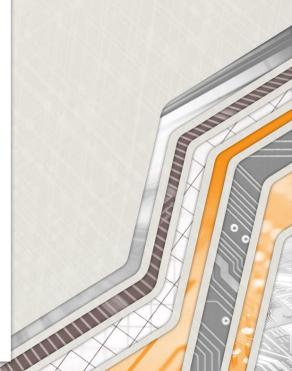
Modern Mixed-Precision Methods in Portable C++ for Accelerated Hardware Platforms

Neil Lindquist
SIAM PP22
February 26, 2021







Modern Numerical Methods

Precision

- Double
- Single
- Half
- Bfloat16
- Integer?
- Posits?

Hardware

- CPU
- GPUS
 - NVIDIA
 - AMD
 - Intel
- ML Accelerators
- FPGAs



Mixed Precision Experiments

- Many combinations of precisions to test
 - My GMRES work: 14 combinations in paper
 - Only single, double
 - GMRES-IR5¹: 112 "meaningful" combinations
 - Bfloat16, half, single, double, quad



Possible Types of Configuration

- 1. Read time
- 2. Compile time
- 3. Run time



Read time configuration

- Limited to string replacements
 - E.g., macros
- Error prone



Possible Types of Configuration

- 1. Read time
- 2. Compile time
- 3. Run time



Compile time configuration

- Templating Parameterize code
 - Both types and constants



Templating Type

```
template<class T>
T dot(int n, T^* \times T^* \times Y) {
  T sum = 0.0;
  for (int i = 0; i < n; i++) {
    sum += x[i]*y[i];
  return sum
```



Compile time configuration

- Templating Parameterize code
 - Both types and constants
- Can specialize for specific configurations
 - Different vendor libraries
 - Host vs device memory



Templating Devices

```
template<class T, class Target>
T dot(Vect<T, Target> x,
      Vect<T, Target> y);
template <>
double dot<double, MKL>(Vect<double, MKL> x,
                        Vect<double, MKL> y) {
    return cblas_ddot(x.n(), x.data(), 1,
                             y.data(), 1);
```

Templating Devices

```
template <>
double dot<double, Cuda>(Vect<double, Cuda> x,
                          Vect<double, Cuda> y) {
    double result;
    cublasDdot(cublas handle, x.n(),
               x.data(), 1,
               y.data(), 1,
               &result);
    return result;
```



Kokkos

- Performance Portability Library
- View generic multi-dimensional array
 - Template: type, device, memory layout
- Device-portable kernels

Similar ideas in Raja, ect.



Templating with Kokkos

```
template<class T, class S, class MemSpace,
         class R=std::common type<T, S>>
R dot(Kokkos::view<T*, MemSpace> x, Kokkos::view<S*, MemSpace> y){
 R out; int n = x.n();
 Kokkos::parallel reduce(
  Kokkos::RangePolicy<typename MemSpace::execution space>(∅,n),
  KOKKOS_LAMBDA(int i, R& partial_sum) {
   partial_sum += x(i)*y(i);
  }, Kokkos::Sum(out));
 return out;
```



Possible Types of Configuration

- 1. Read time
- 2. Compile time
- 3. Run time



Run time configuration

- More flexible APIs
- Manually wrap templated code



Wrapping Templated Routines

```
double dot(std::string target,
           Vect<double> x, Vect<double> y) {
  if (target == "MKL") {
     return dot<MKL>(x, y);
  if (target == "CUDA") {
     return dot<CUDA>(x, y);
```



Wrapping Templated Routines

- 2 devices & 2 precisions → 8 branches
- 2 devices & 3 precisions → 18 branches
- 3 devices & 4 precisions → 48 branches



Run time configuration

- More flexible APIs
- Can manually wrap templated code
- Runtime types uncommon

- DPC++: runtime devices
- OCCA: runtime types, devices



Conclusions

- Combinatorial explosions of implementations
 - Precisions and accelerators
- Can alleviate with
 - Templating
 - Portability libraries

