# GAIO.jl

## Preparing for a 1.0 release using julia 1.9

**April Herwig**

Department of Mathematics
Technical University of Munich

# Motivation

## Attractors

**Definition 1.** *An invariant set $A$ is attracting if there is a neighborhood $U$ of $A$ such that for every open set $V \supset A$ there is $K \in \mathbb{N}$ such that*

$$f^k(U) \subset V \quad \text{for all } k \geq K.$$

**Proposition 1.** *If $A$ is a closed attracting set then*

$$A = \bigcap_{k \in \mathbb{N}} f^k(U).$$

**Basic idea:** Successively refine an approximation of $A$ using subdivision

# Computing Attractors

## The subdivision algorithm

Generate a sequence $\mathcal{B}_0, \mathcal{B}_1, \mathcal{B}_2, \ldots$ of finite families of compact sets as follows:

Let $\mathcal{B}_0 = \{Q\}$, $\theta \in (0,1)$. For $k = 1, 2, \ldots$ do

- construct $\hat{\mathcal{B}}_k$ such that

$$|\hat{\mathcal{B}}_k| = |\mathcal{B}_{k-1}| \quad \text{and} \quad \operatorname{diam} \hat{\mathcal{B}}_k \leq \theta \cdot \operatorname{diam} \mathcal{B}_{k-1}.$$

- set

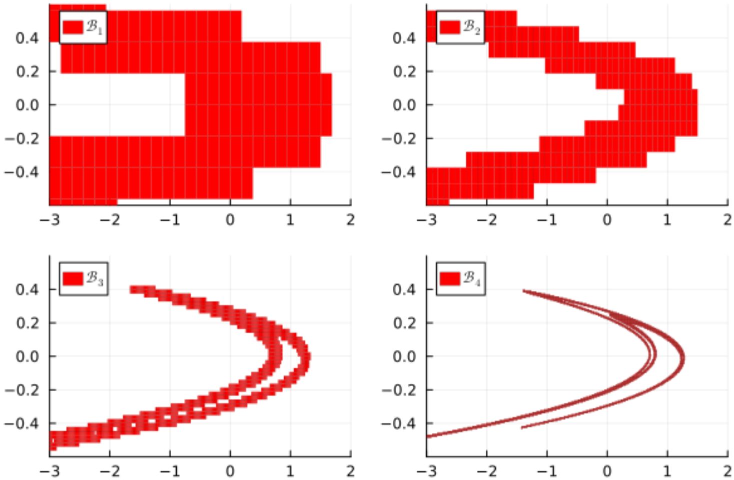
$$\mathcal{B}_k = f(\hat{\mathcal{B}}_k) \cap \hat{\mathcal{B}}_k, \quad \text{where}$$

$$f(\hat{\mathcal{B}}_k) \cap \hat{\mathcal{B}}_k \stackrel{def}{=} \{B \in \hat{\mathcal{B}}_k \mid \exists B' \in \hat{\mathcal{B}}_k : f(B') \cap B \neq \varnothing\}.$$

**Theorem 1.** *$|\mathcal{B}_k| \to A_Q$ as $k \to \infty$ in the Hausdorff metric.*

# Computing Attractors

## The subdivision algorithm



# Representation of Cubical Complexes

`BoxPartition`: partition the domain into equally sized grid of hypercubes, or "Boxes"

```
struct BoxPartition{N,T,I<:Integer}
    domain::Box{N,T}
    dims::SVector{N,I}
end
```

`TreePartition`: binary tree holding successive subdivisions

```
struct Node{I<:Integer}
    left::I
    right::I
end

struct TreePartition{N,T,I,V<:AbstractArray{Node{I}}}
    domain::Box{N,T}
    nodes::V
end
```

# Representation of Cubical Complexes

BoxSet: collections of Boxes within a partition

```
struct BoxSet{B,P<:AbstractBoxPartition{B},S<:AbstractSet} <: AbstractSet{B}
    partition::P
    indices::S
end
```

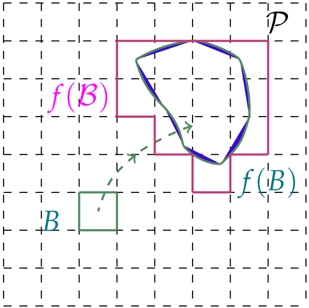
We can use the built-in set data types and setwise operations for BoxSets using multiple-dispatch

```
function Base.⊆(B₁::BoxSet, B₂::BoxSet)
    ( B₁.partition == B₂.partition ) && ( B₁.indices ⊆ B₂.indices )
end
```

# Cell Mapping

So how do we compute

$$f(\mathcal{B}) = \{B \in \mathcal{P} \mid \exists B' \in \mathcal{B} : f(B') \cap B \neq \varnothing\} \;?$$



# Cell Mapping
## test point sampling

```
function map_boxes(g::SampledBoxMap, source::BoxSet)

  B() = empty(source)                  # Function to initialize empty BoxSet
  P = source.partition

  @floop for box in source
    for p in g.domain_points(box)      # Generate sample points
      fp = typesafe_map(g, p)          # Wrap user-function output
      hit = cover(P, fp)               # Box in P covering the point fp
      @reduce(image = B() ∪ hit)       # Each thread collects hits,
    end                                # after loop completion the
  end                                  # result is reduced

  return image
end
```

# Cell Mapping

## test point sampling

- choose test points within $\mathcal{B}$ and record their images under $f$
- memory-efficient "lazy" test point sampling with `Generators`
- ensure type-stability to shorten generated code
- spread load across multiple compute threads using `@floop` macro
- collect hits and reduce per-thread result into single result using `@reduce` macro
- can harness the GPU using CUDA.jl

# Cell Mapping

All `BoxMap` types work under a common API: they must define

```
map_boxes(g::MyBoxMap, source::BoxSet)
construct_transfers(g::MyBoxMap, domain::BoxSet)
construct_transfers(g::MyBoxMap, domain::BoxSet, codomain::BoxSet)
```

You now have all the knowledge to understand `TransferOperator`, `BoxGraph` as well!

# Optimization: Solving "Time-To-First-Attractor"

Investigating the first execution shows an interesting problem: a GPU kernel gets compiled... even when no GPU is present?

Solve by converting the CUDA dependency to an extension

```
name = "GAIO"
uuid = "33d280d1-ac47-4b0f-9c2e-fa6a385d0226"
authors = ["The GAIO.jl Team"]
version = "1.0.0"

[deps]
...

[weakdeps]
CUDA = "052768ef-5323-5732-b1bb-66c8b64840ba"
...

[extensions]
CUDAExt = "CUDA"
```

# Optimization: Solving "Time-To-First-Attractor"

With dynamic dispatch reduced, we can now save precompiled code in a package image. To force precompilation of common workloads, use `PrecompileTools.jl`

```
using PrecomileTools

@setup_workload begin
  # set local variables for common workload, e.g. attractor of Henon map

  @compile_workload begin
    # track which code gets generated and force full compilation
  end

end
```