Bit–Accurate Simulation of Convolution–Based Filtering on Reconfigurable Hardware

Holger Scherl*
Friedrich–Alexander–Universität Erlangen–Nürnberg
scherl@informatik.uni-erlangen.de

Markus Kowarschik†
Siemens Medical Solutions
markus.kowarschik@siemens.com

Joachim Hornegger‡
Friedrich–Alexander–Universität Erlangen–Nürnberg
hornegger@informatik.uni-erlangen.de

Abstract

In 3D cone–beam reconstruction of modern medical scanners, the most time–consuming preprocessing step is typically represented by a 1D shift–invariant filtering of the projection data. In order to meet the processing requirements of state–of–the–art medical scanners, we have implemented the convolution–based filtering step in hardware using Field–Programmable Gate Array (FPGA) technology. A significant downside to our FPGA–based approach is that it is no longer possible to use floating–point arithmetic, as is the case for conventional CPU–based implementations. Consequently, it is necessary to realize the whole processing chain of the convolution using fixed–point numbers instead. In order to simulate the effects of fixed–point calculations on the accuracy of the final numerical results, we have developed a highly flexible and bit–accurate software prototype of the hardware design. It covers both the FFT routines as well as different scaling strategies of the involved fixed–point data types. In this paper, we will demonstrate our simulation methodology and present results for the convolution–based filtering, employing various scaling approaches and bit widths in the different computation stages. Finally, we arrive at a satisfying configuration for the FPGA–based implementation. Our simulation results reveal that the numerical results delivered by the selected implementation are sufficiently accurate in comparison to a reference code based on double–precision floating–point arithmetic.

*Department of Computer Science, Institute of Pattern Recognition, D–91058 Erlangen, Germany
†Siemens Medical Solutions, Components Division (CO), D–91052 Erlangen, Germany
‡Department of Computer Science, Institute of Pattern Recognition, D–91058 Erlangen, Germany

1 Motivation

In 3D cone–beam reconstruction, the most time–consuming preprocessing step is typically represented by a 1D shift–invariant filtering of the projection data. This task can most efficiently be addressed by applying the convolution theorem, which states that the convolution of any two vectors is given by the component–wise product of their Fourier coefficients [Ueb97]. Hence, it is required to determine the discrete Fourier transforms (DFTs) of the input vectors as well as the DFT of the filter kernel [GW02]. After the subsequent component–wise multiplications of the Fourier coefficients, the inverse DFTs of the products are computed. Of course, the overall efficiency of this scheme results from the use of computationally efficient Fast Fourier Transform (FFT) algorithms.

In order to meet the processing requirements of state–of–the–art medical scanners, we have implemented the convolution–based filtering step in hardware using Field–Programmable Gate Array (FPGA) technology [MB04]. As the name suggests, these logic circuits are reconfigurable and can thus be programmed specifically for a certain task. The main advantage of FPGAs is their ability to process data in a massively parallel manner, where the degree of parallelism is of course restricted by the available resources (e.g., multipliers and lookup tables) as well as algorithmic constraints. In the case of convolution–based filtering, many arithmetic operations can be executed in parallel. Therefore, enormous processing speeds can be achieved, even when using moderate clock rates only.

A significant downside to our FPGA–based approach is that it is no longer possible to use floating–point arithmetic, as is the case for conventional CPU–based implementations. Consequently, it is necessary to realize the whole processing chain of the convolution using fixed–point numbers instead. In order to simulate the effects of fixed–point calculations on the accuracy of the final numerical results, we have developed a highly flexible and bit–accurate software prototype of the hardware design. It covers both the FFT routines as well as different scaling strategies of the involved fixed–point data types. Additionally, various hardware restrictions of the FPGA architecture had to be considered throughout the simulation task to achieve optimal performance and to meet the space restrictions on the chips.

In our paper, we demonstrate our simulation methodology and present results for the convolution–based filtering, employing various scaling approaches and bit widths in the different computation stages. Our simulations include different scaling strategies for both the FFT computations as well as the intermediate processing steps of the convolution chain. Finally, we will arrive at a satisfying configuration for the FPGA–based implementation.

2 Convolution–Based Filtering

In order to perform a convolution–based filtering, the DFT of each one–dimensional input signal has to be computed. The actual convolution in Fourier space is performed by component–wise multiplication of its Fourier components with that of the filter kernel. Finally, the IDFT of this product has to be computed. We focus on convolutions of block–scaled 16 bit data both for the input and output signals. As vector lengths, both 2048 (2K)

and 4096 (4K) are considered. In order to avoid aliasing artifacts, the input signals have to be zero–padded up to 4K or 8K, respectively [GW02]. An important property is that the DFT of an even and real–valued sequence is again real–valued [Ueb97]. This property is exploited in our implementation of the convolution since the filter sequence is assumed to be real–valued as well as even. Thus, the convolution of two input–vectors can be performed simultaneously with only one convolution chain. In our implementation we use power of two FFT algorithms only (Radix2). In the course of each stage, the FFT algorithm passed through the entire data set. It operates on pairs of complex numbers of the sequence at a time and replaces them by the calculated results (so–called *butterfly* operations).

3 Simulation Approach

In order to simulate the convolution chain using fixed–point data types, we based our software prototype on the *SystemC* library [Ini]. This library offers the possibility to use variables with limited accuracy, thus facilitating the simulation of configurable bit–widths of the involved data types. The simulation itself can be divided into two different parts. First, the involved FFT and IFFT (inverse Fast Fourier Transform) computations have to be simulated with different input and output bit–widths and with an appropriate scaling strategy. In a second step, the intermediate computing stages have to be implemented taking into account the input and output bit–widths of the FFT and the IFFT. In order to keep the input signals properly scaled within the FFT computations, we selected two different approaches. We will refer to them as *block–floating point mode* and *unscaled precision mode*, respectively.

3.1 Block–Floating Point Mode

In the block–floating point mode, the complexity of scaling is integrated almost completely into the FFTs. As we move from stage to stage through the calculation, the magnitudes of the numbers in the sequence generally increase, which means that they can be properly scaled by right shifts. In this case, we test after each butterfly computation, whether an overflow has occurred. Whenever an overflow has occurred, the entire sequence (part of which will be new results, part of which will be entries yet to be processed) is shifted right by one bit and the computations are continued at the point at which the overflow has occurred. It can be shown that there are only two overflow events possible within each FFT stage [Wel69]. One advantage of this approach is that only a minimum of computations is required for scaling purposes in between the FFT computations.
The processing chain looks as follows. The 16 bit entries of the block–scaled input vectors are padded with zeros to a length of 24 bit. Then, the FFT and the multiplication with the DFT of the filter kernel (32 bit numbers in our case) are performed. After the multiplication, the resulting numbers are 56 bit wide. We simply chop off the trailing 32 bits to apply the IFFT with again 24 bits on input. Finally, the results of the IFFT (24 bit each) are truncated to 16 bits for the output. For improved accuracy, we also tried out this approach with 35 as the input and output bit–width of both the FFT and the IFFT.

3.2 Unscaled Precision Mode

The scaling inside the FFT blocks is done differently in unscaled precision mode. Within each FFT stage, the bit–width of the results is increased by one. Therefore, during the computations, overflows cannot occur anymore. Analogously, this strategy can be understood as introducing a global right shift before each stage without loosing the least significant bit. In contrast to the previous approach, this technique exhibits the disadvantage that the output sequence of the FFT may not be properly scaled anymore. Therefore, a more advanced scaling technique is needed in between the FFT and the IFFT block. We will refer to our implemented technique as *dynamic scaling*. During the dynamic scaling, the minimum number of irrelevant bits to the right of the sign bit of all numbers of the current data vector is computed. Afterwards, all vector elements are left–shifted by that number, and the block exponent is adapted appropriately. This results in a simplification of the hardware implementation of the FFT and IFFT routines, which in turn leverages their optimization for processing speed. The loss of efficiency in the dynamic scaling stages in between the FFT and the IFFT can easily be accounted for by the use of pipelining in the hardware implementation [HP03].
The processing chain now involves the computation of the FFT of input vectors of (zero-padded) 24 bit numbers using the unscaled precision mode. Therefore, the output bit–width after the FFT block is 37 bit for the case of a 4K convolution (or 38 bit for the case of an 8K convolution). Due to FPGA hardware constraints related to the internal architecture of multipliers, the values of the output sequence are truncated to 35 bits, and a dynamic rescaling is performed right before the multiplication stage. The 67 bit wide entries of the resulting product vector (recall that the DFT of the filter sequence is given as a vector of 32 bit numbers) are truncated to 36 bits, and a dynamic rescaling is performed again. Then, the vector entries are truncated to the input bit–width of the IFFT (i.e., 24 bit). After the IFFT, a final dynamic rescaling is introduced before truncating the vector entries to the required output bit–width with minimal loss of accuracy.

4 Results

Throughout the evaluation of the accuracy of our different scaling approaches, we used projection images that are generated with the simulation tool *DRASIM* that had been provided by Siemens Medical Solutions. We used two phantom descriptions: the Head Phantom and the Thorax Phantom. Descriptions of these phantoms can be found on the FORBILD website [Gro]. In order to measure the accuracy of our results, we compared each computed signal to the corresponding one that was computed using a reference code based on double–precision floating–point arithmetic. For the bit comparison of two numbers, we counted the number of vanishing most significant bits of their absolute difference. Then, for each result vector, both the minimum and the average of the determined corresponding leading bits are used.
In Table 1, the measured accuracies using the block–floating point scaling strategy are given. The achieved accuracy is about 9 to 10 bits both for 24 bit and 35 bit FFT structures. The loss of several valid bits results from the bit truncation after the multiplication with

	Head phantom	Thorax phantom
4K convolution (block–floating point)		
24 bit FFT	10 (10.98)	11 (11.00)
35 bit FFT	10 (11.38)	11 (11.35)
4K convolution (block–floating point improved)		
24 bit FFT	11 (11.98)	
35 bit FFT	14 (14.98)	
8K convolution (block–floating point)		
24 bit FFT	10 (10.44)	9 (9.96)
35 bit FFT	10 (10.65)	10 (10.43)

Table 1: Simulation results for block–floating point scaling strategy

	Head phantom	Thorax phantom
4K convolution (unscaled precision)		
23 bit FFT	14 (14.98)	15 (15.00)
24 bit FFT	14 (14.98)	15 (15.00)
8K convolution (unscaled precision)		
22 bit FFT	14 (14.84)	14 (14.95)
24 bit FFT	14 (14.84)	14 (14.95)

Table 2: Simulation results for unscaled precision scaling strategy

the DFT of the filter sequence. Therefore, we extended the scaling strategy of the block–floating point mode after the multiply stage by performing first a truncation to 35 bits and second by a dynamic rescaling afterwards. Then, the sequence is truncated to match the input size of the IFFT. Now, the results are as expected for the 35 bit case, but not much better for the 24 bit case (see block–floating point improved in Table 1).
Because of its advantageous structure with regard to hardware implementation and efficiency, we turned our attention more on the unscaled precision arithmetic. Table 2 shows the measured accuracies. The results demonstrate that our preferred approach yields very good accuracy in comparison with the double–precision floating–point results. The measured accuracy does not change even when we tweak the input bit width of the FFTs to match optimally the hardware restrictions both in the 4K convolution case with an optimal input bit–width of 23 bits and in the 8K convolution case with an optimal input bit–width of 22 bits.

5 Conclusions

We focused on the simulation of bit–accurate convolution–based filtering using fixed–point data types exclusively. In this paper, we have presented a simulation framework that allows

to investigate, if the selected scaling strategies yield sufficient accuracy with different possible bit–widths of the involved data types.
The constraints on the results represent a trade–off between available resources within the FPGA and the accuracy of the involved computations. However, our simulation results demonstrated that the numerical results delivered by our unscaled precision scaling strategies are sufficiently accurate for our target application. On output, at least 14 bits of the 16 bit block–scaled result could be gained in comparison to the computation with double–precision floating–point arithmetic.

References

[Gro] Phantom Group. `http://www.imp.uni-erlangen.de/phantoms`.

[GW02] R.C. Gonzales and R.E. Woods. *Digital Image Processing*. Prentice Hall, 2. edition, 2002.

[HP03] J.L. Hennessy and D.A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 3. edition, 2003.

[Ini] The Open SystemC Initiative. `http://www.systemc.org`.

[MB04] U. Meyer-Baese. *Digital Signal Processing with Field Programmable Gate Arrays*. Springer, 2. edition, 2004.

[Ueb97] C.W. Ueberhuber. *Numerical Computation 2*. Springer, 1997.

[Wel69] P.D. Welch. A fixed–point fast fourier transform error analysis. *IEEE Transactions on Audio and Electroacoustics*, 17(2), 1969.