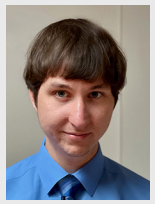
Parallelizing your Fortran Codes on CPUs and GPUs with 'do concurrent'

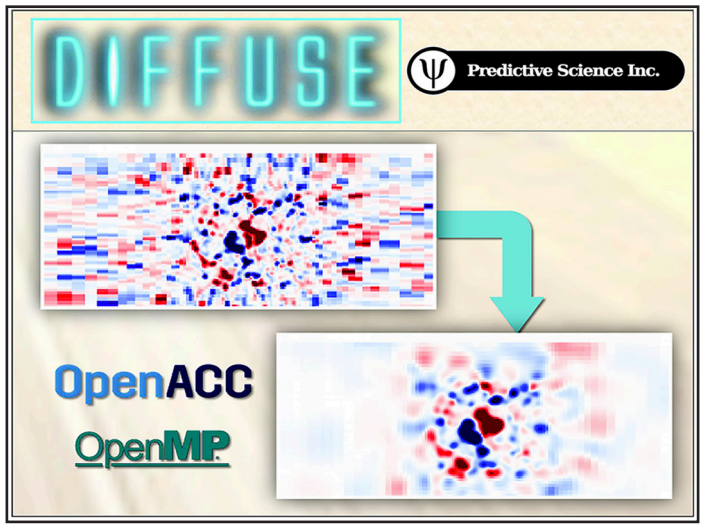


Recently, there has been growing interest in using standard language constructs such as Fortran's DO CONCURRENT loops for accelerated computing. These constructs enable parallelization of codes on multi-core CPUs and GPUs. Fortran's DO CONCURRENT construct is a simpler alternative to implement compared to directive-based APIs such as OpenMP and OpenACC. With Fortran's DO CONCURRENT construct, do loops that are parallelizable (no data dependencies between iterations) only need to be replaced with Fortran's DO CONCURRENT loops. Utilizing a specific compiler flag, these loops are recognized and parallelized. The construct approach has the potential to be more portable, and some compilers already (or have plans to) support such standards. We looked at the performance of Fortran's DO CONCURRENT and directive-based APIs with a mini-app called diffuse. We utilize NVIDIA's nvfortran compiler and compare speedups on CPUs and GPUs to the Serial CPU code. We find that the DO CONCURRENT constructs perform as well as directives in our case, which may not hold in more complicated codes, but both are significantly faster than the serial code. Fortran's DO CONCURRENT construct is a simple first step to parallelizing Fortran codes on CPUs and GPUs.

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

This research is supported by Predictive Science Inc, via NSF-NASA (AGS 202815 and 80NSSC20K1582) grants, the National Science Foundation (DUE- 1930546 and OAC 2019194) grants, the National Science Foundation XSEDE program's Pittsburgh Supercomputer Center allocation (TG-MCA03S014), and the Computational Science Research Center (CSRC) at San Diego State University

Fortran DO CONCURRENT



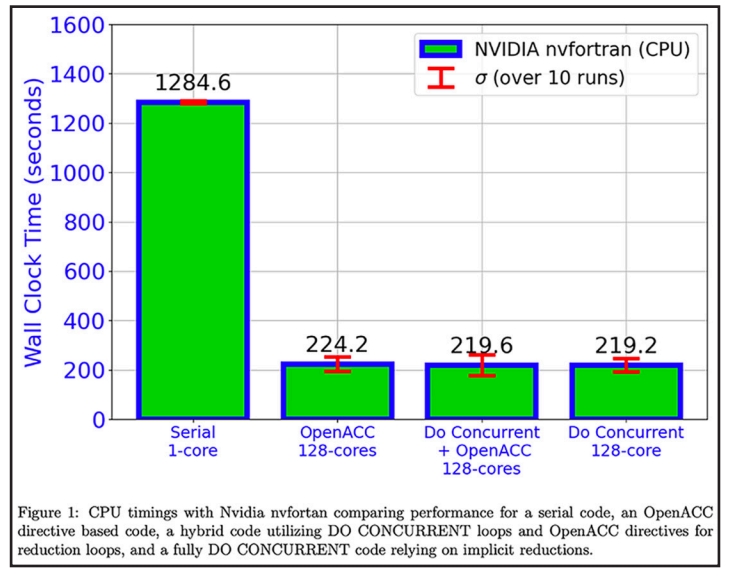


Figure 1: CPU timings with Nvidia nvfortan comparing performance for a serial code, an OpenACC directive based code, a hybrid code utilizing DO CONCURRENT loops and OpenACC directives for reduction loops, and a fully DO CONCURRENT code relying on implicit reductions.

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
```

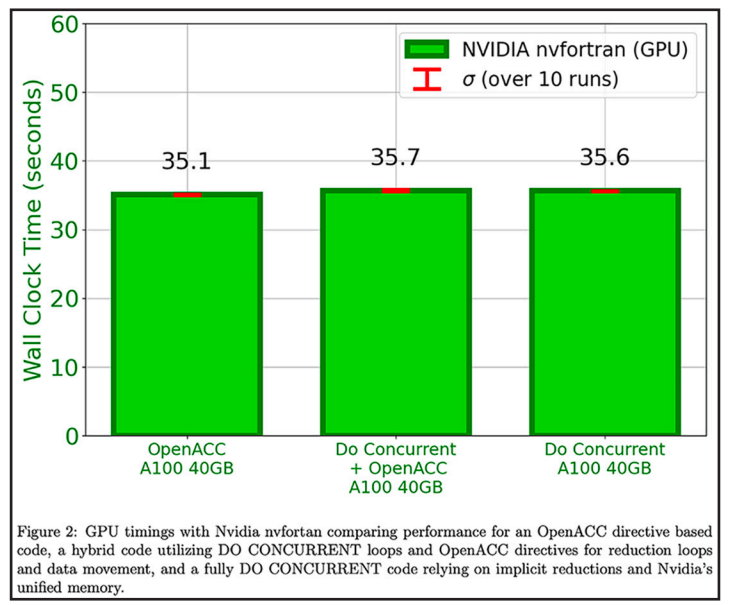


Figure 2: GPU timings with Nvidia nvfortan comparing performance for an OpenACC directive based code, a hybrid code utilizing DO CONCURRENT loops and OpenACC directives for reduction loops and data movement, and a fully DO CONCURRENT code relying on implicit reductions and Nvidia's unified memory.