Exploring Processor Parallelism: Estimation Methods and Optimization Strategies

Roel Jordans, Rosilde Corvino, Lech Jóźwiak, Henk Corporaal
Department of Electrical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands
{r.jordans, r.corvino, l.jozwiak, h.corporaal}@tue.nl

***Abstract*—Former research on automatic exploration of ASIP architectures mostly focused on either the internal memory hierarchy, or the addition of complex custom operations to RISC based architectures. This paper focuses on VLIW architectures and, more specifically, on automating the selection of an application specific VLIW issue-width. An accurate and efficient issue-width estimation strongly influences all the important processor properties (e.g. processing speed, silicon area, and power consumption). We first compare different methods for estimating the required issue-width, and subsequently introduce a new force-based parallelism measure which is capable of estimating the required issue-width within 3% on average. Moreover, we show that we can quickly estimate the latency–parallelism Pareto-front of an example ECG application with less than 10% error using our issue-width estimations.**

I. Introduction[1]

Highly customized application specific instruction-set processors (ASIPs) are increasingly used in products requiring programmability, high-performance, and/or a limited energy consumption. Several industrial strength tool-flows, e.g. [2]–[6], are available to specify such processors, however, the exploration of design-space trade-offs is generally left to the user. Only a few approaches have been presented to automate the design space exploration, but they all assume that an existing architecture is used as a starting point. After selecting this starting point, two strategies are generally considered, *growing* [4]–[9], i.e. extending the initial architecture until no further performance is gained, or *shrinking* [4]–[7], [10], [11], 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. Shrinking and growing can be time-consuming processes, depending on the method used to evaluate the different architectural solutions. Selecting a good search strategy is therefore an important step towards reducing the required design-time.

Through analysis of different kinds of hand crafted designs utilizing the target VLIW technology [12]–[15], we found that the processor area is distributed on average as follows: 30–40% is occupied by the VLIW data-path, 25% by program memory, 25% by data memories, and 10–20% by the register files. Moreover, the data memory and register file sizes (35–45%) are largely determined by the application mapping, while the program memory size (25%) is largely determined by the VLIW issue-width, and specifically the number of operations that can be executed in parallel by the VLIW. The data-path size is substantially influenced by the issue-width, but also by some other design choices such as instruction-set selection and custom operations. Moreover, the issue-width influences not only the memory and data-path sizes, but also their speed and power consumption. This makes selecting the correct issue-width a critical part of the VLIW ASIP design.

This paper focuses on estimating the minimal VLIW issue-width that guarantees the required performance of the target application. In the past, several methods have been presented for estimating the issue-width of a VLIW ASIP [16]–[21]. They estimate the issue-width required for obtaining a specific maximum latency when executing a specific part of the application. Similarly to most of the previously proposed algorithms, we also work on basic blocks, i.e. application parts with single entry and exit points. The control-flow of the application is not considered. In our method, we can account for both basic blocks and larger application parts (e.g. superblocks or hyperblocks). This is possible because our method is independent on the precise scheduling algorithm and will work at any granularity, as long as the scheduler is deterministic.

This paper addresses two very important issues not addressed in the previous works. Firstly, we investigate various methods, including new one, for *estimating* the required issue-width without having to completely schedule the application, and provide a quantitative comparison of these methods. Secondly, we assess the advantage of the newly proposed method over the existing when the ASIP development environment is a black-box. The previously presented methods of growing and shrinking can require many runs of a time-consuming scheduling or synthesis process. Scheduler runs are especially costly in our framework, because we can only use the compiler for our ASIP development environment as a black-box. Compiling the a relatively small target application for a given processor configuration and simulating the resulting application mapping can already take 1–2 minutes. This makes the exploration time strongly dependent on the selection of an appropriate exploration strategy and its starting-point. This paper compares several existing, as well as, newly proposed strategies for *finding* the required issue-width, and we investigate the required number of scheduler runs for each of them.

This paper is organized as follows. Section II briefly intro-

[1]This work was performed as part of the European project ASAM [1] that has been partially funded by ARTEMIS Joint Undertaking, grant no. 100265.

duces the existing methods for VLIW issue-width estimation and discusses the related research. Section III provides more detail on the considered parallelism estimation methods and presents a quantitative comparison. Section IV presents different exploration strategies to find the required issue-width. Section V discusses the extensions of the method to account for the control-flow. Section VI concludes the paper.

II. Issue-width estimation

Traditionally, the issue-width decision for a VLIW processor has been based on an analysis of the available instruction-level parallelism in the target application. Previous research [16]–[18], [22] focused mostly on computing the *average* parallelism that can be obtained for a specific application on an unconstrained platform, only considering the *true dependencies* imposed by the target application. Wall [17] being a notable exception, focussing on the upper-bound of parallelism over traces of a complete applications. More recently, Cabezas and Stanley-Marbell [20] published a method for computing the *distribution* of parallelism across a program's execution. They showed that, in some cases, over 80% of the programs execution stream provides a parallelism that is an order of magnitude smaller than the mean value. Our goal is to provide real-time performance, it is therefore important that enough parallelism is provided for the high-performance parts of the application, even when these parts are only a small portion of the application. In order to better quantify the high variation in parallelism Theobald *et. al* [19] defined their *smoothability* metric. This metric provides a score from 0–100%, a program which exhibits short bursts of high parallelism separated by long sequential sections will get a low score while a program that has a more evenly distributed parallelism will obtain a higher score.

While both the parallelism distribution and the smoothability metric do provide insight in the parallelism variability of a whole program, they only provide a lower-bound on the parallelism required for obtaining a specific performance. Our method attempts to compute the exact parallelism required for obtaining a specific performance for a given program part with real-time constraints. The computed parallelism can be translated directly in an issue-width requirement for a VLIW ASIP, or can be explored as part of a high-level design space exploration such as the data-memory organization exploration.

In this paper, we will compare several methods to compute instruction-level parallelism based on their appropriateness for issue-width estimation and their computational complexity. The considered methods are as follows:

1) *Average parallelism (AP)* [16]–[18], [21], [22], computed by dividing the number of operations by the expected latency of the program (part).
2) *Force based parallelism (FBP)* a contribution of this paper introduced in section III-A2.
3) *Maximum parallelism (MP)* [20], [21], computed by finding the maximal number of operations which can be scheduled in parallel.
4) *Required parallelism (RP)* [17], [21], computing the minimal upper-bound on the parallelism as required for scheduling an application part within a given latency bound.

In order to ensure practical relevance of our solutions and provide more control on the issue-width estimation by the end-user, we have added the option of explicitly constraining specific types of hardware resources in the exploration. Common uses of this are constraining the number of ports on the data memories and/or constraining the number of instances of specific (costly) resources (e.g. a maximum number of dividers).

III. Parallelism estimation

This section introduces three different methods for providing a quick estimate of the parallelism of an application, including our novel force based parallelism, and presents the results of our experimental research with these methods.

A. Methods

1) Average parallelism: Perhaps the most commonly used measure to estimate the parallelism of an application is the average parallelism. It is computed by dividing the number of operations by the required latency and provides a lower bound on the required issue-width.

$$\Phi_{AP} = \frac{|V|}{\lambda}$$

2) Force based parallelism: Another estimate of the required issue-width can be computed using a concept found in Force Directed Scheduling [23].

During force directed scheduling, a *distribution graph* is computed from ASAP-ALAP schedule intervals as the sum of the probabilities of all operations executed which may be executed for each given cycle. An example is shown in figure 1. Both operations v_1 and v_3 can be scheduled at 3 different moments as shown by their ASAP-ALAP schedule interval in figure 1b. Their scheduling probability is therefore $1/3$ for each cycle. The distribution of the summed scheduling probabilities is shown in figure 1c. The maximum of the distribution graph can be used to estimate the parallelism. For example, from figure 1c one will find the value of $1^2/3$, which could lead to the conclusion that a parallelism of 2 is an appropriate solution. Computing the force based parallelism for the 8 point IDCT algorithm results in a value of 7.85, closely corresponding to the required parallelism of 8.

It should be noted that the force based parallelism does not provide an upper nor lower bound on the parallelism. Figure 1 shows an under-estimation while 2 shows a graph that results in an over-estimation. In this example, the operation v_x can be scheduled in parallel to operations v_1 and v_2. This results in a FBP of 2.5 whereas the required parallelism for this graph is only 2. More extreme cases, resulting in larger overestimations, can be constructed in a similar fashion.

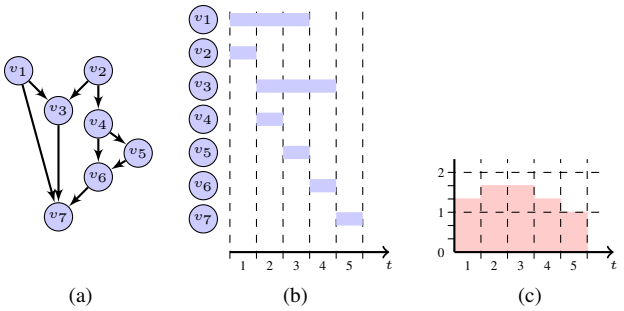

Fig. 1. Example DFG (a) with ASAP-ALAP schedule intervals (b), and the corresponding distribution graph used in computing the force based parallelism (c).

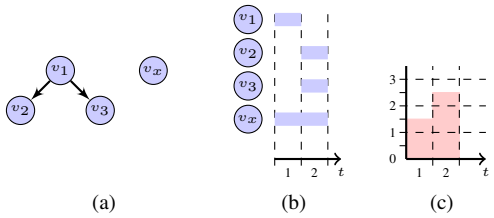

Fig. 2. An example DFG resulting in an over-estimation of the required parallelism by the FBP method.

3) Maximum parallelism: The maximum parallelism [21] can be computed in a way that is similar to the computation of the force based parallelism. The only difference being that all nodes are counted with the same weight and that the length of the schedule interval is not taken into account as shown in figure 3. Computing the maximum parallelism for the example DFG shown in figure 1 results in a parallelism of 3, a parallelism which cannot be obtained in any valid schedule of the DFG, but which, when provided, does guarantee the required latency.

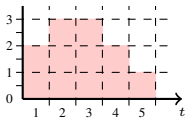

Fig. 3. Potential parallelism graph used in computing the maximum parallelism for the example DFG given in figure 1a.

Care should be taken though when computing the maximum parallelism under resource constraints, as the minimal schedule latency will increase due to the added constraints.

B. Experimental results

All three methods for estimating VLIW issue-width have been implemented at the IR level of the LLVM compiler framework [24] and used to compare their respective quality as an approximation of the required issue-width. The experimental results have been analyzed using R [25].

The experiments reported in this paper have been performed on a set of 3667 basic blocks taken from an MPEG4-SP encoder application. This application contains a representative set of basic blocks showing different kinds of processing. Each of these basic blocks was taken as a separate experiment and the parallelism was estimated for it's ASAP schedule. Almost all these basic blocks fall within the range of 1–150 operations but there are several larger blocks with sizes up to 1279 operations (e.g. `fdct`). In the experiments, all three parallelism estimation methods have been applied to each of the basic blocks with and without adding a constraint on the number of parallel memory accesses. The memory constraint was selected as a common example of an explicit resource constraint and other resource constraints (e.g. constraining costly function units) can be added in a similar fashion.

The experiments have been grouped as *unconstrained* and *constrained* cases, referring respectively to the experiments without and those with the added resource constraint. Figure 4 shows a box-plot of the results obtained from our experiments normalized to the required parallelism (RP) of each basic block found through a full search.

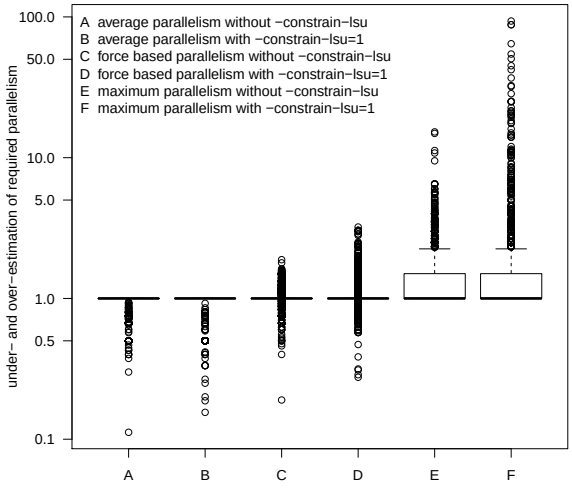

Fig. 4. The deviation of various parallelism estimation methods from the required parallelism. Normalized to the required parallelism.

From our experiments we found, as expected, that the AP provides a *lower-bound* on the VLIW issue-width required for executing the application, while the MP provides an *upper-bound*. The AP underestimates the RP, on average by 7% independent of the presence of extra resource constraints. However, this under-estimation of issue-width can be up to a factor of 8.9x as shown in our experiments. The MP provides an overestimation of up to two orders of magnitude and, on average, 31% for the unconstrained and 72% for the constrained experiments. The FBP delivers the most accurate estimation, on average resulting in a 3% overestimation for the unconstrained and a 6% overestimation for the constrained experiments. The worst-case result for FBP is an underestimation of the required parallelism by 5.2x.

C. Conclusion

From our experiments, it follows that the average parallelism usually provides a quite accurate view of the issue-width requirement of an application. However, in the worst case, it underestimated the required issue-width by a factor of 8.9x.

We also conclude that, the maximum parallelism provides an upper bound with a large error margin. We therefore consider the maximum parallelism to be less useful for a direct issue-width estimation. However, as it will be shown in the next section, the maximum parallelism can be used to create an improved search strategy for finding the required parallelism.

Finally, we have shown that the force-based parallelism estimation is more precise than the average parallelism estimation and has a much smaller worst-case deviation. Our force-based parallelism measure should therefore be the preferred method for making initial estimates of the required parallelism.

IV. VLIW ISSUE-WIDTH EXPLORATION

The required issue-width can be computed both using *growing* and *shrinking* strategies, where growing is the most common [4], [7]–[9] strategy for exploring parallelism. However, applying a linear search for finding the optimal issue-width that guarantees a given latency may not be the best choice since this requires Φ_{RP} iterations of the scheduling algorithm, with Φ_{RP} equal to the required parallelism.

Another possibility, when an upper-bound to the parallelism is known, for example through computing the maximum parallelism, is to perform a binary search, which requires a number of scheduler iterations logarithmic to the size of the considered parallelism range.

Both the previous work (e.g. [17]) and our initial experiments have shown that the required issue-width Φ_{RP} usually has a relatively small value in comparison to the maximum parallelism Φ_{MP}, often even smaller than $\log \Phi_{MP}$. For example (cf. figure 5), a naive growing technique would find the Φ_{RP} in 8 scheduling steps. A binary search strategy starting on the range 1–Φ_{MP} would also require 8 scheduling steps. Starting the growing technique at the average parallelism Φ_{AP} improves the performance of the growing strategy by reducing the number of required scheduling steps to 4. Similarly, changing the range partitioning within the binary search algorithm can help in improving the average performance of the binary search strategy. For example, dividing the solution range into the lower-third and upper-two-thirds partitions results in 5 scheduling steps. Furthermore, it is also possible to select the first pivot independent of the division strategy of the remaining ranges. Using the force-based parallelism Φ_{FBP} as first pivot, and to continue from there with a balanced binary search, results in 4 scheduling steps. Finding the best starting-point and search strategy are therefore critical to an optimal performance of the required parallelism estimation.

A. Possible search strategies

As stated above, two main search strategies are possible, linear search and binary search. Several starting points are possible for both of them. This section will further explain the different possibilities for both strategies.

1) Linear search: Both growing and shrinking strategies can be combined with linear search depending on the selected starting point. The simplest approach is to start at a parallelism of 1 and increase the parallelism until a satisfactory latency is obtained.

A faster way to obtain a single design point satisfying the latency requirements is to start from a measure which is closer to the final result. Both the average parallelism and the force based parallelism are good candidates for this. However, both Φ_{AP} and Φ_{FBP} are fractional numbers which makes the selection of the rounding strategy important. The average parallelism can be rounded up to the next integer since it provides a lower-bound. However, deciding upon the rounding for the force based parallelism estimation is not so straightforward as it can both over- and under-estimate the required parallelism. In our experiments we therefore provide the results for separate experiments using either rounded up (ceil Φ_{FBP}) or rounded down (floor Φ_{FBP}) values as starting points.

A downside of starting at Φ_{FBP} is that, if the initial estimate provides us with a satisfying result, it is required to verify that this is the optimal result, which requires an extra run of the scheduler with a parallelism of one less. This extra scheduler run can be avoided if Φ_{AP} and Φ_{FBP} are equal since this implicitly verifies optimality by proving that Φ_{FBP} is in fact the minimal value in such cases.

2) Binary search: This method requires a starting range and a rule for selecting the pivot. Only one reasonable starting range is available from our parallelism estimation methods which can provide both upper- and lower-bounds for the required parallelism. However, the balance of the search and selection of an initial pivot are critical to the performance of the binary search, as shown in the example accompanying figure 5. Both the balance of the search and the selection of the initial pivot have therefore been explored in our experiments. The first set of experiments with the binary search strategy varies the search balance, through a parameter α of the algorithm. The second set of experiments with the binary search strategy used Φ_{FBP} as initial pivot and was performed for the same values of α. Algorithm 1 shows how such an unbalanced binary search can be implemented using a balancing parameter α while using Φ_{FBP} as first pivot.

B. Experimental results

For our experiments we used the same framework and benchmark set as was used for the comparison presented in the previous section. We have again grouped the experiments as *unconstrained* and *constrained* cases, referring respectively to the experiments without and those with the added resource constraint. This time we focused on the number of scheduler runs required for finding the required parallelism for obtaining an ASAP schedule. We did not count the extra scheduler run required for determining the ASAP latency of the blocks. The results of our experiments are presented in table I. The results of the binary search strategy with Φ_{FBP} as initial pivot are only shown for an α of 2 as variations in α had negligible influence on the number of scheduler runs.

It should be noted that the quality of our results is strongly dependent on the quality of the internal scheduler. Our im-

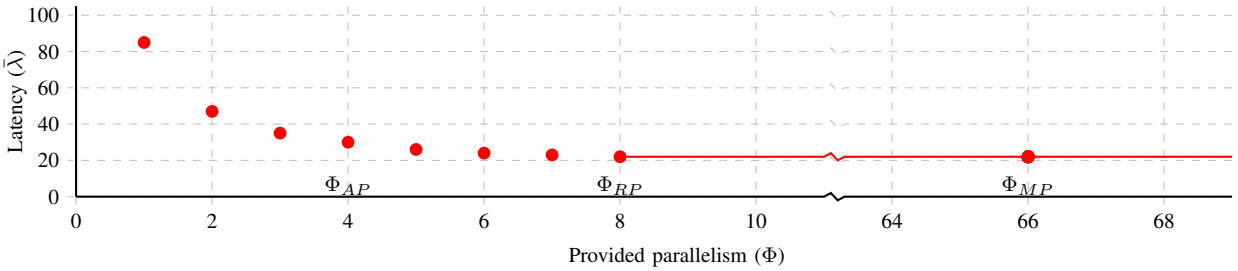

Fig. 5. Latency versus parallelism plot for an 8-point IDCT function with average parallelism (Φ_{AP}), required parallelism for its ASAP latency (Φ_{RP}), and maximum parallelism (Φ_{MP}) marked

Algorithm 1 Computing required parallelism using an unbalanced binary search where the balance is controlled by parameter α

Require: Basic block BB and latency bound $\bar{\lambda}$
Ensure: Calculate the issue-width Φ of BB such that the scheduled latency λ is the greatest integer inferior to $\bar{\lambda}$

$\Phi_{max} \leftarrow \Phi_{MP}$
$\Phi_{min} \leftarrow \Phi_{AP}$
$\Phi_{pivot} \leftarrow \text{floor}\ (\Phi_{FBP})$
while $\Phi_{max} > \Phi_{min}$ **do**
  $\lambda \leftarrow \text{Schedule}\ (BB, \Phi_{pivot})$
  if $\lambda > \bar{\lambda}$ **then**
    $\Phi_{min} \leftarrow \Phi_{pivot} + 1$
  else
    $\Phi_{max} \leftarrow \Phi_{pivot}$
  end if
  $\Phi_{pivot} \leftarrow \Phi_{min} + \lfloor(\Phi_{max} - \Phi_{min})/\alpha\rfloor$
end while
return $\Phi_{RP} \leftarrow \Phi_{min}$

TABLE I
TOTAL NUMBER OF SCHEDULER ITERATIONS DURING RP SEARCH OVER ALL 3667 BLOCKS OF AN MPEG4-SP ENCODER FOR BOTH LINEAR AND BINARY SEARCH STRATEGY

method	start	α	unconstrained	constrained
linear	1		8523	6797
	AP		4705	4457
	ceil(FBP)		7787	7671
	floor(FBP)		7238	7091
binary search	AP–MP	2	6134	6463
		5	5560	5690
		10	5521	5565
		25	5510	5513
binary search	AP–FBP–MP	2	4149	4198

plementation uses a list scheduler but other schedulers can be used as long as they provide a deterministic result. From the many available list-scheduler heuristics [26], we selected the *dependency height* as main criterion and we *prioritize* load-operations in order to increase the scheduler's freedom for scheduling shorter sequences. We found that using this combination of instruction selection criteria we can obtain a high quality result[2] without sacrificing computational complexity. Observe that it is possible to achieve higher quality results using more effective scheduling algorithms, but at the cost of their higher computational complexity. Our RP method only requires that the scheduler is deterministic, but is otherwise independent of the specific scheduling algorithm used.

[2]On average within 3% of the actual required parallelism as computed using an optimal scheduler based on constraint programming [27]

C. Conclusion

Using a binary search strategy with the FBP estimate as the first search point to find a single design point for a parallelism-latency trade-off optimization results in the fewest required search steps. In our experiments this resulted in a 11% reduction from the currently used method of linear search from the average parallelism for the unconstrained experiments and a 6% reduction for the constrained experiments. We therefore recommend using a combination of our FBP metric with binary search when looking for a single design point, but recognize that the full Pareto-front is more interesting in many cases. Computing the Pareto-front requires a full linear search.

V. APPLICATION LEVEL ANALYSIS

The methods presented above for VLIW issue-width estimation effectively provide useful results, but they only provide information based on analysis of specific basic blocks within the application. However, constructing a VLIW ASIP capable of running a complete target application requires predicting the issue-width for larger parts of the application code. Two solutions are possible for this issue.

Firstly, it is possible to compute the required parallelism for larger application parts [17] by extending the currently used scheduling algorithms to work with super-blocks or traces. Such an extension of our current method is straight forward, as our approach is independent of the scheduling algorithms used internally. This approach can also be used when integrating more complex scheduling techniques such as software pipelining.

Secondly, the information provided by the analysis of the basic blocks can be combined with a high-level information on the control-flow of the application. This can be achieved in many ways, especially if further information on the issue-width versus latency trade-off is known for each basic block. In particular, combining the Pareto-fronts of several basic blocks allows us to extend our issue-width estimation to account for larger application parts.

The total execution latency of the application's control-flow-graph can be computed for a specific issue-width by adding the execution latencies of its various blocks (basic blocks or super-blocks), weighted by their execution counts. The execution counts of blocks can either be obtained through static analysis or by profiling of the application. Initial experiments on a small ECG application [15] have shown that this method is capable of predicting all three points of the latency-parallelism Pareto-front, shown in table II, with less than 10% error in less than one second. Our initial analysis of this example case has shown that main source of inaccuracy can be found in *a)* the abstractions of the LLVM IR (e.g. static-single-assignment form) at which the estimation is performed, and *b)* the lack of intra-block optimizations (e.g. speculation).

TABLE II
PREDICTED AND SIMULATED CYCLE COUNTS FOR ECG

issue-width	predicted	simulated	ratio
1	47648	52012	109%
2	31992	34373	107%
3	31977	34329	107%

VI. CONCLUSION

In this paper, we presented and compared three methods for estimating the required VLIW issue-width of an ASIP for a target application. We have found that our force based parallelism estimation proposed in this paper provides a 3% over-estimation on average, outperforming the commonly used average parallelism estimation regarding both the average and maximum error. Moreover, our algorithms can be controlled by the ASIP designer to account for resource constraints such as the maximum number of instances for a specific type of function unit (e.g. a divider), etc. We have also presented several different strategies for obtaining the required VLIW issue-width for a specific latency and were able to reduce the number of required scheduler runs by 11% on average. We also found that the parallelism–latency tradeoff is often more important. Using the parallelism–latency tradeoff information, we were able to predict the execution time of an ECG application with less than 10% error within a fraction of the time required when exploring the issue-width using the ASIP development framework. This way, we substantially improved the effectiveness and efficiency of automatic design space exploration for VLIW based ASIP architectures.

REFERENCES

[1] ASAM, "Project website." [Online]. Available: http://www.asam-project.org

[2] Synopsys, "Synopsys Processor Designer." [Online]. Available: http://www.synopsys.com

[3] Target, "Target Compiler Technologies: IP Designer." [Online]. Available: http://www.retarget.com/

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

[5] 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.

[6] 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.

[7] H. Corporaal and J. Hoogerbrugge, "Cosynthesis with the MOVE framework," in *Symposium on Modelling, Analysis, and Simulation*, 1996, pp. 184–189.

[8] L. Pozzi, K. Atasu, and P. Ienne, "Exact and approximate algorithms for the extension of embedded processor instruction sets," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 25, no. 7, pp. 1209–1229, 2006.

[9] C. Wolinski and K. Kuchcinski, "Automatic selection of application-specific reconfigurable processor extensions," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2008*. IEEE, 2008.

[10] 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.

[11] 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 Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 8, pp. 1–11, August 2011.

[12] P. Qiao, "Design and optimization of digital hearing aid system based on Silicon Hive technology," Master's thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, August 2010. [Online]. Available: http://alexandria.tue.nl/repository/books/709025.pdf

[13] P. Qiao, H. Corporaal, and M. Lindwer, "A 0.964 mW digital hearing aid system," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*. IEEE, 2011, pp. 1–4.

[14] Y. Ökmen, "SIMD floating point processor and efficient implementation of ray tracing algorithm," Master's thesis, TU Delft, Delft, The Netherlands, October 2011. [Online]. Available: http://repository.tudelft.nl/assets/uuid:b0a8ae03-18b9-4a0e-9761-64ffd2851074/Yunus_Okmen_MSc_Thesis.pdf

[15] E. Diken, R. Jordans, R. Corvino, and L. Jozwiak, "Application analysis driven asip-based system synthesis for ecg," in *Embedded World Conference*, Germany, February 2012.

[16] G. S. Tjaden and M. J. Flynn, "Detection and parallel execution of independent instructions," *IEEE Transactions on Computers*, vol. 19, no. 10, pp. 889–895, October 1970.

[17] D. W. Wall, "Limits of instruction-level parallelism," in *ASPLOS 1991 – Proceedings of the fourth international conference on architectural support for programming languages and operating systems*, 1991, pp. 176–188.

[18] T. M. Austin and G. S. Sohi, "Dynamic dependency analysis of ordinary programs," in *ISCA 1992 – Proceedings of the 19th annual international symposium on Computer architecture*, 1992, pp. 342–351.

[19] K. B. Theobald, G. R. Gao, and L. J. Hendren, "On the limits of program parallelism and its smoothability," in *MICRO 1992 – Proceedings of the 25th annual symposium on microarchitecture*, 1992, pp. 10–19.

[20] V. C. Cabezas and P. Stanley-Marbell, "Parallelism and data movement characterization of contemporary application classes," in *SPAA 2011 – Proceedings of the 23rd ACM symposium on parallelism in algorithms and architectures*. New York, NY, USA: ACM, 2011, pp. 95–104.

[21] R. Jordans, R. Corvino, and L. Jóźwiak, "Algorithm parallelism estimation for constraining instruction-set synthesis for VLIW processors," in *DSD 2012 - 15th Euromicro Conference on Digital System Design*, Cesme, Izmir, Turkey, September 2012, pp. 1–4.

[22] E. M. Riseman and C. C. Foster, "The inhibition of potential parallelism by conditional jumps," *IEEE Transactions on Computers*, vol. 21, no. 12, pp. 1405–1411, December 1972.

[23] P. Paulin and J. Knight, "Force-directed scheduling for the behavioral synthesis of asics," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 8, no. 6, pp. 661–679, 1989.

[24] LLVM, "Project website." [Online]. Available: http://www.llvm.org

[25] The R project for statistical computing, "Project website." [Online]. Available: http://www.r-project.org/

[26] M. Smotherman, S. Krishnamurthy, P. S. Aravind, and D. Hunnicutt, "Efficient dag construction and heuristic calculation for instruction scheduling," in *MICRO 1991 – Proceedings of the 24th anual international symposium on microarchitecture*, 1991, pp. 93–102.

[27] A. M. Malik, J. McInnes, and P. van Beek, "Optimal basic block instruction scheduling for multiple-issue processors using constraint programming," *International Journal on Artificial Inteligence Tools*, vol. 17, no. 1, pp. 37–54, February 2008.