April 15, 2025

Announcements

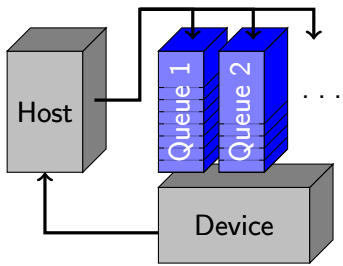
Goals

- Rest of GPA
- Trends
- Modality

Review

Host-Device Concurrency

- Host and Device run asynchronously
- Host submits to queue:
  - Computations
  - Memory Transfers
  - Sync primitives
  - ...
  - *Batches* of these
    - *Mutable* batches of these
    - Nvidia: "CUDA Graphs"
    - OpenCL: "Command buffers"
- Host can wait for:
  - *drained* queue
  - Individual "events"
- Profiling



Timing GPU Work

How do you find the execution time of a GPU kernel?

Warm-up rounds
Wait
Start the timer
Repeat the work for timing granularity
Wait
Stop the timer.

Wait
Start a timer
Enqueue the work
Stop a timer
+ wait

How do you do this asynchronously?

via the queue

Host-Device Data Exchange

Sad fact: Must get data onto device to compute

- Transfers can be a bottleneck
- If possible, overlap with computation
- Pageable memory incurs difficulty in GPU-host transfers, often entails (another!) CPU side copy
- "Pinned memory": unpageable, avoids copy
  - Various *system-defined* ways of allocating pinned memory

"Unified memory" (CUDA)/"Shared Virtual Memory" (OpenCL):

- GPU directly accesses host memory
- Main distinction: Coherence
  - "Coarse grain": Per-buffer fences
  - "Fine grain buffer": Byte-for-byte coherent (device mem)
  - "Fine grain system": Byte-for-byte coherent (anywhere)

Performance: Ballpark Numbers?

Bandwidth host/device:

v2 8GB/s v3: 16GB/s NvLink v3: .5TB/s

Bandwidth on device (A100 PCIe):

Register 100 TB/s Scratch 20TB/s Global: 1.5 TB/s

Flop throughput? (A100 PCIe)

10 TF single 10 TF double

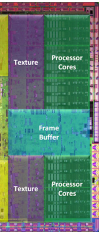
Kernel launch overhead?

10 µs

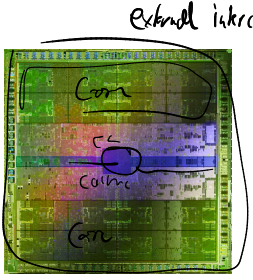
Good source of details: Wikipedia: List of Nidia GPUs

Outline

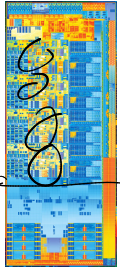
Die Shot Gallery (old)



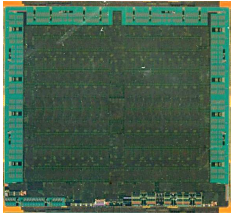
T200
8)



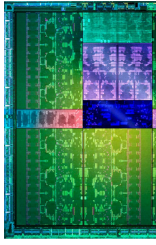
Nv Fermi
(2010)



Intel IVB
(2012)

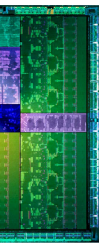


AMD Tahiti
(2012)

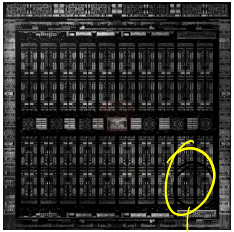


Nv GK11
(2012)

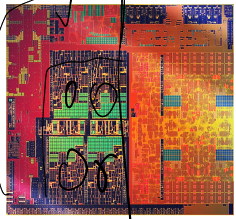
Die Shot Gallery (new)



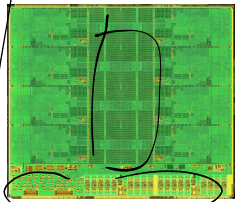
K110
2)



Nv GV100
(2018)



Intel ICL
(2018)



AMD Zen 5
(2024)

AMD MI3
(2024)

Trends in Processor Architecture

What can we expect from future processor architectures?

Commodity chips

Infinitely many cores

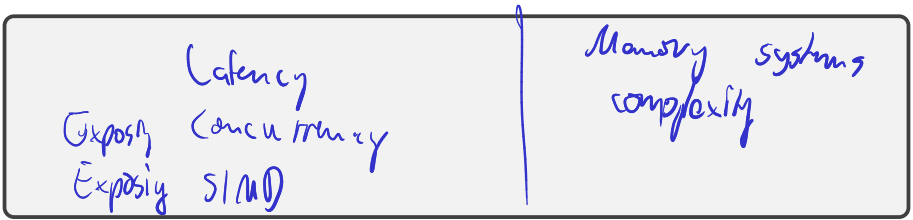
SIMD lanes

Compute BW >> mem bw

Can't keep the whole chip powered

Common Challenges

What are the common challenges encountered by both CPUs and GPUs?



Goal: Try to see CPUs and GPUs as points in a design space 'continuum' rather than entirely different things.

Outline

PTX: Demo

Demo: machabstr/PTX and SASS
Nvidia PTX manual

PTX: Cache Annotations

Loads:

`.ca`	Cache at all levels–likely to be accessed again
`.cg`	Cache at global level (cache in L2 and below and not L1)
`.cs`	Cache streaming–likely to be accessed once
`.lu`	Last use
`.cv`	Consider cached system memory lines stale–fetch again

Stores:

`.wb`	Cache write-back all coherent levels
`.cg`	Cache at global level (cache in L2 and below and not L1)
`.cs`	Cache streaming–likely to be accessed once
`.wt`	Cache write-through (to system memory)

Lost/hidden at the C level!

SPIR-V

Currently: C (OpenCL C, GLSL, HLSL) used as intermediate representations to feed GPUs.
Downsides:

- Compiler heuristics may be focused on human-written code
- Parsing overhead (preprocessor!)
- C semantics may not match (too high-level)

SPIR-V:

- Goal: Common intermediate representation ("IR") for all GPU-facing code (Vulkan, OpenCL)
- "Extended Instruction Sets":
  - General compute (OpenCL/CUDA) needs: pointers, special functions
- Different from "SPIR" (tweaked LLVM IR)

SPIR-V Example

```
          %2 = OpTypeVoid
          %3 = OpTypeFunction %2                          ; void ()
          %6 = OpTypeFloat 32                             ; 32-bit float
          %7 = OpTypeVector %6 4                          ; vec4
          %8 = OpTypePointer Function %7                  ; function-local vec4*
         %10 = OpConstant %6 1
         %11 = OpConstant %6 2
         %12 = OpConstantComposite %7 %10 %10 %11 %10 ; vec4(1.0, 1.0, 2.0, 1.0)
         %13 = OpTypeInt 32 0                             ; 32-bit int, sign-less
         %14 = OpConstant %13 5
         %15 = OpTypeArray %7 %14
[...]
         %34 = OpLoad %7 %33
         %38 = OpAccessChain %37 %20 %35 %21 %36      ; s.v[2]
         %39 = OpLoad %7 %38
         %40 = OpFAdd %7 %34 %39
               OpStore %31 %40
               OpBranch %29
         %41 = OpLabel                                   ; else
         %43 = OpLoad %7 %42
         %44 = OpExtInst %7 %1 Sqrt %43                  ; extended instruction sqrt
         %45 = OpLoad %7 %9
         %46 = OpFMul %7 %44 %45
               OpStore %31 %46
```

Outline

Outline

Qualifying Performance

- What is *good* performance?
- Is speed-up (e.g. GPU vs CPU? C vs Matlab?) a meaningful way to assess performance?
- How else could one *form an understanding* of performance?

Model

Hager et al. '17
Hockney et al. '89

A Story of Bottlenecks

Imagine:

- A bank with a few service desks
- A revolving door at the entrance

What situations can arise at *steady-state*?

Line at the door
Line at the desk

What numbers do we need to characterize performance of this system?

b : Throughput of door [customers/s]
I : "intensity" [questions/customer]
P_{peak} : Desk throughput [questions/s]

A Story of Bottlenecks (cont'd)

- P_{peak}: [task/sec] Peak throughput of the service desks
- I: [tasks/customer] Intensity
- b: [customers/sec] Throughput of the revolving door

What is the aggregate throughput?

$P \leq \min\left(b \cdot I, P_{\text{peak}}\right)$ [questions/s]

Hager et al. '17