Instruction-set Architecture Exploration Strategies for Deeply Clustered VLIW ASIPs

Roel Jordans, Rosilde Corvino, Lech Jóźwiak, Henk Corporaal
Department of Electrical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands
r.jordans@tue.nl www.asam-project.org

***Abstract*—Instruction-set architecture exploration for clustered VLIW processors is a very complex problem. Most of the existing exploration methods are hand-crafted and time consuming. This paper presents and compares several methods for automating this exploration. We propose and discuss a two-phase method which can quickly explore many different architectures and experimentally demonstrate that this method is capable of automatically achieving a 50% improvement on the energy-delay product cost of an automatically generated architecture for an ECG detection application and a 1% energy-delay product cost improvement compared to a hand-crafted design.**

I. Introduction

More and more modern products in telecommunications, multi-media, medical instrumentation, and several other important areas require programmability, high-performance, and/or limited energy consumption. Highly customized application specific instruction-set processors (ASIPs) and, more specifically, very-long instruction word (VLIW) architectures, are increasingly used in such products. Several industrial tool-flows, e.g. [1]–[3], are available to specify, simulate, and/or synthesize such processors. Some tool-flows exist for the automatic exploration of these architectures, but they commonly focus on VLIW processors with a centralized register file [3]. However, as was shown in [4], VLIW architectures with a centralized register file and more than 8 issue-slots quickly become impractical, as the resulting processors slow down due to their complex wiring and the many ports required on the centralized register file.

Clustered VLIW architectures offer a solution by partitioning the centralized register file into several smaller register files. This technique makes it possible to construct fast VLIW architectures with many issue-slots. However, until now clustered processor architecture exploration has been performed by hand. Terechko et al. [4] defined a taxonomy for different variations of clustered VLIW processor architectures. In a generic clustered VLIW architecture, one cluster contains one register file and or more issue-slots. The so called *deeply clustered VLIW* architectures are a sub-set of the generic clustered VLIW architectures in which a cluster contains only a single issue-slot.

In this paper we focus on these deeply clustered VLIW architectures and present and compare several methods to automatically explore: *1)* the number of issue-slots, *2)* the operations available within the issue-slot, *3)* the size of the register files. This paper focuses exploration methods which are using a *shrinking* technique. This allows us to take an oversized architecture, produced as part of a high-level architecture exploration, and turn it into an efficient processor design. The presented exploration strategies are evaluated regarding to both the efficiency of the resulting VLIW ASIP architecture and the time required for performing the exploration.

This paper is organized as follows. Section II explains the target architecture, the general exploration method, its exploration parameters, and introduces the considered exploration strategies. Section III discusses the experimental results and section IV concludes the paper.

This work was performed as a part of the European project ASAM that has been partially funded by ARTEMIS Joint Undertaking, grant no. 100265

II. Architecture exploration methods

Figure 1 shows the general structure of a clustered VLIW processor. The VLIW data-path is composed of a set issue-slots (IS), each containing one or more function-units (FU). The data-path is controlled by a sequencer which executes instructions from the program memory. The function-units in the issue-slots implement the operations and can require pipelining. Each issue-slot is capable of starting a new operation per cycle. The inputs of the operations are taken from the register file (RF) connected to the issue-slot. The output of the issue-slot can be written to one or more register files through the result select network (not shown in figure 1). Each register file can have one or more input ports to allow parallel writes to the register file. One or more local memories (not shown in figure 1) can be present in the processor. These local memories are accessed through a special load/store function-unit (LSU). Only one LSU can be connected to a single local memory.

Our ASIP architecture exploration method uses a shrinking technique and assumes that an *initial prototype* is provided.

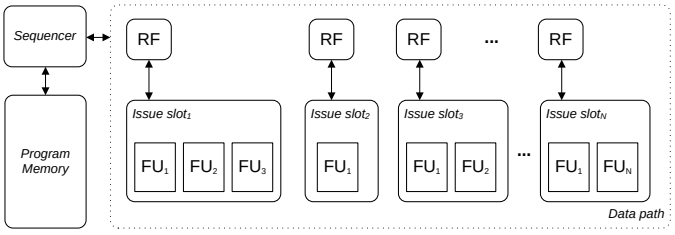

Fig. 1: Architecture template of a VLIW ASIP data-path and sequencer

This initial prototype consists of an oversized ASIP architecture accompanied by a coarsely optimized version of the target application. In our ASIP design flow, this initial prototype is the product of a coarse, high-level, ASIP architecture exploration [5] which aims at finding the best combination of several possible application loop-transformations (e.g. loop fusion, tiling, and vectorization) and corresponding ASIP architecture. However, other sources of initial prototypes (e.g. a human designer) can also be used.

The automatic ASIP instruction-set architecture exploration method explores the following properties:

- The number of issue-slots
- The types of function units available in each issue-slot
- The operations available in each function-unit
- The sizes of register files
- The size of the program memory

Our automatic ASIP instruction-set architecture exploration does not explore the number and sizes of local memories. In our total ASIP design flow, these have been explored as part of the application transformation (parallelization) and coarse ASIP architecture synthesis performed at a higher level, and are already decided by [5] when constructing the initial prototype. Previous research (e.g. [6], [7]) has shown that a large part (50–80%) of the architecture cost is due to the data storage and transfer. The memory and communication exploration is therefore commonly separated from and performed before the precise data-path exploration.

A. Architecture cost evaluation model

The energy and area cost of each proposed processor architecture is evaluated using our architecture cost model. This model uses the scheduled assembler code from the target compiler, which is configured to only use resources available in the proposed processor, and combines it with the application's profile to provide activity counts for the internal components of the proposed processor. These activity counts are then used to estimate the energy consumption of the proposed processor architecture.

We have redirected some of the architecture optimization work to the architecture model in order to keep the exploration time within reasonable bounds. In our architecture model we have decided to ignore any unused resources that are still remaining from the platform instantiation of the initial prototype. Any resource that is unused in a candidate prototype will not be counted towards the area and energy estimation of that candidate prototype. The architecture model recomputes the required size of the program memory, based on the set of the actually used resources and the length of the compiled target application.

Currently, our architecture model is capable of ignoring the following unused ASIP:

- *Complete function units* from the area and energy estimation, and the program memory size computation.
- *Partial function units* from the program memory size computation based on their unused operations.
- *Partial and complete register files* from the area and energy estimation, and program memory size computation based on the register file pressure of the compiled application.

The architecture model also takes changes in the interconnect of the ASIP resulting from the removal of components into account for both the area and energy estimation. This allows us to quickly see which resources are required and which are not, and what the effect is of the resource's removal on the resulting ASIP. It also gives us a clear view on which resources may form bottlenecks in the ASIP, and which are good candidates for further exploration.

B. Exploration style

Using our intelligent architecture model, we are capable of exploring the ASIP architecture using three different exploration styles. We can perform a coarse exploration removing complete *issue-slots* and rely on the architecture model to remove any remaining redundant resources. A more fine-grained exploration is possible by removing *function-units* directly and only removing issue-slots when they are empty. Or we can take a *two-phase* approach combining the above two strategies.

In this two-phase approach we first perform a coarse exploration, where we investigate which issue-slots can be removed from the initial prototype. This provides us with an initial set of promising candidate architectures. We then use the architecture model to determine which resources were actually used by each candidate prototype, and explore one or more of the candidate architectures in more detail by fine-tuning the available resources.

For example, when the first exploration stage finds a register file with a register pressure of 33, the second exploration stage will attempt to resize that register file to 32 elements. Resizing the register files to fit within a smaller power of 2 both results in a physically smaller register file and in a shorter encoding for the register file entry. This encoding is especially interesting as it has a significant impact on the size of the program memory. For each bit in the register file index encoding, the program memory requires a bit for each input- and each output-port of the register file. With 2 input, 3 output register files, a size commonly used in our ASIP template, this saves 5 bits in the instruction word.

C. Exploration strategies

Similarly to [8], we provide two exploration strategies, *best match* and *first match*, both based on the same cost metrics.

1) Best match: The best match exploration considers the removal of each separate component and selects the component that shows the best improvement of the cost metric. For example, when removing an issue-slot from a n issue-slot initial prototype during the first exploration phase, selecting the best match exploration strategy will construct a set of candidate prototypes with $n-1$ issue-slots containing each possible subset of $n-1$ issue-slots from the initial prototype.

2) First match: The first match exploration considers different alternative components sequentially and selects the first component that shows an improvement of the cost metric. The first match strategy results in a much smaller number of considered design points. This is not a problem when choices are symmetric, i.e. there is no difference between two choices (e.g. removing one of two equal issue-slots), but may result in suboptimal solutions when the design choices are asymmetric (e.g. register file sizing).

D. Cost metrics

As briefly mentioned above, both the best match and first match strategy depend on the ability to identify one candidate to be 'better' than another one. For this purpose, our ASIP exploration framework offers two different, commonly used, cost metrics: energy-delay (ED) product and energy-delay-squared (EDD) product. Where the energy-delay product puts more emphasis on the energy consumption, the energy-delay-squared product puts more emphasis on the delay. In both cases, a design is better when it has a lower score. Our ASIP exploration framework automatically computes, and displays, both scores normalized relatively to the initial prototype, making it very easy to select the final architecture from a set of constructed promising candidate architectures.

III. Experiments

To demonstrate the value and benefits of our new ASIP instruction-set architecture exploration method on a real-life application, we selected the Pan-Tompkins QRS detection algorithm [9] for an electro-cardiogram (ECG) monitor, performed the architecture exploration for this application, and mapped this application onto the selected architecture. For this application, both the power consumption and real-time behaviour of the created architecture are critical.

A. The algorithm

The goal of the Pan-Tompkins QRS detection algorithm is to detect the QRS peak (illustrated in figure 2) in an ECG signal. The distance between two consecutive QRS peaks, the RR interval, is commonly used to measure the hart rate of a patient.

The Pan-Tompkins QRS detection algorithm is implemented as a set of five filters and a detection function. Firstly, the input signal is filtered by a band-pass filter (constructed using a low-pass followed by a high-pass filter). Thereafter, the derivative of the signal is taken, this derivative is then squared and, finally, the last 30 squared results are integrated using a moving-window integrator. The so filtered signal is presented to a detection function which implements a set of thresholds which are dynamically adjusted based on the previously measured inputs. Please refer to [9] for more information on the algorithm.

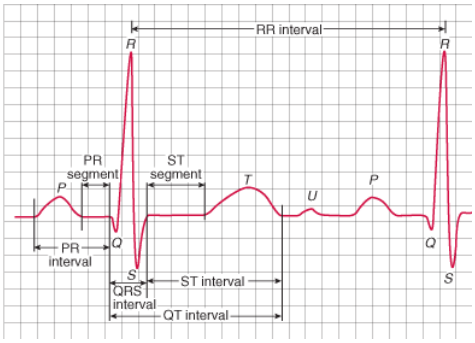

Fig. 2: Example ECG signal showing the QRS complex

B. Constructing the initial prototype

A C-code application specification mapped on a PC platform was used as an initial version of the algorithm. This specification is first analyzed in order to estimate the maximal parallelism of the algorithm using the method proposed in [10]. From the analysis, it follows that the main bottleneck of the application has its maximum performance with an instruction-level parallelism of 6. Further application analysis shows that this application needs only small buffers between the filters which could probably be mapped into register files. Therefore, a VLIW ASIP processor with 6 issue-slots and a single data memory is constructed as the initial prototype.

The initial C implementation is then ported to our initial processor (platform instantiation) to complete the initial prototype. Subsequently, the so constructed initial HW/SW prototype is passed through the simulator to verify its functionality and to create the application profile for the profile-based estimation based exploration. We used 10000 samples taken from the Pysionet ECG signal database [11] as a reference input set for the simulations.

C. Comparing exploration strategies

Table I shows an overview of our experiments with different exploration strategies, their total exploration time, and their final cost results. From this table, we can see that the coarse issue-slot exploration can be performed very quickly and that the result for both the best match and the first match strategies is exactly the same. This is due to the symmetry of the available issue-slots in our initial architecture. However, this symmetry is not available when exploring the function-units directly and there we clearly see that the first-match strategy gets stuck in a locally optimal solution.

Figure 3 illustrates this by visualizing the cost of consecutive solutions found by the different approaches. It shows

TABLE I: Different search configurations and their final solution

exploration style	exploration strategy	designs considered	total time	normalized final cost
issue-slot	best match	17	5m23s	0.503
	first match	8	2m38s	0.503
function-unit	best match	835	4h17m	0.649
	first match	83	24m54s	0.950
two-phase	best match	320	1h32m	0.498
	first match	186	48m34s	0.498

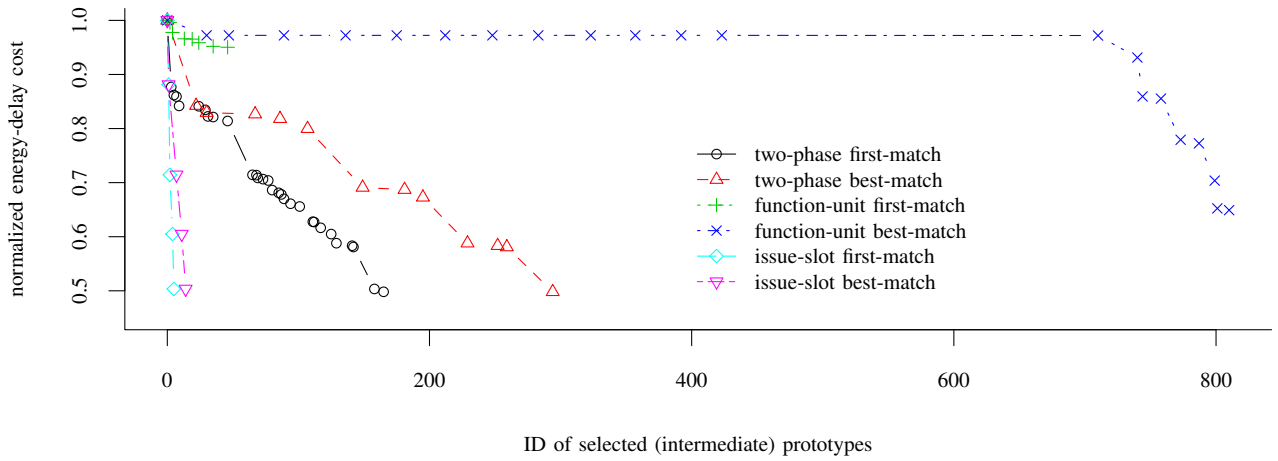

Fig. 3: The cost of selected intermediate prototypes for various search strategies. Each trace starts from the initial prototype which can be found in the top-left, points along the trace show improved energy-delay product cost solutions

how each strategy starts with the initial architecture at the top left and successively finds reduced architectures with a lower energy-delay cost. From table I we can see that the two-phase methods produce the best results, although only 1% better than the issue-slot methods. Again, both the best match and the first match method produce the same result. However, the reason why both exploration strategies produce the same result is less evident in this case and it is quite likely that different results can be obtained for both methods when a different application is being explored.

Overall, we can see that our exploration method reduced the energy consumption by over 50%, compared to the automatically created initial prototype, while keeping the total cycle count of the application within 10% of its original value. However, manually optimizing the initial prototype also focusses on optimizing the issue-width of the processor, and thus produces a similar result as the issue-slot style exploration. Our automated two-phase exploration is capable of finding a slightly better result (1% cost improvement) but greatly improves on the design time of a hand-crafted processor.

IV. Conclusion

In this paper we have introduced and compared several strategies for instruction-set architecture exploration of clustered VLIW ASIPs. We have shown that our two-phase approach gives the best results. Using this method, we are able to find a highly optimized customized VLIW ASIP. It allowed us to automatically reduce the energy-delay product of an automatically generated architecture for the ECG test-case application by 50%, thereby producing an architecture that is on par with a hand-crafted design while greatly reducing the effort required for the customization of VLIW ASIP architectures. Future research will focus on more thorough benchmarking, improving the initial prototype, and experimenting with different exploration strategies.

References

[1] FlexASP project, "TTA-based co-design environment." [Online]. Available: http://tce.cs.tut.fi/

[2] S. Aditya, B. Rau, and V. Kathail, "Automatic architecture synthesis and compiler retargeting for VLIW and EPIC processors," in *ISSS 12 — The 12th International Symposium on System Synthesis*, vol. 9, 1999.

[3] V. Kathail, S. Aditya, R. Schreiber, B. Ramakrishna Rau, D. Cronquist, and M. Sivaraman, "PICO: automatically designing custom computers," *IEEE Computer*, vol. 35, no. 9, pp. 39–47, 2002.

[4] A. Terechko, E. Le Thenaff, M. Garg, J. Van Eijndhoven, and H. Corporaal, "Inter-cluster communication models for clustered vliw processors," in *HPCA-9 — The 9th International Symposium on High-Performance Computer Architecture*. IEEE, 2003, pp. 354–364.

[5] R. Corvino, A. Gamatie, M. Geilen, and L. Jozwiak, "Design space exploration in application-specific hardware synthesis for multiple communicating nested loops," in *SAMOS XII — 12th International Conference on Embedded Computer Systems*, Samos, Greece, July 2012.

[6] P. R. Panda, F. Catthoor, N. D. Dutt, K. Danckaert, E. Brockmeyer, C. Kulkarni, A. Vandercappelle, and P. G. Kjeldsberg, "Data and memory optimization techniques for embedded systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 6, no. 2, pp. 149–206, 2001.

[7] K. Danckaert, K. Masselos, F. Cathoor, H. J. De Man, and C. Goutis, "Strategy for power-efficient design of parallel systems," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 7, no. 2, pp. 258–265, 1999.

[8] J. Hoogerbrugge and H. Corporaal, "Automatic synthesis of transport triggered processors," in *ASCI — The First Annual Conference of the Advanced School for Computing and Imaging*, 1995, pp. 1–10.

[9] J. Pan and W. J. Tompkins, "A real-time qrs detection algorithm," *IEEE Transactions on Biomedical Engineering*, no. 3, pp. 230–236, 1985.

[10] R. Jordans, R. Corvino, L. Jóźwiak, and H. Corporaal, "Exploring processor paralellism: Estimation methods and optimization strategies," in *DDECS 2013 — 16th International Symposium on Design and Diagnostics of Electronic Circuits and Systems*. Karlovy Vary, Czech Republic: IEEE, April 2013, pp. 1–6.

[11] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, pp. e215–e220, 2000 (June 13).