Reducing Energy Consumption of Data Transfers Using Runtime Data Type Conversion

Michael Bromberger[1,2(✉)], Vincent Heuveline[2], and Wolfgang Karl[1]

[1] Karlsruhe Institute of Technology, Karlsruhe, Germany
{bromberger,karl}@kit.edu, michael.bromberger@h-its.org
[2] Heidelberg Institute of Theoretical Studies, Heidelberg, Germany
vincent.heuveline@h-its.org

Abstract. Reducing the energy consumption of today's microprocessors, for which Approximate Computing (AC) is a promising candidate, is an important and challenging task. AC comprises approaches to relax the accuracy of computations in order to achieve a trade-off between energy efficiency and an acceptable remaining quality of the results. A high amount of energy is consumed by memory transfers. Therefore, we present an approach in this paper that saves energy by converting data before transferring it to memory. We introduce a static approach that can reduce the energy up to a factor of 4. We evaluate different methods to get the highest possible accuracy for a given data width. Extending this approach by a dynamic selection of different storage data types improves the accuracy for a 2D Fast Fourier Transformation by two orders of magnitude compared to the static approach using 16-bit data types, while still retaining the reduction in energy consumption. First results show that such a conversion unit can be integrated in low power processors with negligible impact on the power consumption.

Keywords: Energy reduction · Approximate computing · Data type conversion

1 Introduction

Due to the slowing of Moore's law as Dennard scaling reaches the physical lithographic limits at around 5 nm, new methods of increasing performance per watt will have to be found [4]. One possible answer is to support specialized hardware for particular applications, but this specialization results in so-called "dark silicon", i.e. silicon that is not used in all use-cases and presents a fixed cost overhead in such cases. Reducing the energy consumption is essential in low power processors and embedded systems where the battery or heat dissipation are often critical limitations. Memory accesses consume a considerable part of the energy in today's computing systems. An integer operation is 1,000× less energy consuming than an access to a memory like DRAM [10]. The idea of approximate computing (AC) has been suggested as a possible means of increasing performance per watt across the gamut of computing systems. AC relaxes

F. Hannig et al. (Eds.): ARCS 2016, LNCS 9637, pp. 239–250, 2016.
DOI: 10.1007/978-3-319-30695-7_18

the accuracy of results produced by hard- and software in order to get an energy-efficient execution. Many algorithms from machine learning or image processing have an inherent resilience to such inexact operations.

We consider different approaches to reduce the amount of data that has to be transferred to memory, while getting the best achievable accuracy in each case. Our focus lies on conversion methods for a single data type rather than a bunch of data, because we want to improve energy efficiency of loads and stores inside a processor. Therefore, the first contribution of the paper is an evaluation of different conversion methods. A dynamic selection of data types provides a higher accuracy of results, while retaining the benefit of reducing the energy consumption. Secondly, a detailed measurement of the energy consumption of different embedded system platforms is given. We use memory footprints based on converted data for the measurements. Finally, we give preliminary results of a design that implements our results.

2 Related Work

Current approaches often focus on reducing the accuracy of hardware execution units [2]. An approximation of a software function with a certain number of inputs and outputs is given by a trained neural network (NN) [7]. Since software execution of a NN is slow, hardware support like a neural network processing unit is required. AC tools exist that lower the burden of programmers to decide which parts of an application can be executed approximately [5]. Hardware memoization techniques approximate mathematical and trigonometric functions [6] as well as fuzzy floating point (FP) operations [1]. Furthermore, there exist self-tuning off-line approaches, which use different AC kernels running on a GPU. Such approaches find the best available performance for a given result quality [16].

However, a considerable amount of energy is consumed by memories and memory transfers. Therefore, Sampson et al. have introduced EnerJ which uses approximated data types on a high level of abstraction [17]. Increasing the refresh cycle of certain DRAM memory regions raises the probability to read incorrect data, but reduces energy consumption [11,12]. Cache misses caused by loads are very expensive in terms of latency and energy consumption in modern architectures. Instead of loading a value that is missing in the cache, an approximated value is generated according to a history of loaded values [19]. Such approximated values have minor impact on the accuracy of output results for some applications. There exist approaches to store approximated data using Solid State Disks (SSD) [18] and Phase-Change Memory (PCM) [14]. Compressing data before storing it into memory reduces the overhead for transferring the data and increases the amount of data that fits into a certain memory region [20]. Block FP formats, in which several mantissae use the same exponent, also reduce the amount of data. Our approach reduces the size of data on a single data type level where we only consider information given by the current data type. Afterwards, several data types can be collected and compressed by above approaches. Such an approach requires domain-specific user knowledge. This issue has been addressed by Baek et al. [3].

3 Consideration of Different Conversion Methods

In this section, we evaluate which conversion methods are suitable for different algorithms. We used `Octave` to get a rapid prototype implementation. A statistical metric like the Mean Square Error (MSE) is sufficient enough for getting a deeper understanding of how different conversion methods influence the accuracy of results.

$$MSE(x_i, y_i) = \frac{1}{n} \sum_{i=0}^{n} (x_i - y_i)^2,$$

whereby x_i are results of a 64 bit FP implementation and y_i results of an approximated execution.

Instead of using an objective metric like MSE to evaluate the quality, several models exist, i.e. a mathematical formulation, which return a numerical value for a subjective quality. Task performance analysis is an approach that correlates the image quality to the success of a following operation on the data of the image. For example, the image quality of a radiography is good enough if a radiologist is able to see the bone fraction. Therefore, the required accuracy of the results depends strongly on the task that should be fulfilled by the application. In the absence of a general model that gets knowledge about the needed accuracy, knowledge from domain experts is required. This paper does not provide hints to the required accuracy of results. Instead, we give a domain expert the possibility to easily adapt the accuracy of the results of his application to get a performance improvement as well as an energy-efficient execution.

In the following, we use an IEEE 754-based 64-bit floating point (FP64) unit as execution unit. Internal architecture registers can store values in 64 bit. Our approach is similar to extending the accuracy to 80 bits inside a FP unit like in x86 architectures. This approach reduces errors, because internal calculations are performed with higher accuracy. But our approach also further reduces the overhead for transferring data from FP registers to memory in terms of energy and transfer time. Furthermore, such an approach avoids having FP units for different data types, which results in so-called dark silicon because not all units can be used at the same time. We do not consider additionally required data like loop counters, though such counters can be represented as FP values.

Methods which we consider for converting a FP64 value into one with 32, 21 or 16 bits are summarized in Table 1. The first method (opcode 0, op0) converts a FP64 value to one with less bits according to the IEEE-754 standard. Due to the absence of a FP21 data type, we assume 1 sign bit, 5 bits for the exponent and 15 bits for the mantissa. We investigate if such a data type is useful. The approach is to pack three FP21 values into a 64 bit word before transferring it to memory. It is tolerable for some applications to set values smaller than 1 to 0. Therefore, we can increase the data range by a factor of 2 for all FP data types, because we do not have to consider negative superscripts. This is implemented for opcode 1. Opcode 2 and 3 convert a FP64 value to a fixed point representation QX.Y or Q.Y, where X is the number of bits for the integer and Y for the

Table 1. Methods for converting a FP64 value to one with lower accuracy. Signed numbers are represented by a sign bit.

Opcode (op)	Conversion method	Applications
0	IEEE-754 standard	High data range
1	Values < 1 set to zero	Small numbers not needed
2	Unsigned/signed QX.Y	Small data range
3	Unsigned/signed Q0.Y	Small data range (adapting the scale value improves accuracy)

fractional part. The conversion is achieved by dividing the FP64 value by an adaptable scale value. Lines named with FP16, FP21, and FP32 in the following figures are based on opcode 0, lines with Q8.8, Q8.13 and Q8.24 are based on opcode 2, and Q.16, Q.21, and Q.32 are based on opcode 3. Opcode 1 was not used for the first two benchmarks. Additionally, an approach that changes dynamically between 16, 21 and 32 bit conversion data types is considered and corresponding lines are named with dynamic data type (*dyn dt (th=j)*, where j specifies a threshold). This threshold can be set by a programmer and specifies the maximum absolute error allowed for a conversion into a certain data type. The hardware itself, i.e. the conversion unit, checks whether a conversion into a lower data type is below this given threshold j or not. The conversion unit uses the smallest data type for which the resulting conversion error is less than the given threshold. A programmer can adapt the threshold during run-time, in order to trade off accuracy against energy consumption. This is useful if some parts of an algorithm need more accurate calculations. A line named with *Full FP32* means that all internal operations are executed in FP32 and not FP64. Due to the absence of a FP16 execution unit in the test system, we do not consider a *Full FP16* execution.

The first benchmark is a 2D convolution

$$I^{'}[x,y] = I[x,y] * f[x,y] = \sum_{m=-k/2}^{k/2} \sum_{n=-k/2}^{k/2} I[x,y] \cdot f[m-x, n-y],$$

where I is the original image, f is a $k \times k$ 2D Gaussian filter and $I^{'}$ is the deblurred output image. The 2D convolution is executed up to 10 times, where the pixels of image $I^{'}$ are converted and used for following iterations. Pixel intensities are chosen randomly between 0 and 255 for a input image of size 1024×1024.

We specify for the first test that the values of a kernel are not preconverted before executing the first iteration and that the register set is large enough to store all kernel values (results see Fig. 1). Instead of a preconversion where values of a kernel are stored as converted values in the memory and have to be deconverted before transferring to the registers, kernel values are transferred as FP64 values to the FP register set. As mentioned above, image values are 8 bit integers, hence converting the image data is unnecessary. Therefore, the output of the first iteration is equal to a FP64 execution ($MSE = 0.0$). The MSE

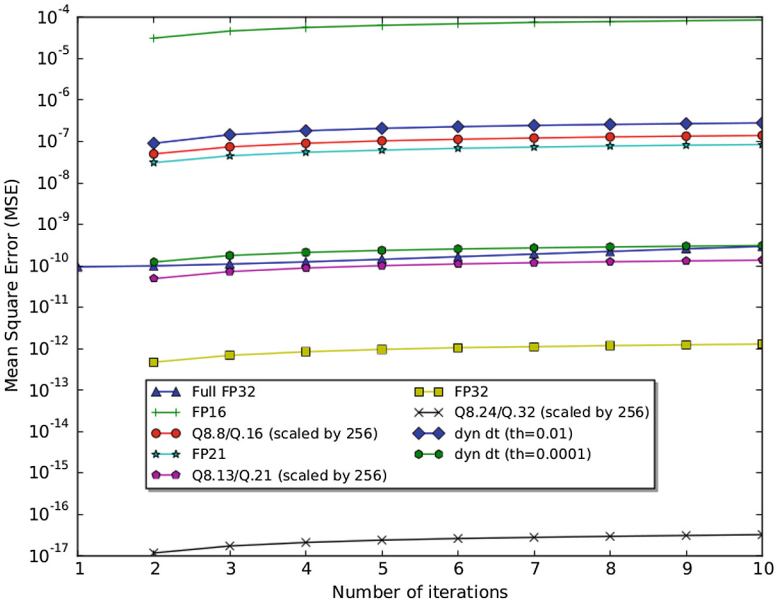

Fig. 1. MSE values of 2D convolutions using different conversion methods without preconversion. As input an image of size 1024×1024 with random values was used.

of a *Full FP32* execution is slightly higher than an execution using Q8.13 or Q.21, but about two orders of magnitude higher than our FP32 based approach and even seven orders of magnitude higher than Q8.24 and Q.32. Hence, our approach results in a much smaller MSE than a *Full FP32* execution, but has the same factor of data reduction. For the dynamic approach *dyn dt (th=0.01)*, the threshold 0.01 implies a usage of more FP16 data types, hence the MSE is closer to the MSE of the FP16 execution. More FP32 data types are used in the case of *dyn dt (th=0.0001)*, therefore the resulting MSE is closer to the MSE of the FP32 execution. For the second case, the kernel is preconverted into different conversion formats (see Fig. 2). The fixed point conversion formats (QY.X and Q.Y) are less accurate in terms of MSE than their FP relatives. According to the results in Figs. 1 and 2, higher accuracy is achieved in the first case. Hence, it turns out that for a higher accuracy of results frequently used data like the kernel values should be stored into the register set without conversion.

The second benchmark is a 2D `Richardson-Lucy Deconvolution`:

$$u^{(t+1)} = u^{(t)} \cdot \left(\frac{g}{u^{(t)} * K} * \widehat{K} \right),$$

where $u^{(t)}$ is the latent image, g the observed image, K a point spread function (PSF) and $\widehat{K}$ the flipped PSF. The Conversion method for the PSF values is fixed to FP16, FP21, and FP32 respectively (opcode 0). Conversion methods used for values of the intermediate results as well as the output image are shown in Fig. 3. Due to the higher data range of the algorithm, we have to adapt the

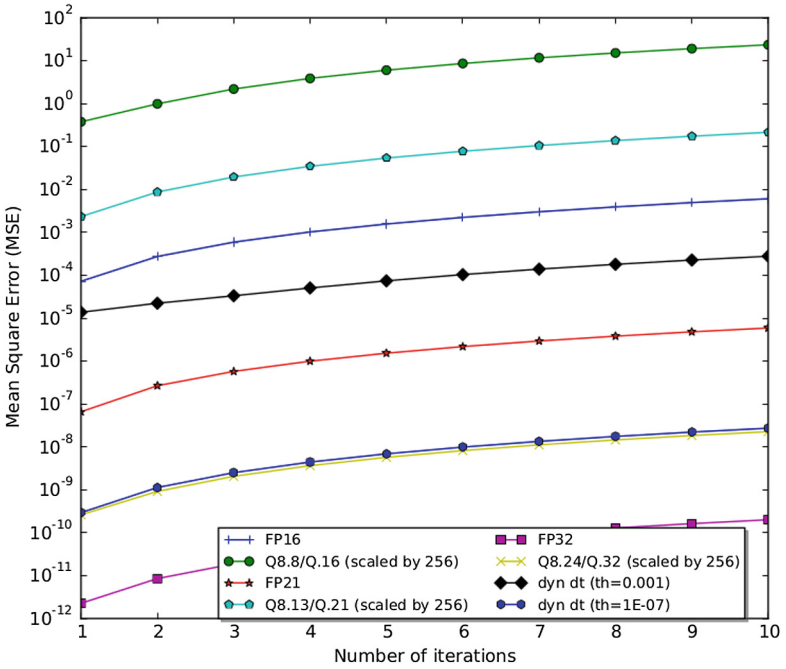

Fig. 2. MSE values of 2D convolutions using different conversion methods with pre-conversion. As input an image of size 1024×1024 with random values was used.

QX.Y conversion methods to Q.16, Q.21, Q.32, Q11.5, Q11.10, and Q11.21. Again the MSE of a *Full FP32* execution is higher than our FP32 approach. The FP approaches (FP16, FP21, FP32) have a smaller error than their fixed-point relatives (Q.16, Q.21, Q.32, Q11.5, Q11.10, and Q11.21). With the dynamic approach (*dyn dt (th=j)*) we can adapt the accuracy between FP16 and FP32 by adapting the threshold.

A 2D Fast Fourier Transformation[1] (FFT) is the last benchmark. As input we use an image with a size of 1024×1024, where the pixel intensities are chosen randomly between 0 and 255. A 2D FFT is done by row-wise 1D FFTs followed by column-wise 1D FFTs. The formula

$$X(n) = \sum_{k=0}^{N-1} x(k) e^{-jk2\pi \frac{n}{N}}, n = 0...N-1$$

is a forward FFT, where $x(k)$ is a complex series with N samples. We consider opcode 0, opcode 1, opcode 3, and the dynamic approach as conversion methods. The maximum absolute value during an execution has to be used as scale value for opcode 3. We also consider opcode 3 where we adapt the scale value by multiplying with 2 after each FFT butterfly beginning with 256.

[1] Code is based on the implementation of Paul Bourke http://paulbourke.net/miscellaneous/dft/.

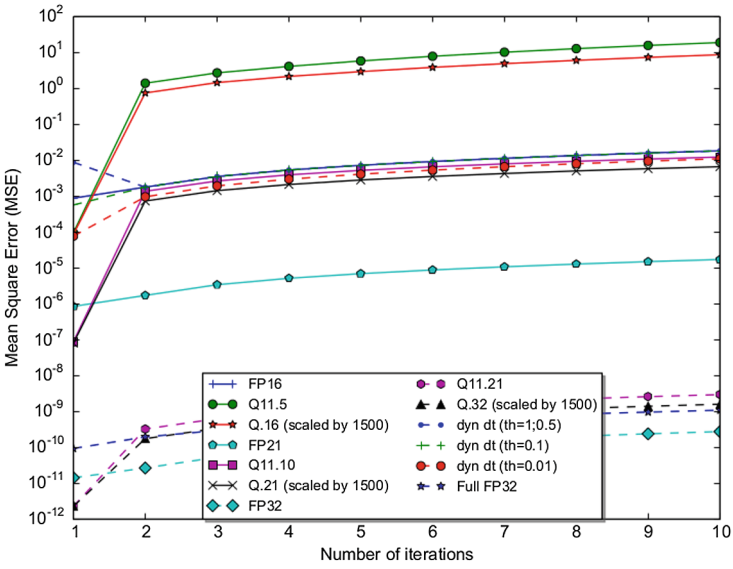

Fig. 3. MSE values for a 1024×1024 2D Richardson Lucy Deconvolution using different conversion methods for intermediate as well as the output image. The kernel values (psf) are converted using FP16, FP21 and FP32 (opcode 0), respectively.

The MSE and the factor of reduced data (RD) is shown in Table 2. Most of the static approaches using either 16 or 21 bits are assumedly not applicable in real applications. This is caused by the large codomain of the FFT algorithm. The only exception is opcode 1 using a 21 bit data type. A *Full FP32* execution has a higher MSE compared to FP32. Formats based on opcode 1 enable the reduction of the MSE compared to FP16 and FP21. Adapting the scale value for Q.32 results in a smaller MSE compared to FP32. Compared to a *Full FP32*, the dynamic approach with a threshold of 0.1 has roughly the same MSE, but reduces the amount of data by a factor of about 3. Hence, the dynamic approach is assumedly applicable for all used thresholds. We integrate the 2D FFT into an algorithm that reconstructs an image taken by a lens-free microscopy (see Table 3) [9]. The so-called spectral method is used to reconstruct holography images acquired by lens-free microscopy. Using FP32 yields to MSE of 0.0. The dynamic approach can reduce the MSE by two orders of magnitude while slightly decreasing the reduction of data transfers compared to FP16.

According to above benchmarks, it is sufficient for a conversion unit to have the conversion methods with opcode 0/1, because they achieve the best results in terms of MSE. It also turned out that Q.Y yields a lower MSE for algorithms with a small data range.

Table 2. MSE values for 2D FFTs using different conversion methods and an image of size 1024×1024 with random values.

	Full FP32		FP (op 0)		FP (op 1)		Q.Y (op 3)		Q.Y adapt. (op3)	
	MSE	RD	MSE	RD	MSE	RD	MSE	RD	MSE	RD
16 bit	-	-	∞	4	1.60E+05	4	2.00E+10	4	9.87E+07	4
21 bit	-	-	∞	3	2.84E+02	3	3.66E+09	3	9.53E+04	3
32 bit	1.63E+02	2	1.67E-04	2	-	2	9.02E+02	2	2.27E−05	2
Dynamic approach										
Threshhold	1E00		1E-01		1E-03		1E-05		1E-07	
	MSE	RD	MSE	RD	MSE	RD	MSE	RD	MSE	RD
	1.16E+03	3.65	1.85E+02	3.14	2.75E-02	2.34	1.67E-04	2.21	1.67E−04	2.21

Table 3. MSE values for the spectral method that reconstructs holography images using the above different FFT methods.

FP16 (Op 0)		FP21 (Op 0)		FP32 (Op 0)			
MSE	RD	MSE	RD	MSE	RD		
4.93E−02	4	4.78E−02	3	0.00E+00	2		
Dynamic approach							
Thesh = 1		Thesh = .5		Thresh = .1		Thresh = .01	
MSE	RD	MSE	RD	MSE	RD	MSE	RD
2.16E−04	3.94	1.63E−04	3.91	8.00E−05	3.77	1.40E−05	3.35

4 Measuring Energy Consumption

After the evaluation of promising conversion methods, we extract memory footprints of the 2D FFT benchmark that are created by using different FP conversion methods. Due to the fact that FP21 is not supported in current computing environments, we consider FP16, FP32 and FP64. The function `memcpy`, which is part of the standard C library, is used to transfer data according to the 2D FFT extracted memory footprints. We used `memcpy`, because we are only interested in the energy consumption of the data transfers. The used data is the result of a 2D FFT of a random image. Platforms used for the measurements are the Odriod-XU [8], the Parallela board [15], and the Myriad 1 development board [13]. An overview of the integrated compute units in each platform is given in Table 4. We selected these platforms because reducing energy consumption is especially important in embedded systems.

Table 4. Overview about the considered platforms.

Platform	Host processor [Technology]	Coprocessor [Technology]
Odriod-XU	Exsynos5 Octa (5410) [28 nm]	PowerVR SGX544MP3 GPU [28 nm]
Parallela	Zynq-7010 [28 nm]	E16G301 [65 nm]
Myriad 1	Leon 3 [65 nm]	SHAVEs [65 nm]

The Odriod-XU includes a Samsung Exsynos Octa processor that is based on the ARM big.LITTLE architecture and integrates a Cortex-A15 and a Cortex-A7. An operating system can switch between both clusters depending on the workload and the required performance, but the clusters cannot be used concurrently. We used a script to measure the energy consumption of the Odriod-XU. The script reads the values of different sensors for voltage, current, and power of the A7, A15 and the main memory. However, such a script influences the measurement. Therefore, we plan to use an external measuring setup using a GPIO pin of the Odriod-XU in the future. *POSIX Threads* (`Pthreads`) enable the usage of all four available cores. It is possible to decide on which cluster the benchmark is executed by specifying the frequency of the Exsynos5. We used an infinite loop around the `memcpy` calls to get a stable value of the electric power. We also measured the execution time in a different run.

The Parallela board, which includes a Dual-core ARM A9 (600 MHz) and a 16-core Epiphany (666 MHz), was developed for an energy efficient execution of high performance applications. We used the Odriod Smart Power, which is a deployable power supply to specify a fixed voltage and to measure the current as well as the electric power. The Odriod Smart Power measures the values for the entire board. For the first test, we used `Pthreads` again for starting two threads on the A9 and did not consider the energy consumption of the Epiphany. We measured the execution time in a second run and calculated the energy consumption. To measure the electric power of the A9 host processor together with the Epiphany, the A9 calls all 16 cores of the Epiphany. The 2D FFT data is transferred from the host memory to the local memory by Direct Memory Access (DMA). On each Epiphany core, `memcpy` transfers data to another region in the local memory of the core.

The Myriad I combines a Leon processor with eight Streaming Hybrid Architecture Vector Engines (SHAVEs). To measure the electric power of the Myriad 1, power cables are connected directly to supply the processor with electrical energy. A switched-mode power supply together with an ampere-meter enable to specify the voltage and to measure the current. To transfer the memory footprints inside the host memory, the function `memcpy` is used on the Leon processor. In the second test, eight DMA units, that are assigned to each SHAVEs, transfer data to the local memory. All SHAVES are used to transfer data inside the local memory according to the memory footprints.

The results of all setups are summarized in Table 5. Columns named with *Reduction* specify the factor of reduction in energy consumption compared to FP64. Energy consumptions are calculated by multiplying the execution time with the measured electric power. Our expectation is that a reduction of the amount of data, that has to be transferred, results in a reduction of the energy consumption of the same factor. According to the measurements, the expectation turned out to be true. The only exception is the dual-core A9 on the Parallela board, which is presumably caused by the underlying inefficient `memcpy` implementation.

Table 5. Measured energy consumption for a 2D FFT memory footprint.

	Size [MB]	Time [ms]	Power [W]	Energy [mJ]	Reduction	Time [ms]	Power [W]	Energy [mJ]	Reduction
		Odroid-XU							
		A15 & Global memory				A7 & Global memory			
FP64	16	5.503	4.849	26.682	1.0	11.226	0.702	7.881	1
FP32	8	2.588	4.875	12.618	2.1	5.104	0.724	3.695	2.1
FP16	4	1.185	4.962	5.880	4.5	2.503	0.729	1.825	4.3
		Parallela board							
		ARM A9				ARM A9 & Epiphany			
FP64	16	39.836	5.767	229.734	1.0	16513.133	6.179	102034.649	1.0
FP32	8	26.174	5.828	152.542	1.5	8257.387	6.026	49759.014	2.1
FP16	4	12.534	5.805	72.760	3.2	4135.034	4.034	16680.727	6.1
		Myriad I board							
		Leon 3				Leon 3 & SHAVES			
FP64	16	1826.062	0.190	346.952	1.0	168.571	0.499	84.117	1.0
FP32	8	910.723	0.187	170.305	2.0	83.152	0.482	40.079	2.1
FP16	4	456.516	0.188	85.825	4.0	40.162	0.478	19.197	4.4

5 Preliminary Design of a Conversion Unit

The Conversion Unit (CU) converts a FP64 value into a data type with fewer bits before storing it to memory (see Fig. 4). While reading the data, the value is converted back to FP64. The decision about the accuracy of the stored data is made statically using a conversion instruction. As a first step to realize such a CU, we described the *Converter* and *Deconverter* in Verilog. The *Converter* can convert a FP64 value to a FP16 or a FP32 value. This is realized by adapting the exponent part of the type and selecting the leading 10 or 23 bits of the mantissa. Instead of truncation, the round to nearest method is used for rounding. A FP64 value can also be converted to a FP format where no negative exponents are considered (opcode 1). If the exponent part of a FP64 value is smaller than the bias, the exponent part is set to 0. If the bias is equal to the exponent part, the exponent part is set to 1. 2 is added to the FP64 exponent for all other cases. The Q.16 and Q.32 data types are calculated by scaling the FP64 value. Such a scaling is only allowed for values $x_i = 2^i$, where $i = 0, 1, 2....$ Therefore, the scaling is

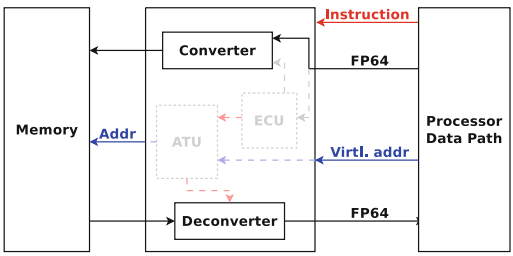

Fig. 4. Structure of the conversion unit.

performed by subtracting i from the exponent part. This avoids the use of a FP divider which increases the electric power as well as the latency of the unit. Each conversion is performed in a clock cycle. The *Deconverter* converts a conversion data type back to a FP64 value. Deconverting a FP value is always exact. As we restrict the factors to be a power of 2 for the scaling, the deconversion from a fixed point to the 64 bit floating point format is also exact. Using the Synopsis tools, we got an estimation about the electric power and the area of the units. We use the TSMC 28 nm HPM High Speed library for the synthesis. The results are summarized in Table 6. Compared to the measured values in Table 5, these units will not significantly increase the energy consumption. Supporting the dynamic approach will require an Address Translation Unit (ATU), as well as storing further information about the conversion method that was used for a specific data type. The ATU will avoid fragmentation inside memory. An Error Check Unit (ECU) decides, which conversion method is used depending on the error that will occur after the conversion. We will design and implement both units in future.

Table 6. Area and estimated power for the *Converter* and the *Deconverter*.

	500 MHz		600 MHz	
	Area	Power	Area	Power
	$[\mu m^2]$	$[\mu W]$	$[\mu m^2]$	$[\mu W]$
Converter	1958.681	336.900	2187.844	435.100
Deconverter	1199.812	188.300	1214.325	225.800

6 Conclusion

Memory accesses are expensive in terms of energy consumption and latency. Image processing applications can tolerate an execution on inaccurate hardware. Therefore, we presented an approach in this paper for a conversion unit (CU), that could be integrated in today's low power processors. Such a unit offers a static and a dynamic conversion of data types before the transfer to memory, which yields a reduction of the energy consumption for data transfers by a factor up to about 4. Using the dynamic approach, the accuracy of the results produced by a 2D FFT is improved by two orders of magnitude, while retaining the potential gain in the reduction of energy consumption. Dealing with different thresholds enables to trade off accuracy against energy consumption. In the future, we will integrate the implemented units into an existing processor design. Furthermore, we want to implement a CU that also supports the dynamic approach. Additionally, we will also design a memory architecture that enables an efficient transfer of data types for the dynamic approach.

Acknowledgements. The work was mainly performed during a HiPEAC internship at Movidius, Ireland. Special thanks to Fergal Connor and David Moloney. Additionally, this work was also funded by the Klaus Tschira Foundation.

References

1. Alvarez, C., Corbal, J., Valero, M.: Fuzzy memoization for floating-point multimedia applications. IEEE Trans. Comput. **54**(7), 922–927 (2005)
2. Avinash, L., Enz, C.C., Palem, K.V., Piguet, C.: Designing energy-efficient arithmetic operators using inexact computing. J. Low Power Electron. **9**(1), 141–153 (2013)
3. Baek, W., Chilimbi, T.M.: Green: a framework for supporting energy-conscious programming using controlled approximation. In: ACM Sigplan Notices, vol. 45, pp. 198–209. ACM (2010)
4. Borkar, S., Chien, A.A.: The future of microprocessors. Commun. ACM **54**(5), 67–77 (2011)
5. Chippa, V., Chakradhar, S., Roy, K., Raghunathan, A.: Analysis and characterization of inherent application resilience for approximate computing. In: DAC, pp. 1–9, May 2013
6. Citron, D., Feitelson, D.G.: Hardware Memoization of Mathematical and Trigonometric Functions. Hebrew University of Jerusalem, Technical report (2000)
7. Esmaeilzadeh, H., Sampson, A., Ceze, L., Burger, D.: Neural acceleration for general-purpose approximate programs. In: MICRO, pp. 449–460 (2012)
8. Hardkernel.: Odriod-XU. http://odroid.com/dokuwiki/doku.php?id=en:odroid-xu. Accessed 03 May 2015
9. Hennelly, B., Kelly, D., Pandey, N., Monaghan, D.: Zooming algorithms for digital holography. J. Phys: Conf. Ser. **206**(1), 012027 (2010)
10. Horowitz, M.: Computing energy problem: and what we can do about it. In: Keynote, International Solid-State Circuits Conference, February 2014. https://www.futurearchs.org/sites/default/files/horowitz-ComputingEnergyISSCC.pdf. Accessed 03 May 2015
11. Liu, S., Pattabiraman, K., Moscibroda, T., Zorn, B.: Flikker: saving DRAM refresh-power through critical data partitioning. In: ASPLOS, March 2011
12. Lucas, J., Alvarez-Mesa, M., Andersch, M., Juurlink, B.: Sparkk: Quality-scalable approximate storage in DRAM. In: The Memory Forum, June 2014
13. Movidius Ltd.: Myriad 1. http://www.hotchips.org/wp-content/uploads/hc_archives/hc23/HC23.19.8-Video/HC23.19.811-1TOPS-Media-Moloney-Movidius.pdf. Accessed 03 May 2015
14. Nelson, J., Sampson, A., Ceze, L.: Dense approximate storage in phase-change memory. In: ASPLOS (2011)
15. Parallela Project: Parallela board. http://www.parallella.org/board/. Accessed 03 May 2015
16. Samadi, M., Lee, J., Jamshidi, D.A., Hormati, A., Mahlke, S.: SAGE: self-tuning approximation for graphics engines. In: MICRO, pp. 13–24 (2013)
17. Sampson, A., Dietl, W., Fortuna, E., Gnanapragasam, D., Ceze, L., Grossman, D.: Enerj: approximate data types for safe and general low-power computation. In: ACM SIGPLAN Notices, vol. 46, pp. 164–174. ACM (2011)
18. Sampson, A., Nelson, J., Strauss, K., Ceze, L.: Approximate Storage in Solid-state Memories. In: Proceedings of the MICRO, MICRO-46, pp. 25–36. ACM, New York (2013)
19. San Miguel, J., Enright Jerger, N.: Load value approximation: approaching the ideal memory access latency. In: WACAS (2014)
20. Sardashti, S., Wood, D.A.: Decoupled compressed cache: exploiting spatial locality for energy-optimized compressed caching. In: MICRO, pp. 62–73 (2013)