Implementing An Agent-Based Model In OpenCL

Gregory J. Davis - University of Notre Dame
Klaus Kofler - University of Innsbruck
Overview

• AGiLESim - Model for investigating mosquito lifecycle

• OpenCL - programming model for general purpose GPU computing

• Challenges and solutions

• Model verification

• Performance comparisons
AGiLESim - Modeling Anopheles gambiae Populations

- *Anopheles gambiae* - vector of malaria in sub-Saharan Africa

- Simulates individual mosquitoes’ lifecycle

- Look at population level response to external factors
  - Weather
  - Vector-targeted interventions

Image source: VectorBase (http://www.vectorbase.org)
AGiLESim - Modeling Anopheles gambiae Populations

• Implemented in both Java & C++ (independent implementations)

• Takes advantage of multiple cores (threaded/OpenMP) and multiple machines (MPI for C++ version)

• Work requires many MANY simulation runs

• Most time spent iterating over lists of agents

• Candidate for parallelization and GPU computing
GPU Computing

CPU (Multiple Cores)
- Core 1
- Core 2
- Core 3
- Core 4
- Cache
- System Memory

GPU (Hundreds of Cores)
- Device Memory

Thousands
OpenCL

- Based on C99 language specification
- Supported by many hardware vendors
  - Altera, AMD, ARM, Intel, NVIDIA, ...
- Support for many architectures
  - CPU, GPU, FPGA, Accelerator
- Requires different programming model!
Split Architecture

- Host code (CPU) and device code (GPU)

- Host:
  - set up environment/allocate memory on device
  - transfer memory to/from device

- Device:
  - perform operations on memory (kernel function)
Challenges

• Device code is non-object-oriented
• Device code cannot allocate/reshape memory
• Memory transfers occur over PCI bus
• No standard libraries
  • i.e. no random()
Memory Solution

- Agents are structured memory allocated in buffers on device

- Fixed size agent array (double buffered)
  - Large enough to hold maximum number of agents

- Agents in same state are packed together
  - Padded to wavefront size (minimize divergent branching)

- Copied from one buffer to another on each iteration
  - Uses parallel prefix scan at each iteration of the simulation
Memory Solution

- All agents reside only in device memory
- no copying back to host
- summary statistics computed by parallel reduction on GPU
- Only random number seeds and summary statistics are communicated between host and device
Random Number Solution

- Host generates random number seeds
  - 4 seeds (32-bit integers)
  - Use agent’s position in memory to mangle seed
  - Feed through Tausworthe algorithm (x3) and then LCG
    - Gives a period of $2^{121}$
Model Verification

- Overall adult population numbers

![Graph showing overall adult population numbers over days with two lines representing SAMPO and AGiLESim.](image)
Model Verification

- Hypothetical vs. observed age distribution
Performance Comparison

- Simulate 1 year in 1 hour time steps (8760 iterations)
- Baseline
  - Java version
  - Intel Xeon X5650 (6 cores @2.66 GHz)
  - 16 GB ram
Performance Comparison
CPU vs. Multiple Devices

Population Size
80K
700K
5500K

Speedup vs. Java

- Opteron 6168
- Xeon X5650
- Radeon HD5870
- FirePro S9000
- GeForce GTX480
- Tesla k20c
Memory Comparison

<table>
<thead>
<tr>
<th></th>
<th>Device</th>
<th>Host</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>5500</td>
<td></td>
</tr>
<tr>
<td>CPU</td>
<td>1375</td>
<td>0</td>
</tr>
<tr>
<td>NVIDIA GPU</td>
<td>2750</td>
<td></td>
</tr>
<tr>
<td>AMD GPU</td>
<td>4125</td>
<td></td>
</tr>
</tbody>
</table>
Summary

• Simulation of ABM *can* benefit from GPU computing paradigm
  • Requires workarounds for limitations
  • in turn - specialized programming skills/knowledge
• Speed-up partially dependent on # of cores, but also memory usage
  • Nowhere near theoretical speedup :)
• Less flexible in terms of development
  • small changes can have severe impact on performance and other code
• Download it - play with it - IMPROVE IT!
  • [https://github.com/klois/SAMPO](https://github.com/klois/SAMPO)
Thanks!