Sub-word Handling in Data-parallel Mapping

Georgia Psychou[1], Robert Fasthuber[2], Jos Hulzink[3], Jos Huisken[3], Francky Catthoor[2]

[1] Comp. Eng. and Inform. Dep., Univ. of Patras, Greece
(currently at Electr. Eng. and Comp. Syst. Dep., RWTH Aachen Univ. Germany);
psychou@eecs.rwth-aachen.de
[2] IMEC, Leuven, Belgium; {fasthuber,catthoor}@imec.be
[3] Holst Center/IMEC, Eindhoven, The Netherlands; {hulzink,huisken}@imec-nl.nl

Abstract: Data-parallel processing is a widely applicable technique, which can be implemented on different processor styles, with varying capabilities. Here we address single or multi-core data-parallel instruction-set processors. Often, handling and reorganisation of the parallel data may be needed because of diverse needs during the execution of the application code. Signal word-length considerations are crucial to incorporate because they influence the outcome very strongly. This paper focuses on the broader solution space of selecting sub-word lengths (at design time) including especially hybrids, so that mapping on these data parallel single/multi-core processors is more energy-efficient. Our goal is to introduce systematic exploration techniques so that part of the designers effort is removed. The methodology is evaluated on a representative application driver for a number of data-path variants and the most promising trade-off points are indicated. The range of throughput-energy ratios among the different mapping implementations is spanning a factor of 2.2.

1 Introduction

Today designers are using a very simplified methodology to exploit the available signal quantisation freedom. That is in general the case for mapping applications to instruction-set processors, which is our target in this paper. Exceptions are present for cost-sensitive custom ASIC or FPGA oriented designs, but those are not our focus here. Usually, the width of the available data-path (e.g. 32 bit) is used for representing all the signals in a fixed-point notation. Alternatively, all data are just left in floating-point notation when the processor platform supports it. That is the easiest and the most effective approach from a design-time reduction point of view. Many processor vendors explicitly use that argument to motivate the support of expensive floating-point arithmetic in their processors, even in the embedded DSP domain where processor cost overhead is seen as undesirable (see e.g. TI C6x DSPs). In the best case, designers use a quantisation technique that reduces the word-length of all signals in a uniform way to a few power-of-2 values (e.g. [Ha04, Wi11]). This allows to better exploit SIMD or sub-word parallel operation. Usually, that leads to a division of the signals in a few categories: 8, 16 and 32 bit data.

However, we believe that recently introduced and still developing techniques for advanced quantisation exploration [Pa10, No10a] allow to change this state-of-the-art paradigm. With these new emerging techniques, a methodology is enabled where the minimal word-length for all signals is determined individually based on quality of service (QoS) require-

ments. Examples are the bit-error rate (BER) in a wireless system or the signal-to-noise ratio (SNR) for a multi-media system. As a result, a wide range of signals with a very non-uniform distribution is produced, e.g. ranging from 6 to 24 bit [No10a]. So, we can exploit a much wider variation of SIMD groupings and this motivates in its turn to go beyond state-of-the-art hardware-based SIMD concepts where only a few power-of-2 word-lengths are supported. In an ad hoc way, custom ASIC designers are sometimes using this fine-grain quantisation information already, but the resulting range of word-lengths is then used to hardwire different operators and not used for a time-multiplexed SIMD processor. We are trying to explore such solutions with our methodology. Moreover, the methodology is not only applicable on single processing elements (PEs) but also on multi-core processors with multiple homogeneous PEs which are organised in a SIMD fashion.

Our first contribution is the systematic exploration of selecting data word-lengths on different SIMD platform options, with a given distribution over the signals of minimal word-length requirements, including especially the attractive hybrid options. Secondly, we also show how we can obtain a broad range of near-optimal mapping results in terms of performance, energy and area trade-offs for realistic embedded applications on an single or multi-core SIMD processor.

The structure of the paper is organized as follows: Section 2 presents related work, while Sect. 3 exhibits the proposed systematic approach for data organisation within the data-parallel processing context. A case study in Sect. 4 is used to illustrate the potential use of the methodology and to show the benefits through some high-level comparisons with alternative techniques. Section 5 concludes the paper.

2 Related work

In the literature, a number of mapping solutions on data-parallel platforms have been proposed, especially for vector or conventional hardSIMD architectures. Regarding SIMD implemented in software, a number of instantiations are discussed in [BD06, La07, No10b] but without any systematic classification or selection process.

In order to exploit the data-level parallelism present in the application, often the data layout (the order in which the data is stored in the memory) has to be modified and the data sub-words that will be operated in parallel have to be (re-)packed together into words. This process, together with the restructuring of the application code to perform the data parallel execution, is called vectorization or SIMDization. Vector compiler-focused work has been published in [LA00, Te05].Various SIMDization and transformations techniques have been proposed in the general or embedded compiler literature [KL00, XOK07]. They are focused on the basic parallelisation, potentially together with some cost functions to reduce the overhead of dealing with various sub-word lengths. But they do not try to describe or explore the broad search space that is available with non-conventional techniques or hybrids for effectively handling sub-words with different word-lengths.

Many researchers have addressed SIMD mapping on homogeneous SIMD architectures in the past (see e.g. [CF04, Qi09]). They have focused however only on the pure hardware-supported SIMD styles, and very seldom even on the sub-word parallel processing. Exceptions to this are the papers [Fr05, CVE09] which do address sub-word processing but again without covering any design space exploration of hybrids.

3 Systematic exploration of sub-word handling hybrids

3.1 Target platforms description

Typically, the elements, that can be found in sub-word parallel SIMD architectures, are the data memory and the register file, arithmetic operators such as shift-add units, and a sub-word rearrangement unit such as a shuffler (if repacking or other operations are necessary). Traditionally, within SIMD, the arithmetic hardware is capable of handling the sign and carry (or other bits) propagation for each of the operands during the calculations, so that they do not pollute each other (called hardSIMD further on). More recently, a software-controlled technique has been introduced, named softSIMD [BD06]. SoftSIMD does not include dedicated circuitry in the arithmetic units to support the data-parallel operations. When operations move operands across their limits, techniques like resizing of the sub-words (repacking) and masking through placing guard bits [BD06] then have to be employed. The difference between hardware-implemented SIMD and the software version, lies in the shuffler operator that now handles the bits on the MSB or LSB side that can potentially pollute the adjacent sub-words. This sub-word parallel operation with the corresponding support can be realized in a single core processor but also in a homogeneous SIMD multi-core processor. In that case, the overall data parallel solution is constructed of a hybrid between sub-word processing inside the cores and vector SIMD across the cores. For instance if we have 8 cores of 64 bit each and we have support for sub-words of 8, 16 and 32 bit, then the mapping can contain 64 parallel operations on 8 bit, 32 operations on 16 bit and 16 operations on 32 bit.

3.2 Systematic exploration of sub-word lengths

For this analysis, a set of proposed word-lengths for each variable in the application code is taken as input. The set comes from emerging systematic and hence automatable quantization techniques such as [Me10]. It includes information about hard and soft constraints regarding the word-lengths. This allows the mapping stage to effectively exploit the word-lengths that are reduced compared to conventional worst-case techniques.

The methodology to select the appropriate sizes for the signals of an application is illustrated through a compact classification scheme (Fig. 1). This scheme can be used by designers to find optimized options for the application word-lengths in a systematic way, which makes sure that no important option is missed. Some of these solutions are well-known but a number of them are novel (to the best of our knowledge), since the techniques for exploring reduced word-length are not so widely applied yet. More importantly, this top-down split-based tree organization allows a number of very promising hybrid solutions to pop up which are not so easy to find simply by trying out different combinations. Hybrids come up by combining options. One such more obvious illustration is that software SIMD can be used to improve the capabilities of a hardware SIMD platform. An example mentioned in [BD06] is that softSIMD could be used in DSPs when, for instance, hardware support for 16-bit SIMD operations is provided, but not for operands with a lower word-length. In that case, 16-bit hardSIMD instructions are combined with lower word-length softSIMD instructions inside the 16-bit segment. But the searching space is broad and this is what we are trying to explore here.

Initially, the quantization information of all variables in the application code is collected. Then, an important distinction has to be made between selecting signal sizes for applications that allow accuracy loss and those that do not (split **a** in Fig. 1). The information about which direction we follow in the diagram, comes from the application designers, who code the programs.

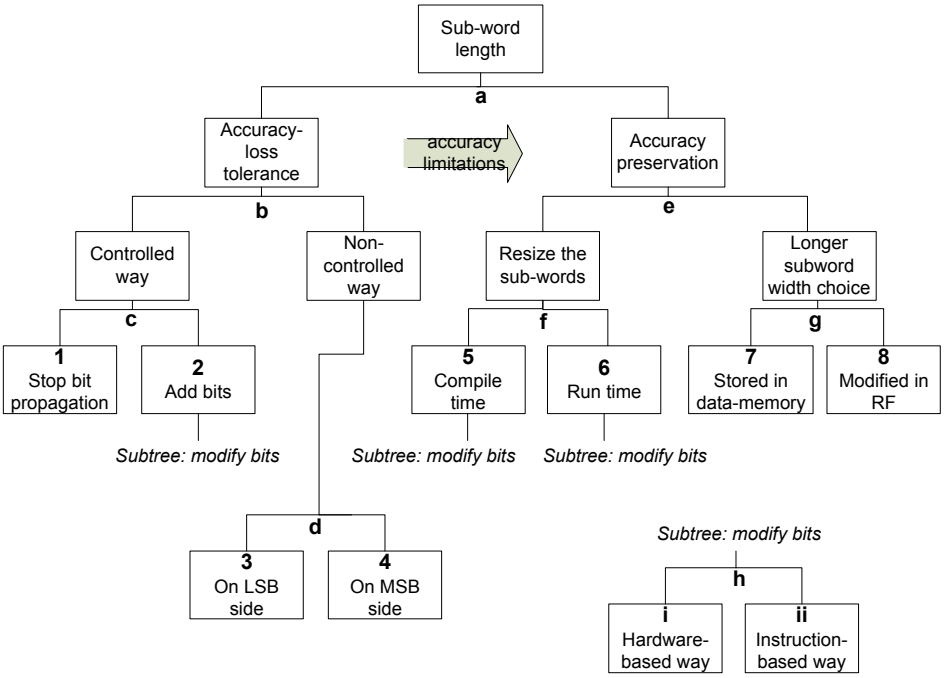


Figure 1: Sub-word length selection

3.2.1 Accuracy loss tolerance.

When the type of application allows that data have some accuracy loss, two potential cases can be found (split **b**). In the first case (left-hand side of split **b**), at compile time it can be known exactly how many bits can be discarded from each data item, namely accuracy loss takes place in a completely controlled way. In the second case (right-hand side of split **b**), the loss of accuracy can not be fully determined until run-time. The bit loss is bounded by a pre-analysed upper limit, to ensure correct functionality. This is the non-controlled subcategory. Whether we are in the controlled or non-controlled branch depends on whether we have sufficient knowledge of how the application evolves during execution time. An example application in the controlled accuracy loss subcategory is a video encoder. Since the code and typical (natural) data streams of the video encoder and the compression standards are predefined, it can be analysed at design time how much accuracy loss will be present. This happens by profiling representative video input sequences and extrapolating from these information regarding the accuracy loss. On the other hand, the subcategory of uncontrolled accuracy loss solutions can be applied in cases like data transmission applica-

tions that are best effort-oriented, such as Skype-like protocols. Since it can not be known exactly at design time how many users will appear, namely the load changes at run-time, it is impossible to accurately preanalyse the data.

An example can be used to illustrate more effectively these cases. We use simple operations as additions and right shifts, which are representative cases of handling bits both on the MSB and LSB side. In this example, the operands, resulting from the quantization analysis, are 5-bit long and the data-path is 30 bits wide. Each of the operands takes part in an addition and/or a right shift. Given a 30-bit data-path, we can operate 6 sub-words of 5 bits in parallel.

In the controlled approach (left-hand side of split **b**), the accuracy loss can be guaranteed either by stopping bit propagation (carry or LSBs) by means of hardware (leaf **1**) or by adding bits (leaf **2**). This guarantee refers to making certain that the operands do not cross their boundaries and pollute their neighbouring ones. Thus, the results remain valid. Leaf **1** applies both to hardSIMD and softSIMD. For softSIMD, the bit cut off can be similar to the hardSIMD case, but with a different internal hardware structure to create less overhead especially when many (non power of 2) sub-word sizes exist. Adding bits can be accomplished by extending the word-lengths and placing guard bits or modifying bits in the existing space (applies for the softSIMD case). In the example shown in Fig. 2a, the accuracy loss is being controlled leading to only 1-bit loss (instead of potentially 2 due to the right shift). This happens by extending the 5-bit operands to 6 bits and zeroing out their LSB before they are shifted.

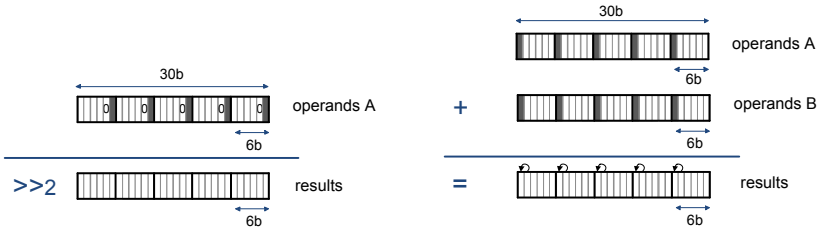


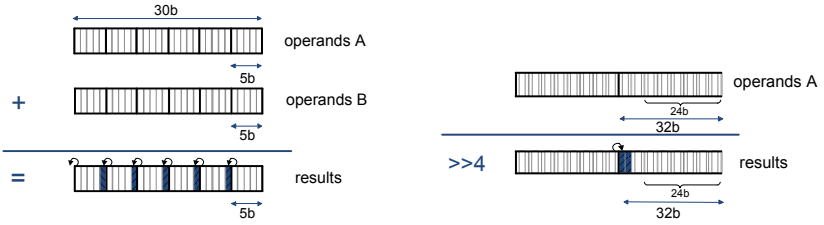
Figure 2: Extended sub-words for controlled accuracy loss and preservation

The placing of the guard bits can either take place in a hardware-based fashion (leaf **2i**), for instance by shuffling of bits, or in an instruction-based way (leaf **2ii**). In the last case, the data-word to be masked is entered as one input of the masking unit. Another word, with the desired composition of guard bits, needs to have been stored upfront in the data memory. The (re)placement occurs through a logic operation.

When accuracy loss occurs in the non-controlled way (right-hand side of split **b**), no special care is taken for the sub-words of the output (after the addition or the shift). Only an upper limit exists for the bit loss (see beginning of Subsect. 3.2.1). In the example, the 5-bit values of the output may contain inaccurate data. Here, the case that there is pollution of data of the neighbouring sub-words on the LSB side (leaf **3**) or on the MSB side (leaf **4**) has to be discussed separately. Regarding leaf **3**, inaccurate data appears (as shown in Fig. 3a) when an overflow occurs during an addition. If no precaution is taken,

the overflow bit moves out of the sub-word and generates an inaccurate result for both this and its neighbouring sub-word (the MSB enters the left-hand sub-word).

Also right-shift operations can lead to accuracy loss in an uncontrolled way (the shifted bits -LSB- enter the right-hand sub-word). These instances are relevant to leaf **4**. Since the MSBs are influenced, it is possible that the results come out totally wrong. However, cases exist that this does not happen, as the one illustrated in Fig. 3b. Here, the operands are two 32-bit sub-words. The data values needed to be represented are only 24 bits. If these 24 bits are chosen to be positioned on the right-hand side (LSB side) of the sub-words, then the 4 bits that are shifted in from the left-hand side do not influence the results on the MSB side. In this way, the results remain valid and no overhead operations need to take place.



(a) during an addition (b) during a right shift

Figure 3: Accuracy loss that occurs in a non-controlled way

3.2.2 Accuracy preservation.

The second category (right-hand side of split **a**) considers the case where a hard guarantee exists for the accuracy preservation. This is true in mission critial applications where not meeting these hard requirements on the results will have an unacceptable impact.

The accuracy preservation can happen either by enlarging the sub-words to prevent bit loss during the operations (split **e** left-hand side) or by upfront (during the word-length decision phase) selecting a bigger word-length before the beginning of the execution (right-hand side of split **e**). In Fig. 2b although the operands are 5-bit long, 6 bits are reserved for each one of them. The 6th bit (MSB) provides space for the overflow bit.

In the case of resizing (left-hand side of split **e**), sub-words are extended to prepare for the operations that follow. The decision on the resizing can be made either at compile (leaf **5**) or at run-time (leaf **6**). The run-time mechanism could be realised with preinstalled hardware that supports precoded senario-based mapping options. The latter are then selected during run-time. For the addition case of our example, this means 6-bit sub-words are selected before the beginning of the operation. For the right-shift operation, (5+n)-bit sub-words are selected before the beginning of the shift by n. In this case, less sub-words are processed in parallel, i.e. only 5 instead of 6 in the case of the addition. Still, every sub-word of the output has a correct result and no sub-word is affected by its neighbouring one. Thus, at the cost of extra resizing operations and a reduced number of parallel sub-words, a hard guarantee can be provided for the accuracy. To achieve an efficient resizing for the softSIMD case, guard bits are used. The placing of the guard bits can take place in a similar way as for leaf **2** (subtree **h**).

At the right-hand side of split **e**, during the initial selection of the word-lengths, we choose a bigger word-length in order to provide enough space for the extra bits i.e. the overflow bit in the case of the addition. In our example instead of the minimal length of 5 bits, we select 6-bit word-lengths. In that case, the correctness of the results is maintained, and extra resizing is avoided, but we operate only 5 sub-words in parallel. The bigger word-lengths can either have been stored in the data-memory in the intended extended form (leaf **7**) or the extension can take place during their stay in the register file (leaf **8**). Within softSIMD, the extra space provided by the bigger word-length selection is filled with guard bits on the one or both sides of the sub-words depending on the next operations.

3.2.3 Hybrid solutions.

Most importantly, many hybrid solutions can be derived in a fully systematic way by combining characteristics of any subset of leaves in Fig. 1. All these leaves are namely potentially compatible with each other. Here, we will describe only a few to illustrate this property. These combinations lead to new, non-obvious choices.

One such case is that within the controlled accuracy loss approach (left-hand side of split **b**), the stopping of bit propagation (leaf **1**) can be combined with the addition of bits at specific places (leaf **2**). In that way, space can be provided for the MSBs by cutting off LSBs. The guard bits for this case are placed at the MSB side of the sub-words. In Fig. 4 such an example is shown. The LSB of the 6-bit operands is stopped by means of hardware during the right shift by 1. In combination to that, a guard bit is placed on the MSB of the sub-words to potentially prepare them for an addition. If we would choose to stop bit propagation on the MSB side as well (during the addition), then the neighbouring sub-words would still be protected but the accuracy of the current sub-words would be altered (as they cannot expand to the extra bit any longer). The other option to preserve accuracy would be to resize the sub-words to a bigger word-length before the execution of the operations. But this potentially means less parallelism exploitation. This hybrid approach provides a way to avoid the use of a bigger word-length. A second hybrid

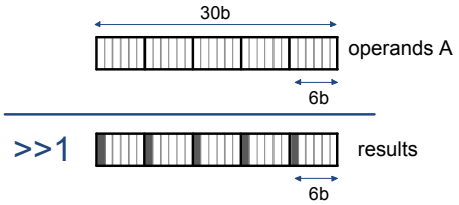


Figure 4: Hybrid approach for controlled accuracy loss

illustration considers the combination of the resizing option (left-hand side of split **e**) with controlled loss of accuracy (left-hand side of split **b**). This could be considered when the loss of accuracy would be too big if no resizing takes place but a flawless accuracy preservation is also not preferable (e.g. because it leads to lower parallelism exploitation). Many other promising hybrids exist in the overall search space.

It becomes obvious that this diagram can be effective for the selection of the platform characteristics, since certain platform options can lead to more optimized mapping results depending on the application domain and the overall system requirements. However, even

when the platform is already defined this diagram remains partly relevant. For example, instruction-based masking can be performed as long as the basic logical operations are supported on the platform. This is common for conventional instruction-set processors.

The way to select and evaluate the different possibilities is dependent of course on the application and platform characteristics. But also trade-offs are present. We briefly discuss here the main issues that play a role in this decision making. A more systematic evaluation is part of future work. The total number of groups of lengths should be small so that the overhead involved with resizing the sub-words is minimized (e.g. in the shuffler). If, for example, in an application among other word-lengths needed, 13- and 16-bit data occur at similar frequencies, it would make sense to use the 16-bit value for both cases. In that way, the support of an extra word-length is avoided. In addition, a resizing operation should be avoided when the accuracy loss is acceptable compared to the gain due to more difficult masking and repacking. For example, if an application allows that some LSBs are lost then it is better to avoid repacking to preserve accuracy and make use of this tolerance of the application so that energy consumption is further minimized. A specific example illustrating trade-offs regarding this selection decision is discussed in Subsect. 4.2.

4 Evaluation of the proposed methodology

4.1 Driver algorithm

An image filter is used as driver for the evaluation of the proposed sub-word handling techniques. The illustration regards the mapping of a 1-dimensional Gauss loop filter applied to an input frame. Different architectures are considered for mapping the Gauss loop. However, due to space limitations, only one mapping implementation will be discussed more thoroughly now and the others will be summarized and compared in Sect. 4.3. The

program 1 Optimized Gauss loop (1D)

for $x = 1 \rightarrow N - 1$ **do**
  for $y = 2 \rightarrow M - 1$ **do**
    Op1: $MulRes0 = imsub[x-1][y] + imsub[x+1][y]$;
    Op2: $MulRes7 = MulRes0 * Gauss[0][1]$;
    Op3: $MulRes4 = imsub[x][y] * Gauss[0][2]$;
    Op4: $imgauss\_x[x][y] = MulRes4 + MulRes7$;
  end for
end for

optimized code of the 1D filter is depicted in program 1. The operations involved are two additions (Op1, Op4) and two constant multiplications (Op2, Op3). The filter coefficients are shown in Table 1. The multiplications can be performed either by using a multiplier unit or by a shift-add unit (by applying strength reduction). For the current illustration, when the multiplications are performed in a shift-add way, the right-shift approach is chosen and for the constants the Canonical Signed Digit (CSD) [Ba07] encoding is considered (see Table 1). The first multiplication is performed in three shift-add/subtract steps by 4, by 3 and by 2. Similarly, for the second multiplication right shifts by 3, by 2 and by 5 and the corresponding additions/subtractions need to take place.

	Gauss[0][1]	Gauss[0][2]
Decimal	655	983
Binary	1010001111	1111010111
CSD coding	101001000-	10000-0-00-

Table 1: Gauss coefficients

Signal	Length (bits)	Signal	Length (bits)
imsub	7	imgauss_x	12
MulRes7	10, 11, 13	MulRes4	9, 10, 13

Table 2: Gauss loop signals

4.2 Application of the methodology

We will illustrate the mapping of the code using the softSIMD related leaves in Fig. 1. SoftSIMD makes sense to use while performing simpler operations that do not lead to a potential explosion of the operands' word-length [BD06]. Thus, for the multiplications, the shift-add scheme is employed. We consider a 48-bit data-path that consists of a shift-add-subtract unit (SAS) for the arithmetic operations, a shuffler unit for the repacking operations and intermediate registers (softSIMD data-path).

For the mapping, minimal quantization-based word-lengths (for the input signals, the intermediate results of the constant multplications and the final results), as shown in Table 2, are our starting point. Then, in correspondence to the Fig. 1, the word-length choices of this Gauss filter code occur as follows:

1. The first operation (*Op1*) is an addition of 7-bit values (*imsub*). The result needs to be stored in 8-bit values. To avoid a repacking operation from 7 to 8 bits, the signals are stored as 8-bit in the data memory (leaf **7**). That means 6 sub-words are in parallel in the 48-bit data-path. The result of this operation (MulRes0) is used in the second operation which is a multiplication with a Gauss coefficient. The second multiplication starts with a right shift by 4. Since the operands need to be shifted by 4, they need to be extended to 12-bit values, before they enter the data-path. So that accuracy is preserved during the right shift, 1 LSB is cut-off from the first addition result (leaf **1**) and guard bits are added (leaf **5i** or **5ii**) so that the sub-words are 12-bit long to prepare for the next multiplication. Combining these leaves is possible as explained in Subsect. 3.2.3.

2. During the second operation (*Op2*), after the first shift-add operation by 4 (explained before) and prior to the second shift operation by 3, a repacking operation of the variable to 16 bits needs to take place so that accuracy is preserved (leaf **5i** or **5ii**). Because two guard bits are required at the left-hand side of the repacked multiplicand (one for the additions within the multiplication and one extra for the addition of the fourth operation of the Gauss loop), the result is shifted out of the sub-words during the last shift by 2. In this case, care must be taken so that the extra bits are cut off and do not enter the adjacent operand (leaf **1**). This choice is a hybrid combination of the resizing option (leaf **5i** or **5ii**) and the stopping of bit propagation (leaf **1**), that allows us to exploit the given word-length in an efficient way while we are also obeying the accuracy limitations. If the guard bits are not placed, potential overflows coming from the additions can corrupt the results. On the other hand, if the 2 LSBs are not cut-off, right-hand sub-words may be polluted. Another alternative would be that an extra word-length is then supported, for example the 18-bit or 24-bit word-length. But this would increase the hardware overhead (for example in the shuffler because an additional repacking operation would have to be supported) and would limit the parallelization potential. With the 18-bit or 24-bit option only two sub-words

could operate in parallel in the 48-bit data-path. This allows this hybrid combination to appear as an optimal option.

3. For the third operation (*Op3*) similar choices with the previous operation are made. The imsub data are stored as 8-bit in the data memory (leaf **7**) and are resized to 12-bit by the register file (leaf **8**) before they enter the data-path. As before, the sub-words operate as 12-bit during the first cycle and as 16-bit during the two subsequent cycles.

4. The fourth operation (*Op4*) is an addition of the multiplication results (*Op2*, *Op3*) stored into a 12-bit value (according to quantization results). In this case, 4 sub-words are working in parallel.

More detailed information on the exact mapping is available in [Ps10].

4.3 Comparison with mapping on alternative data-paths

Alternative data-paths. Besides the softSIMD discussed above, alternative data-paths are considered. Like softSIMD, hardSIMD data-paths also include a shift-add-subtract (SAS) unit and a shuffler. The differences with softSIMD have been discussed in Sect. 3. The cases considered are one more unconventional hardSIMD option (HardSIMD SAS, 3-wl) that supports all three sub-word sizes (8-, 12-, 16-bit), one more conventional (HardSIMD SAS, 2-wl) that supports only power of 2 sub-word sizes (in this case 8-, 16-bit). Moreover, the SIMD case where only one sub-word size (24-bit) is considered (SIMD SAS, 1-wl). In the last case, no shuffler is needed and the SAS unit does not need to handle the carry and other bits (the operands remain within their boundaries).

As further reference, we also consider hardwired multiplier data-paths. In this case, a multiplier is used in the position of the SAS unit. Operands' width during the multiplications is calculated as follows. The input operands (*imsub[x...]*) are 7-bit and the Gauss coefficients are 10-bit long. That means that the width needed for each multiplication result is at least 17 bits, and when rounded to the next power of 2, this becomes 32 bits. For the multiplier approaches discussed in this paper, a more optimized choice of 24 bits is used as an optimistic comparison reference. Moreover, for the additions existing in the application an adder is needed and when more than one word-length is supported, a shuffler unit is present. In this category, two cases are considered, one (HardSIMD Mult, 3-wl) where three sub-word sizes (8-bit for the initial addition, 24-bit for the multiplications and 12-bit for the final addition) are supported and one (SIMD Mult, 1-wl) where only one size is supported. In the last case, six groups of two 24-bit operands are operating in parallel and operands remain within their limits during the operations.

Relative area, energy consumption and throughput. The different data-path options with the required functionalities have been implemented in TSMC 40 nm std. cell technology, using Cadence RTL Compiler and SoC Encounter for synthesis and place and route. To obtain the area, energy and throughput numbers for the components, a similar estimation flow as in [Fa09] has been applied. Each data-path is synthesized for the maximum clock frequency; no pipelining in the arithmetic data-path is applied. The algorithm is manually scheduled and optimized for each data-path variant. Software pipelining is applied in cases where more than one operator is present. The number of required clock cycles (to complete one Gauss iteration; 12 pixels) together with the critical path is then

used to determine the relative throughput. For the energy estimation, the actual activation count for each data-path unit is considered. Table 3 summarizes the results.

Data-path option	Accesses of operators	Area	Energy	Through-put	Through. /energy
SoftSIMD	27*SAS + 19*Shuf	1.63	0.80	1.69	2.12
HardSIMD SAS, 3-wl	27*SAS + 11*Shuf	1.48	0.72	1.27	1.76
HardSIMD SAS, 2-wl	30*SAS + 4*Shuf	1.28	0.72	1.33	1.85
SIMD SAS, 1-wl	48*SAS	1.00	1.00	1.00	1.00
HardSIMD Mult, 3-wl	5*Adder + 12*Mult + 9*Shuf	7.84	2.05	2.00	0.96
SIMD Mult, 1-wl	12*Adder + 12*Mult	6.63	2.00	2.29	1.15

Table 3: Estimation of the total relative area, energy consumption, throughput and throughput-energy quotient for one Gauss iteration (data-path = 48 bits).

As it can be seen in Table 3, SIMD SAS 1-wl requires the lowest area, which is reasonable since no bit handling among sub-words needs to take place. However, the delay significantly increases in this case. The softSIMD and hardSIMD SAS (3-wl) data-paths, that exploit maximally the different solutions presented in Sect. 3, are among the near-optimal solutions in the table. SoftSIMD achieves a higher throughput, compared to the other SAS implementations (because of the small critical path and the dense scheduling). More importantly, it achieves the highest throughput-energy ratio. HardSIMD SAS data-paths exhibit energy-efficiency because of the simpler shuffler unit. HardSIMD SAS (2-wl) has a more optimal throughput-energy combination than the hardSIMD (3-wl) because of the slightly smaller critical path. The multiplier data-paths have a big area and energy overhead but achieve the highest throughput. The throughput-energy ratio is a viable metric because at the circuit level, it is the most relevant trade-off. A faster circuit indeed requires more energy to perform a computation than a slower one, due to buffer sizing. For instance, the softSIMD circuit could be made slower and therefore it will also consume less energy. Note that these results scale as well to a multi-core processor composed of a homogeneous set of PEs with sub-word parallel support, as indicated already in subsection 3.1. In that case the mapping above is simply copied for each of the PEs due to the homogeneous overall mapping principle.

The table demonstrates a wide range of options in the total space, which lead to a wide range of quantitative results. According to the objectives, the options are worth exploring. For instance, when high timing constraints are present using a multiplier data-path is a better option. When design time is not a hard limitation, softSIMD as well as hardSIMD which supports non power-of-2 sub-word sizes are worth exploiting, since they exhibit the best energy-efficiency.

5 Conclusion

In this paper, the broad solution space of organizing parallel data with minimal data word-length requirements in domain-specific processors during mapping (at design time) is explored in a systematic way. These processors can consist of single or multi-core architectures organized in a homogeneous SIMD fashion. A methodical way reveals a broad number of options and can potentially save design time. The application of the proposed

sub-word handling analysis on a driver algorithm has substantiated a wide mapping solution space. Our goal is to facilitate an intermediate step of the mapping process, after having selected the minimal word-length requirements and before the compiler tool decides and applies the final mapping. This precompiler methodology can provide pragma or intrinsic-based guidance to support the traditional compilers.

References

[Ba07] E.Backenius, Reduction of Substrate Noise in Mixed-Signal Circuits, PhD thesis, Linkoping University, 2007.

[BD06] Emulate SIMD in software, `http://www.bdti.com/InsideDSP/2006/12/06/Bdti`, 2006.

[CF04] G.Cichon, G.Fettweis, Mouse: a shortcut from Matlab source to SIMD DSP assembly code, *SAMOS'IV*, Greece, July 2004.

[CVE09] J.Corbal, M.Valero, R.Espasa, Exploiting a new level of DLP in multimedia applications, *MICRO conf*, New York, Dec. 2009.

[Fa09] R.Fasthuber et al, Novel energy-efficient scalable soft-output SSFE MIMO detector architectures, *Proc. Intnl. Conf. (IC-SAMOS)*, July 2009.

[Fr05] F.Franchetti et al, Efficient utilization of SIMD extensions, *Proc. Of the IEEE*, Vol.93, 2005

[Ha04] K.Han et al, Data wordlength reduction for low-power signal processing software, *Proc. IEEE Wsh. on Signal Proc. Syst. (SIPS)*, Austin TX, IEEE Press, pp.343-348, Oct. 2004.

[KL00] A. Krall, S.Lelait, Compilation techniques for multimedia processors, *Int. Journal on Parallel Programing, 28(4)*, 2000.

[LA00] S. Larsen, S.Amarasinghe, Exploiting superword level parallelism with multimedia instruction sets, *ACM SIGPLAN Notices, 35(5):145156*, 2000.

[La07] A.Lambrechts et al, Enabling word-width aware energy optimizations for embedded processors, *Proc. ODES*, San Jose CA, pp.66-75, March 2007.

[Me10] D.Menard et al, Quantization mode opportunities in fixed-point system design, *Proc. 18th Eur. Sign. Proc. Conf.(EUSIPCO)*, Denmark, pp.-, Aug. 2010.

[No10a] D.Novo et al, Exploiting Finite Precision Information to Guide Data Flow Mapping, *Proc. 47th ACM/IEEE Design Automation Conf. (DAC)*, Anaheim CA, pp.248-253, June 2010.

[No10b] D.Novo et al, Ultra Low Energy Domain Specific Instruction-set Processor for On-line Surveillance, *Proc. SASP co-located with DAC*, Anaheim CA, pp.30-35, June 2010.

[Pa10] K.Parashar et al, Fast performance evaluation of fixed-point systems with un-smooth operators, *Proc. IEEE Intnl. Conf. Comp. Aided Des. (ICCAD)*, San Jose, pp.9-16, Nov. 2010.

[Ps10] G.Psychou, Optimized SIMD scheduling and architecture implementation for ultra-low energy bioimaging processor, M.Sc. thesis, Univ. of Patras and IMEC NL, 2010.

[Qi09] M.Qiu et al, Energy-aware loop scheduling and assignment for multi-core, multi-functional unit architecture, *J.Signal Processing Systems*, Vol.57, pp.363-379, 2009.

[Te05] C.Tenllado et al, Improving superword level parallelism support in modern compilers, *Wsh. Codes-ISSS*, NY, pp.303-308, Oct. 2005.

[Wi11] Wikipedia, `http://en.wikipedia.org/wiki/Fixed_point_arithmetic`, read Sep.2011.

[XOK07] L.Xue, O.Ozturk, M.Kandemir, A memory-conscious code parallelization scheme, *Proc. 44th Conf. Design automation*, pages 230–233, NY, USA, 2007. ACM Press.