コード生成容易なMATLAB上のデータ並列DSLによるプログラミングシステム

瀬川淳一[†] 城田祐介[†]
樽家昌也[†] 金井達徳[†]

近年のプロセッサはマルチコア化や SIMD 命令の採用で高速化しており，性能を引き出すにはプログラムの並列化が必要になっている．しかし，組込み機器向けの画像処理や信号処理のアルゴリズムの開発者にとって，マルチスレッド化や SIMD 化によるプログラムの並列化は容易でなく自動化が望まれている．この問題に対し，我々はデータ並列的な処理を簡潔に記述できる配列処理言語の開発を行ってきた．この配列処理言語は並列化 C コードへの変換が容易にできるという特徴があるが，開発者が新たに言語を習得しなければならないという課題があった．一方，アルゴリズム開発者には MATLAB や Octave などが広く使われているが，開発したアルゴリズムを元に C コードを再実装する場合，人手で並列化を行わなければならないことが多い．そこで，我々は配列処理言語の機能をデータ並列 DSL として MATLAB に統合したプログラミングシステムを開発した．データ並列 DSL では多重配列（配列を要素とする配列）を扱う関数を MATLAB の基本関数と組み合わせることで複雑なデータ並列のプログラムを簡潔に記述できるとともに，並列化 C コードの生成に必要な，配列のアクセス範囲や並列実行可能な処理単位を簡単な解析で検出できるため，自動並列化が容易である．データ並列 DSL は MATLAB 文法に準拠しているため習得のコストも小さく，MATLAB のライブラリや周辺ツールも利用できるため，アルゴリズム開発者の開発効率を高めることができる．

A Programming System Combining a Data-Parallel DSL Suitable for Code Generation with MATLAB

Jun'ichi Segawa,[†] Yusuke Shirota,[†] Masaya Tarui[†]
and Tatsunori Kanai[†]

Since current processors offer more parallelism by adding cores and SIMD instructions, fully exploiting parallelism is indispensable for generating high-performance codes. Automation of this process is important for algorithm-developers in the field of embedded systems including image processing and signal processing, who want to concentrate on their algorithms, rather than coping with the increasing complexity of the underlying processor architecture. To address this issue, we have been developing an array processing language which is a functional language dedicated to data-parallel processing on multicores. While the array processing language is suitable for parallel code generation, new language is not always easily accepted. On the other hand, prototyping tools such as MATLAB and Octave are widely accepted by algorithm developers in many fields. When developing softwares using MATLAB, however, MATLAB algorithms are generally reimplemented with C language by hand. To accelerate development, we propose a programming system using the array processing language redesigned as a data-parallel DSL on MATLAB. The DSL allows to naturally express algorithms, while enabling translation into C codes tuned for target processors, thus automating the reimplementation process. In addition, since the DSL is integrated with MATLAB, the language is easy to learn and development is accelerated using MATLAB's rich set of built-in utilities.

1. はじめに

近年のプロセッサは，搭載するコア数を増やし，SIMD 命令をサポートすることで性能を伸ばす傾向にあり，プログラムを高速化するには，マルチスレッド化や SIMD 化が欠かせない．そのため，組込み機器向けの画像処理や信号処理など，処理性能が求められる分野では，ソフトウェア開発の生産性が問題になっている．特に，ソフトウェア開発の中でもアルゴリズムの開発を主に行っている開発者にとって，マルチスレッド化や SIMD 化によりプログラムを並列化する

† 株式会社東芝 研究開発センター
Corporate Research & Development Center, Toshiba Corporation

ことは開発の本質ではないため，自動化できることが望ましい．

この問題に対し，これまで我々は配列処理ドメインに特化したプログラミング言語（以降，配列処理言語と呼ぶ）と並列化 C コードを自動生成するトランスレータを開発してきた[1)~4)]．配列処理言語は，ターゲットを画像処理や信号処理などの配列処理に絞り込み，データ並列的な処理を簡潔に記述できる関数や演算子を提供することで，プログラムの記述のしやすさと並列化 C コードの生成の容易さを両立させる設計にしている．しかしながら，開発者にとっては，新たに言語を習得しなければいけないという課題があった．

一方で，画像処理や信号処理などの分野で，MATLAB[5)] や Octave[6)] といったプロトタイプ開発ツールがアルゴリズム開発者を中心に広く使われている．MATLAB は，C/C++よりも抽象度の高いアルゴリズムレベルでプログラムを記述することができ，ライブラリや周辺ツールも充実しており使い勝手が良い．MATLAB は，アルゴリズムの評価を効率よく行うために用いられることが多く，評価したいアルゴリズムを記述し，インタプリタ上でシミュレーションを行うという使われ方が多い．ところが，MATLAB で評価したアルゴリズムを元に，組込み機器向けやシミュレーションの高速化向けに C コードを再実装する場合，人手で並列化を行わなければならないという問題があった．Real-Time Workshop Embedded Coder[7)] のように MATLAB から C コードを自動生成するツールもあるが，可読性の高い C コード生成が目的であり，マルチコアや SIMD 命令を活用するには，人手による並列化が避けられない．

そこで我々は，アルゴリズム開発者に広く利用されている MATLAB 上に，並列化 C コードの自動生成が容易な配列処理言語をベースにしたデータ並列 DSL(Domain Specific Language) を設計し，これを用いるプログラミングシステムを開発した．データ並列 DSL は，MATLAB の配列処理を記述するための基本機能はそのまま利用し，データ並列的な処理を簡潔に記述するための機能を MATLAB の関数として導入している．関数の導入で実現しているため，データ並列 DSL は MATLAB 文法に準拠しており，導入した関数も少数であることから，アルゴリズム開発者にとって習得しやすい．更に，データ並列 DSL は MATLAB 上に実装しているため，ライブラリや周辺ツールが充実した MATLAB 環境と連携させて開発効率を向上させることができる．

本論文では，MATLAB 上に構築したデータ並列 DSL と並列化 C コードの自動生成手法について説明する．まず，2 章でプログラミングシステムの概要について説明する．次に，3 章でデータ並列 DSL の説明を行い，続く 4 章でデータ並列 DSL からの並列化 C コードの生成手法について説明する．そして，5 章で生成した並列化 C コードの性能評価を行い，6 章で関連研究について述べる．

2. プログラミングシステムの概要

2.1 データ並列 DSL の設計

データ並列 DSL のベースとなる配列処理言語[1),2)] は，次のような特徴を持つプログラミング言語である．

- 1 つの配列から切り出した複数の部分配列を要素とする配列（以降，多重配列と呼ぶ）を基本のデータ構造として導入し，多重配列の生成や結合などの演算子を提供．
- 多重配列および単純な配列に対する基本演算の適用を関数型言語で用いられる高階関数 Map, Scan, Reduction を使って組み合わせることで，複雑なデータ並列型の配列処理プログラムを簡潔に記述可能．
- 変数が単一代入であり，処理の依存関係の解析が容易．

配列処理言語では，多重配列と高階関数により，多重配列の生成時に指定した部分配列の切り出しパターンや利用されている高階関数を解析することで，並列化 C コードが生成しやすくなる．

データ並列 DSL は，以上のような配列処理言語の特徴を MATLAB 上の DSL として設計したものである．MATLAB は元々配列処理向けの言語であり，データ並列 DSL では配列間の基本演算子などは MATLAB のものをそのまま利用する．その上で多重配列の演算と高階関数を新たに MATLAB 上の関数として導入し，関数を組み合わせて配列処理プログラムをループレスで記述できるようにしたのがデータ並列 DSL である．

2.2 プログラミングシステムによる開発フロー

図 1 に本プログラミングシステム全体の概要を示す．データ並列 DSL で記述した配列処理プログラムは，MATLAB のインタプリタ上で実行できるのに加え，トランスレータにより並列化 C コードに変換できる．トランスレータは 2 種類の並列化 C コードに対応している．MATLAB 上でのシミュレーション等の実行を高速化するためには，実行するプロセッサに合わせて並列化した C コードを MEX-file 形式 (MATLAB から呼び出し可能な C コード) で生成する．組

込み機器の開発など，C 言語で記述されたアプリケーションにデータ並列 DSL で記述したアルゴリズムを組み込むためには，並列化した実装用の C コードを生成する．

実装用の C コードは，ターゲットアーキテクチャに応じて自動生成するが，ターゲットに応じた生成方法の詳細については 1) を参照することとし，本論文では MATLAB が動作する Intel® プロセッサに焦点を絞って説明を行う．

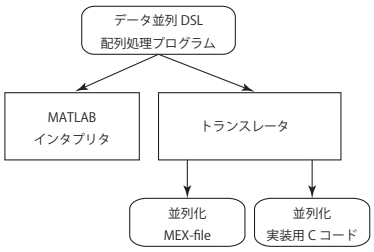

図 1 プログラミングシステムの概要
Fig. 1 Overview of the programming system.

本プログラミングシステムを用いた典型的な開発フローは次のようになる．開発の初期には，MATLAB インタプリタ上で実行し，MATLAB 環境のデバッガや可視化ツールを利用して開発を行う．そして，より大きなデータを用いたテストやシミュレーションは，トランスレータで MEX-file 化したプログラムを用いて行う．そして，開発したプログラムは実装用の並列化 C コードに変換してアプリケーションに組み込む．

3. データ並列 DSL

3.1 データ並列 DSL の主な関数

データ並列 DSL は，多重配列のための演算子や高階関数を MATLAB の関数として導入したものである．その中でも，特に画像処理や信号処理で頻繁に利用され，並列化 C コードを生成する際に重要な働きをするのが，

- 多重配列の切り出し関数 `MExtract()`
- 高階関数 `Map()`
- 変数の型指定の関数 `Type()`

である．

多重配列の切り出し関数 `MExtract()`

配列から複数の部分配列を切り出して多重配列を生成するのが `MExtract()` である．`MExtract()` は**図 2** に示すように，第 1 引数の配列から部分配列を繰り返し切り出し，それを要素とする多重配列 (図 2 の返り値 `N`) を返す．切り出しパターンは第 2〜5 引数で指定し，順に，切り出し開始位置の原点からの距離 `base`，切り出す部分配列の間隔 `step`，切り出す部分配列の各軸方向の個数 `size`，部分配列のサイズ `esize` をそれぞれベクトルで指定する．また，切り出した部分配列同士には重なりがあってもよい．生成された多重配列はサイズが `size` であり，要素がそれぞれサイズ `esize` の配列である．多重配列は，記述の簡潔さや並列化情報の抽出を目的として導入した仮想的なものであり，実行時に配列をコピーするような大きなオーバヘッドは生じない．

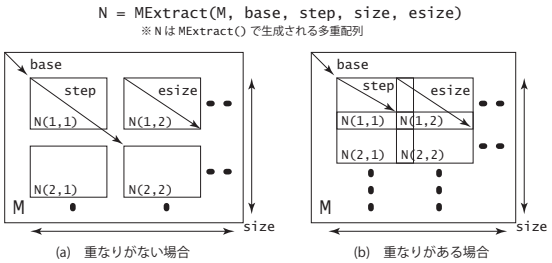

図 2 `MExtract()` による部分配列の切り出し
Fig. 2 Multiple sub-arrays extraction by `MExtract()`.

`MExtract()` が複数の部分配列を切り出して多重配列を返すのに対し，単一の部分配列を切り出す `Extract()` も用意している．第 1 引数で切り出す対象配列を，第 2 引数で切り出し開始位置の原点からの距離 `base`，第 3 引数で切り出す部分配列のサイズ `size` を指定する．`MExtract()`，`Extract()` ともに元の配列をはみ出す部分配列の指定が可能である．はみ出した部分には不定値が入っていると解釈する．これにより，画像処理や信号処理によく現れる配列の境界付近の例外処理の記述を簡略化している．

高階関数 `Map()`

`Map()` は，指定された関数を引数配列の要素に繰り返し適用し，その返り値を要素と同じインデックスに格納する関数である．第 1 引数で呼び出す関数への参照，第 2 引数以降に処理対象となる配列をそれぞれ指定する．**図 3** に示すように，`Map()` は `MExtract()` と組み合わせることで，C 言語などで多重ループで記述していた複雑な処理が簡潔に記述できる．

`Map()` 以外にも，データ並列 DSL では逐次的な配列処理を記述するための高階関数として `Scan()`，`Reduction()` を提供している．

Intel, Xeon は，米国およびその他の国における Intel Corporation の商標です．その他本論文に掲載の商品，機能等の名称は，それぞれ各社が商標として使用している場合があります．

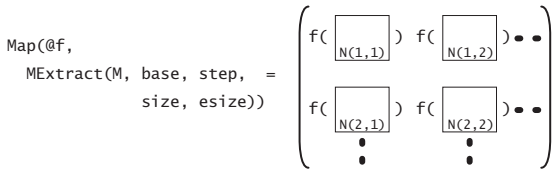

図 3 Map() と MExtract() の組み合わせ
Fig. 3 Map() and MExtract() combination.

型宣言関数 Type()

変数の型宣言に用いるのが Type() である．データ並列 DSL では，より高性能な C コードを生成するために，入出力変数，すなわち，データ並列 DSL で記述された配列処理プログラムと呼び出し側のプログラムの間でやりとりされるデータに対して，型宣言をすることができる．扱える型はスカラー型と配列型であり，スカラー型は 8/16/32/64 ビットの整数，単精度/倍精度の浮動小数点数と複素数に対応している．配列型は，多次元配列に対応しており，配列のサイズと要素の型で決まる．Type() の第 1 引数で型を指定し，第 2 引数で対象の入力変数（省略時は出力変数）を指定する．

3.2 データ並列 DSL による記述例

データ並列 DSL では，MExtract() と Map() を組み合わせることで配列処理をトップダウンに詳細化していくのが基本的なプログラミングスタイルである．これにより，画像処理や信号処理によく見られる典型的な配列処理をアルゴリズムに近い形で簡潔に記述することを可能にしている．画像処理や信号処理の代表的な具体例を用いて，データ並列 DSL によるプログラムの記述方法を説明する．

Prewitt フィルタ

図 4 は，画像のエッジ検出などに使われる Prewitt フィルタ[8] の記述例である．Prewitt フィルタは，各画素を中心とする部分配列に対して，畳み込み計算を行う処理を，画像全体に対して行うプログラムである．

データ並列 DSL では，トップの function 構文で外部から呼ばれる関数を定義する．Prewitt() の中では最初に，Type() を用いて入出力変数の型宣言を行っている．この例では，入出力配列ともにサイズが [480,720] で要素が uint8 の配列であることを示している．

Prewitt() の中では，入力画像 M から MExtract() を用いてサイズ [3,3] の部分配列を複数切り出して多重配列を生成し，切り出した各々の部分配列に対して畳み込み計算を行う関数 conv() を Map() で呼び出す．conv() は，水平方向と垂直方向のフィルタ定数による畳み込み計算を行う．conv() の中の Sum() は，配列の要素の総和を返す関数である．Sum() と同様に配列からスカラー値やインデックスを計算する関数として，Prod()，Max()，Min()，Maxpos()，Minpos() を提供しており（順に総積，最大値，最小値，最大値の要素のインデックス，最小値の要素のインデックス），これらを総称してアグリゲーション関数と呼んでいる．

また，Size() は，配列のサイズを返す関数である．

なお，MATLAB では配列を [1,2; 3,4] というように，要素を “[”, “]” で囲み行の区切りを “;” で記述する．配列の要素間の積，商は，行列積や逆行列の積との混同を避けるため “.*”，“./” と記述し，関数への参照は “@関数名” と記述する．

```
function img = Prewitt(M)
 Type({[480, 720], 'uint8'});
 Type({[480, 720], 'uint8'}, M);

 base = [-1, -1]; step  = [1, 1];
 size = Size(M);  esize = [3, 3];
 img = Map(@conv,
   MExtract(M, base, step, size, esize));
end

function ret = conv(m)
 fh = [ 1,  0, -1;
        1,  0, -1;
        1,  0, -1];
 fv = [-1, -1, -1;
        0,  0,  0;
        1,  1,  1];
 pixel = abs(Sum(fh .* m)) / 2 +
         abs(Sum(fv .* m)) / 2;
 if pixel > 255
  ret = 255;
 else
  ret = pixel;
 end
end
```

図 4 Prewitt フィルタプログラム
Fig. 4 Prewitt filter program.

OpticalFlow

次に，より複雑な画像処理の記述例として，OpticalFlow[9] の記述例を**図 5** に示す．OpticalFlow は，ブロックマッチングにより 2 枚の画像間の動きベクトルを検出するプログラムで，部分配列の切り出しが階層的に行われるのが特徴である．

```
function ret = OpticalFlow(P, C)
 Type({[60, 90], [2], 'int32'});
 Type({[480, 720], 'uint8'}, P);
 Type({[480, 720], 'uint8'}, C);

 Psize = [8, 8]; Csize = [32, 32];
 Pbase = [0, 0]; step = [8, 8];
 Cbase = Pbase - (Csize - Psize) / 2;
 blks = (Size(P)-Psize-Pbase) ./ step + 1;
 ret = Map(@Search,
   MExtract(P, Pbase, step, blks, Psize),
   MExtract(C, Cbase, step, blks, Csize));
end

function ret = Search(src, dst)
 thr = 30;
 bs = [0, 0]; st = [1, 1];
 sz = Size(dst) - Size(src) + 1;
 es = Size(src);
 if Max(src) - Min(src) >= thr
  pos = Minpos(Map(@SAD, Star(src),
           MExtract(dst, bs, st, sz, es)));
  ret = pos - (Size(dst) - Size(src)) / 2;
 else
  ret = [0, 0];
 end
end

function ret = SAD(x, y)
 ret = Sum(abs(x - y));
end
```

図 5 OpticalFlow プログラム
Fig. 5 OpticalFlow program.

OpticalFlow は，入力が 2 枚の画像で，出力が動きベクトルを要素とする配列である．OpticalFlow() では，前画像 P からサンプルブロックを，現画像 C からサンプルブロックの周囲のブロックをそれぞれ MExtract() で切り出す．そして，切り出したサンプルブロックとサンプルブロックの周囲のブロックの対応する組に対して，Search() を Map() を用いて適用する．Search() は，dst の中から src と最も近似度の高いブロックの位置を求める関数である．Search() は，近似度を求めるために MExtract() を用いて dst から src と同じ大きさのブロックをすべて切り出し，各ブロックと src との近似度を SAD(Sum of Absolute Differences) 計算で求め，一番近似度の高いブロックの位置を返す．Star() は Map() と組み合わせて使う関数であり，通常の Map() が指定された関数を引数配列の要素に適用するのに対し，Star() が用いられている引数に対しては，引数配列そのものに対して適用する．なお，Search() が if 文になっているのは，src 内のコントラストが低い場合の有効なマッチング計算ができないことによる無駄な計算を省くためである．

CFAR 処理

最後に信号処理の記述例として，レーダ信号処理の一つである CFAR (Constant False Alarm Rate) 処理[10] の記述例を示す．CFAR 処理とはクラッタと呼ばれる目標物以外からの不要な反射波を抑圧して目標物からの反射波を抽出する処理である．**図 6** に示すように，受信信号 V の注目点に対し，近距離側 near と遠距離側 far の部分配列の総和を求め注目点の値を総和値で割り対数を取る処理を受信信号に対して繰り返し行うものである．

```
function ret = CFAR(V)
 Type({[2048], 'single'});
 Type({[2048], 'single'}, V);

 n = 16;
 ret = Map(@f, V,
           MExtract(V, [-(n + 1)], [1],
                    Size(V), [n]),
           MExtract(V, [2], [1],
                    Size(V), [n]));
end

function ret = f(p, near, far)
 d = Sum(near) + Sum(far);
 ret = 10 * log10(32 * p / d);
end
```

図 6 CFAR 処理プログラム
Fig. 6 CFAR program.

4. トランスレータによる並列化 C コードの生成

4.1 並列化 C コード生成の概要

データ並列 DSL で記述されたプログラムは，

- 高階関数を用いてループレスで配列処理が記述されており，ソースコードレベルでのプログラム変換が容易
- 並列化可能な処理単位が明確で並列化 C コード

の生成が容易
という性質を持つ．

この性質を踏まえ，トランスレータは，
(1) ソースコードレベルのプログラム変換
(2) トップレベルの並列処理のマルチスレッド化
(3) ボトムレベルの並列処理の SIMD 化
の 3 つの変換ステップでデータ並列 DSL で記述されたプログラムを並列化 C コードに変換する．各変換ステップについて，これ以降で説明する．

4.2 プログラム変換

データ並列 DSL のソースコードレベルのプログラム変換を容易に実現できるという性質を利用し[4]，トランスレータでは，SIMD 化に適したプログラム構造に変換するための MExtract2Extract 変換[3] や，MEX-file 生成時に必要となる C 言語と MATLAB の配列のメモリ上の配置の違いを吸収するための変換などを行う．

MExtract2Extract 変換は，`MExtract()` と `Map()` の組み合わせで直観的に記述された画像フィルタ処理などのプログラムを，SIMD 化向けのプログラム構造に変えるプログラム変換である．**図 7** は，MExtract2Extract 変換を行う前の Prewitt フィルタプログラムから生成した C コードの概要である．変換前のプログラムは，`MExtract()` で切り出す不連続データに対して処理を行うため，生成される C コードは，最内ループの繰り返し数が少なく SIMD 化しても効果は小さい．

```
for (i = 1; i < 479; i++) {
 for (j = 1; j < 719; j++) {
  sum_h = 0; sum_v = 0;
  for (y = 0; y < 3; y++) {
   for (x = 0; x < 3; x++) {
    sum_h += fh[y*3+x] *
             M[((i-1)+y)*720+(j-1)+x];
    sum_v += fv[y*3+x] *
             M[((i-1)+y)*720+(j-1)+x];
   }
  }
  pixel = abs(sum_h)/2 + abs(sum_v)/2;
  ret[i*720+j] = pixel>255 ? 255 : pixel;
 }
}
```

図 7 Prewitt フィルタプログラムの C コードの概要
Fig. 7 Pseudo C code of Prewitt filter program.

図 8 に示すように，MExtract2Extract 変換は，`MExtract()` を用いるプログラムを `Extract()` を用いるプログラムに変換することで連続領域の大きいデータに対する処理に変える．SIMD 化は連続領域の大きいデータに対する処理に対して適用したほうが効率的である．そのため，`MExtract()` を `Extract()` に変更し `Map()` で呼び出す関数を処理が等価になるように変換する MExtract2Extract 変換を行うことで，SIMD 化に適したプログラムに変換できる．

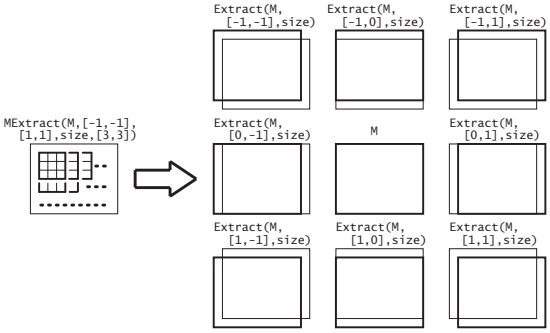

図 8 MExtract2Extract 変換
Fig. 8 MExtract2Extract transformation.

MExtract2Extract 変換ルールを図 4 の Prewitt フィルタに適用した結果が**図 9** のプログラムである．`MExtract()` で細かい部分配列を切り出して畳み込み計算を行っていた処理が，`Extract()` による大きな部分配列の切り出しと，切り出した部分配列の要素を参照して計算を行う処理に変換される．

図 9 の MExtract2Extract 変換後のプログラムから生成した C コードの概要が，**図 10** である．サイズの大きい連続した部分配列に対する処理に変換されているため，最内ループの繰り返し数が十分に確保されており，ループの段数も減らせているため，ループのオーバヘッドも削減できる．

一方，MEX-file 生成時に必要となる C 言語と MATLAB の配列のメモリ上の配置の違いも，同様にプログラム変換にて対応する．MEX-file が MATLAB プログラムから受け取る配列は，メモリ上での配置が C 言語と異なり，2 次元配列は列優先で配置されており，多次元配列でも同様に配置が逆転している．そのため，MEX-file を生成するには，入出力配列のメモリ上での配置の違いを吸収するための変換を行わなければならない．この変換もデータ並列 DSL では，`MExtract()` や `Extract()` の切り出しパターンのサイズ指定の各軸の入れ替えや，定数配列の軸の入れ替え，および，入出力配列のサイズの各軸の長さを入れ替えを行うプ

```
function img = Prewitt(M)
  .....
 img = Map(@f, Extract(M, [-1, -1], size),
               Extract(M, [-1,  0], size),
               Extract(M, [-1,  1], size),
                 .....
               Extract(M, [ 1,  1], size));
end

function ret = f(m11, m12, m13, ... , m33)
  pixel = abs(m11 - m13 +
              m21 - m23 +
              m31 - m33) / 2 +
          abs(-m11 - m12 - m13 +
               m31 + m32 + m33) / 2;
  if pixel > 255
    ret = 255;
  else
    ret = pixel;
  end
end
```

図 9 MExtract2Extract 変換後の Prewitt フィルタプログラム
Fig. 9 Transformed Prewitt filter program using MExtract2Extract.

```
for (i = 1; i < 479; i++) {
 for (j = 1; j < 719; j++) {
  pixel = abs(
   M[(i-1)*720+(j-1)] - M[(i-1)*720+(j+1)] +
   M[( i )*720+(j-1)] - M[( i )*720+(j+1)] +
   M[(i+1)*720+(j-1)] - M[(i+1)*720+(j+1)])/2
    + abs(
   -M[(i-1)*720+(j-1)] - M[(i-1)*720+( j )] -
    M[(i-1)*720+(j+1)] + M[(i+1)*720+(j-1)] +
    M[(i+1)*720+( j )] + M[(i+1)*720+(j+1)])/2;
  ret[i*720+j] = pixel > 255 ? 255 : pixel;
 }
}
```

図 10 MExtract2Extract 変換後の Prewitt フィルタプログラムの C コードの概要
Fig. 10 Pseudo C code of transformed Prewitt filter program using MExtract2Extract.

ログラム変換ルールを適用することで実現できるため対応が容易である．

4.3 マルチスレッド化

続いて，マルチスレッド化，SIMD 化を行うため，並列化可能な処理単位の検出を行う．

データ並列 DSL では，配列に対して，`Map()` による関数適用，アグリゲーション関数の適用を行っている箇所を見つけるだけで並列に実行可能な処理単位を取り出すことができる．`Map()` による関数適用は，呼び出される関数の適用が互いに独立であり，また，アグリゲーション関数の適用も，各要素の演算間，あるいはそれらを複数のグループに分けて部分的に集約する演算間には依存関係が無いためである．また，`Map()` と組み合わせて用いられている `MExtract()` の切り出しパターンから，各処理単位を実行するのに必要な配列の範囲を求めることができる．このようにデータ並列 DSL で記述されたプログラムでは，並列化に必要な解析対象が絞り込まれている．

こうした並列化に必要な情報を従来の C 言語などループで記述された配列処理から抽出するには，処理の依存関係や，実行における配列の参照・更新の範囲の特定など複雑な解析が必要であったが，データ並列 DSL は，必要な解析対象が絞り込まれており，解析を大幅に簡素化することができる．

並列に実行可能な処理単位は，`Map()` で呼ばれる関数が入れ子になる場合があるので，階層的になっている．そのうちのトップレベルをマルチスレッド化，ボトムレベルを SIMD 化するのがこれ以降の並列化の基本的な流れである．(図 10 では，インデックス変数 `i`, `j` の `for` 文がマルチスレッド化の対象であり，SIMD 化の対象は，マルチスレッド化で分割されたインデックス `j` の `for` 文である．)

トップレベルの並列に実行可能な処理単位を取り出して，単純にスレッドに分配してしまうと，処理単位の粒度が細かすぎてオーバヘッドが大きくなったり，実行する順序によってはキャッシュミスが多く発生する場合がある．そこで，並列化可能な処理単位を，レンジと呼ぶ単位でグルーピングしてスレッドに分配する単位とし，レンジの実行順序をキャッシュのヒット率が高くなるように制御するレンジスケジューリング[1)]を導入した．

レンジのサイズ，すなわち 1 つのレンジに属する並列に実行可能な処理単位の数は，レンジを実行中のメモリアクセス量とコア内部の L1 キャッシュの容量から算出する．あるレンジを実行中のメモリアクセス量は，`MExtract()` の切り出しパターンと配列の要素の型の大きさから近似解を算出できる．こうして決定したレンジサイズで入出力配列を分割してレンジのプールを用意する．そしてプールの中から各コアの負荷状況に応じて順次レンジを分配することで，各コアの負荷を均等化する．このとき，各コアに隣接するレンジ

表 1 評価環境
Table 1 Evaluation environment.

Intel® Xeon® Processor X3440	2.53GHz
L1 データキャッシュ	32KB 8-way
コア数	4
C コンパイラ	gcc 4.4.3(-O3)
MATLAB	2011b

を分配することで，キャッシュ上のデータをできるだけ再利用できるように処理順序を制御する．

4.4 SIMD 化

MExtract2Extract 変換による SIMD 化しやすいプログラム構造への変換や，キャッシュを考慮したマルチスレッド化によるメモリアクセスの最適化により，SIMD 化の効果を大きくするための条件は整っている．SIMD 化は，基本的にはマルチスレッド化のために分割したレンジ内の連続データに対する処理を，C コンパイラのイントリンシックを利用した SIMD 化コードへの変換を行う．イントリンシックを利用することでレジスタ割り当てや命令スケジューリングはコンパイラに任せており，トランスレータでは，SIMD 化を行う上で一般的な最適化テクニック[12] である，整数演算の中間変数の必要最低限のビット長を求めて並列度を高めるビット長推論，アライメント最適化，レジスタ上でのデータの再利用を行っている．C コンパイラの自動ベクトル化のように複雑な依存解析に左右されず，より確実に SIMD 化できるのが特徴である．

5. 性 能 評 価

データ並列 DSL から生成された並列化 C コードを評価するため，代表的な画像処理や信号処理のプログラムで性能評価を行った．測定に用いたプログラムは，3 章で記述例を示した Prewitt フィルタ, OpticalFlow, CFAR 処理，そして MIP 法によるボリュームレンダリング[11]，である．これらのプログラムをデータ並列 DSL で記述してトランスレータにより MEX-file を生成し，それを MATLAB プログラムから呼び出して性能測定を行った．評価に用いた環境は，**表 1** に示すとおりである．

5.1 並列化を行わない場合の性能

最初に，MExtract2Extract 変換，マルチスレッド化，SIMD 化を行わない並列化無し (Off) の C コードが妥当な性能であるかを調べた．まず，並列化無し (Off) の設定にしたトランスレータで MEX-file を生成し，続いて，同じアルゴリズムを通常の MATLAB で記述し，それを MATLAB プログラムの C コード自動生成ツールである Real-Time Workshop Embedded Coder で MEX-file を生成し，性能比較を行った．その結果，トランスレータが生成した MEX-file が，Real-Time Workshop Embedded Coder の生成した MEX-file に対して 1.0〜7.3 倍と同程度以上の処理性能であることが確認できた．Real-Time Workshop Embedded Coder は，マルチスレッド化や SIMD 化による並列化は行っていないため，この結果から並列化無し (Off) の段階で，生成された C コードに大きなオーバヘッドが無いことが確認できた．

5.2 並列化による性能向上

続いて，トランスレータによる変換ステップを，

(1) 並列化無し (Off)
(2) MExtract2Extract 変換 (M2E)
(3) MExtract2Extract 変換＋マルチスレッド化 (Mthread)
(4) MExtract2Extract 変換＋マルチスレッド化＋SIMD 化 (SIMD)

と，順に変換ステップを増やしていき，それぞれの変換ステップにおける並列化無しのプログラムに対する性能向上率を測定した．

各変換ステップによる性能向上の割合を**表 2** に示す．

表 2 並列化による性能向上率
Table 2 Speed-ups of the optimized programs.

プログラム（入力配列）		Speed-up
Prewitt	Off	1.0
(480x720 uint8)	M2E	25.3
	Mthread	57.4
	SIMD	373.3
OpticalFlow	Off	1.0
(480x720 uint8)	M2E	1.0
	Mthread	3.6
	SIMD	6.5
CFAR	Off	1.0
(2048 single)	M2E	3.2
	Mthread	3.1
	SIMD	13.4
Volume Rendering	Off	1.0
(256x256x256 uint16)	M2E	1.0
	Mthread	3.5
	SIMD	4.2

MExtract2Extract 変換が適用されるのは，入力配列の部分配列に対して畳み込み計算や総和計算を行う処理に対して行うプログラム変換であるため，性能評価に用いたプログラムの内，対象となるのは Prewitt フィルタ，CFAR 処理である．どちらも性能向上が確認できるが，特に Prewitt フィルタの方が性能向上率が高い．これは処理対象の配列が大きく，プログラム変換に伴うループのオーバヘッドの削減効果が大きい

ためである．

マルチスレッド化は，Prewitt フィルタ，OpticalFlow，ボリュームレンダリングで性能向上が確認できた．一方で，CFAR 処理ではほとんど性能向上が見られない．これは処理対象の配列が他の画像処理のプログラムと比べて小さく，マルチスレッド化によるオーバヘッドが相対的に大きくなったためである．そのため，処理対象の配列の大きさによっては，マルチスレッド化をしない方がより高速に処理できる場合があり，この判定を自動化するのが今後の課題である．

SIMD 化は，すべてのプログラムで性能向上が確認できており，特に Prewitt フィルタの性能向上率が高い．これは，ビット長推論にて必要最低限のビット長を求めることで，8 並列の SIMD 化を適用できることがわかり，並列度の高い SIMD 命令が適用されているためである．

以上の性能評価から，データ並列 DSL を用いて記述された画像処理や信号処理の代表的なプログラムからマルチスレッド化，SIMD 化を行った効率的な C コードが自動生成できることが確認できた．

次に，トランスレータが自動生成した並列化 C コードが人手で並列化したプログラムに対してどれだけの性能劣化に抑えることができたかを測定した．Prewitt フィルタ，CFAR 処理，ボリュームレンダリングは，1.1 倍〜2.1 倍の性能劣化に抑えられており，並列化 C コードを人手で開発する手間を考慮すると十分な性能と考えられる．一方 OpticalFlow は，人手による並列化が SAD のための SIMD 命令を活用しており，この SIMD 命令の生成に現在のトランスレータの実装が対応していないことから 16.6 倍の性能劣化となった．今後，トランスレータの実装を進めることで，特殊な SIMD 命令にも対応していく予定である．

6. 関連研究

並列プログラミング言語の研究の中でも X10[13] や Sequoia[14] は，ターゲットアーキテクチャを抽象化し，その上でプログラムの分割をユーザが明示する手法を取っている．これに対し，本研究は，プログラムに記述されている暗黙的な分割ポイントの情報を抽出し，それをターゲットアーキテクチャに応じて分割するところまで自動化しているところが特徴である．

Intel ArBB[15] は，C++で配列処理を簡潔に記述するためのライブラリである．ArBB に比べ，本研究のデータ並列 DSL が，多重配列と `Map()` の組み合わせにより複雑な配列処理をより簡潔に記述できる．

5 章の性能評価で利用した Real-Time Workshop Embedded Coder は，MATLAB から標準的な可読性の高い C コードの生成を目的としており，特定のプロセッサ向けの並列化は行っていないが，データ並列 DSL より多様な MATLAB プログラムからの C コード生成が可能であるため，互いに補完する関係である．

7. おわりに

本論文では，提案するプログラミングシステムを用いることで，画像処理や信号処理のプログラムをデータ並列 DSL で簡潔に記述することができ，トランスレータにより効率の良い並列化 C コードが生成できることを示した．データ並列 DSL によるプログラミングは，並列化を意識する必要がないため，アルゴリズム開発者にとって使い易い．更に，データ並列 DSL が MATLAB 文法に準拠しているため，習得のコストも小さく，また，MATLAB との連携もできるため MATLAB の充実したライブラリや周辺ツールを利用することができ，実用性が高い．

今後もプロセッサは複雑化する一方であり，最近では高負荷な処理をアクセラレータにオフロードして高速化する手法の利用も盛んになってきていることから，必要とされる並列化手法はますます複雑化，多様化していく傾向にある．本論文で提案する手法は，並列化に必要なプログラム解析が容易であり，並列化のための抽象度の高いプログラム変換も容易であることから，様々な並列化手法を導入しやすい[4]．更にフロントエンドが MATLAB であることから，アルゴリズム開発者が最新の並列化手法を手軽に利用できるプログラミングシステムとして期待できる．

本研究の今後の課題はアーキテクチャ展開とアプリケーション展開である．今後は GPU や FPGA に代表されるアクセラレータへの対応を行うことで，適用可能なプロセッサのバリエーションを増やす．アクセラレータは，配列処理の高速化に向いているがプログラミングが難しいため，本研究の手法が効果的に適用できると考えられる．また，本論文では代表的な画像処理や信号処理のプログラムで性能評価を行ったが，より多様なアプリケーションの配列処理に適用することで，適用可能性の評価や新たな並列化手法の開拓につなげていく．

参考文献

1) 金井達徳, 瀬川淳一, 武田奈穂美: 組み込みプロセッサのメモリアーキテクチャに依存しない画像処理プログラムの記述と実行方式, 情報処理学会論文誌コンピューティングシステム, Vol.48, No.SIG

13(ACS 19), pp.287–301 (2007).
2) 瀬川淳一, 金井達徳, 城田祐介: 配列処理向けドメイン特化言語によるマルチコアプログラミング, 情報処理学会研究報告, Vol.2008-ARC, No.75, pp.19–24 (2008).
3) 城田祐介, 瀬川淳一, 金井達徳: 配列処理言語における SIMD 化向けプログラム変換, 情報処理学会研究報告, Vol.2008-HPC, No.74, pp.193–198 (2008).
4) Shirota, Y., Segawa, J., Tarui, M. and Kanai, T.: Autotuning in an Array Processing Language using High-level Program Transformations, International Workshop on Automatic Performance Tuning(2011 年 6 月発表予定).
5) MATLAB: http://www.mathworks.co.jp/products/matlab/
6) Octave: http://www.gnu.org/software/octave/
7) Real-Time Workshop Embedded Coder: http://www.mathworks.com/products/rtwembedded/
8) 田村秀行: コンピュータ画像処理, オーム社 (2002).
9) 安居院猛, 長尾智晴: C 言語による画像処理入門, 昭晃堂 (2000).
10) 吉田孝: 改訂レーダ技術, 電子情報通信学会 (1996).
11) 今里悠一, 大橋昭南: 医用画像処理, 昭晃堂 (1993).
12) Bik, A.J.C.: The Software Vectorization Handbook, Intel Press (2004).
13) X10: http://x10.sourceforge.net/
14) Fatahalian, K., Knight, T.J., Houston, M., Erez, M., Horn, D.R., Leem, L., Park, J.Y., Ren, M., Aiken, A., Dally, W.J. and Hanrahan, P.: Sequoia: Programming the Memory Hierarchy, Proceedings of the 2006 ACM/IEEE conference on Supercomputing (2006).
15) Intel ArBB: http://software.intel.com/en-us/articles/intel-array-building-blocks/