ホワイトペーパー

FPGA
人工知能(AI)

インテル® Stratix® 10 NX FPGAを用いたWaveNetの実装によるリアルタイムな音声合成の実現

著者

Jonny Shipton
Myrtle.ai
ソフトウェア・リード

Jon Fowler
Myrtle.ai
FPGAリード

Chris Chalmers
Myrtle.ai
上級開発者

Sam Davis
Myrtle.ai
マシンラーニング責任者

Sam Gooch
Myrtle.ai
マシンラーニング・エンジニア

Giuseppe Coccia
Myrtle.ai
マシンラーニング・エンジニア

目次

概要

リアルタイムの音声合成モデルは、音声アシスタントやスマートスピーカーなどの対話型音声サービスに広く導入されています。しかし、こういったモデルには厳しいレイテンシー要件があることから、導入が困難になるケースも少なくありません。このホワイトペーパーでは、最先端のボコーダーであるWaveNetの実装について説明します。WaveNetは、人間に近いレベルの音質で、256個の16kHzオーディオストリームをリアルタイムで生成することができ、これは最適化されたGPUソリューションと比べて8倍のスループットに相当します。この実装は、インテル® Stratix® 10 NX FPGAによって実現されており、ニューラル・ネットワークを搭載した実際のアプリケーションで高スループットかつ低レイテンシーの推論を実行するFPGAの能力を実証しています。

I. はじめに

音声アシスタントの普及は、主にテキスト読み上げ(TTS)分野の進歩に支えられてきました。最先端の音声合成システムは、オーディオを生成するために順次実行される2つのニューラル・ネットワークで構成されています。1つ目のモデルがテキストを入力として受け取ってスペクトログラムなどの音響的特徴を生成し、ボコーダーと呼ばれる2つ目のモデルがこの中間的な特徴を受け取って音声を生成します。この1つ目のモデルとしてよく用いられるのがTacotron 2です。このホワイトペーパーでは、音声合成システムの2つ目のモデルを中心に説明します。WaveNet[1]は、人間に近いレベルの自然な音声[2]を生成できる最先端のボコーダーです。モデルの品質を決めるカギとなるのは自己回帰ループですが、これはリアルタイム・アプリケーションのネットワークへの実装を非常に難しくしている特性でもあります。

そのためTTSの研究では、Parallel-WaveNet[3]、WaveRNN[4]、ClariNet[5]、WaveGlow[6]など、既存のハードウェアで効率的な推論を実行するためのより適した選択肢となるボコーダー・アーキテクチャーを見つけることに重点が置かれてきました。オーディオ品質の評価は主観的であるため、最高品質のボコーダーといってもある程度あいまいさは残りますが、WaveNet本来のアーキテクチャーだけで、最新のアプローチを上回るとはいかないまでも、同等以上の品質でオーディオを生成できると参考文献のすべての著者が認めています。[3],[6] ~ [9]

ほかのFPGAソリューションも含め、代わりのアーキテクチャーを使用するのではなく、WaveNetモデルを直接アクセラレーションする取り組みは続けられていますが[10]、この手法でリアルタイムの音声合成が可能になることは知られていません。確認できている範囲で最高クラスの性能は、高度に最適化された手書きのCUDAカーネルを使用してリアルタイムの音声合成を実装できるNVIDIA* nv-wavenet[1]です。

このホワイトペーパーでは、サンプル・リポジトリーで提供されているnv-wavenetの最大構成とほぼ同数のパラメーターをネットワークの繰り返し部分で使用したWaveNetモデルを実装します。ネットワークの繰り返し部分は、レイテンシーの影響を最も受けやすい領域です。インテル® Stratix® 10 NX FPGAを使用したブロック浮動小数点(BFP16)の量子化によって、このモデルを導入し、256個の16kHzストリームをリアルタイムで生成できます。160個の24kHzストリームと128個の32kHzストリームをリアルタイムで出力することで、FPGA実装の高い周波数でも効率を維持できることを示しています。

このホワイトペーパーの以降の構成は次のとおりです。セクションIIでは、WaveNetモデル・アーキテクチャーについて詳しく説明します。セクションIIIでは、量子化とBFP16データ形式を使用したモデル圧縮の概念について説明します。セクションIVでは、AI Tensorブロック、BFP16演算、広帯域幅メモリー(HBM)の使用など、インテル® Stratix® 10 NX FPGAへのWaveNetモデルの実装について説明します。セクションVでは、実装の性能や生成されるオーディオの品質など、結果を示します。最後にセクションVIでは、結果をまとめ、データセンターでのニューラル・ネットワークの将来の展開におけるFPGAの重要性を示します。

[1] https://github.com/NVIDIA/nv-wavenet(英語)

層	タイプ	パラメーター数		GOP/秒のオーディオ	
		層当たり	合計	層当たり	合計
前処理層					
埋め込み	埋め込み		30,720		-
特徴量のアップサンプリング	ConvTranspose1d		5,120,080		0.82
繰り返し層					
拡 張	拡張Conv1d	57,840	925,440	1.84	29.49
条件付き	Conv1d	19,440	311,040	0.61	9.83
残 差	Conv1d	14,520	217,800	0.46	6.91
スキップ	Conv1d	29,040	464,640	0.92	14.75
後処理層					
出 力	Conv1d		61,440		1.96
終 了	Conv1d		65,536		2.09
合 計			7,196,696		65.85

表 1. モデルのパラメーターと 1 秒の合成オーディオ出力当たりに必要な演算回数の内訳

II. モデルの説明

オーディオ生成のタスクでは、値のシーケンス $[x_1,...,x_n]$ を生成します。通常のオーディオファイルと同様に、各値 x_i は波形から切り取った個々のサンプルを表す整数です。オーディオストリームには、サンプルレートとビット深度という 2 つの重要なパラメーターがあります。サンプルレートは、1 秒のオーディオを構成するサンプルの数であり、ヘルツで測定されます。ビット深度は、各 x_i が取ることができる離散値の範囲です。この 2 つのパラメーターを増やせばオーディオストリームの品質は向上しますが、1 秒により多くのサンプルを生成するか、各サンプルの演算数を増やさなければなりません。このホワイトペーパーでは、人間に近いレベルの音質と高い性能の両方を得られるという理由から、サンプルレートを 16kHz、ビット深度を 16 (サンプルごとに 65,536 個の値を使用可能) としています。

WaveNet は、前のステップでのサンプルすべてを条件として各サンプルを生成する条件付き確率 $p(x_t|x_1,..., x_{i-1})$ をモデル化する畳み込みニューラル・ネットワークであり、特定ステップでのモデルの第 1 層への入力は、前のタイムステップからのモデル出力の埋め込みです。ネットワークのコアは、それぞれが 4 つの畳み込みを含む一連の繰り返し層で構成されます (図 1 を参照)。各繰り返し層の第 1 の畳み込みは、現在のステップ t と過去のステップでの前層の出力を連結したものを入力とする拡張畳み込みです。k 番目の繰り返し層の過去のステップは、拡張周期パラメーター D によって異なり、ステップ $t - 2^{k \bmod D}$ に設定されます。

各繰り返し層ではスキップ出力も生成し、このスキップ出力を合計して、さらに 2 つの畳み込み層での後処理に続いて最終モデル出力の生成に使用します。前の層からの出力に加えて、各繰り返し層はどういったオーディオが生成されるのかを制御する条件付き入力も受け取ります。条件付き入力は、アップサンプリング転置畳み込みを条件付き特徴量に適用することによって生成されます。WaveNet の初期バージョンでは、この条件付き特徴量は単純な言語特徴量でしたが、現在のアプローチでは通常、Tacotron 2 [2] や Deep Voice 3 [11] などの別のモデルによって生成されたスペクトログラムを使用します。

2 つの後処理畳み込み層が適用された後、モデルは確率分布 $p(x_t|x_1,..., x_{i-1})$ からサンプリングし、これにより確率が最も高い値を選択するといったほかの方法よりも忠実度の高い音声が生成されることが分かっています。このモデルは、ビット深度 16 のオーディオ出力を生成しますが、2^{16} = 65,536 個の値を生成する際の演算オーバーヘッドを削減するために、代わりに 8 ビットの μ -law で符号化された値を生成し、その後でこれらの値を復号して、最終的な 16 ビット・オーディオ・サンプルを生成します。

推論では 1 つのステップの出力を算出するには前のステップの値が必要となるため、各ステップを個別に順次計算しなければならず、16kHz のオーディオの場合、リアルタイムで音声を生成するには、このモデルを 1 秒当たり 16,000 回実行することになります。つまり、モデルを介した 1 回のフォワードパスにかけられる時間は最大でも 62.5 µs ということです。

WaveNet は、畳み込みのチャネル数 (スキップ / 残差 / オーディオチャネルの数) と、繰り返し層の数、拡張周期パラメーター D によってパラメーター化されます。このホワイトペーパーでは、**s** = 240 個のスキップチャネル、**r** = 120 個の残差チャネル、**a** = 256 個のオーディオチャネル、**L** = 16 個の繰り返し層、D = 8 のモデルを使用します。表 1 に示すように、このモデルには 720 万個のパラメーターがあり、1 秒のオーディオの生成に約 65.85 GFLOPS の演算が必要となります。ここでは ConvTranspose1d 層を CPU に、ほかのすべての層を FPGA にマップしました。これにより FPGA で保持するモデル・パラメーター数が大幅に削減され、FPGA 全体で格納するパラメーターは約 210 万個になります。それでも FPGA で実行する層が演算の大部分を占め、モデルの全演算の 98.8% となっています。

III. モデル圧縮

量子化はモデル圧縮の一手法です。モデルの量子化は、パラメーターやアクティベーションに別の数値形式を使用することで成り立ち、これによりメモリー要件を緩和できるだけでなく、ハードウェアで演算をより効率的に実行できるため、消費電力とレイテンシーの低減につながります。

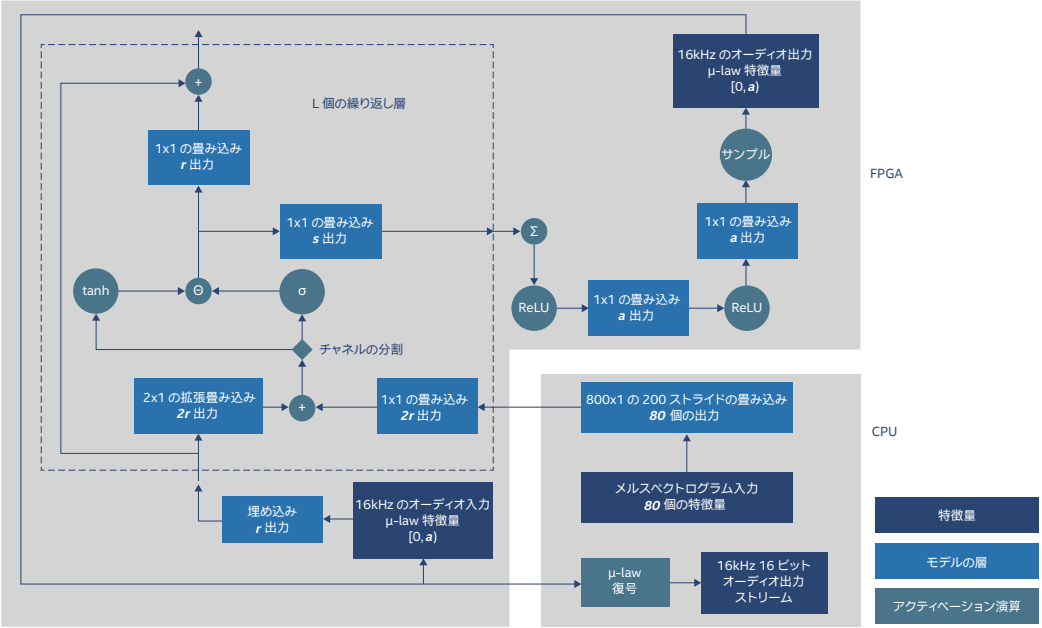

図 1. スキップチャネル、残差チャネル、オーディオチャネルの数を **s**、**r**、**a**、層の数を **L** でパラメーター化した WaveNet アーキテクチャー

パラメーターやアクティベーションを別の数値形式に変換するには、各値を表すビット数を減らすだけでなく(例 : 32 ビットの代わりに 16 ビットを使用)、これらの値で演算を実行するときの算術演算の変更(例 : 浮動小数点演算の代わりに整数演算を使用) が必要になる場合もあります。

モデルの量子化で取るアプローチには 2 つあり、1 つが、モデルをトレーニングした後にパラメーターを目的の数値形式に変換するトレーニング後の量子化(PTQ) です。このアプローチはシンプルですが、場合によっては、モデルの品質や精度に悪影響を及ぼす可能性があります。もう片方は Quantization Aware Training(QAT)[12] で、重みに対する目標数値精度をトレーニング中にシミュレートすることで品質低下を防ぎ、この方法により量子化プロセスで発生するノイズのより適切な除去が可能になります。

BFP16 量子化

インテル® Stratix® 10 NX FPGA には、ブロック浮動小数点 16 (BFP16) で計算する AI Tensor ブロックが組み込まれています。BFP16 は、同じビット数を維持しながら、値を量子化するプロセスを変更することで、IEEE 規格の 16 ビット浮動小数点数値形式(FP16) と比較して量子化誤差を抑えることを目的とした数値形式です。浮動小数点ユニットよりもはるかに効率的な固定小数点演算ユニットで浮動小数点演算を使用できるため、特に FPGA/ASIC プロセッサーで使用される数値形式となっています。

数値群を BFP16 に量子化するには、目的のブロックサイズ N を定義する必要があります。対象の数値群は、事前に設定したパラメーターによって決定されたサイズで複数のブロックに分割されるため、各ブロック X は次のようになります。

$$X = (x_1,...,x_i,...,x_N)$$

x_i は、数値群内の i 番目 の値です。

次のステップでは、各ブロックの最大指数 ϵ_X を求めます。

$$\epsilon_X = \max_{i \in [1,N]} e_i$$

ei は、数値群内で i 番目の数値の指数です。

この指数は量子化後に対象群内のすべての数値で共有されますが、仮数部は最大指数と一致するようにシフトされます。

$$m'_i = m_i > (\epsilon_X - ei)$$

$m'i$ は量子化後の i 番目の数値の仮数、mi は量子化前の i 番目の数値の仮数です。> は、右シフトのビット幅演算を表します。

L_e が指数のビット数、L_m が仮数のビット数、1 ビットが符号の表現に使用される場合、n 個の数値の平均長は、$1 + L_m + L_{e/n}$ となるため、ブロック浮動小数点表現はメモリーとインターコネクト帯域幅の両方の要件を緩和することができます。これに対して、浮動小数点表現の場合、すべての数値の指数値が異なるため、n 個の数値の平均長は、$1 + L_m + L_e$ となります。

BFP16 では、仮数に 7 ビット、指数に 8 ビット、符号に 1 ビットを使用します。ブロックサイズは、量子化プロセス中に生じる誤差と各演算に必要なメモリー / 帯域幅のトレードオフを制御するため、重要なパラメーターです。ブロックサイズが大きいほど、量子化誤差も大きくなりますが、表現に必要な領域と帯域幅は少なくなります。

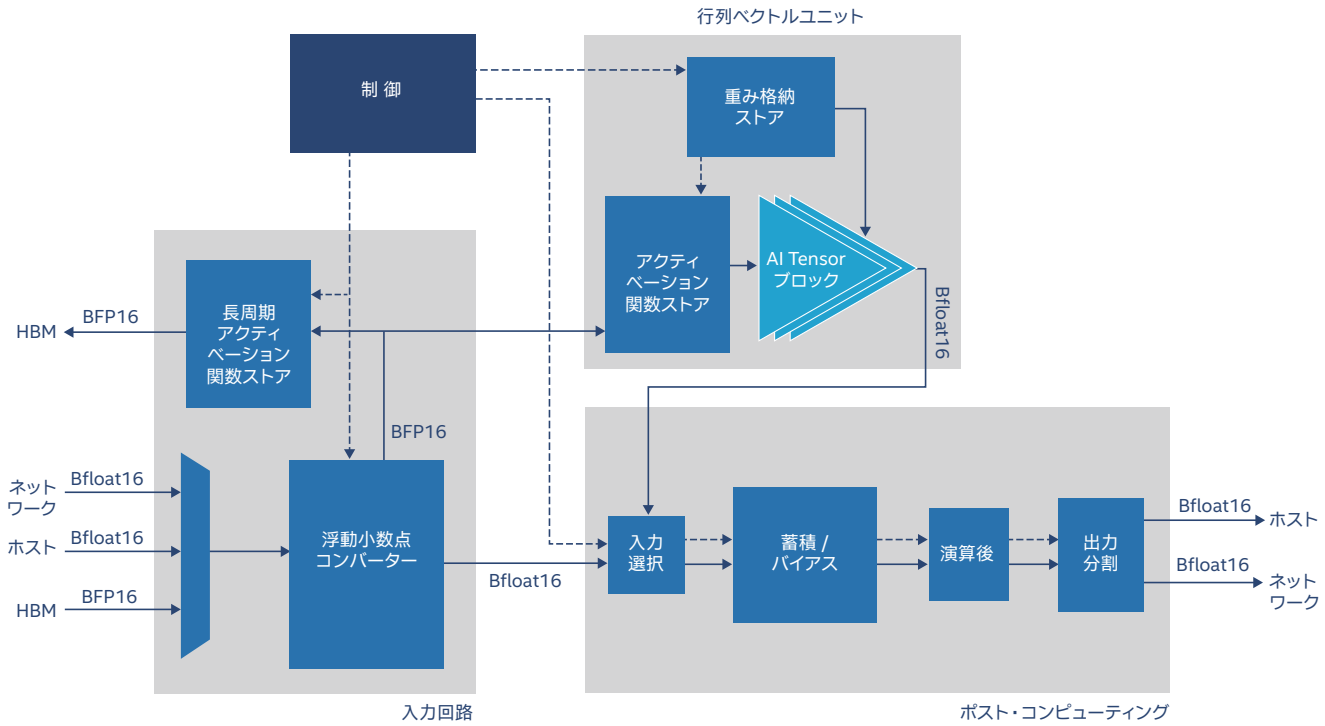

図 2. インテル® Stratix® 10 NX FPGA の MAU* コア・アーキテクチャー

IV. ハードウェアへの実装

ここでは WaveNet モデルをインテル® Stratix® 10 NX FPGA に実装し、入力での ConvTranspose1d 層を除いて、すべての層が FPGA で処理されます。この前処理層は前のステップで生成された値に依存しないため、コアの自己回帰 WaveNet ループの外側で実行できます。また、12.5ms の自己回帰ループよりも処理の頻度が少なく、メイン自己回帰ループにはレイテンシー要件もありません。この層を CPU に実装しているのは、FPGA 処理をレイテンシーが重視されるネットワークの自己回帰部分専用にするためです。ネットワークの残りの部分は FPGA で実行され、この中には条件付きアップサンプリング層に続く 1x1 の畳み込み層も含まれます。厳密にはコアの自己回帰ループの一部ではありませんが、この畳み込み層がネットワークの全体的な演算の大部分を占めるため、コンピューティング性能を活かすために FPGA に配置しました。

FPGA 処理アーキテクチャー

ネットワークを FPGA で実装するために、Myrtle.ai プログラマブル MAU* コア・アーキテクチャーを選択しました。MAU* コアは、ディープ・ニューラル・ネットワーク用のプログラマブル処理エンジンであり、FPGA ファブリックのオーバーレイにより柔軟でランタイムの構成が可能な推論エンジンを実現します。ここではインテル® Stratix® 10 NX FPGA アーキテクチャーに最適化された 4 つの MAU* コアを FPGA に配置し、高いクロック周波数を維持しながら、高レベルのロジック使用率を確保できるように、インテル® Quartus® Prime 開発ソフトウェアで MAU* コアのフロアプランを作成しました。このデザインで FPGA は 240 MHz のコア処理周波数を使用します。

各 MAU* コアは、専用の広帯域幅メモリー (HBM) インターフェイスである PCIe* を介してホスト CPU に接続し、ルーティング・ネットワーク経由でほかの MAU* コアに接続します。隣接する MAU* コアの各ペア間のルーティング・ネットワークは、クロックサイクルごとに 120 個の bfloat16 値を伝送するので、内部バス帯域幅は 60GBps になります。

インテル® Stratix® 10 NX FPGA に最適化された MAU* コアのアーキテクチャーを図 2 に示します。このアーキテクチャーには、各層内で畳み込みを実装するために必要な機能がすべて含まれています。MAU* コアは、畳み込み層に加えて、MAU* コアのポスト・コンピューティング・ユニットを形成する tanh、シグモイド、ReLU、ゲート・アクティベーション、softmax サンプリング、ワンホット埋め込み演算といった関数も備わっています。

インテル® Stratix® 10 NX FPGA の AI Tensor ブロックの活用

インテル® Stratix® 10 NX FPGA には、インテル® Stratix® 10 FPGA シリーズのほかの FPGA と比較して 15 倍の INT8 TOPS を実現する革新的な AI Tensor ブロックが組み込まれています。インテル® Stratix® 10 NX FPGA の AI Tensor ブロックを活用して 1D 畳み込み層を効率的に実装するために、Tensor モードで動作するようにブロックを構成しました。演算は 10 個の行列要素の各ブロックに共有指数を使用して BFP16 で実行されます。インテル® Stratix® 10 NX FPGA の各 AI Tensor ブロックでは、クロックサイクルごとにサイズ 10 の 3 つのドット積を計算できます。4 つの AI Tensor ブロックをカスケード接続して、120 のドット積幅が形成されます。インテル® Stratix® 10 NX FPGA の AI Tensor ブロックのアーキテクチャーを図 3 に示します。

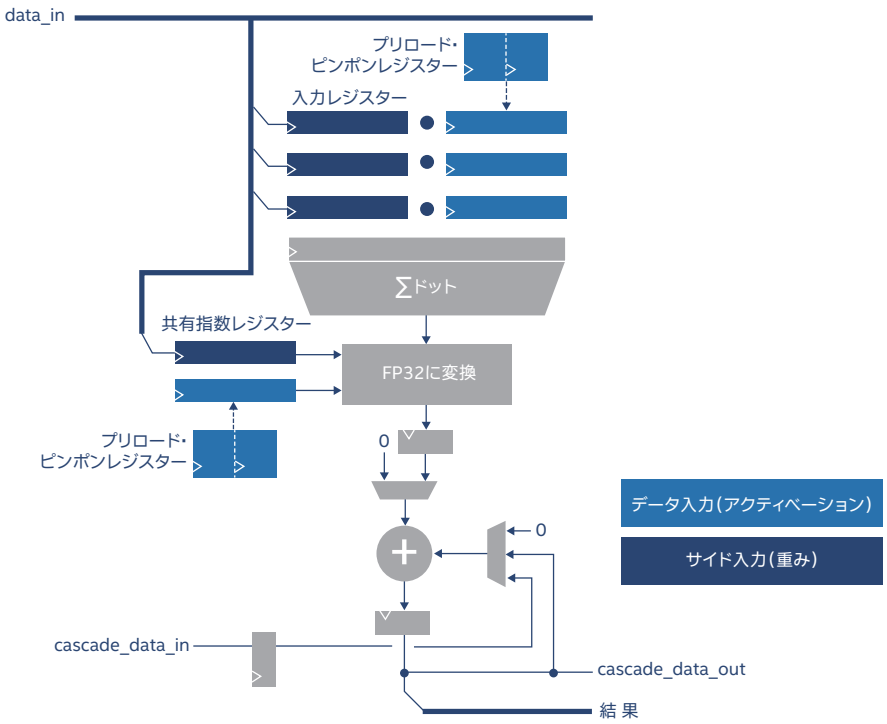

図 3. インテル® Stratix® 10 NX FPGA の AI Tensor ブロック[2]

モデルの重みはサイド入力によって読み込まれます。AI Tensor ブロックでは、新しい重みを読み込むために 18 クロックサイクルを必要とし、この更新はピンポンレジスターによる演算と並行して発生します。ここでは 32 のバッチで異なるチャネルを処理して、バッチごとに重みを更新できるように、Tensor モードの演算を最大効率で活用し、各 MAU* コアで 480 個の AI Tensor ブロックを使用しました。その結果、各 MAU* コアでクロックサイクルごとに単一の 120x120 行列ベクトル乗算を実行できるようになっています。こうしてこのアクセラレーター・デザインで、全 4 つのコアで合計 27.6 TOPS の性能を実現できます。

モデル・パラメーターは、オンチップ RAM にローカルに保持されます。WaveNet は比較的小さなモデルであるため、すべてのネットワーク・パラメーターを AI Tensor ブロックにローカルに保持するのもオンチップメモリーで十分です。この設計では、1.3TBps の帯域幅で 4.5MB のローカルストレージをネットワーク・パラメーターに割り当てています。

FPGA での拡張畳み込みの実装

推論には、モデル・アーキテクチャーの拡張畳み込みを、通常の畳み込みと、前のステップで計算した値を格納するバッファーに置き換える生成アルゴリズム Fast WaveNet [13] を使用しました。これにより、拡張畳み込みをシミュレートする適切な値をバッファーから選択でき、バッファー内の値を検索すればよいため、新しい各値の生成時に過去のサンプルから取得した値を再計算する必要がなくなり、モデルの性能が大幅に向上します。

ネットワークの拡張畳み込みには、最大で過去 128 タイムステップ分のアクティベーション・データが必要です。これにより、中間パラメーターの格納に高スループットかつ低レイテンシーの要件が課せられます。WaveNet ネットワークの内部では、アクティベーションを取得して拡張畳み込みに渡すための帯域幅の要件は、MAU* コア当たり 3.1GBps です。最大で過去 128 タイムステップ分のアクティベーションを格納するには、各コアで計 10.3MB のストレージが必要になりますが、このサイズのデータストアはコア内部にローカルで収容するには大きすぎるため、HBM メモリーを使用してこの情報を格納します。

プラットフォーム上の HBM から利用可能な帯域幅は 256GBps であるのに対し、外部 DDR4 メモリーから利用可能な帯域幅は約 21GBps であり、システム内のデータ移動で帯域幅にかかるオーバーヘッドが大きくなるため、プラットフォーム上では HBM のほうが外部 DDR4 メモリーを使用するよりも優先されます。

プログラミング・モデル

WaveNet モデルでは、MAU* コアは実行時にプログラムされるため、FPGA を別のモデル・アーキテクチャーにリターゲティングすることができ、FPGA 実装を再コンパイルする必要もありません。

このモデルでは各コアで複数の WaveNet 層を実行し、コアロジックを時分割することで、WaveNet を 4 つの MAU* コアにマッピングしました。MAU* コア・コントローラーがコアロジックの実行を処理します。各コアは 32 音声チャネルのバッチで動作し、連続するクロックサイクルでこのバッチの各チャネルに対して同じ命令を実行。バッチで層のすべての演算が完了すると、後続の層での処理のために、アクティベーションは次の MAU* コアに進みます。32 音声チャネルのバッチが 4 つのコアそれぞれで同時に処理され、FPGA で常に 128 の音声チャネルが使用できるように、4 つのコアがパイプライン化されています。

128 以上の同時音声チャネルをリアルタイムで処理するために、音声チャネルはチャンクとして処理されます。チャンクはメルスペクトログラム入力の整数値です。FPGA アクセラレーションの時分割を可能にすることで、FPGA はリアルタイムよりもはるかに高速でサンプルを生成できます。FPGA は、HBM メモリー内に保持されているすべてのアクティブな同時音声チャネルに対し中間状態で、チャンクを非連続動作モードで処理できるようにデザインされています。

[2] 提供 : インテル プログラマブル・ソリューションズ事業本部、Graham McKenzie

表 2 に示すように、WaveNet ではネットワーク層を MAU* コアに配置しています。

コア番号	層マッピング
1	入力埋め込み層、WaveNet 層 1 ~ 3
2	WaveNet 層 4 ~ 8
3	WaveNet 層 9 ~ 13
4	WaveNet 層 14 ~ 16、出力畳み込み層

表 2. MAU* コアへの WaveNet のマッピング

V. 結果

手 法

ここでは 16kHz にサブサンプリングされた LJSpeech データセットを用いています。[14] このプロジェクトを開始するにあたり、検証セット用、テストセット用としてそれぞれに 100 個のサンプルがデータセットからランダムに選択されました。残りのサンプルはすべてトレーニング・セットとして使用されます。データ補完は適用されません。

WaveNet アーキテクチャーのパラメーターは、セクション II で定義されているとおりです。混合精度トレーニングを使用し、1 ~ 8 個の GPU に分散されたバッチサイズ 16 で、すべてのモデルの 14 万ステップに対してトレーニングを実行します。バッチ内の 1 つの要素は、トレーニング・セット内のオーディオクリップからランダムに選択された 1 秒のセグメントで構成され、必要に応じてゼロで埋められます。固定学習率が 10^{-3}、$\beta_1 = 0.9$、$\beta_2 = 0.999$、$\epsilon = 10^{-8}$ の設定で、Adam オプティマイザー [15] を使用しました。WaveNet の実装に PyTorch*[16] を、インテル® QAT による BFP16 のシミュレートに QPyTorch[17] を使用しています。

教師強制交差エントロピーの検証 / テスト損失と、各モデルの平均オピニオン評点 (MOS) の両方を記録。モデルごとにトレーニング・プロセスを 3 回繰り返し、検証損失の中央値を持つモデルを選択して、テスト損失とテスト MOS スコアを算出計算して記録しました。MOS は、30 人の異なる Amazon Mechanical Turk* ワーカーに、生成された各サンプルがどの程度自然に聞こえるかを 5 段階で評価してもらうことによって割り出します (表 3 を参照)。記載している MOS はこれらのスコアの平均として、95% の信頼区間は t 分布で算出しています。

評 価	ラベル	説 明
1	Bad	完全に不自然な音声
2	Poor	大部分が不自然な音声
3	Fair	自然な音声と不自然な音声が均等
4	Good	大部分が自然な音声
5	Excellent	完全に自然な音声

表 3. 平均オピニオン評点の尺度

モデルの品質

検証 / テストセットの損失、ベースライン FP32 モデルと PTQ/QAT それぞれを使用した BFP16 モデルの MOS を表 4 に示します。BFP16 (PTQ) モデルと BFP16 (QAT) モデルはどちらも、ベースライン FP32 モデルに近い高忠実度のオーディオ合成が可能であることが分かりました。

モデル	検証損失	テスト損失	テスト MOS
人 間	-	-	4.056 ± 0.034
FP32	2.189	2.182	3.976 ± 0.027
BFP16 (PTQ)	2.203	2.197	3.711 ± 0.029
BFP16 (QAT)	2.195	2.188	3.823 ± 0.028

表 4. FP32/BFP16 WaveNet モデルの品質結果

モデルの性能

インテル® Stratix® 10 NX FPGA で実行した WaveNet モデルの主要パラメーターと処理性能を表 5 に示します。V100 GPU で実行された nv-wavenet との比較です (このホワイトペーパーの執筆時点で参照できる範囲で最速の WaveNet 実装であるため)。

消費電力は、2 つのプラットフォームの熱設計電力 (TDP) に基づいて比較しました。どちらのプラットフォームも、消費電力が実際のアプリケーションより大きく見積もられています。インテル® Stratix® 10 NX FPGA の TDP は、PCIe* ベースのアクセラレーター・カードの導入用にインテルが提供しているデザイン解析に基づく数値です。

同時音声チャネルの数は、16kHz のサンプルレートを満たしてリアルタイムで生成できるチャネルの最大数です。FPGA 実装では GPU 実装と比べてわずかに小さい層サイズを使用していますが、モデルの高速化部分に、nv-wavenet 実装にはない条件付き層が含まれているため、モデルのパラメーター数と演算数が多くなっています。

インテル® Stratix® 10 NX FPGA の実装では、nv-wavenet と比較して 8 倍の同時音声ストリーム数を示し、WaveNet モデルで 8.6 倍高い TOPS を実現します。

FPGA ならば、消費電力当たりの音声チャネル数を 9.3 倍、GOPS/W を 10 倍向上できる、電力効率の極めて高いソリューションを構築することが可能です。

表 6 では 16kHz 以上のオーディオ周波数の同時音声チャネル数を示しています。ここから、単一タイムステップを生成する際のレイテンシー要件を緩和したとしても、FPGA ソリューションの効率は維持できることが分かります。

	Myrtle.ai WaveNet	nv-wavenet
プラットフォーム	インテル® Stratix® 10 NX FPGA	NVIDIA* V100 GPU
周波数(MHz)	240	1530
数値精度	BFP16/bfloat16	fp16
WaveNet 構成	r=120、s=256、L=16、a=256、D=8	r=128、s=256、L=16、a=256、D=8
1 秒のオーディオ当たりの演算数(GOPS/ 秒)	65.03	60.36
モデル・パラメーター数(単位 : 100 万)	2.08	1.99
同時音声チャネル数	256	32
アプリケーションの TOPS	16.6	1.93
TDP(W)	215	250
消費電力当たり性能(GOPS/W)	77.4	7.7
消費電力当たりの音声チャネル数(1/W)	1.19	0.128

表 5. 16kHz での WaveNet 実装の性能結果

オーディオ周波数	同時音声チャネル数	
	Myrtle.ai WaveNet	nv-wavenet
16kHz	256	32
24kHz	160	16
32kHz	128	8

表 6. オーディオ周波数別の WaveNet 実装の性能結果

ConvTranspose1d は、2.80GHz で動作するインテル® Xeon® プロセッサー(デュアルソケット、16 コア CPU) に実装されています。各入力ステップが 200 の出力ステップ (ストライド) に相当するため、16kHz のオーディオの出力ステップを生成するには、1 つの入力ステップ当たり 12.5ms で処理しなければなりません。そこで PyTorch* のネイティブ実装よりも桁違いの速さで動く独自の ConvTranspose1d バリアントを実装しました。各ソケットで独立して実行されるバッチサイズ 64 を使用しています。バッチごとに 3.85ms で 99.999 パーセンタイルのレイテンシーを計測しました。これは 12.5ms の処理要件を満たし、各ソケットでサイズ 64 の 3 つのバッチを十分に実行できる速度です。この構成により、CPU は最大 384 の同時音声チャネルの ConvTranspose1d を計算することが可能になり、FPGA 実装への入力にも十分対応します。また、256 の同時音声チャネルを生成できる WaveNet の完全なシステム実装も示されています。

VI. まとめ

このホワイトペーパーでは、FPGA ベースの専用アクセラレーターを活用し、最先端の WaveNet モデルで 256 の同時ストリームにより人間に近いレベルの合成音声を生成するリアルタイム性能を実現できることを示しました。これは現時点でこのモデルに使用できる最高レベルの GPU ソリューションと比べて 8 倍の性能です。

より高い周波数では FPGA の優位性がさらに高まり、現在利用可能な最高レベルの GPU ソリューションと比較して、24kHz では 10 倍、32kHz では 16 倍高い性能が得られることが分かりました。これにより、1 つのアクセラレーターで 32kHz のオーディオの 128 同時ストリームが可能になり、より高いオーディオ周波数で WaveNet を展開するコスト効率の高いプラットフォームを構築できます。

FPGA は、WaveNet の実装で大幅な消費電力の低減を示しました。GPU 実装と比較すると 10 分の 1 に抑えることができるため、リアルタイムの音声合成を大規模に展開する場合の省電力効果は絶大です。

トレーニング後に BFP16 形式を適用することで、精度の損失を最小限に抑えながら、FP32 からシンプルな量子化フローが可能になることも実証されています。FPGA により、マシンラーニング・フレームワークからシンプルかつ効果的な量子化フローだけでなく、さらに効率的なハードウェア実装のメリットが得られるのは間違いありません。

参考資料

[1] A. v. d. Oord、S. Dieleman、H. Zen、K. Simonyan、O. Vinyals、A. Graves、N. Kalchbrenner、A. Senior、K. Kavukcuoglu、「Wavenet: A generative model for raw audio」、arXiv preprint arXiv:1609.03499、2016年。

[2] J. Shen、R. Pang、R. J. Weiss、M. Schuster、N. Jaitly、Z. Yang、Z. Chen、Y. Zhang、Y. Wang、R. Skerrv-Ryan他、「Natural tts synthesis by conditioning wavenet on mel spectrogram predictions」、2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)。IEEE、2018年、pp. 4779–4783。

[3] A. Oord、Y. Li、I. Babuschkin、K. Simonyan、O. Vinyals、K. Kavukcuoglu、G. Driessche、E. Lockhart、L. Cobo、F. Stimberg他、「Parallel wavenet: Fast high-fidelity speech synthesis」、International Conference on Machine Learning、2018年、pp. 3918–3926。

[4] N. Kalchbrenner、E. Elsen、K. Simonyan、S. Noury、N. Casagrande、E. Lockhart、F. Stimberg、A. v. d. Oord、S. Dieleman、K. Kavukcuoglu、「Efficient neural audio synthesis」、arXiv preprint arXiv:1802.08435、2018年。

[5] W. Ping、K. Peng、J. Chen、「Clarinet: Parallel wave generation in end-to-end text-to-speech」、arXiv preprint arXiv:1807.07281、2018年。

[6] R. Prenger、R. Valle、B. Catanzaro、「Waveglow: A flow-based generative network for speech synthesis」、ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)。IEEE、2019年、P.3617-3621。

[7] S. Kim、S.-g. Lee、J. Song、J. Kim、S. Yoon、「Flowavenet: A generative flow for raw audio」、arXiv preprint arXiv:1811.02155、2018年。

[8] Q. Tian、Z. Zhang、H. Lu、L.-H. Chen、S. Liu、「Featherwave: An efficient high-fidelity neural vocoder with multi-band linear prediction」、arXiv preprint arXiv:2005.05551、2020年。

[9] P.-c. Hsu、H.-y.Lee、「Wg-wavenet: Real-time high-fidelity speech synthesis without gpu」、arXiv preprint arXiv:2005.07412、2020年。

[10] S. Hussain、M. Javaheripi、P. Neekhara、R. Kastner、F. Koushanfar、「Fastwave: Accelerating autoregressive convolutional neural networks on fpga」、arXiv preprint arXiv:2002.04971、2020年。

[11] W. Ping、K. Peng、A. Gibiansky、S. O. Arik、A. Kannan、S. Narang、J. Raiman、J. Miller、「Deep voice 3: Scaling text-to-speech with convolutional sequence learning,」、arXiv preprint arXiv:1710.07654、2017年。

[12] B. Jacob、S. Kligys、B. Chen、M. Zhu、M. Tang、A. Howard、H. Adam、D. Kalenichenko、「Quantization and training of neural networks for efficient integer-arithmetic- only inference」、IEEE Conference on Computer Vision and Pattern Recognitionの議事録、2018年、P.2704–2713。

[13] T. L. Paine、P. Khorrami、S. Chang、Y. Zhang、P. Ramachandran、M. A. Hasegawa-Johnson、T. S. Huang、「Fast wavenet generation algorithm」、arXiv preprint arXiv:1611.09482、2016年。

[14] K. Ito、L. Johnson、「The lj speech dataset」、https://keithito.com/LJ-Speech-Dataset/ (英語)、2017年。

[15] D. P. Kingma、J. Ba、「Adam: A method for stochastic optimization」、arXiv preprint arXiv:1412.6980、2014年。

[16] A. Paszke、S. Gross、F. Massa、A. Lerer、J. Bradbury、G. Chanan、T. Killeen、Z. Lin、N. Gimelshein、L. Antiga、A. Desmaison、A. Kopf、E. Yang、Z. DeVito、M. Raison、A. Tejani、S. Chilamkurthy、B. Steiner、L. Fang、J. Bai、S. Chintala、「Pytorch: An imperative style, high-performance deep learning library」、Advances in Neural Information Processing Systems 32、H. Wallach、H. Larochelle、A. Beygelzimer、F. d'Alch´e-Buc、E. Fox、R. Garnett編。Curran Associates, Inc.、2019年、P.8024–8035。[オンライン] 入手先：http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (英語)

[17] T. Zhang、Z. Lin、G. Yang、C. De Sa、「Qpytorch: A low-precision arithmetic simulation framework」、arXiv preprint arXiv:1910.04540、2019年。

WP-01304-1.0/JP