OpenABL Artifact Overview Document

Biagio Cosenza, Nikita Popov, Ben Juurlink
Paul Richmond, Mozhgan Kabiri Chimeh
Carmine Spagnuolo, Gennaro Cordasco, Vittorio Scarano

May 29, 2018

1 Getting Started

1. Download and extract https://github.com/OpenABL/OpenABL/archive/master.zip:

   ```
   wget https://github.com/OpenABL/OpenABL/archive/master.zip
   unzip master.zip
   cd OpenABL-master/
   ```

2. General installation and usage instructions are available in the `README.md` file. A more condensed version of these instructions is reproduced in the following steps. The used commands target Ubuntu systems, but equivalent packages should be available for other Linux distributions.

3. Install build dependencies:

   ```
   sudo apt-get install flex bison cmake g++
   ```

4. Perform an out-of-source build:

   ```
   mkdir ./build
   cmake -Bbuild -H.
   make -C build -j4
   ```

5. Install build and runtime dependencies for backend libraries:

   ```
   sudo apt-get install git autoconf libtool libxml2-utils xsltproc default-jdk
   ```

6. Install backend libraries (might take a few minutes):

   ```
   make -C deps
   ```

7. FlameGPU requires a CUDA 8 installation. If CUDA is not installed in `/usr/local/cuda`, the path needs to be specified:

   ```
   export CUDA_PATH=/usr/local/cuda-8.0 # for example
   ```

   Additionally the SM architecture of the GPU needs to be specified:

   ```
   export SMS=52 # for SM architecture 5.2 (default: SMS="30 35 37 50 52")
   ```

8. Test that all backends work using the `circle` model:

   ```
   build/OpenABL -i examples/circle.abl -b c -R
   build/OpenABL -i examples/circle.abl -b mason -R
   build/OpenABL -i examples/circle.abl -b dmason -R
   build/OpenABL -i examples/circle.abl -b flame -R
   build/OpenABL -i examples/circle.abl -b flamegpu -R
   ```

   Also test Mason visualizations using the `ants` model:

   ```
   build/OpenABL -i examples/ants.abl -b mason -R -C visualize=true
   # Click play. After about 500 steps a pheromone trail starts to form.
   ```

2 Step-by-Step Instructions

2.1 Single-Node Performance

The single-node performance comparison results in Fig. 3 of the paper can be obtained by running the bundled benchmarking and plotting scripts using Python:

```
python bench/bench.py -r results/

# Install matplotlib if not available yet
sudo apt-get install python-matplotlib
python bench/plot.py results/
```

By default, the benchmarking script will use the same backends, models and agent counts as those in the paper. On our test platform this takes approximately two hours to execute. The benchmarking time can be reduced by passing the `--max-time` parameter, which (approximately) restricts the time spent per backend/model combination:

```
python bench/bench.py -r results/ --max-time 60 # in seconds
```

In the default configuration of 4 backends and 6 models, this will take approximately $4 \cdot 6 \cdot 60$ s = 24 min.

It is also possible to only execute the benchmarks for specific backends and models, and to manually specify an agent number range. The following example executes benchmarks for the `circle` and `boids2d` models on Mason, with 250 to 64000 agents:

```
python bench/bench.py -r results/ -b mason -m circle,boids2d \
       --num-agents=250-64000
```

This can be used to rerun individual benchmarks in case of irregularities, or if the initial run used too few agents.

Our own benchmarks were performed on a system with an Intel Core i5-4690K CPU at 3.50GHz, 16GB of memory, running on Ubuntu 16.04. For FlameGPU, we used an NVIDIA Titan Xp (Pascal architecture) with 12GB of memory.

2.2 Cluster Strong Scaling Results

Unfortunately, setting up the D-Mason library to work in a cluster environment is relatively involved and can at present not be automated. For this reason the D-Mason strong scaling results in Fig. 4 of the paper cannot be reproduced in a short amount of time.

As an alternative, if a machine with a sufficient number of cores is available, it may be possible to perform a local multi-core simulation with somewhat similar characteristics. For this purpose additional benchmarking and plotting script are provided:

```
python bench/bench_dmason.py -r results/ -t 32
python bench/plot_dmason.py results/
```

The `-t` parameter specifies the maximum number of threads that will be used and should match the core count of the machine. By default the benchmark runs with 100000 agents and 100 timesteps, which can be changed using the `--num-agents` and `--num-timesteps` parameters respectively.

Our own benchmarks were performed on a cluster of 12 nodes equipped with two Intel Xeon E5-2430 (six core) processors each, with hyper-threading disabled and connected by I350 Gigabit network adapters. One node was reserved for the coordination of the simulation, while the others allocated one D-Mason logical processor for each core, running on Oracle JVM 1.8. Apache ActiveMQ was used for communication/synchonization, with the broker being allocated on the coordinating node.