Why are machine learning forecast systems so fast?

Performance measurement of traditional and ML forecast systems

Christopher Subich
Environment & Climate Change Canada
christopher.subich@ec.gc.ca

Outline

1. Introduction and Scope
    - The surprising computational powers of AI
    - Scope of this talk
    - Deepmind's GraphCast vs ECCC's GEM
2. Performance measurements
    - Roofline model of performance
    - Hardware capabilities at ECCC
3. Results
4. Discussion & Conclusions

Introduction

- The age of data-driven, AI-based weather forecasting is new
  - Just more than two years since the publication of FourCastNet (v1, NVidia)
  - GraphCast (Deepmind) is less than eighteen months old.
  - Other recent entries into the field:
    - Pangu-Weather (Huawei), FengWu (Shanghai AI Lab), FuXi (Fudan University), AIFS (ECMWF)
  - All promise state of the art forecast performance with "orders of magnitude" shorter running times
- These models focus on *quarter-degree, global atmospheric forecasts* at lead times from a few hours to about 10 days
  - Comparable to state-of-the art traditional forecast systems from a few years ago
  - Limited set of model levels, output times, and output variables
  - Common but yet-unproven assumption that it's only a matter of time before these models overtake traditional NWP at the cutting edge.

This talk

- Narrow objective of this talk: examine the *performance claims* of GraphCast compared to Environment & Climate Change Canada's (ECCC's) operational forecast system
- Forecast accuracy (and physical consistency) is a matter of ongoing research, and any talk is sure to be immediately out of date as a new system breaks score records.
  - Performance, however, is more a matter of basic architecture
- What supports the incredible speed claims for AI forecasting systems?
  - Have they learned a better / more efficient representation of the Earth system?
  - Are AI systems just really good at running on GPUs?
  - Are the claims misleading because the AI systems do less work?
    - Fewer vertical levels / output variables / timesteps
- The answer affects the long-term future of traditional NWP
  - We might never beat AI at a "more efficient" representation of weather on Earth
  - Gains from sparse outputs will be lost if we demand more complete forecasts

The contenders

- Representing AI forecasting systems: GraphCast
  - Open-sourced in August 2023 by Google Deepmind
  - Quarter-degree, 6h forecast iterated for longer lead times
  - 37 pressure levels, predicting wind, temperature, geopotential, and humidity plus several surface variables
  - Run on a single NVidia A100 GPU
  - Graph Neural Network architecture
- Representing traditional NWP: ECCC's GEM
  - One third-degree global configuration, matching operational ensemble forecast
  - 84 vertical layers (log hydrostatic pressure coordinate), 15 minute timestep
  - Run on 80 CPU cores (one compute node)
  - Semi-Lagrangian advection, implicit Crank-Nicholson timestepping scheme with FFT-based innermost solver

The contest

- Both systems are asked to produce a single 24-hour forecast
- Closest possible comparison between stock/operational systems, but still not equal
  - GraphCast has slightly higher horizontal resolution
  - GEM has much higher vertical resolution and a much shorter timestep
    - Necessary for stability, but also allows more frequent outputs "for free"
  - GEM was run with full microphysics enabled, tracking about 200 variables internally
  - GEM would normally run operationally with multi-node parallelism
- Measurements:
  - Overall wall-clock time, including necessary I/O to read inputs and write the forecast
  - Computation time, excluding I/O
  - FLOPS executed on CPU or GPU during "core" computation
  - Amount of memory accessed during computation
- Measurements taken via CPU/GPU performance counting registers

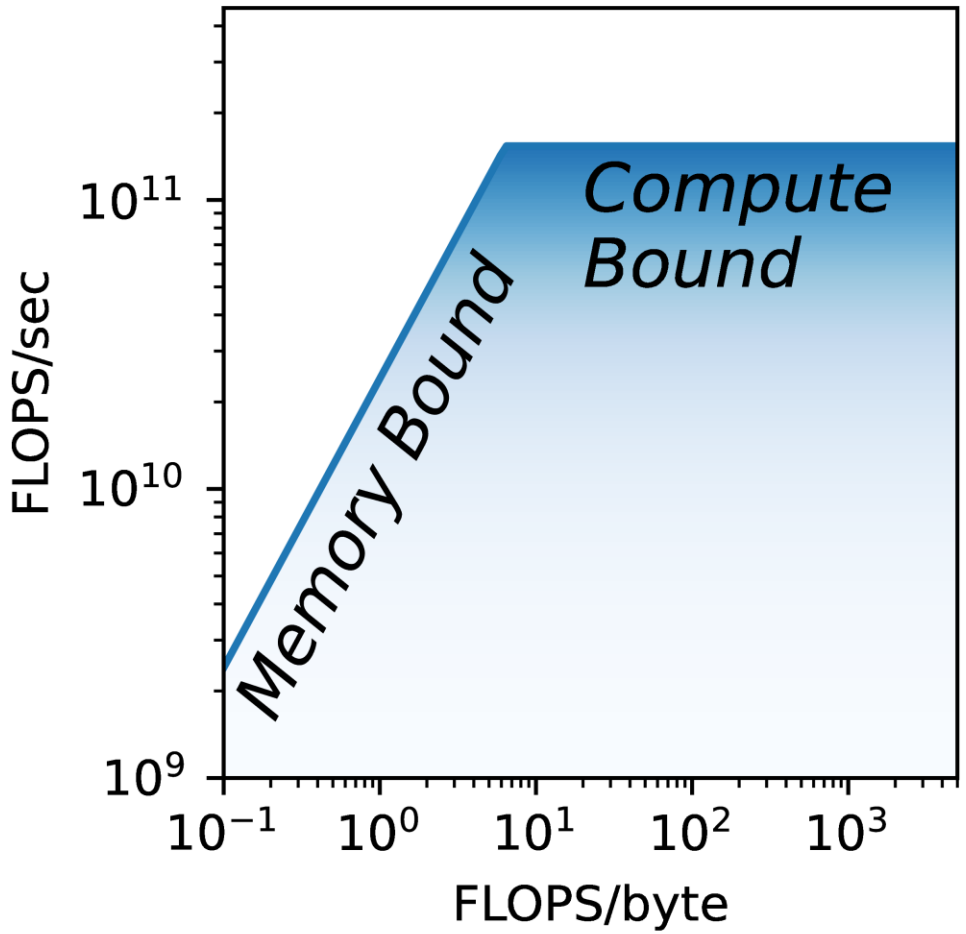
Roofline model

- It's difficult to compare performance across both algorithms and hardware architectures
  - How many CPUs are equivalent to one GPU?
  - Instead, compare each algorithm to the hardware's peak performance
- Schematic view:
  - Numerical forecasting mostly does math (FLOPS)
  - Forecast systems also must move data around in memory
  - These things happen simultaneously, but both take time

$$\text{runtime} = \max\left(\frac{\text{memory loads}}{\text{memory bandwidth}}, \frac{\text{FLOPS}}{\text{compute rate}}\right)$$

- Idealized two-parameter model
  - Assumes that the ideal algorithm should perfectly use computing resources
  - No serialization or communication bottlenecks
  - No real code will ever reach the peak, but we'd like to come close

Roofline model

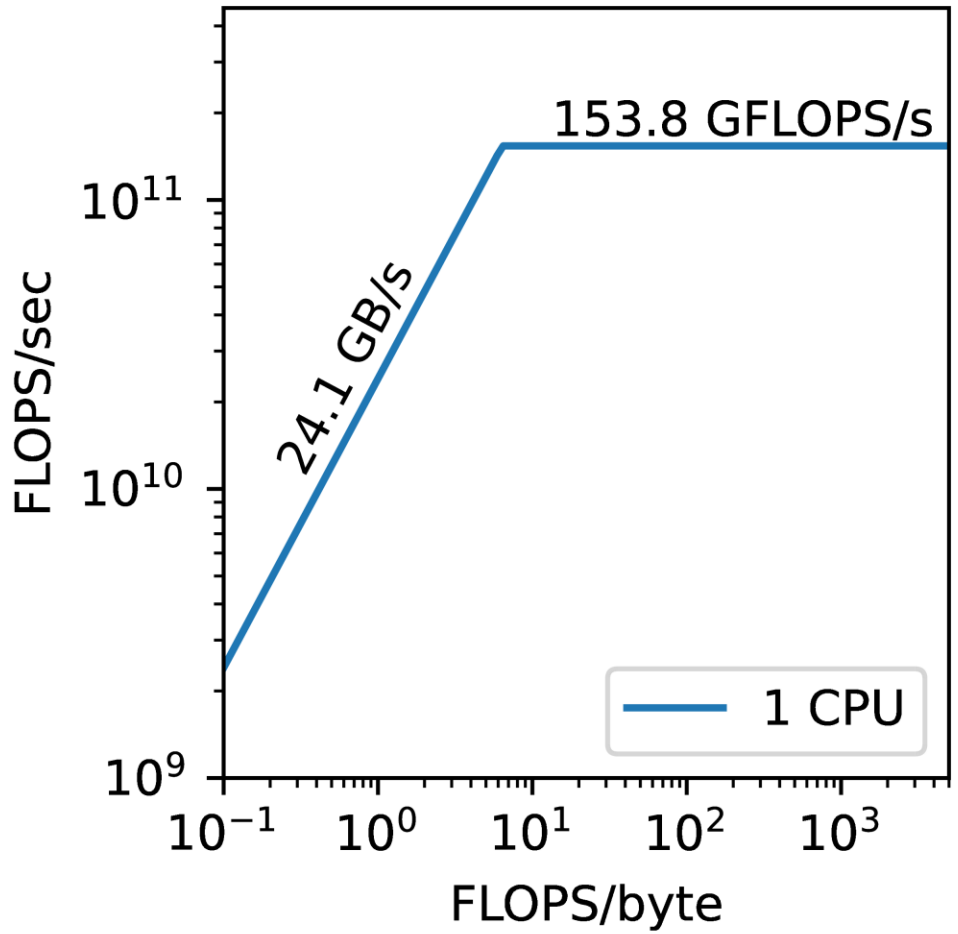


- On a log-log plot, peak performance looks like a "roofline"
- Left side: *memory bound* algorithms
  - Faster with better memory/cache
- Right side: *compute-bound* algorithms
  - Faster with more/wider vector units, faster floating-point math
- Transition happens at a particular compute intensity
  - Measured in FLOPS/byte
  - Hardware defines the roofline shape
  - Software can exist anywhere beneath the roofline

The computer system

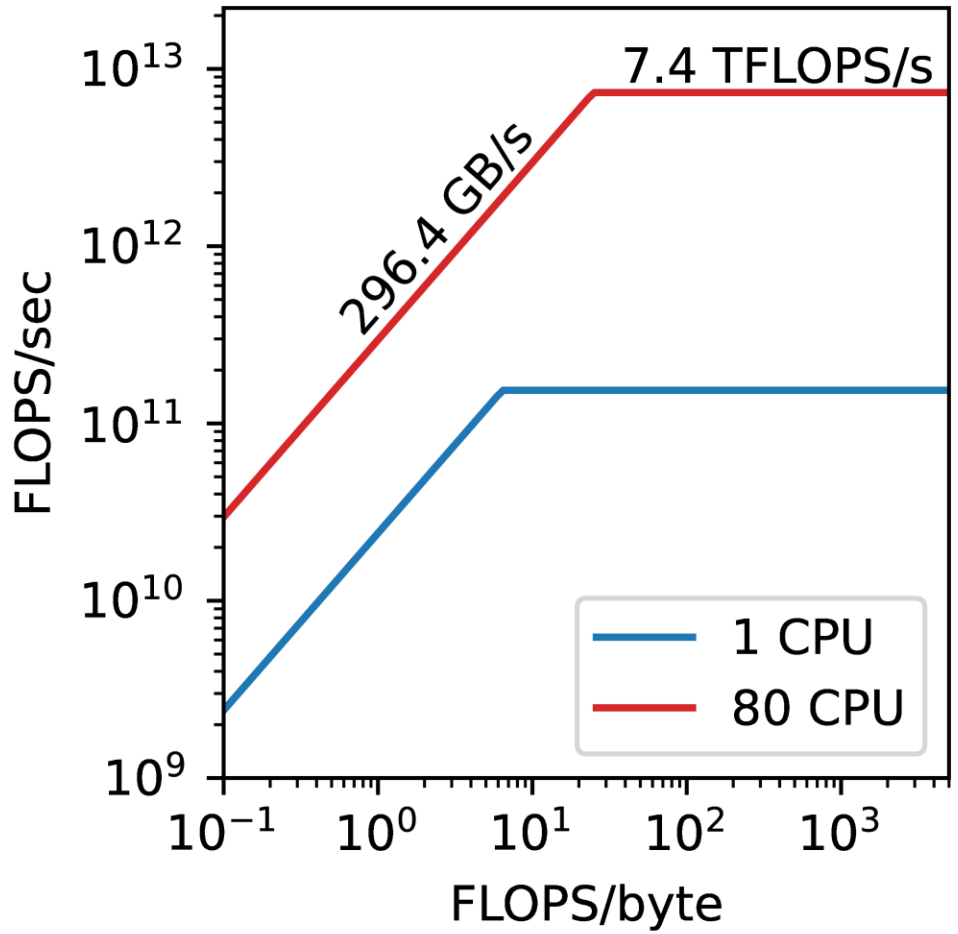
- Tests were run on Environment & Climate Change Canada's `underhill` system
  - Debut in June 2022 at rank 69 on Top500 list
  - Lenovo Thinksystem SD650 V2 system
  - Primary compute nodes have 2 Xeon Platinum 8380 40-core processors @ 2.3GHz
    - 80 CPU cores per compute node (no hyperthreading used)
  - Supplemented with two GPU nodes, each with 4 NVidia A100 GPUs (40GB)
    - 2 × 24-core CPUs on each GPU node, but that's not particularly relevant
- Testing configurations:
  - GEM: 80 CPU cores (one compute node; inherently parallel)
  - GraphCast:
    - 1 GPU (typical deployment)
    - 1 CPU (minimum possible configuration)
    - 80 CPU ("equivalent" to GEM)

System performance – 1 CPU core



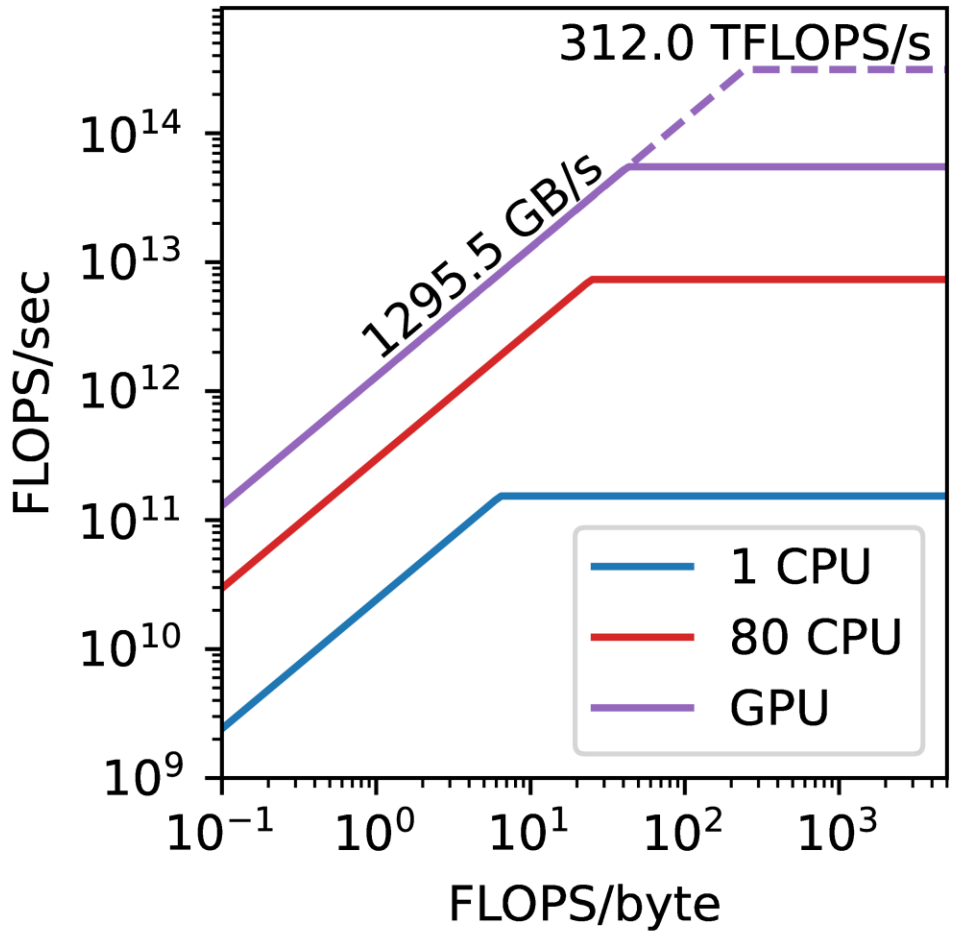
- Measurements made via Empirical Roofline Tool
  - Measures effective system performance, rather than on-paper specification
- 1 CPU core: the most basic "compute unit"
  - 24.1 GB/second memory bandwidth
  - About 150 GFLOPS/sec compute rate (single precision)
  - Requires about 8 FLOPS/byte for an algorithm to become compute-bound (32 FLOPS per single-precision float)

System performance – 80 CPU cores



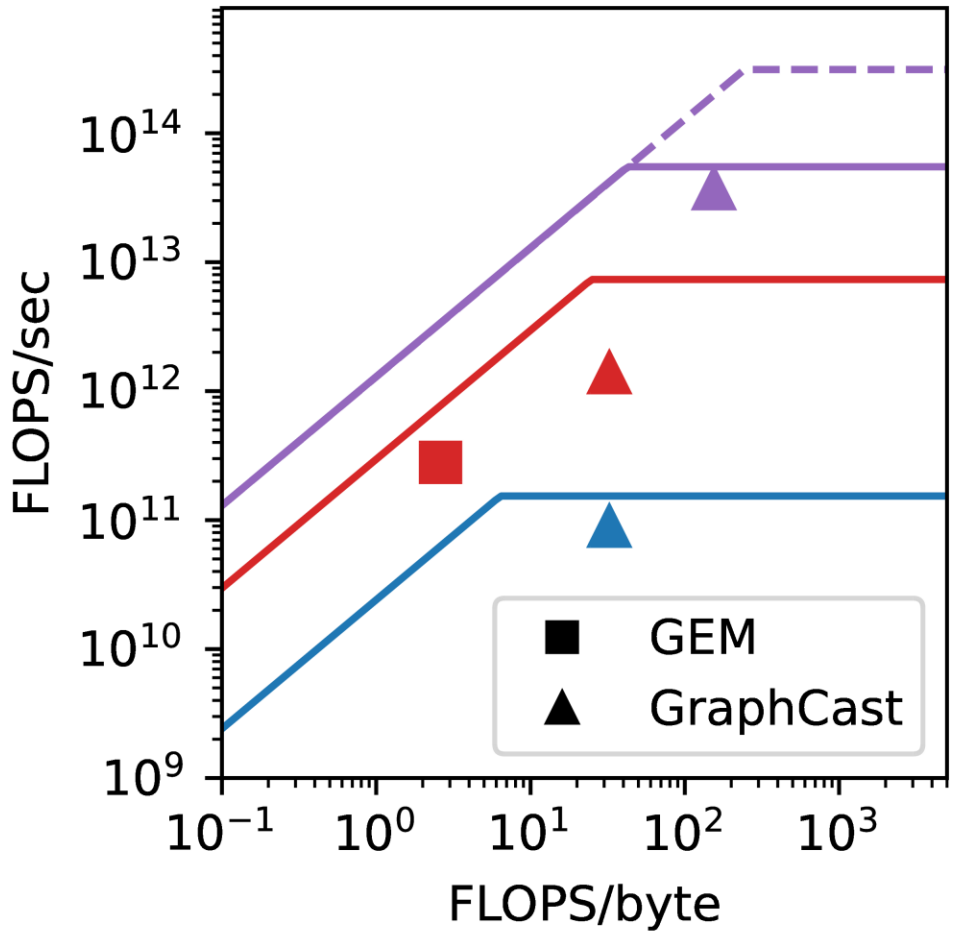
- 80 CPU cores: one full compute node
  - 295 GB/sec memory bandwidth
  - 7.35 TFLOPS/sec compute rate
- Values are obviously better than for 1 CPU core, but not 80× better
  - About 12× memory bandwidth
  - About 50× compute rate
  - Limits due to saturation of memory bus and CPU boost clock rate
- Need greater compute-intensity (now 25 FLOPS/byte) for an algorithm to become compute-bound

System performance – 1 GPU



- One single A100 GPU
  - 1.3 TB/sec memory bandwidth to on-card high-bandwidth memory
  - 55 TFLOPS/sec baseline compute rate (half-precision)
  - 312 TFLOPS/sec with tensor cores (on-paper)
- One GPU has 4–40× the performance of an entire compute node
- Requires even greater compute intensity for best use: up to 240 FLOPS/byte

Results



- FLOPS and compute intensity measured for compute kernels only, not I/O or initialization
- GEM: 150 TFLOPS, 2.6 FLOPS/byte
  - 80 CPU: 596s runtime, 530s compute
  - Clearly limited by memory bandwidth
- GraphCast: 110 TFLOPS
  - CPU: 32 FLOPS/byte
    - 1 CPU: 1265s, 1239s compute
    - 80 CPU: 103s, 80s compute
  - GPU: 154 FLOPS/byte
    - 1 GPU: 25s, 3.1s compute
  - Compute-limited on CPU, close on GPU

Discussion

- GraphCast is indeed very fast, but it's not doing less work than a traditional NWP
  - 110 TFLOPS versus 150 TFLOPS for GEM, with comparable number of grid points
  - Fewer simulated variables
- Decomposition of the speed advantage:
  - 1.36 × from fewer FLOPS (but no physics parameterizations)
  - 4.85 × from better compute efficiency (80 CPU result)
  - 25.8 × from running on a GPU (1 GPU vs 80 CPU)
  - Net speed advantage of 170×
- 3 seconds per 24-hour forecast is almost impractically fast
  - Computation time can easily become dominated by I/O
  - Renewed emphasis on high-performance, easily-parsed, parallelizable file formats
  - Easy to accidentally leave expensive GPUs idle

Conclusions

- Return to the questions from the beginning:
- Have AI forecast systems learned a more efficient representation of the weather?
  - *Probably not* – interperability is hard, but no evidence that GraphCast does more with fewer FLOPS
- Are AI forecast systems really good at running on GPUs?
  - *Definitely* – the GPU advantage was responsible for over half of the 170× performance improvement
  - Enabled because GraphCast has much greater compute-intensity than GEM; a straightforward port of GEM would not receive the same benefit
- Are the performance claims misleading because AI systems do less work?
  - *Probably not*, but there's room for nuance
  - A 170× advantage is a lot, even if an AI system needs to be larger/run more to produce complete outputs
  - Open questions about model capacity & what can be added without a penalty

Conclusions – Next steps & tomorrow's models

- I think this story is oddly encouraging for traditional NWP
- We leave a lot of compute capacity underutilized
  - GEM could do an order of magnitude more computation "for free" with its current memory access patterns
  - Hard to retrofit this onto current models, but for the future we need to re-think our intuitions about what is fast and slow
    - Very high-order dynamical cores?
- AI models are designed around accelerator use and high compute-intensity, but face their own challenges
  - We aren't yet seeing weather models on the scale of the largest language models
  - What price will we pay for GPU parallelism, especially across nodes?
  - High-resolution, global, multimodal weather data is a very large working set

References

- GraphCast — Remi Lam et al. 2023. “Learning Skillful Medium-Range Global Weather Forecasting.” *Science* 382 (6677): 1416–21. https://doi.org/10.1126/science.adi2336.
- GEM — https://github.com/ECCC-ASTD-MRD/gem
- PAPI (Performance measurement library) — https://icl.utk.edu/papi, https://github.com/icl-utl-edu/papi
- NVidia GPU performance counters — https://developer.nvidia.com/cupti
- Empirical Roofline Tool — https://crd.lbl.gov/divisions/amcr/computer-science-amcr/par/research/roofline/software/ert

Technical addendum

- Measurement procedure:
  - GEM: dynamical core and physics parameterizations were broken out via internal profiling for measuring compute time and memory/FLOPS (roughly equal time in each)
  - GraphCast: Based on Deepmind sample code, discarding one forecast to exclude JAX compilation times.
- Floating point events:
  - CPU: Aggregate `PAPI_SP_OPS` and `PAPI_DP_OPS`, which combine scalar and vector operations with appropriate factors; double-precision received 2× weight.
  - GPU: `smsp__ops_path_tensor_src_bf16_dst_fp32`, counting tensor-core operations (about 98% of all FLOPS)
- RAM events:
  - CPU: `OCR:READS_TO_CORE_DRAM`, counting cache lines moved from RAM to CPU
  - GPU: `dram__bytes_read`, reads from on-GPU memory