Improving Virtual Prototype Driven Hardware Optimization by Merging Instruction Sequences

Jan Zielasko¹,², Rune Krauss¹, Marcel Merten¹, Rolf Drechsler¹,²
¹Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
²Cyber-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 VP¹ and our proposed approach² are provided as open-source software.

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

\[ \text{score}(I) = w_I \cdot \#I \]

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 1 that is executed \( w_1 = 100 \) times and contains \( \#I = 8 \) instructions results in \( \text{score}(I) = 800 \).

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™ suite 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 picopjpeg 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.

1https://github.com/agra-uni-bremen/riscv-vp
2https://github.com/agra-uni-bremen/opt-vp
3https://github.com/agra-uni-bremen/embench

1https://github.com/agra-uni-bremen/riscv-vp
2https://github.com/agra-uni-bremen/opt-vp
3https://github.com/agra-uni-bremen/embench
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.

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 increases 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.
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 \( B_1, \ldots, B_{\#S-1} \in B \), 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 \( M_{0\ldots,k} \) 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 \( M_{0\ldots,k} \) 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 \( M_{0\ldots,k} \) and \( M_{k+1} \), and merges each instruction node containing an outgoing dependency from \( M_{0\ldots,k} \) 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 \( M_0 = (C,A,B,A,D) \) and \( M_1 = (A,B) \) 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, \( A_{M_1} \) can be mapped to \( A_{M_0} \) (highlighted in black) leading to \( A' \), but afterwards no further merges are possible. Taking dependencies into account, \( A_{M_1} \) can first be mapped to \( A_{M_0} \), which then allows \( B_{M_1} \) to be merged with \( B_{M_0} \), 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 \( s_{\text{new}} \leq s_{\text{old}} \) is satisfied (Lines 39–44), which is calculated by

\[
\text{score}'(B, M, k) = \sum_{i=0}^{k-1} \text{score}(B_i) - (\#M_i - \#B_i) \cdot w_B \cdot c
\]

where \( c \) is the cost factor for No Operations (NOPs). If a HA is built for \( M_{0\ldots,k} \), processed base sequences can still be executed. Since \( M_{0\ldots,k} \) 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 \( M_{0\ldots,k} \) (Line 45) can be designed.

**Example 5.** Fig. 3 shows the computed base sequences \( M_0 = (\text{ADD, SUB, MUL, DIV, AND}) \) with the highest score, \( M_1 = (\text{SUB, MUL, MUL, DIV, ADD, DIV}) \) with the most mappings to \( M_0 \), and \( M_2 = (\text{AND, ADD, ADD}) \), 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, \( \text{DIV}_M \) cannot be mapped to \( \text{DIV}_M \), because there is an incoming dependency from \( \text{AND}_M \) 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 \( \text{DIV}_M \) must be redirected to the merged \( \text{DIV}' \). 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 \( M_{0,1,2} = (\text{SUB}', \text{MUL}', \text{DIV}', \text{MUL}, \text{DIV}, \text{AND}', \text{ADD}, \text{ADD}') \).

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\(^8\) applications 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
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 `arduino` 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. 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

<table>
<thead>
<tr>
<th>Application</th>
<th>MSS</th>
<th>Default</th>
<th>Subsequence</th>
<th>Variant</th>
<th>Full</th>
</tr>
</thead>
<tbody>
<tr>
<td>aha-mont64</td>
<td>2,969,930</td>
<td>1,876,752</td>
<td>9</td>
<td>29</td>
<td>2,078,290</td>
</tr>
<tr>
<td>huffbench</td>
<td>1,754,030</td>
<td>661,439</td>
<td>10</td>
<td>26</td>
<td>219,411</td>
</tr>
<tr>
<td>md5sum</td>
<td>1,145,350</td>
<td>958,464</td>
<td>3</td>
<td>34</td>
<td>2,805,400</td>
</tr>
<tr>
<td>minver</td>
<td>954,229</td>
<td>500,570</td>
<td>6</td>
<td>35</td>
<td>553,937</td>
</tr>
<tr>
<td>nisseu</td>
<td>3,949,130</td>
<td>1,018,784</td>
<td>7</td>
<td>27</td>
<td>5,492,240</td>
</tr>
<tr>
<td>picojpeg</td>
<td>1,227,180</td>
<td>1,227,108</td>
<td>7</td>
<td>22</td>
<td>1,227,270</td>
</tr>
<tr>
<td>primecount</td>
<td>3,047,680</td>
<td>1,399,464</td>
<td>7</td>
<td>23</td>
<td>1,881,740</td>
</tr>
<tr>
<td>sglib-combined</td>
<td>916,351</td>
<td>489,435</td>
<td>10</td>
<td>31</td>
<td>492,747</td>
</tr>
<tr>
<td>site</td>
<td>1,565,790</td>
<td>749,049</td>
<td>8</td>
<td>25</td>
<td>836,406</td>
</tr>
<tr>
<td>tarfind</td>
<td>2,154,990</td>
<td>443,520</td>
<td>9</td>
<td>26</td>
<td>923,201</td>
</tr>
<tr>
<td>ud</td>
<td>479,456</td>
<td>298,888</td>
<td>9</td>
<td>24</td>
<td>1,075,360</td>
</tr>
</tbody>
</table>

**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](image)

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


