Program Optimization – GPUs

simon.marechal@synacktiv.com

2023/2024

Introduction

History

graphics cards are really good at parallel processing,

- early efforts for general purpose computation with *shaders*
- 2006, CUDA
- 2009, OpenCL
    - also for CPUs, FPGAs, etc.
- 2010, (old) *oclHashcat*, very limited
- 2011, *oclHashcat-plus*
- 2013, *oclHashcat*

When to use GPUs ?

- very parallel workloads (same operations on a large amount of data),
- ideally, cut in a lot of parts,
- high arithmetic density (ratio of calculation over memory operations)

Difference with CPUs

- much, *much* simpler execution units
  - in order execution (except RTX 4xxx series?)
  - no branch prediction
- many threads in parallel (1k / 2k)

OpenCL programming concepts

Platform model

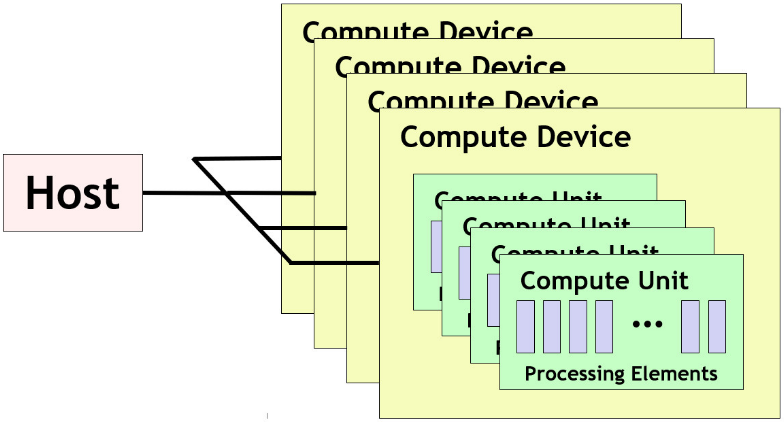

Figure 1: from Khronos Group OpenCL Guide

Execution model

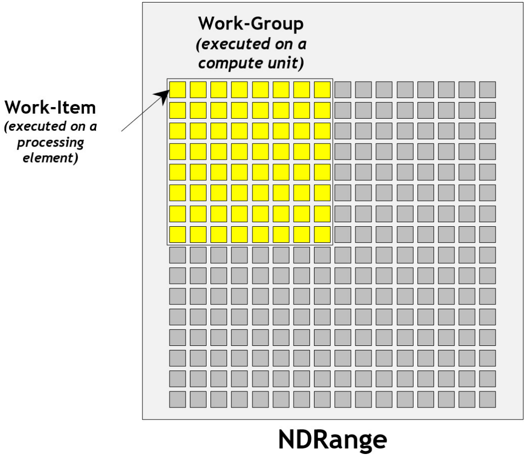
Work-Group
(executed on a compute unit)
Work-Item
(executed on a processing element)
NDRange

Figure 2: from Khronos Group OpenCL Guide

Memory model

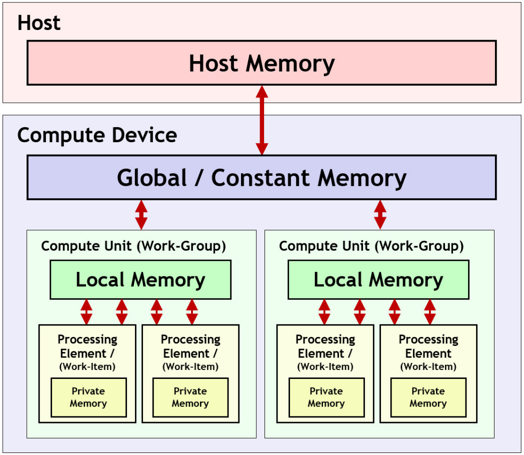
Host
Host Memory
Compute Device
Global / Constant Memory
Compute Unit (Work-Group)
Local Memory
Processing Element / (Work-Item)
Private Memory
Compute Unit (Work-Group)
Local Memory
Processing Element / (Work-Item)
Private Memory
Processing Element (Work-Item)
Private Memory

Figure 3: from Khronos Group OpenCL Guide

OpenCL C

- looks a lot like C, but isn't (we will not study how it differs)
- specific concepts:
  - address spaces, local, global, private, constant
  - work item / work group
  - command queues
- `get_global_id()`

Task - GPU cracker

- you are given an initial C adapter program
- write the `.cl` file!
  - simple cracker
  - optimize it
  - implement better queuing

Why are GPUs fast?

- many, many threads can run on a single processing unit
- they are scheduled so as to mask memory latencies
- this means it is easier to approach peak execution rate for those processors

QUESTIONS?

Thank you for your attention

SYNACKTIV DIGITAL SECURITY