社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2009－SLDM－139 (15)
2009－ＥＭＢ－ 12 (15)
2009／3／6

MXコアにおける部分積剰余を用いたRSA暗号の実装

黒木 涉† 飯田 全広† 末吉 敏則†

† 熊本大学大学院 自然科学研究科 〒860-8555 熊本市黒髪 2-39-1
E-mail: †kuroki@arch.cs.kumamoto-u.ac.jp, ††{iida,sueyoshi}@cs.kumamoto-u.ac.jp

あらまし MX コアは細粒度の演算器（PE：Prpcessing Element）を複数搭載した超並列 SIMD（Single Instruction Multiple Data）型プロセッサである．MX コアは処理対象をマルチメディア処理としており，データサイズが数千ビットの多倍長整数を扱うアプリケーションの実装は想定していない．本稿では，多倍長整数演算を扱う公開鍵暗号方式の一種である RSA 暗号の暗号化処理を取り上げ，部分積剰余を用いた MX コアへの実装方法の提案と評価結果について報告する．シミュレーション結果より，鍵長 2,048 ビット RSA 暗号の暗号化処理のスループットは 1,550kbps（MX コア 200MHz 動作時）との結果が得られ，従来の実装手法と比較するとサイクル数は 31.7%の削減，スループットは 2.92 倍の速度向上が得られた．

キーワード MX コア，RSA 暗号，SIMD

A implementation of RSA encryption using Interleaved Modular Multiplication for MX Core

Wataru KUROKI†, Masahiro IIDA†, and Toshinori SUEYOSHI†

† Department of Mathematics and Computer Science,
Graduate School of Science and Technology, Kumamoto University,
2-39-1 Kurokami, Kumamoto-shi, 860-8555 Japan
E-mail: †kuroki@arch.cs.kumamoto-u.ac.jp, ††{iida,sueyoshi}@cs.kumamoto-u.ac.jp

Abstract MX Core is a massively parallel SIMD（Single Instruction Multiple Data） type processor which have fine-grained computing units (PE：Prpcessing Element). The main target of operation in MX Core is multimedia processing. Therefore the MX core does not assume the implementation of the application that handle the multiprecision integer with several thousands bits. In this paper, we focus on the RSA encryption which is a kind of public-key crypto system with multiprecision integer arithmetic operation. Then we report the implementation method of the RSA encryption, which use the interleaved modular multiplication, and describe the evaluation result. From the simulation result, the throughput of 2,048-bit RSA encryption reaches to 1,550kbps on the MX Core that the processing frequency is 200MHz. As compared with the conventional implementation, the number of cycles is reduced by 31.7% and the throughput has improved 2.92 times.

Key words MX Core, RSA, SIMD

1. はじめに

近年，組込みシステムはデジタル家電や携帯電話，デジタルカメラ，携帯電話など様々な製品で使用されている．これらの機器では音声，画像，動画等のメディア処理を主に扱い，CPU，DSP（Digital Signal Processor）等を用いたソフトウェア処理または ASIC（Application Specific Integrated Circuit）等の専用ハードウェアのどちらかを用いるのが一般的である．

しかし，メディア処理を扱うアプリケーションのデータ量の増大や演算の複雑性が上昇した新規格の登場により，CPU や DSP で処理させるには動作周波数は GHz 以上が必要になってきている．その様なプロセッサは，消費電力とコストが高いため組込みシステムに搭載するのは困難である．一方，専用ハードウェアは特定のアプリケーションに対し最適化されており，消費電力や性能の面では優れているが他のアプリケーションに転用することは困難である．また，近年は FPGA（Field Programmable Gate Array）等のようにアプリケーションに応じて回路構成を変更可能なデバイスも注目されているが，面

積，消費電力の面で課題は多い．

このような背景から，ルネサステクノロジは低消費電力，小面積で SoC(System on a Chip) に組込むことを前提としたアクセラレータとしてプログラマブルマルチマトリクスプロセッサ (MX コア) [1] を開発し，それを搭載した MX-SoC [2] を発表している．MX コアはマトリクスアーキテクチャによる超並列 SIMD (Single Instruction/Multiple Data) 型プロセッサコアである．MX コアは 1,024 個の 2 ビット PE (Processing Element) と PE を制御するコントローラ，MX コア外部との通信を仲介する I/O からなり，各 PE には 512 ビットのデータレジスタを 2 セット持つ．MX-SoC に搭載されている MX コアは 90nm プロセス (7 Cu, CMOS Low Standby Process) で 162MHz 動作 (Worst Condition)，2.26mm^2/Core, 145mW/Core と小型で低消費電力のアクセラレータである．

MX コアは音声処理や画像処理等のメディア処理に対して有効であることは，先行研究 [3] [4] で評価されてきた．それ以外にも，MX コアに CAM（Content Addressable Memory）を付加することで，共通鍵暗号方式の一つである 128 ビット AES（Advanced Encryption Standard）暗号の暗号化処理に対しても有効であることが報告 [5] されている．

先行研究では，8〜128 ビットのデータを扱うアプリケーションを実装し評価を行っていた．本研究は，データのビット長が 1,000 ビットを超える公開鍵暗号方式の RSA 暗号の暗号化処理を MX コアに実装する．RSA 暗号の暗号化処理は，乗算と剰余演算の組み合わせで構成されており，暗号化対象のデータ毎に並列に行える．しかしながら，MX コアは先程述べたように各 PE に対して 1,024 ビットのデータレジスタしか持たないため，それ以上のビット数の処理は，複数の PE のデータレジスタを使用してデータを配置し，演算を行う必要が有る．この時，使用するレジスタ数は，演算の中間結果の最大ビット長で確定し，使用するレジスタ数が増加すると，同時に暗号化処理できるデータ数が減少するため，スループットが低下する．乗算剰余演算を乗算後に剰余を行う処理では，積がデータの最大値となる．そこで本稿では，乗算と剰余を同時に演算する部分積剰余を用いることで，使用するデータレジスタ数を削減する実装手法を提案し，シミュレーションで有効性を評価する．

以下，第 2 章では MX コアの特徴や構造，演算方法を紹介し，第 3 章で本稿での実装アプリケーションである RSA 暗号の暗号化処理概要を述べる．第 4 章では，実装アプリケーションの MX コアでの演算方法を示し，本稿で提案する手法について述べる．第 5 章で評価環境と評価結果を示す．最後に，第 6 章でまとめと今後の課題について述べる．

2. MX コア

2.1 MX コアの構造

図 1 のように MX コアは主に PE とデータレジスタで構成される．データレジスタは，PE の両翼に 512 ビットづつの SRAM が配置される．これを MX コアの基本構成単位とし，PE と 1,024 ビットのデータレジスタをあわせてエントリと呼ぶ．MX コア全体で 1,024 エントリあり，これらの PE を SIMD 命令で実行することにより，最大 1,024 個のデータに対し並列処理を行う．また，各 PE 間でデータ転送を行う場合は，垂直方向のデータ通信用配線（V-ch）を使用する．垂直チャネルは，一定の距離にある PE 間を並列にデータ転送することが可能である．これらの SIMD 型処理は，全て図中下部のコントローラ部により制御される．MX コアは一つのチップ上にホスト CPU となるメインプロセッサおよび DMAC (Direct Memory Access Controller)，メモリコントローラ，SDRAM と合わせて実装され，さまざまなアプリケーションにおいてアクセラレータとして機能する．

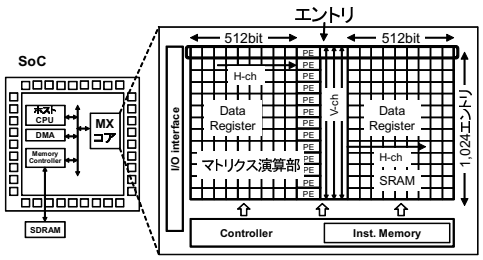

図 1　MX コアの構成

2.2 MX コア内の PE

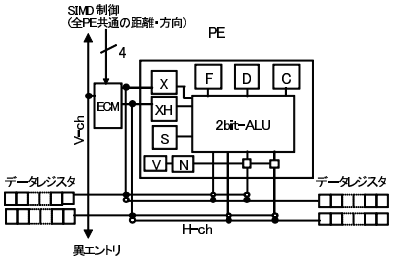

図 2　PE の構成

図 2 に MX コアの PE の詳細を示す．PE は，1 ビットフラグ用レジスタ S，F，D，C，V，N，1 ビットアキュムレータ X，XH および 2 ビット単位で処理できる ALU から成る．また，X，XH は PE 間通信を制御している ECM（Entry CoMmunicator）を介して V-ch とも接続されている．データレジスタから X，XH へデータを読み出し，2 ビット ALU にて演算を行い，演算結果をどちらかのデータレジスタへ書き込む構成となっている．フラグ用レジスタ C，V は直接操作することが可能であり，操作する際は一度 X レジスタを介して各レジスタへ値がセットされる．C はキャリーフラグ，S，F および D は 2 ビット演算における各種処理レジスタとして動作する．各 PE に設けられた V と N は Valid フラグとして，H-ch，V-ch のデータ転送や演算時のマスキングを行うことができる．また，PE は 2 入力 1 出力である．これは，図 2 のように PE の両翼に SRAM を分割することでシングルポートの SRAM で実現している．

2.3 MX コアの演算方法の特徴

図 3(a) のように汎用プロセッサの演算は逐次的にデータの数だけ繰り返す必要がある．一方，MX コアは図 3(b) に示すように 1,024 個のデータを同時に 2 ビットずつ演算を行う．このため，1 つのデータに対しては複数サイクルを要するが，超並列構造を生かした処理で総演算サイクル数を小さくできる．また，PE が 2 ビットという細粒度構造であるために，データ幅の制約が無く様々なビット幅のデータに対応可能となる．

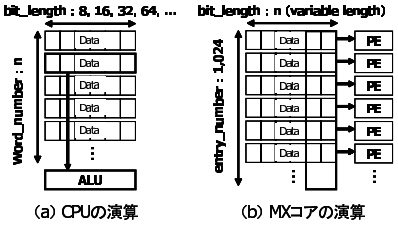

図 3 演 算 方 法

3. RSA 暗号

本章では，RSA 暗号 [6] の概要について説明する．RSA 暗号は，桁数の多い合成数の素因数分解が難しい事を利用した暗号である．暗号の強度は，法 n のビット長（以下，鍵長）で決まる．鍵長が短いと処理は高速になるが，安全性が低下する．逆に鍵長を長くすると処理は遅くなり，安全性は向上する．一般的には，安全性を考慮すると鍵長を 1,024 ビット以上にするのが望ましいとされる．しかし，計算機の性能向上に伴い，2,010 年以降は鍵長を 2,048 ビット以上に設定することが推奨されている [7] .

RSA 暗号の暗号化，復号の変換式は以下の通りである．但し，$M(< n)$ は平文，C は暗号文，e と n は公開鍵のパラメータ，d と n は秘密鍵のパラメータである．

暗号化

公開鍵（暗号鍵）：(e, n)

$$\text{暗号化処理：} C = M^e \ mod\ n \tag{1}$$

復号

秘密鍵（復号鍵）：(d, n)

$$\text{復号処理：} M = C^d \ mod\ n \tag{2}$$

上式で用いられる e, d, n には，以下の関係式が成り立つ．

$$n = p \cdot q$$
$$e \cdot d \ mod\ (p-1)(q-1) = 1$$

式 (1),(2) より，暗号化と復号は引数が違うだけのべき乗剰余演算を行う事がわかる．しかし，復号する際は n の素因数が既知であるため，中国人の剰余定理を用いることにより高速化が可能である．従って，本稿では RSA 暗号の暗号化に絞って実装する．

4. 実　　装

4.1 実 装 方 針

本節では，RSA 暗号の暗号化処理を MX コアへ実装する方針について述べる．実装する RSA 暗号の暗号化処理の鍵長は，512 ビット，1,024 ビット，2,048 ビットの 3 パターンとし，公開鍵は既に生成されているものとする．MX コアでは，公開鍵のパラメータである n と平文 M を入力し，式 (1) を実行し暗号文 C を導出する．e の値は，暗号化において一般的に良く用いられる 65,537 とする．$M^{65,537} \ mod\ n$ をそのまま演算するのは，計算コストが高いので，次節で述べる左向きバイナリ法 [8] を用いることで，乗算剰余演算回数を削減する．

4.2 左向きバイナリ法

表 1 は左向きバイナリ法のアルゴリズムである．左向きバイナリ法とは，指数を $a^{2x+1} = (a^x)^2 \times a$ と式変形することでべき乗回数の削減するものである．左向きバイナリ法を用いることで，$M^{65,537}$ は $(M^{256})^2 \times M$ に変換でき，さらに M^{256} は $(M^{16})^2$ に変換できる．この操作を繰り返すことにより，$M^{65,537} = (\cdots((((M^2)^2)^2)^2)^2\cdots) \times M$ と展開でき，その結果，M の 65,537 回のべき乗算は 16 回のべき乗算と 1 回の乗算で実現できる．MX コアは，演算負荷の高い Step4 および Step5 の乗算および剰余演算を行い，残りの処理はホスト CPU にて行う．

表 1 左向きバイナリ法を用いたべき乗剰余演算

Input:	$N, 0 < M < n, e$
Output:	$M^e mod\ n$
Step1:	$A = M$
Step2:	$i = \lceil log_2 e \rceil, j = 0$
Step3:	$if(e_0 = 1)\ C = M$
	$else\ C = 1$
Step4:	$\mathbf{A = A \times A \bmod n},\ j{+}{+}$
Step5:	$if(e_j = 1)\ \mathbf{C = C \times A \bmod n}$
Step6:	$if(i \neq j)\ return\ Step4$
	$else\ output\ C$

4.3 従来の実装手法

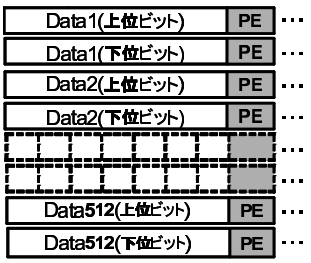

図 4 複数エントリに跨るデータ配置例

現状の MX コアでは，1 エントリあたり 1,024 ビットの SRAM で構成されたデータレジスタを持ち，1 エントリで複数の 8〜

128 ビットのデータを扱うことが可能である．しかしながら，扱うデータが RSA 暗号のように数千ビットクラスの超長ビットになると，図 4 に示すように複数エントリに跨ってデータを配置する必要がある．跨るエントリ数が多くなると，同時に処理できるデータ数が減少するため，スループットが低下する．

また，従来の MX コアへのアプリケーションの実装手法は，C 言語のソースを一部 MX 命令に書き換えることであるため，処理対象のビット幅が適切でない事が多い．

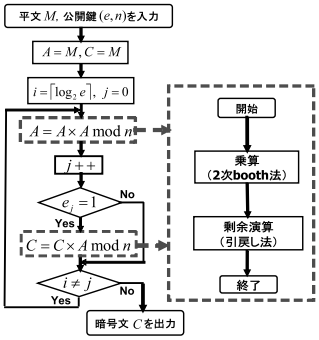

図 5　従来の実装手法での RSA 暗号 暗号化処理のフローチャート

従来の実装手法での RSA 暗号 暗号化処理のフローチャートを図 5 に示す．従来の実装手法は，式 (1) に対して左向きバイナリ法のみを適応しており，破線で囲んだ乗算剰余演算は，2 次 booth 法の乗算を行った後，引戻し法による剰余演算を行っている．この時，平文 M が k ビットだと中間結果の最大ビット長は，積の 2k ビットである．

4.4 提 案 手 法

多倍長整数を保持するのに必要なメモリ量は，演算の中間結果の最大ビット長で決まる．例えば，被乗数又は乗数 X，積 X^2，剰余 X^2 mod Y を演算したときの最大のビット長は積 X^2 の 2k ビットである．そのため，積のビット長を基準として跨るエントリ数を決定する．しかしながら，剰余 X mod Y のビット長は k ビットであり，保持に不要な領域が k ビット存在する．従って，中間結果の最大ビット長を削減できると推論できる．

本稿では，多倍長整数での乗算剰余演算における演算手法として，部分積剰余を用いた剰余アルゴリズム [9] [10] を用い，多倍長整数を保持するのに必要なメモリ量を削減し，メモリの使用効率向上を図る．

表 2　部分積剰余を用いた乗算剰余アルゴリズム

Input:	$n, 0 < A < n, 0 < B < n$
Output:	$W = A \cdot B \ mod \ n$
Step1:	$i := \frac{k}{2}$, $W := 0$
Step2:	$W := W \cdot 4 + A \times (b_{2i-1} + b_{2i} - 2b_{2i+1})$, $i := i - 1$
Step3:	$W := W \ mod \ n$
Step4:	もし $i \geqq 0$ ならば Step2
Step5:	W を出力

表 2 に部分積剰余を用いた乗算剰余アルゴリズムを示す．表 2 のアルゴリズムは，$A \cdot B \ mod \ n$ の中間結果 W に $A \cdot B$ の部分積 $A \times (b_{2i-1} + b_{2i} - 2b_{2i+1})$ を加算し，その和に対して n の剰余を求めることにより，中間結果 W を一定の大きさに保ちながら，乗算剰余演算の解を出力する．但し，A,B,n は全て k ビット，$B = (b_{k-1}, \ldots, b_0)$, $b_{k+1} = b_k = b_{-1} = 0$．また，$(b_{2i-1} + b_{2i} - 2b_{2i+1})$ は，2 次 booth デコードした結果である．部分積は MSB（Most Significant Bit）から求めている．

このアルゴリズム中のデータサイズの最大値は，STEP2 の出力 W か STEP3 の出力 W である．STEP2 の $W \cdot 4$ は $k+2$ ビット，$A \times (b_{2i-1} + b_{2i} - 2b_{2i+1})$ は $k+3$ ビットであることから，その和である STEP2 の出力 W は $k+4$ ビットである．STEP3 の $W \ mod \ n$ は n の剰余であることから k ビットである．従って，本アルゴリズムを用いることで多倍長整数を保持するメモリ量は，$2k$ ビットから $k+4$ ビットに削減できる．

図 6 に提案手法を用いた RSA 暗号 暗号化処理のフローチャートを示す．提案手法も従来手法と同じく，左向きバイナリ法を用いて RSA 暗号の暗号化を演算している．図 5 の従来手法との違いは，破線で囲まれた部分が表 2 のアルゴリズムを用いた点である．$W = W \cdot 4$ の処理は，2 ビットシフトで行う．

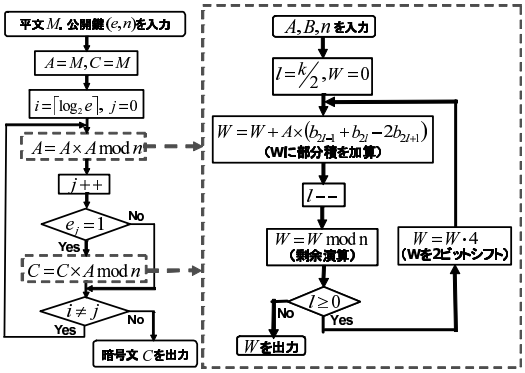

図 6　提案手法を用いた RSA 暗号 暗号化処理のフローチャート

4.5 RSA 暗号 暗号化処理の実装

従来手法での RSA 暗号のデータ配置例を図 7，提案手法でのデータ配置例を図 8 に示す．なお，鍵長は 2,048 ビットである．図中の積 **B** および剰余データ **D** は，乗算および剰余演算の解を示す．また，テンポラリ領域は演算結果を一時的に格納するために使用し，マスクビットやシステム領域は，各 PE の制御に使用する．鍵長 2,048 ビット時の RSA 暗号で使用する平文データのサイズは 2,048 ビットである．MX コアのデータレジスタの片翼のサイズは 512 ビットであり，さらに複数の演算データを配置する必要があるため，実装では 1 エントリあたり 160 ビットごとに折り返し，データ配置を行う．従来の実装手法は，乗算後，剰余演算を行うことで乗算剰余演算を実現している．一方，提案する実装手法は，前章で述べた部分積に対して剰余演算を行うことで乗算剰余演算を実現している点で異なる．

通常のプロセッサでは，一度に 1 個の平文に対してのみ暗号化処理を行う，しかし，MX コアでは，暗号化処理を複数の平文に対して並列に行うことが可能である．図 7 の提案手法適応前のデータ配置例では，積 B を格納するために 1 平文を 26 エントリにわたってデータを配置し，最大 39 並列で平文を暗号化処理を行う．提案手法を適応することにより，図 8 に示すように積 B が平文 M と同じエントリ数で格納でき，最大 78 並列で平文を暗号化が行われる．

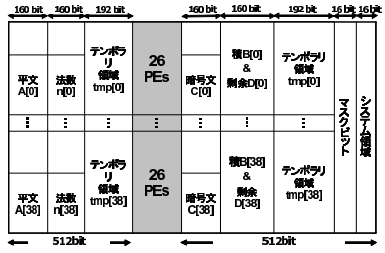

図 7　RSA 暗号実装時のデータ配置（従来手法）

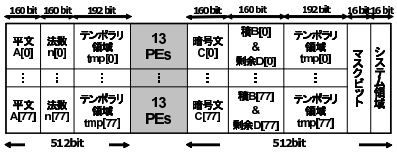

図 8　RSA 暗号実装時のデータ配置（提案手法）

従来の実装手法，提案する実装手法共に，乗算のアルゴリズムは MX コア用の乗算命令と同じ 2 次 booth 法を用いる．MX コア用の乗算関数は，複数エントリに跨るデータを考慮していないので，2 次 booth デコードを行い，PE の各フラグをセットする関数を作成した．以下に複数エントリに跨るデータの 2 次 booth デコードの方法を示す．

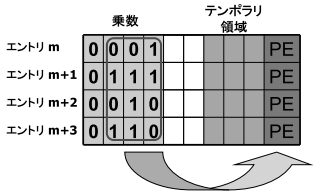

図 9　乗算命令での booth デコード

図 9, 図 10 は，4 エントリに跨る乗数の 2 次 booth デコードを図示したものである．図 9 が MX 用の乗算命令中の 2 次 booth デコード，図 10 が複数エントリに跨る乗数用の 2 次 booth デコードを表す．

MX コア用の乗算関数では，2 次 booth デコードを行うと図 9 に示すように全てのエントリからビット列を読み込み PE 内

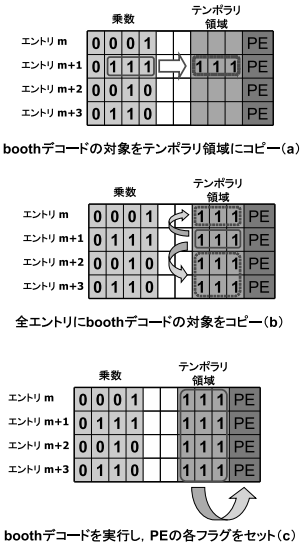

図 10　複数エントリでの booth デコード

のフラグをセットする．これでは，複数エントリに跨る乗数の 2 次 booth デコードには利用できない．従って，図 10（a）, 図 10（b）に示すように，一旦テンポラリ領域にデコード対象のデータ 3 ビットを全エントリにコピーしてから，図 10（c）に示すように 2 次 booth デコードを行うことで，複数エントリに跨る乗算を実現している．

剰余演算のアルゴリズムは，MX コア用の除算関数を参考にして，引き放し法を用いる．MX コア用の除算関数は，乗算と同じく複数エントリに跨るデータを考慮していないが，こちらは PE のフラグ操作と加減算を組み合わせることで実現した．

5. 評　　価

5.1 評 価 手 法

提案手法の比較評価として，まず，提案手法を用いない従来の実装手法による実装．次に，提案手法による実装．最後に，プロセッサのみを用いたソフトウェアでの実装，という 3 つの手法における比較を行う．比較に用いるプロセッサは OpenSSL 0.98e ライブラリ [11] を使用する．MX コアでの評価は，ルネサステクノロジ提供の方式シミュレータ ver0.03.01，および，このシミュレータに提案手法を追加したものを用いる．なお，MX コアの処理時間については，外部とのデータ転送時間およびコントローラ部の処理時間は含まず，演算部のみの処理時間である．また，MX コアの動作周波数は 200MHz として計算を行っている．比較するプロセッサとして，Intel 社の Core Solo プロセッサ 1.20GHz を用いる．評価項目として，リソース量，サイクル数，スループットを選択する．リソース量とは，鍵長 n に対応した平文 M を処理するのに必要なエントリ数である．また，スループットは，MX が各々の鍵長のときに同時に処理できる平文 M の最大サイズをもとに算出している．平文 M の

最大サイズを表 3 に示す.

表 3　各鍵長における同時に処理できる平文 M の最大サイズ

鍵長 (ビット)	従来手法（ビット）	提案手法（ビット）
512	74,752	131,072
1,024	79,872	149,504
2,048	79,872	159,774

5.2　評価結果及び考察

表 4 に，RSA 暗号の暗号化処理を実装したときのそれぞれの鍵長での 1 平文 M を暗号化するのに必要なリソース量を示す. 表 4 の結果を見ると，各鍵長においてほぼ半分のエントリで処理できることがわかる．これは，従来の演算手法では k ビットの乗算剰余演算を行うとビット長が最大 2k の積ができるのに対し，部分積剰余を用いると最大 k+4 ビットのデータ長で演算を行うことができるからである．一度に処理できるビット長が半減することで，使用エントリ数が半分になり，その結果 MX コアで並列演算できる量が倍になる．これらのことから，処理に応じた演算処理を使用することにより処理の高速化，リソース量の削減および並列度の向上につながることがわかる.

表 4　RSA 暗号の MX 実装におけるリソース量 (エントリ)

鍵長 (ビット)	従来手法	提案手法
512	7	4
1,024	13	7
2,048	26	13

表 5　RSA 暗号の MX 実装におけるサイクル数

鍵長 (ビット)	従来手法（サイクル）	提案手法（サイクル）
512	6,796,587	5,264,292
1,024	13,907,806	9,840,214
2,048	30,140,273	20,603,658

表 6　RSA 暗号の暗号化処理におけるスループット (kbps)

鍵長	MX コア (200MHz)		Core Solo
(ビット)	従来手法	提案手法	(1.20GHz)
512	2,200	4,980	4,201
1,024	1,149	3,039	3,340
2,048	530	1,550	2,087

表 5 に，RSA 暗号の暗号化処理を実装したときのサイクル数，表 6 に，RSA 暗号の暗号化処理を実装したときのスループットを示す.

表 5 の結果より，提案手法を用いることで，従来の実装手法に比べ，鍵長 2,048 ビットのときに最大 31.7% サイクル数を削減することができた．また表 6 の結果より，鍵長 2,048 ビットのときに最大で 2.92 倍の速度向上が得られることがわかる．これは，部分積剰余を用いることにより，リソース削減により同時に処理できる平文 M の数が増加したことと，演算に必要なサイクル数が削減されているからである．Core Solo プロセッサと比較した結果，鍵長 512 ビットの場合は 1.19 倍の高速化が得られるが，鍵長が大きくなると性能が低下し，鍵長 2,048 ビットでは 0.74 倍と速度向上が得られなかった．しかし，MX コアと Core Solo プロセッサでは動作周波数に 6 倍の開きがあることを考慮すると，MX コアの方がより効率の良い処理を行っていると考えられる.

6.　まとめと今後の課題

本稿では，公開鍵暗号方式の一つである RSA 暗号の暗号化処理の MX コアへの実装方法と評価結果について述べた．部分積剰余を $M^e \ mod \ n$ に適用することで，演算の中間結果を $2k$ ビットから $k+4$ ビットまで削減できた．提案手法を用いた鍵長 2,048 ビットの RSA 暗号の暗号化処理において，1,550kbps のスループットを実現した．従来手法と比較すると，サイクル数で最大 31.7%削減でき，データが跨るエントリ数も半減できた．その結果，スループットで最大 2.92 倍の改善が得られた.

CoreSolo プロセッサと比較すると，スループットが 0.74 倍と速度向上を得ることが出来なかった．しかしながら，MX コアと Core Solo プロセッサでは動作周波数に 6 倍の開きがあることを考慮すると，MX コアの方がより効率の良い処理を行っていると考えられる.

今後の課題として，今回用いたシミュレータは外部との入出力が考慮されていないため，外部との入出力を考慮したシステム全体の評価が必要であると考えられる．また，今回の提案手法は，乗算と剰余演算を対象としていたが，他の演算の組み合わせにも適応できるかどうか調査する必要がある.

文　　献

[1] H. Noda, et al, “he Design and Implementation of the Massively Parallel Processor Based on the Matrix Architecture”, IEEE J.Solid-State Circuits, vol.42,pp.182-192 (2007).

[2] K. Mizumoto, et al, “A multi matrix-processor core architecture for real-time image processing Soc,” A-SSCC, no.6-2, pp.180-183 (2007).

[3] 卜部公介，黒木渉，大野隆行，飯田全広，末吉敏則：“MX コアにおけるメディアアプリケーションの実装と評価”，信学技報 CPSY2007-14，vol.107，no.175，pp.49-54，Aug. 2007.

[4] 中野光臣，飯田全広，末吉敏則：“超並列 SIMD 型プロセッサ MX コアへのアントコロニー最適化法の実装と評価”，信学技報 CPSY2007-26，vol.107，no.276，pp.13-18，Oct. 2007.

[5] 石崎 雅勝，熊木 武志，田上 正治，小出 哲士，マタウシュ ハンス ユルゲン，行天 隆幸，野田 英行，奥野 義弘，有本 和民: “CAM を有する超並列 SIMD 型プロセッサによる効果的な AES 暗号化処理”，信学技報 CPSY2007-28, vol.107, no.276 pp. 25-30, Oct. 2007.

[6] 岡本 栄司: “暗号理論入門［第 2 版］”，共立出版株式会社 (2002).

[7] “経済産業省 暗号技術検討会 2006 年度報告書，” http://www.meti.go.jp/policy/netsecurity/cryptrec2006.pdf

[8] Johan Torkel Hastad: “Pseudo-random generators under uniform assumptions,” Proc. of Annual ACM symposium on Theory of computing, pp395-404 (1990).

[9] 岡本龍明，太田和夫，情報処理学会　監修: “暗号・ゼロ知識証明数論”，共立出版株式会社 (1995).

[10] 葛毅，櫻井隆雄，ルオンディンフォン，阿部公輝，坂井修一 “インタリーブ型剰余乗算回路の評価”，電子情報通信学会論文誌. A，J88-A No.12，pp.1497-1505 (2005).

[11] The OpenSSL Project *http : //www.openssl.org/* (参照 2007-02)