PARISCV: A Profiler for Application-Specific Acceleration on RISC-V

M. R. Ashuthosh, Joyen Benitto, Spoorthi S, and Madhura Purnaprajna

Centre for Heterogeneous and Intelligent Processing Systems, PES University, Bengaluru, India
Email: (ashuthoshmr@, pes2ug20ec028@pesu.,pes2ug21ec905@pesu.,madhurap@)pes.edu

I. Introduction

To meet the demand for accelerating applications efficiently, custom architectures have become a necessity. With the advent of the open-source RISC-V ISA, architects can now build better architectures by extending the ISA through custom instructions and extensions. While custom instructions can significantly improve processor performance for specific applications, identifying the right custom instruction along with the right set of extensions can be challenging and time-consuming. When multiple candidates exist, it becomes crucial to determine which custom instruction can provide the highest speedup in terms of clock cycles. In response, we have developed a profiler to aid in the identification of extensions and integration of custom instructions within RISC-V processors.

In this work, we present PARISCV [1], a profiler for finding and incorporating application-specific extensions, custom instructions in a RISC-V processor. Leveraging the VexRiscv [2] processor, which facilitates the quick and easy inclusion of custom instructions, we showcase the process of using the profiler. The profiler takes the trace output from Verilator [3] simulation of the processor running the application and generates a list of potential custom instructions. Furthermore, the profiler provides insights into the cycles that can be saved by extending the core with custom instructions. By employing this open-source instruction profiler, the RISC-V community gains a valuable tool for hardware-software co-design, enabling them to maximize the performance benefits by tailoring processors for specific application domains.

II. Methodology

For application-specific acceleration using PARISCV [1], we have devised the flow as shown in Figure 1. Our methodology of application specific acceleration is targeted to VexRiscv [2] platform. VexRiscv [2] is an FPGA friendly 32 bit RISC-V CPU implementation that is easily customisable to include RV32I[M][A][F[D]][C] extensions. A support for extending the core with custom instructions is available.

PARISCV [1] methodology can be divided into two i.e finding the appropriate extension and tailoring custom instruction.

Finding appropriate extension: This part involves in compiling the target application enabling all extensions as provided by RISC-V tool-chain [4]. The executable generated is simulated through an instruction accurate simulator called Spike [4], which generates the trace of instructions executed. Based on this trace, PARISCV suggests extensions that have significant presence in the target application.

Custom instruction: Next, we generate VexRiscv core supporting the suggested extensions. The application is recompiled based enabling only the selected extensions. The compiled executable is run on the cycle-accurate VexRiscv using Verilator [3]. The trace generated through cycle-accurate simulation is then fed into PARISCV for identifying the potential custom instruction that might accelerate the application. The output of PARISCV comprises a set of instructions poised for conversion into custom instructions, tailored to enhance application performance. Additionally, the package generates files containing statistics such as the theoretical cycles saved through the conversion to cycle-accurate information, alongside other miscellaneous data like the percentage constituency of instructions within the application. To ascertain the actual speedup, the provided custom instructions are integrated into the VexRiscv platform through hand written HDL code.The application is then modified to include custom instruction through inline assembly. The code involving custom instructions is recompiled and run on the VexRiscv equipped with required extensions, custom instructions. This way, one can accelerate application by hardware-software co-design. Through this process, performance impact and area utilization of the custom instructions can be gathered providing further insights in the process of acceleration. This methodology represents a systematic approach to analyzing and optimizing application performance by harnessing instruction and cycle-accurate information for tailoring the RISC-V core.

III. Experimental setup

Our experiments are carried out using six widely used applications from the Embench benchmark suite [5] and attempting to accelerate them on the VexRiscv [2] platform. RISC-V toolchain [4] is used for compilation and instruction accurate simulation. Verilator [3] is used for cycle accurate simulation and generating trace file (.fst file) of VexRiscv. Runtime of the application is adjusted by varying the iteration count for quick cycle accurate simulations. We modify the GDBwave [6] that is used for post-simulation waveform-based RISC-V GDB server to extract the necessary signals from the trace generated by Verilator. PARISCV [1] is a software written using python and the entire flow as described in Figure 1 is automated using bash script. Currently the search space of custom instruction in PARISCV is set to instruction window size of 2 and 3. We

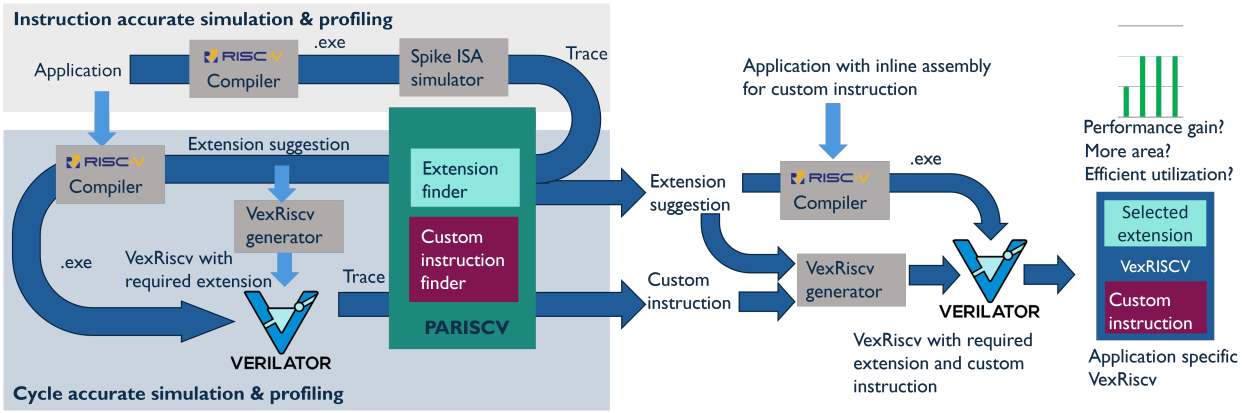

Fig. 1: PARISCV [1] flow.

use Xilinx Vivado 2022.1 [7] for synthesizing VexRiscv cores targeting XC7A100TCSG324 FPGA device.

IV. Results

Figure 2 shows the speedup obtained using appropriate extension and custom instructions as compared to baseline i.e VexRiscv core with RV32I implementation. We observe all applications except huffbench performing better than the baseline by including M (Multiplication/Division) extension as suggested by PARISCV. For huffbench, PARISCV suggest other extensions are not necessary, that otherwise would have incurred extra area (2.38× than baseline if we include M and F extensions). A maximum speedup of 22.3× is seen for edn. crc32 and picojpeg benefited from custom instruction further by 34% and 16% as compared to speedup after extension selection. PARISCV also suggests MAC (multiply-add) instruction as a custom instruction for further speedup of matmult-int and edn but due to limited number of source operands, we weren't able to realise it. For the rest of the applications, we weren't able to find a feasible custom instruction that can provide desired acceleration. We also tabulate the extra area due to extension + custom instruction (E+C) and speedup achieved for added area (in terms of LUTs) ratio in Table I. From the table, we observe with minimal increase in the area we are able to significantly accelerate the applications.

Application	Extension (E)	Area (E+C)			Speedup / Area
		LUT	FF	DSP	
crc32	I, M	1.20×	1.14×	7×	1.79
aha-mont64	I, M	1.19×	1.14×	4×	2.08
edn	I, M	1.19×	1.14×	4×	18.74
huffbench	I	1×	1×	1×	1
matmult-int	I, M	1.19×	1.14×	4×	7.33
picojpeg	I, M	1.27×	1.14×	4×	1.03

TABLE I: Speedup and area as compared to RV32I core for different applications of Embench [5].

V. Conclusion and Future Work

In conclusion, the PARISCV methodology presents a systematic approach to analyze and optimize application performance within the RISC-V architecture. Through this work, we were able to efficiently accelerate six Embench applications by enabling the necessary extensions and custom instructions. Although for our target applications, we weren't able to find custom instruction that provide significant acceleration as compared to extension selection, the methodology serves as a valuable resource for application-specific RISC-V based acceleration. Future work involves, extended PARISCV to suggest subset of instructions from an extension. We also intend to explore accelerating through coarse grained accelerators by varying the window size of search space in PARISCV.

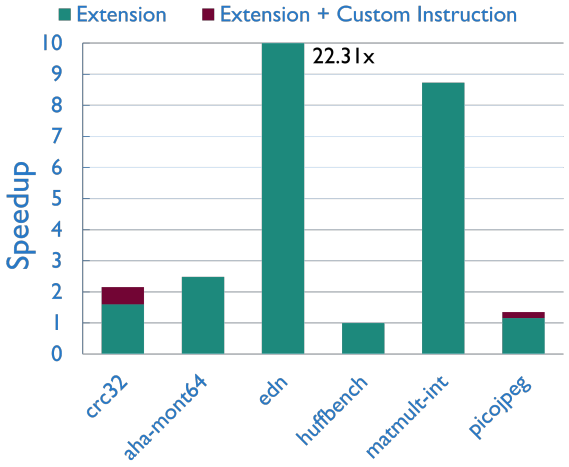

Fig. 2: Speedup obtained for using extensions and custom instructions as suggested by PARISCV.

References

[1] "Pariscv." https://github.com/CHIPS-PESU/pariscv/.
[2] "Vexriscv." https://github.com/SpinalHDL/VexRiscv/.
[3] "Verilator." https://www.veripool.org/verilator/.
[4] "Risc-v toolchain." https://github.com/riscv-collab/riscv-gnu-toolchain/.
[5] "Embench." https://github.com/embench/embench-iot/.
[6] "Gdbwave." https://github.com/tomverbeure/gdbwave.
[7] "Vivado." https://www.xilinx.com/products/design-tools/vivado.html.