Towards Automatically Compiling Efficient FPGA Hardware

Jean Baptiste Note, Jean Vuillemin
Ecole Normale Supérieure- Paris

We detail some aspects of our current research on compiling efficient FPGA designs from the source code of data flow applications. The output from our compiler is a FPGA hardware design for the Pamette [1] re-configurable co-processor. Three requirements are met by construction:

1. the source code software specifies the transform applied to the host memory content at each system cycle;
2. the compiled FPGA design bit-wise computes the very same cycle transform- yet at much higher speed thanks to [1];
3. the hardware design area is automatically minimized to meet a throughput requirement, which is specified a-priori within the source code.

Accordingly, the compiler carries its analysis/synthesis in three stages:

1. Unfold to SSA list, perform range analysis, bit-size and translate to RTL design:
    - The input source code is presented here in a C like syntax. In our experimental implementation, the Jazz language [2] is used- such source could be expressed just as well in other synchronous language: Lustre [3], Esterel [4], ...
    - Transform the source code to an equivalent SSA list operating on integers.
    - Analyse the range of each integer variable, and code each by a finite vector of bits; accordingly represent integer operations by Boolean functions.
    - The result is a RTL design equivalent to the bit-sized SSA description.

2. Map RTL to SPF form, re-time and FPGA compile:
    - Map RTL design to Serial/Parallel/Feedback SPF form.
    - Minimize re-timing registers to achieve optimal latency- based on reliable delay models for FPGA integer operators, and less reliable routing models.
    - Produce FPGA design from vendors tools (PamDC [5] and Xilinx [6]) and system software support [5].

3. FPGA design size/power meets set bandwidth requirement:
    - For RTL throughput below requirement, trade area for bandwidth according to [7].
    - For RTL throughput way above requirement, we first generate the Bit-Serial Design Realization: BSDR minimizes logic and throughput among designs.
        i. For BSDR throughput below requirement, unfold space as above.
        ii. For BSDR throughput much above requirement, we fold space at the expense of bandwidth through hyper-serial designs [7].
        iii. For low required throughput, a pure software implementation on the host processor is chosen, without using the co-processor.
        iv. Between these extremes, valuable hardware/software co-design tradeoffs are automatically met.

An experimental validation of the proposed techniques has been obtained.

- Including over half a dozen real-life designs in ***current multi-media***.
- This specific presentation highlights ***dithering in present digital printers***.
- Excluding so far ***automatic bandwidth adaptation***.

We automatically compile *efficient* FPGA designs for applications in the above list:

- *Efficient* means that, the compiled design compared to any hand-crafted for the specific case is no worse, by a factor of two in size/bandwidth/power.
- To permit efficiency, we let each stage automatically performed by the compiler be guided by annotations/pragmas- manually put in the source code.
- A fair measure of our system is thus the number of annotations added to the source code in order to compile an efficient hardware design, as defined above. Our experimental evidence, from all test cases, is: ***very few***!

This talk presents and explains part of the ***theoretical*** and ***experimental*** evidence, in the context of two half-toning algorithms: ***random*** and ***Floyd Steinberg*** digital dithering.

The conclusion from this study is quite **optimistic**: once a *small number* of specific *compiler directives* are added by the *learned designer*, the source code for many current multi-medias applications can be **automatically compiled** into **efficient FPGA** based hardware co-processors. We expect the compiler methodology to extend to other hardware/software technology targets as well- including SIMD machines and multi-core processors.