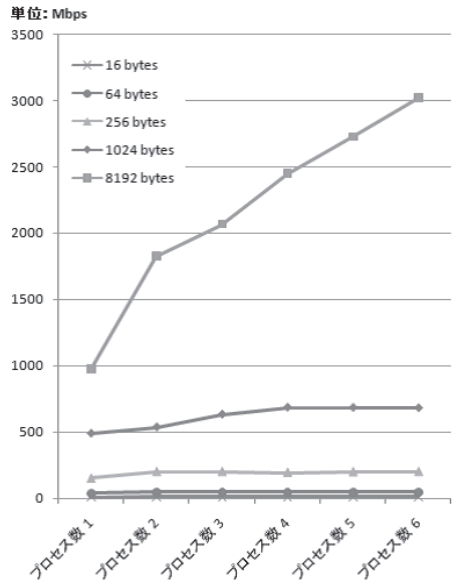

図 8 プロセス数を変化させた場合の提案システムのスループット
Fig. 8 Throughput of the proposed system when the number of processes.

化・復号処理の単位となる SSL レコードサイズが，最大約 2^{14} bytes [*3] である．たとえば，SSL-Web サーバへの適用を考えた場合，その際，比較的大きいサイズで，暗号化・復号が行われると判断できる．また，Web サーバの 1 つである Apache において，worker モードの利用の場合はスレッド単位で，prefork モードでの利用の場合はプロセス単位で，SPU を利用すれば，大量のアクセスにおける暗号処理のオフロードに適用できる．よって，一例を示しただけではあるが，提案方式が一定の有効性を持つことが分かる．

7. まとめと今後の課題

本論文では，Cell/B.E. で OpenSSL の暗号処理を SPE にオフロードするための一方式を提案した．また，提案方式について実装を行い，そのパフォーマンスを計測した．これにより，既存技術である並列化ライブラリ Mars と比べて，高速な演算ができることを示した．

今後の課題として，提案方式および既存方式における詳細な分析を行ったうえでのボトルネックの発見，分析やその解消がある．また，様々な暗号アルゴリズムへの拡張や，より効果的なスケジューラの実装なども課題である．

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．また，本論文を作成にするにあたり協力いただいた村上智祐氏に感謝する．最後に，本論文の査読過程において，有益なコメントをいただいた匿名の査読者と編集委員にも感謝する．

[*3] 実装によって正確なサイズは多少異なる [23] ため，「約」と表現した．

参考文献

[1] available from ⟨http://cell.scei.co.jp/index_j.html⟩.
[2] available from ⟨http://www-128.ibm.com/developerworks/power/cell/docs_documentation.html⟩.
[3] available from ⟨http://www.openssl.org/⟩.
[4] Viega, J. et al.：OpenSSL 暗号・PKI/SSL/TLS ライブラリの詳細，オーム社 (2004).
[5] available from ⟨http://cell.scei.co.jp/pdf/CBE_Architecture_v101_j.pdf⟩.
[6] Pham, D. et al.: The Design and Implementation of a FirstGeneration Cell Processor, *IEEE Internationl SolidState Circuits Symposium*, pp.184–186 (2005).
[7] Williams, S. et al.: The Potential of the Cell Processor for Scientific Computing, *the ACM International Conference on Computing Frontiers* (2006).
[8] Scarpino, M.: *Programming the Cell Processor*, Prentice Hall PTR.
[9] Barry, P. et al.: *Designing Embedded Networking Applications*, Intel Press (2005).
[10] Freier, A. et al.: The SSL Protocol Version 3.0 draft (Mar. 1996).
[11] Dierks, T. et al.: RFC2246: The TLS Protocol Version 1.0 (Jan. 1999).
[12] available from ⟨http://www.openssl.org/docs/crypto/engine.html⟩.
[13] available from ⟨http://www.bsc.es/projects/deepcomputing/linuxoncell/⟩.
[14] available from ⟨http://ftp.uk.linux.org/pub/linux/Sony-PS3/mars/latest/mars-docs-1.1.4/html/⟩.
[15] 村瀬正名ほか：Cell Broadband Engine プロセッサを利用したセキュアソフトウェアプラットフォームの実現，情報処理学会論文誌，Vol.49, No.9, pp.2989–3000 (2008).
[16] 鎌田俊昭ほか：Cell Broadband Engine 向けオフロード機構の提案，研究報告計算機アーキテクチャ（ARC），2010-ARC-190, No.21, pp.1–6 (2010-07-27).
[17] 太田篤志ほか：Cell/B.E. の SPE 向け軽量カーネルの設計と試作，研究報告システムソフトウェアとオペレーティング・システム（OS），2009-OS-111, No.37, pp.1–8 (2009-04-15).
[18] 前田誠司ほか：ヘテロマルチコアプロセッサ Cell 上でのスレッド実行環境，情報処理，Vol.47, No.1, pp.34–40 (2006).
[19] 齋藤孝道ほか：IXP425 における暗号処理の効率的なオフロード方式の実装と評価，情報処理学会論文誌，Vol.51, No.9, pp.1530–1541 (2010).
[20] Tolly Group, Intel Corp.: IXP425 Network Processors, Performance Analysis of VPN Devices, No.204132 (July 2004).
[21] Hughes, J. et al.: Transparent Multi-core Cryptographic Support on Niagara CMT Processors, *Proc. IWMSE09* (2009).
[22] Badia, L.: Real World SSL Benchmarking, Rainbow Techologies Whitepaper (Sep. 2001).
[23] Rescorla, E.：マスタリング TCP/IP SSL/TLS 編，オーム社 (2003).

© 2012 Information Processing Society of Japan

齋藤 孝道 （正会員）

明治大学理工学部情報科学科准教授．博士（工学）．

杉浦 寛

2008 年明治大学理工学部卒業．2010 年明治大学大学院理工学研究科博士前期課程修了．2010 年新日鉄ソリューションズ株式会社入社．

© 2012 Information Processing Society of Japan