Towards Extreme-Scale Agent-Based Simulation with BioDynaMo

Lukas Breitwieser, ACAT 2022
Introduction to Agent-Based Simulation
Modeling complex systems – e.g. a swarm of birds
The agent-based model

- **Agent**: bird
  - position
  - velocity
  - shape

- **Behaviors**:

Agent-based simulation

Source: [https://github.com/tescande/boids.git](https://github.com/tescande/boids.git) by Thierry Escande
Agent-based simulation is very versatile

Source: Macal and North (2014), https://doi.org/10.1109/WSC.2014.7019874
Rising Number of Publications in this field

Keywords used: "agent-based modeling" or "agent-based simulation"

Source: https://app.dimensions.ai

Exported: October 14, 2022

Criteria: "agent-based model" OR "agent-based simulation" in full data.

© 2022 Digital Science and Research Solutions Inc. All rights reserved. Non-commercial redistribution / external re-use of this work is permitted subject to appropriate acknowledgement. This work is sourced from Dimensions® at www.dimensions.ai.

Source: https://app.dimensions.ai
Performance considerations
The problem

Existing simulation platforms do not always take full advantage of modern hardware.
Impact of low performance

- Limitation of the size and complexity of models
- Longer development time
- Limited capability to explore parameter space → less optimal solution
- Increased cost
Our solution: BioDynaMo

BioDynaMo is a modular and high-performance agent-based simulation platform written in C++.

https://biodynamo.org

Developed by the BioDynaMo collaboration:

and other universities:
CERN Knowledge Transfer

FROM CERN TECHNOLOGY TO SOCIETY

Source: https://kt.cern
## Features and abstraction layers

### Simulation

- **Agent geometry**: sphere, cylinder
- **Agents**: Cell, NeuronSoma, NeuriteElement
- **Behaviors**: Secretion, Chemotaxis, Proliferation, GeneRegulation
- Extracellular diffusion
- Agent interaction force

### BioDynaMo's model building blocks

- Parameter management
- Parameter optimization
- Hierarchical model support
- Hybrid-modeling
- Space boundary conditions

### BioDynaMo's high-level features

- Web-based interface
- Backup & restore of simulations
- Quality assurance infrastructure

### BioDynaMo's low-level features

<table>
<thead>
<tr>
<th>Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
</tr>
</tbody>
</table>

### Hardware

<table>
<thead>
<tr>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux / MacOS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Multi-core) CPUs</td>
</tr>
<tr>
<td>GPU</td>
</tr>
</tbody>
</table>

Source: Breitwieser et al. 2021, [https://doi.org/10.1093/bioinformatics/btab649](https://doi.org/10.1093/bioinformatics/btab649)
Demo: Neuroscience use case

Source: https://www.youtube.com/watch?v=taWMFs5D5Pg
Demo: Neuroscience use case

Demo: Neuroscience use case

Source: https://www.youtube.com/watch?v=MA74wZbhQ7w
Performance Challenges and Improvements

Maximize parallelization

• Optimized algorithm to search for neighbors
• Parallelize the addition and removal of agents

Efficient thread synchronization during agent updates

Minimize data transfers and memory access latency

• NUMA-aware iteration
• Agent Sorting and Balancing
• Pool-based memory allocator

Avoid unnecessary work

• Pair-wise force calculation for static regions

Offload computation to the GPU
Challenge: Agent-based workload is memory-bound
Minimize Memory Access Latency
NUMA-aware iteration

ND 0
CPU
DRAM
ND 0
ResourceManager
ND 1
CPU
DRAM
ND 1

Hardware
Simulation
BioDynaMo
Agent sorting and balancing mechanism

A. Agents in 3x3 grid
B. Grid box indices
C. Morton order of 4x4 grid

D. Determine offsets

E. Determine Morton order

F. Partition

G. Sort and balance
BioDynaMo memory allocator

A  Layout

Memory Block

N-page aligned segment

unused memory  free memory elements  allocated element  list node pointing to next free element

NumaPoolAllocator*

increasing memory addresses

B  Deallocate

NumaPoolAllocator*

T

free_list_[thread_id].push(p)

F

allocator.numa_id == thread.numa_id

central_list_.push(p)
Performance Evaluation
Benchmark simulations
### Benchmark simulations characteristics

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Cell proliferation</th>
<th>Cell clustering</th>
<th>Epidemiology use case</th>
<th>Neuroscience use case</th>
<th>Oncology use case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create new agents during simulation</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Delete agents during simulation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Agents modify neighbors</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load imbalance</td>
<td></td>
<td></td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Agents move randomly</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simulation uses diffusion</td>
<td>✗</td>
<td></td>
<td></td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Simulation has static regions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of iterations</td>
<td>500</td>
<td>1000</td>
<td>1000</td>
<td>500</td>
<td>288</td>
</tr>
<tr>
<td>Number of agents (in millions)</td>
<td>12.6</td>
<td>2</td>
<td>10</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>Number of diffusion volumes</td>
<td>0</td>
<td>54m</td>
<td>0</td>
<td>65k</td>
<td>0</td>
</tr>
</tbody>
</table>
## TABLE II: Benchmark hardware

<table>
<thead>
<tr>
<th>System</th>
<th>Main memory</th>
<th>CPU</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>504 GB</td>
<td>Server with four Intel(R) Xeon(R) E7-8890 v3 CPUs @ 2.50GHz with a total of 72 physical cores, two threads per core and four NUMA nodes.</td>
<td>CentOS 7.9.2009</td>
</tr>
<tr>
<td>B</td>
<td>1008 GB</td>
<td>Server with two Intel(R) Xeon(R) E5-2683 v3 CPUs @ 2.00GHz with a total of 28 physical cores, two threads per core and two NUMA nodes.</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>62 GB</td>
<td></td>
<td>CentOS Stream 8</td>
</tr>
</tbody>
</table>
Runtime and Memory Complexity

![Runtime and Memory Complexity Graphs](image)

- **Runtime per Iteration [s]**
  - Y-axis: $10^{-3}$ to $10^1$
  - X-axis: $10^3$ to $10^9$

- **Memory consumption [GB]**
  - Y-axis: $10^0$ to $10^2$
  - X-axis: $10^3$ to $10^9$

Legend:
- **Cell prolif.**
- **Cell clustering**
- **Epidemiology**
- **Neuroscience**
- **Oncology**
Comparison with Biocellion

- Single-node 16 CPU cores; 13.4 million cells → BioDynaMo is **4.15x faster**

- Biocellion: 21 nodes, 672 CPU cores, 281 million cells
  BioDynaMo: one node, 72 CPU cores
  → same runtime, but **9.3x fewer CPU cores** used

Comparison with Cortex3D and NetLogo

- **Cortex3D**
  - Cell prolif.
  - Cell clustering
  - Neuroscience
  - Epidemiology

- **NetLogo**
  - HPSE standard implementation
  - Plus HPSE uniform grid
  - Plus memory improvements
  - Plus extra memory
  - Plus static agent detection

The graph shows the speedup for each category with different improvements.
Strong scaling

![Graph showing runtime vs. number of threads for different configurations. The x-axis represents the number of threads ranging from $10^0$ to $10^2$, and the y-axis represents runtime in milliseconds ranging from $10^4$ to $10^6$. The graph illustrates the impact of BDM standard implementation, BDM uniform grid, memory improvements, extra memory, and static agents detection on runtime performance. The first NUMA domain boundary is indicated on the graph.]
Publications about BioDynaMo

2021


2017

Publications using BioDynaMo

2022


2021


https://biodynamo.org/team/application_publications/
Summary

- Agent-based simulation can be used to model many complex systems
- BioDynaMo is up to three orders of magnitude faster than state-of-the-art tools.
- These improvements allow BioDynaMo simulating billions of agents on a single server.
- BioDynaMo is currently being used in:
  - neuroscience
  - oncology
  - epidemiology
  - cryobiology
  - socioeconomics
  - finance
  - ...
- BioDynaMo is open-source and we would be very happy to welcome new users and contributors.
Thank you for your attention!

Lukas.Breitwieser@cern.ch
Backup Slides
Distributed simulation engine
Agent-based simulation algorithm

```plaintext
ModelInitialization()
for i ∈ iterations do
  for op ∈ pre_standalone_operations do
    op();
  end
  wait()
  parallel for a ∈ agents do
    for op ∈ agent_operations do
      op(a);
    end
  end
  for op ∈ standalone_operations do
    op();
  end
  wait()
  for op ∈ post_standalone_operations do
    op();
  end
end
```
The process of developing an ABM

Important building blocks

A  Agents

- Cell
- Person

B  Behavior

- Move
- Divide
- Grow
- Substance secretion

C  Environment

- Observed agent
- Agents inside environment
- Agents outside environment

D  Simulation Algorithm

// Define initial model
Place simulation objects in space
Set their attributes
Define behavior

// Run simulation
for each simulation step
  Update environment
  for each agent
    for each agent operation
      Run agent_operation(agent)
    for each standalone operation
      Run standalone_operation()
Modular software design

Cell clustering model

- **Agent:** Cell
  - Spherical shape
  - Cell type

- **Behaviors**
  - Secrete a substance into the extracellular matrix
  - Follow the concentration gradient (chemotaxis)

- **Initial condition**
  - Randomly distributed in 3D space
Cell clustering result

Maximize parallelization
Optimized uniform grid to search for neighbors

Source: Ahmad Hesam
Parallel agent removal mechanism

Removed agents:

<table>
<thead>
<tr>
<th>idx</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>RessourceManager::agents_</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td>7</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

Thread 0: {2, 8}
Thread 1: {7}

1. Initialize
to_right

2. Fill
to_right

3. Reorder
to_right

4. Swap
to_right

5. Resize

Thread 0 to_right
Thread 1 to_left

#swaps

not_to_left

not_to_left

to_left

1. Initialize

2. Fill

3. Reorder

4. Swap

5. Resize
Optimize Thread-Synchronization
Thread-synchronization (TS) during agent-updates

- Only necessary if agents modify their local environment.
  - Two agents (updated by two different threads) could attempt to modify the same neighbor.

- BioDynaMo provides two TS mechanisms
  - Automatic TS
  - User-defined TS

---

**Algorithm 1: Agent-based simulation algorithm**

```plaintext
1 ModelInitialization()
2 for i ∈ iterations do
3     for op ∈ pre_standalone_operations do
4         op();
5     end
6     wait();
7     parallel for a ∈ agents do
8         for op ∈ agent_operations do
9             op(a);
10        end
11     end
12     for op ∈ standalone_operations do
13         op();
14     end
15     wait()
16     for op ∈ post_standalone_operations do
17         op();
18     end
19 end
```
Automatic thread-synchronization

determine box of current agent
get lock of current box and all neighboring boxes
sort locks
lock all locks
execute operations
unlock all locks
User-defined thread-synchronization

```cpp
void NeuronSoma::CriticalRegion(std::vector<AgentPointer>* aptrs) {
    aptrs->reserve(daughters_.size() + 1);
    aptrs->push_back(Agent::GetAgentPtr());
    for (auto& daughter : daughters_) {
        aptrs->push_back(daughter);
    }
}
```
BioDynaMo’s GPU capabilities

- Operations can have implementations for different compute targets (CPU, GPU, and FPGA).
- If an operation has multiple implementations, the scheduler decides which one to use.
- Currently, BioDynaMo provides a GPU operation to calculate mechanical forces between spheres.

\[
\begin{align*}
\delta &= r_1 + r_2 - \|p_1 - p_2\| \\
r &= r_1 \cdot r_2 \\
F &= \left(\kappa \cdot \delta - \gamma \cdot \sqrt{r \cdot \delta}\right) \cdot \frac{p_1 - p_2}{\|p_1 - p_2\|}
\end{align*}
\]

Collision force computation

Source: Hesam et al. 2021, [https://doi.org/10.48550/arXiv.2105.00039](https://doi.org/10.48550/arXiv.2105.00039)
(Ongoing) Use Cases
Oncology use case

Epidemiology use case

An *in silico* hybrid continuum-/agent-based procedure to modelling cancer development: Interrogating the interplay amongst glioma invasion, vascularity and necrosis

Jean de Montigny\textsuperscript{a}, Alexandros Iosif\textsuperscript{b}, Lukas Breitwieser\textsuperscript{c,d}, Marco Manca\textsuperscript{e}, Roman Bauer\textsuperscript{f,a}, Vasileios Vavourakis\textsuperscript{Sg,c}

---

Source: de Montigny et al. 2021, [https://doi.org/10.1016/j.ymeth.2020.01.006](https://doi.org/10.1016/j.ymeth.2020.01.006)
Retinal self-organization

Understand the mechanisms of cells self-organization during early development which is pivotal for their function.

Source: Jean de Montigny et al., 2021, https://doi.org/10.1101/2021.10.22.465398
Radiation-induced lung injury simulation

Simulate onset of radiation pneumonitis and/or lung fibrosis in normal tissue after exposition to thoracic irradiation.

Source: Nicolò Cogno et al., https://doi.org/10.3390/sym14010090
Spatial Spread of HIV in Malawi

• Collaboration with UniGE

• Original simulation written in R (Runtime: ~5.5h)

• Goal: speed up execution time

• Preliminary runtime with BioDynaMo: less than 2 minutes

• Further work needed to make models equivalent

The spatial spread of HIV in Malawi: An individual-based mathematical model

Janne Estill, Wingston Ng’ambi, Liudmila Rozanova, Olivia Keiser
doi: https://doi.org/10.1101/2020.12.23.20248757

Source: https://www.medrxiv.org/content/10.1101/2020.12.23.20248757v3
Evaluation
Table 6. Performance data. The values in column “Agents” and “Diffusion volumes” are taken from the end of the simulation. Runtime measures the wall-clock time to simulate the number of iterations. It excludes the time for simulation setup and visualization.

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Agents</th>
<th>Diffusion volumes</th>
<th>Iterations</th>
<th>System (Table 5)</th>
<th>Physical CPUs</th>
<th>Runtime</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neuroscience use case</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single (Figure 4A in the main manuscript)</td>
<td>1494</td>
<td>250</td>
<td>500</td>
<td>A</td>
<td>1</td>
<td>0.16 s</td>
<td>382 MB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D</td>
<td>1</td>
<td>0.12 s</td>
<td>479 MB</td>
</tr>
<tr>
<td>Large-scale (Figure 4C in the main manuscript)</td>
<td>9036986</td>
<td>65536</td>
<td>500</td>
<td>A</td>
<td>72</td>
<td>35 s</td>
<td>6.47 GB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D</td>
<td>2</td>
<td>11 min 28 s</td>
<td>5.37 GB</td>
</tr>
<tr>
<td>Very-large-scale</td>
<td>1018644154</td>
<td>5606442</td>
<td>500</td>
<td>B</td>
<td>72</td>
<td>1 h 24 min</td>
<td>438 GB</td>
</tr>
<tr>
<td>Oncology use case (Figure 5 in the main manuscript)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000 initial cells</td>
<td>4177</td>
<td>0</td>
<td>312</td>
<td>A</td>
<td>1</td>
<td>1.05 s</td>
<td>382 MB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D</td>
<td>1</td>
<td>0.832 s</td>
<td>480 MB</td>
</tr>
<tr>
<td>4000 initial cells</td>
<td>5341</td>
<td>0</td>
<td>312</td>
<td>A</td>
<td>1</td>
<td>1.76 s</td>
<td>382 MB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D</td>
<td>1</td>
<td>1.34 s</td>
<td>480 MB</td>
</tr>
<tr>
<td>8000 initial cells</td>
<td>7861</td>
<td>0</td>
<td>288</td>
<td>A</td>
<td>1</td>
<td>3.27 s</td>
<td>384 MB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D</td>
<td>1</td>
<td>2.60 s</td>
<td>482 MB</td>
</tr>
<tr>
<td>Large-scale</td>
<td>10003925</td>
<td>0</td>
<td>288</td>
<td>A</td>
<td>72</td>
<td>1 min 42 s</td>
<td>7.42 GB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D</td>
<td>2</td>
<td>43 min 56 s</td>
<td>5.84 GB</td>
</tr>
<tr>
<td>Very-large-scale</td>
<td>986054868</td>
<td>0</td>
<td>288</td>
<td>B</td>
<td>72</td>
<td>6 h 21 min</td>
<td>604 GB</td>
</tr>
<tr>
<td>Epidemiology use case (Figure 6C in the main manuscript)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Measles</td>
<td>2010</td>
<td>0</td>
<td>1000</td>
<td>A</td>
<td>1</td>
<td>0.53 s</td>
<td>381 MB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D</td>
<td>1</td>
<td>0.42 s</td>
<td>479 MB</td>
</tr>
<tr>
<td>Seasonal Influenza</td>
<td>20200</td>
<td>0</td>
<td>2500</td>
<td>A</td>
<td>1</td>
<td>16.41 s</td>
<td>383 MB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D</td>
<td>1</td>
<td>16.40 s</td>
<td>479 GB</td>
</tr>
<tr>
<td>Medium-scale (measles)</td>
<td>100500</td>
<td>0</td>
<td>1000</td>
<td>A</td>
<td>72</td>
<td>1.36 s</td>
<td>1 GB</td>
</tr>
<tr>
<td>Large-scale (measles)</td>
<td>10050000</td>
<td>0</td>
<td>1000</td>
<td>A</td>
<td>72</td>
<td>59.19 s</td>
<td>5.87 GB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D</td>
<td>2</td>
<td>19 min 18 s</td>
<td>5.41 GB</td>
</tr>
<tr>
<td>Very-large-scale (measles)</td>
<td>1005000000</td>
<td>0</td>
<td>1000</td>
<td>B</td>
<td>72</td>
<td>2 h 0 min</td>
<td>495 GB</td>
</tr>
<tr>
<td>Soma clustering (Figure 2)</td>
<td>32000</td>
<td>1240000</td>
<td>6000</td>
<td>A</td>
<td>72</td>
<td>12.91 s</td>
<td>1.02 GB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D</td>
<td>2</td>
<td>2 min 7 s</td>
<td>522 MB</td>
</tr>
</tbody>
</table>

Source: Breitwieser et al. 2021, [https://doi.org/10.1093/bioinformatics/btab649](https://doi.org/10.1093/bioinformatics/btab649)
Agent sorting and balancing

[Graph showing speedup compared to no agent sorting across different agent sorting frequencies for various fields: Cell prolif., Cell clustering, Neuroscience, Oncology, and Epidemiology.]
Operation breakdown
Optimization overview
Environment algorithm comparison

(a) Whole simulation

(b) Build time

(c) Search time (indirect)

(d) Memory consumption
Memory allocator comparison