A Soft Coarse-Grained Reconfigurable Array Based High-level Synthesis Methodology: Promoting Design Productivity and Exploring Extreme FPGA Frequency

Cheng Liu, Colin Yu Lin, Hayden Kwok-Hay So
liucheng@eee.hku.hk, linyu@eee.hku.hk, hso@eee.hku.hk

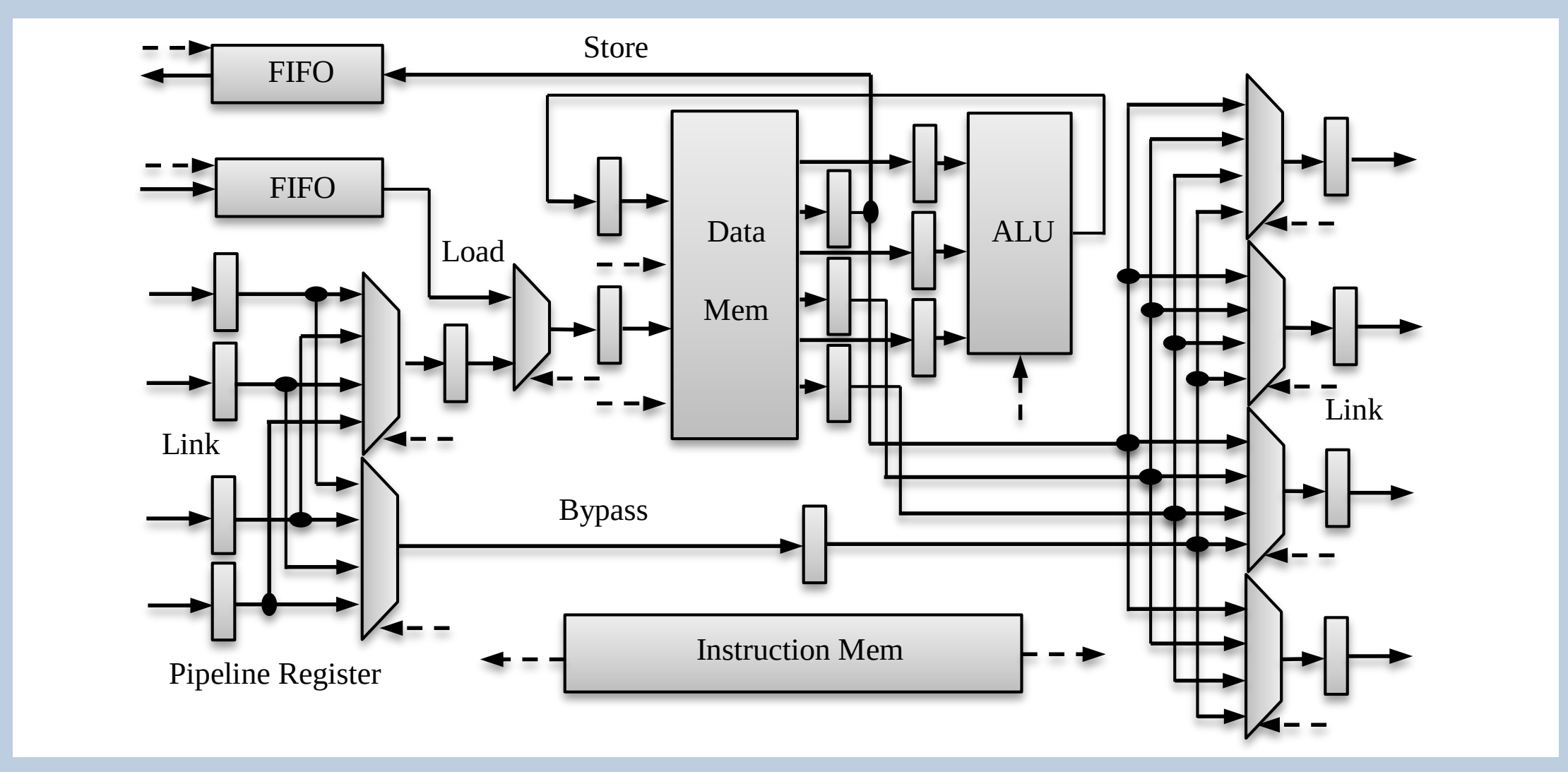
Background and Motivation

Compared to the use of a typical software development flow, the productivity of developing FPGA-based compute applications remains much lower[1]. There are two reasons for this.

- Current high-level synthesis (HLS) tools seldom help to reduce the lengthy low-level FPGA implementation process.
- High-level application developers often lack the intimate hardware engineering experience to achieve high performance on FPGAs.

SCGRA

The implementation of the SCGRA has a significant impact on the overall performance. An instance of such SCGRA design is thus presented to demonstrate the feasibility of producing high performance gateware without incurring long compilation time. One PE of the SCGRA is shown in the Figure below.



Conclusion

- By using SCGRA as an intermediate compile step, the lengthy low-level implementation tool flow is reduced to a relatively rapid operation scheduling problem, which contributes directly to higher application designers' productivity.
- Implementation with close to maximum clock frequency resulting from the highly regular structure of the SCGRA, in combination with an in-house scheduler that can effectively schedule operation to overlap with pipeline latencies guarantee the overall high performance.

References

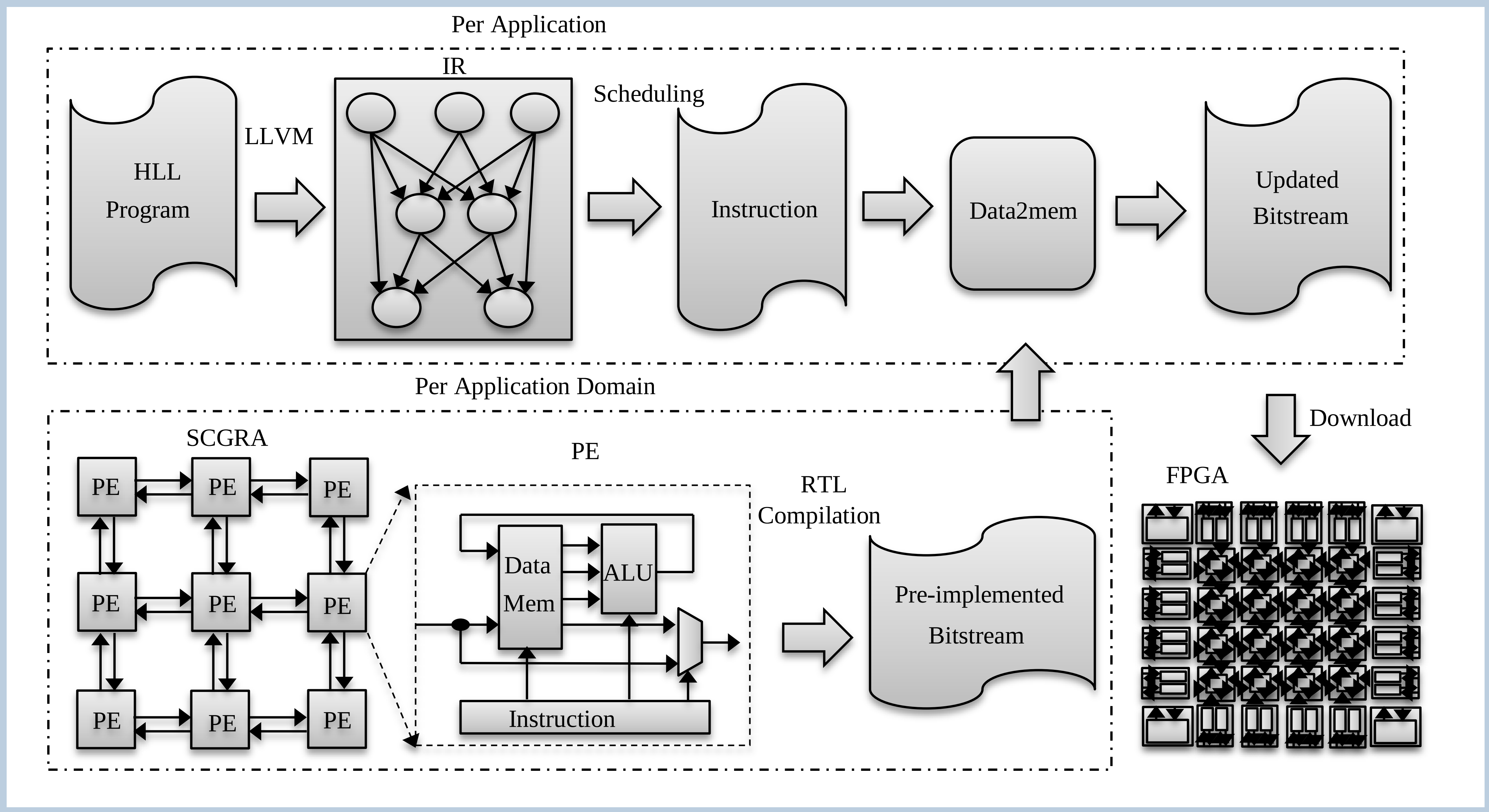
[1] Lavin, C. and Padilla, M. and Ghosh, S. and Nelson, B. and Hutchings, B. and Wirthlin, M., Using hard macros to reduce FPGA compilation time, International conference on Field Programmable Logic and Applications, 2010

[2] The LLVM compiler framework, http://llvm.org

[3] Cong, J. and Liu, B. and Neuendorffer, S. and Noguera, J. and Vissers, K. and Zhang, Z., High-level synthesis for FPGAs: From prototyping to deployment, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2011

Proposed Design Methodology

To Address the productivity and performance problems, a HLS methodology that utilizes soft coarse-grained reconfigurable arrays (SCGRAs) as an intermediate compilation step as shown in the figure below is presented. Instead of compiling high-level applications directly to circuits, the compilation process is reduced to an operation scheduling task targeting the SCGRA.



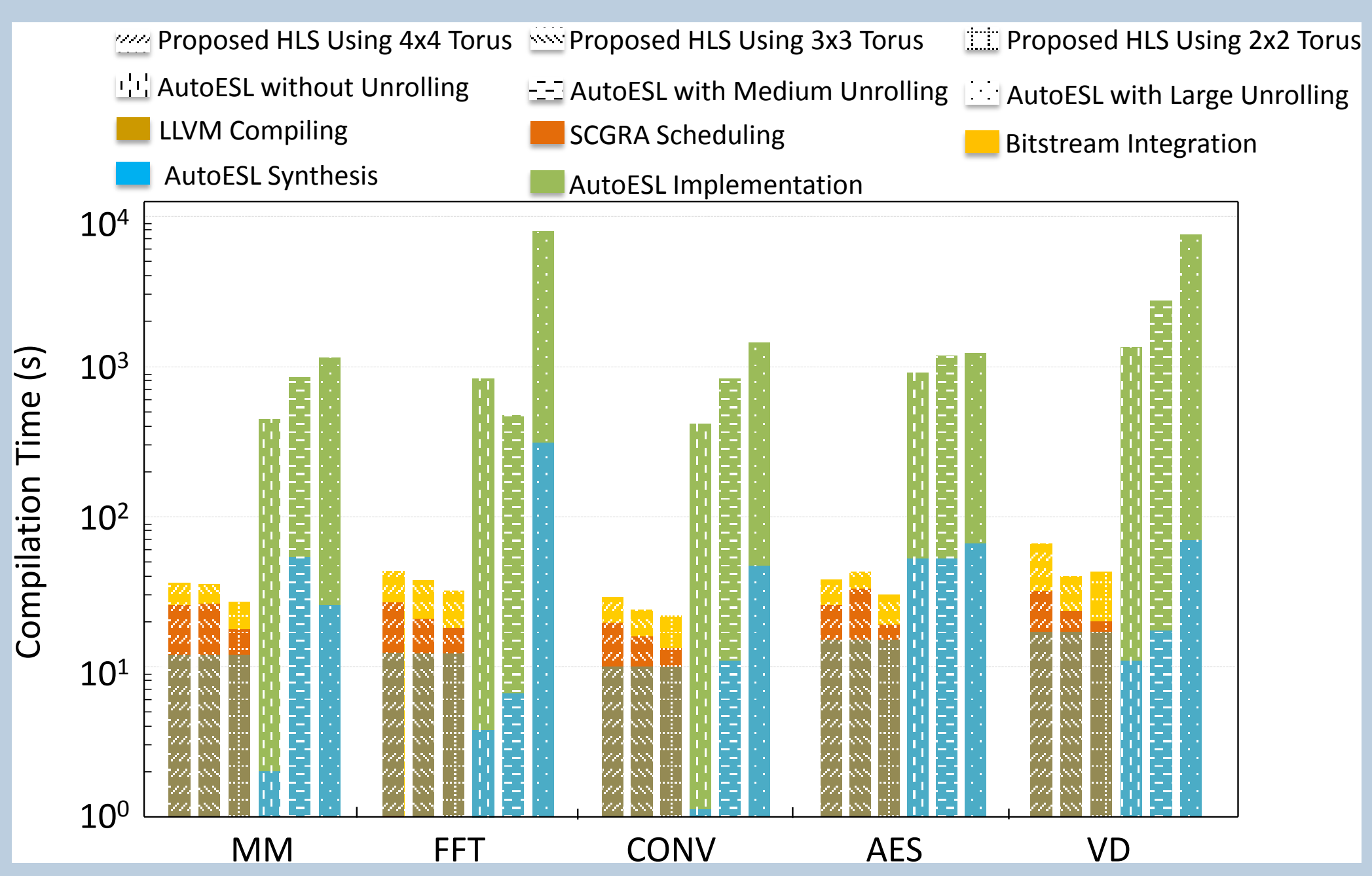
Experiment setup

We take 5 computation kernels including matrix multiply (MM), fast Fourier transform (FFT), discrete convolution (CONV), advanced encryption standard (AES) and Viterbi decoder (VD) as our benchmark.

All runtimes were obtained on a Linux workstation with an Intel(R) Xeon(R) CPU E5345 and 8GB of RAM. All the implementations targeted Xilinx Virtex6 FPGA (xc6vlx240t784- 1). The proposed HLS methodology employed LLVM v3.1 and clang v3.1 [2] for C program compiling and PlanAhead v13.4 for the SCGRA implementations. As for the direct mapping methodology, AutoESL 2011.4.2 [3] was taken as a representative.

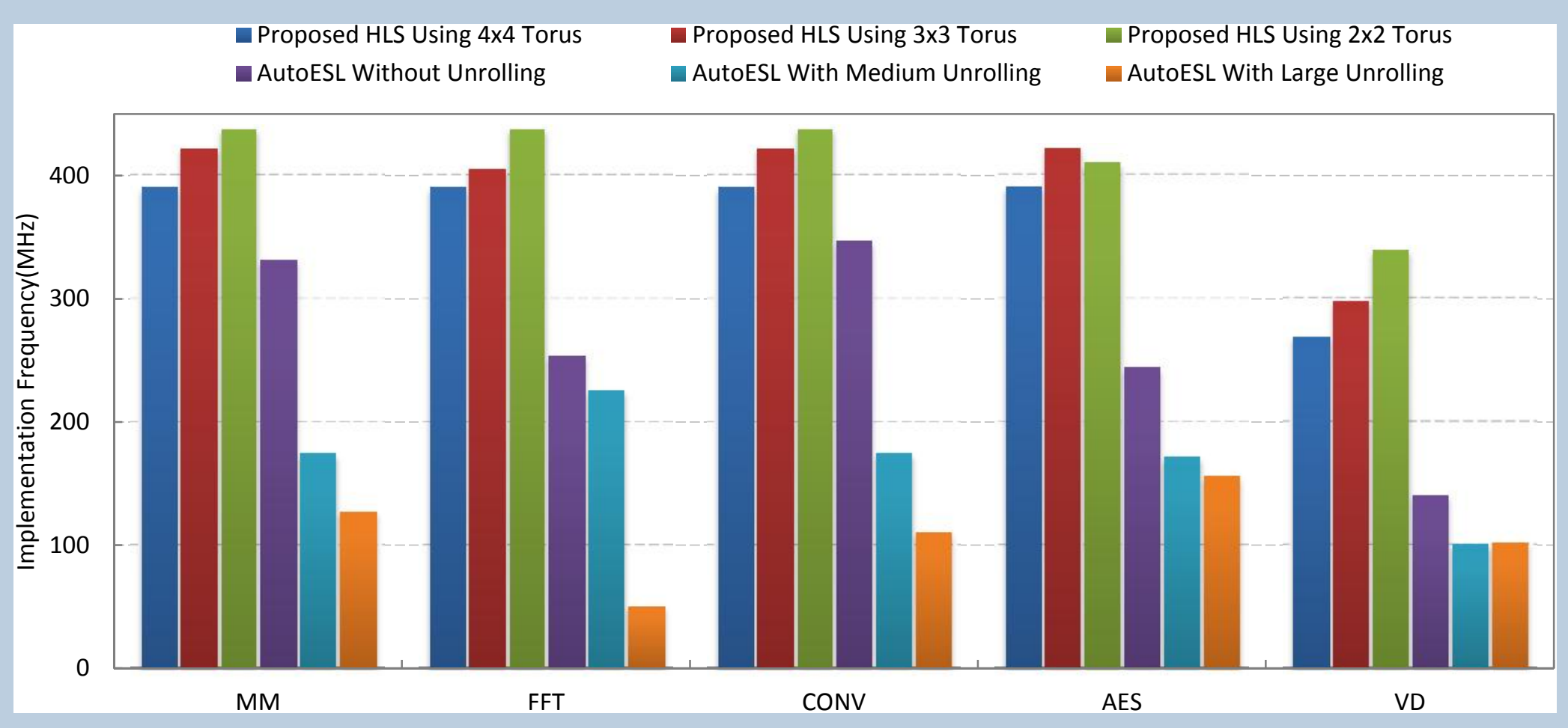
Experiment-1

AutoESL includes two steps: AutoESL synthesis and AutoESL implementation. The proposed HLS methodology bypasses the lengthy low-level implementation steps and simply needs three high-level steps: LLVM compiling, SCGRA scheduling and bitstream integration. As shown in the figure below, it is generally 10X-100X faster.

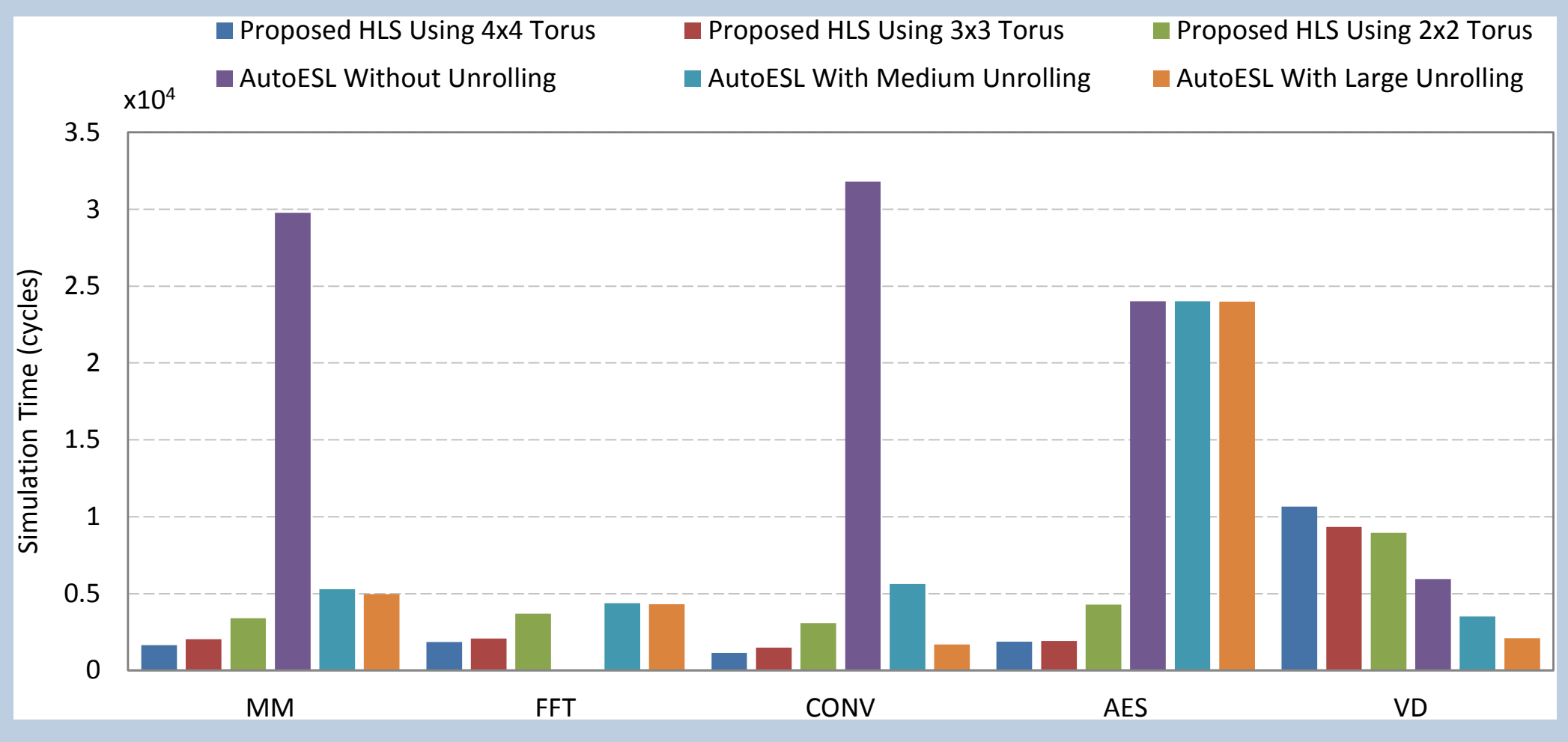


Experiment-2

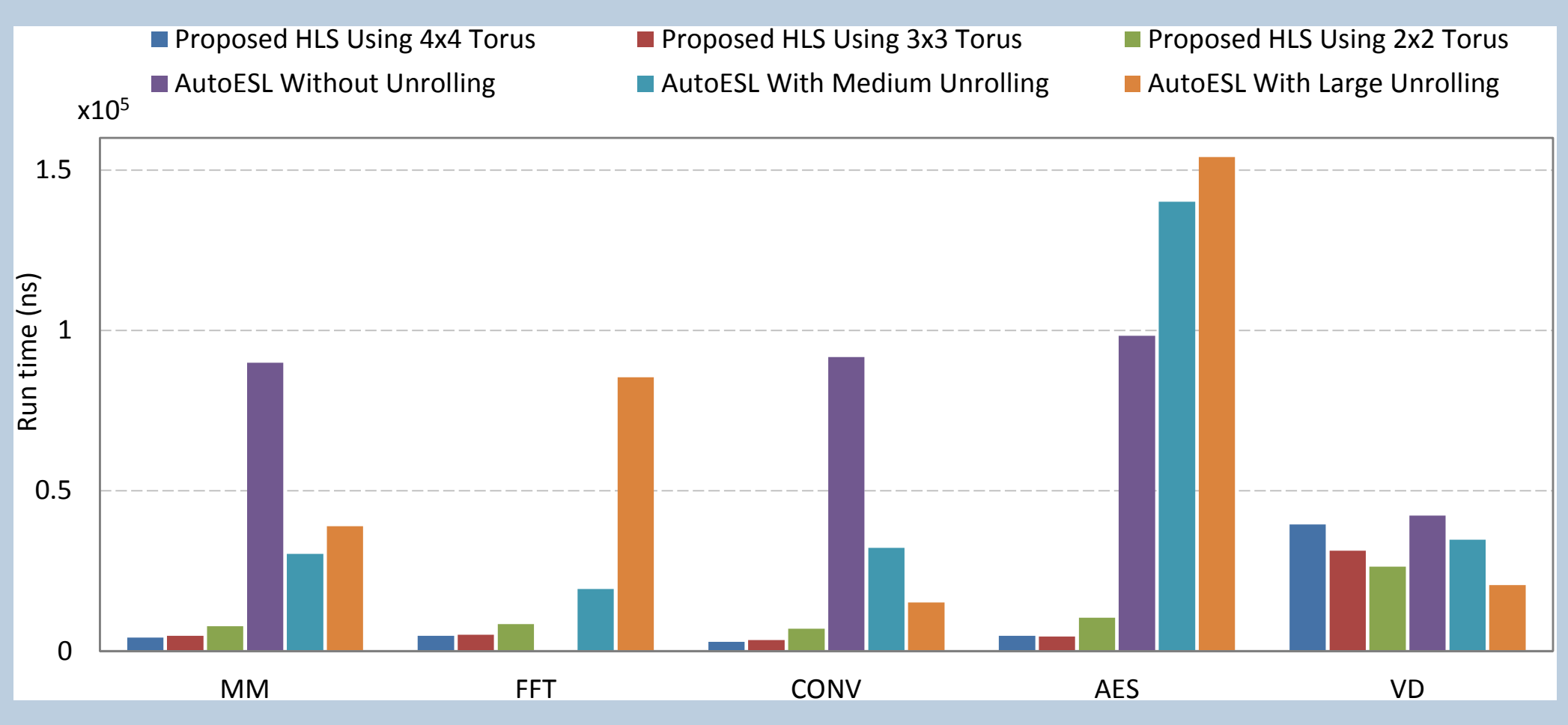
The figure below displays the implementation frequency. It is clear that the implementations using the proposed HLS methodology for all the kernels are much higher than those using AutoESL.



The following figure shows the simulation performance. The proposed design methodology outperforms AutoESL for most of the kernels.



With both implementation frequency and simulation performance given in previous two figures, the proposed design methodology achieved 0.8-21x times speedup in the application run time.


Proposed HLS Using 4x4 Torus
Proposed HLS Using 3x3 Torus
Proposed HLS Using 2x2 Torus
AutoESL Without Unrolling
AutoESL With Medium Unrolling
AutoESL With Large Unrolling
Run time (ns)
MM
FFT
CONV
AES
VD