STriTuVaD UISS FLAME GPU

Nov 19, 2019

Contents:

1 Model and Implementation 3

2 Running the Simulator 7

3 Troubleshooting 9

4 Tools 11

5 Resources at the University of Sheffield 17

6 Funding 19

This is the public facing documentation for the FLAME GPU 1.5 implementation of the UISS 6 simulator, as part of the STriTuVaD H2020 project.

CHAPTER 1

Model and Implementation

The model is based on UISS 6.0, developed and provided by Francesco Pappalardo et Al. at the University of Catania.

The current implementation is built using a modified version of FLAME GPU 1.5, which imposes some restrictions on how much of the original model can be directly reproduced (to be addressed with a FLAME GPU 2 implementation).

1.1 Model

The model contains agents located at discrete locations within a toroidal, hexagonal lattice. Each lattice site contains quantities of chemicals, and some lattice-site behaviours occur (such as diffusion).

The agents include:

- BCell
- THelper
- TCytotoxic
- DendricCell
- EpiteliaCell
- MCell
- PlasmaCell
- NKCell
- ICList
- AntibodyList
- AntigenList

1.2 Implementation

The current implementation is based on UISS 6.0, using a modified version of FLAME GPU 1.5.0

1.2.1 FLAME GPU

FLAME GPU is a high performance Graphics Processing Unit (GPU) extension to the FLAME framework for Agent Based Modelling (ABM). It provides a mapping between a formal agent specifications with C based scripting and optimised CUDA code.

Users formally describe the model using the XMLModelFile (following provided XML Schema) and implement behaviours in CUDA C, using the pragmatically generated API. This abstracts the complexities of GPU programming away from the modeller.

FLAME GPU is open source, and can be found on github at FLAMEGPU/FLAMEGPU.

Documentation is also host both on github at FLAMEGPU/docs and online at http://docs.flamegpu.com/en/master/.

Modifications

To enable the model to be implemented using FLAME GPU, modifications were made to FLAME GPU to enable certain behaviours not previously required by FLAME GPU models.

Currently these are implemented in a branch of a separate repository (uiss-6 branch in ptheywood/flamegpu), as they are not yet ready for general purpose use.

1.2.2 XMLModelFile

The XMLModelFile.xml in the source repository is the formal definition of the model. It contains:

- Parameters of the Model
- Agent descriptions
- Message List definitions
- Agent behaviour descriptions
- Directed Acyclic Graph (DAG) of per-iteration behaviour

1.2.3 Functions Files

As the model is complex, rather than using a single CUDA C file, functions.c, for all agent behaviour implementation, it is split across several implementation files (.h, .c, .cuh, .cu), in the src/model/ directory. Further information can be found in the source repository.

1.2.4 Input File

FLAME GPU simulations expect and XML file describing the initial state of the simulation as an input file.

For the UISS implementation, this is an XML formatted version of the UISS 6 input file format. This can be converted automatically using one of the provided tools·

1.2.5 Output Files

The FLAME GPU UISS implementation aims to match the output of the FLAME GPU

1.2.6 Limitations

Due to the nature of the GPU implementations, and limitations of the current version of FLAME GPU, not all of the UISS features can be replaced exactly.

For instance, the order of interactions is not shuffled on subsequent iterations. The source repository contains further information on the limitations of the implementation.

1.2.7 Further Information

Further information can be found within the source repository, containing non-public information.

CHAPTER 2

Running the Simulator

To run the FLAME GPU UISS simulator, the code must first be compiled to be executed on an NVIDIA GPU.

See the source repository contains more precise requirements and instructions.

2.1 Requirements (Linux)

- CUDA 8.0 or greater
- GCC 7 or greater
- xsltproc
- xmllint
- make
- git

To run FLAME GPU simulations a Compute Capability 3.0 or greater Nvidia GPU is required, with an appropriate CUDA driver.

2.2 Compilation

To build the executable, navigate into the relevant source directory and run *make console*.

Other options are available, as described in the source repository.

2.3 Running the Simulator

FLAME GPU Console mode simulations take several arguments as follows:

```
usage: FLAMEGPU_UISS [-h] [--help] input_path num_iterations [cuda_device_id] [XML_
↪output_override]
```

Where

- input_path corresponds to an input xml file, typically called 0.xml. This should contain the model parameters.
- num_iterations is the number of iterations for the simulator to run. For UISS this should be set to 0, as the number of iterations is related to several input parameters, and calculated at runtime.
- *cuda_device_id* controls which GPU should be used in a multi-GPU system. Typically the value of 0 should be used.
- XML_output_override controls how often a FLAME GPU XML output file is produced. This has significant performance impact and is not required for the UISS implementation. Use the value *0*.

I.e.

```
./bin/linux-x64/Release_Console/FLAMEGPU_UISS iterations/0.xml 0 0 0
```

Output files will be produced in the same directory as *0.xml*, i.e. *iterations/* in the above example.

2.4 Tools

Several scripts and tools have been developed to simplify the running of the model. See *Tools*. for further information.

CHAPTER 3

Troubleshooting

The source code for the FLAME GPU implementation can be found on github at StrituvadModelling/strituvad-uiss-flamegpu. This is a private repository, if you do not have access please contact p.heywood@sheffield.ac.uk or p.richmond@sheffield.ac.uk.

3.1 Bugs Reports and Feature Requests

As with all software you may encounter crashes, bugs or missing features.

Bugs, Feature Requests and source-related discussion should happen using the github issue tracker.

An issue can then be created via *New issue*, adding an appropriate title and a description of the problem. To improve the chances that we can address the issue, please include:

- The input file (you can attach files to issues)
- The relevant commands used to launch the executable
- Observed behaviour
- Expected behaviour

CHAPTER 4

Tools

Several scripts / tools have been developed to simplify the use of the simulator, which can be found in the StrituvadModelling/strituvad-uiss-flamegpu repository.

Further information and examples of all these scripts can be found in the source repository.

4.1 Input File Conversion Script

The FLAME GPU implementation of UISS 6 takes a FLAME GPU input file as an input parameter. A Python script has been developed to convert a UISS input file in to an appropriately formatted FLAME GPU input file.

Usage is as follows:

```
# Convert 'datafile' to '0.xml'
python3 dat2xml.py datafile -o 0.xml

# use --help for further information
python3 dat2xml.py --help
usage: dat2xml.py [-h] [-o OUTPUT] [-f] input

Convert a UISS input data file to a FLAME GPU 1.x initial states file. This is
very brittle. The data file must contain the correct number of lines

positional arguments:
  input                 UISS Datafile

optional arguments:
  -h, --help            show this help message and exit
  -o OUTPUT, --output OUTPUT
                        path to output file
  -f, --force           Force overwriting of output files
```

4.2 Input File Mutation Script

As the UISS model is highly stochastic, many runs using different RNG seeds are required.

To avoid doing this by hand, a script was developed to automate this process.

Usage is as follows:

```
python3 gendat.py --help
usage: gendat.py [-h] [-f FILENAME] [-p PARAMETER [PARAMETER ...]] [-o OUTPUT]
                 [--list-parameters] [-v]

UISS Datafile modifier / Randomiser

optional arguments:
  -h, --help            show this help message and exit
  -o OUTPUT, --output OUTPUT
                        Directory for output files. Defaults to `.`
  --list-parameters     list the parameters of the program
  -v, --verbose         print verbose output

required arguments:
  -f FILENAME, --filename FILENAME
                        template for the parameter file
```

Note: This may be extended in the future to simplify parameter sweeps.

4.3 Output File Visualisation Script

Along side the existing gnuplot script for visualising populations, a python script was developed to support the visualisations of many runs, and view the range of outputs capture.

Usage is as follows:

```
python3 uiss-plot.py --help
usage: uiss-plot.py [-h] [-f FILES [FILES ...]] [-s MIN MAX] [-o OUTPUT]
                    [--verbose]

Tool to plot one or more UISS output files as figure(s)

optional arguments:
  -h, --help            show this help message and exit
  -f FILES [FILES ...], --files FILES [FILES ...]
                        Files to plot. One graph per passed argument
  -s MIN MAX, --shade MIN MAX
                        Min, max value pairs for shaded regions.
  -o OUTPUT, --output OUTPUT
                        path to output file
  --verbose             Verbose output
```

4.4 TUOS HPC Job Submission Scripts

Batch job submission scripts are used to submit jobs to HPC systems, such as ShARC and Bessemer at the university of Sheffield.

Example submission scripts are provided for each job submission system.

4.4.1 SGE (ShARC)

ShARC (and Iceberg) at the University of Sheffield use the SGE job submission system.

The tools sub-repository contains example job submissions scripts to compiler the FLAME GPU UISS implementation, and run the 5 example datafiles.

Alternatively it is possible to use interactive sessions.

Compilation via SGE

Compilation via SGE can occur using the `tools/sge/compile.sharc.sh` job submissions script.

```
# Navgate to the tools/sge directory
cd strituvad-uiss-flamegpu/tools/sge/
# Submit the batch job to compile the code
qsub compile.sharc.sh
# Ensure that the job has finished before running the model
qstat
```

Alternatively this can be ran within an interactive sesssion.

```
# Get an interactive session, no GPU required
qrshx
# Navgate to the tools/sge directory
cd strituvad-uiss-flamegpu/tools/sge/
# Run the compilation script
./compile.sharc.sh
```

Note: Compilation does not require a GPU.

Execution via SGE

To run an example, once the executable has been compiled (see above) the appropriate job submission script can be submit.

I.e. for the `datafile_1` example, with a single invocation of a single seed

```
# Navgate to the tools/sge directory
cd strituvad-uiss-flamegpu/tools/sge/
# Submit the batch job to compile the code
qsub datafile_1.sharc.sh
```

Alternatively this can be executed in a GPU interactive session, although this is not reccomended.

```
# Get a GPU interactive session (in the private queue for this project on ShARC)
qrshx -l gpu=1 -P rse-strituvad
# Navigate to the SGE directory
cd strituvad-uiss-flamegpu/tools/sge/
# Run the appropriate script (or runs the commands contained within manually)
./datafile_1.sharc.sh
```

This will place generated `.out` files in the `iterations/datafile_1` directory, which can be retrieved using scp.

```
# From your local machine, replacing $CICS_USERNAME with your cics username
scp -r $CICS_USERNAME@sharc.shef.ac.uk:~/strituvad-uiss-flamegpu/simulator/iterations/
↪datafile_1/\*.dat path/to/local/destination
```

Checking job progress via SGE

To check the progress of your jobs on SGE, use the `qstat` command

```
qstat
```

4.4.2 Slurm (Bessemer)

Note: Bessemer does not yet contain the private GPU nodes we will be using.

Job submission scripts will be updated to reflect this at a later date.

Bessemer at the University of Sheffield use the Slurm job submission system.

The tools sub-repository contains example job submissions scripts to compiler the FLAME GPU UISS implementation, and run the 5 example datafiles.

Alternatively it is possible to use interactive sessions.

Compilation via Slurm

Compilation via slurm can occur using the `tools/slurm/compile.bessemer.sh` job submissions script.

```
# Navgate to the tools/slurm directory
cd strituvad-uiss-flamegpu/tools/slurm/
# Submit the batch job to compile the code
sbatch compile.bessemer.sh
# Ensure that the job has finished before running the model
squeue -u $USER
```

Alternatively this can be ran within an interactive sesssion.

```
# Get an interactive session, no GPU required
srun --pty bash -i
# Navgate to the tools/slurm directory
cd strituvad-uiss-flamegpu/tools/slurm/
# Run the compilation script
./compile.bessemer.sh
```

Note: Compilation does not require a GPU.

Execution via Slurm

To run an example, once the executable has been compiled (see above) the appropriate job submission script can be submit.

I.e. for the `datafile_1` example, with a single invocation of a single seed

```
# Navgate to the tools/slurm directory
cd strituvad-uiss-flamegpu/tools/slurm/
# Submit the batch job to compile the code
sbatch datafile_1.bessemer.sh
```

Alternatively this can be executed in a GPU interactive session, although this is not reccomended.

```
# Get a GPU interactive session
srun --pty bash -i --gres=gpu:1 --partition=gpu
# Navigate to the slurm directory
cd strituvad-uiss-flamegpu/tools/slurm/
# Run the appropriate script (or runs the commands contained within manually)
./datafile_1.bessemer.sh
```

This will place generated `.out` files in the `iterations/datafile_1` directory, which can be retrieved using scp.

```
# From your local machine, replacing $CICS_USERNAME with your cics username
scp -r $CICS_USERNAME@bessemer.shef.ac.uk:~/strituvad-uiss-flamegpu/simulator/
↪iterations/datafile_1/\*.dat path/to/local/destination
```

Checking job progress on Slurm

To check the progress of your jobs via slurm, use the `squeue` command

```
squeue -u $USER
```

CHAPTER 5

Resources at the University of Sheffield

The University of Sheffield (TUOS) has several HPC facilities which can be used as a part of the Strituvad project for running simulations, including CPU and GPU resources.

The facilities are namely ShARC and Bessemer.

5.1 Gaining Access: TUOS Staff

Members of staff at the University of Sheffield can gain access to HPC systems by emailing helpdesk@sheffield.ac.uk.

To access private GPU nodes, please contact p.richmond@sheffield.ac.uk.

5.2 Gaining Access: External Users

Project members who are not members of the University of Sheffield require a University of Sheffield computing account to access these resources. Individuals must register for an External UCard and Computing Account by completing the relevant form. Once complete please email this to p.richmond@sheffield.ac.uk.

5.3 Connecting to the Clusters

To connect to TUOS HPC facilities, you must either be on campus, or connected to the university of Sheffield VPN.

To connect to the VPN, please follow the instructions provided by CiCS.

It is then possible to connect via SSH. Please see the Sheffield HPC Docs for further information.

5.4 Transferring Files

Please see the Sheffield HPC Docs for information on transferring files to and from the HPC clusters.

5.5 ShARC

ShARC is one of the University of Sheffield HPC Clusters, optimised for multi-node computing.

ShARC also contains some GPU nodes, notably. K80 GPUs in the public queue, and P100 GPUs in various private queues. For the purposes of the Strituvad project the private GPU queues can be made available.

ShARC uses the Son of Grid Engine Scheduler (SGE). Details on how to make use of this scheduler can be found on the Sheffield HPC Documentation.

5.6 Bessemer

Bessemer is one of the University of Sheffield's HPC clusters, optimised for single-node or high-throughput computing.

Bessemer contains 1 public GPU node containing 4 V100 GPUs, but will also contain many private V100 nodes, which will be made available for the Strituvad project.

Bessemer uses the Slurm Scheduler. Details on how to make use of this scheduler can be found on the Sheffield HPC Documentation.

CHAPTER 6

Funding

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement n. 777123).

https://cordis.europa.eu/project/rcn/212940/factsheet/en