M5: Multi-modal Multi-task Model Mapping on Multi-FPGA with Accelerator Configuration Search

Akshay Karkal Kamath[1], Stefan Abi-Karam[1,2], Ashwin Bhat[1] and Cong Hao[1]
[1]School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA
[2]Georgia Tech Research Institute, Atlanta, GA, USA
{akshay.k2, stefanabikaram, ashwinbhat, callie.hao}@gatech.edu

***Abstract*—Recent machine learning (ML) models have advanced from single-modality single-task to multi-modality multi-task (MMMT). MMMT models typically have multiple backbones of different sizes along with complicated connections, exposing great challenges for hardware deployment. For scalable and energy-efficient implementations, multi-FPGA systems are emerging as the ideal design choices. However, finding the optimal solutions for mapping MMMT models onto multiple FPGAs is non-trivial. Existing mapping algorithms focus on either streamlined linear deep neural network architectures or only the critical path of simple heterogeneous models. Direct extensions of these algorithms for MMMT models lead to sub-optimal solutions. To address these shortcomings, we propose M5, a novel MMMT Model Mapping framework for Multi-FPGA platforms. In addition to handling multiple modalities present in the models, M5 can flexibly explore accelerator configurations and possible resource sharing opportunities to significantly improve the system performance. For various computation-heavy MMMT models, experiment results demonstrate that M5 can remarkably outperform existing mapping methods and lead to an average reduction of 35%, 62%, and 70% in the number of low-end, mid-end, and high-end FPGAs required to achieve the same throughput, respectively. Code is publicly available[1].**

***Index Terms*—Multi-FPGA, DNN Model Mapping Framework**

I. Introduction

Multi-FPGA acceleration has become imperative for state-of-the-art machine learning (ML) models due to the growing sizes of these models as well as their inputs. The prohibitively large computation complexity makes their deployment on a single device infeasible. In addition, hardware development tools are becoming the bottlenecks, requiring days or even weeks to synthesize large-scale designs, drastically increasing the development cycle and the time-to-market. Thus, multi-FPGA is appealing for energy-efficient and scalable ML system.

Related work has discussed ways for mapping streamlined ML models to multi-FPGA systems [1]–[7]. For instance, [1] and [7] propose dynamic-programming-based model partitioning to maximize multi-FPGA throughput. CNN-on-AWS [2] formulates the model partitioning as a mixed-integer non-linear programming problem, while [6] further extends to dynamic re-programming of FPGAs at runtime to reduce power. However, ML models have been rapidly evolving from single-modality single-task to multi-modality multi-task (**MMMT**) [8]–[13]. For example, in AR/VR applications, image, gesture, and speech are learned together [14]; in emotion recognition, MoCap [15] learns from speech, text, and motions jointly.

Despite the success of existing mapping algorithms on multi-FPGA systems [1], [2], [4], [5], [7], there are significant **limitations** for their application towards MMMT models. First, most multi-FPGA mapping algorithms focus on *streamlined and linear DNNs* with a single backbone and no branches. However, MMMT models [8]–[10], [16] usually have *multiple backbones of different sizes and complicated connections* (e.g., cross-talk between backbones). Fig. 1 illustrates the situation where a direct extension of existing mapping algorithms produces a sub-optimal solution (4 FPGAs), whereas a better solution (3 FPGAs) exists. Second, H2H [9] is the only mapping algorithm that uses an iterative heuristic to map MMMT models to multi-FPGA. However, H2H ignores the configurability of the FPGAs by assuming a fixed accelerator architecture, and thus does not explore *optimal resource sharing and allocation* when multiple layers are being mapped to the same FPGA. As we will demonstrate in Sec. II, failing to do so can also result in sub-optimal mapping. Third, most previous works focus on optimizing either the end-to-end latency or the throughput for a fixed number of FPGAs. Consequently, optimization of energy consumption is under-explored. In applications such as video processing, the energy consumption is expected to be minimized in addition to meeting the performance objectives. This can be achieved by reducing the number of FPGAs needed for mapping.

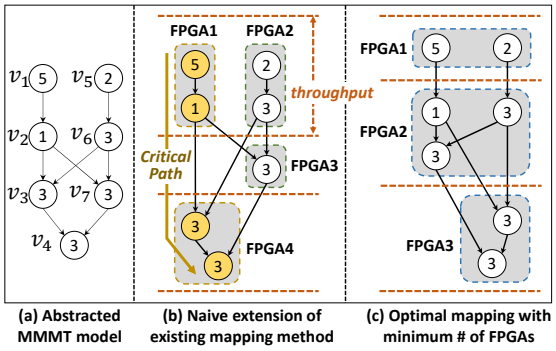

Fig. 1: For MMMT models, naive extension of existing mapping algorithms can result in sub-optimal mapping.

In this work, we address the limitations of existing algorithms by proposing a novel mapping framework, named **M5**, specifically targeted for MMMT models on Multi-FPGA. M5 not only handles multiple backbones with complicated connections, but also flexibly explores accelerator configurations and possible resource sharing opportunities to greatly boost the mapping quality while decreasing the resources (number of FPGAs) required to minimize the system power consumption. We summarize our contributions as follows:

- **M5** is the *first* mapping algorithm for MMMT models on multi-FPGA which explores flexible accelerator configurations and possible resource sharing among layers. It is both computation- and communication-aware and aims to minimize the number of FPGAs for given throughput objective and resource constraints.
- Since **M5** tackles MMMT models, it has a large solution space compared to streamlined DNNs. We propose a two-stage algorithm: (1) generating *uniformly-sampled* topological sorts within the solution space; (2) for each topological sort, using *dynamic programming* to generate load-balanced mappings and searching for the best accelerator configuration for each model layer.
- Our proposed novel Uniform Sampling (US) method introduces significantly *higher randomness* that covers more design points compared to traditional topological order generation methods, enabling a much broader and efficient design space exploration.
- **M5** demonstrates remarkable improvements over existing mapping algorithms by reducing the number of low-end, mid-end, and high-end FPGAs required, on an average, by 35%, 62%, and 70% respectively for popular computation-heavy MMMT models.

[1]https://github.com/akshay-k2/m5

978-3-9819263-7-8/DATE23/© 2023 EDAA

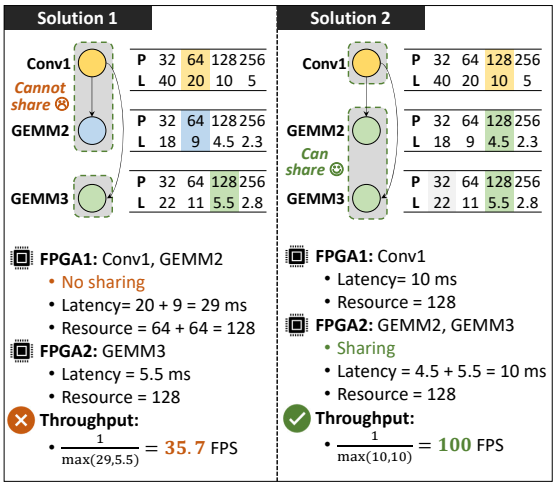

Fig. 2: Considering accelerator configuration and resource sharing leads to higher chance of obtaining better solutions.

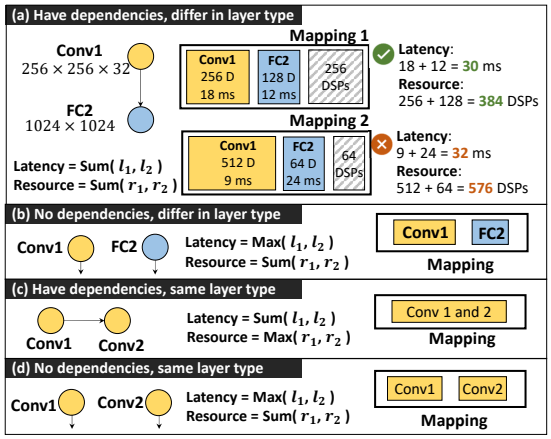

Fig. 3: Based on the layer type and layer dependency, the latency and resource formulations are different. Meanwhile, different accelerator configurations (e.g., parallelism) result in different mapping quality.

II. Motivating Examples

Fig. 1 demonstrates that direct extensions of existing mappers designed for linear DNN architectures may result in sub-optimal solutions. **Fig. 2** demonstrates that resource sharing can be beneficial and must not be ignored. In this example, we enumerate possible accelerator configurations with **P** representing parallelism (i.e., the number of DSPs) and **L** denoting latency. In example Solution 1, assuming a video processing application, *Conv1* and *GEMM2* are mapped to one FPGA but do not share resources due to different layer types, and the throughput is 35.7 FPS (frames per second). In contrast, Solution 2 maps *GEMM2* and *GEMM3* to one FPGA with resource sharing, achieving a higher throughput (100 FPS) than Solution 1 for the same resource usage. *This motivating example explains the necessity of resource sharing* while partitioning the MMMT model. **Fig. 3** demonstrates the benefit of accelerator configuration. We categorize resource sharing into four situations (a to d), depending on whether neighboring layers are of the same type or have data dependency. In situation (a), for instance, two adjacent layers are of different types but have dependency. Mapping solution 1 achieves a latency of 30 ms with 384 DSPs, while mapping solution 2 achieves a latency of 32 ms with 576 DSPs, evidently worse than solution 1. *This example motivates the necessity of accelerator configuration.*

III. Problem Formulation

Model Representation. An MMMT model can be represented as a Directed Acyclic Graph (DAG), $G = (V, E)$, with one or more sources (inputs) and one or more sinks (outputs). Each node $v \in V$ represents a layer of the model, associated with attributes including layer type (e.g., convolution, fully connected, LSTM, attention, etc.) and computation cost (e.g., number of MACs). Each edge $e \in E$ represents a data dependency between two layers and is associated with a weight quantifying the cost of inter-layer data transfer (communication overhead).

Target Platform: Chained FPGAs. We target a linearly chained FPGA system and utilize *throughput* as the performance indicator as opposed to end-to-end latency, since for real-world applications such as real-time video processing, throughput across consecutive video frames is more important than the latency of processing an individual frame. For a system with N FPGAs, the overall throughput T_{sys} is decided by the FPGA with the highest latency: $T_{\text{sys}} = \frac{1}{\max(L_1, L_2, \cdots, L_N)}$, where L_i is the $i\text{-}th$ FPGA latency.

Computation and Communication Costs. We use the number of MAC operations to describe a layer's computation cost. This cost is decided by the model parameters and the input size (e.g., HD images). The communication cost associated with edges $W(e_{i,j})$, as described in [2], is the ratio of the size of i^{th} layer output D_i and the inter-FPGA bandwidth BW. In this work, we assume a homogeneous cluster of FPGAs with identical inter-FPGA bandwidth.

Mapping Model Layers on FPGAs. For an MMMT model with $|V|$ layers, the goal is to find cost-balanced mappings on N FPGAs, where N is to be minimized as long as the target objective (e.g., throughput) is met. Each FPGA can contain one or more accelerator instances with different types and parallelism factors, and each accelerator can execute one or more model layers. Specifically, for each layer, the following mapping variables must be decided -

1) FPGA allocation: which FPGA the layer is mapped to;
2) Accelerator configuration: parallelism factor of its accelerator;
3) Resource sharing: can its accelerator be shared with other layers.

Notably, very limited existing works explore accelerator configuration, while no existing work discusses resource sharing to the best of our knowledge. In this work, we explicitly explore the situation that two or more layers share the same accelerator when they are of the same layer type or have the same computation pattern.

Mapping Objectives and Constraints The objective of M5 optimization problem is to *minimize the total number of FPGAs under throughput and resource constraints*. We use the number of DSPs as the resource constraint for each FPGA and assume that each DSP performs 1 MAC operation per cycle. Throughput constraint is described by its inverse, the Initiation Interval (II). As long as the layer with highest computation cost has a latency less than the target II, the throughput constraint is satisfied.

Layer modeling for accelerator configuration. We adopt the layer performance model from the most recent work, H2H [9], to model layer latency l_i and resource utilization r_i. Since the number of DSPs is usually linearly proportional to the accelerator parallelism factor and is the dominating resource, we let r_i denote the DSP usage and use it interchangeably with parallelism. Let P_i be the accelerator performance model for layer i (e.g., P_i differs from convolution to fully connected) so that $l_i = P_i(r_i)$. An accelerator can have multiple choices of parallelism; we enumerate a fixed set of parallelism factors (divisible by power of two) for each layer, and each factor corresponds to a determined layer latency. Consequently, i^{th} layer's performance and resource model is a set of possibilities, denoted as $\{< r_i, l_i > | r_i \in \{32, 64, 96, 128, \cdots\}, l_i = P_i(r_i)\}$.

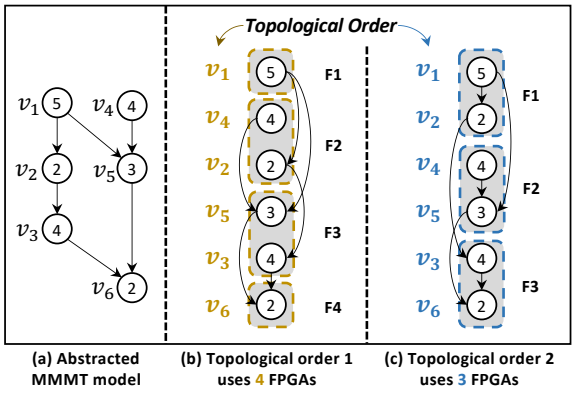

Fig. 4: The choice of topological order impacts mapping results. Finding *all* topological orders is #P-complete problem.

FPGA modeling for resource sharing. As discussed in Sec. II, since we explore layer sharing on an FPGA, there are four scenarios for layers i and j, as shown in Fig. 3.

1) Two layers have data dependency and differ in layer type, as in Fig. 3(a). In this scenario, two layers cannot share the same accelerator. Therefore, the FPGA latency is $LAT = \text{Sum}(l_i, l_j)$ and resource is $RES = \text{Sum}(r_i, r_j)$.
2) Two layers are not dependent and differ in layer type, as in Fig. 3(b). It is beneficial to use two separate accelerators which run in parallel. Thus, $LAT = \text{Max}(l_i, l_j)$, $RES = \text{Sum}(r_i, r_j)$.
3) Two layers have data dependency and are of the same type, as in Fig. 3(c). Because of data dependency, the two layers cannot be executed in parallel, so it is beneficial for them to share resource. Thus, $LAT = \text{Sum}(l_i, l_j)$, $RES = \text{Max}(r_i, r_j)$.
4) Two layers are not dependent but are of the same type. In this scenario, the two layers can either share the same accelerator or use two separate ones. For simplicity, we consider the latter and thus $LAT = \text{Max}(l_i, l_j)$, $RES = \text{Sum}(r_i, r_j)$.

IV. Proposed M5 Algorithm

The primary input to our M5 algorithm is an MMMT model specification, expressed as a DAG. Since an MMMT model has multiple backbones, branches, and complicated connections, together with accelerator configuration and resource sharing, the design space is much larger compared to streamlined DNNs. To efficiently explore the design space, we propose a two-stage algorithm:

1) Find valid topological sorts of the given DAG, which serve as initial solutions for mapping (Sec. IV-A);
2) Based on each topological sort, determine the optimal mapping, accelerator configuration, and resource sharing (Sec. IV-B).

Note that the first step samples initial solutions from the entire design space, and thus its sampling efficiency is crucial to the algorithm quality: *how well can the algorithm visit as many diverse design points as possible by maximizing sampling randomness?* In the following sections, we first discuss our proposed sampling algorithm which significantly outperforms revised deterministic algorithms with much higher randomness. Then, we discuss dynamic programming based mapping given each sampled initial solution.

A. Generating Topological Orderings of MMMT Model Graph

A **topological sort** or topological ordering of a directed graph is a linear ordering of its vertices, such that for every directed edge (u, v) from vertex u to vertex v, u comes before v in the ordering. Given a DAG for an MMMT model, finding a proper topological sort can serve as an *initial solution* for determining a valid mapping together with accelerator configuration and resource sharing. Fig. 4(a) is an example of a DAG; (b) and (c) show two of its topological sorts.

Finding a good topological sort, however, is non-trivial. In contrast to streamlined ML models which have only one topological sort, MMMT models have multiple topological orders. Determining an optimum mapping solution can be sensitive to the topological order chosen. This is demonstrated in Fig. 4 using a simplified example with fixed accelerator configurations, where the node values refer to the layer resource usage. Assuming that each FPGA capacity is 7 units, the first topological order results in 4 FPGAs, while the second one results in only 3 FPGAs. However, enumerating all possible topological orders is equivalent to finding all the linear extensions for a given partially ordered set, which is a *#P-complete problem* [17] and is as least as hard as NP-complete problems [18]. For instance, the total number of topological sorts for an MMMT model with just 20 layers can be in billions and require several days or even weeks for enumeration. This effectively highlights the fundamental difference between MMMT and streamline models. Combined with accelerator configuration search and resource sharing exploration, the complexity is further elevated.

Sampling Topological Sorts. As a brute-force search of the entire space of valid topological sorts is intractable, we instead focus on *how to randomly sample the space of all valid topological sorts in an efficient and uniform manner*. Through this approach of sampling, a fixed number of sampled sorts can be evaluated and the best mapping can be presented as the final mapping. To achieve this goal, we first discuss two sampling algorithms that modify traditional deterministic methods to non-deterministic variations in order to expand the solution space (as shown in Fig. 5):

1) Kahn's algorithm based [19]: forward sort and reverse sort.
2) List-scheduling based: As-Soon-As-Possible (ASAP) and As-Late-As-Possible (ALAP).

(1) *Kahn's algorithm based*. Typical implementations of topological sorting algorithms are variations of Kahn's algorithm [19]. It iteratively finds vertices with no incoming edges and removes all outgoing edges from these vertices, resulting in an ordering with all the dependency constraints satisfied. To make this algorithm non-deterministic, we introduce randomness when there are multiple nodes that qualify for selection at any given stage. As shown in Fig. 5(a), first, we randomize which of the zero in-degree nodes will be selected for removal and inserted into the sort list at each iteration of the algorithm. We refer to this variation as the "forward sort" algorithm. Second, we modify the algorithm to build the topological sort in reverse order by removing nodes with zero out-degree and adding them to the front of the sort list. We refer to this variation as the "reverse sort" algorithm.

(2) *List-scheduling based*. Another approach is to use the classic list scheduling algorithms. We modify the ASAP and ALAP algorithms to make them non-deterministic by introducing randomness, as shown in Fig. 5(b). ASAP and ALAP organize graph nodes into different levels. The nodes within the same level can be considered a contiguous block of nodes in the topological sort. The ordering of the nodes within a block can be permuted since nodes at the same level do not have any dependence on each other. The blocks of nodes corresponding to each level are then concatenated together in level-order to produce the final topological sort. To make this algorithm non-deterministic, we randomize the permutation of node blocks at each level.

The drawback of these two algorithms, however, is that their "randomness" is limited, i.e., the possible randomness is based on a deterministic solution, which largely constrains the solution space. To address this limitation, we propose a *fully randomized* Uniform Sampling (US) algorithm, which achieves the best sampling efficiency by covering the largest solution space.

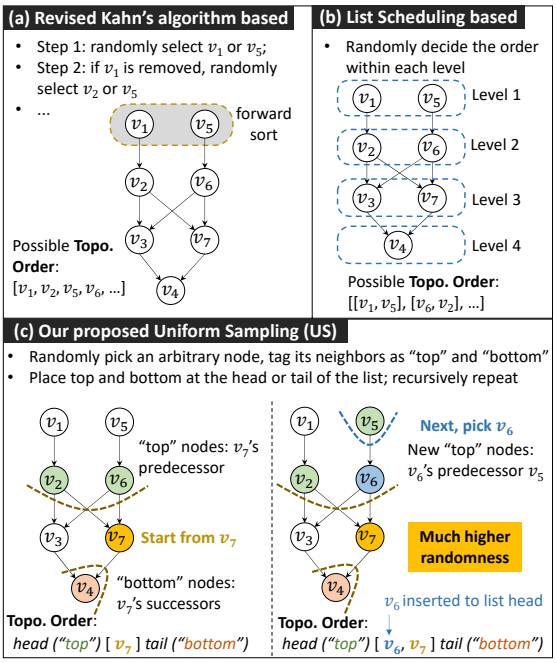

Fig. 5: Examples of sampling based on Kahn's algorithm (a) and list-scheduling (b) along with our proposed Uniform Sampling (c).

Uniform Random Start Topological Sort. To increase the randomness greatly, we propose a novel sorting algorithm, described by Algorithm 1, that builds a topological sort from any arbitrary starting node in the DAG. This approach combines ideas from the forward and reverse versions of Kahn's algorithm to build the sort list in two directions as shown in Fig. 5(c). First, a starting node is randomly selected, removed from the graph, and added to the sort list (line 10-11). All predecessors of the starting node are tagged as "top" while all the successors are tagged as "bottom". The algorithm then iteratively removes nodes tagged as "top" or "bottom" and adds them to the *head* or *tail* of the sort list respectively. When these nodes are removed, they also tag their predecessors and successors as "top" or "bottom" nodes without overwriting any existing tags. During node selection, a filtering step is needed to remove candidate nodes who also have dependence on other candidate nodes (line 15-16). To make this algorithm non-deterministic, the "top" or "bottom" node for removal at each iteration is randomly selected out of all the possible candidate nodes at that step. We randomize the sort generation further by also randomly selecting the start node that is to be used for the algorithm. We refer to this implementation as "uniform random start" sort.

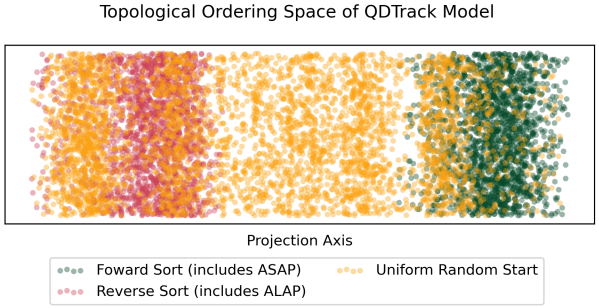

Fig. 6: Projection of sampled topological orders by different sampling algorithms for QDTrack [20] model graph.

Our proposed algorithm can generate topological orders *uniformly* in the topological sort space, covering much larger solution space

Algorithm 1: Uniform Random Start Topological Sort

Input : Model Graph G
Output: Topological Sort List S
1 **Function** `TagConnectedNodes(`n`)`**:**
2 **for** *child in n.successors if child.tag != "top"* **do**
3 child.tag ← "bottom"
4 **for** *parent in n.predecessors if child.tag != "bottom"* **do**
5 child.tag ← "top"
6 **Function** `RemoveAndAddToSort(`n, G, S`)`**:**
7 S ← Append n, to the left **if** $n.tag$ == "top" **else** to the right
8 `TagConnectedNodes(`n`)`
9 $S \leftarrow []$
10 $startNode$ ← Randomly choose a node in G
11 `RemoveAndAddToSort(`$startNode$`)`
12 **while** *len(S) < len(G.nodes)* **do**
13 $topNodes$ ← All nodes with "top" tag
14 $bottomNodes$ ← All nodes with "bottom" tag
15 $topFiltered$ ← `FilterTopNodes(`G, $topNodes$`)`
16 $bottomFiltered$ ← `FilterBottomNodes(`G, $bottomNodes$`)`
17 $possibleNodes$ ← $topFiltered$ and $bottomFiltered$
18 $randomNode$ ← Randomly choose a node in $possibleNodes$
19 `RemoveAndAddToSort(`$randomNode$, G, S`)`
20 **return** S

than the other two methods. Fig. 6 visualizes the differences between our proposed uniform sampling and the deterministic-based sampling. First, we find that the traditional "forward sort" and "reverse sort" algorithms are biased and generate sorts that are clustered together in the topological sort space. This can be explained based on the branching choice effect when building the sorts using Kahn's algorithm. The initial set of candidate nodes is the same for each run of the algorithm leading to similar starting node sequences. After the first few nodes, there is a combinatorial branching effect for the order in which the rest of the nodes are selected. We also find that generating sorts using ASAP and ALAP are subsets of forward and reverse sort algorithms.

With random start nodes, we reduce the bias of having predictable sequences within a sort across all sampled sorts. We justify the generality of "random start" algorithm based on the projections of topological sorts from various algorithms as shown in Fig. 6. Our proposed approach covers the space of all other topological sorts while creating additional sorts in spaces between the traditional algorithms.

B. Finding the Optimal Mapping

After uniformly sampling a large enough set of topological sorts of the input DAG, for each sort, we explore its optimal mapping. First, we generate k partitions of the topological order, $1 \leq k \leq N$, where N is the upper bound of the total number of FPGAs. For different k, we evaluate each to find the smallest k while satisfying the given resource, throughput and bandwidth constraints.

We adopt the dynamic-programming-based polynomial-time load-balancing algorithm from [7] (after correcting a minor error in the original equation – changing the limit from $r = 1$ to $r = k - 1$) for mapping $|V|$ layers on to 1, 2, ..., N FPGAs based on the computation cost as shown in Eq. 1:

$$C_{j,k} = \begin{cases} \sum_{l=1}^{j} C_l & \text{if } k = 1 \\ \min_{r=k-1}^{j-1}(max(C_{r,k-1}, \sum_{l=r+1}^{j} C_l)) & \text{if } 1 < k \leq N \end{cases} \quad (1)$$

In Eq. 1, C_i represents the computation cost of i^{th} layer where $i \in [1, N]$. $C_{j,k}$, on the other hand, represents the total computation cost (computation load) of the first k FPGAs when mapping layers from 1 to j. For a single FPGA in the cluster, the computation load is obtained by adding individual computation costs of the first j layers.

Algorithm 2: Optimal Mapping using Dynamic Programming

Input : G, $Topo\_Order$, DSP, II, T_{CLK}, BW, AC_En
Output: Valid mapping with minimal partitions

Function `DPsolver`$(G(V,E),\ Topo\_Order)$**:**
  $N \leftarrow |V|$ `// Number of nodes in G`
  $Mappings \leftarrow$ Map G to $\{1, 2, ..., N\}$ FPGAs as per Eq. 1
  return $Mappings$

Function `CriticalPartition`$(Mapping)$**:**
  Find $Partition$ of $Mapping$ with highest cumulative cost
  return $Partition$

Function `findValidConfigs`$(criticalPart)$**:**
  $validConfigs \leftarrow []$
  for *every node* $\in criticalPart$ **do**
    for *every resource-latency pair of node* **do**
      if *usage does not exceed DSP-II budget* **then**
        Update $validConfigs$ with this pair
  if $validConfigs$ *is not empty* **then**
    return $True$
  return $False$

$N \leftarrow |V|$ `// Max. number of partitions`
$allMappings[1:N] \leftarrow$ `DPsolver`$(G, Topo\_Order)$
$bestMapping \leftarrow allMappings[N]$
$N \leftarrow |V| - 1$
while $N \neq 0$ **do**
  $currentMap \leftarrow allMappings[N]$
  $criticalPart \leftarrow$ `CriticalPartition`$(currentMap)$
  $configExists \leftarrow$ `findValidConfigs`$(criticalPart)$
  if $configExists$ **then**
    $bestMapping \leftarrow allMappings[N]$
  $N \leftarrow N - 1$
return $bestMapping$

For a cluster with multiple FPGAs, the computation load is calculated based on assigning the first r layers to $k-1$ FPGAs and the remaining layers ($r+1$ to j) to the last FPGA. The former component acts as the sub-problem in this dynamic programming algorithm.

Once all the mappings are generated, we evaluate the validity of each mapping to check if it satisfies the specified constraints and objectives starting with the mapping having the highest number of partitions. Since the overall system throughput is limited by the latency of the partition with the highest computation cost, termed critical partition, as long as there is at least one resource-latency combination (described in Sec. III) in this partition that does not exceed the DSP, II and BW budgets, the mapping can be considered valid as described in Algorithm 2. Moreover, since the computation latency is much higher than communication delays [21], we cross-check the validity by adding the maximum of incoming edge weights to the critical partition latency. We assume the usage of double buffer to hide the communication delays within a partition.

V. Experiments

A. Experiment setup and baselines.

MMMT Models. To evaluate our mapping framework, we experiment using six MMMT models as shown in Table I, following the same setting from H2H [9]. The dominating layer types are convolution, fully connected, and LSTM. We also include near-streamlined ResNet50 model with SD inputs (480×640) for reference.

Experiment Settings. For evaluation, we consider three Xilinx FPGA boards: Ultra96v2, KC705 and ZCU104 with 360, 840 and 1728 DSP units respectively. The clock frequency is set to 125 MHz. The target throughput constraint is 30 FPS as is typical of any video processing application. We assume 16-bit fixed point as the data precision but M5 can be easily extended to model other quantized precision types. For instance, with 8-bit data, two MAC operations can be executed by a single DSP unit in one clock cycle. The input image size varies from HD to SD such that the bottleneck layer of the model, when mapped to a single FPGA, achieves the target throughput requirement (so that we can guarantee there is a solution). Our proposed algorithms and the baselines are implemented using Python and executed on Intel i9 8-core CPU with 16 GB memory.

Domain	Model	GMAC
Image Classification	ResNet-50 (SD) [22]	25.04
Augmented Reality	VLocNet [16]	59.05
Face Recognition	CASUA-SURF [23]	7.10
Sentiment Analysis	VFS [11]	21.46
Multiple Object Tracking	QDTrack (SD) [20]	57.63
Face Recognition	FaceBagNet [24]	9.68
Emotion Recognition	MoCap [15]	0.68

TABLE I: MMMT Models used for evaluation.

Baselines. As there is no existing algorithm that directly solves the problem discussed in this work, we make the best effort to extend the existing approaches to be applicable and conduct ablation study to demonstrate the effectiveness of our proposed M5 framework. In particular, we evaluate the following algorithms:

1) **Baseline-1**. Direct extension of existing mapping algorithm [7] *without* accelerator configuration and resource sharing. The algorithm in [7] prioritizes layers (vertices) on the critical path (**CP**) and adopts dynamic programming; the algorithm then iteratively maps the critical paths of sub-graphs obtained by removing the nodes in the previously processed critical paths. Although H2H [9] is the only known work targeting MMMT model mapping, its algorithm centers at optimizing end-to-end latency for a fixed number of FPGAs and does not consider accelerator configuration or resource sharing.
2) **Baseline-2**. On top of Baseline-1, we traverse all possible accelerator configurations and explore resource sharing possibilities as discussed in Sec. III.
3) **Baseline-3**. We apply the classic list scheduling [25] (**ALAP** and **ASAP**) with the exploration for accelerator configuration and resource sharing.
4) **Ours**. Our proposed framework based on evaluation of uniformly sampled (**US**) topological orders as discussed in Sec. IV with accelerator configuration and resource sharing.

Overall Mapping Results. Fig. 7 shows the overall mapping results on Ultra96v2, KC705 and ZCU104 respectively across all the MMMT models considered in this work with baselines and our proposed framework. For a computation-heavy model like QDTrack, we can observe that the number of mid-end ZCU104 FPGA boards required reduces by around 85% and around 78% for VLocNet. Fig. 8 illustrates the actual mapping of VLocNet model on 11 ZCU104 FPGAs using our approach involving accelerator configuration and resource sharing. In contrast, the same model requires over 30 such FPGAs with any of the baselines. Furthermore, for every model, our uniform sampling approach results in mappings that are as good as or better than those obtained from the classic list scheduling algorithms. This affirms the validity of our sampling approach in efficiently exploring the topological ordering search space.

Overall, compared to the baseline, M5 reduces the number of low-end, mid-end and high-end FPGAs required for mapping these MMMT models, on an average, by 35%, 62% and 70% respectively. These results conclusively highlight the efficacy of our approach of uniformly sampling the topological ordering space and the advantages of using accelerator configuration and resource sharing in drastically

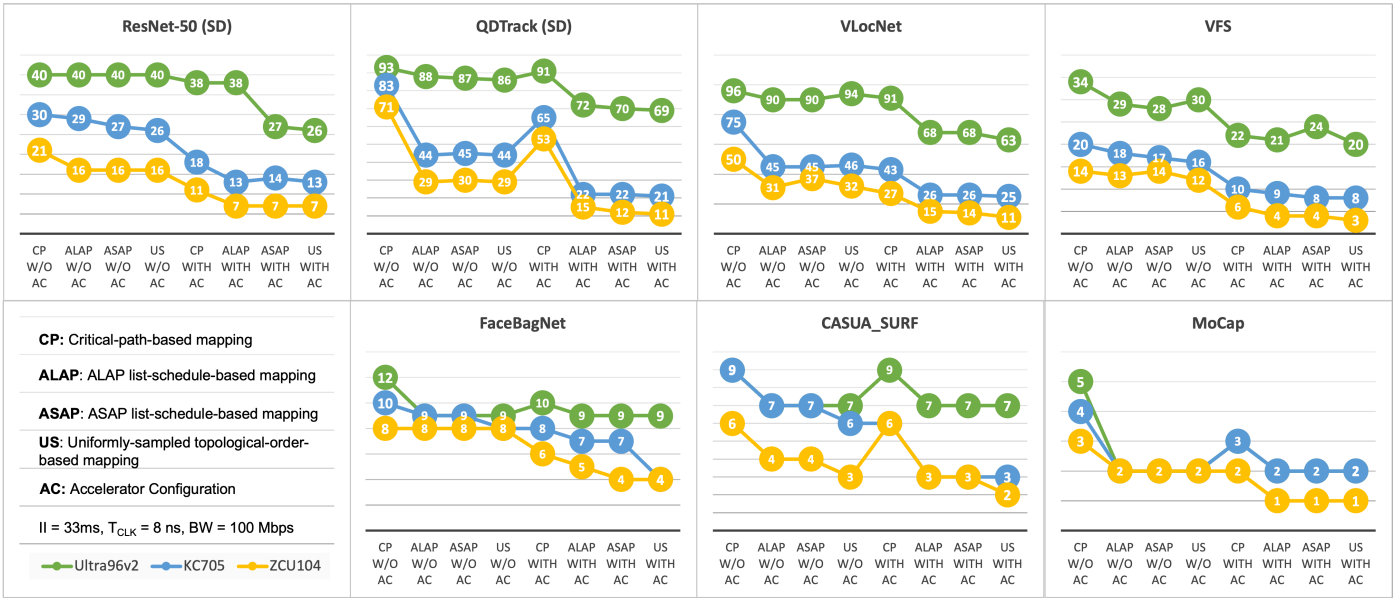

Fig. 7: M5 results showing the number of partitions (i.e. FPGAs required) in the optimal mapping for each configuration

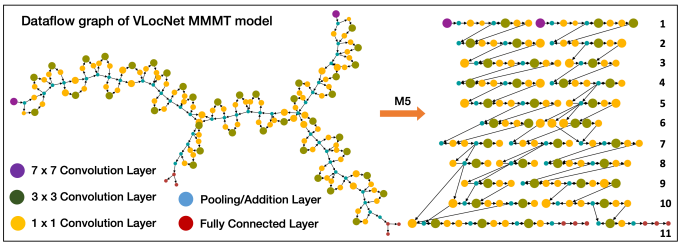

Fig. 8: Optimal mapping of VLocNet on 11 ZCU104 FPGAs.

reducing the number of FPGAs required for model implementation, thereby demonstrating a highly energy-efficient alternative compared to existing baseline approaches.

VI. Conclusion

In this work, we tackle the problem of mapping Multi-Modal Multi-Task (MMMT) deep learning models on a multi-FPGA system using a novel framework, named M5. Through comprehensive experiments, we demonstrate that the existing mapping algorithms, such as critical-path based and list-schedule based mappings, lead to sub-optimal solutions when applied to MMMT models. In our framework, we generate uniformly-sampled topological orderings of the given task graph to efficiently explore the solution search space and employ a comprehensive evaluation method that uses dynamic programming to determine the optimal mapping of the model layers onto the FPGAs. We also perform accelerator configuration search for better resource utilization by analyzing the model layer types and layer dependencies in the model. Using popular MMMT models such as VLocNet and QDTrack as reference, we show that M5 provides a remarkable gain in terms of energy efficiency by significantly lowering the number of FPGA boards required for model implementation.

References

[1] W. Zhang *et al.*, "An efficient mapping approach to large-scale dnns on multi-fpga architectures," in *DATE*. IEEE, 2019, pp. 1241–1244.

[2] J. Shan *et al.*, "Cnn-on-aws: Efficient allocation of multikernel applications on multi-fpga platforms," *IEEE TCAD*, 2020.

[3] W. Jiang *et al.*, "Achieving super-linear speedup across multi-fpga for real-time dnn inference," *ACM TECS*, vol. 18, no. 5s, pp. 1–23, 2019.

[4] Y. Yamauchi *et al.*, "Horizontal division of deep learning applications with all-to-all communication on a multi-fpga system," in *CANDARW*. IEEE, 2020, pp. 277–281.

[5] Y. Sun and H. Amano, "Fic-rnn: A multi-fpga acceleration framework for deep recurrent neural networks," *IEICE Transactions on Information and Systems*, vol. 103, no. 12, pp. 2457–2462, 2020.

[6] J. Shan *et al.*, "Fast energy-optimal multi-kernel dnn-like application allocation on multi-fpga platforms," *IEEE TCAD*, 2021.

[7] S. Biookaghazadeh *et al.*, "Toward multi-fpga acceleration of the neural networks," *ACM JETC*, vol. 17, no. 2, pp. 1–23, 2021.

[8] C. Hao and D. Chen, "Software/hardware co-design for multi-modal multi-task learning in autonomous systems," in *AICAS*. IEEE, 2021.

[9] X. Zhang *et al.*, "H2h: Heterogeneous model to heterogeneous system mapping with computation and communication awareness," *DAC*, 2022.

[10] M. Iqbal *et al.*, "A multimodal recommender system for large-scale assortment generation in e-commerce," *arXiv:1806.11226*, 2018.

[11] S. Thuseethan *et al.*, "Multimodal deep learning framework for sentiment analysis from text-image web data," in *2020 WI-IAT*. IEEE, 2020.

[12] S. Ruder, "An overview of multi-task learning in deep neural networks," *arXiv:1706.05098*, 2017.

[13] R. Hu and A. Singh, "Unit: Multimodal multitask learning with a unified transformer," in *ICCV*, 2021, pp. 1439–1449.

[14] A. Mehler *et al.*, "Vannotator: A framework for generating multimodal hypertexts," in *Hypertext and Social Media*, 2018, pp. 150–154.

[15] S. Tripathi *et al.*, "Multi-modal emotion recognition on iemocap with neural networks." *arXiv:1804.05788*, 2018.

[16] A. Valada *et al.*, "Deep auxiliary learning for visual localization and odometry," in *ICRA*. IEEE, 2018.

[17] G. Brightwell and P. Winkler, "Counting linear extensions is #p-complete," in *ACM STOC*, 1991, pp. 175–181.

[18] L. G. Valiant, "The complexity of enumeration and reliability problems," *SIAM Journal on Computing*, vol. 8, no. 3, pp. 410–421, 1979.

[19] A. B. Kahn, "Topological sorting of large networks," *Commun. ACM*, p. 558–562, nov 1962.

[20] J. Pang *et al.*, "Quasi-dense similarity learning for multiple object tracking," in *IEEE/CVF CVPR*, June 2021.

[21] R. Ramezani, "Dynamic scheduling of task graphs in multi-fpga systems using critical path," *The Journal of Supercomputing*, 2021.

[22] K. He *et al.*, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[23] S. Zhang *et al.*, "A dataset and benchmark for large-scale multi-modal face anti-spoofing," in *Proceedings of the CVF*, 2019, pp. 919–928.

[24] T. Shen *et al.*, "Facebagnet: Bag-of-local-features model for multi-modal face anti-spoofing," in *Proceedings of the CVF*, 2019.

[25] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM journal on Applied Mathematics*, vol. 17, no. 2, pp. 416–429, 1969.