SC22

Dallas, TX | hpc accelerates.

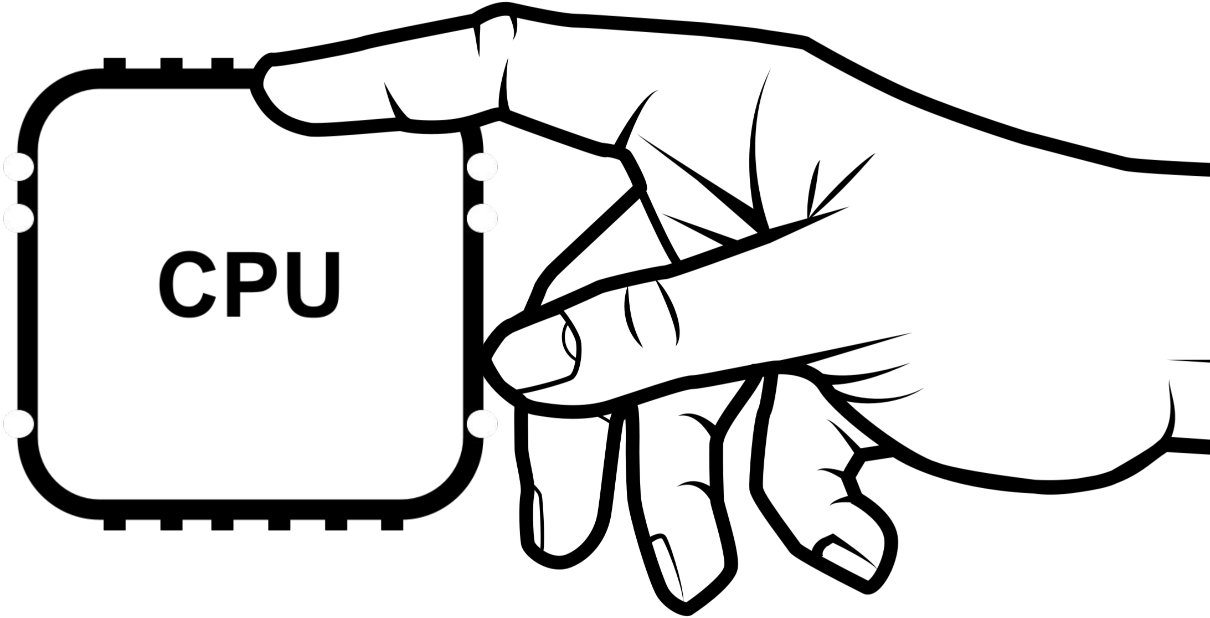
A Methodology for Evaluating Tightly-integrated and Disaggregated Accelerated Architectures

Taylor Groves, Chris Daley, Rahulkumar Gayatri, Hai Ah Nam, Nan Ding, Lenny Oliker, Nicholas J. Wright, Samuel Williams

Lawrence Berkeley National Laboratory

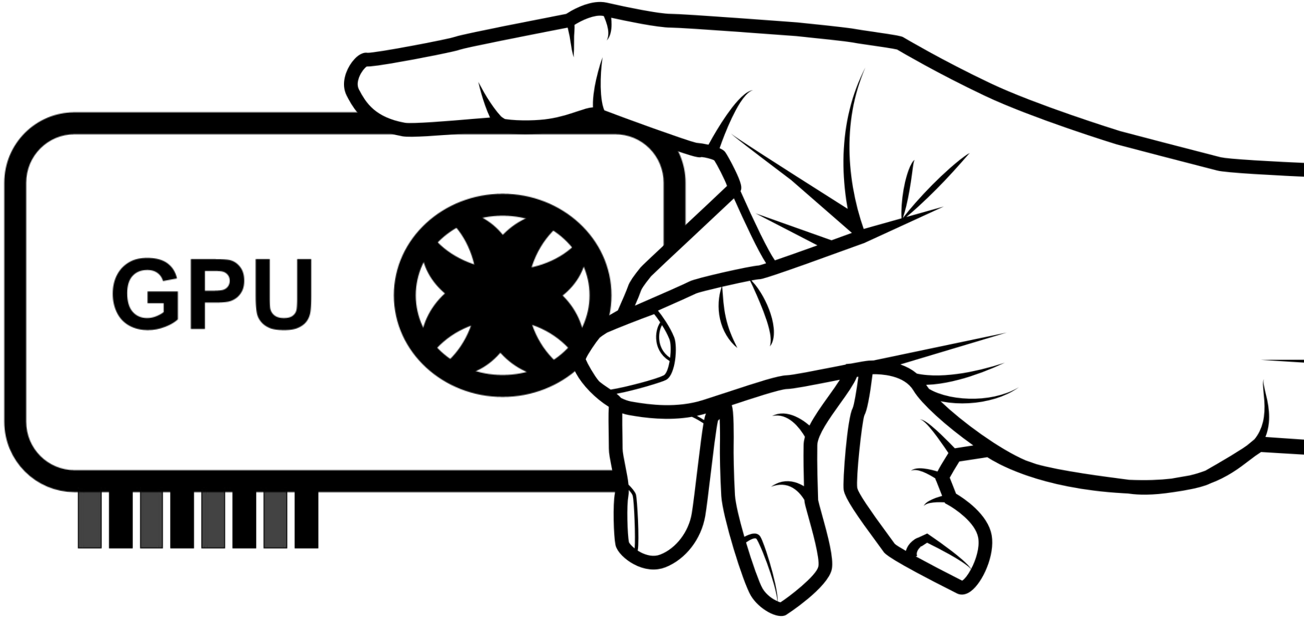
In the (Not-quite) Beginning We Were Given CPUs

However, it had limited parallelism and FLOPS



Sometime Later People Added GPUs

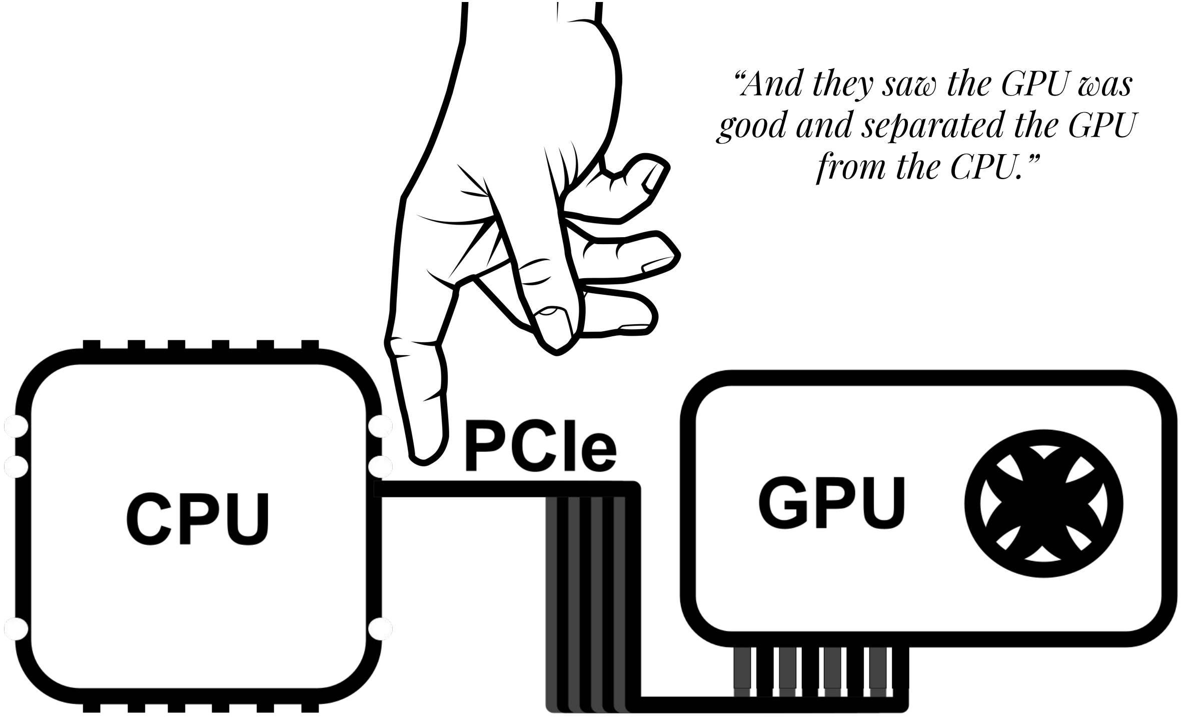
Lots of parallelism, but still need to be managed by the CPU



PCIe Connected the CPU and GPU (discrete)

Offload Strategy…
use the GPU if you have enough SIMD work to pay the costs of:

- GPU memory allocation
- PCIe data transfer
- Kernel launch



"And they saw the GPU was good and separated the GPU from the CPU."

For over a decade PCIe connected the GPU to the CPU, but two opposing trends have emerged:

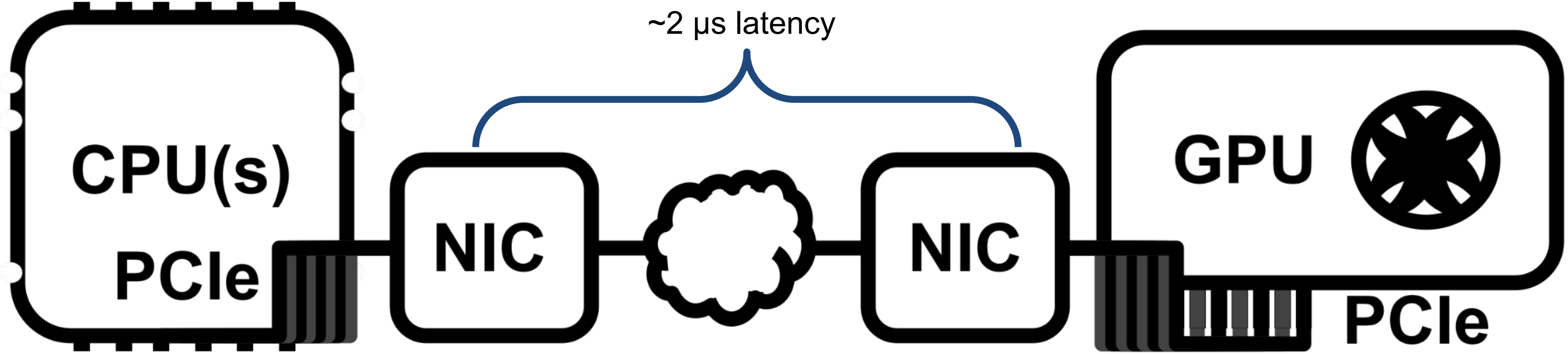
(1) Disaggregation and

(2) Tighter coupling

Idea 1: Loosely-coupled GPU (disaggregated)

Separate the CPU and GPU further across a network.
Composable architecture, GPU shared with CPUs on the network
Save money, but this adds overheads to the data transfer.

- But maybe the added cost isn't too bad?

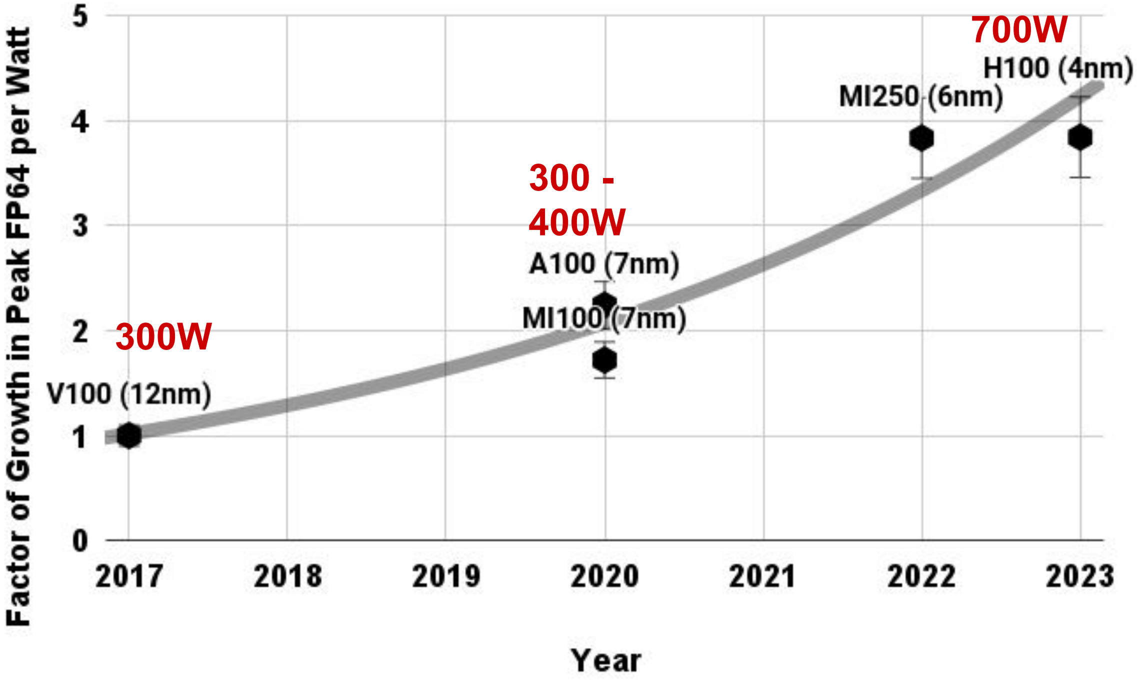


Idea 2. Tightly Coupled GPU and CPU

FLOPS per Watt and GPU power are increasing.

SIMD work time shrinking faster than data transfer time.

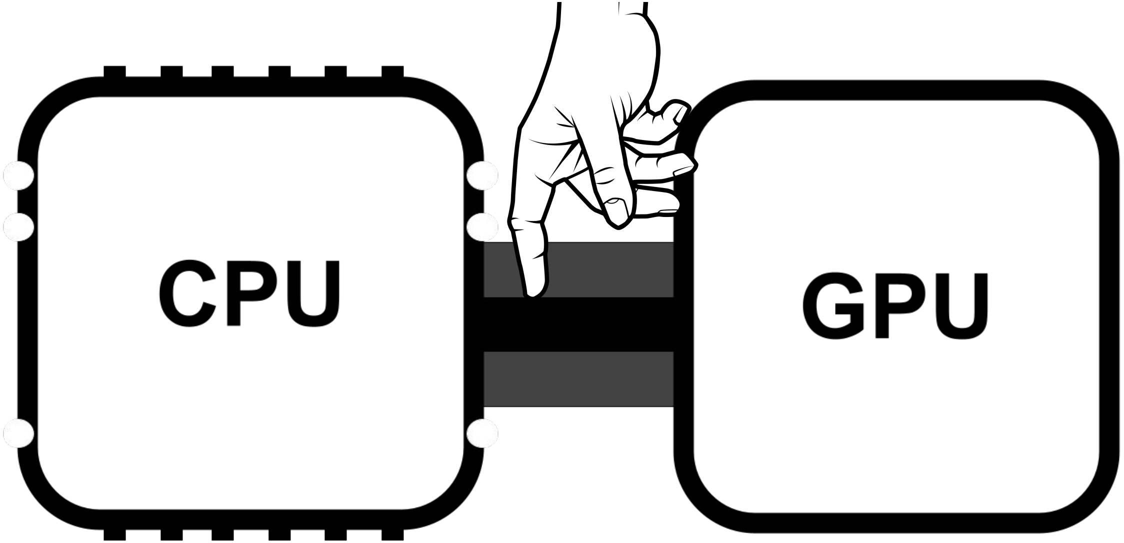
Amdahl's Law means data transfer time is becoming increasingly important.



Idea 2a. Replace PCIe With Something Faster (On-board)

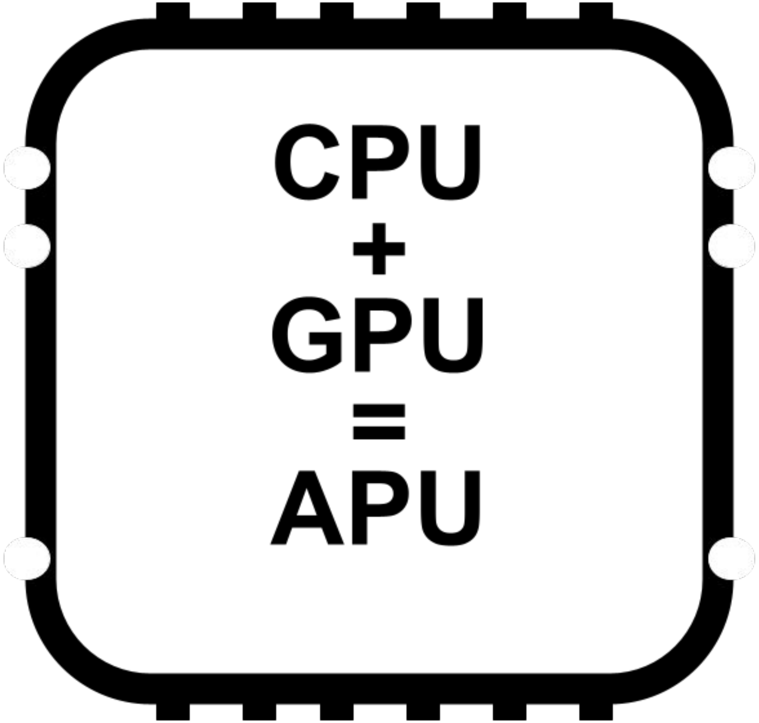
- High bandwidth and low latency interconnect
- Greatly reduces data transfer overheads
- Still maintains two separate memories (e.g. HBM and DDR)

Typically an order of magnitude higher bandwidth than PCIe



Idea 2b. Put the GPU + CPU on the Same Package (APU)

- Shares a single memory – eliminates data transfer!
- Enables acceleration of small kernels that aren’t otherwise offloaded
- CPU and GPU must share limited space and power budget



Our Idea: Use a General Model to Evaluate These Designs

Keep it simple.

Overheads:

- Init. (malloc, kernel launch)
- Data transfer in
- Data transfer out

Benefits

- Speedup of GPU computation

Focus is CPU-to-GPU coupling (i.e. data transfers)

We use the LogGP model

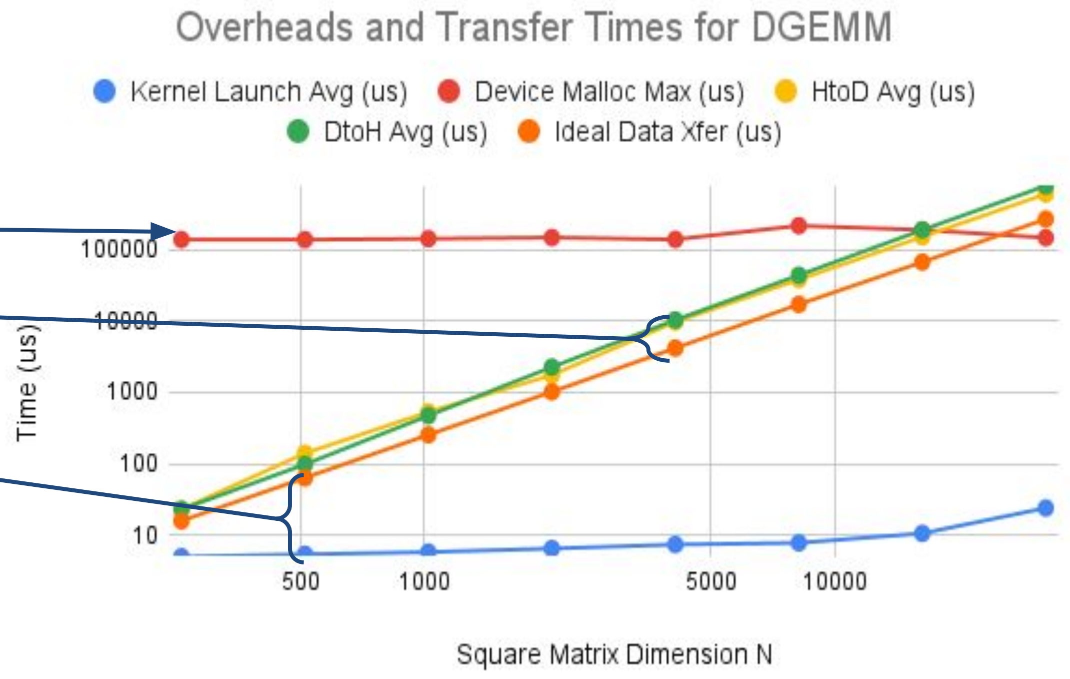
- latency
- overhead
- gap (1 / bandwidth)
- Gap (msg size / bandwidth)

Computation model: peak FLOPS

Measuring Accelerator Overheads and Transfer Times

Study the model parameters for well understood DGEMM (N^3 FLOPS : N^2 data xfer)

- Mallocs are expensive, but potentially amortized.
- Derive LogGP from the recorded data transfer time.
- For matrix size > 512^2, data transfer dominates vs. kernel launch



Measured on NERSC Perlmutter Nvidia A100 GPU

How to Capture Complex Behavior?

Multi-phase application with varying patterns of compute

Data transfers may overlap with computation

Multi-tier accelerator networks (e.g. NVlink)

Runtime vs Kernel Window in summary statistics.

Derive LogGP parameters of data transfers at varying size?

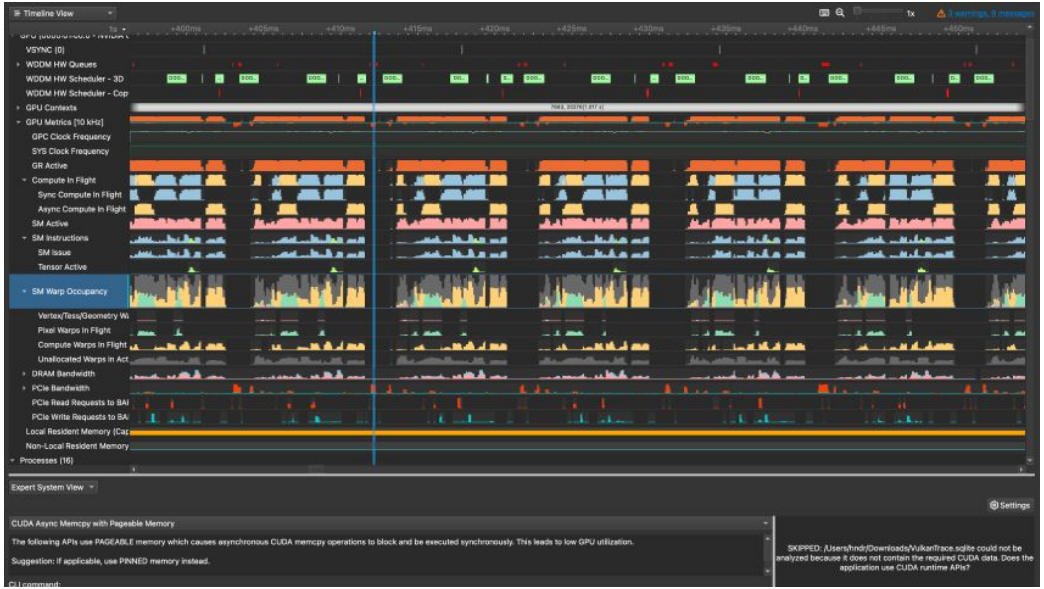
Can we amortize memory allocations? How many times do they occur?

Need Insights Without App. Expertise: Nvidia Nsight Systems

Fantastic profiler featuring:

- timelines for CPU and GPU
- PCIe data transfer
- NVLink (peer to peer) transfers
- memory allocation
- supports Mellanox NICs

However, we still don't have everything we need to study CPU-GPU coupling



CUDA Async Memcpy with Pageable Memory

The following APIs use PAGEABLE memory which causes asynchronous CUDA memcpy operations to block and be executed synchronously. This leads to low GPU utilization.

Suggestion: If applicable, use PINNED memory instead.

CLI command:

SKIPPED: /Users/hndr/Downloads/VulkanTrace.sqlite could not be analyzed because it does not contain the required CUDA data. Does the application use CUDA runtime APIs?

(https://developer.nvidia.com/nsight-systems)

Where Do We Need to Extend the Features?

Multi-phase application with varying patterns of compute

Data transfers may overlap with computation

Multi-tier accelerator networks (e.g. NVlink)

Runtime vs Kernel Window in summary statistics.

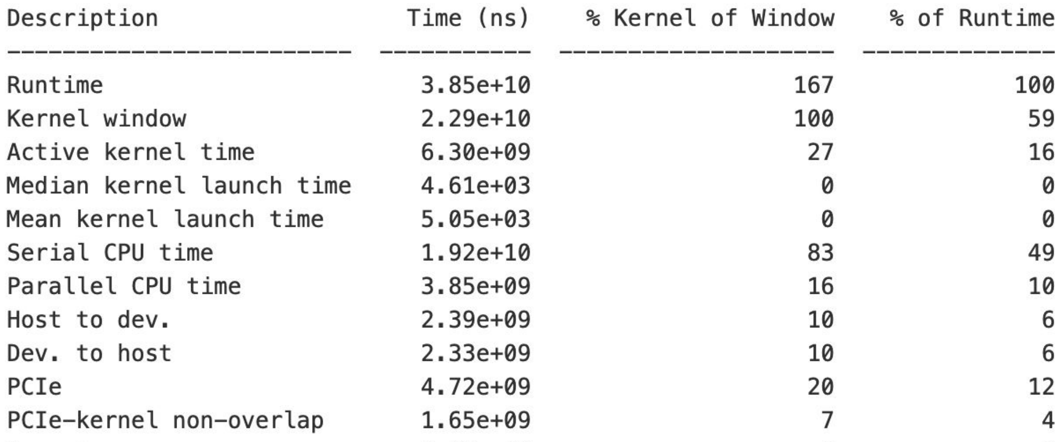
Derive LogGP parameters of data transfers at varying size?

Can we amortize memory allocations? How many times do they occur?

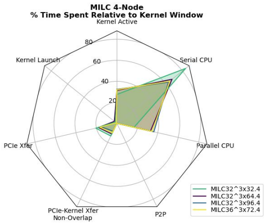
Introducing NEthing (https://gitlab.com/NERSC/nething)

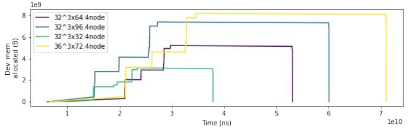
NEthing takes existing sqlite profiles and extends Nsight Systems capabilities.

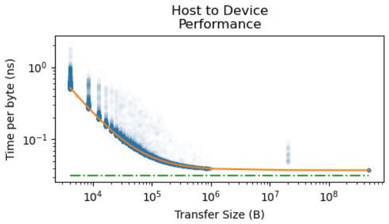
- Merges of different time series to calculate computation and data transfer overlap,
- Generates the parameters for a LogGP based model and
- Provides useful summary statistics and visualizations for both *kernel window* and *runtime*



```
Description                   Time (ns)   % Kernel of Window   % of Runtime
--------------------------   ----------   -------------------   -------------
Runtime                        3.85e+10                   167             100
Kernel window                  2.29e+10                   100              59
Active kernel time             6.30e+09                    27              16
Median kernel launch time      4.61e+03                     0               0
Mean kernel launch time        5.05e+03                     0               0
Serial CPU time                1.92e+10                    83              49
Parallel CPU time              3.85e+09                    16              10
Host to dev.                   2.39e+09                    10               6
Dev. to host                   2.33e+09                    10               6
PCIe                           4.72e+09                    20              12
PCIe-kernel non-overlap        1.65e+09                     7               4
```







NEthing to Examine CPU-GPU Coupling

Three apps: DGEMM, MILC and LAMMPS-SNAP

Running across (4) A100 GPUs (NERSC Perlmutter)
Increasing problem size:

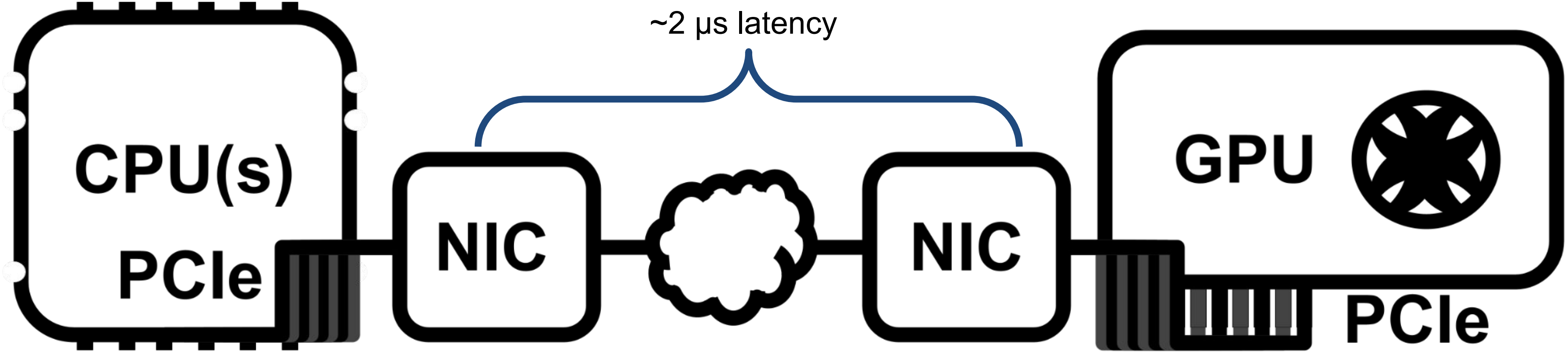
- 2-8GB GPU memory

CPU assumed to be 1/8th FLOPS of GPU

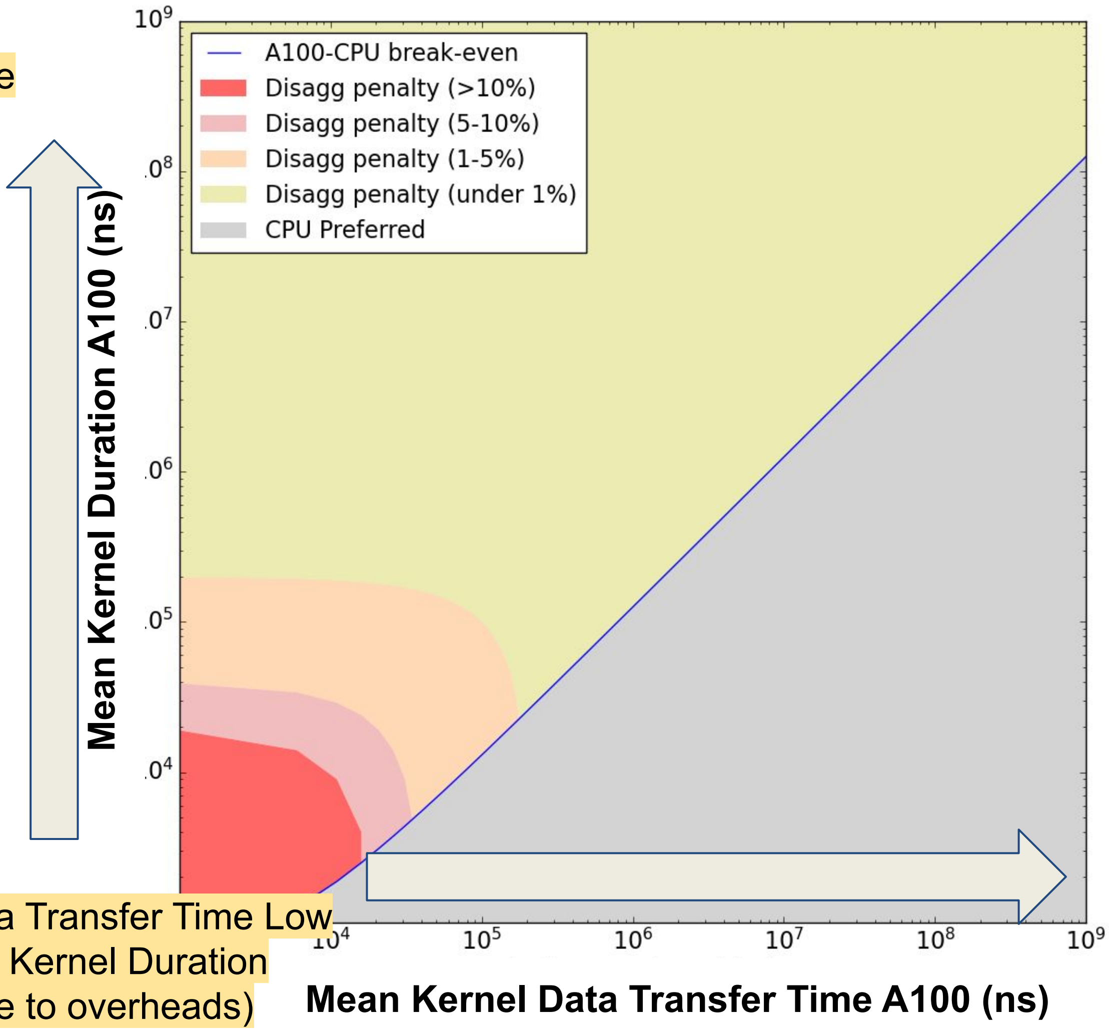
Adjust CPU-GPU connectivity for 2022 and 2026 architecture design points.

Idea 1: Loosely-coupled GPU (disaggregated in 2022)

- Use Average Transfer Size and Kernel Duration from profiles
- Insert an additional 2 μs delay per transfer
- Impact on performance?



Lots of GPU Time and Low Data Transfer

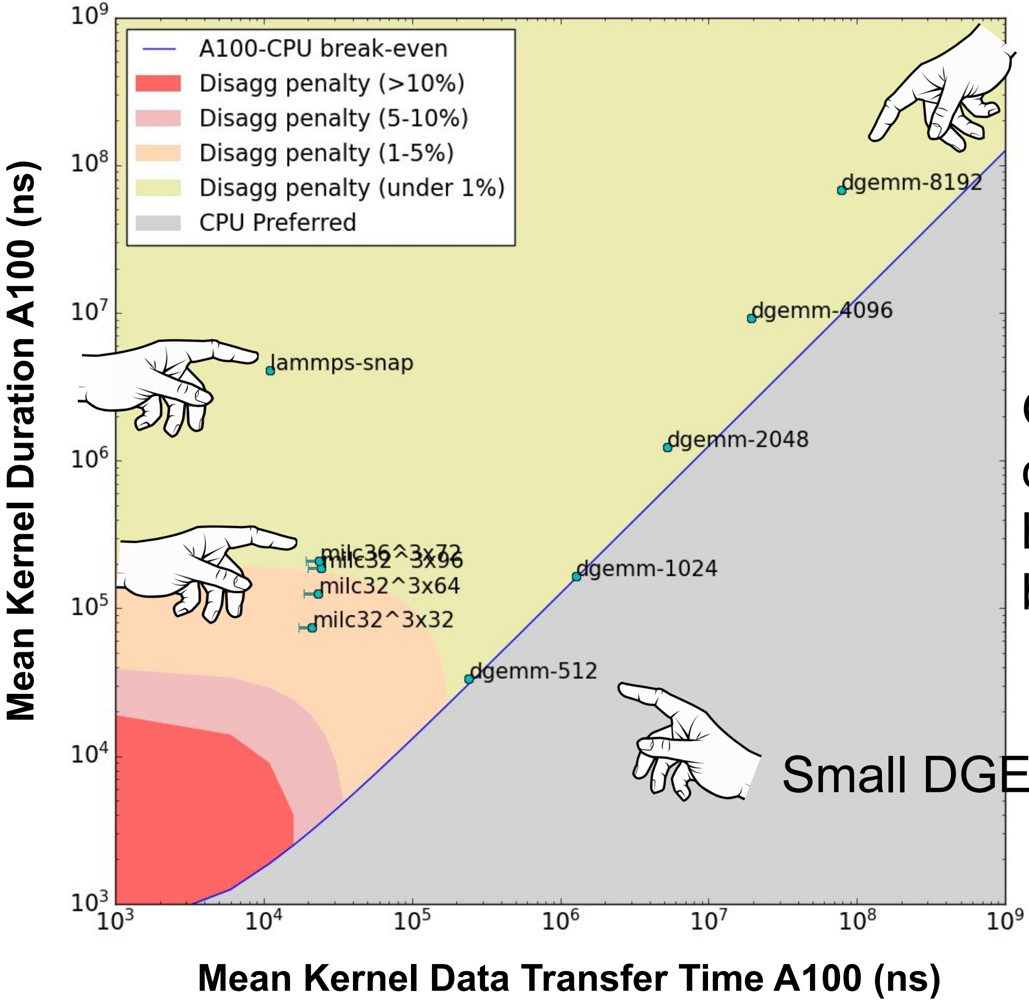


High Data Transfer Costs with Low Opportunities for Speedup

Low Data Transfer Time Low and Low Kernel Duration (sensitive to overheads)

Yellow area signifies a disaggregation penalty of < 1%

- LAMMPS-SNAP
- Large MILC problems



Large DGEMM compute scales N^3 data xfer scales N^2

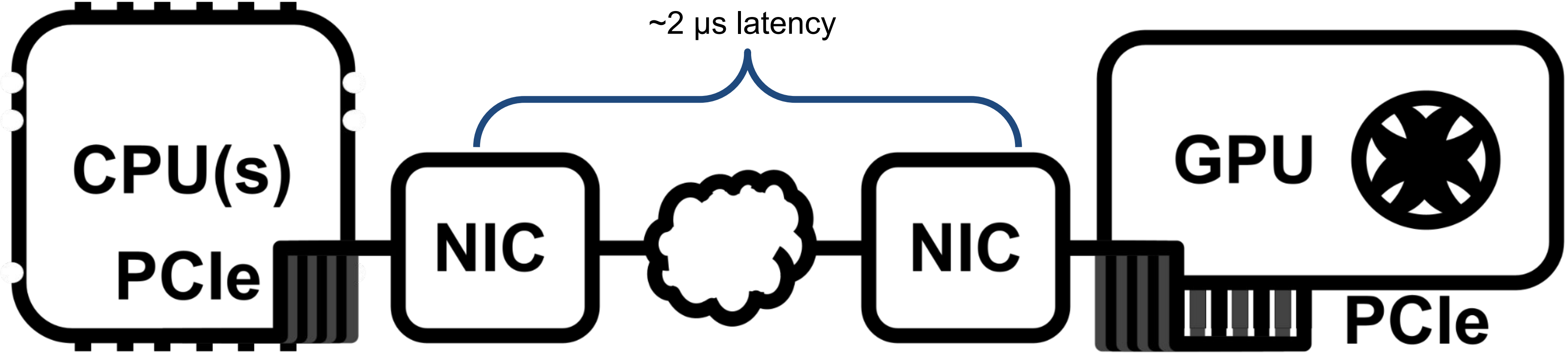
Grey area signifies codes that do not have a performance benefit on the GPU

Small DGEMM

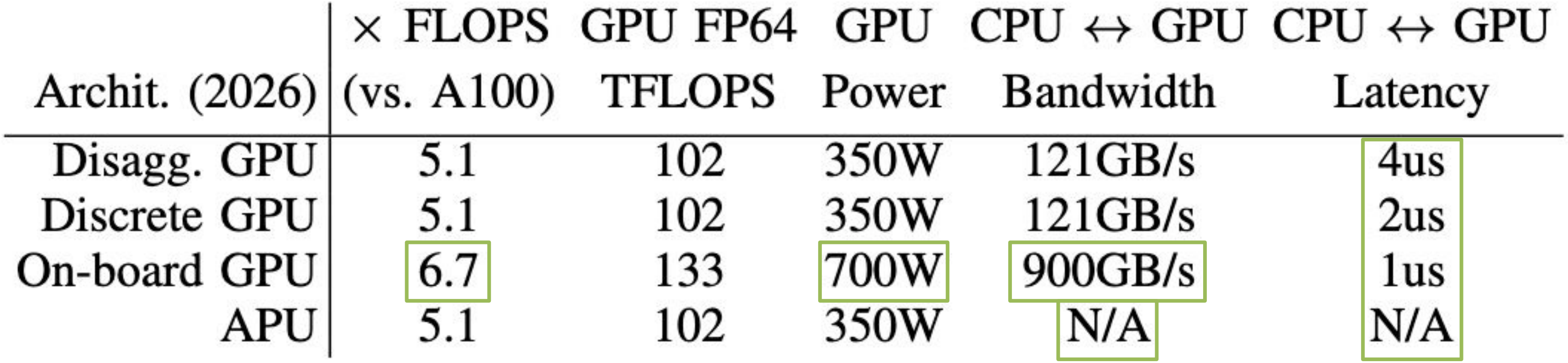
Idea 1: Loosely-coupled GPU (disaggregated)

The workloads evaluated (DGEMM, MILC, LAMMPS-SNAP) seem amenable to disaggregation for **today's** systems…

But what about for future architectures?



Develop Four Future Design Points



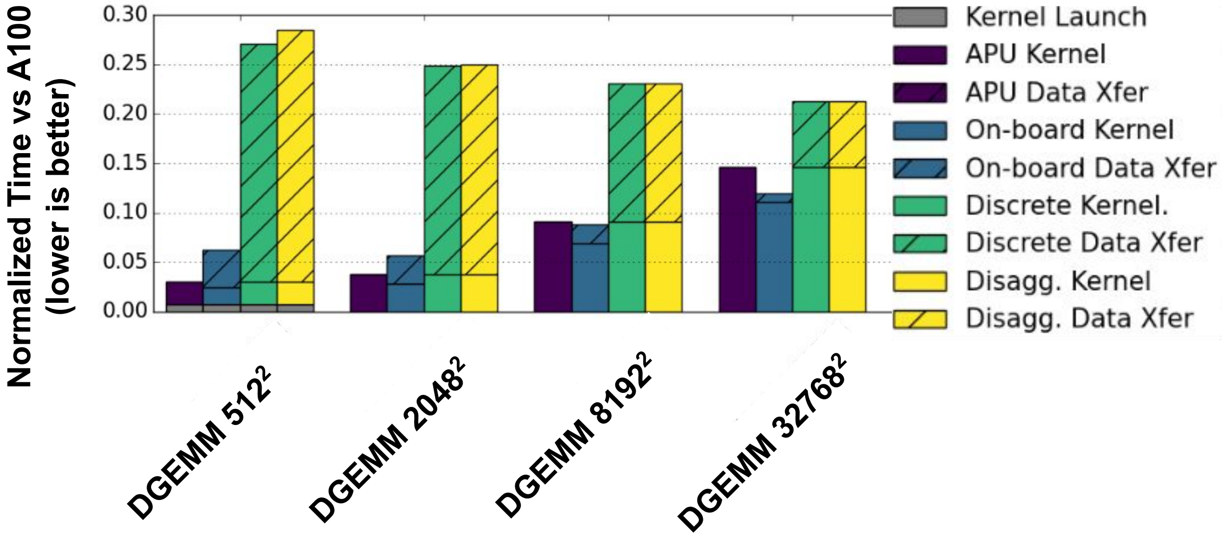
	× FLOPS	GPU FP64	GPU	CPU ↔ GPU	CPU ↔ GPU
Archit. (2026)	(vs. A100)	TFLOPS	Power	Bandwidth	Latency
Disagg. GPU	5.1	102	350W	121GB/s	4us
Discrete GPU	5.1	102	350W	121GB/s	2us
On-board GPU	6.7	133	700W	900GB/s	1us
APU	5.1	102	350W	N/A	N/A

Assumptions made

- 5.1 - 6.7X increase in FLOPS of A100
- For APU, GPU uses 50% of package power (700W)
- 50% power results in 76% FLOPS

Many areas where we could shift the design parameters of each architecture.
These are reasonable and illustrative of the modeling approaches.

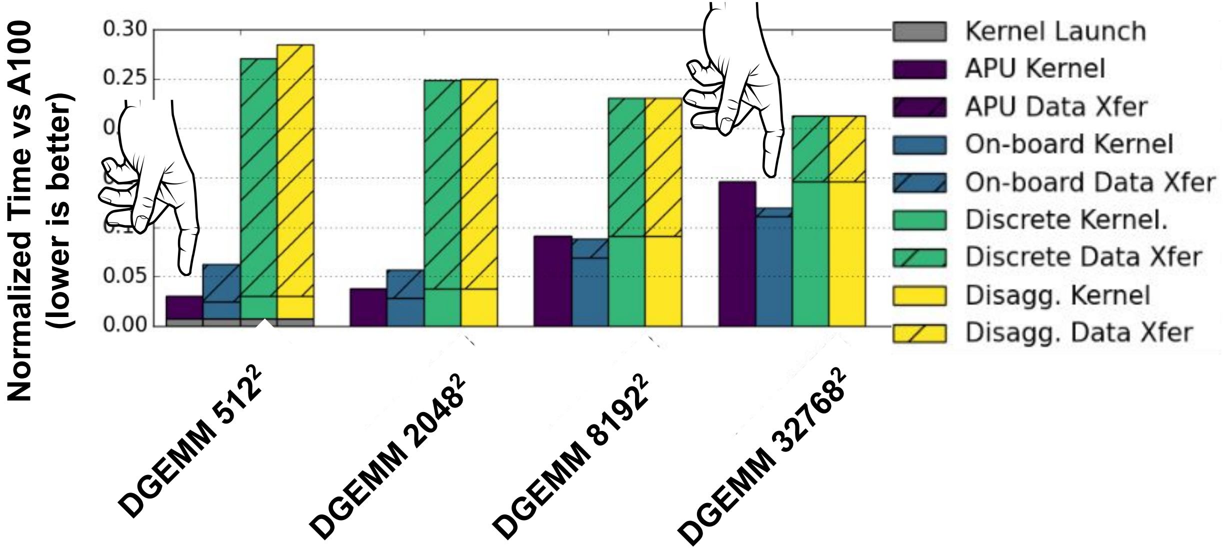
2026 CPU-GPU Coupling Comparison (DGEMM)



2026 CPU-GPU Coupling Comparison (DGEMM)

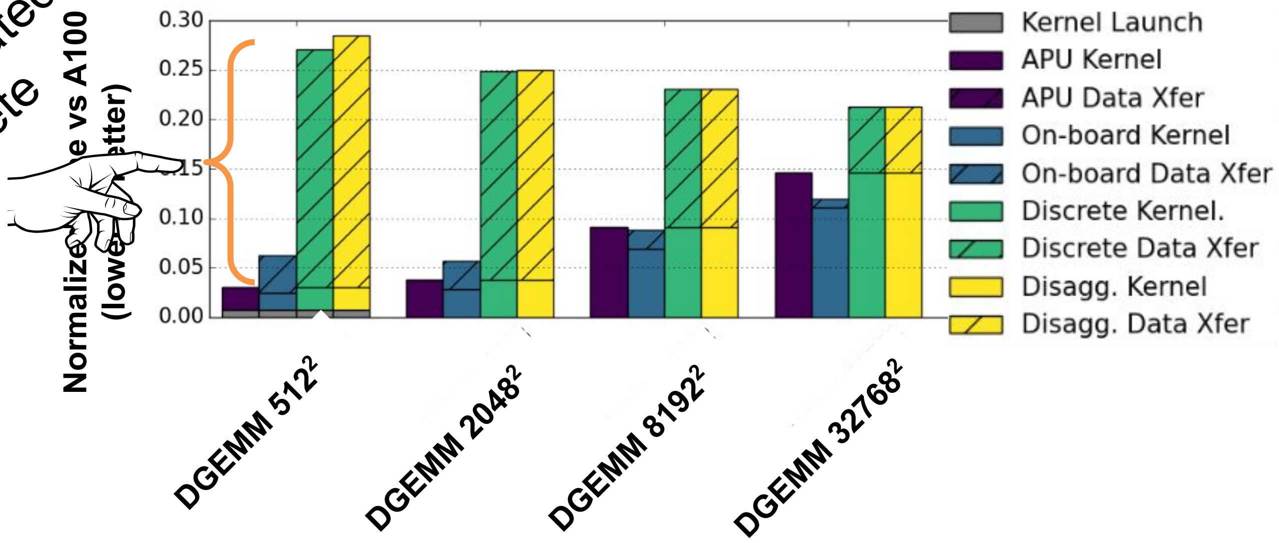
APU greatly benefits small DGEMMs and short-lived kernels

Once computation scales up sufficiently On-board GPU sees benefits of higher FLOPS



2026 CPU-GPU Coupling Comparison (DGEMM)

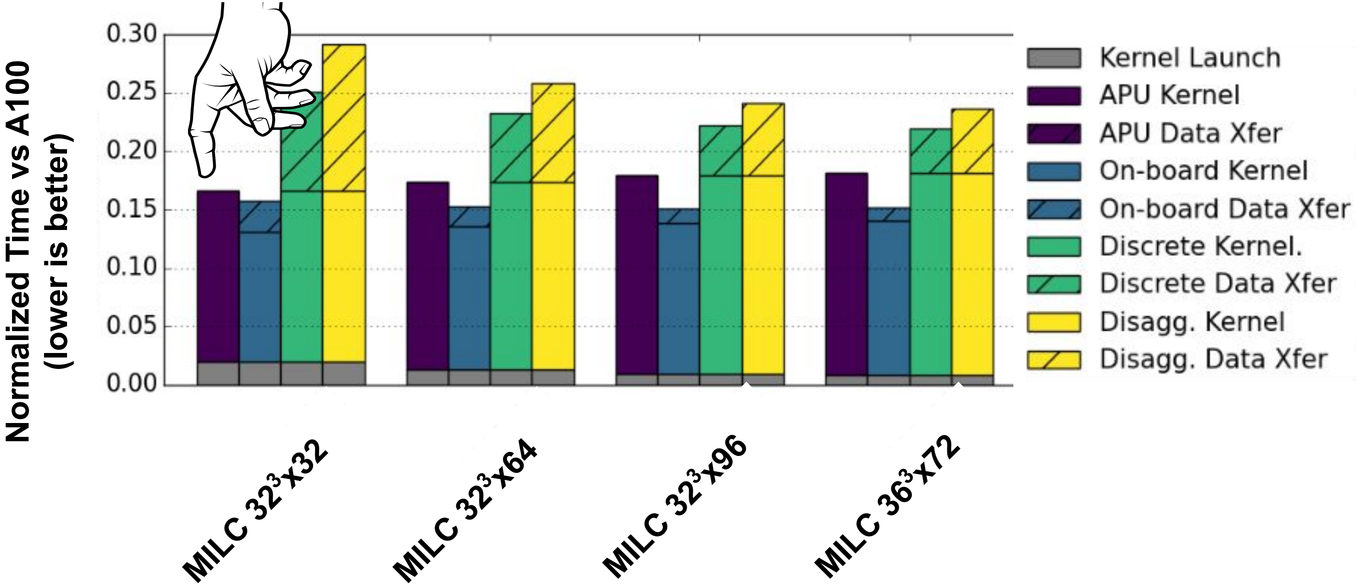
The relatively large data transfers of DGEMM are all benefited by tighter coupling of CPU and GPU



up to 10X speedup over disaggregated and discrete

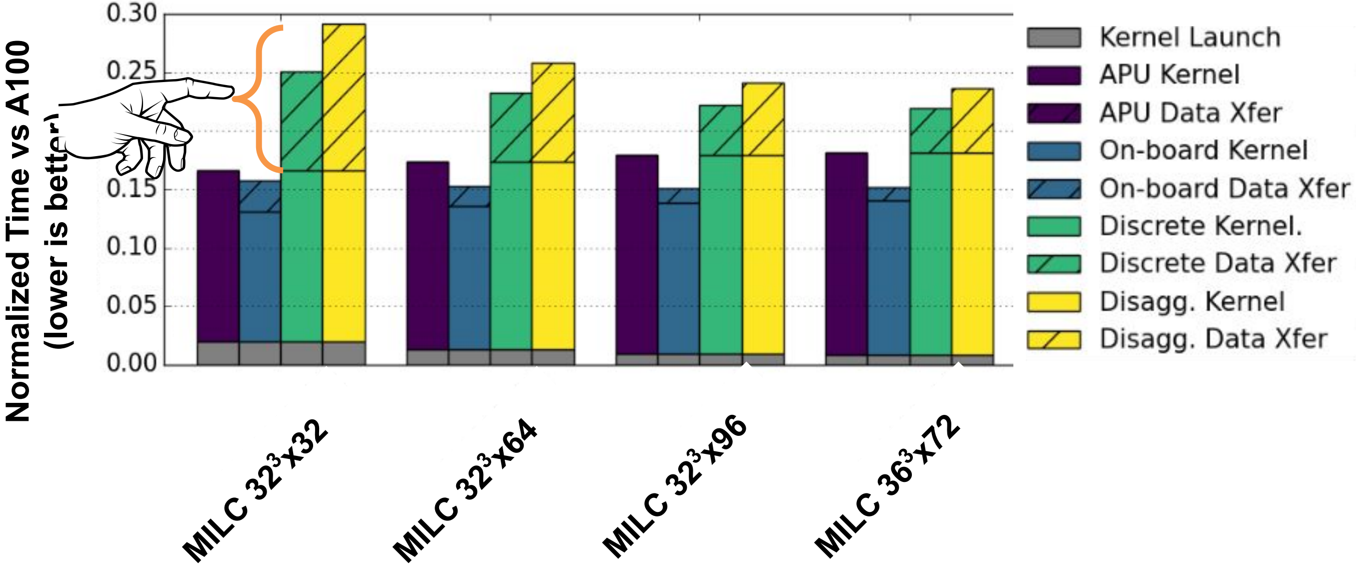
2026 CPU-GPU Coupling Comparison (MILC)

For codes with regular data transfers, APU is competitive with On-board GPU, (up to 17% slower with assumed 350 vs 700 W)



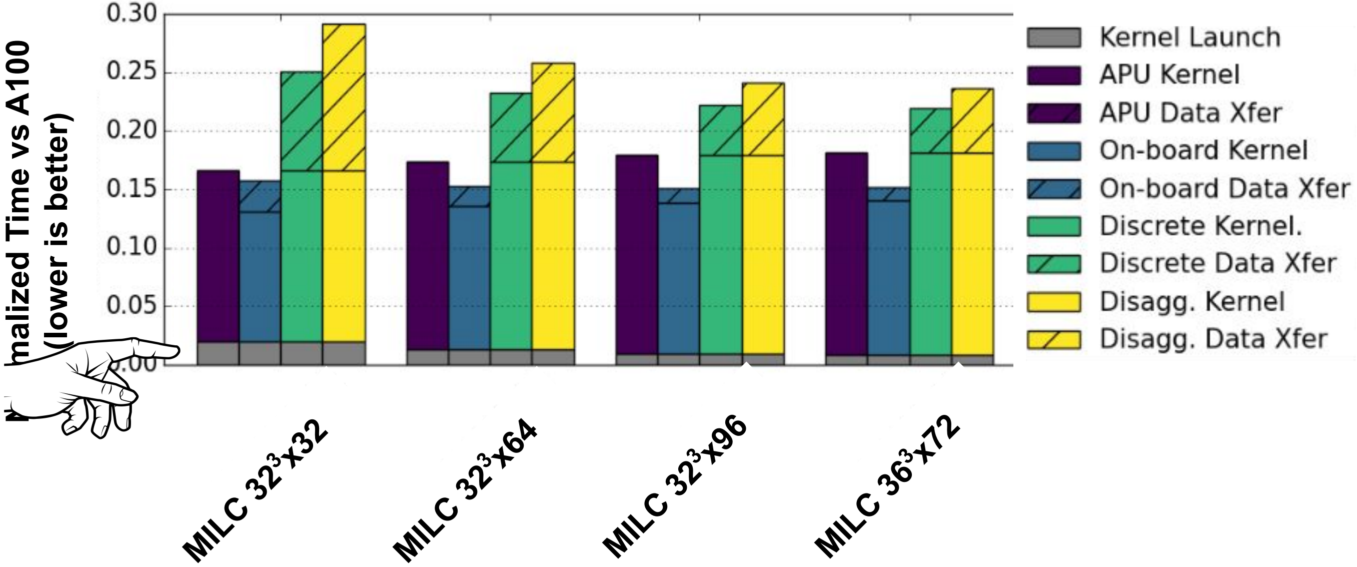
2026 CPU-GPU Coupling Comparison (MILC)

Discrete or Disaggregated twice as slow due to data transfers in MILC



2026 CPU-GPU Coupling Comparison (MILC)

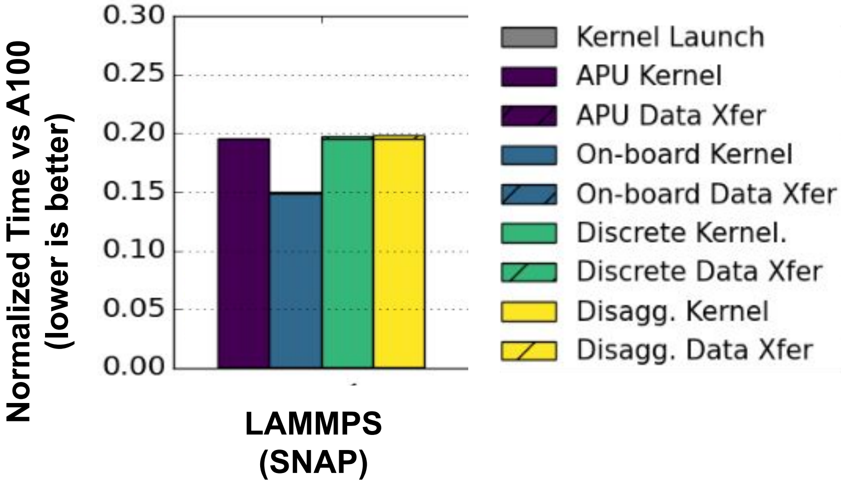
Kernel launches (which were < 1% for A100) are of growing importance.



2026 CPU-GPU Coupling Comparison (LAMMPS-SNAP)

LAMMPS-SNAP has minimal data transfers and is FLOPS limited.

- Discrete and Disaggregated and APU solutions are only limited by the available power.
- On-board is ~25% faster (700W vs 350W)
- For similar workloads, dependent on price, system architects could build a scale-out system of cheaper components



Conclusions

Choices of GPU and accelerator options are increasing as time goes on.

- APU present new opportunities for codes that were overlooked as poor performers on current generation GPUs.
- Crucial to understand power limitations of GPU components on APU

Options for tighter coupling limit opportunities for disaggregation

- Higher bandwidth connectivity (e.g. optics) for disaggregation could help

Understanding the workload of a HPC center is increasingly important

- If GPU prices increase, performance to $ becomes increasingly relevant.
- Tradeoffs between scale-up vs. scale out system

Questions?

tgroves@lbl.gov

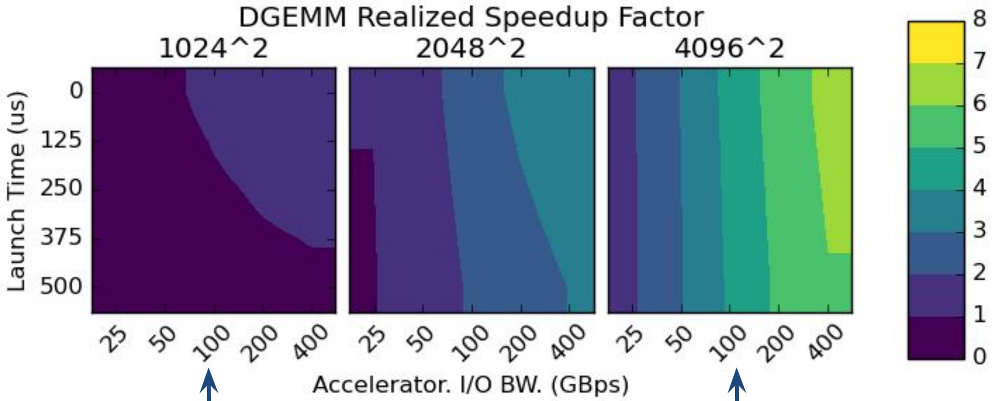
What can our offload model tell us about DGEMM?

Assume, that our CPU computes 8X slower than our A100 GPU

For three different DGEMM sizes (1024^2, 2048^2, 4096^2)

What is the performance impact of:

- increasing or decreasing the launch time, and
- increasing the CPU-to-GPU bandwidth?



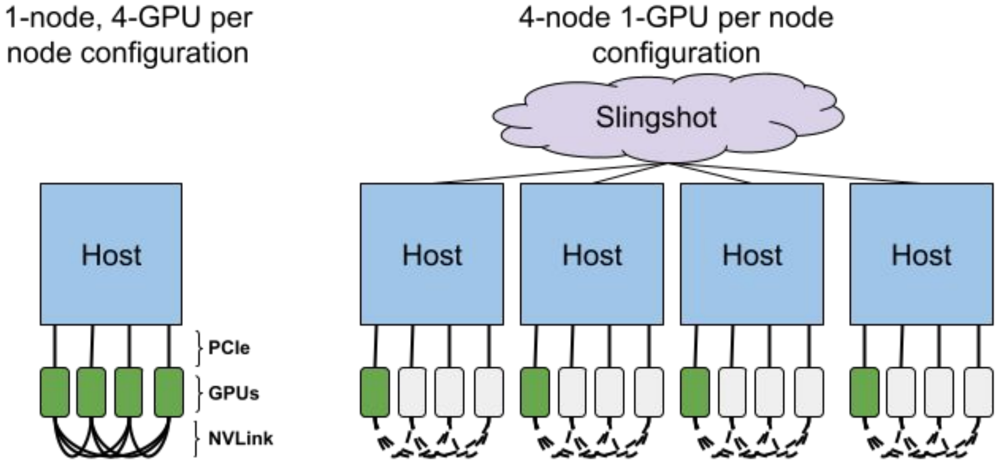
Latency bound data xfer. (Not worth offloading.)

Work (N^3) grows faster than data transfer (N^2).

Example Usage of NEthing: MILC

Lattice QCD generation problem (su3 rhmd hisq)

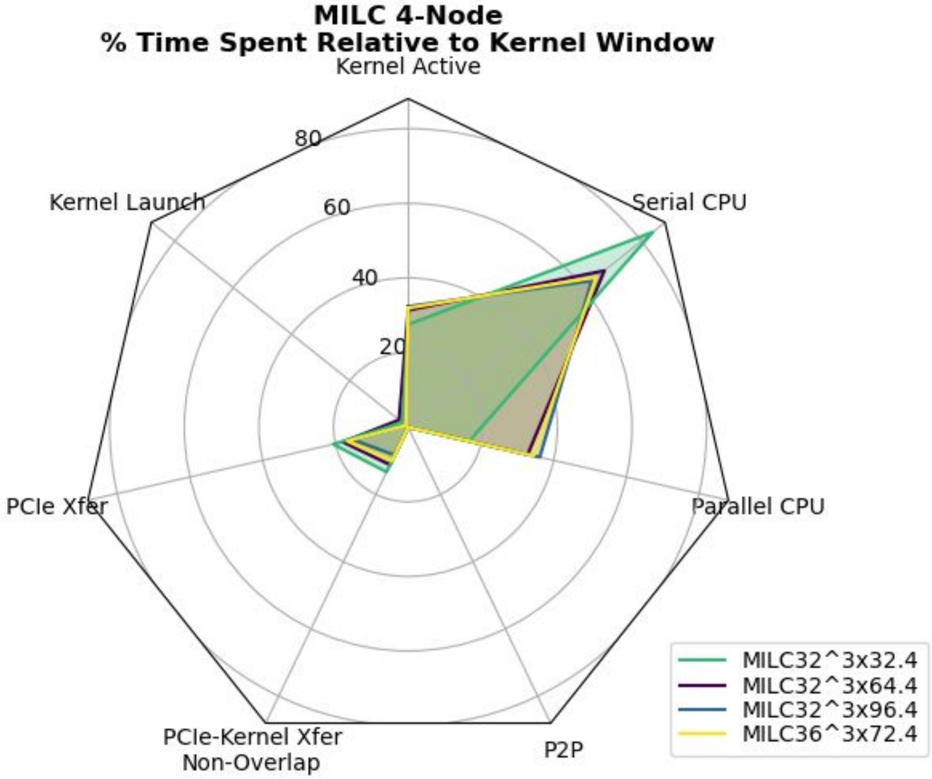
- Nvidia QUDA backend
- Four problem sizes evaluated
- Four A100 GPUs
- Two configurations to measure NVLink and Slingshot usage


1-node, 4-GPU per node configuration
4-node 1-GPU per node configuration
Slingshot
Host
Host
Host
Host
Host
PCIe
GPUs
NVLink

Example Usage of NEthing: MILC Summary

Four Problem Sizes Compared on Four A100's with fairly similar profiles.

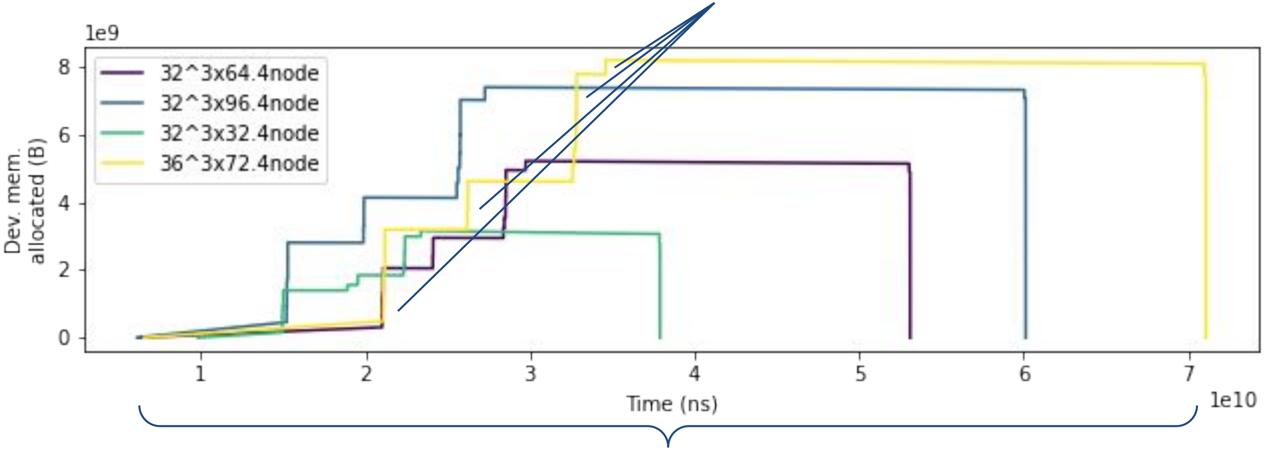
- 15-20% time in PCIe Xfer
- 35-50% of the transfers are overlapping
- <1% time in kernel Launch
- Kernel is active ~30% of the kernel window
- CPU active 60-80% of the time in serial and 40-20% of time in parallel



Example Usage of NEthing: MILC Memory Usage

~5 Memory Allocations

2-8 GB of GPU memory depending on problem



Runtime of ~35-70 seconds depending on problem size

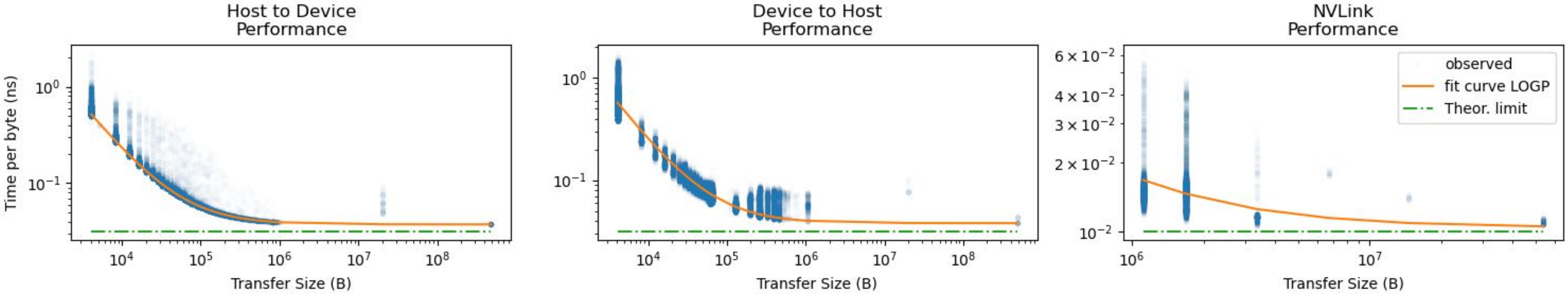
Example Usage of NEthing: MILC Data Transfers

We parameterize our LogGP model by plotting the time per byte for varying message sizes.

- Host to Device PCIe
- Device to Host PCIe
- NVLink

y = *ax* + *b* of each curve

- *b* (y-intercept) is L + o
- *x* is transfer size (bytes)
- *a* is the time per byte (Gap)


Host to Device Performance
Time per byte (ns)
10^0
10^{-1}
10^4
10^5
10^6
10^7
10^8
Transfer Size (B)
Device to Host Performance
10^0
10^{-1}
10^4
10^5
10^6
10^7
10^8
Transfer Size (B)
NVLink Performance
6×10^{-2}
4×10^{-2}
3×10^{-2}
2×10^{-2}
10^{-2}
10^6
10^7
Transfer Size (B)
observed
fit curve LOGP
Theor. limit