BRINGING HIGH-PERFORMANCE RECONFIGURABLE COMPUTING TO EXACT COMPUTATIONS

Esam El-Araby, Ivan Gonzalez, and Tarek El-Ghazawi

NSF Center for High-Performance Reconfigurable Computing (CHREC),
ECE Department, The George Washington University
801 22nd Street NW, Washington, DC 20052, USA
{esam, ivangm, tarek}@gwu.edu

ABSTRACT

Numerical non-robustness is a recurring phenomenon in scientific computing. It is primarily caused by numerical errors arising because of fixed-precision arithmetic in integer and/or floating-point computations. Exact computation, based on arbitrary-precision arithmetic, has been developed over the last decade as an emerging numerical computation paradigm in response to this problem of numerical non-robustness. Exact arithmetic, specifically arbitrary-precision arithmetic, has been traditionally implemented using efficient software libraries such as GNU Multi-Precision (GMP). However, this results in a slower arithmetic performance when compared to fixed-precision arithmetic. In this paper we present a first effort, to the best of our knowledge, of reconfigurable hardware support for arbitrary-precision arithmetic. The proposed hardware architectures are based on virtual convolution sche[1]duling which is derived from a formal representation of the problem. Targeting high performance and efficiency, dynamic (non-linear) pipelines techniques were exploited to eliminate the effects of deeply-pipelined operators. Referenced to GMP, our experiments showed promising results.

1. INTRODUCTION

Numerical non-robustness is a recurring phenomenon in scientific computing. It is primarily caused by numerical errors arising because of fixed-precision arithmetic. Most of these errors can be considered harmless, but occasionally there are "catastrophic" errors in the computation that cause non-robust behavior such as crashing of the program and/or infinite loops [1]. In response to this problem, exact computation, based on exact/arbitrary-precision arithmetic, was first introduced in 1995 by Yap and Dub'e [2] as an emerging numerical computation paradigm. Arbitrary-precision arithmetic, also called bignum arithmetic, allows computer programs to perform arithmetic operations in integer and/or floating-point representations with an arbitrary number of digits of precision. This is typically limited only by the available memory of the host system [3, 4]. Exact/arbitrary-precision arithmetic is applied in most problems in computational sciences and engineering such as public-key cryptography, computational metrology and Coordinate Measuring Machines (CMMs), computation of fundamental mathematical constants such as π to millions of digits, rendering fractal images, computational geometry, geometric editing and modeling, and Constraint Logic Programming (CLP) languages [1, 2, 3, 4, 5].

Arbitrary-precision arithmetic is mostly implemented in software, perhaps embedded into a computer language. Over the last decade, a number of big number software packages have been developed. These include the GNU Multi-Precision (GMP) library, CLN, LEDA, Java.math, BigFloat, BigDigits, Crypto++, etc. [4, 5]. In addition, there exist stand-alone application software/languages such as PARI/GP, Mathematica, Maple, Macsyma, dc programming language, REXX programming language, etc. [4].

Arbitrary-precision numbers are often stored as large-variable-length arrays of digits in some base related to the system word-length. Because of this, arithmetic performance is slower compared to fixed-precision arithmetic which is closely related to the size of the processor internal registers [2]. There have been some attempts for hardware implementations. However, those attempts usually amounted to specialized hardware for small-size discrete multi-precision and/or large-size fixed-precision [6, 7, 8, 9, 10] integer arithmetic rather than real arbitrary-precision arithmetic.

In this paper we present a first effort of reconfigurable hardware support for arbitrary-precision arithmetic. We propose the use of High-Performance Reconfigurable Computers (HPRCs) as a promising candidate for arbitrary-precision arithmetic. These platforms are characterized by higher performance and processing power compared to conventional platforms [11, 12, 13] as well as by higher flexibility, i.e. Run-Time-Reconfigurability (RTR), compared to ASIC solutions. The SRC-6 is an example of this category of computers [14] and is used here for this purpose.

The proposed hardware architectures, which will be derived from a formal representation of the problem, are

This work was supported in part by the I/UCRC Program of the National Science Foundation under the NSF Center for High-Performance Reconfigurable Computing (CHREC).

based on virtual convolution scheduling. Dynamic (non-linear) pipelines techniques will be exploited to eliminate the effects of deeply-pipelined reduction operators. The efficiency of the proposed hardware will also be analyzed. The experimental work will be verified for both correctness and performance in reference to the GMP library.

2. PROBLEM FORMULATION

Exact arithmetic supports exact computation with the four basic arithmetic operations (+,-,x,÷) over the rational field $\boldsymbol{Q}$ [1, 2, 3, 4, 5]. Therefore, within the context of hardware support, the problem of exact computation is reduced to implementing the four basic arithmetic operations with arbitrary-precision of operands.

Generally, the asymptotic computational complexity of each operation depends on the bit length of operands [5, 15], see Table. 1. Therefore, the main challenge of arbitrary-precision arithmetic on hardware is the physical/spatial limitations of the hardware. In other words, the problem can be formulated as: given a fixed-precision arithmetic unit, e.g. p-digit by p-digit multiplier, it is required to realize an arbitrary-precision arithmetic unit, e.g. arbitrary large-variable-size m_1-digit by m_2-digit multiplier. In achieving this objective, our approach is based on leveraging previous work and concepts that were introduced for solving similar problems. For example, Tredennick [16] proposed architectural solutions for variable-length byte string processing. Similarly, Olariu [17] formally analyzed and proposed solutions for the problem of sorting arbitrary large number of items using a sorting network of small fixed I/O size. Finally, ElGindy [18] investigated the problem of mapping recursive algorithms on reconfigurable hardware.

Table 1. Computational Complexity of Arithmetic Operations [15]

Operation	Input	Output	Algorithm	Complexity
Addition	Two n-digit numbers	One $(n+1)$-digit number	Basecase/Schoolbook	$O(n)$
Subtraction	Two n-digit numbers	One $(n+1)$-digit number	Basecase/Schoolbook	$O(n)$
Multiplication	Two n-digit numbers	One $2n$-digit number	Basecase/Schoolbook	$O(n^2)$
			Karatsuba	$O(n^{1.585})$
			3-way Toom-Cook	$O(n^{1.465})$
			k-way Toom-Cook	$O(n^{1+\varepsilon})$, $\varepsilon > 0$
			Mixed-level Toom-Cook	$O(n(\log n)2^{\sqrt{(2\log n)}})$
			Schönhage-Strassen	$O(n(\log n)(\log\log n))$
Note: The complexity of multiplication will be referred to as M(n) below				
Division	Two n-digit numbers	One n-digit number	Basecase/Schoolbook	$O(n^2)$
			Newton's method	$M(n)$
Square root	One n-digit number	One n-digit number	Newton's method	$M(n)$
Polynomial evaluation	n fixed-size polynomial coefficients	One fixed-size	Horner's method	$O(n)$
			Direct evaluation	$O(n)$

Our formal representation of the problem will consider only the multiplication operation. This is based on the fact that multiplication is a core operation from which the other basic arithmetic operations can be easily derived, e.g. division as a multiplication using invariant integers [19, 20]. We will show how our proposed arithmetic unit can perform arbitrary-precision addition, subtraction, multiplication, as well as arbitrary-length convolution operations. Division is reserved for future support and will not be included in this work only because of time constraints. We decided to investigate the Basecase/Schoolbook algorithm, see Table 1. Although this algorithm is not the fastest algorithm $O(n^2)$, it is the simplest and most straight forward algorithm with the least overhead. In addition, this algorithm is usually the starting point for almost all available software implementations of arbitrary-precision arithmetic. For example, in the case of GMP, the Basecase algorithm is used up to a pre-determined operand size threshold, 3000-10,000 bit length depending on the underlying microprocessor architecture, beyond which the software adaptively switches to a faster algorithm [20, 21]. Our emphasis was on investigating the feasibility and/or the potential of reconfigurable hardware within the domain of exact computations. Therefore, our performance comparisons with GMP will be maintained up to this threshold of data size. We preserve the adaptive implementation of the faster algorithms for future investigation. This will be supported by the native RTR adaptability of HPRCs.

3. APPROACH AND ARCHITECTURES

3.1. Formal Problem Representation

An arbitrary-precision m-digit number in arbitrary numeric base r can be represented by:

$$A = a_0 + a_1 r + a_2 r^2 + \ \ldots\ldots\ \ldots\ldots + a_{m-1} r^{m-1}$$

$$A = \sum_{j=0}^{m-1} a_j r^j, \quad 0 \le a_j < r \tag{1}$$

It can also be interpreted as an n-digit number with base r^p, where p is dependent on the underlying hardware word-length, e.g. 32-bit or 64-bit. This is represented by equation (2) as follows:

$$A = \sum_{i=0}^{n-1} \sum_{j=ip}^{(i+1)p-1} a_j r^j = \sum_{i=0}^{n-1} \sum_{k=0}^{p-1} a_{k+ip} r^{k+ip} = \sum_{i=0}^{n-1} \left[\sum_{k=0}^{p-1} a_{k+ip} r^k \right] r^{ip} = \sum_{i=0}^{n-1} A_i r^{ip}$$

$$\text{where } A_i = \sum_{k=0}^{p-1} a_{k+ip} r^k, \ \ n = \left\lceil \frac{m}{p} \right\rceil \tag{2}$$

Multiplication, accordingly, can be formulated as shown in Fig. 1 and expressed by equation (3). In other words, as implied by equation (4a), multiplication of high-precision numbers can be performed through two separate processes in sequence. The first is a low-fixed-precision, i.e. p-digits, multiply-accumulate (MAC) process for calculating the coefficients/partial-products C_i's as given by equation (4b). This is followed by a merging process of these coefficients/partial-products into a final single high-precision product as given by equation (4a). Equation (4b) shows that the coefficients C_i's can be represented at minimum by $2p$-digit precision. The extra digits are due to

the accumulation process. Therefore, C_i's can be expressed as shown by equation (4c).

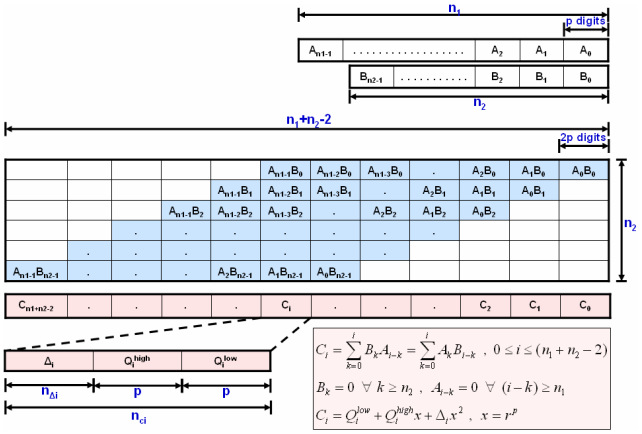


Fig. 1. Multiplication matrix of high-precision numbers

$$A=\sum_{i=0}^{n_1-1}A_i r^{ip}=\sum_{i=0}^{n_1-1}A_i x^i=A(x)\ ,\ B=\sum_{i=0}^{n_2-1}B_i r^{ip}=\sum_{i=0}^{n_2-1}B_i x^i=B(x)$$
$$C=AB=A(x)\cdot B(x)=C(x) \qquad (3)$$
$$where\ n_1=\left\lceil\frac{m_1}{p}\right\rceil,\ n_2=\left\lceil\frac{m_2}{p}\right\rceil,\ x=r^p$$

$$C(x)=C_0+C_1x+C_2x^2+C_3x^3+\ .....\ +C_{(n_1+n_2-2)}x^{(n_1+n_2-2)} \qquad (4a)$$

$$where\ \begin{cases} C_i=\sum_{k=0}^{i}B_kA_{i-k}=\sum_{k=0}^{i}A_kB_{i-k}\ ,\ 0\le i\le(n_1+n_2-2) \\ B_k=0\ \forall\ k\ge n_2 \\ A_{i-k}=0\ \forall\ (i-k)\ge n_1 \end{cases} \qquad (4b)$$

$$C_i=Q_i^{low}+Q_i^{high}x+\Delta_i x^2\ ,\ x=r^p \qquad (4c)$$

3.2. Multiplication as a Convolve-and-Merge Process

It can be easily noticed that the coefficients C_i's given by equation (4b) are in the form of a convolution sum. This led us to believe that virtualizing the convolution operation and using it as a scheduling mechanism will be a straight forward path for implementing multiplication, and hence the remaining arithmetic operations. The different sub operands, i.e. A's and B's, being stored in the system memory, will be accessed according to the convolution schedule and passed to the MAC process. The outcome of the MAC process is then delivered to the merging process which merges the partial products, according to another merging schedule, into the final results. The final results are then scheduled back into the system memory according to the same convolution schedule.

The convolution schedule, on one hand, can be derived from equation (4b). It is simply a process that generates the addresses/indexes for A's, B's and C's governed by the rules given in equation (4b). On the hand, the merging schedule can be derived from equation (5) which results from substituting equation (4c) into equation (4a). Fig. 2 shows the merging schedule as a high-precision addition of three components. The first component is simply a concatenation of all the first p-digits of the MAC output. The second component is a p-digit shifted concatenation of all the second p-digits of the MAC output. Finally, the third component is a $2p$-digit shifted concatenation of all the third p-digits of the MAC output.

$$C(x)=\sum_{i=0}^{n_1+n_2-2}Q_i^{low}x^i+\left(\sum_{i=0}^{n_1+n_2-2}Q_i^{high}x^i\right)\cdot x+\left(\sum_{i=0}^{n_1+n_2-2}\Delta_i x^i\right)\cdot x^2$$
$$C(x)\equiv C^{low}+C^{high}\cdot x+C^{carry}\cdot x^2 \qquad (5)$$
$$where$$
$$C^{low}=\sum_{i=0}^{n_1+n_2-2}Q_i^{low}x^i\ ,\ C^{high}=\sum_{i=0}^{n_1+n_2-2}Q_i^{high}x^i\ ,\ C^{carry}=\sum_{i=0}^{n_1+n_2-2}\Delta_i x^i$$

The merging schedule, as described above, is a high-precision schedule which will work only if the merging process is performed after the MAC process has finished completely. Given the algorithm complexity $O(n^2)$, and allowing the two processes to work sequentially one after another would dramatically impact the performance. However, modifying the merging process to follow a small-fixed-precision scheduling scheme that works in parallel and in synchrony with the MAC process would bring back the performance to its theoretical complexity $O(n^2)$. The modified merging scheme can be very easily derived either from equation (5) or Fig. 2 resulting in equation (6):

$$S_i=\delta_{i-1}+Q_i^{low}+Q_{i-1}^{high}+\Delta_{i-2}\ ,\ i=0,1,2,\ \ldots\ ,(n_1+n_2)$$
$$\delta_i=S_i\cdot x^{-1}=S_i\cdot r^{-p}=SHR(S_i,p\ digits)$$
$$Q_k^{low}=Q_k^{high}=\Delta_k=0\ \forall\ k<0,\ k>(n_1+n_2-2) \qquad (6)$$
$$\delta_k=0\ \forall\ k<0,\ k\ge(n_1+n_2)$$

This would mean that as soon as the MAC process finishes one partial result, C_i in $3p$-digit precision, the merging process, in-place, produces a final partial result S_i in p-digit precision, see Fig. 3. This precision matches the word-length of the supporting memory system which allows easy storage of the final result without stalling either the MAC or the merging process. The merging process

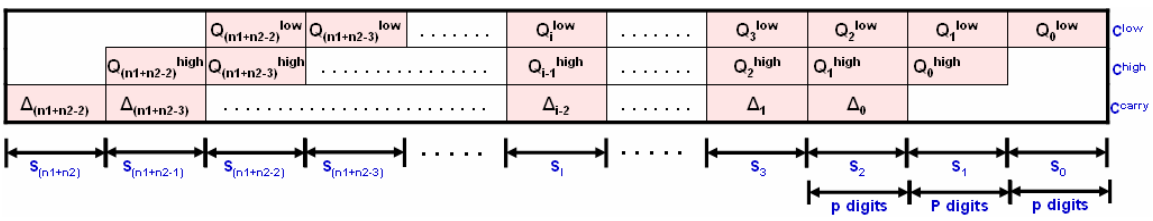


Fig. 2. Merging schedule

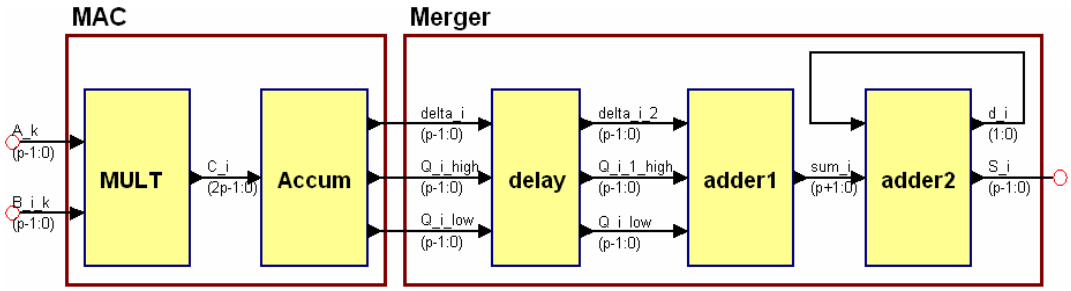


Fig. 3. Arithmetic unit architecture

registers the remaining high-precision digits for use in subsequent calculations of S_i's.

In addition to performing multiplication, the derived architecture in Fig. 3 can also natively perform the convolution operation for sequences of arbitrary-size. This is because the MAC process generates the coefficients in equation (4b) according to a convolution schedule and in fact they are the direct convolution result. In other words, only the MAC process is needed for convolution operation. Furthermore, the same unit can be used to perform addition and/or subtraction by passing the input operands directly to the merging process without going through the MAC process.

It is necessary at this point to investigate the growth of the MAC process as it represents an upper bound for the unit precision. As discussed earlier, shown in Fig. 1, and given by equation (4c), the outcome of the MAC process, C_i, consists of three parts: the multiply digits, Q_i^{low}, Q_i^{high}, and the accumulation digits, Δ_i. The corresponding number of digits, n_{Ci}, n_{Qi}^{low}, n_{Qi}^{high}, $n_{\Delta i}$, can be expressed as given by equations (7a), (7b) , (7c), and shown in Fig. 4.

$$n_{C_i} = n_{Q_i^{low}} + n_{Q_i^{high}} + n_{\Delta_i}\ ,\ n_{Q_i^{low}} = n_{Q_i^{high}} = p \tag{7a}$$

$$n_{\Delta_i} = \begin{cases} \log_r(i+1) & ,\ 0 \le i \le (n_2-2) \\ \log_r(n_2) & ,\ (n_2-1) \le i \le (n_1-1) \\ \log_r(n_1+n_2-1-i) & ,\ n_1 \le i \le (n_1+n_2-2) \end{cases} \tag{7b}$$

$$n_{\Delta_{\max}} = \left\lceil \log_r\left(\frac{m_2}{p}\right) \right\rceil ,\ 0 \le n_{\Delta_{\max}} \le p \Rightarrow p \le m_2 \le p \cdot r^p \tag{7c}$$

$$when\ p = 64 \Rightarrow 64 \le m_2 \le 64 \cdot 2^{64} \equiv 128\ EiB\ (ExibiByte)$$

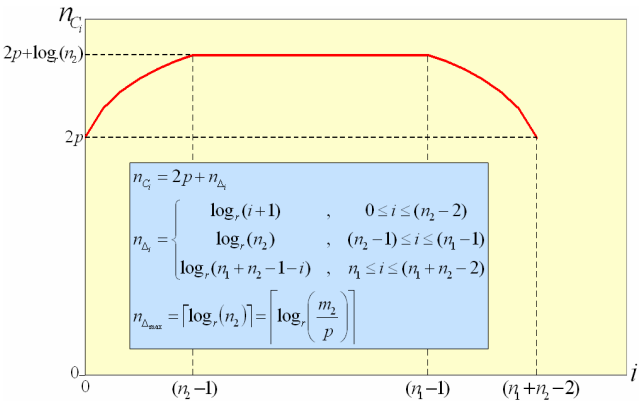


Fig. 4. Growth of MAC process (accumulation/carry digits)

Controlling the growth of the MAC process by keeping the accumulation/carry digits less than or equal to p-digits, equation (7c) gives the upper bound on the precision of the input operands. This is in terms of the hardware unit word-length, i.e. p-digits, and the numeric system base r. For example, for a binary representation, i.e. $r=2$, and a 64-bit arithmetic unit, i.e. $p=64$, the accommodated operand precision is 128 *ExibiByte* which is beyond any realistic storage system. In other words, a hardware unit with such parameters can provide almost infinite-precision arithmetic.

4. EXPERIMENTAL WORK

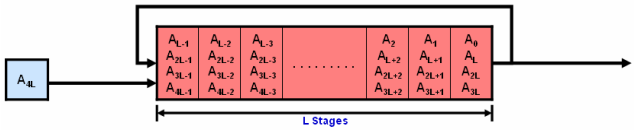
Our experiments have been performed on one of the-state-of-the-art HPRCs, i.e. SRC-6 [14]. SRC-6 is based on Xeon 2.8GHz microprocessors and Xilinx Virtex-II XC2V6000-4 100MHz FPGAs, with six local memories each 512K x 64 bits wide. The proposed architecture was developed partly in Xilinx System Generator v8.2 environment as well as in VHDL. In both environments, the architectures were highly parameterized.

The hardware performance was referenced to one of the most efficient [21] software libraries supporting arbitrary-precision arithmetic, namely GMP library on Xeon 2.8GHz. We considered two versions of GMP. The first was the compiled version of GMP which is highly optimized for the underlying microprocessor. The other was the highly portable version of GMP.

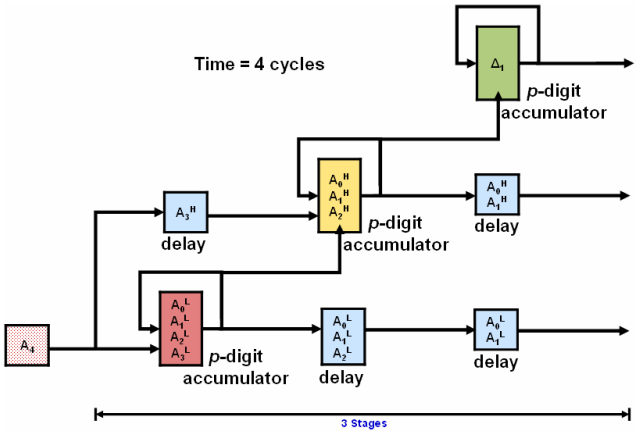
4.1. Implementation Issues

Large-precision reduction operations used in both the MAC, i.e. $3p$-digit accumulation, and the merging processes proved to be a challenge due to critical-path issues. Techniques of deep-pipelining proposed in [22, 23], and those of non-linear pipelining presented in [24, 25] were considered to eliminate those effects of deeply-pipelined operators, see Fig. (5a). The buffering mechanism, presented in [22, 23], showed either low throughput and efficiency, or high latency and resources usage for our case. Therefore, we leveraged the techniques of non-linear pipelines [24, 25] which proved to be effective, see Fig. (5b).

We analyzed the pipeline efficiency, as defined in [24, 25], of our proposed architecture. This is expressed by



(5a) Deeply-pipelined accumulator



(5b) Proposed non-linear-pipelined accumulator

Fig. 5. Accumulator of the MAC process

equations (8a), (8b) and shown in Fig. 6. We implemented two versions of the arithmetic unit, i.e. 32-bit and 64-bit. For very large data, the efficiency for the 32-bit unit was lower bounded to 80% while the efficiency for the 64-bit unit was lower bounded to 84.62%, see equation (8b) and Fig. 6.

$$\eta = 1 - \frac{L_{merger}(n_1 - 1)(n_2 - 1)}{L_{total} n_1 n_2} = 1 - \frac{1}{1 + \frac{L_{mac}}{L_{merger}}} \cdot \left(1 - \frac{1}{n_1}\right)\left(1 - \frac{1}{n_2}\right) \tag{8a}$$

where L_{mac} *is the latency of the MAC – process*

and L_{merger} *is the latency of the merging – process*

$$\eta_\infty = \lim_{n_1, n_2 \to \infty} \eta = \frac{L_{mac}}{L_{total}}$$

$$when \begin{cases} p = 32bits,\ L_{mac} = 4,\ L_{merger} = 1 \ \Rightarrow\ \eta_\infty = 80.00\% \\ p = 64bits,\ L_{mac} = 11,\ L_{merger} = 2 \ \Rightarrow\ \eta_\infty = 84.62\% \end{cases} \tag{8b}$$

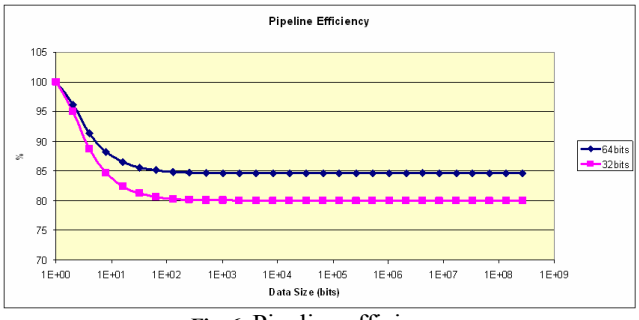


Fig. 6. Pipeline efficiency

4.2. Eperimental Results

Our experiments were performed for two bascases, i.e. 32-bit and 64-bit bases. Both cases showed similar behavior for execution time and speedup. In this section, we only show the results of the 64-bit bascase.

The arbitrary-precision addition/subtraction performance was measured on SRC-6 and compared to both the compiled and portable versions of GMP. As shown in Fig. 7, *T_COMP_HW*, i.e. the total computation time of the hardware on SRC-6, is lower than the execution time of both the compiled and portable versions of GMP. The performance speedup is shown in Fig. 8. The hardware implementation asymptotically outperforms the software, by a factor of approximately 5, because of the inherent parallelism exploited by the hardware. We can also notice that for small-precision addition/subtraction the speedup factor starts from approximately 25. This is due to the large overhead, relative to the data size, associated with the software while the only overhead associated with the hardware is due to the pipeline latency. This latency is independent on the data size. It is also worth to notice the linear behavior, *O(n)*, of both the software and the hardware. This is because both execute the same algorithm, i.e. Bascase addition/subtraction [20, 21], see Table 1.

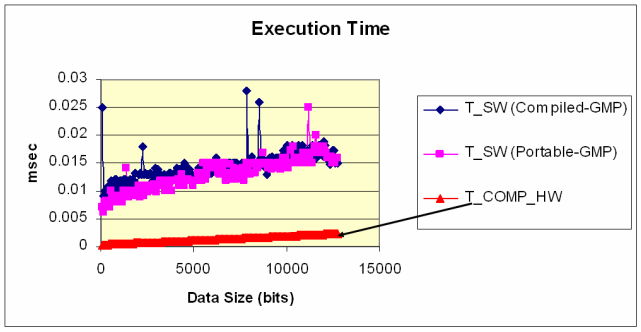


Fig. 7. Addition/subtraction execution time

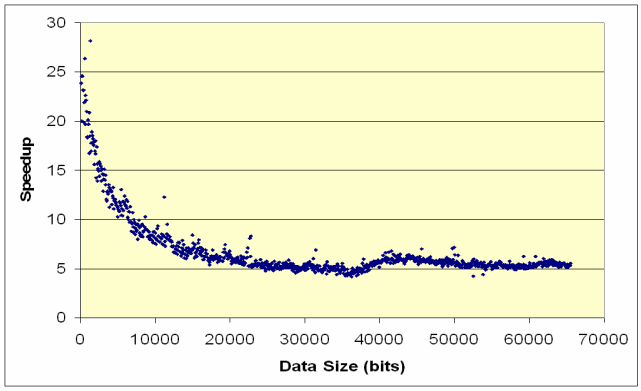
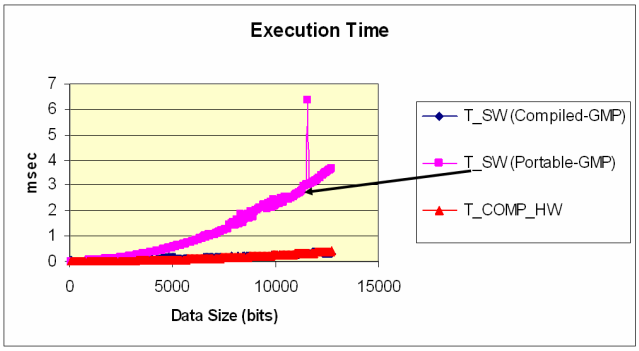
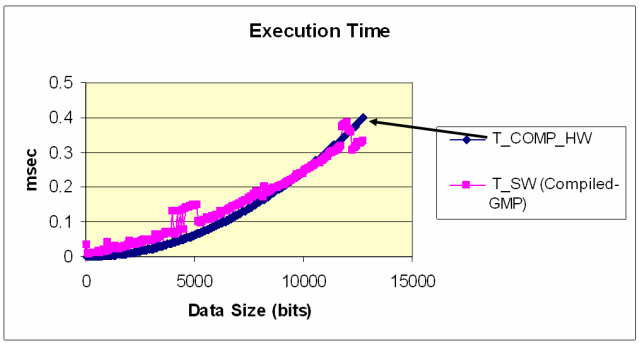


Fig. 8. Addition/subtraction hardware speedup versus GMP

In the case of multiplication, we notice a non-linear behavior $O(n^{1+e})$, $0<e<1$; see Fig. 9 and Table 1. We notice also a similar behavior to the addition/subtraction for small-size operands. In this case the hardware outperforms GMP in general with at least a factor of approximately 25. Fig. (9a) shows a significant performance for the hardware



(9a) Hardware versus compiled and portable GMP



(9b) Hardware versus compiled GMP

Fig. 9. Multiplication execution time

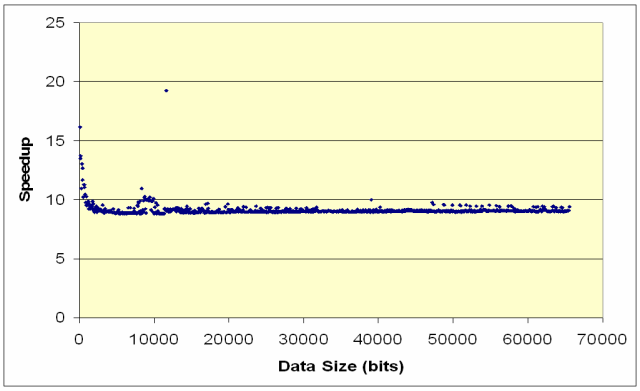


Fig. 10. Multiplication hardware speedup versus portable GMP

compared to the portable version of GMP. This is because this version of GMP uses the same algorithm as ours, i.e. Basecase with $O(n^2)$ see Table 1, independent of the data size [20, 21]. As shown in Fig. 10, the hardware behavior asymptotically outperforms the portable GMP multiplication by a factor of approximately 9. However, this is not the case with the compiled GMP multiplication which is highly optimized and adaptive. Compiled GMP uses four multiplication algorithms and adaptively switches from a slower to a faster algorithm depending on the data size and according to pre-determined thresholds [20, 21]. For these reasons, the hardware, as can be seen from Fig. (9b), outperforms the compiled GMP up to a certain threshold, approximately 10Kbits, beyond which the situation reverses.

5. CONCLUSIONS AND FUTURE WORK

In this paper we presented an unprecedented effort, to the best of our knowledge, of reconfigurable hardware support for arbitrary-precision arithmetic, and hence exact arithmetic/computations. Our emphasis was on investigating the feasibility and/or the potential of reconfigurable hardware within the domain of exact computations. Our approach was based on leveraging previous work and concepts that were introduced for solving similar problems. The proposed solution was derived from a formal representation of the problem, and was based on virtual convolution scheduling. In our formal representation we considered only the multiplication operation. That was due to the fact that multiplication is a core operation from which the other basic arithmetic operations can be easily derived. We showed how our proposed arithmetic unit could perform arbitrary-precision addition, subtraction, multiplication, as well as arbitrary-length convolution operations. Division was reserved for future support and was not included in this work only because of time constraints. Dynamic (non-linear) pipelines techniques were exploited to eliminate the effects of deeply-pipelined reduction operators. The efficiency of the proposed hardware was also analyzed.

The experimental work was verified for both correctness and performance in reference to the GMP library on the SRC-6 HPRC. The hardware outperformed GMP by a factor of 5x speedup for addition/subtraction, while the speedup factor was lower bounded to 9x compared to the portable version of GMP multiplication. We found HPRCs a promising candidate for arbitrary-precision arithmetic and exact computations.

However, for the sake of completeness a lot of work remains needed. For example, implementation of faster algorithms and the adaptive switching among them should be investigated. This can be supported by the native RTR adaptability of HPRCs. In addition, the support of floating-point arbitrary-precision arithmetic might be needed as well.

6. REFERENCES

[1] Vikram Sharma, "Complexity Analysis of Algorithms in Algebraic Computation", *PhD dissertation*, Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, January 2007.

[2] C.K. Yap, T. Dube, "The Exact Computation Paradigm", *In Computing in Euclidean Geometry, D.Z. Du and F. K. Hwang, Editors*, 2nd ed., Vol. 4 of Lecture Notes Series on Computing, pp. 452—492, World Scientific Press, Singapore, 1995.

[3] Donald E. Knuth, "The Art of Computer Programming", Vol. 2, "Seminumerical Algorithms", 3rd edition, Addison-Wesley, 1998.

[4] http://en.wikipedia.org/wiki/Arbitrary_precision_arithmetic

[5] Chen Li, "Exact Geometric Computation: Theory and Applications", *PhD dissertation*, Department of Computer Science, Institute of Mathematical Sciences,, New York University, January 2001.

[6] Julie Langou, Julien Langou, Piotr Luszczek, Jakub Kurzak, Alfredo Buttari and Jack Dongarra, "Exploiting the Performance of 32 bit Floating Point Arithmetic in Obtaining 64 bit Accuracy (Revisiting Iterative Refinement for Linear Systems)", *Proceedings of the ACM/IEEE SC 2006 Conference*, November 2006, Tampa, Florida, USA.

[7] Javier Hormigo, Julio Villalba, Emilio L. Zapata, "CORDIC Processor for Variable-Precision Interval Arithmetic", *Journal of VLSI Signal Processing Systems*, Vol. 37 Issue 1, May 2004.

[8] S. Balakrishnan, S.K. Nandy, "Arbitrary Precision Arithmetic --- SIMD Style", *Eleventh International Conference on VLSI Design: VLSI for Signal Processing*, p. 128, 1998.

[9] Saha, A., Krishnamurthy, R., "Design and FPGA Implementation of Efficient Integer Arithmetic Algorithms", *Proceedings of IEEE Southeastcon'93*, Vol. 4, Issue 7, April 1993.

[10] D.M. Chiarulli, W.G. Rudd, and D.A. Buell, "DRAFT--- A Dynamically Reconfigurable Processor for Integer Arithmetic", *In Proc. 7th Symp. on Computer Arithmetic*, pp. 309-321, IEEE Computer Society Press, 1989.

[11] A. Michalski, K. Gaj, D.A. Buell, "High-Throughput Reconfigurable Computing: A Design Study of an IDEA Encryption Cryptosystem on the SRC-6e Reconfigurable Computer", *FPL 2005*, pp.681-686.

[12] E. El-Araby, T. El-Ghazawi, J. Le Moigne, and K. Gaj, "Wavelet Spectral Dimension Reduction of Hyperspectral Imagery on a Reconfigurable Computer," *IEEE International Conference on Field-Programmable Technology, FPT 2004*, Brisbane, Australia, December 2004.

[13] E. El-Araby, M. Taher, T. El-Ghazawi, and J. Le Moigne, "Prototyping Automatic Cloud Cover Assessment (ACCA) Algorithm for Remote Sensing On-Board Processing on a Reconfigurable Computer", *IEEE International Conference on Field-Programmable Technology (FPT 2005)*, Singapore, 11-14 Dec., 2005.

[14] SRC Computers, Inc., "SRC Carte™ C Programming Environment v2.2 Guide (SRC-007-18)", August 2006.

[15] http://en.wikipedia.org/wiki/Computational_complexity_of_mathematical_operations

[16] H. L. Tredennick, T. A. Welch, "High-speed Buffering for Variable Length Operands", *Proceedings of the 4th annual symposium on Computer architecture ISCA '77*, Vol. 5, Issue 7, pp. 205-210, March 1977.

[17] Stephan Olariu, M. Christina Pinotti, S. Q. Zheng, "How to Sort N Items Using a Sorting Network of Fixed I/O Size", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 10, No. 5, pp. 487-499, May 1999.

[18] H. ElGindy and G. Ferizis, "Mapping Basic Recursive Structures to Runtime Reconfigurable Hardware", *Proceedings of FPL 2004*, August 2004.

[19] Torbjorn Granlund and Peter L. Montgomery, "Division by Invariant Integers using Multiplication", *in Proceedings of the SIGPLAN PLDI'94 Conference*, June 1994.

[20] GMP Manual, "GNU MP The GNU Multiple Precision Arithmetic Library", Edition 4.2.1, 2 May 2006.

[21] http://gmplib.org/

[22] Ling Zhuo, Gerald R. Morris, Viktor K. Prasanna, "High-Performance Reduction Circuits Using Deeply Pipelined Operators on FPGAs", *accepted for publication in IEEE Transactions on Parallel and Distributed Systems*.

[23] Ling Zhuo, Viktor K. Prasanna, "High-Performance and Area-Efficient Reduction Circuits on FPGAs", *Proceedings of the 17th International Symposium on Computer Architecture and High Performance Computing*, October 2005, Rio de Janeiro, Brazil.

[24] Hwang, K., "Advanced Computer Architecture: Parallelism, Scalability, Programmability", McGrawHill, 1993.

[25] Hwang, K., and Xu, Z., "Scalable Parallel Computing: Technology, Architecture, Programming", McGrawHill, 1998.