electronics

MDPI

Article

An Optimized Architecture for CGA Operations and Its Application to a Simulated Robotic Arm

Salvatore Vitabile [1,*], Silvia Franchini [1] and Giorgio Vassallo [2]

[1] Department of Biomedicine, Neuroscience, and Advanced Diagnostics (Bi.N.D.), University of Palermo, Via del Vespro, 129, 90127 Palermo, Italy
[2] Department of Engineering, University of Palermo, Viale delle Scienze, Edificio 6, 90128 Palermo, Italy
* Correspondence: salvatore.vitabile@unipa.it

Abstract: Conformal geometric algebra (CGA) is a new geometric computation tool that is attracting growing attention in many research fields, such as computer graphics, robotics, and computer vision. Regarding the robotic applications, new approaches based on CGA have been proposed to efficiently solve problems as the inverse kinematics and grasping of a robotic arm. The hardware acceleration of CGA operations is required to meet real-time performance requirements in embedded robotic platforms. In this paper, we present a novel embedded coprocessor for accelerating CGA operations in robotic tasks. Two robotic algorithms, namely, inverse kinematics and grasping of a human-arm-like kinematics chain, are used to prove the effectiveness of the proposed approach. The coprocessor natively supports the entire set of CGA operations including both basic operations (products, sums/differences, and unary operations) and complex operations as rigid body motion operations (reflections, rotations, translations, and dilations). The coprocessor prototype is implemented on the Xilinx ML510 development platform as a complete system-on-chip (SoC), integrating both a PowerPC processing core and a CGA coprocessing core on the same Xilinx Virtex-5 FPGA chip. Experimental results show speedups of 78× and 246× for inverse kinematics and grasping algorithms, respectively, with respect to the execution on the PowerPC processor.

Keywords: application-specific processors; Clifford Algebra; computational geometry; conformal geometric algebra; FPGA-based prototyping; grasping; human-like robotic arms; inverse kinematics

Citation: Vitabile, S.; Franchini, S.; Vassallo, G. An Optimized Architecture for CGA Operations and Its Application to a Simulated Robotic Arm. *Electronics* **2022**, *11*, 3508. https://doi.org/10.3390/electronics11213508

Academic Editor: Rui Pedro Lopes

Received: 27 September 2022
Accepted: 23 October 2022
Published: 28 October 2022

1. Introduction

The use of computer vision-guided robots for the manipulation of 3D objects plays a fundamental role in the automation of industrial processes [1,2]. In particular, collaborative robots interacting with humans are increasingly used in current production environments [3,4]. Mathematical tools used to handle the robot kinematics and dynamics are traditionally based on standard vector algebra and projective geometry [5,6]. In the last few years, however, several studies have been published that have demonstrated the effectiveness of conformal geometric algebra (CGA) in the visually guided robotics field [7–9]. CGA is a comprehensive mathematical tool that allows us to handle both conformal geometric transformations and incidence algebra operations in a unified framework. As it has been proven in various studies [10,11], this framework is, therefore, useful in the robotics field to solve in a simple and intuitive way different problems as grasping and inverse kinematics of robot arms [12–15], which are usually handled by the classical geometry. CGA is based on five-dimensional Clifford Algebra (5D CA) that embeds the three-dimensional (3D) Euclidean space in a 5D space adding two extra dimensions, namely, the point at origin and the point at infinity [16-21]. This embedding allows us to identify 3D geometric objects, such as points, lines, circles, and spheres, with specific 5D algebra elements. Moreover, geometric transformations of 3D objects, such as rigid body

motions (reflections, rotations, translations, and dilations), can be easily obtained by applying a unique algebra operator, known as the "sandwich" geometric product, to the algebra elements corresponding to geometric objects to be transformed, while the "meet" operator is used to determine the intersection of two geometric entities in a simple and unified fashion. Existing studies have demonstrated that the CGA-based versions of the grasping and inverse kinematics algorithms show less computational complexity and achieve a competitive runtime performance with respect to the traditional approaches based on the standard projective geometry [10,11]. However, the interaction between robots and human operators in a shared workspace poses important safety issues and requires, therefore, real-time computations and fast response times in the robot reaction to avoid collisions and prevent potential injuries [22,23]. For these reasons, the support of a dedicated hardware accelerator for the native execution of CGA-based geometric calculations is demanded in embedded robotic platforms.

In this paper, a novel embedded coprocessor for accelerating CGA calculations in robotic tasks is presented. Two robotic algorithms, namely, the grasping and inverse kinematics of a human-like robotic arm, are used to demonstrate the effectiveness of the proposed hardware accelerator in this specific application domain. The proposed architecture is the latest evolution in the geometric algebra coprocessor family that we have proposed in the last few years [24–36]. The CliffordALU5 core presented in [33] only supported basic CGA operations as products and sums, while the ConformalALU coprocessor presented in [34] was able to natively execute only rigid body motion operations. Therefore, the ConformalALU was not able to execute basic CGA operations, while rigid body motion operations had to be decomposed into sequences of CGA products and sums to be executed on the CliffordALU5, which led to the slowdown of the high-level applications. However, several real applications, including the robotic algorithms presented as case study in this paper, require both basic CGA operations and composite CGA operations as rigid body motions. For this reason, the CGA coprocessor proposed in this work was conceived and designed to natively support the entire set of CGA operations including basic operations (products, sums, differences, and unary operations), as well as complex operations as rigid body motion operations (reflections, rotations, translations, and dilations). The novel CGA coprocessor embeds, in a single integrated circuit, different dedicated hardware units, including a CGA ALU, based on the CliffordALU5, and a motor unit, based on the ConformalALU, and it introduces pipelining and parallelism techniques to further speedup the CGA operation execution. The coprocessor is designed such that the different hardware units work in parallel and guarantee the parallel execution of multiple different CGA operations. Each hardware unit is a pipeline capable to execute a specific CGA operation. The CGA coprocessor is composed of four pipelines, each dedicated to the execution of one of the following CGA operations: products, sums/differences, unary operations, and rigid body motion operations. The Xilinx ML510 development board containing a Xilinx Virtex-5 XC5VFX130T FPGA device is used to implement the coprocessor prototype. This particular Virtex-5 FPGA chip embeds a PowerPC processing hardcore such that the coprocessor prototype is realized in the form of a complete system-on-chip (SoC) where the PowerPC is used as the general-purpose processor, while the reconfigurable logic of the FPGA is used to implement the special-purpose CGA coprocessor. A design space exploration is performed to analyze and compare several design configurations based on different parameters such as the number of DSP units, LUTs, and slices utilized on the FPGA chip. The different design points are compared in terms of operating frequency, power consumption, and area cost. The experimental tests performed on the coprocessor prototype showed average speedups of about 45× for basic CGA operations and 519× for rigid body motion operations with respect to the use of a software library specialized for the execution of CGA operations running on the standard PowerPC processor. Furthermore, a 78× speedup was observed for the inverse kinematics algorithm exploiting only basic CGA operations, while the grasping algorithm, which requires both basic CGA operations and a significant number of complex rigid body motion operations, achieved

a 246× speedup against the execution on the conventional PowerPC processor. Lastly, the comparison with a different CGA implementation based on the Gaalop software compiler [37–39] generating the high-level C instructions showed execution times comparable or better for the CGA coprocessor.

The main contributions of the paper can be summarized as follows: (i) to the best of our knowledge, the novel coprocessor proposed in this paper is the first specialized architecture able to natively support the entire set of 5D CGA operations and to accelerate complete real-time CGA-based robotic applications. The previous cores were not able to effectively accelerate real-world robotic algorithms, which require the real-time execution of a large number of both basic and complex CGA operations [33,34]; (ii) the new architecture was completely reorganized, integrated, and optimized introducing design optimizations, such as multistage pipelining and multicore parallelism, which led to speedups improved of about one order of magnitude with respect to the prior separated cores (iii) the performance comparison between the coprocessor and a latest-generation CPU, usually utilized for robotics, showed significant results of the CGA coprocessor and proved, for the first time, the real-world applicability of the proposed CGA-based system. Since the widespread adoption of the CGA-based methods has been so far hindered by their high computational complexity, for all the reasons described above, we think that the results of this work open new perspectives on the applicability of CGA in different application domains and provide a significant contribution to this research field.

The rest of the paper is organized as follows: Section 2 summarizes related studies, while Section 3 presents the coprocessor architecture, as well as its FPGA implementation and the design space exploration results. Experimental results are reported in Section 4, while Section 5 contains the discussion and conclusions.

2. Related Work

An introduction to the basic concepts of geometric algebra (GA) and conformal geometric algebra (CGA) can be found in [16–21]. Since CGA operates in the 5D space adding two extra dimensions to the 3D space objects, the use of this algebra leads to high numerical complexity and demands for software tools and/or hardware devices to accelerate CGA operators and guarantee a good runtime performance of CGA-based engineering applications. Several software, mixed software/hardware, and full-hardware implementations of GA have been proposed in the last few years.

The software approaches consist of software packages that can be used to execute GA operations on conventional general-purpose processors. They include the standalone program CluCalc [40], as well as the software libraries Gaigen [41] and GluCat [42]. Several software packages have been also developed to support GA operators in the most used numerical computing environments as Matlab, Maple, and Mathematica: the Clifford Multivector Toolbox [43] and Gable [44] packages for Matlab, the Clifford [45] and GA [46] packages for Maple, and the Grassmann algebra package [47] for Mathematica.

A software/hardware codesign approach was proposed by D. Hildenbrand in [37–39]. This approach exploits the Gaalop pre-compiler to compile and accelerate GA-based algorithms. The Gaalop output consists in a representation of GA operations as parallel basic arithmetic operations. This intermediate representation can be then further compiled to run on various hardware devices, such as FPGAs or GPUs.

Lastly, full-hardware solutions are based on coprocessing architectures capable of directly supporting GA operations. An ASIC implementation of a specialized coprocessor for GA-based edge detection tasks in color images was proposed by Mishra and Wilson [48]. The coprocessor is designed to execute only the 3D GA operations required by the specific color edge detection algorithm. Perwass et al. proposed an FPGA-based coprocessor that supports only geometric product operations for algebras of dimension up to 8 and uses 24 bit integer numbers to represent the GA operand coefficients [49]. The FPGA-based coprocessors presented in [24–27] support 4D GA operations between homogeneous elements. In the first architecture [24], each GA operation is decomposed into the

proper sequence of basic arithmetic operations that are executed by a sequential ALU under the supervision of a microprogrammed control unit, while the second architecture [25–27] contains dedicated parallel units for the native execution of 4D GA products, sums, and differences. A novel representation of 4D GA based on fixed-size elements called quadruples is presented in [28,29], along with a novel coprocessor design capable to support 4D GA operations on quadruples. In [31], the authors proposed a specialized hardware architecture to accelerate the GA-based color edge detection algorithm introduced in [30]. A design space exploration of dedicated hardware architectures for GA was presented in [32], while [35] outlined a coprocessor family for the native support of up to 5D GA operations. The CliffordALU5 coprocessor presented in [33] directly executes 5D GA or CGA basic operations (products, sums/differences, and unary operations) using quadruples as basic elements of computation, while, in [34], the authors presented the ConformalALU coprocessor that natively supports complex CGA operations such as rigid body motion operations by exploiting a novel simplified formulation of these operations. The proof-of-concept coprocessor, named GAPPCO, presented in [36], is a configurable coprocessor conceived to accelerate the parallel arithmetic computations generated by the Gaalop precompiler. Gaalop compiles and optimizes the GA-based algorithms converting GA operations into parallel sums of products and generates the proper configuration data for the FPGA-based GAPPCO coprocessor.

3. Proposed Coprocessor

3.1. Coprocessor Architecture

The proposed CGA coprocessor supports the entire set of CGA operations in the 5D space including basic operations (products, sums, differences, and unary operations) and rigid body motion operations (reflections, rotations, translations, and uniform scaling or dilations). The supported operations, along with the related operation codes, are listed in Table 1.

As depicted in Figure 1, the CGA coprocessor is composed of two pipelined computing units, namely, the CGA ALU that executes basic operations and the motor unit that executes the rigid body motion operations. The CGA ALU contains in turn three pipelined operating blocks for the execution of CGA products, sums/differences and unary operations, respectively. The CGA coprocessor is, therefore, composed of four pipelines that work in parallel and guarantee the parallel execution of multiple different CGA operations. Figure 1 shows also that different read/write FIFO buffers (input FIFO and output FIFO) are used to feed the pipelines and collect the operation results from the pipelines.

Table 1. Supported operations and related opcodes.

Operation Class	Operation	Opcode
Product operations	Geometric product	0000
	Outer or wedge product	0001
	Left contraction	0010
	Right contraction	0011
Sum/difference operations	Sum	0100
	Difference	0101
Unary operations	Dual	1000
	Reverse	1001
	Conjugate	1010
	Grade involution	1011
Rigid body motion operations	Reflection	1100
	Rotation	1101
	Translation	1110

	Dilation	1111

The controller unit supervises the CGA coprocessor operation. It receives the instruction stream from the PowerPC processor, decodes the instruction opcodes, and distributes the instructions to the different pipelines on the basis of the two most significant bits of the opcode that, as described in Table 1, indicate the operation type (00 for product operations, 01 for sum/difference operations, 10 for unary operations, and 11 for rigid body motion operations). The other two bits of the opcode, which define the specific operation, are decoded and used within each operating block.

The four computing units have different execution times. As an example, the unary operations require the shortest execution times, while the motor unit operations take the longest execution times; therefore, it may occur that multiple products, sums/differences, and unary operations are processed during the execution time of a single rigid body motion operation. The high-level API, where the instructions are packed to be redirected to the coprocessor, assigns to each instruction an identification number; when the coprocessor processes the instruction, the same number is attached to the related result to allow the API to properly rearrange the result stream according to the original order of the instructions.

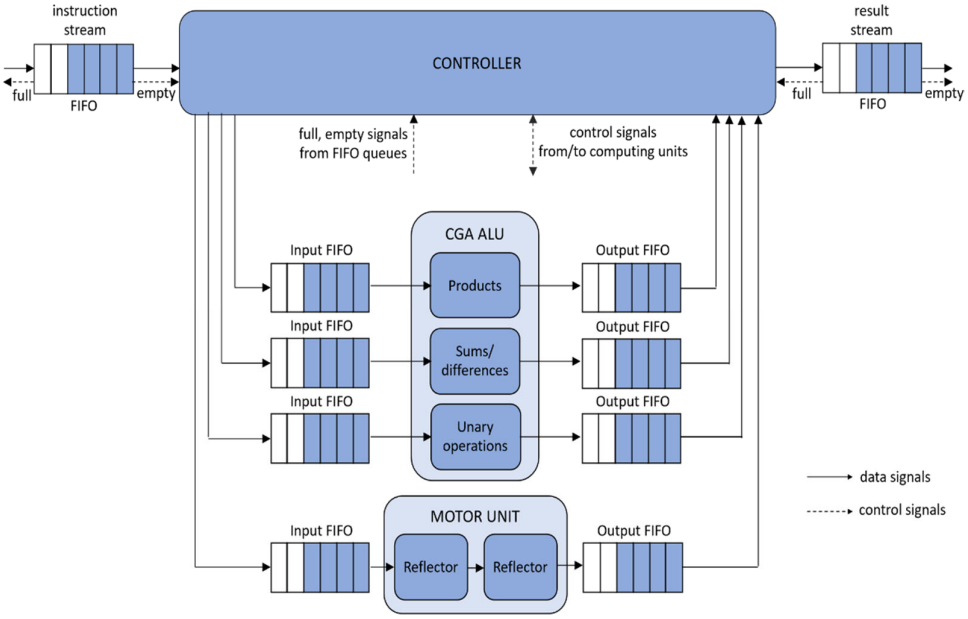

Figure 1. CGA coprocessor block diagram. The controller unit fetches the instructions from the instruction FIFO, decodes the opcodes, distributes the instructions on the four parallel pipelines (the products unit, the sums/differences unit, and the unary operations unit of the CGA ALU, and the motor unit), collects the results from the pipelines, and forwards them to the result FIFO. Input/output FIFO buffers are used to feed the pipelines and receive the results from the pipelines.

The CGA ALU uses quadruples introduced in [33] as basic elements of computation to execute CGA products, sums, and unary operations. The high-level API converts operations on Clifford homogeneous numbers, usually required in high-level CGA-based applications, into operations on quadruples. The motor unit exploits the simplified algorithm introduced in [34] for executing reflection, rotation, translation, and dilation operations. Each rigid body motion operation is reduced to two consecutive reflections executed by two pipelined cascade reflector units. The proposed coprocessor integrates, therefore, into a unique computing architecture the CliffordALU5 unit described in [33], as well as the ConformalALU unit presented in [34]. The two previous architectures were redesigned to be integrated in a novel coprocessor that introduces pipelining and parallelism

techniques to further speedup the CGA operation execution. The FIFO queues were introduced to allow us to exploit the pipelined architecture of the four computing modules (products, sums/differences, unary operations and motor unit), while the controller unit was designed to receive a continuous instruction stream from the PowerPC processor and distribute the instructions to the four pipelines so as to guarantee the parallel execution of multiple different CGA operations. The novel CGA coprocessor supports, therefore, the entire set of CGA operations required by the CGA-based high-level applications as the robotic algorithms presented in Section 4. As e demonstrated in Section 4, the experimental results show that the introduction of pipelining and parallelism in the novel coprocessor, as well as the integration of the dedicated motor unit for accelerating rigid body motion operations, allows for a further speedup factor of the CGA-based algorithms with respect to the use of the two separated CliffordALU5 and ConformalALU cores, while maintaining low values of both the area cost and the power consumption.

CGA basic operations have quadruples as input operands, while rigid body motion operations have 5D vectors as input operands. It has to be observed that we use only 5D vectors as input operands for rigid body motion operations since, as demonstrated in [34] (Eq. 42), each rigid body motion operation on higher-grade operands (bivectors, trivectors, etc.) can be always transformed into a sequence of operations on 5D vectors. Each quadruple has four coefficients, while 5D vectors have five coefficients. Coefficients are represented by 32 bit floating-point numbers coded according to the IEEE 754 specification for single-precision floating-point numbers. Therefore, 1 bit is used for the sign, 8 bits are used for the exponent, and 23 bits are used for the mantissa. The instruction format is reported in Table 2. The ID number field contains the identification number of the instruction. The tag1 and tag2 fields, used only in the case of basic CGA operations, represent the types of the two input quadruples (a 3 bit tag is used to represent the eight possible quadruple types); the opcode field specifies the operation to be executed, as reported in Table 1. Lastly, to represent the input operands, 15 32 bit coefficients are needed. The reason is that, in the case of rigid body motion operations, the input operands consist of three 5D vectors, namely the vector to be transformed, the vector normal to the first reflection plane, and the vector normal to the second reflection plane. In the case of basic CGA operations between quadruples, only the first eight 32 bit coefficients are used to represent the two input quadruples. The basic CGA operations on quadruples give as result one or two output quadruples. Therefore, the result format, reported in Table 3, is composed of the ID number field containing the identification number of the related instruction followed by the tag1 and tag2 fields that represent the types of the two output quadruples, and the eight 32 bit coefficients of the output quadruples. In the case of rigid body motion operations, the tag1 and tag2 fields are not used, and only the first five 32 bit coefficients are used to represent the resulting 5D vector.

Table 2. Instruction format.

ID Number (22 bit)	Tag1 (3 bit)	Tag2 (3 bit)	Opcode (4bit)	Coefficients
32 bit				15 × 32 bit

Table 3. Result format.

Zero Padding (4 bit)	ID Number (22 bit)	Tag1 (3 bit)	Tag2 (3 bit)	Coefficients
32 bit				8 × 32 bit

3.2. FPGA Implementation

The CGA coprocessor prototype, as depicted in Figure 2, is implemented on the Xilinx ML510 development board containing a Xilinx Virtex-5 XC5VFX130T FPGA device. The proposed implementation has the form of a complete system-on-chip (SoC) that integrates both a general-purpose processing core and a CGA specialized coprocessing core on the same Virtex-5 FPGA chip. As the general-purpose processor, the PowerPC440 hardcore available on the Xilinx Virtex-5 XC5VFX130T FPGA device is used, while the CGA coprocessor is implemented exploiting the configurable logic of the same FPGA chip. The CGA coprocessor design is described using the VHDL hardware description language and synthesized using the Xilinx ISE design suite. The Xilinx Platform Studio (XPS) environment is used to configure the embedded system composed of the PowerPC440 hardcore and the configurable coprocessor implemented as a custom peripheral (IP core) connected to the Processor Local Bus (PLB) of the PowerPC processor. The operating frequency is 125 MHz for both the PowerPC processor and the PLB bus. To obtain a higher read/write data transfer rate on the PLB bus, we adopted the *burst* transfer mode, which allows for a transfer rate of 128 bits per clock cycle instead of 32 bits per clock cycle as in the standard *single-beat* data transfer mode. The PowerPC–coprocessor communication is based on the use of read/write FIFO buffers. The system operation consists of the following phases: (1) the PowerPC writes instructions and operands on the instruction FIFO; (2) the coprocessor reads instructions and operands from the instruction FIFO; (3) the coprocessor executes the instructions; (4) the coprocessor writes the results on the result FIFO; (5) the PowerPC reads the results from the result FIFO. Each FIFO has a width of 128 bits and a depth of $2^{14} = 16{,}384$ words. FIFO buffers are implemented by using the 36 kbit block RAM (BRAM) available on the FPGA chip. The RS232 serial interface is used for the PowerPC-terminal standard input/output operations.

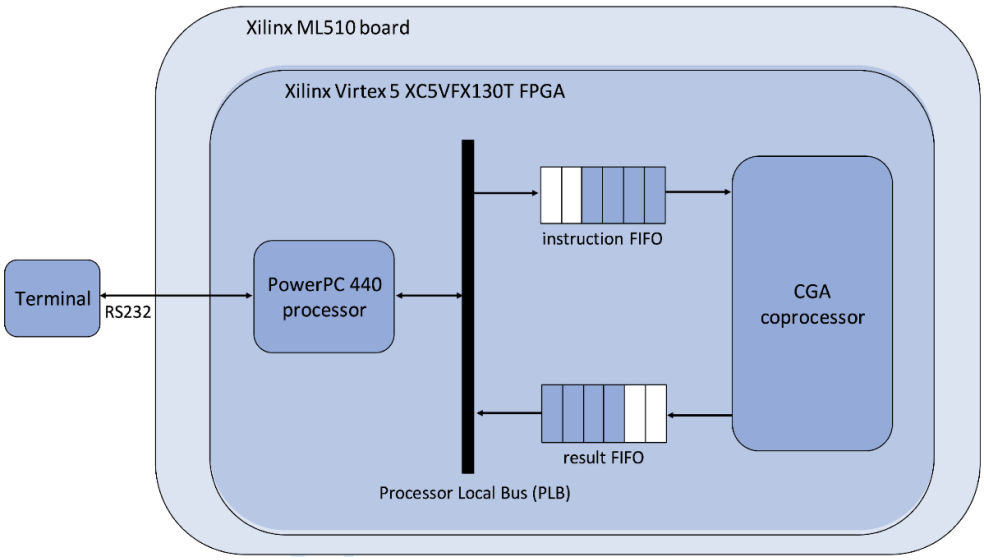

Figure 2. CGA coprocessor prototype on the Xilinx ML510 board.

3.3. Design Space Exploration

The basic floating-point arithmetic operations (products, sums, and differences) needed to execute CGA operations are implemented using the DSP units (DSP48E slices) integrated in the Xilinx Virtex-5 FPGA device. The Xilinx ISE IP Core Generator tool is used to generate floating-point multiplier, adder, and subtractor units. This tool allows the user to configure the floating-point multiply/add units using a different number of DSP units. The possible configurations are as follows: (1) no usage (no DSP unit is used to implement the floating-point arithmetic module); (2) medium usage (one DSP unit is used); (3) full usage (two DSP units are used); (4) max usage (three DSP units are used).

Each floating-point multiplier can be configured to use up to three DSP units, while the floating-point adders/subtractors can use up to two DSP units. A design space exploration is performed to compare several design configurations on the basis of the use of different numbers of DSP units for the implementation of the floating-point arithmetic modules. The different design points are explored and compared in terms of operating frequency, area cost, and power consumption. Results are reported in Table 4 and Figure 3 for eight different CGA coprocessor configurations synthesized and implemented on the Xilinx Virtex-5 FPGA using from zero to three DSP units per multiplier and zero or two DSP units per adder/subtractor.

Table 4. Design space exploration—resource utilization, max frequency, and power consumption for different design configurations using different numbers of DSP units for implementing floating-point multipliers, adders, and subtractors.

	Design 1 **Multipliers 0 DSP** **Add/Sub 0 DSP**	**Design 2** **Multipliers 1 DSP** **Add/Sub 0 DSP**	**Design 3** **Multipliers 2 DSP** **Add/Sub 0 DSP**	**Design 4** **Multipliers 3 DSP** **Add/Sub 0 DSP**	**Design 5** **Multipliers 0 DSP** **Add/Sub 2 DSP**	**Design 6** **Multipliers 1 DSP** **Add/Sub 2 DSP**	**Design 7** **Multipliers 2 DSP** **Add/Sub 2 DSP**	**Design 8** **Multipliers 3 DSP** **Add/Sub 2 DSP**
Slices	13,935 68%	11,304 55%	10,123 49%	9508 46%	12,294 60%	9370 46%	8199 40%	7252 35%
LUTs	32,090 39%	22,139 27%	17,852 22%	17,491 21%	27,419 33%	17,227 21%	12,840 16%	12,466 15%
Flip-flops	34,973 43%	26,640 33%	21,459 26%	19,590 24%	29,193 36%	20,759 25%	15,515 19%	13,623 17%
DSP48E slices	0 0%	26 8%	52 16%	78 24%	54 17%	80 25%	106 33%	132 41%
Max frequency (MHz)	183	192	162	172	159	149	173	162
Total dynamic power (W)	0.356	0.364	0.325	0.344	0.320	0.308	0.345	0.325
Total quiescent power (W)	4.208	4.208	4.208	4.208	4.208	4.208	4.208	4.208
Total power (W)	4.564	4.572	4.533	4.552	4.528	4.516	4.553	4.533

It can be observed that, when the number of DSP units used in the floating-point modules increases, the utilization of the general-purpose resources (slices, LUTs, and flip-flops) reduces. Considering that the power consumption of the various configurations is almost constant, and the frequency is always above 125 MHz (i.e., the frequency we chose

since it is the maximum operating frequency of the PLB bus), we chose for the final coprocessor design the configuration with the max usage of DSP units (three DSP units for multipliers and two DSP units for adders and subtractors), since it guarantees the lowest area cost (in terms of FPGA slices) allowing for a future dual-core implementation of the CGA coprocessor on the considered Virtex-5 FPGA chip or a multicore implementation on a FPGA device with more resources.

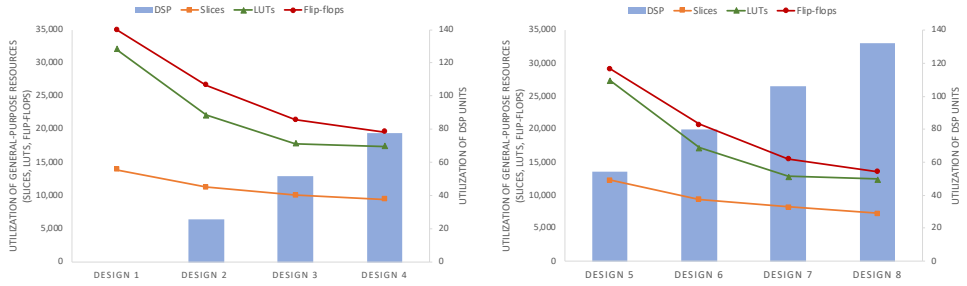

Figure 3. Design space exploration—utilization of DSP units and general-purpose resources (slices, LUTs, and flip-flops) for eight different design points using different numbers of DSP units for each floating-point multiply/add module: design 1 (zero DSP per multiplier, zero DSP per adder/subtractor), design 2 (one DSP per multiplier, zero DSP per adder/subtractor), design 3 (two DSP per multiplier, zero DSP per adder/subtractor), design 4 (three DSP per multiplier, zero DSP per adder/subtractor), design 5 (zero DSP per multiplier, two DSP per adder/subtractor), design 6 (one DSP per multiplier, two DSP per adder/subtractor), design 7 (two DSP per multiplier, two DSP per adder/subtractor), and design 8 (three DSP per multiplier, two DSP per adder/subtractor).

4. Experimental Results

4.1. Performance Analysis

The coprocessor performance was evaluated in terms of area cost, power consumption, and speedup with respect to the execution on a standard general-purpose processor.

4.1.1. Resource Utilization

Regarding the resource utilization, three different architectures were implemented on the Xilinx ML510 board and compared in terms of area cost (occupied FPGA slices, LUTs, flip-flops, DSP units, and block RAM): the architecture containing only the CGA ALU (based on CliffordALU5 [33]), the architecture containing only the motor unit (based on ConformalALU [34]), and the novel CGA coprocessor presented in this paper integrating both the CGA ALU and motor unit, redesigned to exploit pipelining and parallelism and guarantee the parallel execution of multiple CGA operations. The resource utilization of the architectures on the Xilinx Virtex-5 XC5VFX130T FPGA device is reported in Table 5.

It can be observed that the two architectures presented in [33,34] were already designed to support pipelining, but they were tested for scalar execution, without exploiting pipelining (i.e., one instruction at a time was completed by the coprocessor before to process the following instruction). In the novel coprocessor, the introduction of the FIFO queues using the block RAM (BRAM) of the Virtex-5 FPGA device allows us to fully exploit the pipelined architecture of both CGA ALU and motor unit. Furthermore, the CGA coprocessor is designed so as to exploit more dedicated DSP slices and save general purpose resources such as slices and LUTs. It can be observed that the percentage of occupied slices and LUTs of the CGA coprocessor is lower than the sum of the percentages of occupied slices and LUTs of the previous architectures CliffordALU5 and ConformalALU, respectively.

Table 5. Resource utilization on the Xilinx Virtex-5 XC5VFX130T FPGA device.

	CGA ALU (Based on CliffordALU5 [33])	Motor unit (Based on ConformalALU [34])	CGA Coprocessor
Slices	4576 22%	4754 23%	7252 35%
LUTs	8642 10%	8109 9%	12,466 15%
Flip-flops	10,721 13%	10,167 12%	13,623 17%
DSP48E slices	32 10%	20 6%	132 41%
36 kbit Block RAM	-	-	256 86%

4.1.2. Power Consumption

The power consumption of the CGA coprocessor on the Xilinx Virtex-5 FPGA chip was measured using the Xilinx ISE XPower Analyzer tool. Table 6 reports the power consumption for the three architectures described in Section 4.1.1: the CGA ALU, the motor unit, and the novel parallel CGA coprocessor proposed in this paper integrating both the CGA ALU and the motor unit. It can be observed that, in the novel CGA coprocessor, the three computing units (products, sums/differences, and unary operations) of the CGA ALU work in parallel such that we have four pipelines (the three pipelines of the CGA ALU and the pipeline of the motor unit) working in parallel, while, in the old versions of the architectures, only one computing unit at a time was used. The parallel operation of the new coprocessor leads to higher values of the power consumption, as reported in Table 6.

Table 6. Power consumption on the Xilinx Virtex-5 XC5VFX130T FPGA device.

	CGA ALU	Motor Unit	CGA Coprocessor
Total quiescent power	2.271 W	2.273 W	4.208 W
Total dynamic power	0.085 W	0.130 W	0.325 W
Total power	2.356 W	2.403 W	4.533 W

4.1.3. Speedup

To measure the speedup achieved by the CGA coprocessor against the execution on a standard general-purpose CPU, the same CGA operations are executed on both the PowerPC440 processor available on the Xilinx Virtex-5 FPGA device and the CGA coprocessor. The Gaigen software library [41] is used for the execution of CGA operations on the PowerPC processor. Gaigen (Geometric Algebra Implementation Generator) is a library generator that allows the user to generate optimized software libraries for the execution of specific geometric algebras by setting several parameters such as algebra dimension, metric, and signature. In our case, the Gaigen generator is used to derive the specific software library optimized for the execution of CGA operations in the 5D space. Two different systems were designed and implemented on the Xilinx ML510 board: the former is a fully software system based on the PowerPC processor and the Gaigen software library, while the latter is a mixed software/hardware system that uses both the PowerPC processor and the specialized CGA coprocessor. In the latter system, the Gaigen software library running on the PowerPC is modified so as to redirect CGA operations to the coprocessor. Several

experimental tests are performed to measure the execution times of different CGA operations, including basic operations as products, sums/differences, and unary operations on 5D homogeneous elements (scalars S, vectors V, bivectors BV, trivectors TV, pseudovectors PV, and pseudoscalars PS), as well as reflections, rotations, translations, and dilations of 5D vectors. Both the PowerPC processor and the CGA coprocessor operate at the same frequency of 125 MHz on the same gate technology such that a direct comparison of execution times is derived. Execution times, measured in clock cycles at 125 MHz, and related speedups are reported in Table 7 for a selection of CGA operations.

The average speedups are 45× for basic CGA operations and 519× for rigid body motion operations. Regarding translation, rotation, and dilation operations, we can observe that most of the execution time of the coprocessor (81.8% on average) is taken by the instruction preparation in the high-level API, which has to calculate the two reflection vectors with respect to which the two successive reflections have to be performed starting from the translation direction and distance (for translations), the rotation plane and angle (for rotations), or the scaling factor (for dilations).

Table 7. Execution times measured in clock cycles at 125 MHz—Gaigen/CGA coprocessor comparison (reported values are average values related to 10,000 executions of each operation on randomly generated operands).

Operation	Operand 1	Operand 2	Gaigen Software Library (t_1)	CGA Coprocessor (t_2)	Speedup (t_1/t_2)
Outer product	S	V	5412	182	29.73×
Outer product	V	V	43,503	710	61.27×
Outer product	V	BV	73,425	1434	51.20×
Outer product	BV	BV	79,795	1567	50.92×
Left contraction	S	V	5496	178	30.87×
Left contraction	V	V	11,695	309	37.84×
Left contraction	V	BV	50,921	1034	49.24×
Left contraction	BV	BV	25,261	486	51.97×
Left contraction	BV	TV	79,878	1616	49.42×
Left contraction	TV	TV	25,075	480	52.23×
Left contraction	PV	PV	10,329	158	65.37×
Right contraction	V	V	11,691	305	38.33×
Right contraction	BV	BV	25,294	491	51.51×
Geometric product	S	V	5401	181	29.83×
Geometric product	V	V	54,962	856	64.20×
Geometric product	V	BV	124,361	2120	58.66×
Geometric product	BV	BV	270,562	4612	58.66×
Sum	V	V	6211	185	33.57×
Sum	BV	BV	14,998	575	26.08×
Subtraction	V	V	6426	184	34.92×
Subtraction	BV	BV	15,908	575	27.66×

Average			**45,076**	**868**	**45.40×**
Reflection	V	-	646,306	566	1141.88×
Translation	V	-	1,174,200	3893	301.61×
Rotation	V	-	1,841,440	3878	474.84×
Dilation	V	-	608,531	3821	159.25×
Average			**1,067,619**	**3039**	**519.39×**

To reduce this preparation time in the high-level API, we precalculated the reflection vectors to be included in the instructions for different input data and stored them in precompiled tables. As an example, the precompiled table related to the translation operations (used in the grasping application) contains the precalculated reflection vectors needed to translate the base circle of the object to be grasped by different distances (expressed in cm) in the given translation direction. The use of the precalculated tables allowed us to reduce the instruction preparation time of about 80%. This choice, along with the use of the pipelined execution, led to speedups higher of about one order of magnitude with respect to the previous architecture ConformalALU.

According to the standard formulation of conformal geometric algebra, the rigid body motion operations are obtained by the so-called "sandwich" product consisting in pre-multiplying and post-multiplying the entity to be transformed (5D vector) by the versor that represents the geometric transformation (rotor for rotations, translator for translations, and dilator for dilations). This standard formulation of rigid body motion operations requires a sequence of basic CGA operations (geometric products) that can be executed on the CGA ALU. Conversely, the motor unit executes each rigid body motion operation as a whole operation using the novel formulation that requires two successive reflection operations [34]. We experimentally measured the average execution times required to execute a rotation operation using both the standard formulation on the pipelined CGA ALU and the novel formulation on the pipelined motor unit integrated in the CGA coprocessor. This comparison is reported in Table 8. This result confirms that the integration of the motor unit as an accelerator for rigid body motion operations within the CGA coprocessor leads to a further 12× speedup factor with respect to the execution of the same rigid body motion operations on the basic CGA ALU.

Table 8. Comparison of average execution times (measured in clock cycles at 125 MHz) of a rotation operation on the CGA ALU and on the motor unit, along with related speedup.

Rotation on CGA ALU (t1)	Rotation on Motor Unit (t2)	Speedup (t1/t2)
46,360	3878	12×

4.2. Robotic Applications

To evaluate the CGA coprocessor effectiveness in a real-world application domain, two CGA-based robotic algorithms, namely, grasping and inverse kinematics of a human-arm-like kinematics chain, were executed on the coprocessor [7–9]. These applications were chosen since they massively use CGA operations that can be accelerated by the CGA coprocessor. The kinematics chain used in the experiments is depicted in Figure 4. It has five degrees of freedom given by five joint angles: q1 (rotation of the robot around y-axis), q2, q3, and q4, (angles between the links and between the last link and the end effector or gripper), and q5 (rotation of the gripper). In general, the grasping task consists of calculating the proper contact point that the gripper has to reach to grasp the object starting from a calibrated stereo pair of images of the object, while the inverse kinematics is used

to derive the right positions of the robot joints that allow the gripper to reach the target point in the 3D space.

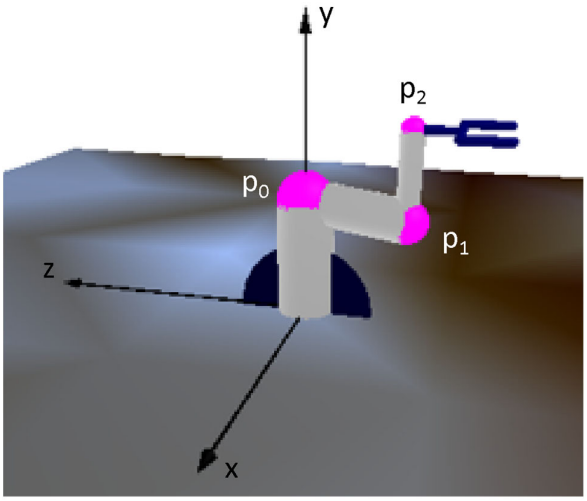

Figure 4. The human-arm-like kinematics chain.

Several studies [7–11] have demonstrated that the reformulation of these robotic algorithms within the CGA framework allows for simplification, intuitiveness, and better computational performance. However, the hardware support of a specialized coprocessor capable of natively executing CGA operations is fundamental for real-time applications where the interaction of collaborative robots with human operators poses safety issues and demands fast response times of the involved robots to avoid collisions and prevent injuries.

The CGA-based versions of the proposed algorithms were formulated to exploit the simplified geometric constructions of CGA, whose basic geometric entities are circles and spheres. Operating in the 5D conformal space allows us to reduce geometric objects to 5D algebra elements (spheres and planes are 5D vectors, while circles are 5D bivectors) and geometric transformations to 5D algebra operators (as an example, translations and rotations are obtained by the "sandwich" geometric product, while intersections are obtained by the "meet" operator).

As illustrated in Figure 5, the grasping algorithm takes a calibrated stereo pair of images of the object to be grasped, i.e., a regular prism in our application, and extracts four non-coplanar points belonging to the corners of the object from these images. The corresponding 3D points are derived using triangulation. Then, the distances between all four points and the plane spanned by the three other points are computed. The point with the greatest distance *da* is the apex point, while the other three points are those belonging to the base of the object. The base circle through the base points of the object is then computed and translated in the direction and magnitude of da/2 to produce the grasping circle. Lastly, the point of contact to be reached by the gripper is calculated as the closest point of the grasping circle to the *y*-axis (see [7–9] for a detailed description of the CGA-based grasping algorithm).

Starting from the target point, the inverse kinematics algorithm calculates the joint points (p_0, p_1, and p_2) and joint angles, as well as the quaternions needed to rotate the robot and reach the proper joint positions by using geometric entities as circles, spheres, and planes and operating on them by CGA operators (Figure 6 shows a visual simulation using CluCalc [40]). A more detailed description of the CGA-based formulation of the inverse kinematics algorithm can be found in [7–9]. An accurate profiling of the proposed algorithms was performed to find the CGA operations most frequently required during

the algorithm execution and the related execution times. The profiling results are summarized in Tables 9 and 10 for the grasping and inverse kinematics algorithms, respectively. As it can be observed, the inverse kinematics algorithm requires only basic CGA operations, while the grasping algorithm requires both basic CGA operations and complex CGA operations, particularly translation operations. Translation operations take 80.78% of the total execution time.

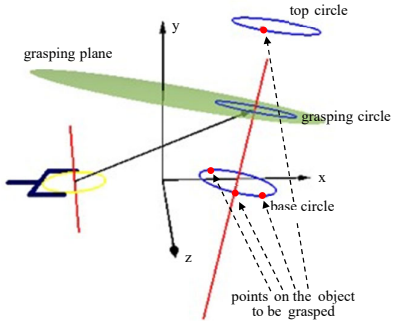

Figure 5. Grasping algorithm.

Table 9. Grasping algorithm profiling.

Operation	Operands	% of Required Operations	% of Execution Time
Geometric product	S-V	7.5%	0.16%
	V-V	12.5%	2.70%
	V-BV	10%	4.89%
	BV-TV	5%	5.32%
Outer product	V-V	10%	1.71%
	V-BV	2.5%	0.72%
	V-TV	2.5%	0.72%
	BV-BV	5%	1.57%
Left contraction	S-V	2.5%	0.05%
	V-V	10%	0.46%
	V-BV	2.5%	0.50%
Dual	BV	2.5%	0.10%
	TV	5%	0.21%
	PV	5%	0.11%
Translation	V	17.5%	80.78%

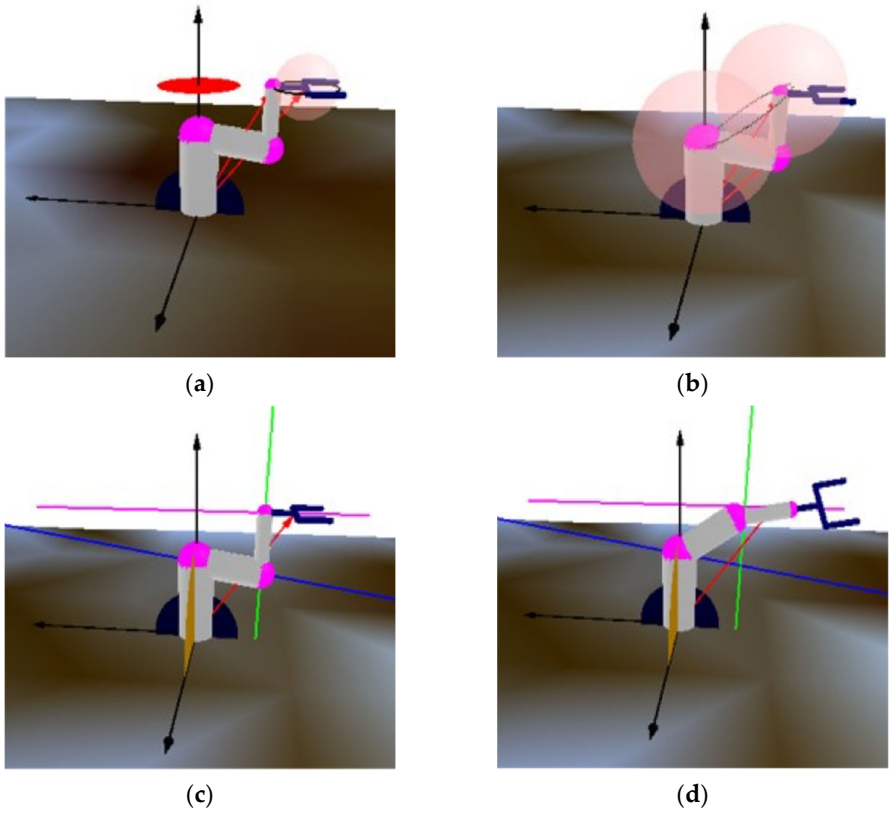

Figure 6. Inverse kinematics algorithm: (**a**) computation of the joint point p_2; (**b**) computation of the joint point p_1; (**c**) computation of the joint angles; (**d**) robot rotation.

The two algorithms were executed on the full-software system based on the PowerPC processor, as well as on the mixed software/hardware system based on the PowerPC processor and the CGA coprocessor. Regarding the execution on the coprocessor, we modified the two algorithms to exploit as much as possible the parallelism of the CGA coprocessor: in both algorithms, the instructions were reordered so as to execute in parallel all the operations that could be parallelized. The measured execution times and the achieved speedups are listed in Table 11.

The grasping algorithm shows a higher speedup since it exploits the further acceleration provided by the motor unit for the execution of translation operations. For further validation, the grasping algorithm was executed on the CGA ALU only, as well as on the complete CGA coprocessor composed of both the CGA ALU and the motor unit. The comparison results are reported in Table 12. As can be observed, the use of the motor unit for the execution of rigid body motion operations allows for a further 5.1× speedup. These results confirm the usefulness to integrate the specialized motor unit into the CGA coprocessor.

Table 10. Inverse kinematics algorithm profiling.

Operation	Operands	% of Required Operations	% of Execution Time
Geometric product	S-V	12.8%	1.85%
	V-V	4.3%	6.32%
Outer product	V-V	17%	19.79%
	V-BV	17%	33.39%

	BV-BV	2.1%	4.48%
Left contraction	V-V	6.4%	2.00%
	V-BV	4.3%	5.86%
	BV-BV	4.3%	2.91%
	BV-TV	4.3%	9.19%
	TV-TV	19%	12.75%
Subtraction	V-V	8.5%	1.46%

Table 11. Algorithm speedup.

Algorithm	Execution Time on PowerPC (t_1)	Execution Time on CGA Coprocessor (Parallel Execution) (t_2)	Speedup (t_1/t_2)
Grasping	81.4 ms	0.33 ms	246.67×
Inverse kinematics	30 ms	0.38 ms	78.94×

Table 12. Grasping algorithm execution times on CGA ALU and on CGA ALU + motor unit and related speedup.

Grasping Algorithm on CGA ALU (t_1)	Grasping Algorithm on CGA ALU + Motor Unit (t_2)	Speedup (t_1/t_2)
1.7 ms	0.33 ms	5.1×

To further evaluate the coprocessor effectiveness and its real-world applicability, we compared its performance with a latest-generation CPU implementation, usually used for robotics. Table 13 reports the comparison between the execution times of the robotic algorithms on the CGA coprocessor and on an Intel quad-core i7 CPU running at 2.9 GHz. Since the two systems operate at different clock frequencies, we could measure the potential speedup of the coprocessor by comparing the execution times expressed in clock cycles.

Lastly, we also compared our coprocessor performance with the CGA implementation based on the Gaalop software compiler [37–39]. We used Gaalop to compile the grasping and inverse kinematics algorithms. The optimized C code of the two algorithms provided by Gaalop was executed on the PowerPC processor of the Xilinx ML510 board, and its runtime performance was compared with the CGA coprocessor. Results of this comparison are listed in Table 14, where Gaalop compilation times are not considered, and only the execution times after the compilation phase are reported.

Table 13. Comparison between CGA coprocessor and Intel quad-core I7 CPU.

Algorithm	Execution Time on Intel Quad-Core i7 CPU (t_1) (Expressed in Clock Cycles at 2.9 GHz)	Execution Time on CGA Coprocessor (t_2) (Expressed in Clock Cycles at 125 MHz)	Speedup (t_1/t_2)
Grasping	3,862,800	41,250	93×
Inverse kinematics	1,174,500	47,500	24×

Table 14. Gaalop/CGA coprocessor comparison.

Algorithm	Execution Time of the Gaalop-Optimized Algorithm on PowerPC	Execution Time of the Algorithm on CGA Coprocessor
Grasping	0.55 ms	0.33 ms
Inverse kinematics	0.37 ms	0.38 ms

As can be observed, the execution times of the inverse kinematics algorithm are comparable, while a better runtime performance of the CGA coprocessor was observed for the grasping algorithm. As already remarked above, the grasping algorithm involves many translation operations (above 80%); hence, it exploits the higher speedup provided by the Motor unit.

5. Discussion and Conclusions

A novel embedded coprocessor to accelerate conformal geometric algebra (CGA) operations in robotic tasks was presented in the paper. The proposed CGA coprocessor integrates two cores into a single computing architecture: the first core, namely, the CGA ALU (based on the CliffordALU5 unit described in [33]), natively supports basic CGA operations (products, sums, and unary operations), while the second core, namely, the motor unit (based on the ConformalALU unit presented in [34]), executes directly in hardware complex CGA operations, such as rigid body motion operations (reflections, rotations, translations, and uniform scaling). The novel architecture is, therefore, able to natively support the entire set of CGA operations in the 5D space. The novel coprocessor was designed as a parallel pipelined architecture composed of four pipelines that work in parallel to execute multiple CGA operations simultaneously. The four pipelines support CGA products, sums/differences, unary operations, and rigid body motion operations. The coprocessor prototype was implemented as a system-on-chip (SoC) on the Xilinx ML510 platform containing both a PowerPC processor and a Xilinx Virtex-5 FPGA device. The reconfigurable logic of the FPGA device was used to place the CGA coprocessor connected to the processor local bus of the PowerPC. The block RAM of the target Virtex-5 FPGA device was exploited to implement a set of FIFO buffers capable of feeding the pipelines and collect results from the pipelines, while the *burst* transfer mode, which allows for a transfer rate of 128 bits per clock cycle instead of 32 bits per clock cycle as in the standard single beat transfer mode, was adopted to speedup the transfer of instructions and data between the PowerPC and the coprocessor. The introduction of pipelining and parallelism in the novel CGA coprocessor allowed us to achieve a better runtime performance with respect to the previous CliffordALU5 and ConformalALU cores that were implemented and tested to receive from the PowerPC processor a single CGA instruction at a time, execute this single instruction, and provide back the result to the PowerPC before to receive the next instruction to be executed.

The observed average speedups of the novel CGA coprocessor over the software execution were 45× for basic CGA operations and 519× for rigid body motion operations. The proposed CGA coprocessor was used to accelerate two CGA-based robotic algorithms, namely, the grasping and the inverse kinematics of a robotic arm, which require a mix of basic CGA operations and rigid body motion operations. A 78× speedup was observed for the inverse kinematics algorithm exploiting only basic CGA operations, while the grasping algorithm, which requires both basic CGA operations and rigid body motion operations, achieved a 246× speedup against the execution on the conventional PowerPC processor.

As reported in Table 14, the proposed CGA coprocessor also showed execution times comparable or better when compared with the CGA implementation based on the soft-

ware Gaalop [37–39], although these results do not consider the compilation times required by the Gaalop pre-compiler, but only the execution times after the compilation phase.

Table 15 summarizes the main features and performance figures of the previously presented cores (CliffordALU5 [33] and ConformalALU [34]) and the new coprocessor proposed in this work, when the three architectures were implemented on the same Xilinx Virtex-5 XC5VFX130T FPGA device.

As can be observed, although the new CGA coprocessor integrates the two previous cores (CliffordALU5 and ConformalALU), it achieves better performance in terms of area cost. The reason is that the previous hardware modules were redesigned to exploit as much as possible the DSP slices available on the Virtex-5 FPGA device for the execution of floating-point arithmetic operations so as to save general-purpose resources (slices). The power consumption of the proposed coprocessor is higher with respect to the two previous separated cores. The reason is that the new coprocessor is composed of four pipelined units that work in parallel, while, in the previous architectures, only one hardware unit worked at a time and the other units were idle. Thanks to the introduction of pipelining and instruction parallelism, the novel coprocessor also presents better speedups over software when compared with the previous architectures. Regarding the speedups of the robotic applications, it can be observed that the grasping algorithm, which massively uses complex CGA operations as translation operations, shows a better speedup since it exploits the further acceleration given by the motor unit for the execution of translation operations. Observed results confirm, therefore, that the new proposed computing architecture, based on the integration of the specialized motor unit into the general-purpose CGA ALU and on the introduction of pipelining and parallelism techniques, provides several benefits in terms of both algorithm speedup and area cost.

The proposed coprocessor can be used not only for robotic applications, but also in other application domains that require the fast and efficient execution of a large number of geometric operations, such as computer graphics, computer vision, and image processing. With particular regard to robotic algorithms, the CGA framework offers a simple and intuitive mathematical tool to handle the robot kinematics and dynamics, while the hardware acceleration provided by the proposed CGA architecture improves the real-time performance required in embedded robotic platforms.

Table 15. Comparison of CliffordALU5, ConformalALU, and the proposed CGA coprocessor. Reported performance figures are related to the implementation of the three coprocessors on the same Xilinx Virtex-5 XC5VFX130T FPGA device.

	CliffordALU5 [33]	**ConformalALU [34]**	**Proposed CGA Coprocessor**
Supported Operations	Basic CGA operations (products, sums, differences, and unary operations)	Rigid body motion operations (reflections, rotations, translations, and dilations)	Both basic CGA operations and rigid body motion operations
Pipelining	No (predisposed, but not used)	No (predisposed, but not used)	Yes
Parallelism	No	No	Yes
N. of slices	4576 (22%)	4754 (23%)	7252 (35%)
No. of DSP48E slices	32 (10%)	20 (6%)	132 (41%)
No. of Block RAM	-	-	256 (86%)
Power consumption	2.356 W	2.403 W	4.533 W

Future developments will be aimed at GPU algorithm implementations by selecting modern GPU technology requiring less power consumption and making it compatible with an autonomous robotic system [50,51].

Author Contributions: Conceptualization, S.V., S.F., and G.V.; methodology, S.V., S.F., and G.V.; implementation, simulation, and validation, S.F. and S.V.; writing—original draft preparation, S.F.; writing—review and editing, S.V. and S.F.; supervision, S.V.; project administration, S.V. All authors read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: All data, models, or code that support the findings of this study are available from the corresponding author upon reasonable request.

Acknowledgments: We would like to thank the editors and the anonymous reviewers for their insightful comments and constructive suggestions.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Thoben, K.D.; Wiesner, S.; Wuest, T. Industrie 4.0 and smart manufacturing—A review of research issues and application examples. *Int. J. Autom. Technol.* **2017**, *11*, 4–16. https://doi.org/10.20965/ijat.2017.p0004.
2. Robla-Gómez, S.; Becerra, V.M.; Llata, J.R.; Sarabia, E.G.; Torre-Ferrero, C.; Pérez-Oria, J. Working Together: A Review on Safe Human-Robot Collaboration in Industrial Environments. *IEEE Access* **2017**, *5*, 26754–26773. https://doi.org/10.1109/ACCESS.2017.2773127.
3. Zanchettin, A.M.; Croft, E.; Ding, H.; Li, M. Collaborative Robots in the Workplace. *IEEE Robot. Autom. Mag.* **2018**, *25*, 16–17. https://doi.org/10.1109/MRA.2018.2822083.
4. Fryman, J.; Matthias, B. Safety of industrial robots: From conventional to collaborative applications. In *German Conference on Robotics*; ROBOTIK: Munich, Germany, 2012, pp. 1–5.
5. Reza, N.J. *Theory of Applied Robotics: Kinematics, Dynamics, and Control*, 2nd ed.; Springer: New York, NY, USA, 2010.
6. Ruf, A.; Horaud, R. Closing the loop between articulated motion and stereo vision: A projective approach, Ph.D Thesis, INP: Grenoble, France, 2000.
7. Zamora, J.; Corrochano, E.B. Inverse Kinematics, Fixation and Grasping using Conformal Geometric Algebra. In Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Sendai, Japan, 28 September–2 October 2004; pp. 3841–3846, Volume 4. https://doi.org/10.1109/IROS.2004.1390013.
8. Hildenbrand, D.; Bayro-Corrochano, E.; Zamora, J. Advanced Geometric Approach for Graphics and Visual Guided Robot Object Manipulation. In Proceedings of the 2005 IEEE International Conference on Robotics and Automation, Barcelona, Spain, 18-22 April 2005; pp. 4727–4732. https://doi.org/10.1109/ROBOT.2005.1570850.
9. Hildenbrand, D.; Zamora, J.; Bayro-Corrochano, E. Inverse Kinematics Computation in Computer Graphics and Robotics Using Conformal Geometric Algebra. *Adv. Appl. Clifford Algebr.* **2008**, *18*, 699–713. https://doi.org/10.1007/s00006-008-0096-5.
10. Hildenbrand, D.; Fontijne, D.; Wang, Y.; Alexa, M.; Dorst, L. Competitive runtime performance for inverse kinematics algorithms using conformal geometric algebra. In Proceedings of the European Association for Computer Graphics (Eurographics), Vienna, Austria, 4–8 September 2006; pp. 5–9.
11. Wo¨rsdo¨rfer, F.; Stock, F.; Bayro-Corrochano, E.; Hildenbrand, D. Optimizations and Performance of a Robotics Grasping Algorithm Described in Geometric Algebra. In *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*; Bayro-Corrochano, E., Eklundh, J.O., Eds.; Springer: Berlin/Heidelberg, Germany; CIARP 2009. Lecture Notes in Computer Science, Volume 5856, 2009.
12. Pengwen, X.; Kongfei, H.; AiGuo, S.; Peter, L. Robotic Haptic Adjective Perception Based on Coupled Sparse Coding. *Sci. China Inf. Sci.* **2022**. doi: 10.1007/s11432-021-3512-6.
13. Xiong, P.; Tong, X.; Song, A.; Liu, P.X. Robotic Multifinger Grasping State Recognition Based on Adaptive Multikernel Dictionary Learning. *IEEE Trans. Instrum. Meas.* **2022**, *71*, 2511014. https://doi.org/10.1109/TIM.2022.3178500.
14. Xiong, P.; Liao, J.; Zhou, M.; Song, A.; Liu, P.X. Deeply Supervised Subspace Learning for Cross-Modal Material Perception of Known and Unknown Objects. *IEEE Trans. Ind. Inform.* **2022**, 1–10. https://doi.org/10.1109/TII.2022.3195171.
15. Xiong, P.; He, K.; Wu, E.Q.; Zhu, L.-M.; Song, A.; Liu, P.X. Human-Exploratory-Procedure-Based Hybrid Measurement Fusion for Material Recognition. *IEEE/ASME Trans. Mechatron.* **2022**, *27*, 1093–1104. https://doi.org/10.1109/TMECH.2021.3080378.
16. Clifford, W.K. On the Classification of Geometric Algebras. In *Mathematical Papers*; Tucker, R., Ed.; Macmillian: London, UK, 1882; pp. 397-401.
17. Hestenes, D. *New Foundations for Classical Mechanics*; Kluwer Academic: Dordrecht, The Netherlands, 1986.
18. Hestenes, D.; Sobczyk, G. *Clifford Algebra to Geometric Calculus: A Unified Language for Mathematics and Physics*; Kluwer Academic: Dordrecht, The Netherlands, 1987.

19. Dorst, L.; Mann, S. Geometric Algebra: A Computational Framework for Geometrical Applications (Part 1: Algebra). *IEEE Comput. Graph. Appl.* **2002**, *22*, 24–31. https://doi.org/10.1109/MCG.2002.999785.
20. Dorst, L.; Mann, S. Geometric Algebra: A Computational Framework for Geometrical Applications (Part 2: Applications). *IEEE Comput. Graph. Appl.* **2002**, *22*, 58–67. https://doi.org/10.1109/MCG.2002.1016699.
21. Dorst, L.; Fontijne, D.; Mann, S. *Geometric Algebra for Computer Science: An Object Oriented Approach to Geometry*; Morgan Kaufmann: Burlington, MA, USA, 2007.
22. Haddadin, S.; Albu-Schaeffer, A.; Hirzinger, G. Requirements for safe robots: Measurements, analysis and new insights. *Int. J. Robot. Res.*, **2008**, *28*, 1507–1527. https://doi.org/10.1177/0278364909343970.
23. Zanchettin, A.M.; Ceriani, N.M.; Rocco, P.; Ding, H.; Matthias, B. Safety in human-robot collaborative manufacturing environments: Metrics and control. *IEEE Trans. Autom. Sci. Eng.* **2016**, *13*, 882–893, April 2016. https://doi.org/10.1109/TASE.2015.2412256.
24. Franchini, S.; Gentile, A.; Grimaudo, M.; Hung, C.A.; Impastato, S.; Sorbello, F.; Vassallo, G.; Vitabile, S. 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, 29–31 August 2007; pp. 436–439. https://doi.org/10.1109/DSD.2007.4341505.
25. Gentile, A.; Segreto, S.; Sorbello, F.; Vassallo, G.; Vitabile, S.; Vullo, V. CliffoSor: A Parallel Embedded Architecture for Geometric Algebra and Computer Graphics. In Proceedings of the IEEE International Workshop on Computer Architecture for Machine Perception (CAMP 2005), Palermo, Italy, 4–6 July 2005, pp. 90–95.
26. Gentile, A.; Segreto, S.; Sorbello, F.; Vassallo, G.; Vitabile, S.; Vullo, V. CliffoSor, an Innovative FPGA-based Architecture for Geometric Algebra. In Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms, (ERSA 2005), Las Vegas, Nevada, USA, 27-30 June 2005; pp. 211–217.
27. Franchini, S.; Gentile, A.; Sorbello, F.; Vassallo, G.; Vitabile, S. An Embedded, FPGA-based Computer Graphics Coprocessor with Native Geometric Algebra Support. *Integr. VLSI J.* **2009**, *42*, 346–355. https://doi.org/10.1016/j.vlsi.2008.09.010.
28. Franchini, S.; Gentile, A.; Sorbello, F.; Vassallo, G.; Vitabile, S. An FPGA Implementation of a Quadruple-Based Multiplier for 4D Clifford Algebra. In Proceedings of the 11th IEEE Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2008), Parma, Italy, 3–5 September 2008; pp. 743–751. https://doi.org/10.1109/DSD.2008.91.
29. Franchini, S.; Gentile, A.; Sorbello, F.; Vassallo, G.; Vitabile, S. Fixed-Size Quadruples for a New, Hardware-Oriented Representation of the 4D Clifford Algebra. *Adv. Appl. Clifford Algebr.* **2011**, *21*, 315–340. https://doi.org/10.1007/s00006-010-0258-0.
30. Franchini, S.; Gentile, A.; Vassallo, G.; Vitabile, S.; Sorbello, F. Clifford Algebra based Edge Detector for Color Images In Proceedings of the Proc. 6th Int. Conf. on Complex, Intelligent, and Software Intensive Systems (CISIS-2012), Palermo, Italia, 4-6 July 2012; pp. 84–91. https://doi.org/10.1109/CISIS.2012.128.
31. Franchini, S.; Gentile, A.; Vassallo, G.; Vitabile, S.; Sorbello, F. A Specialized Architecture for Color Image Edge Detection Based on Clifford Algebra. In Proceedings of the 7th International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS-2013), Taichung, Taiwan, 3–5 July 2013; pp. 128–135. https://doi.org/10.1109/CISIS.2013.29.
32. Franchini, S.; Gentile, A.; Sorbello, F.; Vassallo, G.; Vitabile, S. Design Space Exploration of Parallel Embedded Architectures for Native Clifford Algebra Operations. *IEEE Des. Test Comput.* **2012**, *29*, 60–69. https://doi.org/10.1109/MDT.2012.2206150.
33. Franchini, S.; Gentile, A.; Sorbello, F.; Vassallo, G.; Vitabile, S. Design and Implementation of an Embedded Coprocessor with Native Support for 5D, Quadruple-Based Clifford Algebra. *IEEE Trans. Comput.* **2013**, *62*, 2366–2381. https://doi.org/10.1109/TC.2012.225.
34. Franchini, S.; Gentile, A.; Sorbello, F.; Vassallo, G.; Vitabile, S. ConformalALU: A Conformal Geometric Algebra Coprocessor for Medical Image Processing. *IEEE Trans. Comput.* **2015**, *64*, 955–970. https://doi.org/10.1109/TC.2014.2315652.
35. Franchini, S.; Gentile, A.; Sorbello, F.; Vassallo, G.; Vitabile, S. Embedded Coprocessors for Native Execution of Geometric Algebra Operations. *Adv. Appl. Clifford Algebr.* **2017**, *27*, 559–580. https://doi.org/10.1007/s00006-016-0662-1.
36. Hildenbrand, D.; Franchini, S.; Gentile, A.; Vassallo, G.; Vitabile, S. GAPPCO: An Easy to Configure Geometric Algebra Coprocessor Based on GAPP Programs. *Adv. Appl. Clifford Algebr.* **2017**, *27*, 2115–2132. https://doi.org/10.1007/s00006-016-0755-x.
37. Hildenbrand, D.; Pitt, J.; Koch, A. *Gaalop—High Performance Parallel Computing Based on Conformal Geometric Algebra, Geometric Algebra Computing*; Springer: London, UK, 2010; pp. 477–494.
38. Hildenbrand, D. Geometric Algebra Computers. In *Foundations of Geometric Algebra Computing*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 179–188.
39. Hildenbrand, D. *Introduction to Geometric Algebra Computing*; Chapman and Hall/CRC: *Boca Raton, Florida,* 2018.
40. Perwass, C. The CLUCalc Home Page. Available online: https://www.CluCalc.info (accessed on 31 July 2019).
41. Fontijne, D. Gaigen 2: A Geometric Algebra Implementation Generator. In Proceedings of the 5th International Conference on Generative Programming and Component Engineering, GPCE '06, Portland, Oregon, USA, 22–26 October 2006; pp. 141–150.
42. Leopardi, P. The GluCat Home Page. Available online: https://glucat.sourceforge.net/(accessed on 31 July 2019).
43. Hitzer, E.; Sangwine, S. Clifford Multivector Toolbox, A Toolbox for Computing with Clifford Algebras in Matlab. 2018. Available online: https://sourceforge.net/projects/clifford-multivector-toolbox/. (accessed on 31 July 2019).
44. Mann, S.; Dorst, L.; Bouma, T. The making of GABLE: A geometric algebra learning environment in Matlab. In *Geometric Algebra with Applications in Science and Engineering*; Bayro-Corrochano, E., Sobczyk, G., Eds.; Springer: New York, NY, USA, 2001; pp. 491–511.

45. Ablamowicz, R.; Fauser, B. CLIFFORD—A Maple Package for Clifford Algebra Computations. 2017. Available online: https://math.tntech.edu/rafal/cliff2017/index.html. (accessed on 31 July 2019).
46. Ashdown, M. GA package for Maple. 2018. Available online: https://www.mrao.cam.ac.uk/~maja1/software/GA/. (accessed on 31 July 2019).
47. Browne, J. The Grassmann Algebra Book Home Page. Available online: https://grassmannalgebra.com. (accessed on 31 July 2019).
48. Mishra, B.; Wilson, P.; Wilcock, R. A geometric algebra coprocessor for color edge detection. *Electronics* **2015**, *4*, 94–117.
49. Perwass, C.; Gebken, C.; Sommer, G. 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*; Ablamowicz, R., Ed.; Springer: New York, NY, USA, 2004; Volume 34.
50. Martinez-Terán, G.; Ureña-Ponce, O.; Soria-García, G.; Ortega-Cisneros, S.; Bayro-Corrochano, E. Fast Study Quadric Interpolation in the Conformal Geometric Algebra Framework. *Electronics* **2022**, *11*, 1527. https://doi.org/10.3390/electronics11101527.
51. Teran, G.M.; Urena-Ponce, O.; Garcia, G.S.; Ortega-Cisneros, S.; Corrochano, E.B. Fast GPU Interpolation for Medical Robotics Using the Conformal Geometric Algebra Framework. *Biomed. J. Sci. Tech. Res.* **2021**, *38*, 30661–30670. https://doi.org/10.26717/BJSTR.2021.38.006201.