UiT

THE ARCTIC
UNIVERSITY
OF NORWAY

INF-3201 – Assignment 3 - GPGPU

Frode Opdahl
opdahl.frode@gmail.com

Assignment outline

- Parallelize the given piece of code with CUDA/OpenCL and multiprocessing (combined)

Environment

- Your own computer
- The lab computers
  - “ssh abc123@ifilab87.stud.cs.uit.no”

- UVRocks frontend (only a few computers, and older hardware)
- Precode in Python
  - But feel free to solve it using whichever language you prefer
  - There is also a more complex precode in C for those that prefer working more directly with CUDA
    - Recommended for those that feel comfortable with C
    - Note: This might not be available straight away

Example programs

- Included in the handout (src/hello_mandelbrot/):
- Example of using CUDA C (cuda_kernel.cu)
- Example of using PyCUDA to use the CUDA C kernel
- Example of a Python extension written in C
- Example of how to use a C library in Python

- Requirements:
  - Numpy
  - Matplotlib (if you want to look at the images)
  - Pycuda
  - Gcc
  - CUDA Toolkit
  - The ifilab computers should have all of this ready

Precode

- The precode consists of code to encode and decode messages using XTEA and CBC, as well as a simple mangling algorithm
  - Frontends (encrypt.py and decrypt.py)
  - Main code (precode.py)
- Also has code to attempt to guess the password given a known part of the plaintext

Encoding

- Each block of decoded data consists of two data points
  - Each data point consists of 24 bits location information and 8 bits of data
- final_result[datapoint & location_mask] = datapoint & data_mask
- The final result is random bits, with the secret embedded in a random location
- Each block of the final result (64 bits) encoded with XTEA and CBC
  - Very simple encryption algorithm

Decoding

- Decoding is the same, but in reverse order
- When we don’t have a password, we guess by iterating through the printable characters for several levels
  - The number of possible passwords grows incredibly fast
- To test if a password is correct, the data is decoded with the guessed password, then we try to unmangle it
- If we find something we know is in the plaintext in the decoded data, we assume the password is correct

Additional details about the precode

- Four encrypted files included (easy.npy, medium.npy, hard.npy, veryhard.npy)
  - If implementing using the C precode, use the “easy_bin, medium_bin, …)
- All have a secret text which includes the word <<Secret>>
  - Uppercase S
- A very simple GPU implementation decrypts the hard.npy file in ~7 seconds
  - The precode decrypts the easy.npy file in ~26 seconds

If you want to implement it in C

- Use nvcc to compile (included in the CUDA toolkit)
  - Precode comes with a simple makefile
- http://docs.nvidia.com/cuda/

- Alternatively: OpenCL
- https://www.khronos.org/opencl/
- https://software.intel.com/en-us/intel-opencl

Assessment criteria

- Your solution
  - Does it fulfill the requirements?
  - Is it well-commented and understandable?
- Your report
  - Have you critically evaluated your solution?
  - Have you adequately explained your solution?
  - Have you made your assumptions clear and separate from your measurements?

Requirements

- Parallelize the pre-code using GPU
- Analyze your solution using the Nvidia Visual Profiler or equivalent for other GPUs
- Not required but for the best possible speedup also parallelize using the CPU, combined with the GPU
  - Hint: Only one thread/core is needed to communicate with the GPU
  - Rest are just idle

Practical details

- Report
  - Short report describing your algorithms and results, as well as reasons for choosing this approach
  - A critical review of your achieved performance (this requires profiling and/or tracing)
  - Accompanying code should be commented
- I am available by e-mail (opdahl.frode@gmail.com) at most hours

Report

- Describe how you have approached the problem (e.g. «profiling shows that functionX() is expensive, so I have paralellized this», «tracing shows that the workload goes from X to Z at this point, which means that the work has to be distributed like so»)
- Try to make assumptions clear, and separate from measurements
- What I want to know:
  - How did you solve the problems?
  - Why did you solve them in this way?
  - Are the results of your approach any good?

Deadline

- Report and code: Friday November 4th

Competition

- Another informal competition!
- Let me know if you want to be excluded from the results
- Same procedure as last time
- I will run the code on the machines similar to the ones at ifilab

Resources

- http://docs.nvidia.com/cuda
- https://www.khronos.org/opencl/resources/opencl-tutorials-white-papers-and-how-to-guides

- http://mathema.tician.de/software/pycuda/
- http://mathema.tician.de/software/pyopencl/