Benchmarking HPL and HPCG and beyond

Optimization & Performance Analysis on Heterogeneous Clusters

The Two Benchmarks

- HPL — **Peak Compute Benchmark**
  - Dense linear algebra (LU factorization)
  - Stresses compute pipelines and exposes synchronization costs through panel broadcasts.
- HPCG — **Realistic Memory-Bound Benchmark**
  - Irregular access patterns
  - Dominated by SpMV
  - Stresses memory bandwidth & communication costs

TOP 500
The List.

acm TURING AWARD 1966
CELEBRATING 50 YEARS
OF COMPUTING'S GREATEST ACHIEVEMENTS

The LINPACK Benchmark: Past, Present, and Future*

Jack J. Dongarra†, Piotr Luszczek†, and Antoine Petitet‡

July 2002

Abstract

This paper describes the LINPACK Benchmark [41] and some of its variations commonly used to assess performance of computer systems. Aside from the LINPACK benchmark suite, the TOP500 [43], and the HPL [48] code are presented. The latter is frequently used to obtained results for TOP500 submissions. Information is also given on how to interpret results of the benchmark and how the results fit into performance evaluation process.
Keywords: BLAS, benchmarking, high performance computing, HPL, linear algebra, LINPACK, TOP500

1 Introduction

The original LINPACK Benchmark is, in some sense, an accident. It was originally designed to assist users of the LINPACK package [15] by providing information on execution times required to solve a system of linear equations. The first "LINPACK Benchmark" report appeared as an appendix in the LINPACK Users' Guide [15] in 1979. The appendix comprised of data for one commonly used path in the LINPACK software package. Results were provided for a matrix problem of size 100, on a collection of widely used computers (23 computers in all). This was done so

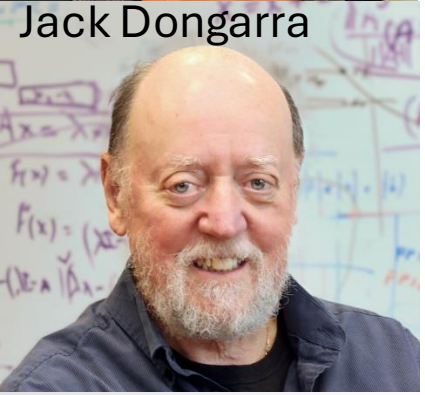
New Mexico

Jack Dongarra
(Far left)

Clev Moler's PhD
Student

Michael Heroux

Jack Dongarra



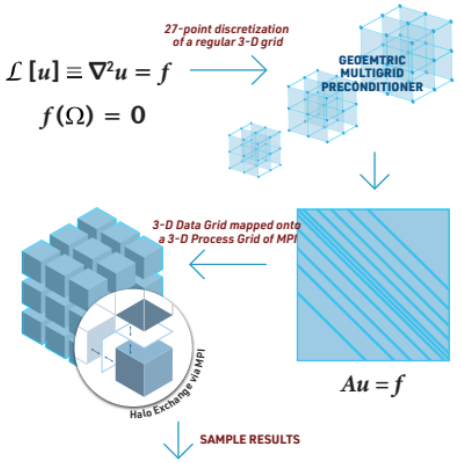
Piotr Luszczek

HPCG

NOVEMBER 2021 TOP 5	SITE	COUNTRY	RMAX PFLOP/S	HPCG PFLOP/S
1 Fugaku	RIKEN	Japan	442	16
2 Summit	DOE/SC/ORNL	USA	149	3
3 Perlmutter	DOE/SC/LBNL	USA	65	2
4 Sierra	DOE/NNSA/LLNL	USA	95	1.8
5 Selene	NVIDIA	USA	64	1.6

The HPC Conjugate Gradient (HPCG) benchmark uses a preconditioned conjugate gradient (PCG) algorithm to measure the performance of HPC platforms with respect to frequently observed, yet challenging, patterns of execution, memory access, and global communication.

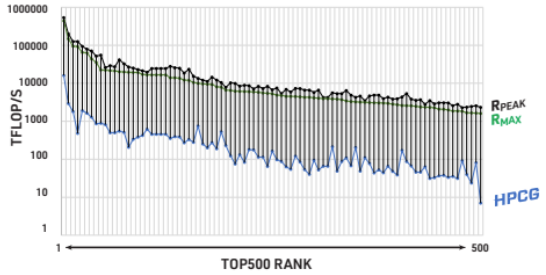
The PCG implementation uses a regular 27-point stencil discretization in 3 dimensions of an elliptic partial differential equation (PDE) with zero Dirichlet boundary condition. The 3-D domain is scaled to fill a 3-D virtual process grid of all available MPI process ranks. The CG iteration includes a local and symmetric Gauss-Seidel preconditioner, which computes a forward and a back solve with a triangular matrix. All of these features combined allow HPCG to deliver a more accurate performance metric for modern HPC hardware architectures.



SAMPLE RESULTS

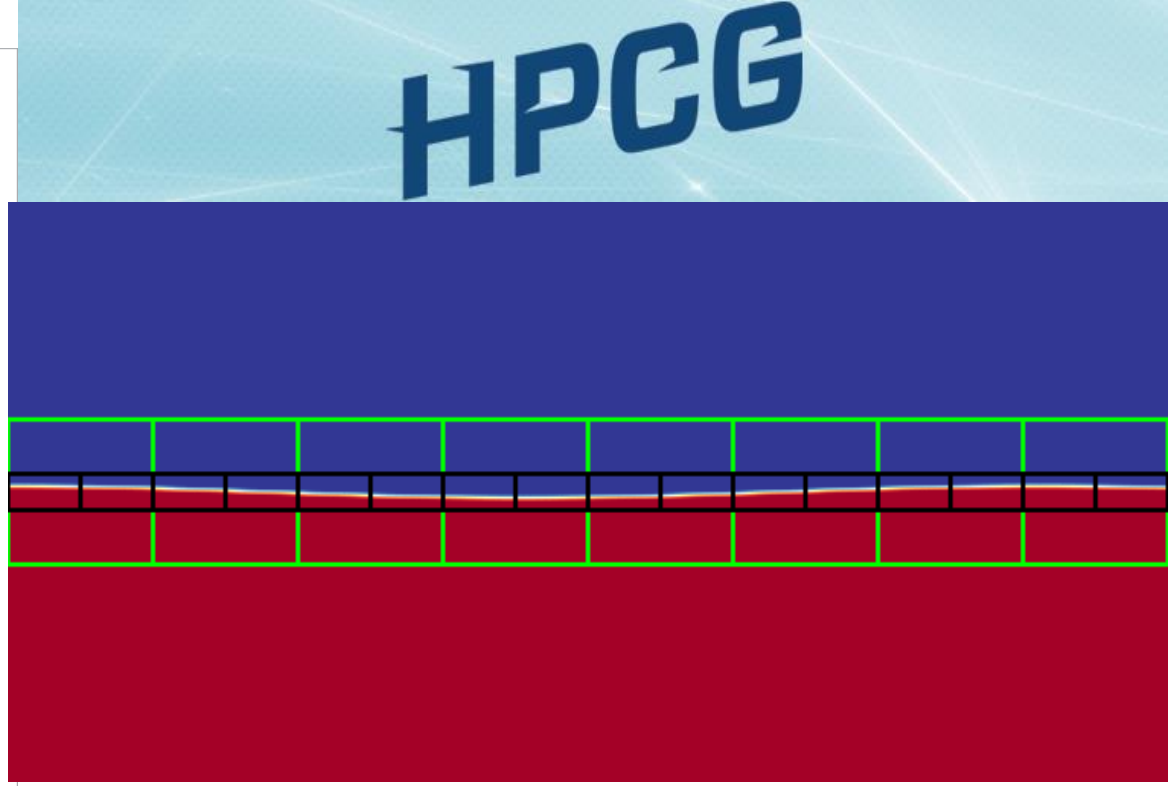
PRECONDITIONED CONJUGATE GRADIENT SOLVER

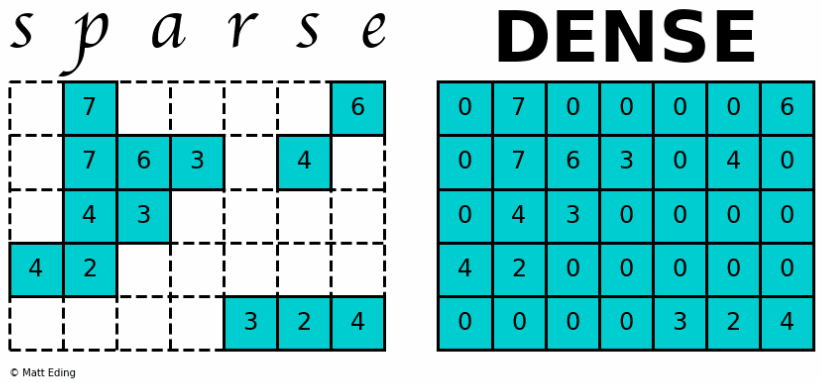
$p_0 \leftarrow x_0$, $r_0 \leftarrow b - Ap_0$
for $i = 1, 2,$ to max_iterations **do**
  $z_i \leftarrow M^{-1} r_{i-1}$
  if $i = 1$ **then**
    $p_i \leftarrow z_i$
    $\alpha_i \leftarrow$ dot_prod(r_{i-1}, z_i)
  else
    $\alpha_i \leftarrow$ dot_prod(r_{i-1}, z_i)
    $\beta_i \leftarrow \alpha_i / \alpha_{i-1}$
    $p_i \leftarrow \beta_i p_{i-1} + z_i$
  end if
  $\alpha_i \leftarrow$ dot_prod(r_{i-1}, z_i)/dot_prod(p_i, Ap_i)
  $x_{i+1} \leftarrow x_i + \alpha_i p_i$
  $r_i \leftarrow r_{i-1} - \alpha_i A p_i$
  if $\|r_i\|_2 <$ tolerance **then**
    STOP
  end if
end for



FIND OUT MORE AT
http://www.hpcg-benchmark.org/

ICL INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY OF TENNESSEE KNOXVILLE
Sandia National Laboratories
SPONSORED BY U.S. DEPARTMENT OF ENERGY NNSA





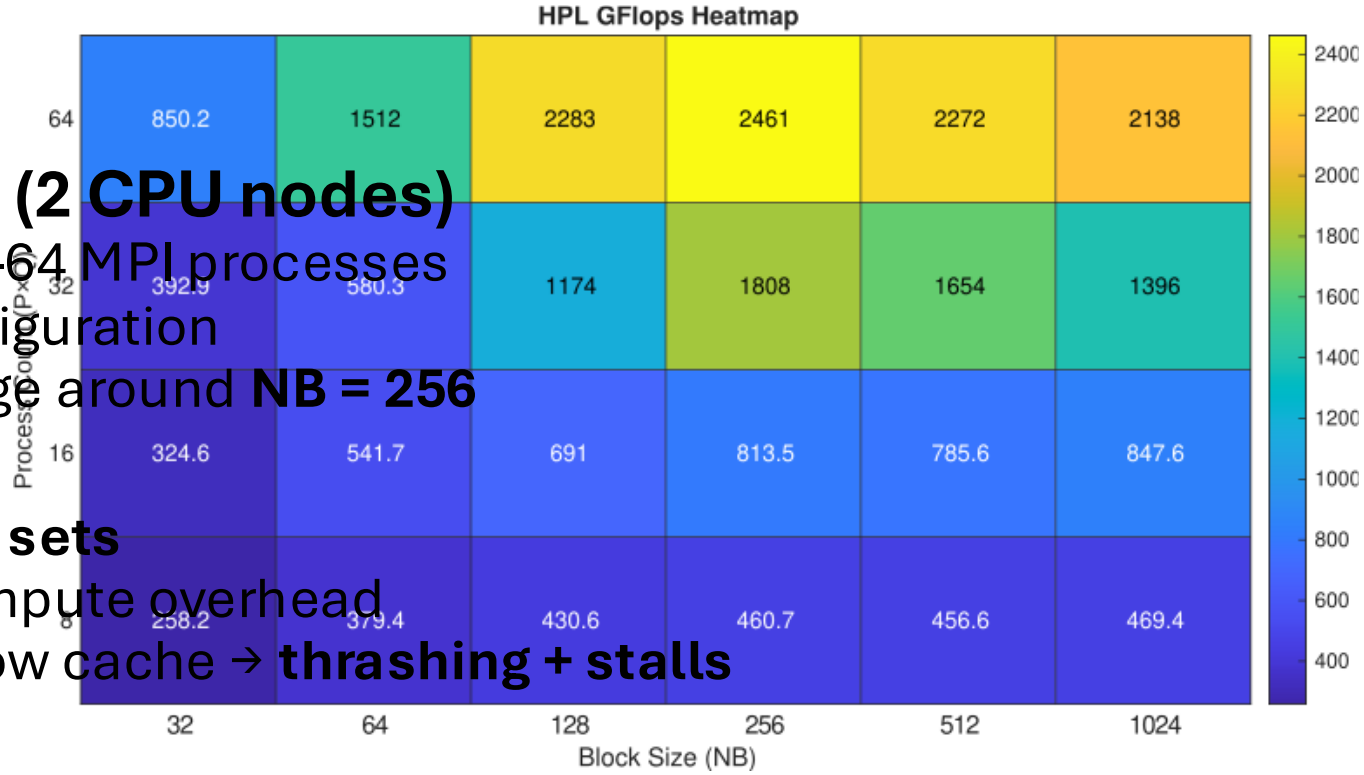
HPL on the Clusters

- **Team CPU Cluster — Baseline CPU Architecture**
  - Dual-socket Intel Sandy Bridge
  - 16 cores, large L3 Cache
- **Hopper CPU Cluster — Modern CPU Node**
  - Dual-socket Intel Xeon Gold 6226R
  - 32 cores, high-bandwidth DDR4
  - Mellanox CX interconnect
- **Easley GPU Node — Heterogeneous Accelerator System**
  - 2× NVIDIA H100 NVL GPUs (HBM3 @ 7.7 TB/s)
  - NVLink + NCCL + NVSHMEM communication stack
  - Ideal for GPU-heavy memory intensive workloads

HPL Optimization: What We Learned About Compute-Bound Scaling

- **CPU Optimization (Team + Hopper)**
  - Throughput hotspot consistently at **NB = 256**
  - Larger blocks (NB ≥ 1024) → **L3 cache thrashing + degraded locality**
- **GPU Optimization (Easley H100 NVL)**
  - Achieved **73.1 TFLOPS** (61% of theoretical peak)
  - Requires large NB (**1536**) to saturate tensor cores
  - High memory usage (~95% HBM3) → better occupancy
- **Architectural Insight**
  - CPU performance hinges on **cache fit**
  - GPU performance hinges on **kernel throughput + synchronization jitter**
  - HPL characterizes the ***compute ceiling*** of each architecture — but not real application behavior.

HPL on CPUs: Throughput Hotspot at NB = 256



- **Systematic sweep on Hopper (2 CPU nodes)**
  - Varied **NB = 32 → 1024**, across 8–64 MPI processes
  - Measured GFLOPS for each configuration
  - Observed clear performance ridge around **NB = 256**
- **Why NB = 256 Performs Best**
  - Fits cleanly in **L3 cache working sets**
  - Balances communication vs compute overhead
  - Larger blocks (NB ≥ 1024) overflow cache → **thrashing + stalls**
- **Key Result**
  - Peak CPU throughput: **2.46 TFLOPS** at **NB = 256**, **64 processes**, **1:1 P×Q grid**
  - This plateau appears consistently across CPUs, confirming that cache behavior—not raw FLOPs—dominates HPL scaling

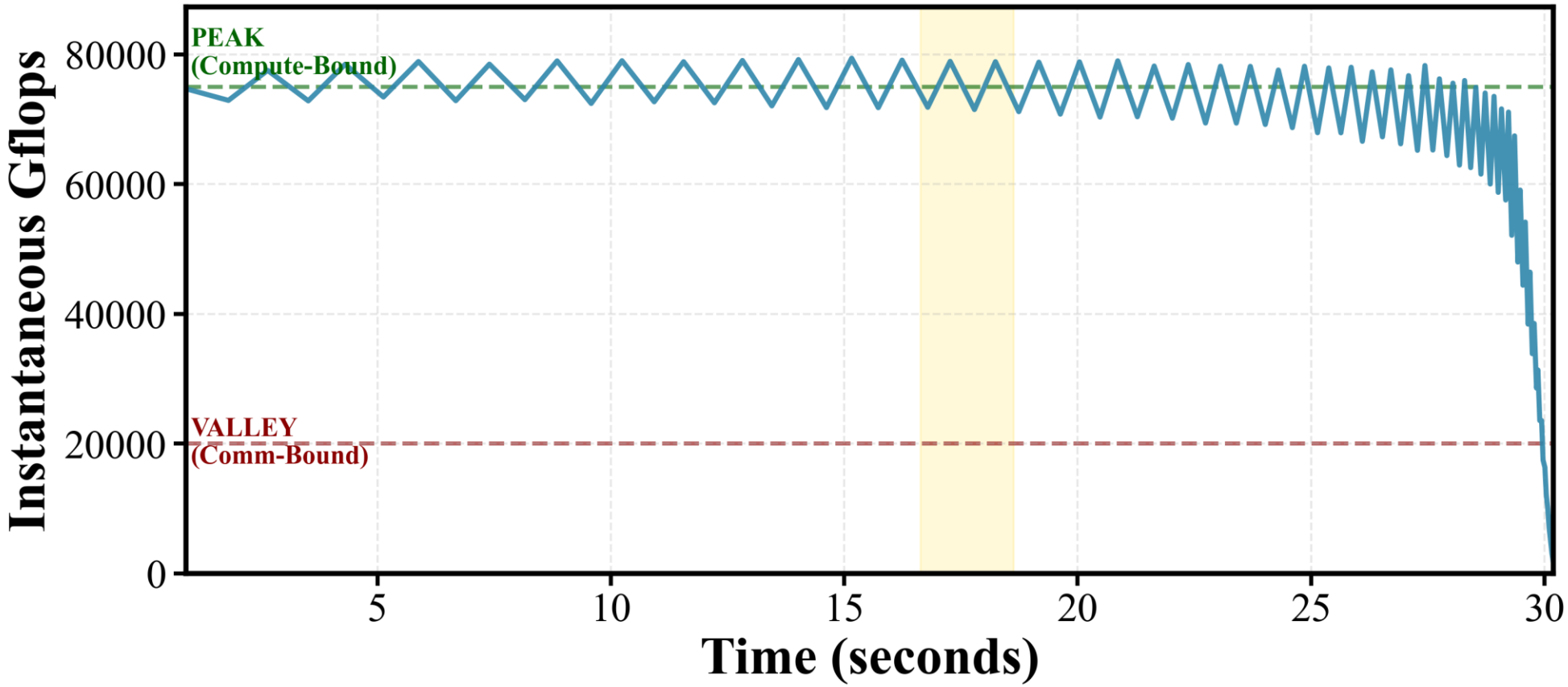
HPL on H100 GPUs: High-Frequency GFLOPS Oscillations

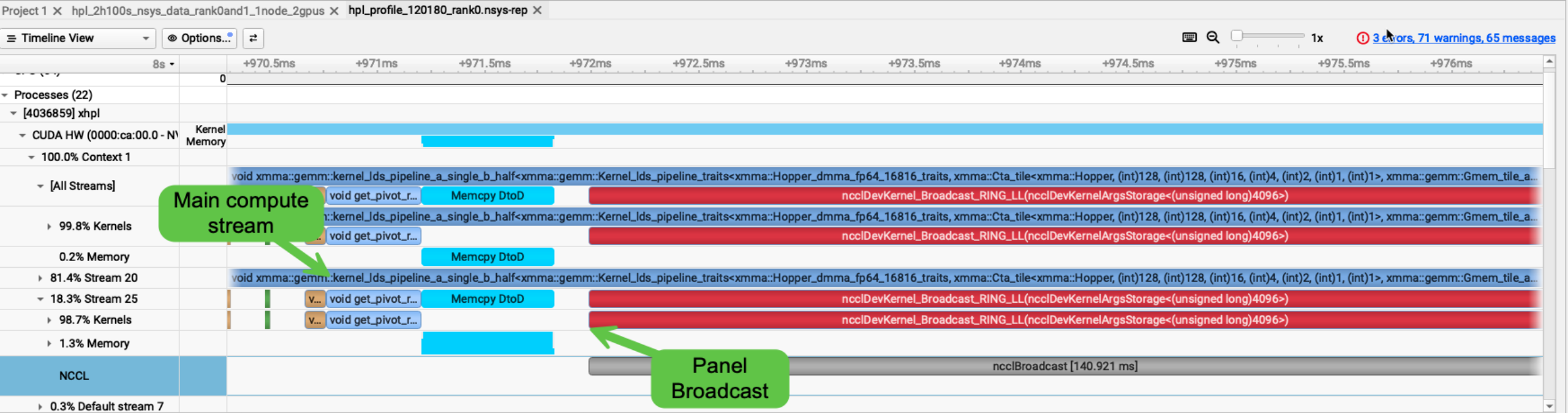
- Observed:
  - 73.1 TFLOPS sustained
  - Rapid oscillations: **~70k ↔ 80k GFLOPS**
  - Aligned with HPL algorithm phases
- Why?
  - **Trailing updates:** compute-intensive → GFLOPS spike
  - **Panel broadcasts:** synchronization wait → GFLOPS dip
- Key Insight
  - Even in dense LA, **communication phases dominate overall performance**.

Kernel Jitter Analysis: Root Cause of HPL Oscillations

- **Nsight Systems Profiling Findings**
  - The primary compute kernel (kernel_lds_pipeline) shows **extreme execution-time variance**
  - **Coefficient of variation > 460%** across iterations
  - Compute kernel jitter aligns precisely with **GFLOPS peaks and valleys**
- **Interpretation**
  - During **trailing updates**, the compute kernel runs at full throughput → **GFLOPS spike**
  - During **panel factorization + broadcast**, GPU threads stall → **GFLOPS dip**
  - Jitter is a *structural consequence* of HPL's alternating algorithmic phases
- **Key Insight**
  - **Performance oscillations originate from kernel-level synchronization behavior — not hardware instability or noise.**

HPL Performance Oscillation Pattern





Stats System View

CUDA API Summary
CUDA API Trace
CUDA GPU Kernel Summary
CUDA GPU Kernel/Grid/Block Summary
CUDA GPU MemOps Summary (by Size)
CUDA GPU MemOps Summary (by Time)
CUDA GPU Summary (Kernels/MemOps)
CUDA GPU Trace
CUDA Kernel Launch & Exec Time Summary
CUDA Kernel Launch & Exec Time Trace
CUDA Summary (API/Kernels/MemOps)
DX11 PIX Range Summary
DX12 GPU Command List PIX Ranges Summary
DX12 PIX Range Summary
MPI Event Summary
MPI Event Trace
MPI Message Size Summary
NVTX GPU Projection Summary

CLI command::
nsys stats -r cuda_api_sum "/Users/afasulo3/Desktop/hpc_project1/nsys data h100's/hpl_profile_120180_rank0.sqlite"

Time	Total Time	Num Calls	Avg	Med	Min	Max	StdDev	Name
86.7%	30.053 s	95	316.343 ms	244.060 ms	25.709 ms	905.613 ms	262.690 ms	cudaEventSynchronize
11.6%	4.017 s	1975	2.034 ms	38.629 µs	16.943 µs	285.514 ms	9.330 ms	cuLibraryLoadData
0.4%	129.496 ms	44	2.943 ms	5.558 µs	1.929 µs	83.772 ms	13.227 ms	cudaDeviceSynchronize
0.4%	122.382 ms	93	1.316 ms	4.692 µs	829 ns	51.840 ms	7.417 ms	cudaStreamSynchronize
0.2%	75.266 ms	44	1.711 ms	6.875 µs	323 ns	35.089 ms	7.257 ms	cudaFree
0.2%	57.068 ms	398	143.386 µs	49.399 µs	28.631 µs	32.324 ms	1.619 ms	cuMemSetAccess
0.1%	41.214 ms	11866	3.473 µs	2.964 µs	2.772 µs	2.916 ms	26.754 µs	cudaLaunchKernel
0.1%	28.688 ms	196	146.368 µs	3.339 µs	2.898 µs	26.359 ms	1.885 ms	cuLaunchKernelEx
0.1%	26.233 ms	512	51.235 µs	4.163 µs	2.403 µs	23.841 ms	1.053 ms	cudaMemcpyAsync
0.0%	16.617 ms	8	2.077 ms	1.612 ms	1.262 ms	4.155 ms	967.494 µs	cudaGetDeviceProperties_v2_v1200
0.0%	12.987 ms	34	381.962 µs	4.874 µs	3.345 µs	12.160 ms	2.082 ms	cudaMalloc
0.0%	12.929 ms	560	23.087 µs	5.486 µs	121 ns	113.336 µs	26.913 µs	cuMemRelease
0.0%	12.767 ms	278	45.923 µs	44.356 µs	22.444 µs	374.184 µs	28.703 µs	cuMemUnmap
0.0%	11.907 ms	3613	3.295 µs	3.177 µs	2.925 µs	19.428 µs	712 ns	cudaLaunchKernelExC_v11060

Why HPL Alone Isn’t Enough: Transitioning to HPCG

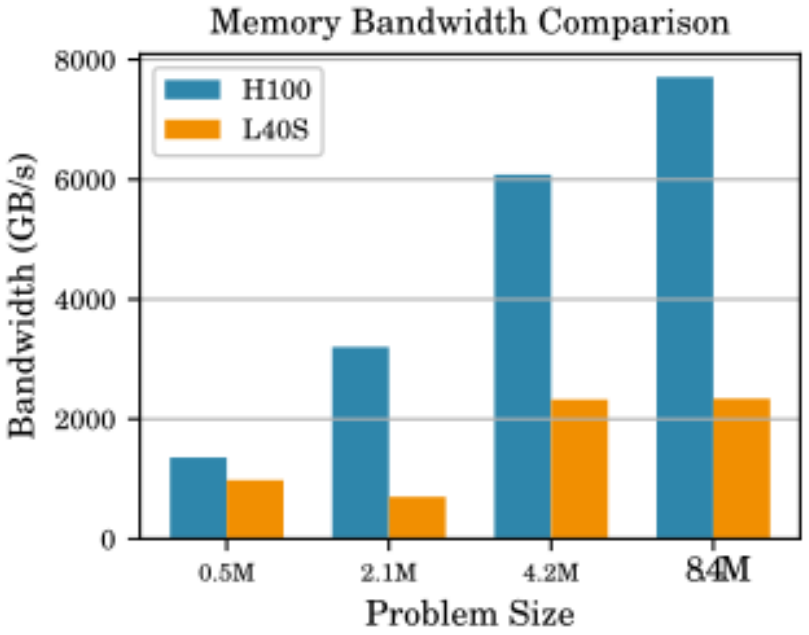
- **Limitations of HPL (Dense, Compute-Bound)**
  - Optimistic benchmark: measures *peak FLOP throughput*
  - Dominated by **dense GEMM** with ideal locality
  - Communication pattern: **predictable panel broadcasts**
  - Not representative of LANL’s sparse, irregular physics workloads
- **Motivation for HPCG**
  - Designed to mimic **real application bottlenecks (SpVM)**
  - Dominated by **memory bandwidth** and **network behavior**
  - Provides a lower-but-more-realistic view of system performance
- **Key Takeaway**
  - **HPL tells us how fast our hardware *could* be.**
    HPCG tells us how fast it *will* be for real scientific workloads.

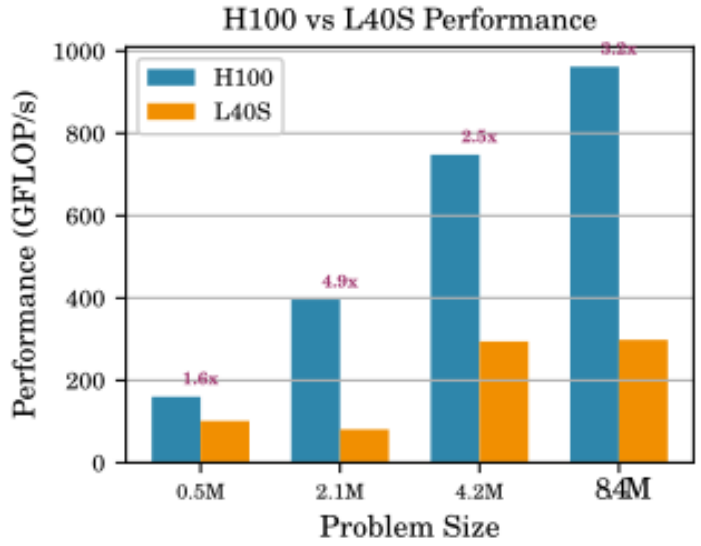
HPCG Results Overview: Key Performance Findings

- **CPU Scaling**
  - Strong scaling: sublinear due to communication
  - Weak scaling: near-linear GFLOPS growth
  - Behavior matches Amdahl/Gustafson predictions
- **GPU Scaling**
  - For weak scaling (256^3 per GPU):
    - **571 → 1095 GFLOPS**
    - **96% efficiency**
- **Key Insight**
  - HPCG performance is tied to **memory bandwidth** and **communication latency**, not FLOPs.

HPCG H100 Results

- **Result:** H100 is **3.06× faster** than L40S
  Memory Bandwidth: HBM3 = 7.7 TB/s vs. GDDR6 = 2.3 TB/s (**3.3× higher**)
- **Conclusion:**
  HPCG performance correlates almost linearly with memory bandwidth.





Takeaways: Understanding Performance Across Architectures

- **HPL teaches us:**
  - Compute-bound limits
  - Synchronization bottlenecks
  - GPU kernel behavior
- **HPCG teaches us:**
  - Memory bandwidth is often the limiting factor
  - Communication patterns define scalability
- **Overall:**
  Performance = interaction of **algorithm + architecture + runtime environment**.

Closing

- **I’m excited about the opportunity to contribute to LANL’s HPC ecosystem — building tools, runtimes, and systems that make large-scale scientific computing faster, more predictable, and easier for researchers to use.**
- My goal is to do great work and make an impact on the team, contributing to LANL’s overall mission.