Gaalop Precompiler Manual

Patrick Charrier (patrick.charrier@stud.tu-darmstadt.de)

January 13, 2014

1 Introduction and The Horizon example

In the following, the so called "Horizon example" is used to explain the setup and basic workflow of Gaalop Precompiler [1]. This guide is mainly intended as a technical manual. For a more mathematical description of Geometric Algebra we refer to [2].

However, it is not necessary to understand the example, in order to learn how to work with Gaalop GPC in the succeeding sections. For a quick start, jump to:

- section 2 to learn using Gaalop Precompiler for C/C++
- or section 3 for Gaalop Precompiler targeting OpenCL.

1.1 Mathematical background

Consider an observer standing on a planet. Given a description of the particular planet and the viewpoint of the observer, we try to find an algebraic expression of the horizon as seen by the observer, provided there is no occlusion of any sort other than the planet itself, in the scene.

We define P as the viewpoint of the observer, S as a sphere describing the planet with center point M and radius r. Let m_x, m_y, m_z be the 3D coordinates of the planet's center and p_x, p_y, p_z be the ones of the viewpoint, then M, P and S have the following definition in 5D conformal space.

$$M = m_x e_1 + m_y e_2 + m_z e_3 + \frac{1}{2}(m_x^2 + m_y^2 + m_z^2)e_\infty + e_0$$

$$P = p_x e_1 + p_y e_2 + p_z e_3 + \frac{1}{2}(p_x^2 + p_y^2 + p_z^2)e_\infty + e_0$$

$$S = M - \frac{1}{2}r^2 e_\infty$$

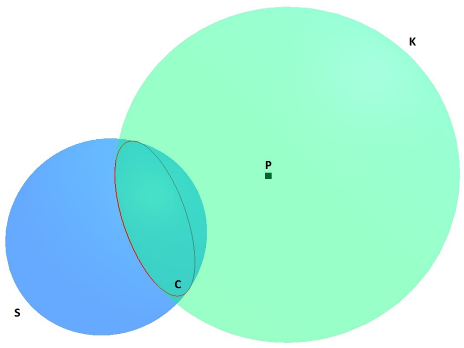


Figure 1: Calculation of the intersection circle (horizon)

Given these definitions, we may construct another sphere K around P. The radius for this second sphere is computed by the inner product $S \cdot P$. The circle presenting the horizon may then be calculated by the outer product of both spheres. Figure 1 illustrates the calculation.

$$K = P + (S \cdot P)e_\infty$$

$$C = S \wedge K$$

2 How To Use **Gaalop GPC for C++**

This manual shows the particular steps that need to be performed, to create, setup and build an application using Gaalop GPC for C++.

1. Create the file *horizon.cpg* with the contents below. For a mathematical description of the code, you may refer to section 1.

> **Beware:** Do **NOT copy-and-paste** this code. The PDF format uses strange character encodings that have lead to a huge variety of errors in the past. A much better way is to copy the source code package from `http://www.gaalop.de/wp-content/uploads/gpc_tutorials.zip`.

```
#include <iostream>

int main() {
  #pragma gpc begin
    #pragma clucalc begin
      P = VecN3(1,1,0);
      r = 1;
      S = e0-0.5*r*r*einf;
      C = S^(P+(P.S)*einf);
      ?homogeneousCenter = C*einf*C;
      ?scale = -homogeneousCenter.einf;
      ?EuclideanCenter = homogeneousCenter / scale;
    #pragma clucalc end
    std::cout << mv_get_bladecoeff(homogeneousCenter,e0)
              << std::endl;
    std::cout << mv_get_bladecoeff(EuclideanCenter,e0)
              << std::endl;
    std::cout << mv_get_bladecoeff(EuclideanCenter,e1)
       << "," << mv_get_bladecoeff(EuclideanCenter,e2)
       << "," << mv_get_bladecoeff(EuclideanCenter,e3);
  #pragma gpc end
  return 0;
}
```

Listing 1: *horizon.cpg* source file

mv_get_bladecoeff(homogeneousCenter,e0)
As you might have guessed from the code above, the command mv_get_bladecoeff(homogeneousCenter,e0) will retrieve the value of the coefficient of blade e0 from the multivector homogeneousCenter. It works accordingly for other multivectors like EuclideanCenter and other blades such as e1, e2 and e3. A full list of all blades of Conformal Geometric Algebra may be found in section 4, table 4.

Helper Functions
Above, mv_get_bladecoeff is called a helper function. There are many other possibilities for accessing multivector blade coefficients, which are for example useful for reading and writing from/to arrays or OpenCL-vectors. All helper functions are explained in the Language Specification in section 5. For quick reference, refer to table 5 directly.

#pragma-layers
The structure of the code of listing 1 consists of three layers, also called blocks:

- **#pragma clucalc-block:**
  contains pure CLUCalc-code.
- **#pragma gpc-block:**
  contains C/C++-code and helper functions.
- **outer block:**
  contains pure C/C++-code.

Scoping
Gaalop Precompiler allows for regular scope handling across **#pragma**-blocks. Refer to section 6 for more information on this feature.

2. In the same directory as the *horizon.cpg* (project directory) create the *CMakeLists.txt* build script:

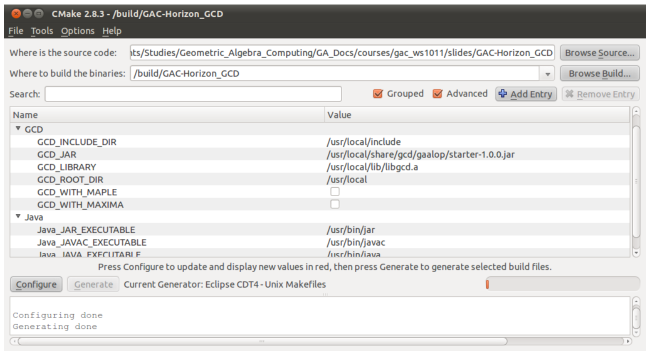
Beware: Do **NOT copy-and-paste** this code. The PDF format uses strange character encodings that have lead to a huge variety of errors in the past. A much better way is to copy the source code package from `http://www.gaalop.de/wp-content/uploads/gpc_tutorials.zip`.

```
PROJECT(horizon)
SET(CMAKE_MODULE_PATH ${PROJECT_SOURCE_DIR})
FIND_PACKAGE(GPC)
GPC_CXX_ADD_EXECUTABLE(horizon "horizon.cpg")
```

Listing 2: *CMakeLists.txt* build script

3. The Gaalop Precompiler installation contains the file *FindGPC.cmake* under *share/cmake-2.8/Modules*. Copy this file to the project directory, where *horizon.cpg* and *CMakeLists.txt* reside.
4. Start CMake.
5. Fill in the source directory (first input field). Fill in the destination directory (second input field).
6. In the window opening, choose GNU make generator.
7. Click Configure.
8. Fill in the root path to Gaalop GPC in the *GPC_ROOT_DIR* field. On Linux this should be automatically discovered.
9. Click Configure again. All other Gaalop GPC-related paths should now be discovered.
10. Click Generate.
    Figure 2 shows how CMake may look like after Configuration and Generation.
11. Open the CMake generated destination directory in your terminal or console.
    Enter make (Unix) or MinGW (Windows) and confirm with the Enter key. Wait until the build processes finishes.

    Hint: Using CMake and Gaalop GPC, builds are also easily possible with Visual Studio, Borland Builder or any other build tool of your choice.
12. Start the compiled application (figure 3).
    Unix: *./horizon*
    Windows: *horizon.exe*



Where is the source code: ts/Studies/Geometric_Algebra_Computing/GA_Docs/courses/gac_ws1011/slides/GAC-Horizon_GCD

Where to build the binaries: /build/GAC-Horizon_GCD

Search:

✓ Grouped ✓ Advanced

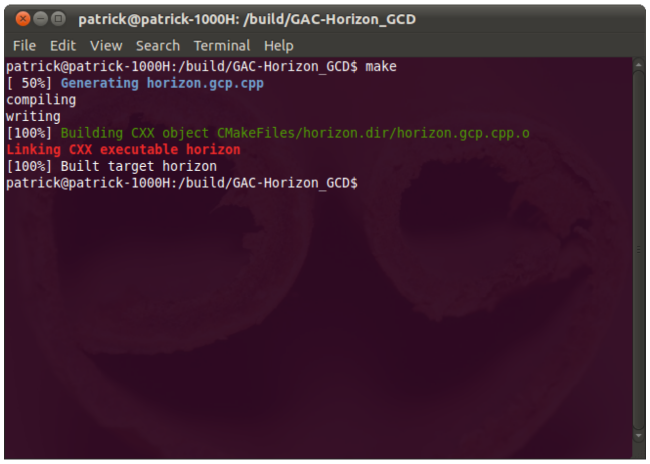
Name	Value
▼ GCD	
GCD_INCLUDE_DIR	/usr/local/include
GCD_JAR	/usr/local/share/gcd/gaalop/starter-1.0.0.jar
GCD_LIBRARY	/usr/local/lib/libgcd.a
GCD_ROOT_DIR	/usr/local
GCD_WITH_MAPLE	☐
GCD_WITH_MAXIMA	☐
▼ Java	
Java_JAR_EXECUTABLE	/usr/bin/jar
Java_JAVAC_EXECUTABLE	/usr/bin/javac
Java_JAVA_EXECUTABLE	/usr/bin/java

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Eclipse CDT4 - Unix Makefiles

```
Configuring done
Generating done
```

Figure 2: CMake-configuration for Gaalop GPC for C++


patrick@patrick-1000H: /build/GAC-Horizon_GCD
File Edit View Search Terminal Help

```
patrick@patrick-1000H:/build/GAC-Horizon_GCD$ make
[ 50%] Generating horizon.gcp.cpp
compiling
writing
[100%] Building CXX object CMakeFiles/horizon.dir/horizon.gcp.cpp.o
Linking CXX executable horizon
[100%] Built target horizon
patrick@patrick-1000H:/build/GAC-Horizon_GCD$
```

Figure 3: Screenshot of Gaalop GPC for C++ build process.

3 How To Use Gaalop GPC for OpenCL

The Horizon example is not perfectly suited as an OpenCL application, but here is how the horizon could be computed for thousands of observers at once.

1. Create the file *horizon.clg* with the following contents:

```
__kernel void horizonKernel(__global float* circleCenters,
                            __global const float* points,
                            const unsigned int num_points){
  const int id = get_global_id(0);
  #pragma gpc begin
    P = VecN3(points[id],
              points[id]+num_points,
              points[id]+2*num_points);
    #pragma clucalc begin
      r = 1;
      S = e0-0.5*r*r*einf;
      C = S^(P+(P.S)*einf);

      ?homogeneousCenter = C*einf*C;
      ?scale = -homogeneousCenter.einf;
      ?EuclideanCenter = homogeneousCenter / scale;
    #pragma clucalc end
    circleCenters = mv_to_stridedarray(EuclideanCenter,
                                       id, num_points,
                                       e1,e2,e3);
  #pragma gpc end
}
```

Listing 3: *horizon.clg* OpenCL kernel source file

Refer to point 1 of section 2 for more information on the syntax and the **#pragma**-statements. You may also refer to section 5 (table 5) for direct information on the mv_to_stridedarray() helper function.

2. The fastest way to carry out the following steps is to copy one of the examples that come with your OpenCL distribution and modify it according to your needs. Only the most important parts of the code are pointed out in the following listings. The code resides in the *horizon.cpp* source-file. Remember to put all files (*horizon.cpp*, *horizon.clg* and *CMakeLists.txt*) into the same directory (project directory).

3.
```
// list platforms
std::vector<cl::Platform> platforms;
cl::Platform::get(&platforms);
std::cout << "listings_platforms\n";
for (std::vector<cl::Platform>::const_iterator it =
         platforms.begin(); it != platforms.end(); ++it)
std::cout << it->getInfo<CL_PLATFORM_NAME> () << std::endl;

// create context
cl_context_properties properties[] = {CL_CONTEXT_PLATFORM,
         (cl_context_properties)(platforms[0])(), 0 };
cl::Context context(CL_DEVICE_TYPE_ALL, properties);
std::vector<cl::Device> devices = context.getInfo<
         CL_CONTEXT_DEVICES> ();
cl::Device& device = devices.front();

// create command queue
cl::CommandQueue commandQueue(context, device);
```

Listing 4: List platforms and create context and command queue.

4.
```
// settings
const size_t numPoints = 10000;
```

```
cl_float circleCenters[3*numPoints];
cl_float points[3*numPoints];
```

Listing 5: Create a host buffer.

5.
```
// Allocate the OpenCL buffer memory objects for source
// and result on the device GMEM
clDeviceVector<cl_float> dev_circle_centers(context,
        commandQueue,numPoints * 3,CL_MEM_READ_ONLY);
clDeviceVector<cl_float> dev_points(context,
        commandQueue,numPoints * 3,CL_MEM_READ_ONLY);
```

Listing 6: Create a device buffer with the same size.

6.
```
// Asynchronous write of data to GPGPU device
dev_points = points;
```

Listing 7: Copy the host buffer to the device buffer.

7.
```
// read the OpenCL program from source file
std::string sourceString;
readFile(sourceString, "horizon.gcl.cl");
cl::Program::Sources clsource(1, std::make_pair(
        sourceString.c_str(), sourceString.length()));
cl::Program program(context, clsource);

// build
program.build(devices);
std::cout
<< program.getBuildInfo<CL_PROGRAM_BUILD_LOG>(device)
<< std::endl;

// create kernel and functor
cl::Kernel horizonKernel(program, "horizonKernel");
cl::KernelFunctor horizonFunctor =
        horizonKernel.bind(commandQueue,
                cl::NDRange(numPoints),cl::NullRange);
```

Listing 8: Load the OpenCL-kernel.

8.
```
// Launch kernel
horizonFunctor(dev_circle_centers.getBuffer(),
        dev_points.getBuffer());

// Synchronous/blocking read of results,
// and check accumulated errors
dev_circle_centers.copyTo(circleCenters);
```

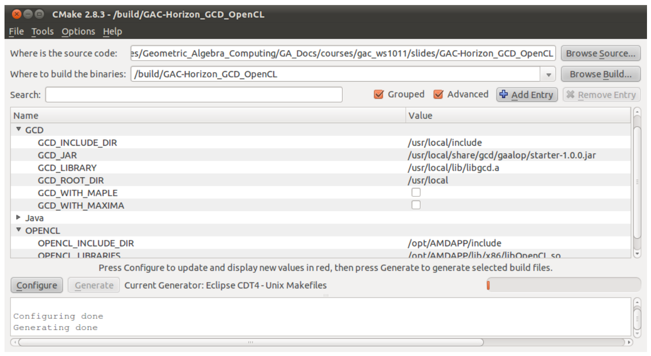
Listing 9: Set the device buffers as kernel arguments and start the kernel by using the functor.

9.
```
// Synchronous/blocking read of results,
// and check accumulated errors
dev_circle_centers.copyTo(circleCenters);
```
Listing 10: Read back the results from device to host.

10.
```
// print first circle center
std::cout << circleCenters[0] << "," << circleCenters[1]
         << "," << circleCenters[2] << std::endl;
```
Listing 11: Print the center of the first circle.

11.
```
CMAKE_MINIMUM_REQUIRED(VERSION 2.6)
PROJECT(horizon)
SET(CMAKE_MODULE_PATH ${PROJECT_SOURCE_DIR})
FIND_PACKAGE(OpenCL REQUIRED)
FIND_PACKAGE(GPC REQUIRED)
GPC_OPENCL_ADD_EXECUTABLE(horizon
                              "horizon.cpp"
                              "horizon.clg")
```
Listing 12: Create the file *CMakeLists.txt* with the following contents.

12. The Gaalop Precompiler installation contains the file *FindGPC.cmake* under *share/cmake-2.8/Modules*. Copy this file to the project directory, where *horizon.cpg* and *CMakeLists.txt* reside.
13. Start CMake.
14. Fill in the source directory (first input field). Fill in the destination directory (second input field).
15. In the window opening, choose GNU make as generator.
16. Click Configure.
17. Fill in the path to Gaalop GPC in the *GPC_ROOT_DIR* field.
18. Set *OPENCL_INCLUDE_DIR* to the include directory of your OpenCL distribution and *OPENCL_LIBRARIES* to the corresponding library. (Hint: ATI Stream SDK OpenCL library is located in */lib/x86/*OpenCL.lib*.)
19. Click Configure again.
20. Click Generate.
    Figure 4 shows how CMake may look like after Configuration and Generation.
21. Open the CMake generated destination directory in your terminal or console.
    Enter make (Unix) or MinGW (Windows) and confirm with the Enter key. Wait until the build processes finishes.



Where is the source code: es/Geometric_Algebra_Computing/GA_Docs/courses/gac_ws1011/slides/GAC-Horizon_GCD_OpenCL

Where to build the binaries: /build/GAC-Horizon_GCD_OpenCL

Search:

[✓ Grouped] [✓ Advanced]

Name	Value
GCD	
GCD_INCLUDE_DIR	/usr/local/include
GCD_JAR	/usr/local/share/gcd/gaalop/starter-1.0.0.jar
GCD_LIBRARY	/usr/local/lib/libgcd.a
GCD_ROOT_DIR	/usr/local
GCD_WITH_MAPLE	☐
GCD_WITH_MAXIMA	☐
Java	
OPENCL	
OPENCL_INCLUDE_DIR	/opt/AMDAPP/include
OPENCL_LIBRARIES	/opt/AMDAPP/lib/x86/libOpenCL.so

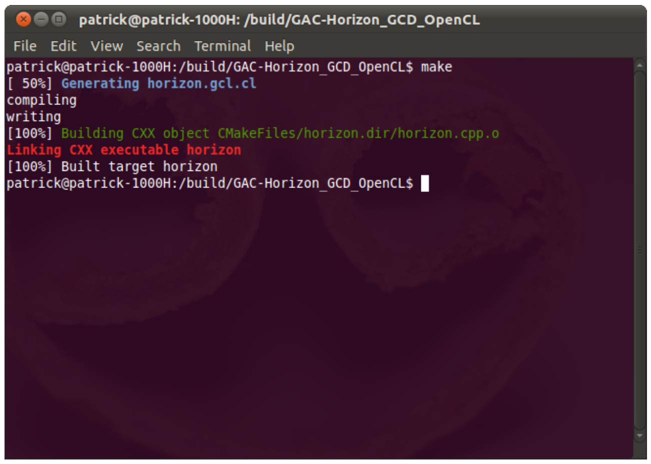
Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Eclipse CDT4 - Unix Makefiles

```
Configuring done
Generating done
```

Figure 4: CMake configuration for Gaalop GPC for OpenCL.

22. Start the compiled application (figure 5).
    Unix: *./horizon*
    Windows: *horizon.exe*


patrick@patrick-1000H: /build/GAC-Horizon_GCD_OpenCL
File Edit View Search Terminal Help

```
patrick@patrick-1000H:/build/GAC-Horizon_GCD_OpenCL$ make
[ 50%] Generating horizon.gcl.cl
compiling
writing
[100%] Building CXX object CMakeFiles/horizon.dir/horizon.cpp.o
Linking CXX executable horizon
[100%] Built target horizon
patrick@patrick-1000H:/build/GAC-Horizon_GCD_OpenCL$
```

Figure 5: Screenshot of a Gaalop GPC for OpenCL build process.

4 Multivectors and Blades of Conformal Geometric Algebra

An element of Conformal Geometric Algebra (CGA) is referred to as a multivector. A multivector consists of a linear combination of so called blades. Blades define the basis of CGA and are combinations of the vectors e_1, e_2, e_3, e_0 and e_∞. All possible blades and their grading are listed in table 4.

blade	**grade**
1	0
e_1	1
e_2	1
e_3	1
e_∞	1
e_0	1
$e_1 \wedge e_2$	2
$e_1 \wedge e_3$	2
$e_1 \wedge e_\infty$	2
$e_1 \wedge e_0$	2
$e_2 \wedge e_3$	2
$e_2 \wedge e_\infty$	2
$e_2 \wedge e_0$	2
$e_3 \wedge e_\infty$	2
$e_3 \wedge e_0$	2
$e_\infty \wedge e_0$	2
$e_1 \wedge e_2 \wedge e_3$	3
$e_1 \wedge e_2 \wedge e_\infty$	3
$e_1 \wedge e_2 \wedge e_0$	3
$e_1 \wedge e_3 \wedge e_\infty$	3
$e_1 \wedge e_3 \wedge e_0$	3
$e_1 \wedge e_\infty \wedge e_0$	3
$e_2 \wedge e_3 \wedge e_\infty$	3
$e_2 \wedge e_3 \wedge e_0$	3
$e_2 \wedge e_\infty \wedge e_0$	3
$e_3 \wedge e_\infty \wedge e_0$	3
$e_1 \wedge e_2 \wedge e_3 \wedge e_\infty$	4
$e_1 \wedge e_2 \wedge e_3 \wedge e_0$	4
$e_1 \wedge e_2 \wedge e_\infty \wedge e_0$	4
$e_1 \wedge e_3 \wedge e_\infty \wedge e_0$	4
$e_2 \wedge e_3 \wedge e_\infty \wedge e_0$	4
$e_1 \wedge e_2 \wedge e_3 \wedge e_\infty \wedge e_0$	5

Table 1: The 32 blades, that a multivector in 5D-Conformal Geometric Algebra is composed of.

5 **Gaalop Precompiler** Language Specification

Multivectors have a limited number of blades. For example in Conformal Geometric Algebra their size is limited to 32 blades. A multivector storage for CGA therefore has to save a maximum of 32 blade coefficients. A naive approach may therefore simply save the maximum number of coefficients in an array.

The problem with this approach is, that the number of blades grows exponentially with dimensionality. A 9D-Algebra [3] for example, that is proven to be useful in some cases, has exactly 512 blades and 512 blade coefficients, which are too many to save them efficiently in an array for each multivector. Since we want to support even higher dimensions, this is not an option.

Fortunately, the simple observation that the majority of multivector blade coefficient of a multivector equals zero, helps us to overcome this problem. The obvious solution is to save only non-zero blade coefficients. This technique is explained in full detail in section 7. To assist with this approach, several helper functions are defined in table 5.

The purpose of these helper functions, listed in table 5, is the transformation between multivectors and C/C++/OpenCL/CUDA language concepts like **float**-variables, arrays, or vectors. For example, mv_get_bladecoeff() is responsible for extracting a blade coefficient from a multivector, whereas mv_from_array() constructs a multivector from a C-like array.

coeff = mv_getbladecoeff(mv,blade);	Get the coefficient of blade blade of multivector mv.
mv = mv_from_vec(vec);	Construct multivector mv from OpenCL-vector vec.
mv = mv_from_array(array,blades,..);	Construct multivector mv from array array.
mv = mv_from_stridedarray(array,index,stride,blades ,...);	Construct multivector mv from array array at index index with stride stride. Example mv = mv_from_stridedarray(array,0,nummvs, e1,e2,e3,e0, einf);.
array = mv_to_array(mv,blades,...);	Write the blades blades ,... of multivector mv to array array. Example array = mv_to_array(mv,e1,e2,e3,e0,einf);.
array = mv_to_stridedarray(mv,index,stride,blades ,...);	Write the blades blades ,... of multivector mv to array array at index index with stride stride. Example array = mv_to_stridedarray(mv,0,nummvs, e1,e2,e3,e0, einf);.
vec = mv_to_vec(mv);	Write the multivector mv to OpenCL-vector vec.

Table 2: Gaalop GPC helper functions

6 Multivector Scoping

The Scoping feature allows multivectors from one **#pragma** gpc-block to be accessed in other **#pragma** gpc-blocks.

The following code is valid Gaalop GPC-syntax:

```
#pragma gpc begin
#pragma clucalc begin // block A
        mv1 = ...;
        mv2 = ...;
        ?a = mv1*mv2;
#pragma clucalc end

        ... // some C++ code

#pragma clucalc begin // block B
        // automatically imports variable a from block A
        ?b = a + 10;
#pragma clucalc end
```

```
#pragma gpc end
```
Listing 13: Simplified way of reusing multivectors from previous **#pragma**-blocks.

The solution guarantees correct scoping. For example, the following listing will cause a compilation error:

```
#pragma gpc begin
        { // scope 1
#pragma clucalc begin // block A
                mv1 = ...;
                mv2 = ...;
                ?a = mv1*mv2;
#pragma clucalc end
        }

... // some code

        { // scope 2
#pragma clucalc begin // block B
                ?b = a + 10; // Compilation will fail,
                // because a was declared in a different scope.
#pragma clucalc end
        }
#pragma gpc end
```
Listing 14: Multivectors are not available in different scopes.

Whereas outer scopes are imported into inner scopes as usual (listing 15). The scoping rules work in a way a programmer would expect them to work, without any knowledge of the underlying concept.

```
#pragma gpc begin
        { // begin outer scope
#pragma clucalc begin // block A
                mv1 = ...;
                mv2 = ...;
                ?a = mv1*mv2;
#pragma clucalc end

                ... // some C++ code

                { // begin inner scope
#pragma clucalc begin // block B
                        ?b = a + 10; // Will work as expected.
#pragma clucalc end
                } // end inner scope
        } // end outer scope
#pragma gpc end
```
Listing 15: Outer scope multivectors are handled as expected.

7 Compressed Multivector Storage

The naive approach to multivector storage is to save all multivector blade coefficients in one array sequentially, including the ones equal to zero. This leads to a non-optimal memory and cache efficiency, ultimately with higher dimensional algebras. A higher efficiency is achieved by only storing the non-zero entries of a multivector in one array sequentially.

An example output, including all meta-info, may then look like listing 16. Note that these aspects are internal details. It is not necessary to understand this listing in order to start programming with Gaalop GPC. The listing is intended for those who wish to understand the internal works of Gaalop GPC.

```
//#pragma gpc multivector V0_t_dt
float V0_t_dt[6];
//#pragma gpc multivector V1_t_dt
float V1_t_dt[6];

//#pragma gpc multivector_component V1_t_dt e1^e2 V1_t_dt[0]
V1_t_dt[0] = ((I_2 - I_1) * V013 * V023 - am12) / I_3;
//#pragma gpc multivector_component V1_t_dt e1^e3 V1_t_dt[1]
V1_t_dt[1] = ((I_3 - I_1) * V012 * V023 + am13) / I_2;
//#pragma gpc multivector_component V1_t_dt e1^einf V1_t_dt[2]
V1_t_dt[2] = (-(array_lmom[0] / mass));
//#pragma gpc multivector_component V1_t_dt e2^e3 V1_t_dt[3]
V1_t_dt[3] = ((I_3 - I_2) * V012 * V013 - am23) / I_1;
//#pragma gpc multivector_component V1_t_dt e2^einf V1_t_dt[4]
V1_t_dt[4] = (-(array_lmom[1] / mass));
//#pragma gpc multivector_component V1_t_dt e3^einf V1_t_dt[5]
V1_t_dt[5] = (-(array_lmom[2] / mass));
//#pragma gpc multivector_component V0_t_dt e1^e2 V0_t_dt[0]
V0_t_dt[0] = dt / 2.0 * V1_t_dt[0] + array_V0[(index) +
        0 * (numMolecules)];
//#pragma gpc multivector_component V0_t_dt e1^e3 V0_t_dt[1]
V0_t_dt[1] = dt / 2.0 * V1_t_dt[1] + array_V0[(index) +
        1 * (numMolecules)];
//#pragma gpc multivector_component V0_t_dt e1^einf V0_t_dt[2]
V0_t_dt[2] = dt / 2.0 * V1_t_dt[2] + array_V0[(index) +
        2 * (numMolecules)];
//#pragma gpc multivector_component V0_t_dt e2^e3 V0_t_dt[3]
V0_t_dt[3] = dt / 2.0 * V1_t_dt[3] + array_V0[(index) +
        3 * (numMolecules)];
//#pragma gpc multivector_component V0_t_dt e2^einf V0_t_dt[4]
V0_t_dt[4] = dt / 2.0 * V1_t_dt[4] + array_V0[(index) +
        4 * (numMolecules)];
//#pragma gpc multivector_component V0_t_dt e3^einf V0_t_dt[5]
V0_t_dt[5] = dt / 2.0 * V1_t_dt[5] + array_V0[(index) +
        5 * (numMolecules)];
```

Listing 16: An example output of codegen−compressed.

References

[1] Patrick Charrier. Geometric algebra enhanced precompiler for c++ and opencl. Master's thesis, TU Darmstadt, 2012.

[2] Dietmar Hildenbrand. *Foundations of Geometric Algebra Computing*. Springer, 2013.

[3] Julio Zamora-Esquivel. G6,3 geometric algebra. In *ICCA9, 7th International Conference on Clifford Algebras and their Applications*, 2011.