Fast Energy-Optimal Multi-Kernel DNN-like Application Allocation on Multi-FPGA Platforms

Fast Energy-Optimal Multi-Kernel DNN-like Application Allocation on Multi-FPGA Platforms

Junnan Shan, *Student Member, IEEE,* Mihai T. Lazarescu, *Senior Member, IEEE,* Jordi Cortadella, *Fellow, IEEE,* Luciano Lavagno, *Senior Member, IEEE,* and Mario R. Casu, *Senior Member, IEEE*

***Abstract*—Platforms with multiple Field Programmable Gate Arrays (FPGAs), such as Amazon Web Services (AWS) F1 instances, can efficiently accelerate multi-kernel pipelined applications, e.g., Convolutional Neural Networks for machine vision tasks or transformer networks for Natural Language Processing tasks. To reduce energy consumption when the FPGAs are underutilized, we propose a model to (1) find off-line the minimum-power solution for given throughput constraints, and (2) dynamically reprogram the FPGA at runtime (which is complementary to dynamic voltage and frequency scaling) to match best the workloads when they change. The off-line optimization model can be solved using a Mixed-Integer Non-Linear Programming (MINLP) solver, but it can be very slow. Hence, we provide two heuristic optimization methods that improve result quality within a bounded time. We use several very large designs to demonstrate that both heuristics obtain comparable results to MINLP, when it can find the best solution, and they obtain much better results than MINLP, when it cannot find the optimum within a bounded amount of time. The heuristic methods can also be thousands of times faster than the MINLP solver.**

***Index Terms*—CNN, NLP, transformer, multi-FPGA, allocation, optimization, heuristic, AWS.**

I. Introduction

OUTSTANDING Deep Neural Networks (DNN) results for, e.g., image recognition, object detection, and natural language processing (NLP), dramatically increase their use and energy consumption, weighing on the environment and increasing data center costs. The main cloud providers, e.g. Amazon, Alibaba, Microsoft, offer multi-FPGA platforms that use less energy than CPUs or GPUs. ASICs consume even less energy, but FPGAs are incomparably more configurable, supporting rapid application evolution.

As data center workloads change over time, accelerators sized for the highest application throughput are often underused, wasting FPGA resources and energy. Frequency scaling can reduce energy consumption, but not the resource occupation.

We propose using FPGA reconfiguration to optimize both resource usage and energy for a given throughput, on Amazon web service (AWS) F1 instances with the architecture shown in Fig. 1. Eight Xilinx UltraScale+ FPGAs, each with an independent clock and local DDR DRAM, are connected with a PCI Express (PCIe) bus to an x86 host CPU, which orchestrates

J. Shan, M.T. Lazarescu, L. Lavagno and M.R. Casu are with the Department of Electronics and Telecommunications, Politecnico di Torino, I-10129 Torino, Italy, e-mail: mario.casu@polito.it.

J. Cortadella is with the Computer Science Department, Universitat Politècnica de Catalunya, 08034 Barcelona, Spain, e-mail: jordi.cortadella@upc.edu

Manuscript received Month XX, 20ZZ; revised Month YY, 20ZZ.

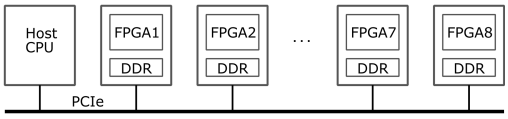

Fig. 1. Architecture of the Amazon web service (AWS) F1 instance

application execution on FPGAs and PCIe communications. We pipeline multi-kernel applications at the kernel level, with all kernels running at the same time, hence the Initiation Interval (*II*, the inverse of the throughput) is determined by the slowest kernel. We ignore reconfiguration energy because we are considering infrequent reconfiguration (e.g., once per minute, or once per hour) to meet daily workload variations depending on the time of the day, not on second- or millisecond-scale variations [1] [2].

As in our previous work [3], we split the applications into several pipelined kernels (e.g., one per macro-layer in a Deep Neural Network) and profile each of them with Xilinx SDAccel [4] for resource usage (LUTs, FFs, DSPs, BRAMs), execution time, and DDR memory bandwidth. Then we input the profiled kernels and available FPGA resources into our power and performance model to obtain a power-optimal implementation to program the FPGAs. The model uses integer variables and non-linear functions, and we solve it with a Mixed-Integer Non-Linear Programming (MINLP) solver, which is very slow, from several hours for small problems to days or more for larger problems. To speed up the optimization, we propose two heuristic methods. One reduces the problem size before applying the MINLP solver, while the other uses greedy kernel allocation while minimizing power consumption.

Efficient resource allocation for high-performance data center applications is a well-studied topic for GPUs, CPUs, and FPGAs. Previously, we provided detailed resource allocation models to maximize the application *throughput*, optimized with both an MINLP solver and a heuristic method, but *without considering power consumption* [5], [6] and we demonstrated in [3] a multi-objective optimization that minimizes energy consumption while meeting performance requirements using an MINLP solver, which is *very slow*. Here, we propose two heuristic solvers, which can be several orders of magnitude faster.

Li *et al.* [7] use a similar greedy resource allocation to the most critical kernel, balancing resource usage until exhaustion, but without minimizing power consumption or considering multi-FPGA allocation. Our model satisfies performance constraints while minimizing the overall, multi-FPGA power consumption. Cong *et al.* [8] proposed a task-parallel static dataflow graph execution model with multiple compute unit

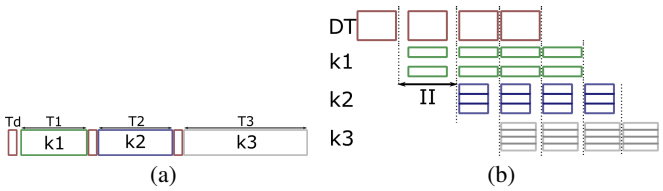

Fig. 2. Scheduling of multi-kernel application (a) without and (b) with balanced resource utilization: k1 with 2 CUs, k2 with 3 CUs, and k3 with 4 CUs.

(CU) instances, with efficient scheduling modeled as a set of difference constraints, but for single-FPGA targets and optimizing only for performance, not power.

For multi-FPGA targets, Zhang *et al.* [9] map pipelined Convolutional Neural Network (CNN) layers to a multi-FPGA platform exploring the design space for optimal performance and energy consumption using dynamic programming. They assume constant FPGA power consumption, thus reducing the problem of energy minimization to execution time minimization. We use a separate power model to minimize energy consumption for a given *II* constraint.

Tarafdar *et al.* [10] and Caulfield *et al.* [11] propose performance improvements using direct network communication between FPGAs. However, they do not optimize the power of FPGA clusters, and current PCIe-based multi-FPGA cloud platforms do not support this communication model.

II. Multi-FPGA Power Optimization

We model an application as K kernels organized in a linear pipeline, as shown in Fig. 2a (e.g., the layers of a CNN).

In a pipelined execution, all kernels compute concurrently, and simultaneous data transfers occur between the host CPU and the FPGA DDRs. The slowest pipeline stage, K3 in this example, determines the *II* of the pipeline and therefore determines the application throughput. To meet a target throughput, we aim to keep *II* below a maximum value II_{max} by balancing the resource allocation to kernels, as shown in Fig. 2b. We can split the kernel workloads into one or more CUs running concurrently, similar to OpenCL workgroups or CUDA thread groups. This execution model is well supported by commercial FPGA design tools, such as Xilinx SDAccel [4]. Power consumption depends on the number of CUs of each kernel and on their allocation to FPGAs. We seek a solution that minimizes the power for a given II_{max}. Since the target throughput can change at runtime due to dynamic changes of the workload, we determine the optimal solution for each II_{max} value in a discretized range.

A. Problem formulation

We aim to minimize the total power P_{total} while keeping the kernel pipeline initiation interval *II* shorter than II_{max} to satisfy the required throughput (1). As shown later, *II* depends on the number $n_{k,f}$ of CUs of each kernel k allocated to each FPGA f, and on the clock frequency Fck_f of each FPGA. Each CU of kernel k requires $R_{k,t}$ resources of type $t \in \{$FF, LUT, DSP, BRAM, DDR bandwidth$\}$, and must not exceed the available amount on each FPGA board, R_t (2), while the clock Fck_f of any FPGA f must be slower than the maximum supported by the board (3). Moreover, each kernel k must run on at least one CU (4)

$$II \leq II_{max} \tag{1}$$

$$\textstyle\sum_k n_{k,f} R_{k,t} \leq R_t, \qquad \forall f,\ \forall t \tag{2}$$

$$Fck_f \leq FCK, \qquad \forall f \tag{3}$$

$$CU_k = \textstyle\sum_{f=1}^{F} n_{k,f} \geq 1, \qquad \forall k. \tag{4}$$

The resulting problem is MINLP because it includes non-linear constraints, as shown next, with both integer (e.g., $n_{k,f}$) and real (e.g., Fck_f) variables.

B. Initiation interval (II) modeling

The top-level computation consists of pipelined data transfers and kernel executions. We use double buffers in the FPGA DDR so that execution can overlap data transfer with the host CPU (using single-buffering requires just a simple change of our model, and we will not discuss it here).

II is limited by the maximum among the data transfer time from host CPU to FPGA DDR forward, T_{h2f}, backward, T_{f2h}, and the CU execution time, T_{exe}

$$II \geq \max(T_{h2f} + T_{f2h}, T_{exe}). \tag{5}$$

We analyze now separately the terms in (5).

1) Execution phase: we assume that kernel workloads are arbitrarily parallelizable. It is an assumption which holds for many machine learning, big data, and scientific applications, and is well supported by the OpenCL and CUDA computation models. If $T_{wc,k}$ is the *single-CU*, *worst-case* execution time of kernel k at maximum FPGA frequency *FCK*, and the kernel workload is split over $n_{k,f}$ CUs on one or several FPGAs, then the *actual* kernel execution time in FPGA f, $T_{k,f}$, scales with the number CU_k of CUs and the actual frequency Fck_f of FPGA f (6). T_{exe} is the maximum across all kernels and FPGAs

$$T_{k,f} = \frac{T_{wc,k}}{CU_k} \cdot \frac{FCK}{Fck_f}, \qquad \forall f,\ \forall k \tag{6}$$

$$T_{exe} = \max_{k,f} T_{k,f}. \tag{7}$$

2) Data transfer phase: data transfer time, T_{h2f}, is the ratio between the total input data size transferred from host memory to the DDR of FPGA f, DI_{h2f}, and PCIe bandwidth, B_{h2f}

$$T_{h2f} = \frac{DI_{h2f}}{B_{h2f}}. \tag{8}$$

Since the AWS F1 platform does not support FPGA-to-FPGA direct data transfers, we assume that all data is transferred from CPU to FPGAs. We also assume that all CUs need the entire input data set, DI_k, which is true for CNNs and can be a worst-case assumption for other applications. Hence, we must replicate the input data if the CUs of a kernel k are allocated to multiple FPGAs, and the replication factor, α_k, is equal to the total number of FPGAs used for kernel k. The amount of data transferred in the host-to-FPGA phase is

$$DI_{h2f} = \textstyle\sum_{k=1}^{K} \alpha_k DI_k. \tag{9}$$

Similar to T_{h2f}, T_{f2h} is the ratio between the total size of output data transferred from the DDR of FPGA f to host memory, DO_{f2h}, and PCIe bandwidth, B_{f2h}

$$T_{f2h} = \frac{DO_{f2h}}{B_{f2h}}. \tag{10}$$

We assume that the output data computed by a kernel, DO_k, are equally divided among its CUs, hence we transfer

$$DO_{f2h} = \sum_{k=1}^{K} DO_k. \tag{11}$$

C. Power modeling

The average FPGA power consumption has a constant *static* component, P_s, and a *dynamic* one, P_d, including both the data transfer with the host and the FPGA processing. Total power consumption is thus

$$P_{total} = P_s + P_d. \tag{12}$$

The detailed power model is discussed in [3].

1) Static power: includes the DDR static power, P_{DDRs}, and the FPGA static power, P_{fs}. In addition, it is proportional to the number of active FPGAs, F

$$P_s = F\left(P_{DDRs} + P_{fs}\right). \tag{13}$$

2) Dynamic power: is proportional to the average dynamic energy, E_d, spent during one *II*

$$P_d = \frac{E_d}{II}. \tag{14}$$

Dynamic energy consists of DDR dynamic energy, E_{ddrd}, due to data transfer between host and DDR, and the processing energy, E_c, due to the CUs allocated on FPGA f

$$E_d = E_{ddrd} + E_c \tag{15}$$

$$E_c = \sum_f^F P_{fd,f} \cdot T_{exe} \tag{16}$$

$$P_{fd,f} = \sum_k n_{k,f} \cdot P_k \cdot \frac{Fck_f}{FCK}. \tag{17}$$

The dynamic power of FPGA f, $P_{fd,f}$, depends on the number of CUs of each kernel allocated to it, $n_{k,f}$, and scales with the clock frequency. The detailed equation for the calculation of the DDR dynamic power is discussed in [3].

III. Heuristic solutions

The optimization problem in Sec. II can be solved using a Mixed-Integer Non-Linear Programming (MINLP) solver. However, this may need a very long time to solve [3], being often impractical, especially for explorations that invoke multiple times the MINLP solver. Hence, we propose two heuristic methods to improve the exploration efficiency. The first still uses a MINLP solver, but over a much smaller exploration space. The second avoids completely the MINLP solver and is much faster.

A. First heuristic solver, using MINLP

To speed up the MINLP solver, we fix the number of active FPGAs, limit the number of possible CUs for each kernel, and simplify the power model. In our previous work [3], we empirically noticed that the best solutions save static power by always using the minimum number of FPGAs, F_{min}, most likely because of the static power consumption. Hence, our first heuristics uses F_{min} as a hard bound on resources instead of exploring allocations on more FPGAs. To obtain F_{min}, we first determine the minimum number of CUs for each kernel that satisfies the II_{max} constraint

$$CU_{min_k} = \left\lceil T_{wc,k}/II_{max} \right\rceil \tag{18}$$

and then we find F_{min} by using the resource utilization

$$F_{min} = \max_t \left\lceil \frac{\sum_{k=1}^{K} R_{k,t} \cdot CU_{min_k}}{R_t} \right\rceil. \tag{19}$$

In our experiments with CNNs and NLP, the maximum in (19) is always determined by the total number of required and available DSP units (i.e., $R_{k,t}$ and R_t, with t = DSP) on a single FPGA.

We introduce an additional binary variable in the problem, $extraCU_k = \{0, 1\}$, to limit the number of CUs per kernel, CU_k, to be at most one higher than CU_{min_k} from (18)

$$CU_k = CU_{min_k} + extraCU_k. \tag{20}$$

We do this because additional CUs may reduce the execution time of a kernel, which may become closer in speed to other kernels, hence allowing us to reduce the frequency when these kernels are all allocated to the same FPGA. With CU_k limited to only two values in (20), the possible values of the allocation variables $n_{k,f}$—the sum of which over the active FPGAs is CU_k per (4)—is also largely reduced, which has a substantial effect on the execution time of the MINLP solver.

We do not include the data-transfer power in the model because it is typically much lower than the computation power. In this way, the model is further simplified. Notice, however, that this power contribution will be implicitly minimized by our method, since the CUs of the same kernel are likely allocated in the same FPGA because they have the same execution time. As a result, the input data of that kernel will not be duplicated, and data transfer and its associated power consumption will also be reduced.

B. Second heuristic solver, without MINLP

For larger problems, even the simplified model introduced in Sec. III-A can be too slow. To further speed up the solution, we propose another heuristic method that does not rely on external solvers.

For the same reasons explained in Sec. III-A, we determine the number of active FPGAs $F = F_{min}$ as in (19), and we determine before the allocation with (18) the minimum number of CUs for each kernel, and use the auxiliary binary variable as in (20). This leads to 2^K possible combinations for the number of CUs, which we test exhaustively as shown in Alg. 1 (line 4). Notice that although this is exponential, the number of kernels K is usually small and the run time is short, as we report in the experiments. Moreover, the combinations that do not fit in the FPGAs are pruned early by the filter on line 10 to further reduce the run time.

R_{total} in the pseudo-code refers to the DSP resources needed by each kernel, which limit the allocation of computation-intensive applications, such as CNNs and transformer networks, before other resources are exhausted (LUT, FF, BRAM).

Prior to allocation, we set the FPGA resource slack R according to resource constraints (line 11). Then we sort the kernels (line 13) in descending order of the execution time obtained with CU_{min_k} CUs. In this way, we favor the allocation of kernels with similar execution time to the same FPGA, helping to reduce the operating frequency, power, and input

Algorithm 1: Second heuristic allocation method

1 ***procedure*** *AllocateCUs(*$CU_{min}, R_{total}, II_{max}$*)*
   // Vector of min required CUs per kernel
2 $CU_{min} = (CU_{1_{min}}, CU_{2_{min}}, \ldots, CU_{K_{min}})$
3 boolean $extraCU = (CU_{1_e}, CU_{2_e}, \ldots, CU_{K_e})$
4 **for** $c = 1$ *to* 2^K **do** // 2^K combinations in total, cause each kernel has two possible CUs
5 assign *extraCU* according to c
6 $alloc$ = False; $R_{total} = 0$
7 **for** $k = 1$ *to* K **do**
8 $CU_k = CU_{k_{min}} + extraCU_k$
9 R_{total} += $CU_k \cdot R_k$
10 **if** $R_{total} < F \cdot R$ **then**
   // Vector of FPGA resource slack initialized to constraint value
11 $S = (S_1, S_2, \ldots, S_F)$; $\forall f: S_f = R$
   // Allocated CUs initialized to zero
12 $\forall k, f: n_{k,f} = 0$
   // Sort by descending exec. time
13 sortCU(*CU*)
14 $k = 0$
15 **for** $f = 1$ *to* F **do**
16 $R_{acc} = 0$
17 **while** $k < K$ **do**
18 R_{acc} += $CU_k \cdot R_k$
19 **if** $R_{acc} \leq R$ **then**
20 $\delta CU_k = CU_k$
21 $S_f = S_f - CU_k \cdot R_k$
22 $CU_k = 0$
23 $n_{k,f} = \delta CU_k$
24 **else**
25 $\delta CU_k = \lfloor S_f / CU_k \rfloor$
26 $CU_k = CU_k - \delta CU_k$
27 $S_f = S_f - \delta CU_k \cdot R_k$
28 $n_{k,f} = n_{k,f} + \delta CU_k$
29 break;
30 $k = k + 1$
31 **if** $\sum_k CU_k > 0$ **then**
32 $k = K - 1$
33 **for** $f = F$ *to* 1 **do**
34 $R_{acc} = 0$
35 **while** $k > 0$ *and* $CU_k > 0$ **do**
36 R_{acc} += $CU_k \cdot R_k$
37 **if** $R_{acc} \leq S_f$ **then**
38 $\delta CU_k = CU_k$
39 $S_f = S_f - CU_k \cdot R_k$
40 $CU_k = 0$
41 $n_{k,f} = \delta CU_k$
42 **else**
43 $\delta CU_k = \lfloor S_f / CU_k \rfloor$
44 $CU_k = CU_k - \delta CU_k$
45 $S_f = S_f - \delta CU_k \cdot R_k$
46 $n_{k,f} = n_{k,f} + \delta CU_k$
47 break;
48 $k = k - 1$
49 **if** $\sum_k CU_k = 0$ **then**
50 $alloc$ = True
51 Calculate $T_{f2h} + T_{h2f}$
52 **if** *alloc and* $T_{f2h} + T_{h2f} < II_{max}$ **then**
53 Update Fck_f
54 Calculate *Power*
55 $n_{k,f} = \arg(\min(Power))$

data duplication (lines 15 to 30). At the end of this loop, some kernels might have residual CUs that could not be allocated. Therefore, we go through the kernels in the opposite order, allocating the kernels with the smallest execution time to FPGAs in increasing operating frequency order (lines 33 to 50). Note that the kernels are ordered in descending order of execution time on FPGAs, and they are allocated on FPGA in the same order, so the first FPGA will have the highest frequency, and the last one the slowest frequency. We accept only the allocations with total data transfer times below II_{max} (lines 51 to 54), and we select the allocation with minimum power consumption.

TABLE I
TRANSFORMER KERNEL CHARACTERIZATION ON THE AWS F1 PLATFORM

Kernels	BRAM (%)	DSP (%)	T_{wc} (ms)	Bw / Br (%)	tw / tr (ms)	bw / br (%)	P_k (W)
Attention1	20.9	31.5	9.5	1.23 / 0.39	8.9 / 0.3	1.03 / 0.004	4.12
Attention2	9	16.5	6.3	9.52 / 0.89	0.59 / 0.12	1.03 / 0.004	3.04
feed_forward1	0.9	3.7	16.7	77.4 / 3.63	0.28 / 0.11	0.275 / 0.01	0.85
feed_forward2	0.9	3.7	16.8	11.5 / 0.95	1.94 / 0.11	0.28 / 0.011	0.85
norm	0.5	0.5	0.3	20.5 / 10.24	0.1 / 0.1	0.315 / 0.16	0.92

TABLE II
OPTIMIZATION TIME BY THREE DIFFERENT METHODS.

DNNs	MINLP	H1	H2
ALEX-16	1 h	20 min	0.4 s
ALEX-32	*24 h	*15 h	2 s
TRANSFORMER-16	*24 h	*10 h	3 s
VGG-16	*30 h	*15 h	5 s

* stopped at the solver time limit.

IV. EXPERIMENTS

We use the same MINLP solver from [3] to implement the first heuristic method (see Sec. III-A), and we implemented the second one in C++ (see Sec. III-B). We validate the methods using two widely used CNNs, 8-layer AlexNet [12] (32-bit floating-point, ALEX-32, and 16-bit floating-point, ALEX-16) and 17-layer VGG [13] (16-bit floating-point, VGG-16), and a transformer network for NLP [14] (16-bit fixed-point, one decoder, one encoder, four heads in the attention layer, 11 layers total, TRANSFORMER-16). TABLE I shows the transformer kernel characterizations on the AWS F1 platform. Kernel characterizations for the other three implementations are reported in [3]. The execution time is measured on the AWS platform. Since the power measurement is not available on AWS F1, we rely on an estimation made using datasheets (e.g. for the DDR) and the Xilinx power analysis tools.

Fig. 3 shows the experimental results obtained on a CentOS Linux v6.9 machine with an Intel Core i7-6900K processor with 128 GB RAM. MINLP denotes the optimization with the method described in [3], H1 denotes the first heuristic method, and H2 denotes the second heuristic method. Figs. 3a–3d show the minimum power consumption obtained for a range of *II* using each optimization method. TABLE. II shows the optimization time for each application and for different methods. Figs. 3e–3h show the corresponding number of FPGAs. For small-size problems, e.g., ALEX-16 and ALEX-32, all optimization methods yield the same result, proving the effectiveness of the proposed heuristics. However, these are much faster than MINLP.

For larger problems, e.g., TRANSFORMER-16 in Fig. 3c, H1 and H2 find better results, especially for $II = 2$ ms. H2 returns a suboptimal solution for $II = 4$ ms, but it finishes

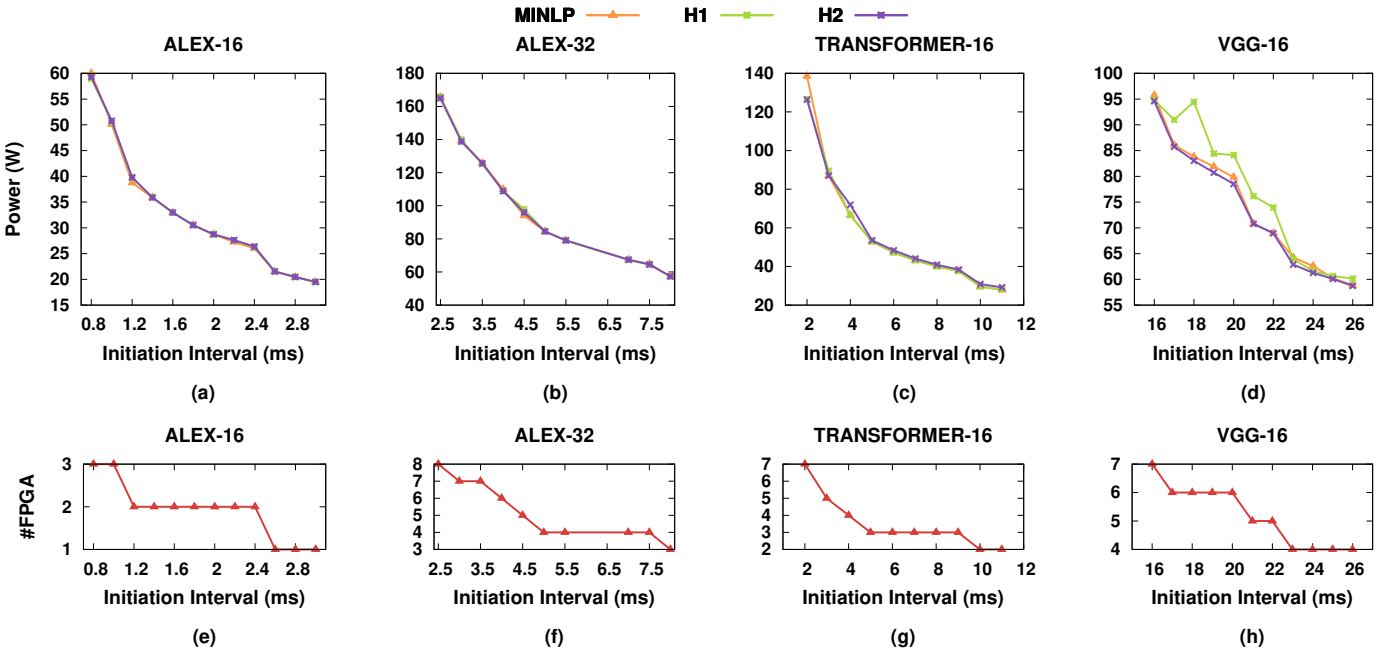

Fig. 3. Minimum power and number of FPGAs function of the initiation interval obtained with optimization method in [3] (MINLP), first (H1) and second (H2) heuristic methods for (a) 16-bit floating-point AlexNet, (b) 32-bit floating-point AlexNet, (c) 16-bit fixed-point Transformer, and (d) 16-bit floating-point VGG.

comparatively much faster, in a few seconds. For even larger problems, e.g., VGG-16 in Fig. 3d, H1 often misses the best solution by roughly $12\,\%$ compared to MINLP. H2 solutions are the best with run times around $5\,\mathrm{s}$.

Summarizing, for small problems both our heuristic methods, H1 and H2, find good solutions. H1 may even obtain better solutions, such as for $II = 4\,\mathrm{ms}$ for TRANSFORMER-16. For larger problems, H2 is better and much faster.

Note that for larger problems we had to stop MINLP early. Hence, the results in Fig. 3d are the best after $24\,\mathrm{h}$ to $30\,\mathrm{h}$, but still sub-optimal. This explains why the proposed heuristic methods H1 and H2 may reach better results (e.g., for VGG-16).

The number of FPGAs used for the best solution achieved using the three different methods are shown in Figs. 3(e)-(h). In all cases, the optimal number of FPGAs is the same.

V. Conclusion

We propose two heuristic methods to efficiently obtain energy-optimal or near-optimal solutions to configure a multi-FPGA platform for a given *II*. The first heuristics constrains the exploration space to significantly reduce the runtime, while achieving the same or even better results than the exact algorithm. Similarly, the second heuristics first reduces the exploration space, then groups the kernels with similar execution time on a single FPGA to minimize the FPGA frequency, thus minimizing the power consumption. In addition, it is thousands of times faster than the exact algorithm.

References

[1] X. Tang, X. Liao, J. Zheng, and X. Yang, "Energy efficient job scheduling with workload prediction on cloud data center," *Cluster Computing*, no. 3, pp. 1581–1593, 2018.

[2] H. Wang, J. Pannereselvam, L. Liu, Y. Lu, X. Zhai, and H. Ali, "Cloud workload analytics for real-time prediction of user request patterns," in *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 2018, pp. 1677–1684.

[3] J. Shan, M. T. Lazarescu, J. Cortadella, L. Lavagno, and M. R. Casu, "Power-optimal mapping of cnn applications to cloud-based multi-fpga platforms," *IEEE Trans. Circuits Syst. II*, vol. 67, no. 12, pp. 3073–3077, 2020.

[4] "SDAccel Development Environment." [Online]. Available: https://www.xilinx.com/products/design-tools/software-zone/sdaccel.html

[5] J. Shan, M. T. Lazarescu, J. Cortadella, L. Lavagno, and M. R. Casu, "Cnn-on-aws: Efficient allocation of multi-kernel applications on multi-fpga platforms," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, pp. 1–1, 2020.

[6] J. Shan, M. R. Casu, J. Cortadella, L. Lavagno, and M. T. Lazarescu, "Exact and heuristic allocation of multi-kernel applications to multi-fpga platforms," in *Proc. 56th Annual Design Automation Conference 2019*, ser. DAC '19. New York, NY, USA: ACM, 2019.

[7] B. Li, S. Pandey, H. Fang, Y. Lyv, J. Li, J. Chen, M. Xie, L. Wan, H. Liu, and C. Ding, "Ftrans: Energy-efficient acceleration of transformers using fpga," in *Proc. 2020 Int. Symp. on Low Power Electronics and Design*, ser. ISLPED '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 175–180.

[8] J. Cong, M. Huang, and P. Zhang, "Combining computation and communication optimizations in system synthesis for streaming applications," in *Proc. 2014 ACM/SIGDA Int. Symp. on FPGAs*, 2014, pp. 213–222.

[9] C. Zhang, D. Wu, J. Sun, G. Sun, G. Luo, and J. Cong, "Energy-efficient cnn implementation on a deeply pipelined fpga cluster," in *Proc. 2016 Int. Symp. on Low Power Electronics and Design*, ser. ISLPED '16. New York, NY, USA: ACM, 2016, pp. 326–331.

[10] N. Tarafdar, T. Lin, E. Fukuda, H. Bannazadeh, A. Leon-Garcia, and P. Chow, "Enabling flexible network fpga clusters in a heterogeneous cloud data center," in *Proc. 2017 ACM/SIGDA Int. Symp. on FPGAs*. New York, NY, USA: ACM, 2017, pp. 237–246.

[11] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, "A cloud-scale acceleration architecture," in *49th IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, Oct 2016, pp. 1–13.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[13] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.