A FAMILY OF EMBEDDED COPROCESSORS WITH NATIVE GEOMETRIC ALGEBRA SUPPORT

S. Franchini [a], **A. Gentile** [a], **F. Sorbello** [a], **G. Vassallo** [a], and **S. Vitabile** [b]

[a] Computer Science and Artificial Intelligence Lab, DICGIM Department,
University of Palermo, Palermo, Italy
silvia.franchini@unipa.it [presenter, corresponding], antonio.gentile@unipa.it,
filippo.sorbello@unipa.it, giorgio.vassallo@unipa.it

[b] Department of Biopathology and Medical Biotechnologies
University of Palermo, Palermo, Italy
salvatore.vitabile@unipa.it

ABSTRACT. Clifford Algebra or Geometric Algebra (GA) is a simple and intuitive way to model geometric objects and their transformations. Operating in high-dimensional vector spaces with significant computational costs, the practical use of GA requires, however, dedicated software and/or hardware architectures to directly support Clifford data types and operators. In this paper, a family of embedded coprocessors for the native execution of GA operations is presented. The paper shows the evolution of the coprocessor family focusing on the latest two architectures that offer direct hardware support to up to five-dimensional Clifford operations. The proposed coprocessors exploit hardware-oriented representations of GA elements and operators properly conceived to obtain fast performing implementations. The coprocessor prototypes, implemented on Field Programmable Gate Arrays (FPGA) development boards, show significant speedups of about one order of magnitude with respect to the baseline software library Gaigen running on a general-purpose processor. The paper also presents an execution analysis of different GA-based applications, namely inverse kinematics of a robot, optical motion capture, raytracing, and medical image processing, showing good speedups with respect to the baseline general-purpose implementation.

1. INTRODUCTION

Geometric Algebra (GA), also known as Clifford Algebra, is a powerful mathematical tool that allows for a simple and intuitive representation of geometric objects and their transformations [1]-[6]. The coordinate-free representation of GA is made possible by using extended objects in high-dimensional vector spaces to represent three-dimensional (3D) geometry. Four-dimensional (4D) and five-dimensional (5D) Clifford Algebras are used to implement the most powerful models of 3D geometry, namely the homogeneous model and the conformal model, respectively. Since the generic element of an n-dimensional GA has 2^n coordinates, the computational complexity increases quickly when the algebra dimension increases, growing rapidly from 16 to 32 coefficients when moving from 4D to 5D, and requiring more than a thousand multiply-adds between coefficient combinations. These significant computational costs demand in turn dedicated software and/or hardware architectures to directly support Clifford data types and operators [7]-[19]. Software implementations of GA are conceived to execute Clifford operations on general-purpose processors using dedicated software libraries, such as Gaigen [7] and GluCat [8], packages for symbolic and numerical environments, such as the Clifford [9] and GA [10] packages for Maple, the Gable [11] and Clifford Multivector Toolbox [12] packages for Matlab, and the Grassmann algebra package [13] for Mathematica, or stand-alone programs, such as CluCalc [14]. Since faster performing solutions are needed, recently the attention has turned toward fully hardware

implementations or hardware-software codesigns. A combined software-hardware approach is proposed in [15]-[17] to accelerate GA-based algorithms. A pre-compiler, named Gaalop (Geometric Algebra Algorithm Optimizer), compiles GA algorithms to an intermediate representation converting GA operations in basic arithmetic operations [15]. This intermediate representation can be further compiled to run on different hardware platforms, such as Field Programmable Gate Arrays (FPGAs) or Graphics Processing Units (GPUs) [16]-[17]. A specialized coprocessor implemented on an Application Specific Integrated Circuit (ASIC) is presented in [18]. The coprocessor is specifically designed for color edge detection applications and supports only the 3D GA operations required in the target applications. The FPGA-based coprocessor prototype presented in [19] executes product operations for algebras of dimension up to 8 and uses 24-bit integer numbers to represent scalar coefficients of Clifford operands.

In this paper, a family of embedded coprocessors for the native execution of GA operations is presented, which we have been designing and implementing in the last few years [22]-[27]. The paper shows the evolution of the coprocessor family based on several factors, namely algebra dimension, supported operations, Clifford number representation, execution flow (sequential or parallel, scalar or pipeline), datapath width, data precision. First, the design space of computing architectures to natively execute GA operations has been explored taking into account different architectural parameters, such as number of multiply-add units and coefficient precision. Several alternative architectures, as resulted from the design space exploration, have been implemented and compared in terms of area cost, relative error, latencies, and speedup [25]. The resulting family of coprocessors offers direct hardware support to up to 5D GA operations. Different hardware-oriented representations of GA elements and operations have been properly conceived to obtain fast performing implementations. The representation based on the variable-length homogeneous elements, used for the first two architectures of the family [22]-[23], has been then replaced with a fixed-size representation based on quadruples in the latest coprocessors [24],[26],[27]. This novel representation of algebra elements allows for important simplifications of algebraic operations that in turn lead to a faster and more compact hardware architecture. The latest presented coprocessing architecture exploits a simplified formulation of the Conformal Geometric Algebra (CGA) operations, namely reflections, rotations, translations, and dilations, which results in faster execution of such operations [27]. The coprocessors have all been prototyped using development boards based on FPGA devices. The latest two coprocessors have been implemented as complete embedded Systems-on-Chip (SoCs) [26]-[27]. Experimental tests performed on the prototypes have shown significant speedups with respect to the baseline software library Gaigen running on a general-purpose processor [7]. The latest two coprocessors, named CliffordALU5 and ConformalALU, respectively, show native support of all GA operations in both 4D and 5D spaces with average speedups of about one order of magnitude over the baseline software implementation. To evaluate the coprocessor effectiveness in specific application domains, an application suite composed of different GA-based algorithms, namely inverse kinematics of a robot, optical motion capture, raytracing, and medical image processing, has been used as testbench, showing good speedups with respect to the baseline general-purpose implementation.

The rest of the paper is organized as follows: Section 2 outlines the design space exploration of GA-based computing architectures, while the resulting coprocessor family is presented in Section 3. Section 4 presents experimental results, while conclusions are contained in Section 5. A comprehensive introduction to the fundamental concepts of GA can be found in [1]-[6].

2. Design Space Exploration

The design space of hardware implementations that natively support GA operations has been analyzed along several axes, namely, Clifford number representation (homogeneous elements versus quadruples), execution flow (sequential versus parallel, scalar versus pipeline), number of multiply-add units, coefficient precision, datapath width, instruction word length. As described in Table 1, several alternative architectures based on different sets of architectural parameters have been explored and different design points have been implemented and prototyped. The resulting family of coprocessing architectures is detailed in the following section. One of the design parameters is related to the representation to be used for algebra elements. The standard variable-length representation, based on homogeneous elements, used for the first two architectures, has been replaced, in the latest coprocessors, with the fixed-size representation based on quadruples. A description of this representation is provided in Sections 3.3 and 3.4.

Table 1. Design space exploration

Coprocessor	GA operations	Clifford number representation	Execution flow	Parallelism techniques	N. of multipliers	Precision	Instruction word length
S-CliffoSor [22]	4D products, sums, differences	Homogeneous elements	Sequential	-	-	32-bit integers	15x32-bit words = 480 bits
CliffoSor [23]	4D products, sums, differences 3D rotations (Operations on bivectors not implemented)	Homogeneous elements	Parallel	-	24	32-bit integers	15x32-bit words = 480 bits
Quad-CliffoSor [24]	4D products, sums, differences, unary operations	Quadruples	Parallel	Pipeline	16	16-bit floating point	9x16-bit words = 144 bits
CliffordALU5 [26]	4D/5D products, sums, differences, unary operations	Quadruples	Parallel	Pipeline	16	32-bit floating point	9x32-bit words = 288 bits
ConformalALU [27]	5D conformal geometric operations (reflections, rotations, translations, dilations)	5D vectors	Parallel	Pipeline	5	32-bit floating point	16x32-bit words = 512 bits

3. Coprocessor Family

In this section, the family of embedded coprocessors for the native support of GA operations is presented. A brief description of the implemented architectures, whose main features are listed in Table 1, is provided in the following subsections.

3.1 S-CliffoSor

The first coprocessor to be implemented was S-CliffoSor (Sliced Clifford coprocesSor), which offers direct hardware support to 4D Clifford operations between homogeneous elements using 32-bit integers numbers to represent scalar coefficients of basis blades. A homogeneous element or homogeneous multivector contains only blades of the same grade. Each 4D Clifford multivector is an ordered tuple of 5 homogeneous elements m = (s, v, b, t, p), as listed in Table 2. Each homogeneous element is composed of a different number of blades, where each blade is a coefficient-basis blade pair. A 4-bit mask is associated with each basis blade, where each bit is associated with a basis vector e_i, i=1,2,3,4, with e_1 the least significant bit. S-CliffoSor is based on a 32-bit sequential ALU that executes addiction, subtraction, multiplication and logical operations. Each 4D Clifford operation is decomposed into the proper sequence of these basic operations whose execution is supervised step-by-step by a microprogrammed control unit. The sequential architecture of the coprocessor leads to long per-operation latencies, which can be however hid by replicating the single S-CliffoSor slice to execute in parallel multiple independent Clifford operations.

Table 2. 4D GA homogeneous elements

Homogeneous element	Blades					
Scalar (s)	a_0					
Vector (v)	a_1e_1	a_2e_2	a_3e_3	a_4e_4		
Bivector (b)	$a_{12}e_1e_2$	$a_{13}e_1e_3$	$a_{14}e_1e_4$	$a_{23}e_2e_3$	$a_{24}e_2e_4$	$a_{34}e_3e_4$
Trivector (t)	$a_{123}e_1e_2e_3$	$a_{124}e_1e_2e_4$	$a_{134}e_1e_3e_4$	$a_{234}e_2e_3e_4$		
Pseudoscalar (p)	$a_{1234}e_1e_2e_3e_4$					

3.2 CliffoSor

As S-CliffoSor, the second developed architecture, namely CliffoSor (Clifford coprocesSor), executes 4D Clifford operations between homogeneous elements using 32-bit integer numbers to represent scalar coefficients. While S-CliffoSor is based on a sequential execution flow, CliffoSor uses parallel structures for the fastest execution of Clifford operations. Three different functional units directly support 4D Clifford products, 4D Clifford sums/differences and 3D Clifford rotations, respectively. To save resources, the most complex operations on bivectors are not directly supported in hardware, but they are handled in a higher-level Application Programming Interface (API) with multiple calls to the coprocessor. This design choice allowed us to use 24 parallel multipliers for Clifford products execution, rather than 36 parallel multipliers, as needed to directly support in hardware bivector-bivector products.

3.3 Quad-CliffoSor

As CliffoSor, Quad-CliffoSor (Quadruple-based Clifford coprocesSor) is a parallel architecture composed of three dedicated units for the execution of 4D Clifford products, 4D Clifford sums/differences, and 4D Clifford unary operations, respectively. While the previous architectures use the natural representation of GA elements based on homogeneous elements, Quad-CliffoSor first introduces a novel representation based on fixed-size elements, called quadruples. The variable size of homogeneous elements, as described in Table 2, leads to some storage inefficiencies since a six-element vector is used to implement each homogeneous multivector, but only the bivector uses all six elements. Defining the four-size elements, or quadruples, listed in Table 3, the generic 4D multivector can be written as a tuple of 4 quadruples, m = (V, T, S, P). Each quadruple is composed of four blades. Using quadruples, rather than homogeneous elements, has two important advantages: first, fixed-size operands are better suited to a

hardware implementation; second, using the 4 quadruples listed in Table 3 allows for significant simplifications of product operations leading in turn to a faster and more compact hardware architecture. As demonstrated in [24], the product of two quadruples always gives as result the sum of two quadruples so that a single fixed format can be used for both input data and output result. Furthermore, a simplified algorithm can be used to execute product operations on quadruples. Any quadruple can be reduced to a quadruple of type V by simple sign changing operations on its coefficients. The product between any couple of quadruples can be therefore reduced to the product between two quadruples of type V by simple sign changing and/or swapping operations. The latter operation is the only operation to be implemented. The Quad-CliffoSor multiplier unit was properly designed to permit the fastest execution of this operation. 16 parallel multipliers are used to calculate the 16 products between the input quadruple coefficients, while a three-stage pipeline allows a product operation result to be provided on every clock cycle. Quad-CliffoSor is the first coprocessor of the family that uses 16-bit floating point numbers to represent scalar coefficients of the basis blades.

Table 3. 4D GA quadruples

Quadruple	Blades			
V	a_1e_1	a_2e_2	a_3e_3	a_4e_4
T	$a_{234}e_2e_3e_4$	$a_{134}e_1e_3e_4$	$a_{124}e_1e_2e_4$	$a_{123}e_1e_2e_3$
S	$a_{14}e_1e_4$	$a_{24}e_2e_4$	$a_{34}e_3e_4$	a_0
P	$a_{23}e_2e_3$	$a_{13}e_1e_3$	$a_{12}e_1e_2$	$a_{1234}e_1e_2e_3e_4$

3.4 CliffordALU5

CliffordALU5 is the first coprocessor that natively supports GA operations in the 5D space. The fixed-size representation based on quadruples, introduced in Quad-CliffoSor for 4D Clifford elements, is extended in CliffordALU5 to 5D Clifford elements. The eight quadruples of 5D GA are listed in Table 4.

Table 4. 5D GA quadruples

Quadruple	Blades			
V	a_1e_1	a_2e_2	a_3e_3	a_4e_4
T	$a_{234}e_2e_3e_4$	$a_{134}e_1e_3e_4$	$a_{124}e_1e_2e_4$	$a_{123}e_1e_2e_3$
S	$a_{14}e_1e_4$	$a_{24}e_2e_4$	$a_{34}e_3e_4$	a_0
P	$a_{23}e_2e_3$	$a_{13}e_1e_3$	$a_{12}e_1e_2$	$a_{1234}e_1e_2e_3e_4$
V’	$a_{15}e_1e_5$	$a_{25}e_2e_5$	$a_{35}e_3e_5$	$a_{45}e_4e_5$
T’	$a_{2345}e_2e_3e_4e_5$	$a_{1345}e_1e_3e_4e_5$	$a_{1245}e_1e_2e_4e_5$	$a_{1235}e_1e_2e_3e_5$
S’	$a_{145}e_1e_4e_5$	$a_{245}e_2e_4e_5$	$a_{345}e_3e_4e_5$	a_5e_5
P’	$a_{235}e_2e_3e_5$	$a_{135}e_1e_3e_5$	$a_{125}e_1e_2e_5$	$a_{12345}e_1e_2e_3e_4e_5$

As demonstrated in [26], the simplified algorithm used in Quad-CliffoSor for product operations execution can be extended to 5D operations. Since 4D quadruples are a subset of 5D quadruples, CliffordALU5 is a universal coprocessor that can directly execute all 4D and 5D GA operations (geometric products, outer products, left and right contractions, sums, differences, and unary operations) using quadruples as basic elements of computation. The block diagram of the CliffordALU5 coprocessor (Figure 1(a)) shows three dedicated functional units for the execution of 4D/5D Clifford products, 4D/5D Clifford sums/differences, and 4D/5D Clifford unary operations, respectively. The simplified algorithm used for product operations, described in Section 3.3 for the 4D case, allows for a compact architecture of the Clifford multiplier unit, as shown in Figure 1(b). This pipelined unit contains a 16x multiplier bank for the parallel execution of the 16 multiplications required by a product operation between quadruples.

CliffordALU5 is the first coprocessor of the family that uses 32-bit floating point numbers to represent scalar coefficients of Clifford operands.

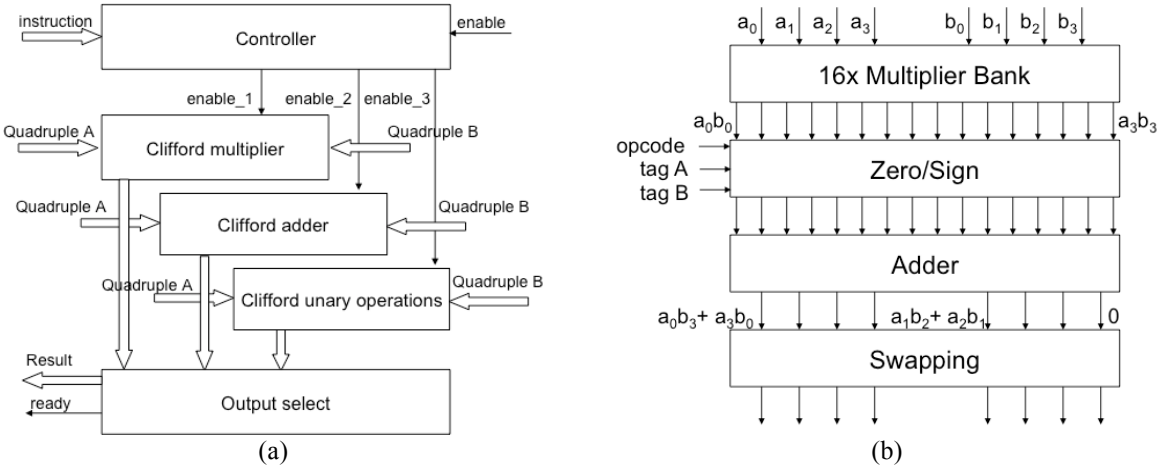


Figure 1. (a) CliffordALU5 block diagram; (b) Clifford multiplier unit block diagram

3.5 ConformalALU

The latest architecture, named ConformalALU, has been designed for the direct hardware support of Conformal Geometric Algebra (CGA) or 5D Geometric Algebra geometric operations, namely reflections, rotations, translations, and dilations. The coprocessor exploits a simplified formulation of these operations aimed at a parallel hardware implementation, which derives from two considerations. First, a conformal geometric operation on a generic k-blade A_k, represented in CGA by the "sandwich" geometric product, can be decomposed in operations on vectors according to the following formula:

$$XA_k\tilde{X} = X(a_1 \wedge a_2 \wedge \ldots \wedge a_k)\tilde{X} = Xa_1\tilde{X} \wedge Xa_2\tilde{X} \wedge \ldots \wedge Xa_k\tilde{X} \qquad (1)$$

where X is the versor (rotor, translator, or dilator) that represents the conformal transformation. Second, rather than using the standard "sandwich" geometric product of CGA, each conformal geometric operation can be obtained by two non-commuting successive reflections. The basic operation becomes therefore the reflection of a 5D vector. In our implementation, each vector reflection is executed in turn using the following simplified formula based on the classical dot product, rather than the standard "sandwich" geometric product of CGA:

$$a' = a_\perp - a_\parallel = a - 2a_\parallel = a - 2|a_\parallel|m = a - 2(a \cdot m)m \qquad (2)$$

where a is the vector to be reflected in a plane with unit-normal m, while a' is the reflected vector, as depicted in Figure 2.

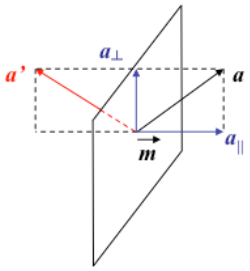


Figure 2. Reflection of a vector *a*

Requiring only one dot product and two subtractions between vectors, this new formulation leads to a computational advantage and therefore a more compact and faster hardware architecture [27]. The block diagram of the ConformalALU coprocessor is depicted in Figure 3(a), while Figure 3(b) shows the block diagram of the Reflector unit. Two cascade Reflector units are used to execute a whole instruction stream in a pipeline fashion. As CliffordALU5, ConformalALU uses 32-bit floating point numbers to represent scalar coefficients.

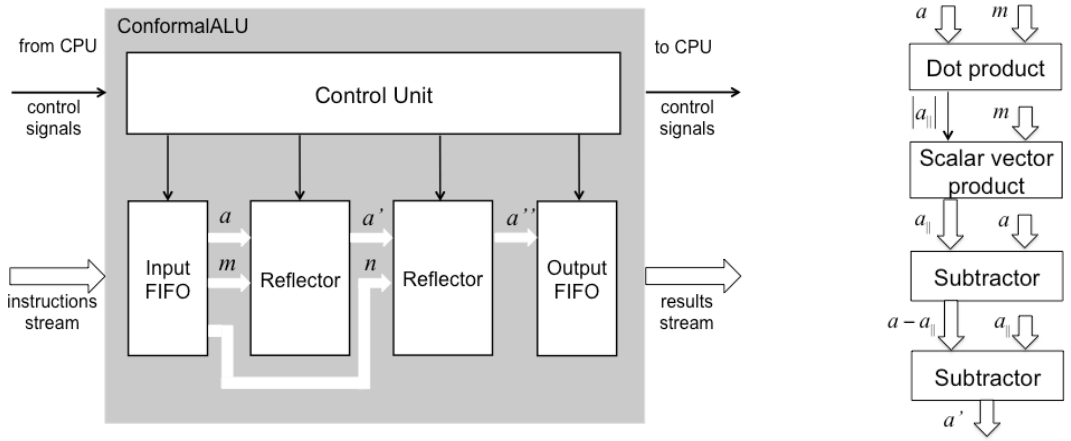


Figure 3. (a) ConformalALU block diagram; (b) Reflector unit block diagram

4. EXPERIMENTAL RESULTS

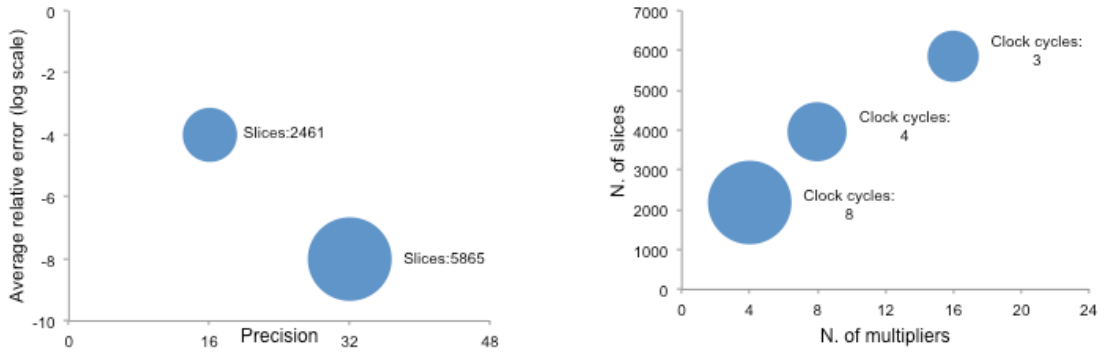
4.1 Coprocessor performance analysis

The designed coprocessors have all been prototyped using development boards based on FPGA devices. Several experimental tests have been performed to evaluate the coprocessor performance and compare it with the fast Gaigen software chosen as baseline general-purpose implementation. Table 5 shows the performance analysis of the coprocessor family in terms of clock frequency, area cost, latency per operation, and speedup. CliffoSor shows a higher area cost as well as reduced latencies per operation with respect to S-CliffoSor. These results depend on the different execution flow of the two architectures, namely, sequential for S-CliffoSor and parallel for CliffoSor. The reduced area cost, as well as the increased speedup, of Quad-CliffoSor in comparison with CliffoSor, is an effect of the computational and architectural simplifications of the quadruple-based representation. The higher area cost of CliffordALU5 when compared with Quad-CliffoSor, is due to the higher precision (32-bit rather than 16-bit) of the scalar coefficients. A scalar version and a pipelined version of the ConformalALU coprocessor have been designed. As reported in Table 5, the pipelined ConformalALU consumes more resources, but allows for reduced latency and, consequently, increased throughput. As a result of the design space exploration, Figure 4 presents a performance analysis, in terms of area cost, relative error, and latency, of different alternative architectures based on different sets of design parameters, such as the number of multiply-add units and the coefficient precision. Figure 4(a) shows average relative errors (with respect to the full-precision Gaigen implementation) and area costs of the multiplier units of Quad-CliffoSor and CliffordALU5, which use 16-bit and 32-bit precision, respectively. The higher-precision architecture consumes over two times more resources than the lower one, but a significant reduction of relative errors is observed. Three different versions of the CliffordALU5 coprocessor, which use 4, 8, and 16 parallel multipliers, respectively, for product operations execution are compared

in Figure 4(b) in terms of area costs and latencies per operation. Increasing the number of multipliers, the area cost increases, as well; however, a reduced latency in the product operation execution between quadruples can be observed.

Table 5. Performance analysis of the coprocessor family

Coprocessor	Clock frequency	Area (n. of FPGA slices)	Latency (clock cycles)	Speedup over Gaigen software
S-CliffoSor	45 MHz	2,295 (single slice)	(average) Products: 91 Sums/Diff.: 78	(potential) Products: 4x Sums/Diff.: 3x
CliffoSor	50 MHz	8,444	Products: 7 Sums/Diff.: 5	(potential) Products: 4x Sums/Diff.: 12x
Quad-CliffoSor	50 MHz	3,201	Products: 3 Sums/Diff.: 1	(potential) Products: 23x Sums/Diff.: 33x
CliffordALU5	100 MHz	6,011	Products: 3 Sums/Diff.: 1	(real) 4D Products: 5x 5D Products: 4x Sums/Diff.: 2x
ConformalALU	125 MHz	5,876 (scalar) 9,640 (pipelined)	(average) 315 (scalar) 88 (pipelined)	(real) Reflections: 56x Rotations: 15x Translations: 46x Dilations: 41x

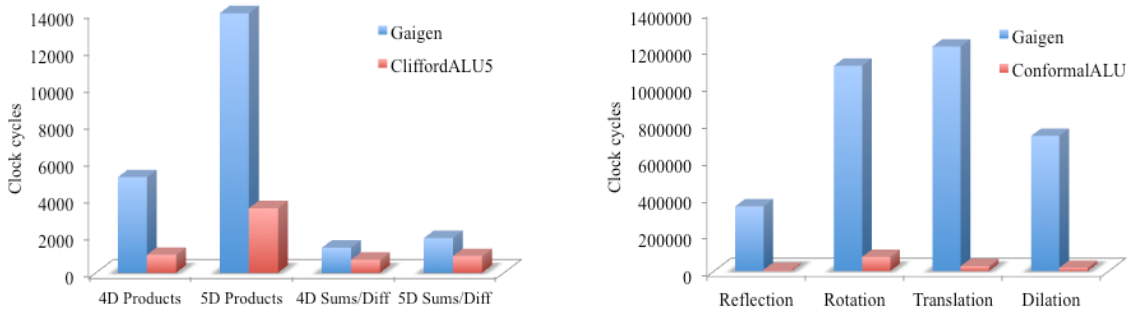


(a) (b)

Figure 4. (a) Average relative error and area cost (number of FPGA slices) versus precision for Quad-CliffoSor and CliffordALU5; (b) Area cost and latency per operation (clock cycles) versus number of multipliers for CliffordALU5.

To evaluate the speedups over the reference Gaigen software, the same test operations were executed using both the Gaigen library and the coprocessor. The first three coprocessors were prototyped on FPGA boards connected via the PCI bus or the Ethernet to the host computer. Only potential speedups in terms of clock cycles were estimated since the coprocessor ran on the FPGA using a clock frequency slower than the software running on the conventional host PC. Conversely, the latest two coprocessors were implemented as complete Systems on Chip (SoCs) using FPGA boards that integrate both a PowerPC general-purpose processor and the specialized coprocessor on the same chip. A real speedup, in terms of wall-clock times, has been therefore measured over the software library running on the PowerPC processor at the same operating frequency as the coprocessor. Gaigen/CliffordALU5 and Gaigen/ConformalALU comparisons are summarized in Figures 5(a) and 5(b), respectively. As reported in Table 5, CliffordALU5 achieves effective average speedups

of 5x for 4D Clifford products, 4x for 5D Clifford products, and 2x for 4D/5D Clifford sums, while effective speedups achieved by ConformalALU are 56x for reflection operations, 15x for rotations, 46x for translations, and 41x for dilations, respectively.



(a) (b)

Figure 5. (a) Gaigen/CliffordALU5 comparison; (b) Gaigen/ConformalALU comparison

4.2 Application suite

A suite of GA-based applications, including inverse kinematics of a robot, optical motion capture, raytracing, and medical image processing, has been used as testbench to evaluate the effectiveness of the coprocessor family in specific application domains. A description of these applications can be found in [26], [27]. The testbench algorithms have been executed using the latest two coprocessors, namely CliffordALU5 and ConformalALU, and their performance has been compared with the baseline general-purpose implementation based on the Gaigen software. Table 6 lists the observed speedups for each application. Taking into account the mix of Clifford operations required by each algorithm, the first three applications have been executed on the CliffordALU5 coprocessor, while the ConformalALU coprocessor has been used to accelerate medical image processing algorithms [20],[21]. The medical imaging applications, accelerated by the ConformalALU, massively use CGA operations (translations and rotations). The higher speedups of these applications are an effect of the simplified formulation of CGA operations that allows for faster execution of these operations.

Table 6. Observed speedups for the test applications

Application	Inverse kinematics	Motion capture	Raytracing	Medical image segmentation	Medical image registration
Observed speedup	3.4x	3.8x	4.8x	46x	43x

5. CONCLUSIONS

A family of embedded coprocessors that offer direct hardware support to GA operations has been presented in this paper. As overall result, the latest two coprocessors, namely CliffordALU5 and ConformalALU, natively execute all 4D and 5D GA operations showing speedups of about one order of magnitude relative to the baseline software implementation Gaigen. It has been observed that the novel simplified formulation of 5D CGA operations, used in ConformalALU, allows for a further speedup of about 10x with respect to the execution on the CliffordALU5 coprocessor. Future work will be aimed therefore to integrate the two coprocessors CliffordALU5 and ConformalALU in a single architecture to obtain a complete System-on-Chip that supports all basic operations of up to 5D GA (products, sums, unary operations) and accelerates geometric operations (reflections, rotations, translations, uniform scaling) of the 5D conformal model using the fast dedicated unit ConformalALU.

REFERENCES

[1] D. Hestenes, *New Foundations for Classical Mechanics*, Kluwer Academic, 1986.

[2] D. Hestenes and G. Sobczyk, *Clifford Algebra to Geometric Calculus: A Unified Language for Mathematics and Physics*, Kluwer Academic, 1987.

[3] L. Dorst and S. Mann, *Geometric algebra: A computational framework for geometrical applications (part 1: Algebra)*, in IEEE Comput. Graph. Appl., vol. 22, no. 3, pp. 24–31, May/Jun. 2002.

[4] L. Dorst and S. Mann, *Geometric algebra: A computational framework for geometrical applications (part 2: Applications)*, in IEEE Comput. Graph. Appl., vol. 22, no. 4, pp. 58–67, Jul./Aug. 2002.

[5] J. Lasenby, W. J. Fitzgerald, C. J. L. Doran and A. N. Lasenby, *New Geometric Methods for Computer Vision*, in Int. J. Comp. Vision, vol. 36, no. 3, pp. 191-213, 1998.

[6] D. Hestenes, Hongbo Li, A. Rockwood, *New Algebraic Tools for Classical Geometry*, in Geometric Computing with Clifford Algebras, G. Sommer (ed.), Springer Heidelberg, pp. 3-26, 2000.

[7] D. Fontijne, *Gaigen 2: A geometric algebra implementation generator*, in Proc. 5th Int. Conf. Generative Programming and Component Eng., 2006, pp. 141–150.

[8] P. Leopardi, The GluCat Home Page. [Online]. Available: http://glucat.sourceforge.net/.

[9] R. Ablamowicz and B. Fauser, CLIFFORD - A Maple package for Clifford algebra computations. [Online]. Available: http://math.tntech.edu/rafal/.

[10] M. Ashdown, GA package for Maple. Available: http://www.mrao.cam.ac.uk/~maja1/software/GA/.

[11] S. Mann, L. Dorst, and T. Bouma, *The making of GABLE: A geometric algebra learning environment in Matlab*, in Geometric Algebra with Applications in Science and Engineering, E. Bayro-Corrochano and G. Sobczyk, Eds., New York, NY, USA, Springer, 2001, pp. 491–511.

[12] E. Hitzer, S. Sangwine, Clifford Multivector Toolbox, A toolbox for computing with Clifford algebras in Matlab. [Online]. Available: http://sourceforge.net/projects/clifford-multivector-toolbox/.

[13] J. Browne, The Grassmann Algebra Book Home Page. [Online]. Available: http://www.grassmannalgebra.info/grassmannalgebra/.

[14] C. Perwass, The CLUCalc Home Page. [Online]. Available: http://www.CluCalc.info.

[15] D. Hildenbrand, J. Pitt, and A. Koch, *Gaalop-high performance parallel computing based on conformal geometric algebra*, in Geometric Algebra Computing, Springer, 2010, pp. 477–494.

[16] P. Charrier and D. Hildenbrand, *Geometric algebra computing technology for accelerated processing units*, presented at the Embedded World Conf., Nürnberg, Germany, 2013.

[17] D. Hildenbrand, *Geometric algebra computers*, in Foundations of Geometric Algebra Computing, New York, NY, USA, Springer, 2013, pp. 179–188.

[18] B. Mishra, P. Wilson, and R. Wilcock, *A Geometric Algebra Co-Processor for Color Edge Detection*, in Electronics 2015, 4(1), pp. 94-117.

[19] C. Perwass, C. Gebken, G. Sommer, *Implementation of a Clifford algebra co-processor design on a field-programmable gate array*, in Clifford Algebras: Applications to Mathematics, Physics, and Engineering, Series: Progress in Mathematical Physics, vol. 34, R. Ablamowicz, Ed., Springer, 2004.

[20] J. Rivera-Rovelo and E. Bayro-Corrochano, *Surface approximation using growing self-organizing nets and gradient information*, Appl. Bionics Biomech., vol. 4, no. 3, pp. 125–136, 2007.

[21] E. Bayro-Corrochano and J. Rivera-Rovelo, *The use of geometric algebra for 3d modeling and registration of medical data*, J. Math. Imaging Vis., vol. 34, no. 1, pp. 48–60, May 2009.

[22] S. Franchini, A. Gentile, M. Grimaudo, C.A. Hung, S. Impastato, F. Sorbello, G. Vassallo, and S. Vitabile, *A Sliced Coprocessor for Native Clifford Algebra Operations*, in Proceedings of the 10th IEEE Euromicro Conference on Digital System Design - Architectures, Methods and Tools (DSD 2007), Lübeck, Germany, August 29-31, 2007, pp. 436-439, IEEE Computer Society Press.

[23] Silvia Franchini, Antonio Gentile, Filippo Sorbello, Giorgio Vassallo, and Salvatore Vitabile, *An Embedded, FPGA-based Computer Graphics Coprocessor with Native Geometric Algebra Support*, in Integration, The VLSI Journal, Volume 42, Issue 3, pp. 346-355, June 2009.

[24] Silvia Franchini, Antonio Gentile, Filippo Sorbello, Giorgio Vassallo, and Salvatore Vitabile, *Fixed-size Quadruples for a New, Hardware-Oriented Representation of the 4D Clifford Algebra*, in Advances in Applied Clifford Algebras, Volume 21, Issue 2, pp. 315-340, June 2011.

[25] Silvia Franchini, Antonio Gentile, Filippo Sorbello, Giorgio Vassallo, and Salvatore Vitabile, *Design Space Exploration of Parallel Embedded Architectures for Native Clifford Algebra Operations*, in IEEE Design and Test of Computers, Volume 29, Issue 3, pp. 60-69, May-June 2012.

[26] Silvia Franchini, Antonio Gentile, Filippo Sorbello, Giorgio Vassallo, and Salvatore Vitabile, *Design and Implementation of an Embedded Coprocessor with Native Support for 5D, Quadruple-based Clifford Algebra*, in IEEE Transactions on Computers, Vol. 62, No. 12, pp. 2366-2381, Dec. 2013.

[27] Silvia Franchini, Antonio Gentile, Filippo Sorbello, Giorgio Vassallo, and Salvatore Vitabile, *ConformalALU: a Conformal Geometric Algebra Coprocessor for Medical Image Processing*, in IEEE Transactions on Computers, Vol. 64, No. 4, pp. 955-970, April 2015.