Instructions to Reproduce the Results in the Paper: High-Performance and Scalable Agent-Based Simulation with BioDynaMo

Lukas Breitwieser Ahmad Hesam Fons Rademakers
Juan Gómez Luna Onur Mutlu

Contents

1 Introduction

This document describes how to reproduce the results in the paper: **High-Performance and Scalable Agent-Based Simulation with BioDynaMo** available at https://doi.org/10.1145/3572848.3577480.

All supplementary files are hosted on Zenodo (https://doi.org/10.5281/zenodo.6463816). The repository consists of additional evaluations (`SF0-additional-evaluations.pdf`), this document (`SF1-readme.pdf`), all code (`SF2-code.tar.gz`), a self-contained Docker image (`SF3-bdm-publication-image.tar.gz`), and the obtained raw benchmark results (`SF4-raw-results.tar.gz`).

We executed all simulations and benchmarks in the paper using the provided Docker image. The table below shows an overview of our benchmark hardware.

System	Type	Description
System A	CPU	4x Intel(R) Xeon(R) CPU E7-8890 v3 @ 2.50GHz
	Physical CPU cores	72
	Threads per core	2
	NUMA nodes	4
	Memory	504 GB
	Host operating system	CentOS Linux release 7.9.2009 (Core)
	Kernel	3.10.0-1160.53.1
	Docker version	19.03.13
System B	CPU	4x Intel(R) Xeon(R) CPU E7-8890 v3 @ 2.50GHz
	Physical CPU cores	72
	Threads per core	2
	NUMA nodes	4
	Memory	1008 GB
	Host operating system	CentOS Linux release 7.9.2009 (Core)
	Kernel	3.10.0-1160.59.1
	Docker version	19.03.5
System C	CPU	2x Intel(R) Xeon(R) E5-2683 v3 CPUs @ 2.00GHz
	Physical CPU cores	28
	Threads per core	2
	NUMA nodes	2
	Memory	62 GB
	Host operating system	CentOS Stream 8
	Kernel	4.18.0-365
	Docker version	20.10.7

2 Getting started

This section guides you through the necessary steps to setup your machine. Please follow these steps before starting to reproduce the results.

2.1 Extract code repositories

Start by downloading the file `SF2-code.tar.gz`. Extract the archive into an empty directory and change into this directory using the following commands. Make sure that the absolute path to this directory contains no spaces.

```
mkdir reproduce
tar -xzf <path-to>/SF2-code.tar.gz -C reproduce
cd reproduce
```

Now, the folder `reproduce` contains all the necessary code to produce all results shown in the paper. The code consists of the `biodynamo`, and `bdm-paper-examples` repositories. BioDynaMo is the high-performance simulation engine shown in the paper, and the directory `bdm-paper-examples` contains the simulations, benchmarks, plots, and utilities to generate the results.

License information is available in the following files:

- `biodynamo/LICENSE`
- `bdm-paper-examples/LICENSE`

Directory `metadata` contains more detailed information about the hardware and software configuration of the used systems.

- System A: `metadata/system-a`
- System B: `metadata/system-b`
- System C: `metadata/system-c`

2.2 Install host machine prerequisites

2.2.1 Docker

Please install a recent Docker version (>= 19.0.3) and follow the instructions to use docker as a non-root user.

2.2.2 Intel Vtune Sampling Driver

To collect performance results using Intel Vtune, install the Intel Vtune sampling driver (version 2022.2.0) on the host machine by following this guide.

This step is necessary to perform the microarchitecture analysis.

2.2.3 Linux `screen` command

We use the Linux `screen` command to execute long-running commands over SSH connections. With `screen`, processes are not terminated if we close the SSH session. If this command is

not available on the system, please install it:

```
# on Ubuntu
sudo apt install -y screen
# on CentOS
sudo yum install -y screen
```

Here are the most important `screen` commands:

```
# Start a new screen
screen

# Disconnect from the screen by pressing CTRL + a, d
# That means: press 'CTRL' and 'a' at the same time,
# release both, and then type 'd'

# Reconnect
screen -r
```

Have a look at this tutorial for more details on using `screen`.

2.3 Load Docker image

Download the publication's docker image (`SF3-bdm-publication-image.tar.gz`) if you haven't done so already. After finishing this step, load the docker image using the following commands. This step takes around 5 min to complete. The extracted image requires ~30GB of disk space.

```
cd bdm-paper-examples
docker/load.sh <path-to>/SF3-bdm-publication-image.tar.gz
```

If this step completes successfully, you should be able to see an entry for `bdm-publication-image-v7` with ID `9e13327e93a5` in the output of the command `docker images`. A sample output is shown below.

```
REPOSITORY                TAG       IMAGE ID       ...  SIZE
...
bdm-publication-image-v7  latest    9e13327e93a5   ...  28.5GB
...
```

2.4 Verify successful setup

To verify that all previous steps were successful, we provide a script that compiles and executes all simulations shown in Table 1 in the paper with ~1000 agents.

- Executed on System A in 15 min
- Required memory: 12 GB
- Required disk space: 1 GB

The following instruction assumes that the current working directory is `reproduce/bdm-paper-examples`.

```
# Create a new Docker container and run the script
docker/run.sh ./run-functional-evaluation.sh
```

If the script returns exit code 0, it means that the checks were successful and that we can start running the scripts to reproduce all results in the paper.

3 Documentation

This section contains more details about the `bdm-paper-examples` code repository, the Docker image, important information for reproducing the results, and an explanation of the result directory structure.

3.1 The `bdm-paper-examples` repository

The following list explains the directory structure.

- `bdm`: This directory contains the five simulations shown in Table 1 in the paper. These simulations are detailed in https://doi.org/10.1093/bioinformatics/btab649.
- `benchmark`: This directory contains the template scripts that define the compile and runtime parameters used for a specific study.
- `docker`: This directory contains scripts to build, run, load, save, and connect to a Docker container. It also includes the definition of the Docker image.
- `netlogo`: This directory contains the epidemiology implementation for NetLogo and scripts to execute it.
- `other-tools/cx3d`: This directory includes Cortex3D and its simulation implementations.
- `plot`: This directory contains all matplotlib scripts to combine the raw results into figures for the paper.
- `util`: This directory contains various scripts needed to execute the benchmarks.
- `visualization`: Contains ParaView scripts to visualize the simulations. We did not use these scripts for this publication.
- `run-*.sh`: These scripts execute one or more benchmarks for each simulation and gather metadata about the software and hardware configuration of the underlying machine.

3.2 Mapping between simulation names in the paper and `bdm-paper-examples`

Simulation name in the paper	Name in `bdm-paper-examples`
Cell proliferation	`cell-grow-divide`
Cell clustering	`soma-clustering`
Epidemiology use case	`epidemiology`
Neuroscience use case	`pyramidal-cell`
Oncology use case	`tumor-spheroid`

3.3 Docker image

The table below shows the version of selected software products.

Software	Description
BioDynaMo	v1.03.56-8762fa70
C++ compiler	g++ (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0
Python	3.9.1
CMake	3.19.5
Java	openjdk version "11.0.10" 2021-01-19
Maven	Apache Maven 3.6.3
Cortex3D	0.03 with modifications We used maven as build system, added support for headless execution, and removed calls to `Thread.sleep`.
NetLogo	6.2.0
Intel oneAPI	2022.0.0 (build 621730)

The Dockerfile that underlies the provided image (`SF3-bdm-publication-image.tar.gz`) can be found at `bdm-paper-examples/docker/Dockerfile`

3.4 Important information

- Results are always generated in the directory `bdm-paper-examples/result`. Make sure this directory is empty before starting a script.
- The resulting plots can be found in the directory `bdm-paper-examples/result/evaluation`.
- Please ensure that the system you are using has enough memory and disk space to run the simulations.
- The information regarding required memory and disk space given below only includes the execution of the script.
- Execution time on your system might differ significantly, depending on your hardware.
- No manual changes are necessary if the number of CPUs differs from our benchmark hardware. Our scripts make the necessary adjustments automatically.
- Use a local filesystem to avoid authentication issues with network file systems.
- Do not run the scripts as `root` user, and do not use `sudo`.
- Only run one script at the same time.
- If you are facing issues that you cannot resolve, don't hesitate to get in touch with lukas.breitwieser@gmail.com for help.

3.5 Result directory structure

As mentioned above, all results will be generated in the directory `bdm-paper-examples/result`. Please find a description of its directory structure below.

- `evaluation`: This directory contains plots generated from the individual simulation results. The plots are accompanied by a `csv` file containing the plot's raw numbers.
- `reproduce`: Contains information on how to reproduce the generated results.
  - Exact source code used: `bdm-paper-examples` and `biodynamo` repositories
  - Additional details in `metadata/run-*` about used memory, disk space, docker container id, and more.
- `metadata`:
  - `log`: Complete log of the whole execution.
  - `system-info`: Metadata about the hardware and software configuration used.
  - `benchmark`: Subdirectories contain log files of individual benchmark runs (e.g., load balancing study for the epidemiology use case)
- `tmp`: Contains large temporary files (e.g., the raw results from Intel Vtune and advisor).
- Simulation result directories (`cell-grow-divide`, `epidemiology`, `soma-clustering`, `pyramidal-cell`, `tumor-spheroid`) contain the measurements for each benchmark for each simulation. For example, the `epidemiology/operation-breakdown` directory contains the runtime measurement for each operation.
- `run-*.tar.gz`: Archive of the whole result directory excluding the `tmp` folder.
- `run-*.tar.gz.sha256`: Sha256 hash code of the result archive to detect modification.

3.6 Long execution time

The execution time of our scripts lies between one hour and 13 days due to three main reasons.

- First, we provide a rigorous performance analysis of the BioDynaMo simulation engine, which results in a large number of benchmarks and analyses.
- Second, due to our significant speedups of up to three orders of magnitude, the majority of time is spent executing the slow baselines.
- Third, due to BioDynaMo's focus on large-scale simulations, we execute some benchmarks with one billion agents or more.

Reducing the execution time by reducing the simulation size (i.e., using smaller inputs) is not possible since this would not provide necessary evidence for our claims in the paper.

4 Reproducing results

The following instructions assume that the current working directory is `reproduce/bdm-paper-examples`.

4.1 Main results (Figure 5 (left) and 9–12 in the paper)

This script executes the majority of benchmarks described in the evaluation section of the paper.

- Executed on System A in 13d
- Required memory: 13 GB
- Required disk space: 10 GB
- Results in directory `0-main` in archive `SF4-raw-results.tar.gz`

```
# Move the previous result
mv result ../result-<choose-name>
# Create a new screen
screen
# Create a new Docker container and run the script
docker/run.sh ./run-main.sh
# Disconnect from the screen by pressing CTRL + a, d
```

Mapping between the output of the script and the paper:

Figure in the paper	Result file (inside `result/evaluation`)
Figure 5 (left)	`operation-breakdown.svg`
Figure 9 (top)	`optimization-overview/all.svg`
Figure 9 (bottom)	`optimization-overview/all-memory.svg`
Figure 10a	`full-sim-scalability/all.svg`
Figure 10c (left)	`optimization-overview-scalability/cell-grow-divide.svg`
Figure 10c (right)	`optimization-overview-scalability/cell-grow-divide-runtime.svg`
Figure 11a (left)	`environment/all-144.svg` [1]
Figure 11a (right)	`environment/all-18.svg` [2]
Figure 11b (left)	`environment/all-144-update-environment.svg`
Figure 11b (right)	`environment/all-18-update-environment.svg`
Figure 11c (left)	`environment/all-144-agent-ops.svg`
Figure 11c (right)	`environment/all-18-agent-ops.svg`
Figure 11d (left)	`environment/all-144-memory.svg`
Figure 11d (right)	`environment/all-18-memory.svg`
Figure 12 (left)	`load-balancing/all-144-threads.svg`
Figure 12 (right)	`load-balancing/all-18-threads.svg`

[1] 144 corresponds to the number of logical CPU cores on the system.
[2] 18 corresponds to the number of physical CPU cores on the first NUMA domain.

Mapping between the output of the script and `SF0-additional-evaluations.pdf`:

Figure	Result file (inside `result/evaluation`)
Figure 1 (left)	`mem-alloc-comp/all-total-runtime.svg`
Figure 1 (right)	`mem-alloc-comp/all-memory.svg`
Figure 2b (left)	`optimization-overview-scalability/cell-grow-divide.svg`
Figure 2b (right)	`optimization-overview-scalability/cell-grow-divide-runtime.svg`
Figure 2c (left)	`optimization-overview-scalability/soma-clustering.svg`
Figure 2c (right)	`optimization-overview-scalability/soma-clustering-runtime.svg`
Figure 2d (left)	`optimization-overview-scalability/epidemiology.svg`
Figure 2d (right)	`optimization-overview-scalability/epidemiology-runtime.svg`
Figure 2e (left)	`optimization-overview-scalability/pyramidal-cell.svg`
Figure 2e (right)	`optimization-overview-scalability/pyramidal-cell-runtime.svg`
Figure 2f (left)	`optimization-overview-scalability/tumor-spheroid.svg`
Figure 2f (right)	`optimization-overview-scalability/tumor-spheroid-runtime.svg`

These benchmarks provide the following insights. The results are discussed in full depth in Section 6.3 and 6.7–6.10 in the paper.

- **Figure 5 (left)** shows a breakdown of all operations in our benchmark simulations. The majority of the runtime is spent in agent operations (median 76.4%).
- **Figure 9 (left)** shows that the presented optimizations improve performance between 33.1× 524×.
- **Figure 9 (right)** shows that the optimizations increase the median memory consumption by a mere 1.77%, which increases to 55.6% by enabling the use of extra memory during agent sorting.
- **Figure 10a** quantifies the strong scaling behavior of BioDynaMo. The median speedup is 64.7× comparing an execution with 144 threads (72 physical CPU cores) to serial execution.
- **Figure 10c** shows that with the uniform grid and memory improvements, BioDynaMo scales well across 144 threads and 4 NUMA domains.
- **Figure 11a** shows that simulations with BioDynaMo's uniform grid are up to 191× faster than the kd-tree implementation.
- **Figure 11b** shows that building the BioDynaMo uniform grid index is almost three orders of magnitude faster than the kd-tree implementation.
- **Figure 11c** shows that BioDynaMo uniform grid outperforms the baselines for the search operations for all simulations.
- **Figure 11d** shows that simulations with BioDynaMo's uniform grid only consume 11% more memory than the kd-tree implementation in the worst case.
- **Figure 12** shows the speedup of agent sorting for four (left) and one (right) NUMA node. The oncology and cell clustering simulations benefit most of this performance improvement (peak speedup of 5.77 and 4.56× for four NUMA domains).

4.2 Comparison with Cortex3D and NetLogo (Figure 8 in the paper)

This script executes the comparison of BioDynaMo with Cortex3D and NetLogo.

- Executed on System A in 4h
- Required memory: 83 GB
- Required disk space: 10 GB
- Results in directory `1-comparison-with-others` in archive `SF4-raw-results.tar.gz`

```
# Move the previous result
mv result ../result-<choose-name>
# Create a new screen
screen
# Create a new Docker container and run the script
docker/run.sh ./run-comparison-with-others.sh
# Disconnect from the screen by pressing CTRL + a, d
```

Mapping between the output of the script and the paper:

Figure in the paper	Result file (inside `result/evaluation`)
Figure 8	`comparison-with-others/all.svg`

These benchmarks provide evidence that :

- BioDynaMo is between 23 and 89× faster than Cortex3D, even though BioDynaMo was restricted to one thread for these benchmarks.
- BioDynaMo is three orders of magnitude faster than NetLogo on System A.

The results are discussed in full depth in Section 6.6 in the paper.

4.3 Runtime and space complexity (Figure 6 in the paper)

This script analyses the runtime and memory consumption of BioDynaMo, with the number of agents increasing from 10^3 to 10^9.

- Executed on System B in 2d 2h
- Required memory: 570 GB
- Required disk space: 16 GB
- Results in directory `2-runtime-complexity` in archive `SF4-raw-results.tar.gz`

```
# Move the previous result
mv result ../result-<choose-name>
# Create a new screen
screen
# Create a new Docker container and run the script
docker/run.sh ./run-runtime-complexity.sh
# Disconnect from the screen by pressing CTRL + a, d
```

These benchmarks provide evidence that :

- The runtime and memory consumption increases linearly with the number of agents increasing from 10^3 to 10^9.
- BioDynaMo is capable of simulating one billion agents.

The results are discussed in full depth in Section 6.4 in the paper. The following table shows the mapping between the output of the script and the paper:

Figure in the paper	Result file (inside `result/evaluation`)
Figure 6 (left)	`runtime-complexity/all.svg`
Figure 6 (right)	`runtime-complexity/all-memory.svg`

4.4 Workload profiling (Figure 5 (right) in the paper)

This script performs the microarchitecture analysis for all simulations in Table 1 in the paper.

- Executed on System A in 4h
- Required memory: 38 GB
- Required disk space: 32 GB
- Results in directory `3-profiling` in archive `SF4-raw-results.tar.gz`

```
# Move the previous result
mv result ../result-<choose-name>
# Create a new screen
screen
# Create a new Docker container and run the script
docker/run.sh ./run-profiling.sh
# Disconnect from the screen by pressing CTRL + a, d
```

The file `result/evaluation/uarch-analysis.svg` corresponds to Figure 5 (right) in the paper.

This analysis provides evidence that the five benchmark simulations are predominantly memory bound. Section 6.3 in the paper provides more details.

4.5 Biocellion comparison small (Figure 7 in the paper)

This script executes the small-scale comparison with Biocellion, and the optimization analysis in Figure 6b (right).

- Executed on System C in 7h
- Required memory: 12 GB
- Required disk space: 1 GB
- Results in directory `4-biocellion-cmprsn-single-node` in archive `SF4-raw-results.tar.gz`

```
# Move the previous result
mv result ../result-<choose-name>
# Create a new screen
screen
# Create a new Docker container and run the script
docker/run.sh ./run-biocellion-cmprsn-single-node.sh
# Disconnect from the screen by pressing CTRL + a, d
```

Figure in the paper	Result file (inside `result/evaluation`)
Figure 7b (right)	`optimization-overview-16-cpus/biocellion-cell-clustering.svg`

Open the spreadsheet at `bdm-paper-examples/bdm/biocellion-cell-clustering/comparison.ods` and follow the instructions below to calculate the speedups.

- Copy the content of `biocellion-cell-clustering/single-node/runtime` into the cells `B32:B34`.
- The calculated speedup is shown in cell `B41`.

This benchmark provides evidence that BioDynaMo is 4.15x faster than Biocellion for the cell sorting simulation with 26.8 million cells using 16 physical CPU cores. The results are discussed in full depth in Section 6.5 in the paper.

4.6 Biocellion comparison large (Figure 7 in the paper)

This script executes the large-scale comparison with Biocellion, the optimization analysis in Figure 6b (left), and also renders the visualization in Figure 6a.

- Executed on System B in 16.5h
- Required memory: 470 GB
- Required disk space: 1 GB
- Results in directory `5-biocellion-cmprsn-cluster` in archive `SF4-raw-results.tar.gz`

```
# Move the previous result
mv result ../result-<choose-name>
# Create a new screen
screen
# Create a new Docker container and run the script
docker/run.sh ./run-biocellion-cmprsn-cluster.sh
# Disconnect from the screen by pressing CTRL + a, d
```

Figure in the paper	Result file (inside `result/evaluation`)
Figure 7b (left)	`optimization-overview/biocellion-cell-clustering.svg`
Figure 7a	`biocellion-cell-clustering-final-state-highres.png`

Open the spreadsheet at `bdm-paper-examples/bdm/biocellion-cell-clustering/comparison.ods` and follow the instructions below to calculate the speedups.

- Copy the content of `biocellion-cell-clustering/cluster/runtime` into the cells `B7:B9`
- Copy the content of `biocellion-cell-clustering/cluster-281m/runtime` into the cells `E7:E9`
- Copy the content of `biocellion-cell-clustering/single-node/runtime` into the cells `E32:E34`
- Enter the number of physical CPU cores of the used benchmark hardware into `B18` and `E18`
- The improved efficiency of BioDynaMo per CPU core is shown in cell `B21` and `E21`.

This benchmark provides evidence that:

- BioDynaMo is 9.64x more efficient per CPU core than Biocellion for the cell sorting simulation with 1.72 billion cells using 72 physical CPU cores.
- Figure 6a shows a good qualitative agreement to the simulation results in the Biocellion paper.
- Figure 6b illustrates that the memory optimizations have a higher speedup for the larger scale simulation with more CPU cores (Figure 6b left) than the smaller scale benchmark (Figure 6b right). Figure 6b right can be found in the results from the previous section.

The results are discussed in full depth in Section 6.5 in the paper.

5 Reusing and repurposing this artifact

Besides reproducing the results, researchers can build upon this artifact in numerous ways. A non-exhaustive list of possibilities is given in this section.

5.1 Add additional benchmarks

This artifact provides all necessary functionality to easily add additional benchmarks. Let's assume that we want to investigate the impact of the agent batch size parameter on simulation runtime. This study was omitted from the paper, due to the page limit of the PPoPP conference, but is included in this artifact.

Let's go through the main steps.

1) Create a benchmark template (see file `bdm-paper-examples/benchmark/batch-size.sh`)

```
#!/bin/bash

source $BDM_SCRIPT_DIR/../../util/default-compile-script.sh ""

for f in 10 100 1000 10000 50000; do
  benchmarkIC $(CPUCount) \
    "$RESULT_DIR/$SIMULATION/batch-size/$f/runtime" \
    $CMD --inline-config="{ \"bdm::Param\": { \"scheduling_batch_size\": $f }}"
done
```

The benchmark template uses the default parameters to compile BioDynaMo and the simulation and executes the simulation for five different batch sizes using the `benchmarkIC` function. The first parameter determines the result path, while the remaining parameters form the executed simulation command.

2) Create a script for each simulation that should be benchmarked

The benchmark template from the previous step has placeholders for the simulation name (`$SIMULATION`) and the command that should be executed (`$CMD`). These values are set in separate scripts for each simulation.

Let's take the neuroscience use case as an example (`bdm-paper-examples/bdm/pyramidal-cell/batch-size.sh`)

```
#!/bin/bash

SIMULATION="pyramidal-cell"
CMD="./pyramidal-cell --mode=cover-image --config=../cover.json"

BDM_SCRIPT_DIR=$(readlink -e $(dirname "${BASH_SOURCE[0]}"))
source $BDM_SCRIPT_DIR/../../benchmark/batch-size.sh
```

The script sets the required variables (`SIMULATION`, and `CMD`), determines the location of itself (`BDM_SCRIPT_DIR`), and sources the benchmark template file.

3) Create a script to plot the results (`plot/batch_size.py`)

Creating the batch size study plot can be done in less than 20 lines of code, reusing the `parameter_study` definition in this artifact.

```
#!/usr/bin/env python3

import numpy as np
import matplotlib.pyplot as plt
import os
import sys
from parameter_study import generate


def main(result_dir, sim_name, suffix):
    plt.rcParams.update({'font.size': 12, 'legend.fontsize': 16})

    generate(result_dir, sim_name, 'batch-size{}'.format(suffix), 'Batch Size',
             'Speedup compared to batch size 10', True, False)


if __name__ == '__main__':

    if len(sys.argv) < 2:
        print("Missing argument.")
        print("  ./{0} all/sim_name [suffix]]".format(sys.argv[0]))
        sys.exit(-1)

    suffix = ""
    if len(sys.argv) == 3:
        suffix = sys.argv[2]

    main(os.environ["RESULT_DIR"], sys.argv[1], suffix)
```

4) Execute the benchmark and generate the plot for all simulations

To execute the benchmark for each simulation with a `batch-size.sh` file inside its project directory, run the following command in the docker container:

```
source util/main.sh
util/foreach-sim.sh batch-size
```

To generate a plot summarizing the results for all simulations, execute the following:

```
python plot/batch_size.py all
```

This step concludes the tutorial on how to extend this artifact with a new benchmark type.

5.2 Evaluate the effectiveness of additional optimizations

This artifact can also be used to develop new performance optimizations for the BioDynaMo simulation engine. The researcher can focus solely on developing the optimization and rely on the infrastructure in this artifact to evaluate the speedups, memory consumption, etc.

The benchmark template could look like this:

```
#!/bin/bash

source $BDM_SCRIPT_DIR/../../util/default-compile-script.sh ""

benchmarkIC $(CPUCount) \
  "$RESULT_DIR/$SIMULATION/new-optimization/off" \
  $CMD --inline-config='{ "bdm::Param": {"new_optimization": 0 }}'

benchmarkIC $(CPUCount) \
  "$RESULT_DIR/$SIMULATION/new-optimization/on" \
  $CMD --inline-config='{ "bdm::Param": {"new_optimization": 1 }}'
```

For the remaining steps, please follow the instructions in Section 5.1.

5.3 Evaluate BioDynaMo's performance for additional simulations

Adding additional simulations to evaluate the performance of BioDynaMo is straightforward:

1) Copy the new BioDynaMo simulation to the `bdm-paper-example/bdm` folder.
2) Add the headers and source files in `bdm/shared` in the simulations `CMakeLists.txt` file. (see `bdm/soma-clustering/CMakeLists.txt:L27-30`).

```
include_directories("../shared")
file(GLOB_RECURSE HEADERS src/*.h ../shared/*.h)
file(GLOB_RECURSE SOURCES src/*.cc ../shared/*.cc)
```

3) Include two benchmark-related headers
(see `bdm/soma-clustering/src/soma_clustering.h:L18-19`).

```
#include "benchmark_param.h"
#include "benchmark_init.h"
```

4) Make sure that the simulation uses the `bdm::BenchmarkParam`
(see `bdm/soma-clustering/src/soma_clustering.h:L50`).

```
Param::RegisterParamGroup(new BenchmarkParam());
```

5) Make sure that the simulation uses `bdm::SimParam` that contains the parameters: `iterations` and `num_agents`
(see `bdm/soma-clustering/src/soma_clustering.h:L37-45`).

```
struct SimParam : public ParamGroup {
  BDM_PARAM_GROUP_HEADER(SimParam, 1);
```

```
  // determines the number of iterations the simulation is executed for
  uint64_t iterations = 100;
  // determines the number of agents the simulation is executed for
  uint64_t num_agents = 100;
  // other parameters ...
};
```

6) Call the `InitBenchmarkObjects` functions right after instantiating the BioDynaMo `Simulation` object. (see `bdm/soma-clustering/src/soma_clustering.h:L62`).
7) Add the simulation name in `plot/common.py` to the functions `getAllSim` and `getLegendName`
8) Implement all benchmark templates that the new simulation should be tested with, e.g., microarchitecture analysis:

```
#!/bin/bash

SIMULATION="new-simulation-name"
CMD="./new-simulation-name"

BDM_SCRIPT_DIR=$(readlink -e $(dirname "${BASH_SOURCE[0]}"))
source $BDM_SCRIPT_DIR/../../benchmark/microarchitecture.sh
```

Now the new simulation is fully integrated into the evaluation framework and will be used in the scripts presented in Section 4. Try, for example, to run the microarchitecture analysis.

6 Contact

Please contact us with any feedback or if you need any help building on BioDynaMo. You can reach us at lukas.breitwieser@gmail.com and omutlu@gmail.com.