

H2H: Heterogeneous Model to Heterogeneous System Mapping with Computation and Communication Awareness

Xinyi Zhang¹, Cong Hao², Peipei Zhou¹, Alex Jones¹, and Jingtong Hu¹

¹University of Pittsburgh, ²Georgia Institute of Technology
{xiz173, peipei.zhou, akjones, jthu}@pitt.edu, callie.hao@ece.gatech.edu

ABSTRACT

The complex nature of real-world problems calls for heterogeneity in both machine learning (ML) models and hardware systems. The heterogeneity in ML models comes from multi-sensor perceiving and multi-task learning, i.e., multi-modality multi-task (MMMT), resulting in diverse deep neural network (DNN) layers and computation patterns. The heterogeneity in systems comes from diverse processing components, as it becomes the prevailing method to integrate multiple dedicated accelerators into one system. Therefore, a new problem emerges: *heterogeneous model to heterogeneous system mapping (H2H)*. While previous mapping algorithms mostly focus on efficient computations, in this work, we argue that it is indispensable to consider computation and communication simultaneously for better system efficiency. We propose a novel H2H mapping algorithm with both computation and communication awareness; by slightly trading computation for communication, the system overall latency and energy consumption can be largely reduced. The superior performance of our work is evaluated based on MAESTRO modeling, demonstrating 15%-74% latency reduction and 23%-64% energy reduction compared with existing computation-prioritized mapping algorithms. Code is publicly available at <https://github.com/xyxinyizhang/H2H>.

1 INTRODUCTION

As DNNs are applied in more and more complicated tasks, both machine learning (ML) models and hardware acceleration systems call for **heterogeneity** [1, 2] to address emerging challenges. First, ML algorithms are evolving from handling single-modality single-task to multi-modality multi-task (MMMT) [1]. For instance, in the recommendation system, visual and textual data are jointly learned in a multi-modality fashion [3]; in AR/VR applications, image, gesture, and speech are jointly learned [4]. Such changes result in increasingly complicated ML models with larger size and complex inter-block connections. The top of Fig. 1 depicts heterogeneous ML models as well as a real-life model, VlocNet [5], for semantic visual localization. Second, recent advanced systems are introducing great heterogeneity by integrating diverse acceleration components with different capabilities to pursue low latency and high energy efficiency. For example, at the System-on-Chip (SoC) level, Xilinx Versal [6], Nvidia Xavier [7], and Tesla FSD [8] integrate various processing components on a single chip. At the cloud level, Microsoft’s Brainwave [2] and AWS [9] are composed of multiple FPGAs which can be flexibly reconfigured. In this paper, we target a cloud-scale multi-FPGA [2] as the target system architecture, shown in the bottom part of Fig. 1, where each leaf device is an FPGA that can be configured as an arbitrary accelerator.

We refer to mapping and scheduling a heterogeneous ML model onto a heterogeneous system as **H2H**. The H2H problem is non-trivial because of the following complexities. (1) **Computation Awareness**. The heterogeneous MMMT model components can vary largely in terms of layer type, layer shape, and data dimension. For instance, an MMMT model can be composed of convolution

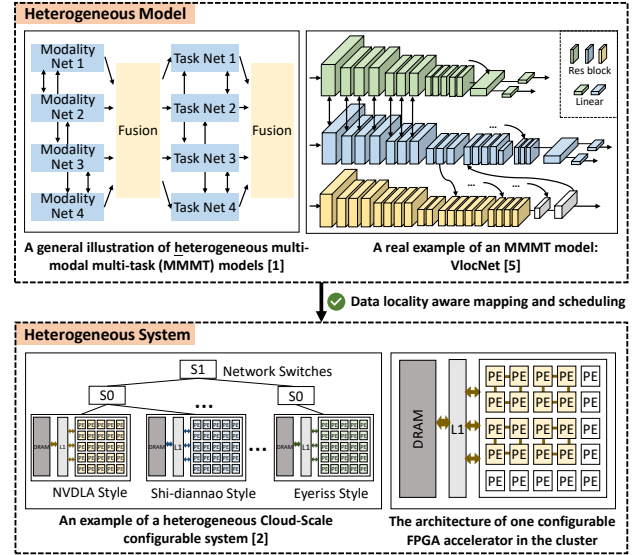


Figure 1: The overview of H2H mapping and scheduling: from heterogeneous models to heterogeneous system.

(Conv), fully connected (FC), long short-term memory (LSTM), and transformer layers, with significantly different dataflow patterns and preferred accelerator architectures [10]. A first challenge is revealed, how to map layers to the desirable accelerators, suitable for their computation patterns. (2) **Communication Awareness**. Computation-prioritized mapping does not necessarily lead to global best performance if communication, i.e., data transfer across different accelerators, is ignored. Fig. 2 demonstrates the difference between computation-prioritized mapping and communication-aware mapping: the former maps each layer purely based on its preferable dataflow pattern, while the latter slightly sacrifices computation efficiency but in turn reduces the overall system latency by avoiding expensive data transfer. Thus, a second challenge emerges, as modern MMMT models have more complex dependencies (e.g., the cross-layer connections in VlocNet [5] in Fig. 1), the data transfer overhead and optimization difficulty become exaggerated.

Existing mapping algorithms for DNNs are primarily prioritized based on computation. For instance, Fowers et al. [2] improve a single accelerator computation efficiency by augmenting the dataflow, but do not discuss system-level cross-accelerator communication. Chen et al. [11] map DNN layers to different accelerators to fully utilize DSP and RAM resources; Kwon et al. [10] propose the state-of-the-art mapper, which improves the convolution efficiency by 65% by mapping Conv layers to different styles of accelerators such as Eyeriss, NVDLA, and Shi-diannao [12–14]. The data transfer overhead, however, is not considered.

To address the discussed challenges and limitations, in this work, we propose the first **H2H mapping algorithm with both computation and communication awareness**, aiming at system-level formulation, modeling, and optimization. We first formulate the H2H problem using two graphs to depict the model layer dependency and accelerator execution order. We then build a system-level modeling infrastructure to model arbitrary heterogeneous systems, based on each accelerator’s computation and communication performance models. Guided by the infrastructure, we propose an H2H mapping algorithm, aiming to largely reduce system overall latency and energy. We summarize our contributions as follows:

- **H2H problem formulation and system modeling.** We formulate the H2H problem as two directed graphs to describe layer dependency of the heterogeneous model and accelerator execution order in the heterogeneous system. We then develop a configurable system-level infrastructure based on MAE-STRO [15], which takes arbitrary accelerators with user-defined performance models in a plug-in manner to obtain system latency and energy.
- **H2H mapping algorithm** We propose an H2H mapping algorithm with both computation and communication awareness, including computation-prioritized mapping, weight locality and activation transfer optimization, and data locality aware remapping. An optimized mapping can be found within seconds.
- **Performance evaluation.** We comprehensively evaluate our mapping algorithm on the infrastructure using 6 real-world heterogeneous DNN models on a heterogeneous system composed of 12 off-the-shelf accelerators. We achieve 15% to 74% latency reduction and 23% to 64% energy reduction compared with existing computation-prioritized mapping algorithms.

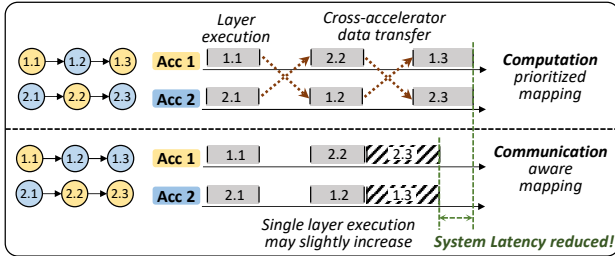


Figure 2: An example of communication-prioritized mapping and communication-aware mapping.

2 BACKGROUND AND MOTIVATION

Heterogeneous ML Model. Modern real-life ML applications usually involve multiple inputs and handle multiple tasks simultaneously, i.e., multi-modal multi-task (MMMT) [1]. Multi-modal models aim to process and relate information from multiple sources to capture the correspondences between modalities, while multi-task models aim to efficiently enforce cross-task information exchange for improving performance and robustness. Such MMMT models expose great heterogeneity with various layer types, complex inter-layer connections, and data exchange. In practice, convolution layers are widely seen in vision tasks; FC layers are mostly used in small-scale data extraction; LSTM/Transformer layers are prevalent in text and language tasks [16]. Different applications can

Table 1: System Performance Modeling Parameters

Acc Type	Parameters	Explanation
Conv	$\langle N, M, R, C, K, S \rangle$	ofm_channel_num, ifm_channel_num, ofm_height, ofm_width, kernel_size, stride
FC	$\langle N, M \rangle$	in_features, out_features
LSTM	$\langle N, H, L \rangle$	in_size, hidden_size, layers
–	BW_{acc}	accelerator-to-host bandwidth
–	M_{acc}	local DRAM size

have largely varied layer types with different preferable computing patterns, accelerator designs, and data transfer cost.

Heterogeneous System. A configurable multi-FPGA system is one representative heterogeneous system, since each FPGA can be flexibly configured with different accelerator architectures, such as Eyerriss [12], NVDLA [13], and Shi-diannao [14]. Microsoft FPGA datacenter [2] is a good heterogeneous example as shown in Fig. 1 (bottom). Heterogeneous systems can largely improve computation efficiency, because a certain accelerator is typically optimized only for a subset of ML layers, and different layers are expected to be mapped to their specific accelerators.

Motivation. Given the complexity of ML models and heterogeneous systems, it is non-trivial to find a good H2H mapping that can effectively balance **computation** and **communication**. First, DNN accelerators are highly specialized for certain dataflows. For instance, NVDLA [13] optimizes convolution channel-wise parallelism, while Shi-diannao [14] optimizes feature-map-wise parallelism. Model layers should be mapped to the accelerators with preferable computation patterns to reduce computation latency. Most existing approaches highly prioritize the computation pattern but usually ignore cross-layer communication [10]. Second, there are also communication-prioritized mapping algorithms [17] by forming task clusters and assigning a cluster to a processor. However, this may largely hurt the computing efficiency since the tasks within the same cluster do not necessarily run efficiently on the same accelerator. Meanwhile, the heavy cross-layer dependency (cross-talk) in the heterogeneous models may also lead to ineffective clustering. Third, existing mapping algorithms lack the formulation of DNN models and system-level information such as accelerators’ architecture and dataflow. Without hardware awareness, existing algorithms cannot be directly and efficiently applied. Therefore, a hardware-aware mapping algorithm that considers both computation and communication simultaneously is needed.

The cornerstone for communication reduction is **data locality** by efficiently utilizing accelerators’ local DRAM. In a multi-FPGA system, each FPGA is equipped with a local DRAM, which can be utilized to store model weights and to buffer intermediate activations of two adjacent layers to reduce cross-FPGA data movement. The challenge is that the computation-prioritized mapping can achieve best efficiency per FPGA, but the overall performance may be compromised due to the transmission cost (and vice versa). Therefore, we propose an H2H mapping algorithm, which can jointly consider the benefit of high data locality and suitable computation patterns.

3 SYSTEM FORMULATION

Heterogeneous ML Model. As shown in Fig. 1, a heterogeneous model has complicated dependencies especially for cross-talk connections. It is natural to formulate such a model as a directed graph $G_{model} = (V, E)$, where the vertices represent the layers and the

edges represent the dependencies. In G_{model} , each node holds layer information such as Conv, FC, LSTM, etc., as well as their data dimension (e.g., feature map size). We consider three types of popular accelerators with the layer parameters summarized in Table 1.

Heterogeneous System. We also formulate the multi-FPGA system as a directed graph $G_{sys} = \{G_{Acc_i}\}$, where each sub-graph G_{Acc_i} is a computation graph representing the layers' execution scheduling on the i -th FPGA accelerator Acc_i . Initially, each graph G_{Acc_i} is empty without any mapping. An example of G_{model} and initial G_{sys} with three initial empty Acc_i are shown in Fig. 3's input block. After mapping, each G_{Acc_i} will be composed of nodes from G_{model} in their execution order.

In this work, we consider a multi-FPGA system proposed in [2], where each FPGA is connected to a host node via Ethernet switches, enabling FPGA-to-FPGA and FPGA-to-host communication. The host node distributes data to each FPGA's local DRAM memory, whose capacity typically ranges from 512 MB to 8 GB [16] and is usually used as additional buffers to mitigate the scarcity of FPGA on-chip memory. The Ethernet speed ranges from 1 G to 10 G Ethernet (0.125 to 1.25 GB/s) in cloud-FPGA [9], and FPGA local DRAM speed ranges from 6.4 GB/s to 460 GB/s [18]. We consider two most important system-level parameters, accelerator-to-host bandwidth and local DRAM size, denoted by BW_{acc} and M_{acc} , respectively, as shown in Table 1.

System Performance Model. We model the overall heterogeneous system performance at two levels: individual accelerator, and the overall system. First, for individual accelerator, there are plenty of analytical models of different designs, so we directly adopt the performance models from existing literature, denoted by P_{acc} . For each accelerator, we consider the following configurable parameters: (1) BW_{acc} , the accelerator to main memory bandwidth; (2) M_{acc} , the local DRAM size; (3) $Layer_{para}$, other layer parameters as shown in Table 1. For instance, the analytical model for the accelerator proposed in [19] can be expressed as $P_{Acc} < M_{acc}, Layer_{para} >$ with its loop tiling setting $< R_{wei}, D_{type}, F_{acc}, BW_{dram}, T_m, T_n, T_r, T_c >$. Second, for the system-level performance model, we modify the MAESTRO for the target multi-FPGA system by allowing customizing the accelerator-to-host bandwidth as BW_{acc} , which is also configurable by users.

4 PROPOSED H2H MAPPING ALGORITHM

In this section, we discuss our proposed computation and communication aware H2H mapping algorithm, according to our analytical model based infrastructure. As shown in Fig. 3 top, there are four steps. **1 Computation-prioritized mapping.** The heterogeneous ML model is mapped at layer-granularity, that each layer is mapped to the accelerator that best fits its computation dataflow, ignoring all data movement optimizations (i.e., zero data locality). **2 Weight locality optimization.** Since each accelerator has its own local DRAM, we buffer part of the weights in the memory to maximally avoid weight data movement. **3 Activation transfer optimization.** If two adjacent layers are mapped to the same accelerator, their intermediate activation, i.e., the output/input feature maps (OFM/IFM), will no longer need to transfer and thus latency can be reduced. **4 Data locality aware re-mapping.** This step explores the trade-off between computation and communication, aiming to largely reduce communication cost with slight computation efficiency degradation, which still results in overall performance improvement. The H2H algorithm flow is shown in Algorithm 1. It takes G_{model} and P_{Acc_i} as inputs, and produces a mapped and

Algorithm 1: H2H Mapping and Scheduling

Input: G_{model}, P_{Acc_i}
Output: $G_{model}^*, G_{sys}^* = \{G_{Acc_i}^*\}, Syslatency, Sysenergy$

```

1 Function Computation_Prioritized_Mapping():
2   for nodes in  $G_{model}$  without predecessors do
3     Enumerate all possible mappings based on  $P_{Acc_i}$ 
4     Choose the mapping with minimum  $\Delta Syslatency$ 
5 Function Weight_Locality_Opt():
6   Knapsack_Solver( $G_{model}, G_{sys}$ )
7    $Syslatency, Sysenergy \leftarrow$  update_System_Scheduling()
8 Function Activation_Transfer_Opt():
9   for every node pair adjacent in  $G_{sys}$  do
10    activation_Fusion(node pair)
11    $Syslatency, Sysenergy \leftarrow$  update_System_Scheduling()
12 Function Data_Locality_Remapping():
13   Repeat
14     for every  $n \in G_{model}$  do
15       Attempt to remap  $n$  to its neighbors' Acc
16       Weight_Locality_Opt()
17       Activation_Transfer_Opt()
18        $\Delta Syslatency \leftarrow$  update_System_Scheduling()
19       Accept remap if  $\Delta Syslatency < 0$ 
20   Until no more beneficial remapping operations;
```

scheduled solution (G_{model}^*, G_{sys}^*) with modeled system latency and energy ($Syslatency, Sysenergy$).

4.1 Computation-prioritized Mapping

In the first step, we perform Computation_Prioritized_Mapping. It assigns the model layers to the accelerators that result in the best computation performance by assuming zero local DRAM without any data locality at the accelerator. We use performance model P_{Acc_i} to estimate the latency of a layer executing on the i -th accelerator, and assume that all the weights and intermediate results go to the main memory of the host node. To obtain system latency, the layer scheduling on each accelerator must be determined. To guarantee a valid scheduling considering layer dependencies especially across multiple sub-models, the algorithm determines the mapping and scheduling iteratively. In every iteration, it selects all the nodes without predecessors from G_{model} as a group, enumerates all possible mappings within the group (multiple nodes can be mapped to one or more accelerators), and selects the one that results in the smallest system latency increment.

An example is shown in Fig. 3 1, where the color of the nodes represents which accelerator it is mapped to. The gray blocks represent the accelerator execution, where idle periods are introduced by layer dependency. Note that, in this step, we assume zero local DRAM, so that the latency values include both computation and communication: layer computation, weight transfer from the main memory, and IFM/OFM transfer from/to the main memory.

4.2 Weight Locality Optimization

Weight_locality_Opt is performed after computation-prioritized mapping by utilizing the local DRAM of each accelerator. With local DRAM, weight transfer from main memory can be greatly reduced, and it is a common practice to buffer part (or all) of the weights of the DNN layer(s) [16]. In this system, since multiple layers are mapped to the same accelerator, the layer weights must be selectively stored in the local DRAM, under a certain memory budget. Therefore, we propose to use the Knapsack algorithm to store, as much as possible, weights in the accelerators' local DRAM to reduce weight transfer. After pinning weights locally, we first update each layer's

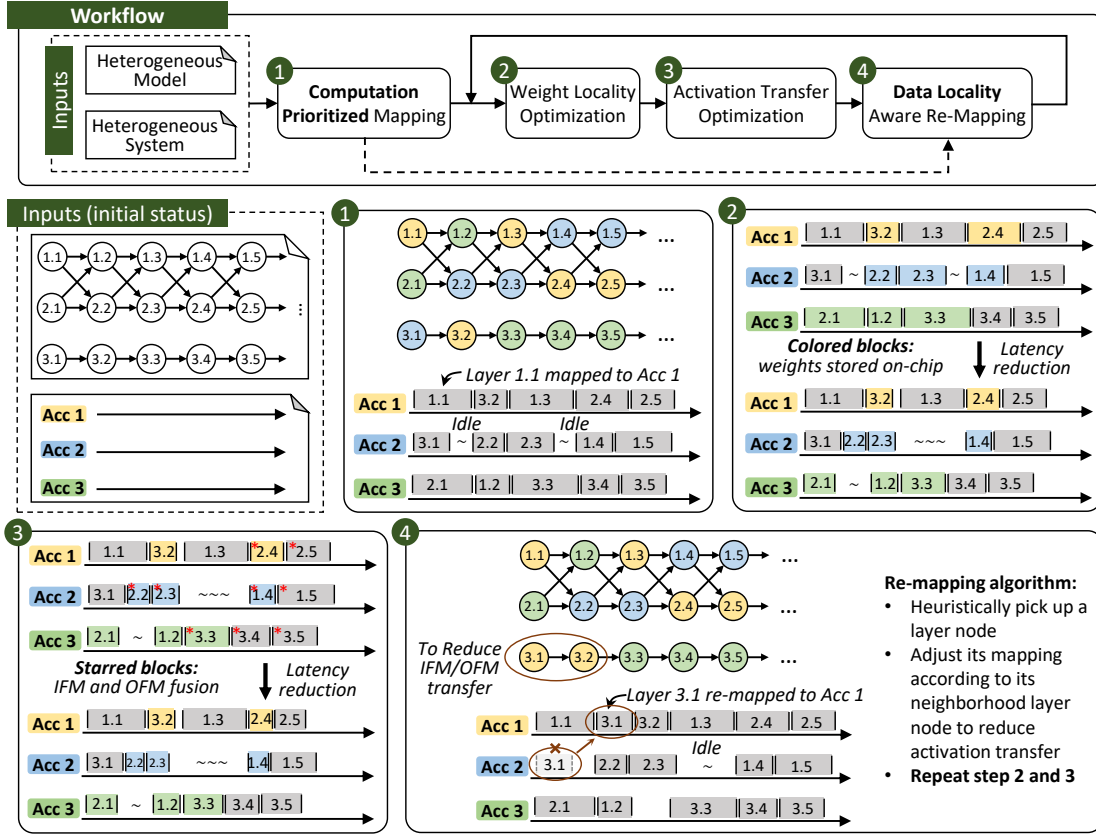


Figure 3: H2H mapping algorithm visualization. It includes 4 major steps: (1) computation-prioritized mapping; (2) weight locality optimization; (3) activation transfer optimization; (4) data locality aware remapping.

latency, and then update the system’s scheduling and overall latency. Note that, since changing the latency and scheduling of one layer can affect all its successor layers, we update the layer scheduling *recursively*. This is especially efficient for graph structures, since in each iteration, we only update a node’s direct successor neighbors without traversing the entire graph. An example is shown in Fig. 3 ②, where the colored blocks represent the layers whose weights are stored in the local DRAM with reduced latency. The system latency is also reduced.

4.3 Activation Transfer Optimization

After weight locality optimization, the activation (IFM/OFM) transfer will be optimized by Activation_Transfer_Opt to further reduce the overall cost. This is based on the assumption that, if two adjacent layers are mapped to the same accelerator, their intermediate IFM and OFM can be reused locally by taking advantage of the local DRAM and thus the activation transfer from/to the main memory can be avoided. We call it activation fusion, which can be performed recursively, similar to the weight locality optimization: for each mapped layer, it checks its successor neighbors for activation fusion, updates its own and its neighbors’ latency, if applicable, and recursively updates the system’s overall scheduling. An example of activation fusion is shown in Fig. 3 ③, where the starred blocks indicate the layers that are applicable for fusion.

4.4 Data Locality Aware Remapping

The weight and activation optimization are post-optimizations for communication given a mapping solution. In this step, we execute

Data_Locality_Remapping, for communication-oriented remapping, i.e., initial mapping tuning aiming at largely reducing communication cost. Specifically, for each layer, we define a *remapping* operation that re-allocates a layer from its source accelerator to a new destination accelerator, on which its predecessors and/or successors are mapped. This remapping reduces the activation transfer time by allowing activation fusion, but may increase the computation latency. The weight transfer latency may be increased or decreased depending on the available local DRAM capacity of the destination accelerator. Therefore, to determine the exact effect of a remapping operation, weight locality and activation transfer optimization, i.e., step 2 and 3, must be re-executed for every remapping attempt. We adopt a greedy algorithm and perform remapping attempt for every layer; a remapping is accepted only if it reduces the system’s overall latency, i.e., the benefit of communication reduction outweighs the computation cost increment. The algorithm terminates when no more layers can be remapped with reduced overall latency.

An example of locality-aware remapping is shown in Fig. 3 ④. In this example, layer 3.1 is remapped from Acc2 to Acc1 since its neighbor layer 3.2 resides on Acc1, so that the activation transfer between layer 3.1 and layer 3.2 can be reduced. Although in this example, the scheduling of layer 2.2 cannot move earlier because of the layer dependency, in most cases, the source accelerator can reduce its latency because of the memory budgeted for weights is released and it can execute earlier because it can take the vacated cycles of the remapped layer.

4.5 Extension for Dynamic Modality Change

The proposed H2H algorithm can be easily extended to handle dynamic modality change scenarios which is common in multi-sensor systems. For instance, a health monitoring system may choose to turn on and off motion sensors based on the surrounding environment and the person’s activity [20], and such dynamic modality change can be as frequent as several times within one second. This motivates an extended H2H mapping for dynamic modality changes, where a new mapping with increased or decreased modalities (i.e., layers) depends on the previous mapping result to maximally re-use the buffered weights. The advantage is to avoid weight loading for frequent modality change.

Therefore, we modify the proposed H2H algorithm as follows for dynamic modality change. Given the previous mapping and weight buffering, for a new set of modalities (layers), it prioritizes the layer mapping if the layer’s weights are already buffered on a certain accelerator. Then, we repeat steps 1 to 4 with a modified Knapsack algorithm, where part of the weight allocation is determined.

5 EVALUATIONS

5.1 Evaluation Settings

Heterogeneous models. Table 2 summarizes 6 heterogeneous DNN models used in evaluation, spanning different domains including Augmented Reality (AR), Face Recognition, Sentiment Analysis, Activity Recognition, and Emotion Recognition. Most models use Convolution Neural Networks (ResNet, VGG, VD-CNN, and their variants) as backbones, and there are typically 3 to 5 backbones placed together for MMT with cross-backbone data dependencies. **Heterogeneous accelerators.** We survey 12 state-of-the-art FPGA-based Convolution/FC/LSTM accelerators and summarize them in Table 3. We replicate their performance models based on the original papers; we honor the local DRAM capacity M_{acc} based on the FPGA boards used, ranging from 512 MB to 8 GB [16].

System modeling. We modify MAESTRO [15] to a system-level infrastructure to model the cloud-scale multi-FPGA system as shown in Fig. 1 (bottom) [2]. The Ethernet speed BW_{acc} between FPGAs and main memory ranges from 1 G to 10 G Ethernet (0.125 to 1.25 GB/s) [9]. The system latency and energy are modeled based on the values reported in the accelerator papers in Table 3.

Table 2: Heterogeneous (MMT) models

Domain	Model	Backbones	Para.
Augmented Reality	VLocNet [5]	ResNet-50 variants	192M
Face Recognition	CASUA-SURF [21]	ResNet-18 variants	13.2M
Sentiment Analysis	VFS [22]	VGG and VD-CNN variants	365M
Face Recognition	FaceBag [23]	ResNet variants	25M
Activity Recognition	CNN-LSTM [24]	ConvNet and LSTM variants	16M
Emotion Recognition	MoCap [25]	Convolution and LSTM unit	8M

5.2 H2H mapping performance

Baseline. As discussed in Section 2, existing mapping algorithms are computation-prioritized that strive for finding the most suitable accelerators based on the dataflow patterns [10]. This mapping strategy is the same as the first step in our H2H mapping. To make a fair comparison, we take the results from H2H mapping after the second step (Section 4.2) including the weight locality optimization, since existing works can also assume local DRAM for the accelerators.

Table 3: State-of-the-art FPGA DNN accelerators

Name	Accelerator Type	Optimization	FPGA
J.Z [26]	Convolution	On-chip memory	GX1150
C.Z [19]	Convolution	Channel parallel.	VC707
W.J [27]	Convolution	Memory and Channel	ZCU102
J.Q [28]	Conv/FC/LSTM	Computing Generality	ZC706
A.C [29]	Convolution	Loop Optimization	XC7Z045
Y.G [30]	Conv/FC/LSTM	Computing Generality	Stratix-V
T.M [31]	Convolution	Loop Optimization	GX1150
A.P [32]	Convolution	Winograd	Stratix-V
X.W [33]	Convolution	Systolic Array	GT1150
S.H [34]	LSTM/FC	Deep Pipeline	XCKU060
X.Z [35]	LSTM	Gate Parallelism	PYNQ-Z1/VC707
B.L [36]	LSTM	Deep Pipeline	VCU118

Latency and Energy Reduction. In Fig. 4, we present the latency and energy reduction of the 6 heterogeneous DNN models. We test the H2H mapping algorithm under different network bandwidth configurations (BW_{acc}): Low- (0.125 GB/s); Low (0.15 GB/s); Mid- (0.25 GB/s); Mid (0.5 GB/s); High (1.25 GB/s). The x-axis refers to the four steps in H2H mapping algorithm, and the y-axis is the modeled system latency in seconds and energy in joule. The H2H mapping algorithm achieves 15% to 74% system latency reduction and 23% to 64% energy reduction compared with the baseline mapping [10] when the system is bandwidth bounded, i.e., under the bandwidth Low- setting. With high bandwidth, the H2H still reduces overall latency by 10% to 50%. In half of the evaluated cases, we achieved over 60% latency reduction.

The detailed latency reduction after each step is shown in Table 4. Since we regard the second step as the baseline, we present the absolute latency values (in seconds) for steps 1 and 2 and the relative values for steps 3 and 4 compared with step 2. Apparently, when the bandwidth increases, the reduction decreases, but even with high bandwidth, network CNN-LSTM and MoCap still reduce latency by almost half from H2H mapping.

H2H performance analysis. In Fig. 5(a), we visualize the communication and computation latency ratio using the mapping results under Bandwidth Low- of the six models. Note that after our H2H mapping, the computation ratio greatly increases (yellow bars), where MoCap increases from 21% to 94%, indicating that the communication overhead is largely reduced. We also show the H2H mapping algorithm execution time in Fig. 5(b). The search time is consistently low across different DNN models, less than one second. The VLocNet requires longer search time since it consists of 141 layers; the CNN-LSTM and MoCap are significantly faster since they consist of less than 30 layers.

6 CONCLUSION

In this work, we proposed a computation and communication aware H2H algorithm, aiming at system-level formulation, modeling, and optimization for mapping heterogeneous models to heterogeneous systems. We achieve up to 74% system latency reduction and 64% energy reduction. The H2H is designed with high flexibility, as it is configurable at system level and adopts a plug-in manner for accelerators. It can be easily configured to catch up with the latest advancements in deep learning society, such as the growing size of DNN models, increasing intensity of accelerator computing resource, and system network bandwidth.

REFERENCES

- [1] Cong Hao et al. Software/hardware co-design for multi-modal multi-task learning in autonomous systems. In *Proceeding of AICAS*, pages 1–5. IEEE, 2021.

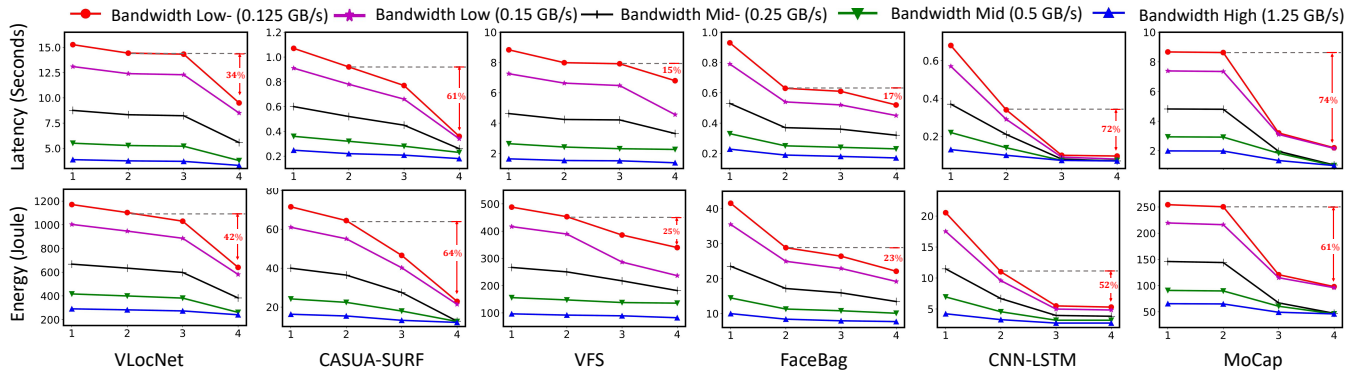


Figure 4: The latency and energy performance comparison.

Table 4: Latency reduction breakdown comparing with the second step (baseline).

Bandwidth	VLocNet				CASUA-SURF				VFS				FaceBag				CNN-LSTM				MoCap			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Low-	15.27	14.43	99.31%	65.84%	1.07	0.92	83.70%	39.13%	8.84	7.99	99.12%	85.11%	0.93	0.63	96.83%	82.54%	0.68	0.34	29.41%	28.24%	8.67	8.63	37.08%	25.49%
Low	13.10	12.40	99.11%	68.55%	0.91	0.78	84.62%	43.59%	7.26	6.64	97.59%	68.67%	0.79	0.54	96.30%	83.33%	0.57	0.29	31.03%	27.59%	7.39	7.35	42.18%	29.39%
Mid-	8.76	8.33	98.80%	66.87%	0.60	0.52	86.54%	50.00%	4.63	4.24	99.29%	78.07%	0.53	0.37	97.30%	86.49%	0.37	0.21	38.10%	33.33%	4.82	4.8	40.83%	21.67%
Mid	5.51	5.28	98.67%	71.78%	0.36	0.32	87.50%	71.88%	2.64	2.42	95.45%	93.39%	0.33	0.25	96.00%	92.00%	0.22	0.14	52.14%	52.14%	2.94	2.92	62.67%	34.25%
High	3.88	3.76	98.67%	88.03%	0.25	0.22	94.55%	81.82%	1.64	1.53	98.69%	90.20%	0.29	0.189	95.24%	89.95%	0.13	0.10	73.00%	70.00%	1.99	1.98	67.68%	50.51%

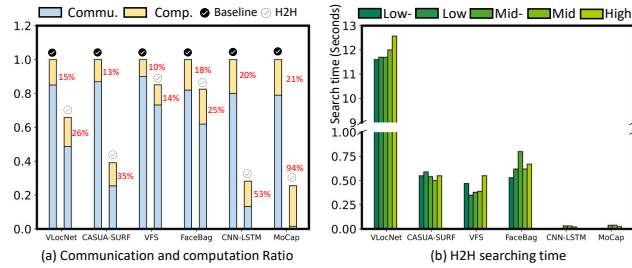


Figure 5: Communication and computation ratio.

- [2] Jeremy Fowers et al. A configurable cloud-scale dnn processor for real-time ai. In *Proceeding of ISCA*, pages 1–14. IEEE, 2018.
- [3] Murium Iqbal et al. A multimodal recommender system for large-scale assortment generation in e-commerce. *arXiv preprint arXiv:1806.11226*, 2018.
- [4] Alexander Mehler et al. Vannotator: A framework for generating multimodal hypertexts. In *Proceedings of the Hypertext and Soc. Media*, pages 150–154. 2018.
- [5] Abhinav Valada, Noha Radwan, and Wolfram Burgard. Deep auxiliary learning for visual localization and odometry. In *2018 ICRA*, pages 6939–6946. IEEE, 2018.
- [6] Brian Gaide et al. Xilinx adaptive compute acceleration platform: Versal™ architecture. In *Proceedings of the 2019 FPGA*, pages 84–93, 2019.
- [7] Michael Ditty et al. Nvidia’s xavier soc. In *Hot chips: a symposium on high performance chips*, 2018.
- [8] Emil Talpes et al. Compute solution for tesla’s full self-driving computer. *IEEE Micro*, 40(2):25–35, 2020.
- [9] Aws network. <https://aws.amazon.com/blogs/aws/new-gigabit-connectivity-options-for-amazon-direct-connect/>.
- [10] Hyoukjun Kwon et al. Heterogeneous dataflow accelerators for multi-dnn workloads. In *Proceedings of HPCA*. IEEE, 2021.
- [11] Yao Chen et al. Cloud-dnn: An open framework for mapping dnn models to cloud fpgas. In *Proceedings of the 2019 FPGA*, pages 73–82, 2019.
- [12] Yu-Hsin Chen et al. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ACM SIGARCH Computer Architecture News*.
- [13] Nvidia. Website. <http://nvidia.org/>.
- [14] Zidong Du et al. Shidiannao: Shifting vision processing closer to the sensor. In *Proceedings of ISCA*, pages 92–104, 2015.
- [15] Hyoukjun Kwon et al. MAESTRO: A data-centric approach to understand reuse, performance, and hardware cost of DNN mappings. *IEEE Micro*, 40(3), 2020.
- [16] Kaiyuan Guo et al. A survey of fpga-based neural network inference accelerators. *ACM Transactions on TRES*, 12(1):1–26, 2019.
- [17] Kenjiro Taura et al. A heuristic algorithm for mapping communicating tasks on heterogeneous resources. In *Proceedings of HCV*, pages 102–115. IEEE, 2000.
- [18] Chris Riley. Basic tutorial for maximizing memory bandwidth with vitis and xilinx ultrascale+ hbm devices, 2019.

- [19] Chen Zhang et al. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the FPGA*, pages 161–170, 2015.
- [20] Da-Ren Chen et al. A power-aware 2-covered path routing for wireless body area networks with variable transmission ranges. *Journal of JPD*.
- [21] Shifeng Zhang et al. A dataset and benchmark for large-scale multi-modal face anti-spoofing. In *Proceedings of the CVF*, pages 919–928, 2019.
- [22] Selvarajah Thuseethan et al. Multimodal deep learning framework for sentiment analysis from text-image web data. In *2020 WI-IAT*. IEEE, 2020.
- [23] Tao Shen et al. Facebagnet: Bag-of-local-features model for multi-modal face anti-spoofing. In *Proceedings of the CVF*, 2019.
- [24] Xinyu Li et al. Concurrent activity recognition with multimodal cnn-lstm structure. *arXiv preprint arXiv:1702.01638*, 2017.
- [25] Samarth Tripathi et al. Multi-modal emotion recognition on iemocap with neural networks. *arXiv preprint arXiv:1804.05788*, 2018.
- [26] Jialiang Zhang et al. Improving the performance of opencl-based fpga accelerator for convolutional neural network. In *Proceedings of the 2017 FPGA*, 2017.
- [27] Weiwen Jiang et al. Achieving super-linear speedup across multi-fpga for real-time dnn inference. *ACM Transactions on TECS*, 18(5s):1–23, 2019.
- [28] Jiantao Qiu et al. Going deeper with embedded fpga platform for convolutional neural network. In *Proceedings of the 2016 FPGA*, pages 26–35, 2016.
- [29] Andre Xian Ming Chang et al. Compiling deep learning models for custom hardware accelerators. *arXiv preprint arXiv:1708.00117*, 2017.
- [30] Yijin Guan et al. Fp-dnn: An automated framework for mapping deep neural networks onto fpgas with rtl-hls hybrid templates. In *2017 IEEE FCCM*.
- [31] Yufei Ma et al. Optimizing loop operation and dataflow in fpga acceleration of deep convolutional neural networks. In *Proceedings of the 2017 FPGA*, 2017.
- [32] Abhinav Podili et al. Fast and efficient implementation of convolutional neural networks on fpga. In *2017 IEEE ASAP*, pages 11–18. IEEE, 2017.
- [33] Xuechao Wei et al. Automated systolic array architecture synthesis for high throughput cnn inference on fpgas. In *Proceedings of the 54th DAC*, 2017.
- [34] Song Han et al. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *Proceedings of the 2017 FPGA*, pages 75–84, 2017.
- [35] Xinyi Zhang et al. Achieving full parallelism in lstm via a unified accelerator design. In *2020 IEEE ICCD*, pages 469–477. IEEE, 2020.
- [36] Bingbing Li et al. Ftrans: energy-efficient acceleration of transformers using fpga. In *Proceedings of the ISLPED*, pages 175–180, 2020.