HEP-BNN：一种用于在异构多处理器平台上寻找二值神经网络低延迟执行配置的框架

Leonard David Bereholschi、Ching-Chi Lin、Mikail Yayla、Jian-Jia Chen 多特蒙德工业大学，德国{leonard.bereholschi, chingchi.lin,mikail.yayla, jian-jia.chen}@tu-dortmund.de

arXiv:2301.05126v1 [cs.DC] 12 Jan 2023

摘要—与全精度神经网络相比，二值化神经网络通过使用二值化权重和激活值，大幅降低了计算和内存需求。在由中央处理器和图形处理器组成的异构多处理器平台上，将二值神经网络的一层部署到不同设备上，会影响推理性能，即准确率和延迟。通常，可使用由中央处理器和图形处理器组成的异构硬件平台来执行二值神经网络工作负载。然而，要有效利用异构硬件平台，必须找到高效的二值神经网络工作负载映射策略。在本研究中，我们提出了一种框架，可为由中央处理器和支持 CUDA 的图形处理器组成的执行平台生成高效的二值神经网络层到设备映射（即为模型的每一层选择合适的并行配置）。我们在三个特性各异的硬件平台上，使用两个知名数据集——*Fashion-MNIST* 和 *CIFAR-10*，对所提出的框架在两种二值神经网络架构上的表现进行了评估。结果表明，与运行完全并行化的图形处理器实现相比，在所测试的硬件平台上，我们的框架生成的高效配置分别可快达 2×、 2.6× 和 11.8× 倍。

索引词—**二值神经网络，推理，图形处理器，CUDA**

I. 引言

过去十年间，神经网络（NN）已应用于各种实际领域，例如计算机视觉中的图像识别、化学领域中的化学模式预测，以及医学领域中的癌症检测。 [1] 给定一个经过充分训练的神经网络模型，推理就是使用该模型对此前未见过的数据进行预测的过程。根据神经网络模型的结构，推理可能会耗费大量时间和资源。

二值神经网络（BNN） [2] 是神经网络的一种资源高效型变体。在二值神经网络中，权重和激活值均采用 1 位表示进行二值化。二值神经网络中的乘法和累加运算分别可以使用同或非运算符和人口计数函数来计算。因此，与全精度神经网络相比，二值神经网络的资源效率显著更高，使其成为在资源受限的边缘设备上运行人工智能应用的理想选择。

对于二值神经网络工作负载的执行，定制硬件加速器（即基于 FPGA 或专用集成电路的加速器）仍处于开发初期 [3], ，而中央处理器和图形处理器已相当成熟且易于获取。近期有多项研究评估了使用图形处理器运行二值神经网络的方案。Hubara 等人 [2] 使用面向图形处理器的同或非 内核来评估二值神经网络。Xu 等人 [4] 实现了一个二值神经网络计算内核。Li 等人 [5] 在 Turing GPU 上运行二值神经网络，重点关注位级并行和内存步幅。Chen 等人 [6] 为移动电话开发了一种二值神经网络加速引擎。

尽管图形处理器具有强大的计算能力，但在图形处理器上执行二值神经网络的每一层并不一定能带来良好的性能。本文揭示，在图形处理器上独占执行二值神经网络的所有层，与在中央处理器上顺序运行模型相比，会显著增加延迟（例如，模型过小且 CPU 开销过大，见图 1）。因此，要在由中央处理器和图形处理器组成的异构多处理器平台上实现高效的二值神经网络推理，恰当的层到设备映射至关重要。

贡献：本文提出了一个框架*HEP-BNN*，可针对在由中央处理器和图形处理器组成的异构多处理器平台上运行的给定二值神经网络模型，自动生成高效的层到设备映射（即每层适用的并行配置）。对于给定的训练好的 BNN 模型，*HEP-BNN* 会系统地评估该模型在不同并行配置下分别于中央处理器和图形处理器上的执行时间，并将执行时间最短的配置标示为目标平台上该二值神经网络的高效 CPU/GPU 配置。这样的框架有助于更有效地利用可用硬件资源。此外，自动生成包含优化映射且可直接应用的代码，能够实现高效的二值神经网络推理。这使研究人员和从业者都能充分发挥现有硬件平台的能力，在资源受限的边缘设备上高效地应用二值神经网络。

我们的贡献总结如下：

- 我们提出了*HEP-BNN* 框架，可针对给定的二值神经网络模型和由中央处理器与支持 CUDA 的图形处理器组成的异构执行平台，自动生成高效的层到设备映射。生成的代码包含模型各层的高效配置，可用于实际采用二值神经网络推理的应用。该框架已发布在 GitHub 上[1]。

[1]https://github.com/LeonardDavid/hep-bnn

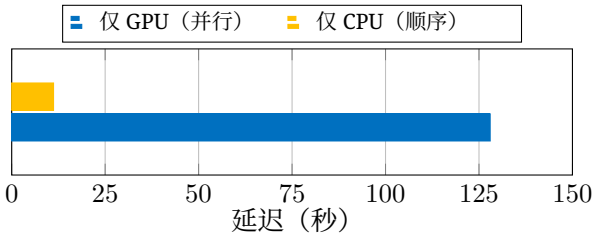


图 1. Jetson TX2 上的 Fashion-MNIST：我们结果中的一个示例，展示了顺序 CPU 模型与并行 GPU 模型的总延迟差异（后者的 CPU 开销更高）。

- 为展示该框架的能力，我们将其应用于两个二值神经网络模型，并使用两个常见数据集*Fashion-MNIST* 和 *CIFAR10*，在具有不同特性的三个硬件平台上进行测试：服务器、笔记本电脑和*Jetson TX2*。对于不同的二值神经网络模型，结果表明，通过为图形处理器上运行的层选择恰当的并行配置，三个执行平台的推理时间分别最多可缩短 2×、 2.6× 和11.8×。

II. 系统模型

第二节 A 小节介绍了二值神经网络的基础知识以及我们模型中使用的层。第二节 B 小节介绍了 GPU 计算，并描述了 CUDA 框架，包括其特性、局限性和一些实现细节。第二节 C 小节介绍了我们框架中实现和使用的不同并行配置，以及本文通篇使用的相应记号。最后，第二节 D 小节阐述了问题定义。

A. 二值化神经网络基础

二值化神经网络（BNN） [2] 是一种节省资源的神经网络变体。在二值神经网络模型中，权重和激活值被二值化为 1 位表示。与全精度神经网络不同，后者必须通过一次矩阵乘法来计算每个神经元的输出；而在二值神经网络中，只需应用 $xnor$ 运算符即可计算神经元的输出。具体而言，一层的输出可计算为

$$2 * popcount(xnor(W_i^l, I^{l-1})) - \#bits > T,$$

其中， W_i^l 是第 l层的权重， I^{l-1} 是第 l层的输入， $popcount()$ 是一个用于统计 $xnor$ 运算结果中 1 的个数的函数，$\#bits$ 是 $xnor$ 运算结果中的位数，而 T 是一个可学习的阈值参数，可根据批归一化参数计算得出。二值输出取决于该语句的真值，该语句表示一个移位的二值化函数 [7]。

在本研究中，我们考虑卷积二值化神经网络。卷积二值化神经网络包含四种基本层类型，即 卷积层、最大池化层、步进层和全连接层，如图 2 所示。本研究中的二值神经网络还采用了展平层。

该卷积层计算输入与滤波器的二维卷积。我们用“Cχ”表示卷积层，其中 χ 表示该层神经元的数量。例如，“C64”是一个包含 64 个神经元的卷积层。在本研究中，所有卷积层的滤波器尺寸均固定为 3×3 。

最大池化层通过在给定窗口大小内选取输入的最大值来对输入进行下采样，我们的模型中窗口大小设为 2×2 。我们用“MPχ”表示最大池化层，其中 χ 表示其输出大小，例如，输出大小为“16×16”时记为“MP16”。

步进层先执行批归一化，再执行二值激活函数。神经网络模型中使用批归一化 [7] ，以加快并稳定训练，从而提高准确率。请注意，即使在二值神经网络中，批归一化的阈值仍为有符号整数。在我们的模型中，我们采用硬双曲正切函数作为二值激活函数。对于二值神经网络的推理，可通过二值阈值化 [7]来计算批归一化和后续的激活。本文其余部分将“步进”层记为“S”。

一个全连接层会将当前层中的每个神经元与下一层中的每个神经元相连。我们将全连接层记作“FCχ”，其中 χ 表示该层中神经元的数量。请注意，全连接层也将二值权重作为可学习参数。

我们用“FLAT”表示展平层。该层将高维矩阵重新排列为低维矩阵，例如，在我们的模型中，将三维矩阵转换为一维数组。重新排列可通过在 CPU 的 C++ 代码中执行一条简单的语句来完成。

B. 图形处理器

由于其架构特性，图形处理器适用于高度并行的应用场景，例如图形密集型任务中的重复矩阵乘法。大多数先进的图形处理器都拥有大量计算核心，并配备高效的大容量、高速内存。在图形处理器上，计算由线程并行执行。要编程控制线程的行为，需要使用专用的 GPU 代码，例如 CUDA 或 OpenCL 等框架。在我们的工作中，我们使用 CUDA 编程语言，将二值神经网络模型的计算工作负载部署到英伟达显卡上。

在 CUDA 程序中，线程可以组织成线程块，每个线程块最多可并行执行 1024 个线程。线程块还可以进一步组织成三维网格结构，从而提升图形处理器上的并行度。网格在不同维度轴上的大小限制如下：$\{x, y, z\} \rightarrow \{2^{31}-1, 65536, 65536\}$。此外，使用 *threadIDx*和*blockIDx*等内部 CUDA变量，可以让每个线程访问与其特定计算相关的数组中的单个值。

在 CUDA 编程语言中，在图形处理器上执行计算任务的函数称为内核。启动*CUDA* 内核之前，需要在显存中分配内存，然后将计算所需的所有数据从主机（中央处理器）传输到设备（图形处理器）。启动内核 之前，还需在遵循前述限制的前提下指定线程块和网格的大小

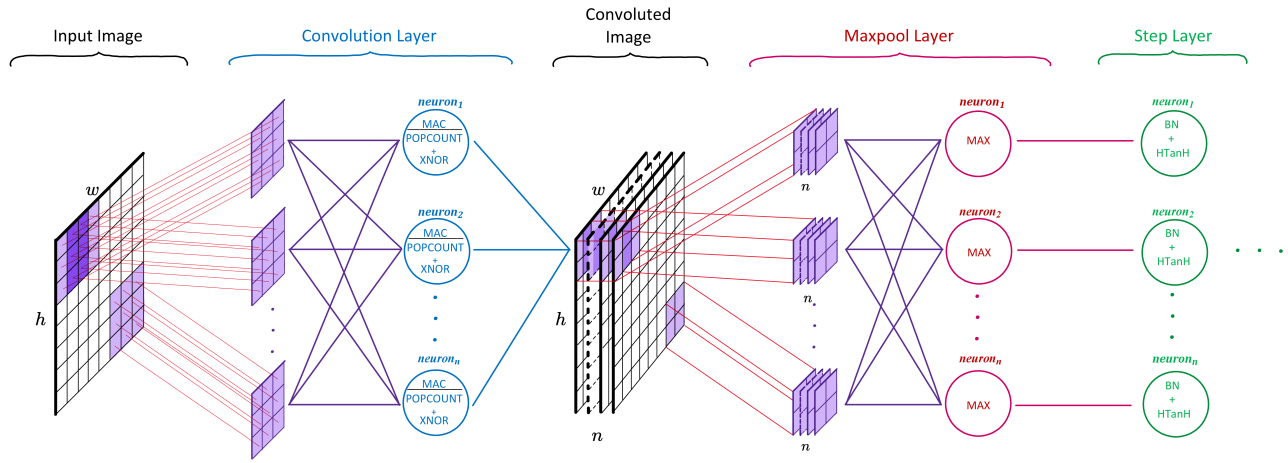


图 2. 卷积二值神经网络模型的结构，展示了三种主要的层类型：卷积、最大池化和阶跃层。

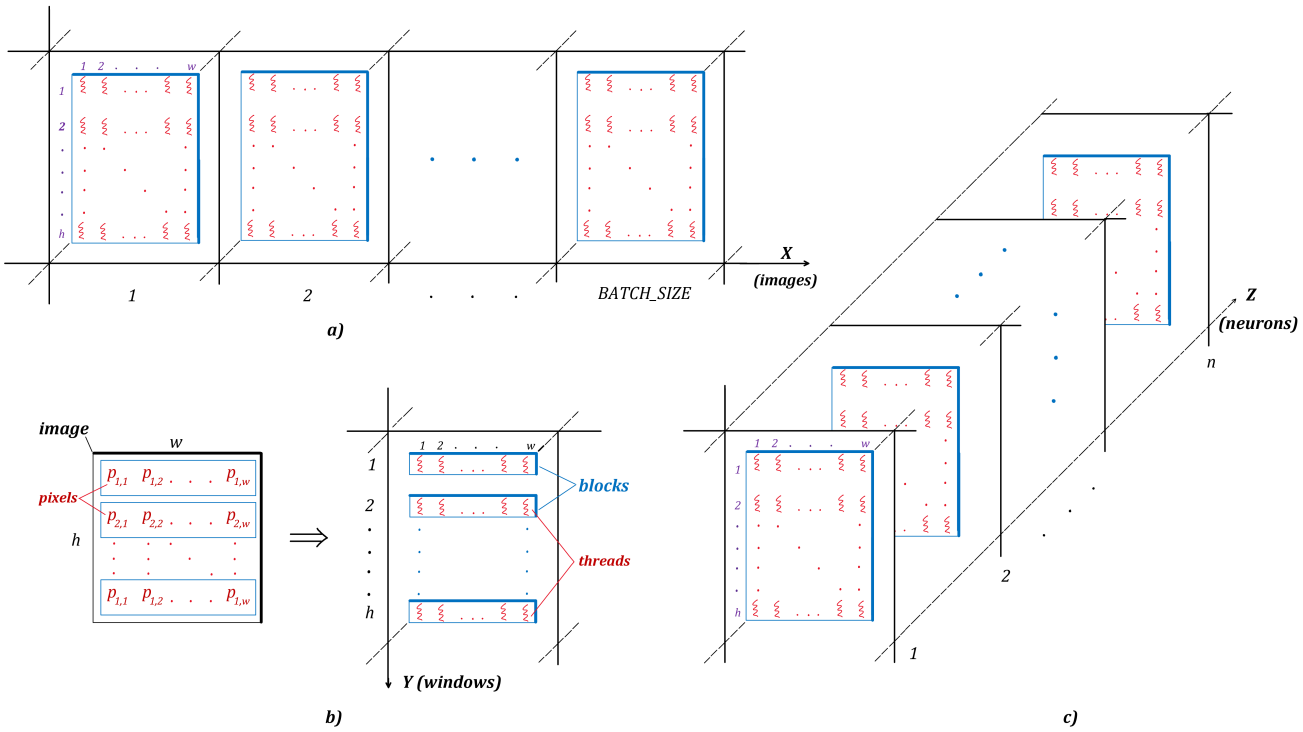


图 3. 并行性的概念：a）基于数据，b）基于窗口，以及 c）基于神经元

。图形处理器完成内核中的所有任务后，结果会从设备复制回主机，并释放先前在显存中分配的内存。

与中央处理器相比，图形处理器能够提供强大的计算能力，因此常被用作许多场景中的加速器，但在图形处理器上运行应用程序并不总能提升性能。在某些情况下，由于通信时间开销等因素，在图形处理器上运行并行代码可能比运行顺序中央处理器代码耗时更长。因此，分析应用程序的特性（例如并行度），有助于判断是否适合在图形处理器上运行该应用程序。对于能够通过图形处理器加速的应用程序，如何组织工作负载以实现最佳性能，是亟待解决的关键问题。

C. 数据并行性方面

二值神经网络模型的工作负载由多个作为输入的数据图像构成。我们将批大小 定义为一个批次中并发处理的数据图像数量。为了并行处理数据集上的二值神经网络推理工作负载，我们从数据并行的三个方面来组织工作负载：

1）基于数据：批次中的每个数据图像都并发进行推理。2）基于窗口：将一幅数据图像划分为由连续像素组成的卷积窗口，并发处理这些窗口。3）基于神经元：并发计算神经网络模型同一层中各神经元的输出。

在数据（X）配置中，批次中的多个数据图像在图形处理器上并发推理，如图 3（a）所示。批次中的每个数据图像都被分配给一个

图形处理器线程块。如果一个线程块被分配了多个数据图像，则这些图像会依次处理。数据图像中的每个像素（及其后续操作）都由线程块中的一个线程处理。

图 3 (b) 展示了窗口 *(Y)*配置的基本思想：数据图像按行划分为由连续像素组成的窗口，并将每个窗口分配给一个图形处理器线程块。同一窗口中像素（线程）相关的工作负载由图形处理器并发处理。

对于神经元 *(Z)* 配置，如图 3 (c) 所示，同一层中神经元的输出会并发计算。神经元的输出是其前驱节点的加权和经过激活函数处理后的结果。每层中的每个神经元都分配给一个图形处理器线程块，线程块中的线程以对应的前一层输出为输入，并计算该神经元的输出。

基于这些方面，我们考虑以下七种并行配置及其记法，并将在本文中始终使用：1）数据 *(X)*，2）窗口 *(Y)*，3）神经元 *(Z)*，4）数据 + 窗口 *(XY)*，5）数据 + 神经元 *(XZ)*，6）窗口 + 神经元 *(YZ)*，以及 7）数据 + 窗口 + 神经元 *(XYZ)*。由多个方面组成的配置，会同时按照各个单独方面的所有实现方式进行实现。请注意，对于所有并行配置，线程块中的线程会根据层执行相同的操作，例如，对卷积层中的像素进行卷积。*blockIDx* 和*threadIDx* 变量决定每个线程负责哪些像素和/或神经元。

D. 问题定义

给定一个训练良好的二值神经网络（BNN）模型，我们旨在借助图形处理器降低该模型的推理时间。然而，在图形处理器上并行执行计算工作负载涉及多个方面。二值神经网络模型中的每一层都可能适合不同的并行配置。尽管如此，由于开销（例如主机与图形处理器设备之间的数据迁移所造成的开销），某些层的工作负载在图形处理器上运行可能并无益处。因此，我们的目标是为给定的二值神经网络模型生成高效的层到设备映射，以最大限度地缩短推理时间。对于映射到图形处理器上执行的层，我们还会确定其适合的并行配置。

III. F 框架介绍

本节将详细介绍所提出的框架。我们的*HEP-BNN* 框架的操作步骤概述见第 III-A 节。映射算法见第 III-B 节。框架的文件夹结构、重要脚本文件和生成文件的相关信息分别详述于第 III-C、III-D 和 III-E 节。

A. 我们的HEP-BNN 框架概述

我们框架执行的操作步骤如图 4 所示。首先，程序接收一个二值神经网络

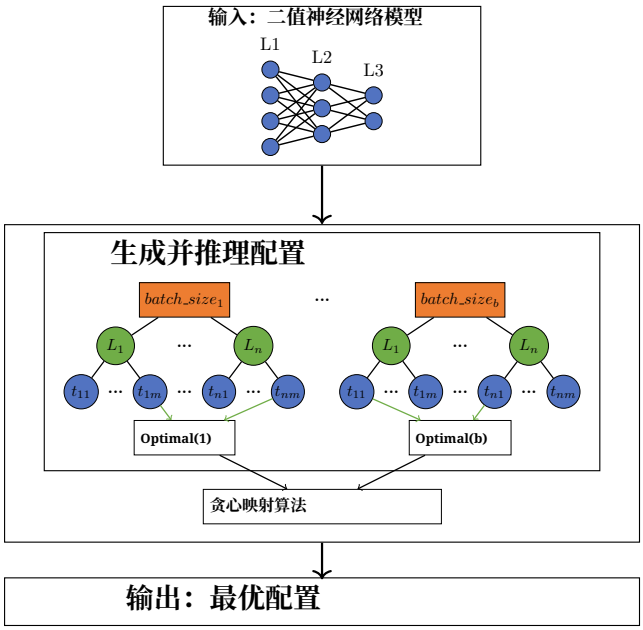


图 4. 我们的*HEP-BNN* 框架的运行步骤。

模型作为输入，该模型采用 ONNX 格式，且已在特定数据集（例如 Fashion-MNIST、CIFAR10）上完成训练。然后，对于给定范围内的每个批大小，都会生成适用于中央处理器和图形处理器的 C++ 及 CUDA 代码。使用不同配置实现每一层（L_1，…，L_n）后，系统会针对应用的每种配置运行模型推理，从而得到不同的运行时间（ t_{11}，…，t_{1m} $\in L_1$ 和 t_{n1}，…，$t_{nm} \in L_n$）。根据第 III-B 节所述的算法 1，这些计时信息将用于以贪心方式选择高效配置。

B. 映射算法

为确定能够实现最低推理时间的合适并行配置，本文采用以下层到设备映射算法（见算法 1）。该算法使用不同的批大小，分别在中央处理器和图形处理器上对二值神经网络模型的每一层进行性能分析。在图形处理器上，实现了第二节 C 小节中的每种并行配置。总计，每层针对 8 种不同的实现进行了性能分析：1）中央处理器，2）数据 *(X)*，3）窗口 *(Y)*，4）神经元 *(Z)*，5）数据 + 窗口 *(XY)*，6）数据 + 神经元 *(XZ)*，7）窗口 + 神经元 *(YZ)*，以及 8）数据 + 窗口 + 神经元 *(XYZ)*。

对于经过性能分析的层，将实现最低推理时间的实现映射到指定的批大小。将每个层的最低推理时间相加，即可得到使用适用于指定批大小的高效实现执行二值神经网络模型的总运行时间。

对每个层进行性能分析后，搜索**最小**总运行时间，以及达到该时间所需的合适批大小。最后，使用合适批大小获取二值神经网络模型中每个层的映射实现。由此生成高效并行配置，实现预期推理时间最低。算法 1 给出了所述映射算法的伪代码。

数据：二值神经网络模型 结果：高效配置

1 $proper_{batch\ size} \leftarrow 0$ **2** $result\ time \leftarrow$ **最大整数_3 对每个** $batch\text{-}size$ **执行**4 $sum_{min\ time} \leftarrow 0$ **5 对每个** $layer$ **执行**6 $min\ time \leftarrow$ **最大整数_7 对每个** $implem$ **执行 8 使用** $implem$ **实现** $layer$ **9（中央处理器**$_{time}$**，图形处理器**$_{time}$**）** $\leftarrow$ **对已实现的**$layer$ **(**$batch\ size$**)进行性能分析**

10 $inference\ time \leftarrow$ **中央处理器**$_{time}$ + **图形处理器**$_{time}$ **11 如果** $inference\ time < min\ time$ **则**

12 $min\ time \leftarrow inference\ time$ **13 将** $implem$**(**$layer$**) 映射到** $batch\ size$ **14 结束 15 结束**

16 $sum_{min\ time} \leftarrow sum_{min\ time} + min\ time$ **17 结束 18 如果** $sum_{min\ time} < result\ time$ **则**

19 $result\ time \leftarrow sum_{min\ time}$

20 $proper_{batch\ size} \leftarrow batch\text{-}size$ **21 结束 22 结束 23 对每个** $layer$ **执行 24 从映射中获取** $implem$ **(**$layer$**)**[$proper_{batch\ size}$]**25 将** $layer_{implem}$ **添加到高效配置 26 结束 27 返回高效配置**

算法 1： 用于确定二值神经网络模型高效配置的映射算法，该配置可实现最低推理时间（注意：*implem* 是 *implementation*的缩写）

C. 文件夹结构

我们的*HEP-BNN* 框架使用 Python 来运行输入模型的实现和优化。为了展示我们的实现，我们采用开源机器学习编译器和代码生成器 Fastinference [8]。具体而言，分别用于生成中央处理器和图形处理器的 C++ 和 CUDA 代码时，会使用模板语言*Jinja2* 。本节将概述文件夹树结构中最重要的部分（*'fastinference/'*）。

每个模型、优化器和实现都在*'fastinference/'*中的单独文件夹内定义。它们进一步按受支持的算法类型划分，例如*'ensemble/'*、*'tree/'*、*'neuralnet/'*。具体而言，在*'fastinference/implemen-tations/neuralnet'* 中，为不同的目标硬件设置了相应的文件夹：*'cpp/'*、*'fpga/'*、*'iree/'*。

我们的工作主体位于*'cuda/'* 文件夹中，其中为每种并行配置都设置了一个独立目录。文件夹名称遵循第二节 C 小节中介绍的记法。

每个文件夹中都有*Jinja2* 文件，其中主要包含用于模型在图形处理器上并行运行的各层 CUDA代码模板。此外，还有一个*'cpu/'* 文件夹，其中包含用于模型在中央处理器上顺序运行的 C++ 模板，供顺序实现使用。每个文件夹中的*'implement.py'* 文件都包含*'to implementation()'* 函数，负责为模型的每一层选择合适的模板文件，并生成实现所需的 C++和 CUDA 文件。

此外，*'automatic/'* 文件夹包含运行映射算法的代码，该算法会自动从相应路径读取模型和数据，生成并推理所有选定的配置，并以贪心方式映射合适的配置。在以下章节中，我们以 CIFAR10 数据集为例，更详细地介绍某些文件的用法。

D. 重要脚本文件

在***test_cuda.py***中，指定了可用并行配置的实现。其标记法与第二节 C 小节中所述的一致。随后，通过调用***test_utils.py***中定义的*HEP-BNN* 框架的*'to implementation()'* 函数来启动该框架。

这里为批大小设置了上限和下限，并以 2 的幂表示（例如，若 'b l = 0' 和 'b u = 4'，则使用的批大小为 $2^0 = 1$、$2^1 = 2$、$2^2 = 4$ 和 $2^3 = 8$）。

随后在嵌套的 for 循环中运行映射算法：一个循环遍历每种配置，另一个循环遍历每个批大小。该算法包含两个重要函数，*'prepare_fastinference()'* 和 *'run_experiment()'*。前者会生成模型编译和运行所需的所有文件（更多详情见第 III-E 节），后者则编译并运行生成的模型，然后输出并存储推理结果。

运行所有生成的模型后，根据最低推理时间为每一层选定最佳配置（见算法 1，第 23–26 行）。最后，生成并推理高效配置，其推理时间低于所有其他组合。

二值神经网络模型以 ONNX 格式存储在以下路径中：

fastinference/实现/神经网络/cuda/自动化/模型/cifar10/模型_cifar10.onnx

测试数据存储在：

fastinference/实现/ 神经网络/cuda/自动化/数据/ cifar10/testing.csv

HEP-BNN 通过在根文件夹（*'fastinference/'*）中运行以下命令来**启动**：

fastinference/implementations/neuralnet/cuda/automatic/test_cuda.py --outpath tmp/fastinference/cuda_auto --数据集 cifar

表 I CIFAR-10 二值神经网络模型的结构。

CIFAR-10 二值神经网络模型结构
在 → C64 → S → C64 → MP16 → S → C256 → S → C256 → 多处理器系统8 → 处理器缓存 → 缓存512 → 多处理器 →系统512 → 多处理器4 → 系统 → 展平层 → 全连接层1024 → 全连接层 →

表 II
FashionMNIST 二值神经网络模型的结构。

FashionMNIST 二值神经网络模型结构
在 → C64 → MP14 → S → C64 → MP7 → S → 展平层 → FC2048 → S → FC2048 → 10

E. 生成文件

本节列出了每种配置对应的**生成文件**，并附有简要说明。

- *'utils.h'* 和*'utils.cuh'*：包含实用函数（分别用于 C++ 和 CUDA 代码）
- *'cuda kernel.h'* 和 *'cuda model.h'*：这些头文件包含将 C++代码与 CUDA 代码连接起来的函数
- *'modelW.hpp'*：包含每一层输出数组的声明，并存储权重、偏置和阈值
- *'model.h'* 和 *'model.cpp'*：根据配置，包含用于中央处理器的顺序 C++ 模型，或用于图形处理器的并行 CUDA 模型调用（两种实现都存在，但未使用的部分会被注释掉，以便调试和比较）
- *'model.cu'*：用于图形处理器的 CUDA 代码（如适用）

此外，还有一些文件无论配置如何都相同。这些文件已经编写完成，并从*'../cuda/automatic/'* 文件夹**复制** 到每个生成的实现中：

- *'main.cpp'*：将数据集拆分为多个批次，调用模型预测器，并计算准确率和延迟
- *'CMakeLists.txt'*：生成用于编译整个项目的 'Makefile'

请注意，修改任何 *Jinja2* 模板或 'implement.py'文件后，必须先运行以下命令*'pythonsetup.py install'*，然后才能调用 *'make'* 命令。

IV. 实验设置与结果评估

本节将介绍实验结果。第 IV-A 节介绍硬件、数据集和模型的相关信息。第 IV-B 节展示并比较了我们的 *HEP-BNN* 框架的结果，即为每个层实现适当的配置，并将其与基线顺序实现及其他并行配置进行比较。

A. 性能分析环境

*1)*硬件：我们在一台基于 Linux 的服务器上进行实验，该服务器配备了 Intel Core i7-8700K 中央处理器和一块具有 8GB 显存的 GTX1080 图形处理器。此外，我们还在配备消费级图形处理器（GTX 1650Ti）的 Windows 系统以及嵌入式 Jetson TX2 板卡上运行相同的实验。表 III 还列出了每个图形处理器可用的 CUDA 核心数量。

表 III 评估所用硬件概览

Name	CPU	GPU	CUDA 核心
服务器	i7-8700K	GTX1080	2560
笔记本电脑	i7-10750H	GTX1650Ti	1024
Jetson TX2	Cortex-A57	基于 Pascal	256

我们在内核启动外部封装 cudaEventRecord() 函数，以便准确测量 GPU 时间。CUDA 代码中的通信成本（即主机与设备之间的内存分配和传输）由中央处理器在内核启动之前执行，并计入 CPU 开销时间。我们为模型实现了相互独立的层，因此中央处理器与图形处理器之间的数据传输会在每层执行之前和之后进行（即使连续的两层都在图形处理器上执行）。代码生成器可以在未来的工作/实现中进行调整，以考虑这种情况。

*2)*数据集：该算法在两个常用的基准测试数据集 FashionMNIST 和 CIFAR-10 上进行评估。FashionMNIST [9] 数据集由 70，000 张灰度图像及其标签组成，涵盖 10 个类别，代表不同的服装。每张图像的尺寸为28×28 像素，包含 1 个通道，其中 0 表示最亮值，255 表示最暗值。在全部图像中，60，000 张用于训练，其余 10，000 张用于测试。CIFAR-10 [10] 数据集包含 60，000 张彩色图像（3 个通道），每张图像的尺寸为 32×32，共 1024 个像素。该数据集分为 50，000 张训练图像和 10，000 张测试图像，分属 10 个不同类别，代表交通工具（即飞机、船、卡车、汽车）和动物（即鸟、猫、狗、鹿、青蛙、马）。

3) 二值神经网络架构：用于推断数据集的二值神经网络模型采用了适用于神经网络二值化变体的 VGG 型架构 [11]。FashionMNIST 网络模型共包含 10 层，每层都属于第 II-A 节所述的类型之一。具体而言，1^{st} 和4^{th} 层是卷积层，尺寸分别为 $28 \times 28 \times 64$和 $14 \times 14 \times 64$。卷积层在紧随其后的最大池化层（即 2^{nd} 和 5^{th} 层）中下采样至输入尺寸的一半。3^{rd}、6^{th}和 9^{th} 层采用步进层，用于执行批归一化和激活函数运算。输入图像经过卷积和下采样后，在第 7 层展平为一维数组。最后，8^{th} 和 10^{th} 层全连接了共 2048 个神经元。

网络结构列于表 II 和表 I 中，所用符号与第 II-A 节中的一致。CIFAR-10 模型还包含二值神经网络的标准层类型，共计 19 层。卷积层位于第 1^{st}、3^{rd}、6^{th}、8^{th}、11^{th} 和 13^{th} 层。仅在第 4^{th}、9^{th} 和 14^{th} 层进行三次最大池化下采样。第 16^{th} 层将图像展平，以供位于第 17 层和第 19 层的全连接层使用。其余层均为步进层。

为模拟推理过程，分别使用两个模型的测试图像集

表 IV CIFAR10 模型的高效配置

	C64	S	C64	MP16	S	C256	S	C256	MP8	S	C512	S	C512	MP4	S	FLAT	FC1024	S	FC1024
服务器	**CPU**	CPU	**Z**	**CPU**	CPU	**XZ**	CPU	**XYZ**	**XY**	CPU	**XZ**	CPU	**XZ**	**X**	CPU	CPU	**X**	CPU	**CPU**
笔记本电脑	**CPU**	CPU	**Y**	**CPU**	CPU	**XYZ**	CPU	**XYZ**	**CPU**	CPU	**XYZ**	CPU	**XYZ**	**CPU**	CPU	CPU	**X**	CPU	**CPU**
TX2	**CPU**	CPU	**XYZ**	**CPU**	CPU	**Z**	CPU	**Z**	**CPU**	CPU	**XZ**	CPU	**XZ**	**CPU**	CPU	CPU	**XY**	CPU	**CPU**

表五 Fashion-MNIST 模型的高效配置

	C64	MP14	S	C64	MP7	S	FLAT	全连接层2048	S	全连接层2048
服务器	**CPU**	**CPU**	CPU	**XZ**	**X**	CPU	CPU	**CPU**	CPU	**CPU**
笔记本电脑	**CPU**	**CPU**	CPU	**CPU**	**CPU**	CPU	CPU	**CPU**	CPU	**CPU**
TX2	**CPU**	**CPU**	CPU	**XZ**	**CPU**	CPU	CPU	**CPU**	CPU	**CPU**

。这为分析二值神经网络模型各层的算法提供了受控的基准测试环境。权重和偏置经过 100 轮训练，FashionMNIST 模型的推理准确率达到 77。24%，CIFAR-10 模型的推理准确率达到 67。08%。

初步观察表明，批大小会影响某些配置的运行时间。因此，实验还对两个二值神经网络模型采用了不同的批大小，从 $\{1, 2, ..., 128\}$ 开始，按 2 的幂次递增。

B. 结果

表VI列出了在每种经过测试的目标硬件上运行二值神经网络模型（每个模型均以 10000 张数据图像作为输入）所测得的推理时间。延迟（以秒为单位显示）是二值神经网络处理**整个** 10000 张图像测试数据集所需的时间。请回想第 III-B 节，合适的配置是在所有不同批大小中选出的，因此务必同时注意批大小和运行时间。具体而言，批大小为 n 意味着并行处理 n 张数据图像。

可以看出，在三种经过测试的硬件中，服务器拥有最多的 CUDA 核心，因此总体运行时间更短。另一方面，资源受限的 TX2 拥有最少的 CUDA 核心，运行时间明显更长。

CIFAR-10 和*FashionMNIST* 模型各层的映射配置分别列于表IV和表V。从中可以观察到以下几点：

- 由于 Fashion-MNIST 二值神经网络模型是两个模型中较小的一个，因此几乎所有层都映射到中央处理器上按顺序执行。一个值得注意的例外是第二个卷积层，它在服务器和 TX2 上均映射到*XZ*配置。
- 对于 CIFAR10 二值神经网络模型，我们观察到，几乎在所有情况下，最大池化层都映射到中央处理器，而卷积层和全连接层则使用不同的并行配置映射到图形处理器。例如，第二个卷积层在服务器上映射到*Z* 配置，在笔记本电脑上映射到*Y*，在 TX2 上映射到*XYZ* 。

最后，图 5 将纯串行的中央处理器实现和两种直观的二值神经网络模型并行化方法，与映射算法的结果进行了比较。朴素 GPU 实现考虑使用仅包含<style id='1'>数据 (X) </style>配置的方式，对

表VI 高效配置的最短推理时间

数据集	Fashion-MNIST			CIFAR10		
硬件	服务器	笔记本电脑	TX2	服务器	笔记本电脑	TX2
运行时间	2.11秒	2.84秒	9.31秒	41.6秒	55s	297s
批大小	16	2	64	16	128	16

每个适合 GPU 加速的层进行并行化。全并行 GPU 实现则尽可能地并行化每个适合的层，即采用数据 + 窗口 + 神经元 *(XYZ)*配置。这是两种最直观的工作负载并行化方式，但没有考虑到某些层可能无法从 GPU 加速中受益这一事实。

对每种批大小运行高效配置所得的结果显示，整体速度显著提升。具体而言，与完全并行实现相比，在服务器上运行 *HEP-BNN* 框架的速度最高可提升 $2\times$；在 Jetson TX2 上最高可提升 $2.6\times$；而在笔记本电脑系统上，高效配置则带来 $11.8\times$ 的性能提升。

V. 结论

我们提出了一种框架，可为由中央处理器和支持 CUDA 的图形处理器组成的异构多处理器平台生成高效的 BNN 层到设备映射。给定一个训练好的 BNN 模型，我们提出的 *HEP-BNN* 框架会系统地评估该模型在中央处理器和图形处理器上采用不同并行配置时的执行时间。我们在三种不同类型的硬件平台上，使用两个 BNN 架构和广为人知的数据集对该框架进行了评估。结果表明，在所测试的数据集/BNN 和不同硬件平台上，与完全并行化的方法相比，我们提出的框架生成的二值神经网络推理映射能够实现显著更高的加速比。具体而言，在所测试的目标硬件上，我们的 *HEP-BNN* 框架生成的高效并行配置分别可将推理时间最多缩短 $2\times$、$2.6\times$ 和 $11.8\times$。此外，*HEP-BNN* 生成的、包含高效配置的图形处理器代码，也可用于实际采用二值神经网络推理的应用。

我们相信，我们的*HEP-BNN* 框架将帮助研究人员和从业者，为使用由中央处理器和图形处理器组成的异构平台的 BNN 推理系统寻找高效的执行配置。

致谢

本论文得到了德国研究联合会（DFG）项目 OneMemory（405422836）、协作研究中心 SFB 876“通过资源受限分析提供信息”（项目编号 124020371）子项目 A1（http://sfb876.tu-dortmund.de），以及德国联邦教育与研究部和北莱茵-威斯特法伦州作为 Lamarr 研究所的一部分提供的支持，

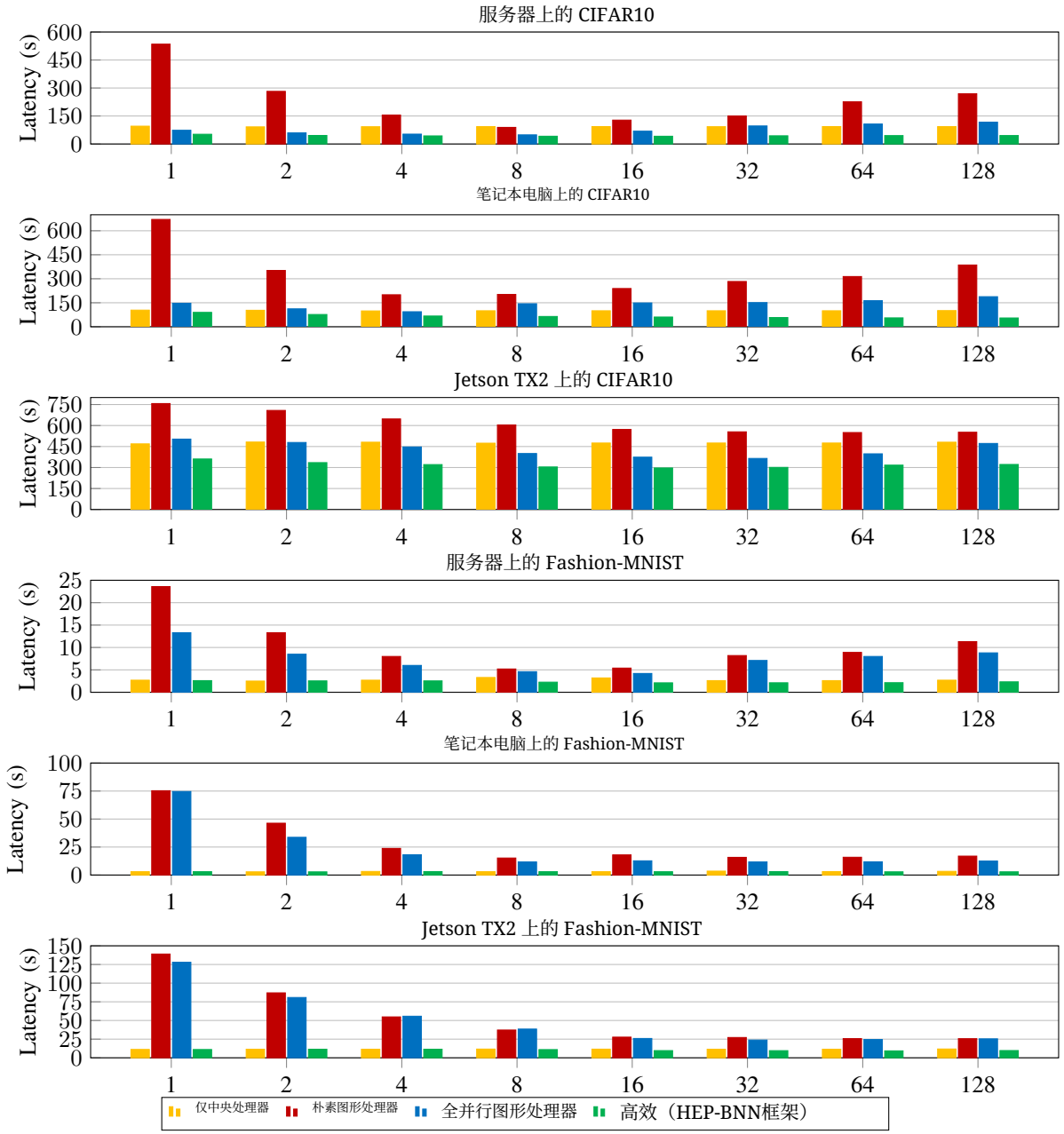


图 5. 整个 FashionMNIST 测试图像数据集（上方三幅图）和整个 CIFAR10 测试图像数据集（下方三幅图）的不同批大小执行时间对比。每个数据集均在三种不同的硬件配置下进行评估，分别为：服务器、笔记本电脑和 TX2。

该研究所致力于机器学习和人工智能，项目代号为 LAMARR22B。本研究获得了德国联邦教育与研究部（BMBF）通过 6GEM 研究中心提供的资助，资助编号为 16KISK038。

参考文献

[1] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed 和 H. Arshad，《人工神经网络应用的最新进展：综述》，*Heliyon*，第 4 卷，第 11 期，文章编号 e00938，2018 年。[2] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv 和 Y. Bengio，《二值化神经网络》，载于神经信息处理系统进展（*NIPS*），2016 年。[3] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh 和 D. Marr，《加速二值化神经网络：FPGA、中央处理器、图形处理器和 ASIC 的比较》，载于*2016* 年现场可编程技术国际会议（*FPT*），第 77–84 页，2016 年。[4] X. Xu 和 M. Pedersoli，《用于在 PyTorch 上进行网络二值化的计算内核》，*CoRR*，第 abs/1911.04477 卷，2019 年。

[5] A. Li 和 S. Su，《通过 Turing GPU 中的位张量核心加速二值化神经网络》，*IEEE*并行与分布式系统汇刊，第 32 卷，第 7 期，第 1878–1891 页，2021 年。[6] G. Chen, S. He, H. Meng 和 K. Huang，《PhoneBit：适用于移动电话的高效 GPU 加速二值神经网络推理引擎》，载于*2020* 年欧洲设计、自动化与测试会议暨展览会（*DATE*），第 786–791 页，2020 年。[7] E. Sari, M. Belbahri 和 V. P. Nia，《批归一化如何助益二值化训练？》，*arXiv:1909.09139*，2019 年。[8] S. Buschj¨ ager，《fastinference GitHub 仓库》。https://github.com/sbuschjaeger/fastinference。[9] H. Xiao, K. Rasul 和 R. Vollgraf，《Fashion-MNIST：用于机器学习算法基准测试的新型图像数据集》，2017 年。[10] A. Krizhevsky，《从微小图像中学习多层特征》，技术报告，2009 年。[11] K. Simonyan 和 A. Zisserman，《用于大规模图像识别的超深卷积网络》，载于国际学习表征会议（*ICLR*），2015 年。