Parallel & Distributed Computing: Lecture 30

Alberto Paoluzzi

December 11, 2019

Project A – Overview

Introduction

Produce a report, using Pandoc (Markdown + LaTeX), including:

1. the link to the personal project repository in GitHub;
2. for each parallelized / optimized ⟨*feature*⟩ you should produce:
   - src/⟨*feature*⟩.jl file, demostrating the working feature;
   - examples/⟨*feature*⟩.jl file, demostrating the working feature;
   - test/⟨*feature*⟩.jl file testing the feature implementation;
   - doc/⟨*feature*⟩.md file with problems description, and type of parallelization / optimization;
   - study of the speed-up obtained, including a graph, in doc/⟨*feature*⟩.md.

The project must include at least 3 features modified / optimized.

For each feature, the project should discuss (why) the preference between shared memory (threads) and distributed parallelization (processes)

Changed / new files should be named as either ⟨*old_name*⟩*_matricola* or ⟨*feature*⟩*_matricola*.

Overview of Arrangement pipeline 1/2

Input Facet selection, i.e., construction of the collection $\mathcal{S}_{d-1}$ from $\mathcal{S}_d$, using `LAR`.

Indexing Spatial index made by intersection of $\boxed{d \text{ interval-trees}}$ on bounding boxes of $\sigma \in \mathcal{S}_{d-1}$.

$\boxed{\text{Decomposition}}$ Pairwise $z = 0$ intersection of line segments in $\sigma \cup \mathcal{I}(\sigma)$, $\boxed{\text{for each } \sigma \in \mathcal{S}_{d-1}}$.

Congruence Graded bases of equivalence classes $C_k(U_k)$, with $U_k = X_k/R_k$ for $0 \leq k \leq 2$.

Connection Extraction of $(X^p_{d-1}, \partial^p_{d-1})$, maximal connected components of X_{d-1} $(0 \leq p \leq h)$.

$\boxed{\text{Bases}}$ Computation of redundant cycle basis $[\partial^+_d]^p$ for each p-component, via TGW.

Overview of Arrangement pipeline 2/2

$\boxed{\text{Boundaries}}$ Accumulation into $H \mathrel{+}= [o]^p$ (hole-set) of outer boundary cycle from each $[\partial_d^+]^p$.

Containment Computation of antisymmetric $\boxed{\text{containment relation } S}$ between $[o]^p$ holes in H.

Reduction Transitive R reduction of S and generation of forest of flat trees $\langle [o_d]^p, [\partial_d]^p \rangle$.

Adjoining of roots $[o_d]^r$ to (unique) outer cell, and non-roots $[\partial_d^+]^q$ to container cells.

Assembling Quasi-block-diagonal assembly of matrices relatives to isolated components $[\partial_d]^p$.

Output Global boundary map $[\partial_d]$ of $\mathcal{A}(\mathcal{S}_{d-1})$, and reconstruction of 0-chains of d-cells in X_d.

Overview of Boolean mapping

$\boxed{\text{Atomic description}}$ Boolean description of CSG terms as d-chains ("oracle" answering if atom $i \subset$ term j)

$\boxed{\text{pointInPolyhedron}}$ Computation of intersection number of a ray from a point with a 3D polyhedron boundary.

Other parallelizable algorithms in LAR

pointInPolygon Computation of intersection number of a ray from a point with a 2D polygon boundary.

Integration Monomial integration on triangular domains (2D/3D signed volumes and inertia)

CSG (Constructive Solid Geometry) expressions

$[\partial_3]$ columns as atoms of Boolean algebra

- See Paoluzzi et al., Finite Boolean Algebras for Solid Geometry Using Julia's Sparse Arrays, ArXiV, 2019

1. Start with an assembly of boundary LAR models $\mathcal{S}_{d-1}$
2. Compute the space Arrangement's $[\partial_3]$ matrix
3. Compute an internal point for each atom
4. Compute the subset of assembly terms that contain each atom
5. Compose the binary description (as 3-chains) of assembly terms
6. apply standard bitwise operators to any combination of binary terms

Parallelizable tasks

d interval-trees

1. Use either threads or processes to parallelize the task:

```
sp_idx = Lar.spaceindex(model)
```

1. Test the symmetry of computed relation.
2. Look within the file: LinearAlgebraicRepresentation/src/refactoring.jl

pointInPolygonClassification

1. Use either threads or processes to parallelize the task:

```
function pointInPolygonClassification(V,EV)
```

1. Test for one million 2D points.
2. Look within the files: LinearAlgebraicRepresentation/src/refactoring.jl LinearAlgebraicRepresentation/examples/2d/point_in_polygon.jl

Integration

1. See Vectorization in Julia
2. See SIMD and SIMD-intrinsics in Julia

```
""" The main integration routine """
function TT(
        tau::Array{Float64,2},
        alpha::Int, beta::Int, gamma::Int,
        signedInt::Bool=false )
```

3. Look within the files: LinearAlgebraicRepresentation/src/integr.jl https://github.com/cvdlab/lar-cc/blob/master/doc/pdf/integr.pdf
4. Test one/more graphical models made by triangles, from The Stanford 3D Scanning Repository.

Atomic description of terms of a CSG expression

1. See Finite Boolean Algebras for Solid Geometry Using Julia's Sparse Arrays
2. Optimize / parallelize the task (do not consider the interior `Arrangement` task)

   ```
   function bool3d(assembly)
   ```

3. Look within the files:
   - LinearAlgebraicRepresentation/src/bool3d.jl
   - LinearAlgebraicRepresentation/examples/3d/bool3d.jl
   - LinearAlgebraicRepresentation/examples/3d/randomcubes.jl
4. Test with `randomcubes.jl` with tens of sparse objects

pointInPolyhedron containment test

1. Use either threads or processes to parallelize the task:

```
for (k,point) in enumerate(innerpoints) # k runs on columm
    cells = containmenttest(point) # contents of columns
    #println(k," ",faces)
    rows = [span(h) for h in cells]
    for l in cat(rows)
        boolmatrix[k+1,l+1] = 1
    end
end
```

1. Test random 3D points against random 3D cubes (very high numbers)
2. Look within the files: LinearAlgebraicRepresentation/src/bool3d.jl

$\boxed{\text{aaaaaaaaaa}}$

$\boxed{aaaaaaaaaa}$

$\boxed{aaaaaaaaaa}$

aaaaaaaaaa