Introduction to the YAKL C++ Portability Library

Matt Norman

E3SM All Hands Meeting, March 3, 2022

ORNL is managed by UT-Battelle, LLC for the US Department of Energy

Acknowledgements

- This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

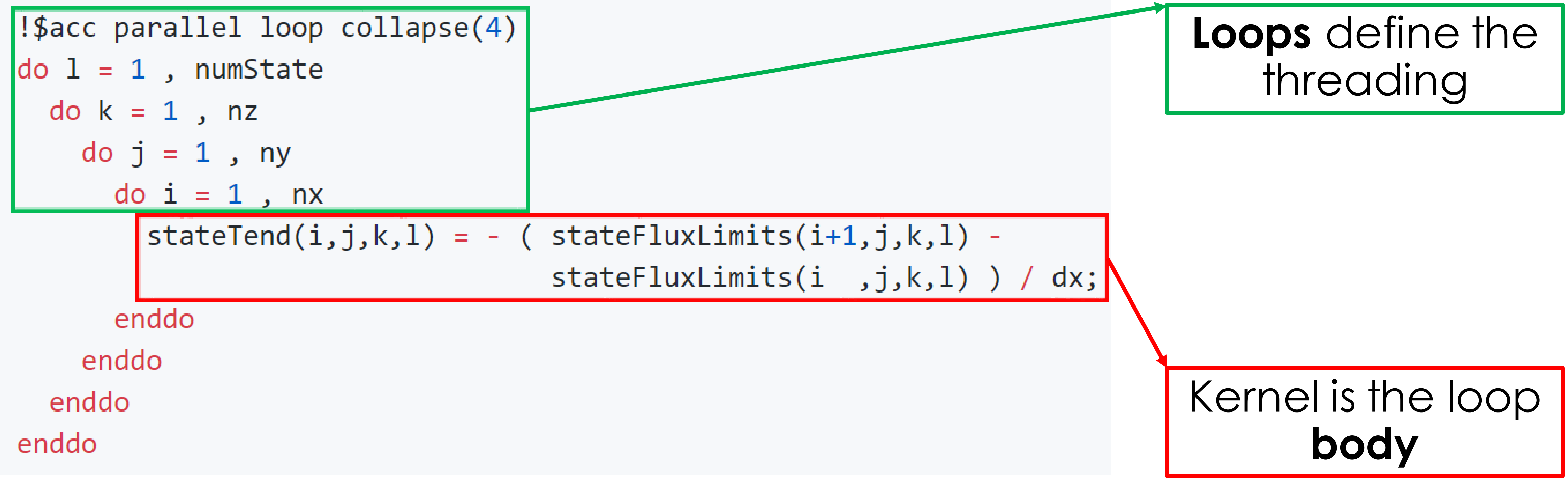
- This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy’s Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation’s exascale computing imperative.

What is Portable C++

- It is a C++ library, not a separate language or language extension
- It uses the same information you already give in non-parallel code
- You can pass code as an object in C++
- That code, along with information about your loops, is sent to a backend
  - CUDA, HIP, OpenMP, SYCL, etc.
- A single source code runs in parallel on many different hardware backends
- Portable C++ libraries often come with other features
  - Multi-dimensional arrays
  - Ways to handle race conditions (reductions, atomics, etc.)
  - Ways to manage data between two different devices

What is Portable C++?

- Just a C++ library, not a separate language or a language extension
- Based on the “kernel” paradigm (CUDA and HIP):
  - A kernel performs work on a single thread
  - Let the launcher know how many threads to launch
  - Requires no more work or information than you’re already used to providing

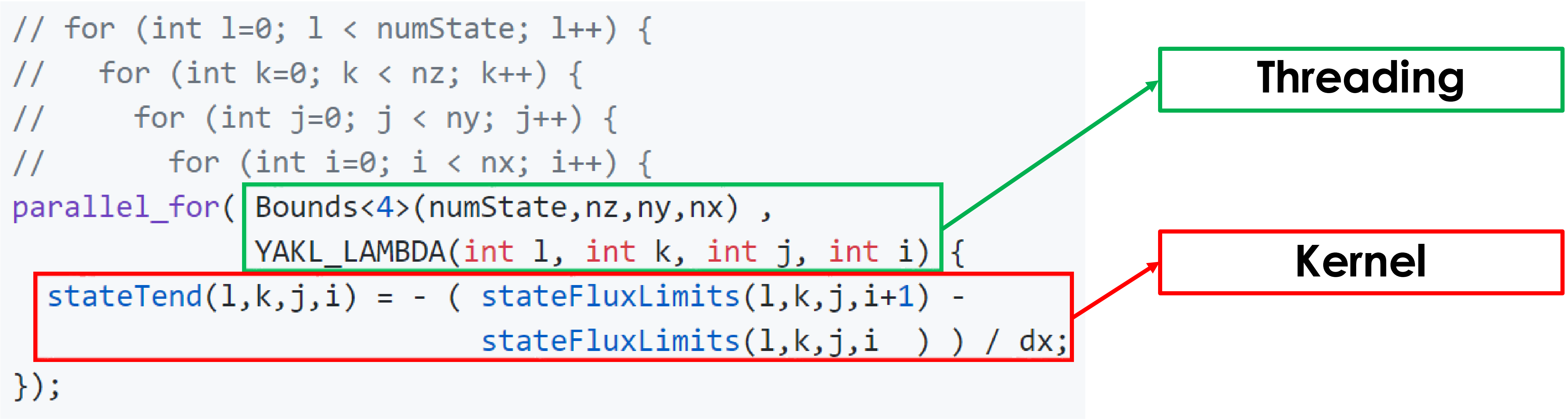


```
!$acc parallel loop collapse(4)
do l = 1 , numState
  do k = 1 , nz
    do j = 1 , ny
      do i = 1 , nx
        stateTend(i,j,k,l) = - ( stateFluxLimits(i+1,j,k,l) -
                                 stateFluxLimits(i  ,j,k,l) ) / dx;
      enddo
    enddo
  enddo
enddo
```

Loops define the threading

Kernel is the loop **body**

The Core of Portable C++



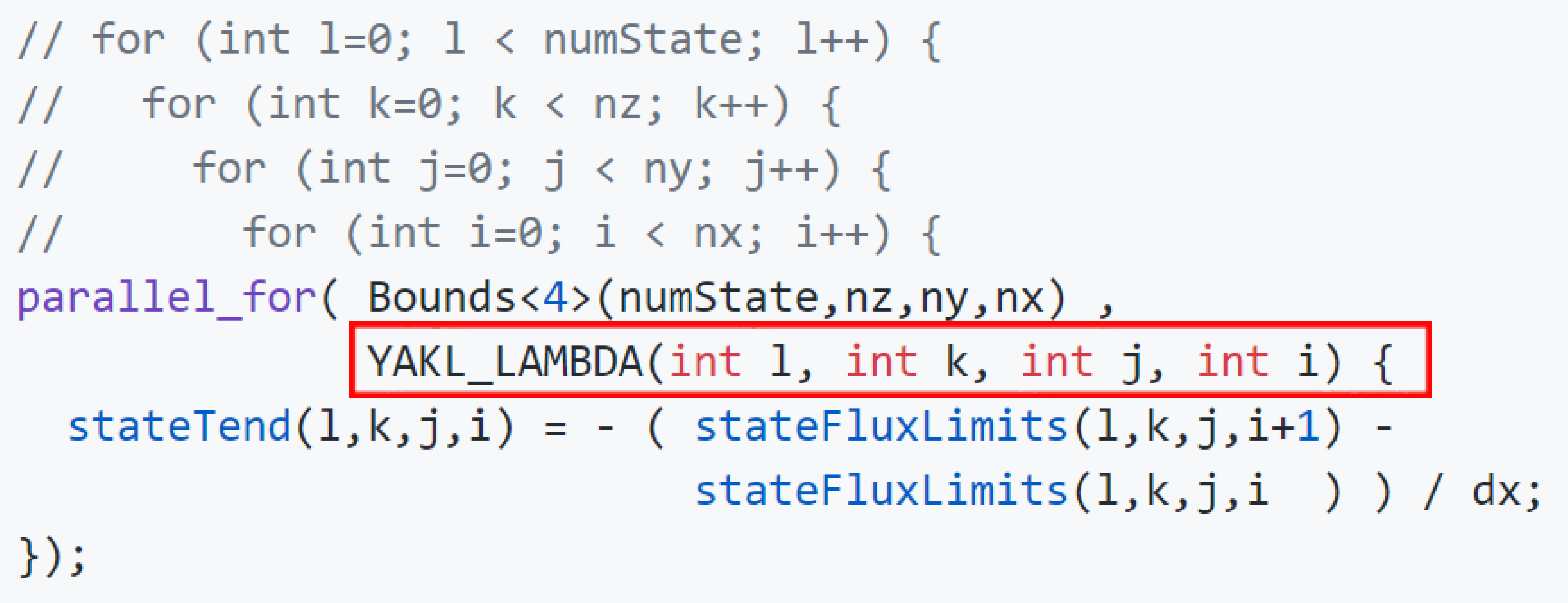
```
// for (int l=0; l < numState; l++) {
//   for (int k=0; k < nz; k++) {
//     for (int j=0; j < ny; j++) {
//       for (int i=0; i < nx; i++) {
parallel_for( Bounds<4>(numState,nz,ny,nx) ,
              YAKL_LAMBDA(int l, int k, int j, int i) {
  stateTend(l,k,j,i) = - ( stateFluxLimits(l,k,j,i+1) -
                           stateFluxLimits(l,k,j,i  ) ) / dx;
});
```

Threading

Kernel

The Core of Portable C++

- C++ can pass code as an object



```
// for (int l=0; l < numState; l++) {
//   for (int k=0; k < nz; k++) {
//     for (int j=0; j < ny; j++) {
//       for (int i=0; i < nx; i++) {
parallel_for( Bounds<4>(numState,nz,ny,nx) ,
              YAKL_LAMBDA(int l, int k, int j, int i) {
  stateTend(l,k,j,i) = - ( stateFluxLimits(l,k,j,i+1) -
                           stateFluxLimits(l,k,j,i  ) ) / dx;
});
```

- C++ “lambdas” convert code into a class object for you
- You can then pass the code to whatever backend you want
  - “parallel_for” can launch with CUDA, HIP, OpenMP, OpenMP 4.5+, SYCL, etc.
- Just as flexible and generic as directives

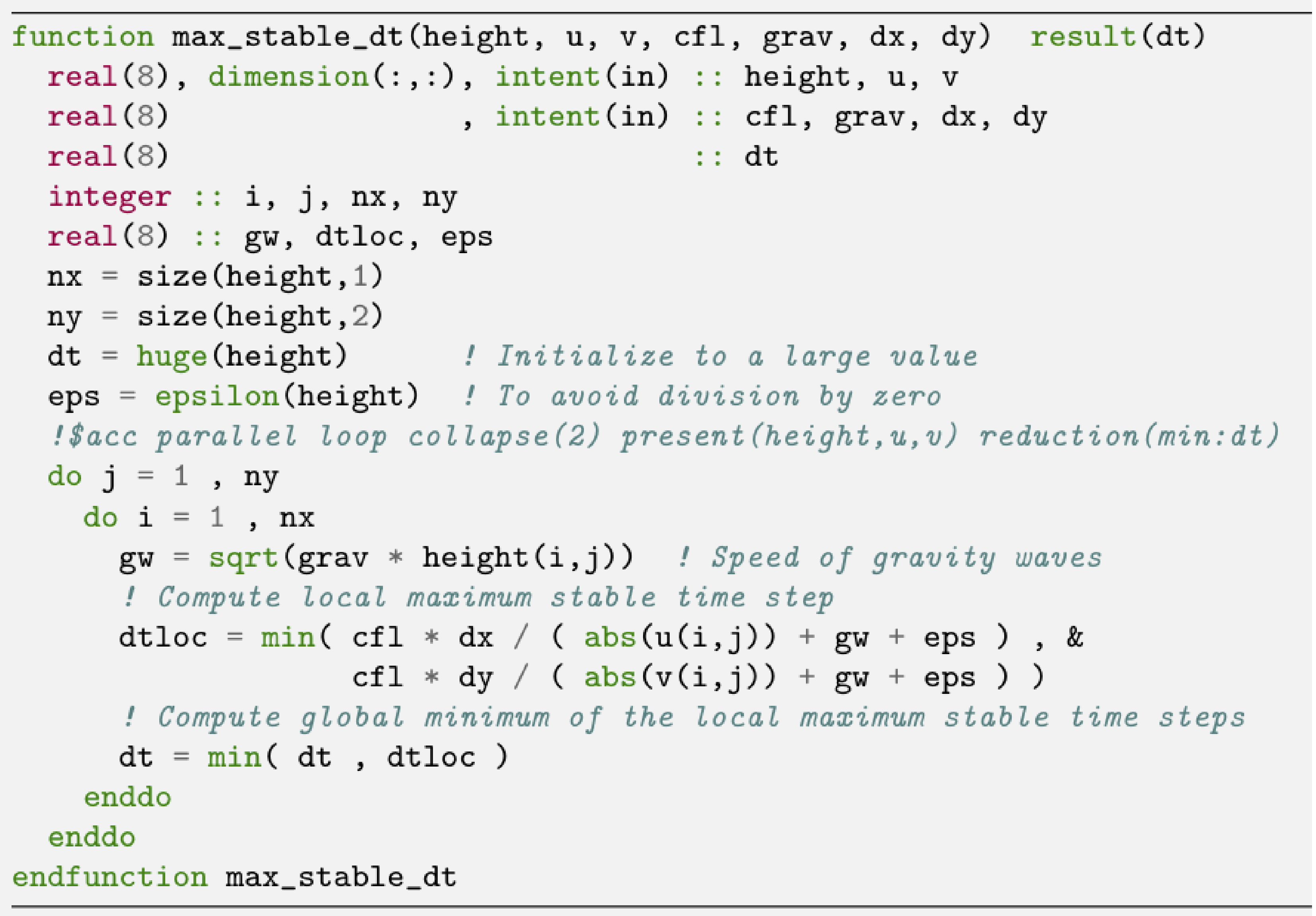
Yet Another Kernel Launcher (YAKL)

- C++ portability library emphasizing simplicity and porting Fortran code to C++
  - https://github.com/mrnorman/YAKL
- Currently supports:
  - CUDA (Nvidia GPUs)
  - HIP (AMD GPUs)
  - SYCL (Intel GPUs)
  - CPUs in serial and with OpenMP CPU threading
  - OpenMP target offload (in progress)
- YAKL started as a stop gap while HIP was unsupported by Kokkos
- Turned into a helpful avenue to handling large Fortran codes
- YAKL is quite small (8K lines of code), developed with < 1 FTE total effort

YAKL Features

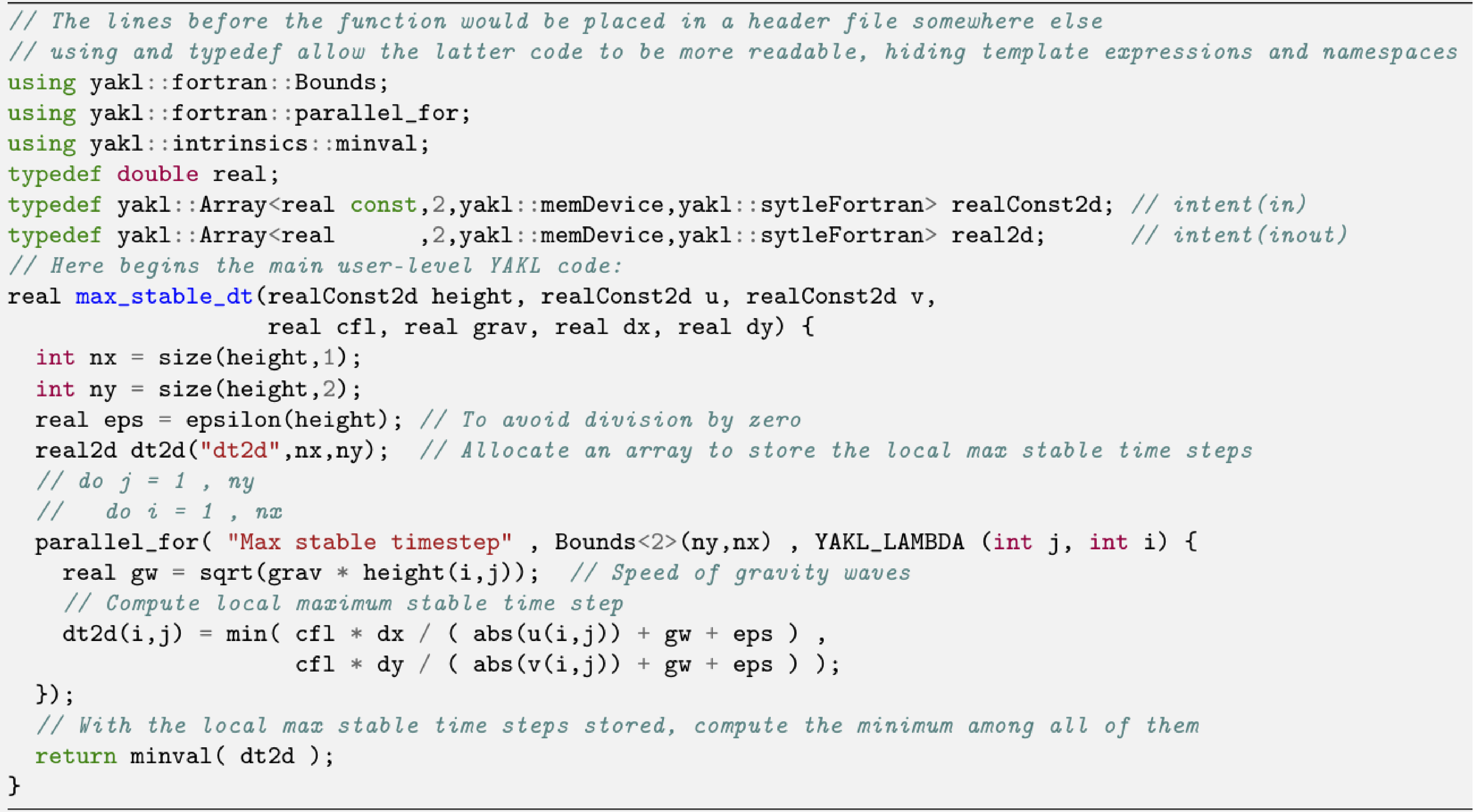
- parallel_for kernel launchers
- Multi-dimensional arrays (dynamic and static) in C and Fortran styles
- Functions to move data between host and GPU memory spaces
- An efficient non-blocking pool allocator for cheap allocation / free
  - With fortran bindings to share data with Fortran codes
- Atomic and reduction operators for race conditions
- Synchronization for asynchronous work
- Limited Fortran intrinsics library (minval, sum, size, allocated, pack, etc.)
- NetCDF and PNetCDF I/O and automated timers

Example YAKL Conversion (Fortran Code)



```fortran
function max_stable_dt(height, u, v, cfl, grav, dx, dy)  result(dt)
  real(8), dimension(:,:), intent(in) :: height, u, v
  real(8)                , intent(in) :: cfl, grav, dx, dy
  real(8)                             :: dt
  integer :: i, j, nx, ny
  real(8) :: gw, dtloc, eps
  nx = size(height,1)
  ny = size(height,2)
  dt = huge(height)       ! Initialize to a large value
  eps = epsilon(height)   ! To avoid division by zero
  !$acc parallel loop collapse(2) present(height,u,v) reduction(min:dt)
  do j = 1 , ny
    do i = 1 , nx
      gw = sqrt(grav * height(i,j))  ! Speed of gravity waves
      ! Compute local maximum stable time step
      dtloc = min( cfl * dx / ( abs(u(i,j)) + gw + eps ) , &
                   cfl * dy / ( abs(v(i,j)) + gw + eps ) )
      ! Compute global minimum of the local maximum stable time steps
      dt = min( dt , dtloc )
    enddo
  enddo
endfunction max_stable_dt
```

Example YAKL Conversion (YAKL Code – Fortran-style)



```
// The lines before the function would be placed in a header file somewhere else
// using and typedef allow the latter code to be more readable, hiding template expressions and namespaces
using yakl::fortran::Bounds;
using yakl::fortran::parallel_for;
using yakl::intrinsics::minval;
typedef double real;
typedef yakl::Array<real const,2,yakl::memDevice,yakl::sytleFortran> realConst2d; // intent(in)
typedef yakl::Array<real      ,2,yakl::memDevice,yakl::sytleFortran> real2d;      // intent(inout)
// Here begins the main user-level YAKL code:
real max_stable_dt(realConst2d height, realConst2d u, realConst2d v,
                   real cfl, real grav, real dx, real dy) {
  int nx = size(height,1);
  int ny = size(height,2);
  real eps = epsilon(height); // To avoid division by zero
  real2d dt2d("dt2d",nx,ny);  // Allocate an array to store the local max stable time steps
  // do j = 1 , ny
  //   do i = 1 , nx
  parallel_for( "Max stable timestep" , Bounds<2>(ny,nx) , YAKL_LAMBDA (int j, int i) {
    real gw = sqrt(grav * height(i,j));  // Speed of gravity waves
    // Compute local maximum stable time step
    dt2d(i,j) = min( cfl * dx / ( abs(u(i,j)) + gw + eps ) ,
                     cfl * dy / ( abs(v(i,j)) + gw + eps ) );
  });
  // With the local max stable time steps stored, compute the minimum among all of them
  return minval( dt2d );
}
```

Codes Developed or Ported with YAKL

1. System for Atmsopheric Modeling (SAM)
2. Portable Atmosphere Model (PAM)
3. RRTMGP radiation code
4. "AWFL Shallow" Shallow-Water Model
5. MiniWeather mini-app (github.com/mrnorman/miniWeather)
6. Preliminary investigations into using YAKL for MPAS-O