RAPID DESIGN EXPLORATION FRAMEWORK FOR APPLICATION-AWARE CUSTOMIZATION OF SOFT CORE PROCESSORS

Alok Prakash, Siew-Kei Lam, Amit Kumar Singh, Thambipillai Srikanthan (Senior Member IEEE)

CHiPES, School of Computer Engineering, NTU
email: {alok0001,assklam,amit0011,astsrikan}@ntu.edu.sg

ABSTRACT

Off-the-shelf soft core processors are becoming increasingly popular in embedded systems design today as they provide for application specific customization, in particular through instruction subsetting. However, choosing the right processor configuration remains a challenge as the search space becomes prohibitively large when the configurable options increase. In this paper we propose a framework to rapidly explore the processor configuration design space for a given application. Unlike existing approaches that require time-consuming synthesis process, the proposed method relies only on a single-pass output of the LLVM compiler infrastructure. Experimental results based on widely used benchmarks show that the proposed framework can reliably predict the actual performance and area trends of various configurable options.

1. INTRODUCTION

In recent years, parameterized soft core processors have become a popular Intellectual Property in FPGAs as they provide for application-specific customization. Instruction subsetting is one of the most common and effective means for customizing soft-core processors. Dougherty et al. [1] have demonstrated the advantages of instruction subsetting as a means for reducing power consumption by manually optimizing the assembly codes of the applications. Commercial FPGA soft-core processors [2],[3] facilitates instruction subsetting by providing a number of configurable functional units e.g. hardware multiplier, barrel shifter, etc. that can be enabled/disabled during the processor specification phase. These configurable units incur additional FPGA area, and hence they should be used prudently.

Sheldon et al. [4], Yiannacouras et al. [5] and Padmanabhan et al. [6] showed with Microblaze, NIOS and LEON processors respectively that application specific customization of these soft core processors can lead to significant area savings with better performance, at times even compared to the fully blown CPU. However, as the number of configurable parameters increase with modern soft core processors, these methodologies which necessitate time-consuming synthesis/execution runs for each configuration option become impractical. For example, an exhaustive search for a suitable Microblaze configuration can take up to 11 hours [4]. The LEON processor has approximately 90 parameters [6] to choose from and about $5 * 10^{24}$ possible configurations.

Hence, there is a need to develop faster design exploration methodologies that can assist designers in choosing the best processor configuration for a given application. Our framework does not require time consuming synthesis and execution runs for each configuration option. Instead, it relies on a single-pass compilation output of the open source LLVM compiler [7] infrastructure to predict the performance and area trends when different functional units are selected for the soft core processor. The tool exploration time is within seconds compared to tens of minutes of synthesis runtime in previous methods making it highly scalable for a large number of configuration parameters. Moreover, the proposed framework is not restricted to a particular target processor, and hence it can be employed to identify the most appropriate processor for a given application. Lastly, the proposed strategies can also be incorporated in existing frameworks such as SPREE [5] to rapidly identify the best configurations for generating a customized processor.

2. PROPOSED FRAMEWORK

In this section we describe the proposed design exploration framework (see Fig. 1) for application specific customization of soft core processors. Similar to [4], we assume that all the configuration parameters affect the system performance independently. This implies that the resulting gain and cost incurred by instantiating two components is the sum of the gain and cost of the individual components. This assumption does not limit the effectiveness of our estimation tool because as shown in [4], the dependence of different parameters is negligible. As shown in Fig. 1, the framework has three major phases: 1) Pre-characterization Phase, 2) Architecture-independent Phase, and 3) Application-aware Design Exploration and Customization.

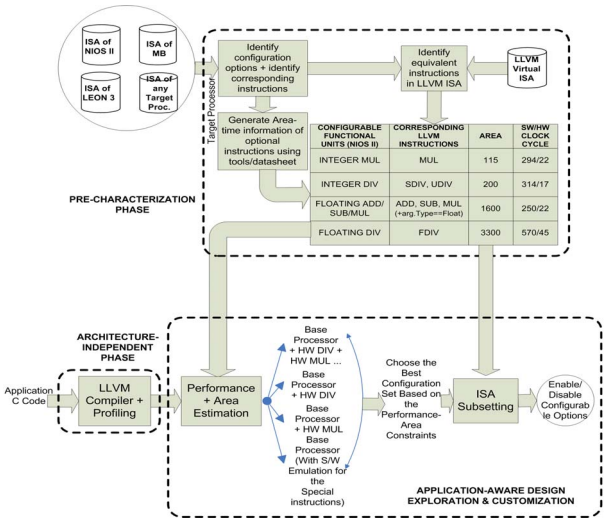


Fig. 1. Design Exploration Framework.

2.1. Pre-Characterization Phase

The pre-characterization phase is a one-time process that needs to be performed for each target processor. In this phase, the configurable functional units of the target processor and the corresponding instructions which can benefit from these units are identified. For example in the NIOS II processor, the hardware Integer Multiplier unit corresponds to the following instructions: *mul, muli, mulxss, mulxsu, mulxuu*. Similarly the instructions *div* and *divu* benefit from the hardware Integer Divider.

Next, the execution time of these optional instructions in software and hardware are obtained from either the datasheet or through test applications that are subjected to the toolchain. Software execution time is the number of clock cycles required to implement the optional instruction on the base processor when the corresponding functional unit is unavailable. Hardware execution time is the number of clock cycles required by the hardware functional unit to implement the corresponding optional instruction. In addition, we also determine the hardware area required to implement the configurable functional units.

Finally, the corresponding instructions in the LLVM virtual instruction set that implement the optional instructions are identified. Fig. 1 shows an example of the information pertaining to the optional instructions of a particular soft-core processor that are stored in a table. It can be observed that these informations include the configurable functional units, corresponding LLVM instructions, hardware area of the configurable functional units and their software/hardware execution time. Note that different configurable functional units is associated with different hardware/software execution time. For example, the NIOS II processor executes the multiplication operation approximately 10 times faster in hardware than in software, while the floating point operations are executed about 12 to 15 times faster in hardware.

2.2. Architecture-Independent Phase

In this phase, the given application is first compiled and analyzed using the open-source LLVM compiler without any reference to the target processor. An application C Code is compiled using LLVM to produce the LLVM IR (Intermediate Representation). It should be noted that the LLVM IR uses the native LLVM virtual instruction set. A "LLVM Pass" has been written to parse the IR in order to identify the function, basic block, instructions in these basic block, their arguments and their types. Next the application is profiled using the LLVM-profiler (which uses the LLVM Just in Time Compiler) to obtain the basic block execution counts. At the end of this phase, all the optional instructions in an application along with their execution count are obtained.

2.3. Application-aware Design Exploration and Customization

In this final phase, we rely on the information obtained in the pre-characterization step and the architecture-independent step to explore the design trade-offs of various processor configurations. In particular we calculate the hardware/software execution time (in clock cycles) of each of the optional instructions identified in the architecture-independent phase in order to predict the performance gain of various processor configurations for a given application.

It is noteworthy that we do not estimate the execution time of the instructions that cannot benefit from the configurable functional units as they do not lead to notable differences across the various configurations. We will show in the experimental results section that restricting the performance estimation to only the optional instructions can sufficiently enable the designers to choose the most suitable configuration for a given application. Restricting the estimation process to only the optional instructions also results in reduced complexity in the various phases of the our framework.

In addition to performance estimation, each configuration is annotated with the corresponding hardware area of the functional units used. Hence, for a given application, we can obtain a performance-area plot that describes the design trade-offs for different processor configurations. This performance-area plot assists the designer in identifying the most suitable processor configuration for a given application without the need to undergo time-consuming synthesis and execution runs. Having decided on the best configuration, the designer can then enable/disable the corresponding functional units during the processor specification phase in the target processor. This is the ISA subsetting step as shown in Fig. 1, which enables us to effectively remove the support for instructions that are never used or do not

lead to significant gains in the application. Alternatively, the performance-area plot can be incorporated in frameworks such as SPREE to rapidly explore the large design space in order to determine the best configuration before generating the RTL description of the customized processor.

Although the experimental results in this paper is based on the NIOS II processor, the proposed framework is generic enough to be applied to any target processor. This also allows us to use the proposed framework to select the most suitable processor and the corresponding configuration for a given application from a pool of processors.

3. EXPERIMENTS, RESULTS AND DISCUSSIONS

In this section, we present experimental results for our work. Our experiments are based on the NIOS II Processor from Altera [8] that is targeted on the STRATIX II device. The applications used in our experiments are taken mostly from two popular benchmark suite, MiBench and Mediabench. The applications from the MiBench suite (i.e. basicmath, fft and CRC32) are less compute intensive, and hence they do not need most of the configurable functional units. Therefore, instruction subsetting can lead to substantial area reduction for such applications. The applications from the Mediabench suite (i.e. MPEG-2 Encoder, MPEG-2 Decoder, G721 Encoder, EPIC Encoder and EPIC Decoder) are quite computationally intensive and hence they require most of the configurable functional units. Also, a handwritten application for the raytrace algorithm was tested with the tool.

For the NIOS II (fast core) processor, the configurable units considered in our work are Integer Multiplier (IM), Integer Divider (ID), single precision Floating Point Unit (FU) and Floating point divider unit (FD). We have not considered the cache dependency in this work. Instead, we instantiated the largest possible cache on the device. Selection of cache sizes will be considered in the future expansion of this work.

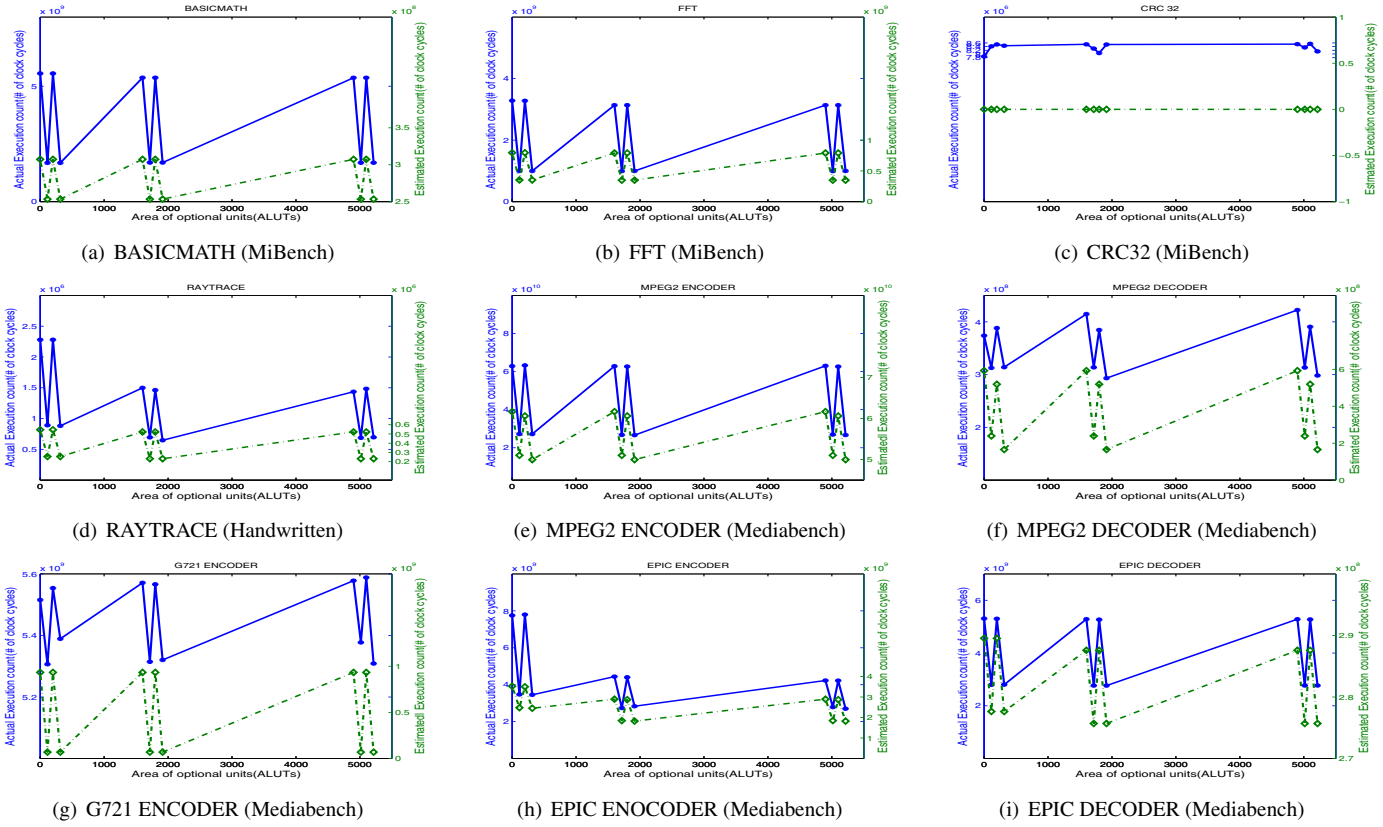
During our experiments we observed that most of the soft core processors have an optional IEEE 754 compliant Floating point unit, which when instantiated, computes the *single precision* floating point operations in hardware. The double precision operations on the other hand are always executed using the software libraries. The FU is not used for such operations. We have noted that in the NIOS II processor, the double precision instructions utilizes the hardware integer multipliers (i.e. IM). The execution time for the double precision instructions is approximately 3 to 5 times faster when a hardware integer multiplier unit is present. This demonstrates that the some of the configurable units can be used indirectly, and must be taken into account during the estimation process. The proposed framework can successfully capture such subtle anomalies during the precharacterization phase. In addition, we have included a procedure in the design exploration stage of the framework to estimate the number of double precision instructions.

The graphs in Fig. 2 show the performance-area plots of different processor configurations for the applications. The Y-Axis is the execution time (number of clock cycles) and the X-Axis is the approximate hardware area incurred for different configurations. The dotted line is the estimated execution time of the optional instructions from our framework. The second line shows the actual performance of the application on different configuration of the processor. In order to get the actual performance, we used the NIOS II performance counter macros in the application code. This macro is minimally intrusive and thus the overhead added is extremely low compared to the application runtime. We synthesised different possible configurations and executed the applications on them to get the execution count on each of the configuration. Since there are four configurable functional units in our case (i.e. IM, ID, FU and FD), we should perform the estimation for 16 configurations. However in the NIOS II architecture, we cannot instantiate FD without instantiating FU. This leaves us with 12 possible configurations. This does not limit the estimation framework as we can predict the performance of all the 16 configurations.

As expected, the numerical value of the points on the dotted line in the graphs is much lower than the actual implementation results, as the estimation is performed on the optional instructions only, whereas the actual implementation results consist of execution time of the entire application. It is noteworthy that when comparing the two lines, we are only interested in the performance-area trends (rather than the absolute values) as they provide sufficient information to enable the designer to choose the most suitable processor configuration for a given application. It is evident from the graphs in Fig. 2 that the proposed strategy for restricting the performance estimation to only the optional instructions is justified, as this can reliably predict the performance-area trends of various processor configurations.

In Fig. 2, we see that Basicmath and fft have a zigzag behavior. This is due to the fact that for these applications, the only significant amount of optional instructions is the integer multiplication. Hence whenever we instantiate the hardware integer multiplier unit, there is a significant dip in the execution count. Applications like CRC32, do not contain any optional instructions, hence the proposed framework outputs a flat performance-area curve. This is consistent with the actual implementation results, wherein we observe that the performance of the CRC32 application remains almost the same for different processor configurations. The minor fluctuations in the actual performance of the CRC2 application can be ignored as the execution count can never be the same even in consecutive runs of the application.

Overall, it can be observed that the proposed framework can reliably predict the performance-area trends of different processor configurations. We also see that our method per-



(a) BASICMATH (MiBench) (b) FFT (MiBench) (c) CRC32 (MiBench)

(d) RAYTRACE (Handwritten) (e) MPEG2 ENCODER (Mediabench) (f) MPEG2 DECODER (Mediabench)

(g) G721 ENCODER (Mediabench) (h) EPIC ENOCODER (Mediabench) (i) EPIC DECODER (Mediabench)

Fig. 2. Performance vs. Area Curves for Benchmark Applications.

forms best when predicting the advantages of different configurable functional units for the Mediabench applications like the EPIC Encoder (Fig. 2.(h)), which contain a good mixture of the optional instructions. These results demonstrate the advantages of the proposed framework as a means for rapid design exploration of processor configurations.

4. CONCLUSION

In this paper, we proposed an efficient design exploration framework for rapid customization of soft core processors through instruction subsetting. As the framework relies on the open-source compiler infrastructure for compilation and analysis, it is not restricted to a particular target platform. Only a one-time pre-characterization step is required for a new target platform and the proposed strategies in the pre-characterization step successfully capture subtle points like nested dependence of the functional units. In addition, the runtime of the framework is only in the order of seconds as the performance-area estimation of different processor configurations can be achieved without undergoing time-consuming synthesis and execution. The output of the proposed framework is a performance-area plot, which can assist the designer in deciding a suitable configuration given a performance area constraint.

5. REFERENCES

[1] W. Dougherty and et al., "Instruction subsetting: Trading power for programmability," in *VLSI '98. System Level Design. Proceedings. IEEE Computer Society Workshop on*, 1998.

[2] D. Caldern and et al., "Soft core processors and embedded processing: a survey and analysis," in *Proceedings of -ProRISC*, November 2005, pp. 483–488.

[3] J. Tong and et al., "Soft-core processors for embedded systems," in *Microelectronics, 2006. ICM '06. International Conference on*, Dec 2006, pp. 170–173.

[4] D. Sheldon and et al., "Application-specific customization of parameterized fpga soft-core processors," in *ICCAD '06: Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, 2006.

[5] P. Yiannacouras and et al., "Exploration and customization of fpga-based soft processors," *Computer Aided Design of Integrated Circuits and Systems IEEE Transactions on*.

[6] S. Padmanabhan and et al., "Automatic application-specific microarchitecture reconfiguration," in *In Proc. of 13th Reconfigurable Architectures Workshop*, 2006.

[7] "LLVM compiler infrastructure (http://llvm.org/)."

[8] "Altera NIOS 2 processors.(http://www.altera.com/products/ip/processors/nios2/ni2-index.html)."