Acceleration of a production Solar MHD code with Fortran standard parallelism: From OpenACC to `do concurrent`

Ronald M. Caplan, Miko M. Stulajter, and Jon A. Linker
Predictive Science Inc.
caplanr@predsci.com

Predictive Science Inc.

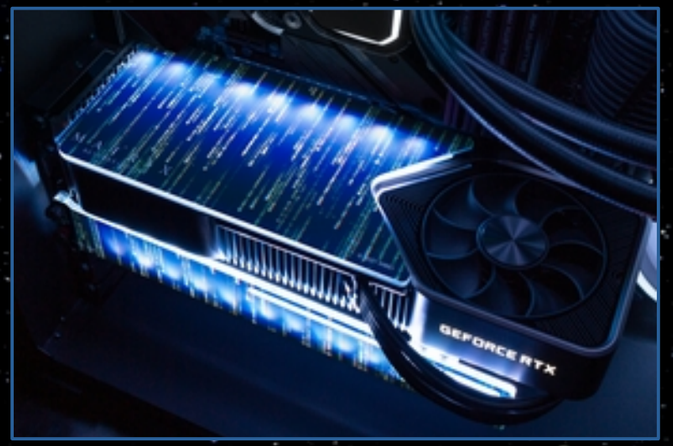
Supported by:
NSF NASA

OpenACC
WEBINAR
DIGITAL EVENT
JULY 11, 2023
1 PM EDT/10 AM PDT

arXiv:2303.03398

Outline

- Accelerated Computing
- Directives and Standard Parallelism
- Previous Implementations
- The MAS Code
- From OpenACC to Do Concurrent
- Performance
- Summary and Future Outlook



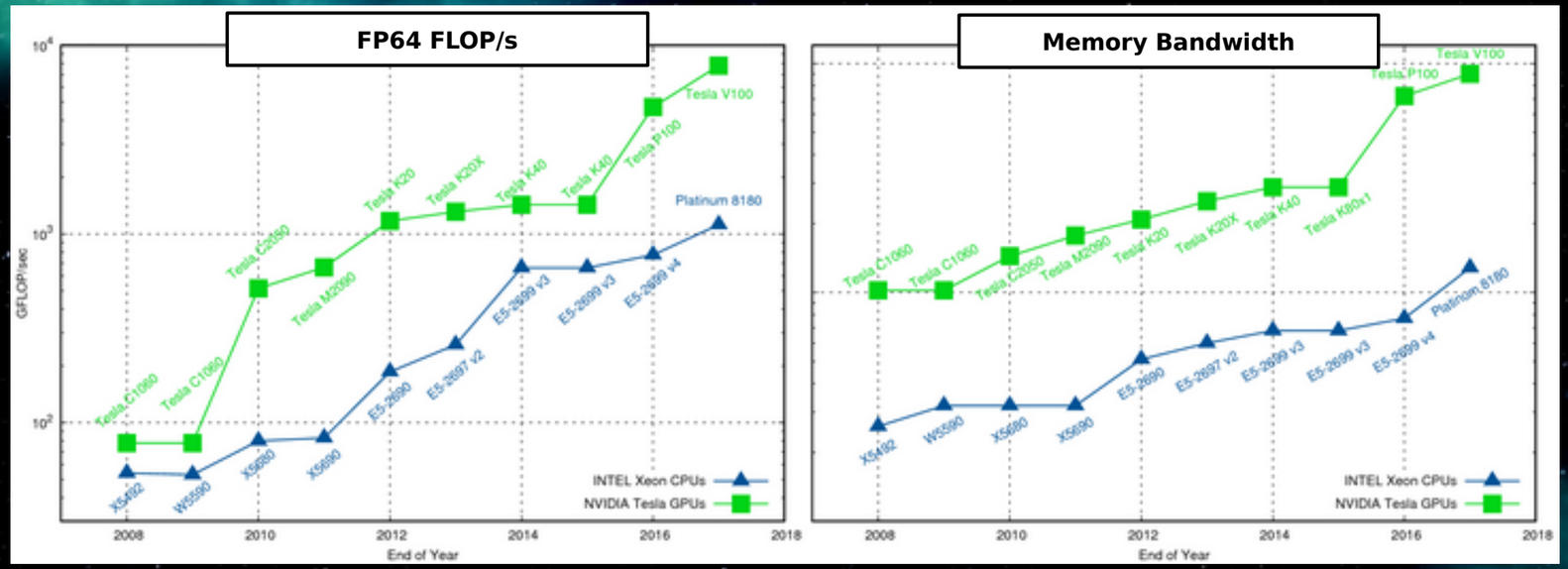
OpenACC

More Science, Less Programming

MAS

Why Accelerated Computing?

- *Overall* performance
  - FLOP/s
  - *Memory Bandwidth*
  - Specialized hardware (e.g. ML/DL tensor cores)
- *Compact* performance
  - *In-house workstations*
  - Reduce HPC real estate
- *Efficient* performance
  - Lower energy use
  - Save money



Directives

- Special comments that direct/allow the compiler to generate code that the base language does not support (e.g. parallelism, GPU-offload, data movement, etc.)
- Can produce single source code base for multiple targets (GPU, Multi-core CPU, FPGA, etc.)
- Low-risk - can ignore directives and compile as before
- Vendor-independent (NVIDIA, AMD GCN, Intel, GCC, Cray, Flang, etc.)
- Great for rapid development and accelerating legacy codes
- Two major directive APIs for accelerated computing: **OpenACC** and **OpenMP®**

OpenACC

More Science, Less Programming

```
!$acc enter data copyin(x) create(y)
!$acc parallel loop
    do i=1,n
      y(i) = a*x(i) + b
    enddo
!$acc exit data delete(x) copyout(y)
```

OpenMP

Enabling HPC since 1997

```
!$omp target enter data map(to:x) map(alloc:y)
!$omp target teams distribute parallel do
    do i=1,n
      y(i) = a*x(i) + b
    enddo
!$omp end target teams distribute parallel do
!$omp target exit data map(delete:x) map(from:y)
```

Fortran Standard Parallelism: Do Concurrent

- Introduced in ISO Standard Fortran 2008
- Indicates loop can be run with **out-of-order** execution
- Can be hint to the compiler that loop **may** be parallelizable
- No current support for reductions, atomics, device selection, conditionals, etc.
- Fortran 202X (2023) specification will add reductions

```
do i=1,N
  do j=1,M
    Computation
  enddo
enddo
```

```
do concurrent (i=1:N,j=1:M)
  Computation
enddo
```

Compiler	Version	DO CONCURRENT parallelization support
nvfortran	≥ 20.11	CPU and GPU with `-stdpar`
ifort/ifx	≥ 19.1 ≥ 23.0	CPU with `-fopenmp` CPU and GPU with `-fopenmp-target-do-concurrent`
gfortran	≥ 9	CPU with `-ftree-parallelize-loops=<#Threads>`

Directives vs. Standard Parallelism

Why use Fortran standard parallelism instead of directives?

- Longevity (ISO)
- Lower code footprint
- Less unfamiliar to domain scientists
- For accelerated computing, directives are currently more portable (may change)

These reasons also apply to codes that already use directives

Original Non-Parallelized Code

```
do k=1,np
  do j=1,nt
    do i=1,nrm1
      br(i,j,k) = (phi(i+1,j,k)-phi(i,j,k))*dr_i(i)
    enddo
  enddo
enddo
```

OpenACC Parallelized Code

```
!$acc enter data copyin(phi,dr_i)
!$acc enter data create(br)
!$acc parallel loop default(present) collapse(3) async(1)
do k=1,np
  do j=1,nt
    do i=1,nrm1
      br(i,j,k) = (phi(i+1,j,k)-phi(i,j,k))*dr_i(i)
    enddo
  enddo
enddo
!$acc wait
!$acc exit data delete(phi,dr_i,br)
```

Fortran's DO CONCURRENT

```
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
```

GPU Portability

A. Herten:
github.com/AndiH/gpu-lang-compat

DC

OpenACC
More Science, Less Programming

NVIDIA	AMD	Intel
👍	👎	👍*
👍	👍👎	👎
👍	👍	👍

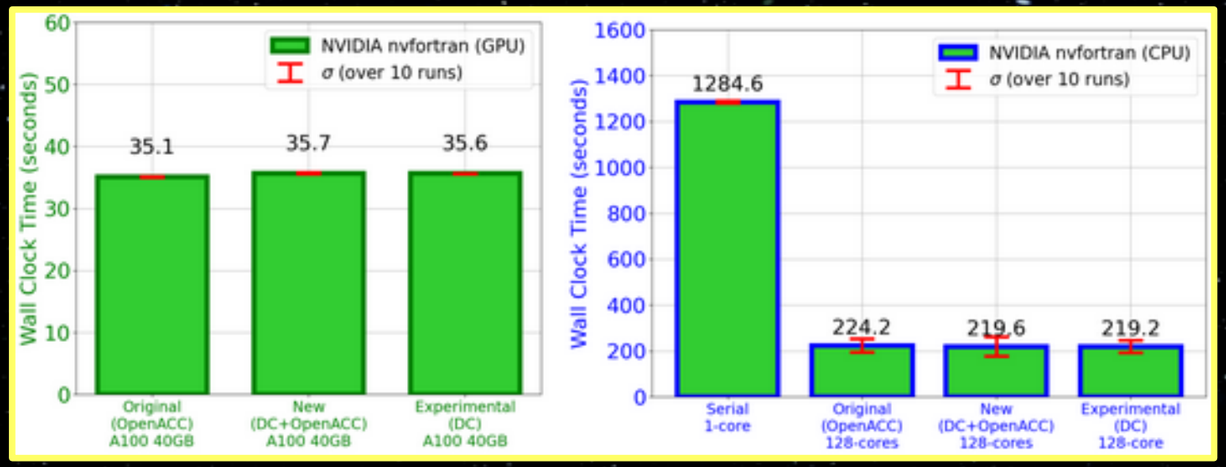
* No performance results found

Previous Implementations

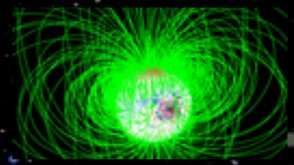
- We have previously tested replacing OpenACC with DC in a small surface diffusion tool and in our medium-sized potential field solver code
- The number of OpenACC directives were substantially or completely removed while maintaining similar performance

DIFFUSE

Stulajter, et. al. "Can Fortran's `do concurrent' Replace Directives for Accelerated Computing?"
Lecture Notes in Computer Science, 13194, 3-21. Springer, Cham. (2021)

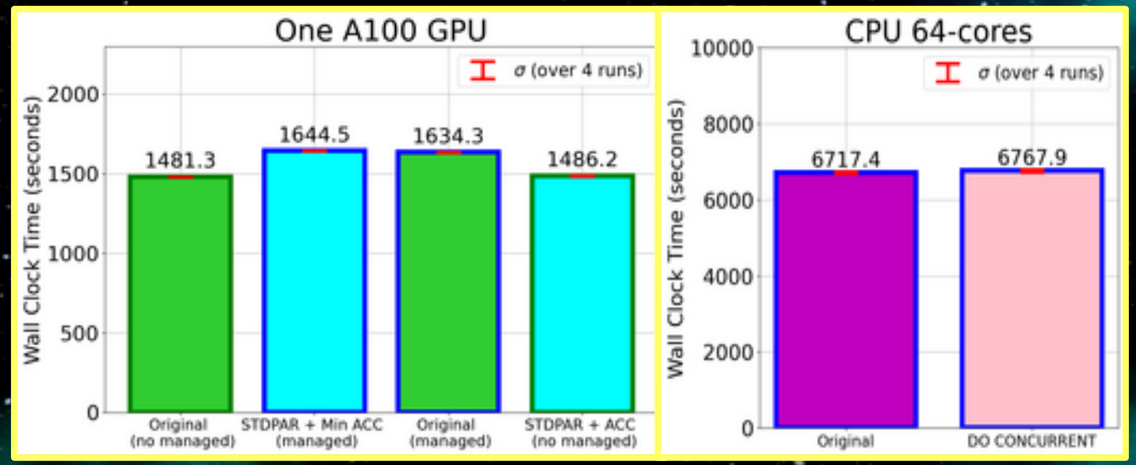


POT3D



github.com/predsci/POT3D

https://developer.nvidia.com/blog/using-fortran-standard-parallel-programming-for-gpu-acceleration



Here, we apply DC to our large-scale production code: MAS

The MAS Code

MAS — MAGNETOHYDRODYNAMIC ALGORITHM OUTSIDE A SPHERE

Purpose: General-purpose simulations of the corona and heliosphere for use with solar physics and space weather research

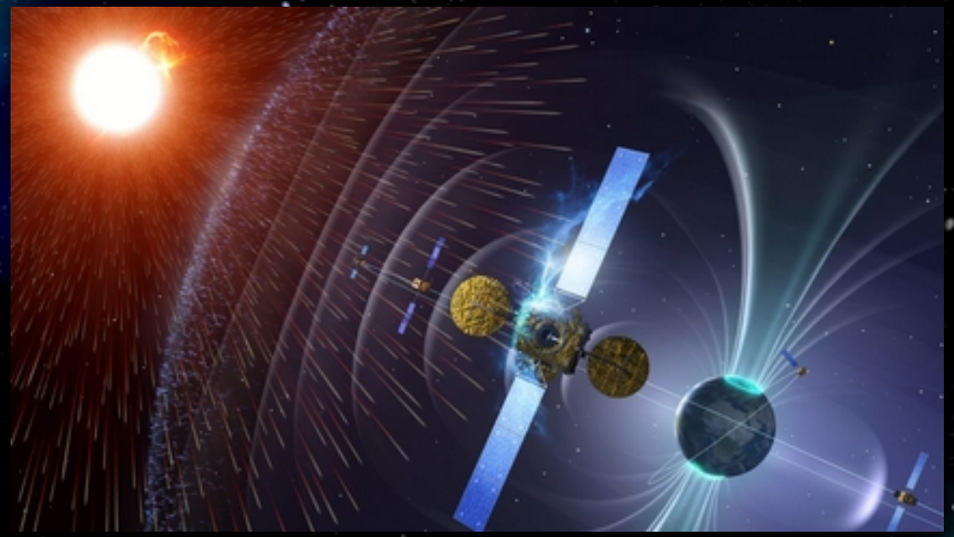
Model: Spherical 3D resistive thermodynamic MHD equations.

Algorithm: Implicit and explicit time-stepping with finite-difference stencils. Implicit steps use sparse matrix preconditioned iterative solver

Highly memory bandwidth bound!

Code: ~70,000 lines of Fortran

Parallelism: MPI + OpenACC + StdPar



predsci.com/mas

MAS OpenACC Implementation

Basic Loop

```
!$acc parallel default(present)
!$acc loop collapse(2)
   do j=1,n
     do i=1,m
       y(i,j) = a*x(i,j) + y(i,j)
     enddo
   enddo
!$acc end parallel
```

Fortran Array-syntax

```
!$acc kernels default(present)
    y(:,:) = a*x(:,:) + y(:,:)
!$acc end kernels
```

Reductions

```
!$acc parallel loop present(y)
!$acc& reduction(+:sum)
   do j=1,m
     sum = sum + y(j)
   enddo
```

```
!$acc parallel loop collapse(2)
!$acc& default(present)
   do j=1,m
     do i=1,n
!$acc atomic update
       sum(i) = sum(i) + y(i,j)
     enddo
   enddo
```

CPU↔GPU Data transfers

”y” is allocated and initialized on CPU.

```
!$acc enter data copyin (y)
!$acc update self (y)
!$acc exit data delete(y)
```

(Can now use “y” in OpenACC regions)
(CPU version of “y” updated for I/O, etc.)
(Free up GPU memory)

MAS OpenACC Implementation Cont.

Use multiple GPUs on one or more compute nodes

MPI-3

```
call MPI_Comm_split_type (MPI_COMM_WORLD,MPI_COMM_TYPE_SHARED, &
                          0,MPI_INFO_NULL,comm_shared,ierr)
call MPI_Comm_rank (comm_shared, iprocsh, ierr)
!$acc set device_num(iprocsh) !Assumes #GPUs/Node = #MPIranks/Node
```

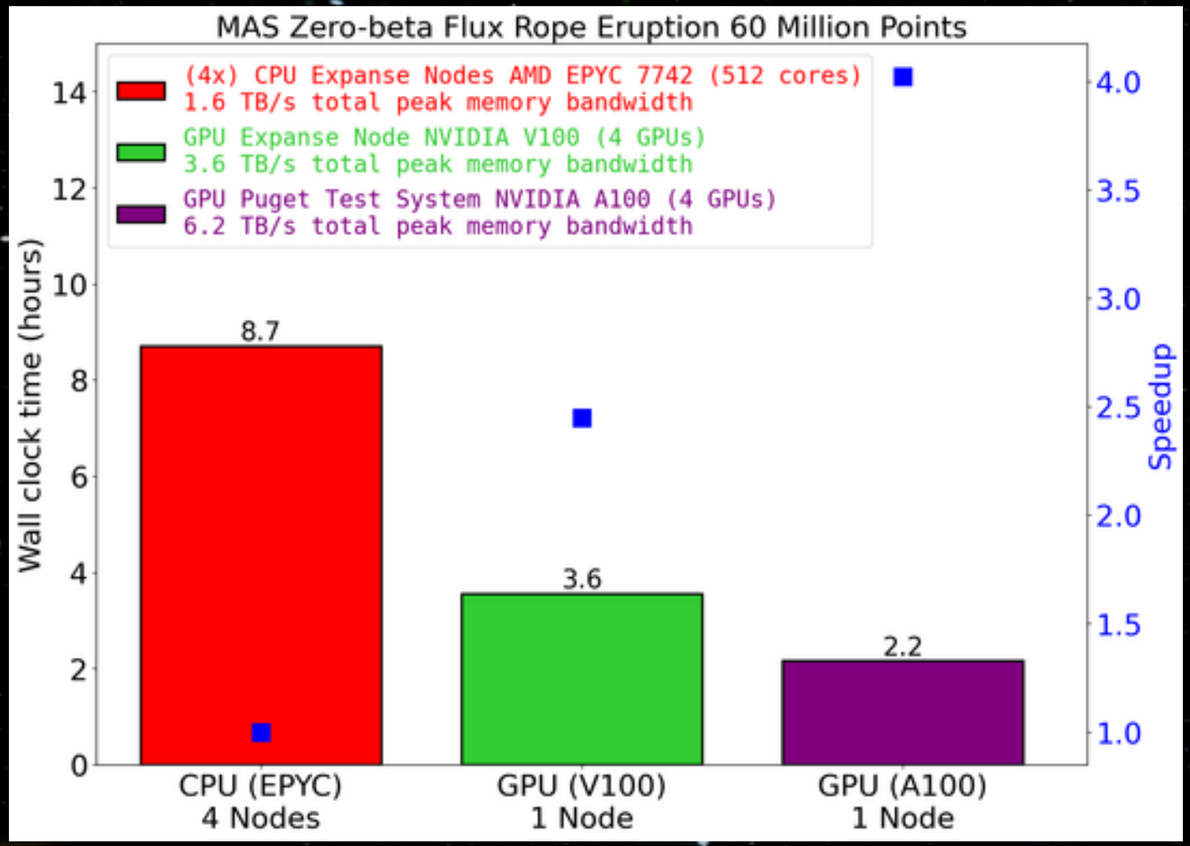
Use GPU data directly with MPI calls (CUDA-aware MPI)

```
!$acc host_data use_device(y) if_present
  call MPI_Allreduce(MPI_IN_PLACE,y,n,MPI_DOUBLE, &
                     MPI_SUM,MPI_COMM_WORLD,ierr)
!$acc end host_data
```

Code 1 [A]

Single portable source for GPU & CPU!

Total lines of code: 73,865 (~2% OpenACC)



OpenACC directives in original GPU branch of MAS (Code 1).

OpenACC directive type	# of lines
`parallel, loop`	997
data management: `enter, exit, update, host_data, declare`	320
`atomic`	34
`routine`	12
`kernels`	6
`wait`	6
`set device_num`	1
continuation lines (`!$acc&`) (spread across all directive types)	82
Total	1458

MAS OpenACC Implementation Cont: Performance Check

- A **roofline** analysis shows how well given hardware is being utilized compared to the theoretical maximum for the given code

```
do k=2,nz-1
 do j=2,ny-1
  do i=2,nx-1
    result(i,j,k) =
    -6*x(i,  j,  k  )
     + x(i-1,j  ,k  )
     + x(i+1,j  ,k  )
     + x(i  ,j-1,k  )
     + x(i  ,j+1,k  )
     + x(i  ,j  ,k-1)
     + x(i  ,j  ,k+1)
  enddo
 enddo
enddo
```

Floating Point Operations: **7 FLOP**
Data movement (loads/stores):
8*8 Bytes = **64 Bytes**
Arithmetic Intensity:
FLOP / BYTE = 0.11

MAS OpenACC to Do Concurrent

- Multiple versions based on these considerations:
- Avoid code refactoring (or not)
- Adhere to ISO 2018 Fortran Standard (or not)
  - Data affinity statements (part of specification) not used as they are not currently supported by some compilers (e.g. GCC)
- Using 202X preview & special features (or not)
  - DC Reduction clause (nvfortran only)
  - OpenACC directives within DC loops
- Less OpenACC directives vs. performance loss
  - Unified managed memory vs. manual memory management

SUMMARY OF ALL MAS CODE VERSIONS DEVELOPED AND TESTED.

Code Version	Code description and `nvfortran` GPU compiler flags	Total Lines	`$acc` Lines
0: CPU	Original CPU-only version	69874	∅
1: A	Original OpenACC implementation `-acc=gpu -gpu=cc80`	73865	1458
2: AD	OpenACC for DC-incompatible loops and data management, DC for remaining loops `-acc=gpu -stdpar=gpu -gpu=cc80,nomanaged`	71661	540
3: ADU	OpenACC for DC-incompatible loops, DC for remaining loops, Unified memory `-acc=gpu -stdpar=gpu -gpu=cc80,managed`	71269	162
4: AD2XU	OpenACC for for functionality , DC2X for remaining loops, Unified memory `-acc=gpu -stdpar=gpu -gpu=cc80,managed`	70868	55
5: D2XU	DC2X for all loops, some code modifications, Unified memory `-stdpar=gpu -gpu=cc80 -Minline=reshape,name:s2c, boost,interp,c2s,sv2cv`	68994	∅
6: D2XAd	DC2X for all loops, some code modifications, OpenACC for data management `-acc=gpu -stdpar=gpu -gpu=cc80,nomanaged -Minline=reshape,name:s2c, boost,interp,c2s,sv2cv`	71623	277

Code 2 [AD]

- Start with Fortran 2018 specification compliance; no unified managed memory; no refactoring
- No DC reduction support in current standard, so array reduction code would need refactoring
- Functions/routines inside loops:
  - DC requires they are “pure”
  - Even so, nvfortran does not currently support them, so need OpenACC routine directives
- Removing **`kernels`** used for array syntax and intrinsics (e.g. **`MINVAL`**) would need refactoring

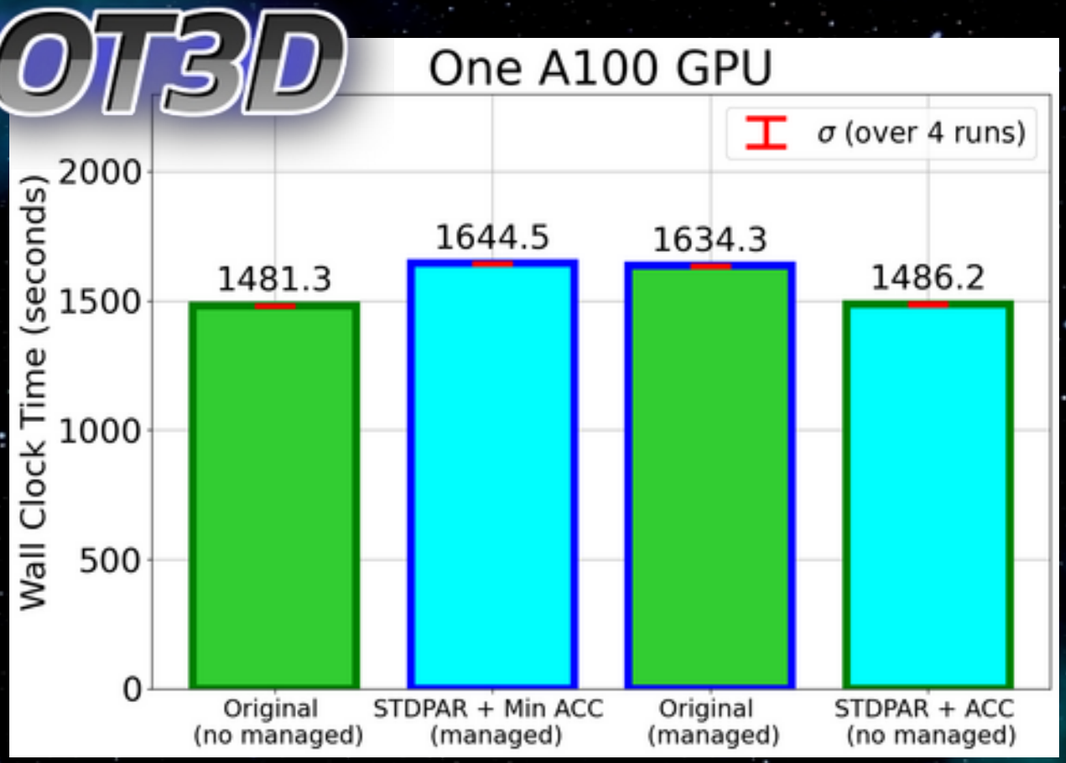
```
!$acc parallel default(present)
!$acc loop collapse(2)
do j=1,n2
  do i=1,n1
!$acc atomic update
    sum0(i)=sum0(i)+array(i,j)*...
  enddo
enddo
!$acc end parallel
```

```
      module c2s_interface
!$acc routine(c2s) seq
      interface
        pure subroutine c2s (x,y,z,r,t,p)
          use number_types
          implicit none
          real(r_typ), intent(in) :: x,y,z
          real(r_typ), intent(out) :: r,t,p
        end subroutine c2s
      end interface
      end module
```

```
!$acc kernels default(present)
      min_field_val_local=MINVAL(field,mask)
!$acc end kernels
```

Code 2 [AD] Cont.

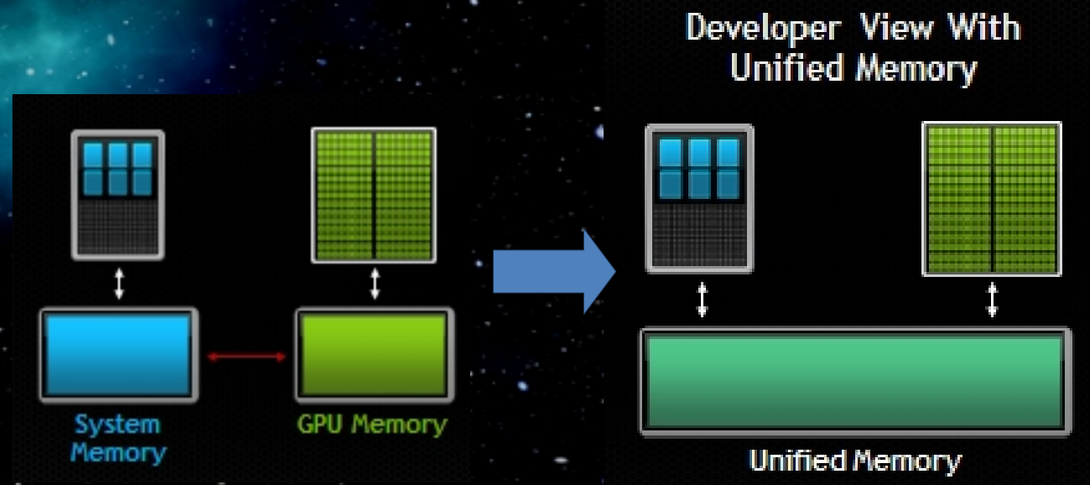
- Previous DC results show using unified managed memory (UM) can reduce performance, so we leave OpenACC data movement directives
- Kernel fusion (OpenACC **`parallel`** regions) and asynchronous computation (OpenACC **`async`** clause) are not available in DC
- Code 2 [**AD**]:
  OpenACC for DC-incompatible loops and data management
  DC for all remaining loops



Code Version	Code description and `nvfortran` GPU compiler flags	Total Lines	`$acc` Lines
1: A	Original OpenACC implementation `-acc=gpu -gpu=cc80`	73865	1458
2: AD	OpenACC for DC-incompatible loops and data management, DC for remaining loops `-acc=gpu -stdpar=gpu -gpu=cc80,nomanaged`	71661	540

Code 3 [ADU]

- Here, activate NVIDIA unified managed memory (UM) allowing removal of OpenACC data directives
- Can't remove *all* OpenACC data directives:
  - **`declare`** in function calls
  - Derived-type structures in OpenACC loops when using **`default(present)`**
- Code 3 [**ADU**]:
  OpenACC for DC-incompatible loops,
  DC for all remaining loops,
  Unified managed memory




```
      real(r_typ) :: t_cutoff1=500000._r_typ
!$acc declare create(t_cutoff1)
```

```
!$acc update device(t_cutoff1)
```

```
      allocate (a%r(nrm1,nt,np))
      allocate (a%t(nr,ntm1,np))
      allocate (a%p(nr,nt,npm1))
c
!$acc enter data create(a)
```

Code Version	Code description and `nvfortran` GPU compiler flags	Total Lines	`$acc` Lines
2: AD	OpenACC for DC-incompatible loops and data management, DC for remaining loops `-acc=gpu -stdpar=gpu`	71661	540
3: ADU	OpenACC for DC-incompatible loops, DC for remaining loops, Unified memory `-acc=gpu -stdpar=gpu -gpu=cc80,managed`	71269	162

Code 4 [AD2XU]

- Here, we use the Fortran 202X preview implementation in nvfortran
- DC reduction clause
- No array reductions, but can use OpenACC atomics in DC loop
- Able to remove most data clauses as all loops using derived types are now DC
- Some OpenACC directives remain:
  - **`atomic, declare, update, set device_num, routine, kernels`**
- Code 4 [**AD2XU**]:
  OpenACC for functionality,
  DC 202X for all loops,
  Unified managed memory

```
do concurrent (k=2:npm1) reduce(+:sum0)
  sum0=sum0+a(2,k)*dph(k)*pl_1*two
enddo
```

```
do concurrent (j=1:n2,i=1:n1)
!$acc atomic update
  sum0(i)=sum0(i)+array(i,j)*...
enddo
```

Code Version	Code description and nvfortran GPU compiler flags	Total Lines	$acc Lines
3: ADU	OpenACC for DC-incompatible loops, DC for remaining loops, Unified memory `-acc=gpu -stdpar=gpu -gpu=cc80,managed`	71269	162
4: AD2XU	OpenACC for for functionality , DC2X for remaining loops, Unified memory `-acc=gpu -stdpar=gpu -gpu=cc80,managed`	70868	55

Code 5 [D2XU]

- Here, we allow minor code refactoring
- Replaced **`kernels`** with expanded DC loops
- Array reduction loops modified to avoid **`atomic`**
- Removed **`set_device_num`** by using BASH launching script (OpenMPI-based)
  `mpiexec -np <#> ./launch.sh ./mas ...`
- Use nvfortran flags to in-line routines, explicitly listing routines that can't be automatically in-lined
- Can eliminate duplicate CPU-only routines
- Code 5 [**D2XU**]:
  DC 202**X** for all loops,
  Some code refactoring,
  Unified managed memory

```
do concurrent (i=1:n1)
  tmp=0.
  do concurrent (j=1:n2) reduce(+:tmp)
    tmp=tmp+array(i,j)*...
  enddo
  sum0(i)=tmp
enddo
```

launch.sh

```
#!/bin/bash
# Assume 1 GPU per MPI local rank
# Set device for this MPI rank:
export CUDA_VISIBLE_DEVICES="
    $OMPI_COMM_WORLD_LOCAL_RANK"
# Execute code:
exec $*
```

Code Version	Code description and nvfortran GPU compiler flags	Total Lines	$acc Lines
4: AD2XU	OpenACC for for functionality , DC2X for remaining loops, Unified memory `-acc=gpu -stdpar=gpu -gpu=cc80,managed`	70868	55
5: D2XU	DC2X for all loops, some code modifications, Unified memory `-stdpar=gpu -gpu=cc80 -Minline=reshape,name:s2c, boost,interp,c2s,sv2cv`	68994	∅

Code 6 [D2XAd]

- Spoiler alert: Unified Managed Memory currently results in a non-trivial performance hit, especially across many MPI ranks
- Here, we take **Code 5**, but add back in (in minimal form) OpenACC data directives
- Result is minimal number of directives, while retaining original performance
- Code 6 [**D2XAd**]:
  DC 202**X** for all loops,
  Some code refactoring,
  Open**A**CC for **d**ata management

MIN(

)

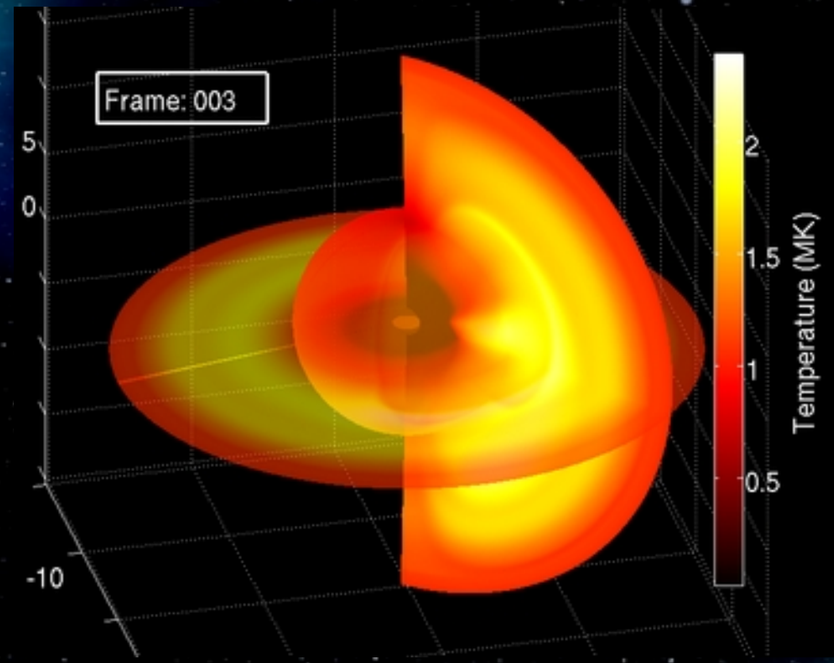
Code Version	Code description and `nvfortran` GPU compiler flags	Total Lines	`$acc` Lines

5: D2XU	DC2X for all loops,
	some code modifications,
	Unified memory
	`-stdpar=gpu -gpu=cc80`
	`-Minline=reshape,name:s2c,`
	`boost,interp,c2s,sv2cv`
		68994	∅

6: D2XAd	DC2X for all loops,
	some code modifications,
	OpenACC for data management
	`-acc=gpu -stdpar=gpu`
	`-gpu=cc80,nomanaged`
	`-Minline=reshape,name:s2c,`
	`boost,interp,c2s,sv2cv`
		71623	277

MAS Test Case and Computational Environment

- Medium-sized production thermodynamic coronal relaxation
- 36 million cells, 24 minutes physical time – Can fit on single 40GB GPU
- DC has no effect on performance for CPU-only MPI runs:



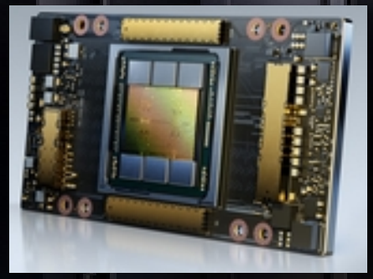
EXPANSE
COMPUTING WITHOUT BOUNDARIES
SDSC

# CPUs x Model	**(2x)** EPYC 7742
# Total Cores	128 *(we use 64)*
Peak FLOP/s	7.0 TFLOP/s
Memory	256 GB
Total Memory Bandwidth	381.4 GB/s

# Nodes	Code 1 (A)	Code 2 (AD)
1	725.54	725.53
8	79.58	79.64

ACCESS

DELTA
NCSA



A100 40GB SXM4

NCSA Delta 8xGPU Node

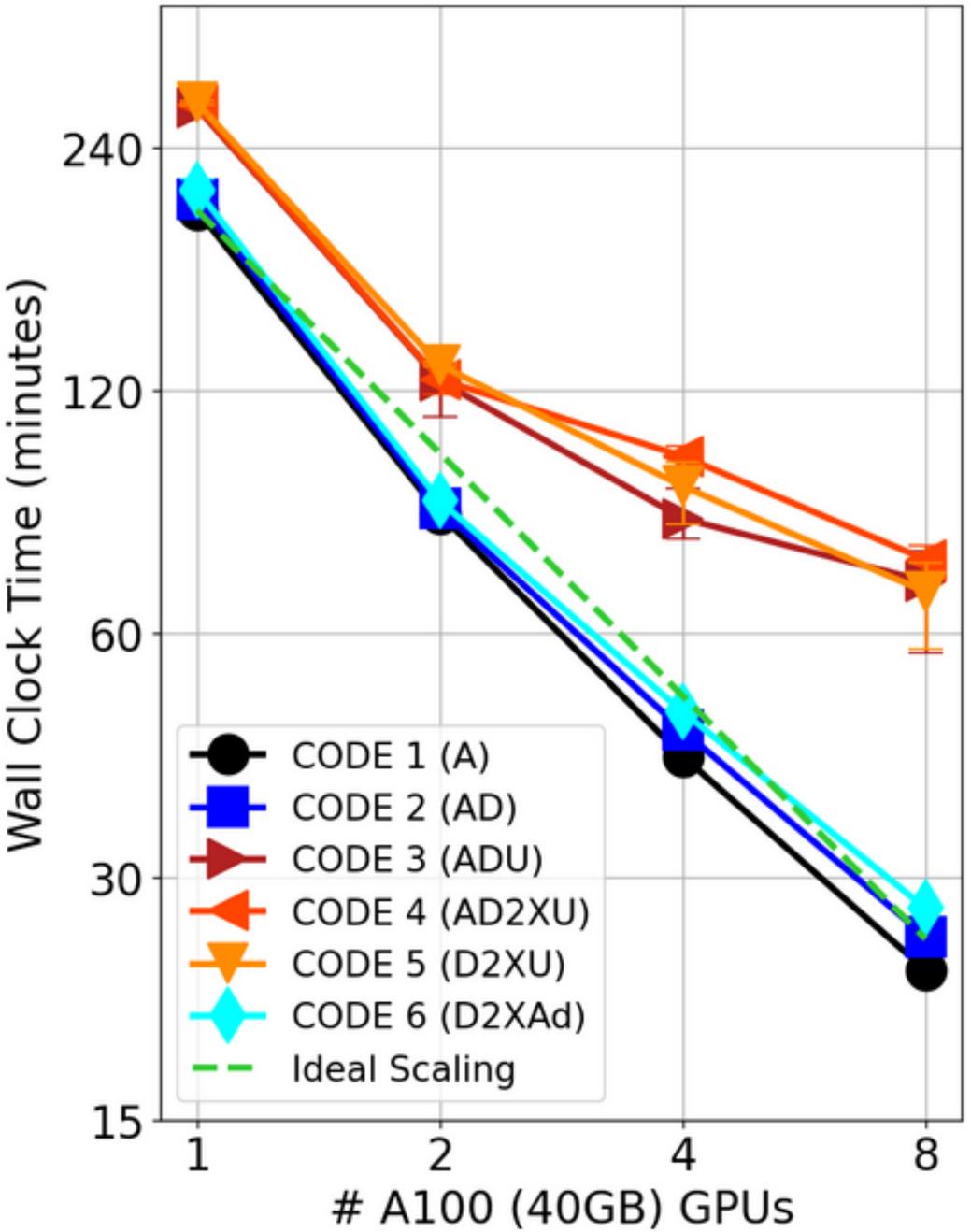
# CPUs x Model	**(2x)** EPYC 7742
# GPUs x Model	**8x** A100-40GB SXM4
Peak DP FLOP/s / GPU	9.8 TFLOP/s
Memory / GPU	40 GB
Memory Bandwidth/GPU	1555 GB/s

Code Summary and Performance

- Codes 1, 2, and 6 all show good scaling and similar performance
- Code 5 uses ZERO directives
- Code 6 uses 5.2x fewer directives than the original code, while Code 2 (within current spec) uses 2.7x fewer – with both exhibiting similar performance!

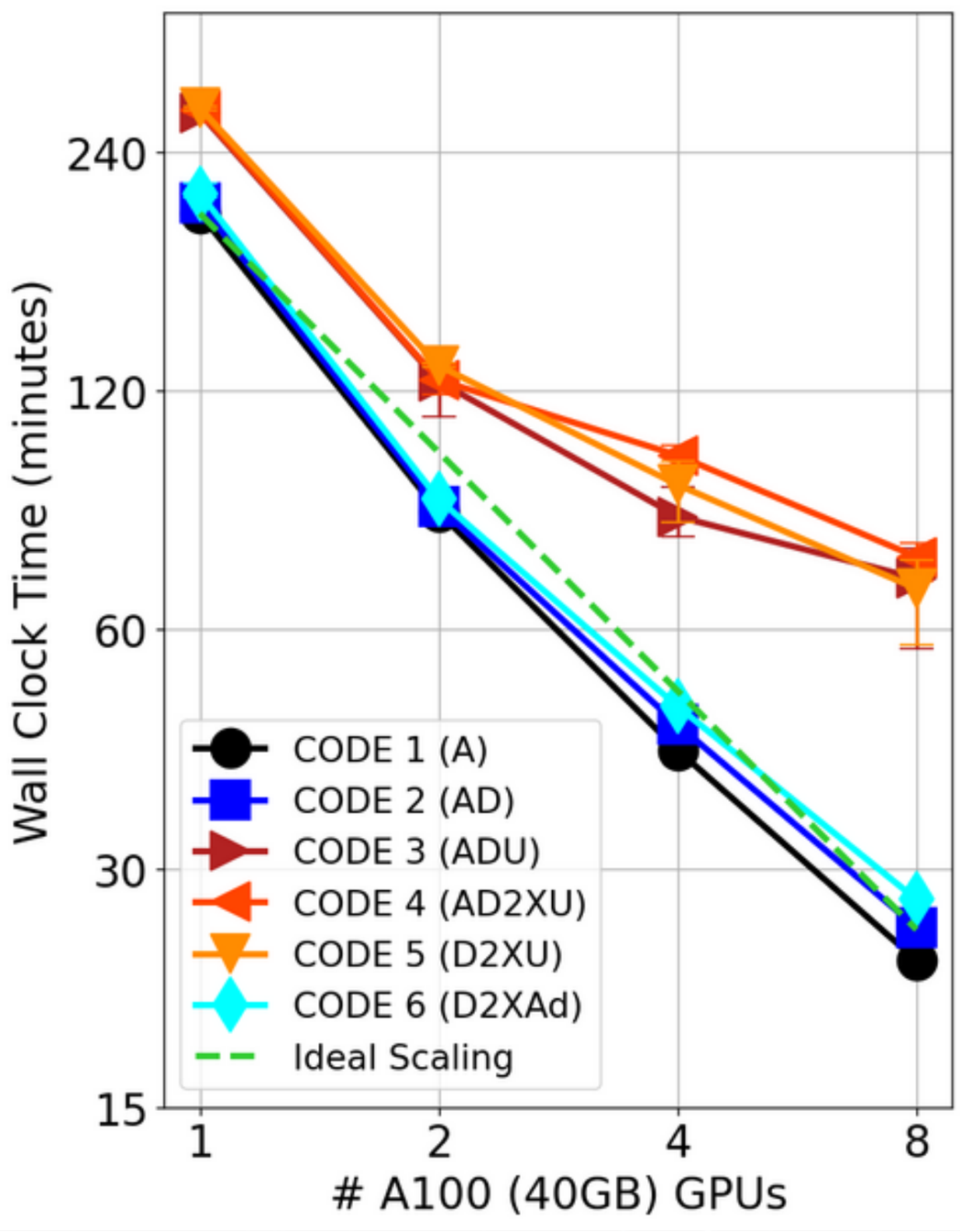
SUMMARY OF ALL MAS CODE VERSIONS DEVELOPED AND TESTED.

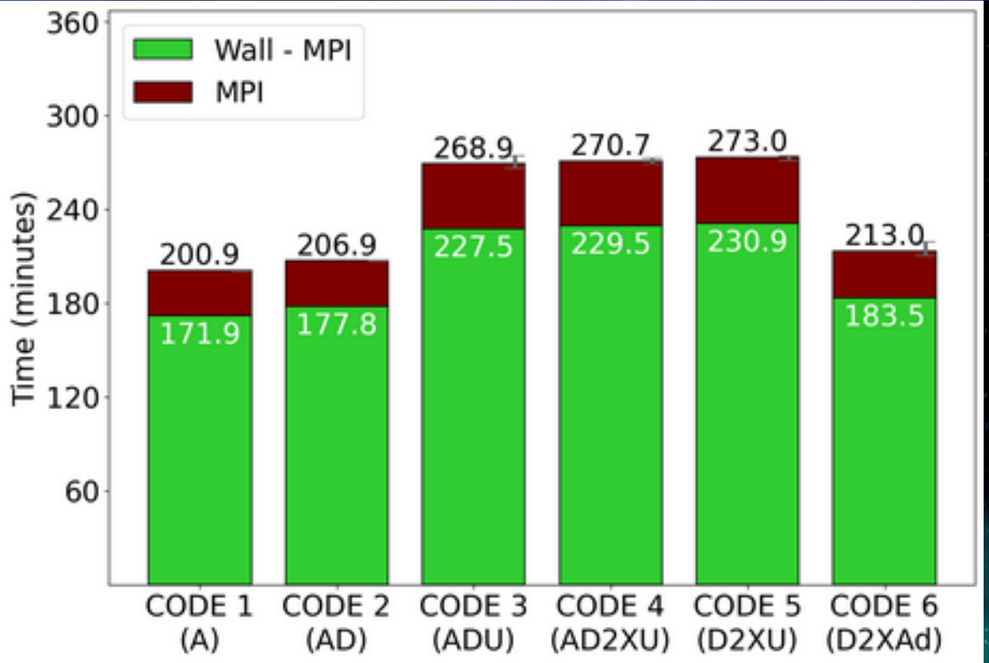
Code Version	Code description and `nvfortran` GPU compiler flags	Total Lines	`$acc` Lines
0: CPU	Original CPU-only version	69874	∅
1: A	Original OpenACC implementation `-acc=gpu -gpu=cc80`	73865	1458
2: AD	OpenACC for DC-incompatible loops and data management, DC for remaining loops `-acc=gpu -stdpar=gpu -gpu=cc80,nomanaged`	71661	540
3: ADU	OpenACC for DC-incompatible loops, DC for remaining loops, Unified memory `-acc=gpu -stdpar=gpu -gpu=cc80,managed`	71269	162
4: AD2XU	OpenACC for for functionality , DC2X for remaining loops, Unified memory `-acc=gpu -stdpar=gpu -gpu=cc80,managed`	70868	55
5: D2XU	DC2X for all loops, some code modifications, Unified memory `-stdpar=gpu -gpu=cc80 -Minline=reshape,name:s2c, boost,interp,c2s,sv2cv`	68994	∅
6: D2XAd	DC2X for all loops, some code modifications, OpenACC for data management `-acc=gpu -stdpar=gpu -gpu=cc80,nomanaged -Minline=reshape,name:s2c, boost,interp,c2s,sv2cv`	71623	277

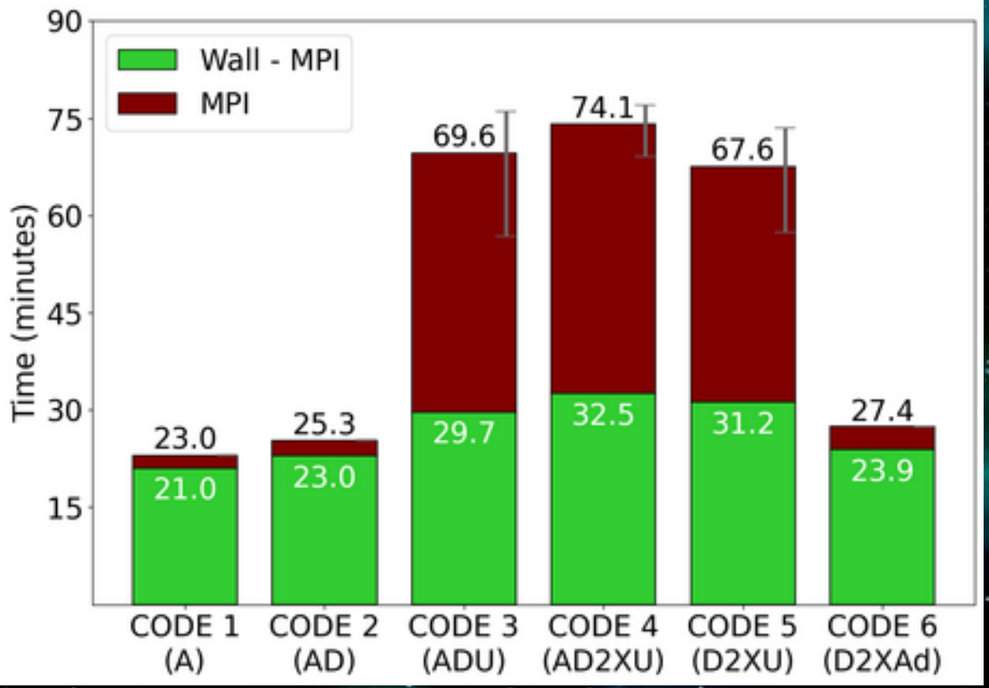


Performance Cont.

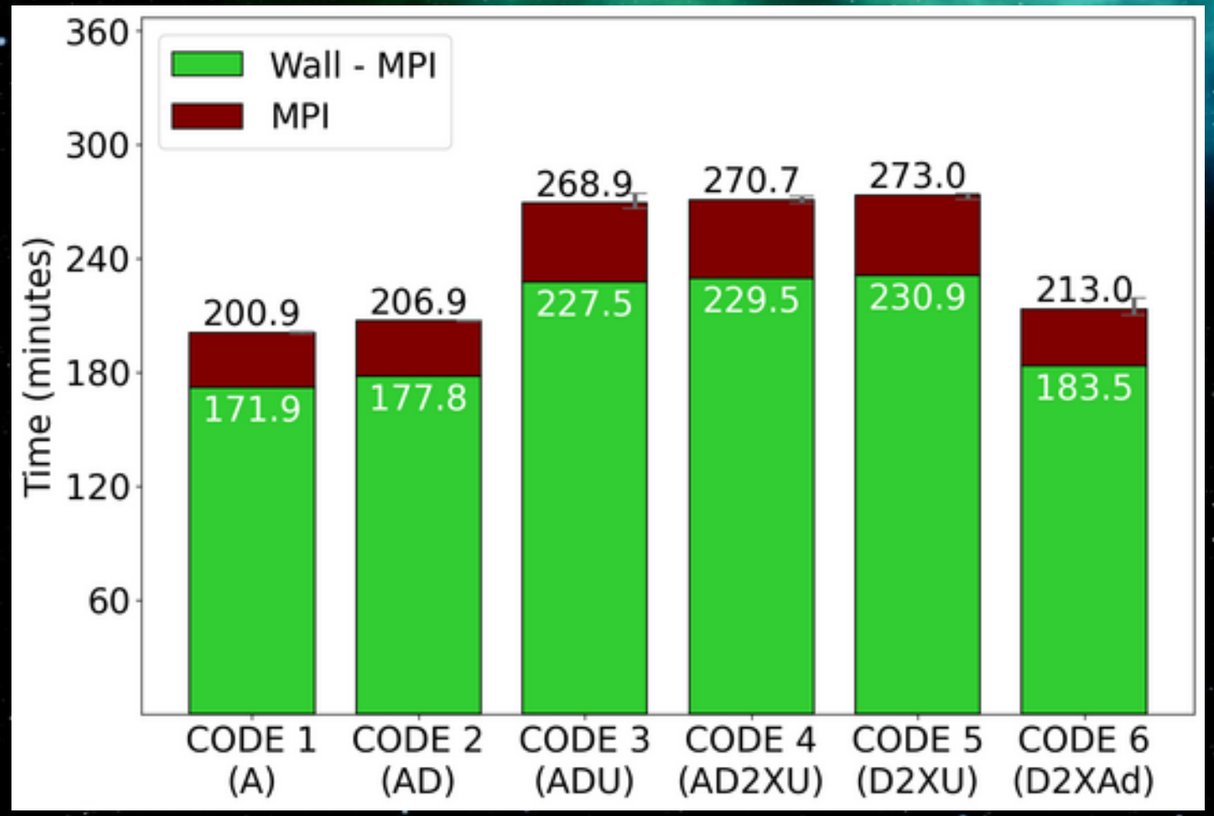
- Codes 3, 4, and 5 show poor performance and poor scaling – all use unified managed memory
- They have ~25% lower performance with 1 GPU
- With multiple GPUs over MPI (even on the same node) the performance is over 2x slower
- NVIDIA compiler developers are aware of this issue and working on a fix

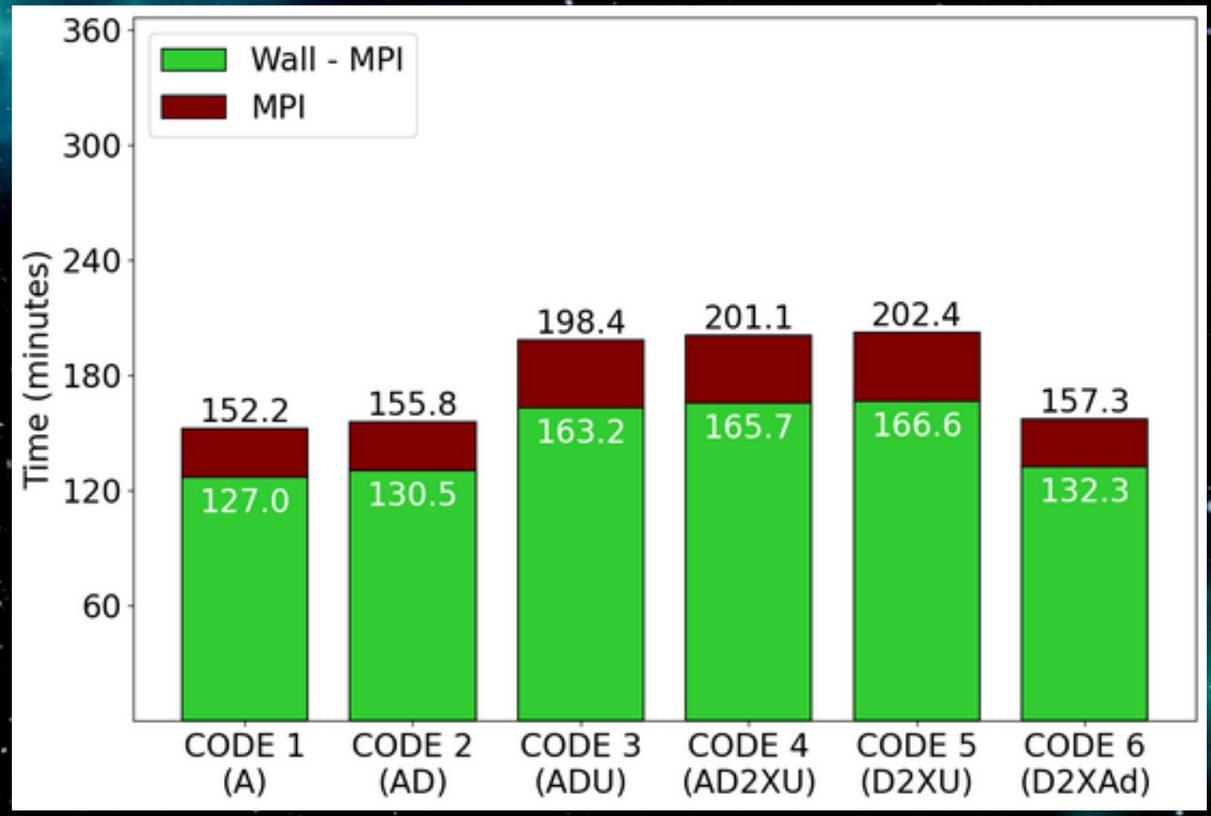




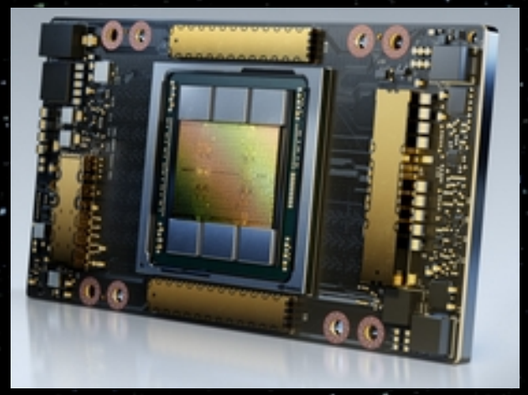


Performance Cont.

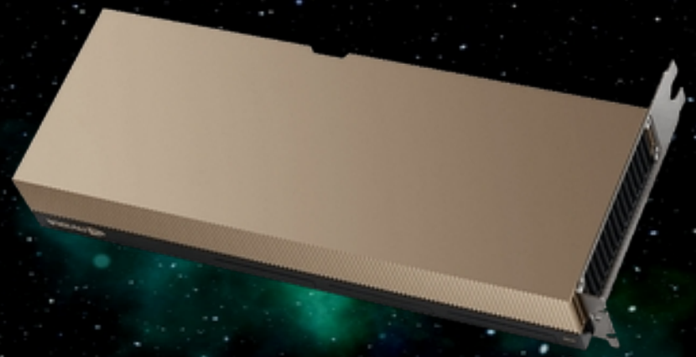




A100 40GB SXM4
Memory Bandwidth: 1.56 TB/s



H100 80GB PCIe
Memory Bandwidth: 2.00 TB/s



Georgia Tech

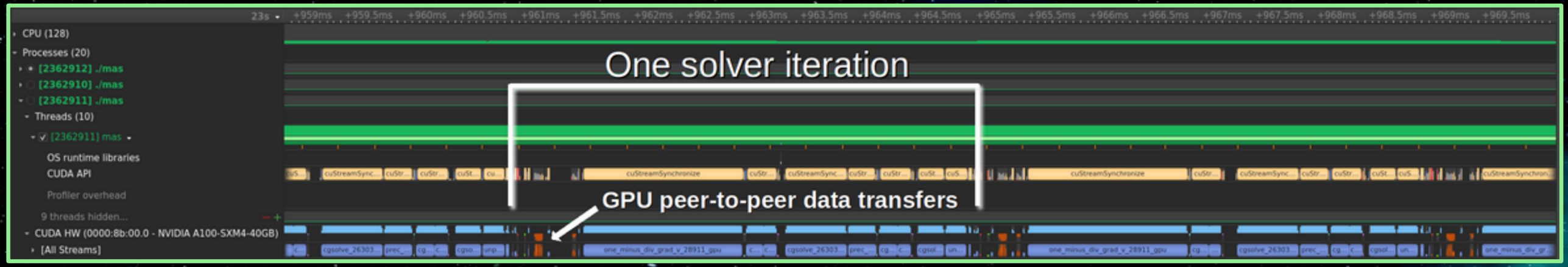
Quorra

Performance Cont.

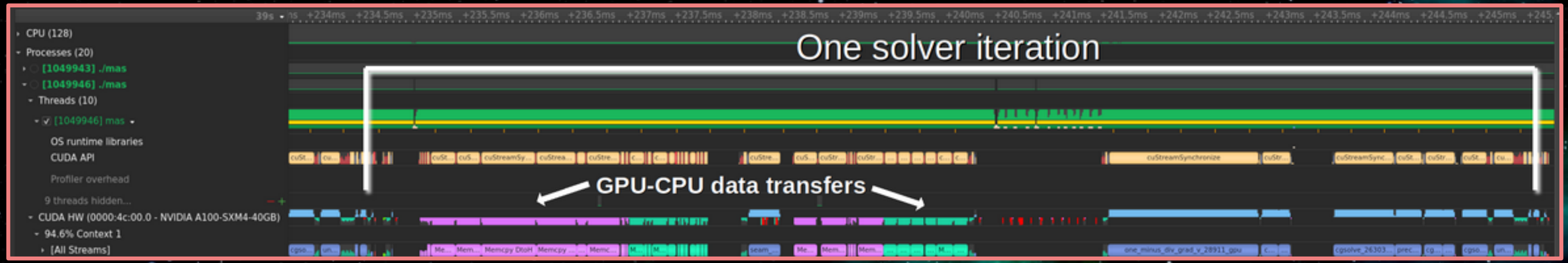
- Unified Managed Memory is not utilizing CUDA-aware MPI, resulting in a lot of CPU-GPU data transfers:

```
nsys profile --stats=true mpiexec -np 8 ./mas mas
```

NVIDIA NSIGHT™ SYSTEMS



Code 2 [AD]



Code 3 [ADU]

MAS Production Code Implementation

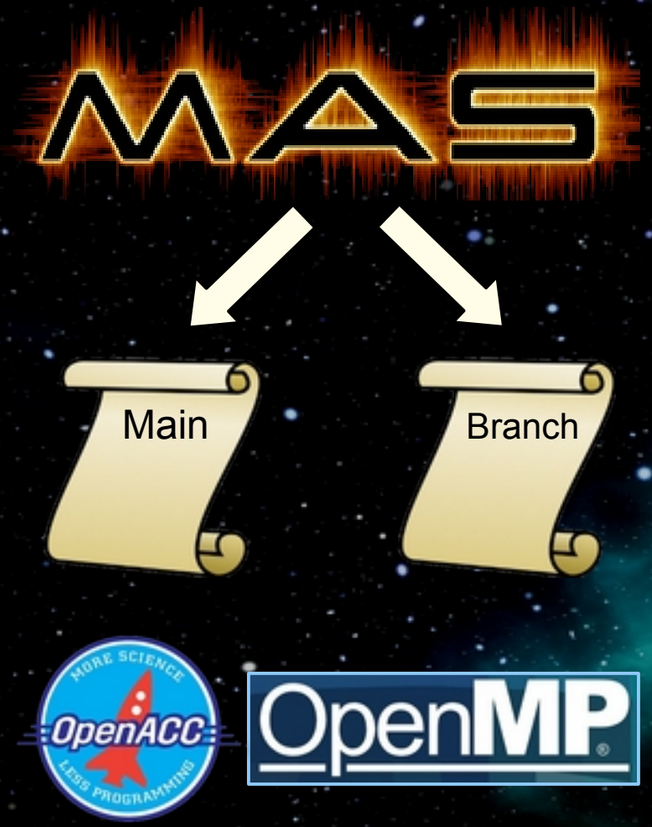
- Which version did we pick for our production code?
- Two code versions (both easy to maintain):
  - Main code has minimal OpenACC directives so it can be compiled for GPUs using unified managed memory

```
Lines:   Total: 68,972    OpenACC: 132 (0.2%)
```

  - "ACC" branch of code has OpenACC data movement added in; used in production releases

```
Lines:   Total: 70,658    OpenACC: 593 (0.8%)
```

- We have added OpenMP to OpenACC loops in ACC branch to allow hybrid-CPU mode
- May convert OpenMP to OpenMP target off-load for future use with Intel GPUs



Summary and Future Outlook

Do Concurrent in Fortran for accelerated computing

- **ISO standard** (~0 chance of deprecation)
- More compact/simple code
- Computation loops exhibit similar performance to directives (when combined with directive manual data management)
- Can add hybrid MPI+thread CPU parallelism
- Currently supported by NVIDIA and Intel for their GPUs

- Lack of performance features (no async, no kernel fusion, no memory management)
- Non-trivial performance drop for CUDA-aware MPI with NVIDIA (fix in progress)
- No current GPU DC support in GCC, AMD, Cray, Flang, etc. (some in progress...)

- Still need OpenACC/OpenMP target for some time
- Future Fortran language additions & compiler support for multi-vendor accelerators can lead to

one code to run on them all...