OpenABL: A Domain-Specific Language for Parallel and Distributed Agent-Based Simulations

Biagio Cosenza[1], Nikita Popov[1], Ben Juurlink[1], Paul Richmond[2], Mozhgan Kabiri Chimeh[2], Carmine Spagnuolo[3], Gennaro Cordasco[3], Vittorio Scarano[3]

[1]Technische Universität Berlin
Germany

[2]The University of Sheffield
United Kingdom

[3]University of Salerno
Italy

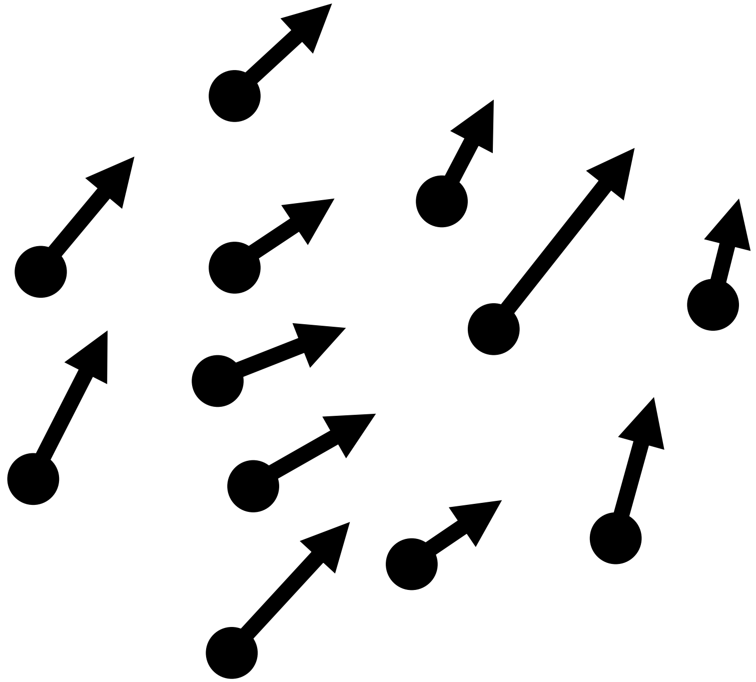
Outline

- Agent-based Simulations (ABS)
  - Parallel and distributed ABS
- OpenABL
  - Language
  - Compilation infrastructure
- Experimental evaluation

Agent-Based Simulations

the agent-based computational model is well-suited to the study of phenomena where agent populations are heterogeneous, there is no central control over individuals (autonomy), the space where the agents work is explicit (e.g., an n-dimensional grid), and agents only have local interactions with neighboring agents

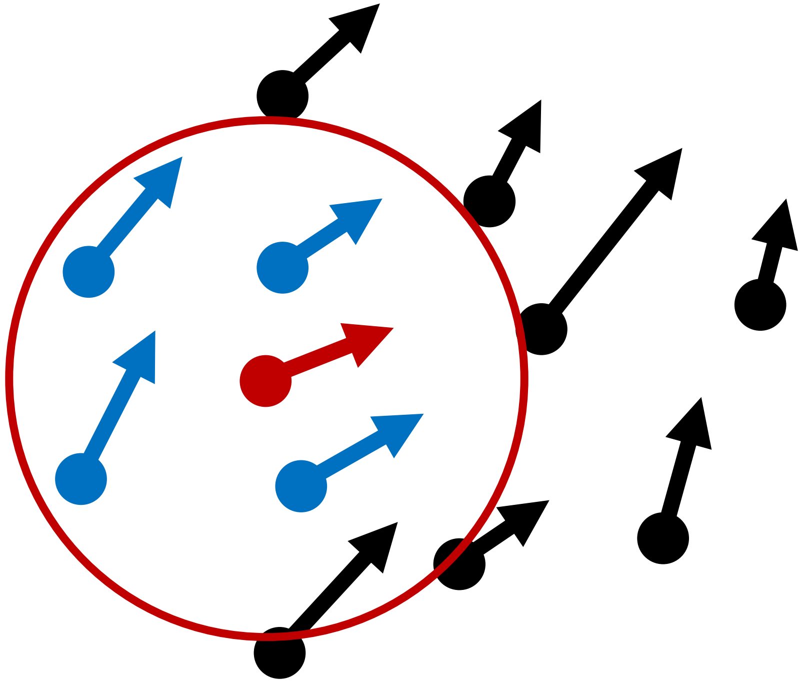
See "Agent-based computational models and generative social science". Epstein. Complexity (5) 1999. 41–60



Agent-Based Simulations

the agent-based computational model is well-suited to the study of phenomena where agent populations are heterogeneous, there is no central control over individuals (autonomy), the space where the agents work is explicit (e.g., an n-dimensional grid), and agents only have local interactions with neighboring agents

See "Agent-based computational models and generative social science". Epstein. Complexity (5) 1999. 41–60



Parallel and Distributed ABS: The Implementation Zoo

- ABS parallel implementations
  - Several problems from different fields
  - Different architectures and platforms
  - Different solutions
- OpenABL goals
  - Performance
  - Programmability
  - Portability
  - Reproducibility

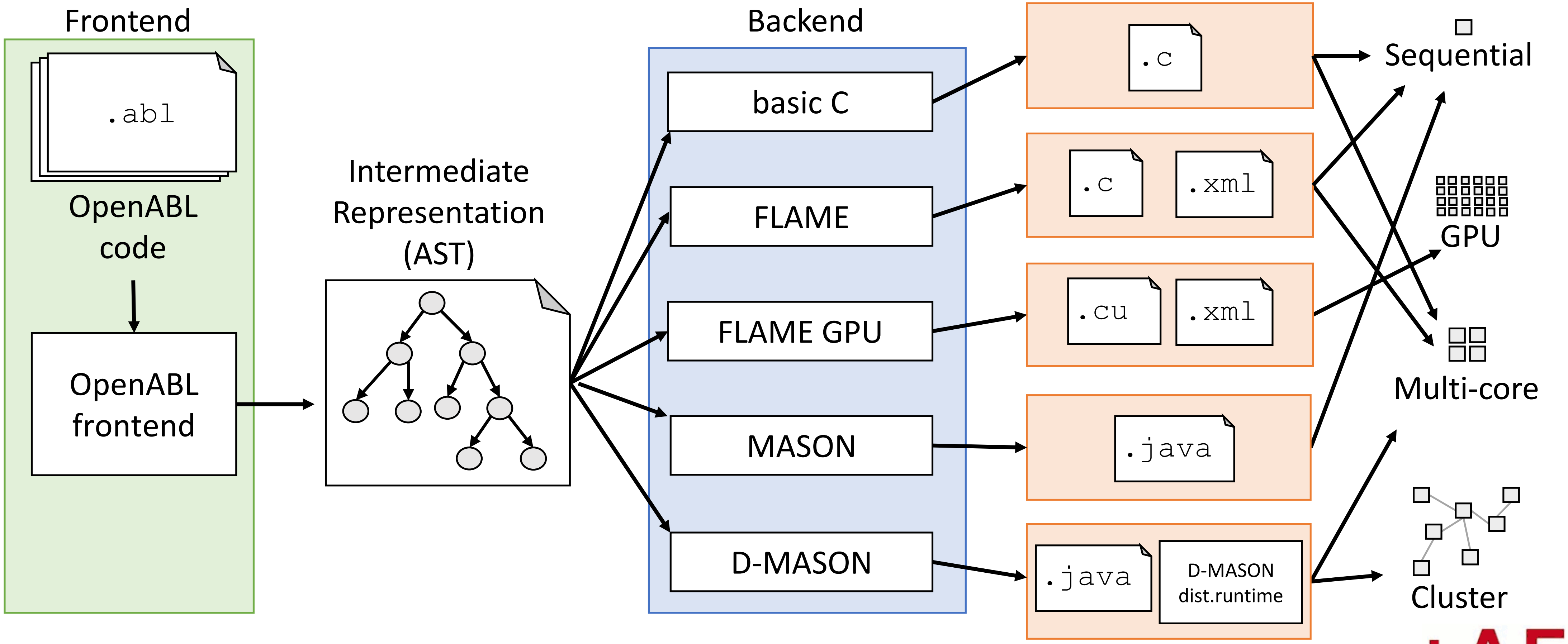
ecology
transport engineering
social science
economy
biology
animation and graphics

libraries

Mason
Flame
DistrSteer
D-Mason
FlameGPU
REPAST-HPC

CPU
GPU
multi-core CPU
HPC large-scale cluster
cloud computing

OpenABL: Overview



The OpenABL Language

- Example: simple agent motion
- Agent declaration
- Environment and param definition
- Step function
  - ➢ state of agent at time t computed from state of agents at time t-1
  - ➢ agents can be updated in parallel
  - ➢ using information from local neighborhood (`near`)
- Simulate
- Helper functions

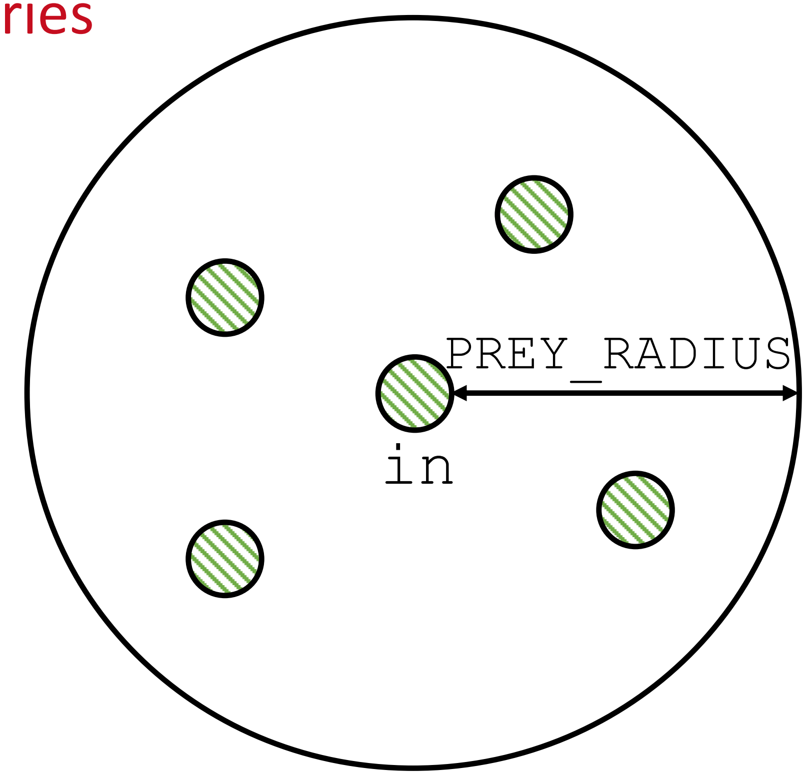
```
// Agent declarations
agent Point {
  position float3 pos;
}
// Simulation parameters and environment definition
float radius = 5;
float env_size = 100;
param int num_agents = 1000;
param int num_timesteps = 100;
environment { max: float3 ( env_size ) }
// Step function
step move_point ( Point in -> out) {
  // Move towards the average direction of the neighbors
  float3 dir = float3 (0);
  int num_neighbors = 0;
  for ( Point other : near (in , radius )) {
    dir += normalize ( other .pos - in.pos );
    num_neighbors += 1;
  }
  out.pos = clamp(in.pos+dir/num_neighbors,float3(env_size));
}
// Main code : Initialization and execution
void main() {
  for (int i : 0..num_agents )
    add ( Point {pos : random ( float3 ( env_size ))});
  simulate ( num_timesteps ) { move_point }
  save ("result.json");
}
```


AES
Embedded Systems Architecture

The OpenABL Language: Locality and Neighborhood Queries

- `near` with the homogenous types
- Example from `predator-prey`

```
step prey_flock(Prey in -> out) {
  // ...
  for(Prey py : near(in, PREY_RADIUS)) {
    // ...
    cohesion_velocity += ... py.pos;
  }
  // ...
  out.steer += cohesion_velocity;
}
```

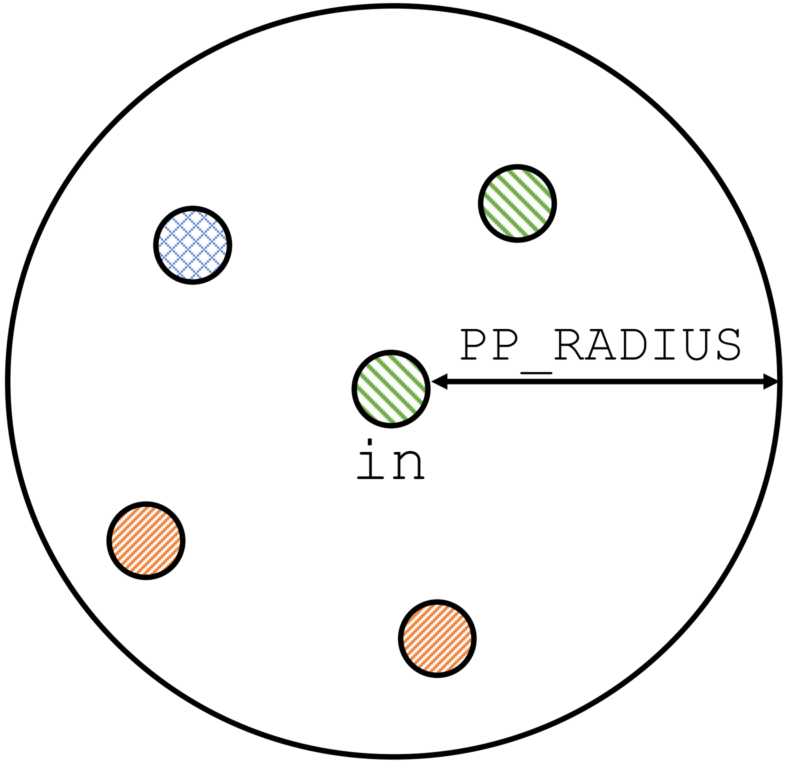


`Prey py : near(in,PREY_RADIUS)`

The OpenABL Language: Heterogeneity Support

- `near` with heterogenous types
- Example from `predator-prey`

```
step prey_avoid_pred(Prey in -> out) {
  // ...
  for(Predator pt : near(in, PP_RADIUS)) {
    // ...
    avoid_velocity += ... pt.pos
  }
  // ...
  out.steer += avoid_velocity;
}
```



`Predator pt : near(in,PP_RADIUS)`

The OpenABL Language: Dynamic Agent Addition and Removal

- Dynamically add and remove agents
- Example from `predator-prey`
  - ➢ probabilistic agent creation with `add`
  - ➢ same position of the father
- Challenging for distributed backend

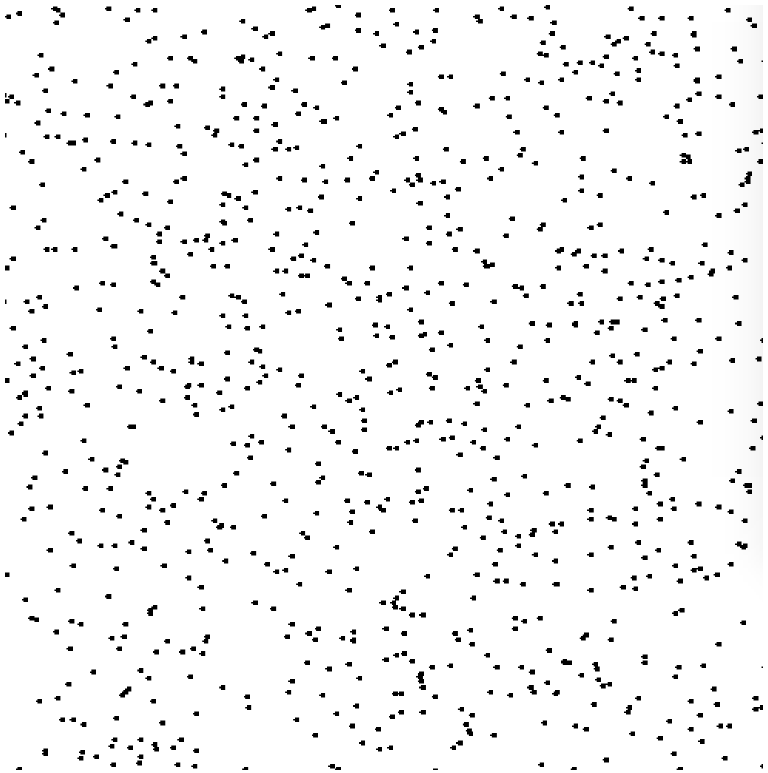
```
step prey_reproduction(Prey in -> out) {
  if (random(1.0) < REPRODUCE_PREY_PROB) {
    add(Prey {
      pos: in.pos, // same position
      dir: -in.dir, // opposite direction
      steer: -in.steer,
      life: in.life/2 // life split btwn fath.&child
    });
    out.life = in.life/2;
  }
}
```

The OpenABL Compiler

- Source-to-source
  - `flex` and `bison`
- The frontend produces a high-level intermediate representation
  - an AST, with domain-specific nodes
  - experimental support for optimizations (e.g., step functions merge)
- Backends
  - Mapping to library-specific model and concepts
  - C, Flame, FlameGPU, Mason, D-Mason
  - Visualization backend

OpenABL Examples

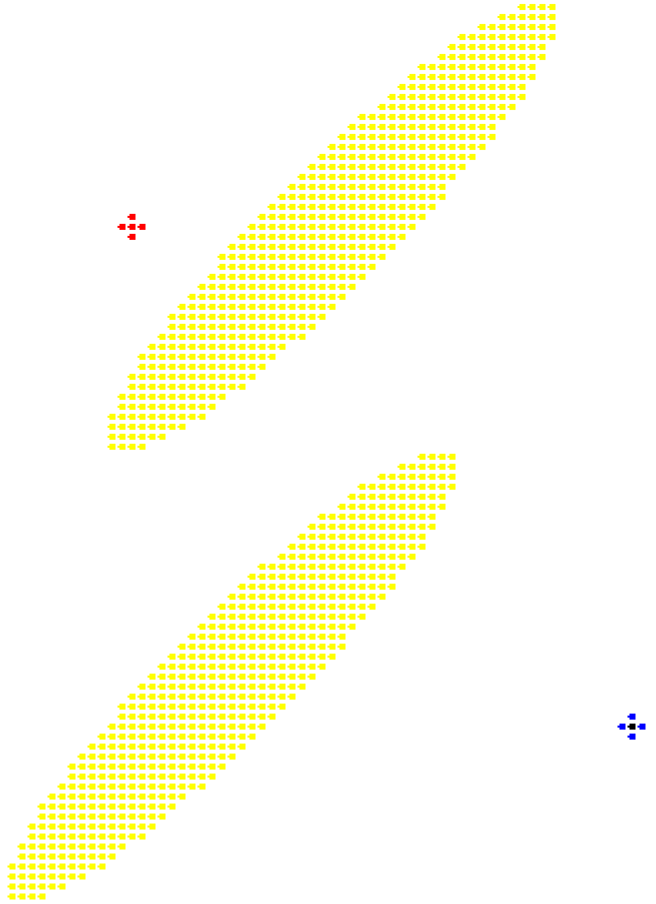
- Test benchmarks
  - Circle (1 type, 1 step)
  - Boids
  - Game of life
  - Sugarscape
  - Ants foraging
  - Predator-prey



Source: https://github.com/OpenABL/OpenABL/blob/master/examples/circle.abl

OpenABL Examples

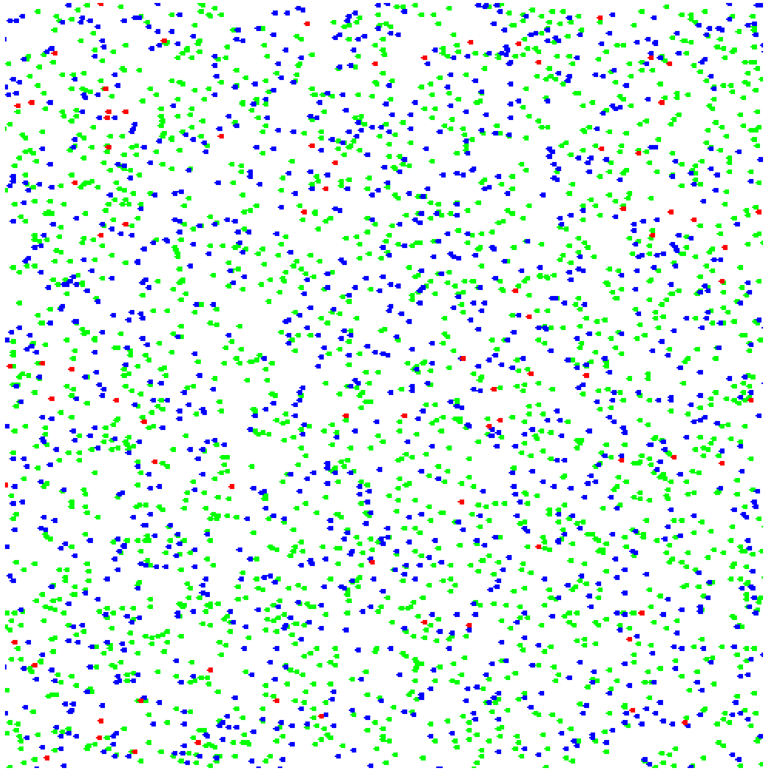
- Test benchmarks
  - Circle
  - Boids
  - Game of life
  - Sugarscape
  - Ants foraging (2 types, 3 steps)
  - Predator-prey



Source: https://github.com/OpenABL/OpenABL/blob/master/examples/ants.abl

OpenABL Examples

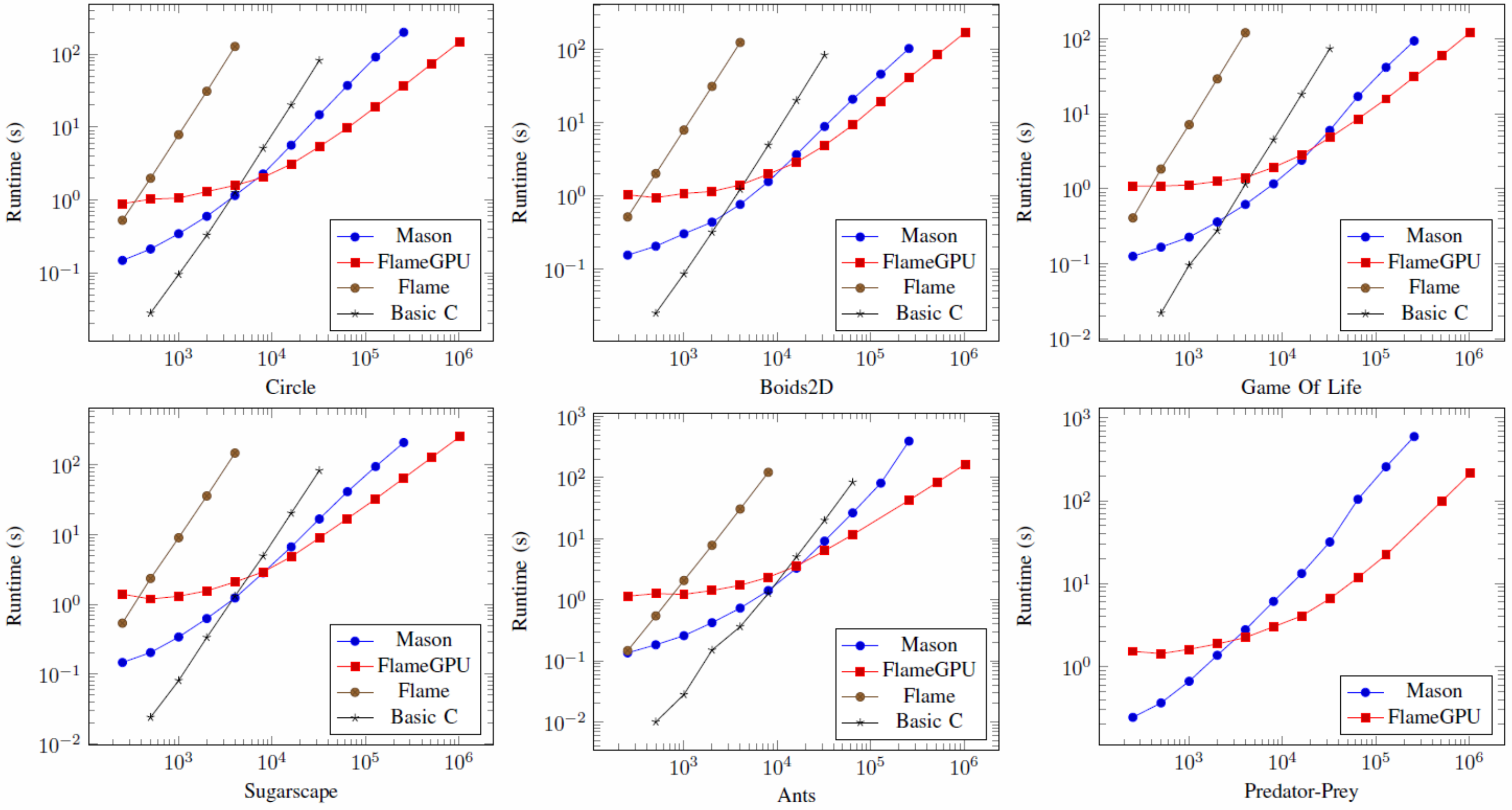
- Test benchmarks
  - Circle
  - Boids
  - Game of life
  - Sugarscape
  - Ants foraging
  - Predator-prey (3 types, 13 steps, add/remove)

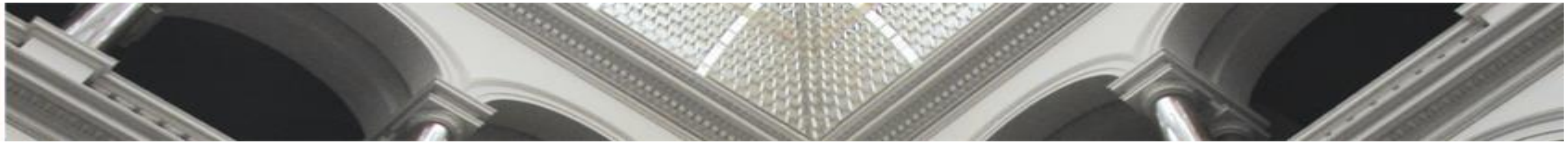


Source: https://github.com/OpenABL/OpenABL/blob/master/examples/predator_prey.abl

Experimental Results

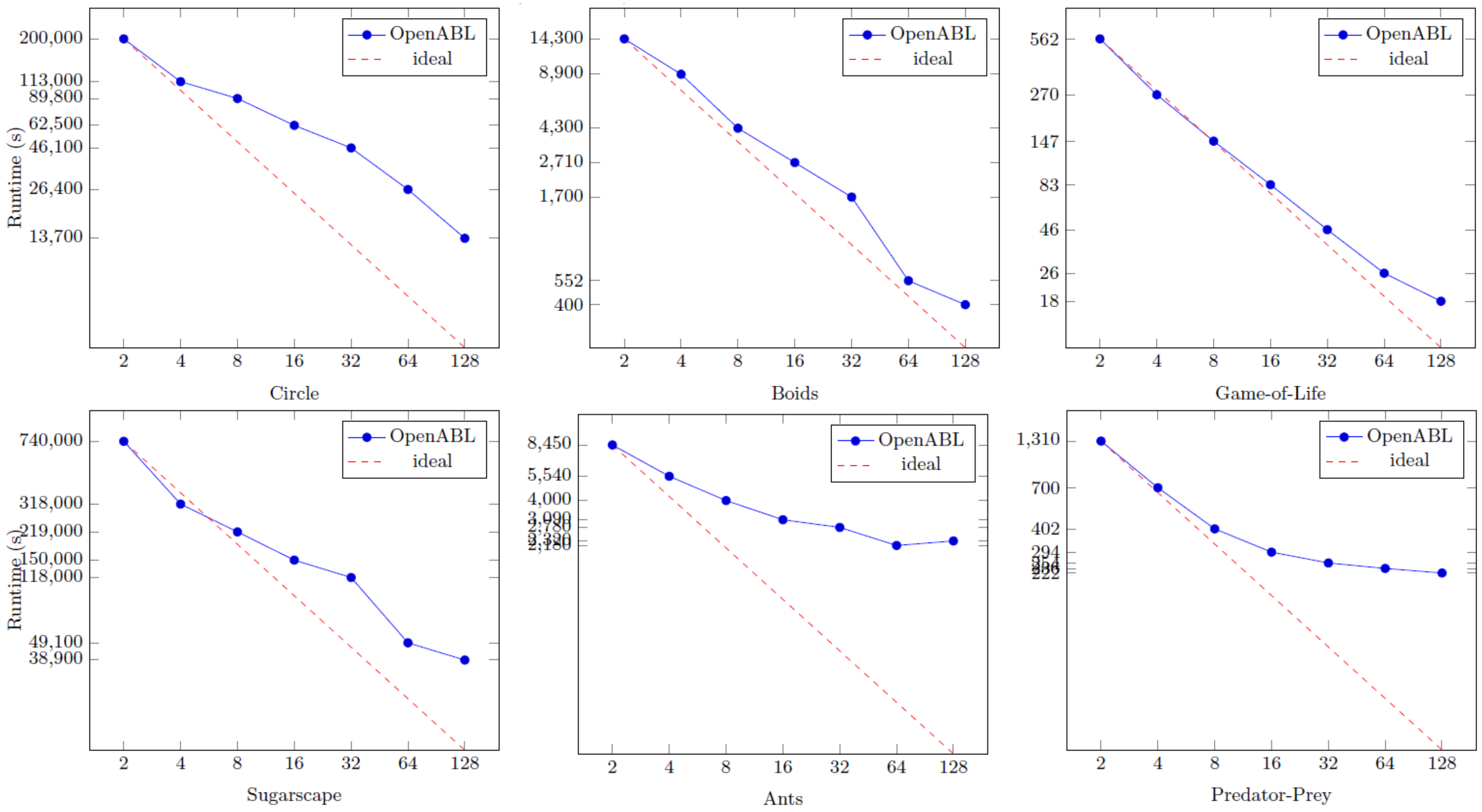
- Single-node performance
- Cluster scaling
- Programmability evaluation
- Overhead analysis





Experimental Results

- Single-node performance
- Cluster scaling
- Programmability evaluation
- Overhead analysis



Experimental Results

- Single-node performance
- Cluster scaling
- Programmability evaluation
- Overhead analysis

OpenABL code is much shorter!
In eLOC (effective lines of code):

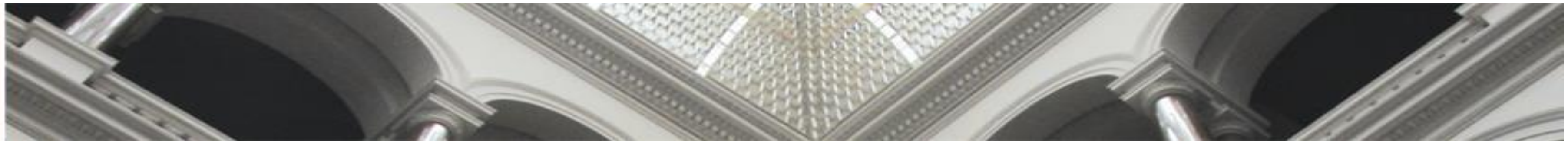
- From 2.2 up to 5.1 x more code on FlameGPU
- From 5.1 up to 14.9 x more code on D-Mason

For `boids`, against manually-tuned code:

- Mason 9% (because of double-buffering)
- Flame n.a. (Flame too slow to compare, (impractical with > 5000 agents)
- FlameGPU 0% (perfect programming model match)
- D-Mason 30% (because of double-buffering and additional synchronization)

Conclusion

- OpenABL: a new domain-specific language designed for agent modeling
  - ➢ high-level abstractions for programmability
  - ➢ explicitly exploits agent parallelism to deliver high-performance
- A source-to-source compiler implementation
- Backends targeting high-performance parallel and distributed architectures
  - ➢ multi-core CPUs, massively parallel GPUs, large clusters and cloud systems
- Tested on a collection of six applications from various fields
- A program written in OpenABL is much smaller than one written for non-portable platform-specific libraries, and its performance is very close to manual implementations



Thanks for your attention!

OpenABL: A Domain-Specific Language for Parallel and Distributed Agent-Based Simulations

Biagio Cosenza[1], Nikita Popov[1], Ben Juurlink[1], Paul Richmond[2], Mozhgan Kabiri Chimeh[2], Carmine Spagnuolo[3], Gennaro Cordasco[3], Vittorio Scarano[3]

[1]Technische Universität Berlin
Germany

[2]The University of Sheffield
United Kingdom

[3]University of Salerno
Italy

Project source code: https://github.com/OpenABL
Evaluated artifact: https://figshare.com/s/3ef16d36a5896000b85a
Paper preprint: http://biagiocosenza.com/papers/CosenzaEUROPAR18.pdf

Acknowledgments: DFG project CELERITY CO 1544/1-1 and EPSRC fellowship EP/N018869/1