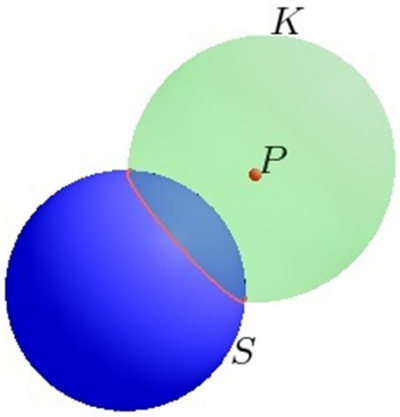
Geometric Algebra Computing

Gaalop GPC for OpenCL / Molecular Dynamics

20.11.2014

TECHNISCHE UNIVERSITÄT DARMSTADT

Dr.-Ing. Dietmar Hildenbrand
Technische Universität Darmstadt

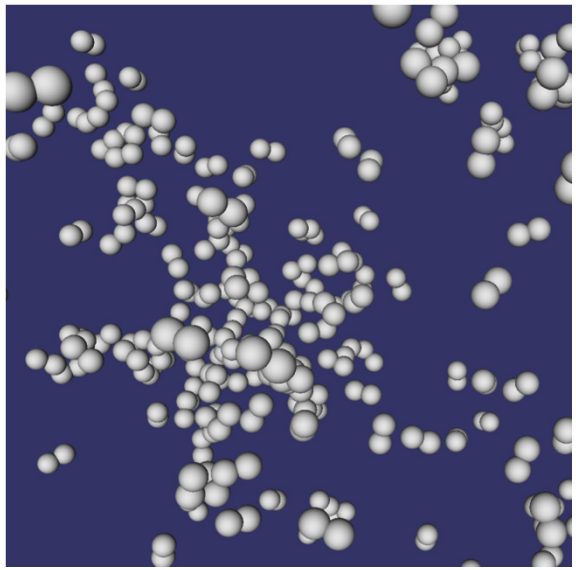


Gaalop GPC for OpenCL …

TECHNISCHE UNIVERSITÄT DARMSTADT

- … available for Windows and Linux (www.gaalop.de).

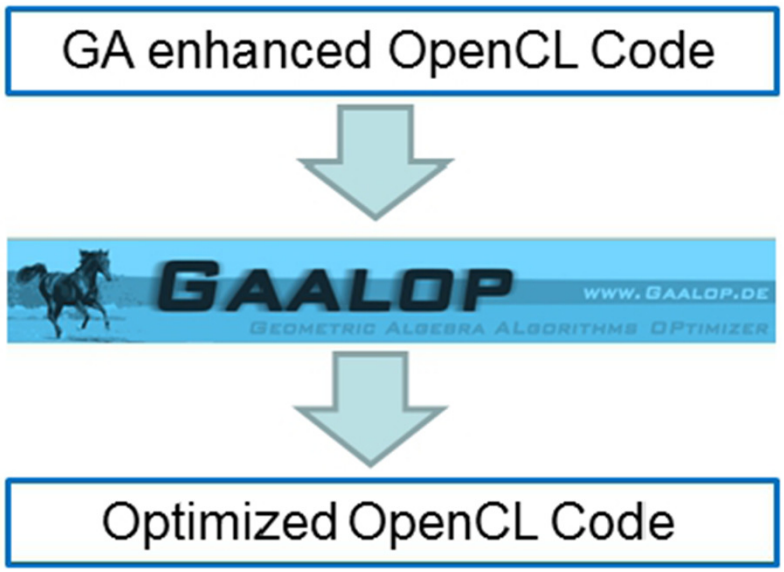
- Application example: Molecular dynamics …



Gaalop Precompiler

TECHNISCHE UNIVERSITÄT DARMSTADT

- Gaalop GPC for OpenCL



List of Precompiler Functions

TECHNISCHE UNIVERSITÄT DARMSTADT

coeff = mv_getbladecoeff(mv,blade);	Get the coefficient of blade **blade** of multivector **mv**.
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** with stride **stride**. Example **array = mv_to_array (mv,nummvs, e1,e2,e3,e0,einf);**.
vector = mv_to_vector(mv, blades,...);	Write the multivector **mv** to vector **vector**.
mv = mv_from_vector(vector,blades,..);	Construct multivector **mv** from vector **vector**.
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** with stride **stride**.

Strided Arrays

TECHNISCHE UNIVERSITÄT DARMSTADT

Nonstrided arrays are simply a concatenation of instances in memory. For example, having a structure

```
struct Vec {
        float x;
        float y;
        float z;
};
```

and creating an array containing N instances of this structure will yield the following image in memory:

x_0	y_0	z_0	x_1	y_1	z_1	x_2	y_2	z_2	$\dots$	x_{N-1}	y_{N-1}	z_{N-1}

Strided arrays are simply a concatenation of individual structure elements in memory, thereby breaking with the traditional memory layout:

x_0	x_1	x_2	$\dots$	x_{N-1}	y_0	y_1	y_2	$\dots$	y_{N-1}	z_0	z_1	z_2	$\dots$	z_{N-1}

As can be seen, we now put into the array all x-elements of all structures, followed by all y-elements of all structures, followed by all z-elements. This may seem unusual at first, but yields coalesced memory access when the GPU threads read their data into the register space. Coalesced reads are by far the fastest reads possible from GPU global memory. Similar advantages arise from coalesced writes, which also occur in the following examples.

Gaalop Precompiler

TECHNISCHE UNIVERSITÄT DARMSTADT

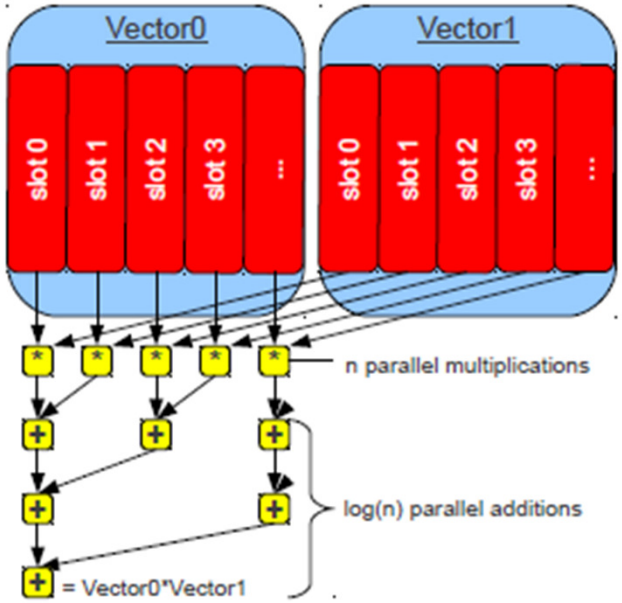
- Horizon example
- in OpenCL

```
__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

}
```

Advantages for GPUs

TECHNISCHE UNIVERSITÄT DARMSTADT

- Sum of products -> SIMD

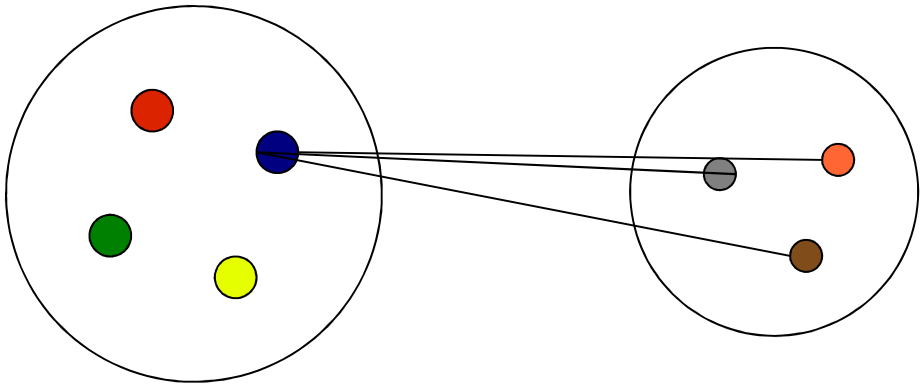


- Few special cases
- Optimized memory management

Molecular Dynamics in a Nutshell

TECHNISCHE UNIVERSITÄT DARMSTADT

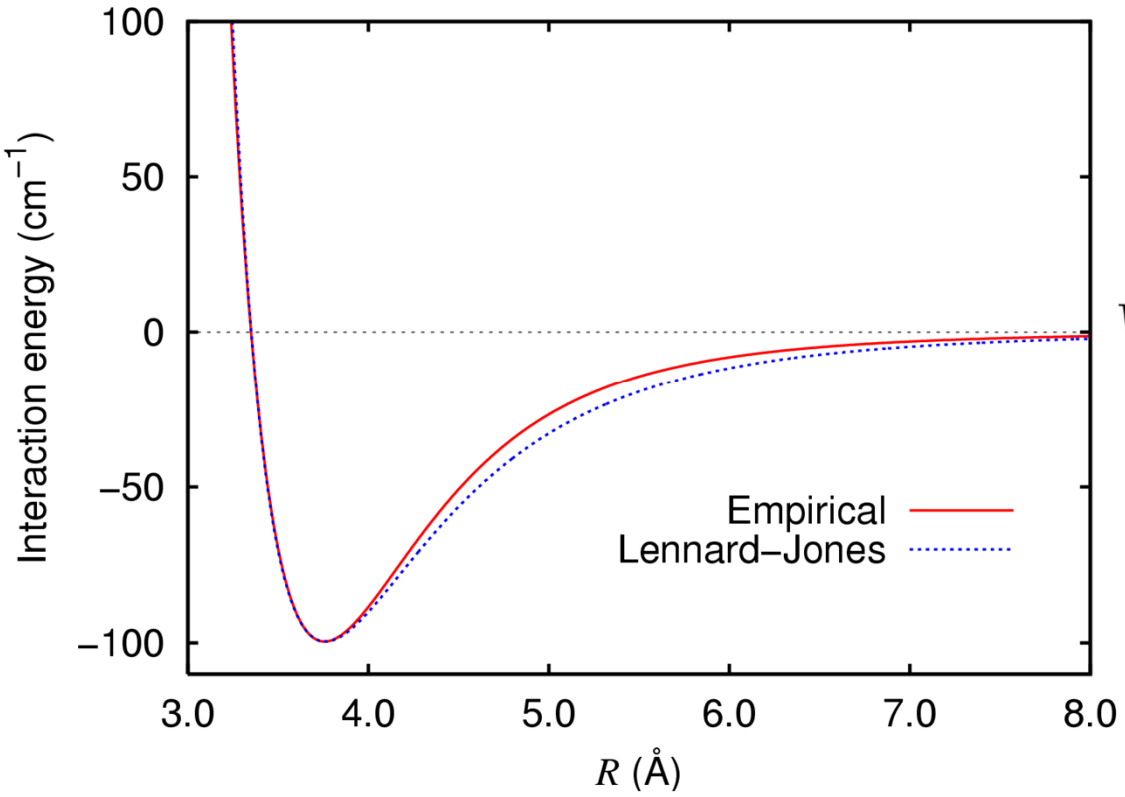
- A Molecule is a compound of several atoms,
- which are assumed to be static inside the molecule.
- Every atom sends out attraction or repulsion forces to every other atom.
- These forces then result in a movement of the molecules according to Newton's and Euler's laws.
- This is simulated for 1000s of molecules in parallel.



Molecular dynamics simulation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Lennard Jones potential



www.wikipedia.de

$$V(\mathbf{r}) = 4\epsilon \left[\left(\frac{\sigma}{\mathbf{r}} \right)^{12} - \left(\frac{\sigma}{\mathbf{r}} \right)^{6} \right]$$

$$\mathbf{r} = \mathbf{r}_i - \mathbf{r}_j \in \mathbb{R}^3$$

Molecular dynamics simulation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Lennard Jones force
  - is the negative gradient of the Lennard Jones potential

$$F(\mathbf{r}) = -\nabla_{\mathbf{r}} V$$

  - and results in the following force vector acting upon a pair of atoms

$$F(\mathbf{r}) = 24\epsilon \frac{\mathbf{r}}{|\mathbf{r}|} \left[2 \frac{\sigma^{12}}{|\mathbf{r}|^{13}} - \frac{\sigma^{6}}{|\mathbf{r}|^{7}} \right]$$

  - All forces (and torques) acting on a particular molecule can be summed up to a single net force.
  - With Newton's law *F=m*a* this force results in an accelaration of the molecule.

Conventional molecular dynamics solver

TECHNISCHE UNIVERSITÄT DARMSTADT

- Conventional Velocity Verlet timestep integration method

1. Calculate: $\vec{x}(t+\Delta t) = \vec{x}(t) + \vec{v}(t)\,\Delta t + \frac{1}{2}\vec{a}(t)(\Delta t)^2$
2. Calculate: $\vec{v}(t+\frac{\Delta t}{2}) = \vec{v}(t) + \frac{\vec{a}(t)\Delta t}{2}$
3. Derive $\vec{a}(t+\Delta t)$ from the interaction potential.
4. Calculate: $\vec{v}(t+\Delta t) = \vec{v}(t+\frac{\Delta t}{2}) + \frac{\vec{a}(t+\Delta t)\Delta t}{2}$

www.wikipedia.de

Note that those steps have to be computed separately

1. for the translation part
2. for the rotational part

CGA molecular dynamics solver

TECHNISCHE UNIVERSITÄT DARMSTADT

- Velocity Verlet in Conformal Geometric Algebra:

$$
\begin{aligned}
&\text{Displacement propagation:} && D(t+\Delta t) &&= D(t) + \dot{D}(t)\Delta t + \tfrac{1}{2}\ddot{D}(t)(\Delta t)^2 \\
&\text{Midpoint velocity:} && V_b\left(t+\tfrac{\Delta t}{2}\right) &&= V_b(t) + \dot{V}_b(t)\tfrac{\Delta t}{2} \\
&\text{Acceleration:} && \dot{V}_b(t+\Delta t) &&= e_\infty \dot{v}_b(t+\Delta t) - e_{123}\dot{\omega}_b(t+\Delta t) \\
&\text{Velocity propagation:} && V_b(t+\Delta t) &&= V_b\left(t+\tfrac{\Delta t}{2}\right) + \tfrac{1}{2}\dot{V}_b(t+\Delta t)\Delta t
\end{aligned}
$$

with

$$\dot{D} = \frac{1}{2} D V_b \left(= \frac{1}{2} V_b D = \frac{1}{2} D V_b D^{-1} D \right)$$

$$\ddot{D} = \frac{1}{2}\dot{D}V_b + \frac{1}{2} D\dot{V}_b = \frac{1}{4} D V_b^2 + \frac{1}{2} D\dot{V}_b$$

and with the Euclidean pseudoscalar

$$e_{123} = e_1 \wedge e_2 \wedge e_3$$

Florian Seybold (HLRS), based on David Hestenes's work

- CGA combines translational and rotational parts into one compact algorithm.

General Solver concept

TECHNISCHE UNIVERSITÄT DARMSTADT

- solver is separated into 3 parts

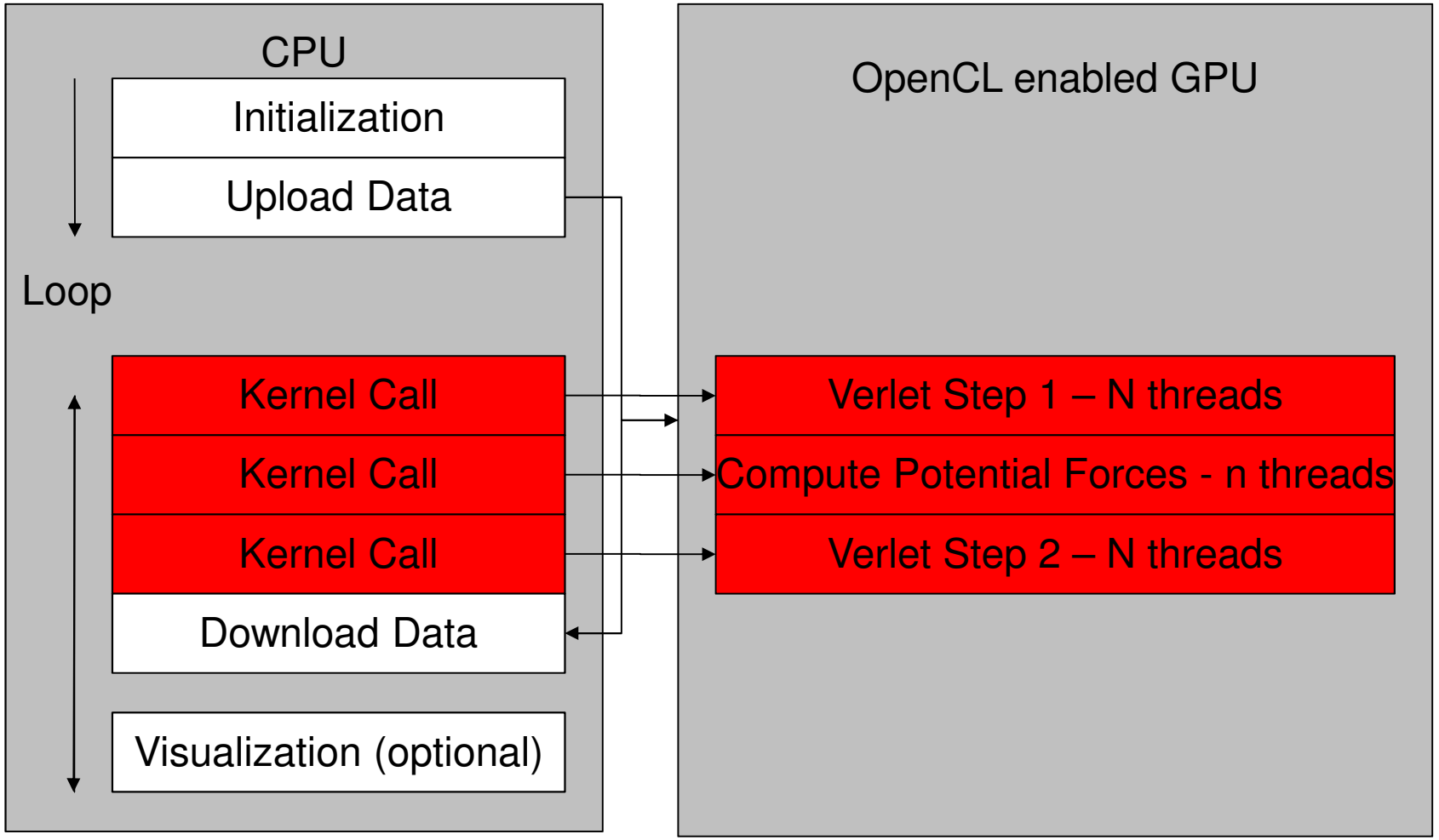
1. molecule verlet time integration step 1
   - updates the molecule's position and orientation
   - N computations for N molecules

2. computation of potential forces
   - updates each molecule's force and torque
   - n x (n-1) computations for n atoms

3. molecule verlet time integration step 2
   - updates the molecule's linear and angular velocity
   - N computations for N molecules

OpenCL-Solver concept

TECHNISCHE UNIVERSITÄT DARMSTADT



Implementation

TECHNISCHE UNIVERSITÄT DARMSTADT

Listing 13.1 shows the host/CPU code for the initialization of the molecular dynamics simulation.

```
void convertStandardModelToSolverModel(const BaseModel& model) {
  const MoleculeVector& molecules = model.molecules;
  const AtomVector& atoms = model.atoms;
  const int numMolecules = molecules.size();
  const size_t numAtomPositions = atoms.size();
  for (int index = 0; index < numMolecules; ++index) {
    // get molecule
    const Molecule& molecule = molecules[index];

    #pragma gpc begin
      //map to multivectors
      lp = mv_from_array(molecule.lpos,e1,e2,e3);
      rotor = mv_from_array(molecule.arot,1,
                            e2^e3,e3^e1,e1^e2);
      lv = mv_from_array(molecule.lvel,e1,e2,e3);
      av = mv_from_array(molecule.avel,e1,e2,e3);
```

Implementation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Gaalop Precompiler for OpenCL code:

```
#pragma clucalc begin
        // calculation
  const floatmv D1_t = 0.5 * D0_t * V0_t;
  const floatmv D2_t = 0.5 * D1_t * V0_t + 0.5 * D0_t * V1_t;

  const floatmv D0_t_dt = D0_t + D1_t * dt + 0.5 * D2_t * dt * dt;
  const floatmv V0_t_05dt = V0_t + 0.5 * V1_t * dt;
#pragma clucalc end
```

Displacement propagation: $D(t+\Delta t) = D(t) + \dot{D}(t)\Delta t + \frac{1}{2}\ddot{D}(t)(\Delta t)^2$

Midpoint velocity: $V_b\left(t+\frac{\Delta t}{2}\right) = V_b(t) + \dot{V}_b(t)\frac{\Delta t}{2}$

Acceleration: $\dot{V}_b(t+\Delta t) = e_\infty \dot{v}_b(t+\Delta t) - e_{123}\dot{\omega}_b(t+\Delta t)$

Velocity propagation: $V_b(t+\Delta t) = V_b\left(t+\frac{\Delta t}{2}\right) + \frac{1}{2}\dot{V}_b(t+\Delta t)\Delta t$

with

$$\dot{D} = \frac{1}{2}DV_b \left(= \frac{1}{2}V_b D = \frac{1}{2}DV_b D^{-1} D\right)$$

$$\ddot{D} = \frac{1}{2}\dot{D}V_b + \frac{1}{2}D\dot{V}_b = \frac{1}{4}DV_b^2 + \frac{1}{2}D\dot{V}_b$$

and with the Euclidean pseudoscalar

$$e_{123} = e_1 \wedge e_2 \wedge e_3$$

Implementation

TECHNISCHE UNIVERSITÄT DARMSTADT

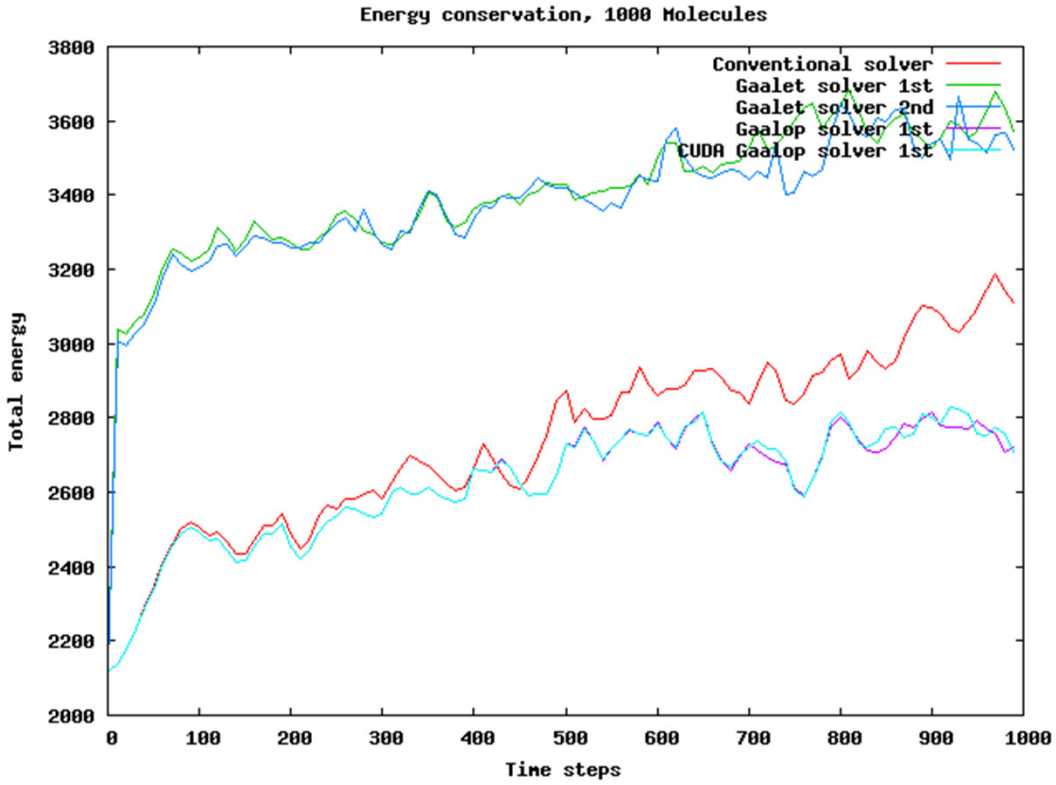
```
      D0_t = mv_from_stridedarray(array_D0,index,numMolecules,
                          1,e1^e2,e1^e3,e1^einf,
                          e2^e3,e2^einf,e3^einf,e1^e2^e3^einf);
      V0_t = mv_from_stridedarray(array_V0,index,numMolecules,
                          e1^e2,e1^e3,e1^einf,
                          e2^e3,e2^einf,e3^einf);
      V1_t = mv_from_stridedarray(array_V1,index,numMolecules,
                          e1^e2,e1^e3,e1^einf,
                          e2^e3,e2^einf,e3^einf);
      #pragma clucalc begin
        ?D1_t = 0.5 * D0_t * V0_t;
        ?D2_t = 0.5 * D1_t * V0_t + 0.5 * D0_t * V1_t;

        ?D0_t_dt = D0_t + D1_t * dt + 0.5 * D2_t * dt * dt;
        ?V0_t_05dt = V0_t + 0.5 * V1_t * dt;
      #pragma clucalc end
        //map from multivectors
        array_D0 = mv_to_stridedarray(D0_t_dt,index,numMolecules,
                                1,e1^e2,e1^e3,e1^einf,
                                e2^e3,e2^einf,e3^einf,e1^e2^e3^einf);
        array_V0 = mv_to_stridedarray(V0_t_dt,index,numMolecules,
                                e1^e2,e1^e3,e1^einf,
                                e2^e3,e2^einf,e3^einf);
    #pragma gpc end
}
```

Listing 13.2. Compute-intensive Gaalop GPC for OpenCL code for the first step of the velocity Verlet numerical integration of the displacement versor and velocity screw for a molecule.

Results

TECHNISCHE UNIVERSITÄT DARMSTADT

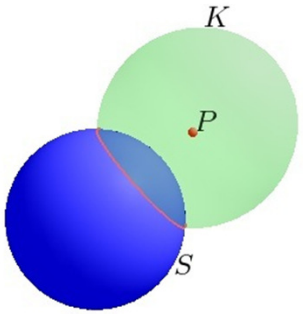
- Better runtime performance
- Better Numerical Stability (Energy Conservation)



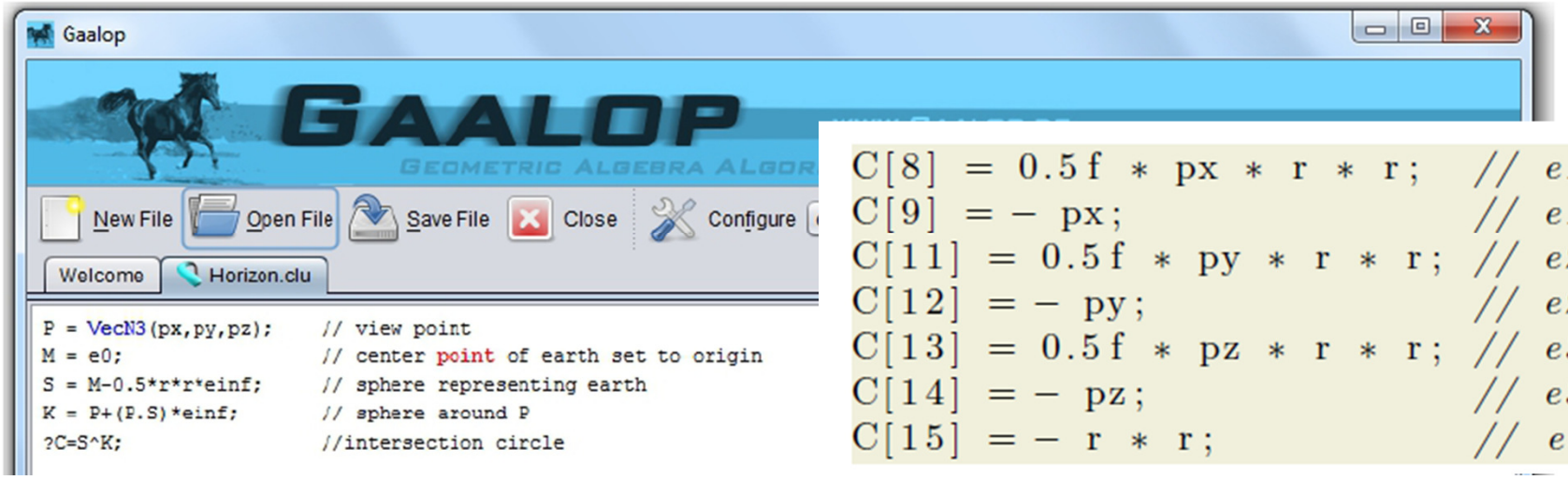
Conclusion

TECHNISCHE UNIVERSITÄT DARMSTADT

- Geometrically intuitive



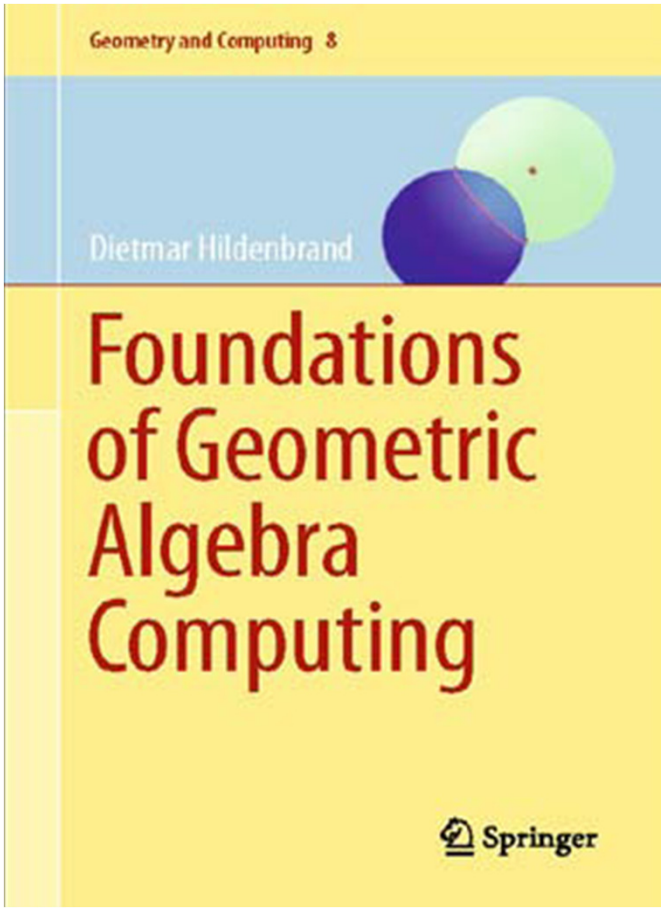
- Fast and robust implementations



Reference

TECHNISCHE UNIVERSITÄT DARMSTADT

- „Foundations of Geometric Algebra Computing“
- Dietmar Hildenbrand
- Springer, 2013



Thanks a lot

Google Dietmar Hildenbrand