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

OpenACC BoF

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

Predictive Science Inc.

www.predsci.com

Introduction

- Directives are popular for parallelization on CPUs & GPUs

OpenACC More Science, Less Programming

OpenMP Enabling HPC since 1997

PROS

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

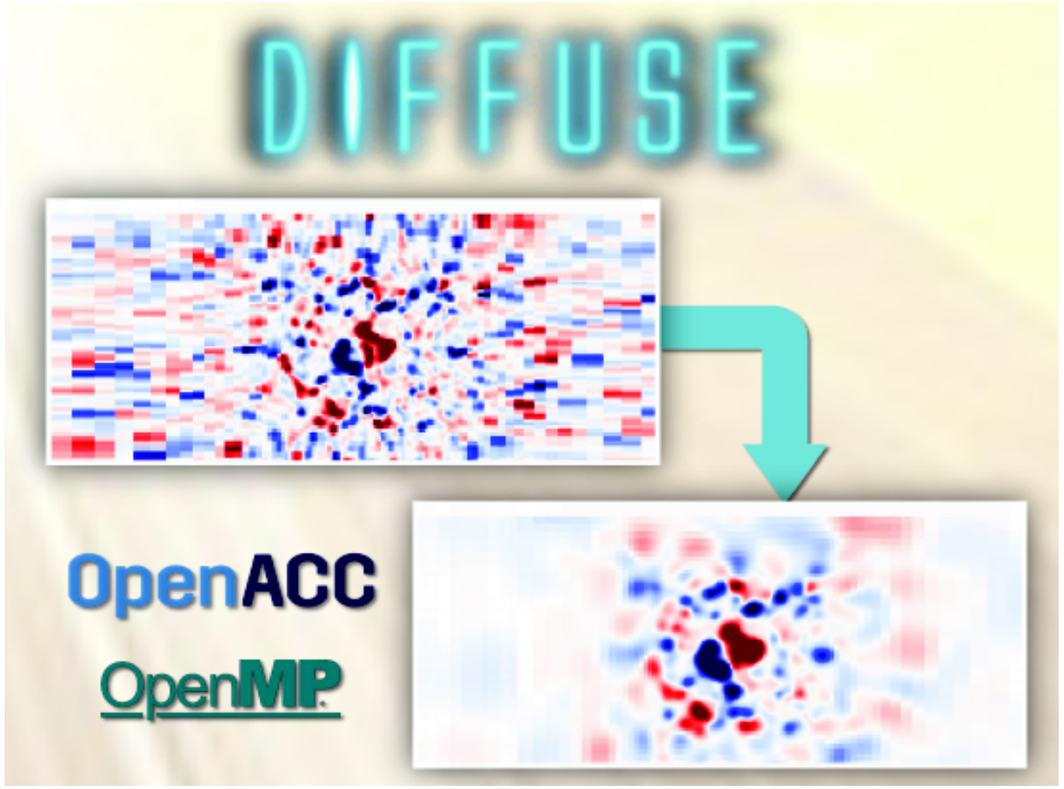
CONS

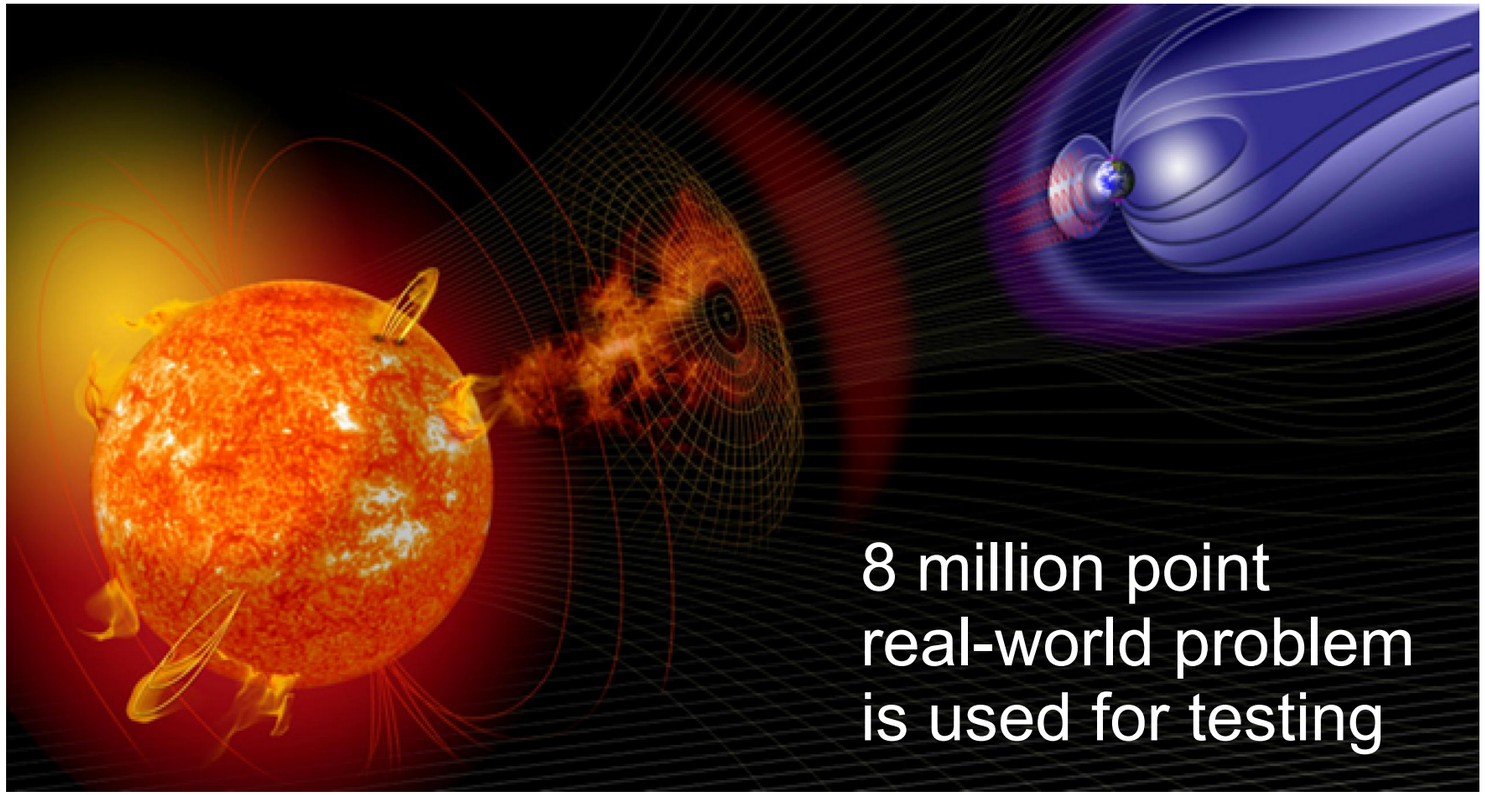
- Not always supported
- Spec more fluid than language, so may need re-writes
- Can make code harder to read (e.g. deep copy, device type optimizations, etc.)

- Standard languages have begun to add features that compilers can use to parallelize code without directives:
  - C++17's Standard Parallel Algorithms and Fortran's **`do concurrent`**

GOAL: Test the current status of being able to replace directives with `do concurrent` for accelerated computing

Test Code Description





Integrates the spherical surface heat equation on a logically rectangular non-uniform grid with a finite difference scheme which includes:

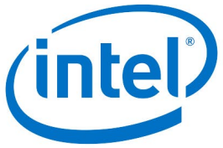
- **Vector/array operations**
- **Stencil operations**
- **Reduction operations**

Does NOT have:

- **MPI (CUDA/RoCm-aware)**
- **Atomics**
- **Multi-GPU**
- **Derived types**
- **Function calls in loops**

Computational Environment

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

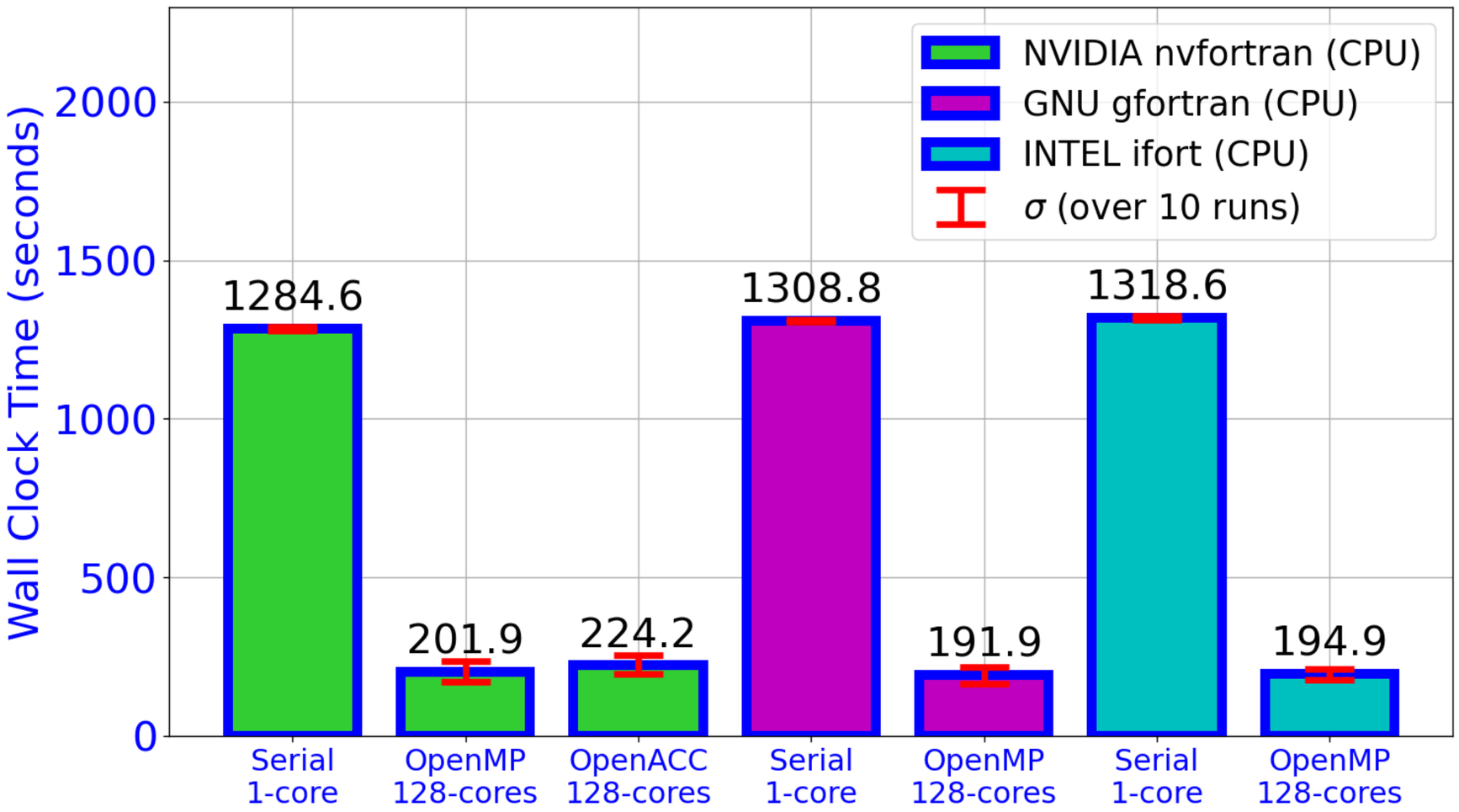


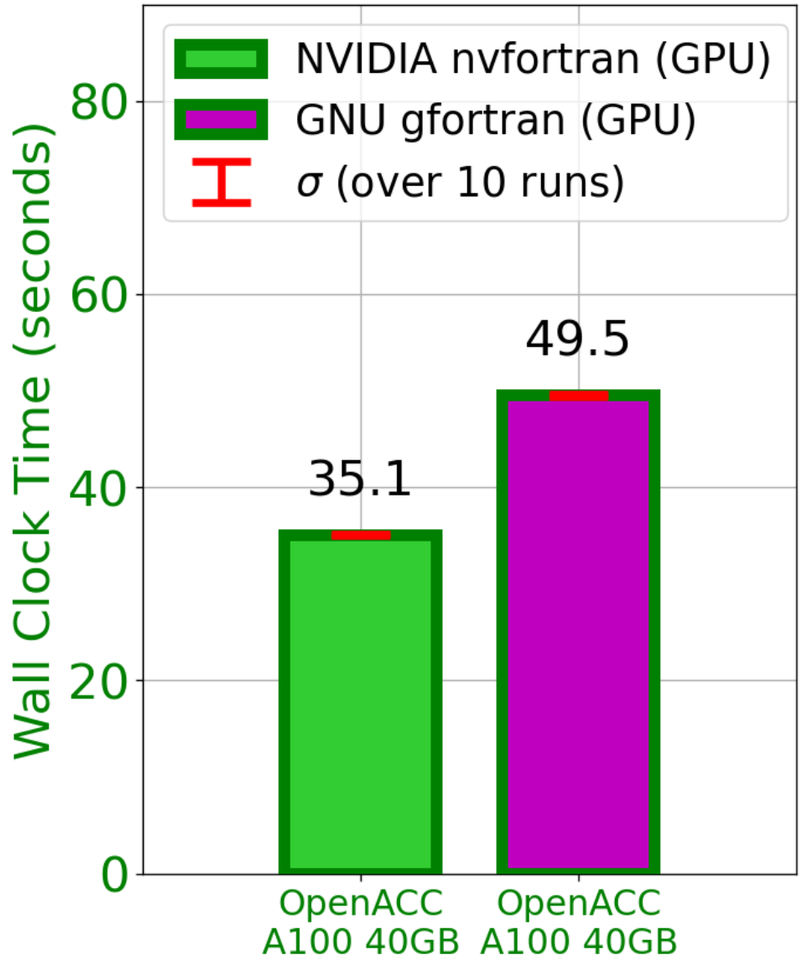
XSEDE
Extreme Science and Engineering Discovery Environment

PSC

	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





- All CPU runs have similar performance for all compilers
- **`nvfortran`** runs faster on GPU than **`gfortran`**

Implementation: Fortran's `do concurrent'

- Introduced in Fortran 2008
- Indicates loop can be run with out-of-order execution
- Can be used as hint to the compiler that loop may be parallelizable
- Current specification has no support for reductions, atomics, device selection, conditionals, etc.

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
```

Implementation: Code Versions

Serial

Original

New

Experimental

	`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

Experimental version represents "ideal" situation of having no directives

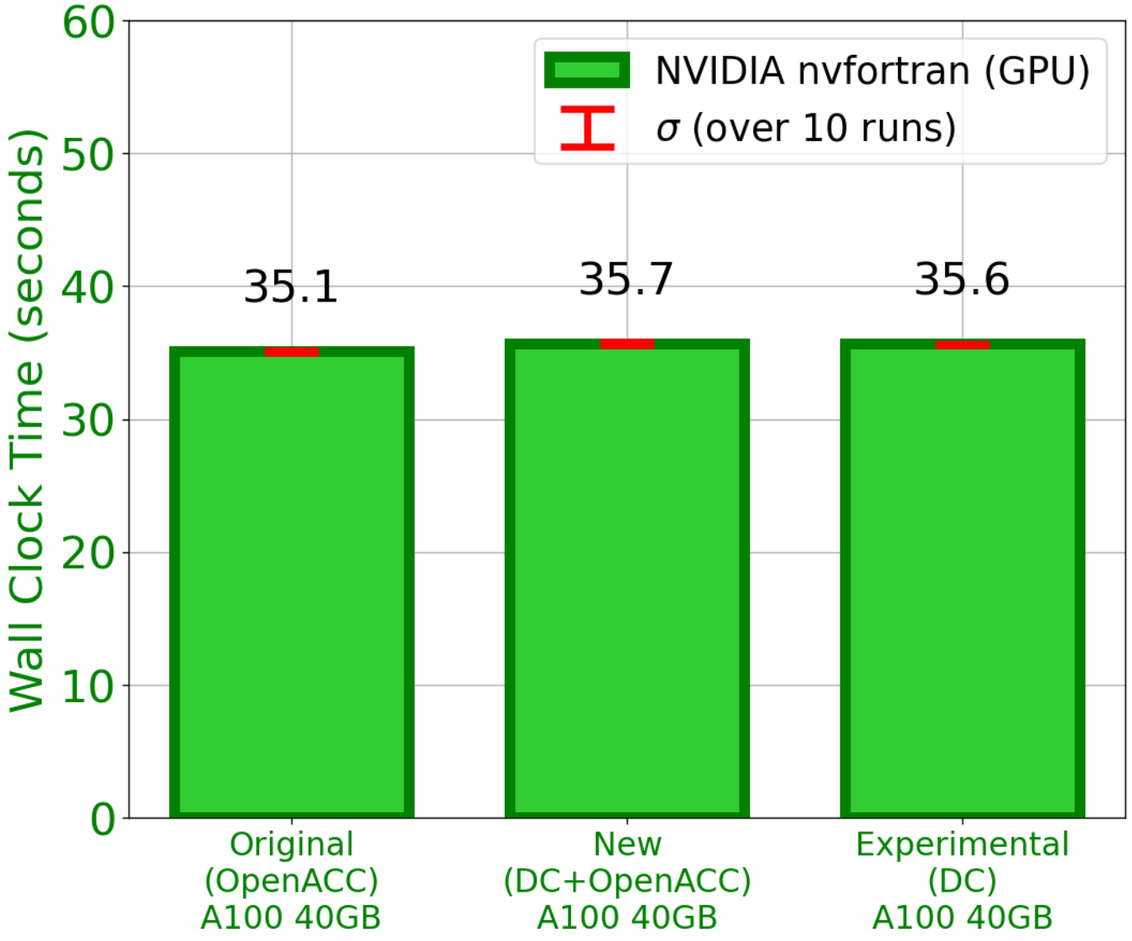
Implementation: Compiler Options

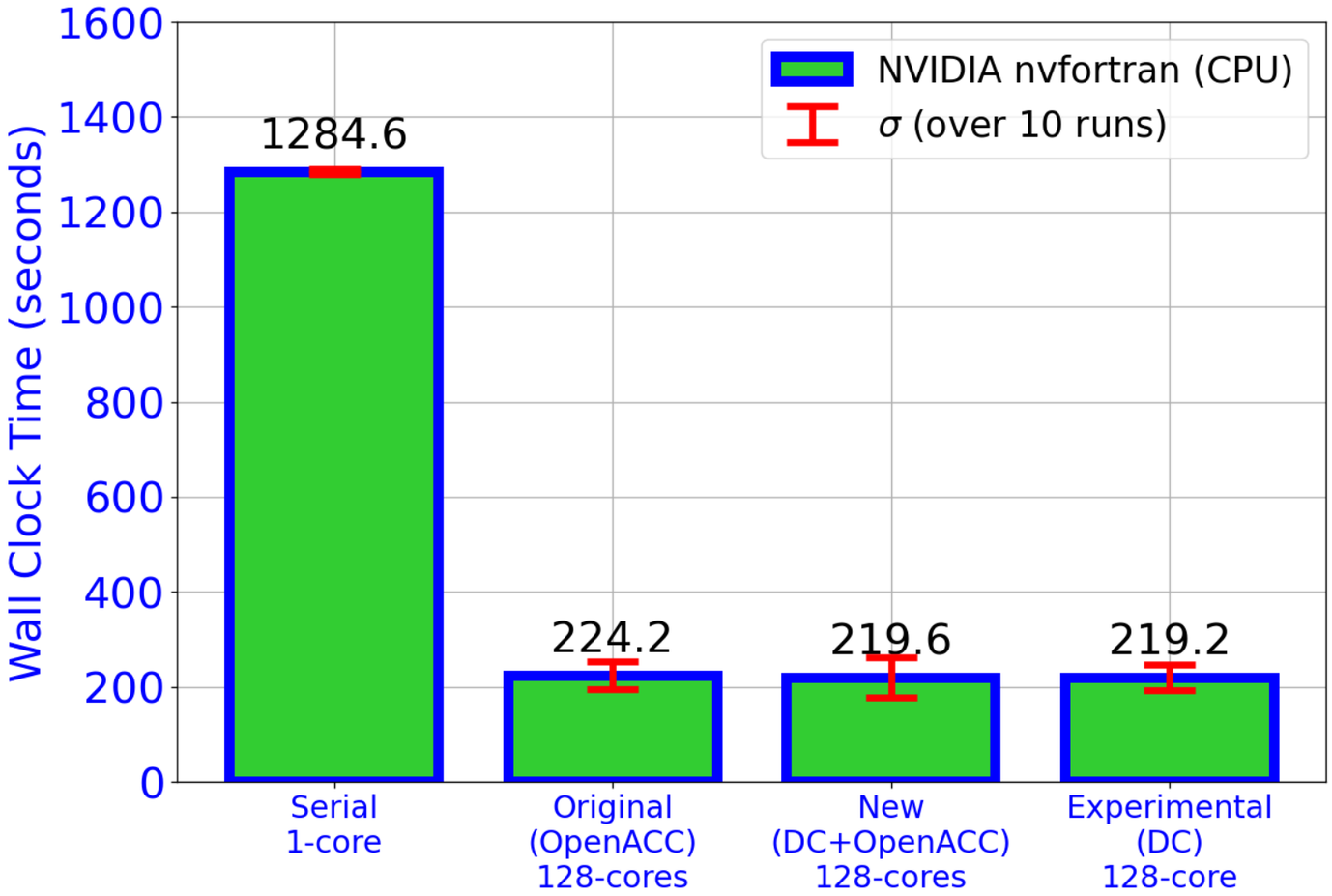
- Utilized `-O3` and `-march=<ARCH>` for all compilers
- **`gfortran`**:
  - CPU: `-fopenacc` and/or `-ftree-parallelize-loops=<N>`
  - GPU: `-fopenacc -foffload=nvptx-none -fopenacc-dim=::128`
- **`nvfortran`**:
  - CPU: `-mp` or `-acc=multicore`
  - GPU: `-stdpar=gpu` and/or `-acc=gpu -gpu=cc<XY>,cuda<X>.<Y>`
  - Note: managed memory is enabled by default when using `stdpar`! (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`

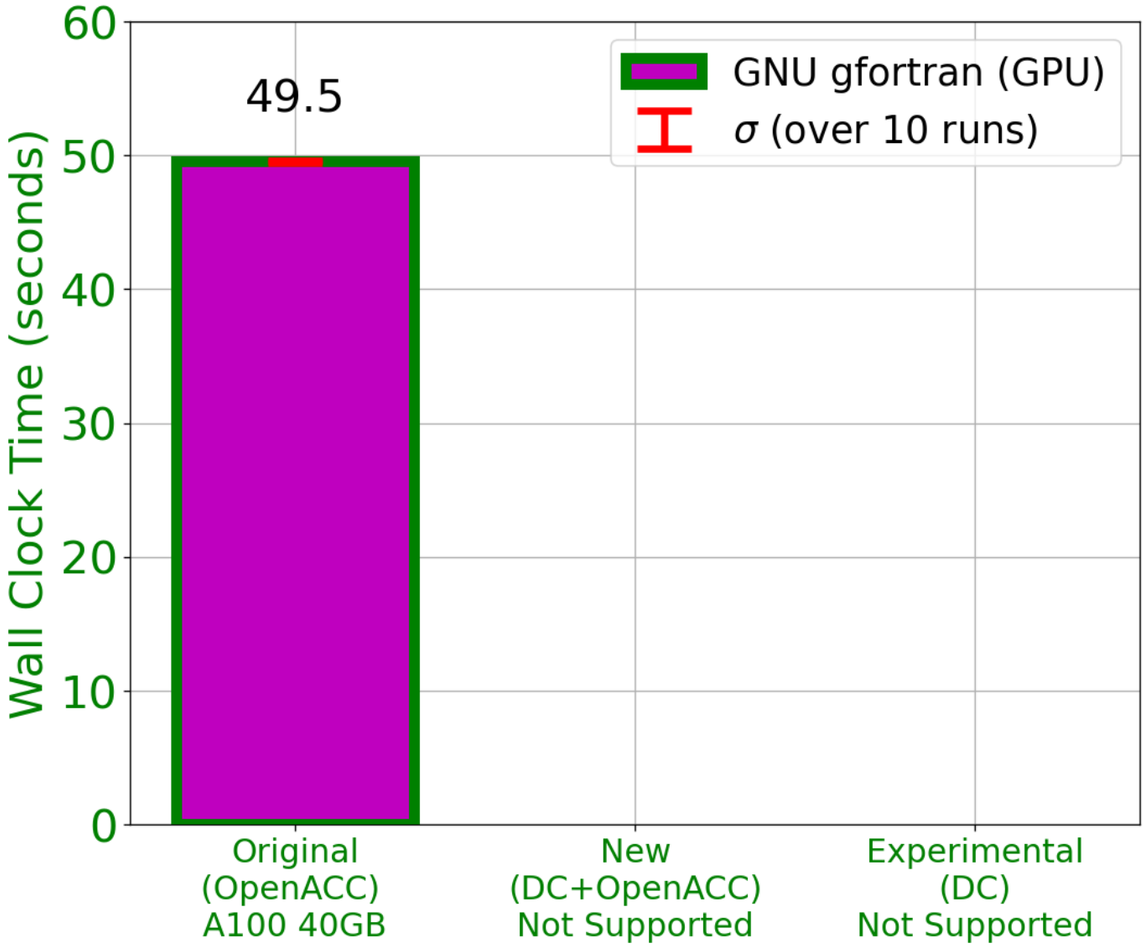


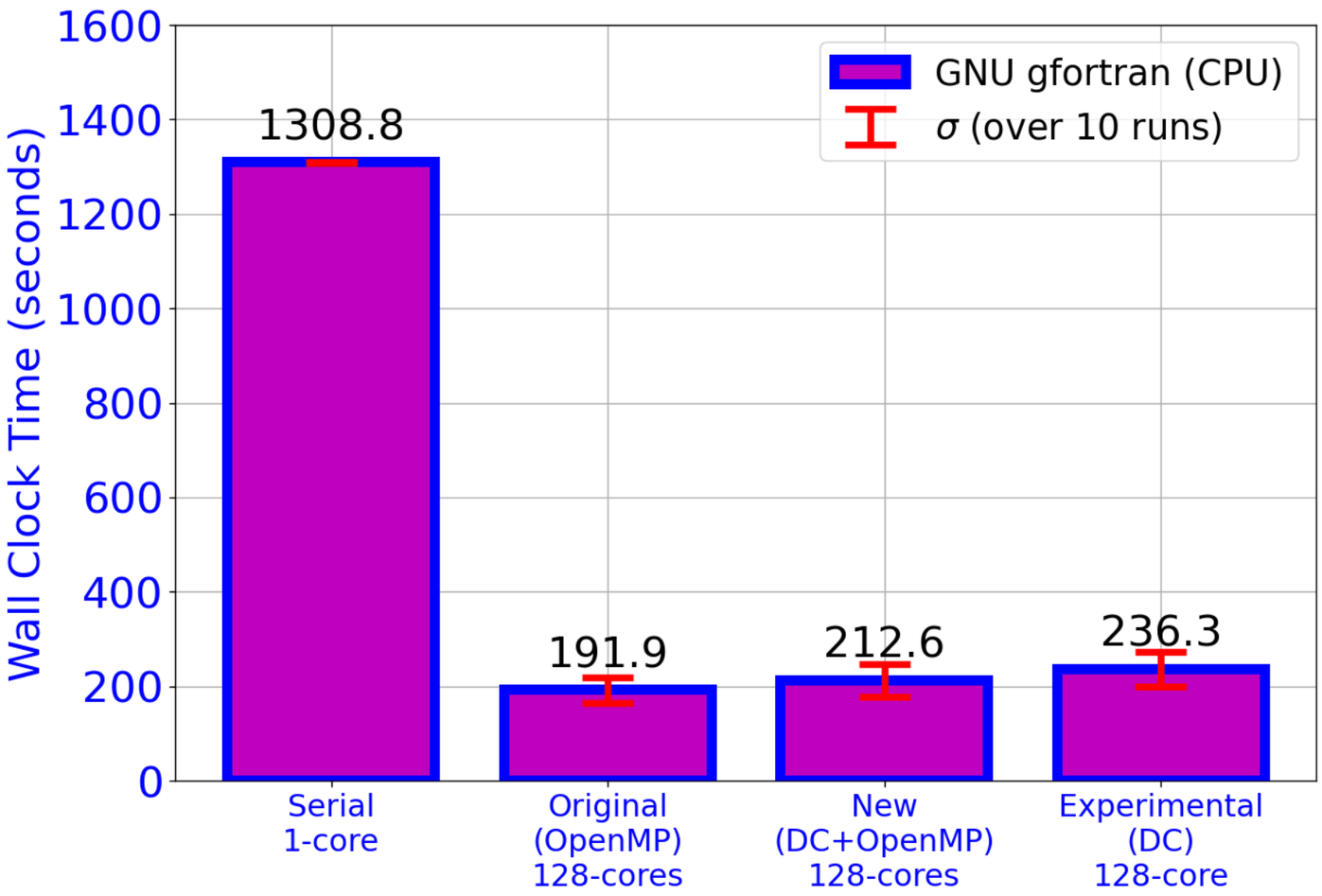


Results: gfortran

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`

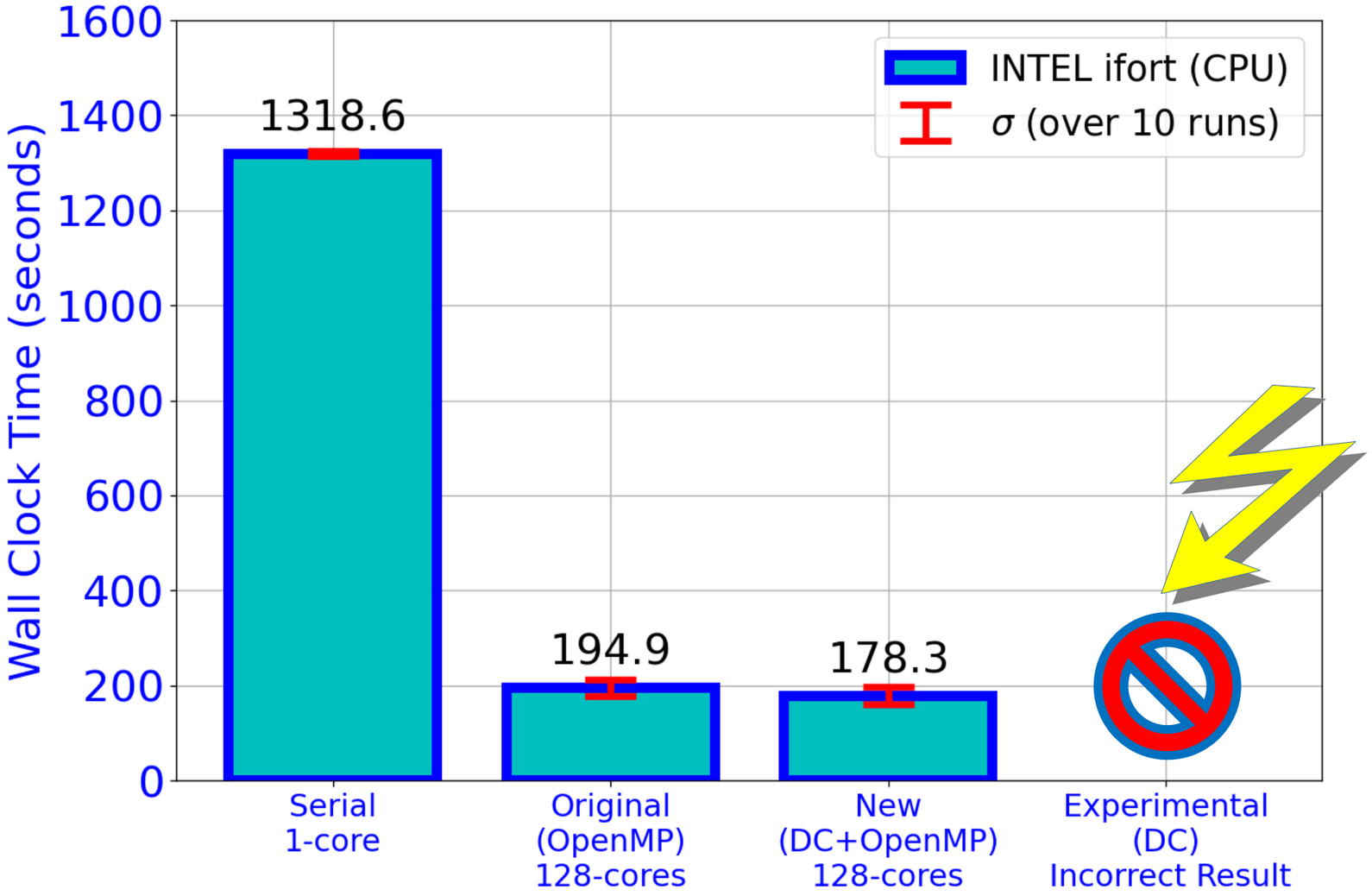




Results: `ifort`

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

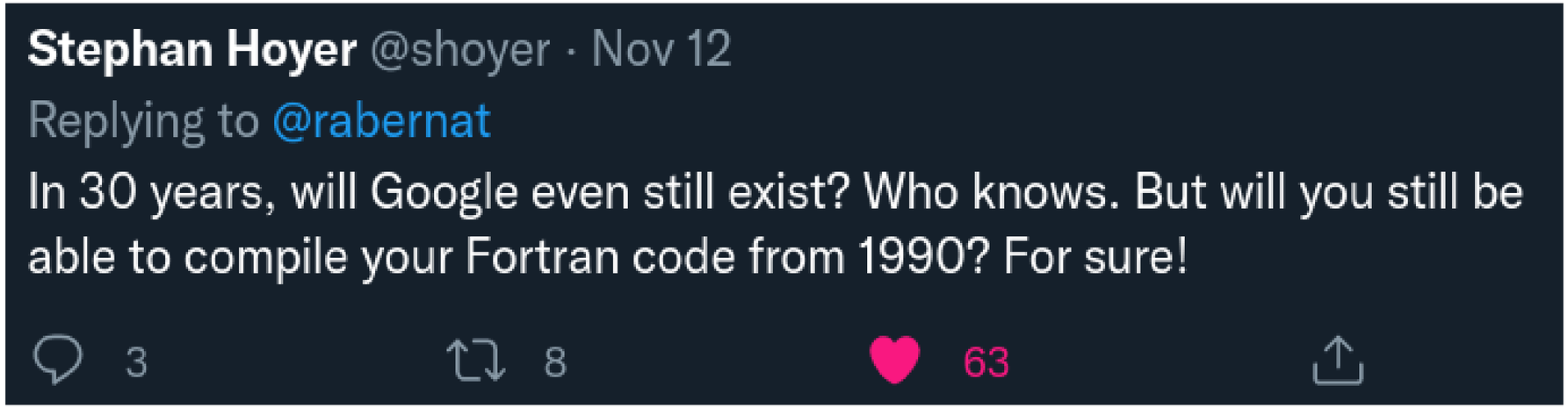
- **Compatibility (GPU)**:
  - Currently only `nvfortran` has `do concurrent` support for GPUs
  - Using `do concurrent` *loses* `gfortran` GPU support for now…
    … but may *gain* Intel GPU support with planned update to `ifort`
  - Relying on unified memory is possible performance loss (but not here)

- **Portability (CPU)**:
  - CPU multi-core parallelization was retained
    (except for *Experimental* code on `ifort`)
  - `nvfortran` and `ifort` have direct support of `do concurrent` on CPUs, while `gfortran` relies on auto-parallelization
  - Implicit reductions with `do concurrent` not supported everywhere

Discussion Continued

- **Performance**:
  - Comparable performance between stdpar and directives on CPUs/GPUs

- **Summary**:
  - **`do concurrent`** allows cleaner code and increases “future-proofiness”



  - **`nvfortran`** allowed us to eliminate *all* directives!
  - … but using a combination of directives and **`do concurrent`** yields better cross compiler/hardware compatibility

Summary/Future

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

YES! ...

… with (1) `nvfortran`, (2) NVIDIA GPUs, (3) for *some* codes (like ours)

MORE DETAILS

arxiv.org/abs/2110.10151

WACCPD 2021

DOI 10.5281/zenodo.5253520

Next step:

POT3D

github.com/predsci/POT3D

(spec.org/hpc2021)

SPOILER ALERT

It works!

(with some !$acc)

Multi-GPU with MPI (CUDA/RoCm-aware), atomics, static arrays