浮動小数点 DSP デザイン・フローと アルテラ FPGA 上でのパフォーマンスに関する 第三者機関による分析

Berkeley Design Technology, Inc.

2012 年 10 月

概要

FPGA は、高負荷なデジタル信号処理アプリケーション向けの並列処理エンジンとして使用されることが多くなっています。ベンチマーク結果によれば、FPGA は、高度に並列化可能なワークロードの場合、DSP プロセッサや汎用 CPU よりも高い性能と優れたコストパフォーマンスを実現することができます。しかし、これまではほぼ例外なく固定小数点信号処理デザインのみに使用されています。FPGA は、高性能な浮動小数点演算を必要とするアプリケーション向けの有効なプラットフォームと見なされていません。FPGA の浮動小数点効率と性能は、長い処理レイテンシと配線密集によって制限されます。加えて、従来の FPGA デザイン・フローは、Verilog または VHDL によるレジスタ転送レベルのハードウェア記述に基づいており、複雑な浮動小数点アルゴリズムの実装には適していません。

アルテラは、アルテラ FPGA への浮動小数点デジタル信号処理アルゴリズムの実装プロセスを合理化し、従来よりも高い性能と効率を実現するために、新しい浮動小数点デザイン・フローを開発しました。そのデザイン・フローでは、例えば、乗算、加算、二乗を順次実行するといった、基本的な浮動小数点演算子からなるデータパスを構築するのではなく、浮動小数点コンパイラによって基本的な演算子を単一の関数またはデータパスに統合したフューズド・データパスを生成します。それにより、従来の浮動小数点 FPGA デザインに存在する冗長性を排除しています。また、アルテラのデザイン・フローは、アルテラの DSP Builder アドバンスト・ブロックセットと MathWorks の MATLAB および Simulink ツールによるハイレベル・モデル・ベースのデザイン・フローです。アルテラは、FPGA 設計者がハイレベルで作業することにより、複雑な浮動小数点アルゴリズムの実装と検証を従来の HDL ベースのデザインよりも素早く行えるようになると期待しています。

BDTI は、アルテラの浮動小数点信号処理デザイン・フローに関して独自に分析を行いました。目的は、高負荷な浮動小数点信号処理アプリケーションに対してアルテラ FPGA で達成できる性能、およびアルテラの浮動小数点信号処理デザイン・フローの使いやすさを評価することでした。本書では、背景および方法論の詳細とともに BDTI の所見を示します。

目次

1. はじめに

2 つの浮動小数点デザインの例

デジタル・チップの進歩により、従来は研究環境に限られていた複雑なアルゴリズムを組込みコンピューティング・アプリケーションの分野に応用することが可能になりつつあります。例えば、大規模な計算リソースが利用可能であり、通常はリアルタイム計算を必要としない研究環境では、長年、主に線形代数が使用されてきました。その具体的な目的は、多数の連立一次方程式を含む系の解を求めることです。大規模な系の解を求めるには、逆行列演算やある種の行列分解が必要です。これらの手法では、計算負荷が非常に高いことに加え、十分なダイナミック・レンジを使用しないと、数値的不安定に悩まされることがあります。そのため、そうしたアルゴリズムを効率的かつ正確に実装するには、浮動小数点デバイスが事実上不可欠でした。

アルテラは最近、従来の FPGA デザイン手法に比べて浮動小数点デザインの性能と効率を改善するとともに、アルテラ FPGA 上での浮動小数点 DSP アルゴリズムの実装を簡素化するために、DSP Builder アドバンスト・ブロックセット・ツール・チェインに浮動小数点機能を導入しました。前のホワイトペーパー[1]では、BDTI は 40nm Stratix IV および Arria IV FPGA デバイス向けに合成したシングル・チャネル浮動小数点コレスキー・ソルバの実装を例に、アルテラ Quartus II 開発ソフトウェア v11.0 ツール・チェインの性能および効率を分析・評価しました。

このホワイトペーパーでは、最新版である Quartus II 開発ソフトウェア v12.0 ツール・チェインを使用してアルテラのアプローチの有効性を評価するほか、アルテラの 28nm Stratix V および Arria V FPGAの性能を評価します。この評価では、マルチチャネル・コレスキー行列分解とグラム・シュミット法による QR 分解という 2 つのタイプの分解を用いて多数の連立一次方程式の解を求めることに焦点を当てます。これらの分解と前進代入および後退代入を併用して、連立一次方程式 **Ax** = **B** のベクトル **x** の値を求めます。

行列分解は、時空間適応信号処理 (STAP) などの高度な各種軍用レーダー・アプリケーションや、デジタル通信におけるさまざまな推定問題に使用されています。QR 分解は、m x n の一般行列 **A** によく使用されます。それに対し、コレスキー分解は計算効率が高いことから、正方行列、対称行列、および正定値行列に適したアルゴリズムです。いずれの分解も、計算負荷が非常に高いアルゴリズムを使用し、高いデータ精度を要求するため、浮動小数点演算が不可欠です。さらに、本書で評価する 2 つのサンプルは、線形代数と有限インパルス応答 (FIR) フィルタを含むさまざまなデジタル信号処理アプリケーションに見られるベクトル・ドット積と入れ子ループをアルゴリズムの中核に使用します。

表記と定義

M 太字の大文字は行列を表します。
z 太字の小文字はベクトルを表します。

L* 行列 L の共役転置。
l* 要素 l の共役転置。

エルミート行列 自己の共役転置に等しい複素正方行列。これは実対称行列の複素拡張です。

正定値行列 エルミート行列 **M** は、すべて 0 でない複素ベクトル z について **z*Mz** > 0 の場合、正定値です。**M** は本書の目的上、エルミート行列であるため、量 **z*Mz** は常に実数です。

正規直交行列 行列 **Q** は、$\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$ (**I** は恒等行列)である場合、正規直交行列です。*コレスキー分解* **M** = **LL*** となるように正定値エルミート行列 **M** を下三角行列 **L** と共役転置 **L*** に分解すること。

QR 分解 サイズが *m* x *n* の行列 **M** を **M** = **QR** となるように、サイズが *m* x *n* の正規直交行列 **Q** とサイズが *n* x *n* の上三角行列 **R** に因数分解すること。

F_{max} FPGA デザインの最大周波数。

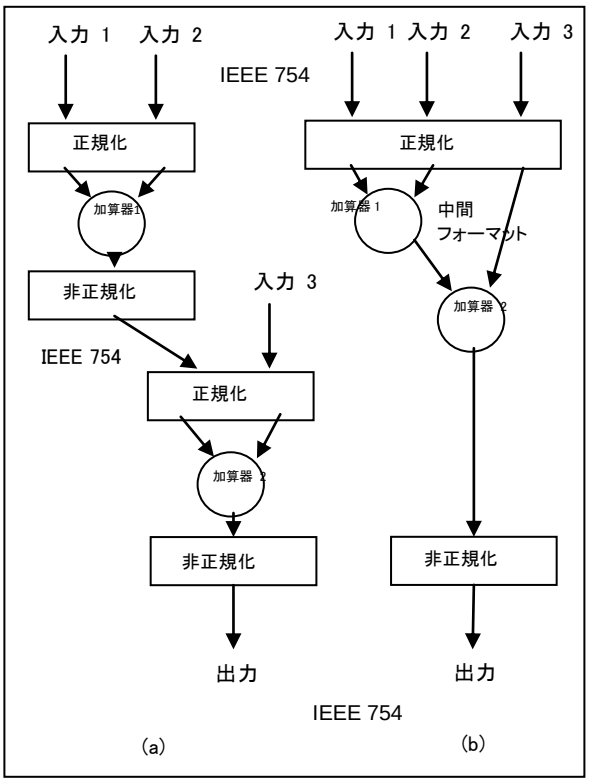

図 1 (a) 従来の浮動小数点実装、
(b) フューズド・データパスによる実装

本書で説明する実装例では、両方の方法で複素数データ型の連立一次方程式の解を求め、その結果を示します。アルテラ Stratix V FPGA は、セクション 4 で示す QR ソルバを使用した場合、サイズが 400 x 400 の行列に対して 1 秒当たり 315 回の行列変換を実行することが可能であり、203 MHz 動作時で 162 GFLOPS (1 秒当たり 162 x 10^9 回の浮動小数点演算) の浮動小数点演算性能を達成します。

本書で評価するデザイン例はいずれも、DSP Builder ソフトウェア v12.0 の一部として、提供されています。入手のお問い合わせは、floatingpoint@altera.com までお寄せください。

浮動小数点デザイン・フロー

従来、FPGA は高負荷な浮動小数点アプリケーションに適したプラットフォームではありませんでした。FPGA ベンダーは浮動小数点プリミティブ・ライブラリを提供していますが、浮動小数点アプリケーションにおける FPGA の性能は非常に限られています。従来の浮動小数点 FPGA デザインの効率が低い理由の 1 つは、深いパイプライン化と浮動小数点演算子の広い演算構造によって、大きなデータパス・レイテンシと配線密集が生じることです。レイテンシは、データ依存性が高いデザインにおいて、対処が難しい問題をもたらし、最終的に得られるデザインは動作周波数の低いデザインにつながることが少なくありませんでした。

アルテラ DSP Builder アドバンスト・ブロックセット・ツール・フローは、アーキテクチャ・レベルとシステム・デザイン・レベルの両面からこの問題に対処しています。アルテラの浮動小数点コンパイラは、基本的な浮動小数点演算子の組み立てによってデータパスを構築するのではなく、データパスの大部分を単一の浮動小数点関数に統合します。これは、図 1 に示すように、データパス全体に十分な精度を割り当てて正規化および非正規化ステップをできる限り省くことが目的であり、データパス内のビット増加を分析し、最適な入力正規化を適切に選択することによって行われます。IEEE 754 形式を使用するのはデータパス境界のみで、データパス内ではより大きな仮数幅を使用してダイナミック・レンジを広げ、連続する演算子間での非正規化および正規化処理の必要性を減らします。正規化および非正規化処理の実現には、単精度浮動小数点数の場合、最大 48 ビット幅のバレル・シフタが必要となります。これは大量のロジックと配線リソースを消費し、FPGA への浮動小数点実装が効率的でない主な原因となります。*フューズド・データパス・メソドロジ*は、これらのバレル・シフタの数を大幅に削減します。より高い精度の仮数を必要とする乗算には、Stratix V および Arria V デバイスの 27 x 27 乗算器モードが使用されます。図 1 (b) は、フューズド・データパス・メソドロジによる 2 つの加算器チェインの実装を図 1 (a) の従来の実装と比較したものです。図 1 (b) のフューズド・データパスでは、第 1 加算器の出力の非正規化、および第 2 加算器の入力の正規化が省かれるため、演算子間の冗長性が排除されます。余分なロジック/ルーティングの排除とハード乗算器の使用により、複雑なデータパス間のタイミングとレイテンシが予測可能になります。単精度と倍精度のいずれの IEEE 754 浮動小数点アルゴリズムも、従

来よりも少ないロジックで実装でき、より高い性能が得られます。アルテラによれば、フューズド・データパスを使用することにより、基本的な演算子で構成された同等のデータパスに比べて、ロジックの 50% 削減とレイテンシの 50% 低減につながります [2]。また、内部データ表現の幅が広くなるため、基本的な IEEE 754 浮動小数点演算子を含むライブラリを使用する場合に比べて、全体的なデータ精度が概して高くなります。

アルテラの浮動小数点 DSP デザイン・フローには、アルテラの DSP Builder アドバンスト・ブロックセット、Quartus II 開発ソフトウェア RTL ツール・チェイン、ModelSim シミュレータのほか、MathWorks の MATLAB および Simulink ツールが組み込まれています。Simulink 環境では、複雑なシステムの記述、デバッグ、および検証をアルゴリズム動作レベルで行うことが可能です。DSP Builder アドバンスト・ブロックセットには、データ型伝播やベクトル・データ処理などの Simulink の機能が組み込まれており、アルゴリズム・デザイン・スペースを素早く探索することができます。

本書では、DSP Builder アドバンスト・ブロックセット・ツールを使用して、単精度浮動小数点表現を使用した複素数データ型の連立一次方程式の解を求める 2 つの例を検証することにより、アルテラの浮動小数点デザイン・フローの効率と性能を評価します。一方ではコレスキー分解を使用し、もう一方ではグラム・シュミット法による QR 分解を使用します。セクション 2 では、この 2 つの浮動小数点実装例について説明します。セクション 3 では、デザイン・フローおよびツール・チェインを実際に使用して気付いた点について述べます。セクション 4 では、ハイエンドの Stratix V 5SGSMD5K2F40C2N デバイスとミッドレンジの Arria V 5AGTFD7K3F40I3N デバイスという 2 種類のアルテラ FPGA への実装の性能を示します。最後に、セクション 5 で BDTI の結論を示します。

2. 実装

背景

Ax = **b** という形の一連の一次方程式は、多くのアプリケーションで登場します。線形最小二乗を含む最適化問題にしても、予測問題のためのカルマン・フィルタにしても、MIMO コミュニケーション・チャネル推定にしても、一連の一次方程式 **Ax** = **b** の数値解を求めることが問題であることに変わりありません。サイズが *m* x *n* の一般行列の場合 (*m* は行列の高さ、*n* は行列の幅)、QR 分解を使用してベクトル **x** の値を求めることができます。アルゴリズムでは、**A** をサイズ *m* x *n* の正規直交行列 **Q** とサイズが *n* x *n* の上三角行列 **R** に分解します。**Q** は正規直交行列であるため、$\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$ かつ $\mathbf{Rx} = \mathbf{Q}^T\mathbf{b}$ となります。**R** が上三角行列であることから、元の行列 **A** の逆行列を求めなくても、後退代入によって **x** の値を容易に求めることができます。本書で示す QR 分解の例では、$m \geq n$ の優決定系の行列を扱います。

多くの問題で登場する共分散行列などのように、行列 **A** が対称かつ正定値である場合、一般にコレスキー分解とコレスキー・ソルバが使用されます。アルゴリズムでは行列 **A** の逆行列を求め、したがって $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ のベクトル **x** の値を求めます。分解に使用するアルゴリズムにもよりますが、コレスキー分解は **QR** 分解に比べて 2 倍以上効率的です。これらの分解アルゴリズムは再帰的であり、除算を含むため、行列サイズが大きくなるにつれて広い数値ダイナミック・レンジが必要になります。例えば、行列サイズが 4 x 4 程度の MIMO チャネル推定の場合、ほとんどの実装では浮動小数点演算を使用して実行されます。軍用アプリケーションに見られるもののように高いスループットを必要とする大規模システムの場合、必要な浮動小数点演算の比率が高いことから、一般に組込みシステムでは対応できませんでした。設計者は、アルゴリズム全体をあきらめて準最適なソリューションを採用するか、複数の高性能浮動小数点プロセッサの使用に頼ることが多く、結果としてコストとデザイン作業の増加につながります。

アーキテクチャの概要

コレスキー・ソルバ

ここでのデザイン例では、コレスキー・ソルバをパイプライン化し、並列動作する 2 つの

サブシステムとして FPGA に実装します。最初のサブシステムは、コレスキー分解と前進代入 (囲み欄「*コレスキー・ソルバ*」のステップ 1 および 2) を実行します。2 番目のサブシステムは後退代入 (囲み欄のステップ 3) を実行します。入力行列はエルミート行列であり、この分解によって共役転置の関係にある三角行列が生成されます。そこで、入力行列 A の下三角行列の半分のみをロードし、下三角行列 L が生成されたとして、上書きすることにより、メモリ使用率が最適化されます。どちらのサブシステムもパイプライン化されており、入力ステージと処理ステージを利用して、あるメモリ領域で処理を実行し、残りのメモリ領域を新規データのロードに使用することができます。図 2 に示すように、分解および前進代入パイプライン・ステージの出力は後退代入の入力ステージに入ります。

コレスキー・ソルバ

Ax = **b** のベクトル **x** の値を求めるための再帰的コレスキー・アルゴリズムには、以下の 3 つのステップがあります。

ステップ 1: 分解 (**A** = **LL*** として下三角行列 **L** の解を求める)

$$l_{11} = \sqrt{a_{11}} \quad (1)$$

for i = 2 to n,

$$l_{i1} = {}^{a_{i1}}/_{l_{11}} \quad (2)$$

for j = 2 to (i-1),

$$l_{ij} = (a_{ij} - \sum_{k=1}^{j-1} l_{ik} \times l_{jk}^{*})/l_{jj} \quad (3)$$

end

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} (l_{ik} \times l_{ik}^{*})} \text{end} \quad (4)$$

end

上記の方程式における依存関係に注意してください。方程式 (4) の対角線要素は、同じ行の左側の要素にのみ依存します。非対角線要素は、同じ行の左側の要素、および上の対応する対角線要素の左側の要素に依存します。

ステップ 2: 前進代入 (方程式 **Ly** = **b** の **y** の値を求める)

$$y_1 = {}^{b_1}/_{l_{11}} \quad (5)$$

for i = 2 to n,

$$y_i = (b_i - \sum_{k=1}^{i-1} y_k \times l_{ik})/l_{ii} \quad (6)$$

end

ステップ 3: 後退代入 (方程式 **L*** **x** = **y** の **x** の値を求める)

$$x_n = y_n/l_{nn}^{*} \quad (7)$$

for i = n-1 to 1,

$$x_i = (y_i - \sum_{k=i+1}^{n} x_k \times l_{ik}^{*})/l_{ii}^{*} \quad (8)$$

end

ここで、

n = 行列 **A** の次元
l_{ij} = 行列 **L** の行 *I* 列 *j* の要素
a_{ij} = 行列 **A** の行 *I* 列 j の要素
y_i = ベクトル **y** の行 *I* の要素
b_i = ベクトル **b** の行 *I* の要素
x_i = ベクトル **x** の行 *I* の要素

ステップ 1 の出力はコレスキー分解、ステップ 3 の出力は一次方程式 **Ax** = **b** の解 **x** です。このアルゴリズムでは、**x** =$\mathbf{A}^{-1}$ **b** を解くために行列 **A** の逆行列を間接的に求めます。

分解の中核は、方程式 (3) および (4) を計算する複素ベクトル・ドット積エンジン (ベクトル乗算器とも呼ばれます) です。Stratix V デバイスでは、最大 90 個、Arria V デバイスでは最大 45 個の複素数データ要素からなるベクトル・サイズ (VS) を実装することができます。また、ベクトル・サイズは、クロック・サイクルごとにドット積エンジンに新規データ・セットを供給するのに必要な並列メモリ読み出し数にも対応し、したがって内部で使用されるデュアル・ポート・メモリの幅および分割を決定します。実装上の理由から、所与のサイズの行列を、*ceil(N/VS)* メモリ・バンクに分割します (*ceil()* はシーリング関数、*N* は行列のサイズ)。

使用するコレスキー・ソルバ・デザインは、マルチチャネル実装です。最大チャネル数はコンパイル時のパラメータであり、デバイス内の利用可能なメモリによってのみ制限されます。メモリ分割は、同じ構造の複数のコピーが使用されること以外はシングル・チャネル実装と同じです。

分解は、左上隅から垂直方向にジグザグ状に右下隅まで、列単位で 1 要素ずつ実行しま

す (図 3 (a) 参照)。まず各列の対角線要素を計算し、続いて同じ列内の下にある非対角線要素をすべて計算した後、右隣列の一番上の対角線要素に移動します。イベントおよび繰り返しのスケジュールは、4 レベルの入れ子 *for* ループによって制御します。最外ループは列単位の処理、2 番目のループはバンク単位の処理を実行し、3 番目のループは行を処理し、最内ループは複数のチャネルを処理します。チャネル処理を最内ループに配置することにより、浮動小数点アキュムレータは事実上、時分割アキュムレータとなり、レイテンシをより効果的に隠すことができます。DSP Builder アドバンスト・ブロックセットの *NestedLoops* ブロックは 1 個の処理ブロック内に最大 3 レベルの入れ子ループを統合しており、別個の *for-loop* ブロック 3 個によって実装された同様の機能に比べて、高速かつリソース効率に優れています。このブロックによって複雑なループ制御信号が抽象化されるため、ループ構造全体がわかりやすくなり、設計/デバッグ時間の短縮につながります。

ドット積エンジンは行列の行に対して動作し、方程式 (3)、(4)、および (6) の加算項でベクトル・サイズまでの乗算を 1 つのサイクルで同時に計算します。ドット積エンジンの入力には、複数の入力行列の行に対して繰り返すために循環メモリ構造が使用されます。ベクトル・ドット積がベクトル・サイズより短い場合、使用しない項をマスクし、加算に含まないようにします。ドット積がベクトル・サイズにより長い場合、部分積和を計算し、バンク境界に保存します。そして、その行に含まれる所与の要素のバンク出力がすべて得られた時点で、総和を実行します (図 3 (b) 参照)。バンク出力の総和は、DSP Builder アドバンスト・ブロックセットの浮動小数点加算器ブロックを使用して、単一のアキュムレータ・ループで実行します。このフィードバック・ループには、13 サイクルのレイテンシがあります。*for Banks* ループと *for Rows* ループの順序を従来のソフトウェア実装の場合とは逆に入れ替え、マルチチャネル処理を追加することにより、浮動小数点アキュムレータのレイテンシが隠され、ハードウェア使用率が向上します。このタイプのレイテンシに対処するためのループ遅延は、DSP Builder アドバンスト・ブロックセットによって自動的に計算されます。*Loop Delay* ブロックの *Minimum delay* チェックボックスをオンにすることにより、最小遅延を計算し、追加するように設定することができます。また、同一の遅延が発生するパスに同じグループ番号を割り当て、グループ内のすべてのパスに同じ遅延値を割り当てることも可能です。本書で評価した例では使用していませんが、DSP Builder アドバンスト・ブロックセットは特定用途向け浮動小数点アキュムレータを備えています。これはユーザーによるカスタマイズが可能で、最大入力サイズや必要なアキュムレータ精度などのパラメータのコンフィギュレーションによって速度およびリソース要件を最適化することができます。このブロックは、高クロック・レートにおいて単一ストリームの浮動小数点数の累算を 1 サイクル/サンプルで実行することが可能です。

2 番目のサブシステムは後退代入を実行します。このサブシステムは、固有の入力および出力メモリ・ブロックを備えており、コレスキー分解/前進代入サブシステムと同様に入力ステージと処理ステージにパイプライン化されています。後退代入の複雑度は分解の N^3 に対して N^2 程度であるため、ドット積にはベクトル処理を使用せず、単一の複素数乗算器を使用します。これで、コレスキー分解/前進代入サブシステムに十分に対応可能です。

QR ソルバ

Ax = **b** の **x** の値をグラム・シュミット法による QR 分解で求めるには、以下の 3 つのステップが必要です。

ステップ 1:サイズが *m* x *n* の行列 **A** の分解 (**A** = **QR** として、サイズが *m* x *n* の行列 **Q** とサイズが *n* x *n* の行列 **R** を求める)

$$\mathbf{u}_1 = \mathbf{a}_1 \quad (1)$$

for k = 1 to n,

$$r\_sq_{kk} = \sum_{j=1}^{m} u_{jk}^{*} \times u_{jk} \quad (2)$$

$$r_{kk} = \sqrt{r_{sq_{kk}}} \quad (3)$$

for i = (k+1) to n, and k < n,

$$r_{ki} = \frac{1}{r_{kk}} \sum_{j=1}^{m} u_{jk}^{*} \times a_{ji} \quad (4)$$

end

t = k + 1, and k < n, (5)

for i = 1 to m, and k < n,

$$u_{it} = a_{it} - \sum_{j=1}^{k} r_{jt} \times u_{ij} / r\_sq_{jj} \quad (6)$$

end

end

正規直交行列 **Q** = [$\mathbf{q}_1$ $\mathbf{q}_2$ …$\mathbf{q}_n$] は明示的に計算されず、直交行列 **U** = [$\mathbf{u}_1$ $\mathbf{u}_2$ … $\mathbf{u}_n$] が計算されて、この再構築された一連の方程式で使用されることに注意してください。ここで、 $\mathbf{q}_i = \mathbf{u}_i / \|\mathbf{u}_i\|$ です。

ステップ 2:**d** の計算 (**d** = $\mathbf{Q}^T\mathbf{b}$)

for k = 1 to n,

$$d_k = \sum_{j=1}^{m} u_{jk}^{*} \times b_j / r_{kk} \quad (7)$$

end

ステップ 3:後退代入 (方程式 **Rx = d** の **x** の値を求める)

for i = n-1 to 1,

$$x_i = (d_i - \sum_{k=i+1}^{n} r_{ik}^{*} \times x_k) / r_{ii}^{*} \quad (8)$$

end

ここで、

m = 行列 A の行次元
n = 行列 A の列次元
$\mathbf{u}_i$ = 行列 **U** の列
iu_{ij} = 行列 **U** の行 *i* 列 *j* の要素
r_{ij} = 行列 **R** の行 *i* 列 *j* の要素
$\mathbf{a}_i$ = 行列 **A** の列
ia_{ij} = 行列 **A** の行 *i* 列 *j* の要素
d_i = ベクトル **d** の行 *i* の要素
b_i = ベクトル **b** の行 *i* の要素
x_i = ベクトル **x** の行 i の要素

QR ソルバは、行列 **A** の不確定な逆行列を求めずに、一次方程式 **Ax** = **b** の解 **x** を求めます。

QR ソルバ QR 分解と QR ソルバは、パイプライン化され、並列動作する 2 つのサブシステムとして実装されます (図 4 参照)。最初のサブシステムは、囲み欄「*QR ソルバ*」のステップ 1 および 2 を実行し、2 番目のサブシステムはステップ 3 の後退代入を実行します。後退代入サブシステムは、コレスキー・ソルバに使用されているものと同じです。コレスキー・ソルバでは方程式 (1) ～ (6) を 4 レベルの単一の入れ子ループとして実装しているのに対し、QR ソルバは単純な有限ステート・マシン (FSM) を使用して 4 つの主要演算を繰り返します。つまり、方程式 (2) でベクトルの振幅二乗、方程式 (4) で 2 つのベクトルのドット積、方程式 (6) でベクトルからドット積の差、方程式 (7) でドット積を計算することにより、ベクトル **d** の値を求めます。DSP Builder アドバンスト・ブロックセットの *NestedLoop* ブロックは、FSM の全段階でデータパスのすべての制御およびイベント信号を生成するために使用されます。

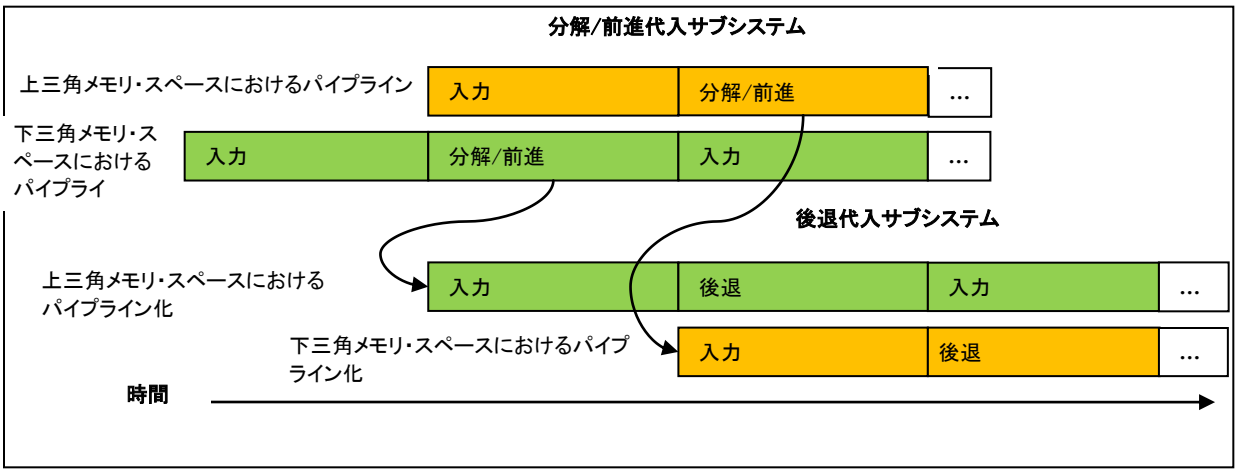

図 2 コレスキー・プロセスのパイプライン化とメモリの再利用

上記の 4 つの演算に共通する処理ブロックはドット積エンジンです。性能向上のために、ドット積の計算にはベクトル処理が使用されています。ベクトル・サイズは、コレスキー・ソルバ・デザインと同様にコンパイル時のパラメータです。FSM の 4 つのステートすべてにおいてこのエンジンを再利用するために、入力部分にデータ・マルチプレクサを使用し、FSM イベント・コントローラによって制御します。このマルチプレクサにより、FSM のステートごとにドット積エンジンに合った入力が選択されます。ドット積エンジンおよび浮動小数点アキュムレータの詳細は、コレスキー・ソルバ・デザインのものと同様ですので、ここでは示しません。

QR 分解に必要なメモリは、最初に行列 **A** および入力ベクトル **b** を保持するメイン・コア・メモリ・ブロックを再利用することによって最適化されます。処理は、コア・メモリ内で左から右に向かって列単位で実行されます。列は、使用されて不要になった時点で、新しい行列の対応する列によって上書きされます。このメモリ・ブロックの元の内容は、元の行列 **A** が分解されるときまでに新しい行列によって置き換えられるため、行列の処理を停止することなく連続して実行する能力が保たれます。

FSM の最初の 2 つのステートでは、行列 **R** の要素は各行の対角線要素から行単位で 1 要素ずつ生成されます。FSM の減算ステートでは、コア・メモリ内の行列 **A** の列は再帰的に更新され、部分的に計算されたベクトル u_i によって置き換えられます。例えば、列 *k* では、列 *k+1* ～ *n* はすべて、メモリ内でスケーリング後の u_k を *k+1* ～ *n* の各列から減

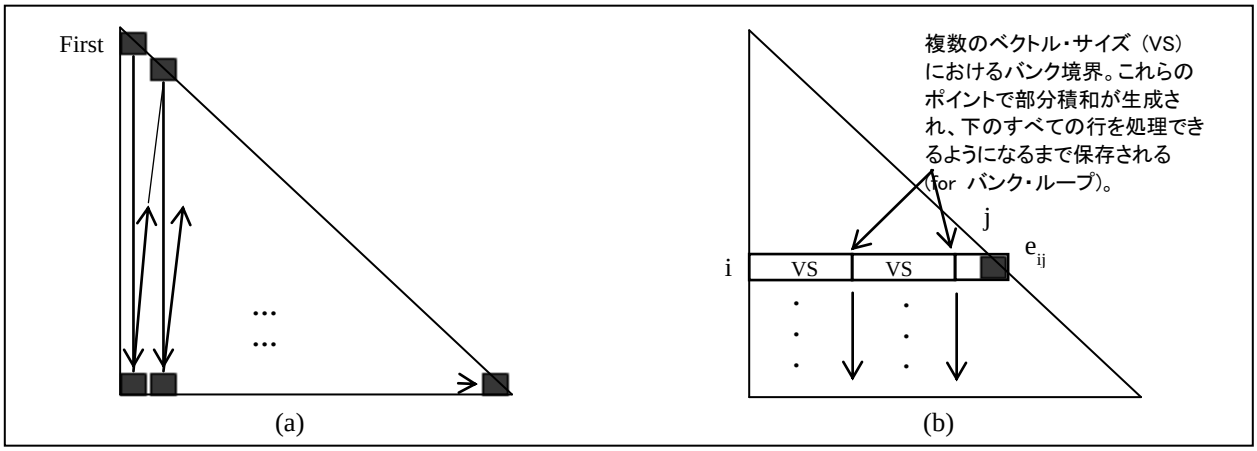

図 3 (a) 処理シーケンス、(b) 対角線要素 e_{ij} の計算には、j =VS と j = 2*VS における 2 つの部分積和のほか、最後に残るドット積部分の加算を含む

Last

算することによって更新されます。この処理　　(方程式 (3) および (4) 参照)。矩形メモリ構

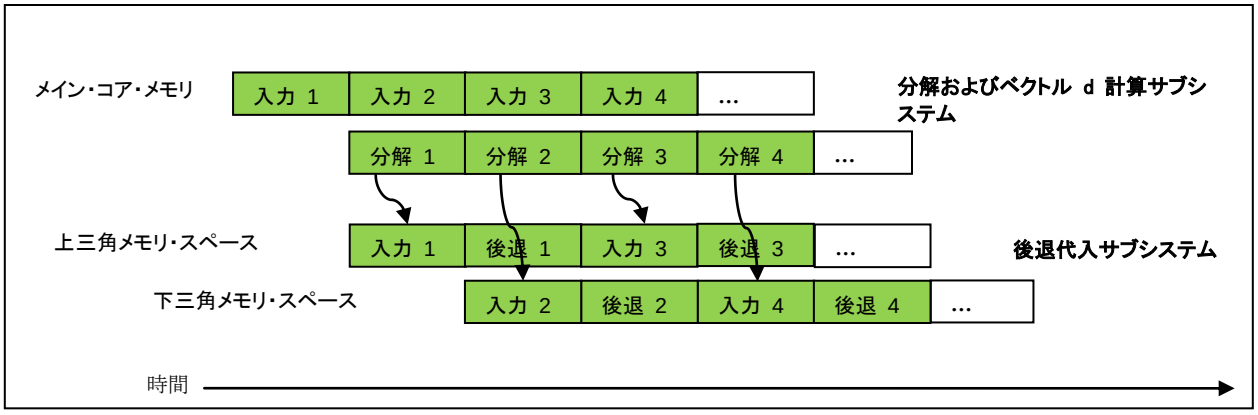

図 4 QR プロセスのパイプライン化とメモリの再利用

は、ベクトル u_{k+1} ～ u_n を再帰的に計算するものであり、方程式 (6) をハードウェア実装した場合、効率に優れています。FSM の 4 番目のステートでは、列単位で処理することにより、新たに計算された各 u_k と入力ベクトル **b** を乗算して列ベクトル **d** の単一要素 d_k を生成します。行列 **Q** は明示的に生成されませんが、直交列 u_k が生成され、後続の FSM の段階で使用され、FSM の次のサイクルでの新しい行列の対応する列 a_k によって上書きされます。図 5 は、QR 分解サブシステムのメモリ構成および処理順序を示しています。

最初のサブシステムの出力は、行列 **R** と列ベクトル **d** です。行列 **R** は、上三角行列であり、行単位で左から右に向に生成されます

造は、ピンポン型で使用されます。下三角セクションは、上三角セクションが後退代入サブシステムによって処理されている間は分解サブシステムによって埋められており、逆に上三角セクションが分解サブシステムによって埋められている間は後退代入サブシステムによって処理されます。列ベクトル **d** は、行単位で上から下に生成され、行列 **R** に付加されます。後退代入の出力が一次方程式 **Ax** = **b** の 解ベクトル **x** です。

コレスキー・ソルバと同様に、QR ソルバもマルチチャネル形式で実装することで使用率の向上、レイテンシの低減、およびデザインのスループット向上を図ることが可能です。スループット向上は、主に浮動小数点アキュムレータのレイテンシの効果的な低減によってもたらされます。

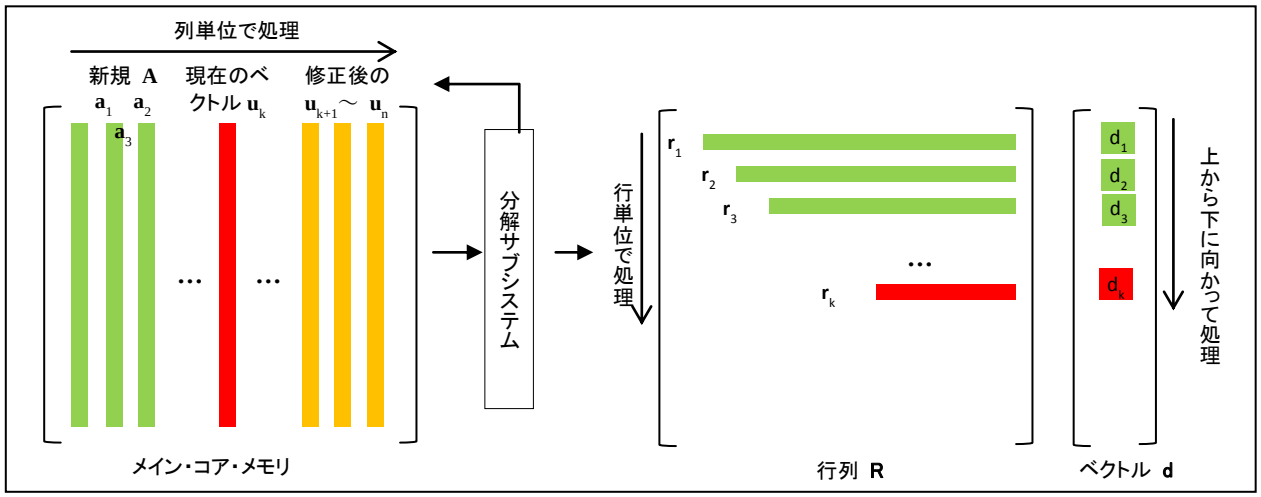

図 5 QR 分解サブシステムのメモリ構成

3. デザイン・フローとツール・チェイン

デザイン評価方法論

この評価のために、DSP Builder アドバンスト・ブロックセットを使用して作成されたコレスキー・ソルバおよび QR ソルバの実装をアルテラから提供していただきました。また、評価に必要なアルテラおよび MATLAB のツールがインストールされた PC もアルテラから提供していただきました。その上で、BDTI のエンジニアはアルテラのデザインの検討、シミュレート、および合成をいずれも Simulink 環境で行いました。さらに、合成したデザインを Stratix V FPGAと Arria V FPGAという 2 種類のデバイスで実行しました。その過程で、アルテラのデザイン・フローと 2 つのサンプル・デザインの性能を評価しました。この評価で使用したボードは、サンプル・デザイン用のスティミュラスを生成する機能を備えていないため、どちらのデザインもスティミュラス生成ブロックが搭載されています。これは、DSP Builder アドバンスドを使用して実装されており、評価対象アプリケーション・デザインと共にコンパイルされます。スティミュラス・ブロックが評価対象デザインの性能および FPGA リソース使用量に与える影響を最小限に抑えるために、行列 **A** はスティミュラス・ブロックによって、ごく少数のランダム・データからオンザフライで生成されます。この少数のデータは、スティミュラス・ブロック・メモリにロードされ、デザインと共にコンパイルされる MATLAB m ファイル・スクリプトによって生成されます。この MATLAB スクリプトにより、評価対象デザインのスティミュラス・ブロックと同じアルゴリズムを使用して解ベクトル **x** の基準データを倍精度浮動小数点形式で生成し、Simulink モデルおよびデバイス上で実行されるデザインの誤差性能の評価基準とします。このアルゴリズムにより、コレスキー・ソルバ・デザインでは行列 **A** のエルミート正定値を保証し、QR ソルバ・デザインでは適切な行列 **A** の生成を保証します。

コレスキー・ソルバ・デザインでは、ベクトル、行列、およびチャネル・サイズの 4 つの構成を実装しました。Stratix V FPGA にはそのうちの 3 つ、Arria V デバイスには 2 つを実装し、1 つはデバイス間の共通構成としました。QR ソルバ・デザインでは、4 つの構成を Stratix V FPGAに実装し、そのうちの 2 つを Arria V デバイスにも実装しました。そして、すべての構成について、FPGA リソース使用量、達成可能なクロック・レート、スループット、および機能的な正確性を評価しました。クロック・レート、デバイス選択、スピード・グレードなどの FPGAデザイン 制約は、Simulink 環境で指定します。

スループットおよび性能の評価は、Simulink モデル・レベルとハードウェア・プラットフォーム・レベルで行いました。前進処理サイクル、および前進処理サイクルと後退処理サイクルの合計を表示するためのインスツルメンテーションを各デザインの Simulink モデルに加え、構成ごとに Quartus II 開発ソフトウェアを実行して、達成される F_{max} を得ます。その上で、各構成をハードウェア・プラットフォームにダウンロードし、動作周波数を F_{max} に設定し、処理を開始します。各構成について、各ソルバの解ベクトル **x** をキャプチャし、対応する MATLAB 倍精度浮動小数点基準と比較します。

シミュレーション後のスクリプトにより、Simulink による IEEE 754 単精度浮動小数点出力と、生成した MATLAB 倍精度浮動小数点基準との差を計算します。同様に、キャプチャされたハードウェア・シミュレーション出力と、生成した MATLAB 倍精度浮動小数点基準との差も計算します。

ツール・チェインの評価

Simulinkは、MATLAB フレームワークを基礎に構築されているため、MATLAB フレームワークが必要です。Simulink 環境では、評価対象のデザインはアルテラ DSP Builder アドバンスト・ブロックセットのブロックを使用します。これは DSP Builder スタンダード・ブロックセットとは別のブロックセットです。DSP Builder アドバンスト・ブロックセットは、ブロック・ベースの DSP アルゴリズムおよびデータパスの実装を対象にしており、より一般的かつ基本的なファンクション・ブロックで構成される DSP Builder スタンダード・ブロックセットに比べ、抽象化レベルが高くなっています。このライブラリには、標準のものよりも複雑な高速フー

リエ変換 (FFT) および FIR フィルタ・ビルディング・ブロックのほか、50 種類以上の一般的な三角関数、算術関数、およびブール関数が含まれています。Quartus II 開発ソフトウェア v12.0 の主な追加機能は、低レイテンシの平方根、入れ子 *for* ループ、およびカスタマイズ可能な浮動小数点アキュムレータ・ブロックです。スタンダード・ブロックセットのエレメントと DSP Builder アドバンスト・ブロックセットのエレメントを同じ階層レベルのデータパス構造内で併用することはできません。浮動小数点コンパイラをサポートしているのは、DSP Builder アドバンスト・ブロックセットのブロックのみであり、スタンダード・ブロックセットのブロックは浮動小数点処理に最適化されていません。また、スタンダード・ブロックセットは、ハンド・コーディングされた HDL のインポートに対応していますが、DSP Builder アドバンスト・ブロックセットの場合、ツールが HDL レベルでの最適化を実行できないため対応していません。一般に、ブロック・ベースのデザイン・エントリ手法は DSP アルゴリズムに適していますが、*case* や *switch* などの構造がブロックセットに含まれていないため、制御を多用し、ステート・マシンを含むデザインの場合は、テキスト・ベースの手法の方が直観的に行えます。

DSP Builder アドバンスト・ブロックセットでシミュレーションを開始すると、Simulinkモデルがコンパイルされ、アルテラ Quartus II 開発ソフトウェア環境用の HDL コードと制約が生成され、ModelSim 環境用のテスト・ベンチとスクリプト・ファイルが作成された後、Simulinkモデル・シミュレーションが実行されます。各種構成のシミュレーションの実行に必要な時間は、3 GHz Intel Xeon W3550 PC の場合、行列サイズによって 3 分から 28 分程度です。Simulink によるシミュレーションでは、Quartus II 開発ソフトウェアによるコンパイルの実行なしに、詳細なリソース使用量見積もりが生成されるため、必要なデバイス・サイズを素早く決定できます。ハードウェア開発キットを使用する場合、ユーザーがボード・レイアウトに基づいてピンアウト・アサインメントを指定する必要があります。その際、デザインの Quartus II 開発ソフトウェアのプロジェクト・フォルダ内にある *.qsf* ファイルに含まれるピンアウト制約を再利用するか、Quartus II 開発ソフトウェアのピン・プランナ機能を使用してピンアウト・アサインメントを作成・管理することが可能です。

アルゴリズム探索および対応する HDL 生成の容易性を評価するために、モデルで実験を行いました。スティミュラス・ブロックおよびシミュレーション実行でベクトル・ドット積サイズ、行列サイズ、データ型などの入力パラメータを変更したところ、いずれの場合も数分以内に正しい RTL コードが生成され、シミュレーション出力は MATLAB 評価基準と一致しました。

構成の合成はすべて、Simulink 環境から直接起動できる Quartus II 開発ソフトウェアで行いました。設計者は、デフォルトまたはユーザーが選択した最適化パラメータで*プッシュ・ボタン*・モードで Quartus II 開発ソフトウェアを使用するか、デザイン・スペース・エクスプローラ (DSE) ツールを使用することができます。デザイン・スペース・エクスプローラは、Quartus II 開発ソフトウェアの一部として利用可能で、パスごとに異なるシードを使用して複数のルータ・パスを自動的に実行します。そして、最も高いクロック・レートが得られる配線を保存します。これはユーザーの介入が不要な自動プロセスですが、プッシュボタン・モードに比べてはるかに長い時間がかかります。プッシュボタン・モードの場合、デザイン・サイズによって 1 ～ 6.5 時間を要しました。

DSP Builder アドバンスト・ブロックセット・デザイン・フローに施されたより高い抽象化レベルにより、 より高速なアルゴリズム空間探索およびシミュレーション・サイクルが可能になることで、最終的な最適化されたデザインに到達するのに要する時間全体が短縮されます。しかし、この優位性は、例えば“データ型の伝播がSimulink固有の機能である”という意味では 、Simulink固有のものではありません。手書きによるRTLに対するブロック・ベース・デザイン手法の設計空間探索の優位性を生かすためには、設計者はSimulinkモデルを開発する際に新たな手順を踏む必要があります。特に、モデルはパラメータ駆動型アルゴリズム空間探索を行えるよう構成されなければなりません。当文書で検証しているデザイン例では、モデルは異なる行列サイズ、ベクトル・サイズ、および

パラレル・チャネルの数（コレスキー・ソルバの場合）で実験が行えるよう実装されています。一旦、この水準の柔軟性を備えたモデルが開発されると、性能およびさまざまなデザイン構成に対するリソース使用見積はこれらのパラメータを変動させることで実行することができます。当文書のセクション2に記述されている浮動小数点乗加算器ブロックで例示された通り、良好なスループット速度およびリソース使用を達成するにはハードウェア・デザインの理解も求められます。

DSP Builder アドバンスト・ブロックセット・デザイン・フローに関するトレーニングには、アルテラによる 4 時間のクラスのほか、約 10 時間のオンライン・チュートリアルおよびデモが必要です。また、BDTI が実際にツールおよび両モデルの探索に費やした時間は 90 時間程度でした。ツール・チェインで素早く作業するのに必要な時間と労力は、設計者のスキルと経歴によって左右されます。Simulink のブロック・ベースのデザインと FPGA ハードウェア・デザイン経験の両方に熟練したエンジニアであれば、DSP Builder アドバンスト・ブロックセット・アプローチを効率的かつ容易に使用できるものと思われます。MATLAB と Simulink の知識がほとんどない FPGA 設計者にとって、高い抽象化レベルでの設計は新しい考え方であるため、当初は困難が伴い、習熟するまでにかなりの時間を要するかもしれません。しかし、いったん習熟すれば、HDL アプローチよりもはるかに速いデザイン・サイクルを実現できるはずです。なぜなら、パイプライン化などのハードウェア・デザインの詳細について心配する必要がなくなり、アルゴリズムの実装に集中できるようになるからです。機能シミュレーションおよび検証の大部分を Simulink 環境で行えるため、デザインおよび検証時間は大幅に短縮されます。これは、Simulink によるコンパイルからの RTL 出力を ModelSim ソフトウェアで実行すれば、完全に機能するシミュレーションを行えるからです。

システム・レベルのデザインの経験はあるものの、ハードウェア・デザインのスキルがほとんどないエンジニアの場合、習熟には長い時間を要する可能性があります。ツール・チェインでは、ハードウェア・コンパイル、合成、配線、および自動スクリプト生成が Simulink 環境内に統合され、データ・パイプライン化や信号ベクトル化などの多くの複雑なデザイン・コンセプトが抽象化されていますが、実装を行うにはやはりハードウェア・デザインの知識がある程度必要です。

4. 性能結果

このセクションでは、アルテラのコレスキー・ソルバおよび QR ソルバ浮動小数点実装例

デザイン	デバイス	構成 (チャネル・サイズ / 行列サイズ / ベクトル・サイズ)	ALUT (K) (使用量 / 全体に占める割合)	レジスタ数 (K) (使用量 / 全体に占める割合)	DSP ブロック数 (使用される可変精度 27 x 27 乗算器数 / 全体に占める割合)	M20K (Stratix) / M10K (Arria) ブロック数 (使用量 / 全体に占める割合)	F_{max}, (MHz) P：プッシュボタン使用 D：DSE 使用
コレスキー	Stratix V	1 / 360×360 / 90	198 / 57%	339 / 49%	391 / 25%	1411 / 70%	189 (P)
		20 / 60×60 / 60	135 / 39%	235 / 34%	268 / 17%	955 / 48%	234 (P)
		64 / 30×30 / 30	74 / 22%	124 / 18%	146 / 9%	793 / 39%	288 (P)
	Arria V	6 / 90×90 / 45	104 / 27%	179 / 24%	214 / 19%	1094 / 45%	176 (P) 198 (D)
		64 / 30×30 / 30	73 / 19%	121 / 16%	154 / 13%	1694 / 70%	185 (P)
QR	Stratix V	1 / 400×400 / 100	184 / 53%	377 / 55%	428 / 27%	1566 / 78%	203 (P)
		1 / 200×100 / 100	180 / 52%	375 / 54%	428 / 27%	504 / 25%	207 (P)
		1 / 200×100 / 50	96 / 28%	201 / 29%	228 / 14%	281 / 14%	260 (P)
		1 / 100×50 / 50	95 / 28%	198 / 29%	227 / 14%	230 / 12%	259 (P)
	Arria V	1 / 200×100 / 50	97 / 25%	202 / 27%	238 / 21%	372 / 15%	171 (P)
		1 / 100×50 / 50	95 / 25%	200 / 26%	237 / 21%	245 / 10%	170 (P)

表 1 リソース使用量とクロック速度

に関する BDTI の独立評価の結果を示します。すべてのデザインは、アルテラの DSP Builder アドバンスト・ブロックセット v12.0 と MathWorks がリリースした R2011b および Simulink 7.8 を使用し、Quartus II 開発ソフトウェア v12.0 SP1 によって構築されています。RTL シミュレーションは、ModelSim 10.1 を使用して実行しました。デザインは、ハイエンドの Stratix V の中規模デバイス 5SGSMD5K2F40C2 とミッドレンジの Arria V 5AGTFD7K3F40I3N デバイスという 2 種類のアルテラ 28nm FPGA デバイス向けに構築しました。この分析で使用されている Stratix V FPGAは、34万5,200個のALUT、1,590個の27×27ビット可変精度乗算器、および2,014個のM20Kメモリ・ブロックを備えています。Arria V FPGAは、38万4,000個のALUT、1,156個の27×27ビット可変精度乗算器、および2,414個のM10Kメモリ・ブロックを備えています。RTL 評価に使用したハードウェア・プラットフォームは、DSP 開発キット Stratix V エディションと Arria V FPGA 開発キットです。1 つの構成については、Simulink 環境からのツールの使いやすさを評価するために ModelSim ソフトウェアを使用しました。

2 種類のデバイス向けの両デザインで合計 11 のケースをシミュレートし、構築しました。リソース使用量、性能、および精度の結果をケースごとに記録しました。表 1 に、コレスキー・ソルバおよび QR ソルバの構成ごとに達成されたリソース使用量とクロック速度を示します。コレスキー・ソルバ・デザインは、最大行列サイズ・パラメータを備えており、最大デザイン・サイズより小さい行列サイズを実行時に使用することができます。表 1 に示したリソース使用量の結果に関して、テスト対象の行列サイズによって消費される実際のリソースを得るために、評価対象の行列サイズに等しい最大行列サイズ・パラメータで各構成を合成しました。なお、スティミュラス・ブロックによって使用されるリソースは合計には含まれていません。また、本書で評価した構成のいずれも、FPGA のキャパシティを使い切っていない点にも留意してください。さらに、Quartus II 開発ソフトウェアによって妥当な合成および配置配線時間で最大の F_{max} を達成するために、同じプリセット最適化パラメータを使用して各デザインの速度向上を図りました。6/90 x 90/45 構成のコレスキー・ソルバ・デザインで Quartus II 開発ソフトウェア デザイン・スペース・エクスプローラ (DSE) を実行し、プッシュボタン・モードに対する速度向上および必要な合成時間を評価しました。この場合、12.5% の速度向上が得られましたが、Quartus II 開発ソフトウェア によるデザインの合成時間は 2 時間から 7.5 時間に増加しました。

FPGA リソース使用量は、評価対象デザインに対する予想と一致しています。メモリ使用は主に行列の格納であり、行列サイズとチャネル数 (マルチチャネル・デザインの場合) に比例します。DSP ブロック使用率は、ベクトル・サイズとともに直線的に増加します。ベクトル乗算器は、27 x 27 ビット複素数値の浮動小数点乗算ごとに 4 個の可変精度 DSP ブロックが必要です。ベクトル・ドット積エンジンには、ベクトル・サイズを 60 個の複素浮動小数点値とすると 240 個の DSP ブロックが必要です。

デザイン	デバイス	構成 (チャネル・サイズ / 行列サイズ / ベクトル・サイズ)	Simulink によって報告されたスループット (千行列 / 秒)	全体のレイテンシ (µsec) @F_{max} (MHz)	GFLOPS (実数データ型)
コレスキー	Stratix V	1 / 360×360 / 90	1.43	1112 @ 189	91
		20 / 60×60 / 60	118.35	330 @ 234	39
		64 / 30×30 / 30	544.28	222 @ 288	26
	Arria V	6 / 90×90 / 45	31.31 35.22	347 @ 176 308 @ 198	34 38
		64 / 30×30 / 30	349.62	344 @ 185	16
QR	Stratix V	1 / 400×400 / 100	0.315	3970 @ 203	162
		1 / 200×100 / 100	8.76	167.0 @ 207	141
		1 / 200×100 / 50	6.17	204.5 @ 260	99
		1 / 100×50 / 50	32.82	43.3 @ 259	66
	Arria V	1 / 200×100 / 50	4.05	311 @ 171	65
		1 / 100×50 / 50	21.54	66 @ 170	44

表 2 性能結果

表 2 に、すべての構成におけるコレスキー・ソルバおよび QR ソルバの性能を示します。いずれも、表 1 に示した F_{max} で得られる性能です。スループットは、ソルバの前進代入サブシステムの実行によって消費されたサイクルで F_{max} を除算することによって計算しました。後退代入サブシステムは、前進代入サブシステムよりも低いレイテンシで並列実行されるため、全体のスループットには影響しません。マルチチャネル・コレスキー・ソルバのスループットについては、この結果に並列処理するチャネル数を乗算しました（表2にてチャネル・サイズを表示）。各ケースの全体のレイテンシは、前進代入および後退代入サブシステムの実行に要した合計サイクルを F_{max} で除算することによって計算しました。行列サイズに対するベクトル・サイズの選択は妥協点をみつけることになり、アプリケーションによって異なります。QR ソルバの 200 x 100 構成の各種ベクトル・サイズにおける結果からわかるように、ベクトル・サイズが行列サイズよりもはるかに小さいデザインでは、レイテンシが大きくなる代わりにリソース効率が高くなります。

マルチチャネル・コレスキー・デザインは、BDTIの以前の文書で分析されたシングルチャネル・デザインを改善します。シングル・チャネルでの実装において、浮動小数点乗加算器で見られるようなレイテンシが、アルゴリズムにおける処理の順番を並べ替えることで部分的に隠されてしまいます。参照情報[1]で報告されている通り、シングル・チャネル実装の効率はほとんど行列およびベクトルのサイズによって左右されます。また、特に行列サイズとベクトル・サイズが小さい場合、マルチチャネル実装が処理効率の点で非常に有利であることは、表2のスループット列を見れば明らかです。マルチチャネル処理によるスループット向上は、本稿第2章で述べているように、実装のレイテンシが完全に隠されることによって得られます。マルチチャネル実装は、所与の行列サイズおよびベクトル・サイズにおいて、シングル・チャネル実装よりも高いピーク・スループットを実現します。

表の最後の列は、各構成の 1 秒当たりの実数データ浮動小数点演算回数を 10^9 単位 (GFLOPS) で示しています。各ソルバに必要な演算回数は、使用する分解アルゴリズムに依存します。報告された数字は、2つのFPGA上の浮動小数点複素データ・フォーマットへのコレスキー・ソルバおよび QR ソルバ・アルゴリズムの実際の実装から得たものです。実数データ浮動小数点演算回数は、コレスキー・ソルバでは二次項 $4n^3/3 + 12n^2$ に近似させ、QR ソルバでは $8mn^2 + 6.5n^2 + mn$ を使用しました。

表 3 に、単精度浮動小数点演算を使用する Simulink シミュレーションおよびハードウェア開発ボードで動作するデザイン実装における

デザイン	デバイス	構成 (報告されたチャネル数 / 行列サイズ / ベクトル・サイズ)	MathWorks Simulink IEEE 754 単精度 浮動小数点誤差 (フロベニウス・ノルム / 最大ノルム誤差)	アルテラ DSP Builder による合成 RTL 単精度 浮動小数点誤差 (フューズド・データパス) (フロベニウス・ノルム / 最大ノルム誤差)
コレスキー	Stratix V	1 / 360×360 / 90	2.11e-6 / 1.02e-4	1.16e-6 / 8.58e-5
		7 / 60×60 / 60	4.24e-7 / 8.59e-6	1.82e-7 / 2.62e-6
		53 / 30×30 / 30	7.48e-8 / 2.08e-6	3.84e-8 / 1.15e-6
	Arria V	3 / 90×90 / 45	4.08e-7 / 9.72e-6	1.99e-7 / 5.52e-6
		63 / 30×30 / 30	8.93e-8 / 2.38e-6	5.91e-8 / 1.24e-6
QR	Stratix V	1 / 400×400 / 100	4.53e-6 / 1.45e-4	5.15e-6 / 1.03e-4
		1 / 200×100 / 100	1.24e-6 / 1.13e-5	9.97e-7 / 8.15e-6
		1 / 200×100 / 50	8.38e-7 / 6.70e-6	8.97e-7 / 4.15e-6
		1 / 100×50 / 50	9.13e-7 / 4.68e-6	6.96e-7 / 4.94e-6
	Arria V	1 / 200×100 / 50	9.27e-7 / 2.33e-5	9.31e-7 / 9.95e-6
		1 / 100×50 / 50	9.13e-7 / 4.68e-6	6.96e-7 / 4.94e-6

表 3 MATLAB 倍精度浮動小数点基準と比較した Simulink モデルおよび合成 RTL の誤差性能

コレスキー・ソルバと QR ソルバの誤差性能を示します。誤差は、Simulink および ハードウェア・プラットフォーム・シミュレーションの各出力を、MATLAB によって生成された解ベクトル **x** の倍精度浮動小数点基準と比較することによって計算したものです。マルチチャネル・コレスキー・ソルバの場合、簡潔にするために、ランダムに選んだ単一チャネルの誤差性能のみを示しています。誤差性能は入力データに左右されるものの、多くの場合、RTL 実装はフューズド・データパス・メソドロジの恩恵を受けており、表 3 の列 (4) および (5) の*ノルム*を比較すれば明らかなように、統計的に標準の IEEE 754 単精度実装以上の精度を達成しています。結果のベクトルにおける全体的な誤差の大きさを測定するために、フロベニウス・*ノルム*を使用しました。これは次式によって計算されます。

$$\|E\|_F = \sqrt{\sum_{i=0}^{N} |e_i|^2}$$

ここで、*N* はベクトルのサイズ、**e** は観測された **x** と MATLAB 評価基準の差分ベクトル、*i* はベクトル e の要素番号です。最大正規化誤差は次式によって計算されます。

$$\max_i \left(\left| (x_{i\_obs} - x_{i\_ref}) / x_{i\_ref} \right| \right)$$

5. 結論

本書では、アルテラの DSP Builder アドバンスト・ブロックセット・デザイン・フローによる FPGA への浮動小数点 DSP アルゴリズムの実装に対する新しいアプローチを評価しました。このデザイン・フローには、Altera DSP Builder アドバンスト・ブロックセット、アルテラの Quartus II 開発ソフトウェア・ツール・チェーン、および ModelSim シミュレータのほか、MathWorks の MATLAB および Simulink が組み込まれています。このアプローチでは、設計者が Simulink 環境においてアルゴリズム動作レベルで作業することが可能です。ツール・チェインにより、アルゴリズムのモデリング／シミュレーション、RTL 生成、合成、配置配線、およびデザイン検証ステージが Simulink 環境内に統合されます。それによってアルゴリズム・レベルと FPGA レベルの両面で迅速な開発とデザイン・スペースの素早い探索が可能になり、ひいては全体的な設計作業の削減につながります。アルゴリズムをハイレベルでモデル化してデバッグした後、デザインを合成し、アルテラ FPGA をターゲットにすることが可能です。

この評価の目的上、アルテラ DSP Builder アドバンスト・ブロックセットを使用して Simulink でモデル化した単精度複素数データ IEEE 754 浮動小数点コレスキー・ソルバおよび QR ソルバをデザイン例として使用しまし

た。今回評価した中で最大のデザインは、行列サイズが 400 x 400、ベクトル・サイズが 100 の複素数値浮動小数点行列に対する QR ソルバでした。このデザインは、203 MHz 動作時で 162 GFLOPS を達成しました。表2で報告されているGFLOPS値は、2つのFPGA上の浮動小数点複素データ・フォーマットへのコレスキー・ソルバおよび QR ソルバ・アルゴリズムの実際の実装から得たものです。他の競合するプラットフォームとの比較のため、これらのプラットフォームには同じアルゴリズムを実装し、性能を評価しなくてはなりません。報告された性能はいずれも、手動の最適化またはフロア・プランニングを一切行わずに、アルテラ DSP Builder アドバンスト・ブロックセット・ツール・フローを使用して達成されたものです。Simulinkによるハイレベルのブロック・ベースのデザインから、ツール・チェインによってデザインのパイプライン化、RTL コード生成、およびデザインの合成が自動的に実行された結果、実用になる速度とリソース使用量が達成されたということです。

アルテラの浮動小数点デザイン・フローでは、単一のプラットフォームの下でツールを合理化することにより、FPGA への複素浮動小数点 DSP アルゴリズムの実装プロセスが単純化されています。また、フューズド・データパス・メソドロジにより、従来の限界を超える高い性能と効率で複素浮動小数点データパスを実装することが可能です。

その一方で、この新しいアプローチでは、DSP Builder アドバンスト・ブロックセットを使いこなすまでにかなりの時間が必要であることも事実です。これは、MATLAB と Simulink に慣れてない設計者に特に言えることです。ブロック・ベースのデザイン・エントリ手法は、従来のハードウェア設計者には当初理解しにくいかもしれません。また、手書きによるRTLに対するブロック・ベース・デザイン手法の設計空間探索の優位性を生かすためには、設計者はSimulinkモデルを開発する際に新たな手順を踏む必要があります。例えば、異なる行列およびベクトルのサイズを使った実験を行えるようにするには、当文書に示された2つのデザイン例で行われたように、パラメータ駆動型デザインを組み込んでSimulinkモデルを構築し、さまざまなデザイン構成を精査できるようにする必要があります。

現在、DSP Builder アドバンスト・ブロックセットを使用したデザインは、ブロックセットによって最適化された性能が得られるように提供されるエレメントに限られています。一方、DSP Builder スタンダード・ブロックセットのエレメントは、浮動小数点コンパイラに最適化されておらず、同じ階層レベルのアドバンスト・ブロックセットと併用することはできません。また、ハンド・コーディングされた HDL ブロックは、スタンダード・ブロックセットにしかインポートできません。さらに、DSP Builder アドバンスト・ブロックセットは、DSP 実装を対象にしており、高負荷な制御やステート・マシンを必要とする設計への使用は制限されることも考えられます。

2012 年末にリリース予定である DSP Builder アドバンスト・ブロックセットの次バージョンには、浮動小数点拡張機能が含まれます。設計者は、2つのIEEE 754標準の単精度および倍精度フォーマットに制約されることはなくなり、16から64ビットに及ぶ合計7つの異なる精度を選べるようになります（指数プラス仮数）。DSP Builder アドバンスト・ブロックセットの新しい*Enhanced Precision Support* ブロックを使用することで、設計者は開発しているアプリケーションに最適なデータ幅を選択することができます。

6. 参考文献

[1] Berkeley Design Technology, Inc., 2011. “An Independent Analysis of Altera’s FPGA Floating-point DSP Design Flow”. Available for download at http://www.altera.co.jp/literature/wp/wp-01166-bdti-altera-floating-point-dsp-j.pdf

[2] S.S. Demirsoy, M. Langhammer, 2009. “Fused datapath floating point implementation of Cholesky decomposition.” FPGA ’09, February, 2009.

