Fortran Standard Language Parallelism on GPUs

Ronald M. Caplan
Predictive Science Inc.

AMD Booth #3028
Wed. Nov. 18th 2026



SC26
Chicago, IL | hpc unites.



Predictive Science Inc.

Supported by:





Fortran?



fortran-lang.org

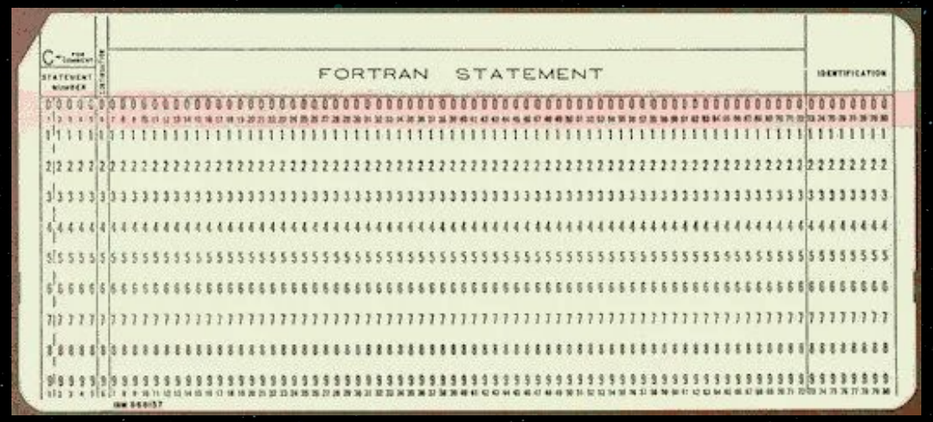
What is the status of Fortran?

Fortran is mature and under active development. The latest revision of the language is Fortran 2023. There are over a dozen open source and proprietary Fortran compilers. Further, open source projects like the Standard Library and the Fortran Package Manager are under active development.

TIOBE
the software quality company

Sep 2026	Sep 2025	Change	Programming Language
1	1		Python
2	3	↑	C
3	2	↓	C++
4	4		Java
5	5		C#
6	6		JavaScript
7	7		Visual Basic
8	11	↑	SQL
9	13	⇈	R
10	18	⇈	Rust
11	12	↑	Fortran

FORTRAN to **Modern Fortran**

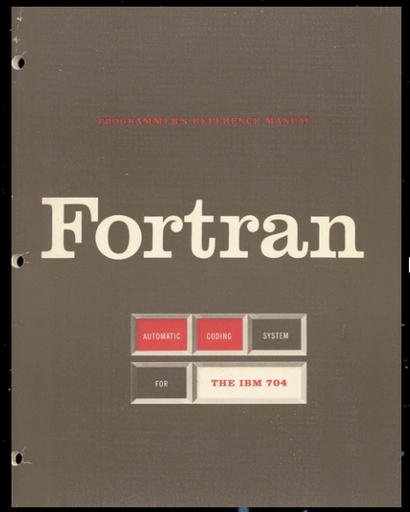


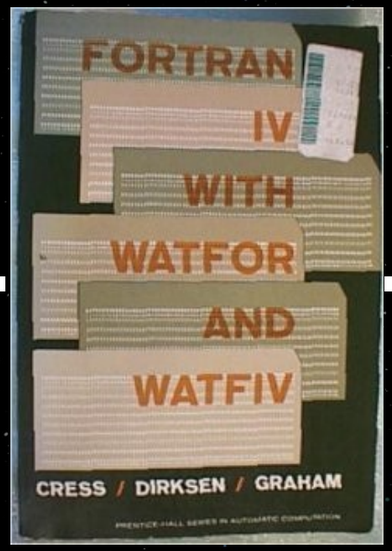


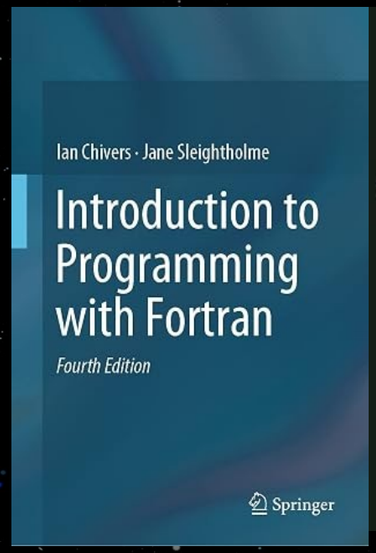


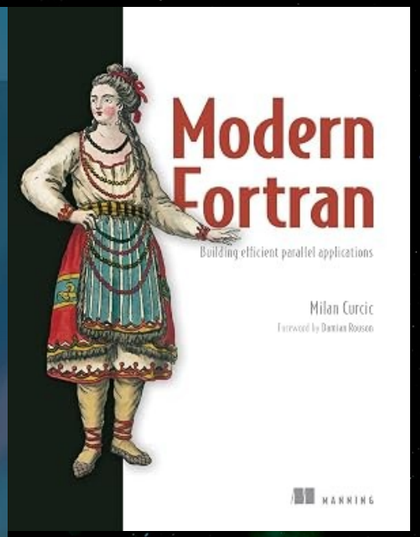


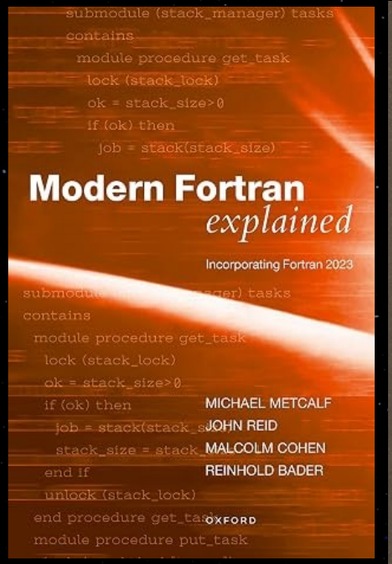
F66 → F77 → F90 → F95 → F2003 → F2008 → F2018 → F2023 → F202Y

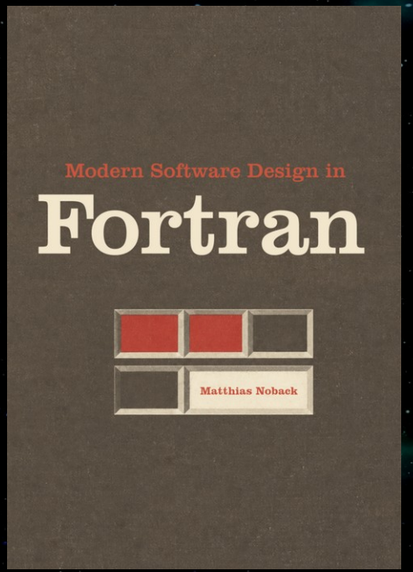






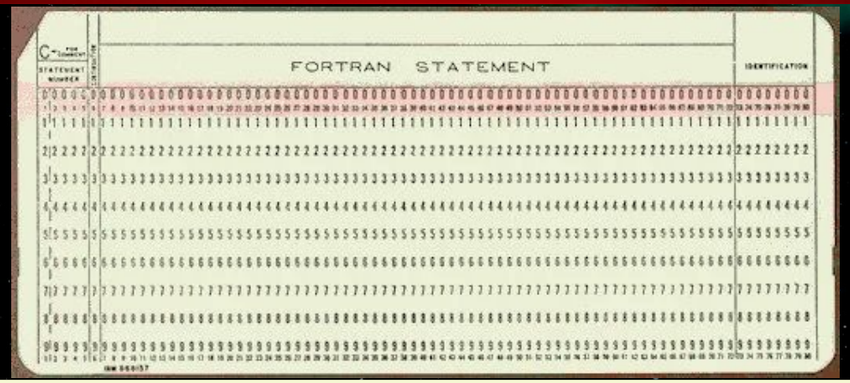






1956 1970 2018 2020 2024 2026

FORTRAN vs Modern Fortran




```
      DOUBLE PRECISION X, Y, Z, A, S
      DIMENSION X(100), Y(100), Z(100)
      N=100
      A=2.0D0
C     Fill x and y somehow…
      DO 10 I=1,N
      Z(I)=A*X(I)+Y(I)
   10 CONTINUE
      S=0.0D0
      DO 11 I=1,N
      S=S+Z(I)
   11 CONTINUE
      IF (S.GT.1000.0) GOTO 30
      GOTO 40
   30 STOP
   40 WRITE (6,20) S
   20 FORMAT (1X, D24.16)
      STOP
      END
```

```
program daxpy_sum

  use iso_fortran_env, only: real64
  implicit none
  real(real64) :: a, s
  real(real64),allocatable :: x(:),y(:),z(:)
  integer :: i,n

  n = 100
  a = 2.0_real64
  allocate(x(n), y(n), z(n))
! Fill x and y somehow…

  do concurrent (i = 1:n)
    z(i) = a * x(i) + y(i)
  end do

  s = 0.0
  do concurrent (i = 1:n) reduce(+:s)
    s = s + z(i)
  end do

  if (s > 1000.0_real64) stop
  print*, s

end program daxpy_sum
```

Fortran vs. Python

```fortran
program axpy_sum
  use iso_fortran_env, only: real64
  implicit none
  integer :: n
  real(real64) :: a, s
  real(real64),allocatable :: x(:),y(:),z(:)

  n = 100
  a = 2.0_real64
  allocate(x(n), y(n), z(n))
  call random_number(x)
  call random_number(y)

  z = a * x + y
  s = sum(z)

  print *, s

end program axpy_sum
```

```python
#!/usr/bin/env python3
import numpy as np
def main() -> None:

  n = 100
  a = np.float64(2.0)

  x = np.random.random(n).astype(np.float64)
  y = np.random.random(n).astype(np.float64)

  z = a * x + y
  s = np.sum(z)

  print(s)

if __name__ == "__main__":
    main()
```

TRANSLATE · INSPECT · EXPERIMENT

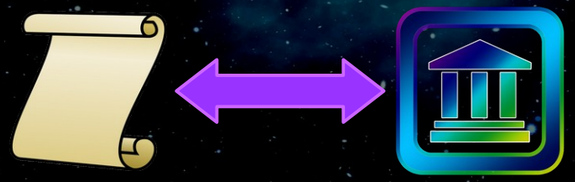
Python → Fortran

A browser playground for p2f. Explore the generated source, one example at a time.

beliavsky.github.io/
python-to-fortran-playground

Fortran on GPUs? - How?

- Libraries (e.g. rocBLAS)



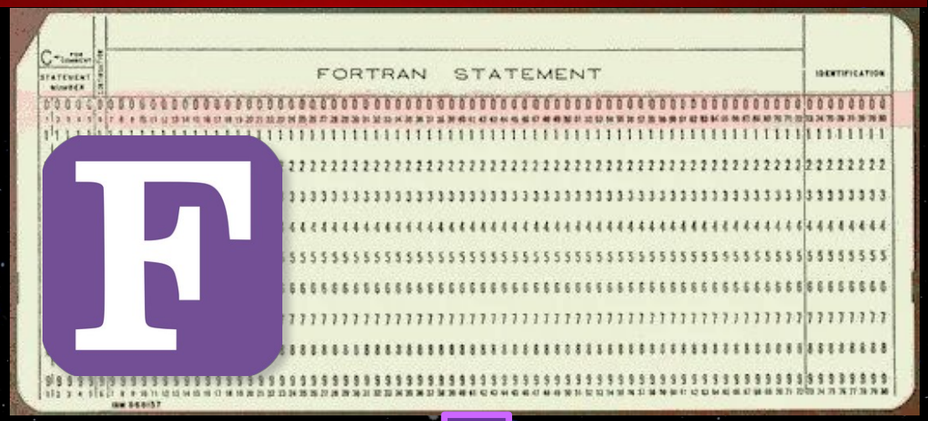
- ISO_C_BINDING:
  - Link to C/C++ code where needed and use C++ APIs (e.g. HIP/Rocm, OpenCL, SYCL, KOKKOS, etc.) – not really a "Fortran" solution...
- External Fortran APIs:
  - CUDA-Fortran (NVIDIA only)
  - OpenCL/CLFORTRAN

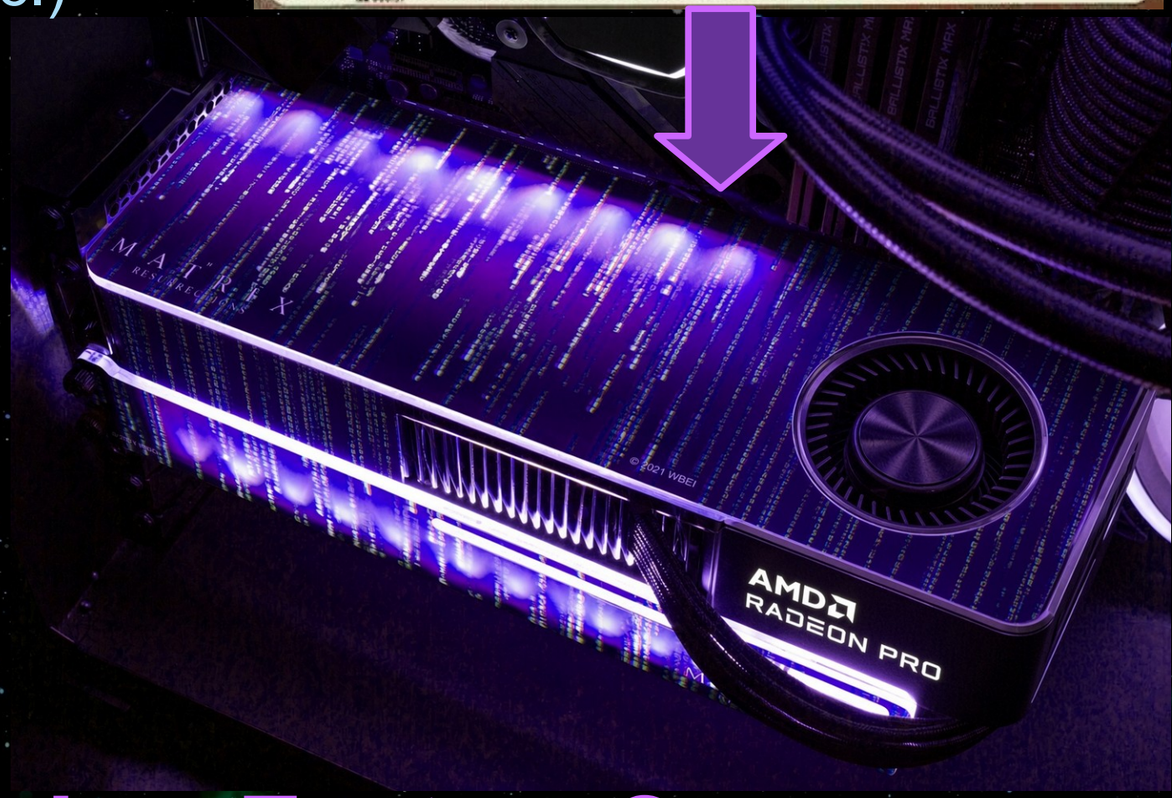
CLFORTRAN – PURE FORTRAN INTERFACE TO OPENCL

- Directive based APIs:
  - OpenACC
  - OpenMP Target

OpenACC

OpenMP®





What about just using Fortran?

Fortran Standard Parallelism: **do concurrent (DC)**

- Why can't compiler compile straight to GPU?
  Automatic parallelization and/or memory paging is difficult!
- What if the language allowed the programmer to help the compiler?
- Fortran introduced **`do concurrent`** in F2008
- DC indicates loop can be computed "out of order" (no data dependencies)

```
do concurrent (i=1:N,j=1:M)
  x(i,j) = y(i,j) + b(i)
end do
```

- Therefore, loop can *probably* be parallelized!
- Variable locality/reduction clauses: **`reduce`**,**`local`**, **`shared`**, etc.
- DC is a good candidate for compilers to offload to GPUs (and multi-core CPUs), without special APIs

Fortran Standard Parallelism: **DC vs. Directives**




```
!$omp target teams distribute
!$omp& parallel do collapse(2) reduction(+:a)
do i=1,N
  do j=1,N
    a = a + b(i,j)
  enddo
enddo
!$omp end target teams distribute parallel do
```



```
do concurrent (i=1:N,j=1:M) reduce(+:a)
  a = a + b(i,j)
enddo
```

Some needed/wanted language features:

- **Memory spaces**
  - **Fix:** Automatic memory management or unified memory
  - Workaround: Data movement with directives
- **Device selection:**
  - **Fix:** DC + Fortran **coarrays** (a dream?)
  - Workaround: MPI+Directive device selection
  - Workaround: MPI+Custom launcher
- **Asynchronous computations**
  - Workaround: Directives around DC loops?

```
!$omp target enter data map(alloc:x)
!$omp target enter data map(to:y)
!$omp target exit  data map(delete:x)
!$omp target update to(x(:,:,:,2))
!$omp target update from(y(:))
```

```
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)
```

```
!$ call omp_set_default_device(iprocsh)
```

run.sh

```
#!/bin/bash
RANK=$SLURM_LOCALID
export ROCR_VISIBLE_DEVICES=$RANK
export HIP_VISIBLE_DEVICES=$RANK
export OMP_DEFAULT_DEVICE=$RANK
exec "$@"
```

Launch: `mpirun -np $N ./run.sh <CODE>`

Compiler Implementation and Support

- Language features are great – if compilers implement and support them...

Feature	gfortran	ifx	nvfortran	flang	ftn	lfortran	amdflang
Compiles DC	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Compiles DC "reduce"	Yes	Yes	Yes	Yes	Yes	Yes	Yes
DC CPU parallelism	Mostly (no reduce)	Yes	Yes	Yes	Yes	Yes	Yes
DC GPU parallelism	No	Yes (Intel)	Yes (NVIDIA)	Yes (AMD, NVIDIA)	Yes (AMD, NVIDIA)	Yes (AMD, NVIDIA, Apple)	Yes (AMD)

- "Support" does not always mean good performance, or "full support"
- Here, we demonstrate DC AMD GPU offload using the `amdflang` compiler on a production code

AMD + LLVM Flang = amdflang



LLVM / Flang

github.com/llvm/llvm-project

flang.llvm.org

flang

amdflang is
AMD-optimized

AMD

ROCm

rocm.docs.amd.com

amdrocm-core-sdk10.1

Pre-release binaries (AFAR)

github.com/amd/InfinityHub-CI/tree/main/fortran

repo.radeon.com/rocm/misc/flang

New!

AOF

(AMD Optimizing Compiler)

amd.com/en/developer/aocc.html

Here, we use AFAR to get cutting edge features

amdflang: Offload "do concurrent" to GPUs

Activate OpenMP

Activate DC parallelism on GPU

```
amdflang -fopenmp -fdo-concurrent-parallel=device
         -offload-arch=gfx942
```

Set GPU target architecture

```
export HSA_XNACK=1
```

Enable unified memory (if available)

```
UCX_MEMTYPE_CACHE=n
HSA_ENABLE_SDMA=0
UCX_RNDV_SCHEME=get_zcopy
```

May be needed for GPU-aware MPI on consumer GPUs

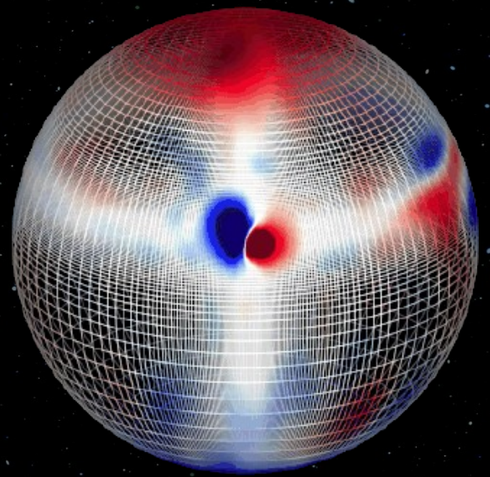


Production Code to Test DC on GPUs: POT3D

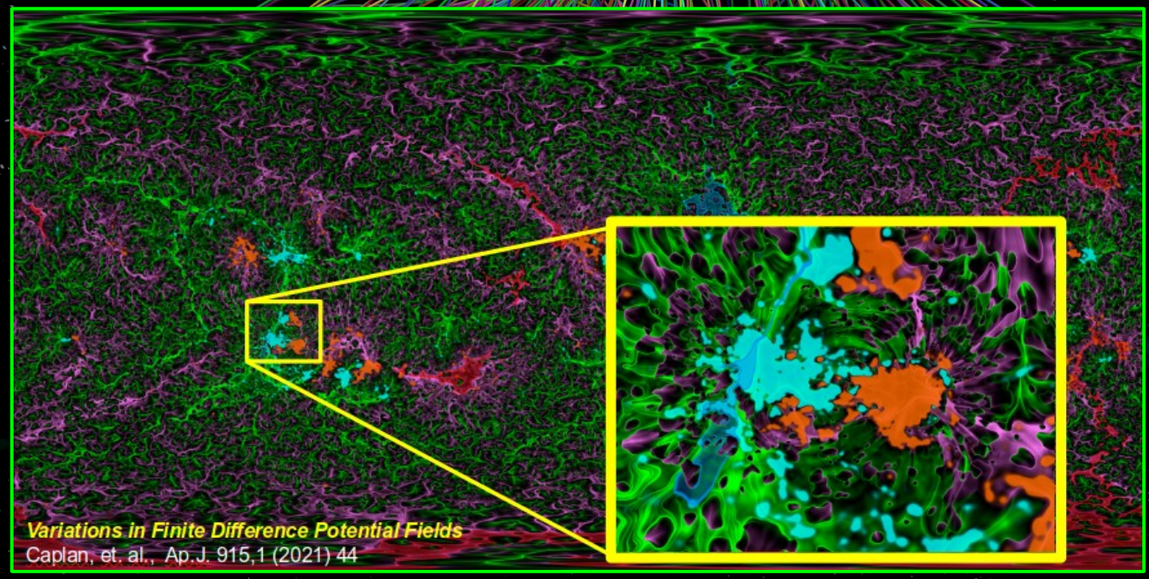
- POT3D computes potential field approximations of the solar coronal magnetic field using observations of the surface magnetic field as a boundary condition
- Included in the Standard Performance Evaluation Corporation's **SPEChpc®2021** and **SPEC CPU®2026** benchmark suites
- Used in the ISC2023 Student Cluster Competition
- Non-uniform logically-rectangular spherical staggered grid



ISC High Performance
STUDENT CLUSTER COMPETITION



github.com/predsci/pot3d



POT3D DC Implementation

Two branches:

Pure Fortran DC (no directives)
stdpar

```
do concurrent (i=1:N,j=1:M)
  y(i,j) = a*x(i,j) + b
enddo
```

DC + OpenMP Target Data Directives
stdpar_ompdata

```
!$omp target enter data map(alloc:x)
!$omp target enter data map(to:y)
!$omp target exit  data map(delete:x)
!$omp target update to(x(:,:,:,2))
!$omp target update from(y(:))
```

stdpar can require unified/automatic memory for good performance

POT3D is *highly* memory bandwidth bound

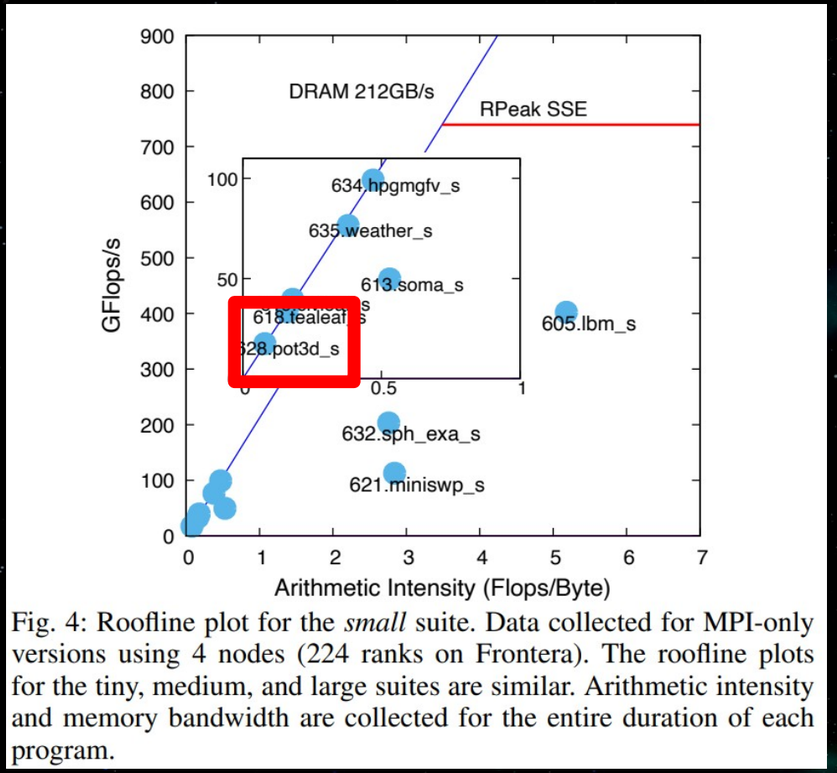
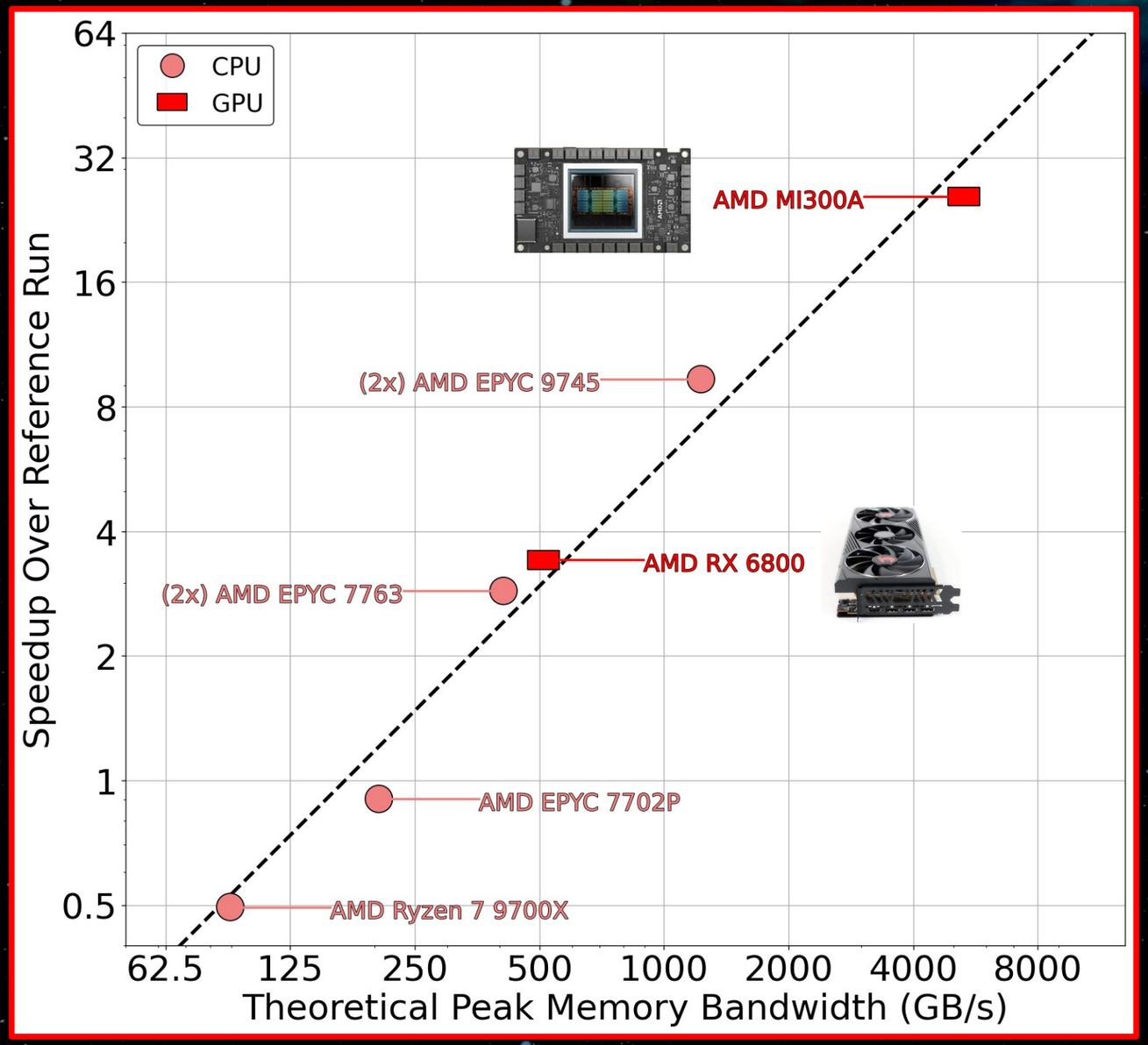
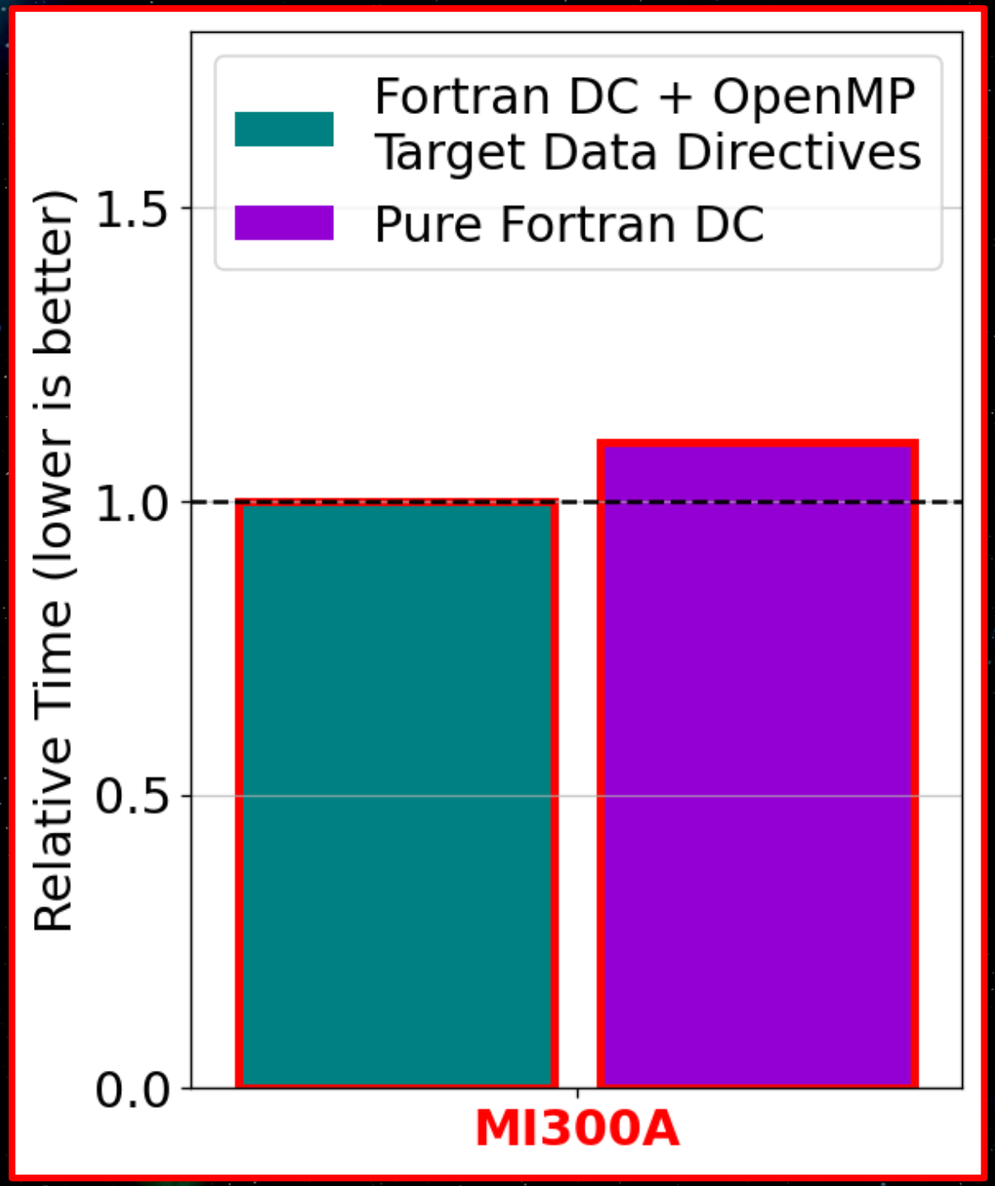


Fig. 4: Roofline plot for the *small* suite. Data collected for MPI-only versions using 4 nodes (224 ranks on Frontera). The roofline plots for the tiny, medium, and large suites are similar. Arithmetic intensity and memory bandwidth are collected for the entire duration of each program.

[Brunst et. al. (2022)]

Results (Single CPU/GPU)

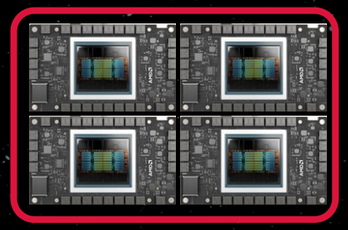




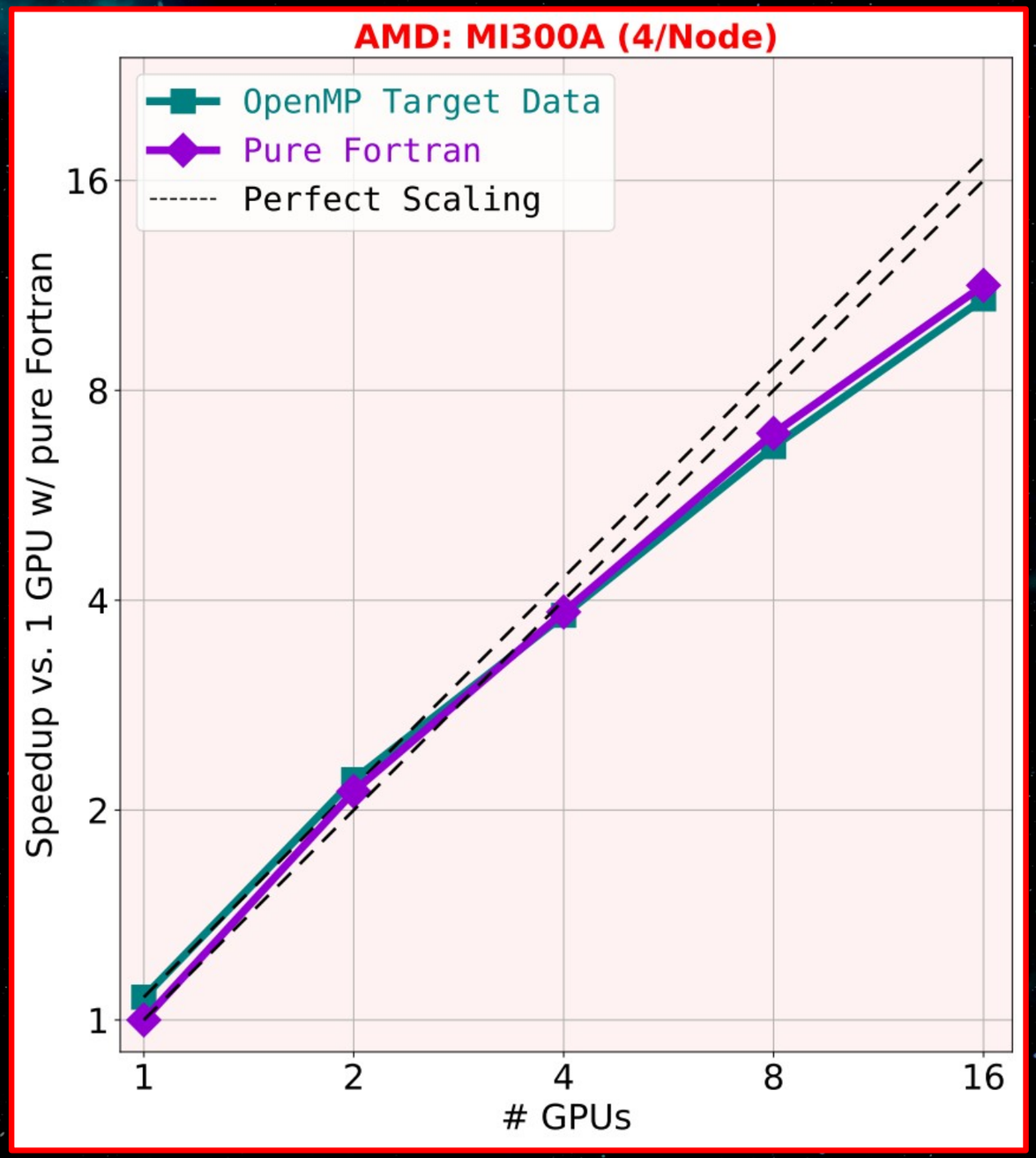
stdpar_ompdata

Results (Multiple GPUs)

- Four MI300A APUs per node

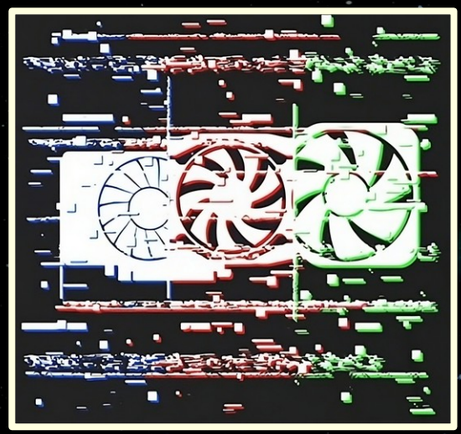


- Strong scaling performance is normal for this run (similar scaling on other platforms)
- Pure Fortran sometimes slightly slower, sometimes slightly faster
- **Key takeaway:**
  “Just Fortran” (+MPI) ran great across multiple AMD GPUs!



Summary

- **Fortran is a modern, well supported language with built-in parallelism** [coarrays and do concurrent (DC)]
- We demonstrated Fortran's DC accelerate a production HPC application on AMD GPUs with the `amdflang` compiler
- On APUs (e.g. AMD MI300A), unified memory yields great performance (with GPU-aware MPI for running across multiple GPUs) – **no directives needed!**
- On consumer/"prosumer" GPUs, using **OpenMP Target** data directives for memory management gives great performance
- **DC on GPUs is portable - other vendors support it, including unified memory capabilities**
- Eventually, "coarrays" could be used to replace/hide MPI for Fortran codes run on distributed GPU clusters



arxiv.org/abs/2608.20586 caplanr@predsci.com

Portability of Fortran's 'do concurrent' on GPUs II

Ronald M. Caplan, Miko M. Stulajter, Jon A. Linker, Jeff Larkin, Nikolaos Tselepidis, Harald Servat, Shiquan Su, Giacomo Capodaglio, Johanna Potyka

fortran-lang.org
fortran-lang.discourse.group
𝕏@fortranlang

FortranFuture
powered by Fortran

fortranfuture.net



“Towards a Digital Twin of the Sun's Atmosphere for Space Weather Applications”
NASA Booth #4038
Wed. Nov 18th 4:30 PM CST
Thurs. Nov. 19st 12:20 PM CST

“Open Polarity”
Art of HPC, Skyline Ballroom West