GEMMA Electromagnetic Code and ADELUS - New Capabilities

Joseph D. Kotulski, Vinh Dang 1352
jdkotul@sandia.gov, vqdang@sandia.gov

Trilinos User Group Meeting 2022
October 25, 2022
GEMMA Description

- Frequency-domain method of moments solution
  - Steady state solution
  - With specialized algorithms (thin-slot, etc.)

- Boundary element formulation
  - Mesh surfaces of parts – interface between regions

- Exact radiation boundary condition
  - Due to Green’s function

- Formulation results in dense (fully populated) matrix
  - Simulations can be limited by available memory
  - Entries are double precision complex

- Code has been ported and used for ND problems on CTS1, ATS1, and ATS2

The next generation version of EIGER
Capability on Next-Generation Hardware

**FORTRAN MPI Paradigm**
- MPI inter- and intranode parallelism
- High processor clock speed
- High memory per processor

**Gemma Heterogenous Paradigm**
- MPI internode parallelism
- Threading intranode parallelism
- Low processor clock speed
- Low memory per processor

Applications
- Gemma

Libraries
- Trilinos

Kokkos abstraction layer

Multi-Core
- Many-Core
- APU
- CPU+GPU
GEMMA – NEW FEATURES

- Improved slot algorithm
  - Takes into account the depth resonance of the slot

- Power Balance
  - Simplified power calculations to determine the high-frequency response.

![Diagram showing frequency response with improved slot algorithm and power balance](image-url)
GEMMA – NEW FEATURES

- Rational Interpolation
  - Algorithm to locate peaks - important for calculation of electromagnetic coupling

- Matrix fill algorithm improved
  - Fill by unknowns (i and j) instead of by elements

33 x Speedup

27 x Speedup
GEMMA – Example Problem (2 million unknowns)
GEMMA – Future Solver Development

- Preconditioner development
  - Matrices have behavior much different than what is experienced with FEM solvers.

- Compression Techniques
  - Reduce the memory footprint
  - Iterative solution using BELOS

- Combining the above concepts
ADELUS – AMD HIP Backend Support

- Necessary changes made for code and CMake to support HIP backend

- Trilinos configuration:
  - hipcc compiler
  - Architecture flag for Kokkos (For MI100: Kokkos_ARCH_VEGA908=ON)
  - Kokkos_ENABLE_HIP=ON
  - KokkosKernels_ENABLE_TPL_ROCBLAS:BOOL=ON/OFF

- rocBLAS wrappers for GEMM, IAMAX, and SCAL kernels to Kokkos Kernels

- Future work: evaluate ADELUS performance on Crusher/Frontier (ORNL)
ADELUS – Factor and Solve Interfaces

- ADELUS previously only provided LU factorization and solve via a single interface `Adelus::FactorSolve` (matrix + RHS packed together)

- Create two separate interfaces which are useful for applications that (i) do not have RHS at the time of factorization OR (ii) need to solve different RHSs with a pre-factorized matrix
  - `Adelus::Factor`: LU factorization
  - `Adelus::Solve`: forward solve + backward solve
  - Support execution on GPUs and multiple RHS vectors

```
Adelus::Factor
  lu()
  factor()
  permute_mat()

Adelus::Solve
  solve()
  permute_rhs()
  forward()
  back_solve6()
  perm1()
```
Enable ADELUS to run on an arbitrary communicator rather than MPI_COMM_WORLD
- Create sub-communicators and launch Adelus to solve many linear equation systems

A new class, AdelusHandle, contains:
- a communicator, global variables, constructor and methods to retrieve these variables

A handle needs to be created and passed through Adelus interfaces from application code

```cpp
class AdelusHandle {
  private:
  // Comm. variables and used-to-be global variables
  int my_rows;       // num of rows I own
  int my_cols;       // num of cols I own
  int my_rhs;        // number of RHSs that I own
  ... 
  MPI_Comm row_comm; // row sub-communicator
  MPI_Comm col_comm; // column sub-communicator
  MPI_Comm comm;     // communicator that I belong to
  public:
  AdelusHandle (MPI_Comm comm_, const int matrix_size_,
                const int npocs_row_, const int num_rhs_, ...) {
    // Calculate global vars and create row and col sub-comms
    ... 
    KOKKOS_INLINE_FUNCTION
    MPI_Comm get_comm() const { return comm; }
    ... 
    KOKKOS_INLINE_FUNCTION
    int get_my_rows() const { return my_rows; }
  }

  // Create handle
  Adelus::AdelusHandle ahandle(my_color, sub_comm,
                              matrix_size, nprocs_row, nrhs);
  ... 
  // Pass through Adelus interfaces
  Adelus::Factor (ahandle, my_A, h_permute, &secs);
```
ADELUS – Backsolve Performance Improvement

- **Issue**: backsolve previously did not scale well with large numbers of RHS vectors
  - Using pipelined communication for the whole RHS mutivectors across MPI processes at each column iteration

- **Improvement**:
  - **Broadcasting only one** current active column within row communicators at each iteration → communication overhead is significantly reduced

---

**RHS Scaling with Factor + Solve**

(N = 111528, 16 GPUs (4 x 4))

- **Factor**
- **Solve (old)**
- **Solve (new)**

1.06x
22x

---

**Time (sec.)**

Number of RHS vectors

1 2 4 6 8 10 12 16 20 24 28 32 40 48 64 128 256 512
ADELUS – Future Performance Improvement

- Allow using tile size greater than 1
- Allow using mixed-precision