

# Relation-based algorithms and the parallelization of stencil-based computations

N. Botta, C. Ionescu, C. Linstead, R. Klein

with contributions of:

- J. Gerlach, Fraunhofer Inst. für Rechnerarch. und Softwaretechnik.
- A. Priesnitz, Chalmers University of Technology.

## Overview

- Triangle centers example and relation-based algorithms
- Parallelization of relation-based algorithms
- Research directions and examples
- To do

## Triangle centers example

- vertex-triangle table  $vt : V \leftarrow T$   
 $vtss : [[\text{Nat}]]$ ,  $vtss!!t$  are the vertexes of  $t$
- vertex coordinates  $xv : \mathbb{R}^2 \leftarrow V$   
 $xvs : [(\text{Real}, \text{Real})]$ ,  $xvs!!v$  is the coordinate of  $v$

## Triangle centers example

- vertex-triangle table  $vt : V \leftarrow T$   
 $vtss : [[\text{Nat}]]$ ,  $vtss!!t$  are the vertexes of  $t$
- vertex coordinates  $xv : \mathbb{R}^2 \leftarrow V$   
 $xvs : [(\text{Real}, \text{Real})]$ ,  $xvs!!v$  is the coordinate of  $v$

Given a list of triangles  $ts : [\text{Nat}]$ , compute their centers:

## Triangle centers example

- vertex-triangle table  $vt : V \leftarrow T$   
 $vtss : [[\text{Nat}]]$ ,  $vtss!!t$  are the vertexes of  $t$
- vertex coordinates  $xv : \mathbb{R}^2 \leftarrow V$   
 $xvs : [(\text{Real}, \text{Real})]$ ,  $xvs!!v$  is the coordinate of  $v$

Given a list of triangles  $ts : [\text{Nat}]$ , compute their centers:

```
tcs = [ (xvs!!(vts!!0) + xvs!!(vts!!1) + xvs!!(vts!!2)) / 3  
      |  
      vts = vtss!!t, t <- ts ]
```

## h-f-R relation-based algorithm

- Replacing  $vt$  with a generic relation  $R : Y \leftarrow X$ ,

## h-f-R relation-based algorithm

- Replacing  $vt$  with a generic relation  $R : Y \leftarrow X$ ,
- $xv$  with a generic function  $f : T \leftarrow Y$  and

## h-f-R relation-based algorithm

- Replacing  $vt$  with a generic relation  $R : Y \leftarrow X$ ,
- $xv$  with a generic function  $f : T \leftarrow Y$  and
- $1/3 \cdot \text{sum}$  with a generic function  $h : T' \leftarrow [T]$ .

## h-f-R relation-based algorithm

- Replacing  $vt$  with a generic relation  $R : Y \leftarrow X$ ,
- $xv$  with a generic function  $f : T \leftarrow Y$  and
- $1/3 \cdot \text{sum}$  with a generic function  $h : T' \leftarrow [T]$ .

yields a generic, relation-based algorithm:

```
rba: [[Nat]] -> [T] -> ([T] -> T') -> ([Nat] -> [T'])
```

```
(rba rss fs h) js = [h [fs!!i | i <- rss!!j] | j <- js]
```

## Relation-based algorithms

h-f-R like algorithms are general enough to suitably represent:

- the computation of geometrical properties of grid elements: center, area, normal vectors, etc.

## Relation-based algorithms

h-f-R like algorithms are general enough to suitably represent:

- the computation of geometrical properties of grid elements: center, area, normal vectors, etc.
- stencil-based algorithms on grids: numerical interpolation, integration, computation of differential operators, etc.

## Relation-based algorithms

h-f-R like algorithms are general enough to suitably represent:

- the computation of geometrical properties of grid elements: center, area, normal vectors, etc.
- stencil-based algorithms on grids: numerical interpolation, integration, computation of differential operators, etc.
- inference in Bayesian networks

however, a few more primitives are needed, e.g., ...

## Relation-based algorithms

... inversion and composition primitives in grid computations to

- compute neighborhoods:
  - $(vt^\circ)v$  are the triangles 'around'  $v$
  - $(vt^\circ \cdot vt)t$  are the triangles sharing one vertex with  $t$
  - $|(vt \cdot vt^\circ)v| - 1$  is the number of 'edges' around  $v$

## Relation-based algorithms

... inversion and composition primitives in grid computations to

- compute neighborhoods:

- $(vt^\circ)v$  are the triangles 'around'  $v$

- $(vt^\circ \cdot vt)t$  are the triangles sharing one vertex with  $t$

- $|(vt \cdot vt^\circ)v| - 1$  is the number of 'edges' around  $v$

- identify 'boundary' grid elements:

- $v$  boundary vertex  $\equiv |(vt \cdot vt^\circ)v| - 1 \neq |vt^\circ|$

## Parallelization of relation-based algorithms

- How to 'distribute'  $f$  and  $R$ -data on partitions ?

## Parallelization of relation-based algorithms

- How to 'distribute'  $f$  and  $R$ -data on partitions ?
- Which  $f$ -data are needed locally and stored remotely ?

## Parallelization of relation-based algorithms

- How to 'distribute'  $f$  and  $R$ -data on partitions ?
- Which  $f$ -data are needed locally and stored remotely ?
- Which  $f$ -data are stored locally and needed remotely ?

## Parallelization of relation-based algorithms

- How to 'distribute'  $f$  and  $R$ -data on partitions ?
- Which  $f$ -data are needed locally and stored remotely ?
- Which  $f$ -data are stored locally and needed remotely ?
- How to 'exchange' needed data between partitions ?

## Research directions

- 1 Develop a simple model of parallel computations to describe, specify, derive and test-implement parallel, distributed relation-based algorithms.

## Research directions

- 1 Develop a simple model of parallel computations to describe, specify, derive and test-implement parallel, distributed relation-based algorithms.

Example (drba specification):

```
(rba rss fs h) js = [h [f!!i | i <- rss!!j] | j <- js]
```

```
d2g ((drba drss dfs h) djs)
```

```
=
```

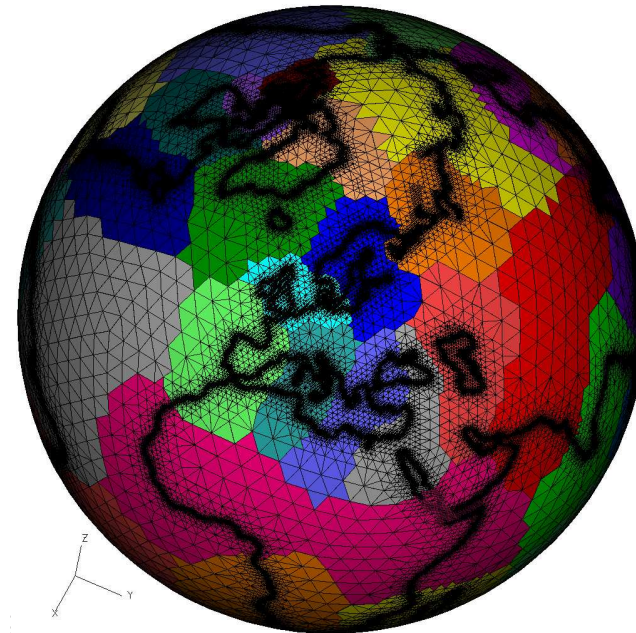
```
(rba (d2g drss) (d2g dfs) h) (d2g djs)
```

## Research directions

- 2 Develop a computational framework for building fast, distributed, parallel applications from:
  - distributed, parallel primitive patterns: h-f-R, conversion, composition;
  - application-specific distributed data structures, e.g. for triangulations;
  - user-defined application-specific  $h$ -functions, e.g. area.

## Example: partitioning, renumbering

```
int main(int argc, char** argv) {  
  
    const Nat n_crds = 3;  
    typedef Rectangular_Coordinate_System<n_crds> CS;  
    ...  
    Triangulation<CS> tri;  
    read_ascii(tri, file);  
    ...  
    init_triangle_vertex(tri);  
    init_triangle_vertex_triangle(tri);  
    subtract_id(r, triangle_vertex_triangle(tri));  
    ...  
    converse(tv, vt);  
    partition_source(vpart, tv, tpart);  
    Array<Nat> vperm = order_preserving_permutation(vpart);  
    renumber_target(vt, invert_permutation(vperm));  
    vpart = compose(vpart, vperm);  
    VERIFY(is_non_decreasing(vpart));  
    vx = compose(vx, vperm);  
    ...  
    return 0;  
}
```



188966 triangles, 43 partitions

## Example: semi-automatic parallelization of stencil-based computations, user viewpoint, sequential

```
int main(int argc, char** argv) {

    typedef Rectangular_Coordinate_System<2> CS;
    Triangulation<CS> tri;
    read_ascii(tri, argv[1]);

    Triangle_Areas<CS> triangle_areas(tri);
    Triangle_Divs_P1<CS, Real> triangle_divs_p1(fv, tri);

    Real a = 0.0; Real div = 0.0;
    for(Nat j = 0; j < n_triangles(tri); j++) {
        const Real aj = triangle_areas(j);
        a += aj;
        div += aj * triangle_divs_p1(j);
    }
    div /= a;

    return 0;
}
```

## Example: semi-automatic parallelization of stencil-based computations, user viewpoint, parallel distributed

```
int main(int argc, char** argv) {  
  
    initialize(argc, argv);  
  
    typedef Rectangular_Coordinate_System<2> CS;  
    Triangulation<CS> tri;  
    read_ascii(tri, argv[1], argv[2], argv[3]);  
  
    Triangle_Areas<CS> triangle_areas(tri);  
    Triangle_Divs_P1<CS, Real> triangle_divs_p1(fv, tri);  
  
    Real a = 0.0; Real div = 0.0;  
    for(Nat j = 0; j < local::n_triangles(tri); j++) {  
        const Real aj = triangle_areas(j);  
        a += aj;  
        div += aj * triangle_divs_p1(j);  
    }  
    div /= a;  
  
    finalize();  
  
    return 0;  
}
```

**Example: semi-automatic parallelization of stencil-based computations, developer viewpoint:**

Provided application-specific distributed data structures, the parallelization of user-defined stencil-based functions requires two steps:

## **Example: semi-automatic parallelization of stencil-based computations, developer viewpoint:**

Provided application-specific distributed data structures, the parallelization of user-defined stencil-based functions requires two steps:

- step 1: wrap user-defined function in a h-type function class;

## **Example: semi-automatic parallelization of stencil-based computations, developer viewpoint:**

Provided application-specific distributed data structures, the parallelization of user-defined stencil-based functions requires two steps:

- step 1: wrap user-defined function in a h-type function class;
- step 2: implement a parallel algorithm in terms of a (parallel) h-f-R specialization.

## Example: semi-automatic parallelization of stencil-based computations, developer viewpoint, step 1

```
template<>
struct Rectangular_Coordinate_System<2>

    template<typename T>
    static
    T
    triangle_area(const Vector<T, 2>& a,
                  const Vector<T, 2>& b,
                  const Vector<T, 2>& c) {

        return T(0.5)
            *
            fabs( (b[0] - a[0])
                * (c[1] - a[1])
                - (c[0] - a[0])
                * (b[1] - a[1]));

    }

    // ...

};
```

```
template<class CS>
struct Triangle_Area {

    typedef Real return_type;

    static const Nat arity = 1;

    template<class RAD>
    Real
    operator()(const RAD& x) const {

        return CS::triangle_area(x[0],
                                  x[1],
                                  x[2]);

    }

};
```

## Example: semi-automatic parallelization of stencil-based computations, developer viewpoint, step 2

```
namespace SPMD_Distributed {

    template<class Coordinate_System>
    class
    Triangle_Areas : public RBA<Triangle_Area<Coordinate_System>,
                          Array<Vector<Real, Coordinate_System::n_crds> >,
                          Reg_Rel<3> > {

    public:
        typedef Triangle_Area<Coordinate_System> H;
        typedef Array<Vector<Real, Coordinate_System::n_crds> > F;
        typedef Reg_Rel<3> R;

        Triangle_Areas(Triangulation<Coordinate_System>& t) :
            RBA<H, F, R>(h, local::vertex_coordinates(t), local::vertex_triangle(t)), h() {
            this->init_offsets();
            this->init_pos_of_exch_access_exch_tables();
            this->complete_f();
        }

        const H h;
    };

} // namespace SPMD_Distributed
```

## To do

- Systematic application of the Haskell BSP model (talk Ionescu) to the analysis and derivation of parallel distributed relation based algorithms.
- Analysis and implementation of h-f-R 'scatter' algorithms.
- Design and implementation of collections of intermediate complexity distributed data structures: relations, triangulations, ...
- Design and implementation of collections of application-dependent, stencil-based algorithms: `triangle_area`, `div_p1`, ...