Assessing the feasibility of OpenCL CPU implementations for agent-based simulations

Nuno Fachada & Agostinho C. Rosa

LaSEEB Evolutionary Systems Biomedical Engineering

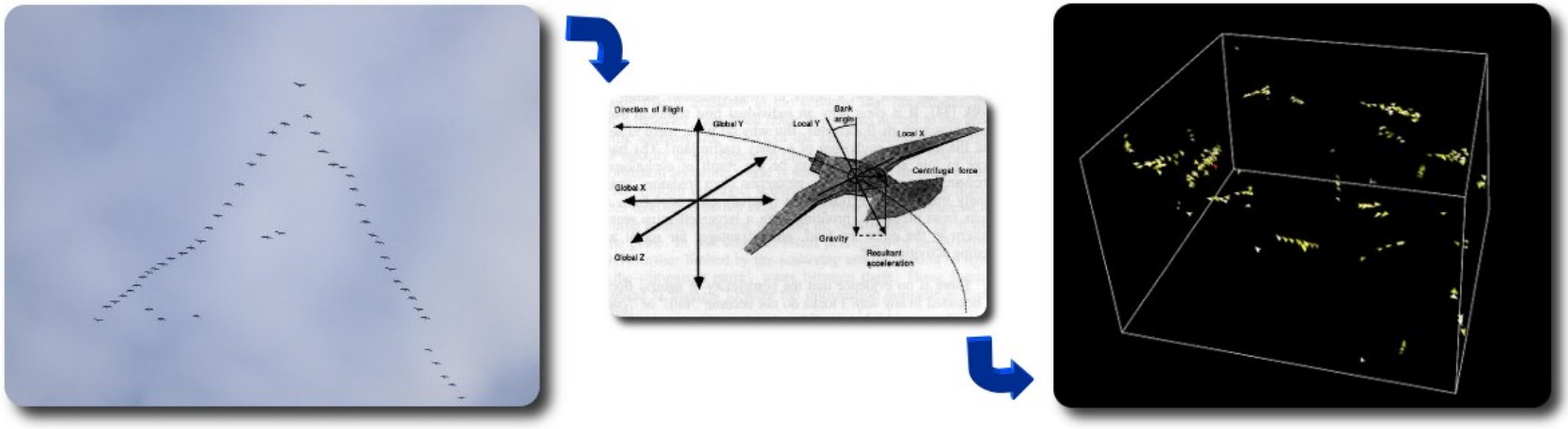
Institute for Systems and Robotics Lisboa

TÉCNICO LISBOA

U LISBOA

What is Agent-based modeling ABM?

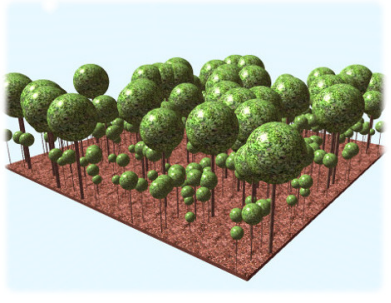
- Bottom-up modeling approach
- Model individual heterogeneous entities
- Entities (agents) make independent decisions
- System behavior emerges from local decisions and interactions



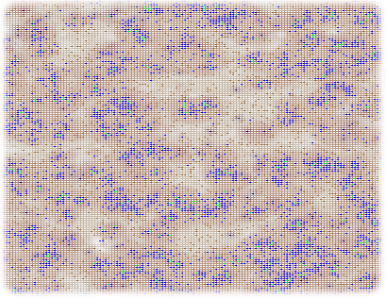
ABM is well-suited for...

Complex systems with many heterogeneous entities

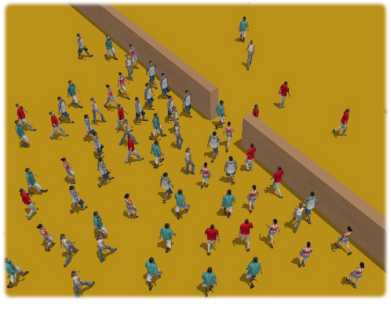
Battlefield



Ecology



Epidemiology



Crowd dynamics

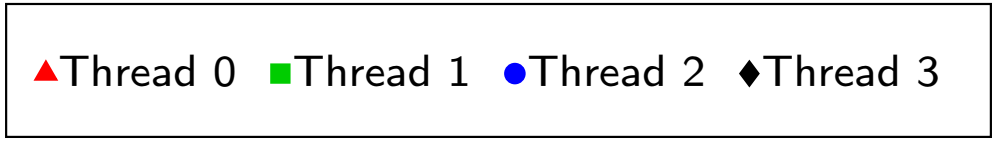
Traffic

Computer games

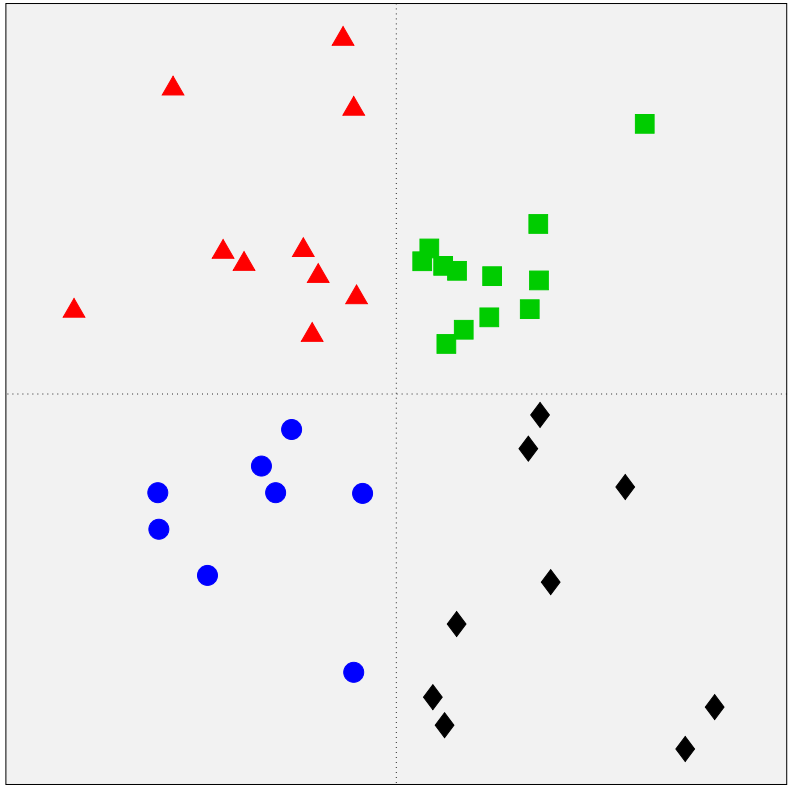
Problems with ABM
and with popular ABM frameworks

- Faithful simulations may require many agents
- Can be very slow
- Solution: parallelization
  - e.g. chip-based parallelism

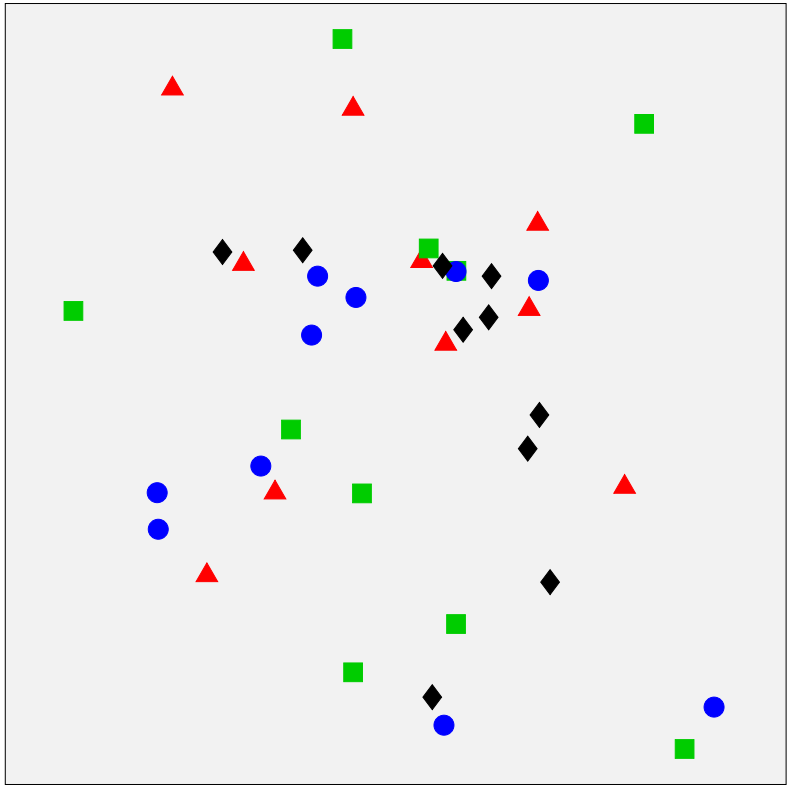
Generic parallelization approaches



Environment-parallel (EP)



Agent-parallel (AP)



In the literature

- CPU
  - Java
  - OpenMP
- GPU
  - CUDA (mainly)
  - OpenCL
- GPU+CPU
  - OpenCL

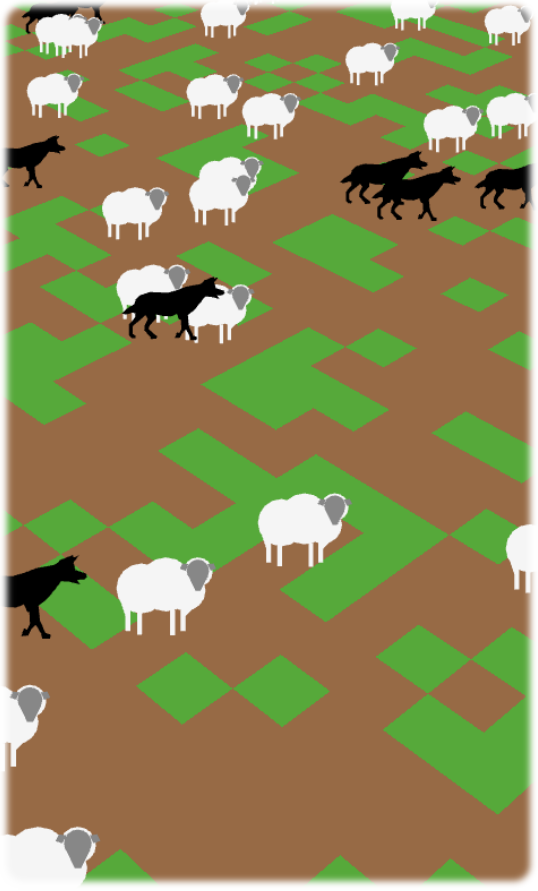
CPU-optimized OpenCL?

- Focus of current study
- Uncommon at best
- Does it offer cost-effective performance gains?

Predator-Prey for High Performance Computing

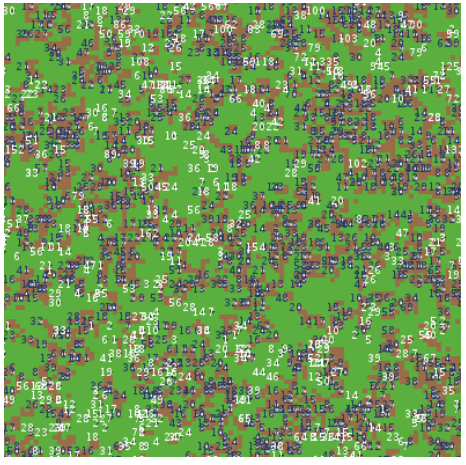
A reference research ABM

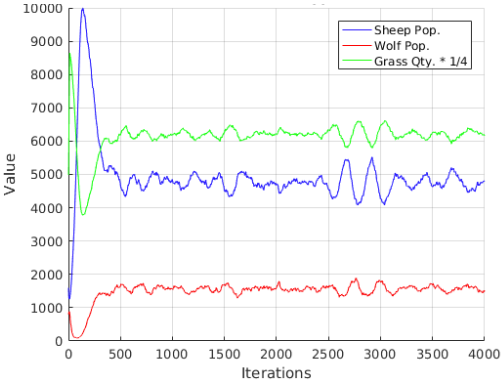
- Reference agent-based model
- Stochastic, well-studied dynamics
- Based on classic predator-prey



Basic rules

- 2D grid
  - 100×100
  - 200×200
  - ...
- Wolves eat sheep
- Sheep eat grass
- Agents reproduce
- Grass regrows





Main algorithm

Processes per time step

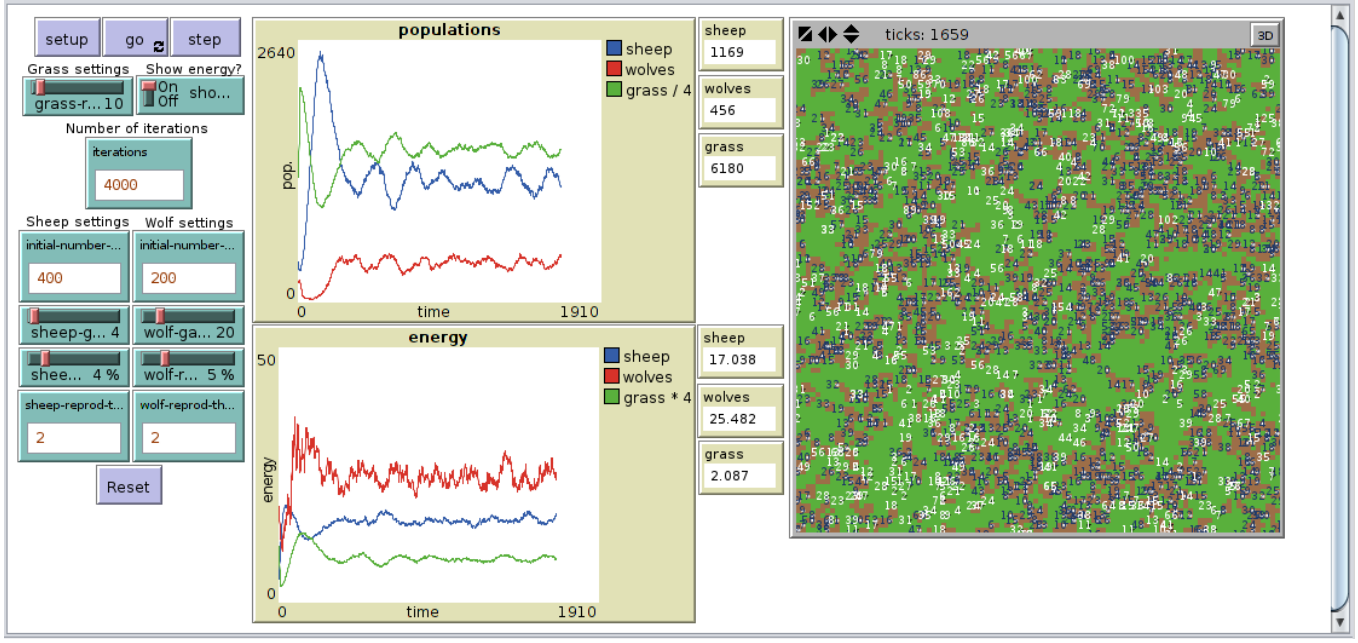
```
 1: for all agent do                    ▷ Any order
 2:     Move()
 3: end for
 4: for all grid cell do                ▷ Any order
 5:     GrowFood()
 6: end for
 7: for all agent do                    ▷ Random order
 8:     Act()
 9: end for
10: GetStats()
```

Implementations

- Reference NetLogo implementation
- Parallel Java implementation
  - Up to 40× faster on 6-core HT CPU

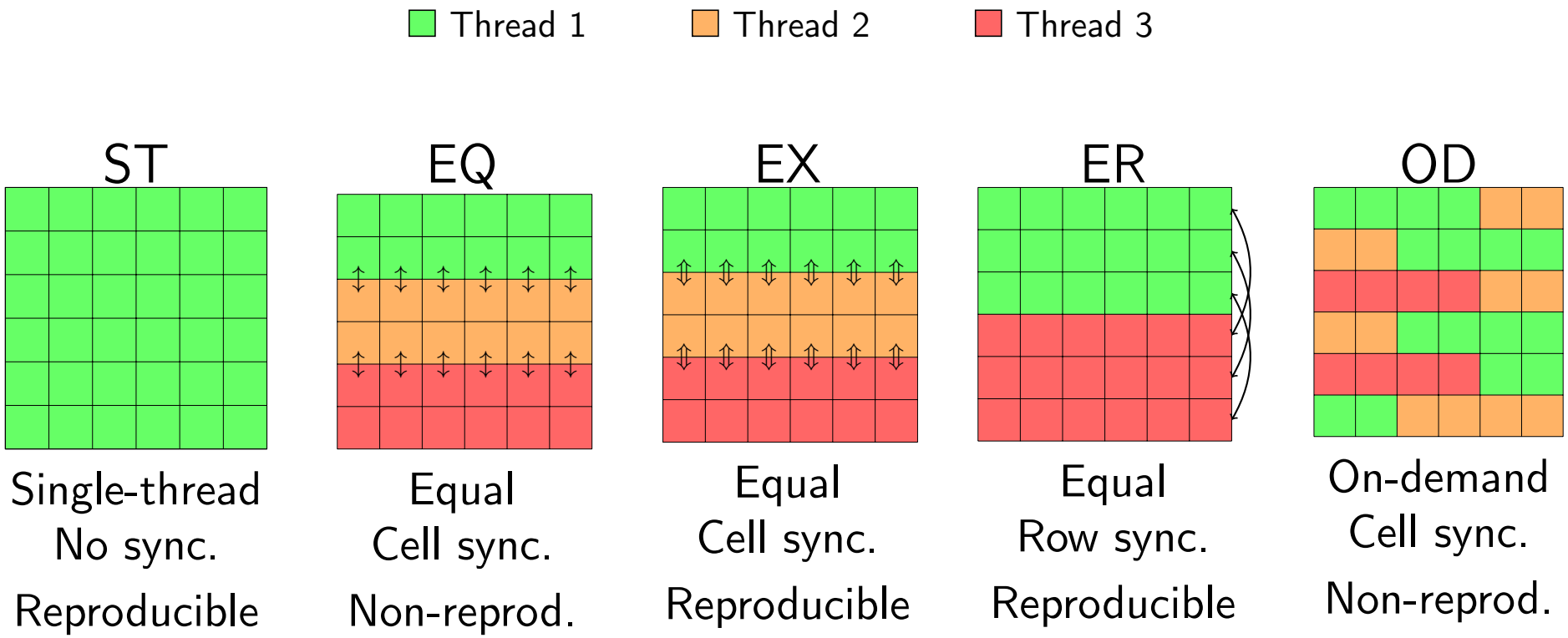
NetLogo implementation

- User-friendly GUI for modelers
- Benchmarked in CLI mode



Java implementation

Six EP parallelization strategies

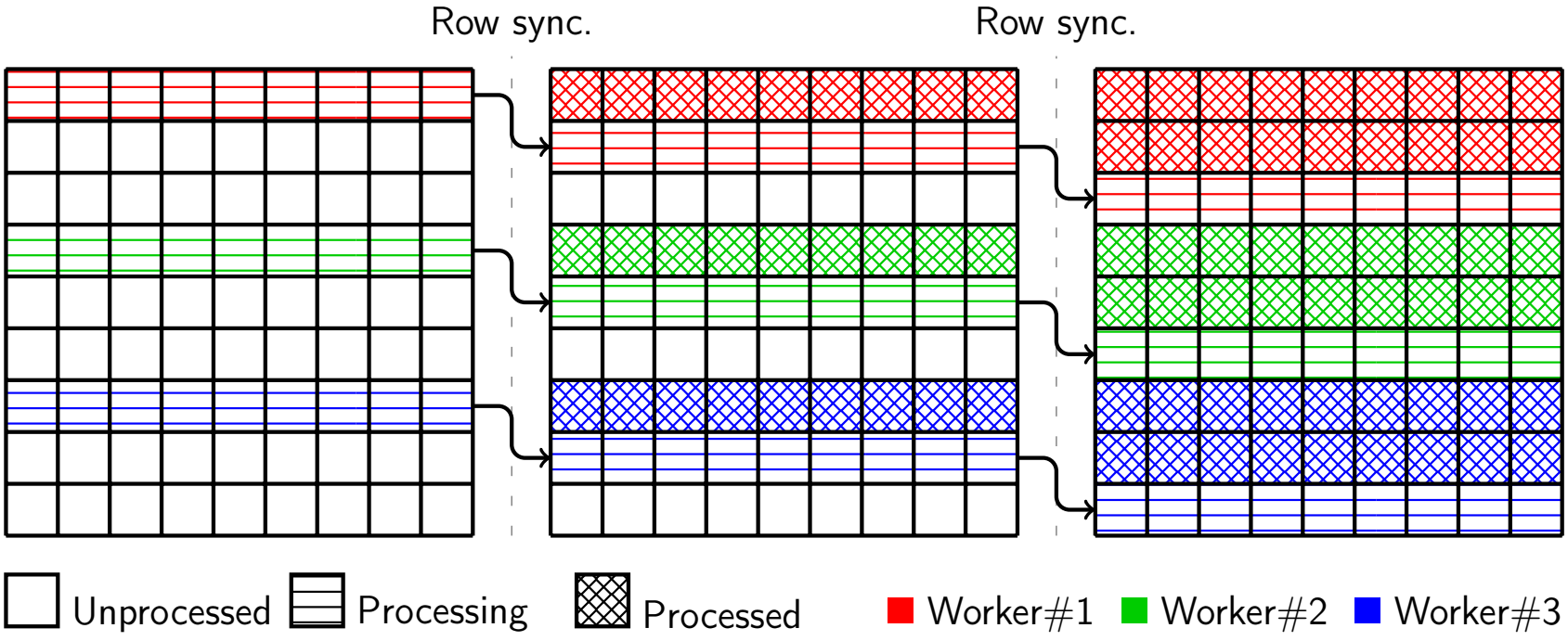


OpenCL CPU
Basics

- C99 host code with helper libraries:
  - Glib
  - cf4ocl
  - cl_ops
- OpenCL 1.2 kernels, runs on:
  - Intel OpenCL (Windows/Linux)
  - AMD APP SDK (Windows/Linux)
  - Apple OpenCL

OpenCL CPU
Parallelization approach

- Environment parallel (EP) approach
- ER strategy (row synchronization)



ER strategy – row synchronization

- No synchronization for agent movement
- Allows numerically reproducible simulations

Main algorithm

Modeler's perspective

Processes per time step

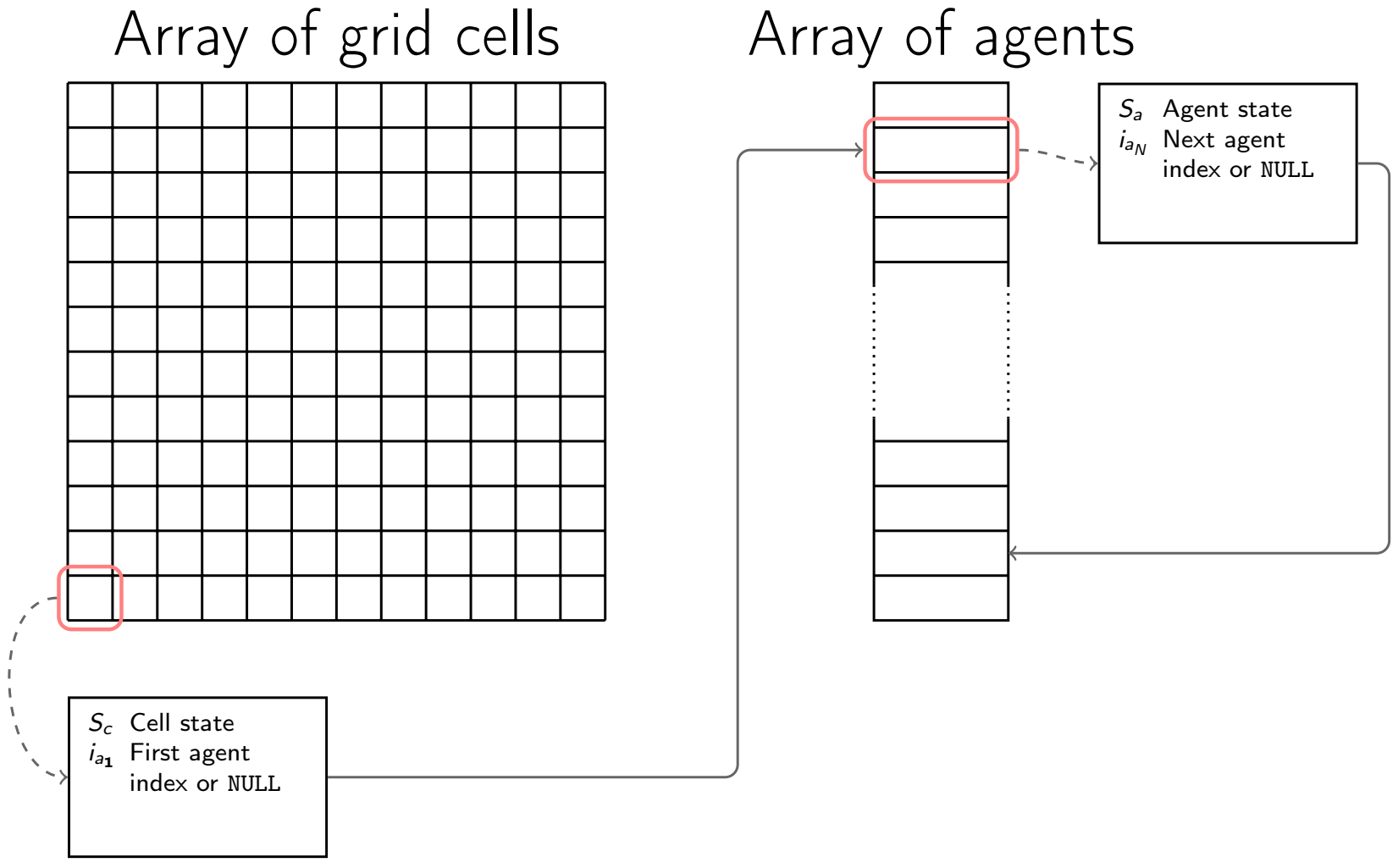
1: $j \leftarrow 0$
2: **for** $j <$ rows per thread **do**
3: Move() + GrowFood() ▷ Environment-parallel
4: $j \leftarrow j + 1$
5: **end for**
6: $j \leftarrow 0$
7: **for** $j <$ rows per thread **do**
8: Act() + GatherStats() ▷ Environment-parallel
9: $j \leftarrow j + 1$
10: **end for**

Main algorithm
OpenCL developer's perspective

Processes per time step

```
1: j ← 0
2: for j < rows per thread do
3:     step1()                          ▷ OpenCL kernel
4:     j ← j + 1
5: end for
6: j ← 0
7: for j < rows per thread do
8:     step2()                          ▷ OpenCL kernel
9:     j ← j + 1
10: end for
```

Data structures



Agent allocation

- Performed in `step2()` kernel
- Atomic compare-exchange on random agent array location
  - If location empty, then allocation successful
  - Otherwise, try again (until max. times)
- Allocation not deterministic...
- ...but irrelevant for simulation reproducibility

Model replication

- Replicating an ABM is difficult
- Parallelization
  - Harder kind of replication
  - Easy to introduce undesired biases
- Crucial: verify if parallelized model is statistically equivalent to serial model

How to test?

- Multiple runs of each model implementation
- Define output summary measures
  - Averages, extremes values, principal components (PCs)
- Apply statistical tests to summary measures
  - H_0: outputs from same distribution
  - H_1: outputs from different distributions
- Multiple parameterizations: increased confidence in replication

Experimental setup
Hardware and software

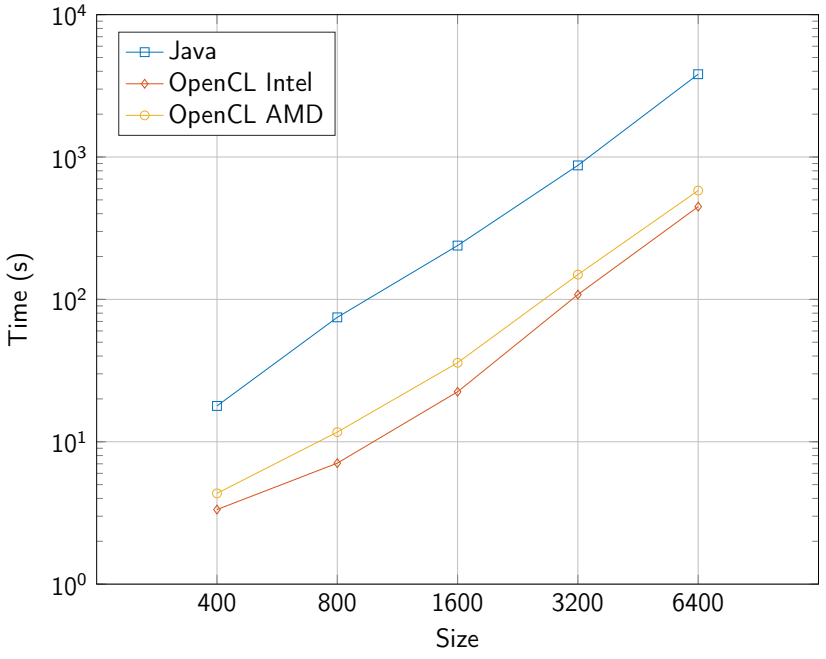
- Hardware
  - Intel Xeon CPU E5-2650 v3 @ 2.30GHz (10 cores, HyperThreading)
  - 64GB RAM
- Software
  - Ubuntu 16.04.1 LTS
  - OpenJDK Java 1.8.0
  - OpenCL:
    - Intel OpenCL CPU Runtime 16.1.1
    - AMD APP SDK 3.0

Experimental setup
Parameterizations

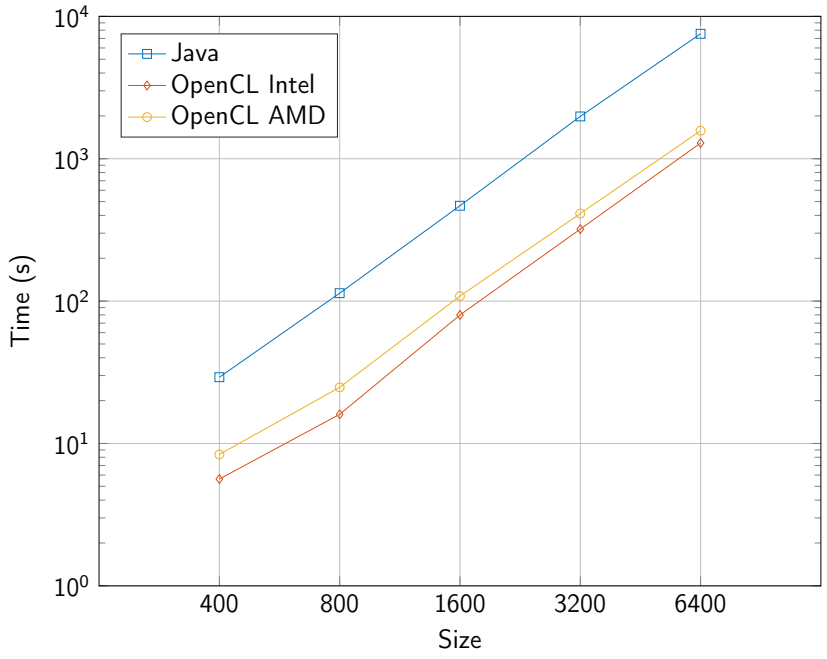
- Implementations:
  - Java: 20 threads
  - OpenCL: 20 work-items/work-groups
- Parameter sets:
  - 1
  - 2 (+agents)
- Model sizes:
  - 400×400
  - ...
  - 6400×6400

Scalability

Parameter set 1

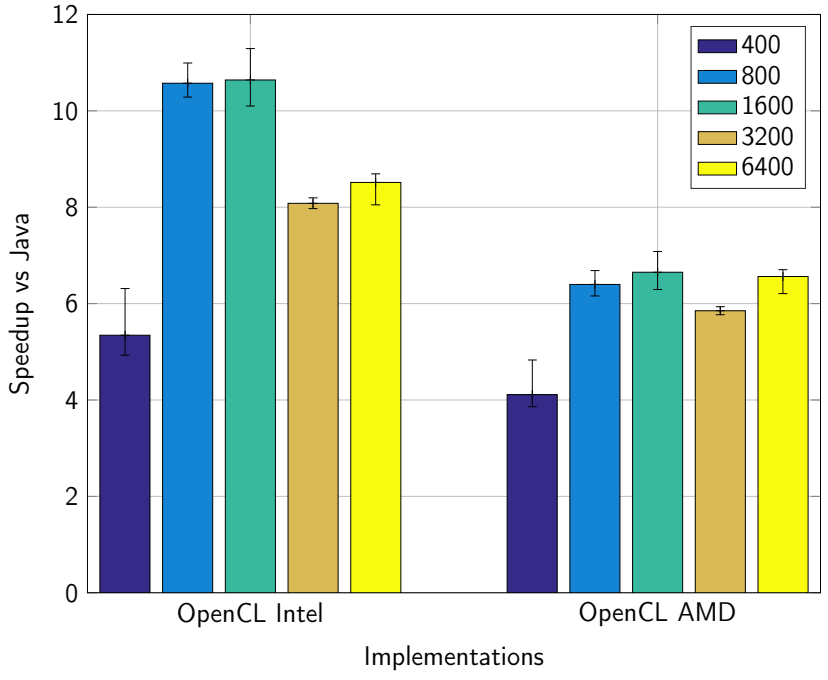


Parameter set 2 (+agents)

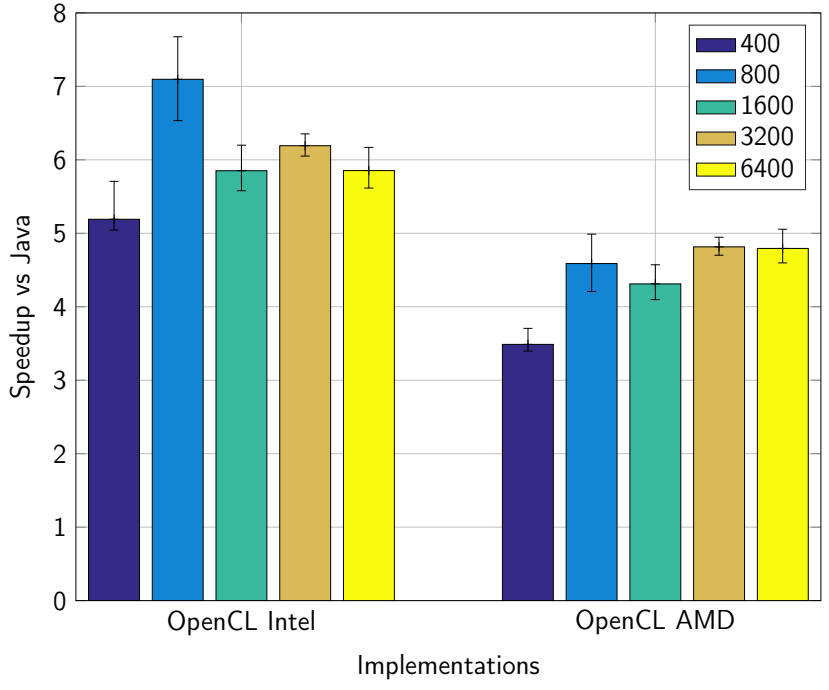


Speedup OpenCL vs Java

Parameter set 1



Parameter set 2 (+agents)



Statistical comparison

Size/set	Outputs					
	P^s	P^w	P^c	$\overline{E}^s$	$\overline{E}^w$	$\overline{C}$
400/1	0.062	0.285	0.029	0.241	0.446	0.030
800/1	0.863	0.482	0.865	0.047	0.494	0.864
1600/1	0.532	0.816	0.759	0.332	0.382	0.768
3200/1	0.125	0.212	0.174	0.119	0.189	0.171
6400/1	0.189	0.117	0.218	0.015	0.452	0.218
400/2	0.557	0.639	0.717	0.735	0.535	0.721
800/2	0.522	0.560	0.558	0.289	0.724	0.559
1600/2	0.623	0.822	0.787	0.297	0.655	0.786
3200/2	0.153	0.567	0.715	0.830	0.654	0.715
6400/2	0.996	0.989	0.990	0.997	0.882	0.990

P-values for the MANOVA test on PCs explaining 90% of variance

H_0: Outputs from Java and OpenCL drawn from same distribution

Conclusions

- CPU-optimized OpenCL is feasible for ABM
- Reproducible simulations possible
- Statistically similar results to Java...
- ...but much faster

Ongoing research

- Agent-parallel (AP) OpenCL GPU implementation
- Similar performance to OpenCL CPU implementation
  - Problem: expensive agent sorting step
- Outputs statistically different
  - Hypothesis: PRNG stream partitioning

Future work

- CPU
  - OpenCL 2.0+
  - Nested parallelism
- GPU
  - Fix incorrect behavior
  - Improve performance
  - Run on Intel and Mali GPUs

Thank you!

Full paper	`https://doi.org/10.1145/3078155.3078174`
Source code	`https://github.com/fakenmc/pphpc`
Data analysis	`https://zenodo.org/record/293014`