TABLA: A Unified Template-based Framework for Accelerating Statistical Machine Learning

TABLA is a framework that can generate FPGA accelerators for many statistical ML algorithms. It leverages the commonalities across these ML algorithms: they can be expressed as stochastic optimization problem, which can be uniformly solved by SGD. It means that developers just need to specify the gradient of object function.

TABLA provides programmers flexible programming interface to specify gradient function. This interface facilitates the following execution of model compiler. One highlight is that the construct makes parallelism part of the code clear for hardware.

Model compiler is responsible for generating an execution schedule for the accelerator. Firstly the template indicates how to integrate stochastic gradient descent. Secondly, complier converts program to dataflow graph. Thirdly, compiler generate schedule of each operation by ML-RCS Algorithm.

Given the DFG and schedule and FPGA specification, TABLA’s design builder will generate synthesizable Verilog code. TABLA holds the prototype template of accelerator, which is scalable and modular. The template can be customized based on DFG and resources of target FPGA. The builder first determine the total number of Processing Engines. Simultaneously the builder generates the control units and buses. Then based on the schedule, control logic is generated. Finally, the builder adds the memory interface unit and the access schedule to the memory.

The PU design and busing logic is simplified because the scheduling is static. Processing Units comprise a set of identical processing engines. PE is good at scalability and customizability, which makes template design more flexible. A single PE is functional enough to carry out the computation of entire algorithm. PUs are connected through a global bus, and they are connected to memory interface through a data buffer. PUs don’t initiate requests , the data buffer fetch data from the external memory and send the data to the PUs. The ALU in PUs support all kinds of operation, but they execute different operation according to DFG. Control unit is the core part of a PU, which stores the PE’s schedule of operation.

In my opinion, the greatest contribution of this work is devising a suitable level of abstraction to indicate algorithms. Using gradient function to represent a wide range of algorithms is really insightful, since gradient descent is friendly to both ML experts and hardware. It is relatively fixed so that it can be recognized easily by model compiler and design builder. What’s more, it facilitates the portability of this framework across different FPGA platform.

From High-Level Deep Neural Models to FPGAs

DNNWEAVER aims at tackling tasks similar to TABLA. The difference is that deep learning more memory and bandwidth compared to machine learning.

The first step is to convert a DNN to a macro dataflow graph. TABLA is equipped with novel ISA, and each instruction represents a node in the graph. The architecture is based on dataflow, which efficiently reduce numbers of memory access.

The core of Design Planner is slicing computation, which reaches the intermediate point of the conflict between parallel operations and data reusing. Template Resource Optimization search algorithm solves the two variables: (1)PEs-per-PU and (2) slice dimensions. The heuristic function of this algorithm is estimated execution cycles. Gradually varying pe in the iterative process, and find the pe that minimizes eec.

The function Design Weaver is to map resource allocation and schedule to the accelerator core. Compared to TABLA, the templates of DNNWEAVER pay more attention to data reuse. For instance, neighboring PEs have a unidirectional link, and each PU has a dedicated buffer for weights sharing. In addition, partial results are stored in a local FIFO.

One great challenge to tackle in this paper is the contradiction between DNN's high memory footprint and FPGA's limited memory and bandwidth. Your team used various techniques to overcome this difficulty. This framework takes into account the rapid evolution of DNN, and great effort are made to improve its scalability and reconfigurability. Static scheduling is also sophisticated, which avoids contention on the bus and complex handshaking.