Update on GPU-enabled NIMROD infrastructure

NIMROD coding camp Jan. 16th, 2023

J King (Tech-X)*

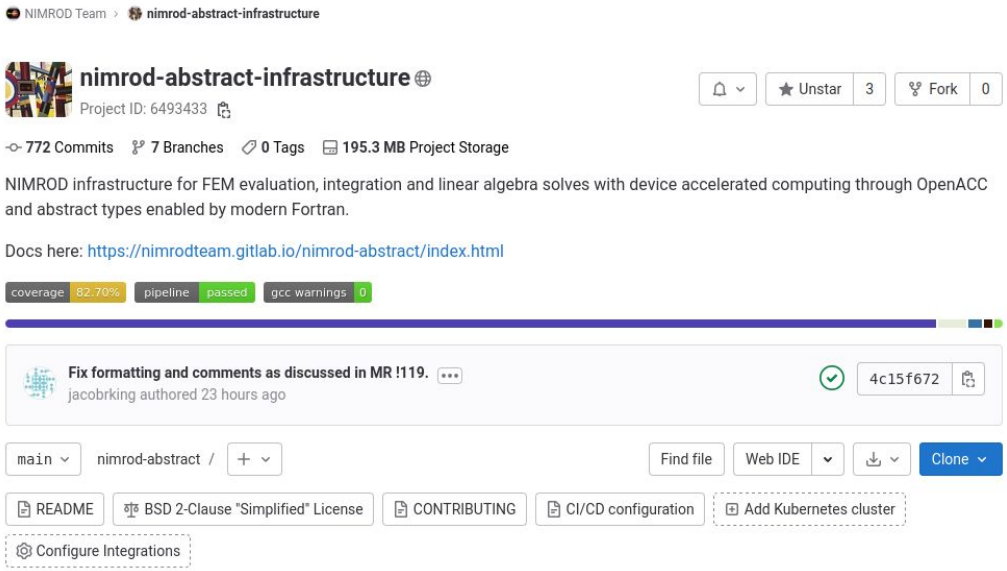
Thanks to comments and code review from
E Howell (Tech-X) and B Cornille (AMD)

Work supported by US DOE

*Also affiliated with Fiat Lux

Abstract infrastructure repo is maturing

- https://gitlab.com/NIMRODteam/nimrod-abstract
- Years of work
- Features:
  - Modern Fortran Abstract Types
  - GPU-enabled blocks with OpenACC
  - CI testing
    - multiple compilers
    - Serial / MPI / GPU
    - unit tests
    - memory testing (valgrind)
  - Using code review and issue tracking
  - Use mpi_f08 module
  - Well documented
- Still much to do



This talk is an overview, not a tutorial

- Topics:
  - GPU optimization of kernels for the Laplace example problem
    - Will demonstrate profiler usage
    - Preliminary performance on Perlmutter GPU
  - Future development topics
    - <list>
- We can work on topics and/or optimization during the camp
- Links to gitlab documentation and issues will be provided as applicable
- If you find a problem with the docs, please edit the wiki to fix it!

Reminder: GPU code is in the infrastructure and integrands,
minimal impact on high-level code

Laplace solve provides test case for optimization

- Known time-discretized solution ensures correctness
- At the end of computation, compare rhs_norm with that expected from self-similar decay
- Time-step loop is focus of optimization
- Link to laplace.f
- Periodic box
- Plan to extend test with regularity, BCs and circular meshes #121

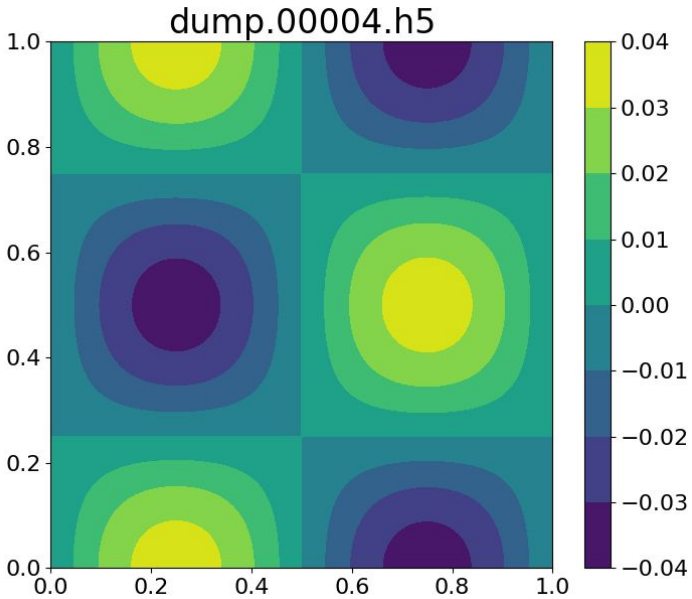


Fig 1: A really boring plot of the solution

Next slides: sample optimizations for the Laplace case

- OpenACC modification: Matvec optimization
- Operations modification: Integrand optimization
- Asynchronous modification: Concurrent execution of kernels

Optimization work not finished but ongoing

Matvec optimization by OpenACC modification

```
2719      !$acc parallel present(mat,output,matrix,vector,iy0,ix0,jy0,jx0)          &
2720      !$acc present(y0off,y1off,x0off,x1off) async(mat%id) if(on_gpu)
2721      !$acc loop gang collapse(3) private(val)
2722      DO iy=iy0,mat%my
2723        DO ix=ix0,mat%mx
2724          DO iq=1,mat%nq_type(ib)
2725            val=0._r8
2726            !$acc loop vector private(jy,jx) collapse(3)                        &
2727            !$acc reduction(+:val)
2728            DO jyl=y0off,y1off
2729              DO jxl=x0off,x1off
2730                DO jq=1,mat%nq_type(jb)
2731                  jy=iy+jyl
2732                  jx=ix+jxl
2733                  IF (jx>=jx0.AND.jx<=mat%mx.AND.jy>=jy0.AND.jy<=mat%my) THEN
2734                    val=val+matrix(jq,jxl,jyl,iq,ix,iy)*vector(jq,jx,jy)
2735                  ENDIF
2736                ENDDO
2737              ENDDO
2738            ENDDO
2739            output(iq,ix,iy)=output(iq,ix,iy)+val
```

```
2702      id=par%get_stream(mat%id,ib,4)
2703      !$acc parallel present(mat,output,matrix,vector,iy0,ix0,jy0,jx0)          &
2704      !$acc present(y0off,y1off,x0off,x1off) async(id) wait(mat%id) if(on_gpu)
2705      !$acc loop gang collapse(2)
2706      DO iy=iy0,mat%my
2707        DO ix=ix0,mat%mx
2708          !$acc loop vector private(val)
2709          DO iq=1,mat%nq_type(ib)
2710            val=0._r8
2711            !$acc loop seq collapse(3) private(jy,jx)
2712            DO jyl=y0off,y1off
2713              DO jxl=x0off,x1off
2714                DO jq=1,mat%nq_type(jb)
2715                  jy=iy+jyl
2716                  jx=ix+jxl
2717                  IF (jx>=jx0.AND.jx<=mat%mx.AND.jy>=jy0.AND.jy<=mat%my) THEN
2718                    val=val+matrix(jq,jxl,jyl,iq,ix,iy)*vector(jq,jx,jy)
2719                  ENDIF
2720                ENDDO
2721              ENDDO
2722            ENDDO
2723            output(iq,ix,iy)=output(iq,ix,iy)+val
```

- Reducing the vector parallelism over the reduction operation (right) makes the GPU code faster
- Code on the right launches concurrent kernels for different basis functions
- Could replace this code with accelerated BLAS GEMV calls
- Could pad matrix/vector structure to avoid bound-checking IF statement

Integrand optimization: on-the-fly alpha transform

```
385     ASSOCIATE (f=>field(bl%ibl)%f,alpha=>field(bl%ibl)%f%qab%alf,              &
386                grad_alpha=>field(bl%ibl)%f%qab%aldf,wdetj=>bl%metric%wdetj)
387       !$acc parallel present(integrand,alpha,grad_alpha,wdetj,bl,f)           &
388       !$acc copyin(fac) async(bl%id) if(on_gpu)
389       !$acc loop gang worker
390       DO ie=1,bl%nel
391         !!$acc cache(wdetj(:,ie))
392         !$acc loop vector collapse(2) independent private(tmpsum)
393         DO iv=1,f%nbasis
394           DO jv=1,f%nbasis
395             tmpsum=0._r8
396             !$acc loop seq
397             DO ig=1,bl%ng
398               tmpsum=tmpsum                                                   &
399                      +wdetj(ig,ie)*(alpha(ig,jv,ie,1)*alpha(ig,iv,ie,1)       &
400                      +fac*SUM(grad_alpha(ig,jv,ie,:)*grad_alpha(ig,iv,ie,:)))
401             ENDDO
402             DO iq=1,f%nqty
403               integrand(iq,iq,ie,jv,iv)=tmpsum
```

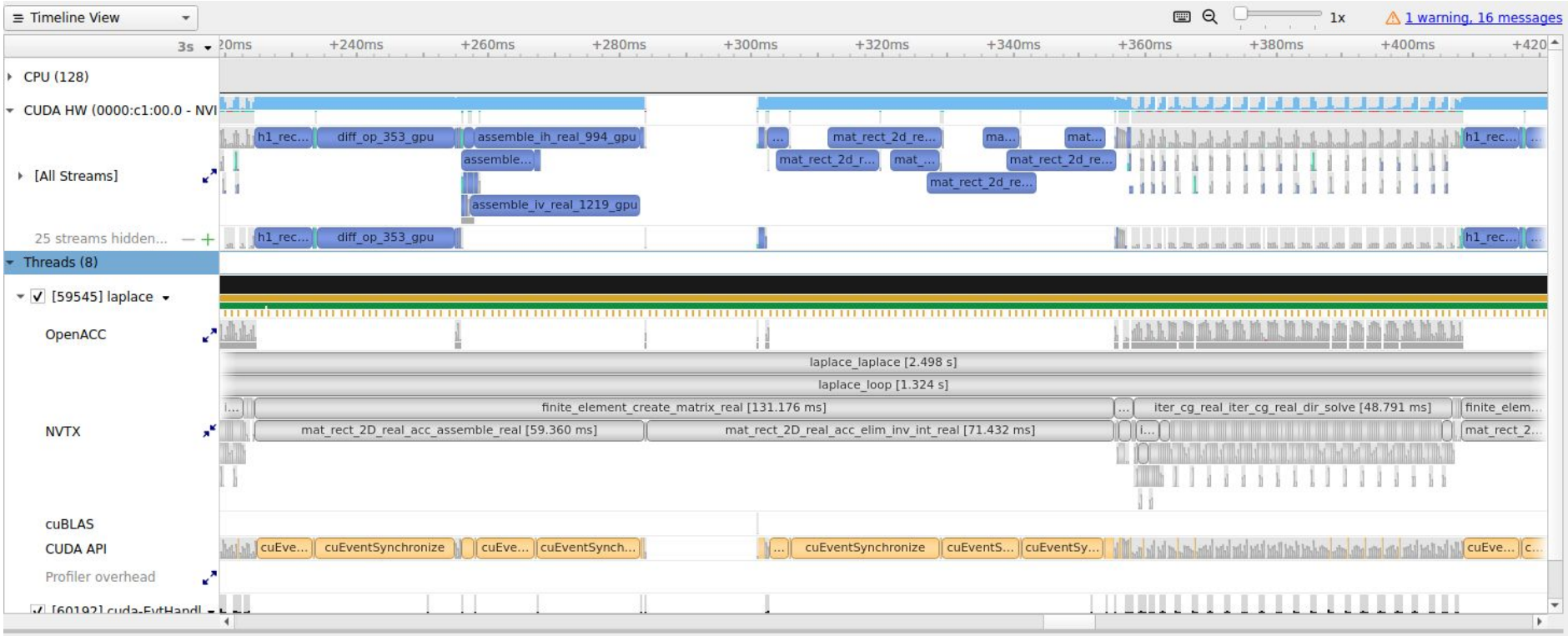
Precomputed by block

```
349     ASSOCIATE (f=>field(bl%ibl)%f,alpha=>field(bl%ibl)%f%qa%alf,               &
350                aldf=>field(bl%ibl)%f%qa%aldf,wdetj=>bl%metric%wdetj,           &
351                ijac=>bl%metric%ijac)
352       !$acc parallel present(integrand,alpha,aldf,wdetj,ijac,bl,f)            &
353       !$acc copyin(fac) async(bl%id) if(on_gpu)
354       !$acc loop gang worker
355       DO ie=1,bl%nel
356         !$acc loop vector collapse(2) independent private(tmpsum)
357         DO iv=1,f%nbasis
358           DO jv=1,f%nbasis
359             tmpsum=0._r8
360             !$acc loop seq
361             DO ig=1,bl%ng
362               tmpsum=tmpsum                                                   &
363                      +wdetj(ig,ie)*(alpha(ig,jv)*alpha(ig,iv)                 &
364                                     +fac*((ijac(ig,ie,1,1)*aldf(ig,jv,1)      &
365                                           +ijac(ig,ie,1,2)*aldf(ig,jv,2))     &
366                                          *(ijac(ig,ie,1,1)*aldf(ig,iv,1)      &
367                                           +ijac(ig,ie,1,2)*aldf(ig,iv,2))     &
368                                          +(ijac(ig,ie,2,1)*aldf(ig,jv,1)      &
369                                           +ijac(ig,ie,2,2)*aldf(ig,jv,2))     &
370                                          *(ijac(ig,ie,2,1)*aldf(ig,iv,1)      &
371                                           +ijac(ig,ie,2,2)*aldf(ig,iv,2))))
372             ENDDO
373             DO iq=1,f%nqty
374               integrand(iq,iq,ie,jv,iv)=tmpsum
```

on-the-fly

- Trades reduced memory bandwidth for increased FLOPs, see #103
- On the GPU, FLOPs are cheap but memory access is expensive
- Produces a 30% slow down on the CPU, but a speed up on the GPU

Concurrent execution of GPU kernels by streams is enabled



- Simultaneous kernel execution with parallelization over blocks and basis function types
- Nsight system profile view of kernel execution vs time shown

Test case parameters and performance expectations

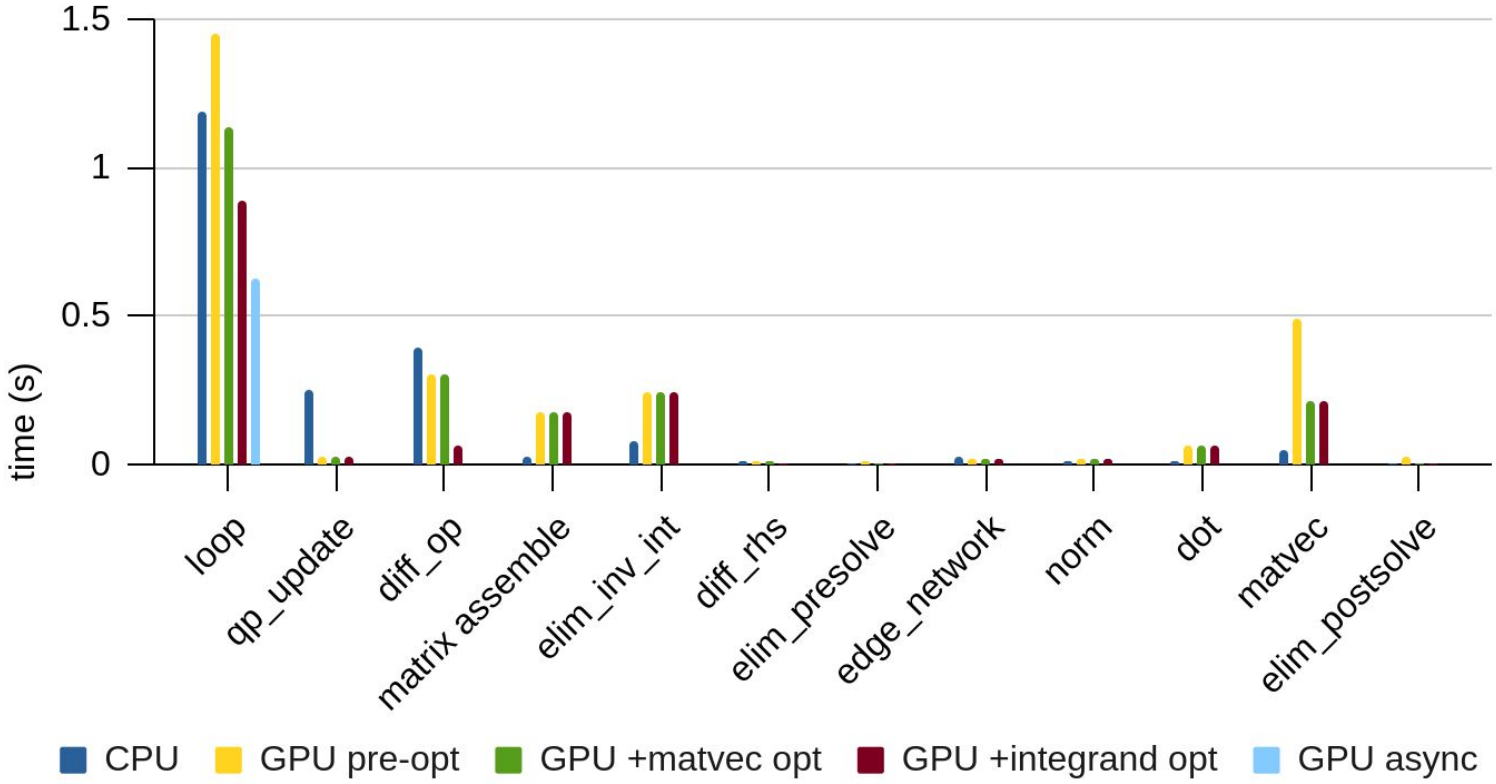
- Grid mx=my=256; pd=6
- Use nxbl=nybl=32 on CPU
- Use nxbl=nybl=2 on GPU
- nstep=4, dt=0.01, diff=1, theta=0.5
- CPU uses 7, 16, 45, 67 iterations per step
- GPU uses 7, 16, 45, 75 iterations per step
- Rough calculations for Perlmutter:

Nodes	**Hardware**	**TFlops (DP)**	**Bandwidth (GB/s)**
3072	2x EYPC Milan CPU	4	400
1536	4x Nvidia A100 GPU	40	8000
	1x EYPC Milan CPU	2	200

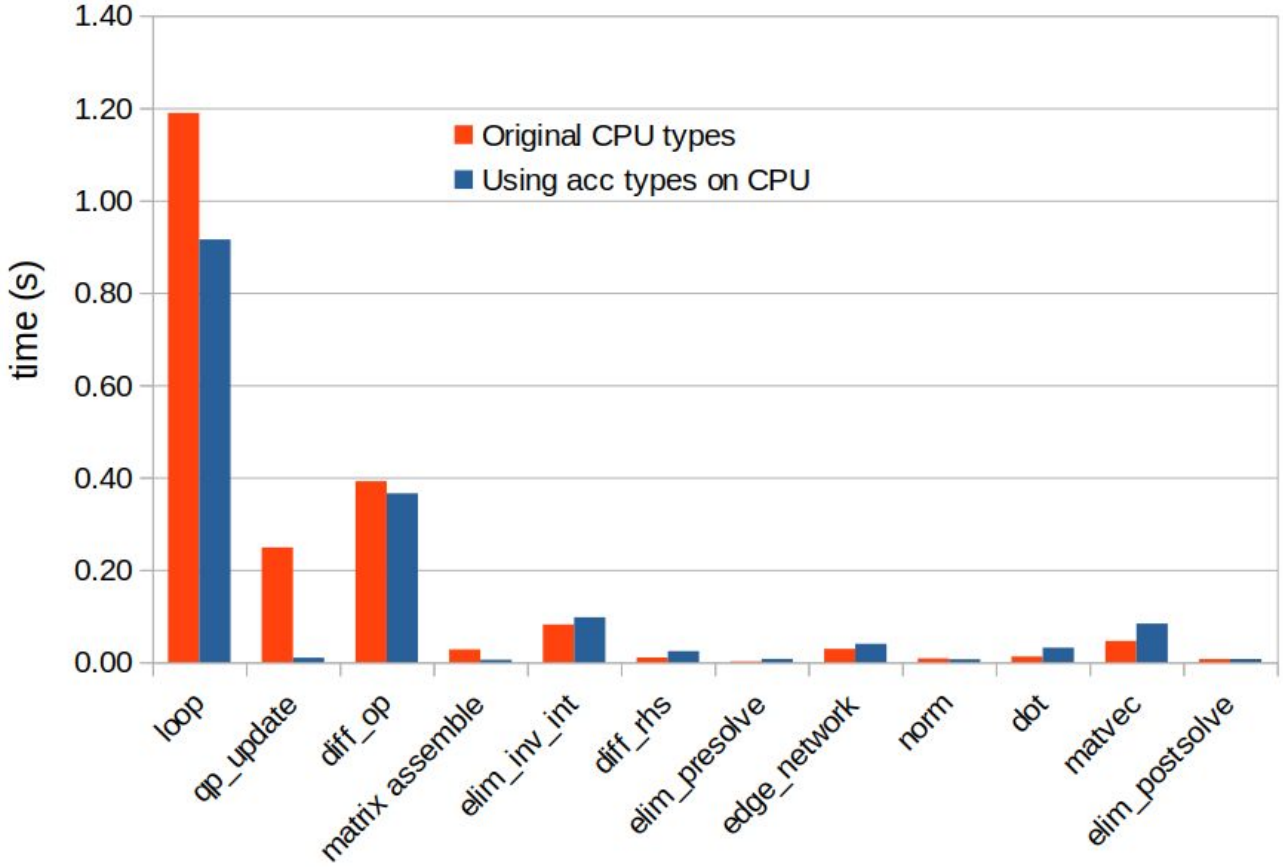
- 1 GPU node ≈ 10-20 CPU nodes
- All GPU nodes ≈ 5-10x all CPU nodes
- Compare performance on 1 CPU node (128 MPI procs) to 1 GPU node (4 MPI procs + 4 GPUs)

Preset status: performance

Optimization work not finished but ongoing



CPU performance improves with GPU acc types



Future development directions

- GPU RDMA in seams #111
- Jacobi preconditioner and abstract interface #113 and #83
- Export to CSR/CSC format #114 and interface with external solvers
- Generalized infrastructure for surface and boundary integrals #115
- Heterogeneous CPU/GPU block decomposition #116
- Batched unstructured FEM evaluation #117
- Abstract interface to solve the inverse problem #118
- Use CI@NERSC to test for performance regression on Perlmutter #119
- Extend Laplace testing with dirichlet BC, regularity and unstructured block meshes #121

GPU particle based methods depend on this infrastructure

- Specifically, solving the inverse problem and batched unstructured FEM evaluations
- Performance considerations on the GPU:
  - Sorting arrays to be ordered by element and block
  - Batching points for inverse mapping and considering sorting
  - Sorting points by block, element for batched unstructured FEM evaluations
- Potential collaboration for PIC and energetic or runaway particle modeling on the GPU

Preconditioning on the GPU challenges performance

- “Does throughput-optimized hardware (GPU) running an easy to parallelize algorithm outperform latency-optimized hardware (CPUs) running hard to parallelize algorithms (triangular solves)?” - question from Sam Williams
- The Laplace example is a bad test of preconditioning
  - The example converges with CG and *no preconditioning*
  - With MHD, GMRES + a weak preconditioner will fail to converge
- Are there a more approximate (than LU), GPU-amenable algorithms that converge with MHD?
- Past hackathon: limited-success with the Ginkgo parILUT algorithm

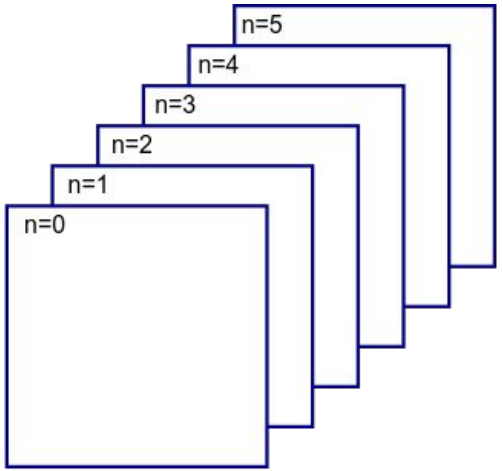
Final remarks

- NIMROD abstract infrastructure is running on the GPUs
- A Perlmutter GPU node is faster than a Perlmutter CPU node
- Plan to continue GPU optimization and improving code features
- Spring 2023: build generalized multispecies code on top of abstract infrastructure
  - Use time-discretized linear waves as a test of implementation
  - Automate eigenfunction calculation with python using sympy

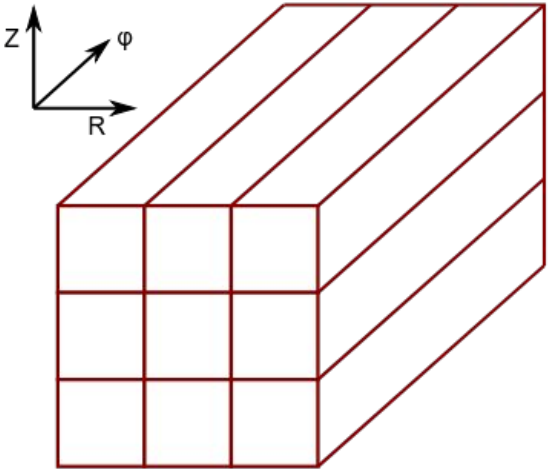
Extra slides

GPU FE integration with Fourier decomp

- Fourier decomposition presently requires communication during FE integration for matrix-vector-dot product and RHS integration
- Anticipate this synchronization to be costly on GPUs
- GPU plan: use a different decomposition with all Fourier modes on each GPU (use (b) not (a) for data layout)



(a) toroidal (φ) decomposition into Fourier modes of RZ-FE planes where complex coefficients correspond to each mode #



(b) decomposition into RZ-blocks containing either all Fourier modes or a real-space representation in φ