National Research Council of Italy

ICAR CNR
Istituto di Calcolo e Reti ad Alte Prestazioni

TECHNISCHE UNIVERSITÄT DARMSTADT

UTS
UNIVERSITY OF TECHNOLOGY SYDNEY

BIOMEDICINA NEUROSCIENZE DIAGNOSTICA AVANZATA

UNIVERSITÀ DEGLI STUDI DI PALERMO

A FPGA Coprocessor for Accelerating Geometric Algebra Algorithms based on the GAPPCO Design

S. Franchini, D. Hildenbrand, E. Saribatir, and S. Vitabile

AGACSE 2024 – 9th Conference on Applied Geometric Algebras in Computer Science and Engineering
University of Amsterdam, Amsterdam, The Netherlands
August 27-29, 2024

Motivation

- High computational complexity of Geometric Algebra (GA) operations
- Efficient implementations of GA algorithms are required to address their high computational costs and improve performance in real-time applications
- Dedicated tools and devices to allow for the practical use of GA in real-world applications

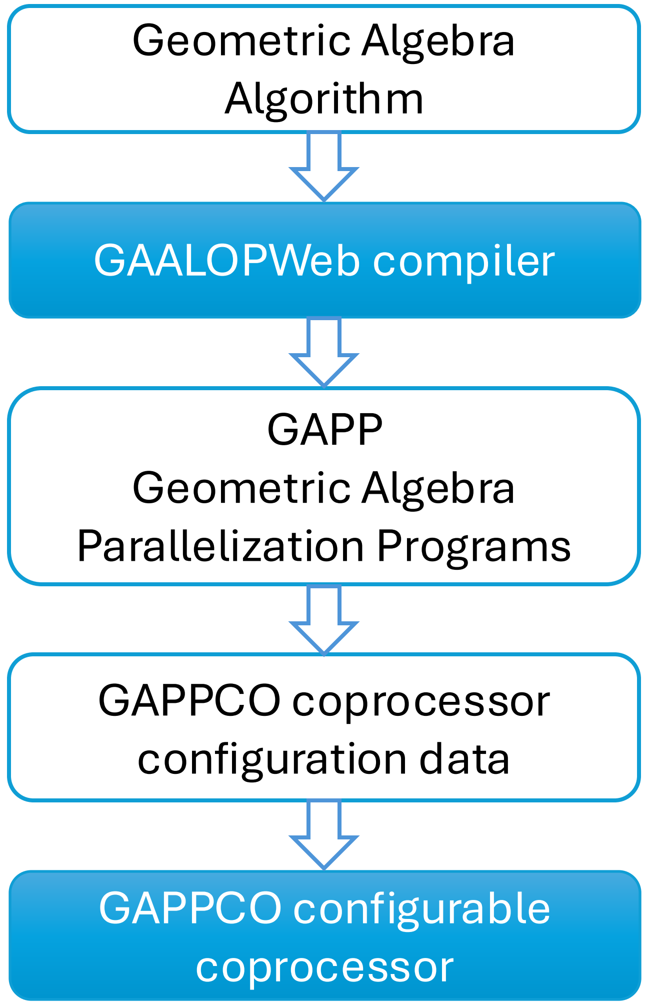
Summary

- State-of-the-art Geometric Algebra implementations
- Proposed approach based on GAALOPWeb compiler + GAPPCO configurable coprocessor
  - GAALOPWeb compiler
  - GAPPCO coprocessor design
  - GAPPCO FPGA implementation
- Experimental results
  - 5x average speedup against standard CPU
- Conclusions and future work

Existing GA Implementations

- Software implementations
  - **Gaigen 2** C++ software library generator by D. Fontijne, L. Dorst
  - **GMac** code generator by Ahmad Hosney Awad Eid
  - **Versor** C++ library by P. Colapinto
  - **Bivector.net** webpage (code generators for C++, C#, Python, Rust, etc.)
- Full-hardware implementations
  - **FPGA coprocessors**
    - Perwass et al. (2003)
    - University of Palermo research group: Gentile et al. (2005), Franchini et al. (2007, 2009, 2013, 2015, 2022)
  - **ASIC implementations**
    - Mishra and Wilson (2006)
- Mixed software-hardware implementations
  - **GAALOPWeb** (by D. Hildenbrand) generates optimized code for different programming languages (C/C++, Python, Rust, Julia, Matlab, Mathematica) and parallel frameworks as OpenCL, CUDA, and FPGA

Proposed approach


Geometric Algebra Algorithm
GAALOPWeb compiler
GAPP Geometric Algebra Parallelization Programs
GAPPCO coprocessor configuration data
GAPPCO configurable coprocessor

- Mixed software/hardware system based on
  - GAALOPWeb compiler
  - GAPPCO configurable coprocessor
- GAALOP (***G***eometric ***A***lgebra ***Al***gorithms ***Op***timizer) produces parallel computations of multivector coefficients each consisting of sums of products to be again parallelized
- GAALOP generates configuration data for GAPPCO configurable coprocessor

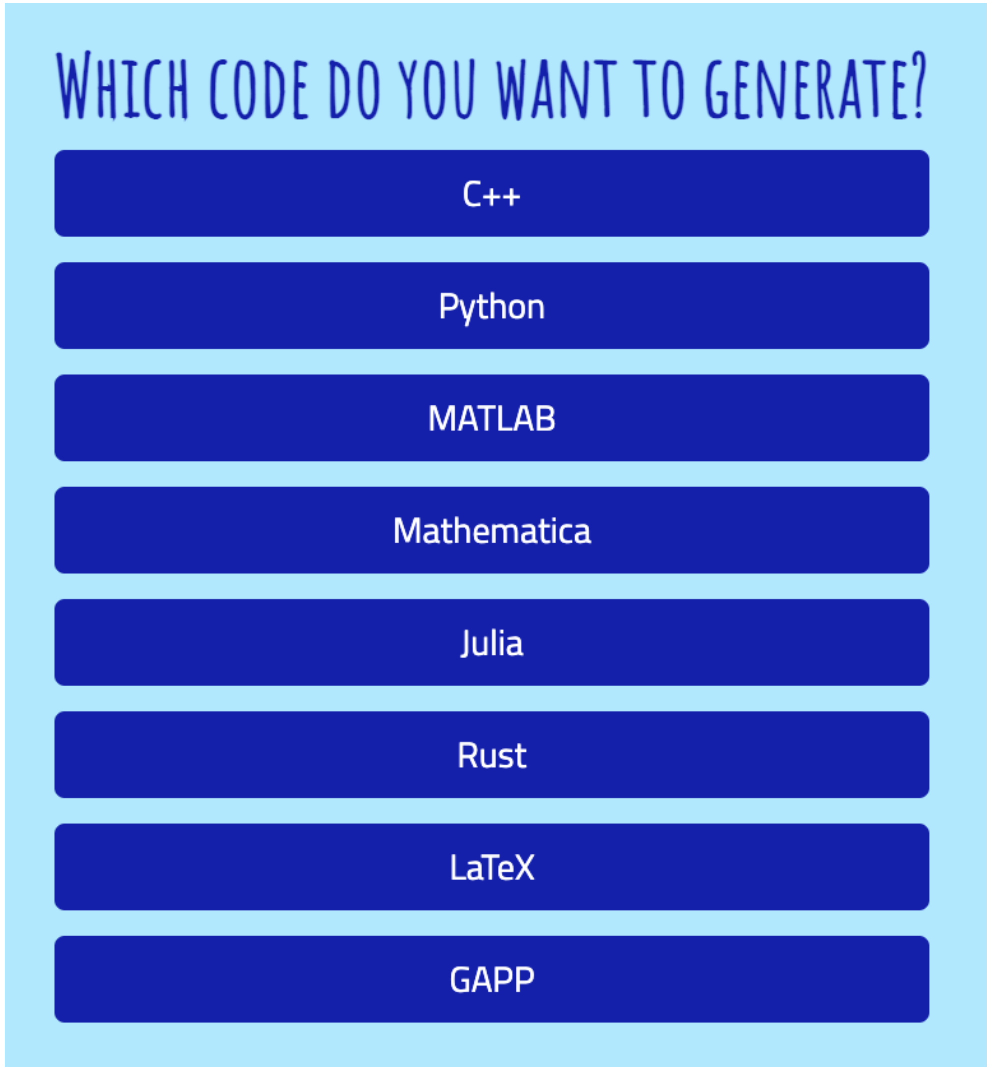
GAALOP - Geometric Algebra ALgorithms OPtimizer

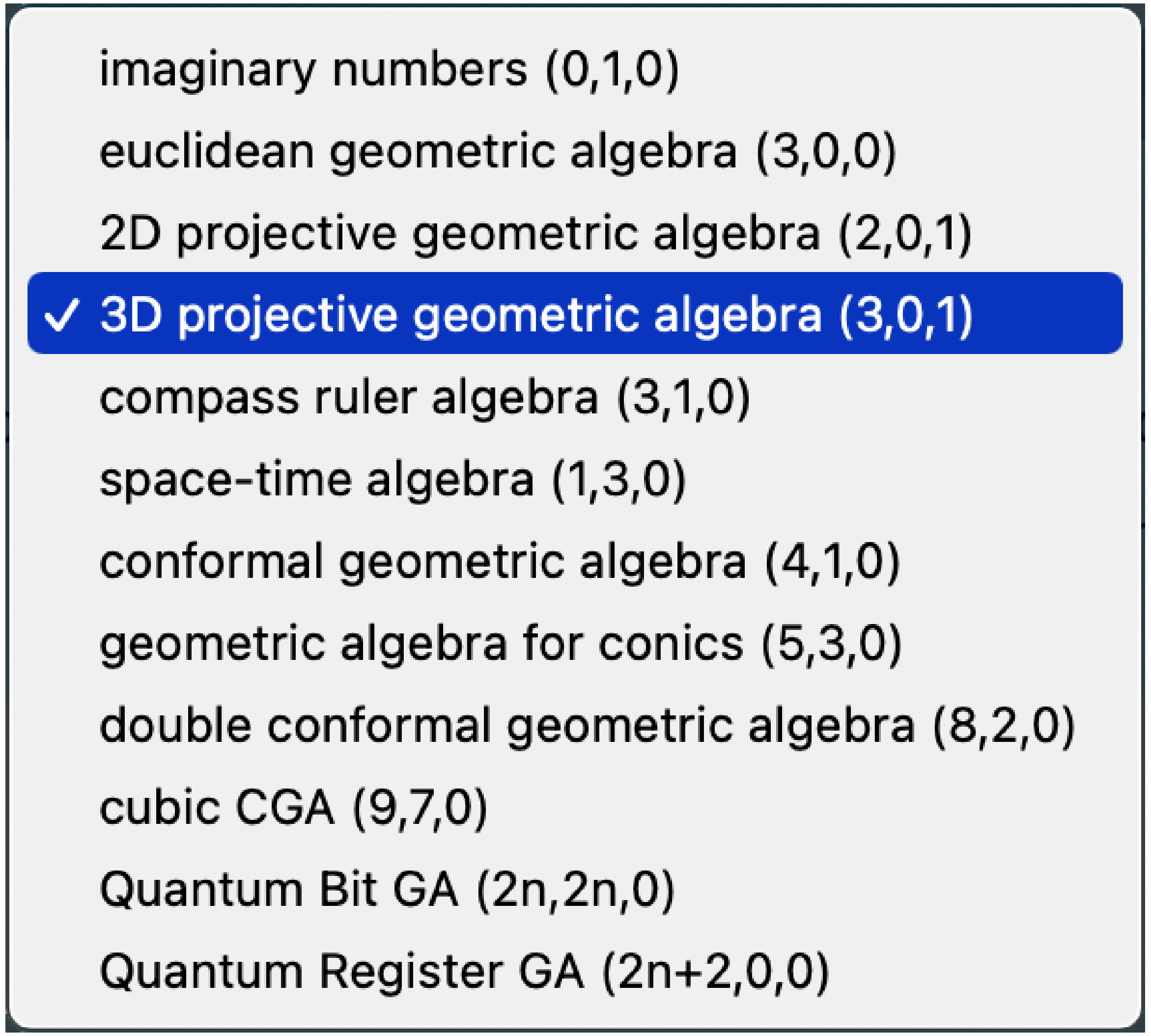
- Is a compiler that can parse GA algorithms described using GAALOPScript, and generate optimised code to target hardware that can perform dot product operations.

- Any hardware device that can perform a dot product operation on vector elements can be used to execute Geometric Algebra algorithms.

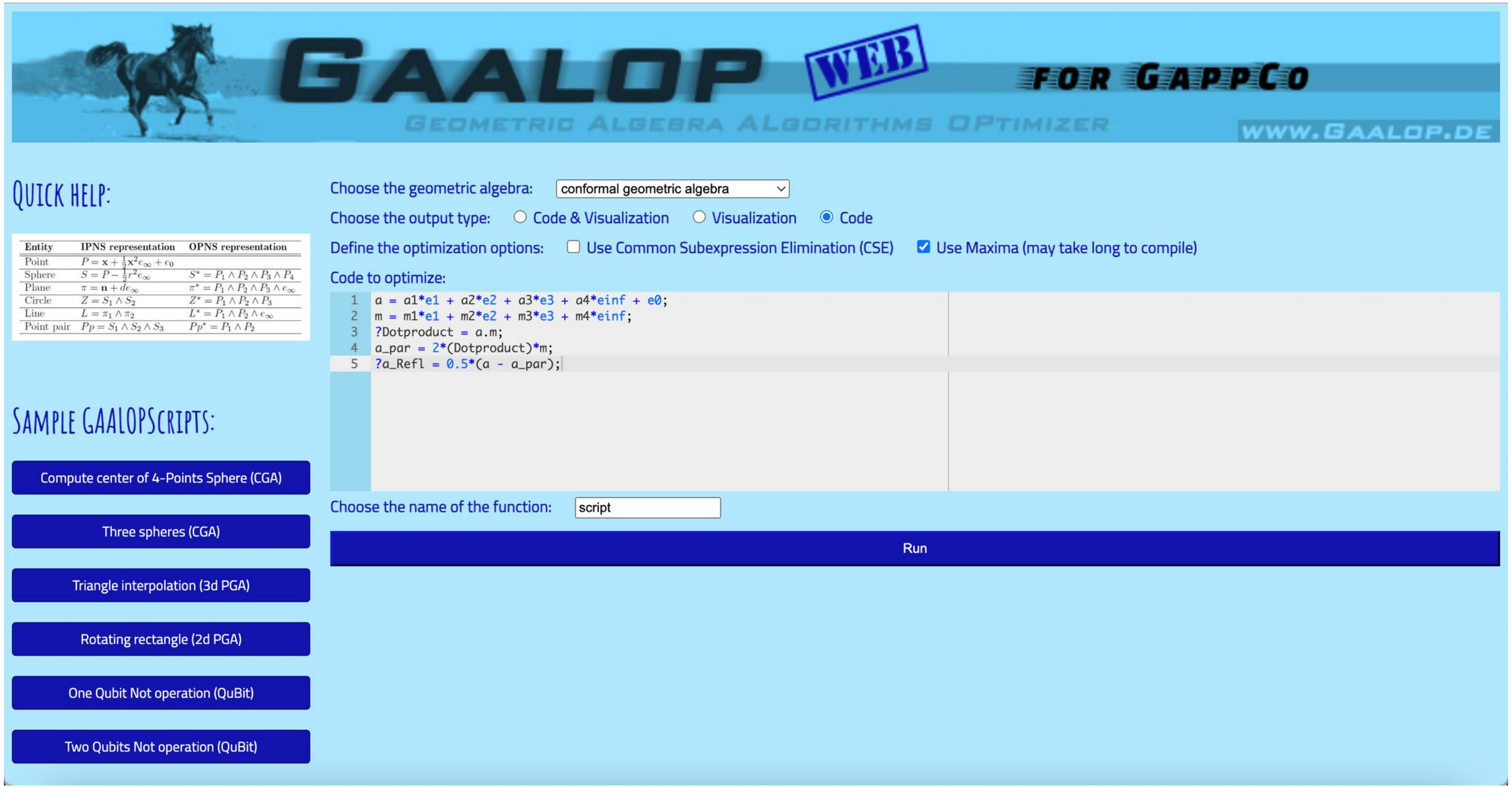
- GAALOP generates optimised code for C/C++, Java, Python, Julia, Mathematica, MATLAB, Rust, LaTeX, OpenCL, CUDA, FPGA, GAPP, RISC-V Vector Extensions, SIMD.

GAALOPWeb – Web Interface for GAALOP




imaginary numbers (0,1,0)
euclidean geometric algebra (3,0,0)
2D projective geometric algebra (2,0,1)
✓ 3D projective geometric algebra (3,0,1)
compass ruler algebra (3,1,0)
space-time algebra (1,3,0)
conformal geometric algebra (4,1,0)
geometric algebra for conics (5,3,0)
double conformal geometric algebra (8,2,0)
cubic CGA (9,7,0)
Quantum Bit GA (2n,2n,0)
Quantum Register GA (2n+2,0,0)

GAALOPWeb – Web Interface for GAALOP



QUICK HELP:

Entity	IPNS representation	OPNS representation
Point	$P = \mathbf{x} + \frac{1}{2}\mathbf{x}^2 e_\infty + e_0$	
Sphere	$S = P - \frac{1}{2}r^2 e_\infty$	$S^* = P_1 \wedge P_2 \wedge P_3 \wedge P_4$
Plane	$\pi = \mathbf{n} + d e_\infty$	$\pi^* = P_1 \wedge P_2 \wedge P_3 \wedge e_\infty$
Circle	$Z = S_1 \wedge S_2$	$Z^* = P_1 \wedge P_2 \wedge P_3$
Line	$L = \pi_1 \wedge \pi_2$	$L^* = P_1 \wedge P_2 \wedge e_\infty$
Point pair	$Pp = S_1 \wedge S_2 \wedge S_3$	$Pp^* = P_1 \wedge P_2$

SAMPLE GAALOPSCRIPTS:

Compute center of 4-Points Sphere (CGA)
Three spheres (CGA)
Triangle interpolation (3d PGA)
Rotating rectangle (2d PGA)
One Qubit Not operation (QuBit)
Two Qubits Not operation (QuBit)

Choose the geometric algebra: conformal geometric algebra

Choose the output type: ○ Code & Visualization ○ Visualization [✓ Code]

Define the optimization options: ☐ Use Common Subexpression Elimination (CSE) ☑ Use Maxima (may take long to compile)

Code to optimize:

```
a = a1*e1 + a2*e2 + a3*e3 + a4*einf + e0;
m = m1*e1 + m2*e2 + m3*e3 + m4*einf;
?Dotproduct = a.m;
a_par = 2*(Dotproduct)*m;
?a_Refl = 0.5*(a - a_par);
```

Choose the name of the function: script

GAPPCO coprocessor concept

- GAPPCO proof-of-concept presented in

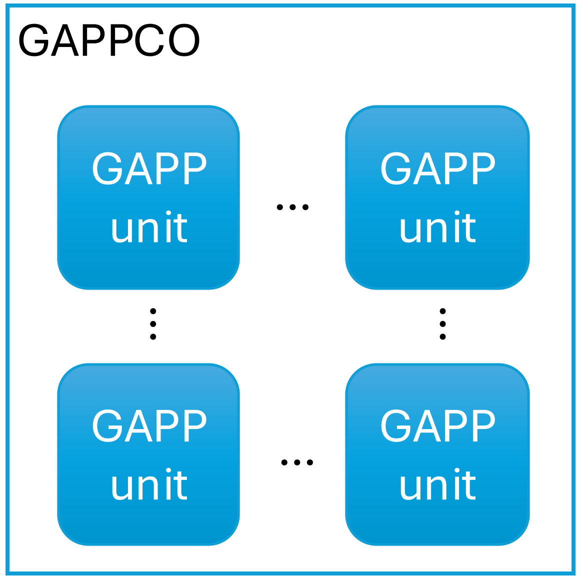
  D. Hildenbrand, S. Franchini, A. Gentile, G. Vassallo, S. Vitabile, “*GAPPCO: An Easy to Configure Geometric Algebra Coprocessor Based on GAPP Programs*”, ADVANCES IN APPLIED CLIFFORD ALGEBRAS, 27(3), pp. 2115-2132, 2017

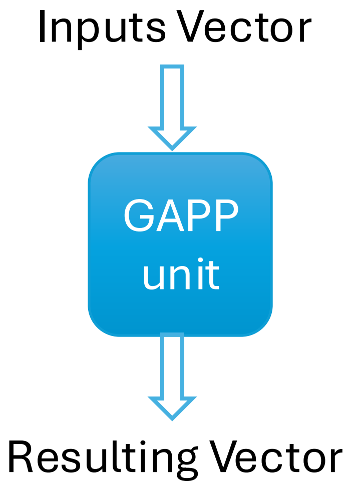
- Can be programmed according to GAALOPWeb configuration data

- Reconfigurable to support different GA algorithms

- Consists of one or more parallel GAPP units

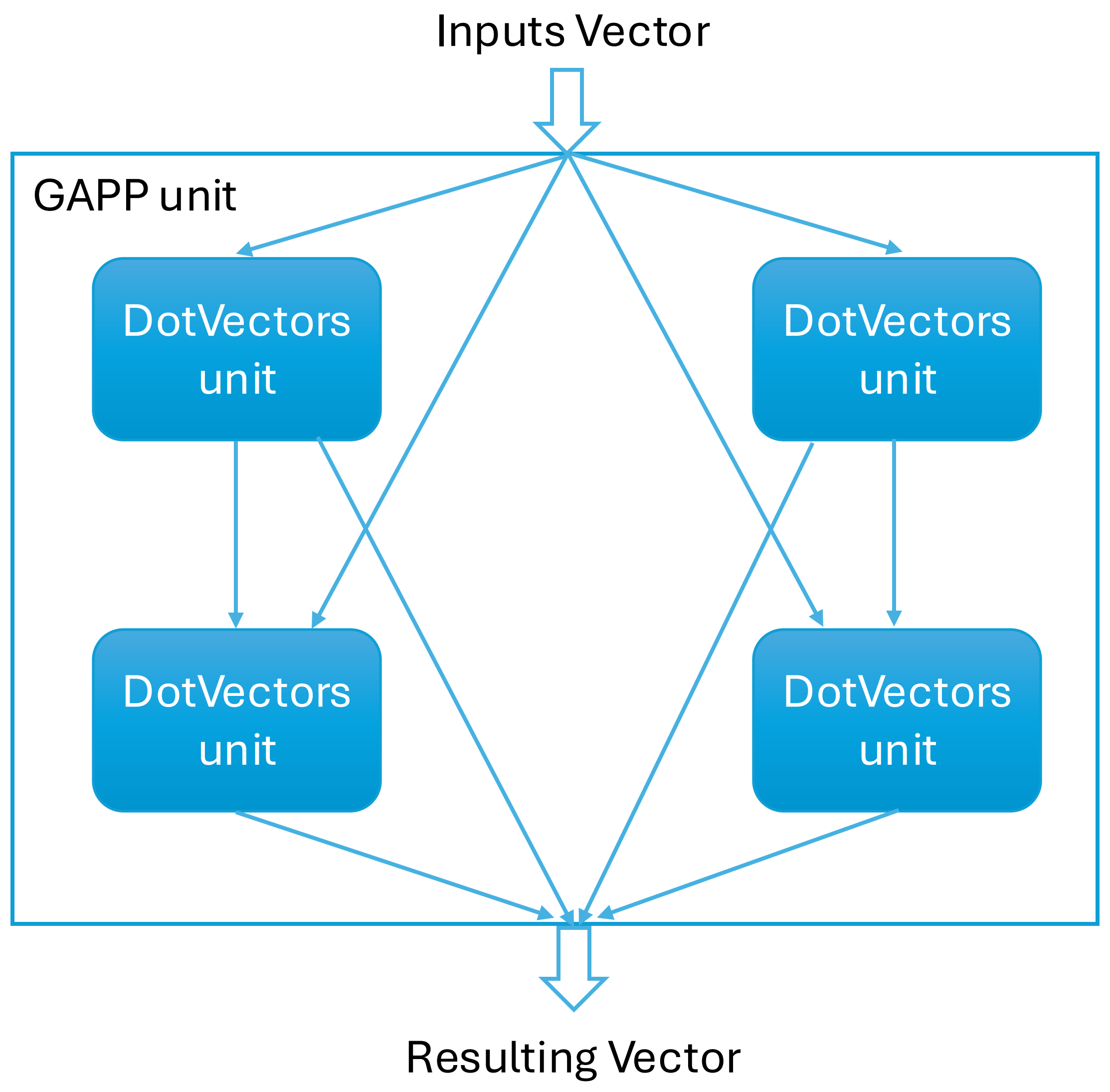
- Each GAPP unit realizes a GAPP program (i.e. a GA algorithm precompiled by GAALOPWeb)



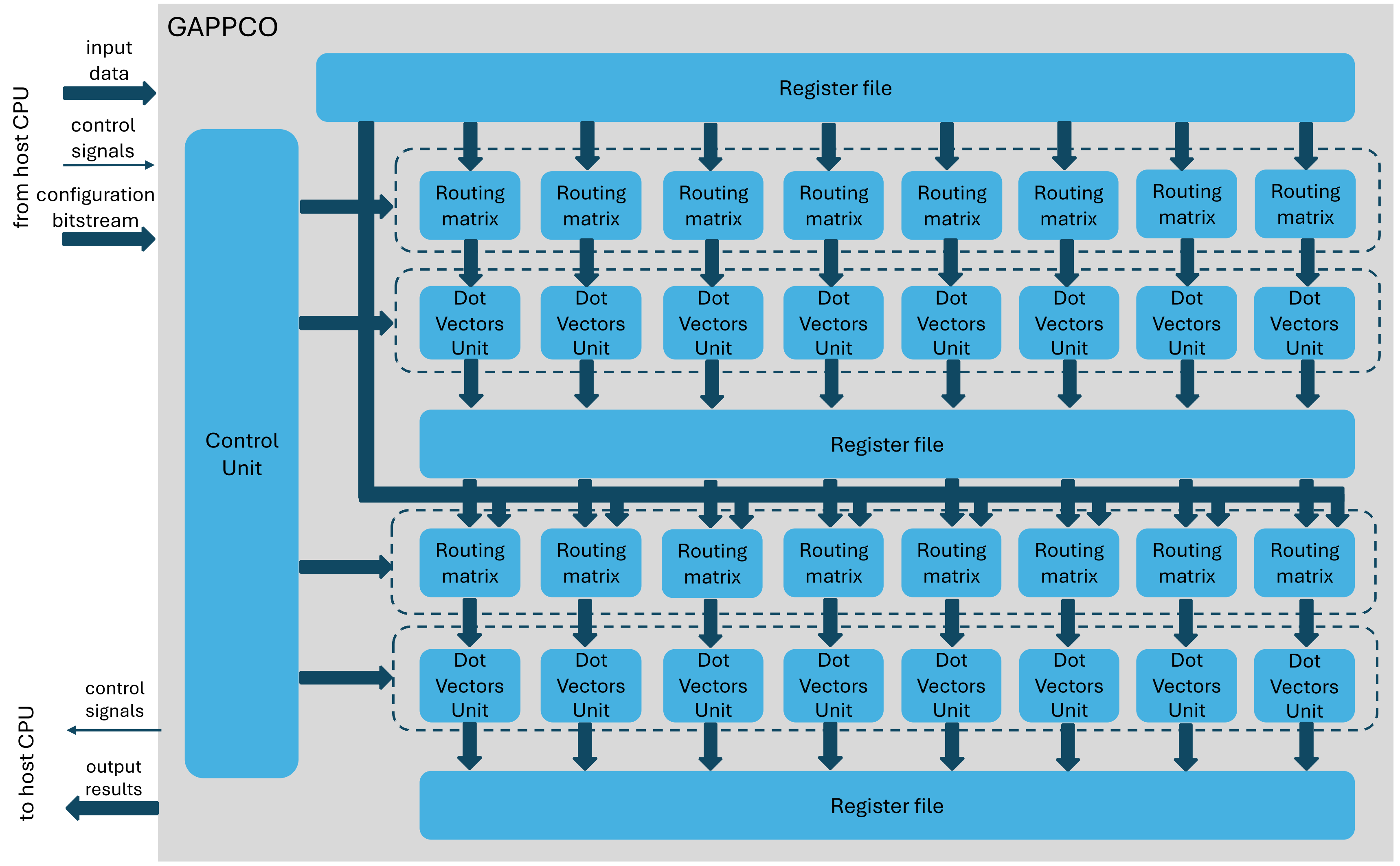


GAPPCO coprocessor concept

- A GAPP unit is a network of one or more DotVectors units
- DotVectors units calculate multivector coefficients by executing dot product operations (sums of products)
- Connections are configured based on the GAALOPWeb configuration data



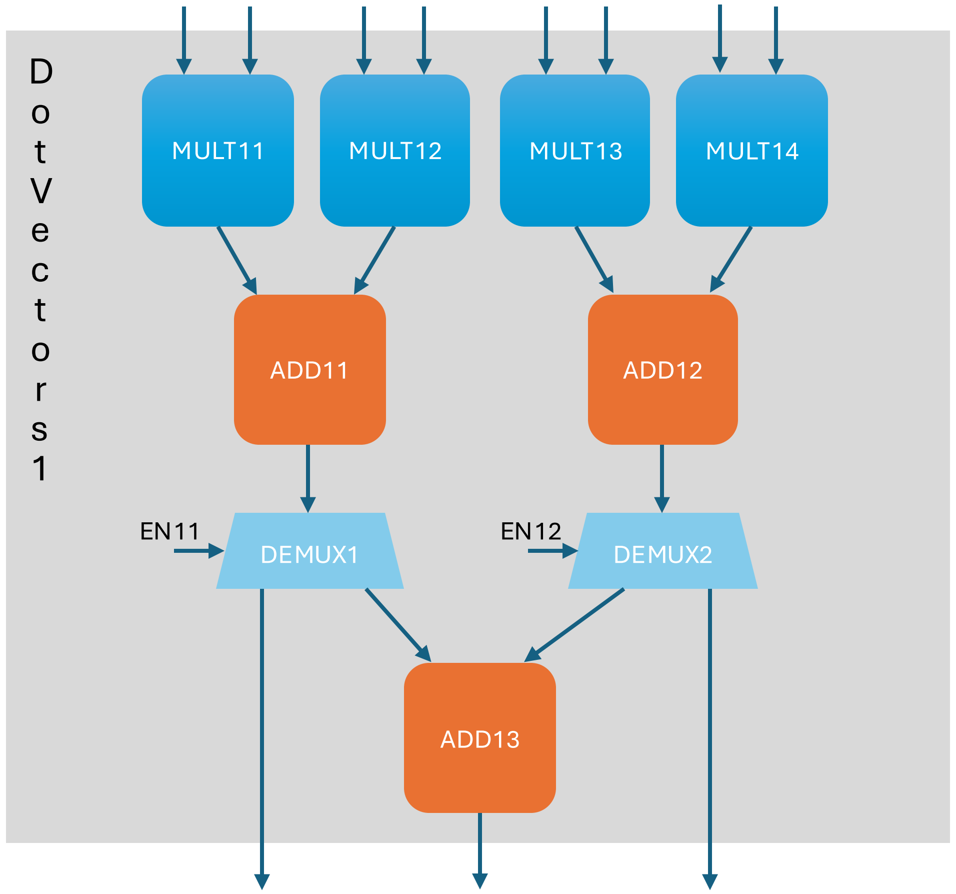
GAPPCO coprocessor design



GAPPCO functional units

- DotVectors units
  - Programmable processing units
- Register files
  - Input data, intermediate and output results
- Routing matrices
  - Programmable interconnection infrastructure
- Control unit
  - Supervises GAPPCO operations and host CPU/GAPPCO data exchange

DotVectors Unit



- Executes basic arithmetic operations (sums of products) on 32-bit floating point numbers
- Can be configured to execute
  - 2 sums of 2 products or
  - 1 sum of 4 products
- DotVectors units can be connected to form larger processing units (GAPP units) able to support complex GA-based computations
- Different GAPP units working in parallel can be configured
- DotVectors units as well as GAPP units have a ***pipeline*** structure
- GAPPCO architecture is based on a set of pipelines working in parallel (***multi-core*** processing)

Application suite

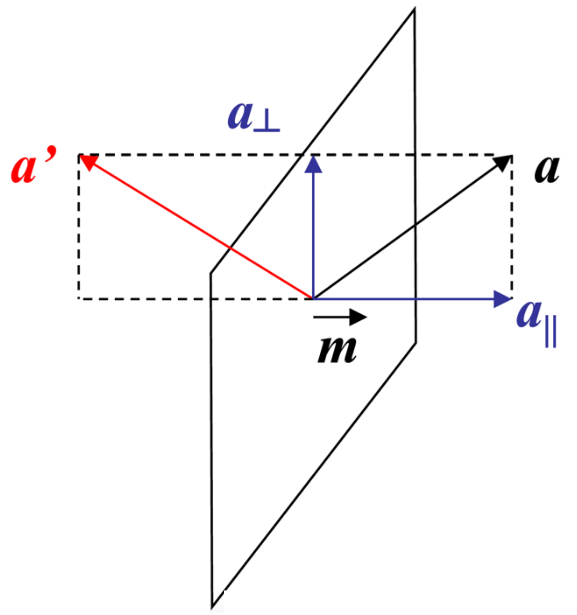
GA-based Application	Geometric computations	Basic GA operations
Medical image registration	Rotations, translations, dilations	5D vector reflections
Raytracing	Ray-bounding sphere and ray-mesh intersections	Line-sphere and line-plane intersections
Line/circle detection in images	Circle passing through points	Circle from 3 points

- Application profiling
- Basic GA computations (GAALOP script)
- GAALOPWeb compiling
- Configuration bitstream for GAPPCO

5D vector reflection

CGA formulation used in the ConformalALU coprocessor

S. Franchini, A. Gentile, F. Sorbello, G. Vassallo, S. Vitabile, "ConformalALU: a Conformal Geometric Algebra Coprocessor for Medical Image Processing", IEEE TRANSACTIONS ON COMPUTERS, 64(4), pp. 955-970, 2015.



Orthogonal transformations (rotations, translations, dilations) can be split into multiple non-commuting consecutive reflections.

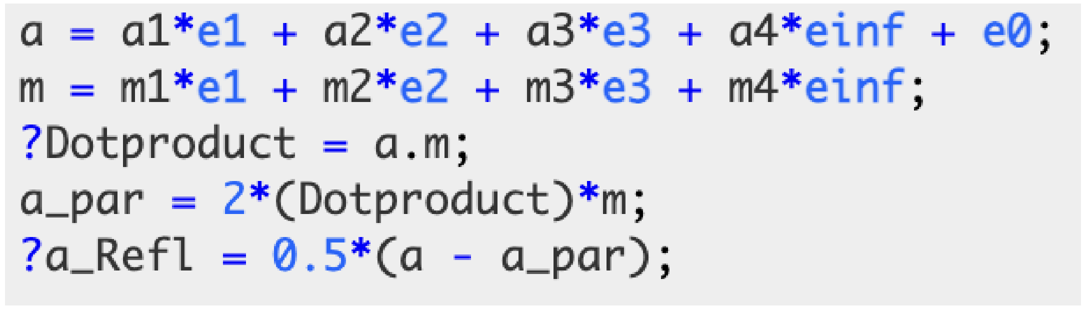
$A_k = a_1 \wedge a_2 \wedge \cdots \wedge a_k$ generic *k*-blade

$$A'_k = -mA_k m = -m(a_1 \wedge a_2 \wedge \cdots \wedge a_k)m = \\ = (-ma_1m) \wedge (-ma_2m) \wedge \cdots \wedge (-ma_km)$$ reflected *k*-blade

$$a' = a_\perp - a_\parallel = a_\perp + a_\parallel - a_\parallel - a_\parallel = a - 2a_\parallel$$

$$\underline{a'} = a - 2|a_\parallel|m = \underline{a - 2(a \cdot m)m}$$

From GAALOP script ...



```
a = a1*e1 + a2*e2 + a3*e3 + a4*einf + e0;
m = m1*e1 + m2*e2 + m3*e3 + m4*einf;
?Dotproduct = a.m;
a_par = 2*(Dotproduct)*m;
?a_Refl = 0.5*(a - a_par);
```

a is either a point or a sphere to be reflected at m

m is a plane with normal vector $(m1, m2, m3)$ and $m4$ as the distance to the origin

The multiplication of the result by 0.5 is done because the resulting computations become easier (and in CGA the multiplication by a scalar does not change the geometric object)

To GAPP code ...

Parallel computations of multivector coefficients (dot products)

$$Dotproduct = a1m1 + a2m2 + a3m3 - m4 \longrightarrow$$

```
assignInputsVector inputsVector = [a1,a2,a3,a4,m1,m2,m3,m4];

//Dotproduct[0] = (((-(inputsVector[7])) + (inputsVector[2] * inputsVector[6])) +
//(inputsVector[1] * inputsVector[5])) + (inputsVector[0] * inputsVector[4])
resetMv Dotproduct[32];
setVector ve0 = {inputsVector[-7,2,1,0]};
setVector ve1 = {1.0, inputsVector[6,5,4]};
dotVectors Dotproduct[0] = <ve0,ve1>;
```

$$a\_Refl[1] = 0.5a1 - (Dotproduct)m1 \longrightarrow$$

```
//a_Refl[1] = (0.5 * inputsVector[0]) - (Dotproduct[0] * inputsVector[4])
resetMv a_Refl[32];
setVector ve2 = {0.5, Dotproduct[-0]};
setVector ve3 = {inputsVector[0,4]};
dotVectors a_Refl[1] = <ve2,ve3>;
```

$$a\_Refl[2] = 0.5a2 - (Dotproduct)m2 \longrightarrow$$

```
//a_Refl[2] = (0.5 * inputsVector[1]) - (Dotproduct[0] * inputsVector[5])
setVector ve4 = {0.5, Dotproduct[-0]};
setVector ve5 = {inputsVector[1,5]};
dotVectors a_Refl[2] = <ve4,ve5>;
```

$$a\_Refl[3] = 0.5a3 - (Dotproduct)m3 \longrightarrow$$

```
//a_Refl[3] = (0.5 * inputsVector[2]) - (Dotproduct[0] * inputsVector[6])
setVector ve6 = {0.5, Dotproduct[-0]};
setVector ve7 = {inputsVector[2,6]};
dotVectors a_Refl[3] = <ve6,ve7>;
```

$$a\_Refl[4] = 0.5a4 - (Dotproduct)m4 \longrightarrow$$

```
//a_Refl[4] = (0.5 * inputsVector[3]) - (Dotproduct[0] * inputsVector[7])
setVector ve8 = {0.5, Dotproduct[-0]};
setVector ve9 = {inputsVector[3,7]};
dotVectors a_Refl[4] = <ve8,ve9>;

//a_Refl[5] = 0.5
assignMv a_Refl[5] = [0.5];
```

To GAPPCO configuration bitstream

GAPPCO register files

0: a1@0
1: a2@0
2: a3@0
3: a4@0
4: m1@0
5: m2@0
6: m3@0
7: m4@0
8: 1.0@0
9: Dotproduct[0]@1
10: 0.5@0
11: a_Refl[1]@2
12: a_Refl[2]@2
13: a_Refl[3]@2
14: a_Refl[4]@2

GAPPCO DotVectors Units

3 DotVectors Units

DotVectors1:
EN: 0
FACTOR1: -7 2 1 0
FACTOR2: 8 6 5 4
RESULT: 9

DotVectors2:
EN: 1
FACTOR1: 10 -9 10 -9
FACTOR2: 0 4 1 5
RESULT: 11 12

DotVectors3:
EN: 1
FACTOR1: 10 -9 10 -9
FACTOR2: 2 6 3 7
RESULT: 13 14

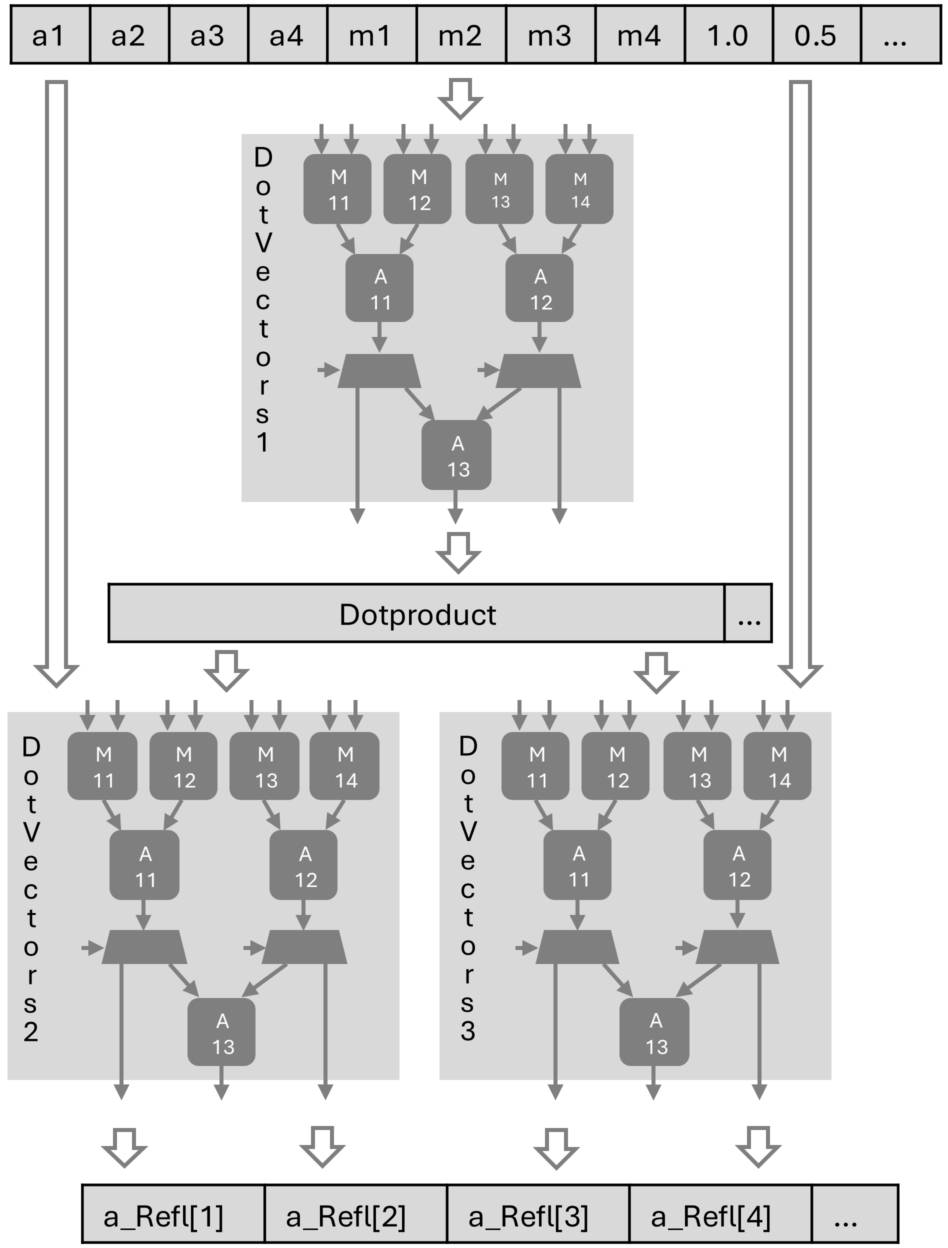
GAPPCO configuration data

```
1 // No. of GAPP units
3 // No. of 4-width DotVectors units for GAPP unit 1

// Configuration bits for DotVectors 1
0/0 // EN11 / EN12
7/1 // MULT11_addr / MULT11_sign
8/0 // MULT11_addr / MULT11_sign
2/0 // MULT12_addr / MULT12_sign
6/0 // MULT12_addr / MULT12_sign
1/0 // MULT13_addr / MULT13_sign
5/0 // MULT13_addr / MULT13_sign
0/0 // MULT14_addr / MULT14_sign
4/0 // MULT14_addr / MULT14_sign
9/1 // RESULT11_addr / RESULT11_type

// Configuration bits for DotVectors 2
1/1 // EN21 / EN22
10/0 // MULT21_addr / MULT21_sign
0/0 // MULT21_addr / MULT21_sign
9/1 // MULT22_addr / MULT22_sign
4/0 // MULT22_addr / MULT22_sign
10/0 // MULT23_addr / MULT23_sign
1/0 // MULT23_addr / MULT23_sign
9/1 // MULT24_addr / MULT24_sign
5/0 // MULT24_addr / MULT24_sign
11/0 // RESULT21_addr / RESULT21_type
12/0 // RESULT22_addr / RESULT22_type

// Configuration bits for DotVectors 3
1/1 // EN31 / EN32
10/0 // MULT31_addr / MULT31_sign
2/0 // MULT31_addr / MULT31_sign
9/1 // MULT32_addr / MULT32_sign
6/0 // MULT32_addr / MULT32_sign
10/0 // MULT33_addr / MULT33_sign
3/0 // MULT33_addr / MULT33_sign
9/1 // MULT34_addr / MULT34_sign
7/0 // MULT34_addr / MULT34_sign
13/0 // RESULT31_addr / RESULT31_type
14/0 // RESULT32_addr / RESULT32_type
```

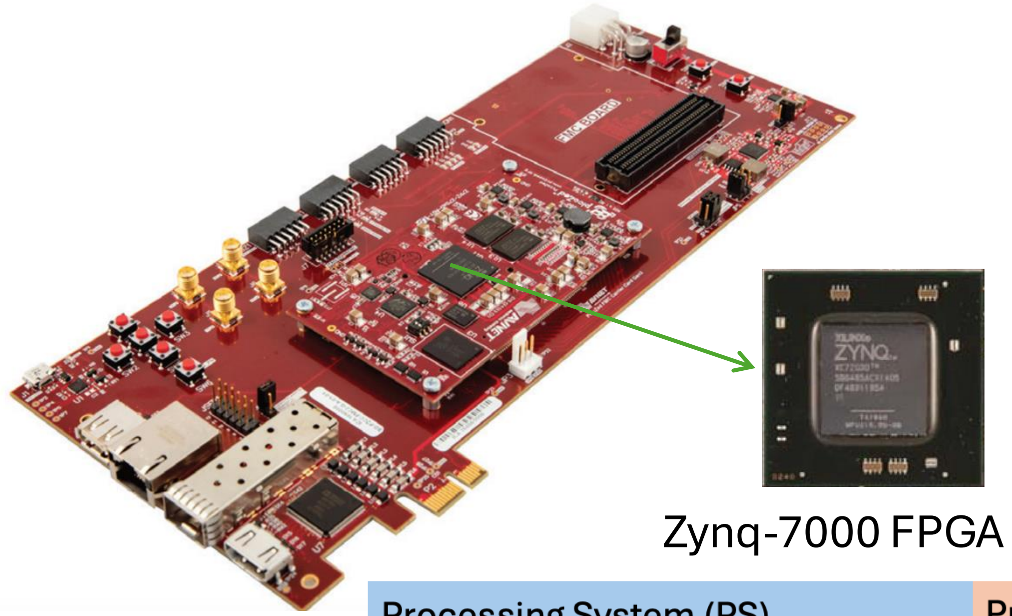
GAPPCO I Configuration data as hexstream (Padded with zeroes at the end)
1c7c08c128089cfa0268a0a6ab63a126ca1a6ed7000000

Configured GAPPCO

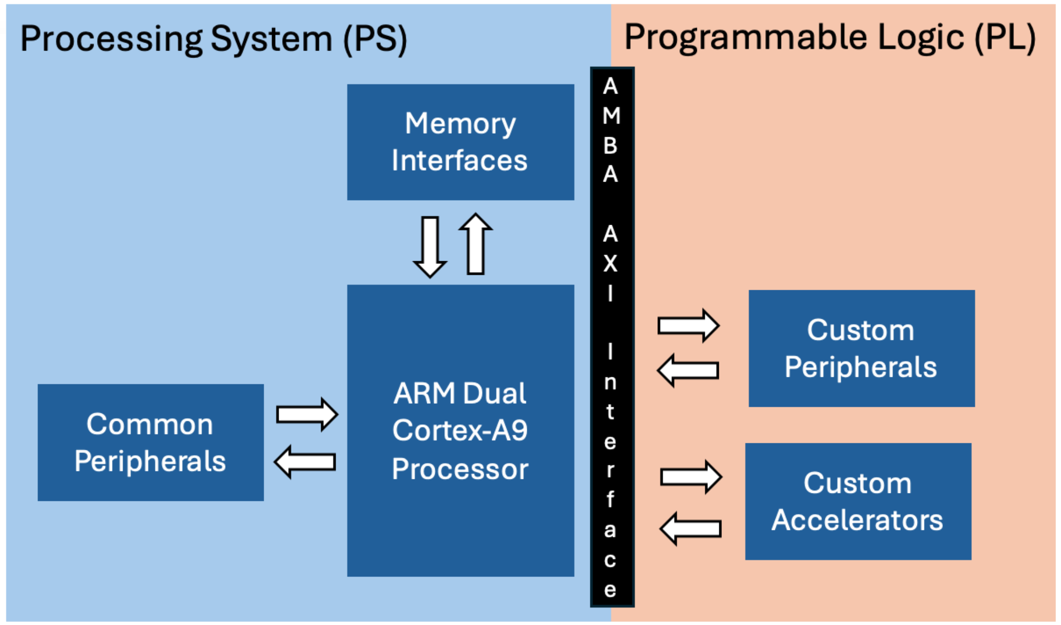
- GAPP unit composed of 3 ***pipelined*** DotVectors units for 5D vector reflection
- The different processing stages work concurrently on different data
- The outputs of each processing stage are the inputs of the successive processing stage



GAPPCO FPGA prototype

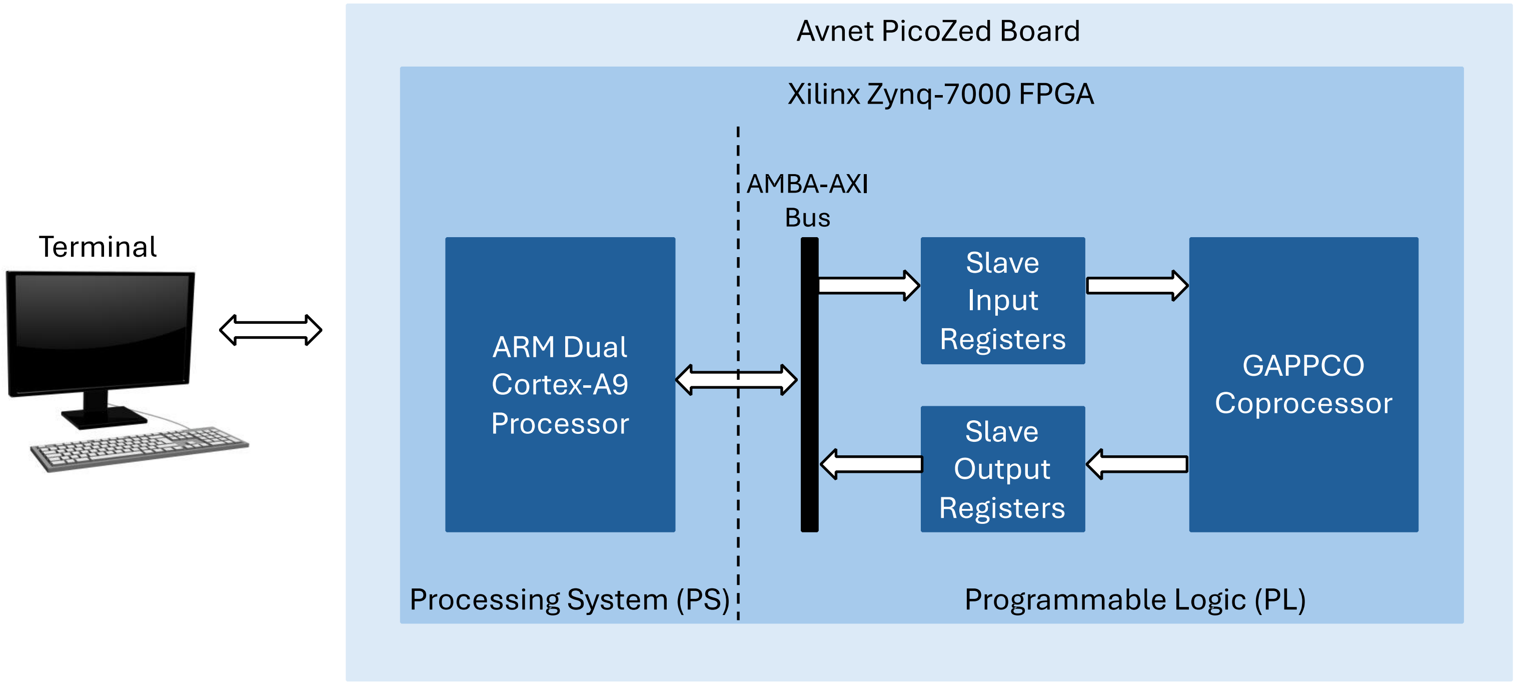


Zynq-7000 FPGA chip



- Avnet Picozed FPGA development board
- AMD Xilinx Zynq-7000 Field Programmable Gate Array (FPGA) device
- System-on-Chip (SoC) containing ARM Processor & Programmable Logic
- Programmable Logic
  - Configurable Logic Blocks
    - Look-Up-Tables (LUT)
    - Flip-Flops
  - Block RAM
  - Digital Signal Processing (DSP) Blocks (Multipliers, Adders/Accumulators)

GAPPCO FPGA prototype



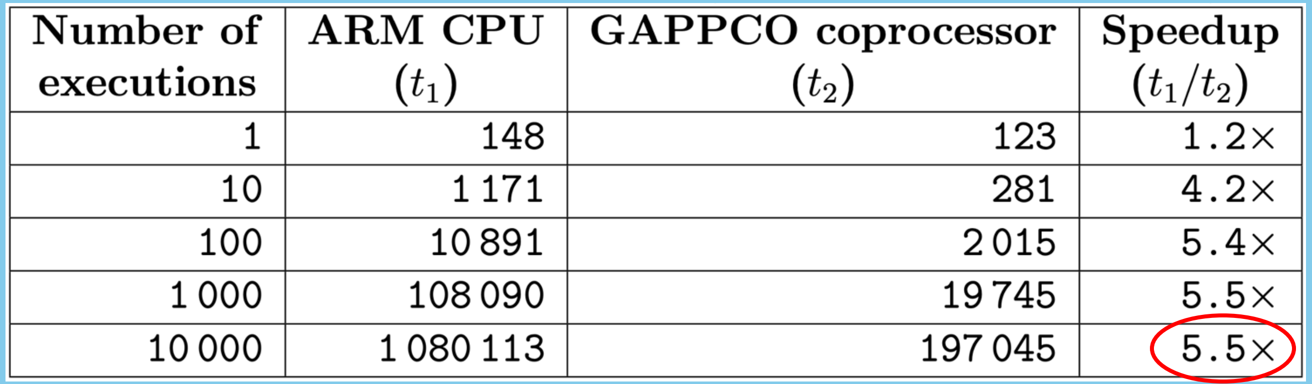
GAPPCO prototype resource utilization (16 DotVectors units)

Resource	Used	Available	Utilization
LUT	38 496	78 600	48.98%
FF	29 818	157 200	18.97%
DSP	288	400	72.00%

- GAPPCO prototype as an embedded SoC composed of
  - ARM standard processor
  - GAPPCO specialized coprocessor
- GAPPCO has been integrated in the PL of the FPGA chip and connected as a custom peripheral (IP-core) to the ARM high-bandwidth bus
- Before runtime, the configuration bitstream generated by the GAALOPWeb compiler is downloaded from the ARM processor to the peripheral in order to customize GAPPCO for the required GA algorithm

Experimental results

Average execution times (in clock cycles at 333 MHz)



Number of executions	**ARM CPU** (t_1)	**GAPPCO coprocessor** (t_2)	**Speedup** (t_1/t_2)
1	148	123	1.2×
10	1 171	281	4.2×
100	10 891	2 015	5.4×
1 000	108 090	19 745	5.5×
10 000	1 080 113	197 045	5.5×

- Computation of a 5D vector reflection in CGA as test algorithm
- Performance comparison between ARM processor and GAPPCO coprocessor
- In the GAPPCO execution, most of the time is taken by ARM-GAPPCO data transfers

Future work

- Upgrading the basic processing unit to support other basic operations (square root, division, etc.)
- Increasing the number of parallel processing units in order to increase parallelism degree and achieve better speedups
- Using larger FPGA devices with faster data transfer interfaces
- Extending the application suite used to validate the GAALOPWeb+GAPPCO system effectiveness

Thank you for your attention!