Improving Virtual Prototype Driven Hardware Optimization by Merging Instruction Sequences

Jan Zielasko[1,2] Rune Krauss[1] Marcel Merten[1] Rolf Drechsler[1,2]
[1]Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
[2]Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
Jan.Zielasko@dfki.de, krauss@uni-bremen.de, mar_mer@uni-bremen.de, drechsler@uni-bremen.de

***Abstract*—Tailoring hardware to an application significantly enhances its performance compared to using a general-purpose processor. While hardware optimization is essential to meet the user requirements for resource-constrained embedded systems, it generally entails considerable costs and a high level of effort. In recent work virtual prototypes have been shown to be an effective analysis tool for guiding this process. In best-case scenarios, it is possible to identify a single recurring instruction sequence that covers approximately 55 % of all executed instructions and is thus suitable for optimization by a *Hardware Accelerator* (HA). However, challenges arise for applications where each identified sequence only covers a small fraction of the total execution. In order to achieve comparable coverage, several HAs can be designed, but this also multiplies the hardware costs. To address these issues, this work proposes an approach to extend and merge identified sequences allowing the design of a single HA for the merged sequence. Experiments show that this approach significantly increases the coverage achievable with a single HA while the resulting performance loss is negligible compared to building multiple HAs.**

***Index Terms*—Virtual prototypes, ASIC, embedded systems, execution tracing, hardware optimization**

I. Introduction

With the ever-increasing demand for high-performance and low-power applications in the areas of IoT and embedded systems, selecting suitable hardware designs for applications is becoming increasingly important [1]. Despite the availability of a wide range of general-purpose processor designs, they fall short of optimal performance as most systems in the aforementioned areas operate with only a single application [2]. Instead of using a general-purpose processor, better results can be achieved by building a customized application-specific integrated circuit that is tailored to the specific requirements of the application [3]. Unfortunately, creating a fully customized design is a complex and expensive process requiring a high level of expertise and effort [4].

Rather than starting the design process from scratch, it is more efficient to start with an existing design and tailor it to the respective application. Previous work has proposed several approaches showing that the design effort is drastically reduced while promising a similar performance improvement as using application-specific integrated circuits: instruction set simulators [5], *Virtual Prototypes* (VPs) [6], and register-transfer level simulations [2]. The VP driven approach as proposed in [7] and [6] uses VPs as a platform for system analysis and therefore combines the advantages of listed high-level and low-level approaches by running target applications on the VP. In this context, the VP analyzes their executions w. r. t. recurring instruction patterns in order to automatically identify the most promising candidates (instruction sequences) for hardware optimization. Experimental results achieved in [6] show that using this approach allows the identification of promising instruction sequences for many different embedded applications: In best-case scenarios, it is even possible to identify a sequence covering around 55 % of all executed instructions. Especially for such a sequence, it is worth designing a *Hardware Accelerator* (HA), which can then improve the performance for these 55 % of instructions.

However, there are applications where each discovered sequence covers only a small fraction of the total execution. Such applications can pose a challenge for the described approach as multiple optimization candidates are recommended, which often only cover around 15 % of the executed instructions. Although it is conceivable to build multiple HAs, this can be expensive due to the increase of hardware costs.

To address these issues, we propose a novel technique for extending and merging instruction sequences. This approach allows a single HA design for a merged sequence that has a higher execution coverage compared to a HA for a single sequence. Experiments confirm that the coverage almost doubles and on average 10 single HAs needed for the same coverage can be replaced with a negligible performance loss.

In summary, the main contributions are as follows:

- Extension of the VP driven approach for providing a variety of instruction sequences;
- Merging of instruction sequences to increase execution coverage and save HAs;
- Evaluation of merging effectiveness based on the variety of instruction sequences.

To stimulate further research, the used VP[1] and our proposed approach[2] are provided as open-source software.

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within projects Scale4Edge under grant no. 16ME0127, ECXL under grant no. 01IW22002 and VE-HEP under grant no. 16KIS1342.

[1]https://github.com/agra-uni-bremen/opt-vp
[2]https://github.com/agra-uni-bremen/opt-seq

II. Preliminaries

This section introduces basic principles in an attempt to keep the paper self-contained. While Section II-A explains some aspects of the RISC-V *Instruction Set Architecture* (ISA), Section II-B briefly describes VPs.

A. RISC-V

RISC-V is an open standard ISA providing a foundation for modular processor design [8]. To this end, it defines an integer instruction set (I), which is the only mandatory base for an implementation. In addition to this set, any of the available standard extensions can be added to extend its functionality [9]: integer multiplication and division (M), compressed instructions (C), etc. A comprehensive survey is available in [8].

The fact that RISC-V follows a load-store architecture and supports unconditional execution makes it suitable for this work as it simplifies the implementation of extending and merging instruction sequences. For example, less information needs to be traced during execution and the simplicity of the instructions enables straightforward merging of sequences.

B. Virtual Prototypes

A VP is an executable abstract model of an entire hardware platform that simulates its architecture at the electronic system level [10]. VPs enable early software development and other system-level use cases before the actual hardware is built shortening the time-to-market. Compared to alternative techniques such as traditional instruction set simulators, they offer greater accuracy, for example in terms of timing behavior [5]. Although simulations on the register-transfer level offer more accuracy than VPs, they are much slower [2].

VPs are suited for our use case as we can quickly obtain all the necessary information at the electronic system level. Thus, the RISC-V-based VP[3] introduced in [11] is used, which is implemented using SystemC transaction-level modeling [12].

III. Related Work

To the best of the authors' knowledge, *RVOPT-VP*[4] proposed in [6] is actually the only RISC-V-based VP driven approach for tailoring hardware to application requirements. Specifically, it extends the RISC-V VP introduced in the last section with a tracing and analysis module for identifying promising instruction sequences so that an existing design can be tailored to a specific application.

In the first step, RVOPT-VP traces the execution of an application based on inputs to generate a bounded execution tree for each encountered instruction, where each tree stores information about every sequence that starts with the instruction at its root node. In a tree, a valid sequence is any path starting from the root node to any node of the same tree, where each node stores a list of data dependencies to previous nodes. In the second step, RVOPT-VP's trees are processed by its analysis module using a score function

$$score(I) = w_I \cdot \#I \tag{1}$$

where w_I (weight) is the number of executions of the respective instruction sequence I and $\#I$ (length) is the number of the affected instructions.

Example 1. A loop I that is executed $w_I = 100$ times and contains $\#I = 8$ instructions results in $score(I) = 800$.

To find the most promising sequence in a tree, the analysis module traverses it using a recursive depth-first search. Taking child nodes into account, for each node it is checked whether it is added to the current best sequence: If there is no branch instruction and the new score calculated with Eq. 1 is greater, the node extends this sequence, otherwise the path extension is stopped here. This way, all possible sequences for a tree are evaluated in order to finally return the most promising sequence with the highest score for each tree.

Considering compiler optimizations, [6] has shown for embedded applications from the de facto standard Embench$^{\text{TM}}$ suite[5] that promising sequences can be identified covering on average about 31 % of the total execution. In the best case, even 55 % of executed instructions for the application *crc32* can be covered, which is therefore particularly suitable for optimization by a HA. However, there are also applications such as *picojpeg* with numerous sequences that only cover around 15 % of the execution. Although it is conceivable to build several HAs, this also increases the hardware costs. If the highest scores additionally consist of a length of 1, these sequences are unusable for a HA design. Hence, the main goal of this work is to overcome these limitations in order to increase the coverage by a single HA design.

IV. Methodology

In this section, we describe the core development of our proposed approach, which is based on [6] and thus designed w. l. o. g. for the RISC-V ISA. First, the VP driven approach described in the last section is extended in Section IV-A with the ability to generate a variety of instruction sequences w. r. t. the extension set *RV32IMC*. Second, in Section IV-B, we present our algorithm to merge instruction sequences for increasing execution coverage and saving HAs.

A. Extending Instruction Sequences

In RVOPT-VP described in Section III, promising instruction sequences can be generated as optimization candidates for a HA. Specifically, one sequence is returned for each different tree, i. e. for each different instruction that is encountered during execution. While this approach guarantees that each sequence covers different execution parts, sequences cannot share a prefix. To offer more opportunities for merges than this default mode, a method is needed that can find additional sequences. The analysis module is therefore extended to allow two new optimization modes: *Subsequence* and *Variant*.

[3]https://github.com/agra-uni-bremen/riscv-vp

[4]https://github.com/agra-uni-bremen/opt-vp

[5]https://embench.org

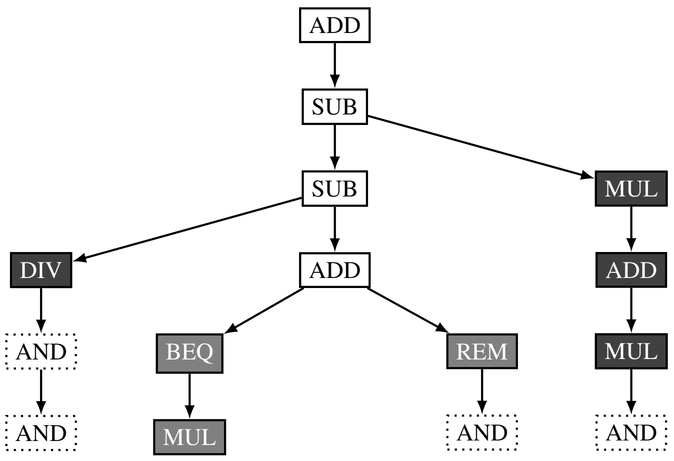


Fig. 1: □ Default, ▨ Subsequence, and ■ Variant

Some instruction nodes are not part of promising optimization candidates that have been generated based on RVOPT-VP's trees due to an insufficient score calculated by Eq. 1. To generate more sequences for improving merge opportunities, the subsequence mode considers the optimization candidates as subsequences and extends their paths at each child of the last node. Since the children can consist completely of branch instructions, this type of instruction is also analyzed if it is always or never executed given the application input.

Example 2. Fig. 1 shows a tree generated by RVOPT-VP with the optimization candidate (`ADD`, `SUB`, `SUB`, `ADD`), where `ADD`, `SUB`, `BEQ`, `MUL`, `REM`, and `AND` are instructions supported by the RV32IMC extension set. Using the subsequence mode, this sequence can be extended by (`BEQ`, `MUL`) and `REM` resulting in a total of 3 sequences. `AND` is missing assuming that the score for this instruction is insufficient.

In addition to stop points set by the analysis module at the end of promising instruction sequences, there are also stop points at branches in the tree generated by RVOPT-VP's tracing module, which can also be extended. In order to increase the sequence variety for such structures, analogous to the generation of subsequences introduced above, the analysis module iterates over the children of each branching node contained in the respective optimization candidates in the different execution trees. The corresponding instruction path is then extended using Eq. 1 to find the next best variant. Depending on a predetermined parameter b, it is thus possible to identify the best b optimization variants with the highest overall execution percentage.

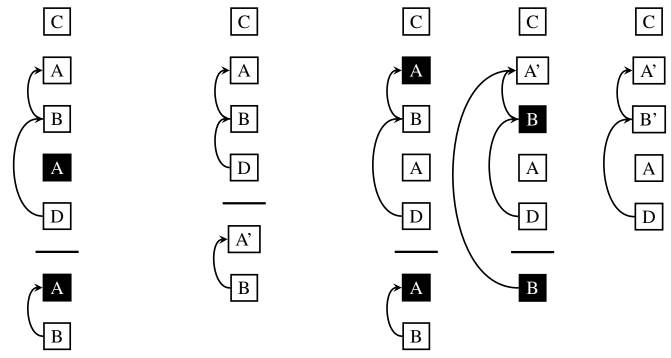
Example 3. Reconsider the promising optimization sequence (`ADD`, `SUB`, `SUB`, `ADD`) from Fig. 1 that is generated by the RVOPT-VP's analysis module. Using the variant mode, the `SUB` instructions are considered as branching nodes and therefore analyzed by iterating over the paths of their children. Let $b = 2$. Then the variants (`ADD`, `SUB`, `MUL`, `ADD`, `MUL`) and (`ADD`, `SUB`, `SUB`, `DIV`) result. The `ADD` instruction is not included as the score is insufficient in this case.

Algorithm 1: Proposed merging of instruction sequences

```
Input: Instruction sequences $S$
Output: Merged sequence
1  $B \leftarrow mapsort(S)$                                        ▷ Base sequences
2  $M \leftarrow$ Concatenated sequences from $B$
3  $s_{new} \leftarrow 0$                                            ▷ Current score
4  $k \leftarrow 0$
5  while $k < \#B - 1$ do
6      $s_{old} \leftarrow s_{new}$
7      for all $v \in reverse(M_{0,\dots,k})$ do                   ▷ BU
8          if $v$ has any outgoing dependency then
9              for all $u \in reverse(M_{k+1})$ do
10                 if $u = v \land$ no dependency conflict then
11                     $merge(u, v)$
12                 end if
13             end for
14         end if
15     end for
16     for all $u \in M_{k+1}$ do                                  ▷ TD
17         if $u$ has any incoming dependency then
18             for all $v \in M_{0,\dots,k}$ do
19                 if $v = u \land$ no dependency conflict then
20                     $merge(v, u)$
21                 end if
22             end for
23         end if
24     end for
25     for all $v \in reverse(M_{0,\dots,k})$ do                   ▷ R-BU
26         for all $u \in reverse(M_{k+1})$ do
27             if $u = v \land$ no dependency conflict then
28                 $merge(u, v)$
29             end if
30         end for
31     end for
32     for all $u \in M_{k+1}$ do                                  ▷ R-TD
33         for all $v \in M_{0,\dots,k}$ do
34             if $v = u \land$ no dependency conflict then
35                 $merge(v, u)$
36             end if
37         end for
38     end for
39     $s_{new} \leftarrow score'(B, M, k + 2)$
40     if $s_{new} \leq s_{old}$ then
41         break
42     end if
43     $k \leftarrow k + 1$
44 end while
45 return $M_{0,\dots,k}$
```

B. *Merging Instruction Sequences*

Although the original RVOPT-VP generates instruction sequences from which a candidate for HA optimization can be selected, this is highly dependent on the application. If optimization candidates only cover a small execution fraction, it may not be worthwhile to build HAs for them, especially if the worst case occurs where only one instruction per tree is affected. Moreover, designing different HAs also increase hardware costs. Thus, a novel merging approach called *RVOPT-SEQ* is proposed in Algorithm 1 that combines the sequences created by the analysis module (Section IV-A) to increase the execution coverage and save hardware costs, and to allow optimization by a single HA in worst-case scenarios.



(a) Without dependencies (b) With dependencies

Fig. 2: Merging of instruction sequences

RVOPT-SEQ starts by sorting input sequences S generated by the analysis module to determine the base sequences B (Line 1). To this end, the sequence with the highest score (Eq. 1) is selected as the start sequence B_0. The next base sequence is a sequence from S that has not yet been selected and matches most of the instructions of the current base sequences without considering data dependencies. This metric is natural because B_0 has the highest execution percentage and therefore the chance is greatest that many merges are possible due to similar sequences $B_1, \dots, B_{\#S-1} \in B$, especially by using the new modes *Subsequence* and *Variant*. The computed list of base sequences is then concatenated to create the vector M (Line 2). The main iteration depends on the number of base sequences $\#B$ to combine sequence pairs where the first element $M_{0,\dots,k}$ is the current sequence already merged and the second element M_{k+1} is the sequence to be merged (Lines 3–6). To map instructions between $M_{0,\dots,k}$ and M_{k+1}, there are 4 subroutines: 1) *Bottom-Up* (BU) in Lines 7–15, 2) *Top-Down* (TD) in Lines 16–24, 3) *Rest-BU* (R-BU) in Lines 25–31, and 4) *Rest-TD* (R-TD) in Lines 32–38. BU iterates backwards through $M_{0,\dots,k}$ and M_{k+1}, and merges each instruction node containing an outgoing dependency from $M_{0,\dots,k}$ with a matching node from M_{k+1} if there is no dependency conflict. TD works vice versa, where nodes from M_{k+1} with incoming dependencies are observed. While 1) and 2) merge nodes w. r. t. dependencies, 3) and 4) select "remaining" source nodes without any dependencies. If this order is not followed, finding the optimal solution is not guaranteed as nodes without dependencies can block the optimization after some of these subroutines have been performed.

Example 4. Let $M_0 = (\mathtt{C}, \mathtt{A}, \mathtt{B}, \mathtt{A}, \mathtt{D})$ and $M_1 = (\mathtt{A}, \mathtt{B})$ be sequences, where A, B, C, and D are instructions. Fig. 2 visualizes the merging of M_0 (top) and M_1 (bottom) separated by a horizontal line, without and with respecting data dependencies (Fig. 2a and Fig. 2b) represented by arrows. If no dependencies are respected, $\mathtt{A}_{M_0}$ can be mapped to $\mathtt{A}_{M_1}$ (highlighted in black) leading to $\mathtt{A}'$, but afterwards no further merges are possible. Taking dependencies into account, $\mathtt{A}_{M_1}$ can first be mapped to $\mathtt{A}_{M_0}$, which then allows $\mathtt{B}_{M_1}$ to be merged with $\mathtt{B}_{M_0}$, resulting in the shortest possible sequence.

Since the main loop is repeated a maximum of $\#B-1$ times in which all instructions can be compared by the subroutines, this results in a worst-case complexity of $\mathcal{O}(n \cdot m^2)$, where n is the number of sequences and m is the number of instructions. The main loop can also be exited beforehand if the termination criterion $s_{new} \leq s_{old}$ is satisfied (Lines 39–44), which is calculated by

$$score'(B, M, k) = \sum_{i=0}^{k-1} score(B_i) - (\#M_i - \#B_i) \cdot w_{B_i} \cdot c \quad (2)$$

where c is the cost factor for *No Operations* (NOPs). If a HA is built for $M_{0,\dots,k}$, processed base sequences can still be executed. Since $M_{0,\dots,k}$ contains more instructions than each of the k merged sequences, all instructions without mapping are discarded and treated as NOPs for that particular base sequence. In this context, Eq. 2 ensures that the performance gain remains reasonable in relation to the increase in hardware complexity. Finally, the best HA for the returned merged sequence $M_{0,\dots,k}$ (Line 45) can be designed.

Example 5. Fig. 3 shows the computed base sequences $M_0 = (\mathtt{ADD}, \mathtt{SUB}, \mathtt{MUL}, \mathtt{DIV}, \mathtt{AND})$ with the highest score, $M_1 = (\mathtt{SUB}, \mathtt{MUL}, \mathtt{MUL}, \mathtt{DIV}, \mathtt{ADD}, \mathtt{DIV})$ with the most mappings to M_0, and $M_2 = (\mathtt{AND}, \mathtt{ADD}, \mathtt{ADD})$, consisting of instructions from RV32IMC analogous to those shown in Fig. 2. By using Algorithm 1, BU cannot be performed when $k = 0$: For example, $\mathtt{DIV}_{M_0}$ cannot be mapped to $\mathtt{DIV}_{M_1}$ because there is an incoming dependency from $\mathtt{AND}_{M_0}$ resulting in a conflict. However, the TD routine working in reverse can be applied as shown in Fig. 3a, whereby the incoming dependency to the highlighted $\mathtt{DIV}_{M_1}$ must be redirected to the merged $\mathtt{DIV}'$. Similar to this routine, R-BU (Fig. 3b) and R-TD (Fig. 3c) can be executed for remaining nodes. In contrast to the first iteration, the BU routine can be used w. r. t. M_2 for $k = 1$ (Fig. 3d). By reapplying R-BU, Fig. 3e finally shows the merged sequence $M_{0,1,2} = (\mathtt{SUB}', \mathtt{MUL}', \mathtt{DIV}', \mathtt{MUL}, \mathtt{DIV}, \mathtt{AND}', \mathtt{ADD}, \mathtt{ADD}'')$.

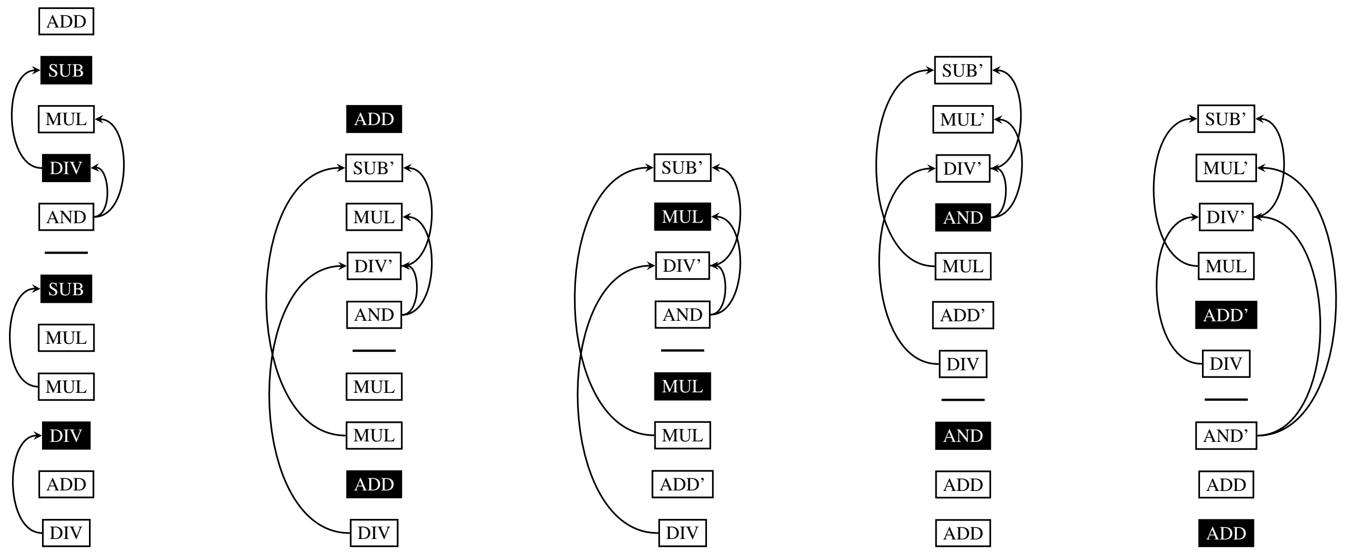
V. Experimental Results

This section summarizes the experiments conducted to empirically evaluate our developed merging algorithm for instruction sequences in order to design a single HA. While Section V-A describes the setup used for the evaluation, Section V-B presents the impact of our proposed approach compared to related work.

A. Experimental Setup

The new optimization modes for extending RV32IMC instruction sequences were implemented in the analysis module of RVOPT-VP[6], which is implemented in C++. For reasons of consistency, the novel merging approach *RVOPT-SEQ*[7] was also implemented in C++ for performance evaluation. For representative reasons, the same set of Embench™ applications[8] compiled with optimization level *O3* and inputs

[6]https://github.com/agra-uni-bremen/opt-vp
[7]https://github.com/agra-uni-bremen/opt-seq
[8]https://embench.org



(a) TD (b) R-BU (c) R-TD (d) BU (e) R-BU

Fig. 3: Merging of base sequences using Algorithm 1 in order to optimize the input for building a HA

as in [6] were used to generate their execution trees using the RVOPT-VP's tracing module. To analyze these trees, the default mode already available in RVOPT-VP, and the new subsequence and variant mode were used. In addition, the combination of these new modes was also used, which is called *Full* below. While using the default mode allows an ideal comparison with the results from [6], the other modes are used for determining to what extent the greater sequence variety increases the number of merges. The associated parameters $b = 3$ and $c = 0.1$ introduced in Section IV were determined experimentally. Specifically, the best 3 variants were analyzed for all experiments and a realistic NOP cost factor of 0.1 was assumed for the termination criterion (Eq. 2) of RVOPT-SEQ.

All evaluations were carried out on a Fedora 39 machine with an Intel Xeon E3-1270 v3 CPU with 3.5 GHz and 32 GB of main memory. For each considered application, 30 runs were performed and the average (AVG) was calculated.

B. Performance Evaluation

In order to evaluate the effectiveness of the proposed merging approach, the results achieved by RVOPT-SEQ were compared with those from [6]. Based on the original RVOPT-VP's *Instruction Sequence Score* (ISS) calculated using Eq. 1, it was measured in relation to the number of base sequences ($\#B$) to what extent these can improve the ISS through a number of merges ($\#M$) expressed as *Merged Sequence Score* (MSS). The experimental results for the embedded applications are shown in Table I and interpreted below.

Regardless of which optimization mode is activated, using RVOPT-SEQ significantly improves the ISS on average where all results are stable. Compared to RVOPT-VP, 8 mappings of instructions can be made based on 28 sequences. This almost doubles the ISS and thus also the coverage of the execution w. r. t. instructions, i. e. 8 HAs can be replaced. In the best case, it is even possible to increase the execution coverage for the application *qrduino* by a factor of around 5 using 11 merges. For *matmult-int* the coverage is hardly increased due to the fact that the coverage of around 40 % achieved in [6] is already comparatively high. A similar case is the benchmark *nsichneu* where the coverage cannot be significantly improved either. It should be noted that in [6] only the instruction `LW` is covered with a percentage of around 55 %, i. e. a HA cannot be used to accelerate the original best sequence. A total of 7 merges now make it possible to design a single HA for this application.

Extended sequences generated by the new modes make it possible for several applications to achieve an even higher number of merges due to additional base sequences compared to the default mode.[9] For e. g. *nettle-aes*, the MSS increases by a considerable factor: While the variant mode has a MSS that is about 7 times higher than ISS, the full mode has a MSS that is nearly 8 times higher. However, the combination of subsequence and variant mode is not worthwhile for every benchmark. For *nettle-sha256*, the coverage increases the most when using only the subsequence mode. This is because some variants are sorted into the front part of the base sequences, i. e. one less mapping is possible in full mode. For better clarity, the corresponding best score improvements are highlighted in bold for each application and visualized in Fig. 4.

In summary, the experimental results confirm that our approach meets the objectives of this work. Taking all optimization modes into account, the execution coverage is almost doubled by using RVOPT-SEQ and on average 10 HAs needed for the same coverage can be replaced by one single HA with a negligible performance loss.

[9]The number of base sequences in full mode does not necessarily correspond to the sum of those analyzed with the subsequence and variant mode as sequences were discarded for merging if they are completely contained in other sequences.

TABLE I: Experimental comparison between optimization modes for merging instruction sequences

Application	Optimization mode															
	Default				Subsequence				Variant				Full			
	MSS	ISS	$\#M$	$\#B$	MSS	ISS	$\#M$	$\#B$	MSS	ISS	$\#M$	$\#B$	MSS	ISS	$\#M$	$\#B$
aha-mont64	**2,969,930**	1,786,752	9	29	2,078,290	1,786,752	9	66	2,275,320	1,786,752	10	70	2,275,320	1,786,752	10	107
crc32	**4,027,420**	2,101,248	6	26	3,656,100	3,655,680	12	72	2,101,340	2,101,248	8	48	3,656,130	3,655,680	13	94
edn	1,970,450	1,509,200	11	29	**3,980,830**	2,956,800	16	89	2,306,440	1,509,200	17	72	**3,980,830**	2,956,800	16	132
huffbench	**1,754,030**	661,439	10	26	719,411	703,200	8	121	960,971	661,439	10	55	719,411	703,200	8	150
matmult-int	2,183,670	2,143,200	4	24	**3,839,160**	3,732,740	7	69	2,176,220	2,143,200	6	50	3,826,040	3,732,740	8	95
md5sum	1,145,350	958,464	3	34	2,805,400	1,078,272	7	104	**4,473,090**	958,464	12	96	2,805,400	1,078,272	7	166
minver	**954,229**	500,570	6	35	553,937	500,570	5	205	910,309	500,570	7	74	553,937	500,570	5	244
nettle-aes	4,309,130	1,018,784	7	27	5,492,240	1,018,784	9	84	6,833,180	1,018,784	9	70	**8,016,290**	1,018,784	11	127
nettle-sha256	952,937	670,684	11	26	**1,161,800**	670,684	14	136	853,372	670,684	7	61	1,138,000	670,684	13	171
nsichneu	1,227,180	1,227,108	7	22	1,227,270	1,227,108	11	91	**1,286,260**	1,227,108	4	44	1,227,270	1,227,108	12	113
picojpeg	949,127	632,636	7	35	2,339,880	814,212	13	199	1,376,820	632,636	8	95	**2,494,210**	814,212	14	259
primecount	**3,047,680**	1,399,464	7	23	1,881,740	1,881,432	9	83	1,400,400	1,399,464	8	46	1,881,920	1,881,432	9	106
qrduino	**3,152,640**	619,974	11	33	628,656	619,974	16	219	775,144	619,974	10	80	628,656	619,974	16	266
sglib-combined	**916,351**	489,435	10	31	492,747	489,435	8	185	798,675	489,435	9	51	492,201	489,435	5	205
slre	**1,565,790**	749,049	8	25	836,406	749,049	11	111	1,269,750	749,049	7	56	976,266	749,049	13	142
tarfind	**2,154,990**	443,520	9	26	923,201	628,320	8	72	602,807	443,520	9	50	652,126	628,320	10	96
ud	479,456	298,888	9	24	1,075,360	298,888	12	86	742,610	298,888	11	54	**1,223,260**	298,888	14	116
AVG	**1,985,904**	1,012,377	8	28	1,981,908	1,341,876	10	117	1,831,924	1,012,377	9	63	2,149,839	1,341,876	11	152

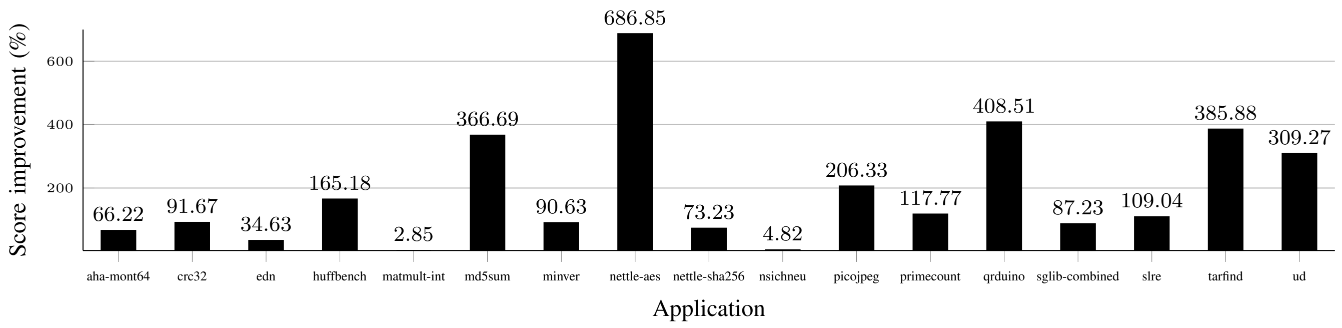


Fig. 4: Improvement of scores in terms of merging instruction sequences using the optimization modes

VI. Conclusion

This paper focused on merging VP-generated instruction sequences to increase their execution coverage for a single HA design. Experiments demonstrated that our approach extends sequences so that the total coverage almost doubles and an average of 10 HAs needed for the same coverage can be replaced by one HA with a negligible performance loss.

Future work will be directed towards further investigating our merging method. Even though its effectiveness was demonstrated by saving HAs, it can be used universally, e. g. for pipeline optimization. It is thus generally planned to design coarse-grained reconfigurable architectures based on the existing high-level results in order to accurately measure possible hardware acceleration for the covered sequences.

References

[1] L. De Micco, F. Vargas, and P. Fierens, "A literature review on embedded systems," *Latin America Transactions*, vol. 18, no. 2, pp. 188–205, 2020.

[2] H. Cherupalli, H. Duwe, W. Ye, R. Kumar, and J. Sartori, "Bespoke processors for applications with ultra-low area and power constraints," in *44th ISCA*. IEEE, 2017, pp. 41–54.

[3] M. Rieger, "Application specific integrated circuits (ASICs)," in *The Electronic Design Automation Handbook*, D. Jansen, Ed. New York: Springer, 2010, pp. 384–397.

[4] A. Shetty, "ASIC design flow and methodology – an overview," *International Journal of Electrical and Electronics Engineering*, vol. 6, no. 1, pp. 1–5, 2019.

[5] D. Shapiro, M. Montcalm, and M. Bolic, "Parallel instruction set extension identification," in *26th Convention of Electrical and Electronics Engineers*. IEEE, 2010, pp. 535–539.

[6] J. Zielasko and R. Drechsler, "Virtual prototype driven application specific hardware optimization," in *Forum on Specification & Design Languages (FDL)*. IEEE, 2023, pp. 1–8.

[7] M. Funck, V. Herdt, and R. Drechsler, "Virtual prototype driven design, implementation and evaluation of RISC-V instruction set extensions," in *25th DDECS*. IEEE, 2022, pp. 14–19.

[8] A. Waterman, Y. Lee, R. Avizienis, H. Cook, D. Patterson, and K. Asanovic, "The RISC-V instruction set," in *HCS*. IEEE, 2013.

[9] Y. Gao, W. Qian, and E. Cui, "RISC-V ISA extension toolchain supports: A survey," in *Proceedings of the 4th International Conference on Computing, Networks and Internet of Things*. ACM, 2023, pp. 924–929.

[10] V. Herdt, D. Große, and R. Drechsler, *Enhanced Virtual Prototyping*. Cham: Springer, 2020.

[11] V. Herdt, D. Große, P. Pieper, and R. Drechsler, "RISC-V based virtual prototype," *Journal of Systems Architecture*, vol. 109, 2020.

[12] F. Ghenassia, *Transaction-Level Modeling with SystemC*. New York: Springer, 2010.