Performance and Power Optimization of Multi-kernel Applications on Multi-FPGA Platforms

Junnan Shan

March 2021

Applications using Convolutional Neural Networks (CNNs) and other Deep Neural Networks (DNNs) for Machine Vision and Natural Language Processing tasks achieved breakthrough results in many challenging domains. To continuously improve these results and approach human abilities in a broad variety of domains, the complexity of the network (i.e., its depth) increases. Most of these applications are run on data-center-class servers, for which processing speed and energy consumption are primary concerns. For those reasons, CPU- and GPU-based platforms are poorly suited and increase operating costs. ASICs can provide the best energy efficiency, but the continuous evolution of CNNs requires flexible ASICs, such as the Google TPU, which are, however, less efficient than theory would predict, for example because they support only a few numerical data types.

FPGAs are a promising option for CNN and DNN acceleration in datacenters, offering energy efficiency coupled with full re-programmability and configurability for both data path and memory architecture. This allows one to tailor the architecture to the application to a much deeper extent than is possible with either CPU/GPU platforms or relatively rigid domain-specific ASICs, like the Google TPU. For these reasons, cloud providers like Amazon Web Service (AWS), Alibaba, and Microsoft offer Virtual Machines coupled with multi-FPGA platforms to accelerate data-center applications with GPU-like performance, but consuming much less energy.

Since network depth and complexity increase, mapping a network on a single FPGA in most of the cases fails to meet performance requirements and would benefit from a multi-FPGA implementation. The problem that we are addressing is as follows. We are given an application modeled as an interconnection of tasks, each with various implementation options with varying performance, memory bandwidth, energy and resource requirements. We would like to statically or dynamically allocate resources to these tasks to optimize various measures of performance, such as throughput, energy per operation, and so on. Platforms like the CPU and the GPU use various kinds of schedulers (Operating System scheduler on the SW side, thread and instruction schedulers on the HW side) for this purpose at compile time or at runtime. The goal of this thesis is to design a compilation-like resource allocator for multi-FPGA acceleration. We devised and implemented an efficient and accurate optimization framework for the allocation of task-level pipelined applications (like Convolutional Neural Networks and Deep Neural Networks) to multiple FPGAs, with the twofold goal of maximizing the application throughput and minimizing the power consumption, under resource and off-chip memory bandwidth constraints. The target Multi-FPGA platform consists of AWS F1 instances with up to eight Virtex Ultrascale+ FPGAs.

First, we implemented in synthesizable C++ and optimized using HLS directives the computing kernel for each and every layer of large CNNs, such as AlexNet, VGG, YOLO, ResNet, and large DNNs, such as Transformer variants. Then, using SDAccel, we implemented individual kernels in hardware using one Compute Unit (CU) for each layer, and orchestrated their execution on the FPGAs by a host code written in OpenCL and executed by the CPU of the AWS board. This allowed us to profile each kernel and get resource and memory bandwidth usage, working frequency, and execution time, which later become the input data of the optimization problem. We provide a model that covers the whole application execution, and includes: 1) input data transfer time from the host CPU to FPGA DDR memory (dynamic RAM), 2) data transfer time from FPGA DDR memory to the FPGA on-chip memory (static RAM), 3) the actual kernel computation, 4) data transfer time from FPGA on-chip memory to FPGA DDR memory, and 5) data transfer from the FPGA DDR memory to the host CPU. This model can be used to mathematically formulate a complex Mixed-Integer Non-Linear Programming (MINLP) optimization problem, which can be solved using a commercial MINLP solver. However, using a MINLP solver is very slow, since the problem is NP-complete. To accelerate the optimization process, we provide a fast heuristic method using a Geometric Programming (GP) solver and an allocator. Not only it can return the solution in a matter of seconds, instead of running several hours or days when using the MINLP solver, but it also offers better results than those returned by the solver when its run time is limited for practical reasons.

Second, we developed another optimization framework to find the solution with minimum power consumption for a given throughput. This model is aimed at data center applications, where energy and cooling costs are significant. To optimize the power consumption we provide a power model on top of the performance model. This model includes the power consumption in different phases: 1) data transfer between host CPU and FPGA memory, 2) data transfer between FPGA and DDR, 3) FPGA computation. Given a throughput constraint, the model will return the best number of parallel number of powered-on FPGAs and their clock frequency and generates the most power-efficient bitstreams to program the FPGAs. This model can also lead to the formulation of another Mixed-Integer Non-Linear optimization problem, which can also be solved using a MINLP solver. We compared the solution obtained by the solver with one that simply clock gates the fastest implementation and one that uses frequency scaling: our method always uses less power. However, a MINLP solver can be very slow especially for design space explorations which need to run the solver several times. Therefore, we provide two different heuristic methods. One of them still uses the MINLP solver but in a reduced exploration space; the other one uses a greedy allocation. Both heuristic methods can be a few orders of magnitude faster than the MINLP solver.

Also for power optimization, we use AlexNet, VGG and Transformer networks to verify our model. The experimental results show that our approach can find the best solution compared to both 1) applying frequency scaling to optimize power under a throughput constraint starting from a fast configuration, and 2) replicating a slow configuration on multiple FPGAs.