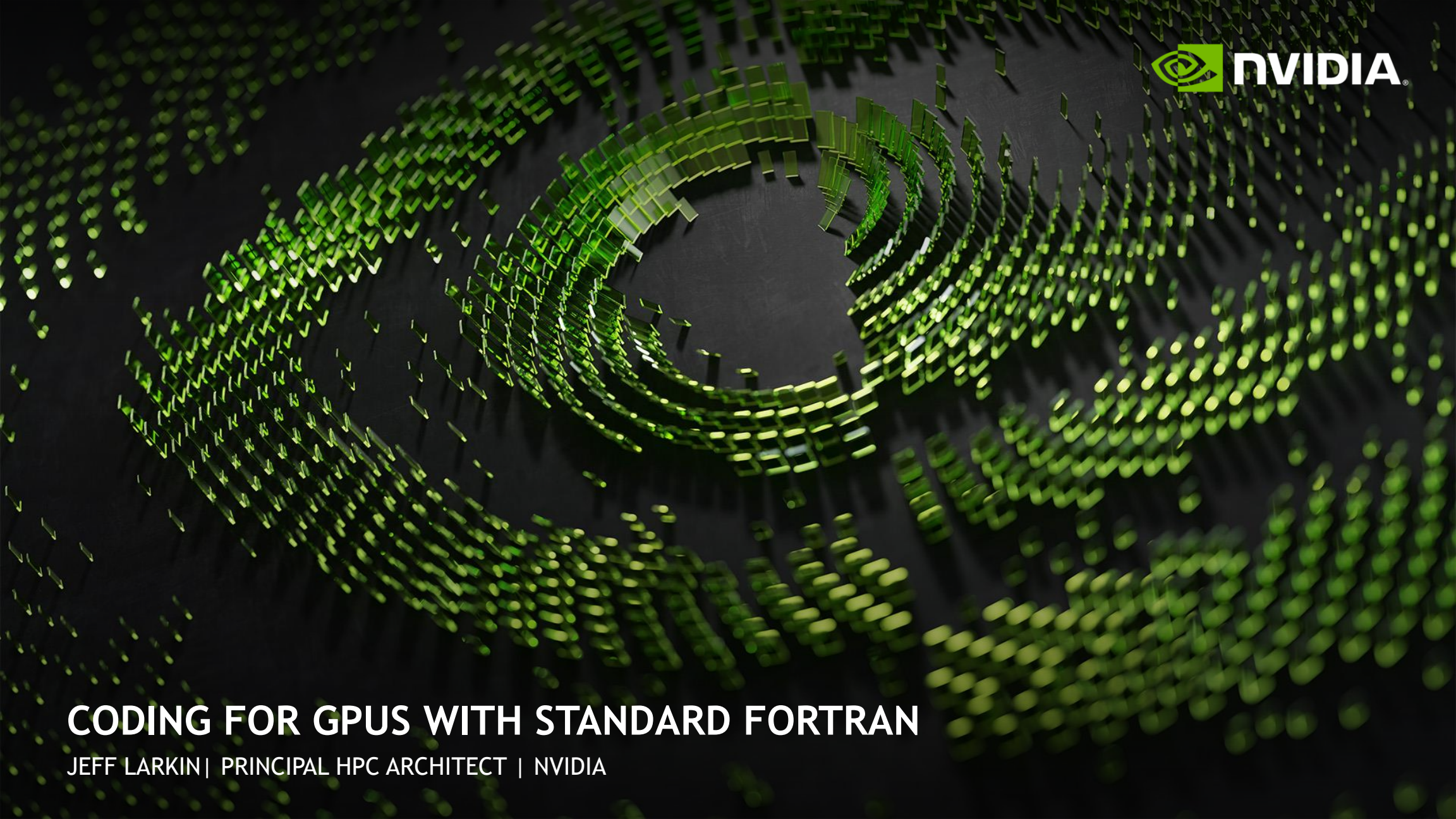


CODING FOR GPUS WITH STANDARD FORTRAN

JEFF LARKIN| PRINCIPAL HPC ARCHITECT | NVIDIA

PROGRAMMING THE NVIDIA PLATFORM

CPU, GPU, and Network

ACCELERATED STANDARD LANGUAGES

ISO C++, ISO Fortran

```
std::transform(par, x, x+n, y, y,
    [=](float x, float y){ return y + a*x; }
);
```

```
do concurrent (i = 1:n)
   y(i) = y(i) + a*x(i)
enddo
```

```
import cunumeric as np
…
def saxpy(a, x, y):
    y[:] += a*x
```

INCREMENTAL PORTABLE OPTIMIZATION

OpenACC, OpenMP

```
!$acc data copy(x,y)
...
do concurrent (i = 1:n)
   y(i) = y(i) + a*x(i)
enddo
...
!$acc end data
```

```
!$omp target data map(x,y)
...
do concurrent (i = 1:n)
   y(i) = y(i) + a*x(i)
enddo
...
!$omp target end data
```

PLATFORM SPECIALIZATION

CUDA

```
attributes(global) subroutine saxpy(x,y,a)
  implicit none
  real :: x(:), y(:)
  real, value :: a
  integer :: i, n
  n = size(x)
  i = blockDim%x * (blockIdx%x - 1) +
threadIdx%x
  if (i <= n) y(i) = y(i) + a*x(i)
end subroutine saxpy

program main
  ...
  x_d = x
  y_d = y
  call saxpy<<<grid, tBlock>>>(x_d,y_d,a)
  y = y_d
  ...
end program
```

ACCELERATION LIBRARIES

Core	Math	Communication	Data Analytics	AI	Quantum

ACCELERATED STANDARD LANGUAGES

Parallel performance for wherever your code runs

ISO C++

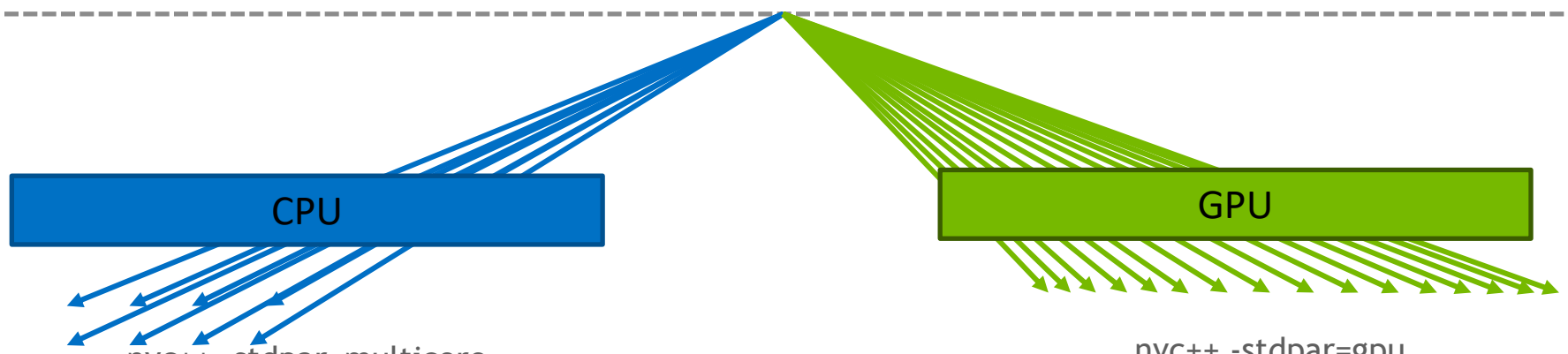
```
std::transform(par, x, x+n, y,
   y,[=](float x, float y){
      return y + a*x;
    }
);
```

ISO Fortran

```
do concurrent (i = 1:n)
   y(i) = y(i) + a*x(i)
enddo
```

Python

```
import cunumeric as np
…
def saxpy(a, x, y):
    y[:] += a*x
```



nvc++ -stdpar=multicore
nvfortran -stdpar=multicore
legate -cpus 16 saxpy.py

nvc++ -stdpar=gpu
nvfortran -stdpar=gpu
legate -gpus 1 saxpy.py


PARALLEL PROGRAMMING WITH ISO
FORTRAN

HPC PROGRAMMING IN ISO FORTRAN

ISO is the place for portable concurrency and parallelism

Fortran 2018

Fortran Array Intrinsics

- NVFORTRAN 20.5
- Accelerated matmul, reshape, spread, …

DO CONCURRENT

- NVFORTRAN 20.11
- Auto-offload & multi-core

Co-Arrays

- Not currently available
- Accelerated co-array images

Preview support available now in NVFORTRAN

Fortran 202x

DO CONCURRENT Reductions

- NVFORTRAN 21.11
- `REDUCE` subclause added
- Support for `+`, `*`, `MIN`, `MAX`, `IAND`, `IOR`, `IEOR`.
- Support for `.AND.`, `.OR.`, `.EQV.`, `.NEQV` on `LOGICAL` values

HPC PROGRAMMING IN ISO FORTRAN

DO CONCURRENT

DO CONCURRENT in NVFORTRAN

- Available since NVFORTRAN 20.11
- Automatic GPU acceleration & multi-core support
- Syntax for nested parallelism / loop collapse; expose more parallelism to the compiler

```
subroutine smooth( a, b, w0, w1, w2, n, m, niters )
 real, dimension(:,:) :: a,b
 real :: w0, w1, w2
 integer :: n, m, niters
 integer :: i, j, iter
  do iter = 1,niters
   do concurrent(i=2 : n-1, j=2 : m-1)
     a(i,j) = w0 * b(i,j) + &
              w1 * (b(i-1,j) + b(i,j-1) + b(i+1,j) + b(i,j+1)) + &
              w2 * (b(i-1,j-1) + b(i-1,j+1) + b(i+1,j-1) + b(i+1,j+1))
   enddo
   do concurrent(i=2 : n-1, j=2 : m-1)
     b(i,j) = w0 * a(i,j) + &
              w1 * (a(i-1,j) + a(i,j-1) + a(i+1,j) + a(i,j+1)) + &
              w2 * (a(i-1,j-1) + a(i-1,j+1) + a(i+1,j-1) + a(i+1,j+1))
   enddo
  enddo
```

MINIWEATHER

Standard Language Parallelism in Climate/Weather Applications

MiniWeather

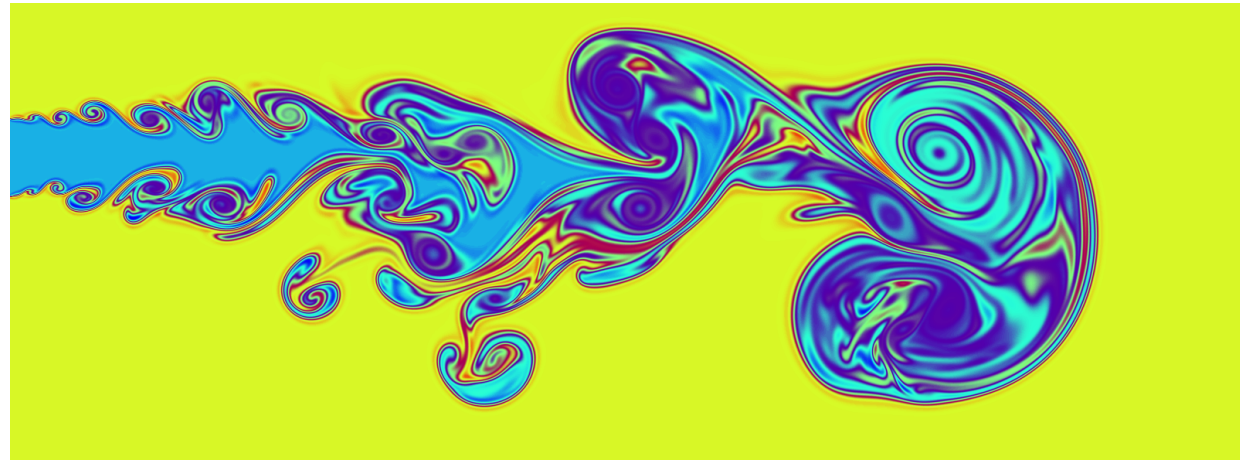
Mini-App written in C++ and Fortran that simulates weather-like fluid flows using Finite Volume and Runge-Kutta methods.

Existing parallelization in MPI, OpenMP, OpenACC, …

Included in the SPEChpc benchmark suite*

Open-source and commonly-used in training events.

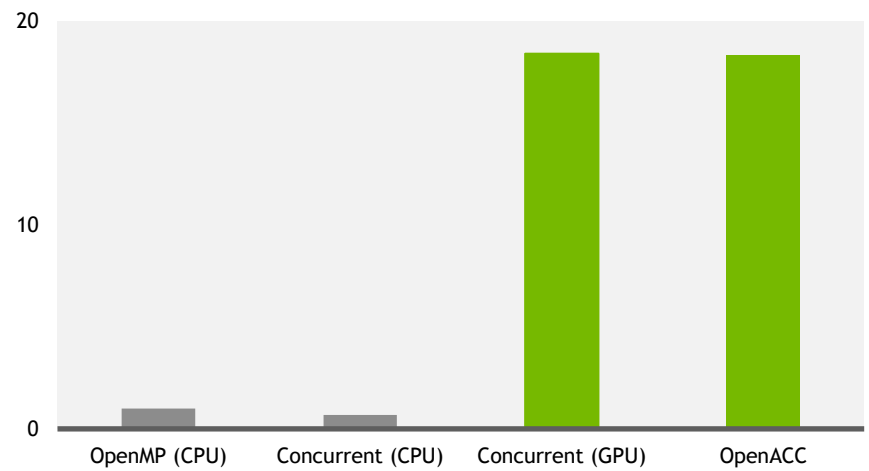
https://github.com/mrnorman/miniWeather/



```
do concurrent (ll=1:NUM_VARS, k=1:nz, i=1:nx)
      local(x,z,x0,z0,xrad,zrad,amp,dist,wpert)

  if (data_spec_int == DATA_SPEC_GRAVITY_WAVES) then
    x = (i_beg-1 + i-0.5_rp) * dx
    z = (k_beg-1 + k-0.5_rp) * dz
        x0 = xlen/8
    z0 = 1000
    xrad = 500
    zrad = 500
    amp = 0.01_rp
    dist = sqrt( ((x-x0)/xrad)**2 + ((z-z0)/zrad)**2 )
         * pi / 2._rp
    if (dist <= pi / 2._rp) then
      wpert = amp * cos(dist)**2
    else
      wpert = 0._rp
    endif
    tend(i,k,ID_WMOM) = tend(i,k,ID_WMOM)
                      + wpert*hy_dens_cell(k)
  endif
  state_out(i,k,ll) = state_init(i,k,ll)
                    + dt * tend(i,k,ll)

enddo
```



Source: HPC SDK 22.1, AMD EPYC 7742, NVIDIA A100. MiniWeather: NX=2000, NZ=1000, SIM_TIME=5. OpenACC version uses -gpu=managed option.

*SPEChpc is a trademark of The Standard Performance Evaluation Corporation

NVIDIA

POT3D: DO CONCURRENT + LIMITED OPENACC

POT3D

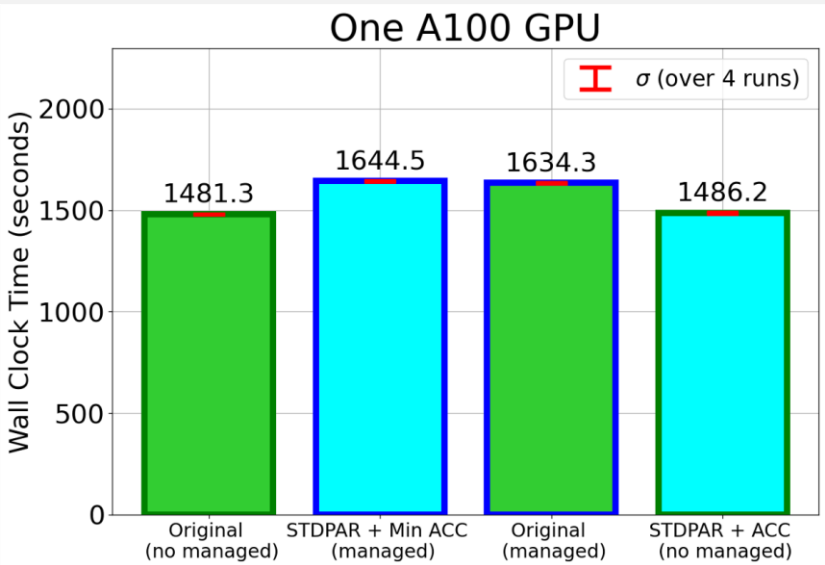
POT3D is a Fortran application for approximating solar coronal magnetic fields.

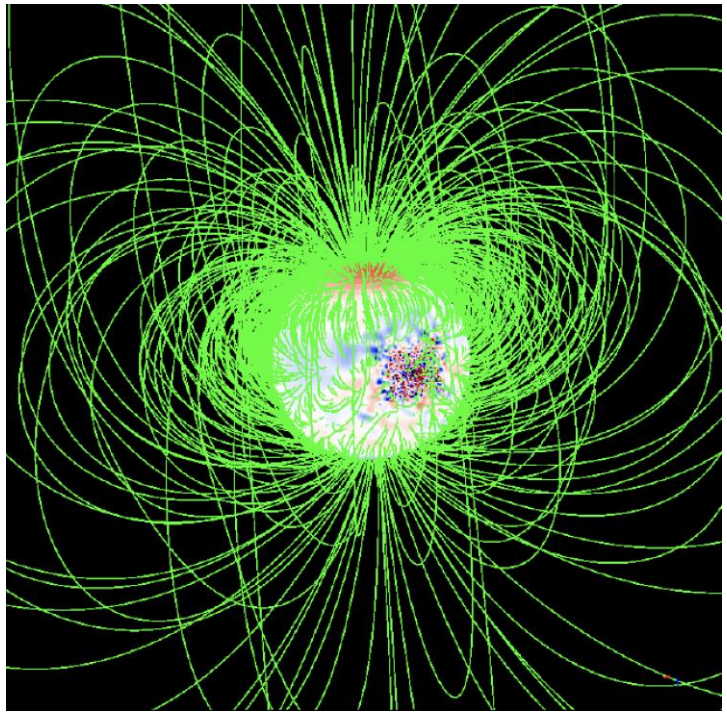
Included in the SPEChpc benchmark suite*

Existing parallelization in MPI & OpenACC

Optimized the DO CONCURRENT version by using OpenACC solely for data motion and atomics

https://github.com/predsci/POT3D





```
!$acc enter data copyin(phi,dr_i)
!$acc enter data create(br)
do concurrent (k=1:np,j=1:nt,i=1:nrm1)
  br(i,j,k)=(phi(i+1,j,k)-phi(i,j,k ))*dr_i(i)
enddo
!$acc exit data delete(phi,dr_i,br)
```

Data courtesy of Predictive Science Inc.

*SPEChpc is a trademark of The Standard Performance Evaluation Corporation

ACCELERATED PROGRAMMING IN ISO FORTRAN

NVFORTRAN Accelerates Fortran Intrinsics with cuTENSOR Backend

```
real(8), dimension(ni,nk) :: a
real(8), dimension(nk,nj) :: b
real(8), dimension(ni,nj) :: c
...
!$acc enter data copyin(a,b,c) create(d)

do nt = 1, ntimes
  !$acc kernels
  do j = 1, nj
    do i = 1, ni
      d(i,j) = c(i,j)
      do k = 1, nk
        d(i,j) = d(i,j) + a(i,k) * b(k,j)
      end do
    end do
  end do
    !$acc end kernels
end do
!$acc exit data copyout(d)
```

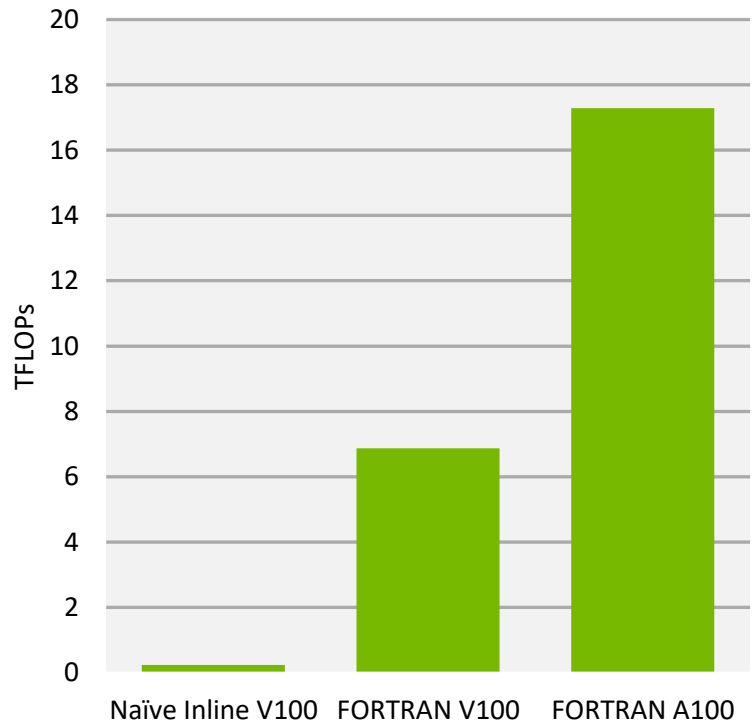
Inline FP64 matrix multiply

```
real(8), dimension(ni,nk) :: a
real(8), dimension(nk,nj) :: b
real(8), dimension(ni,nj) :: c

...

do nt = 1, ntimes
  d = c + matmul(a,b)
end do
```

MATMUL FP64 matrix multiply



HPC PROGRAMMING IN ISO FORTRAN

Examples of Patterns Accelerated in NVFORTRAN

```
d = 2.5 * ceil(transpose(a)) + 3.0 * abs(transpose(b))
d = 2.5 * ceil(transpose(a)) + 3.0 * abs(b)
d = reshape(a,shape=[ni,nj,nk])
d = reshape(a,shape=[ni,nk,nj])
d = 2.5 * sqrt(reshape(a,shape=[ni,nk,nj],order=[1,3,2]))
d = alpha * conjg(reshape(a,shape=[ni,nk,nj],order=[1,3,2]))
d = reshape(a,shape=[ni,nk,nj],order=[1,3,2])
d = reshape(a,shape=[nk,ni,nj],order=[2,3,1])
d = reshape(a,shape=[ni*nj,nk])
d = reshape(a,shape=[nk,ni*nj],order=[2,1])
d = reshape(a,shape=[64,2,16,16,64],order=[5,2,3,4,1])
d = abs(reshape(a,shape=[64,2,16,16,64],order=[5,2,3,4,1]))
c = matmul(a,b)
c = matmul(transpose(a),b)
c = matmul(reshape(a,shape=[m,k],order=[2,1]),b)
c = matmul(a,transpose(b))
c = matmul(a,reshape(b,shape=[k,n],order=[2,1]))
```

```
c = matmul(transpose(a),transpose(b))
c = matmul(transpose(a),reshape(b,shape=[k,n],order=[2,1]))
d = spread(a,dim=3,ncopies=nk)
d = spread(a,dim=1,ncopies=ni)
d = spread(a,dim=2,ncopies=nx)
d = alpha * abs(spread(a,dim=2,ncopies=nx))
d = alpha * spread(a,dim=2,ncopies=nx)
d = abs(spread(a,dim=2,ncopies=nx))
d = transpose(a)
d = alpha * transpose(a)
d = alpha * ceil(transpose(a))
d = alpha * conjg(transpose(a))
c = c + matmul(a,b)
c = c - matmul(a,b)
c = c + alpha * matmul(a,b)
d = alpha * matmul(a,b) + c
d = alpha * matmul(a,b) + beta * c
```

REFACTORING STEP BY STEP

REFACTORING FORTRAN LOOPS

1. Identify an important loop nest that can be run in parallel.

```
!Compute fluxes in the x-direction for each cell
do k = 1 , nz+1
    do i = 1 , nx
      !Use fourth-order interpolation from four cell averages
      !to compute the value at the interface in question
      do ll = 1 , NUM_VARS
        do s = 1 , sten_size
          stencil(s) = state(i,k-hs-1+s,ll)
        enddo
        !Fourth-order-accurate interpolation of the state
      enddo

      !Compute density, u-wind, w-wind, potential
      !temperature, and pressure (r,u,w,t,p respectively)
      r = vals(ID_DENS) + hy_dens_int(k)
      u = vals(ID_UMOM) / r
      w = vals(ID_WMOM) / r
      t = ( vals(ID_RHOT) + hy_dens_theta_int(k) ) / r
      p = C0*(r*t)**gamma - hy_pressure_int(k)

      ...

    enddo
  enddo
```


*Code from MiniWeather mini-app, trimmed for space.

REFACTORING FORTRAN LOOPS

1. Identify an important loop nest that can be run in parallel.
2. Replace existing loops with do concurrent loops
   - Note: Multiple loop iteration variables can be used in the same do concurrent loop, if they are all legal to parallelize

```
!Compute fluxes in the x-direction for each cell
do concurrent (k=1:nz, i=1:nx+1)
      !Use fourth-order interpolation from four cell averages
      !to compute the value at the interface in question
      do ll = 1 , NUM_VARS
        do s = 1 , sten_size
          stencil(s) = state(i,k-hs-1+s,ll)
        enddo
        !Fourth-order-accurate interpolation of the state
      enddo

      !Compute density, u-wind, w-wind, potential
      !temperature, and pressure (r,u,w,t,p respectively)
      r = vals(ID_DENS) + hy_dens_int(k)
      u = vals(ID_UMOM) / r
      w = vals(ID_WMOM) / r
      t = ( vals(ID_RHOT) + hy_dens_theta_int(k) ) / r
      p = C0*(r*t)**gamma - hy_pressure_int(k)

      ...

  enddo
```

*Code from MiniWeather mini-app, trimmed for space.

REFACTORING FORTRAN LOOPS

1. Identify an important loop nest that can be run in parallel.
2. Replace existing loops with do concurrent loops
   - Note: Multiple loop iteration variables can be used in the same do concurrent loop, if they are all legal to parallelize
3. Add local clause for variables that must be privatized for correctness.

```
!Compute fluxes in the x-direction for each cell
do concurrent (k=1:nz, i=1:nx+1)   &
  local(d3_vals,vals,stencil,ll,s,r,u,t,p,w)
   !Use fourth-order interpolation from four cell averages
   !to compute the value at the interface in question
    do ll = 1 , NUM_VARS
      do s = 1 , sten_size
        stencil(s) = state(i,k-hs-1+s,ll)
      enddo
      !Fourth-order-accurate interpolation of the state
    enddo

    !Compute density, u-wind, w-wind, potential
    !temperature, and pressure (r,u,w,t,p respectively)
    r = vals(ID_DENS) + hy_dens_int(k)
    u = vals(ID_UMOM) / r
    w = vals(ID_WMOM) / r
    t = ( vals(ID_RHOT) + hy_dens_theta_int(k) ) / r
    p = C0*(r*t)**gamma - hy_pressure_int(k)

   ...

enddo
```

*Code from MiniWeather mini-app, trimmed for space.

REFACTORING FORTRAN LOOPS

1. Identify an important loop nest that can be run in parallel.
2. Replace existing loops with do concurrent loops
   - Note: Multiple loop iteration variables can be used in the same do concurrent loop, if they are all legal to parallelize
3. Add local clause for variables that must be privatized for correctness.
4. Recompile with –stdpar and test for correctness.
   - Note 1: Only refactor one loop nest at a time to ensure errors aren't introduced, such as forgetting to localize a variable.
   - Note 2: Performance may get worse at first due to increased memory migration.

```
!Compute fluxes in the x-direction for each cell
do concurrent (k=1:nz, i=1:nx+1)   &
  local(d3_vals,vals,stencil,ll,s,r,u,t,p,w)
   !Use fourth-order interpolation from four cell averages
   !to compute the value at the interface in question
    do ll = 1 , NUM_VARS
      do s = 1 , sten_size
        stencil(s) = state(i,k-hs-1+s,ll)
      enddo
      !Fourth-order-accurate interpolation of the state
    enddo

    !Compute density, u-wind, w-wind, potential
    !temperature, and pressure (r,u,w,t,p respectively)
    r = vals(ID_DENS) + hy_dens_int(k)
    u = vals(ID_UMOM) / r
    w = vals(ID_WMOM) / r
    t = ( vals(ID_RHOT) + hy_dens_theta_int(k) ) / r
    p = C0*(r*t)**gamma - hy_pressure_int(k)

   ...

enddo
```

*Code from MiniWeather mini-app, trimmed for space.

REFACTORING FORTRAN LOOPS

1. Identify an important loop nest that can be run in parallel.
2. Replace existing loops with do concurrent loops
   - Note: Multiple loop iteration variables can be used in the same do concurrent loop, if they are all legal to parallelize
3. Add local clause for variables that must be privatized for correctness.
4. Recompile with –stdpar and test for correctness.
   - Note 1: Only refactor one loop nest at a time to ensure errors aren't introduced, such as forgetting to localize a variable.
   - Note 2: Performance may get worse at first due to increased memory migration.
5. Increase the number of concurrent loops to run more work in parallel and reduce memory migration on GPU.

```
!Compute fluxes in the x-direction for each cell
do concurrent (k=1:nz, i=1:nx+1)   &
  local(d3_vals,vals,stencil,ll,s,r,u,t,p,w)
    !Use fourth-order interpolation from four cell averages
   !to compute the value at the interface in question
    do ll = 1 , NUM_VARS
      do s = 1 , sten_size
        stencil(s) = state(i,k-hs-1+s,ll)
      enddo
      !Fourth-order-accurate interpolation of the state
    enddo

    !Compute density, u-wind, w-wind, potential
   !temperature, and pressure (r,u,w,t,p respectively)
    r = vals(ID_DENS) + hy_dens_int(k)
    u = vals(ID_UMOM) / r
    w = vals(ID_WMOM) / r
    t = ( vals(ID_RHOT) + hy_dens_theta_int(k) ) / r
    p = C0*(r*t)**gamma - hy_pressure_int(k)

   ...
enddo

do concurrent (k=1:nz,i=1:nx) reduce(+:mass,te)
    mass = mass + r            *dx*dz ! Accumulate domain mass
    te   = te   + (ke + r*cv*t)*dx*dz
enddo
```

Reduce is a Fortran 202X feature, supported since nvfortran 21.11.

*Code from MiniWeather mini-app, trimmed for space.


HINTS FOR FORTRAN STANDARD
PARALLEL PROGRAMMING

FORTRAN PARALLEL PROGRAMMING HINTS

Index Order Matters

Nested Loops

```
! 221 GB/s
do (i=1,order)   ! Slowest Dimension
  do (j=1,order) ! Fastest Dimension
    B(i,j) = B(i,j) + A(j,i)
  enddo
enddo

! 1050 GB/s
do (j=1,order)   ! Slowest Dimension
  do (i=1,order) ! Fastest Dimension
    B(i,j) = B(i,j) + A(j,i)
  enddo
enddo
```

Do Concurrent

```
! 221 GB/s
! Slowest, Then Fastest Dimension
do concurrent (i=1:order, j=1:order)
  B(i,j) = B(i,j) + A(j,i)
enddo



! 1050 GB/s
! Slowest, Then Fastest Dimension
do concurrent (j=1:order, i=1:order)
  B(i,j) = B(i,j) + A(j,i)
enddo
```

FORTRAN PARALLEL PROGRAMMING HINTS

Interfacing with CUDA Data

To optimize data movement and interface with CUDA libraries (Math, MPI, etc.) OpenACC can be used.

Top opt out of managed memory, build with -gpu=nomanaged

Note: If you take control of some data movement you will have to take control of all, so this is not our preferred approach.

With OpenACC Data Directives

```
!$acc enter data create(sbuf11,sbuf12,rbuf11,rbuf12)

!$acc host_data use_device(sbuf11,sbuf12,rbuf11,rbuf12)
call MPI_Isend (sbuf11,lbuf,ntype_real,iproc_rp,tag,
&               comm_all,reqs(1),ierr)
call MPI_Isend (sbuf12,lbuf,ntype_real,iproc_rm,tag,
&               comm_all,reqs(2),ierr)
call MPI_Irecv (rbuf11,lbuf,ntype_real,iproc_rm,tag,
&               comm_all,reqs(3),ierr)
call MPI_Irecv (rbuf12,lbuf,ntype_real,iproc_rp,tag,
&               comm_all,reqs(4),ierr)
call MPI_Waitall (4,reqs,MPI_STATUSES_IGNORE,ierr)
!$acc end host_data

if (iproc_rm.ne.MPI_PROC_NULL) then
  do concurrent (j=1:n3, i=1:n2)
    a( 1,i,j)=rbuf11(i,j)
  enddo
end if
if (iproc_rp.ne.MPI_PROC_NULL) then
  do concurrent (j=1:n3, i=1:n2)
    a(n1,i,j)=rbuf12(i,j)
  enddo
end if

!$acc exit data delete(sbuf11,sbuf12,rbuf11,rbuf12)
```

FORTRAN PARALLEL PROGRAMMING HINTS

Choosing a CUDA Device

OpenACC can be used to select a GPU device when running on multi-gpu node.

The exact formula may vary according to application needs and node layout.

Some job launchers make this unnecessary.

With OpenACC Set Directive

```
call MPI_Comm_rank (MPI_COMM_WORLD,myrank,ierr)
!$acc set device_num(mod(myrank,gpus_per_node))
```

GTC SPRING 2022 SESSIONS TO REWATCH

For more information on these topics

- No More Porting: Coding for GPUs with Standard C++, Fortran, and Python [S41496]
- A Deep Dive into the Latest HPC Software [S41494]
- C++ Standard Parallelism [S41960]
- Future of Standard and CUDA C++ [S41961]
- Shifting through the Gears of GPU Programming: Understanding Performance and Portability Trade-offs [S41620]
- From Directives to DO CONCURRENT: A Case Study in Standard Parallelism [S41318]
- Evaluating Your Options for Accelerated Numerical Computing in Pure Python [S41645]
- How to Develop Performance Portable Codes using the Latest Parallel Programming Standards [S41618]

Standard Parallelism Resources

NVIDIA Developer Blogs

- Developing Accelerated Code with Standard Language Parallelism
- Accelerating Standard C++ with GPUs
- Accelerating Fortran DO CONCURRENT
- Bringing Tensor Cores to Standard Fortran
- Accelerating Python on GPUs with NVC++ and Cython

Legate and cuNumeric Resources

- https://github.com/nv-legate

Open-source codes

LULESH - https://github.com/LLNL/LULESH
STLBM - https://gitlab.com/unigehpfs/stlbm
MiniWeather - https://github.com/mrnorman/miniWeather/
POT3D - https://github.com/predsci/POT3D

C++ algorithms and execution policy reference

https://en.cppreference.com/w/cpp/algorithm

NVIDIA HPC Compilers Forum

https://forums.developer.nvidia.com/c/accelerated-computing/hpc-compilers


NVIDIA