Model of Evolutionary Game Dynamics Accelerated Using GPU Hardware

Amanda Peters

School of Engineering and Applied Sciences, Harvard University, Cambridge, MA

Introduction

This work demonstrates the results of an effort to parallelize, using GPUs, an important algorithm for simulation of evolutionary processes. **Natural selection dictates survival of the fittest and is thought to promote the emergence of selfish strategies over altruistic ones.** However, we can empirically see many examples of cooperative behavior in biology, from interactions of humans to viruses to bacteria.

This project utilizes the Maynard Smith and Price model to analyze the emergence of cooperation in conflicts during varying behavioral strategies. The scalable parallel algorithm presented here can extend this model to investigate direct and indirect reciprocity, punishment, or kin selection, all key elements behavioral evolution.

Speeding up of the application is required due to the complexity of the problem as these factors are considered. This poster shows the techniques used to accelerate this model via the CUDA programming model on GPU hardware.

The techniques used focused:

- Restructuring of the code to leverage the SIMT nature of the hardware
- Reduction any unnecessary communication between threads
- Memory Bandwidth Optimization

Model

Four key components:

- Player
- Behavioral Strategy
- Round
- Game

The goal of the application is to analyze the conflict between every behavioral strategy with every other behavioral strategy to determine if a dominant strategy arises.

Serial Pseudo Code:

```
Foreach strategy
Foreach strategy
        Game {
                For rounds < max_rounds
                        play;
        }
Process Results
```

CUDA Port

- As new strategies are added and more factors are taken into account, the number of behavioral strategies increases dramatically leading to an exponential growth of the problem space.
- Multilevel parallelism inherent to the algorithm
  - Games
  - Rounds
- Optimizations:
  - Phase 1: GPU kernel call per game
  - Phase 2: Multiple Block Implementation
  - Phase 3: All Data processing on the Device
  - Phase 4: Fine Tuning

Phase 1

- Map problem to CUDA programming model

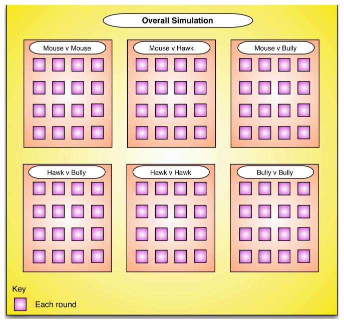

Figure 1. Mapping of problem to CUDA model.

- Inlined kernels for strategies, rounds, and game
- Separate kernel call for each game with parameters to set player's strategies
- Reduce memory bank conflicts through coalesced memory accesses
- Only the results are transferred between host and device for post processing

Phase 2

Leverage full grid:

- Each block computes a different game
- Results of the full game space stored in global memory

Input determined by built-in variables

Phase 3

- Data post-processing is handled within the block
- Reduce memory transfers by only returning the final result per behavioral matchup
- Multiple games per block
- Reduced calls to sync threads

Code Layout:

CPU Code:

```
        Define variables: max_rounds and number of players
        Allocate memory on host and device
        Call gameGPU
        Copy memory from device to host
        Display results
        Free memory
```

GPU Kernels:

```
gameGPU
        Determine player strategies based on blockID
        Allocate shared memory to cache payoff results for all rounds
        Have each thread calculate one round result for all rounds
                call inlined function playRound
        Sync Threads
        Average results for the block
        Return results for that game

playRound
        Until max number of rounds or serious injury incurred play out
                interaction based on players' defined strategies
        Return final payoffs to each player
```

Phase 4

- Reduce the number of games simulated per block to maximize system usage
- Fine tune code
  - Reduce clock cycles through arithmetic instruction optimizations
  - Unroll loops

Results

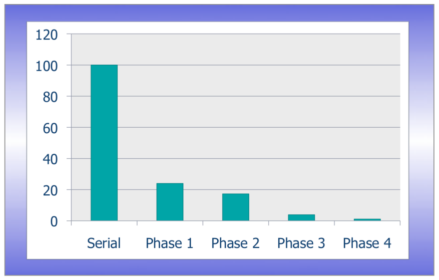

Figure 2. Overall time reduction shown as a percentage of the optimized serial code.

Increase Strategy Space

The parallel implementation allowed a drastically increased strategy space to be explored.

A speedup of 98x was achieved on the Tesla T10 GPU.

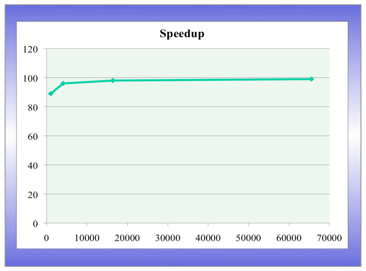

Figure 3. Speedup as compared to optimized CPU code

Future Work

The framework produced was left intentionally generic to allow for adaption in a varieties of models. The next step is to modify the code to show the evolution of strategy adoption in a well dispersed population over time.

This will allow studies of the evolution of cooperation in:

- Large scale social networks
- Quorum sensing bacteria
- Relay nodes in wireless networks

I would also intend to scale this work to run multiple GPU clusters therefore enabling a more exhaustive study of the strategy space and the role of other variables like reciprocity and punishment.

References

1. Nowak MA, K Sigmund (1993). Chaos and the evolution of cooperation. P Natl Acad Sci USA90: 5091-5094.
2. "Evolutionary Stable Strategy." Wikipedia: The Free Encyclopedia. 03 2009. Wikipedia. 7 May 2009 < http://en.wikipedia.org/wiki/Evolutionarily_stable_strategy >.
3. NVIDIA CUDAProgramming Guide.://developer.download.nvidia.com/compute/CUDA/1_0/NVIDIA_CUDA_Programming_Guide_1.0.pdf
4. Axelrod, R. & Hamilton, W. D. The evolution of cooperation. Science 211, 1390 (1981).

Acknowledgments

I thank Dr. Hanspeter Pfister and Dr. Won-Ki Jeong for their help and assistance to present this work, as well as Dr. Efthimios Kaxiras for his continued assistance and support.