High-performance software - Easy gains with simple CUDA

Bernhard Raml

TU Wien
Department of Geodesy and Geoinformation
Research Area Remote Sensing
bernhard.raml@geo.tuwien.ac.at

Vienna | April 24, 2023

How to write high-performance software

Habitability comes before speed

- A habitable code base means automated tests!
- Design should focus on maintainability first.
- Measure using profiling tools to make informed decisions about *what to optimise*.
- Automated performance tests need a stable environment or cover only the most basic components.

Measure before you act

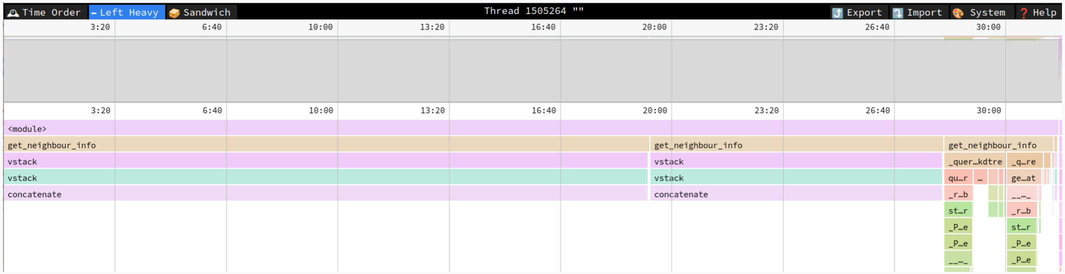


Figure 1: Repeated concatenation slow down due to mem copies

Measure before you act

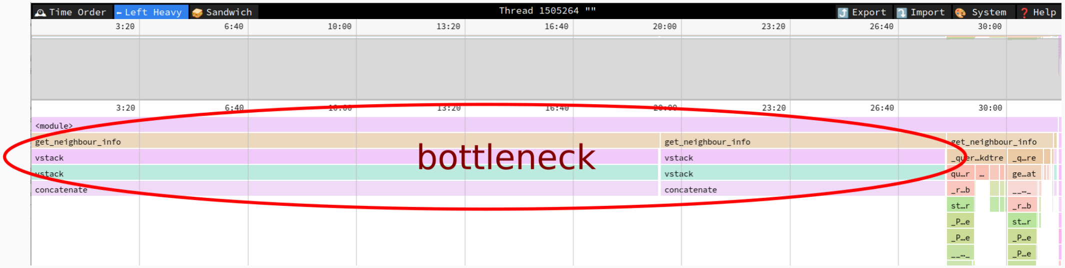


Figure 2: Repeated concatenation slow down due to mem copies

Measure before you act

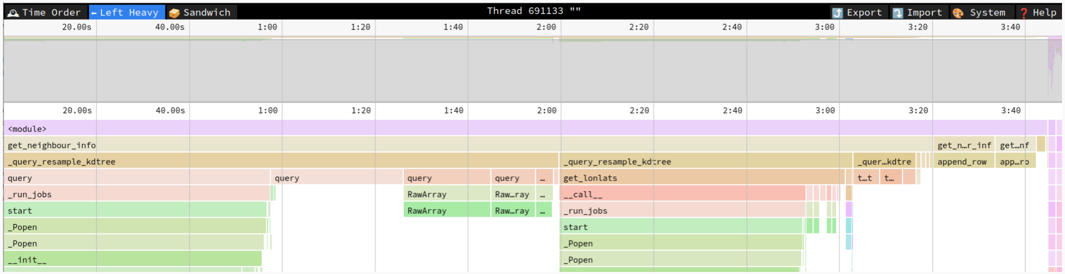


Figure 3: Memory pre-allocation avoids it making index creation dominant

Streaming to increase throughput

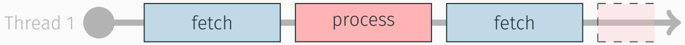


Figure 4: Serial process, waiting for data before processing

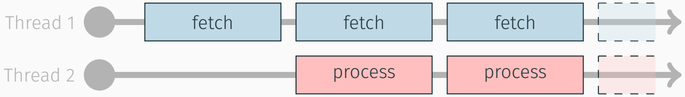


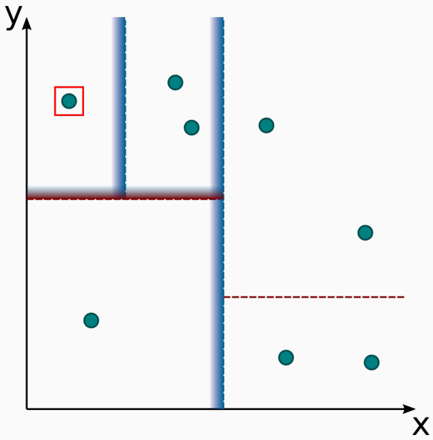
Figure 5: Stream next data-block while processing previous one

GPU vs CPU

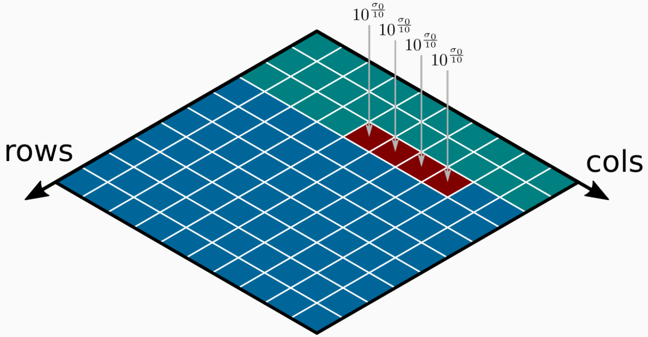
The right tool for the right job

CPU	GPU
Complex control flow	**Embarrassingly parallel tasks**
Tree or graph search, sparse matrix operations	Dense matrix multiplication
Serial processes	**High memory throughput**
IO, compression	Video processing, 3D rendering, high resolution remote sensing
General purpose tasks	**Specialized tasks**
UI, web services, OS	Ray-tracing, video codecs

The right tool for the right job - Examples



CPU - KD tree query


$10^{\frac{\sigma_0}{10}}$
$10^{\frac{\sigma_0}{10}}$
$10^{\frac{\sigma_0}{10}}$
$10^{\frac{\sigma_0}{10}}$
rows
cols

GPU - dB to linear

CPU

- Maximise instructions per cycle
- Low latency of single core
- Deeper cache hierarchy
- Complex instructions

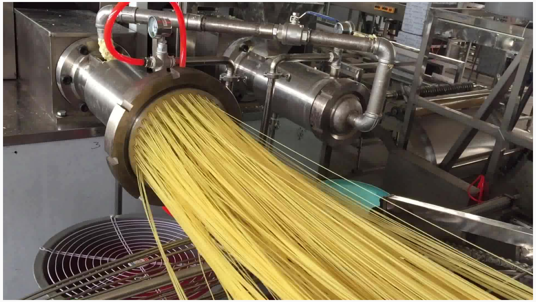
GPU

- Maximise total throughput
- Streamed processes to hide higher latency
- Shallower cache hierarchy
- Simple instructions

The right tool for the right job

CPU

Image: Jorge Zapata - unsplash.com



GPU

Image: pastaproductionline.com

GPU Architecture - Opening the magic box

Hardware Layout CPU vs. GPU

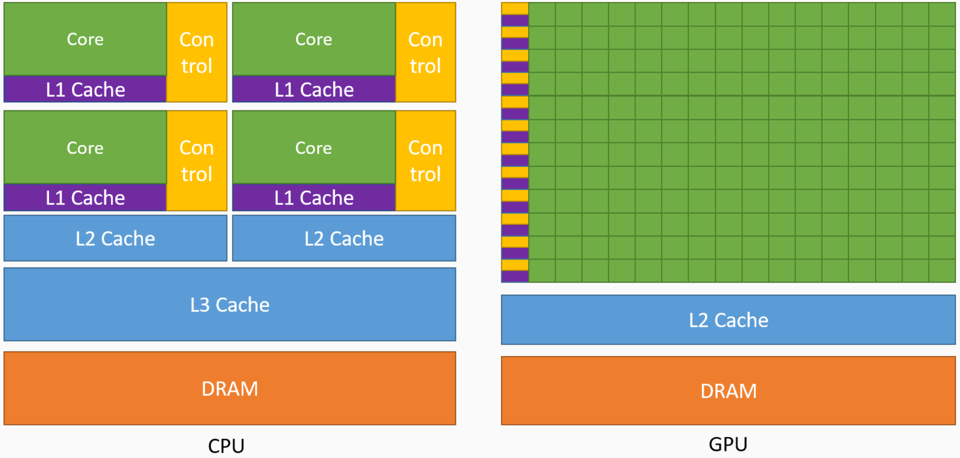
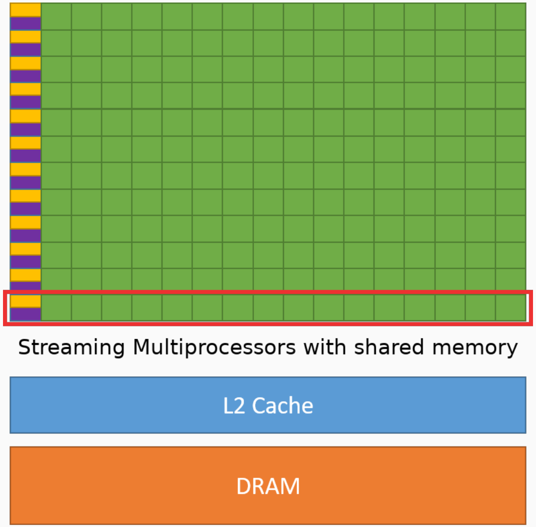


Image: CUDA Programming Guide

Memory latency - One cycle one second metaphor



Register: 1s

Shared Memory: 2s - 4s

L1 Cache: 10s - 30s

L2 Cache: 1min

Device RAM: 2min - 10min

Host to device transfer: hours

Coding Session

Where to go from here?

Some pointers

- Of course, streaming from CPU-RAM to GPU-RAM (*VRAM*) improves throughput as well
- Avoid stalls from branching using clever distributing across *warps* or the step function trick
- Exploit specialised hardware accelerated *intrinsic* functions, e.g., add-mul
- Use different floating point representations like 16-bit halfs
- Look into additional libraries within the CUDA ecosystem, e.g., cuBLAS, cuSolver...

Appendix

Davey Farley’s YouTube Channel *Continuous Delivery* - Hardware cycles:
https://www.youtube.com/watch?v=0reMVgn6kRo

Wong, Henry, et al. ”Demystifying GPU microarchitecture through microbenchmarking.” 2010 IEEE International Symposium on Performance Analysis of Systems Software

CUDA Programming Guide:
https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

Peng Wang’s Fundamental Optimizations in CUDA Presentation:
https://developer.download.nvidia.com/GTC/PDF/1083_Wang.pdf

py-spy: *https://github.com/benfred/py-spy*

PyResample: *https://github.com/pytroll/pyresample*

Python Approval Tests:
https://github.com/approvals/ApprovalTests.Python

Special thanks to Raphael Quast for the Latex template