ProACt: A Processor for High Performance On-demand Approximate Computing

Arun Chandrasekharan[1] Daniel Große[1,2] Rolf Drechsler[1,2]
[1]Group of Computer Architecture, University of Bremen, Germany
[2]Cyber Physical Systems, DFKI GmbH, Bremen, Germany
{arun,grosse,drechsle}@cs.uni-bremen.de

ABSTRACT

We present ProACt, a *Processor for high performance on-demand Approximate Computing*. ProACt is a general purpose processor that can dynamically approximate floating point operations. In ProACt, the approximations are done in hardware, but the software can directly enable, disable or control the accuracy of approximations. In addition, ProACt offers a complete open-source development framework consisting of a hardware processor and associated software tool chain. ProACt uses functional approximations and is proven in FPGA. Further, we show performance improvements of about 30% on case studies in image processing and scientific computing.

1. INTRODUCTION

Approximate computing can deliver significant performance benefits (e.g. in speed or energy) over conventional computing by relaxing the precision of results. In order to harness the full potential of approximations, both hardware and software need to work in tandem [18, 16, 10]. However, in a general application, there are program segments that can be approximated and others which should not be approximated. In addition, it has become good strategy to perform approximation in certain situations, e.g. when the battery goes low. Also, decisions such as the duration and degree of approximations may depend on external factors and input data set of the application, which may not be fully known at system design time. All these calls for an on-demand, rapidly switchable hardware approximation scheme that can be fully controlled by software. This software-control may originate from the application itself or from a supervisory software like the operating system [12, 6].

In this work, we present ProACt, a *Processor for high performance on-demand Approximate Computing* which fullfills all the above requirements. The core idea of ProACt is to functionally approximate floating point operations using previously computed results from a cache thereby relaxing the requirement of having the exact identical input values for the current floating point operation. To enable on-demand approximation we add a *custom instruction* to the *Instruction Set Architecture* (ISA) of the used processor which essentially adjusts the input data for cache look-up and by this controls the approximation behavior. Overall we have devised a ProACt development framework. Our framework consists of the extended hardware processor[1] and the software tool chain (cross-compiler, linker etc) to build a complete system for on-demand approximate computing.

In ProACt we target floating point operations for approximation. These operations are essential to modern multimedia and signal processing engines, and form the basis of several industry standards [2]. However, complex floating point operations such as division and square-root are computationally expensive, and usually span over multiple clock cycles [8, 11]. With ProACt we aim to reduce the number of clock cycles spent on floating point operations with our on-demand approximation scheme. Therefore, these operations and results are stored in an approximation look-up table. This look-up table is checked first before executing a new floating point operation. In this step, *approximation masks* are applied to the operands before the cache look-up. These masks are set by the new custom approximation control instruction (assembly mnemonic `SXL`[2]), and they define the degree of approximation. The `SXL` instruction also controls additional flags for approximation behavior such as enable/disable the cache look-up table. Thus ProACt functions as a normal processor when the approximation look-up table is disabled.

The custom approximation control instruction `SXL` is designed as an *immediate* instruction resulting in very little software overhead and run-time complexity. `SXL` is fully resolved at the *decode* stage of the processor pipeline, and therefore situations such as *control* and *data hazards* [15] are reduced to a minimum. This is significant in a multi-threaded, multi-process execution environment, and in atomic operations. Being able to rapidly switch between approximation and normal modes is an important factor affecting the throughput of the processor in such contexts.

We have prototyped ProACt in a Xlinix FPGA. Experimental results show the benefits in terms of speed for different applications.

To summarize, we make the following key contributions:

- High performance general purpose processor hardware for on-demand floating point approximation
- Dynamic control of approximation from software
- Complete development framework including processor hardware and software tool set

[1]ProACt is based on RISC-V [17], a state-of-the-art open-source 64 bit RISC architecture
[2]`SXL` stands for *Set approXimation Level.*

GLSVLSI '17, May 10-12, 2017, Banff, AB, Canada
 ISBN 978-1-4503-4972-7/17/05…$15.00
DOI: http://dx.doi.org/10.1145/3060403.3060415

The remainder of this paper is structured as follows. In Section 2 related work is discussed. The ProACt architecture is introduced in Section 3 and details on ProACt evaluation is provided in Section 4. Finally, the paper is concluded in Section 5.

2. RELATED WORK

There are several works in approximate computing for hardware and software approximations that range from dual voltage hardware units, to dedicated programming languages (see for e.g., [18], [9]). In particular [10] explains a generalized set of architectural requirements for an approximate ISA with a dedicated programming language. The authors discuss approximation variants of individual instructions that can be targeted to a dual voltage hardware showing the benefits, though only in simulation. Similarly significant progress has been made on custom designed approximation processors such as [14], targeted for one particular area of application. In contrast, ProACt is a general purpose processor with on-demand approximation capabilities. Approximations can be even completely turned off in ProACt. Moreover, our work is focused on functional approximations, rather than schemes like dynamic voltage/frequency scaling (DVFS), that involve fabrication aspects, and are tedious to design, implement, test and characterize. We refer to [18] for a detailed overview of such techniques.

Several schemes have been presented so far to approximate a floating point unit. Hardware look-up tables are used in [8] to accelerate multi-media processing. This technique called *memoing* has been shown to be very effective. Work of [3] extends this further to *fuzzy memoization* using approximations. However, none of these approaches use custom instructions for software controlled approximations and are limited to the scope of the application it is designed for. In addition, they do not offer direct support to treat the critical program segments differently. An example from image processing is the JPEG image header which contains critical information, and should not be approximated, whereas the pixel data content of the image is relatively safe to approximate.

A detailed overview on the ProACt architecture is given next.

3. ProACt ARCHITECTURE

The ProACt system overview is shown in Fig. 1 on the left hand side. As can be seen it consists of the processor hardware and the software units working together to achieve approximation in computations. To operate the approximations in hardware we have added the *Approximate Floating Point Unit* (AFPU) (a zoom is given on the right hand side of Fig. 1 and its details are described in the next section). In normal mode (i.e. approximations disabled), ProACt floating point results are IEEE 754 compliant.

The AFPU is explained next, followed by the ISA extensions for approximation. The ISA and the assembly instructions are the interface between ProACt hardware and the applications targeting ProACt. Other details on the processor architecture and compiler framework is deferred to the end of this section.

3.1 Approximate Floating Point Unit (AFPU)

The AFPU in ProACt consists of an approximation look-up table, a pipelined *Floating Point Unit* (FPU) and an approximation control logic (see Fig. 1, right hand side). The central approach used in this work is that the results of the FPU are stored first, and further operations are checked in this look-up table, before invoking the FPU for subsequent computation. The input arguments to the FPU are checked in the look-up table and when a match is found, the results from the table are fed to the output, bypassing the entire FPU. The FPU will process only those operations which do not have results in the table. This look-up mechanism is much faster, resulting in significant savings in clock cycles. Approximation masks are applied to the operands before checking the look-up table. Thus, the accuracy of the results can be traded-off using these masks. These approximation masks are set by the software (via the custom approximation control instruction `SXL`, details see next section) and varies in precision. The mask value (or alternatively called *approximation level*) denotes the number of bits to be masked from the LSB, before checking for an entry in the look-up table.

These approximation levels are fully configurable and provide a bit level granularity to the approximations introduced. The look-up table stores the last N floating point operations in a round robin fashion. Software controls the approximation mask, look-up table mechanism, and all these units can be optionally turned off. This is achieved by extending the ISA with a custom instruction for approximation. This ISA extension is presented in the next section.

3.2 ProACt ISA Extension

The software compiler relies on the ISA to transform a program to a binary executable. Hence the ISA is extended with a single assembly instruction `SXL` (*Set approXimation Level*) for the software control of approximations. `SXL` is designed as an immediate instruction that takes a 11-bit immediate value. The LSB, when set to '1' enables the hardware approximations. The remaining bits are used to set the approximation level and other special flags. The software can also enable the approximation unit but set the level as 0. Here it will simply act as a result caching and look-up mechanism without any approximation.

3.3 ProACt Processor Architecture

The ProACt processor is based on RISC-V architecture [17]. RISC-V is a modern, general purpose, high quality, instruction set architecture based on RISC principles. Besides, RISC-V is distributed open source; thus making it well suited for academic and research work. Moreover several implementations of this ISA are publicly available. ProACt is based on one such implementation called Rocket chip [5]. ProACt uses 64-bit addressing scheme with 32 general purpose registers. The pipeline used is a 64-bit, 5-stage, in-order pipeline. As shown in Fig. 1, the processor has separate L1 and L2 cache memories.

3.4 Compiler Framework and System Libraries

The ProACt compiler framework consists of cross-compiler, linker, and associated tools based on *GNU Compiler Collection* (GCC) [1].

In addition to this compiler framework, a set of system library routines and macros are provided with the distribution for the convenient use of on-demand approximations in software programming. The need for approximations could be due to a variety of reasons. Architectural choice, external and run-time factors, nature of the algorithm, quick iterations for initial results, power savings due to approximations, are only some of these. Moreover, the impact of approximations may depends on the nature of input data, as can be seen from the experimental results in Section 4.2. Thus, by providing a compact set of software routines all these scenarios are addressed.

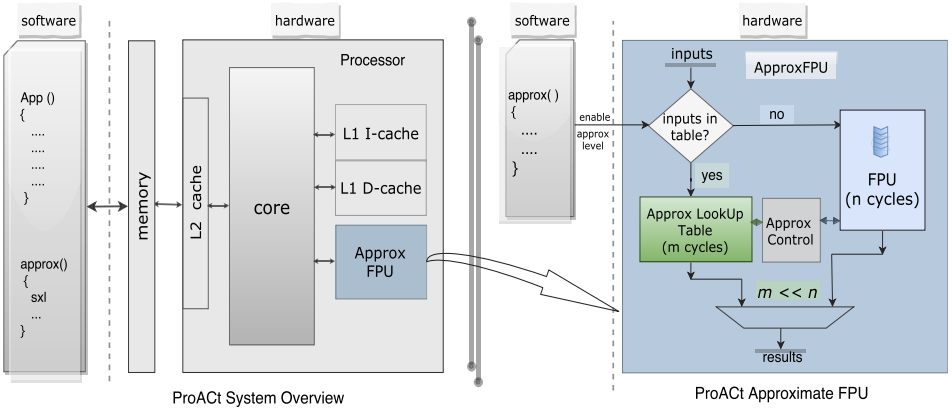


Fig 1: ProACt system overview: SW controlled HW approximations

Table 1: ProACt FPGA hardware prototype details

Platform and Tools:	
FPGA: Xilinx Zynq-XC7Z020	
Prototype Board: Digilent ZedBoard	
FPGA Tools: Xilinx Vivado 2016.2	
Hardware Implementation:	
Frequency:	100MHz⋆
LUTs:	42527
Power/MHz:	18.66 mW/MHz

⋆ provided by ZedBoard

4. ProACt EVALUATION

In this section we present the experimental evaluation of ProACt. First, we give a brief overview of the ProACt FPGA implementation. The experimental results for different applications using ProACt hardware are presented afterwards.

4.1 FPGA Implementation Details

We have implemented a ProACt prototype in hardware using Xilinx Zynq FPGA. We use a fixed 128 entry approximation look-up table throughout our experiments. The table size is set to 128 mainly to utilize the FPGA resources for hardware implementation efficiently. In general, as the table size increases, the hit-rate and thereby the speed-up resulting from approximations increases. The general impact has already been observed by others [3, 8]. Hence, due to page limitation we do not repeat the underlying experiments here. The hardware details of the evaluation prototype is given in Table 1. This FPGA prototype implementation is the basis of all experimental results and benchmarks presented in the subsequent sections.

4.2 Experimental Results

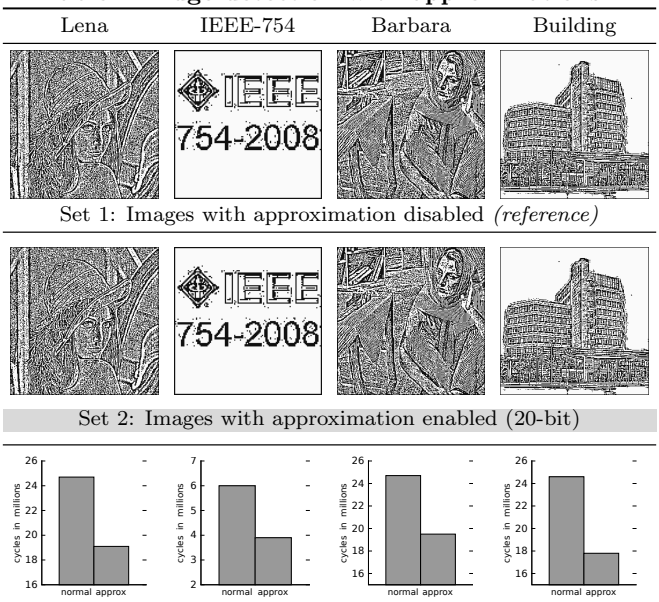
We have used two different categories of applications to evaluate ProACt. The first one is an image processing application and the second set consists of mathematical functions from scientific computing. These experiments evaluate the performance of ProACt and also tests the on-demand approximation switching feature. All applications are written in C, compiled using ProACt GCC compiler and executed in the ProACt FPGA hardware prototype. All programs have a computationally expensive *core algorithm* which is the focus of evaluation. A top level supervisor program controls the approximations and invokes this core algorithm. Thus, the same algorithm is run with different approximation schemes set by the supervisor program. The scope of approximations in this experimental evaluation is restricted to floating point division only. We next discuss the experiments performed in the two categories:

4.2.1 Image edge detection

Table 2 shows the results from a case study on edge detection [13] using ProACt. Image processing applications are very suitable for approximations since there is an inherent human perceptual limitation in processing image details.

The top row of images (Set 1) in Table 2 are generated with approximations disabled by the supervisory program. The middle row (Set 2) are generated with approximations

Table 2: Edge detection with approximations



Set 1: Images with approximation disabled *(reference)*

Set 2: Images with approximation enabled (20-bit)

speed-up: 23% speed-up: 35% speed-up: 21% speed-up: 28%

Hardware cycles taken and speed-up with approximations

Images generated from ProACt FPGA hardware
Set 1 reference images are with normal processing
Set 2 images are with approximation enabled (20-bit)

enabled, and the last row shows bar plots for the hardware cycles taken by the core algorithm, along with the speed-up obtained. As evident from Table 2, ProACt is able to generate images with negligible loss of quality with performance improvements averaging more than 25%. Furthermore, the speed-up is much higher in images with more uniform pixels, as evident from the second image, *IEEE-754* (35% faster). This has to be expected since such a sparse input data set has higher chances of computation reuse.

4.2.2 Scientific functions

We have also evaluated ProACt with several scientific functions as the core algorithm. A subset of the results is given in Table 3. The first row (non-shaded) in each set is the high accuracy version of the algorithms, i.e. with hardware approximations disabled. The second row (shaded)

Table 3: Math functions with approximations

appx level (a)	cycles n (b)	$\|\Delta y\|$ x10^{-3} (c)	speed up% (d)	appx level (a)	cycles n (b)	$\|\Delta y\|$ x10^{-3} (c)	speed up% (d)
$y = sinh(x)$				$y = sinh^{-1}(x)$			
-1	11083	0.00	0.00	-1	76899	0.00	0.00
20	7791	0.15	29.70	20	72506	3.91	5.71
$y = cosh(x)$				$y = cosh^{-1}(x)$			
-1	10820	0.00	0.00	-1	78616	0.00	0.00
20	7501	0.14	30.67	20	73843	2.14	6.07
$y = tanh(x)$				$y = tanh^{-1}(x)$			
-1	10848	0.00	0.00	-1	7698	0.00	0.00
20	7505	0.10	30.82	20	6135	0.93	20.30

Functions $y = f(x)$ evaluated in ProACt FPGA hardware

Shaded rows are results with approximations enabled.

(a) Approximation level (set with `SXL` instruction)
-1: high accurate result (approximation fully disabled)
20: 20-bit approximation in float division.

(b) Number of machine cycles (n) taken for computation.

(c) Accuracy, $|\Delta y| = |y_{-1} - y_{20}| \times 10^{-3}$

(d) Speed up from approximation $= \frac{n_{-1} - n_{20}}{n_{-1}} \times 100\ \%$

shows the result with approximations turned on. The absolute value of the deviation of the results ($|\Delta y|$) with approximation from the high accurate version is given in the third column, along with respective speed-up obtained in fourth column (column d).

The speed-up (d column) and the accuracy loss (c column) in Table 3 show that on-demand approximations can significantly reduce the computation load with an acceptable loss in accuracy. The accuracy loss is only in the 4^{th} decimal place, or lower in all the experiments. Functions such as *cosh* and *tanh* can be approximated very well with a speed-up more than 30% with an accuracy of 0.0001

4.3 Discussion

There is a substantial reduction in the run-time and machine cycles in ProACt with approximations enabled. About 25% speed-up, on an average, is obtained with approximations in the image edge detection applications shown in Table 2. This is also reflected in Table 3, where some of the mathematical functions are more than 30% faster. The dynamic power consumption of the system also decreases when approximations are enabled since this speed-up directly corresponds to a reduction in the whole hardware activity.

The throughput and performance of a system, taken as a whole, is largely governed by Amdahl's law [4]. I/O read and write are the main performance bottleneck in our ProACt system prototype. Consequently we have implemented all the algorithms to read the input data all at once, process, and then write out the result in a subsequent step. To make a fair comparison, the costly I/O read-write steps are not accounted in the reported speed-up.

It is worth mentioning that all experimental results presented in Tables 2, 3 are compiled with GCC flag `-ffast-math` [1]. This flag enables multiple compiler level optimizations such as constant folding for floating point computations, rather than offloading every operation to the FPU hardware. Thus, it potentially optimizes the floating point operations while generating the software binary itself, and the speed up due to ProACt approximation adds on top of that.

5. CONCLUSIONS

In this work, we presented ProACt - a Processor for high performance on-demand Approximate Computing. ProACt is FPGA proven, and comes with a complete open-source development framework. Further, we have demonstrated the advantages of ProACt using image processing and scientific computing programs, that show up to 30% performance improvement with dynamic approximation control. In future we aim to extend ProACt with formal methods such as [7] to guarantee the bounds of approximation.

Acknowledgments:. This work was supported by the German Research Foundation (DFG) in the project MANIAC (DR 287/29-1), by the University of Bremen's graduate school SyDe, funded by the German Excellence Initiative and by the German Academic Exchange Service (DAAD).

ProACt repository: *https://gitlab.com/arunc/proact-processor*

6. REFERENCES

[1] GCC - the GNU Compiler Collection 6.1, 2016.
[2] International Telecommunication Union, 2016.
[3] C. Alvarez, J. Corbal, and M. Valero. Fuzzy memoization for floating-point multimedia applications. *IEEE Trans. Comput.*, 54(7), 2005.
[4] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities, reprinted from the AFIPS conf. *IEEE Solid-State Circuits Society Newsletter*, 12(3):19–20, 2007.
[5] K. Asanovic et al. The Rocket Chip Generator. *Technical Report UCB/EECS-2016-17, EECS Department, University of California, Berkeley*, 2016.
[6] M. Carbin, S. Misailovic, and M. C. Rinard. Verifying quantitative reliability for programs that execute on unreliable hardware. In *OOPSLA*, pages 33–52, 2013.
[7] A. Chandrasekharan, M. Soeken, D. Große, and R. Drechsler. Precise error determination of approximated components in sequential circuits with model checking. In *DAC*, 2016.
[8] D. Citron, D. Feitelson, and L. Rudolph. Accelerating multi-media processing by implementing memoing in multiplication and division units. In *ASPLOS*, ASPLOS VIII, pages 252–261. ACM, 1998.
[9] J. Constantin, L. Wang, G. Karakonstantis, A. Chattopadhyay, and A. Burg. Exploiting dynamic timing margins in microprocessors for frequency-over-scaling with instruction-based clock adjustment. In *DATE*, pages 381–386, 2015.
[10] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Architecture support for disciplined approximate programming. *SIGPLAN Not.*, 47(4):301–312, 2012.
[11] J. R. Hauser. Handling floating-point exceptions in numeric programs. *ACM Trans. Program. Lang. Syst.*, 18(2):139–174, 1996.
[12] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard. Dynamic knobs for responsive power-aware computing. In *ACM SIGPLAN Notices*, volume 46, pages 199–212. ACM, 2011.
[13] R. P. Johnson. Contrast based edge detection. *Pattern Recogn.*, 23(3-4):311–318, 1990.
[14] G. Karakonstantis, A. Sankaranarayanan, M. M. Sabry, D. Atienza, and A. Burg. A quality-scalable and energy-efficient approach for spectral analysis of heart rate variability. In *DATE*, pages 1–6, 2014.
[15] D. A. Patterson and J. L. Hennessy. *Computer Organization and Design, Fourth Edition: The Hardware/Software Interface*. Morgan Kaufmann.
[16] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. Enerj: Approximate data types for safe and general low-power computation. In *PLDI*, PLDI '11, pages 164–174. ACM, 2011.
[17] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovic. The RISC-V instruction set manual, volume i: Base user-level ISA. *EECS Department, UC Berkeley, Tech. Rep. UCB/EECS-2011-62*, 2011.
[18] Q. Xu, T. Mytkowicz, and N. S. Kim. Approximate computing: A survey. *IEEE Design Test*, 33(1):8–22, 2016.