Ginkgo

On the way towards an accelerator-focused C++ sparse linear algebra library

Overview

- Open source sparse linear algebra library
- Sparse linear solvers, preconditioners
- Sparse building blocks (SpMV, reductions, sparse pattern algorithms)
- Generic algorithm implementation

  \+ architecture-specific highly optimized kernels
- Based on C++
- Focused on GPU accelerators (i.e. NVIDIA GPUs)

Software Design

- Based on C++ (11), C bindings planned
- Templated precision (ValueType, Integer)
  - By default, compiles Z,C,D,S
- Smart pointers to avoid memory leaks (unique/shared pointers)
- Runtime polymorphism
  - Kernels have the same signature for different architectures
  - **Executor** determines which kernel is used

    Determines where Data lives
    & operation is executed

  - **LinOp** class for any linear operator:
    - Matrices
    - Solvers
    - Preconditioners
    - …

    generate
    apply
    …

```
// Read data
std::shared_ptr<mtx> A = mtx::create(exec);
A->read_from_mtx("data/A.mtx");
auto b = vec::create(exec);
b->read_from_mtx("data/b.mtx");
auto x = vec::create(exec);
x->read_from_mtx("data/x0.mtx");

// Generate solver
auto solver_gen = cg::create(exec, 20, 1e-20);
auto solver = solver_gen->generate(A);

// Solve system
solver->apply(b.get(), x.get());
```

Software Quality Efforts

- Open source sparse linear algebra library
- git open-source repository
  https://github.com/ginkgo-project/ginkgo/
- Modified BSD license
- Community effort
  - Code review process for pull requests
- Copyright@ UTK, KIT, UJI
- Collaborative effort:
  Karlsruhe Institute of Technology
  Universitat of Jaume
  University of Tennessee

Software Quality Efforts

- Tests for all functionalities (googletest)
  googletest repo gets cloned in the installation step

- Reference kernels for all functionality ensuring correctness
  (not tuned for performance)

- Documentation (doxygen)

- Planned integration into xSDK
  Compliant with the community policies
  https://xsdk.info

- Planned Continuous Integration (CI)

Software Quality Efforts

- CMake build system
- Cross-platform compilation (Unix/Linux, MacOS, Windows)

```
git clone https://github.com/ginkgo-project/ginkgo/
cd ginkgo && mkdir build && cd build

cmake -DBUILD_REFERENCE=ON -DBUILD_GPU=ON ..

make

make test
```

Library Structure

Ginkgo

```
├── core
│   ├── base
│   ├── matrix
│   ├── preconditioner
│   ├── solver
│   └── test
│       ├── base
│       ├── matrix
│       ├── preconditioner
│       ├── solver
│       └── utils
├── reference
│   ├── matrix
│   ├── preconditioner
│   ├── solver
│   └── test
│       ├── base
│       ├── matrix
│       ├── preconditioner
│       └── solver
└── gpu
    ├── matrix
    ├── preconditioner
    ├── solver
    └── test
        ├── matrix
        ├── preconditioner
        └── solver
```

"Core" contains the algorithms, like ILU, CG, GMRES etc. Anything not device-specific.

Unit tests ensure correctness.

The "reference" implementations are ensuring correctness, they are not tuned for performance.

The "gpu" folder contains the tuned numerical kernels, e.g. spmv, dot, but also algorithm-specific kernels for the solvers, preconditioners…

Library Structure

```
├── core
│   ├── base
│   ├── matrix
│   ├── preconditioner
│   ├── solver
│   └── test
│       ├── base
│       ├── matrix
│       ├── preconditioner
│       ├── solver
│       └── utils
├── reference
│   ├── matrix
│   ├── preconditioner
│   ├── solver
│   └── test
│       ├── base
│       ├── matrix
│       ├── preconditioner
│       └── solver
└── gpu
    ├── matrix
    ├── preconditioner
    ├── solver
    └── test
        ├── matrix
        ├── preconditioner
        └── solver
```

Scenario: new ”CGS” solver

- CGS class
- Reference kernels for CGS
- GPU kernels for CGS

Library Structure

```
├── core
│   ├── base
│   ├── matrix
│   ├── preconditioner
│   ├── solver
│   └── test
│       ├── base
│       ├── matrix
│       ├── preconditioner
│       ├── solver
│       └── utils
├── reference
│   ├── matrix
│   ├── preconditioner
│   ├── solver
│   └── test
│       ├── base
│       ├── matrix
│       ├── preconditioner
│       └── solver
└── gpu
    ├── matrix
    ├── preconditioner
    ├── solver
    └── test
        ├── matrix
        ├── preconditioner
        └── solver
```

Scenario: new ”CGS” solver

- CGS class
  + test for class functions
- Reference kernels for CGS
  + test for each kernel
- GPU kernels for CGS
  + test for each kernel
  (compare with reference)

Creating a new solver via script

```
cd ginkgo/dev_tools/scripts
./create_new_solver.sh <name_of_new_solver>
…
```

Creating a new solver via script

```
cd ginkgo/dev_tools/scripts
./create_new_solver.sh cgs
…

Summary:
Created solver file
ginkgo/core/solver/cgs.cpp
    This is where the cgs algorithm needs to be implemented.

Created class header
ginkgo/core/solver/cgs.hpp
    This is where the cgs class functions need to be implemented.
…
A summary of the required next steps has been written to:
todo_cgs.txt
```

How to run the example solver

```
cd ginkgo/build/example/simple_solver/
```

```
./simple_solver
```

Runs reference code

```
./simple_solver gpu
```

Runs on the GPU

```cpp
// Read data
std::shared_ptr<mtx> A = mtx::create(exec);
A->read_from_mtx("data/A.mtx");
auto b = vec::create(exec);
b->read_from_mtx("data/b.mtx");
auto x = vec::create(exec);
x->read_from_mtx("data/x0.mtx");

// Generate solver
auto solver_gen = cg::create(exec, 20, 1e-20);
auto solver = solver_gen->generate(A);

// Solve system
solver->apply(b.get(), x.get());
```

Overhead of Runtime Polymorphism

```
cd ginkgo/build/example/ginkgo_overhead/

[hanzt@a05 ginkgo_overhead]$ ./ginkgo_overhead
Running 1000000 iterations of the CG solver took a total of 0.956637 seconds.
      Average library overhead:     956.637 [nanoseconds / iteration]
```

< 1 second overhead from runtime polymorphism for 1,000,000 iterations.

How about distributed?

- Currently not planned *(even though an easy add-on: just create another executor based on MPI)*
- ***Why?***
  - *One GPU node (potentially multiple GPUs) provides enormous performance.*
  - *Maybe there is something coming from NVIDIA in the next years?*
  - *We do not necessarily want to compete with existing sparse packages based on MPI (SuperLU, PetSC, Trilinos) – but work well in cooperation.*
  - *Limited manpower: Currently 4 developers at KIT, some contributions from NTU*

Why Ginkgo?