Power-Optimal Mapping of CNN Applications to Cloud-Based Multi-FPGA Platforms

Power-Optimal Mapping of CNN Applications to Cloud-Based 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*—Multi-FPGA platforms like Amazon Web Services F1 are perfect to accelerate multi-kernel pipelined applications, like Convolutional Neural Networks (CNNs). To reduce energy consumption, we propose to upload at runtime the best power-optimized CNN implementation for a given throughput constraint. Our design method gives the best number of parallel instances of each kernel, their allocation to the FPGAs, the number of powered-on FPGAs and their clock frequency. This is obtained by solving a mixed-integer, non-linear optimization problem that models power and performance of each component, as well as the duration of the computation phases—data transfer between a host CPU and the FPGA memory (typically DDR), data transfer between DDR and FPGA, and FPGA computation. The results show that the power saved compared to simply clock gating the fastest implementation is obviously very high, but it is also much more significant than simply scaling the frequency of the fastest implementation or replicating the slowest implementation on multiple FPGAs.**

***Index Terms*—CNN, Multi-FPGA, power optimization.**

I. Introduction

MOST data center applications can be easily parallelized, e.g., deep neural networks, big data processing and analysis, scientific (finite element analysis), and energy is a significant running and environmental cost. To offset this while maintaining the performance, cloud providers (Amazon, Alibaba, Microsoft) offer low cost multi-Field Programmable Gate Array (FPGA) platforms. FPGAs are less energy efficient than ASICs, but incomparably more configurable, suitable to support rapid application evolution.

Data center workloads vary and accelerators designed for the highest application throughput may be underutilized most of the time, wasting both FPGA resources and energy. Clock gating and frequency scaling can lower energy consumption, but FPGA reconfiguration adapted to application throughput can lower it even more. Throughput is the inverse of the Initiation Interval (*II*), hence a smaller *II* means a faster throughput.

Fig. 1 outlines a multi-FPGA platform. Here the host CPU controls eight FPGAs over a PCI-express (PCIe) bus and can quickly ($\approx 100\,\mathrm{ms}$) reconfigure them with one of several configurations generated offline, adapting them to the actual application performance needs. Reconfiguration is most likely infrequent, e.g., once per minute (or hour), but it optimizes the number of active FPGAs and their clock to spare energy.

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, Barcelona, Spain, e-mail: jordi.cortadella@upc.edu

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

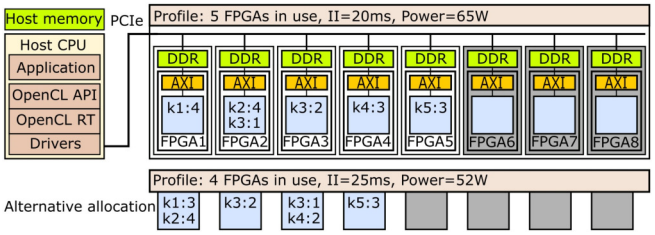

Fig. 1. Multi-FPGA configurations for different power-performance profiles.

Energy-per-computation is the product of power times the initiation interval. Hence at fixed *II*, minimum power is also minimum energy-per-computation. Since we provide the full power and energy-per-computation versus *II* curves, other choices can be made (e.g., find the best *II* to minimize energy-per-computation), according to the application requirements.

We propose a flow to obtain power-optimized multi-FPGA configuration bitstreams that satisfy different application *II* requirements. We consider applications that can be modeled as multi-kernel task-level pipelines, and among these we focus our experiments on Convolutional Neural Networks (CNNs). Each task, which corresponds to a CNN layer, can be computed by parallel kernel instances, termed Compute Units (CUs). They are shown in Fig. 1 as k3:2, k4:3, etc., indicating how many CUs of each kernel are allocated on each FPGA (e.g., k3:2 in FPGA3 means the allocation of two CUs of kernel 3 on it).

After characterizing the multi-FPGA environment and kernels for power, performance, resources, etc., we build a power-performance model that considers both computation and data transfers. Then we solve a Mixed-Integer Non-Linear Problem (MINLP) that, given an *II* constraint, finds the allocation of the CUs to FPGAs and the clock frequency of each FPGA that minimize power and so also energy per computation.

We compare this strategy to two alternatives: 1) finding the fastest multi-FPGA implementation and applying frequency scaling to reduce energy when the *II* requirement decreases; 2) finding the fastest single-FPGA implementation and replicating it on the minimum number of FPGAs needed to meet the *II* constraint. Fig. 2 compares the three strategies showing an allocation example for AlexNet [1] convolutional layers. The fastest solution (not shown in figure) achieves $II = 0.8\,\mathrm{ms}$ with three FPGAs (F1–F3) each running at a fast and individually optimized clock frequency. But if the application requires $II = 1.4\,\mathrm{ms}$, frequency scaling applied to the fastest solution (middle) consumes 14% more power than an optimized configuration (left), which uses only two FPGAs (F1, F2) at a higher clock frequency. The replication solution (right) is also less efficient

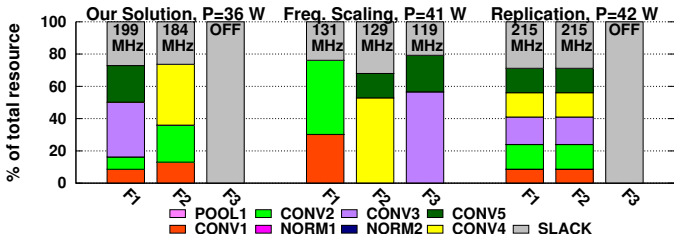

Fig. 2. Comparison of different power optimization strategies for AlexNet.

and consumes 17% more power than our solution.

Resource allocation is a well-studied problem for high-performance data centers with heterogeneous hardware (CPUs with Graphical Processing Unit (GPU) or FPGA accelerators). Tesfatsion *et al.* [2] provide a resource management framework with a hardware scheduler and an optimizer for FPGA-accelerated clouds. Similar to our work, they split workloads into "chunks" run by Virtual Machines on CPUs and sharing FPGA accelerators. But they do not pipeline chunk execution and consider only the FPGA static power.

Zhang *et al.* [3] map pipelined CNN layers to a multi-FPGA platform exploring the design space for optimal performance and energy with dynamic programming. However, they do not explicitly minimize power or energy. Also, they use First In First Out queues (FIFOs) for inter-layer communications, which require in-order production and consumption of activation values. This may be difficult to achieve, and is not supported by current multi-FPGA cloud platforms like Amazon AWS F1 (FPGA-to-FPGA transfers must be mediated by the CPU). On the other hand, we model inter-kernel communication using memory arrays, which is arguably a more general and natural programming model, supported by C, C++, and OpenCL.

The execution model in [4] exploits, like us, application parallelism at task, data, and pipeline level, but they target processors instead of FPGAs. Furthermore, a compiler decides the allocation through heuristic moves, while we solve an optimization problem. A task-parallel static dataflow graph execution model with multiple CU instances is proposed in [5] for FPGA targets, with efficient scheduling formulated as a set of difference constraints. But it does not consider multi-FPGA platforms and optimizes only performance, not power.

For multi-FPGA targets, [6], [7] propose to improve performance by using direct network communication between FPGAs. However, they do not optimize the power of the FPGA clusters, and again this communication model is not offered by current PCIe-based multi-FPGA cloud platforms.

II. Multi-FPGA Power Optimization

We model CNN layers as K kernels organized in a linear pipeline, including Data Transfer (DT) stages between the host CPU and the FPGA DDRs (see Fig. 3). The slowest stage sets the *II* of the pipeline (here the bottleneck is k1, but it could also be DT). To reduce *II*, we split the kernel workloads into one or more CUs running concurrently, like OpenCL workgroups or CUDA thread groups (see Fig. 4(a)), and allocate them to the FPGAs (see Fig. 4(b)). This execution model is well supported by commercial FPGA design tools,

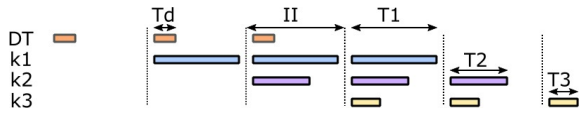

Fig. 3. CNNs modeled as pipelines of kernels, including data transfer DT.

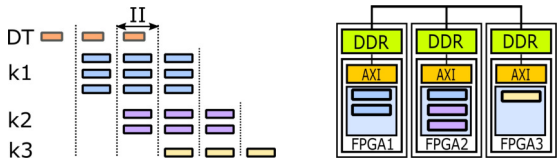

Fig. 4. Kernels are split into multiple compute units allocated on FPGAs.

e.g., Xilinx SDAccel [8], and it approximately divides the computation time by N when allocating N CUs to each layer (data transfer times are accounted for separately in our model). We design a custom IP for each layer grouping convolution, pooling, and normalization in a single kernel.

Power consumption depends on the number of CUs of each kernel and their allocation to FPGAs. We seek the solution that minimizes power for a given *II*. Since the *II* target can change at runtime, we find the optimal solution for each *II* value in a discretized range. Fig. 5 shows the proposed design flow. From a C++ or OpenCL high-level description of kernels, we use Xilinx SDAccel to profile their implementation: FPGA resource utilization (LUTs, FFs, DSPs, BRAMs), DDR memory bandwidth, execution time, etc. We enter the profile and target platform characteristics (AWS F1 x8.large in our experiments, which is the largest publicly available cloud FPGA platform) into our power and performance model, then use a MINLP solver to find the configuration with minimum power for each value of *II* (the points in the graph inset in Fig. 5). Finally, for each configuration we generate the configuration bitstream.

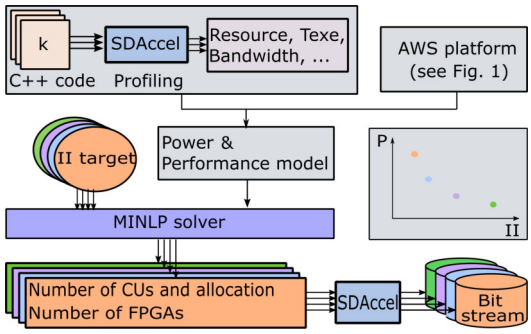

Fig. 5. Design flow to obtain the power-optimal FPGA configurations.

A. Problem formulation

We aim to minimize the total power while keeping the 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 (where $t \in$ {FF, LUT, DSP, BRAM, DDR bandwidth}) and must not exceed the available amount on each FPGA R_t (2), while the clock Fck_f of any

FPGA f must be slower than the maximum supported *FCK* (3). Moreover, each kernel k must run on at least one CU (4).

$$II \leq II_{max} \tag{1}$$
$$\textstyle\sum_k n_{k,f} R_{k,t} \leq R_t, \qquad \forall f,\ \forall t \tag{2}$$
$$Fck_f \leq FCK, \qquad \forall f \tag{3}$$
$$N_k = \textstyle\sum_{f=1}^{F} n_{k,f} \geq 1, \qquad \forall k. \tag{4}$$

The resulting problem is a MINLP one because it includes integer ($n_{k,f}$) and real (Fck_f) variables and non-linear constraints.

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 it will not be discussed further).

II is limited by the maximum among the data transfer time from host CPU to FPGA DDR T_{h2f} and back T_{f2h}, and the CU execution time T_{exe}. Execution can overlap with data transfer (Fig. 3), but all data transfers are managed by the CPU, hence

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

We now analyze separately the terms in (5).

1) Execution phase: We assume that kernel workloads are arbitrarily parallelizable via *doall* top-level loops, which is applicable not only to CNNs but also to many machine learning, big data, and scientific applications, and is well supported by the OpenCL and CUDA models of computation. If $T_{wc,k}$ is the *single-CU* 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 f, then the *actual* kernel execution time in FPGA f, $T_{k,f}$, scales with the number N_k of CUs and the actual frequency Fck_f of FPGA f (7), and T_{exe} is the maximum across all kernels and FPGAs (8)

$$\delta_{k,f} = \begin{cases} 1 & \text{if } n_{k,f} > 0 \\ 0 & \text{otherwise} \end{cases}, \qquad \forall f,\ \forall k \tag{6}$$
$$T_{k,f} = \tfrac{T_{wc,k}}{N_k} \cdot \tfrac{FCK}{Fck_f} \cdot \delta_{k,f}, \qquad \forall f,\ \forall k \tag{7}$$
$$T_{exe} = \max_{k,f} T_{k,f}. \tag{8}$$

Here, the allocation variable $\delta_{k,f}$ is 1 if at least one CU of kernel k is allocated to FPGA f, and 0 otherwise.

2) Host-to-FPGA and FPGA-to-Host phases: T_{h2f} is the ratio between the total size of input data from the host memory to the DDR of the FPGAs, DI_{h2f}, and the PCIe bandwidth, B_{h2f}. Similarly, T_{f2h} is the ratio between the total size of output data from the DDR of the FPGAs to the host memory, DO_{f2h}, and the PCIe bandwidth, B_{f2h}

$$T_{h2f} = \tfrac{DI_{h2f}}{B_{h2f}}, \quad T_{f2h} = \tfrac{DO_{f2h}}{B_{f2h}}. \tag{9}$$

In this paper we assume the worst case, namely that direct data transfers between FPGA DDRs are not supported, since this is the case for the AWS F1 platform (again, relaxing this assumption requires a minor change to the model, and will not be discussed further). 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

$$\alpha_k = \textstyle\sum_{f=1}^{F} \delta_{k,f}, \qquad \forall k. \tag{10}$$

The data transferred in the host-to-FPGA phase amount to

$$DI_{h2f} = \textstyle\sum_{k=1}^{K} \alpha_k DI_k. \tag{11}$$

Note that constant data (e.g., weights and bias in CNNs) are not considered, since they are transferred once during initialization.

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

$$DO_{f2h} = \textstyle\sum_{k=1}^{K} DO_k. \tag{12}$$

TABLE I
VARIABLES USED IN POWER MODEL EQUATIONS.

Notation	Description
P_{total}	total power
Fck_f	actual working frequency of FPGA f
T_{h2f}, T_{f2h}	data transfer time host to DDR and DDR to host, resp.
T_{exe}	execution time
E_{h2f}, E_{f2h}	energy spent during T_{h2f} and T_{f2h}, resp.
E_{exe}	total energy spent during T_{exe}
E_{rw}	energy spent by accesssing to DDR during T_{exe}
DI_{h2f}	total data transferred in the host-to-FPGA phase
DO_{f2h}	total data transferred in the FPGA-to-host phase
P_{DDRdr}, P_{DDRdw}	DDR dynamic power when reading and writing, resp.
P_{fs}	on-chip static power
$P_{fd,f}$	dynamic power of FPGA f

C. Power modeling

FPGA-related average power consumption has a constant *static* contribution, P_s, and a *dynamic* one, P_d, accounting for both data transfer with the host and the FPGA processing. TABLE I shows all the variables involved in the power model.

1) DDR power model: we obtained the FPGA DDR power using a calculator [9] and from experiments on the AWS F1 platform, which includes an API to report power consumption. Idle DDR consumes only static power, P_{DDRs}, while dynamic power depends on the normalized read B_r and write B_w bandwidths (i.e., $B_r = 1$ if the maximum bandwidth is used for reading), and is the sum of the corresponding P_{DDRdr} and P_{DDRdw}. The equations that we used, with coefficients expressed in Watt and coming from the characterization above, are: $P_{DDRs} = 0.5$, $P_{DDRdr}(B_r) = 0.672 B_r$ and $P_{DDRdw}(B_w) = 0.4 B_w$.

2) Single FPGA power: it consists of static and dynamic power obtained with the Xilinx Power Estimator (XPE) [10]. Static power P_{fs} includes logic P_{fls} and memory I/O P_{fio} power

$$P_{fs} = P_{fls} + N_{fio} \cdot P_{fio} \tag{13}$$

with $N_{fio} = 4$ DDR banks per FPGA. One I/O bank consumes $P_{fio} = 0.414\,\text{W}$ from [10] and logic power is $P_{fls} = 2.842\,\text{W}$.

Dynamic power $P_{fd,f}$ of FPGA f depends on each kernel's CUs allocated to f, $n_{k,f}$, and scales with clock frequency

$$P_{fd,f} = \textstyle\sum_k n_{k,f} \cdot P_k \cdot \tfrac{Fck_f}{FCK} \tag{14}$$

with P_k the dynamic power of one CU of kernel k when running at the highest clock frequency *FCK*.

3) Multi-FPGA power: static power P_s depends only on the number of active FPGAs, N_F:

$$P_s = N_F \left(P_{DDRs} + P_{fs}\right). \tag{15}$$

Total dynamic power P_d depends on the energy spent during data transfer from host to FPGA DDRs E_{h2f}, processing E_{exe}, and data transfer from FPGA DDRs to host E_{f2h}

$$P_d = \frac{E_{h2f}+E_{exe}+E_{f2h}}{II} = \frac{E_d}{II}. \tag{16}$$

Here, E_{h2f} depends on the data replication factor α_k (10), DDR write bandwidth Bw_k, and transfer time tw_k obtained from the FPGA profiling reports

$$E_{h2f} = \sum_k^K \alpha_k \cdot P_{DDRdw}(Bw_k) \cdot tw_k. \tag{17}$$

Similarly, E_{f2h} depends on DDR read bandwidth Br_k, and transfer time tr_k also from profiling (note that there is no output data replication)

$$E_{f2h} = \sum_k^K P_{DDRdr}(Br_k) \cdot tr_k. \tag{18}$$

CU execution energy E_{exe} includes the energy to read/write data on DDR E_{rw} and the FPGA processing energy E_c

$$E_{rw} = \sum_k^K N_k \left(P_{DDRdr}(br_k) + P_{DDRdw}(bw_k)\right) \cdot T_{exe} \tag{19}$$

$$E_c = \sum_f^F P_{fd,f} \cdot T_{exe} \tag{20}$$

$$E_{exe} = E_{rw} + E_c. \tag{21}$$

where the data transfer bandwidths that the CUs of kernel k use to read from and write to DDR are br_k and bw_k, respectively.

Total power consumption P_{total} is given by the static power from (15) and the dynamic power from (16)

$$P_{total} = P_s + P_d. \tag{22}$$

The energy per computation is $E_{comp} = P_{total} \cdot II$.

III. Experiments

We check our optimization method against *frequency scaling*, *clock gating*, and *replication* for two widely used and realistically large CNNs, AlexNet [1] and VGGNet [11]. However, note that our technique is not specific to CNNs (even though we evaluate it for well-known CNNs), and only depends on the assumption of *arbitrarily parallelizable kernel pipelines mapped to multiple FPGAs with DDR-based memory transfers*. We show results for AlexNet using 32-bit floating-point and 16-bit fixed-point, and VGGNet using only fixed-point. To solve the minimization problem in (1)–(3) we use a state-of-the-art MINLP solver, Couenne [12]. Note that a MINLP problem can have in principle multiple local minima. We tried running the solver multiple times, but the result was always the same.

We characterize the kernels for the power and performance model discussed in Sec. II using the FPGA profiling reports from Xilinx SDAccel and actual measurements on an Amazon AWS F1 x8-large instance with eight Xilinx UltraScale+ FPGAs, each with four DDR banks and a PCIe connection to the host CPU (see Fig. 1). MINLP and SDAccel run on CentOS Linux 6.9 on a 16-core Intel Core i7-6900K @3.2GHz.

The MINLP solver requires ≈1 hour to optimize one AlexNet fixed-point implementation, ≈1 day for AlexNet floating-point, and ≈30 hours for VGGNet. Note that in all our experiments the solver reaches quickly (after around half an hour) a reasonably good solution, usually within 5% of the best found after multi-day runs. So even if the problem size increases, we can still efficiently get a good solution in a reasonable amount of time. Note also that the solver run time is not critical if it is less than one day, because the characterization of the optimal allocations is done only once, offline, then the networks can run for months on many boards in the cloud. Moreover, this time is comparable with the time required by physical design for several large FPGAs. Faster heuristics are left to future work.

Characterization data from AWS executions for the AlexNet and VGG benchmarks are shown in TABLE II and TABLE III, respectively. The performance of the optimization methods is shown Fig. 6, and the number of FPGAs used as a function of the *II* is shown in Fig. 7. The labels in the figure captions are:

- *Our Solution* is the MINLP optimum using the design flow in Fig. 5; note that this is most likely a local minimum, since the optimization space is not convex;
- *Freq. Scaling* scales down the clock frequency of the fastest-*II* MINLP solution to meet each actual *II* requirement; note that the AWS platform does not support voltage scaling; including it into our model would be a simple modification;
- *Clock Gating* stops the FPGA clock when the CUs of the fastest-*II* MINLP solution finish computation; note that the AWS platform does not support power gating at runtime; since static power is ≈20% of the total power, considering power gating would bring *Clock Gating* closer to *Freq. Scaling*, but still far from *Our Solution*.
- *Replication* makes copies of the MINLP solution that uses the minimal number of FPGAs (hence the slowest solution) until it meets the *II* requirement.

By design, 1) all plots in Fig. 6 except for *replication* start at the best performance point, and 2) *replication* meets *our solution* at the worst performance point. Note that *our solution* always yields equal or superior results to other methods.

Freq. scaling and *clock gating* show a reciprocal dependency between power and *II*, because they keep the same number of kernel CUs, the same FPGA allocation, and the same number of active FPGAs. They satisfy the *II* constraint either by scaling the FPGA clock frequency only, or by disabling the clock in addition to scaling it. Unlike them, *our solution* saves more power when the *II* constraint is relaxed, because it optimizes both the number and allocation of CUs, the number of active FPGAs, and their working frequencies *at the same time*.

Replication starts from the optimal results obtained using *our solution* for the highest *II*. For both AlexNet implementations in Fig. 6(a) and Fig. 6(b), *replication* finds better solutions than *freq. scaling* and *clock gating* for *II* constraints when the execution time is much higher than data transfer time. In fact, from (7) the execution time is inversely proportional to CU number, while from (9)–(11) T_{h2f} is proportional to the number of CUs, because the input data are replicated with the same factor. VGGNet [Fig. 6(c)] shows an extreme case when data transfer time is very high and solution replication mostly increases power by increasing the number of CUs, without a significant reduction of the *II*, since it is dominated by data transfer time. However, as shown in Fig. 2, *our solution* smartly

TABLE II
ALEXNET 32-BIT FLOATING-POINT AND ALEXNET 16-BIT FIXED-POINT KERNEL CHARACTERIZATION. LAYERS: CONVOLUTIONAL CONV, POOLING POOL.

	Alex-32							Alex-16						
Kernels	**BRAM (%)**	**DSP (%)**	**T_{wc} (ms)**	**Bw/Br (%)**	**tw/tr (ms)**	**bw/br (%)**	**P_k (W)**	**BRAM (%)**	**DSP (%)**	**T_{wc} (ms)**	**Bw/Br (%)**	**tw/tr (ms)**	**bw/br (%)**	**P_k (W)**
Conv1	13.07	21.24	13	26.54 / 21.96	0.2 / 0.55	0.193 / 0.130	4.542	10.59	4.31	5.16	16.19 / 15.42	0.2 / 0.39	0.209 / 0.052	1.004
Pool1	2.84	0	1.78	41.48 / 13.62	0.29 / 0.21	1.415 / 0.341	0.633	0.05	0	1.78	26.86 / 8.79	0.23 / 0.17	0.709 / 0.171	0.605
Norm1	6.1	2.11	0.839	12.5 / 11.39	0.23 / 0.26	0.725 / 0.725	1.091	2.53	0.06	0.78	6.26 / 9.62	0.23 / 0.15	0.389 / 0.388	0.596
Conv2	8.73	37.59	7.19	8.45 / 9.4	0.35 / 0.19	0.54 / 0.052	8.367	4.39	7.63	4.11	7.08 / 7.77	0.22 / 0.12	0.475 / 0.046	1.438
Norm2	7.75	2.11	0.807	9.32 / 9.92	0.19 / 0.18	0.466 / 0.466	1.252	6.66	0.06	0.67	4.59 / 7.63	0.2 / 0.12	0.281 / 0.279	0.664
Conv3	5.22	28.13	7.78	7.92 / 26.33	0.23 / 0.1	1.18 / 0.072	6.173	2.63	5.66	6.7	1.864 / 14.4	0.5 / 0.09	0.684 / 0.042	1.109
Conv4	2.13	37.5	9.08	6.63 / 12.7	0.4 / 0.21	1.063 / 0.073	7.979	1.91	7.55	5.06	5.346 / 14.5	0.256 / 0.09	0.737 / 0.056	1.274
Conv5	8.73	37.5	4.84	3.77 / 4.38	0.38 / 0.09	1.027 / 0.017	8.150	4.39	7.55	3.29	2.93 / 2.08	0.24 / 0.09	0.755 / 0.012	1.340

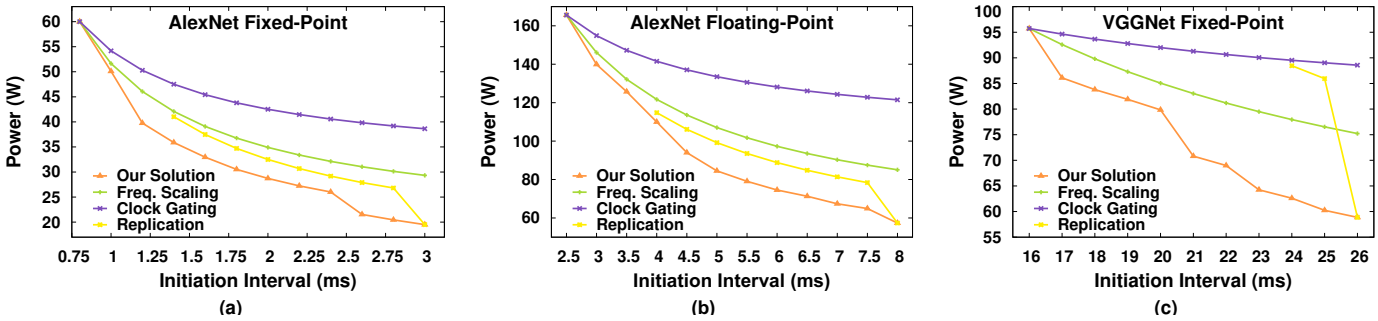

Fig. 6. Power versus initiation interval in (a) AlexNet Fixed-Point, (b) AlexNet Floating-Point, and (c) VGGNet Fixed-Point.

TABLE III
VGGNET 16-BIT FIXED-POINT KERNEL CHARACTERIZATION RESULTS

Kernels	**BRAM (%)**	**DSP (%)**	**T_{wc} (ms)**	**Bw/Br (%)**	**tw/tr (ms)**	**bw/br (%)**	**P_k (W)**
Conv1	3.67	2.95	28.8	17.33 / 24.79	0.18 / 2.70	0.028 / 0.484	0.914
Conv2	9.97	15.14	67.8	83.20 / 22.78	0.80 / 2.94	0.321 / 0.206	2.106
Pool2	11.6	0.03	13.3	83.86 / 23.33	0.80 / 0.72	1.045 / 0.261	0.825
Conv3	9.97	15.14	22.7	49.28 / 24.47	0.34 / 1.37	0.269 / 0.307	2.108
Conv4	9.97	15.14	32.1	78.28 / 24.26	0.46 / 1.38	0.380 / 0.217	2.107
Pool4	2.94	0.03	6.9	72.86 / 22.67	0.92 / 0.74	1.020 / 0.254	0.714
Conv5	8.32	15.07	22.8	23.37 / 22.52	0.36 / 0.74	0.341 / 0.153	2.055
Conv6-7	8.32	15.05	32.9	44.44 / 23.33	0.38 / 0.72	0.472 / 0.106	2.063
Pool7	1.50	0.03	3.5	56.74 / 17.18	0.29 / 0.24	0.985 / 0.246	0.615
Conv8	2.12	15.02	24.5	8.986 / 19.06	0.47 / 0.44	0.455 / 0.071	1.982
Conv9-10	2.12	15.02	37.7	10.93 / 19.28	0.77 / 0.43	0.590 / 0.046	1.979
Pool10	0.05	0.01	2.1	31.87 / 11.84	0.26 / 0.18	0.800 / 0.200	0.582
Conv11-13	2.12	14.99	20.3	3.319 / 10.96	0.63 / 0.19	0.629 / 0.022	1.986

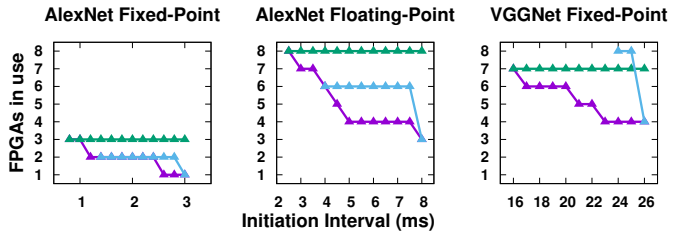

Fig. 7. Number of used FPGAs as a function of the optimization method (FS is frequency scaling, CK is clock gating) and the target initiation interval.

groups the kernel CUs on fewer FPGAs, minimizing both data transfer time and power at the same time.

IV. CONCLUSION

We proposed a power-performance optimization method to optimally configure a multi-FPGA platform running multi-kernel pipelined workloads. Given an *II* target, the solution of a MINLP problem provides an optimal allocation of the best number of CUs for each kernel so as to minimize the overall power consumption. Compared to applying frequency scaling to reduce both *II* and power starting from a fast configuration, or to replicating a slow configuration on multiple FPGAs, our solution provides a much more effective way of saving power.

REFERENCES

[1] 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.
[2] S. K. Tesfatsion *et al.*, "Power and performance optimization in fpga-accelerated clouds," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 18, p. e4526, 2018.
[3] C. Zhang *et al.*, "Energy-efficient CNN implementation on a deeply pipelined FPGA cluster," in *Proc. 2016 Int. Symp. on Low Power Electronics and Design*. ACM, 2016, pp. 326–331.
[4] M. I. Gordon, W. Thies, and S. Amarasinghe, "Exploiting coarse-grained task, data, and pipeline parallelism in stream programs," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 5, pp. 151–162, Oct. 2006.
[5] 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.
[6] N. Tarafdar *et al.*, "Enabling flexible network fpga clusters in a heterogeneous cloud data center," in *Proc. 2017 ACM/SIGDA Int. Symp. on FPGAs*. ACM, 2017, pp. 237–246.
[7] A. M. Caulfield *et al.*, "A cloud-scale acceleration architecture," in *49th IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, Oct 2016, pp. 1–13.
[8] "SDAccel Development Environment." [Online]. Available: https://www.xilinx.com/products/design-tools/software-zone/sdaccel.html
[9] "Power Calculators." [Online]. Available: https://www.micron.com/support/tools-and-utilities/power-calc
[10] "Xilinx Power Estimator (XPE)." [Online]. Available: https://www.xilinx.com/products/technology/power/xpe.html
[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[12] P. Belotti. (2018) Couenne (convex over and under envelopes for nonlinear estimation). [Online]. Available: https://www.coin-or.org/Couenne/