Recent progress on GPU infrastructure optimization

Jacob King*, Eric Howell (Tech-X)

NIMROD Team Meeting
May 24th 2023

Work supported by US DOE

*Also affiliated with Fiat Lux

Periodic code modernization is required

1. Transition to GPU-accelerated architectures (e.g. Perlmutter & Frontier)
2. Language standard improvements (e.g. Fortran 2018 or C++20)
3. Compiler improvements (e.g. what are all these warnings with GCC 12?)
4. Incorporation of changes in design paradigm (e.g. code review & CI)

Option 1:

Start over!

The optimism!

But don’t throw out the baby with the bath water

- Legacy codes exist for a reason
- Ask: why did they survive where so many others failed?
- Leverage prior experience

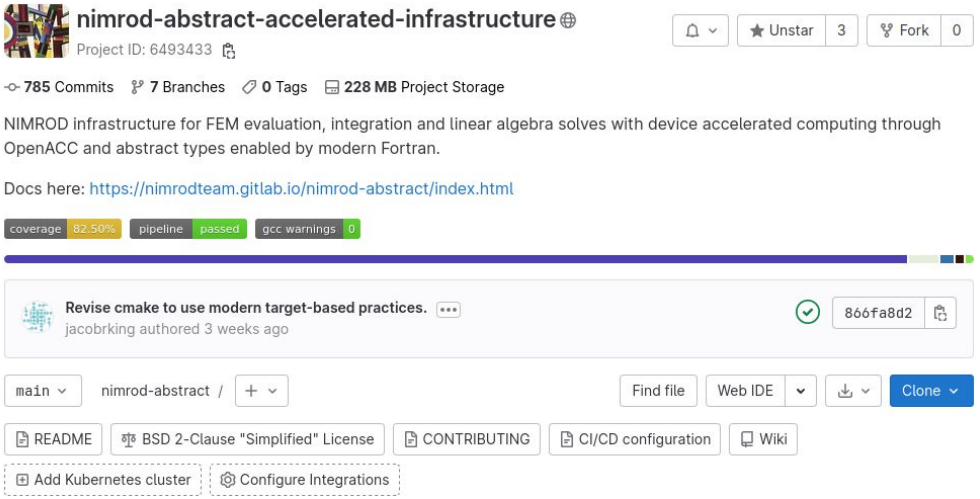
- Don’t start over, refactor!

NIMROD MHD code algorithmic features

- Solves the extended-MHD equations
- Mixed spatial discretization: 2D high-order C0 FEM + Fourier decomposition
  - High-order methods critical for highly-anisotropic equations (e.g. thermal conduction)
  - Static condensation, Schur-complement elimination of spectral element internal DOFs, limits global DOFs
  - Matrix-free Fourier-mode block-Jacobi preconditioner designed for highly magnetized systems
- Distributed-memory parallelism domain decomposition via Fourier mode splitting and tiling blocks of FEs
- Mixed implicit / Semi-implicit time-stepping

NIMROD abstract accelerated infrastructure

- Open source: https://gitlab.com/NIMRODteam/nimrod-abstract
- Open source helps with compiler bug reporting
- 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



Modern Fortran enables abstract types

Fortran 90 – 2 block types
Loops are explicit

```
      CALL timer(timestart)
      DO ibl=1,nrbl
        CALL rblock_make_matrix                                  &
     &    (rb(ibl),matrix_structure(jmode)%rbl_mat(ibl),integrand,    &
     &     MAX(nq,nqdis))
      ENDDO
      DO ibl=nrbl+1,nbl
        CALL tblock_make_matrix                                  &
     &    (tb(ibl),matrix_structure(jmode)%tbl_mat(ibl)%lmat,         &
     &     integrand,nq)
      ENDDO
      CALL timer(timeend)
      time_mat = time_mat + timeend - timestart
```

Fortran 2018 – abstract block types
Types are encapsulated with abstract definition of functionality

```
    DO ibl=1,SIZE(bl)
      ASSOCIATE (mat=>matrix(ibl)%m)
        ALLOCATE(int_mat(mat%nqty,mat%nqty,mat%nel,mat%u_ndof,mat%u_ndof))
        int_mat=0._r8
        CALL integrand_func(bl(ibl)%b,int_mat)
        CALL mat%zero
        CALL mat%assemble(int_mat)
        DEALLOCATE(int_mat)
```

- Use of abstraction not just for brevity – it enables future development
- Extension of Fortran 90 code to 3 block types is unmanageable

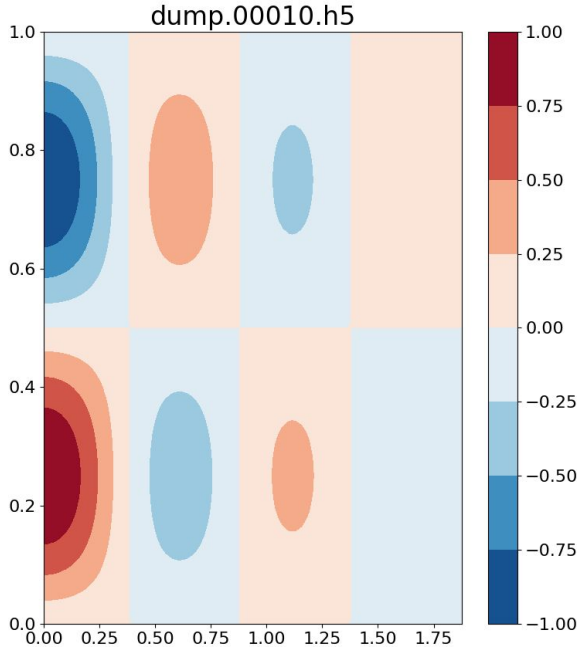
Virtual objects long present in C++, why use Fortran?

- Developer familiarity
- Fortran arrays
- Fortran standards committee focused on scientific computing

- Fortran to C++ terminology guide
- Abstract ⇔ Virtual
- Type ⇔ Object
- Fortran array slicing ⇔ Something in the STL or Eigen? Probably verbose.

Laplace example provides test case

- Known time-discretized solution with self-similar decay ensures correctness of implicit formulation
- Sine/Cosine waves and Bessel functions used in slab and cylindrical geometry, respectively
- Time-step loop is focus of GPU optimization



A plot of the cylindrical solution with a regularity condition (left), periodic boundary conditions (top/bottom) and an essential condition (right)

Test case parameters and performance expectations

- nvhpc nvfortran compiler
- Rough calculations for Perlmutter:

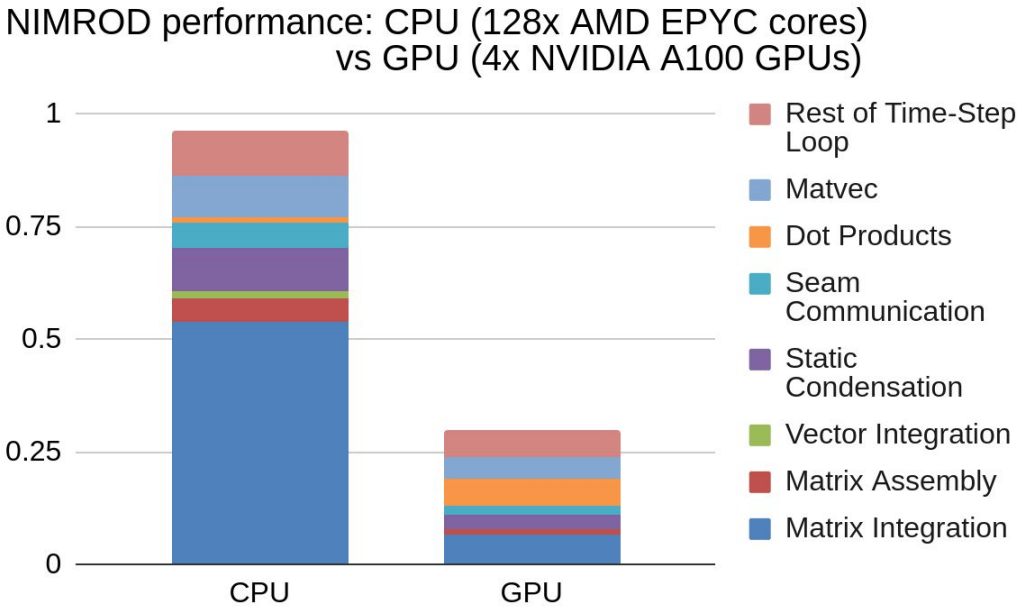
- Grid mx=my=256; pd=6
- Use nxbl=nybl=32 on CPU
- Use nxbl=nybl=2 on GPU
- nstep=4, dt=0.01, diff=1, theta=0.5

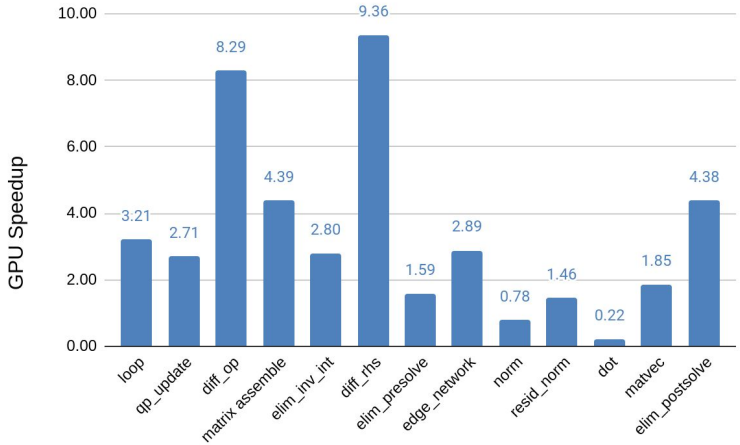
Nodes	Hardware	TFlops (DP)	Bandwidth (GB/s)
3072	2x EYPC Milan CPU	4	400
1536	4x Nvidia A100 GPU	40	8000
	1x EYPC Milan CPU	2	200

- 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 expect 5-10x on application





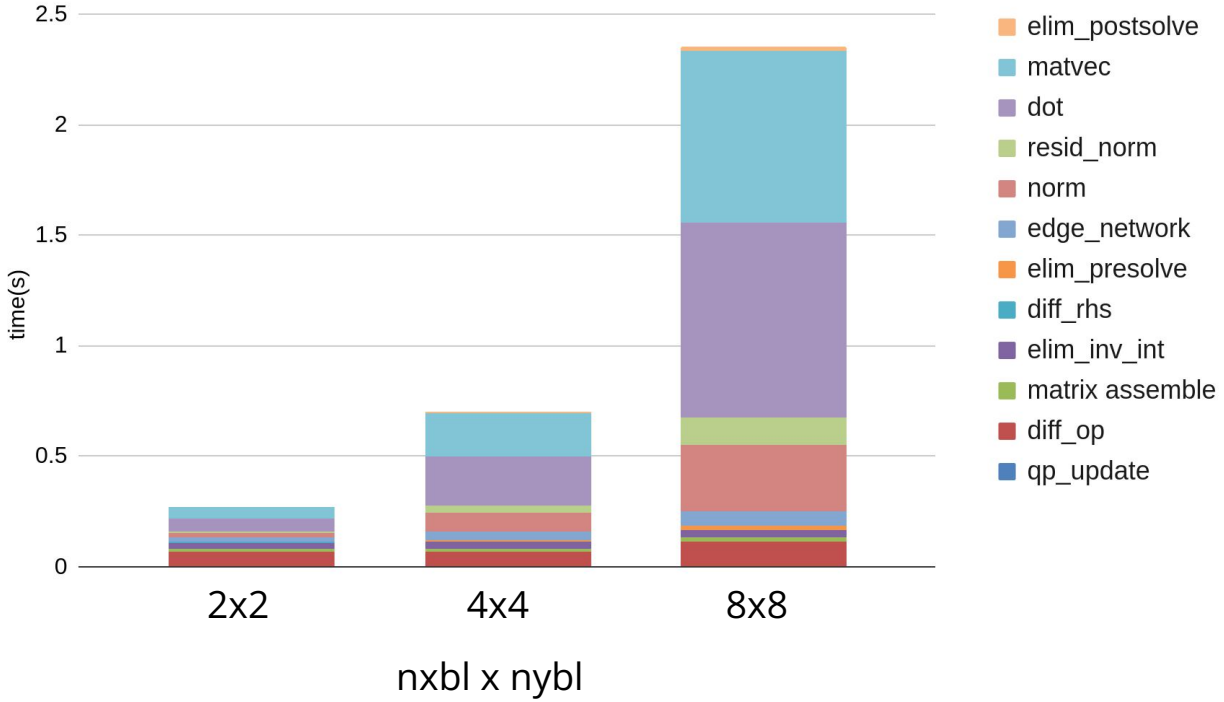
Future development directions

Near term priorities

- Jacobi preconditioner and abstract interface #113 and #83
- Export to CSR/CSC format #114 and interface with external solvers
- Rework seam norm/tang #121 for abstraction

- GPU RDMA in seams #111
- Generalized infrastructure for surface and boundary integrals #115
- Heterogeneous CPU/GPU block decomposition #116
- Batched FEM evaluation at unstructured points #117
- Abstract interface to solve the inverse problem #118
- Use CI@NERSC to test for performance regression on Perlmutter #119

Using more, smaller blocks degrades performance



Items for improvement:

- Kernel fusion
- GPU RDMA
- CPU/GPU block decomp w/ small blocks on CPUs

Where's the fluid advance?

- Plan to work on it in June & July
- Expect an update on it in August
- Will focus on generalized multispecies with COM single-ion-fluid as a specialized case
- Use time-discretized linear waves as a test of implementation

Final remarks

- NIMROD abstract accelerated infrastructure is running on the GPUs
  - A Perlmutter GPU node is faster than a Perlmutter CPU node
- GPU optimization and improving code features is ongoing
- Building generalized multispecies code on top of abstract infrastructure
  - Hierarchy of multicomponent MHD models built into data structures
  - Use time-discretized linear waves as a test of implementation
  - Automate eigenfunction calculation with python using sympy