RC-002

マトリクスアーキテクチャ型超並列演算プロセッサを用いた Mersenne Twister の並列処理実装とその評価

Parallel Processing Implementation and Evaluation of Mersenne Twister with Massive-Parallel Matrix Processor

望月 陽平† 吉田 直之† 松本 直樹† 村上 佑馬†

Youhei Mochizuki Naoyuki Yoshida Matsumoto Naoki Yuma Murakami

熊木 武志‡ 藤野 毅‡

Takeshi Kumaki Takeshi Fujino

1．背景

近年のコンピュータ技術の進歩により，データの処理速度は飛躍的に向上している．そのため自然現象の解析から大規模 LSI 開発に至るまで様々な事象をシミュレートすることが可能になってきている．ここでコンピュータは本質的に決定論的処理を行うため，より精度の高いシミュレーションを行うためには，予測不能な事象に対して確率論的手法を用いる必要がある．また，インターネットの普及に伴い重要性が増している，暗号処理をはじめとするセキュリティアプリケーションの多くにとっては，データ秘匿の安全性，すなわち次の情報の予測を困難にして，秘匿性を高めることが重要となっている．以上より，これらの要求を満たすためには，高品位の乱数を大量，かつ高速に生成することが必要な条件の 1 つとなってきている．そこで我々は高品質，かつ長周期の特徴をもつ疑似乱数生成アルゴリズムの１つとして知られている，Mersenne Twister [1] を，ソフトウェアベースでありマトリクスアーキテクチャ構造を持つ SIMD 型演算プロセッサにて，超並列に乱数生成を行う実装手法の実現に着目した．Mersenne Twister は，生成法や周期等でいくつかの種類に分けることができるが，基本的に逐次処理で乱数を生成するアルゴリズムであるため，独立動作を行うプロセッサを用いて動作周波数や消費電力を抑えつつ高速生成を実現するのは困難であった．また，ASIC を作成してハードウェア的に並列処理を行う手法も提案されているが，柔軟性やハードウェアコストの観点から問題があった．

本研究ではこの Mersenne Twister を C 言語ベースのプログラムによるソフトウェアベースプロセッサながら，並列処理に特化した LSI であるマトリクスアーキテクチャ型超並列演算プロセッサ(以下 MX-1 と呼ぶ) [2] を用いて高並列化実装を行い，その評価を行う．Mersenne Twister は，生成法，及び周期の違いからいくつかの種類が提案されており，本研究では，実装ハードウェアコストが最も大きく，長周期である MT19937 を主な対象とした．CPU，及び GPU 等のソフトウェアベースのプロセッサを用いて並列に乱数を生成する手法としては，MT2203 と呼ばれるアルゴリズムも提案，実装されており [3]，[4]，入力パラメータの異なる小規模な乱数生成器を最大 1,024 個利用することができる．しかしながらこのアルゴリズムは周期が非常に短いため高品位の乱数を連続して得ることが根本的に難しい．我々は最大 1,024 並列の SIMD 処理が可能であり，かつ演算ユニット間の高速データ転送が可能な MX-1 の処理能力を活かして，ソフトウェアベースのプロセッサでは難しいとされていた MT19937 の並列化を実現する手法を提案する．

以降，2 章では，マトリクスアーキテクチャ型超並列演算プロセッサの概要，及び仕組みについて述べ，3 章では，既存の Mersenne Twister に関する研究について報告する．4 章では，MX-1 を用いた Mersenne Twister の並列実装法について詳述し，5 章で実装結果とその評価について述べることとする．

† 立命館大学理工学研究科 〒525-8577
滋賀県草津市野路東 1 丁目 1-1

‡ 立命館大学理工学部 〒525-8577
滋賀県草津市野路東 1 丁目 1-1

2．マトリクスアーキテクチャ型超並列演算プロセッサ MX-1

本章では，マトリクスアーキテクチャ型超並列演算プロセッサである MX-1 について述べる．MX-1 は，大量のデータに対し数値・論理演算処理を行う際の，効率的な並列処理を実現するために研究・開発を行ってきた SIMD 型演算プロセッサである．SRAM をベースとして，演算器(PE：Processing Element)の配置とデータの処理ベクトルを工夫することにより並列度を 1,024 と，従来の汎用プロセッサや DSP と比較して大幅に向上させることを実現している．また，演算器とデータレジスタ領域を密結合することによって PE，メモリ間の I/O 転送にかかる消費電力を削減した．90 nm 7Cu CMOS テクノロジの実装結果では，面積 3.1 mm^2，消費電力 250 mW であり，16 ビットの加算処理では動作周波数 200 MHz で 40 GOPS (Giga Operation per Second)の性能を得ることができるのを確認している [2]．また，これまでに画像圧縮，物体検出処理，電子透かし処理，及び暗号化処理等，様々なアプリケーションを実装してその効果を確認してきた [5]-[15]．マトリクスアーキテクチャ型超並列演算プロセッサは，C 言語ベースのプログラムで様々なアプリケーションを処理するためのハードウェア構成を採用しており，図 1 に示すように大きく分けて，SDRAM，制御用 CPU コア，DMA，及び MX コアの 4 つで構成される．MX コアは 1,024 個の PE と，1,024 エントリ×1,024 ビットの SRAM，外部メモリとのバスであるインターフェースモジュール，MX コアを制御するコントロ

ーラーで構成されており，1 命令で最大 1,024 並列のデータを一度に処理することが可能である．また，制御用 CPU はプログラムのフロー制御，及び逐次処理実行部分に使用され，両者を効率よく動作するようにアルゴリズムを整理しプログラムを作成することが重要となってくる．命令の実行は，制御用コントローラー内部の命令メモリから制御線を通して PE に命令が送られ，1,024 個の PE が左右の SRAM から読み出した値を演算することによって行われる．MX コアの各 PE は加算器，乗算器，及び論理演算器等から構成されており，Carry flag(C フラグ)と Valid flag(V フラグ)が備わっている．C フラグは，演算結果がオーバーフローした場合に，桁上がりの数値が代入される．V フラグは，各エントリ処理の実行を制御するものであり，1 が格納されているエントリは演算可能となり，0 が格納されているエントリは演算を行わない．データの処理単位は，演算器 1 つに対し，左右のエントリ 1 組である．演算器と左右のエントリは水平チャネル(Horizontal channel)で接続されている．これが 1,024 並列垂直方向に並んでおり，単一の命令で全並列の演算を可能にする．また，垂直方向でデータのやり取りをする際には，垂直チャネル(Vertical channel)を利用する．垂直チャネルによるデータの移動は，2 の累乗でバイパスされており，高速な転送が可能となっている．演算処理の方法は，従来のメディアプロセッサや専用プロセッサの多くがビットパラレル・ワードシリアル(Bit-Parallel Word-Serial)であったのに対し，ビットシリアル・ワードパラレル(Bit-Serial Word-Parallel)の手法をとっている．そのため，インターフェースモジュールは，外部 SDRAM に格納されている演算対象データの処理ベクトルを直交に変換して，SIMD 型演算モジュールを効率よく動作させる働きを担う．コントローラーは内部に命令メモリを持ち，アプリケーションにあわせてプログラムを入れ替えることで柔軟にマルチメディアアプリケーションを処理することが可能となっている．

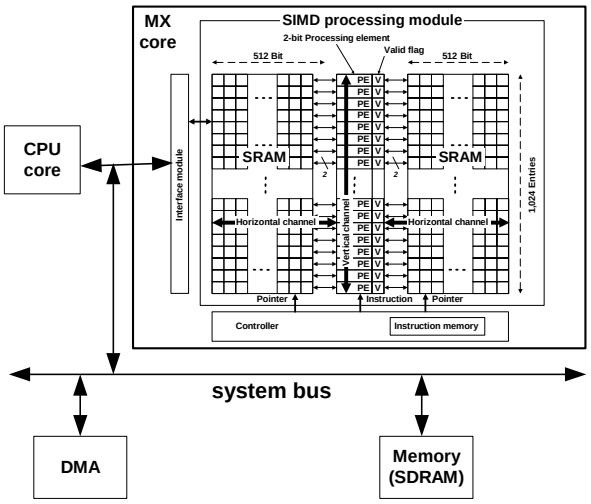

図 1　MX-1 の全体構成ブロック図.

3．疑似乱数生成アルゴリズム Mersenne Twister

3.1 概要

Mersenne Twister は，1997 年に松本眞により発表された疑似乱数生成アルゴリズム [1]でありその代表的なものとして MT19937 と呼ばれるものが有名であり，主に以下のような特徴を持つ．

・長周期($2^{19937}-1$)
・623 次元超立方体中に均等に分布する
・乱数生成時のメモリ使用効率が良い
・高速な乱数生成

図 2 に MT19937 の処理ブロック図を示す． MT19937 は，32 ビットの初期値 (Input Seed)から 32 ビット 624 ワード (Word [0]～Word [623]) の初期データを生成する Pre-Processing モジュール，疑似乱数生成を行う主要部である Next Generator モジュール，及び Next Generator モジュールから出力された疑似乱数の分散を向上させる Tempering モジュールの 3 つの処理部により構成されている．Next Generator モジュールは 624 個のワードから，Word [i], Word [i+1], Word [i+397] (i=0, 1, 2, …, 207)の 3 つのワードを選択して乱数を生成し，Tempering モジュールに出力する．そして Tempering モジュールでは，シフト，及び論理演算を用いて Next Generator モジュールから出力された疑似乱数をさらに攪拌させ出力する．

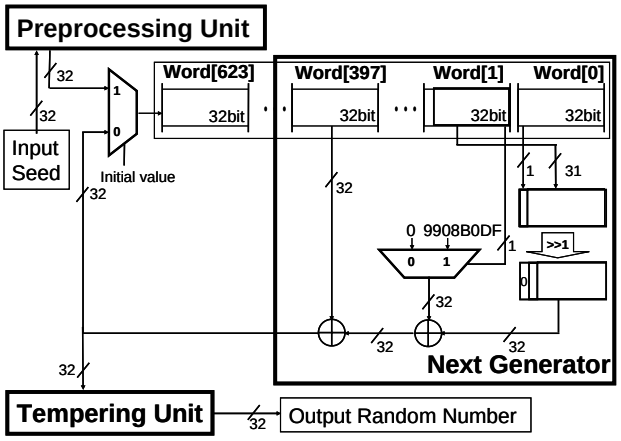

図 2　MT19937 の処理ブロック図.

3.2 乱数生成の流れ

Mersenne Twister は，n 個の符号なし整数である x[0]～x[n-1]，整数 i，符号なし整数定数 v，p，a を用いて疑似乱数列 y を生成する．ここで，*w* をワード長，a，b，及び c をベクトル値，m を中間値，*r* をワードの変わり目，*u*， *s*, *t*，及び *l* を整数値とする．以下にアルゴリズムを詳述する．

Preprocessing 処理

(0) 初期データ作成

$v=1\cdots10\cdots0$ (w-r, r)　$p=0\cdots01\cdots1$ (w-r, r)　$a=a_{w-1}a_{w-2}\cdots a_1a_0$

(1) i を 0 とし，x[0]～x[n-1]に 0 以外の初期値を入れる．

Next Generator 処理

(2) x [i]，y，x [(i+1) mod n]，p より論理演算を行い y を算出する．

y=(x[i] AND v) OR (x[(i+1) mod n] AND p)

(3) x [(i+1) mod n]，y より論理演算を行い x [i]を算出する．

x[i]=x[(i+m) mod n] XOR (y>>1)

XOR { y の LSB が 1 の時 a
　　 { y の LSB が 0 の時 0

Tempering 処理

(4) y をシフトと論理演算を用いて攪拌

y=x[i]
y=y XOR (y>>*u*)
y=y XOR ((y<<s) AND b)
y=y XOR ((y<<*t*) AND c)
y=y XOR (y>>*l*)
y を出力

(5) 次回の初期値作成

i=(i+1) mod n

(6) (2)に戻る

以上の処理を繰り返すことで長周期 に高品質の乱数を生成し続けることが可能である．

3.3 Mersenne Twister に関する既存の実装例

MT19937 は，暗号化処理，及びモンテカルロシミュレーション等，多くのアプリケーションに適用されてきており，その有効性が確認されてきた．しかし，本質的に逐次処理であり，ソフトウェアベースのプロセッサでは並列化が難しいため，高速処理のためには CPLD [16]，及び FPGA [17]，[18]，を用いた逐次処理の高速化，もしくは ASIC [19]を用いた実装による並列化が図られてきた．逐次処理に関しては図 2 にて示した通りである．ここで図 3 にハードウェア実装による並列処理の例を示す．3.1 節で述べた通り，Next Generator モジュールから出力される擬似乱数は 3 ワードのみに依存しているので，3 つのワードの組み合わせとそれらの処理回路を複数用意することで，並列化処理を行うことが可能となる．MT19937 の ASIC による並列化を行った研究では，並列度を変化させたときの評価を行っており，並列度の増加に比例してスループット，及び実装面積が増加することが報告されている [19]．ハードウェア実装によって疑似乱数生成処理は高速化されるが，乱数生成のために専用のハードウェアを作る場合，コストが高くつき，使用用途に柔軟性が無くなるという問題がある．これに対してソフトウェアベースのプロセッサで Mersenne Twister を並列に生成する方法として MT2203 と呼ばれるアルゴリズムが提案，実装されている [3]，[4]．これは，パラメータの異なる最大 1,024 個の小型乱数生成器を用意することで並列に乱数生成が可能となるものであり，図 4 に MT2203 の実装例を示す．MT2203 はすべての小型乱数生成器に対して同一のデータを入力する代わりに Next Generator モジュールで処理する際のパラメータ値を変えることで，最大 1,024 個分の擬似乱数を生成することが可能であるが，並列処理と引き換えに周期が 2^{2203}-1 となり MT19937 の 2^{19937}-1 に比べ極端に短いという問題を抱えている．

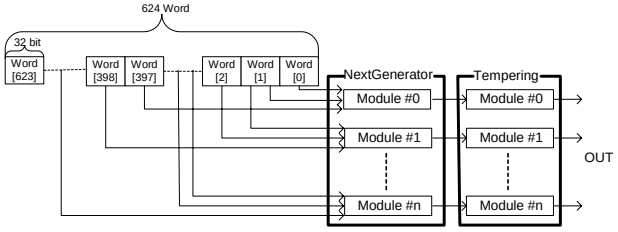

図 3　ハードウェア実装による Mersenne Twister (MT19937)の並列処理例.

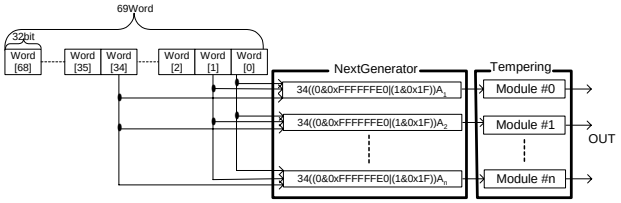

図 4　ソフトウェア実装による Mersenne Twister (MT2203)の並列実装例.

4． MX-1 による MersenneTwister の実装

本章では，Mersenne Twister 中でも代表的な擬似乱数生成アルゴリズムである MT19937 に対して，MX-1 を用いて効果的に並列処理を実現する手法を述べる．

4.1 MX-1 による Mersenne Twister 処理の並列化，及び実装上の高速化について

MT19937 を用いて生成した高品位な乱数を様々なアプリケーションに対してリアルタイムに提供する場合，従来のソフトウェアベースプロセッサでは動作周波数を向上させて逐次処理を高速化する，もしくは周期が短いが並列処理が容易な MT2203 に変更する必要があった．また ASIC を用いた場合，MT19937 を専用回路によって並列処理を行う事が可能ではあるが，適用の柔軟性に欠け，ハードウェアコストが向上することとなる．本研究はこれらの問題を解決するために，MX-1 を用いて MT19937 を並列化するためのハードウェアアルゴリズムを開発し，更に実装上の工夫により長周期の乱数を高速に生成することを可能とした．並列化に際しては，MX-1 の高い並列度による SIMD 演算処理と，水平・垂直チャネルによる柔軟なデータ移動を利用する．図 4 に示す通り MT2203 が全並列に小型乱数生成器に対して同一のデータを入力するのに対して，図 3 に示す通り MT19937 を並列化する場合は 32 ビットの乱数を生成する度に，並列に複数個配置した Next Generator モジュールに対して，各々に異なる 3 ワード分のデータを入力する必要がある．加えて，以前出力された乱数を入力として

後に利用する必要がある．そのため通常のソフトウェアベースプロセッサではデータの入出力が特にボトルネックとなる．これに対して 1,024 個の PE を持つ MX-1 では，図 5 に示す通り，垂直方向にデータを配置して並列に乱数を生成した後に，垂直チャネルを用いて以前のデータを残したまま全データを効率よく移動させることができる．これについては，4.2 節にて詳述する．以上より，MX-1 は並列乱数生成処理と次入力データのための移動を実現することが可能となる．

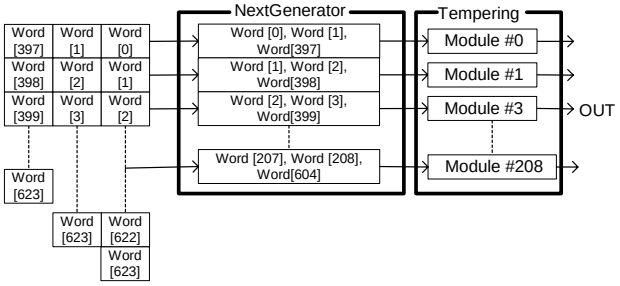

図 5　MX-1 に実装した MT19937 の並列処理例.

次に実装上の工夫について述べる．MX-1 は，SDRAM，CPU，DMA，及び MX コアからなり，MX-1 は CPU のアクセラレータとしての役割を果たすこととなる．MX-1 はビットシリアルに処理を行うために，インターフェースモジュールではデータの直交変換が必要となる．そのため高速に擬似乱数を生成するには，CPU と MX コア間のデータのやり取りを削減すること，及び CPU と MX コアが並列に動作することが重要となる．MT19937 による擬似乱数を連続的に生成する場合，初期入力データから生成された擬似乱数を再度用いて乱数を生成する必要がある．そのためレジスタ容量が小規模である一般的な CPU を用いた場合，初期入力データから生成される擬似乱数をメモリに書き込む処理と，そこから次の入力データを読み込む処理が頻繁に行われることとなる．この実装方法をそのまま MX-1 に採用した場合，データの入出力に伴う直交変換に処理時間が多くかかることとなり効率的とは言えない．MX-1 は通常の CPU と比較して PE の左右に，合計 1M ビットとなる十分な量のレジスタを備えているため，生成された乱数を外部メモリに出力しつつ，同様のデータを内部に保持したまま次の擬似乱数生成を行うことができる．また，CPU と MX コアを並列に動作させて処理速度を向上させるために，本実装では DMA を利用してデータ転送時の CPU の負担を軽減し，ポーリングやプログラム内の変数計算等，並列処理と直接関係のない別の処理をさせるようにした．また，Next Generator，及び Tempering 処理の MX 動作命令をユーザマイクロコードとして 1 つにまとめることで CPU から MX に対する命令数を削減し，並列動作を実現することができた．並列動作のイメージ図を図 6 に示す．並列動作を考慮しない通常の実装 (Straight implementation)では，MX コアの処理の間に CPU の処理が入るが，MX 動作命令をユーザマイクロコードとして 1 つにまとめることで CPU と MX コアを並列に動作させることが可能 (Optimized implementation)となり，高速な処理の実現が期待できる．

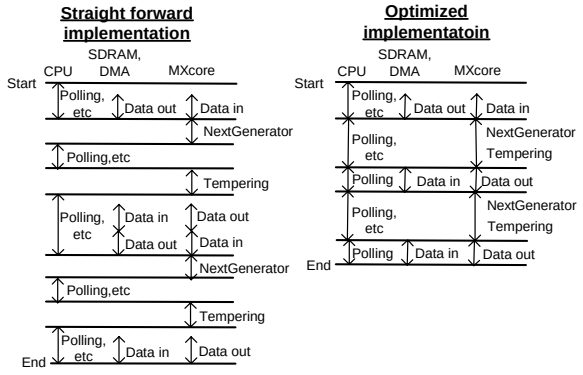

図 6　CPU と MX コアの並列動作イメージ図.

4.2 MT19937 並列処理のハードウェアアルゴリズム

MX-1 を用いて MT19937 を並列処理するためのハードウェアアルゴリズムについて，図 7 に示すフローチャート，及び図 8 に示す MX コア内のデータ配置を基にして，動作について詳述する．なお，図 7 中の(1)～(4)の番号に関しては，以下に示す処理毎の説明番号に対応している．

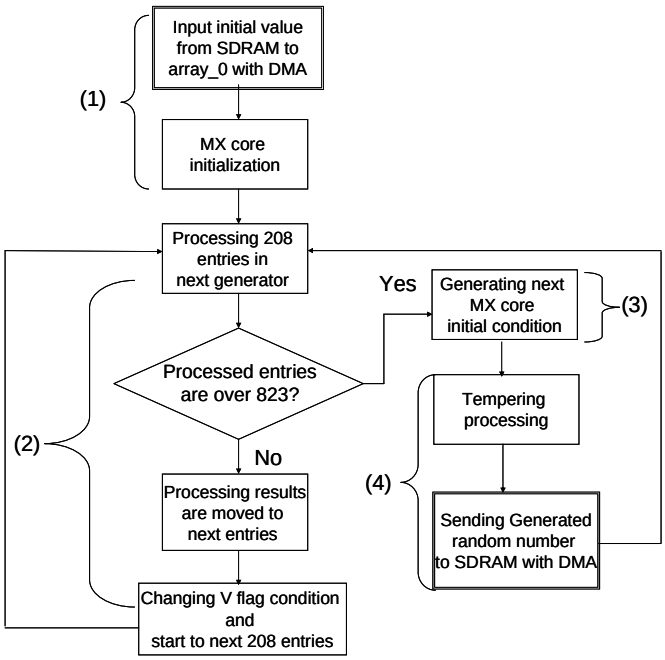

図 7　MX-1 による MT19937 処理のフローチャート.

(1) MX-1 内の初期状態として，各 PE に入力する 3 つのワードを準備するために，外部メモリに格納されている 624 ワードを図 8 - (a)に示す通り Column_A の 624 エントリ分に DMA 経由で格納する．次に Column_A から垂直チャネルを利用して 1 エントリ，及び 397 エントリ分上方にコピーを実行して Column_B に 1～623 ワード，Column_C に 397～623 ワードを格納する．

(2) 図 8 - (a)における 0～207 エントリの V フラグを 1 にセットし，Column_A，B，及び C のデータから Next Generator 処理を実行し，その処理結果 (Word [x0]～Word [x207]) を Column_X に格納する．続いて

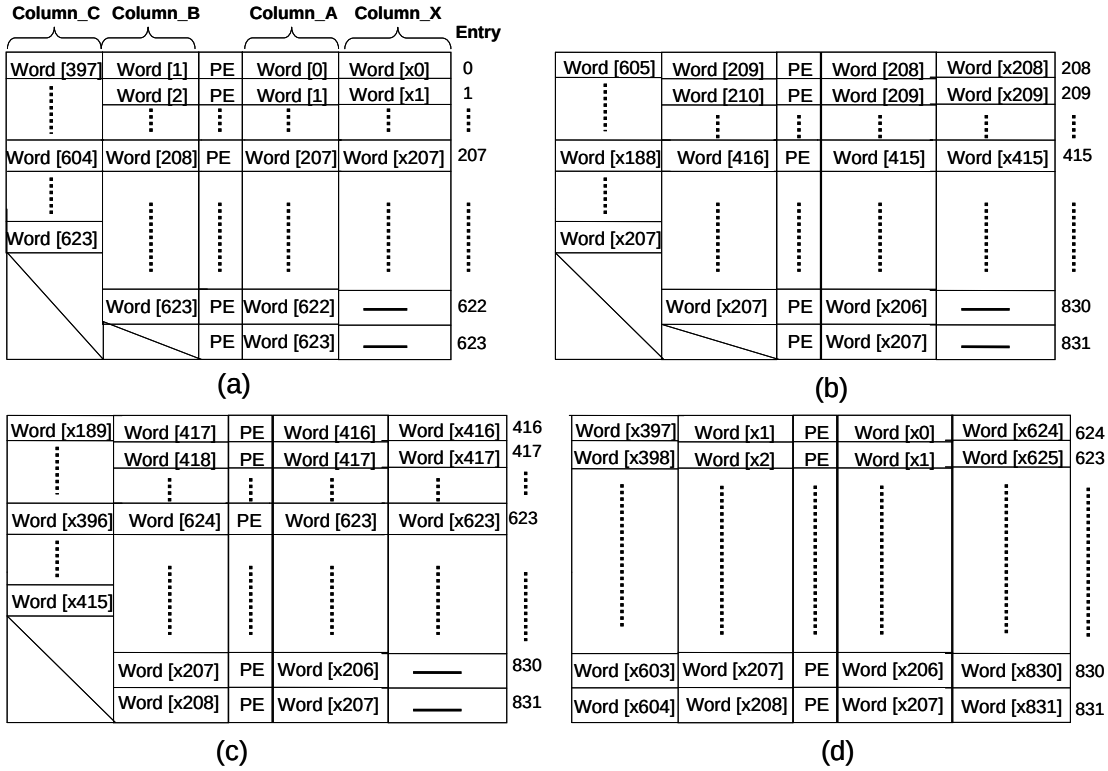

図 8　MT19937 処理時における，MX コア内のデータ配置.

図 8 - (b)に示す様に，垂直チャネルを用いて，Column_A (624～831 エントリ)， Column_B (623～830 エントリ)，Column_C (227～434 エントリ)初期データをコピーする．その後 208～415 エントリの V フラグを 1 にセットし， Next Generator 処理を実行し，その処理結果 (Word [x208]～Word [x415])を Column_X に格納する．その後，図 8 – (c)に示す様に，Column_B (831 エントリ)，及び Column_C (435～642 エントリ)にコピーする．次に 416～623 エントリの V フラグを 1 にセットし， Next Generator 処理を実行し，その処理結果 (Word [x416]～Word [x623])を Column_X に格納し Column_C (643～831 エントリ)にコピーする．最後に図 8 - (d)に示すように，624～831 の V フラグを 1 にセットし， Next Generator 処理を実行する．この処理結果を (Word [x624]～Word [x831])とする．

(3) 次ループ分の初期状態を作るために，処理結果(Word [x208]～Word [x831])を Column_A (0～623 エントリ)，処理結果 (Word [x209]～Word [x831])を Column_B (0～622 エントリ)，処理結果 (Word [x604]～Word [x831])を Column_C (0～226 エントリ)にコピーし図 8 - (a)と同様のデータ配置とする．

(4) 全エントリの V フラグを 1 にセットし，4 回分の Next Generator 処理結果 (Word [x0]～Word [x831])について Tempering 処理を行う．こうして生成した乱数を，DMA 転送により外部メモリに格納し，(2)の処理に戻る．

上記の処理によって，一度に 832 個の疑似乱数を生成することができる．

4.3 Next Generator 処理の手順

本節では，Mersenne Twister の主要処理部である Next Generator モジュールについて，MX-1 による並列処理を実現する手法を述べる．MT19937 のハードウェア実装では全体の 623 ワードから使用するワードである 397 ワードを引いた最大 226 並列の処理が可能であるが，本実装では 624 ワードを 3 回の SIMD 処理で終了させるために，1 回の処理を 208 ワードとする．MX-1 で図 2 における MT19937 に基づく Next Generator モジュールの処理を行う場合，32 ビットの乱数 1 つを生成するため 3 つのワードを 1 エントリに格納し，これらを 208 並列に配置して疑似乱数生成処理を行う．以下処理手順を図 9 に示し，動作を詳述する．

(1) MT19937 で定められている定数$\{9908B0DF\}_{16}$を初期値として設定し，208 エントリの 3 つのワードを SRAM 内の Word[i]，Word[i+1]，Word[i+397] (i=0, 1, 2, …, 207)に配置する．

(2) Word[i]の MSB1 ビットと Word[i+1]の MSB，LSB を除く 30 ビットを PE に送り，Column_X の下位 31 ビットにコピーする．

(3) Word[i+1]の LSB が 1 の場合，V フラグを 1 にセットし，Column_X と定数を PE に送り XOR 演算を行う．この時，V フラグが 0 のエントリは XOR 演算が行われないため 0 との XOR 演算の結果と同一となる．

(4) 全エントリに V フラグをセットし，Column_X と Word[i+397]を PE に送り XOR 演算を行う．

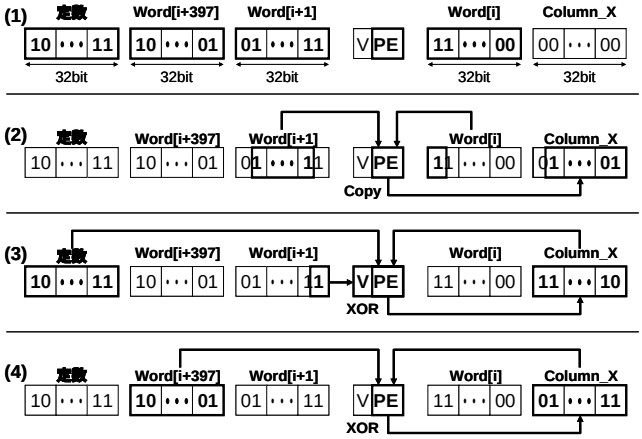

図 9　MX-1 による Next Generator モジュールの処理例.

以上の処理が，208 並列で 4 回連続行われ，832 エントリ分の処理を終えたところで Tempering 処理へと移行する.

4.4 Tempering 処理の手順

本節では，図 2 における MT19937 に基づく Next Generator モジュールで作成した乱数列の分散を向上させるための Tempering モジュールについて，MX-1 による並列処理を実現する手法を述べる．通常 Tempering 処理は，1 回の Next Generator 処理に対して，その都度行われる処理である．しかしながらシフト演算と XOR 演算を複数回繰り返し，ビット列の並びを変更するため，ビットシリアル処理を主体とする MX-1 では，処理クロックサイクル数が大きくなる．そのため本実装では Next Generator 処理で一度に 208 エントリまで生成処理を行い，残りのエントリはその次以降の Next Generator 処理に利用することができることを利用して，最大 832 エントリまで連続して処理を行った後に，全 832 エントリに対して Tempering 処理を施すこととした．その結果 Tempering 処理の回数は通常の 4 分の 1 となり，MX-1 の並列度を最大限活かしつつ，クロックサイクル数の大幅な削減を実現した．以下処理手順を図 10 に示し，その動作を詳述する.

(1) Next Generator 処理の出力である x を別領域にコピーして y とする.
y = x
(2) y の 21 ビット分をコピーして移動させることで y を 11 ビットシフトしたデータ y'を作成し，y のデータと XOR 演算を行う.
y = y XOR (y>>11)
(3) y の 25 ビット分をコピーして移動させることで y を 7 ビットシフトしたデータ y'を作成し，MT19937 で定められている定数$\{9D2C5680\}_{16}$と AND 演算を行い，その処理結果と y のデータを XOR 演算する.
y = y XOR ((y<<7) AND 定数$\{9D2C5680\}_{16}$)
(4) y の 17 ビット分をコピーして移動させることで y を 15 ビットシフトしたデータ y'を作成し，MT19937 で定められている定数$\{EFC60000\}_{16}$と AND 演算を行い，その処理結果と y のデータを XOR 演算する.
y = y XOR ((y<<15) AND 定数$\{EFC60000\}_{16}$)
(5) y の 14 ビット分をコピーして移動させることで y を 18 ビットシフトしたデータ y'を作成し，y のデータと XOR 演算を行う.
y = y XOR (y>>18)
(6)y を乱数として出力

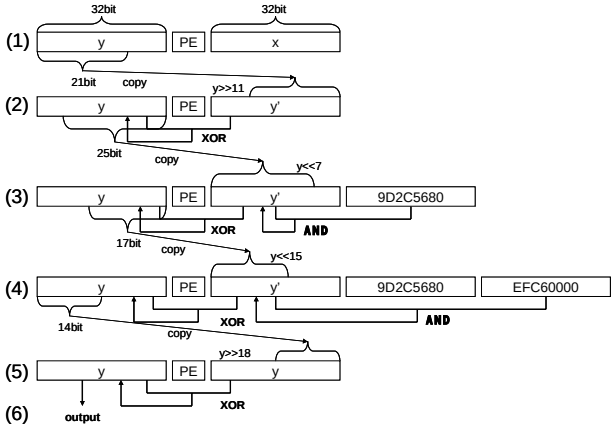

図 10　MX-1 による Tempering モジュールの処理例.

5. 実装結果と評価

MX-1 による並列擬似乱数生成処理の効果を検証するため，ルネサスエレクトロニクスの総合開発環境である High-performance Embedded Workshop (HEW) [20]を用いて実装を行った．なお，MX コアの動作を HEW にてサポートするため，HEW には専用のデバッカ，コードジェネレータを組み込んでいる．実装対象アルゴリズムは MT19937 であり，シミュレーションの動作周波数は文献[2]を基にして 200 MHz としている．Mersenne Twister は，パラメータの違いにより異なる乱数系列を生成する MT19937，MT11213A，MT11213B，TT800 等が提案されており，本研究では MT19937，MT11213A，及び MT11213B を実装している．本章では周期が最も長く，かつハードウェアコストが最も大きい MT19937 の実装結果について述べる.

5.1 MX-1 による MT19937 の処理結果評価

HEW を用いて MT19937 を実装し，データ入力，Next Generator モジュール，Tempering モジュール，及びデータ出力毎に擬似乱数 832 個生成分の MX コア単体のクロックサイクル数を算出した．結果を表 1 に示す．データの入出力は主にホスト CPU と DMA で動作するため，MX コアのクロックサイクル数は少ないが，全体から見ると約 22%程度を占めていることがわかる．しかし，データ入力に関しては，一度初期値を転送すればその後は行われず，MX コア内で連続して初期値を作成できる．よって乱数生成を行う回数が多くなるほど生成効率が上がっていくと考えられる．特に MT19937 は，周期が $2^{19937}-1$ と長大であるためデータの入出力がボトルネックとなることは無いと言える.

表 1　MX-1 による MT19937 処理におけるクロックサイクル数詳細

	MX core
Date input	656
Next Generator	3,770
Tempering	1,380
Date output	864

5.2 乱数生成数による比較

前節で述べたように，乱数生成数が増加すれば入出力のボトルネックは小さくなっていくことを示すために，処理回数の違いでスループットがどのように変化するかを調べた．処理結果を図 11 に示す．

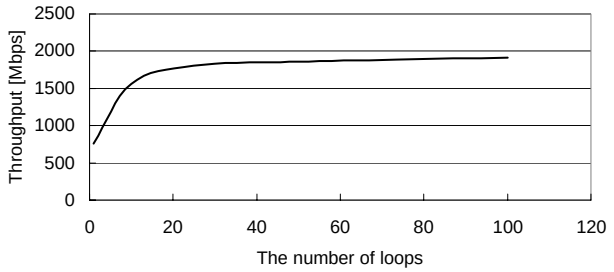

図 11　乱数生成数によるスループットの比較結果.

図より，X 軸が乱数生成のループ回数となっており 832 個の乱数を生成して 1 ループと定義する，Y 軸はスループット値を表している．これより乱数生成数の増加に比例してスループットが向上していることがわかる．また，乱数生成数が増加していくにつれて，スループットが一定の値に近づいていることがわかる．これは図 12 に示す，処理回数によるクロックサイクル数の内訳からも分かる．データ入力は初めの一回だけなのでクロックサイクル数は一定であるが，他処理のクロックサイクル数は線形に増加していくため，データ入力が他の値に比べ非常に小さい値となっていき，スループットは理想値の一定の値に近づいていくと考えられる．理想値は，入力の処理時間が 0 のときであり，その場合のスループットは 1,973 Mbps である．

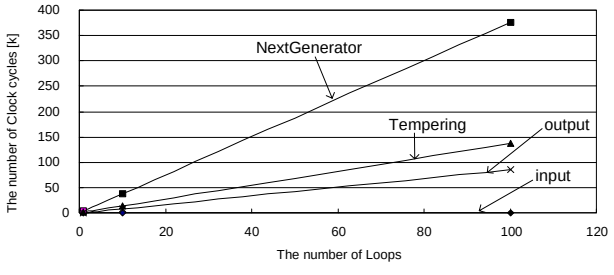

図 12　全体のクロックサイクル数の内訳.

次に，4.1 節で述べた，ユーザマイクロコードを作成した処理の最適化について検証する．CPU と MX コアを並列に動作させた実装と，通常実装の場合の処理時間の変化を調べた．比較結果を表 2 に示す．

表 2　CPU と MX コアを並列に動作させた実装と通常実装の処理時間比較結果.

The number of Loops	Straight forward implementation [us]	Optimazed implementation [us]
1	373.51	33.35
5	1,805.99	89.27
10	3,593.51	158.95

表 2 より，CPU と MX コアを並列に動作させた場合，最大で約 22 倍処理速度が向上していることがわかる．また，乱数生成のループ回数が 1 回の場合は約 11 倍，5 回の場合は約 20 倍であり，10 回の場合は約 22 倍と増加度が一定に近づく．これはユーザマイクロコード作成時に MX コアの命令を連続させ CPU の処理を極力少なくさせることが必要となるが，CPU と DMA の動作を伴う生成乱数データの出力が一定の間隔で出現するためであると考えられる．以上より，CPU と MX コアの並列動作効果を確認することができた．

5.3 既存のプロセッサとの処理性能比較

MX-1 の擬似乱数生成能力を客観的に評価するために，他ベンダのプロセッサによる処理結果，及び関連論文の結果と比較を行った．他ベンダのプロセッサには MX-1 と同じくソフトウェアベースのシステムであり，CPU モジュールのほかに SIMD 機能を備えつつ，低消費電力等ハードウェアコストの低いものを選定する．今回比較対象とした ARM Coretex-A8 は，市場での普及率の高いプロセッサであり，シングルボードコンピュータ超小型組み込みボードである BeagleBoard [21]等に搭載され，720 MHz，300 mW で動作する．また，NEON テクノロジと呼ばれる SIMD 機構を組み込んでおり，MX-1 と同様に特定のアルゴリズムの並列処理が可能となっている．今回，既存のプロセッサと比較するにあたって，ARM Coretex-A8 を搭載している BeagleBoard Rev.C4 と，他研究者の論文である [4]，[16] – [19]から，MT19937 の実装を行い，処理速度，ハードウェアコスト，及び実装上における柔軟性を考慮して FPGA 搭載ボードである Celoxica RC1000 を用いて実装した文献 [17]を引用した．図 13 にスループットの比較結果を示す．なお，動作周波数はそれぞれ BeagleBoard Rev.C4 (ARM Cortex-A8)が 720MHz，Celoxica RC1000 (Xilinx XCV2000E) が 24.234MHz である．

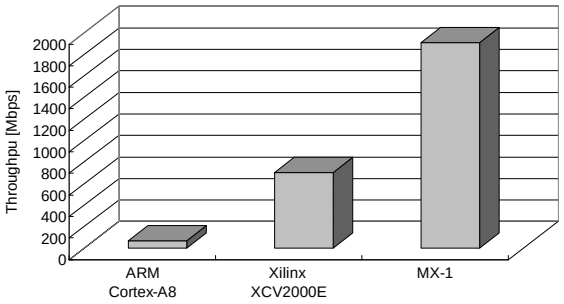

図 13　各プロセッサのスループット比較結果.

図 13 より，MX-1 がこの 3 つのプロセッサの中では最も処理速度が速いことがわかる． ARM Coretex-A8 より約 29

倍，Xilinx XCV2000E より約 2.7 倍速い処理速度を得ることができ，MX-1 による MT19937 の並列実装効果を確認できた．ここで，ARM Cortex-A8 に関してはコンパイラ(Sourcery G++4.4.1)による，MT19937 の並列化がやはり困難であり，NEON による効果を確認することはできなかった．以上の結果より MX-1 はソフトウェアベースながら MT19937 を並列化することが可能であり，高品位，かつ長周期の乱数を効率よく生成できることが分かった．また，今回提案した実装手法は，MX コアの並列度を 1,024 から増加させた場合にも適用できるため，更なる処理速度の向上を目指すことも可能である．

6．まとめ

本研究では，ソフトウェアベースのプロセッサにおける並列化は難しいとされていた疑似乱数生成アルゴリズムの 1 つである Mersenne Twister (MT19937)を，マトリクスアーキテクチャ型超並列演算プロセッサ MX-1 を用いて，実装の工夫により並列化を実現した．実装結果については，プログラムのユーザマイクロコード化により処理速度が最大 22 倍向上するとともに，他のプロセッサとのスループットの比較を行い，ARM Coretex-A8 よりも約 29 倍，Xilinx XCV2000E よりも約 2.7 倍速い処理速度を得ることができた．

謝辞

本研究を行うにあたり，多くのアドバイスをしていただいたルネサスエレクトロニク株式会社の村田乾氏，野田英行氏に深く感謝いたします．

参考文献

[1] M. Matsumoto and T. Nishimura, “Mersenne Twister:A 623-dimentionally equidistributed uniform pseudo-random number generator,” ACM Trans. On Modeling and Computer Simulation, Vol. 8, No1, pp. 3-30, Jan. 1998

[2] M. Nakajima, H. Noda, K. Dosaka, K. Nakata, M. Higasida, O. Yamamoto, K. Mizumoto, H. Kondo, Y. Shimazu, K. Arimoto, K. Saito, and T. Shimizu, “A 40GOPS 250mW massively parallel based on matrix architecture”, ISSCC Dig. Tech. Papers, pp. 410-412, Feb. 2006

[3] Intel® Math Karnel Library Vector Statistical Library notes, Intel Corporation, 2007

[4] Xiang Tian, and Khaled Benkrid, “Mersenne Twister Random Number Generation on FPGA, CPU and GPU,” NASA/ESA Conference on Adaptive Hardware and Systems, pp. 460-464, Aug. 2009

[5] T. Tanizaki, T. Gyohten, H. Noda, M. Nakajima, K. Mizumoto, and K. Dosaka, “A super parallel SIMD processor with time/space conversion bus bridge on the matrix architecture,” IEICE Technical Report, vol. 106, no. 207, pp. 1-6, Aug. 2006

[6] H. Noda, T. Tanizaki, T. Gyoten, K. Dosaka, M. Nakajima, K. Mizumoto, K. Yoshida, T. Iwao, T. Nishijima, Y. Okuno, and K. Arimoto, “The circuits and robust design methodology of the massively parallel processor based on the matrix architecture,” Symp. VLSI Circuits Dig. Tech. Papers, pp. 260-261, June 2006

[7] T. Kumaki, M. Ishizaki, T. Koide, H. J. Mattausch, Y. Kuroda, H. Noda, K. Dosaka, K. Arimoto, and K. Saito, “Acceleration of DCT processing with massive-parallel memory-embedded SIMD matrix processor,” IEICE Trans. Inf. & Syst., vol. E90-D, no. 8, pp. 1312-1315, Aug. 2007

[8] T. Kumaki, T. Koide, H. J. Mattausch, Y. Kuroda, T. Gyoten, H. Noda, K. Dosaka, K. Arimoto, and K. Saito, “Integration architecture of content addressable memory and massive-parallel memory-embedded SIMD matrix for versatile multimedia processor,” IEICE Trans. Electron., vol. E91-C, no.9, pp. 1409-1418, Sept. 2008

[9] Y. Kono, T. Kumaki, M. Ishizaki, M. Tagami, T. Koide, H. J. Mattausch, T. Gyoten, H. Noda, Y. Kuroda, K. Dosaka, K. Arimoto, and K. Saito, “Super parallel SIMD processor with CAM based high-speed pattern matching capability,” IEICE Technical Report, vol. 106, no. 314, pp. 39-44, Oct. 2006

[10] T. Kumaki, Y. Kono, M. Ishizaki, M. Tagami, T. Koide, H. J. Mattausch, T. Gyoten, H. Noda, Y. Kuroda, K. Dosaka, K. Arimoto, and K. Saito, “CAM enhanced super parallel SIMD processor with high-speed pattern matching capability,” Proc. IEEE International Midwest Symposium on Circuits And Systems (MWSCAS’07), pp. 803-806, Aug. 2007

[11] 吉田 直之，松本 直樹，熊木 武志，藤野 毅, “超並列 SIMD 型演算プロセッサ MX-1 への Mersenne Twister の実装(1),” 電子情報通信学会ソサエティ大会，p. 1, Aug. 2010

[12] 松本 直樹，吉田 直之，熊木 武志，藤野 毅, “超並列 SIMD 型演算プロセッサ MX-1 への Mersenne Twister の実装(2),” 電子情報通信学会ソサエティ大会，p. 2, Aug. 2010

[13] 吉川 弘起，黒川 悠一朗，熊木 武志，藤野 毅, “超並列 SIMD 型演算プロセッサ MX-1 のためのガロア体演算による AES 用 SubBytes 変換の高速化,” 電子情報通信学会ソサエティ大会，p. 3, Aug. 2010

[14] 吉川 弘起，黒川 悠一朗，本田 弘，熊木 武志，藤野 毅, “超並列 SIMD プロセッサ MX-1 を用いた AES 暗号の高速処理実装,” SCIS2011，3D1-2, Jan. 2011

[15] 大澤 昌弘，板屋 修平，熊木 武志，藤野 毅, “超並列 SIMD 型演算プロセッサ MX-1 によるモルフォロジカルウェーブレット変換処理の実装,” 情報処理学会第 73 回全国大会，4H-2, Jan. 2011

[16] 黒川 恭一, 藤本 繁伸, “CPLD を用いた Mersenne Twister の開発,” 電子情報通信学会論文誌, 2001 巻, 84 号, pp. 501-504, May. 2001

[17] Shrutisagar Chandrasekaran, and Abbes Amira, “High Performance FPGA implementation of the Mersenne Twister,” 4th IEEE international Symposium on Electronic Design, pp. 482-485, Jan. 2008

[18] 黒川 恭一, 梶崎 浩嗣, “ Mersenne Twister の FPGA による実装,” 防衛大学校理工学研究報告, 40 巻, 2 号, pp. 15-21, Mar. 2003

[19] 渡辺 信吾, 阿部 公輝, “疑似乱数生成器 Mersenne Twister の VLSI 設計,” 情報処理学会研究報告, 2005 巻, 2005 号, pp. 13-18, May.2005

[20] High-performance Embedded Workshop, (http://japan.renesas.com/products/tools/ide/hew/hew_mid_level_landing.jsp)

[21] 米田 聡, “新「BeagleBoard」で最強 PC を作る,” 日経 Linux, 第 11 巻, 第 7 号通巻 118 号, pp. 55-68, Jul. 2009