From Biological Cells to Populations of Individuals: Complex Systems Simulations with CUDA (S5133)

Dr Paul Richmond
Research Fellow
University of Sheffield (NVIDIA CUDA Research Centre)
Overview

• Complex Systems
• A Framework for Modelling Agents
• Degrees of Parallelisation
• Agent Communication
• Putting it all together
Complex Systems

- Many *individuals*
- *Interact* and behave according to simple rules
- System level behaviour *emerges*
Agent Based Modelling

• A method for specification and simulation of a complex system
  • Model is a set of autonomous communicating agents

• Simulation helps to understand complex systems
  • Interventions and prediction

• Presents a computational challenge!
  • Especially for real time or faster
Difficulties in Applying GPUs

• Agents are heterogeneous
  i.e. They diverge
• Agents are born and agents die
  Leads to sparse populations and non coalesced access
• Agents communicate
  No global mechanism for GPU thread communication
• Agents don't stay still
  Acceleration structures used for simulation need to be rebuilt
• Complex Systems
• A Framework for Modelling Agents
• Degrees of Parallelisation
• Agent Communication
• Putting it all together
A Formal Model of an Agent

• Abstract the underlying architecture
  • Let modellers write models not parallel programs
• Describe agents as a form of state machine (X-Machine)
  • Minimises divergence
• Describe state transition functions (agent functions) using high level script
• Describe communication as message dependencies between agent functions
  • Results in Directed Acyclic Graph
  • Identifies synchronisation points for scheduling
FLAME GPU: A Code Generation Framework

• XML Model File
  • Describe Agents and Communication (messages) as a model in XML

• XSLT Templates
  • Code generate a simulation API from agent descriptions

• Scripted Behaviour
  • Scripted behaviour links with dynamic simulation API

• Simulation Program
  • Loads initial data and provides I/O or interactive visualisation
Code Generation using XSLT

- Powerful technique for code generation from Declarative XML model
- Full functional programming language

```xml
<xagents>
  <gpu:xagent>
    <name>Circle</name>
    <memory>
      <gpu:variable>
        <type>int</type>
        <name>id</name>
      </gpu:variable>
      <gpu:variable>
        <type>float</type>
        <name>x</name>
      </gpu:variable>
      <gpu:variable>
        <type>float</type>
        <name>y</name>
      </gpu:variable>
      <gpu:variable>
        <type>float</type>
        <name>z</name>
      </gpu:variable>
      <gpu:variable>
        <type>float</type>
        <name>fx</name>
      </gpu:variable>
      <gpu:variable>
        <type>float</type>
        <name>fy</name>
      </gpu:variable>
    </memory>
  </gpu:xagent>
</xagents>
```

```c
struct __align__(16) xmachine_memory_Circle {
  int id;
  float x;
  float y;
  float z;
  float fx;
  float fy;
};
```

GTC 2015
Mapping an Agent to the GPU

- Each agent function corresponds to a single GPU kernel
  - Each CUDA thread represents a single agent instance
- Agent functions use a dynamically generated API
- Agent Data is transparently loaded from Structures of arrays

```c
typedef struct agent{
  float x[N];
  float y[N];
} xm_memory_agent_list;
```

```c
typedef struct agent_list{
  float x[N];
  float y[N];
} xm_memory_agent_list;
```

```c
__FLAME_GPU_FUNC__ int read_locations(
  xmachine_memory_bird* xmemory,
  xmachine_message_location_list* location_messages)
{
  /* Get the first message */
  xmachine_message_location* location_message =
    get_first_location_message(location_messages);

  /* Repeat until there are no more messages */
  while(location_message)
  {
    /* Process the message */
    if distance_check(xmemory, location_message)
    {
      updateSteerVelocity(xmemory, location_message);
    }

    /* Get the next message */
    location_message =
      get_next_location_message(location_message,
                                location_messages);
  }

  /* Update any other xmemory variables */
  xmemory->x += xmemory->vel_x*TIME_STEP;
...
  return 0;
}```

GTC 2015
• Complex Systems
• A Framework for Modelling Agents
• Degrees of Parallelisation
• Agent Communication
• Putting it all together
Agent Divergence and Sparsity

- **Divergence**: Must group agents (threads)
  - Good News: Agents are already grouped by state
  - Bad News: Agents change states so we are left with sparse lists

- Avoid Sparse Lists by using parallel compaction.
  - Thrust C++ library
Parallelism within the model

- Behaviour consists of function layers
  - Each layer is a synchronisation barrier
- Synchronisation between agents only required when a dependency exists (communication or agent memory)
- This creates parallelism within the function layers of the model
- CUDA Streams can be used to execute independent functions
High Divergence Example

- Single agent ‘cell’ type
  - 5 types of cell within
  - Single message type
- Advantages
  - Large population counts (good utilisation)
  - Simple modelling (but complicated agent transition functions)
- Disadvantages
  - **Lots** of code divergence
  - Unnecessary message reading
Low Divergence Example

- Multiple agent types
  - Different agent type for each cell type
  - Distinction between message

- Advantages
  - Less divergent code
  - More parallelism within the model
  - Less message reading

- Disadvantages
  - Complex dependencies
  - More complex (looking) model
  - Smaller population sizes
Parallelism within the model - performance

Average iteration time of cell behaviour

Simulation Speedup

- High Divergence
- Low Divergence

GTC 2015
• Complex Systems
• A Framework for Modelling Agents
• Degrees of Parallelisation
• Agent Communication
• Putting it all together
Agent Communication

• Brute Force Messaging (N-Body problem)
  • Tile Messages into shared memory

• Spatially Distributed Agents
  • Build data structure to bin agents
  • CUDA Particles
  • Use counting sort to improve performance

• Discrete Space Limited Range (Cellular Automaton)
  • Cache results via texture cache (good locality)
Spatially Distributed Communication

**Radix Sorting**
- Hash Message
- Sort using Thrust (Sort by Key)
  - sort keys
- Reorder
  - scatter messages
  - build partition matrix

**Count Sort**
- Hash Message
- Prefix Sum
  - global index of bins
- Reorder
  - scatter messages

---

GTC 2015
Counting Sort Performance Study

Sorting Performance (1M elements)

- **Tesla K20**
- **Tesla K40**
- **GTX 980**

- **Time (ms)**
- **Element range**

**Thrust Sort**

**Counting Sort**
• Counting sort best suited to smaller population sizes
• Message reading is the bottleneck
Spatially Distributed Communication Benchmark

27k faster than FLAME on CPU with 50k agents (*apples != oranges*)
700x faster than FLAME II with 50k agents on 16 cores (using MPI, vector splitting)
• Complex Systems
• A Framework for Modelling Agents
• Degrees of Parallelisation
• Agent Communication

• Putting it all together
Pedestrian Dynamics

• Pedestrian agents
  • Social Repulsion (Social Forces)
  • Reynolds steering forces
  • Reciprocal Velocity Obstacles

• Navigation agents
  • Global Vector Field
  • Navigation Graph
  • Environment and Goals are calculated as a weighted influence

• An extension: Navigation graphs
Conclusions

• Agent based modelling can be used to represent complex systems at differing biological scales
• FLAME GPU is a framework for model description and CUDA code generation
• Using state based representation avoids divergence and allows parallelism within a model to be exploited
• Counting sort helpful for highly divergent population
• Visualisation is extremely cheap
Thank You

Get the code for free from:

http://www.flamegpu.com
www.github.com/FLAMEGPU

Contact Me:

p.richmond@sheffield.ac.uk
http://www.paulrichmond.staff.shef.ac.uk

Please complete the Presenter Evaluation sent to you by email or through the GTC Mobile App. Your feedback is important!