An Efficient Method for Energy Estimation of Application Specific Instruction-set Processors

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*—Design space exploration for ASIP instruction-set design is a very complex problem, involving a large set of architectural choices. Existing methods are usually handcrafted and time-consuming. In this paper, we propose and investigate a rapid method to estimate the energy consumption of candidate architectures for VLIW ASIP processors. The proposed method avoids the time-consuming simulation of the candidate prototypes, without any loss of accuracy in the predicted energy consumption. We experimentally show the effect of this fast cost evaluation method when used in an automated instruction-set architecture exploration. In our experiments, we compare three different methods for cost estimation and find that we can accurately predict the energy consumption of proposed architectures while avoiding simulation of the complete system.**

I. Introduction

This paper considers our efficient cost estimation method for application-specific instruction-set processors (ASIPs) usable for instruction-set architecture exploration. In particular, we propose and investigate a rapid method to estimate the energy consumption of a candidate architecture sub-set of an oversized initial VLIW ASIP processor. The proposed method avoids the time-consuming construction and simulation of the candidate prototypes, without any loss of accuracy in the predicted area and energy consumption.

Only a few approaches have been presented to automate the ASIP design space exploration, but they all assume that an existing architecture is used as a starting point. After selecting the starting point, two strategies are generally considered, *growing* [1]–[6], i.e. extending the initial architecture until no further performance is gained, or *shrinking* [1]–[4], [7], [8], i.e. removing the (almost) unused components from the architecture until performance is lost. Both approaches show substantial area, energy, and temporal performance improvements over the starting-point designs. Our proposed method applies to exploration processes using a shrinking strategy.

Performing instruction-set architecture exploration is a time consuming process. Accurately predicting the various ASIP quality metrics (i.e. area, delay, and power) within a short time is key to an effective and efficient design space exploration. Especially the cost estimation time strongly influences the number of design points that can be considered in a reasonable time. To ease this problem, ASIP architecture exploration is commonly [1]–[15] restricted to architectures composed according to a certain architecture template. This greatly simplifies the prediction of the cost metrics, as separate components can be pre-characterized individually.

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

The area, delay, and power costs of application specific processors are commonly estimated as aggregates of their (critical) component costs. Previous research (e.g. [12]–[15]) has shown that this results in fast and reasonably accurate (e.g. less than 10% error for energy [13]) cost estimations. To our knowledge however, current estimation tools [12]–[15] require the target applications' simulation traces for computing the delay and active power consumption of a design. This method requires a time-consuming simulation generating the activity traces of the processor's modules. In this paper, we argue that this simulation run is not required, and that equally accurate estimates can be obtained using a static analysis anotated with profiling information, our *static profile-based analysis* in short. To our knowledge, only one of the previous works [11] used profile-based analysis. However, they used their analysis only to predict the expected execution time for different architecture configurations. We extend this approach to perform an accurate prediction of energy consumption.

The extremely fast prototype evaluation provided through our static profile-based analysis allows our design space exploration algorithm to *a)* consider several times more design points within the same time than simulation based exploration, *b)* find better solutions in the same time or the same solutions in several times less time than the simulation based exploration, or *c)* use a more complex architecture model which provides a higher accuracy for complex modules.

In our research, we focus on VLIW ASIPs, however, the method presented in this paper works equally well for other types of processors and other component-based digital systems provided that the application is statically scheduled.

The paper is organized as follows. Section II gives an overview of our architecture exploration method and introduces our ASIP template. Section III presents our new efficient energy consumption estimation technique. Section IV experimentally demonstrates the new cost estimation method and shows its benefits for instruction-set architecture exploration. Section V concludes the paper.

II. Architecture exploration

The proposed ASIP architecture exploration evaluates and selects VLIW ASIP architectures according to a predefined template. Exploring the instruction-set architecture of VLIW ASIPs is much more complex than for simple sequential (e.g.

RISC) processors. It involves decisions on the number of parallel issue-slots the distribution of operations among the issue-slots.

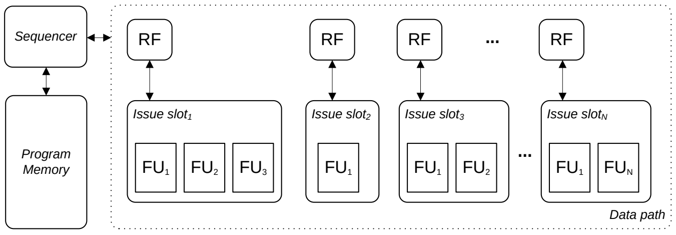


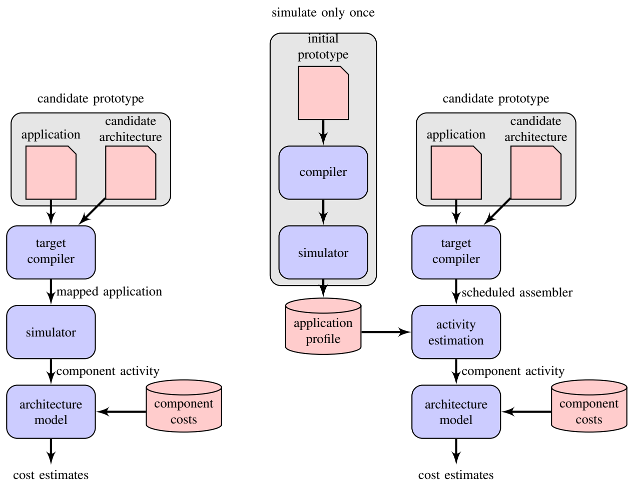
Fig. 1. Architecture template of a VLIW ASIP data-path and sequencer

Figure 1 shows the general structure of the processor template. A VLIW ASIP 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 connected 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.

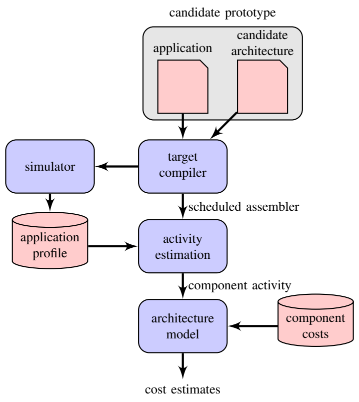
Our ASIP architecture exploration method uses a shrinking technique and assumes that an oversized ASIP architecture specialized for a coarsely optimized version of the target application (the initial prototype) is provided. This initial prototype can either be the product of a coarse, high-level, ASIP architecture exploration [16] or hand-crafted by a human designer. During our ASIP architecture exploration, we compile the target application using only a sub-set of the resources of the initial prototype. Any resource that is not used in the compiled application is expected to be removed in the final construction of the ASIP architecture. Our architecture cost estimation model takes this into account by ignoring any unused resources and correcting for partially used resources. For example, it recomputes *a)* the required width of the instruction memory, *b)* the required sizes of register files, and *c)* the complexity of the remaining result select network. Doing so, we have succeeded in producing exactly the same cost estimates as when we apply our cost model on the reduced architecture (only containing the required elements). This allows us to accurately predict the cost of proposed architecture sub-sets without actually constructing them and saves a lot of time in the architecture exploration.

III. Efficient cost estimation

Our method for the estimation of design costs is a decisive factor of our architecture exploration. Figure 2 illustrates the commonly used trace-based method (figure 2a), our static profile-based method (figure 2b), and a hybrid method (figure 2c).



(a) trace-based estimation (b) static profile-based estimation



(c) hybrid

Fig. 2. Cost estimation methods for ASIP processors

In the *trace-based* approach (cf. figure 2a), the application is compiled for the candidate processor architecture and the resulting mapped application is simulated. Activity counts for the various components of the candidate architecture are collected from a simulation trace, and the dynamic power cost of the candidate is computed using these activity counts. There are two downsides to this approach, *1)* it requires a complete simulation of each candidate prototype, and *2)* the complete simulation trace of the candidate prototype can be very large. Both make the estimation time highly dependent on the size and complexity of the target application and its test-data. In our experiments, we found that trace-based estimation easily becomes impractical, even for smaller applications. Especially when considering that the cost estimation needs to be repeated for each considered design point when exploring ASIP architectures.

Our *static profile-based* method (cf. figure 2b) only needs a single simulation run of the initial prototype to extract its application profile, consisting of the execution count of each basic block of the application code. Based on this information, we can compute the activity counts of each architecture component by analyzing the activity within basic blocks in the scheduled assembler output of the compiler, and multiplying these activities with the execution count from the application profile. This way, we avoid the problems associated with a trace-based approach and can accurately predict the application power consumption within milliseconds instead of minutes or even hours. The application execution time can also be predicted in a strictly analogous way within milliseconds. However, the application profile may not be a constant factor between different compilations of the candidate prototype and it may need to be adjusted to reflect such transformations. This is especially true when code transformations affect the control-flow structure of the candidate prototype as such transformations can influence the application's profile. This problem can be handled as explained in section III-A.

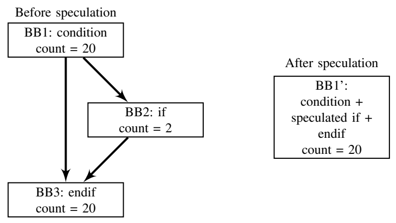
A *hybrid* (cf. figure 2c) approach is also possible. The hybrid approach uses the fast component activity computation from the static profile-based estimation but extracts a new application profile of the candidate prototype from a simulation run. This method is more robust when it comes to code transformations in the compilation process but does require a simulation of each candidate prototype.

A. Handling code transformations

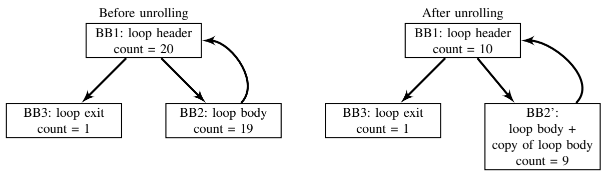
The estimation of the component activity is trivial when the target compiler does not apply control-flow transformations which influence the application's profile.

In general, we can only reuse information from the application profile when the activity estimator can recognize the transformation from the changed control-flow graph of the application. For example, transformations like *if conversion* and *speculation* (cf. figure 3a) can move an operation from a basic block in the program to its parent. In some cases, such a move causes the original basic block to become empty, which results in the basic block being removed from the control-flow graph. Such a transformation can easily be recognized from the changed control-flow graph and the profile can be adjusted accordingly. However, other transformations are more difficult to recognize. For example, *loop unrolling* (cf. figure 3b) duplicates a loop body and adjusts the iteration count of the corresponding loop to compensate for the duplication. Both the original and the unrolled loop have the same control-flow graph shape. This makes it impossible to recognize unrolling without a corresponding annotation from the compiler or an in-depth analysis of the resulting assembly code.

The above described way of handling code transformations after they have been applied is related to our current implementation of the activity prediction module outside of the compiler. Integrating the activity prediction into the compiler will make it possible to directly handle the code transformations that can currently not directly be recognized from the control-flow graph by an external tool. The overhead of merging the activity estimation with the standard compiler activities should be relatively small in most cases, as most of the optimizing compilers are already profile-driven and already have all the necessary data for the activity estimation.



(a) if conversion (speculation)



(b) loop unrolling

Fig. 3. Code transformations and their effects on the application profile

B. Improving accuracy

In principle, our method should produce exactly the same prediction results as the simulation based method. However, some operations (e.g. conditionally executed or guarded operations) may have a different cost depending on the application state. In these cases, the static profile-based estimation needs to be able to guess which is the correct cost. Our current implementation takes the worst-case cost for these operations as it has no way of recognizing the origin of the operation. Better approximations are possible when the activity prediction is combined with the compilation process. If the compiler remembers the basic block from where a guarded operation originated, it can use the profile count of that block to determine how often the guard is true and how often it is false. This would make it possible to accurately model the application state.

IV. 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 [17] 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.

In our experiments, we have compared the quality and speed of our ASIP architecture exploration when using each of the three different cost estimation techniques (trace-based, static profile-based, and hybrid) explained in section III. For this purpose, we have implemented all three cost estimation techniques into our automated instruction-set architecture exploration framework and compare the resulting exploration times.

These experiments were performed using the Pan-Tompkins QRS detection algorithm and based on the experiments presented in [18]. In [18] we presented six different search strategies, each considering between 8 and 835 design points. Figure 4 shows how the exploration time increases as dependent on the number of cosidered design points for each cost evaluation method.

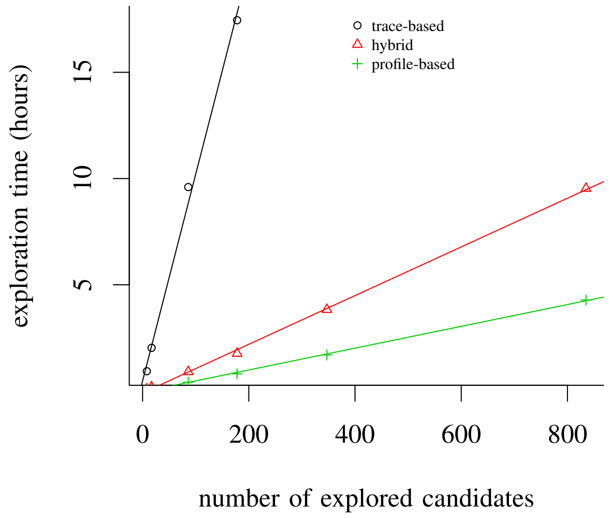


Fig. 4. The total exploration time plotted for all three different cost evaluation techniques with added regression lines

Two experiments, both using the trace-based cost estimation method for evaluation, were found to take more than one day to finish. The exploration time for these experiments has been extrapolated based on a linear regression of the exploration time as a function of the number of considered and was predicted to be approximately 35 hours (for 347 design points) and 85 hours (for 835 design points) respectively.

Figure 4 clearly shows the advantage of using our fast cost-estimation technique when it comes to exploration time. In general, our static profile-based method takes approximately 5 milliseconds to evaluate each candidate prototype. Furthermore, our currently implemented exploration strategies are very simple, this allows us to investigate the distribution of compilation time, simulation time, and cost-estimation time from our measured exploration total times. The total exploration time for our static profile-based exploration method is mostly composed of the compilation time of the prototypes. We can see that the average compilation time is approximately 18 seconds. The average simulation time, 23 seconds, can be found from the hybrid exploration as this exploration only adds the simulation. The average trace-based cost estimation time, 5 minutes, can then be found by subtracting the simulation and compilation times from the time for evaluating a single design point using the trace-based exploration.

Projecting these results onto the related work shows a big advantage of our method. For example, [12], [14] use a simulation based technique for evaluating the performance of TTA-based ASIPs. In their approach, they store the simulation-trace in a database. From this database, they then collect the aggregate usage counts through SQL queries. This is, in fact a highly efficient way of extracting the cumulative values from the trace-based cost estimation which may compete with our hybrid cost-estimation method. Using our static profile-based cost estimation technique, they could remove the simulations from their exploration flow and possibly get a speed-up of the exploration time by a factor of two or more.

V. Conclusion

In this paper, we considered our efficient ASIP cost estimation method. In particular, we proposed and discussed a novel method for the estimation of the energy consumption of VLIW ASIP architectures, and we have experimentally demonstrated its effect on the instruction-set architecture exploration time. Using our method, we can explore more than twice as many design points within the same time compared to other approaches, without any loss in accuracy.

References

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

[2] H. Corporaal and J. Hoogerbrugge, “Cosynthesis with the MOVE framework,” in *Modelling, Analysis, and Simulation*, 1996, pp. 184–189.

[3] S. Aditya, B. Rau, and V. Kathail, “Automatic architecture synthesis and compiler retargeting for VLIW and EPIC processors,” in *Proc. of ISSS*, vol. 9, 1999.

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

[5] L. Pozzi, K. Atasu, and P. Ienne, “Exact and approximate algorithms for the extension of embedded processor instruction sets,” *IEEE Tr.on CAD-ICS*, vol. 25, no. 7, pp. 1209–1229, 2006.

[6] C. Wolinski and K. Kuchcinski, “Automatic selection of application-specific reconfigurable processor extensions,” in *DATE*. IEEE, 2008.

[7] J. Matai, J. Oberg, A. Irturk, T. Kim, and R. Kastner, “Trimmed VLIW: Moving application specific processors towards high level synthesis,” in *The Electronic System Level Synthesis Conference*, 2012.

[8] A. Irturk, J. Matai, J. Oberg, J. Su, and R. Kastner, “Simulate and eliminate: A top-to-bottom design methodology for automatic generation of application specific architectures,” *IEEE Tr.on CAD ICS*, vol. 30, no. 8, pp. 1–11, August 2011.

[9] Synopsys, “Synopsys Processor Designer.” [Online]. Available: http://www.synopsys.com

[10] Target, “Target Compiler Technologies: IP Designer.” [Online]. Available: http://www.retarget.com/

[11] J. Hoogerbrugge and H. Corporaal, “Automatic synthesis of transport triggered processors,” in *ASCI*, 1995, pp. 1–10.

[12] T. Pitkänen, T. Rantanen, A. Cilio, and J. Takala, “Hardware cost estimation for application-specific processor design,” *Embedded Computer Systems: Architectures, Modeling, and Simulation*, pp. 251–264, 2005.

[13] S. Ahuja, D. A. Mathaikutty, G. Singh, J. Stetzer, S. K. Shukla, and A. Dingankar, “Power estimation methodology for a high-level synthesis framework,” in *ISQED*. IEEE, 2009, pp. 541–546.

[14] J. Mäntyneva, “Automated design space exploration of transport-triggered architectures,” Master’s thesis, Tampere University of Technology, Tampere, Finland, July 2009.

[15] G. Qu, N. Kawabe, K. Usami, and M. Potkonjak, “Function-level power estimation methodology for microprocessors,” in *DAC*. ACM, 2000, pp. 810–813.

[16] R. Corvino, A. Gamatie, M. Geilen, and L. Jozwiak, “Design space exploration in application-specific hardware synthesis for multiple communicating nested loops,” in *SAMOS*, Samos, Greece, July 2012.

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

[18] R. Jordans, R. Corvino, L. Jóźwiak, and H. Corporaal, “Instruction-set architecture exploration strategies for deeply clustered vliw asips,” in *ECyPS 2013*, Budva, Montenegro, June 2013, pp. 1–4.