白皮书

FPGA
人工智能

基于英特尔® Stratix® 10 NX FPGA 的 WaveNet 助力实现实时语音合成

作者

Jonny Shipton
软件负责人
Myrtle.ai

Jon Fowler
FPGA 负责人
Myrtle.ai

Chris Chalmers
高级开发人员
Myrtle.ai

Sam Davis
机器学习负责人
Myrtle.ai

Sam Gooch
机器学习工程师
Myrtle.ai

Giuseppe Coccia
机器学习工程师
Myrtle.ai

目录

摘要

实时文本转语音模型在交互式语音服务领域有着广泛的应用，例如语音助手和智能音箱等。但是，由于延迟方面存在严格的要求，企业在部署这些模型时通常面临重重挑战。本文展示了 WaveNet 的实施过程，WaveNet 是一种先进的声码器，可实时生成接近人类质量水平的 256 16 kHz 音频流：吞吐量比手动优化的 GPU 解决方案高 8 倍。我们在实施过程中采用英特尔® Stratix® 10 NX FPGA，结果表明 FPGA 能够为基于神经网络的实际应用提供高吞吐量、低延迟推理等优势。

I. 引言

文本转语音（TTS）领域发展迅猛，这在很大程度上促进了语音助手的普及。最新的语音合成系统包含两个神经网络，它们可按顺序运行，最后生成音频。第一个模型将文本作为输入并生成声谱图等声学特征，第二个模型（被称作声码器）收集这些中间特征并生成语音。Tacotron 2 通常被用作第一个模型。本文主要研究语音合成系统中的第二个模型。WaveNet [1] 是一种先进的声码器，可生成接近人类自然程度的语音 [2]。保证模型质量的关键在于自回归环路，这一点也让在实时应用中网络实施方面极具挑战性。

因此，TTS 的研究重点是寻找可替代的声码器架构，例如并行 WaveNet [3]、WaveRNN [4]、ClariNet [5] 和 WaveGlow [6]，它们更适合在现有硬件上高效执行推理。由于音频质量评估具有一定程度的主观性，因此难以清晰界定哪款声码器质量最高，但是所有作者一致同意原始 WaveNet 架构生成的音频质量至少等于，甚至高于最近的方法 [3], [6]–[9]。

在使用替代架构（包括其他 FPGA 解决方案）之前，过去有人尝试对 WaveNet 模型进行直接加速 [10]，但是实时音频合成的效果并未可知。我们了解到的性能最高的实施是 NVIDIA* nv-wavenet[1]，它可以使用高度优化的手写 CUDA 内核实现实时音频合成。

在本文中，我们实施了一个 WaveNet 模型，其网络重复部分的参数数量大致相当于示例存储库中可用的最大 nv-wavenet 配置的参数数量。网络的重复部分是对延迟最敏感的部分。通过使用块浮点（BFP16）量化和英特尔 Stratix 10 NX FPGA，我们能够部署该模型并实时生成 256 个 16 kHz 流。我们展示了 FPGA 实现方案可以在更高的频率下保持高效运行，并实时生成 160 个 24 kHz 流和 128 个 32 kHz 流。

后文的结构如下：第 2 节详细介绍了 WaveNet 模型架构。第 3 节探讨了使用量化和 BFP16 数据格式进行模型压缩的概念。第 4 节介绍了如何使用 AI 张量块、BFP16 计算和高带宽内存（HBM）等在英特尔 Stratix 10 NX FPGA 上实施 WaveNet 模型。第 5 节展示了实施性能和生成的音频质量等结果。最后，第 6 节对结果进行了总结，并解释了这些结果对于未来使用 FPGA 在数据中心部署神经网络的重要性。

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

层	类型	参数数量		GOP/秒音频	
		每层	总数	每层	总数
预处理层					
嵌入	嵌入		30,720		-
特征上采样	ConvTranspose1d		5,120,080		0.82
重复层					
扩张	扩张 Conv1d	57,840	925,440	1.84	29.49
条件	Conv1d	19,440	311,040	0.61	9.83
残差（Residual）	Conv1d	14,520	217,800	0.46	6.91
跳跃	Conv1d	29,040	464,640	0.92	14.75
后处理层					
输出	Conv1d		61,440		1.96
结束	Conv1d		65,536		2.09
总数			7,196,696		65.85

表 1. 模型参数和每秒合成音频输出所需的千兆次操作数明细。

II. 模型描述

音频生成任务是生成一组值 $[x_1,...,x_n]$。和常规音频文件中的一样，每个值 x_i 都是整数，代表来自波形的离散样本。任何音频流都包含两个重要参数，一个是采样率，即构成一秒钟音频的样本数量，其单位是赫兹；第二个是位深，即每个 x_i 可以接收的离散值范围。增加这两个参数可改善音频流的质量，但是需要每秒生成更多样本，或者增加每个样本的计算量。本文使用 16 kHz 采样率和 16 位深，为每个样本提供 65,536 个可能值，因为我们发现这有助于实现接近人类水平的质量和高性能。

WaveNet 是一个卷积神经网络，模拟根据之前所有样本生成一个样本的条件概率：$p\ (x_t|x_1,..., x_{t-1})$。在给定采样时刻，模型第一层的输入是来自之前时刻的模型输出。网络核心包括一系列重复层，每一层包含 4 个卷积，见图 1。每个重复层的第一个卷积都是扩张卷积，其中，输入是在当前采样时刻 t 和过去采样前一层输出的级联。第 k 个重复层的延时取决于扩张周期参数 D，被设置为 $t - 2^{k \bmod D}$。

每个重复层还生成一个阶跃输出。对阶跃输出求和，在后面两个卷积层进行处理后生成最终的模型输出。除了前一层的输出，每个重复层还会接收条件输入，以控制生成的音频，该条件输入由将条件特征进行上采样转置卷积来生成。在 WaveNet 的早期版本中，这些条件特征是简单的语言特征，但是当前的方法通常使用单独的模型（例如 Tacotron 2 [2] 或 Deep Voice 3 [11]）生成的声谱图。

应用两个后处理卷积层后，模型从概率分布 $p\ (x_t|x_1,..., x_{t-1})$ 中采样；我们发现相比其他方法，例如选择概率最高的值，采样可生成具有更高保真度的语音。模型生成位宽为 16 比特的音频输出。但是，为了降低生成 2^{16}= 65,536 个值的计算开销，模型生成 8 位 μ -law 编码值，然后解码以生成最终的 16 位音频样本。

在推理时，计算一个步长的输出需要上一步的值，因此有必要按顺序单独计算每一步。对于 16 kHz 音频，需每秒运行模型 16,000 次，才能实现实时音频生成。也就是说，模型的单次正向传递最多只能消耗 62.5 微秒。

WaveNet 的参数化与以下几个参数有关：卷积中的通道数量（阶跃、残差和音频通道数量）以及重复层和扩张周期参数 D。对于本文使用的模型，**s** = 240 个阶跃通道，**r** = 120 个残差通道，**a** = 256 个音频通道，**L** = 16 个重复层，D = 8。如表 1 所示，该模型有 720 万个参数，生成一秒钟音频大约需要 65.85 GFLOP 的计算。我们将 ConvTranspose1d 层映射至 CPU，并将所有其他的层映射到 FPGA。这显著减少了必须保存在 FPGA 中的模型参数数量，FPGA 上总共需要存储约 210 万个参数。在 FPGA 上运行的层仍消耗了大部分计算资源，占模型总操作的 98.8%。

III. 模型压缩

量化是一项模型压缩技术。量化模型包括使用参数和/或激活的不同数值格式。量化降低了对内存的要求以及可以在硬件中更高效地执行操作，因此功耗和延迟有所下降。

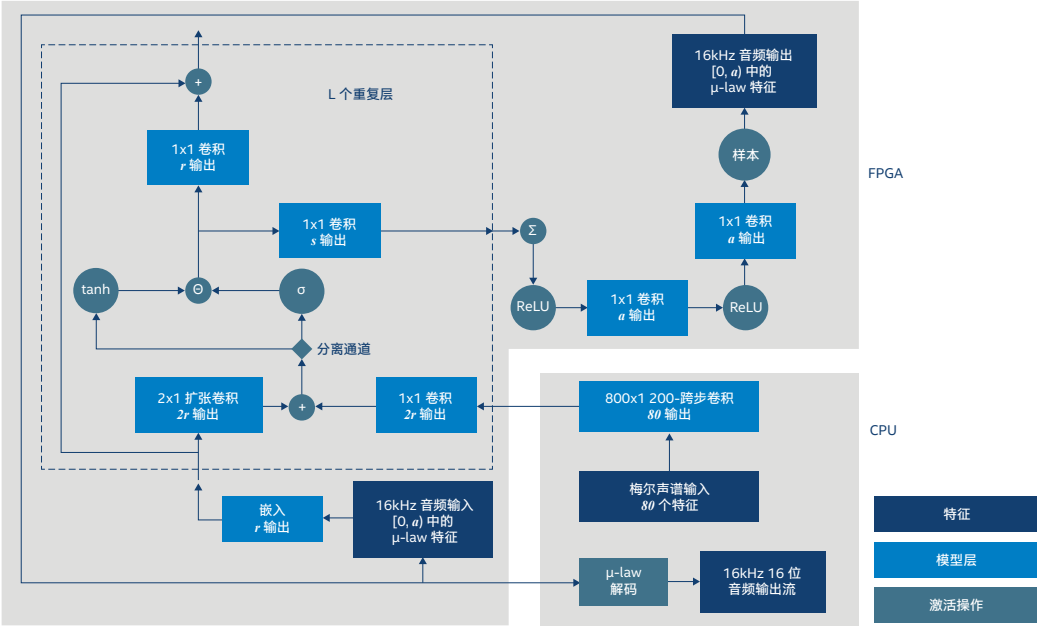

图 1. WaveNet 架构，使用 **s**、**r**、**a** 和对阶跃、残差和音频通道进行参数化，使用 **L** 对层数进行参数化。

将参数和激活转换为不同的数字格式不仅需要减少用于代表每个值的位数（例如从 32 位降至 16 位），还要更改使用这些值执行操作时所用的算法（例如从浮点变为整数运算）。

可以使用两种方法来量化模型。第一个方法是训练后量化（PTQ），指的是在对模型进行训练后，将参数转换为所需的数值格式。该方法很简单，但是在某些情况下会对模型质量或准确性造成不利影响。第二种方法是量化感知训练（QAT）[12]，该方法通过在训练时模拟权重的目标数值精度，从而遏制质量下降。这样，模型能够更好地消除量化流程中产生的噪音。

BFP16 量化

英特尔 Stratix 10 NX FPGA 包含用于执行块浮点 16（BFP16）计算的 AI 张量块。BFP16 是一种数字格式，相比 IEEE 标准 16 位浮点数字格式（FP16），BFP16 保留了相同的位数，但是改变了对数值的量化流程，目的是减少量化误差。FPGA 和一些 ASIC 处理器专门使用该数字格式，并且可以在定点运算单元上使用浮点算法，而定点运算单元的效率远高于浮点单元。

为了将一组数字量化为 BFP16，我们需要定义所需的块大小 N。目标组被分为若干个块，块大小由之前设置的参数决定。因此，每个块 X 为

$$X = (x_1,\ldots,x_i,\ldots,x_N)$$

其中，x_i 是组中的第 i 个值。

以下步骤用于查找每个块的最大指数 ϵ_X

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

其中，e_i 是组中第 i 个数字的指数。

量化后，该指数由目标组内的所有数字共享，尾数值移位，以匹配最大指数

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

其中，m'_i 是量化后第 i 个数字的尾数；m_i 是量化前第 i 个数字的尾数；> 表示右移逐位运算。

块浮点表示有助于降低对内存和互连带宽的要求，因为如果 L_e 是指数的位数，L_m 是尾数的位数，1 位用于表示一个符号，那么 n 个数字的平均长度是 $1 + L_m + L_{e/n}$。相反，通过使用浮点表示，n 个数字的平均长度是 $1 + L_m + L_e$，因为所有数字都有不同的指数值。

BFP16 使用 7 位表示尾数，8 位表示指数，1 位表示符号。块大小是一个重要参数，因为它负责控制量化过程中生成的误差和每项操作所需的内存/带宽之间的平衡。块大小越大，量化误差越大，但是表示所需的空间和带宽越小。

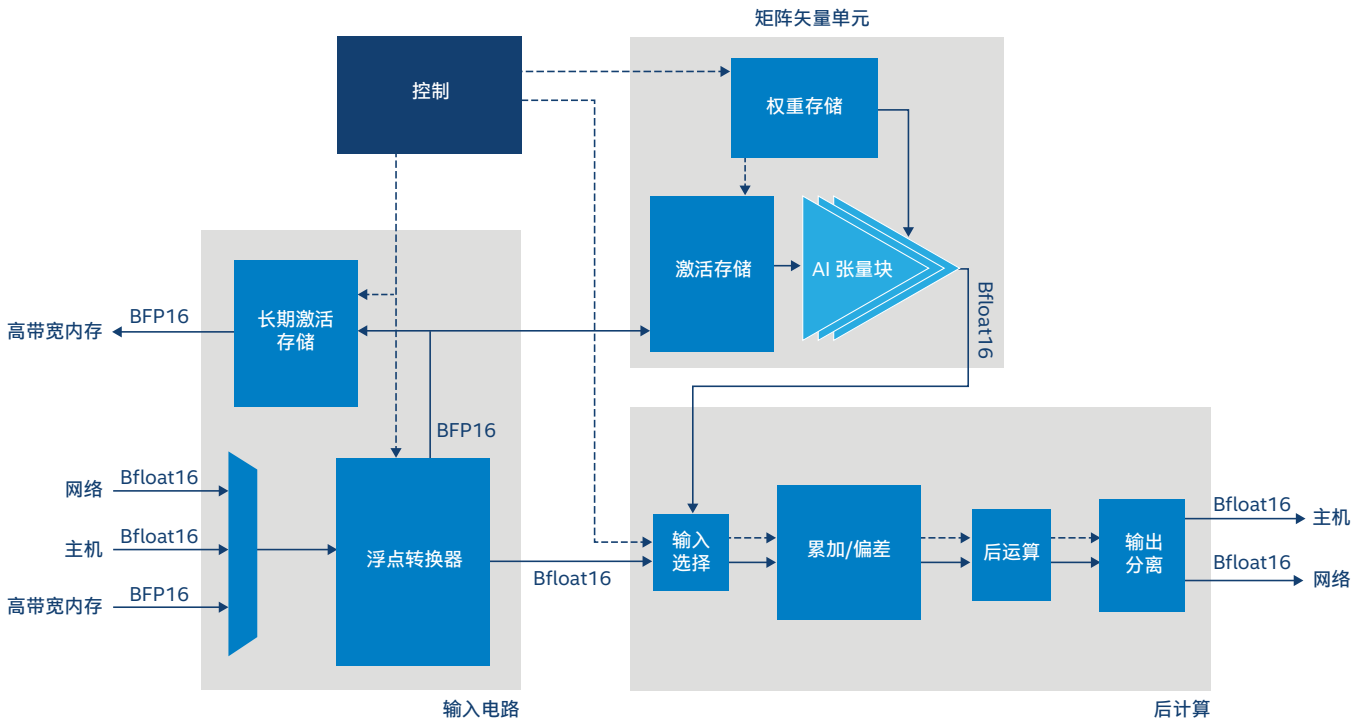

图 2. 英特尔 Stratix 10 NX FPGA 上的 MAU 内核架构。

IV. 硬件实施

我们在英特尔 Stratix 10 NX FPGA 上实施了 WaveNet 模型。除了输入阶段的 ConvTranspose1d 层之外，所有层均在 FPGA 上处理。预处理层不依赖任何之前生成的值，因此可以在核心自回归 WaveNet 循环之外执行。它的处理时间为 12.5 毫秒，低于自回归循环，而且不需要满足主自回归循环的延迟要求。我们在 CPU 上实施该层，这样可以使用 FPGA 专门处理网络的延迟关键型自回归部分。网络的其余部分在 FPGA 上运行，包括条件上采样层之后的 1x1 卷积层。虽然该层严格意义上并不是核心自回归循环的一部分，但是我们将该层放置在 FPGA 上，以利用为该层提供的计算能力，这极大提升了整个网络的计算能力。

FPGA 处理架构

我们使用 Myrtle.ai 可编程 MAU* 内核架构在 FPGA 上实施网络。MAU 内核是一个面向深度神经网络的可编程处理引擎，叠加在 FPGA 结构上，可提供运行时可配置且灵活的推理引擎。我们在 FPGA 上放置了 4 个针对英特尔 Stratix 10 NX FPGA 架构优化的 MAU 内核。我们在英特尔® Quartus® Prime 软件中设计 MAU 内核的平面布局，以实现较高的逻辑利用率，同时保持较高的时钟频率。在该设计中，FPGA 使用 240 MHz 的内核处理频率。

每个 MAU 内核均通过 PCIe 连接主机 CPU，并通过路由网络连接到专用高带宽内存接口（HBM），以及另一个 MAU 内核。每对相邻 MAU 内核之间的路由网络在每个时钟周期发送 120 个 bfloat16 值，内部总线带宽达到 60 GBps。

针对英特尔 Stratix 10 NX FPGA 优化的 MAU 内核架构如图 2 所示。该架构具有在每一层实施卷积所需的所有功能。除了卷积层，MAU 内核还具有 tanh、sigmoid、ReLU、门控激活、softmax 采样和独热嵌入操作等功能，这些功能构成了 MAU 内核的后计算单元。

使用英特尔 Stratix 10 NX FPGA AI 张量块

英特尔 Stratix 10 NX FPGA 具有创新的 AI 张量块，其 INT8 每秒万亿次运算数（TOPS）比英特尔 Stratix 10 FPGA 系列中的其他 FPGA 高出 15 倍。为了使用英特尔 Stratix 10 NX FPGA AI 张量块高效实施 1D 卷积层，我们将块配置为 “张量” 模式。以 BFP16 格式执行计算，每个包含 10 个矩阵元素的块均使用共享指数。每个英特尔 Stratix 10 NX FPGA AI 张量块可在每时钟周期计算块大小为 10 的 3 个点积。将 4 个 AI 张量块级联在一起，以形成宽度为 120 的点积。英特尔 Stratix 10 NX FPGA AI 张量块架构如图 3 所示。

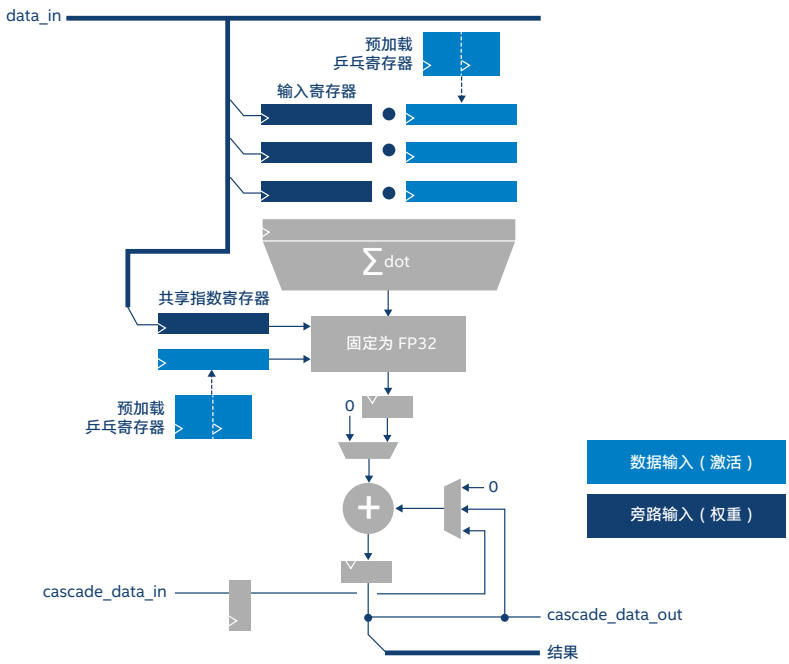

图 3. 英特尔 Stratix 10 NX FPGA AI 张量块。[2]

模型权重由旁路输入加载。AI 张量块加载新权重需要 18 个时钟周期，通过使用乒乓寄存器并行执行更新与计算。我们以 32 的批次大小来处理不同的通道，这样可以在每批次更新权重，发挥张量模式的最高效率。在每个 MAU 内核中都使用 480 个 AI 张量块。结果，每个 MAU 内核每时钟周期均可执行一个 120 x 120 矩阵向量乘法。这为加速器设计的 4 个内核提供了 27.6 TOPS 的总容量。

模型参数保存在本地片上 RAM 中。WaveNet 是一个相对较小的模型，因此有充足的可用片上内存，能够将所有网络参数保存在本地 AI 张量块中。设计为网络参数分配了 4.5 MB 本地存储和 1.3 Tbps 带宽。

在 FPGA 上实施扩张卷积

在推理过程中，我们使用快速 WaveNet 生成算法 [13]，将模型架构中的扩张卷积替换为常规卷积，并使用缓冲区来存储之前计算的值。然后，可以从缓冲区中选择正确的值来模拟扩张卷积。这样在生成每个新值时，便不需要从之前的样本中重新计算值，因为我们可以在缓冲区中轻松查找该值，从而极大提升了模型性能。

网络中的扩张卷积需要从先前的多达 128 个采样值中获取激活数据。这对中间参数存储提出了高带宽、低延迟的要求。在 WaveNet 网络内部，检索激活并馈送到扩张卷积所需的带宽为每 MAU 内核 3.1 GBps。由于需要存储来自先前采样值的多达 128 个激活，因此每个内核总共需要 10.3 MB 的存储空间。由于数据存储量太大，本地内核无法容纳，因此我们使用 HBM 内存来存储这些信息。

相比外部 DDR4 内存，人们更倾向于在片上使用 HBM，因为片上的 HBM 和外部 DDR4 内存分别可提供 256 GBps 和约 21 Gbps 带宽，并且后者还增加了在系统内进行数据传输的带宽开销。

编程模型

MAU 内核在运行时针对 WaveNet 模型进行编程，这支持将 FPGA 重复用于不同的模型架构，无需重新编译 FPGA 实施。

我们通过在每个内核上运行多个 WaveNet 层以及分时共享内核逻辑，将 WaveNet 映射到 4 个 MAU 内核。MAU 内核控制器负责执行内核逻辑。每个内核的批次大小为 32 个语音通道，在连续的时钟周期，在批次的每个通道上运行相同的指令。一旦完成该批次的所有层操作，激活将进入下一个 MAU 内核，以便后面的层进行处理。对 4 个内核进行管道化，这样每个内核将同时处理由 32 个语音通道组成的批次，每次都在 FPGA 中运行 128 个语音通道。

如需实时处理超过 128 个并发语音通道，应将语音通道处理成块（chunk），块就是梅尔声谱输入的一个整数。FPGA 能够以明显高于实时的速度生成样本，支持分时共享 FPGA 加速。FPGA 支持在非连续的操作模式下处理块，将所有活跃并发语音通道的中间状态保存在 HBM 内存中。

[2] 资料来源于英特尔可编程解决方案事业部 Graham McKenzie。

对于 WaveNet，我们将网络层映射到 MAU 内核，如表 2 所示。

内核数量	层映射
1	输入嵌入与 WaveNet 层 1-3
2	WaveNet 层 4-8
3	WaveNet 层 9-13
4	WaveNet 层 14-16 与输出卷积

表 2. 将 WaveNet 映射到 MAU 内核。

V. 结果

方法

我们使用二次采样为 16 kHz 的 LJSpeech 数据集 [14]。在开展项目前，从数据集中随机选择 100 个样本作为验证集，然后随机选择 100 个样本作为测试集。所有剩余的样本用作训练集。未应用数据增强。

WaveNet 架构参数的定义请参见第 2 节。使用混合精度训练对所有模型进行 140,000 步的训练，批次大小为 16，分布在 1–8 个 GPU。批处理中的一个元素包含从训练集的音频剪辑中随机选择的 1 秒钟音频，在必要时填充 0。使用 Adam 优化器 [15]，其固定学习速率为 10^{-3}，$\beta_1 = 0.9$，$\beta_2 = 0.999$，$\epsilon = 10^{-8}$。使用 QAT 时，我们使用 PyTorch [16] 实施 WaveNet，并使用 QPyTorch [17] 来模拟 BFP16。

我们报告了每个模型的 teacher-forced 交叉熵验证和测试损失以及平均意见评分（MOS）。对于每个模型，重复训练过程 3 次，并选择具有中位数验证损失的模型来计算与报告测试损失和测试 MOS 分数。MOS 的计算方法是让 30 位 Amazon Mechanical Turk 独立员工按照样本的自然程度，对生成的每个样本进行打分，共 5 个分数等级，见表 3。报告的 MOS 取这些分数的平均值，并使用 t-分布计算 95% 置信区间。

评级	标签	描述
1	非常差	完全不自然的语音
2	差	多半不自然的语音
3	中等	自然和不自然的语音各占一半
4	良好	多半自然的语音
5	优秀	完全自然的语音

表 3. 平均意见评分量表。

模型质量

表 4 列出了使用 PTQ 和 QAT 的基准 FP32 模型和 BFP16 模型的验证和测试集损失以及 MOS。我们发现 BFP16（PTQ）模型和 BFP16（QAT）模型均能够合成接近基准 FP32 模型的高保真音频。

模型	验证损失	测试损失	测试 MOS
人类	-	-	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 和 BFP16 WaveNet 模型的质量结果。

模型性能

表 5 展示了在英特尔 Stratix 10 NX FPGA 上运行的 WaveNet 模型的关键参数和处理性能。将该模型与在 V100 图形处理单元（GPU）上运行的 nv-wavenet 进行比较。截至撰写本文之时，该模型是我们能够参考的速度最快的 WaveNet 实施。

基于两个平台的热设计功耗（TDP）工具进行功耗比较，结果均高估了两个平台应用的真实功耗。英特尔 Stratix 10 NX FPGA 的 TDP 来自英特尔为基于 PCIe 的加速器卡部署提供的设计分析。

并发语音通道数量是 16 kHz 采样率下可以实时生成的最大通道数量。虽然 FPGA 实施使用的层尺寸要略小于 GPU 实施，但是具有更多模型参数和操作，因为模型的加速部分包含条件层，而 nv-wavenet 实施不包含该层。

相比 nv-wavenet，英特尔 Stratix 10 NX FPGA 实施的并发语音流数量增加了 8 倍。FPGA 能够将 WaveNet 模型的 TOPS 提高 8.6 倍。

FPGA 提供了一款更加节能的解决方案，将每瓦语音通道数量和 GOPS/W 分别提升 9.3 倍和 10 倍。

表 6 展示了 16 kHz 及以上音频频率部署中的并发语音通道数量。这表明即使在生成单个时步的延迟要求降低的情况下，FPGA 解决方案也能保持高效运行。

	MYRTLE.AI WAVENET	NV-WAVENET
平台	英特尔® Stratix® 10 NX FPGA	NVIDIA* V100 GPU
频率（MHz）	240	1530
数值精度	BFP16 / bfloat16	fp16
WaveNet 配置	r=120，s=256，L=16，a=256，D=8	r=128，s=256，L=16，a=256，D=8
每秒音频操作数（GOPS/秒）	65.03	60.36
模型参数（百万）	2.08	1.99
并发语音通道	256	32
应用 TOPS	16.6	1.93
TDP 功耗（W）	215	250
每瓦性能（GOPS/W）	77.4	7.7
每瓦语音通道数（1/W）	1.19	0.128

表 5. 16 kHz WaveNet 实施的性能结果。

音频频率	并发语音通道	
	MYRTLE.AI WAVENET	NV-WAVENET
16 kHz	256	32
24 kHz	160	16
32 kHz	128	8

表 6. 不同音频频率的 WaveNet 实施的性能结果。

我们在英特尔® 至强® 处理器双路 16 核 CPU（运行频率为 2.8 GH）上实施 ConvTranspose1d。每个输入步长都对应 200 个输出步长（stride），因此必须每 12.5 毫秒处理一个输入步长，才能生存 16 kHz 音频的输出步长。我们实施了自己的 ConvTranspose1d 变体，它的运行速度比 PyTorch 原生实施高一个数量级。批次大小为 64，在每个插槽上独立运行。我们测量得出每个批次的 99.999% 延迟为 3.85 毫秒，该速度足以支持在每个插槽上运行 3 个大小为 64 的批次，并满足 12.5 毫秒的处理要求。该配置支持 CPU 为多达 384 个并发语音通道计算 ConvTranspose1d，并且能够满足 FPGA 实施的要求，表明完整的 WaveNet 系统实施能够生成 256 个并发语音通道。

VI. 结论

本文展示了具有 256 个并发语音通道的先进 WaveNet 模型如何使用基于专用 FPGA 的加速器，实现实时性能并生成接近人类水平的合成语音。相比当前可用的最佳 GPU 解决方案，该模型可将性能提升 8 倍。

本文展示了在频率较高的部署中，FPGA 可发挥更大的优势。相比当前可用的最佳 GPU 解决方案，FPGA 可将 24 kHz 和 32 kHz 部署的优势分别扩大 10 倍和 16 倍。FPGA 可提供一个经济高效的平台，支持以更高的音频频率部署 WaveNet，并在一个加速器上生成 128 个 32 kHz 音频并发流。

本文展示了 FPGA 能够显著降低 WaveNet 实施的能耗（能耗只有 GPU 实施的 1/10），并帮助大规模部署实时语音合成的用户极大地节能降耗。

本文展示了可以在训练后应用 BFP16 格式，不同于 FP32，BFP16 可支持简单的量化流程，并将准确性的损失降至最低。这提供了来自机器学习框架的简单有效的量化流程和更高效的硬件实施优势。

参考资料

[1] A. v. d. Oord，S. Dieleman，H. Zen，K. Simonyan，O. Vinyals，A. Graves，N. Kalchbrenner，A. Senior 和 K. Kavukcuoglu，《Wavenet：面向原始音频的生成模型》，arXiv 预印本 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 等，《通过调节 WaveNet 对 Mel 谱图预测的自然 TTS 合成》，2018 年 IEEE 国际声学、语音和信号处理大会（ICASSP）。IEEE，2018 年，第 4779–4783 页。

[3] A. Oord，Y. Li，I. Babuschkin，K. Simonyan，O. Vinyals，K. Kavukcuoglu，G. Driessche，E. Lockhart，L. Cobo，F. Stimberg 等，《并行 wavenet：快速高保真语音合成》，国际机器学习会议，2018 年，第 3918–3926 页。

[4] N. Kalchbrenner，E. Elsen，K. Simonyan，S. Noury，N. Casagrande，E. Lockhart，F. Stimberg，A. v. d. Oord，S. Dieleman 和 K. Kavukcuoglu，《高效的神经音频合成》，arXiv 预印本 arXiv:1802.08435，2018 年。

[5] W. Ping，K. Peng 和 J. Chen，《Clarinet：端到端文本转语音中的并行波形生成》，arXiv 预印本 arXiv:1807.07281，2018 年。

[6] R. Prenger，R. Valle 和 B. Catanzaro，《Waveglow：基于流的语音合成生成网络》，ICASSP 2019-2019 年 IEEE 国际声学、语音和信号处理大会（ICASSP）。IEEE，2019 年，第 3617–3621 页。

[7] S. Kim，S.-g. Lee，J. Song，J. Kim 和 S. Yoon，《Flowavenet：面向原始音频的生成流》，arXiv 预印本 arXiv:1811.02155，2018 年。

[8] Q. Tian，Z. Zhang，H. Lu，L.-H. Chen 和 S. Liu，《Featherwave：一种高效的多频带线性预测高保真神经声码器》，arXiv 预印本 arXiv:2005.05551，2020 年。

[9] P.-c. Hsu 和 H.-y.Lee，《Wg-wavenet：无 GPU 的实时高保真语音合成》，arXiv 预印本 arXiv:2005.07412，2020 年。

[10] S. Hussain，M. Javaheripi，P. Neekhara，R. Kastner 和 F. Koushanfar，《Fastwave：在 FPGA 上加速自回归卷积神经网络》，arXiv 预印本 arXiv:2002.04971，2020 年。

[11] W. Ping，K. Peng，A. Gibiansky，S. O. Arik，A. Kannan，S. Narang，J. Raiman 和 J. Miller，《Deep voice 3：使用卷积序列学习扩展语音转文本》，arXiv 预印本 arXiv:1710.07654，2017 年。

[12] B. Jacob，S. Kligys，B. Chen，M. Zhu，M. Tang，A. Howard，H. Adam 和 D. Kalenichenko，《神经网络的量化和训练，以实现高效的整数运算推理》，IEEE 计算机视觉与模式识别会议记录，2018 年，第 2704–2713 页。

[13] T. L. Paine，P. Khorrami，S. Chang，Y. Zhang，P. Ramachandran，M. A. Hasegawa-Johnson 和 T. S. Huang，《快速 wavenet 生成算法》，arXiv 预印本 arXiv:1611.09482，2016 年。

[14] K. Ito 和 L. Johnson，《lj 语音数据集》，https://keithito.com/LJ-Speech-Dataset/，2017 年。

[15] D. P. Kingma 和 J. Ba，《Adam：一种随机优化方法》，arXiv 预印本 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 和 S. Chintala，《Pytorch：一种命令式风格的高性能深度学习库》，第 32 届神经信息处理系统会议纪要，H. Wallach，H. Larochelle，A. Beygelzimer，F. d'Alch´e-Buc，E. Fox 和 R. Garnett, Eds。Curran Associates, Inc.，2019 年，第 8024–8035 页。[在线] 下载地址：http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[17] T. Zhang，Z. Lin，G. Yang 和 C. De Sa，《Qpytorch：一种低精度算术模拟框架》，arXiv 预印本 arXiv:1910.04540，2019 年。