Efficient Spectral Transforms On NVIDIA Hardware

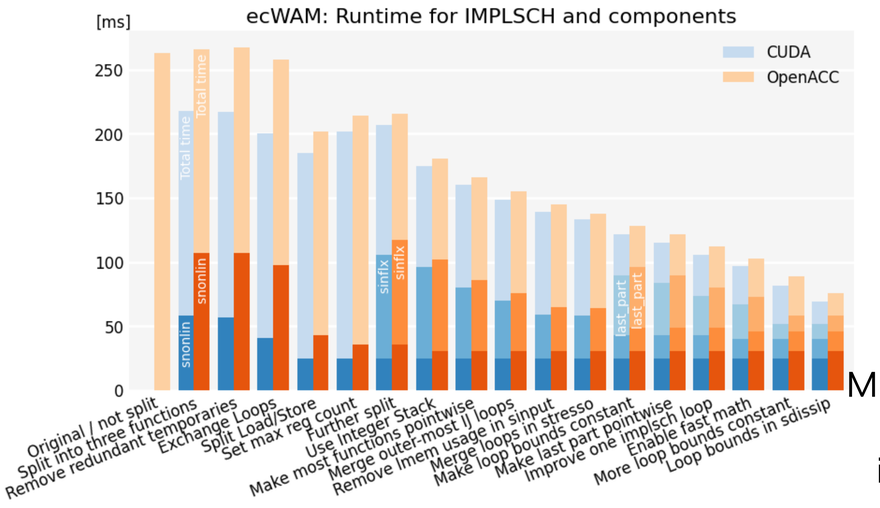
Lukas Mosimann, Devtech HPC, NVIDIA, lmosimann@nvidia.com

ECMWF HPC Workshop, 16. September 2025

Ongoing Collaboration via feedback loop

Efficient collaboration between ECMWF and NVIDIA via separation of concerns

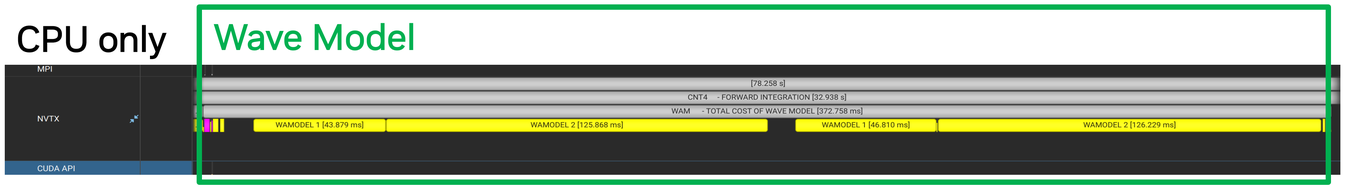
Optimizations in a standalone (A100 GPU)



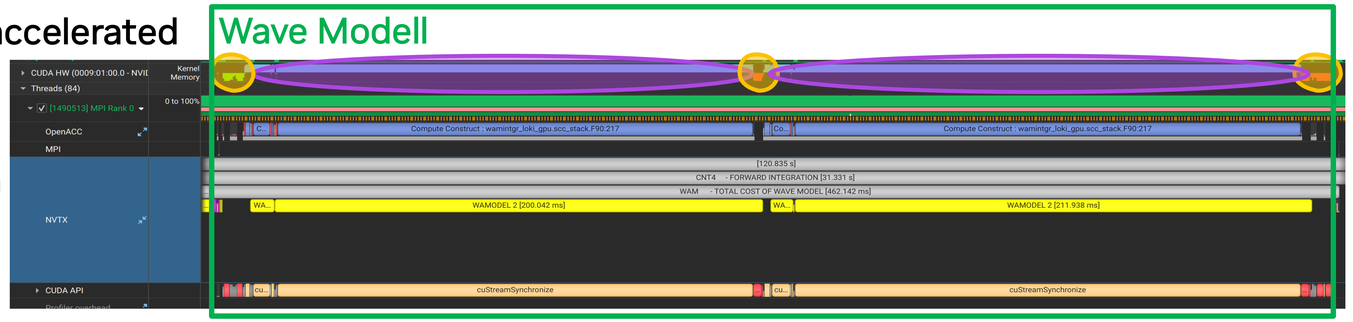
Manual optimization lead to improvements in Loki

Performance in full application

CPU only



GPU-accelerated



Data Transfers H2D/D2H Actual GPU Computations

Performance assessment in full application context shows hot spots for further improvements

NVIDIAs engagement in the IFS/ARPEGE ecosystem

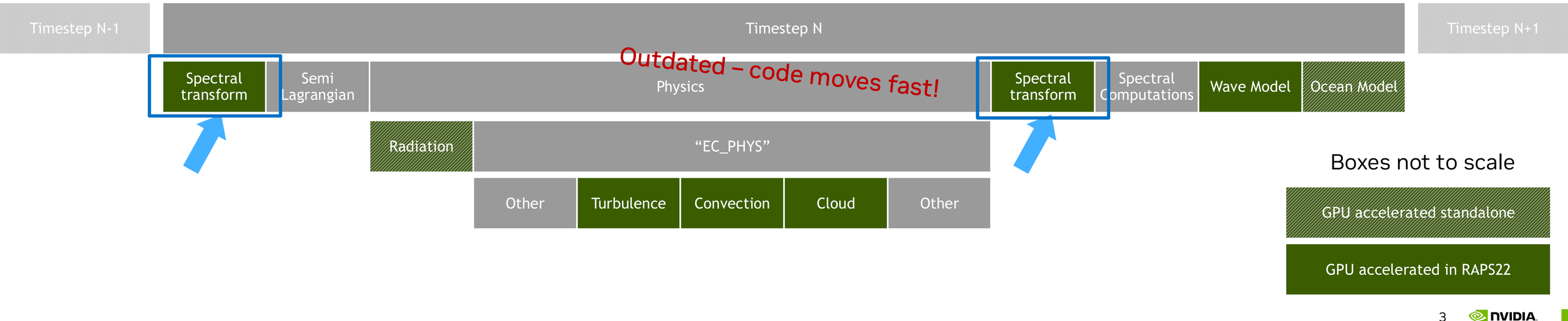
Contributing to the ongoing GPU acceleration projects

- Ongoing collaboration with ECMWF for almost a decade
- Different components require different approaches

Spectral transform (ecTrans)
- CUDA Libraries (cuBLAS, cuFFT)
- OpenACC
- CUDA

Radiation / Ocean Model
- OpenACC
- Unified memory

Physics / Wave Model
- Source-to-Source Translation using Loki
- OpenACC
- Unified memory

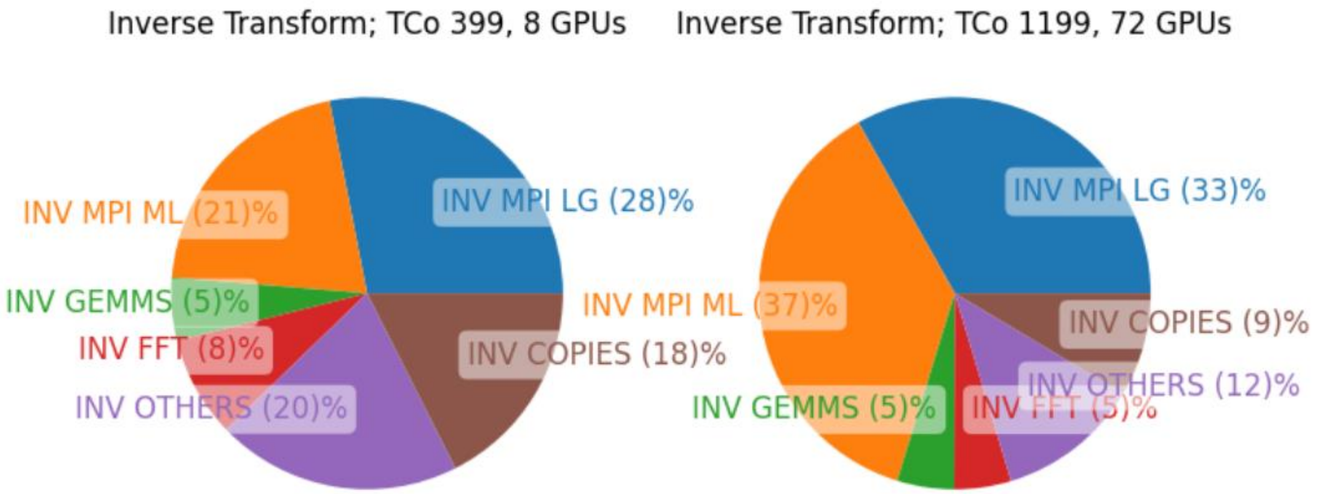


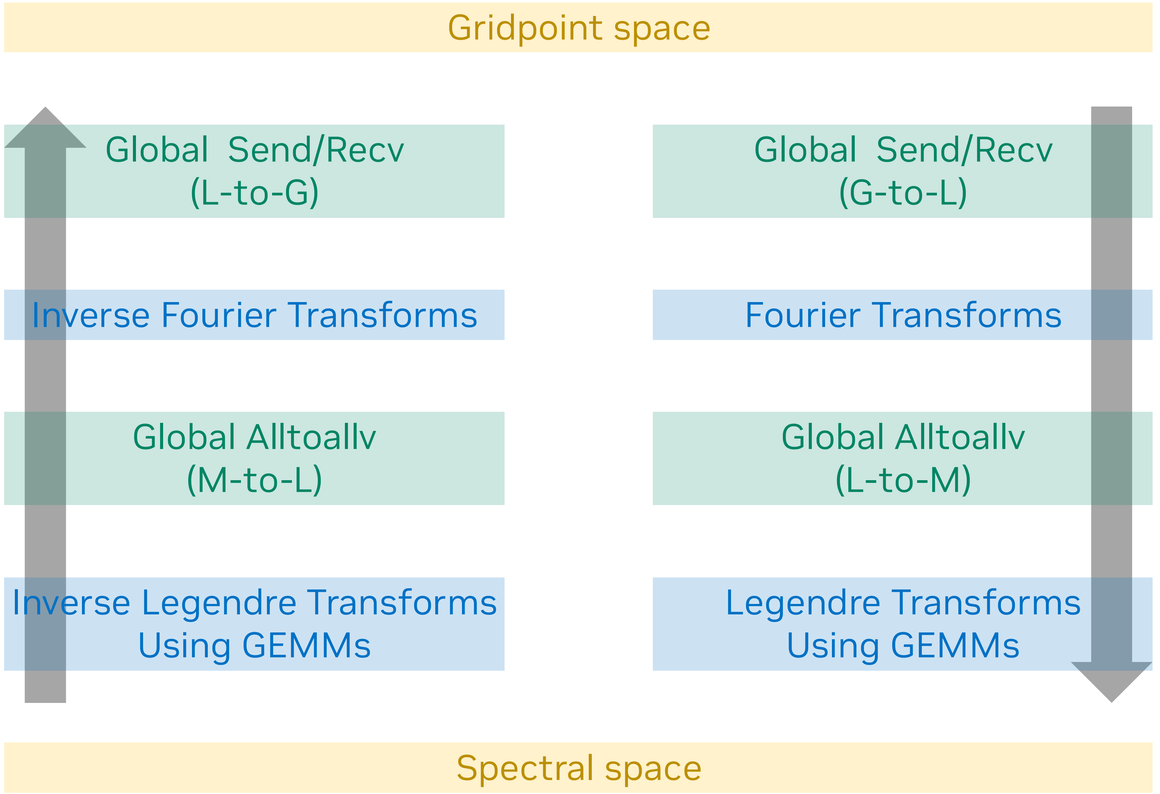
Deep dive into the spectral transform

Various bottlenecks at different resolutions

- Hundreds of small FFTs and GEMMs of various sizes
  → CUDA Graphs
- Non-linear scaling of FFTs [O(N log N)] / GEMMs [O(N^3)]
  → Emulated FP32 for GEMMs
- Large Alltoallv communications limit the weak scaling
  - Local domain remains constant / # ranks increases
  - Avg message size decrease with more ranks
  - Avg number of messages increases with more ranks

  → GB200 NVL72 systems
- Bottlenecks shift a lot depending on # ranks and resolution

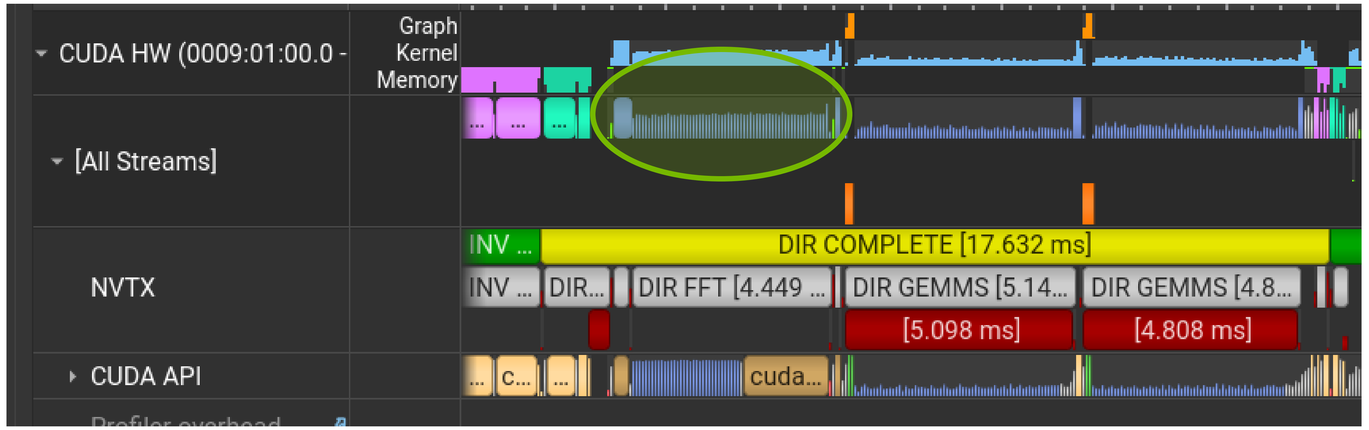




Optimizing Fourier and Legendre transforms

Many tiny kernels turn out to be problematic

- Using Nsight System to spot the bottlenecks
- Apply an FFT for each latitude
  - Each FFT usually has several kernels
  - Easily many hundreds kernels with very bad occupancy

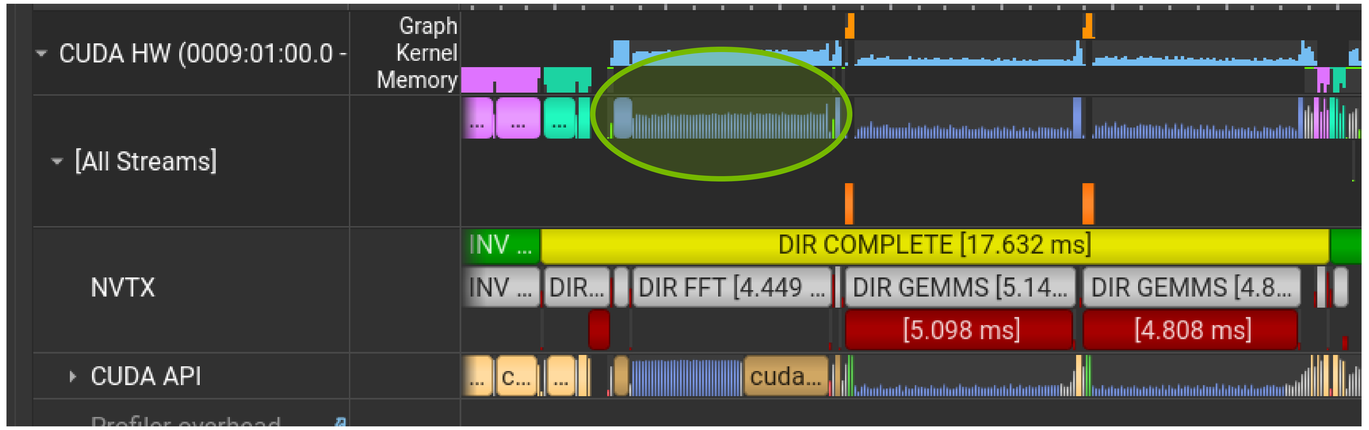


nsys profile –t cuda,nvtx,openacc,mpi –o out –f true ./exe

Optimizing Fourier and Legendre transforms

Many tiny kernels turn out to be problematic

- Using Nsight System to spot the bottlenecks
- Apply an FFT for each latitude
  - Each FFT usually has several kernels
  - Easily many hundreds kernels with very bad occupancy
- Using CUDA Graphs will benefit for two reasons:
  - *Very short kernels*: Kernel launch overhead is more than kernel runtime
    ➔ CUDA Graphs reduce the launch overhead
  - *Very small kernels*: Kernels only need a fraction of the device
    ➔ Launch kernels into many different streams

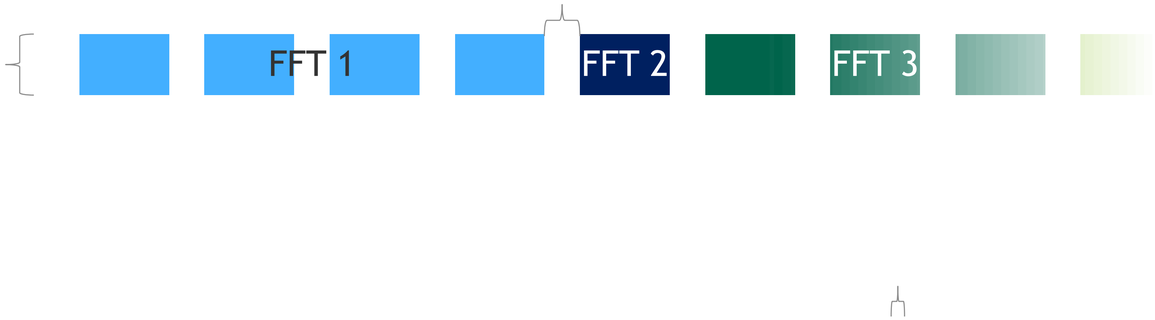


nsys profile –t cuda,nvtx,openacc,mpi –o out –f true ./exe

Using CUDA graphs to optimize code

Capturing mode to create the CUDA graphs

```
template <class Type, cufftType Direction>
void run_group(typename Type::real *data_real,
               typename Type::cmplx *data_complex, int resol_id, int kfield,
               int *loens, int *nmen, int64_t *offsets, int nfft) {
  static constexpr bool is_forward =
    Direction == CUFFT_R2C || Direction == CUFFT_D2Z;

  auto plans = plan_all<Type, Direction>(resol_id, kfield, loens, nfft, offsets);

  for (auto &plan : plans)
    plan.exec(data_real, data_complex);

  cudaDeviceSynchronize();
}
```



Using CUDA graphs to optimize code

Capturing mode to create the CUDA graphs

```
template <class Type, cufftType Direction>
void run_group(typename Type::real *data_real,
               typename Type::cmplx *data_complex, int resol_id, int kfield,
               int *loens, int *nmen, int64_t *offsets, int nfft) {
  static constexpr bool is_forward =
    Direction == CUFFT_R2C || Direction == CUFFT_D2Z;

  auto plans = plan_all<Type, Direction>(resol_id, kfield, loens, nfft, offsets);

  for (auto &plan : plans)
    plan.exec(data_real, data_complex);

  cudaDeviceSynchronize();
}
```

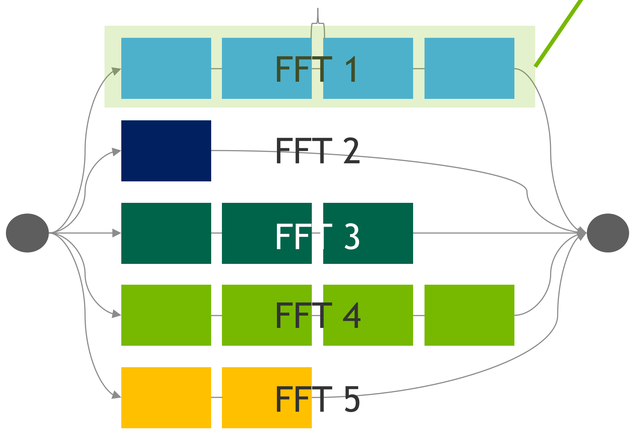

```
template <class Type, cufftType Direction>
void run_group_graph(typename Type::real *data_real,
                     typename Type::cmplx *data_complex, int resol_id,
                     int kfield, int *loens, int *nmen, int64_t *offsets, int nfft) {
  static constexpr bool is_forward =
    Direction == CUFFT_R2C || Direction == CUFFT_D2Z;

  auto key = cache_key{resol_id, kfield};
  auto graph = graphCache.find(key);
  if (graph == graphCache.end()) {
    auto plans = plan_all<Type, Direction>(resol_id, kfield, loens, nfft,
                                           offsets, use_transposition);

    cudaStream_t stream;
    cudaStreamCreate(&stream);
    for (auto &plan : plans) // set the streams
      plan.set_stream(stream);

    cudaGraph_t new_graph;
    cudaGraphCreate(&new_graph, 0);
    for (auto &plan : plans) {
      cudaStreamBeginCapture(stream, cudaStreamCaptureModeGlobal);
      plan.exec(data_real, data_complex);
      cudaGraph_t my_graph;
      cudaStreamEndCapture(stream, &my_graph);
      cudaGraphNode_t my_node;
      cudaGraphAddChildGraphNode(&my_node, new_graph, nullptr, 0, my_graph);
    }
    cudaGraphExec_t instance;
    cudaGraphInstantiate(&instance, new_graph, NULL, NULL, 0);
    cudaStreamDestroy(stream);
    cudaGraphDestroy(new_graph);

    graphCache.insert({key, ...});
  }
  cudaGraphLaunch(*graphCache.at(key), 0);
  cudaDeviceSynchronize();
}
```

If not already cached

Capture and add independent sub-graph

Finalize full graph

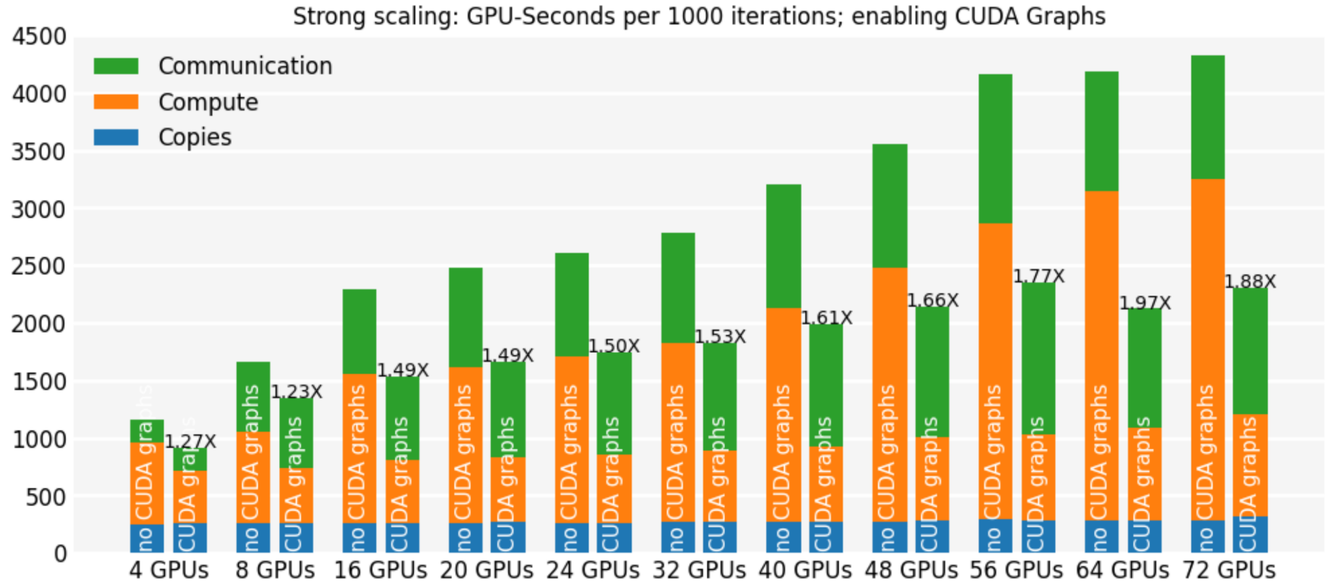
Execute graph



Major benefit from CUDA Graphs

3.2X improvement by using CUDA Graphs for FFTs and GEMMs

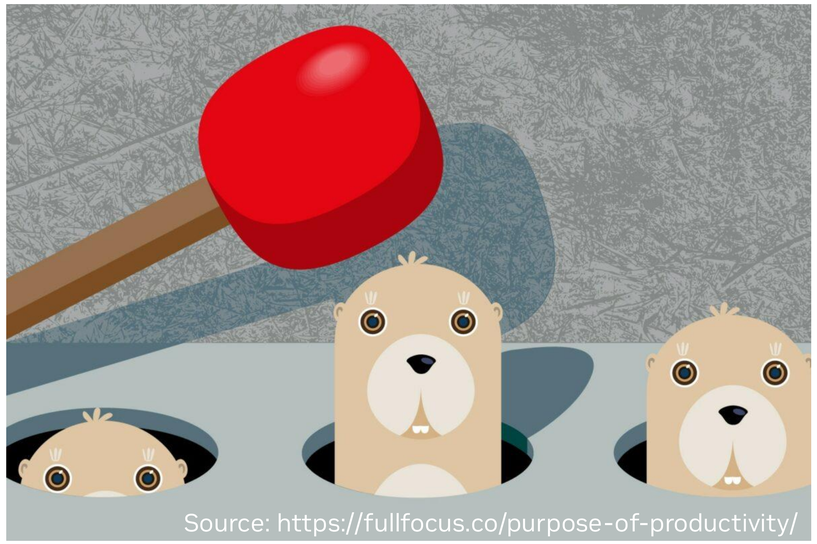
- We have many FFTs/GEMMs running in parallel
  - E.g. One kernel with # warps: 188 / assume 32 warps per SM / # SMs: 152
    ➔ Could fit 25 kernels in parallel
- Caution: Using Nsight Systems with CUDA graphs has a significant overhead ➔ use --cuda-graph-trace=graph
- Much stronger effect at low resolution because average kernel time is smaller

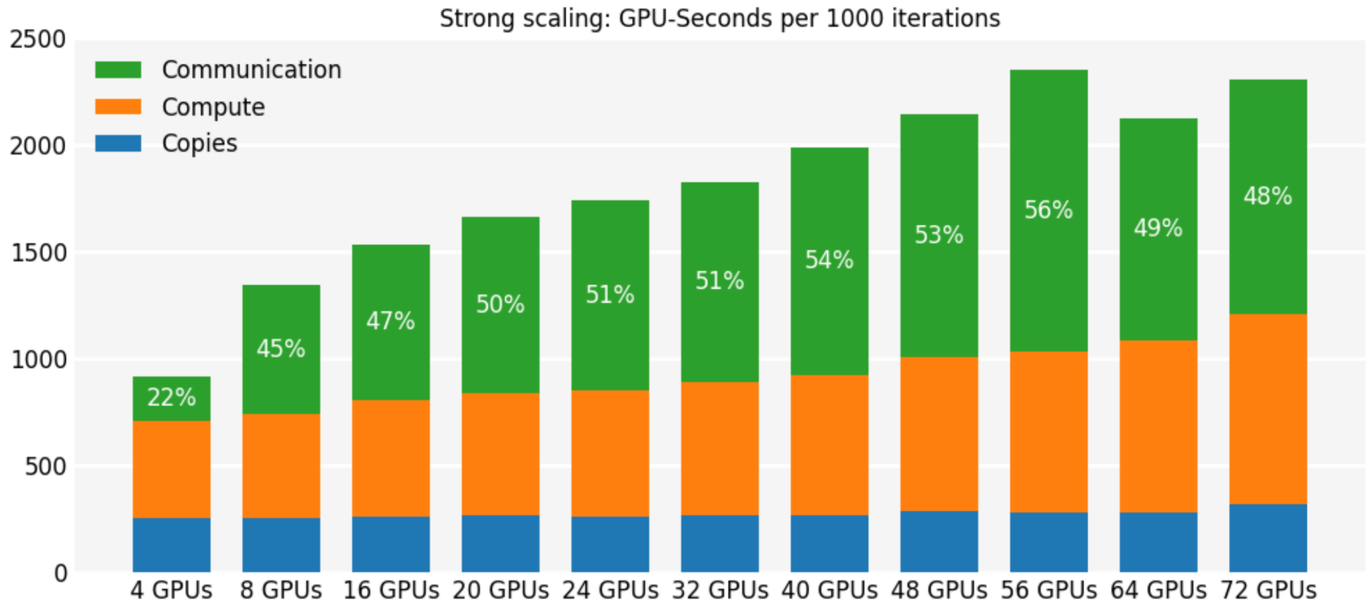


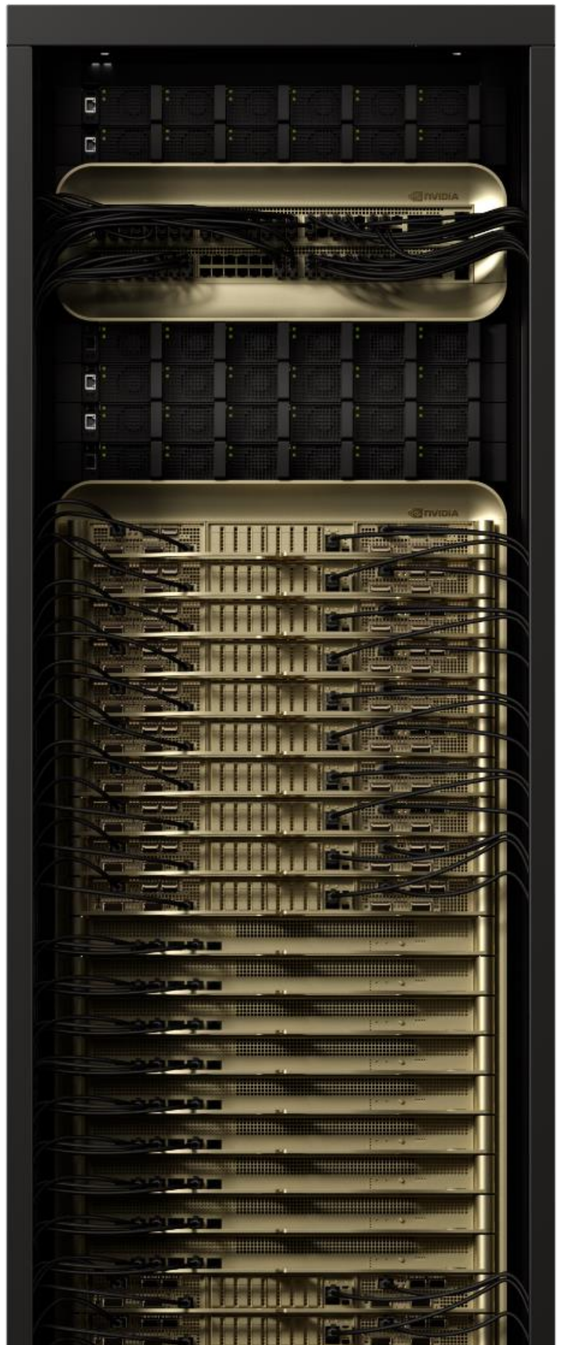
Scaling beyond the limits of traditional IB systems

When the network becomes the bottleneck

- The bottleneck shifts from the GEMMs/FFTs to communication
  - More than 50% of the total runtime at larger scale
- Alltoallv can become very expensive
- Packing and unpacking is only moving data, no productive work



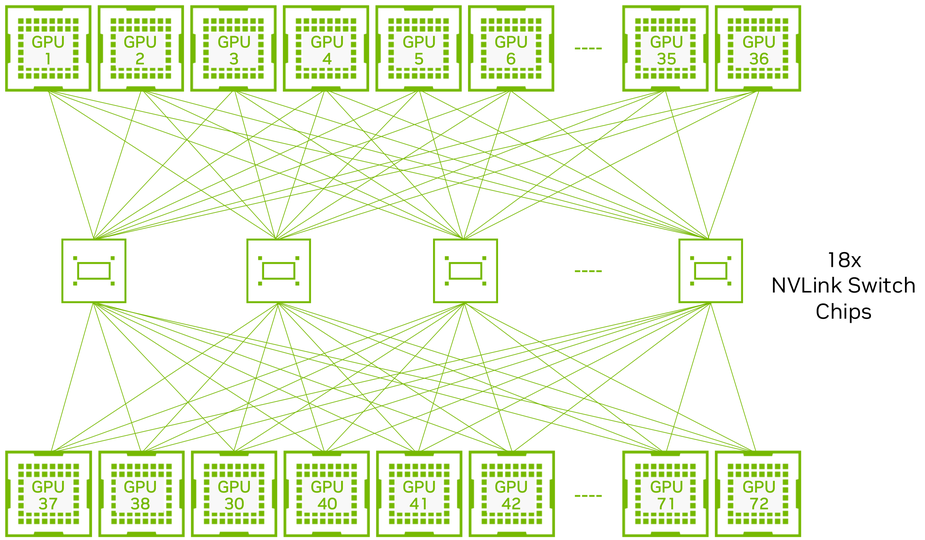


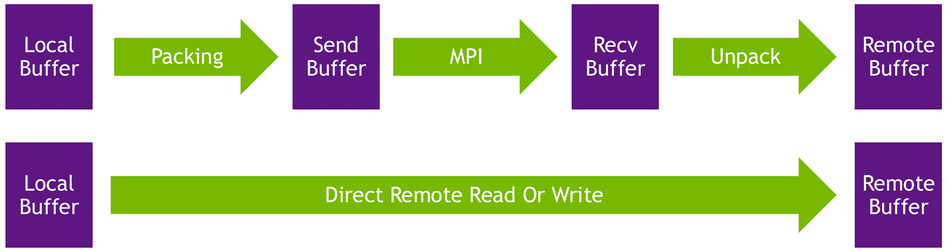


NVIDIA GB200 NVL – One large GPU

The benefits of a single large memory domain

36 GRACE CPUs
72 BLACKWELL GPUs
Fully Connected NVLink Switch Rack

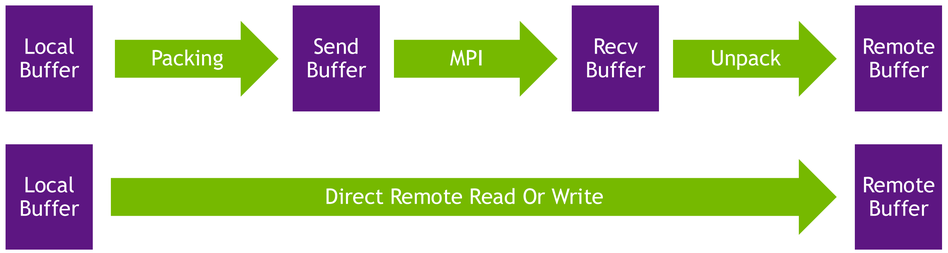


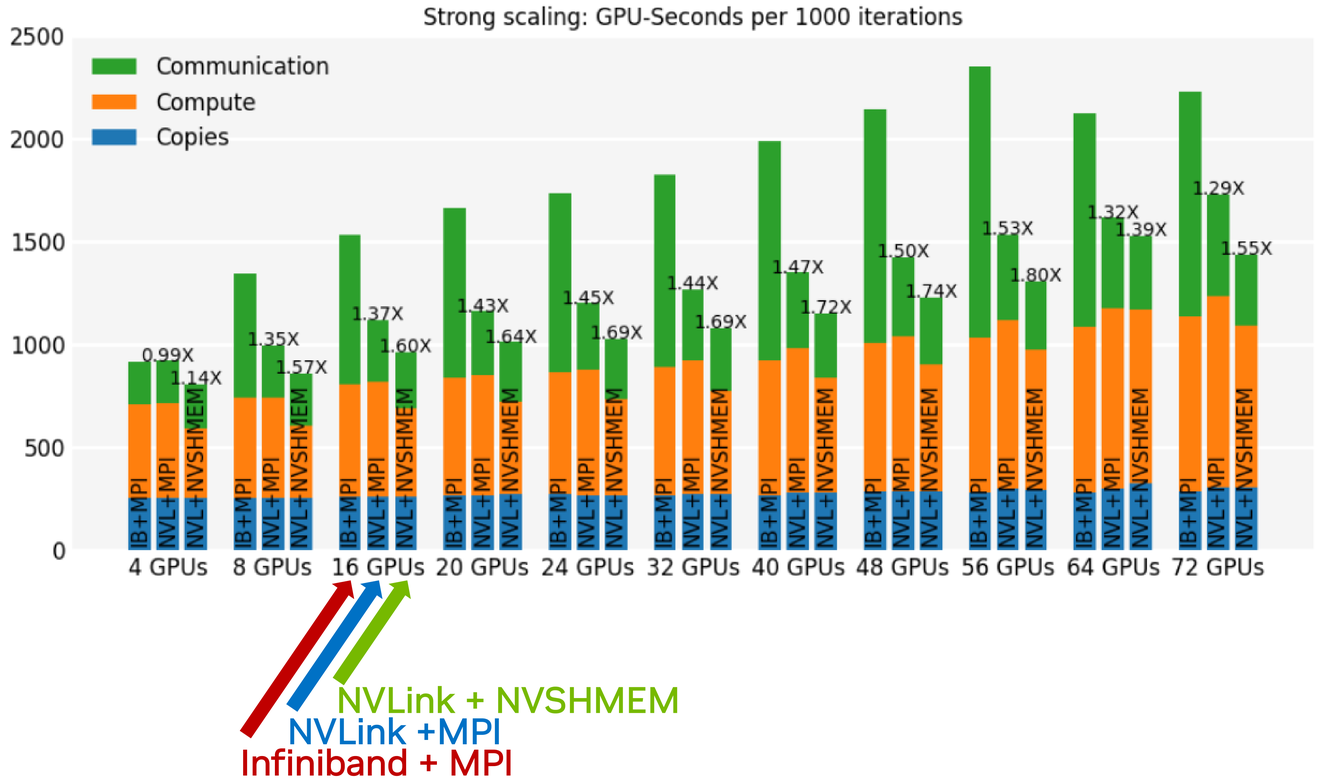


Benefits of GB200 NVL72 and NVSHMEM

Up to 1.8X from a fast interconnects and direct memory access

- Almost 2X over the Infiniband solution at some scales
  - Faster interconnect
  - Removes packing/unpacking completely
- Makes the code easier because directly accessing remote buffers (access data like “1 GPU”)
  - Current implementation assumes a reduced octahedral grid to compute the exact addresses

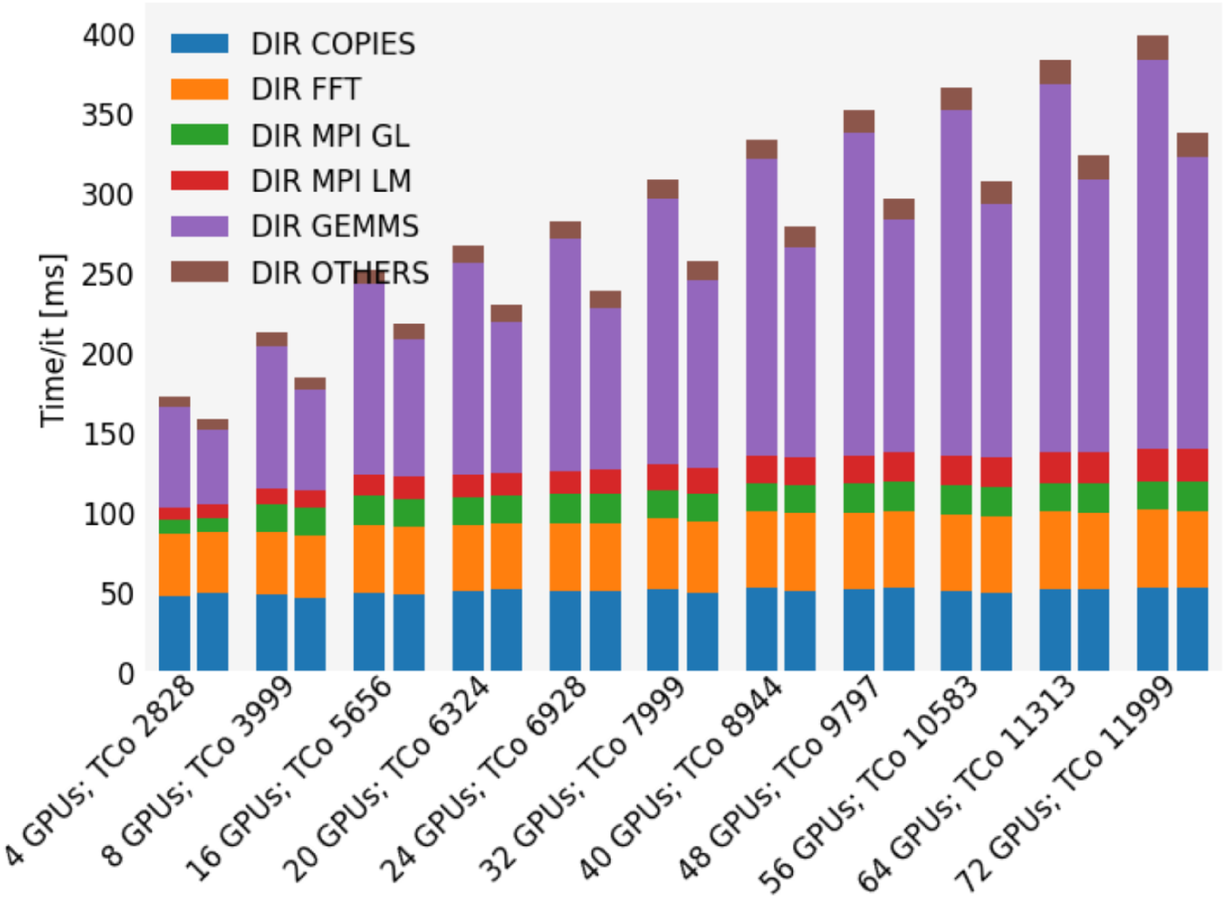




Single precision emulation to improve performance

BF16/9 can give easy benefits when GEMMs become an issue

- Matrix multiplies become a real issue at very high resolutions
- Alternative Pre-Blackwell: Use 3XTF32 to approximate one FP32 GEMM
  - Downside: Measurably lower precision
- With NVIDIA Blackwell GPUs: Use 9xBF16 to emulate FP32 GEMM
  - On average equal precision at potentially higher performance
  - We could also use reduced precision (e.g. 6xBF16)
- More details today 16:10 by Florent Duguet:
  “On the use of different arithmetic precisions and its impact on dynamic systems”



Summary – Towards High-Performance Spectral Transform

General approach applicable to other parts of the code as well

- **Profiling at scale.**
  - Identify bottlenecks with lightweight profiling tools like Nsight Systems
- **Use the full device.**
  - Enable concurrent execution using CUDA Graphs, capturing via CUDA Graph API
- **Use tensor cores.**
  - Maintain full FP32 precision with BF16/9, fully benefit from Tensor Core capabilities
- **Leverage modern network technology.**
  - Leverage fine-granular unified memory provided by NVL72
- **Collaborate with partners.**
  - Maximum efficiency at optimizing the code by separation of concerns.