CS 441
Project II Rough Draft
Paul Gentemann
Bucky Frost
2014-12-02

Proposal

The project shall be the implementation of a checkers artificial intelligence (A.I.) training program on GPU devices using the Cuda language. A stretch goal for this project is to run a version of the CUDA-optimized code on an Amazon EC2 G2 GPU server cluster, and benchmark scale and speed to determine when network-based GPU computations start to become worthwhile options.

Code Base

The A.I. program that will be repurposed from a personal project based off CS 405. The base code repository is https://github.com/arcshock/Checkers_AI.

Code Organization

As of Git commit 883e954315db1dfb119c48f9cda5307374cd07b4

Catch: Git submodule for the Catch C++ testing framework
include: header files of the project
├── checker_board.hpp : definition of the board
├── game.hpp : class manager for games
├── game_piece.hpp : rule definitions of game pieces, kings and pawn currently
├── minimax.hpp : definition of the minimax algorithm
├── network_node.hpp : defines low level functions of network such as modifying weights
├── neural_network.hpp : defines the actual network topography composed of network nodes
├── player.hpp : defines a wrapper class for interactions of games
└── tournament.hpp : will define types of tournaments of games
Makefile: makefile for the project compilation
src: main source code directory for the checkers AI build
└── main.cpp
test: directory for the unit tests of the project
└── tests.cpp : unit tests of project

Requirements

The program implementation will focus on the training aspects of the neural networks (NN). The program will send a number of game instances of the to GPU device for tournaments. From there, the NN will be sorted for continual generations of NN or be removed from the NN pool.

Constraints

For interest of time restrictions, the parallelism aspect of training the A.I. will focus on creating high level parallelism of training versus exploiting aspects of the GPU for implementation tuning of the code base of the A.I.

Timeline

2014-12-02: Rough Draft
2014-12-04: Presentation Lecture Notes
2014-12-06: Working Prototype
2014-12-08: Timing Analysis Complete
2014-12-09: Class Presentation
2014-12-18: Final Draft

Progress

At the beginning of the project, we shared Bucky's codebase, so that we could begin discussing which files and functions would need to be converted into CUDA. The discussion led to a decision to try two separate approaches: having the trainer create a vector containing large quantities of games, which will be run in the GPU, and rewriting the NN's sigmoid function to also run on the GPU to take advantage of GPU trigonometry functions. Also, for the modifications of the NN weights, the randomization functions provided by CUDA could be utilized.

Now that we know what we hope to accomplish, we perused the Nvidia CUDA website in search of installation instructions and tutorials. Paul subscribed to the udacity parallel processing class to access course materials. We installed and ran the sample CUDA programs so that we could view more working examples.

Paul's laptop is apparently running hardware (Tesla architecture) that is in the process of being phased out; because the cuda compiler warnings continue to state that 'compute_11', ‘compute_12’, ‘compute_13’, ‘sm_11’, ‘sm_12’, and ‘sm_13’ are deprecated and may be removed in a future release.

Another point of discussion was how exactly we intended to perform timing analysis. The primary tools available are the *cpu_timer* class from the boost library, running code through NetRun and using the built-in functions, the built-in shell function *times*, the profiling tool *Gprof*, and using c++11 standard for high resolution timer.

Sources

https://aws.amazon.com/ec2/instance-types/
http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-linux/index.html
http://www.boost.org/doc/libs/1_53_0/libs/timer/doc/cpu_timers.html
http://linux.die.net/man/2/times
http://www.arc.vt.edu/userinfo/training/2012Fall/2012Fall_Hybrid_CUDA.pdf