

HARFLOW3D: A Latency-Oriented 3D-CNN Accelerator Toolflow for HAR on FPGA Devices

Petros Toupas^{*†} Alexander Montgomerie-Corcoran^{*} Christos-Savvas Bouganis^{*} Dimitrios Tzovaras[†]

^{*} Dpt. of Electrical and Electronic Engineering
Imperial College London
Email: {p.toupas21,
alexander.montgomerie-corcoran15,
christos-savvas.bouganis}@imperial.ac.uk

[†] Information Technologies Institute
Centre of Research and Technology Hellas
Email: {ptoupas,dimitrios.tzovaras}@iti.gr

Abstract—For Human Action Recognition tasks (HAR), 3D Convolutional Neural Networks have proven to be highly effective, achieving state-of-the-art results. This study introduces a novel streaming architecture based toolflow for mapping such models onto FPGAs considering the model’s inherent characteristics and the features of the targeted FPGA device. The HARFLOW3D toolflow takes as input a 3D CNN in ONNX format and a description of the FPGA characteristics, generating a design that minimizes the latency of the computation. The toolflow is comprised of a number of parts, including (i) a 3D CNN parser, (ii) a performance and resource model, (iii) a scheduling algorithm for executing 3D models on the generated hardware, (iv) a resource-aware optimization engine tailored for 3D models, (v) an automated mapping to synthesizable code for FPGAs. The ability of the toolflow to support a broad range of models and devices is shown through a number of experiments on various 3D CNN and FPGA system pairs. Furthermore, the toolflow has produced high-performing results for 3D CNN models that have not been mapped to FPGAs before, demonstrating the potential of FPGA-based systems in this space. Overall, HARFLOW3D has demonstrated its ability to deliver competitive latency compared to a range of state-of-the-art hand-tuned approaches being able to achieve up to 5× better performance compared to some of the existing works.

I. INTRODUCTION

The growing focus on video-related applications such as video surveillance, autonomous driving, and patient monitoring has necessitated the development of algorithms that integrate and take into account the temporal domain. 3D CNNs, which are often employed to deal with video and volumetric data, augment their learning capacity by extracting input features related to this additional dimension. Due to the temporal dimension, 3D CNNs often have larger computational and memory requirements compared to 2D CNNs. Particularly, 3D CNNs have exhibited high performance in the task of HAR, enabling the interpretation of human motion across video frames and the detection of various activities without the need for specialised time domain approaches (e.g. LSTMs). Whilst vision transformers have recently attained state-of-the-art accuracy, their operation requires orders of magnitude more GFLOPs comparatively.

Devices such as GPUs, FPGAs, and ASICs have been utilised to address the high processing requirements of 3D

CNNs and deliver high-performance systems. FPGAs are particularly attractive as an acceleration platform since they are more flexible than ASICs and more energy efficient than GPUs. The rapid growth and increasing complexity of 3D CNN model designs necessitate high-quality hardware designs that support short design cycles for new 3D CNN model specifications. The goal of this work is to provide an automated way for deploying 3D CNN models onto FPGA systems, with an emphasis on minimising the execution latency. The diversity of the supported models and devices makes it suitable for a number of applications and enables users to select the most appropriate 3D CNN model and device according to their unique demands and budget.

3D CNNs have been studied and developed for quite some time, and the topic of HAR is gaining attention year by year. A few studies have focused on mapping 3D CNNs to FPGAs, but the vast majority of them propose hand-tuned hardware architectures for specific 3D CNN models. While FPGA toolflows for 2D CNNs are well-researched [1], [2], there is an absence of 3D CNN FPGA toolflows. The distinct characteristics of 3D CNNs, such as their large workloads and significantly increased memory and resource requirements, require such toolflows.

Figure 1 displays the pareto-front both for prior works as well as the proposed toolflow, showing their achieved accuracy and latency. HARFLOW3D designs account for most of the points on the pareto-front, demonstrating the toolflow’s ability to generate pareto-optimal designs for a variety of 3D CNN models on HAR. The pareto-optimal relationship between accuracy and latency is an extremely desirable feature for a toolflow, as it allows a designer to trade-off their model’s accuracy for greater performance in a fine-grain manner.

The key contributions of this paper are the following:

- Introduction of HARFLOW3D, the first 3D CNN to FPGA toolflow, which supports a variety of models and devices, achieving competitive results compared to prior hand-tuned works.
- An optimization strategy accompanied by a set of transformations, providing tailored designs based on the characteristics of each 3D CNN model layer.

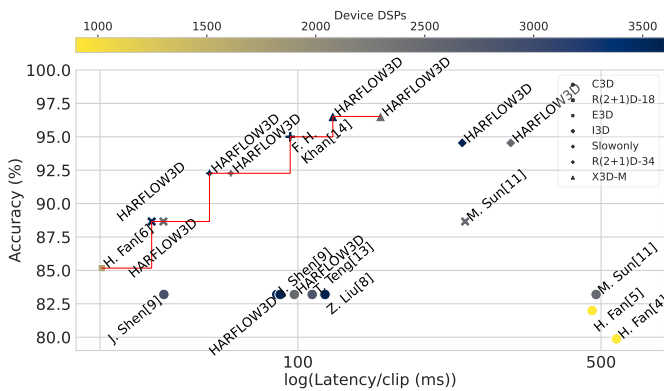


Fig. 1: Pareto front on 3D CNNs: Latency over Accuracy. Designs produced by the proposed HARFLOW3D toolflow dominate the pareto front.

- A set of highly-tuned parameterized building blocks that support runtime parameterization, allowing for significantly lower latency over the non-parameterized equivalent.
- A rich evaluation with experimental results across multiple devices and multiple models, including state-of-the-art 3D CNN HAR models that have not been addressed before, setting the landscape for FPGA-based HAR computation.

II. RELATED WORK

Whilst 3D CNNs have been around for a while, there have been few studies that focus on accelerating these networks on FPGAs. The majority of these studies have focused on older 3D CNNs such as the C3D [3] model, whose accuracy falls short of state-of-the-art models. Fan et al. [4]–[6] have released a series of publications about accelerating 3D CNNs for human action recognition using FPGAs. In their first work [4], they introduced the F-C3D hardware architecture for accelerating the C3D model, which is capable of supporting multiple 3D convolutional layers and includes solutions for resolving some of the challenges posed by 3D CNNs, such as higher processing and memory demands. Additionally, they demonstrated the portability of their design to other FPGA devices. In a following publication [5], they presented an analytical model and a tool for optimising the hardware architecture based on device specifications, accuracy requirements, and the usage of block floating point arithmetic precision to minimise accuracy loss. Their evaluation of the design was likewise performed on the C3D model. In their most recent publication [6], they proposed E3DNet, an effective 3D CNN based on the proposed 3D-1 bottleneck building block. The F-E3D hardware implementation of E3DNet achieves a real-time execution time of 35,3 milliseconds per frame and scores 85.1% accuracy on the UCF101 [7] benchmark.

Based on the similarities between the 2D and 3D convolution computing patterns, Liu et al. [8] presented a hardware design to accelerate both 2D and 3D CNNs. They sought to

turn CNN convolutions into matrix multiplication operations, and concentrated on minimising memory usage to overcome the challenges associated with replicating feature-maps. Applying an analytical model, they configured the accelerators for maximum resource utilisation and evaluated their design using the C3D model. Shen et al. [9] developed a template-based architecture based on the Winograd transform [10] that is capable of handling both 2D and 3D CNNs. In addition, they developed an analytical method for quickly exploring the design space for mapping 2D and 3D CNNs onto FPGA accelerators and validated their design with the C3D model. Sun et al. [11] applied weight pruning to the C3D and R(2+1) [12] 3D CNN architectures using a blockwise approach. Their hardware architecture, which is based on the Alternating Direction Method of Multipliers(ADMM), enables the acceleration of 3D CNNs with minimal accuracy loss as compared to their un-pruned counterparts.

Teng et al. [13] presented a design space exploration strategy for optimizing memory access in 3D CNN models accelerated on FPGAs. The authors proposed a non-overlapping data tiling method for off-chip memory access and explored on-chip data reuse using different loop ordering strategies. They further proposed a hardware architecture that can support these strategies. Their experiments showed that the proposed approach on the C3D model achieved state-of-the-art performance compared to prior FPGA implementations at that time. Khan et al. [14] investigated various 3D CNN design parameters for resource-limited platforms, focusing on the I3D model, a 70-layer deep network for video action recognition. They adjusted the feature-map word lengths and weights in a pre-trained model, which reduced its complexity without affecting its accuracy. They proposed a data tiling technique that utilizes all four dimensions of video data and improves memory bandwidth while reducing DRAM accesses. Based on these optimizations, the proposed FPGA accelerator achieves 684 GOPs/s for 32-bit floating point and 1.29 TOPs/s for 8-bit integer implementations with a 2% accuracy drop.

The majority of research has been largely focused on the C3D [3] model for HAR, which was introduced in 2013. The model’s architecture is rather simple, consisting of only 8 convolutional layers, while it performs poorly in terms of accuracy when compared to recent SoA models in HAR (85.2% in UCF101 vs. the current SoA’s 98.6%). In terms of design complexity, an analogy can be made to AlexNet [15], but in three-dimensional space. Since the aforementioned approaches are mostly focused on the design of the specific C3D model, it is not clear how they can be extended, evaluated, or applied to the more complicated networks of modern state-of-the-art HAR models. The proposed toolflow addresses this by supporting a broad variety of recent 3D CNN designs, such as X3D [16] and Slowonly [17] among others, which include more complex ResNet3D-wise architectures as well as older models like C3D for direct comparison with prior studies.

III. PROPOSED ARCHITECTURE

This section outlines the basics of the hardware-level architecture of the toolflow's generated designs. The suggested toolflow adheres to the same streaming architecture principles as fpgaConvNet [18], which is based on the Synchronous Data-Flow (SDF) computation model.

A. Neural Network Model Parser

To facilitate the process of incorporating various neural network (NN) models into the toolflow, a dedicated NN model parser has been developed. This parser is designed to read and map NN models in the ONNX format, a standardized format supported by many deep learning frameworks such as PyTorch and TensorFlow, into the format required by the toolflow. The 3D CNN model, can be described as a Directed Acyclic Graph (DAG), which is denoted as $M = \{l_1, \dots, l_L\}$, where l_i is the i^{th} layer within the set of model layers L and is expressed as an execution node of M . This is then translated into a Synchronous Data-Flow Graph (SDFG) which is also directed and acyclic, denoted as G with N computation (or hardware) nodes, where $G = \{n_1, \dots, n_N\}$. The core concept of synchronous dataflow modelling is that each node fires whenever data is available at its inputs, resulting in a paradigm of data-driven execution. This format is compatible with the rest of the toolflow's tools such as the latency optimiser and the toolflow's resource and performance models.

B. Building Blocks Description

Each layer of a NN model accepts input data to be processed and returns output data once its operation has been completed. These inputs and outputs are described as feature-maps in and out respectively. The maximum feature-map dimensions supported for a given hardware node n are described as below.

$$\begin{aligned} \mathbf{S}_n^{in} &= \{H_n^{in}, W_n^{in}, D_n^{in}, C_n^{in}\} \\ \mathbf{S}_n^{out} &= \{H_n^{out}, W_n^{out}, D_n^{out}, C_n^{out}\} \end{aligned}$$

where $H_n^{in/out}$, $W_n^{in/out}$, $D_n^{in/out}$, and $C_n^{in/out}$ are the spatial dimensions (Height, Width), followed by the temporal dimension (Depth), followed by the number of Channels for the input and output respectively. The size of the feature-map in regards to the number of elements is referred to as $|\mathbf{S}|$. The \mathbf{S}_n^{in} , and \mathbf{S}_n^{out} parameters exist for all of the layers as part of their parameter space definition. Alongside functional parameters, the hardware accepts different fixed-point precisions at compile-time. The rest of the parameters of the layers, and their intermediate representation definitions are detailed in Table I, and a description is provided below. The runtime parameters are differentiated from compile-time parameters using the \wedge symbol above a parameter. Computation node parameters are subscripted with n , and execution node parameters with l .

• Convolution 3D

Due to the necessity to support a range of 3D CNN models, the toolflow's convolution building block is designed to accommodate the following types of convolution operation: (a) Full convolution $K^D \times K^H \times K^W$ (b) Spatial

TABLE I: Compile time parameters for node n in the hardware graph G , for each layer type.

Convolution	
F_n	Number of filters (output channel dimension)
\mathbf{K}_n	3D kernel size (K_n^D, K_n^H, K_n^W)
\mathbf{J}_n	3D stride (J_n^D, J_n^H, J_n^W)
\mathbf{P}_n	3D Padding ($P_n^{D_s}, P_n^{D_e}, P_n^{H_s}, P_n^{H_e}, P_n^{W_s}, P_n^{W_e}$)
Gr_n	Grouping along the channel dimension
c_n^{in}	parallel streams in
c_n^{out}	parallel streams out
\hat{f}_n	Vector dot product folding
Fully Connected	
F_n	Number of filters (output channel dimension)
c_n^{in}	Number of parallel streams in
c_n^{out}	Number of parallel streams out
Pooling	
T_n	Type of activation
\mathbf{K}_n	3D kernel size (K_n^D, K_n^H, K_n^W)
\mathbf{J}_n	3D stride (J_n^D, J_n^H, J_n^W)
\mathbf{P}_n	3D Padding ($P_n^{D_s}, P_n^{D_e}, P_n^{H_s}, P_n^{H_e}, P_n^{W_s}, P_n^{W_e}$)
c_n	Number of parallel streams in & out
Activation	
T_n	Type of activation
c_n	Number of parallel streams in & out
Global Average Pooling	
c_n	Number of parallel streams in & out
Element-Wise	
T_n	Type of element-wise operation
B_n	Mode of operation (default or broadcast)
c_n	Number of parallel streams in & out

convolution $1 \times K^H \times K^W$ (c) Temporal convolution $K^D \times 1 \times 1$ (d) Depth-wise convolution (e) Point-wise convolution. The computation node is characterised by the following tuple of parameters:

$$\Gamma = \{\hat{\mathbf{S}}^{in}, \hat{\mathbf{S}}^{out}, \hat{\mathbf{K}}, \hat{\mathbf{J}}, \hat{\mathbf{P}}, \hat{G}_r, \hat{c}^{in}, \hat{c}^{out}, \hat{f}\}$$

• Pooling 3D

For 3D pooling layer, the toolflow supports both maximum and average pooling which can be chosen at runtime by the enumerated parameter T . The computation node is characterised by the following tuple of parameters:

$$\Gamma = \{\hat{\mathbf{S}}^{in}, \hat{\mathbf{S}}^{out}, \hat{\mathbf{K}}, \hat{\mathbf{J}}, \hat{\mathbf{P}}, \hat{T}, \hat{c}\}$$

• **Global Pooling, Activation & Element-Wise 3D** Although Global Pooling is a special case of the regular Pooling layer, the hardware for it is optimised for this case. The supported activation functions of the activation layer are the following:

- ReLU activation,
- Sigmoid activation,
- Swish activation defined as: $y = x * \text{sigmoid}(x)$

where the type of activation is given by the parameter T . The runtime choice for broadcasting is defined as B , which is either true or false. The type of element-wise operation is given by the parameter T . The computation node is characterised by the following tuple of parameters:

$$\Gamma = \{\hat{\mathbf{S}}^{in}, \hat{\mathbf{S}}^{out}, \hat{T}, \hat{B}, \hat{c}\}$$

- **Fully Connected** Fully Connected layers share hardware with Convolution layers, but with no feature-map buffering. The computation node is characterised by the following tuple of parameters:

$$\Gamma = \{\hat{S}^{in}, \hat{S}^{out}, \hat{c}^{in}, \hat{c}^{out}\}$$

C. Hardware Design and Implementation

The proposed architecture follows the paradigm of a system consisting of a processor extended by a set of custom instructions. The core building blocks described before are equivalent to custom instructions, and their control is performed through a CPU. Each building block is connected to a crossbar that is responsible for handling the routing of data between the building blocks and off-chip memory, as well as performing inter building block routing. The memory access to and from memory is supported through dedicated DMA blocks.

The architecture of the custom instructions follows the Streaming Architecture paradigm, by implementing direct convolution operations and exploring the opportunity of driving the instantiated building blocks as dictated by the 3D model without accessing the off-chip memory, but at the same time can map multiple operations on the same building blocks in a time-shared manner avoiding the need for reconfiguring the FPGA fabric.

The toolflow is responsible to identify the required building blocks in order to support the computations of a given 3D CNN model, as well as to tune them in order to optimise the performance of the system given the available resources (FPGA resources and off-chip memory bandwidth). As such, the resulting system is a heterogeneous multi-core system, with blocks tailored to the 3D CNN model and the targeted FPGA device. This deviates from the approach taken in other streaming toolflows such as FINN [19], and fpgaConvNet [18], where the hardware is tailored to specific DNN models with layers being deeply pipelined, utilising bitstream reconfiguration to overcome resource constraints. Such design approach leads to high performance designs for throughput oriented applications, however bitstream reconfiguration inhibits the ability to target latency-driven applications with a latency target smaller than the reconfiguration time of the device, which is usually in the order of hundreds of milliseconds.

Figure 2 illustrates a simplified diagram of an example accelerator generated for a given model. Feature-maps are sent to and from off-chip memory via a pair of DMAs, and sent to the hardware nodes through the configurable crossbars. The design has a sandwich-like architecture, with the AXI-Stream crossbars routing data to and from hardware nodes. The output crossbar connects to the input crossbar, allowing for inter-connectivity between hardware nodes.

The fpgaConvNet [18] toolflow is utilised for the implementation of the computation blocks. We regard this as the *baseline* design, where no runtime configuration is supported, and only padded execution can be used to execute variable feature-map sizes. Figure 3 illustrates how the baseline design has been modified to support runtime configurable layers.

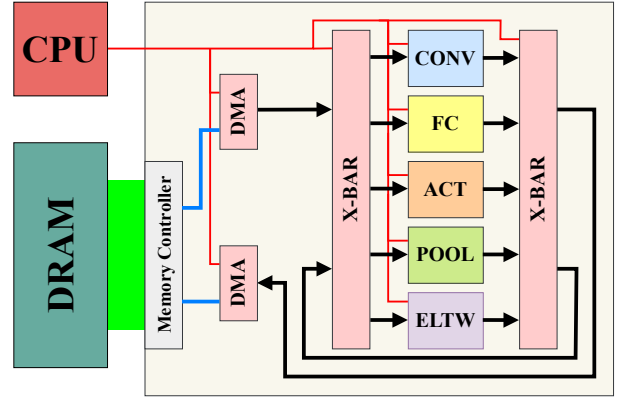


Fig. 2: Block diagram of an accelerator instance produced by the HARFLOW3D toolflow. The black lines describe AXI-Stream signals, where the arrows indicate the directionality of the connection, blue are high-throughput AXI interfaces for DMA access, red are AXI-Lite connections for runtime configuration of the hardware nodes, and green indicate the DDR IO interfaces for communicating with off-chip memory.

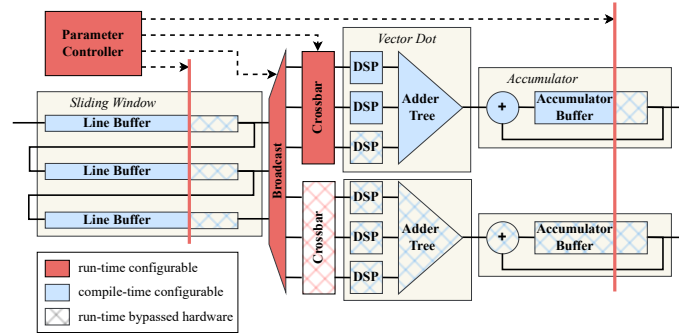


Fig. 3: Diagram of hardware for Convolution, and how it can be used with runtime parameters. The blue blocks represent compile-time configurable hardware modules. The red blocks represent runtime configurable hardware modules. Cross-hatching gives an example of how hardware elements can be bypassed at runtime.

The highlighted blocks constitute the overhead required for supporting runtime configurability. The additional hardware includes a configurable counter to change the depth of the line buffers and accumulation buffers, and a crossbar to map configurable kernel sizes to a configurable number of multipliers. These extra resources are insignificant, and there is no change in targetable clock frequency.

IV. MODELLING

Precise, high-quality performance and resource modelling is essential to support a rapid and valid design space exploration. This section presents a comprehensive performance model for the parameterized building blocks described in Section III. Performance models are described per computation node, and the resource model is for the entire system design.

A. Performance Modelling

Performance modelling is required to evaluate the latency objective during design space exploration. The latency of the execution of a layer can be estimated by a roof-line model consisting of the required memory bandwidth, and the latency from computation. The latency model for each supported hardware type as a function of its runtime parameters Γ is given as follows:

$$\mathcal{L}_{Conv}(\Gamma) = \frac{|\hat{\mathbf{S}}^{out}| \cdot \hat{F} \cdot |\hat{\mathbf{K}}|}{\hat{c}^{out} \cdot \hat{c}^{in} \cdot \hat{f}}$$

$$\mathcal{L}_{FC}(\Gamma) = \frac{\hat{C} \cdot \hat{F}}{\hat{c}^{in} \cdot \hat{c}^{out}}$$

$$\mathcal{L}_{Pool}(\Gamma) = \mathcal{L}_{Act}(\Gamma) = \mathcal{L}_{EltWise}(\Gamma) = \frac{|\hat{\mathbf{S}}^{in}|}{\hat{c}}$$

The preceding models assume unlimited memory bandwidth, however memory accesses have limited throughput. In order to model the effect of memory bandwidth, the consumption and production rates of the layer are required, which are given as,

$$r_n^{in}(\Gamma) = \frac{|\hat{\mathbf{S}}^{in}|}{\mathcal{L}_n(\Gamma) \cdot \hat{c}^{in}}, \quad r_n^{out}(\Gamma) = \frac{|\hat{\mathbf{S}}^{out}|}{\mathcal{L}_n(\Gamma) \cdot \hat{c}^{out}}$$

where $r_n^{in}(\Gamma)$ and $r_n^{out}(\Gamma)$ are the words per cycle per stream in and out of the computation node n when executing runtime parameters Γ .

For convolution and fully-connected layers in particular, extra memory bandwidth is required to stream in the weight parameters. This rate is described as,

$$r_{Conv,FC}^{param}(\Gamma) = \frac{\hat{C}^{in} \cdot \hat{F} \cdot |\hat{\mathbf{K}}|}{\mathcal{L}_{Conv,FC}(\Gamma) \cdot \hat{c}^{in} \cdot \hat{c}^{out} \cdot \hat{f}}$$

Alongside the double-buffering of weights, if the channel dimension of a convolution or full-connected layer is folded, the partial sums must be accumulated. This requires streaming the previous partial sum from off-chip memory, whose rate matches the rate out.

$$r_{Conv,FC}^{psum}(\Gamma) = r_{Conv,FC}^{out}(\Gamma)$$

The constrained bandwidth in and out can be described as,

$$\mathcal{B}_n^{in}(\Gamma) = \min\{\mathcal{B}_{DMA}^{in}, r_n^{in}(\Gamma) \cdot \hat{c}^{in}\}$$

$$\mathcal{B}_n^{out}(\Gamma) = \min\{\mathcal{B}_{DMA}^{out}, r_n^{out}(\Gamma) \cdot \hat{c}^{out}\}$$

where $\mathcal{B}_n^{in}(\Gamma)$ and $\mathcal{B}_n^{out}(\Gamma)$ are the constrained words per cycle for the execution of parameters Γ on the computation node n . This describes a roofline model, where the on-chip bandwidth is capped by the memory bandwidth. The above model summarises the bandwidth in and out for all layers apart from convolution and fully-connected. To describe these, the bandwidth for parameters and partial sums must also be considered,

$$\mathcal{B}_{Conv,FC}^{in}(\Gamma) = \min\{\mathcal{B}_{DMA}^{in}, r_n^{in}(\Gamma) \cdot \hat{c}^{in} + r_{Conv,FC}^{psum}(\Gamma) \cdot \hat{c}^{out} + r_{Conv,FC}^{param}(\Gamma) \cdot \hat{c}^{in} \cdot \hat{c}^{out} \cdot \hat{f}\}$$

For *Fully Connected* layers, $f = 1$. Given the bandwidths, the total latency for executing a layer is given as,

$$\tilde{\mathcal{L}}_n(\Gamma) = \max\left\{\frac{|\hat{\mathbf{S}}^{in}|}{\mathcal{B}_n^{in}(\Gamma)}, \frac{|\hat{\mathbf{S}}^{out}|}{\mathcal{B}_n^{out}(\Gamma)}\right\} \quad (1)$$

It is worth noting that the performance models are functions of runtime parameters, owing to the highly customisable hardware. Streaming Architectures tend to be computationally bounded, as supported by the results presented in Section VII. As the size of the feature-maps is significantly greater than that of the weights, the weights bandwidth is often negligible in comparison to the overall bandwidth required. There is no latency associated with updating the runtime parameters because they are also double-buffered and require negligible information transfer (<100B).

B. Resource Modelling

Resource modelling is used to explore the performance-resource design space whilst keeping designs within the target FPGA's constraints. Modern FPGA devices share four common resource types: DSP, BRAM, LUT and FF. Only the *Conv* and *FC* layers use DSP resources, An analytical model of DSP usage for building blocks n of these layer types is given by,

$$\mathcal{R}_{Conv}^{DSP} = c_n^{in} \cdot c_n^{out} \cdot f_n, \quad \mathcal{R}_{FC}^{DSP} = c_n^{in} \cdot c_n^{out}$$

As a 16-bit fixed-point precision is used throughout the design, each DSP is used for either $16 \cdot 16$ multiplication or multiplication-accumulation. For BRAM modelling, the number of BRAM blocks can be described as,

$$\mathcal{R}^{BRAM}(depth, words) = \left\lceil \frac{depth}{512} \right\rceil \cdot \left\lceil \frac{16 \cdot words}{36} \right\rceil$$

The bus width of the required memory is $16 \cdot words$ as 16-bit fixed-point is used. Only *Conv*, *FC* and *Pool* layers consume BRAM components. Both *Conv* and *Pool* use BRAM for the Sliding Window module, which is described as,

$$\mathcal{R}_{SIW}^{BRAM} = \mathcal{R}^{BRAM}\left(W_n \cdot D_n \cdot \frac{C_n^{in}}{c_n^{in}}, (K_n^H - 1) \cdot c_n^{in}\right) +$$

$$\mathcal{R}^{BRAM}\left(D_n \cdot \frac{C_n^{in}}{c_n^{in}}, K_n^H \cdot (K_n^W - 1) \cdot c_n^{in}\right) +$$

$$\mathcal{R}^{BRAM}\left(\frac{C_n^{in}}{c_n^{in}}, K_n^H \cdot K_n^W \cdot (K_n^D - 1) \cdot c_n^{in}\right)$$

Conv and *FC*, require some extra memory for storing weights on-chip, which is modelled as,

$$\mathcal{R}_{Weight}^{BRAM} = \mathcal{R}^{BRAM}\left(\frac{C_n^{in} \cdot F_n \cdot |\mathbf{K}_n|}{c_n^{in} \cdot c_n^{out} \cdot f_n}, c_n^{in} \cdot c_n^{out} \cdot f_n\right)$$

For *Fully Connected* layers, $\mathbf{K}_n = \{1, 1, 1\}$ and $f_n = 1$. The hardware design uses a large data word design technique, which significantly improves BRAM utilisation.

For the modelling of LUT and FF resources, a regression model is used due to the non-deterministic nature of FPGA synthesis. The regression models are obtained from a data set of 5000 synthesised modules, where the relationship between the module's parameters and resources are inferred.

Using the derived resource models, we can estimate the resource consumption of a complete hardware graph (G),

$$\mathcal{R}_{total} = \left(\sum_{n \in G} \mathcal{R}_n \right) + \mathcal{R}_{DMA} + \mathcal{R}_{xbar}$$

where \mathcal{R} describes the complete resources (DSP, BRAM, LUT, FF), for the given component. The quality of the resource model is evaluated in Section VI.

V. LATENCY-DRIVEN DESIGN SPACE EXPLORATION

The minimization of the 3D-CNN model's execution latency on the runtime-configurable accelerator is regarded as an optimization problem. As such, a Design Space Exploration is performed that searches for both an efficient accelerator for the specific application and a schedule for execution of the 3D CNN model layers on that architecture.

A. Scheduling

Once an accelerator design has been created, a schedule is needed for executing the 3D-CNN model's layers for this given design. The main choices with regards to scheduling are:

- Mapping of the computation nodes for executing the 3D-CNN model's layers (i.e. execution nodes).
- Tiling of the feature-maps of execution nodes on a given computation node.
- Runtime configurations for all the invocations of the computation nodes based on the respective execution nodes parameters.

The mapping between a computation node and the respective execution nodes which it will execute is described as an execution mapping function $\mathcal{E} : G \mapsto \mathcal{P}(M)$, where $\mathcal{P}(M)$ is the power-set of M , which is all the distinct subsets of M . This mapping creates disjoint subsets of M , which can be described as,

$$\mathcal{E}(n) \cap \mathcal{E}(m) = \emptyset \forall n, m \in G, n \neq m$$

where \mathcal{E} is the execution mapping function. The mapping function must give unique mappings for each computation node, such that none of the model's layers are executed more than once. The inverse mapping, \mathcal{E}^{-1} finds the corresponding computation node for a given execution node. The mapping is decided based on the transform described in Section V-C4.

Once a mapping has been decided, the schedule for G , denoted as Φ_G , can be created for executing the 3D CNN model graph M on the hardware graph G . This schedule is outlined in Algorithm 1. For each execution node l of the 3D CNN model graph, the tiling factors across each of its dimensions is obtained, which are then used to find the tile sizes for computation. The algorithm greedily allocates as much of the feature-map as possible onto the computation node, and then chooses the coarse factors based on the tile shape. Additional runtime parameters such as kernel sizes and padding are also chosen based on the execution node's parameters.

Having created the schedule Φ_G , it can be used for executing the workload. The ordering of dimensions in the proposed

Algorithm 1 Scheduling Algorithm

```

1:  $\Phi =$  empty list ▷ initialise an empty schedule
2: for  $l$  in  $M$  do
3:    $n = \mathcal{E}^{-1}(l)$  ▷ get the computation node
4:   for  $i$  in  $\text{range}(\lceil \frac{H_i^{in}}{H_n^{in}} \rceil, \lceil \frac{W_i^{in}}{W_n^{in}} \rceil, \lceil \frac{D_i^{in}}{D_n^{in}} \rceil, \lceil \frac{C_i^{in}}{C_n^{in}} \rceil)$  do
5:      $\hat{H} = \min\{H_n^{in}, H_l^{in} - i^H \cdot H_n^{in}\}$ 
6:      $\hat{W} = \min\{W_n^{in}, W_l^{in} - i^W \cdot W_n^{in}\}$ 
7:      $\hat{D} = \min\{D_n^{in}, D_l^{in} - i^D \cdot D_n^{in}\}$ 
8:      $\hat{C} = \min\{C_n^{in}, C_l^{in} - i^C \cdot C_n^{in}\}$ 
9:     if  $\text{type}(n)$  is Conv or FC then
10:      for  $i^F$  in  $\text{range}(\lceil \frac{F_l^{in}}{F_n^{in}} \rceil)$  do
11:         $\hat{F} = \min\{F_n^{in}, F_l^{in} - i^F \cdot F_n^{in}\}$ 
12:         $c^{in} = \max\{\text{factors } \hat{C}\}$ 
13:         $c^{out} = \max\{\text{factors } \hat{F}\}$ 
14:         $\Gamma = \{\hat{H}, \hat{W}, \hat{D}, \hat{C}, \hat{F}, c^{in}, c^{out}\}$ 
15:         $\Phi$  append  $(n, \Gamma)$ 
16:      else
17:         $c = \max\{\text{factors } \hat{C}\}$ 
18:         $\Gamma = \{\hat{H}, \hat{W}, \hat{D}, \hat{C}, \hat{c}\}$ 
19:         $\Phi$  append  $(n, \Gamma)$ 

```

accelerator is $NHWDC$, where the channel dimension is the fastest changing, and so the schedule is also executed in this order. The total latency for execution is described as,

$$\mathcal{L}_{total}(G) = \sum_{n, \Gamma \in \Phi_G} \tilde{\mathcal{L}}_n(\Gamma) \quad (2)$$

where n is the computation node and Γ is the corresponding set of parameters for each configuration in the schedule. The latency $\tilde{\mathcal{L}}_n(\Gamma)$ of each node n for a configuration Γ is described in Equation (1).

B. Optimization Strategy

Simulated annealing (SA), a meta-heuristic for finding minima in non-convex functions, is adopted as the optimization approach for minimising the latency of a 3D CNN model to FPGA mapping. The implementation of SA for this particular optimisation problem is given in Algorithm 2.

The main objective of the algorithm is to seek the global minimum of a given cost function; in this case, the cost function is latency, which is derived from Equation (2). Starting at an initial state by producing random values for the parameters of the computation nodes, transformations to the hardware graph G are applied iteratively generating new states that may be accepted or rejected based on a policy described in Algorithm 2. Each new state is evaluated based on the system's predetermined constraints, and is only accepted if it satisfies all of them. The constraints that must be satisfied on the proposed system are the following:

- The available memory bandwidth should not be exceeded.
- The total used resources \mathcal{R}_{total} should not exceed the available resources of the device.

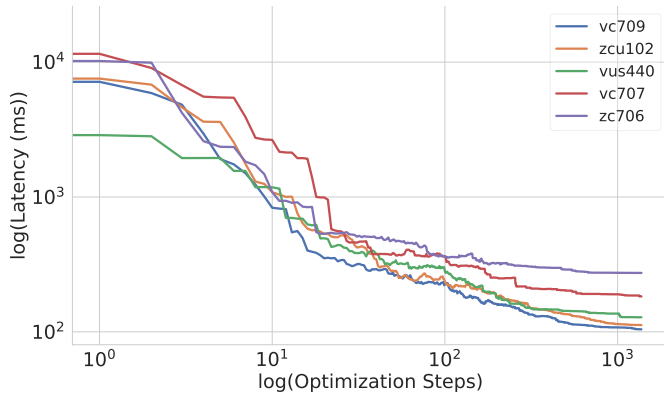


Fig. 4: Latency evolution during Simulated Annealing.

- The streams in and out (c_n^{in}, c_n^{out}) of each computation node should be a factor of the channels in and out respectively.
- The derived scheduled parameters of each layer must be less than the maximum supported by the computation node.

Figure 4 depicts the evolution of latency during Simulated Annealing. The graph depicts how the latency of the C3D model on various FPGA devices evolves over time. As indicated by the graph, the starting point for each run has an extremely high latency, as all of the parameters are set to random values. The latency continues to improve until it reaches a plateau, at which point the optimization is terminated.

C. Transformations

In order to traverse the design space, a set of transforms to the hardware graph G are implemented, and are elaborated below.

1) *Feature-Map Dimensions Reshaping*: The feature-map dimensions dictate the amount of on-chip memory required both for the weight parameters as well as caching required by the Sliding Window module, as given in Section IV-B. At compile time, a fixed shape for the feature-map dimensions must be given to the computation node, and at run time the

Algorithm 2 Simulated Annealing Optimisation Algorithm

```

1:  $\tau = \tau_{start}, G_{new} = G_{init}$  ▷ initialisation
2: while  $\tau > \tau_{min}$  do
3:    $G_{prev} = G_{new}$  ▷ store previous design
4:    $\mathcal{L}_{prev} = \mathcal{L}_{total}(G_{prev})$  ▷ get design latency
5:    $G_{new} = \text{random transformations on } G_{prev}$ 
6:    $\mathcal{L}_{new} = \mathcal{L}_{total}(G_{new})$  ▷ get new design latency
7:   if constraints satisfied then
8:     if  $\mathcal{L}_{new} < \mathcal{L}_{prev}$  then
9:       if  $\psi(\mathcal{L}_{prev}, \mathcal{L}_{new}, \tau) < x \sim U(0, 1)$  then
10:         $G_{new} = G_{prev}$  ▷ reject new design
11:    $\tau = \lambda \cdot \tau$  ▷ reduce temperature

```

where $\psi(\mathcal{L}_{prev}, \mathcal{L}_{new}, \tau) = \exp(-(\mathcal{L}_{prev} - \mathcal{L}_{new})/\tau)$

layer can be executed on the computation node by tiling the feature-map.

When the optimizer searches for the optimal feature-map shape configuration for the computation node for both the input and output, the following conditions must hold,

$$\begin{aligned}
D_n &\leq \max\{D_l : l \in M\} \\
H_n &= \max\{H_l : l \in M\} \\
W_n &\leq \max\{W_l : l \in M\} \\
C_n &\in \{\text{factors } C_l : l \in M\}
\end{aligned}$$

As the choice of row dimension has no impact on resources, the maximum of all rows is chosen. For depth and columns, any dimension that is both less than the max of all layers, and greater than a minimum feasible dimension is acceptable. The channel dimension is chosen to be a factor of any of the existing channel dimensions.

2) *Coarse-grain Folding*: The coarse-grain folding transformation modifies the number of parallel executions of coarse operations in each layer in order to achieve parallelism over the channel dimensions of the input feature-map. The primary operations of each layer can be performed simultaneously by deploying (at max) as many instances of its processing blocks as the number of channels. On Fully Connected and 3D Convolutional layers the coarse level parallelism can be utilised at both the input and output channels. This parallelism is achieved by updating and searching for appropriate values of the compile-time parameters c_n^{in} , c_n , and c_n^{out} during optimization, which are also taken into account by the performance and resource models. To be considered valid, a design must comply with the constraints driving this transformation.

$$c_n^{in}, c_n \in \text{factors } C_n^{in}, c_n^{out} \in \text{factors } C_n^{out}$$

3) *Fine-grain Folding*: The second folding-wise transformation factor determines the parallelism of the vector dot product operation for 3D convolutional layers. This sort of parallelism specifies the number of multipliers to be set in parallel for multiplications and the number of levels on the adder trees for additions. By increasing the compile-time fine folding factor, achievable latency is reduced at the cost of extra DSP resources, as described in Section IV. Evidently, there is a trade-off between performance and resource utilization. This type of parallelism is accomplished by modifying the f_n parameter during the optimization process. The constraint on the compile-time fine-grain folding f_n is given as,

$$f_n \in \text{factors } |\mathbf{K}_n|$$

4) *Combination and Separation of Computation Nodes*: As stated in Section III-C, the toolflow allows several model execution nodes to share the same computation node. The initial mapping creates unique computation nodes n for each execution node l . In NNs with multiple layers, this is impractical since the FPGA resources can be quickly exhausted, and the performance of each computation node would be compromised in order to fit the design. A combination of execution nodes by type for a single computation node (the available types

are depicted in Figure 2) is proposed as a solution to this issue. All execution nodes of the same type are combined and mapped onto a single computation node at the beginning of the optimization. The compile-time parameters of this node are then modified such that it can handle the workload of its associated execution nodes. This transformation is employed throughout the optimization procedure, and can affect the computation and execution nodes in two ways:

- Separate Computation Nodes: The algorithm chooses L_e execution nodes, where L_e is a hyperparameter, and detaches them from their corresponding computation node. The new group of execution nodes are integrated and mapped onto new computation nodes whose characteristics and parameters are adapted respectively.
- Combine Computation Nodes: The algorithm searches for computation nodes of the same type and selects N_c of them, where N_c is a hyperparameter, to combine into a single computation node. The computation node's compile-time parameters are updated to support the new set of workloads.

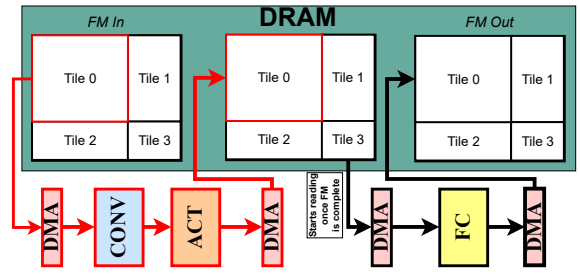
Each time the combination or separation is applied, a set of constraints are considered to assure the result's validity. The required constraints are a combination of the *Feature-Map Dimensions Reshaping*, *Coarse-grain Folding*, and *Fine-grain Folding* constraints.

VI. HARDWARE MODEL VALIDATION

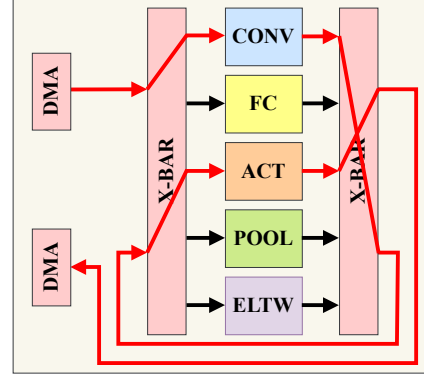
Modeling of performance and resources is used extensively in this work for rapid traversal of the large design space. In this section, the accuracy of the performance and resource models is validated across different hardware designs for C3D [3]. We demonstrate negligible error between modelled and measured results.

Table II shows a direct comparison between predicted resources and resources after synthesis for a C3D design. The DSP and BRAM models are highly accurate due to the deterministic nature of their synthesis, as resource type annotations are used in the hardware design. For the LUT and FF prediction accuracy, the modelling over-predicts LUT usage and under-predicts FF usage. Logic optimisation contributes to fewer LUTs in the final implemented design, and the additional FF resources likely arise from inter-module buffering that is neglected in the modelling. Additional BRAM resources are required by DMAs for the buffering of bursts across the feature-map. This is accounted for during optimisation.

The convolution layers dominate the total resource consumption, with DSPs typically being the limiting factor. The convolution layers are investigated further, where statistical resource modelling information is captured over 16 designs with varying configurations among different layers. Supporting the results of Table II, the Mean Absolute Percentage Error (MAPE) and the Standard Deviation (σ) over the 16 different convolution configurations are shown in Table III.



(a) 3D CNN Model to SDFG mapping



(b) Hardware execution

Fig. 5: The Dataflow of a simple design consisting of a Convolution, ReLU, and FC layers. As the red lines and the crossbar dictate the flow between Convolution and ReLU can be addressed within the FPGA without sending the data back to the off-chip memory. This is the result of Fuse Activation optimization.

TABLE III: Statistical resource modeling information over multiple runs for different convolution layers and configurations

	DSP	BRAM	LUT	FF
MAPE (%)	0.0	0.35	7.21	8.81
σ	0.0	0.38	8.82	2.89

The accuracy of the performance model is evaluated in Figure 6, where the calculation of the absolute percentage error is given by: $error = \frac{|Predicted - Measured|}{Measured} * 100$. The small percentage differences between predicted and measured results on ZCU106 board imply a good level of accuracy and confidence in the toolflow's modelling results. The divergence between the expected and actual latency of the layers is due to the DMA introducing a delay between bursts due to memory access cycles. The MAPE for all convolution layers of the C3D model at the particular design is 6.64%.

VII. EVALUATION

This section focuses on the evaluation of the proposed methodology and its ability to discover optimal designs. The automatically generated designs are benchmarked against existing 3D CNN accelerators.

TABLE II: Comparison of predicted and synthesised resources for C3D designs on a ZCU102 board.

Hardware Node	DSP			BRAM			LUT			FF		
	pred.	act.	error	pred.	act.	error	pred.	act.	error	pred.	act.	error
Conv	2304	2304	(+0%)	1052	1052	(+0%)	151K	138K	(+9.4%)	155K	166K	(-6.6%)
MaxPool	0	0	(+0%)	0	0	(+0%)	22K	17K	(+29.4%)	16K	18K	(-11.1%)
Gemm	128	128	(+0%)	456	456	(+0%)	11K	10K	(+1.0%)	15K	18K	(-16.6%)
ReLU	0	0	(+0%)	0	0	(+0%)	1.0K	1.4K	(-28.5)	2.2K	2.2K	(+0%)
DMA		0			51			2.9K			4.7K	
X-BAR		0			0			1.7K			1.4K	
Total Avail.	2432	2432 (2520)	(+0%)	1559	1559 (1824)	(+0%)	189K	171K (274K)	(+7.8%)	194K	210K (548K)	(-9.4%)

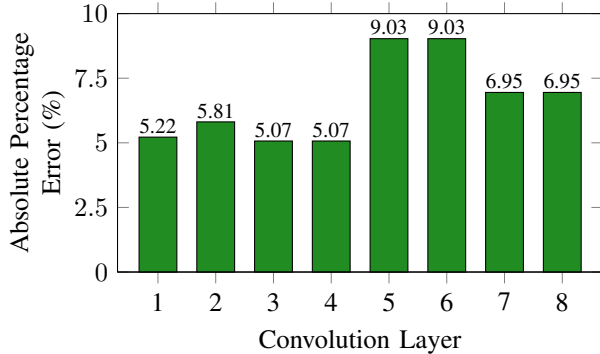


Fig. 6: Comparison of predicted latency to measured latency for all the convolution layers of C3D on the ZCU106 board depicted as absolute percentage error.

TABLE IV: 3D CNN models characteristics

	C3D	Slowonly	R(2+1)D-18	R(2+1)D-34	X3D-M
FLOPs (G) [†]	38.61	54.81	8.52	12.91	6.97
Parameters (M)	78.41	32.51	33.41	63.72	3.82
Num. of Layers	27	174	82	154	396
Num. of Conv Layers	8	53	37	69	115
Spatial dimensions	112 × 112	256 × 256	112 × 112	112 × 112	256 × 256
Num. of Frames	16	8	16	16	16
UCF101 Accuracy (%)	83.2	94.54	88.66	92.27	96.52

[†] FLOPs are reported as MAC operations.

The models of 3D CNN included in the evaluation are listed in Table IV. Each of these models has demonstrated state-of-the-art performance in a number of HAR benchmarks and offers a variety of workload and network parameters. Some of these models also serve as benchmarks for existing FPGA accelerator-focused works. The ONNX files used for all the experimental results have been exported from [20] for *C3D*, *Slowonly*, and *X3D-M* models, and from [21] for *R(2+1)D-18* and *R(2+1)D-34*.

A. Experimental Results

1) *Ablation Study*: In this section, an ablation study was undertaken to assess the effect of several optimization strategies on the final performance of the proposed method. Having a baseline strategy in place, as well as introducing and investigating modifications and additions to it, has yielded significant insights and conclusions regarding the direction that should be followed for improving the optimization process.

For the ablation experiments, the R(2+1)D-18 model was used, although the findings apply to all supported models and devices.

Baseline Design Description:

The baseline optimisation strategy is defined as follows: the SA hyperparameters are configured by, $\tau_{start} = 10$, $\tau_{min} = 1 \times 10^{-6}$, and the cooling rate $\lambda = 0.99$, while a warm start is executed prior to the execution of the optimiser. These parameters remain constant throughout the ablation study. For the baseline experiments, the *Feature-Map Dimensions Reshaping*, *Coarse-grain Folding*, and *Fine-grain Folding* transformations are enabled while the use of runtime parameters and the fusion of activation layers to preceding layer are disabled.

Optimization Strategies:

- **Building Blocks Combination**: Incorporating the *Combination and Separation of Computation Nodes* transformation into the optimization approach improved the optimiser’s performance by $1.14\times$. This optimization is described in detail in Section V-C.
- **Fusion of activation functions into previous layer**: Upon revisiting the optimization findings and assessing each layer type of the model, it was found that the activation layers are typically memory bounded, which hinders the overall performance of the design. To overcome this constraint, the fusion optimization was introduced. Through the crossbars seen in Figure 5b, such layers are fused directly to the preceding ones (mostly convolution layers). Since convolution layers are mainly compute bounded, the fused activation layers also become compute bounded, suppressing the memory bounded limitation. The addition of this specific optimization has provided a $1.52\times$ boost in performance.
- **Runtime reconfiguration of layer parameters**: The introduction of runtime reconfigurability of computation nodes resulted in the most significant improvement boost. As described in Section III-C, in a non-runtime parameterizable node, in order to support a layer with different runtime feature-map dimensions the underlying hardware modules would need to add padding to match the compile-time dimensions. This affects the performance of the computation node greatly, as it must perform redundant operations. With the introduction of runtime parameterisable modules, as shown in Figure 3, the extra

TABLE V: Comparison with existing works on 3D CNN HAR models

Architecture	H. Fan [4] Hand-Tuned	H. Fan [5] Hand-Tuned	Z. Liu [8] Partial*	T. Teng [13] Hand-Tuned	J. Shen [9] Partial*	M. Sun [11] Partial*	H. Fan [6] Hand-Tuned	F. H. Khan [14] Hand-Tuned	HARFLOW3D Toolflow											
Model	C3D	C3D	C3D	C3D	C3D	C3D	R(2+1)D-18	E3D	I3D	C3D		Slowly		R(2+1)D-18		R(2+1)D-34		X3D-M		
GFLOPs [†]	38.61	38.61	38.61	38.61	38.61	38.61	8.52	6.1	110	38.61	38.61	54.81	8.52	12.91	12.91	12.91	12.91	6.97	6.97	
Accuracy (%)	79.87	81.99	83.2	83.2	83.2	83.2	88.66	85.17	95	83.2	83.2	94.54	88.66	92.27	92.27	92.27	92.27	96.52	96.52	
FPGA	ZC706	ZC706	VC709	VC707	VC709	VUS440	ZCU102	ZCU102	Intel SX660	VC709	ZCU102	VC709	ZCU102	VC709	ZCU102	VC709	ZCU102	VC709	ZCU102	VC709
Latency/clip	542.5	476.8	115.5	107.9	89.4	49.1	487	243	35.32	96	98.15	91.03	309.56	239.34	48.99	46.02	70.05	62.55	155.07	120.38
GOPs/s	71.17	80.97	334.28	357.83	431.87	786.35	79.28	35.06	172.8	1145.8	393.37	424.14	177.05	229.01	173.91	185.13	184.29	206.39	43.78	56.14
GOPs/DSP	0.079	0.089	0.092	0.127	0.119	0.273	0.031	0.013	0.102	0.318	0.156	0.117	0.07	0.063	0.069	0.051	0.073	0.057	0.017	0.015
Op/DSP/cycle	0.459	0.449	0.773	0.798	0.799	1.365	0.209	0.092	0.68	1.59	0.781	0.785	0.351	0.424	0.345	0.342	0.365	0.382	0.086	0.104
Frequency (MHz)	172	200	120	160	150	200	150	150	150	200	200	150	200	150	200	150	200	150	200	150
Precision	fp-16	BFP	fp-16	fp-8	fp-16	fp-16	fp-16	fp-16	float-32	fp-8	fp-16	fp-16	fp-16	fp-16	fp-16	fp-16	fp-16	fp-16	fp-16	fp-16
DSP (%)	90	86.6	99.8	96	42	53	48	48	93.3	100	96.51	97.77	62.38	69.11	96.94	97.91	95.67	97.25	53.52	89.61
BRAM (%)	86.6	88.1	26.6	25.3	52	30	100	100	-	79	71.93	63.33	78.56	81.05	69.13	64.62	69.24	54.52	56.14	78.67

* Proposed design supports multiple models (both 2D and 3D), although being tailored to the characteristics of specific 3D CNN models. [†] FLOPs are reported as MAC operations.

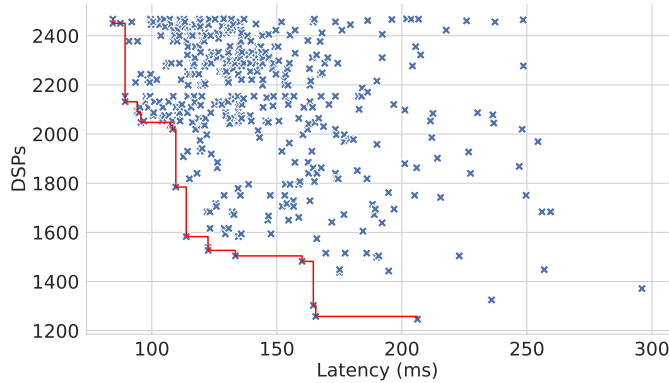


Fig. 7: Pareto front of DSP utilization against latency for R(2+1)D-34 model on a ZCU102 platform during SA

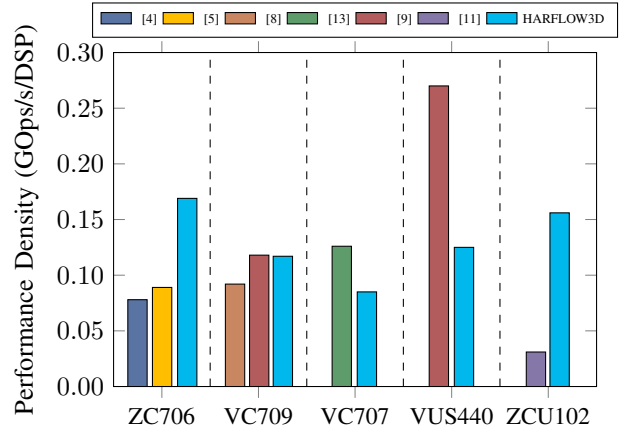


Fig. 8: Comparison on C3D - DSP Efficiency

cost of padding and all the redundant operations are omitted, resulting in a $18.21\times$ performance increase.

2) *Resources against Latency Comparison:* Figure 7 depicts the pareto-front between resources and latency. The DSP utilisation was chosen to represent resources, as the generated designs are typically limited by the number of available DSPs. The figure shows the optimiser’s ability to traverse the resource-latency design space, achieving fine-grain control over this trade-off. Indeed the optimiser is able to double performance along the pareto front at the cost of double the number of DSPs, exploring many design points in between.

B. Comparison to the state-of-the-art

Having evaluated the performance of the hardware and the effectiveness of the design space exploration method, the proposed work is positioned against existing accelerator works. Table V outlines the current space for FPGA-based 3D CNN acceleration, and how HARFLOW3D compares. Please note that all existing accelerators are the product of approaches that target only the specific workload, and thus are hand-tuned, where the produced designs from HARDFLOW3D are generated by a single toolflow. By parametrising all aspects of the hardware and automating the design space exploration, outstanding performance is achieved across a multitude of networks, targeting more than any existing accelerator.

Figure 8 provides a more extensive and direct comparison to existing works on C3D model. Since HARFLOW3D can target all of the platforms that previous studies have targeted for C3D, results for each of them have been collected for a

direct comparison with all of the existing works. The results are evaluated in DSP efficiency, GOPs normalised over the device’s DSPs. As shown in Figure 8, HARFLOW3D achieves $1.89\times$ increased DSP efficiency on ZC706 compared to H. Fan [5]. On ZCU102, $5.03\times$ greater results are achieved than M. Sun [11]. On VC709 HARFLOW3D achieves $1.27\times$ better DSP efficiency than Z. Liu [8], whereas obtains nearly equal performance being only $1.008\times$ off compared to J. Shen [9]. Compared to T. Teng [13] on VC707, the DSP efficiency is $1.48\times$ lesser, however the comparison cannot be considered direct since the specific design uses fixed-point 8 arithmetic precision. In comparison to J. Shen [9] on VUS440, the DSP efficiency is inferior by $2.16\times$.

TABLE VI: Comparison against GPU on C3D

	GPU	FPGA (HARFLOW3D)
Platform	RTX 3090	ZCU106
Clock Frequency	1.7 GHz	200 MHz
Precision	32-bit float	16-bit fixed
Latency/clip (ms)	5.28	182.81
Power (W)	234.1	9.44
Energy/clip (J)	1.24	1.72

Table VI outlines a comparison between an RTX 3090, a server-grade cutting-edge GPU, and a ZCU106, a mid-range FPGA board. Despite the difference in scale between the two devices, the results for energy/clip demonstrate the efficiency of the HARFLOW3D toolflow.

VIII. CONCLUSION

This paper presents HARFLOW3D, the first 3D CNN FPGA toolflow for 3D CNNs that supports a wide range of models and devices. A series of transformations during optimization, in conjunction with a novel hardware implementation supporting runtime parametrization of the hardware nodes, enabled comparable and even greater performance in comparison to prior hand-tuned designs. Future directions may be the support of additional 3D CNN models with different backbones, such as Inception-like architectures (I3D). Furthermore, expansion into domains other than HAR, such as 3D semantic segmentation, medical imaging (CT and MRI scans), 3D object detection from point clouds and Transformer-based HAR models will be investigated.

ACKNOWLEDGMENT

For the purpose of open access, the authors have applied a Creative Commons Attribution (CC BY) license to any Accepted Manuscript version arising.

REFERENCES

- [1] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, "A Survey of Accelerator Architectures for Deep Neural Networks," pp. 264–274, 3 2020.
- [2] S. I. Venieris, A. Kouris, and C. S. Bouganis, "Toolflows for mapping convolutional neural networks on FPGAs: A survey and future directions," *ACM Computing Surveys*, vol. 51, no. 3, 2018.
- [3] S. Ji, W. Xu, M. Yang, and K. Yu, "3D Convolutional neural networks for human action recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 221–231, 2013.
- [4] H. Fan, X. Niu, Q. Liu, and W. Luk, "F-C3D: FPGA-based 3-dimensional convolutional neural network," in *2017 27th International Conference on Field Programmable Logic and Applications, FPL 2017*, 2017, pp. 2–5.
- [5] H. Fan, H. C. Ng, S. Liu, Z. Que, X. Niu, and W. Luk, "Reconfigurable acceleration of 3D-CNNs for human action recognition with block floating-point representation," in *Proceedings - 2018 International Conference on Field-Programmable Logic and Applications, FPL 2018*, no. Section III, 2018, pp. 287–294.
- [6] H. Fan, C. Luo, C. Zeng, M. Ferianc, Z. Que, S. Liu, X. Niu, and W. Luk, "F-E3D: FPGA-based acceleration of an efficient 3D convolutional neural network for human action recognition," in *Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors*, vol. 2019-July, 2019, pp. 1–8.
- [7] K. Soomro, A. R. Zamir, and M. Shah, "UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild," Tech. Rep., 12 2012. [Online]. Available: <http://arxiv.org/abs/1212.0402>
- [8] Z. Liu, P. Chow, J. Xu, J. Jiang, Y. Dou, and J. Zhou, "A uniform architecture design for accelerating 2d and 3d cnns on fpgas," *Electronics (Switzerland)*, vol. 8, no. 1, 1 2019.
- [9] J. Shen, Y. Huang, Z. Wang, Y. Qiao, M. Wen, and C. Zhang, "Towards a uniform template-based architecture for accelerating 2d and 3D CNNs on FPGA," in *FPGA 2018 - Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, vol. 2018-Febru, 2018, pp. 97–106.
- [10] S. Winograd, *Arithmetic Complexity of Computations*. Society for Industrial and Applied Mathematics, 1 1980.
- [11] M. Sun, P. Zhao, M. Gungor, M. Pedram, M. Leeser, and X. Lin, "3D CNN acceleration on FPGA using hardware-aware pruning," in *Proceedings - Design Automation Conference*, vol. 2020-July. IEEE, 2020.
- [12] D. Tran, H. Wang, L. Torresani, J. Ray, Y. Lecun, and M. Paluri, "A Closer Look at Spatiotemporal Convolutions for Action Recognition," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 6450–6459, 2018.
- [13] T. Teng, J. Xi, Z. Letian, W. Xiaotian, W. Jie, and W. Wei, "Exploration of Memory Access Optimization for FPGA-based 3D CNN Accelerator," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. EDA Consortium, 2020, pp. 1650–1655. [Online]. Available: <https://ieeexplore.ieee.org/document/9116376>
- [14] F. H. Khan, M. A. Pasha, and S. Masud, "Towards designing a hardware accelerator for 3D convolutional neural networks," *Computers and Electrical Engineering*, vol. 105, p. 108489, 1 2023.
- [15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," Tech. Rep. [Online]. Available: <http://code.google.com/p/cuda-convnet/>
- [16] C. Feichtenhofer, "X3D: Expanding Architectures for Efficient Video Recognition," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Feichtenhofer Christoph, Ed., 2020, pp. 200–210.
- [17] C. Feichtenhofer, H. Fan, J. Malik, and K. He, "Slowfast networks for video recognition," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2019-October, pp. 6201–6210, 2019.
- [18] S. I. Venieris and C. S. Bouganis, "FpgaConvNet: Mapping Regular and Irregular Convolutional Neural Networks on FPGAs," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 2, pp. 326–342, 2019.
- [19] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A framework for fast, scalable binarized neural network inference," *FPGA 2017 - Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, no. February, pp. 65–74, 2017.
- [20] MMAAction2, "OpenMMLab's Next Generation Video Understanding Toolbox and Benchmark," <https://github.com/open-mmlab/mmaaction2>, 2020. [Online]. Available: <https://github.com/open-mmlab/mmaaction2>
- [21] K. Hara, H. Kataoka, and Y. Satoh, "Can Spatiotemporal 3D CNNs Retrace the History of 2D CNNs and ImageNet?" in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, 12 2018, pp. 6546–6555.