Superword Level Parallelism aware Word Length Optimization

Ali Hassan El Moussawi, Steven Derrien

Superword Level Parallelism aware Word Length Optimization

Ali Hassan El Moussawi
Université de Rennes 1
Rennes, FRANCE
Email: aelmouss@irisa.fr

Steven Derrien
Université de Rennes 1
Rennes, FRANCE
Email: sderrien@irisa.fr

***Abstract*—Many embedded processors do not support floating-point arithmetic in order to comply with strict cost and power consumption constraints. But, they generally provide support for SIMD as a mean to improve performance for little cost overhead. Achieving good performance when targeting such processors requires the use of fixed-point arithmetic and efficient exploitation of SIMD data-path. To reduce time-to-market, automatic SIMDization – such as superword level parallelism (SLP) extraction – and float-to-fixed-point conversion methodologies have been proposed. In this paper we show that applying these transformations independently is not efficient. We propose a SLP-aware word length optimization algorithm to jointly perform float-to-fixed-point conversion and SLP extraction. We implement the proposed approach in a source-to-source compiler framework and evaluate it on several embedded processors. Experimental results illustrate the validity of our approach.**

I. Introduction

Even though, many embedded processors (such as ARM cortex-A), provide hardware support for floating-point arithmetic, a good number of ultra low power embedded processors (such as ARM cortex-M0/1/3, TI TMS320C64x, Recore XENTIUM) do not, in order to reduce die area and/or minimize power consumption. This comes for the cost of restraining programmability to the use of fixed-point arithmetic. However, application prototyping in many domains, such as signal processing and telecommunication applications, employs floating-point representation. Although floating-point can be soft-emulated on such processors, it drastically degrades performance. Instead, fixed-point implementations are preferably used. Hence, float-to-fixed-point conversion is a crucial step for an efficient implementation when targeting such processors. Though, keeping full fixed-point operations precision requires increasingly large word-lengths (WLs) which, unless supported by the target processor, would require software emulation. Rather, quantizations are applied to limit WLs growth and consequently improving performance, for the cost of degrading the application's quality by introducing errors. The conversion process must account for these errors and carefully select WLs that keep the computation's accuracy within an "acceptable" limit while maximizing performance.

This Performance/Accuracy trade-off has been identified and exploited when targeting customizable architectures, in the context of High-level synthesis [1] for instance, where the designer has more flexibility in customizing WLs supported by the architecture. This is not the case when targeting processors with predefined data-path. However, most embedded processors nowadays provide support for small-scale SIMD. This allows the use of smaller data WLs than the word size (sub-word) and allows the application of an operation on several such (packed) data simultaneously, to ultimately improve performance. Intuitively, using narrower WLs in this context, should translate to better performance on one side – due to increased vectorization factor – but lower quality on the other side. Previous work [2] follows this intuition when applying float-to-fixed-point conversion. They aim at minimizing WLs without taking into account SLP, which can be applied, independently, later on.

However, this intuition is unrealistically optimistic since, selecting narrower WLs during word-length optimization (WLO) does not necessarily result in performance improvement after applying SLP extraction mainly because WLO is unaware of SLP grouping possibilities and the associated overhead.

In this work, we address this problem by jointly considering SLP extraction and WLO. More specifically:

- We propose a new SLP-aware WLO algorithm. To the best of our knowledge this is the first work to jointly consider both, WLO and SLP extraction.
- We implement it as an automated source-to-source compilation flow.
- We test our approach on several embedded processors against some signal processing applications.

The paper is organized as follows. In section II, we present some contextual background and related work. Then we present our SLP-aware WLO algorithm and the corresponding source-to-source compilation flow in sections III and IV. Finally we present some experimental results in section V.

II. Background and Related Work

A. Superword Level Parallelism

Taking advantage of SIMD capabilities of a processor to improve performance is not an easy task. Fortunately, many approaches to automatically exploit SIMD have been proposed. Among which, superword level parallelism (SLP) which unlike conventional loop vectorization techniques, exploits SIMD opportunities at the basic block level. The aim is to find groups of scalar operations that can be combined together into SIMD groups. An SIMD group is a set of independent and isomorphic operations (performing the same operation type (add, sub, ...)) of the same size that can be implemented using SIMD instructions. The main limiting factor of SLP performance is the overhead associated with data packing/unpacking operations in case the data is not properly organized in memory. Mainly if the data to be loaded from (stored to) memory are not contiguous and properly aligned. If not handled carefully, the overhead of these operations may easily overcome the benefit of SLP, resulting in performance degradation. That's why all SLP extraction algorithms focus on minimizing packing/unpacking operations when selecting SIMD groups.

Larsen and Amarasinghe introduced SLP and proposed an extraction algorithm [3] based on a local heuristic. Many enhancements [4], [5], [6] have since been proposed.

In 2012, Liu et al. proposed an alternative SLP extraction algorithm [7] based on a estimation of the global superwords reuse. In a given a basic block, it first constructs the set of all possible SIMD group candidates of size 2, among them, it aims at selecting the "best" conflict-free sub-set. Two SIMD group candidates are in conflict if they contain a common operation or they have a cyclic dependency. The groups are selected iteratively based on an estimation of their contribution to the overall superword reuse over packing/unpacking cost ratio. Whenever a group is selected, all remaining candidates in conflict with it are eliminated, until all conflicts are resolved. In order to extend the groups size beyond 2, the group selection is repeated, after replacing the selected groups in the original basic block, as long as groups size is supported. In this work we use a similar SLP extraction algorithm.

B. Floating-to-fixed-point Conversion

Floating-to-fixed-point conversion generally consists of three main parts: (i) Integer word-length (IWL) determination which is specified based on the data values range. The later can be obtained either by gathering simulation statistics [8] or using analytical methods, such as interval arithmetic. The IWL determination is generally used to specify the binary-point position and the required scaling operations (which are needed to align fixed-point formats and are implemented using shift operations). (ii) Word-length (WL) determination, which together with IWL determine the fixed-point specification. The fractional word-length (FWL) is implicitly obtained from WL and IWL. Hence, WL determination directly affects the computation's accuracy, which can be evaluated by simulation [9] or using analytical methods [10], [11], [12]. WLO algorithms [16] are used to perform WL determination and generally trade accuracy for performance. (iii) Fixed-point code generation which converts the floating-point code into fixed-point code that implements the fixed-point specification.

Many floating-to-fixed-point conversion methodologies targeting embedded processors have been proposed. Cilio and Corporaal [13] and Kum et al. [8] presented similar methods allowing the conversion of floating-point C code into fixed-point C code targeting embedded processors with different methods for determining IWLs. However, both approaches consider only the target's native data WL and consequently no WLO is performed.

Menard et al.[2] proposed a float-to-fixed-point conversion methodology targeting embedded processors and taking into account their SIMD capabilities. Indeed, they consider the different WLs (due to SIMD) supported by the target processor and they perform a WLO that aims at selecting for each operation, the instruction (i.e. the WL) that minimizes the overall execution time, subject to an accuracy constraint. The relative execution time associated to an instruction is directly related to the WL of data on which it can operate. For example an execution time of 1 and 0.5, respectively, is associated to 32-bit scalar instruction and 2x16-bit SIMD instruction performing the same operation type. While this approach exposes potential SLP opportunities, by favoring the selection of smaller WLs, it still considers SLP extraction as a completely independent transformation that can be applied afterward. Consequently, during WLO it assumes that, when an SIMD instruction is selected to implement a (scalar) operation, N (depending on the SIMD vector size) such operations will ultimately be executed in parallel by the same instruction – if SLP extraction is later applied – without actually knowing whether or not this is possible. Furthermore, this also ignores the potential overhead associated with data packing/unpacking operations. These assumptions are very optimistic and unrealistic.

To the best of our knowledge, no other work dealing with WLO and SIMD has been proposed.

III. Joint WLO and SLP extraction

Applying WLO without taking into account SLP extraction constraints will most likely yield inefficient solutions. That is because WLO decisions directly dictate the search space of SLP extraction. It may prevent, otherwise possible, SIMD grouping candidates by attributing different WLs for operations that can be, otherwise, grouped together (all operations in a SIMD group must have the same WL).

It is very important to note that WLO is performed under accuracy constraint. In another word, only a limited accuracy-degradation *budget* can be used by WLO to try to improve performance as much as possible. In this context, the impact on performance greatly depends on how well SIMD capabilities are being exploited. It is not wise for the WLO to spend the accuracy-degradation budget on optimizing operations that cannot be efficiently exploited by SLP to improve performance. If WLO is unaware of SIMD grouping possibilities and their associated overhead, it may blindly optimize WLs of operations that cannot form a SIMD group (either due to dependencies or conflicts with other groups) or have a high packing/unpacking overhead.

Besides, the scaling operations can have a major impact on performance, so they should be taken into account as well.

In order to address these problems, we propose an approach that jointly performs WLO and SLP extraction. We couple an accuracy-aware SLP extraction algorithm with a SLP-aware WLO algorithm. In the remainder of this section we present these algorithms.

A. SLP-aware WLO algorithm

The fixed-point specification, representing the set of fixed-point formats of each data and operation in the system, is taken as input, along with the accuracy constraint. The IWLs of all data and operations (called nodes from now on) in the fixed-point specification are pre-determined based on values range evaluation using interval arithmetic (any alternative method can be used instead). The accuracy constraint, specified by the user, represents the maximum allowed noise power of the quantization error at the system's output.

The pseudo code of the SLP-aware WLO algorithm is listed in figure 1a. We start by initializing the WL of all nodes to the maximum WL, say M, supported by the target processor for the corresponding operation (lines 1-3). This generally corresponds to the case where minimum SLP is available, but on the other hand, it represents the most accurate fixed-point specification (that can be obtained using natively supported WLs only).

Next, we process each basic block to be considered for SLP extraction, starting by the higher priority ones. Priority here depends on the contribution of the basic block to the overall execution time. This is to ensure that the accuracy-degradation budget is wisely spent on optimizing most performance impacting basic blocks first. In this work we consider that the selection and sorting of basic blocks to be considered for SLP extraction is performed earlier. It can be done based on profiling analysis for instance.

For each basic block we iteratively apply the accuracy-aware SLP extraction (presented in the next subsection), which yields the set of selected SIMD groups, taking into account

```
Input: BBs: sorted list (by priority) of basic blocks for SLP extraction.
    SPEC: fixed-point specification with IWLs pre-determined.
    A: accuracy constraint
Output: G, set of selected SIMD groups. Determine WLs in spec.
 1: for node in SPEC do
 2:     Set node to maximum WL supported by the target processor
 3: end for
 4: for b in BBs do                                  ▷ visit in priority order
 5:     G ← ∅
 6:     while not done do
 7:         Selected ← SLP(b, spec, A)
 8:         if Selected = ∅ then
 9:             done
10:         end if
11:         Update b and prepare it for next iteration
12:         G ← G \ {e1, e2 : {e1, e2} ∈ Selected}
13:         G ← G ∪ Selected
14:     end while
15:     ScalOptim(G, SPEC, A)
16: end for
```

(a) SLP-aware WLO algorithm

```
 1: procedure ScalOptim(G, SPEC, A)
 2:     for each superword reuse (g1, g2): g1, g2 ∈ G do
 3:         S ← g1.elements.FWL − g2.elements.FWL      ▷ list of
    required scaling amounts
 4:         if all amounts in S are equal then
 5:             skip
 6:         else if all amounts are positive (i.e. right shifts are required) then
 7:             m ← max(S)
 8:             SPEC.save g1
 9:             for e ∈ g1 do
10:                 reduce FWL of e by (m − S[e])
11:             end for
12:             if EvalAcc(SPEC) violates A then
13:                 SPEC.revert g1
14:             end if
15:         end if
16:     end for
17: end procedure
```

(b) SLP-aware Scalings optimization

```
 1:     ▷ EvalAcc(SPEC): evaluates the accuracy of the current fixed-point
    specification
 2: ▷ SetMaxWl(c, SPEC): set all elements of c to maximum possible WL
    as defined by equation 1
 3: procedure SLP(b, SPEC, A)
 4:                                                  ▷ Candidates Extraction
 5:     C ← extract SIMD group candidates from b
 6:     for c in C do
 7:         SetMaxWl(c, SPEC)
 8:         if EvalAcc(SPEC) violates A then
 9:             C ← C \ {c}
10:         end if
11:         revert WL of c
12:     end for
13:     Conf ← ∅
14:     for {ci, cj} in C do                        ▷ Conflicts Detection
15:         if ci and cj have a common operation or a cyclic dependency then
16:             Conf ← Conf ∪ {{ci, cj}}
17:         else
18:             SetMaxWl(ci, SPEC)
19:             SetMaxWl(cj, SPEC)
20:             if EvalAcc(SPEC) violates A then
21:                 Conf ← Conf ∪ {{ci, cj}}
22:             end if
23:             revert WL of ci, cj
24:         end if
25:     end for
26:     Selected ← ∅
27:     while Conf ≠ ∅ do                           ▷ SIMD Groups Selection
28:         for c in C do
29:             estimate global benefit associated with selecting candidate c
30:         end for
31:         g ← select most beneficial candidate in C
32:         Selected ← Selected ∪ {g}
33:         C ← C \ {{g} ∪ {x : {g, x} ∈ Conf}}
34:         SetMaxWl(g, SPEC)
35:     end while
36:     return Selected
37: end procedure
```

(c) Accuracy-aware SLP extraction

Fig. 1: SLP-aware WLO pseudo code.

data packing/unpacking cost. Each time a new SIMD group is selected, the WL of all its elements is reduced to the maximum supported WL, say m, such that:

$$m * Nelem <= SIMD\ size \tag{1}$$

Where $Nelem$ is the number of elements in the group. This is performed by procedure SetMaxWL in algorithm 1c. For example, consider a processor supporting 2x16-bit and 4x8-bit SIMD in addition to 32-bit scalar instructions. In this case $M = 32$. Now when a group of two operations is selected ($Nelem = 2$) during SLP extraction, the WL of each one of them is reduced to $m = 16$ bits, since 16 is the maximum supported WL in this case such as $16 * 2 <= 32$.

The elements of each selected group are then replaced by a new operation (representing the group) in the basic block (line 11) in order to prepare for the next SLP extraction iteration which allows the extension of the groups size when possible, otherwise the processing of the basic block is completed and we move to the next one (lines 8-10).

The global set of selected groups (G) is updated (lines 12-13) and is later used to optimize scaling operations (line 15). The scaling optimization algorithm is presented later on.

As an output we obtain a complete fixed-point specification for the system, along with the set of selected SIMD groups. These information are later used to generate fixed-point and SIMD C code.

B. Accuracy-aware SLP extraction

The pseudo code of the accuracy-aware SLP extraction algorithm is listed in figure 1c.

First we construct the set of SIMD group candidates (C) in the specified basic block and we identify the conflicts ($Conf$) between them. A group candidate is a pair of isomorphic and independent operations. Two groups are in conflict if they contain the same operation or if they have a cyclic dependency. $(C, Conf)$ represents the solution space of SLP.

This is similar to the approach presented by Liu et al.[7]. However, in our accuracy constrained context, some candidates in C may not be valid. This is the case of a candidate c that when selected (i.e. the WL of all its elements is set to m as specified by equation 1) while all other nodes are set to maximum WL (M), the accuracy constraint is violated. In this case, c can never be implemented as SIMD instruction without violating the accuracy constraint. Therefore, we eliminate all invalid candidates in C (lines 6-12).

Furthermore, $Conf$ does not represent all the conflicts in our context. In fact, two candidates can also be in conflict if, when both selected while all other nodes are set to maximum WLs (M), the accuracy constraint is violated. In this case both candidates cannot coexist without violating the accuracy constraint, hence they are in conflict (lines 16-21).

After the set of candidates and conflicts have been determined, the group selection process estimates the global benefit associated with the selection of each remaining candidate and then selects the most beneficial one, g. The benefit of a

candidate is the ratio of superwords reuse it enables, if it get selected, to the overall packing/unpacking cost [7]. The WL of g's elements is specified as described earlier using equation (1), and all candidates in conflict with g are eliminated (line 30-31). This iterative process continues until all conflicts have been resolved (lines 25-33).

C. Scalings Optimization

Another major impact on the performance of a fixed-point implementation is the cost of scaling operations. Their impact depends on how well the target processor supports shifting operations. For example, a barrel shifter can generally perform a shift operation of any amount in constant time. Whereas shift registers require a variable time depending on the shifting amount.

In the context of SLP an additional critical factor is the fact that scaling operations may break some superword (vector) reuse chains and hence require the introduction of packing/unpacking operations. Thus, severely impacting the performance, not only due to the scalings cost but also to the additionally introduced packing/unpacking overhead. This is, in fact, because most embedded processors only support SIMD shifting instructions by the same amount of all the vector elements. Therefore in case two elements of the same vector have to be shifted by different amounts, they need to be unpacked first, shifted independently and then repacked, before being able to be used by an SIMD instruction[1].

In this work, we address this problem by using an SLP-aware, accuracy-aware scaling optimization algorithm. The goal is to try to make all elements of an SIMD group operand (superword) scalable by the same amount, so that the scaling operations can be grouped together and implemented using an SIMD instruction without the need for packing/unpacking.

For each superword reuse, the list of scaling amounts corresponding to each one of the superword elements is determined, as a function of fractional word-lengths (FWLs) of the elements of the source and destination SIMD groups. For instance, in the example of figure 2, the output superword (s) of group the $\{+1, +2\}$ is reused by the group $\{+3, +4\}$. Assuming truncation is used as quantization mode, the elements of s should be right shifted by $f1 - f3$ and $f2 - f4$, respectively. Where fx is the FWL of the corresponding operand of operation x. When the scaling amount is zero, it indicates that no scaling is required. If it is positive a right shift is required. In case $f1-f3 \neq f2-f4$, the scaling amounts are different, thus the shifting operations cannot be grouped and therefore packing/unpacking operations are required as shown in figure 2. To avoid this case, we reduce the FWLs while keeping WLs intact (by increasing the corresponding IWLs) so that the scaling amounts of all elements in a group become equal. By reducing the FWLs, the accuracy of the fixed-point specification may degrade, therefore this optimization is performed as long as the accuracy constraint is not violated.

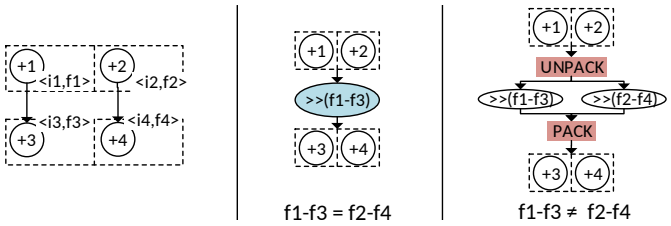

Fig. 2: Example of the impact of scaling operations on packing/unpacking. On the left, an extraction of a data flow graph with 4 operations and their fixed-point formats <IWL,FWL>; dashed lines represent SIMD groups. The center part illustrates the case where scaling operations can be grouped, the other case is illustrated on the right.

[1]In case masking operations are supported, packing/unpacking are not required but multiple SIMD instructions would still be needed to perform such operations. Though masking operations are rarely supported.

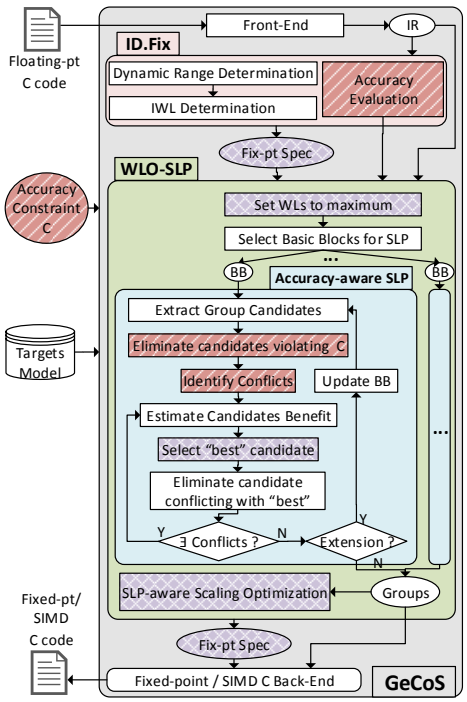

Fig. 3: SLP-aware WLO compiler flow diagram.

IV. Source-to-source compilation flow

The proposed joint WLO and SLP extraction algorithm is implemented as an automated source-to-source transformation in the compiler framework, GeCoS[14]. The flow diagram is depicted in figure 3. Starting from an annotated floating-point C code, we first construct an intermediate representation (IR).

For analyzing the data dynamic ranges and estimating the fixed-point specification accuracy, we use IDFix [15], a floating-to-fixed-point conversion framework integrated to GeCoS. IDFix analyzes the IR and creates the corresponding fixed-point specification, which associate a fixed-point format to each variable and operation (called nodes from now on) in the system. It then determines the dynamic ranges of all nodes, by propagating the values range intervals of the system's inputs, specified using pragma annotations. The IWL of each node is then specified by selecting the minimum IWL that covers the entire corresponding values range, in such way to avoid overflows.

Besides, the analytical expression of the system's output noise power is generated as a function of the fixed-point specification [11]. This is used during WLO as a metric to evaluate the fixed-point specification's accuracy and compare it against the user specified constraint. Note that any alternative way for evaluating the fixed-point specification's accuracy can be used instead since our proposed WLO is completely decoupled.

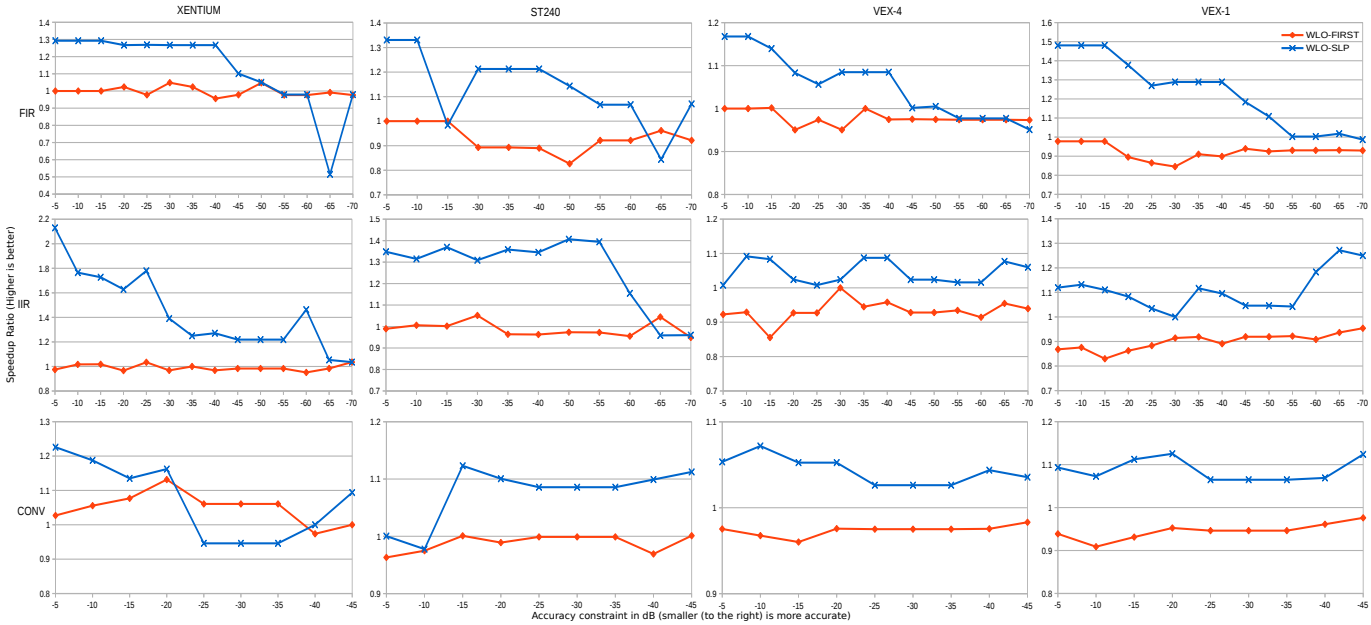

Fig. 4: Speedup comparison between SIMD version of `WLO-First` and `WLO-SLP` vs. accuracy constraint expressed in dB.

The proposed WLO and SLP extraction are used to obtain the fully specified fixed-point specification and the set of SIMD groups. These information are finally used by the back-end to convert the floating-point code into fixed-point using integer C types and explicit cast/scalings in order to match the fixed-point specification. Furthermore, it implements the SIMD groups using an abstract C macros API and generates the API's implementation for the specified target processor using its corresponding SIMD intrinsics.

V. Experimental Results

A. Test Setup

In order to evaluate the proposed SLP-aware WLO flow, we compared it against a compilation flow performing, first, float-to-fixed-point conversion using the same framework (ID.Fix[15]), except for WLO. We used the Tabu search algorithm presented by Nguyen et al.[16] with a cost relative to the operators WL similar to the approach used by Menard et al.[2]. For example, if the target processor provides the possibility to implement an addition using 32-bit or 16-bit word-lengths, then the corresponding cost for 32-bit is set as the double of that for 16-bit. Once the fixed-point specification is obtained, we apply SLP extraction using the same SLP extraction algorithm except that it is not accuracy aware since it is decoupled from the float-to-fixed-point conversion step. Finally, we used the same back-end to generate both fixed-point and SIMD C codes for the target processor. The overview of this baseline flow is represented in figure 5. From now on, we will refer to it as `WLO-First` as opposite to our approach `WLO-SLP`.

Contrary to our approach, `WLO-First` performs WLO independently from SLP i.e. without considering the actual SLP opportunities and the associated overhead due to data packing/unpacking.

Starting from the benchmark's single-precision floating-point C code, we apply `WLO-First` and `WLO-SLP`. The resulting fixed-point C code of `WLO-First` is used as baseline to compare the performance of the SIMD version of

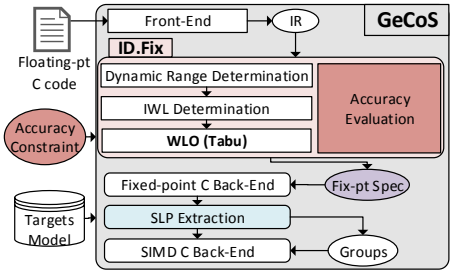

Fig. 5: `WLO-First` compiler flow diagram

`WLO-First` (which perform float-to-fixed-point conversion first then SLP extraction independently) against the one obtained by applying our joint float-to-fixed-point conversion and SLP extraction flow.

The generated C codes are then compiled (with -O3) and simulated on the target processor simulators. The number of cycles spent executing the benchmark is retrieved and used to compute the speedup of the SIMD versions for both `WLO-First` and `WLO-SLP`, over the baseline fixed-point version (of `WLO-First`).

$$speedup = \frac{number\ of\ cycles\ fixedpoint\ version}{number\ of\ cycles\ SIMD\ version} \quad (2)$$

B. Target Processors

We considered several embedded processors for this experimental setup.

XENTIUM [17] is an ultra low power 32-bit 12-issue wide Very Long Instruction Word (VLIW) core from Recore Systems. It has no support for floating-point arithmetic but provides support for 2x16-bit SIMD operations.

ST240 is a 32-bit 4-issue wide VLIW media processor from the ST200 family of ST Microelectronics. ST240 also supports 2x16-bit integer SIMD operations. Additionally, it supports single precision floating-point arithmetic.

Target	Constraint (dB)	-5	-15	-25	-35	-45	-55	-65
XENTIUM	WLO-First	723968	723968	727040	693248	741376	692224	358400
	WLO-SLP	560128	560128	560128	560128	658432	690176	690176
ST240	WLO-First	357726	264430	365005	365005	421730	353096	267861
	WLO-SLP	268823	268823	268823	268823	305011	305011	305011
VEX-4	WLO-First	676872	659464	628744	628744	662536	628744	628744
	WLO-SLP	579592	579592	579592	579592	645128	626696	626696

TABLE I: Number of cycles of SIMD versions for FIR.

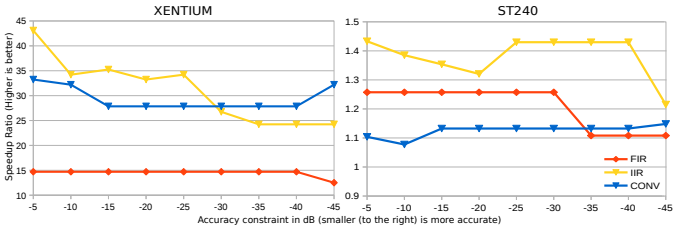

Fig. 6: Speedup of WLO-SLP over the original (single-precision) floating-point version vs. accuracy constraint.

VEX [18] is a parameterizable and extensible VLIW architecture model. We use it in two configurations; VEX-1 and VEX-4 with an issue width of 1 and 4 respectively. Since VEX does not provide support for SIMD, we implemented a 16-bit and 8-bit SIMD instruction extensions for supporting integer arithmetic, shift and data manipulation operations.

C. Benchmarks

A 64-tap FIR and a 10th order IIR filters as well as a 2d (3x3) image convolution (CONV) are used as benchmarks. The C code of each benchmark is annotated to specify the values range of the inputs. The innermost loop in FIR and IIR is partially unrolled by 4 to expose SLP, whereas the convolution kernel (3x3) is fully unrolled. The input samples are pre-normalized to $[-1, 1]$.

D. Results

The graphics of figure 4 plot the speedup of SIMD versions of WLO-First and WLO-SLP over the baseline scalar fixed-point version (of WLO-First), for each benchmark on each target processor, against the accuracy constraint. The later represents the maximum tolerable quantization noise power at the system's output, specified in dB.

The overall results clearly show the advantage of our approach across all benchmarks on all processors.

For FIR, we can see our approach trading accuracy for performance improvement. It manages to efficiently exploit SIMD to achieve performance improvement (up to 1.5 on VEX-1), whereas WLO-First mostly result in performance degradation after applying SLP extraction, illustrating the fact that WLO is blindly optimizing without considering SLP. The few points where our approach yields performance degradation, on XENTIUM and ST240 at -65 db, are due to a reduction in the execution time of the baseline fixed-point version, where the Tabu WLO algorithm manages to find a better fixed-point solution, also at this high accuracy constraint not many SLP opportunities are available.

Table I reports the number of cycles of the SIMD versions for FIR. The number of cycles for WLO-SLP increases, as expected, when the accuracy constraint becomes more strict (smaller values), whereas the number of cycles for WLO-First varies randomly.

Besides, we can notice the impact of instruction level parallelism (ILP) by comparing the results on VEX-4 against VEX-1: on VEX-4, with higher ILP capabilities, the performance improvement is less important than on VEX-1.

For IIR, our approach also yields consistently better performance improvement, up to 2x on XENTIUM and 1.3x on ST240. WLO-First still mostly result in performance degradation after applying SLP extraction.

The same is true for CONV except on XENTIUM between -25 dB and -40 dB where our approach results in slight performance degradation. In this case, the selected SLP solution has a high packing/unpacking cost compared to the performance gain it achieves. However, a performance estimation of the obtained SLP grouping solution, as suggested in [7], can be used to discard the solution in case a performance degradation is detected. In this setup, we intentionally skipped this step in order to emphasize the fact that selecting smaller WLs, which does effectively increase SLP opportunities, does not however always translate to performance improvement.

The graphics of figure 6 plot the speedup of WLO-SLP over the original floating-point code for both XENTIUM and ST240. On XENTIUM, a speedup of 15x to 45x is recorded. This is expected since this processor does not have a hardware support for floating-point arithmetic which needs to be soft-emulated. On ST240, even though it has hardware support for floating-point, a speedup up to 1.4x is obtained. It is mainly due to the exploitation of SIMD capabilities.

VI. Conclusion

In this paper we presented a new SLP-aware WLO algorithm targeting embedded processors with SIMD capabilities. We implemented our approach in an automated source-to-source compilation framework. Experimental results show the advantage of our approach for efficient exploration of the performance/accuracy tradeoff.

References

[1] K.-I. Kum and W. Sung, "Word-length optimization for high-level synthesis of digital signal processing systems," in *SIPS 98*, 1998.

[2] D. Menard, D. Chillet, and O. Sentieys, "Floating-to-fixed-point conversion for digital signal processors," *EURASIP*, 2006.

[3] S. Larsen and S. Amarasinghe, "Exploiting Superword Level Parallelism with Multimedia Instruction Sets," in *PLDI '00*, 2000.

[4] S. Larsen, E. Witchel, and S. Amarasinghe, "Increasing and Detecting Memory Address Congruence," in *PACT '02*, 2002.

[5] A. Kudriavtsev and P. Kogge, "Generation of Permutations for SIMD Processors," *LCTES '05*, 2005.

[6] J. Shin, M. Hall, and J. Chame, "Superword-Level Parallelism in the Presence of Control Flow," in *CGO '05*, 2005.

[7] J. Liu, Y. Zhang, O. Jang, W. Ding, and M. Kandemir, "A Compiler Framework for Extracting Superword Level Parallelism," in *PLDI '12*, 2012.

[8] K.-I. Kum, J. Kang, and W. Sung, "Autoscaler for c: An optimizing floating-point to integer c program converter for fixed-point digital signal processors," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 2000.

[9] H. Keding, M. Willems, M. Coors, and H. Meyr, "Fridge: a fixed-point design and simulation environment," in *DATE '98*, 1998.

[10] J. Lopez, G. Caffarena, C. Carreras, and O. Nieto-Taladriz, "Fast and accurate computation of the roundoff noise of linear time-invariant systems," *Circuits, Devices Systems, IET*, 2008.

[11] D. Menard and O. Sentieys, "Automatic Evaluation of the Accuracy of Fixed-point Algorithms," *DATE* 2002.

[12] G. Caffarena, C. Carreras, J. A. López, and A. Fernández, "Sqnr estimation of fixed-point dsp algorithms," *EURASIP*, 2010.

[13] A. G. Cilio and H. Corporaal, "Floating Point to Fixed Point Conversion of C Code," in *CC*, 1999.

[14] IRISA/INRIA, "GeCoS," "http://gecos.gforge.inria.fr".

[15] "IDFix," http://idfix.gforge.inria.fr.

[16] H. Nguyen, "Novel algorithms for word-length optimization," *EUSIPCO*, 2011.

[17] Recore Systems, "Xentium VLIW DSP IP core," http://www.recoresystems.com/fileadmin/downloads/Product_briefs/2012-2.0_Xentium_Product_Brief.pdf, 2012.

[18] "VEX," http://www.hpl.hp.com/downloads/vex/.