Porting CPU Agent-Based Modelling Applications to GPU

Moving NetLogo Models to FLAMEGPU

V. Li and G. Speyer
Research Computing at Arizona State University, Tempe, AZ 85287
Contact: vincentl@asu.edu

stu_pos108s1

Abstract

Agent-based modelling (ABM) employs concurrent, communicating agents to model emergent, complex behaviors. **NetLogo, a widely used CPU application**, incorporates many ABM models. Graphical processing unit (GPU) implementations of ABMs have demonstrated significant acceleration due to the massive data-parallel computation, custom random-number generators, and the GPGPU "scatter" communication model. This project focuses on converting NetLogo models to run on one such **GPU-based ABM application, FLAMEGPU**. This effort will facilitate researchers within domains that employ NetLogo to access **accelerated computation**.
This application of advanced research computing software and applications should dovetail with the purpose of PEARC, exploiting the performance made feasible on GPUs by empowering state-of-the-art ABM simulations.

Agent-Based Modelling (ABM)

- About:
  - The behaviors and rules of agents are defined
  - Individual agents interact with each other and with the environment to show the collective behavior of the system
  - Used to simulate systems related to physics, biology, social science, and psychology, among other fields

Run-Time Comparison

- Using the Boids simulation (flocking model)
  - "Birds" have position and velocity variables
  - 3 rules of behavior:
    - Separation: birds can't get too close to each other (overcrowding)
    - Alignment: birds update their velocity to correspond with the average velocity of birds within a given radius
    - Cohesion: birds move towards the average position of birds within a given radius
- Recorded the run-time for 100 iterations for doubled data sets
- Ran 20 reps for each problem size and plotted the average
- NetLogo "Flocking" model run on an Intel Xeon E5-2680 v4 using all 28 CPU's in headless mode (no display)
- FlameGPU "Boids_2D" model run on a Nvidia Tesla V100 using 2 GPU's in console mode (no display)

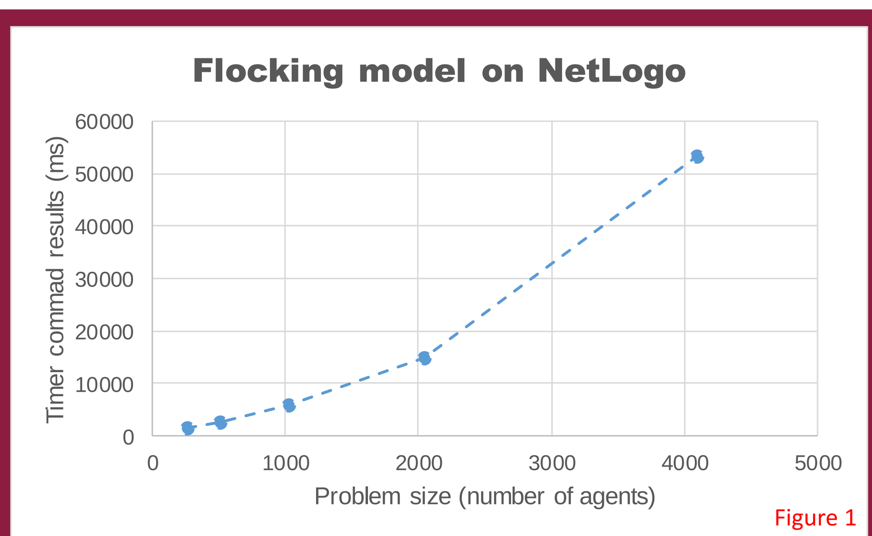

Figure 1

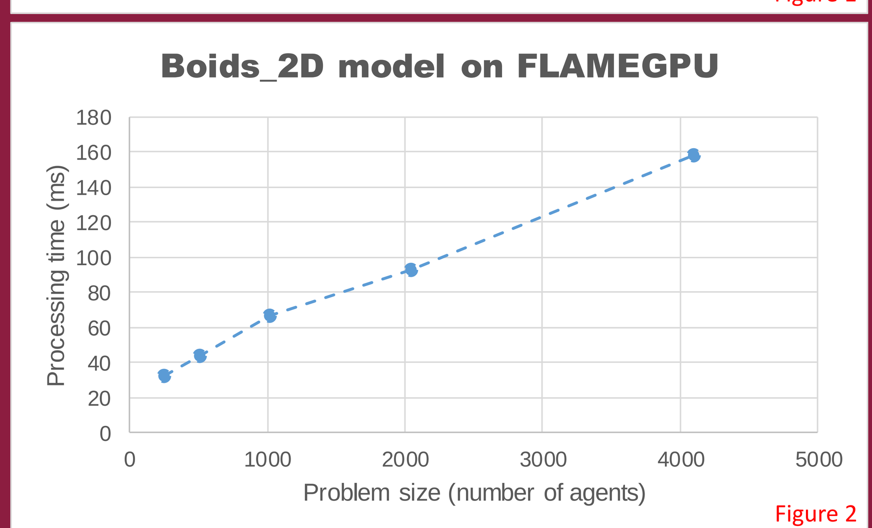

Figure 2

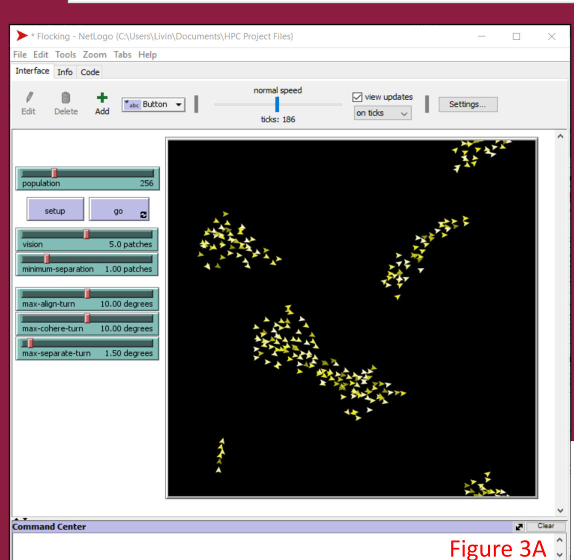

Figure 3A: Screenshot of the NetLogo Flocking model on the desktop app

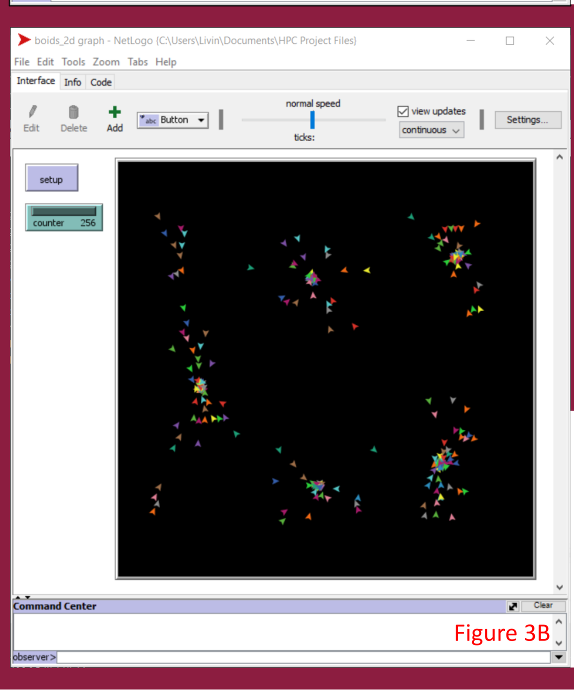

Figure 3B: Screenshot of the FLAMEGPU Boids_2D model with its output (200.xml) translated to NetLogo code to be plotted on the desktop app

Package Descriptions

NetLogo:
- Desktop application
  - Runs on Java Virtual Machine (better for CPU)
  - Can execute simulations through command line and bash scripts with headless mode
  - Automatically run in parallel, if there are multiple processor cores
- Model library includes numerous examples which can be modified

FLAMEGPU:
- On Visual Studio or Linux console
- Console and visualization modes
- Makefile and dynamically generated C header file help with compilation
- FLAMEGPU templates create parallelized CUDA code with an executable
- Package has examples which can be copied with a Python
- command and then modified

Usage Differences

NetLogo:		FLAMEGPU:
1. A 2D/3D environment is created by the app • User does not have to write position and vector variables for the agents • Patches (grid blocks)	-Spatial Environment-	1. A 2D/3D environment must be programmed • Agents have position and vector variables (x, y, z). Movements must be programmed • No patches
2. Agents are continuous; patches are discrete	-Agent Types-	2. Agents can be either continuous or discrete
3. Agents are not state machines	-State Machines-	3. Agents can be state machines
4. The initial orientation of agents (usually random) is generated by setup functions	-Agent Inputs-	4. The initial orientation of agents must be specified by the user in 0.xml
5. XML experiments code shows which simulation parameters change for each run	-Parameter Sweeping-	5. Each simulation parameter combo must be run in a separate job (no parameter sweeping)
6. Function dependencies are implied	-Dependencies-	6. Function dependencies are programmed
7. Max number of agents is not programmed	-Agent Buffer Size-	7. Max number of agents is programmed
8. Time is manually incremented	-Iteration Time-	8. Time is automatically incremented

NetLogo Model File

```
;; Useless.nlogo

;; code shown in the editor of the desktop application

turtles-own [var] ;; internal variable of turtle agent

;; pre-simulation function
to setup
   clear-all
   setup-turtles
   reset-ticks
end

;; pre-simulation function
to setup-turtles
   create-turtles number
   ;; value of number defined in the XML experiments code
   ;; initialize turtles with var >=0 and <5
   ask turtles [
      set var random-float 5
   ]
end

;; agent function
to go
   ask turtles [
      do-stuff
   ]
   tick   ;; all turtles will perform do-stuff in each
          ;; iteration
end

;; arbitrary calculations done by each turtle during each
;; iteration
to do-stuff
   set var var ^ (4)
   set var var ^ (0.25)
   set var var + random-float 1
   set var var - random-float 1
end
```

Figure 4

NetLogo XML-Format Experiments

```
;; Useless.nlogo
<experiments>
  <experiment name="256" repetitions="1"
runMetricsEveryStep="true">
    <setup>setup</setup>
    <go>go</go>
    <exitCondition>ticks = 100</exitCondition>
    <metric>timer</metric>
    <enumeratedValueSet variable="number">
      <value value="256"/>
    </enumeratedValueSet>
  </experiment>
</experiments>
```

Figure 5

Discussion

Figures 1-2 show the run times for each implementation. NetLogo has a "timer" command which can be used to output the run-time of each tick, and FLAMEGPU simulations show a "processing time" of the simulation in standard output. Although it was difficult to create equal footing between both situations, since the hardware and the software packages are different, the orders of growth shown by the data are insightful. There's a clear advantage of using GPU's for ABM.

Figures 4-5 and 6-8 are code implementing an extremely simple example called "Useless" on NetLogo and FLAMEGPU, which has each agent perform arbitrary calculations. The Boids code would have been too large to display on this poster. Code that corresponds to each other are highlighted with the same colors. Light blue code has to do with agent initialization, green specifies function code, yellow shows function dependencies, and pink is related to experiment setup. Additionally, experiment setup in NetLogo, such as the pre-simulation functions (Fig. 4) and the XML code (Fig. 5) aren't addressed in FLAMEGPU but can be worked around with extra C/Python scripts and Bash scripts. Also, FLAMEGPU asks for agent states, specific random number generators configured for continuous and discrete agents, and messages, which are closely related to NetLogo links but have partitioning types specific for the model, as well as their own buffer sizes. These must be specified by the user. More info about FLAMEGPU code syntax can be found at its documentation website.

←Porting a Very Basic Example→

Format Descriptions:

.xml model file (Fig 6) ← .nlogo file (Fig. 4)
- Environmental constants
- Name of c function file "functions.c"
- Agents
  - Internal variables
  - Functions
  - Random number generator? (RNG true/false)
  - Reallocate? (agent death)
  - States
  - Type? (continuous or discrete)
  - Buffer size? (max number of agents in the simulation)
- Messages (communication between agents)
  - Message name and variables
  - Partitioning type? (none, discrete, spatial, graph edge)
- Layers (function dependencies)

.c functions file (Fig. 7) ← .nlogo file (Fig. 4)
- Include header.h, which can be blank and is dynamically generated
- Function definitions
- Specific syntax to:
  - access variables
    - "xmachine_memory_agent-name*" to get internal variables
  - Iterate through agent messages
  - Create or kill agents
  - Use a random number generator
    - Include "RNG_rand48* rand48" as an input of the function
    - "rnd<CONTINUOUS>(rand48)"
    - "rnd<DISCRETE2D>(rand48)"
  - All functions are prefixed with "__FLAME_GPU_FUNC__"

.xml initial states file (Fig 8) for simulation input
- Environmental constants with given values
- Declare each agent that'll be in the model at iteration zero, with each of their internal variables declared
- Easiest to create a Python, C/C++, Java script that accepts a quantity of agents to declare and prints them all to 0.xml

FLAMEGPU Initial States File

```
<!-- 0.xml -->
<states>
<itno>0</itno>  <!-- iteration number -->
   <environment>
      <!-- constants go here -->
   </environment>
   <xagent>
      <name>turtle</name>
      <id>0</id>   <!-- makes this file easier to read -->
      <var>4.0</var>
   </xagent>
   <!-- … same agent code would repeat for id = 0 to 255 -->
</states>
```

Figure 8

Input Types

NetLogo:
1. NetLogo model file in Logo-style syntax
   - Initial and exit functions
   - Agent variables and functions
2. XML-formatted experiment setups inside the model file
   - Can be created through the application's UI or manually written

FLAMEGPU:
1. XML model file
   - Specifying environment, agents, messages, and layers (function dependencies)
2. C functions file with function definitions
3. XML initial states file
   - Define environment constants and initial configuration of agents
   - One per combination of inputs (no parameter sweeping)

Output Types

NetLogo:
1. Using headless mode: CSV spreadsheet or table showing specific data from all iterations

FLAMEGPU:
1. Using console mode: XML output files
   - 1 per iteration
   - Same format as the XML initial states file
   - Need to be parsed by another program to show specifics

FLAMEGPU XML Model File

```
<?xml version="1.0" encoding="utf-8"?>
<!-- XMLModelFile.xml -->
<gpu:xmodel xmlns:gpu="http://www.dcs.shef.ac.uk/~paul/XMMLGPU"
xmlns="http://www.dcs.shef.ac.uk/~paul/XMML">
   <name>Useless</name>
   <gpu:environment>
      <!-- no environmental constants needed -->
      <gpu:functionFiles>
         <file>functions.c</file>
      </gpu:functionFiles>
   </gpu:environment>
   <xagents>
      <gpu:xagent>
         <name>turtle</name>
         <memory>
            <gpu:variable> <!-- how to declare a variable -->
               <type>int</type><name>id</name>
            </gpu:variable>
            <gpu:variable>
               <type>float</type><name>var</name>
            </gpu:variable>
         </memory>
         <functions>
            <gpu:function>
               <name>do_stuff</name>
               <currentState>default</currentState>
               <nextState>default</nextState>
               <gpu:reallocate>false</gpu:reallocate>
                  <gpu:RNG>true</gpu:RNG>
            </gpu:function>
         </functions>
         <states>
            <gpu:state><name>default</name></gpu:state>
            <initialState>default</initialState>
         </states>
         <gpu:type>continuous</gpu:type>
         <gpu:bufferSize>65536</gpu:bufferSize>
      </gpu:xagent>
   </xagents>
   <messages>
      <!-- agents won't be communicating with each other -->
   </messages>
   <layers>
      <layer>
         <gpu:layerFunction>
            <name>do_stuff</name>
         </gpu:layerFunction>
      </layer>
   </layers>
</gpu:xmodel>
```

Figure 6

FLAMEGPU C Functions File

```
// functions.c
#ifndef _FLAMEGPU_FUNCTIONS
#define _FLAMEGPU_FUNCTIONS

#include "header.h" // header.h dynamically generated

__FLAME_GPU_FUNC__ int do_stuff(xmachine_memory_turtle* turtle,
RNG_rand48* rand48) {
   turtle->var = powf(turtle->var, 4.0); // cuda syntax
   turtle->var = powf(turtle->var, 0.25);
   turtle->var += rnd<CONTINUOUS>(rand48); // flamegpu syntax
   turtle->var -= rnd<CONTINUOUS>(rand48);
   return 0;
}

#endif // _FLAMEGPU_FUNCTIONS
```

Figure 7

Conclusion

This project's goal was to find a way to move agent-based modelling applications from a CPU platform to a faster GPU one. This was done by determining a manner of converting NetLogo code (CPU-based ABM application) to the FLAMEGPU syntax and format (GPU-based). The run-time comparison of separate implementations of the Boids simulation demonstrate the GPU's acceleration and overall faster speed, especially for a model in which simple agents are easily parallelized.

Most code such as agent functions and agent variables can be easily transferred from NetLogo to FLAMEGPU, but the latter considers properties like whether an agent is abstract, in a 3D environment, or is it discrete (cellular). Custom random number generators, message partitioning (which isn't shown in the example code), and agent states don't apply to NetLogo code and must be determined by the user during the conversion process. Although the packages don't perfectly match, porting from one to the other is doable. There was not enough time to explore other capabilities such as graph inputs. Conversion software will be worked on in the near future.

Works Cited

Richmond, P., D. Walker, S. Coakley, and D. Romano. "High Performance Cellular Level Agent-based Simulation with FLAME for the GPU." Briefings in Bioinformatics 11, no. 3 (2010): 334-47.
Wilensky, U. 1999. NetLogo. http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.
Wong, Timm. Boids. September 2008. Accessed July 19, 2019. https://cs.stanford.edu/people/eroberts/courses/soco/projects/2008-09/modeling-natural-systems/boids.html.