Can Fortran's `do concurrent' replace directives for accelerated computing?

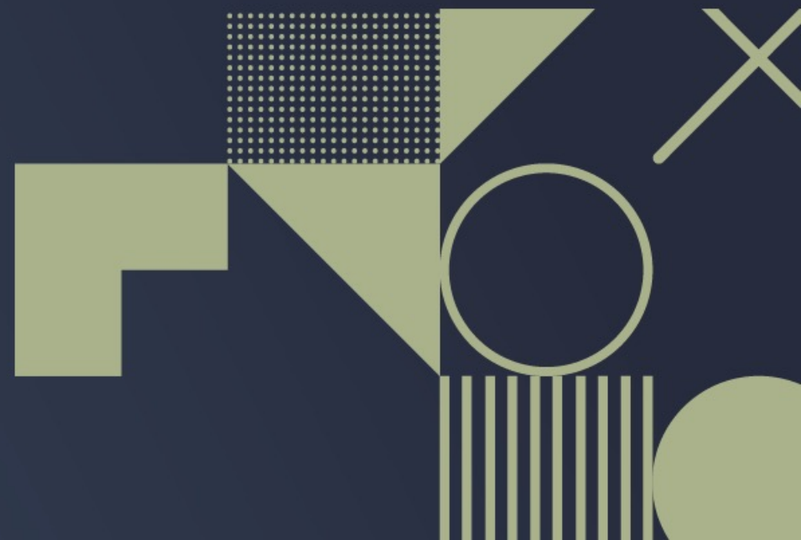
Miko Stulajter, Ronald M. Caplan, and Jon A. Linker

Predictive Science Inc.

www.predsci.com

WACCPD 2021

Eighth Workshop on Accelerator Programming Using Directives



Introduction

- Directives (e.g. OpenMP/OpenACC) are popular for parallelization on CPUs and GPUs
- Standard languages have begun to add features that compilers can use to parallelize code
  - C++17's Standard Parallel Algorithms
  - Fortran's **`do concurrent`**
- GPU-offload of directives supported by many compilers:
  - NVIDIA HPC SDK, Intel OpenAPI HPC Toolkit, LLVM Flang (along with AMD AOCC/AOMP extensions), IBM's XL, HPE's Cray Fortran
- Here, we want to test the current status of being able to replace directives with **`do concurrent`** for accelerated computing

Directives

Directives are widely used for parallelizing codes

PROS

- Performance can be similar to low-level APIs
- Portability
- Minimal code interference

CONS

- Not always supported
- APIs can change, requiring re-writes
- Can make code harder to read

```
for (i=0; i<N; i++)
  y[i] = a*x[i] + y[i];
```

```
#pragma acc kernels
for (i=0; i<N; i++)
    y[i] = a*x[i] + y[i];
```

OpenACC

More Science, Less Programming

"OpenACC is a user-driven directive-based performance-portable parallel programming model. It is designed for scientists and engineers interested in porting their codes to a wide-variety of heterogeneous HPC hardware platforms and architectures with significantly less programming effort than required with a low-level model." - ***openacc.org***

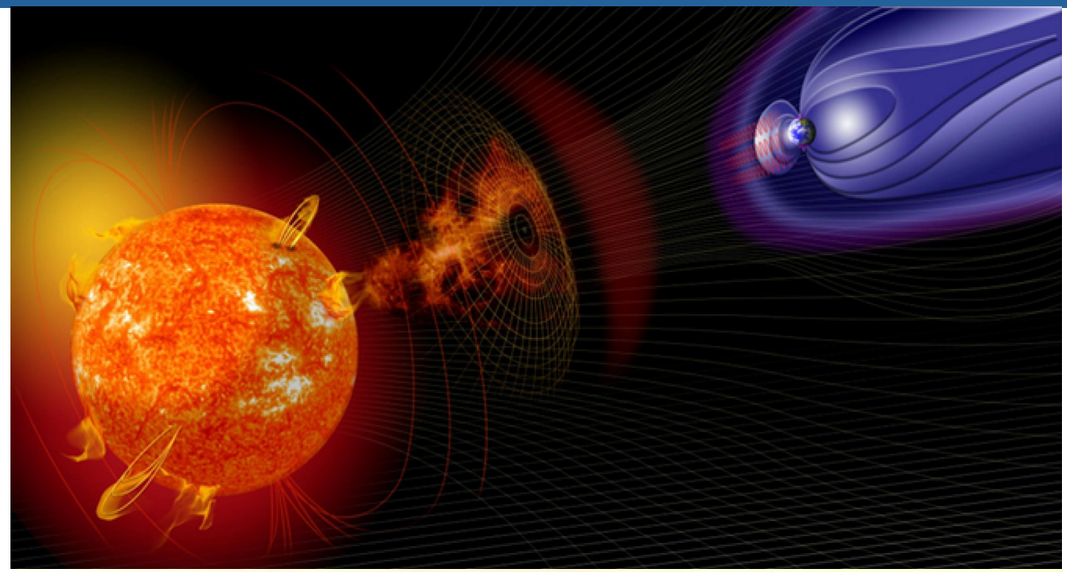
OpenMP®

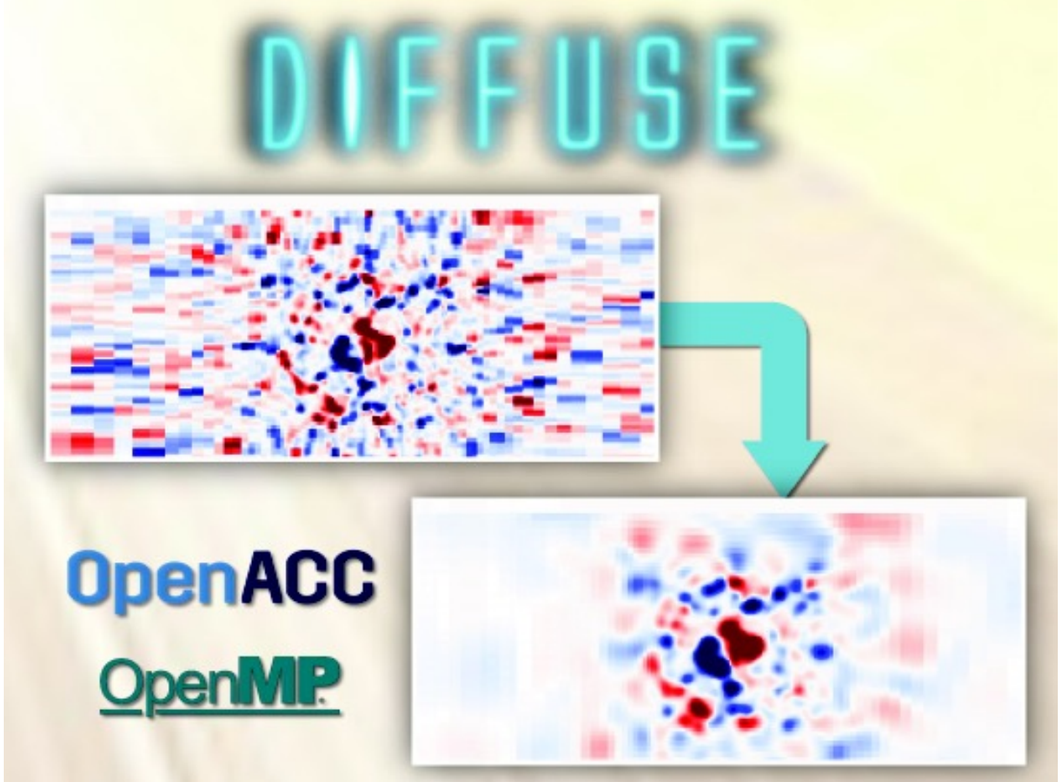
Enabling HPC since 1997

"OpenMP is a specification for a set of compiler directives, library routines, and environment variables that can be used to specify high-level parallelism in Fortran and C/C++ programs." - ***openmp.org***

Code Description

- Space weather events can cause interference & damage to electronic infrastructure
- New NASA+NSF program called SWQU to improve models of solar wind and solar storms
- HipFT: A flux-evolution code to generate observation-based model boundary conditions
- Most expensive computation of HipFT encapsulated in BC-smoothing tool DIFFUSE
- Here we use DIFFUSE as a mini-app of HipFT for DC tests





Code Description cont. / Test Problem

DIFFUSE

- Integrates the spherical surface heat equation
- Logically rectangular non-uniform grid
- Operator is discretized with a second-order central finite-difference scheme
- Time integration with second-order Legendre polynomial extended stability Runge-Kutta scheme (RKL2)

For a test problem, we select a real-world example of using DIFFUSE, that of smoothing the ‘Native res PSI map‘ described in

Caplan, R.M., Downs, C., Linker, J.A., Mikic, Z.: Variations in finite-difference potential fields. The Astrophysical Journal **915**(1), 44 (jul 2021). https://doi.org/10.3847/1538-4357/abfd2f

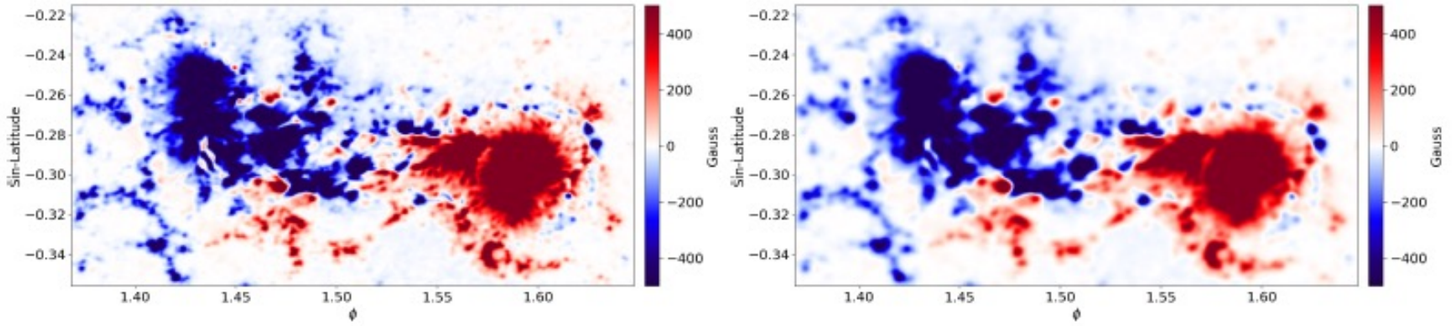


Fig. 1. Zoomed-in detail of the test case magnetic field map before (left) and after (right) smoothing with `diffuse`.

The grid has a resolution of 3974x2013 in theta-phi and the test requires 40,260 iterations of the diffusion operator

Computational Environment

- Utilized the latest compilers at time of testing

Compiler Suite	Compiler	Version
GNU Compiler Collection	`gfortran`	11.2
NVIDIA HPC SDK	`nvfortran`	21.7
Intel OneAPI HPC Toolkit	`ifort` (classic)	21.3

- Singularity containers used to streamline testing and provide reproducibility
- Containers provide performance comparable to bare metal

- Computational resources provided by NSF's XSEDE program and the CSRC at SDSU

PSC

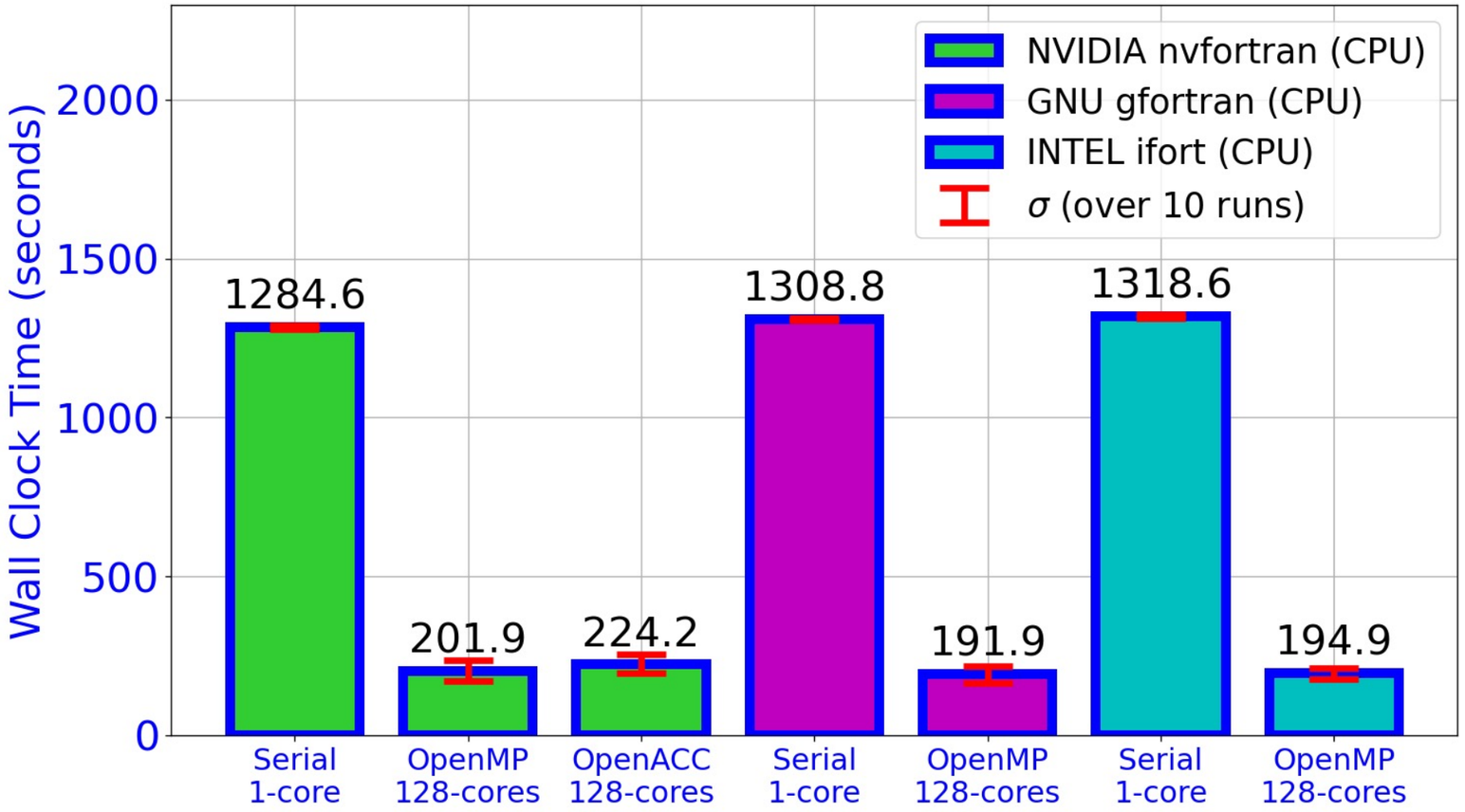
San Diego State University

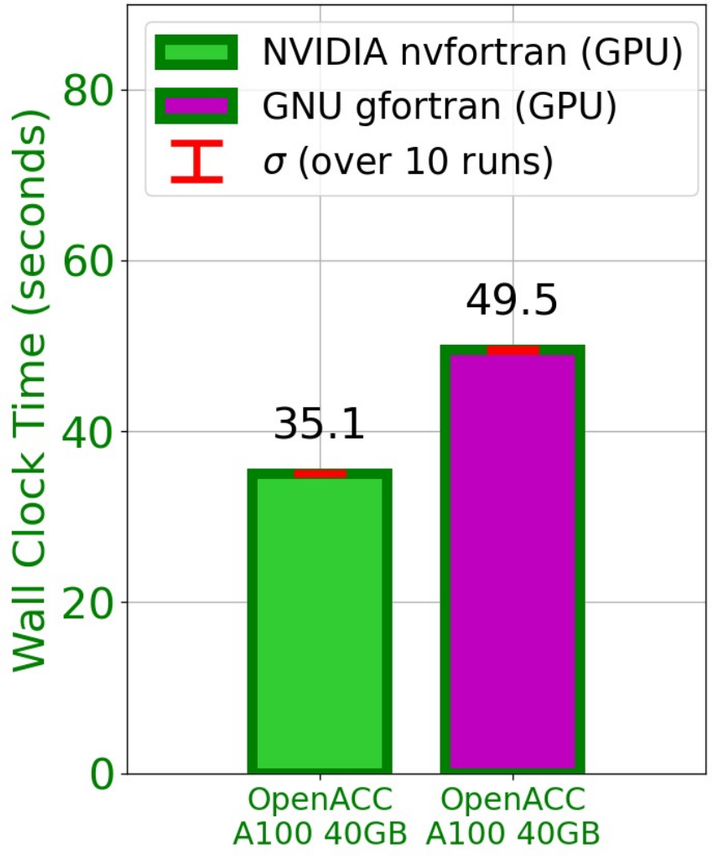
CSRC — an interdisciplinary gateway to the future

NVIDIA DGX A100

	CPU	GPU
CPU/GPU Model	(2x) AMD EPYC 7742 (128 cores)	NVIDIA A100 SXM4
Peak Memory Bandwidth	381.4 GB/s	1555 GB/s
Clock Frequency (base/boost)	2.3/3.4 GHz	1.1/1.4 GHz
RAM	256 GB	40 GB
Peak DP FLOPs	7.0 TFLOPs	9.8 TFLOPs

Baseline Performance Results





- OpenACC and OpenMP similar performance for CPU
- All compilers similar for CPU
- Note DIFFUSE is memory-bound so low speed-up over CPU cores is not unusual
- `nvfortran` faster than `gfortran` for OpenACC GPU

Implementation: Fortran's `do concurrent'

- Introduced in ISO Standard Fortran 2008
- Indicates loop can be run with out-of-order execution
- Can be seen as hint to the compiler that loop may be parallelizable
- Current specification has no support of reductions or atomics

Code 1 Nested `do` loops with OpenMP/ACC directives

```
!$omp parallel do collapse(2) default(shared)
!$acc parallel loop collapse(2) default(present)
      do  i=1,N
          do  j=1,M
              Computation
          enddo
      enddo
!$acc end parallel loop
!$omp end parallel do
```

Code 2 Nested `do` loops as a `do concurrent` loop

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

Compiler	Version	`do concurrent` parallelization support
`gfortran`	≥ 9	Parallelizable on CPU with "-ftree-parallelize-loops=X" flag. Locality of variables is not supported.
`nvfortran`	≥ 20.11	Parallelizable on CPU and GPU with the "-stdpar" flag. Locality of variables is supported.
`ifort`	≥ 19.1	Parallelizable on CPU with the "-fopenmp" flag. Locality of variables is supported.

Implementation: Code Versions

- ***Original:*** Uses directives and data movement directives
- ***New:*** Uses `do concurrent` except for reductions and data movement directives
- ***Serial:*** No directives or `do concurrent` loops
- ***Experimental:*** All directives removed, and all parallelizable loops utilize `do concurrent` including reductions.
  This represents "ideal" situation of having no directives

	`do concurrent`	Directives
Original	None	all loops & data management
New	all loops except reductions	reduction loops & data management
Serial	None	None
Experimental	all loops	None

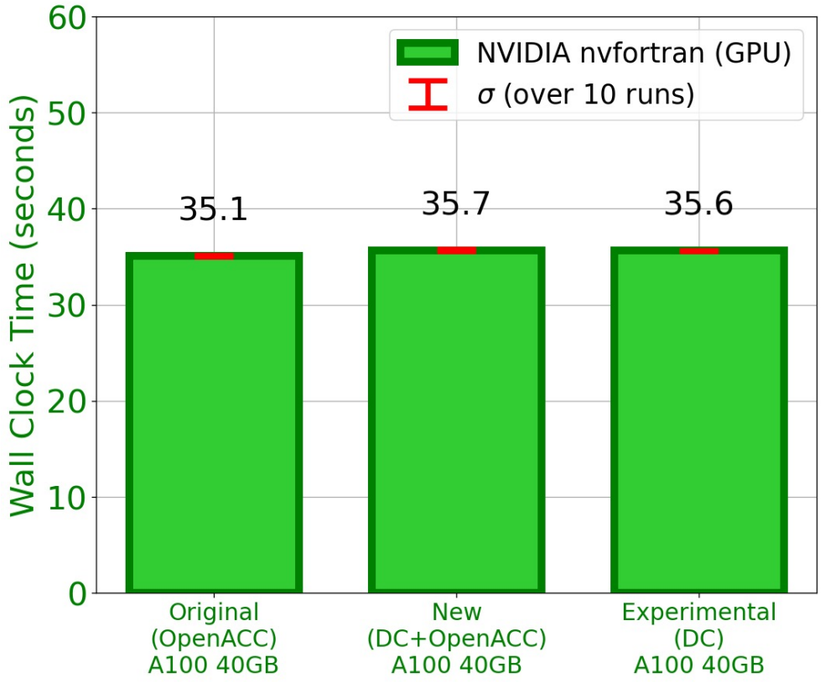
Implementation: Compiler Options

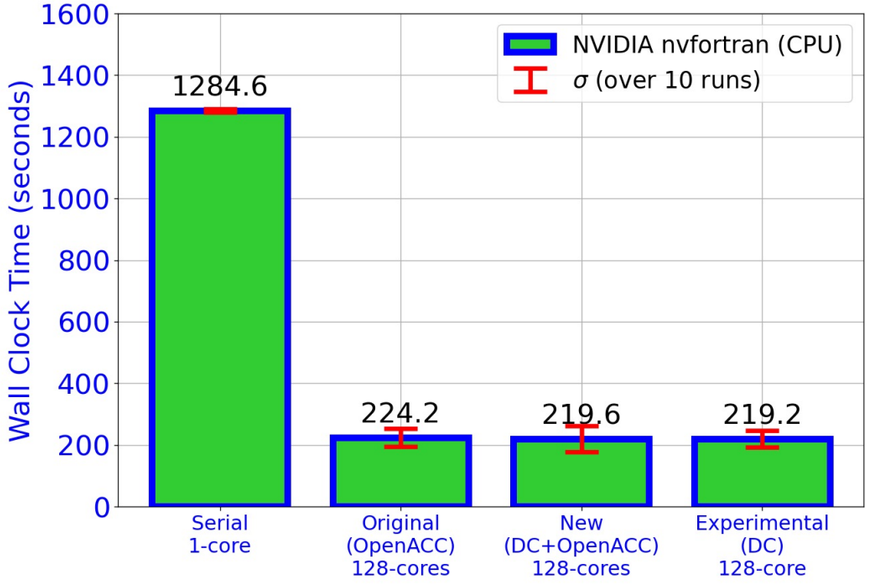
- Utilized `-O3` and `-march=<ARCH>` for all compilers
- **`gfortran`**:
  - CPU: `-fopenmp` and/or `-ftree-parallelize-loops=<N>`
  - GPU: `-fopenacc` and `-foffload=nvptx-none`
- **`nvfortran`**:
  - CPU: `-acc=multicore` and/or `-stdpar=multicore`
  - GPU: `-stdpar=gpu` and/or `-acc=gpu` and `-gpu=cc<X><Y>,cuda<X>.<Y>`
    - Note unified memory is enabled by default (can turn off with `-gpu=nomanaged`)
- **`ifort`**:
  - CPU: `-fopenmp`
  - GPU: No support for NVIDIA GPUs

Results: nvfortran

GPU	
Code	Compiler flags
Original	`-acc=gpu -gpu=cc80,cuda11.4`
New	`-acc=gpu -stdpar=gpu -gpu=cc80,cuda11.4`
Experimental	`-stdpar=gpu -gpu=cc80,cuda11.4`

CPU	
Code	Compiler flags
Serial	
Original	`-acc=multicore`
New	`-acc=multicore -stdpar=multicore`
Experimental	`-stdpar=multicore`





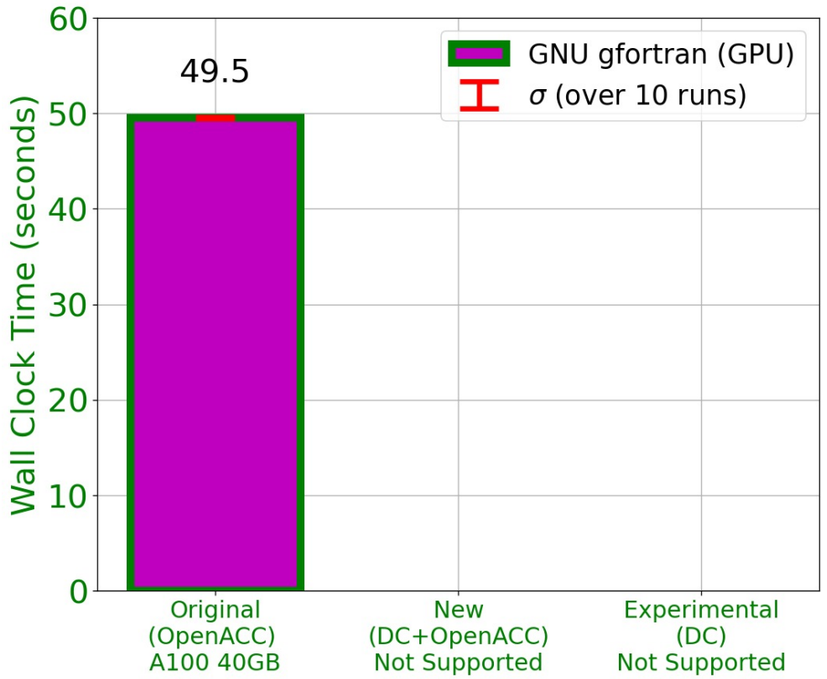
- GPU performance stayed consistent
- CPU performance stayed consistent
- Experimental code worked correctly

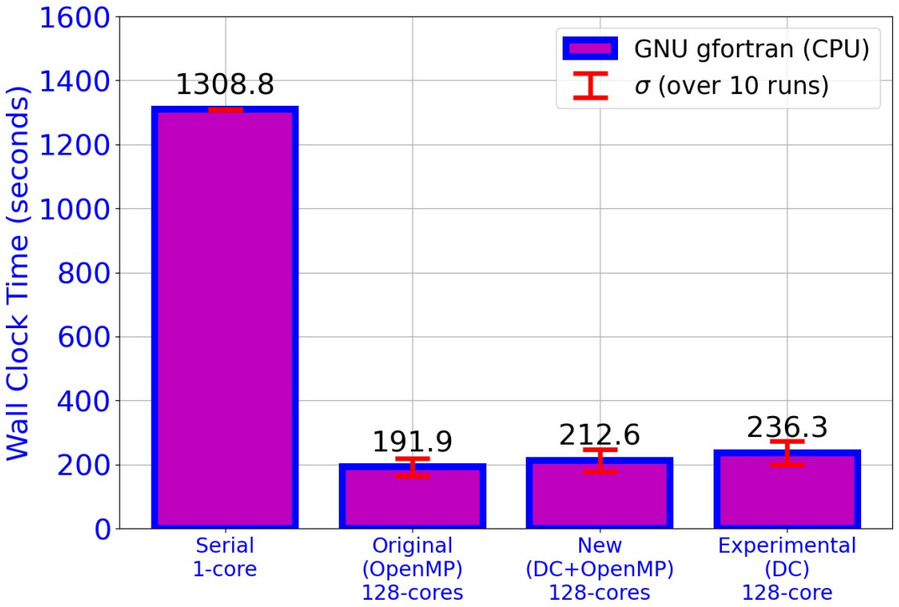
Results: `gfortran`

GPU	
Code	Compiler flags
Original	`-fopenacc -foffload=nvptx-none -fopenacc-dim=::128`
New	No Support
Experimental	No Support

CPU	
Code	Compiler flags
Serial	
Original	`-fopenmp`
New	`-fopenmp -ftree-parallelize-loops=128`
Experimental	`-ftree-parallelize-loops=128`

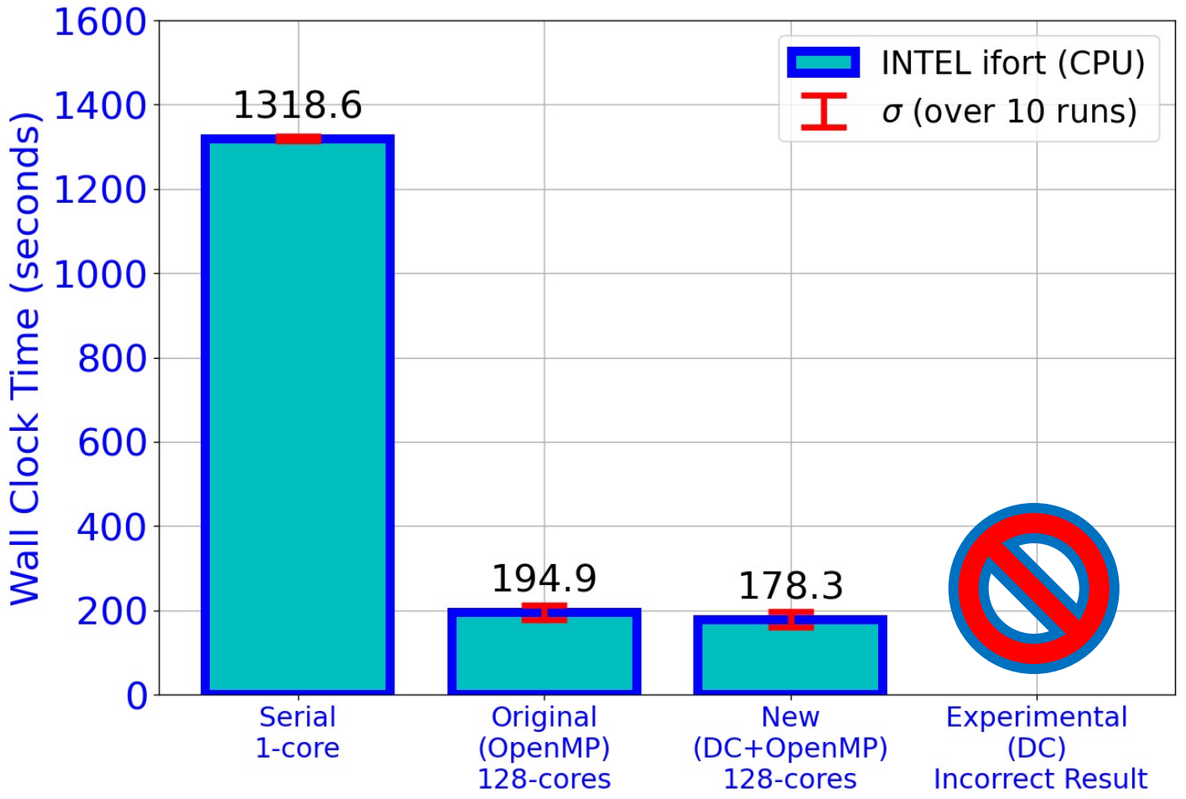
- No GPU support of do concurrent parallelism
- CPU **`do concurrent`** support relies on auto parallelization
- Small performance loss with auto parallelization





Results: `ifort`

- **`do concurrent`** gave better performance on CPU
- Currently no support on the GPU for **`do concurrent`**
- The experimental code gave the wrong answer
  - Compiler failed to correctly detect reductions



GPU	
Code	Compiler flags
Original	No Support
New	No Support
Experimental	No Support

CPU	
Code	Compiler flags
Serial	
Original	`-fopenmp`
New	`-fopenmp`
Experimental	Incorrect Results

Discussion

- **Compatibility on the GPU**:
  - Currently only `nvfortran` has do concurrent support
  - Using do concurrent we lose `gfortran` GPU support
  - Planned `ifort` support of do concurrent on Intel GPUs
  - Removing data movement directives and relying on unified memory could cause performance loss, but doesn't here
- **Portability**:
  - CPU multicore parallelization was not lost (except for *Experimental* code)
  - `nvfortran` and `ifort` have direct support of **`do concurrent`** on CPUs
  - `gfortran` relies on auto-parallelization detection
  - Implicit reductions with do concurrent not supported everywhere

Discussion Continued

- **Performance**:
  - Comparable performance for the *original* and *new* code for both CPUs and GPUs
  - Unified memory on GPU gave comparable performance to manual data management with directives in this case
- **Summary**:
  - **`do concurrent`** allows cleaner looking code and adds robustness
  - **`nvfortran`** allowed for the elimination of *all* directives
  - Using a combination of directives and **`do concurrent`** gives better cross compiler/hardware compatibility at this time

Conclusion

Can Fortran's `do concurrent` replace directives for accelerated computing?

- With `nvfortran` and NVIDIA GPUs, for some codes (such as ours) the answer is **YES**, and with no (or minimal) loss of performance.
- Upcoming language features and compiler implementations, may allow more complicated codes to eventually be parallelized without directives

DOI 10.5281/zenodo.5253520

predsci.com/papers/dc

Make sure singularity container does not have overhead:

Table 12. Timing results on a Bridges2 CPU compute node using `gfortran` 10.2 bare metal and form within a Singularity Container

Code	Run method	real (s)	user (s)	system (s)
Serial	Bare Metal	1306.10	1294.30	0.154
	Singularity	1300.43	1287.50	0.168
Original	Bare Metal	164.87	20782.32	5.935
	Singularity	165.27	20777.85	7.248

Thank you for your attention.

Questions?

https://arxiv.org/abs/2110.10151