Overview of OpenCL

OpenCL Resources

- OpenCL v1.2 Reference Card
  - https://www.khronos.org/files/opencl-1-2-quick-reference-card.pdf
- OpenCL C++ Wrapper v1.2 Reference Card
  - https://www.khronos.org/files/OpenCLPP12-reference-card.pdf
- OpenCL v1.2 Specification
  - https://www.khronos.org/registry/OpenCL/specs/opencl-1.2.pdf

It’s a Heterogeneous world

A modern computing platform may include:

- One or more CPUs
- One or more GPUs
- DSP processors
- Accelerators
- FPGAs
- … and more …

E.g. Intel® Core i7-8700K:

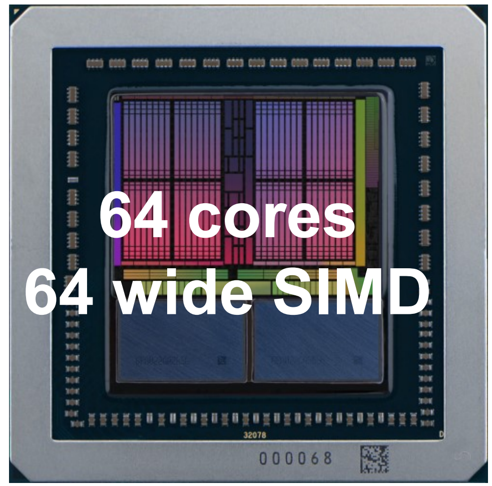
- Six-core Coffee Lake x86 with Intel® UHD Graphics 630

OpenCL lets Programmers write a single portable program that uses ALL resources in the heterogeneous platform

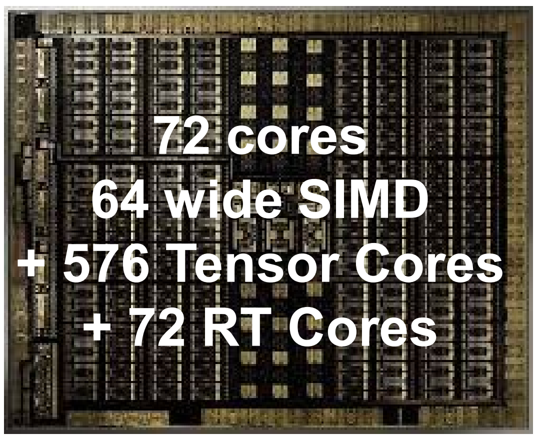
Processor trends

Individual processors have many (possibly heterogeneous) cores.

Intel® Xeon Phi™ (KNL) CPU



AMD® Vega

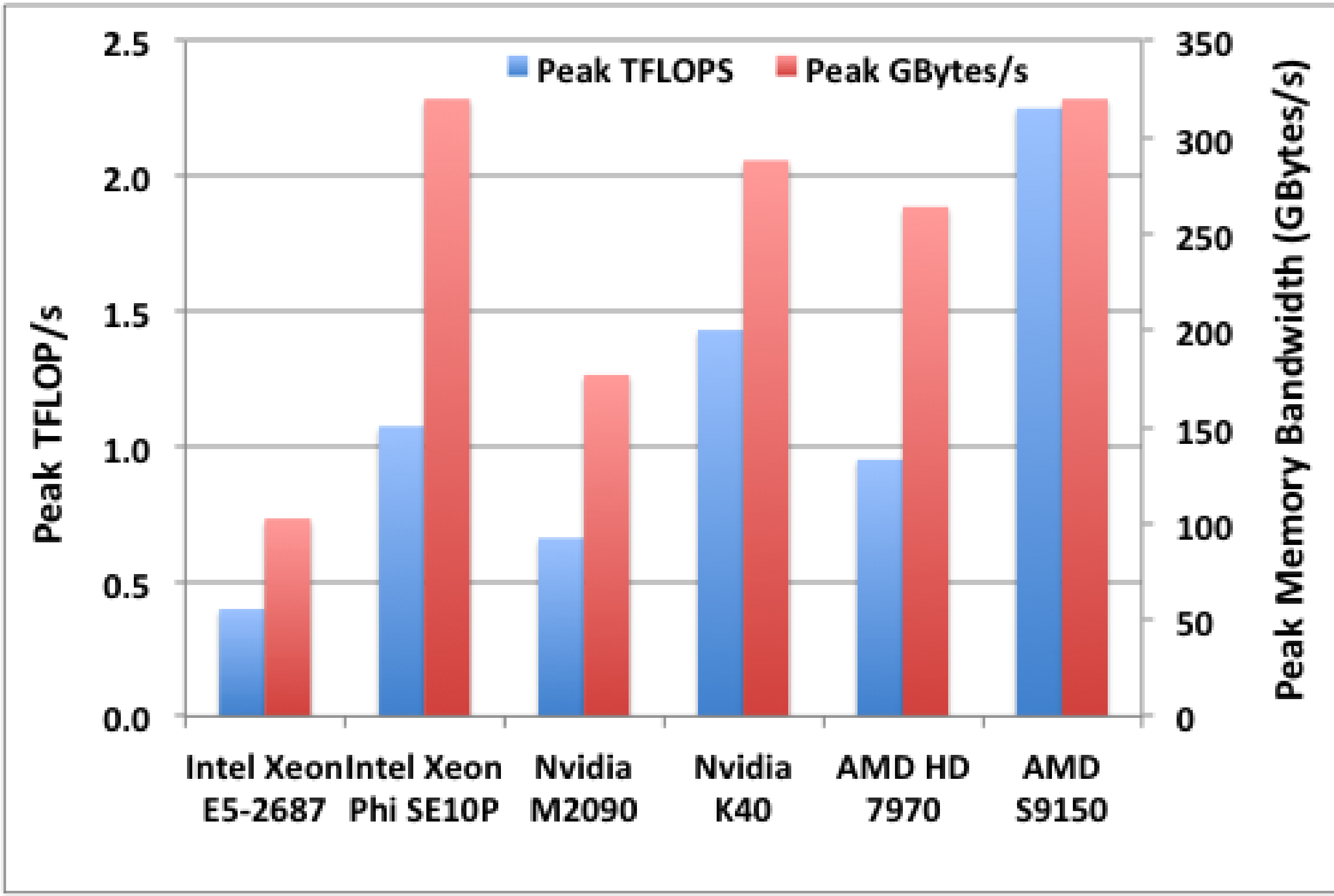


NVIDIA® Turing® RTX 8000

The Heterogeneous many-core challenge:

How are we to build a software ecosystem for the Heterogeneous many core platform?

Many-core performance potential

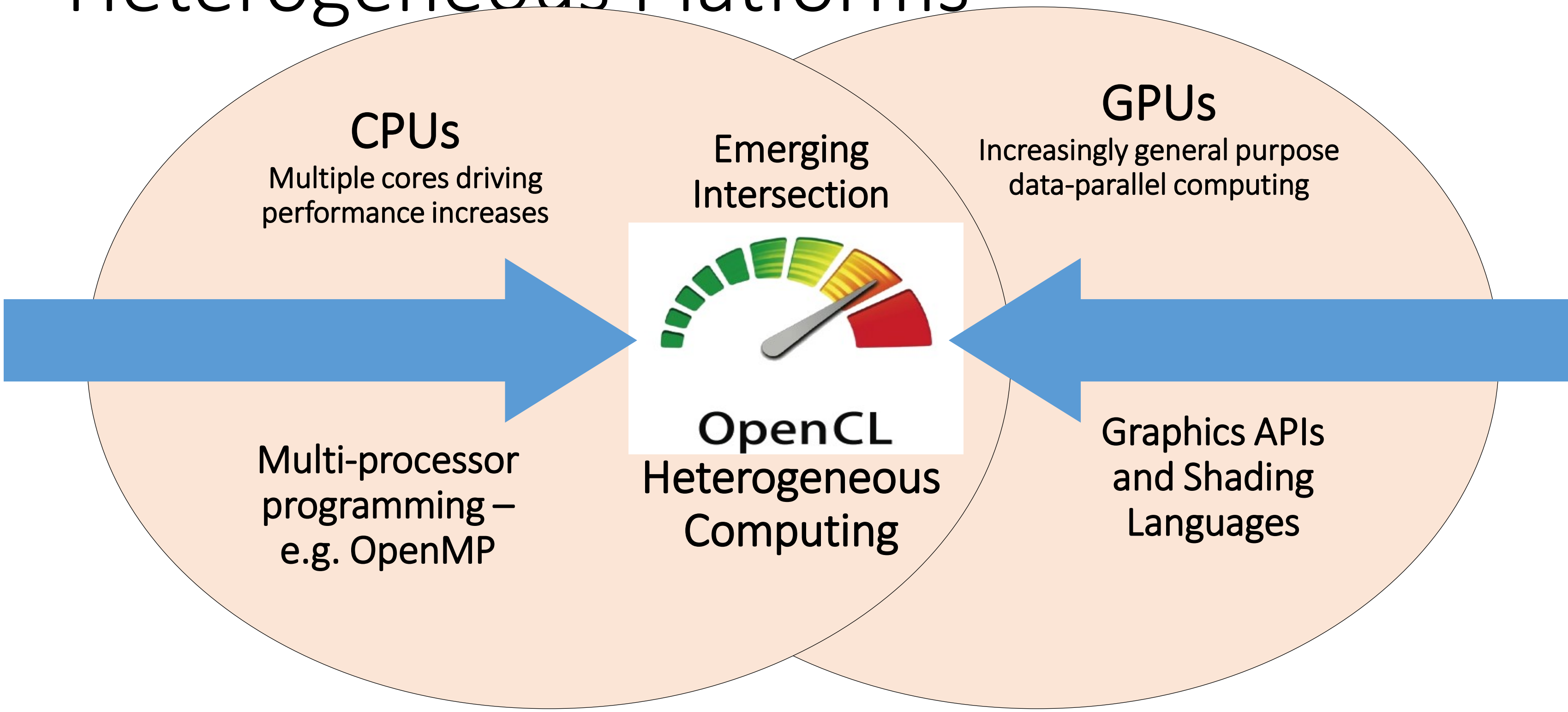


How do we unlock this potential?

- Need efficient, expressive, parallel programming languages
- Also need cross-platform standards
- Ideally not just for HPC so that they have sufficient momentum for the long term

- **OpenCL** is the *only* mainstream parallel programming language that meets all these many-core requirements today

Industry Standards for Programming Heterogeneous Platforms



OpenCL – Open Computing Language

Open, royalty-free standard for portable, parallel programming of heterogeneous parallel computing CPUs, GPUs, and other processors

The origins of OpenCL

AMD

ATI

Merged, needed commonality across products

NVIDIA

GPU vendor – wants to steal market share from CPU

Intel

CPU vendor – wants to steal market share from GPU

Apple

Was tired of recoding for many core, GPUs. Pushed vendors to standardize.

Wrote a rough draft straw man API

Khronos Compute group formed

ARM
Nokia
IBM
Sony
Qualcomm
Imagination
TI
+ many more

OpenCL

OpenCL Working Group within Khronos

- Diverse industry participation
  - Processor vendors, system OEMs, middleware vendors, application developers.
- OpenCL became an important standard upon release by virtue of the market coverage of the companies behind it.

OpenCL 2.2 Released November 2017

- OpenCL first launched Jun’08
- 6 months from “strawman” to OpenCL 1.0
- Rapid innovation to match pace of hardware innovation
    - Committed to backwards compatibility to protect software investments

2011

OpenCL 1.2

Becomes industry baseline for heterogeneous parallel computing

2013

OpenCL 2.0

Enables new class of hardware
SVM
Generic Addresses
On-device dispatch

2015

OpenCL 2.1
SPIR-V 1.0

SPIR-V 1.1 in Core
Kernel Language Flexibility

2017

OpenCL 2.2
SPIR-V 1.2

OpenCL C++ Kernel Language
Static subset of C++14
Templates and Lambdas

SPIR-V 1.2 in Core
OpenCL C++ support

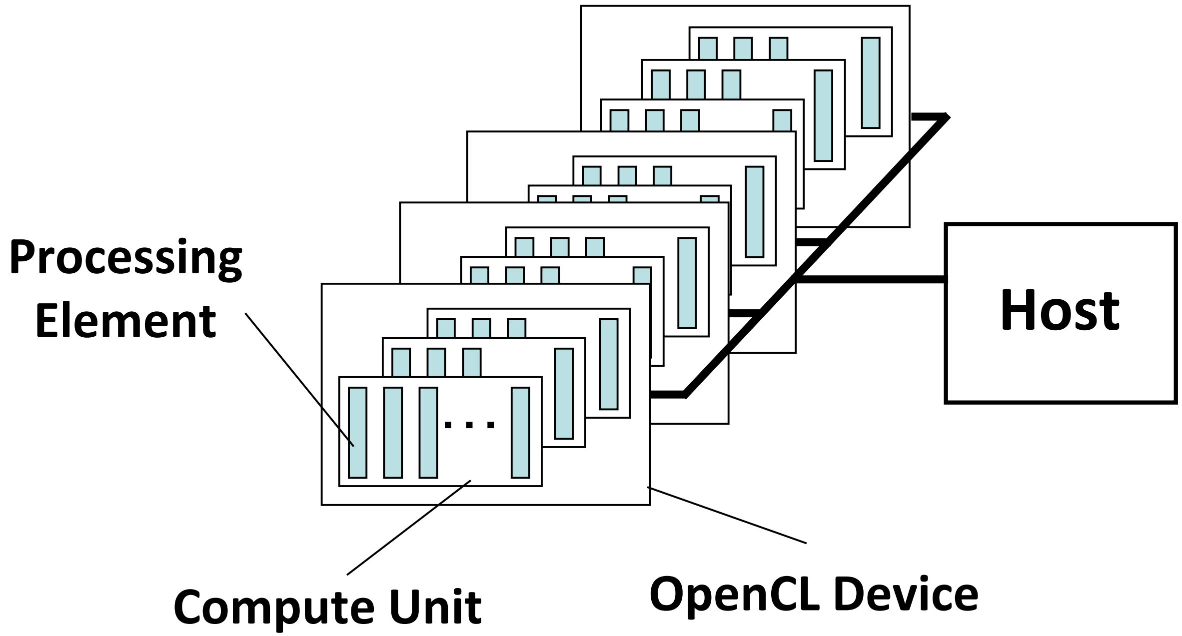
Pipes
Efficient device-scope communication between kernels

OpenCL: From cell phone to supercomputer

- OpenCL Embedded profile for mobile and embedded silicon
  - Relaxes some data type and precision requirements
  - Avoids the need for a separate “ES” specification
- Khronos APIs provide computing support for imaging & graphics
  - Enabling advanced applications in, e.g., Augmented Reality
- OpenCL will enable parallel computing in new markets
  - Mobile phones, cars, avionics

A camera phone with GPS processes images to overlay generated images on surrounding scenery

OpenCL Platform Model



- One ***Host*** and one or more ***OpenCL Devices***
  - Each OpenCL Device is composed of one or more ***Compute Units***
    - Each Compute Unit is divided into one or more ***Processing Elements***
- Memory divided into ***host memory*** and ***device memory***

OpenCL Platform Example (One node, two CPU sockets, two GPUs)

CPUs:

- Treated as one OpenCL device
  - One CU per core
  - 1 PE per CU, or if PEs mapped to SIMD lanes, *n* PEs per CU, where *n* matches the SIMD width
- Remember:
  - the CPU will also have to be its own host!

GPUs:

- Each GPU is a separate OpenCL device
- Can use CPU and all GPU devices concurrently through OpenCL

CU = Compute Unit; PE = Processing Element

The BIG idea behind OpenCL

- Replace loops with functions (a kernel) executing at each point in a problem domain
  - E.g., process an *n* element array with one kernel invocation per element

Traditional loops

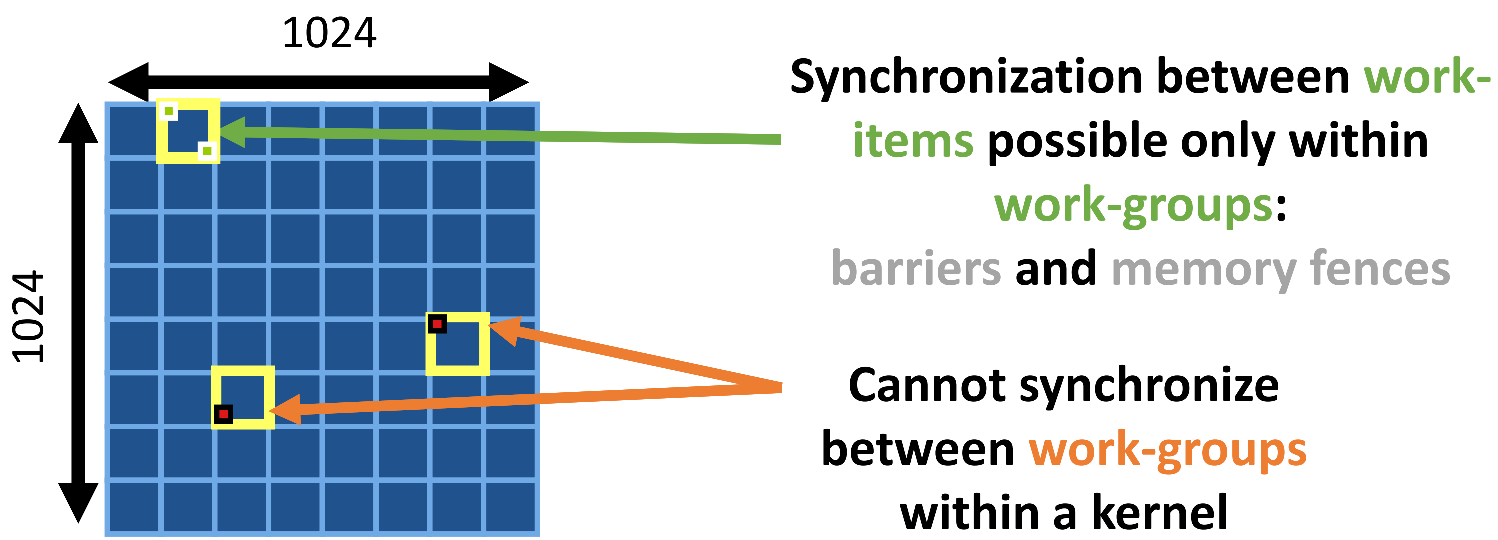
```
void
mul(const int n,
    const float *a,
    const float *b,
          float *c)
{
  int i;
  for (i = 0; i < n; i++)
    c[i] = a[i] * b[i];
}
```

Data Parallel OpenCL

```
__kernel void
mul(__global const float *a,
    __global const float *b,
    __global       float *c)
{
  int i = get_global_id(0);
  c[i] = a[i] * b[i];
}
// many instances of the kernel,
// called work-items, execute
// in parallel
```

An N-dimensional domain of work-items

- Global Dimensions:
  - 1024x1024 (whole problem space)
- Local Dimensions:
  - 128x128 (**work-group**, executes together)



- Choose the dimensions that are "best" for your algorithm

OpenCL N Dimensional Range (NDRange)

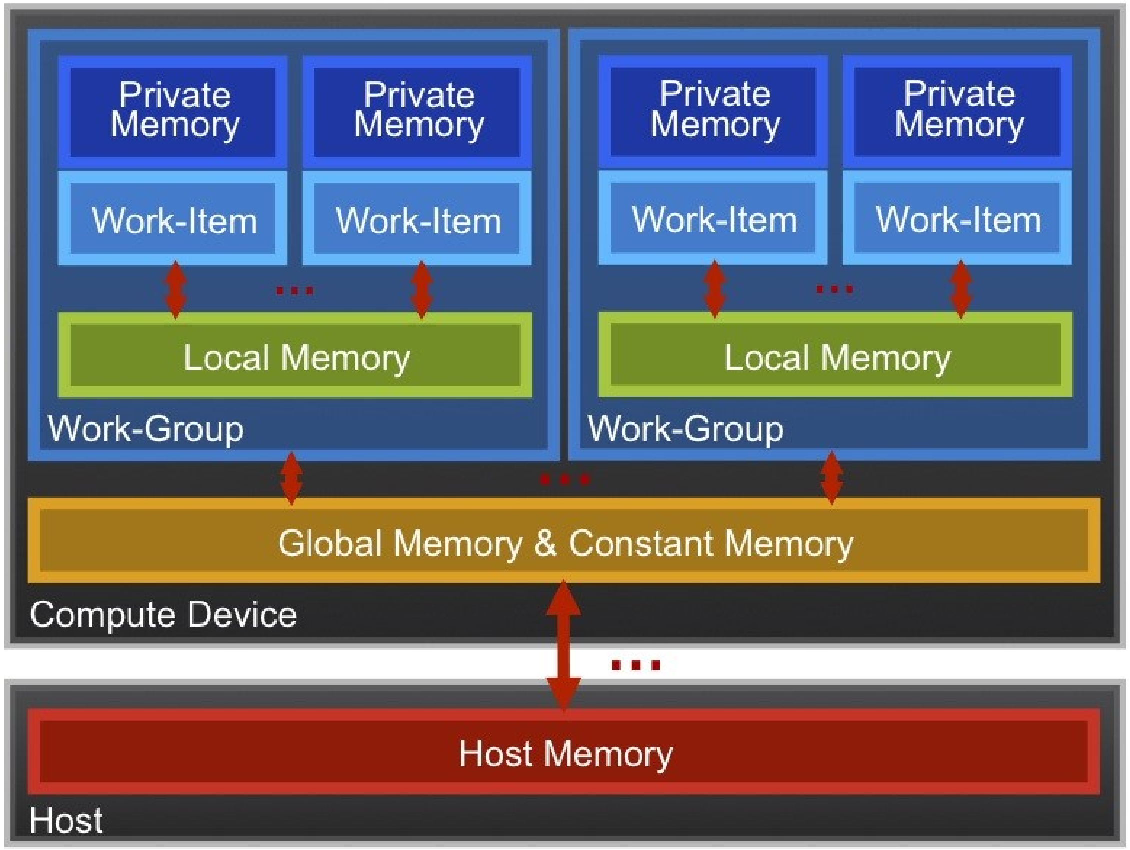
- The problem we want to compute will have some **dimensionality**;
  - E.g. compute a kernel on all points in a rectangle
- When we execute the kernel we specify **up to 3 dimensions**
- We also **specify the total problem size** in each dimension; this is called the **global size**
- We associate each point in the iteration space with a **work-item**

OpenCL N Dimensional Range (NDRange)

- Work-items are grouped into **work-groups**; work-items within a work-group can share **local memory** and can **synchronize**
- We can specify the number of work-items in a work-group; this is called the **local size** (or work-group size)
- Or you can let the OpenCL run-time choose the work-group size for you (may not be optimal)

OpenCL Memory model

- ***Private Memory***
  - Per work-item
- ***Local Memory***
  - Shared within a work-group
- ***Global Memory / Constant Memory***
  - Visible to all work-groups
- ***Host memory***
  - On the CPU



Memory management is **explicit**:
You are responsible for moving data from host → global → local *and* back

The Memory Hierarchy

Bandwidths

Private memory
O(2-3) words/cycle/WI

Local memory
O(10) words/cycle/WG

Global memory
O(800-1,000) GBytes/s

Host memory
O(10) GBytes/s

Sizes

Private memory
O(10) words/WI

Local memory
O(1) KBytes/WG

Global memory
O(10) GBytes

Host memory
O(10-100) GBytes

Speeds and feeds approx. for a high-end discrete GPU, circa 2018