Performance-Portable GPU Acceleration of the EFIT Tokamak Plasma Equilibrium Reconstruction Code

Oscar Antepara
oantepara@lbl.gov
Lawrence Berkeley National Laboratory
Berkeley, California, USA

Samuel Williams
swwilliams@lbl.gov
Lawrence Berkeley National Laboratory
Berkeley, California, USA

Scott Kruger
kruger@txcorp.com
Tech-X Corporation
Boulder, Colorado, USA

Torrin Bechtel
bechtelt@fusion.gat.com
General Atomics
San Diego, California, USA

Joseph McClenaghan
mcclenaghanj@fusion.gat.com
General Atomics
San Diego, California, USA

Lang Lao
lao@fusion.gat.com
General Atomics
San Diego, California, USA

ABSTRACT

This paper presents the steps followed to GPU-offload parts of the core solver of EFIT-AI, an equilibrium reconstruction code suitable for tokamak experiments and burning plasmas. For this work, we will focus on the fitting procedure that consists of a Grad–Shafranov (GS) equation inverse solver that calculates equilibrium reconstructions on a grid. We will show profiling results of the original code (CPU-baseline), as well as the directives used to GPU-offload the most time-consuming function, initially to compare OpenACC and OpenMP on NVIDIA and AMD GPUs and later on to assess OpenMP performance portability on NVIDIA, AMD and Intel GPUs. We will make a performance comparison for different spatial grid sizes and show the speedup achieved on NVIDIA A100 (Perlmutter-NERSC), AMD MI250X (Frontier-OLCF) and Intel PVC GPUs (Sunspot-ALCF). Finally, we will draw some conclusions and recommendations to achieve high-performance portability for an equilibrium reconstruction code on the new HPC architectures.

CCS CONCEPTS

• **Software and its engineering → Parallel programming languages**.

KEYWORDS

GPU, GPU offloading directives, OpenMP, OpenACC, performance portability

ACM Reference Format:
Oscar Antepara, Samuel Williams, Scott Kruger, Torrin Bechtel, Joseph McClenaghan, and Lang Lao. 2023. Performance-Portable GPU Acceleration of the EFIT Tokamak Plasma Equilibrium Reconstruction Code. In *Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023), November 12–17, 2023, Denver, CO, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3624062.3624607

SC-W 2023, November 12–17, 2023, Denver, CO, USA

ACM ISBN 979-8-4007-0785-8/23/11.
https://doi.org/10.1145/3624062.3624607

1 INTRODUCTION

Real-time and offline simulations are important tools to improve experimental research, and the need for faster computer simulations to keep up with real-time experiments is of great importance today with the relative improvement of HPC platforms [15–18]. The new HPC platforms recently deployed rely on GPU architectures, putting most of them on the Top500 list of fastest supercomputers [30], so fusion plasma simulations must adapt their codes to this new landscape. GPU architectures bring a notable increase of compute capabilities that make them attractive to achieve considerable speed-up for high-resolution simulations, making it possible to acquire much higher accuracy simulations to provide better experimental control in real-time, which will be essential for burning plasma experiments currently under construction. Here, we consider a fusion plasma application, EFIT-AI, that aims to improve real-time and offline simulations for tokamak experiments and burning plasmas using different computational techniques including improving the core solver code and introducing machine learning and model order reduction surrogate models [18]. The EFIT-AI core solver consists of the solution of an inverse problem by solving the Grad–Shafranov equation on a spatial grid to provide a complete equilibrium description of a plasma. The algorithmic solution involves loop iterations on a grid, where each point in space requires information on physical constraints or computation of derivatives where neighboring values are needed. Most of those time-consuming loops are typically constrained by memory bandwidth, and thus GPU-porting algorithms should exploit data reuse to exhibit high performance.

Here, we will focus on the acceleration of the core solver and detail the process of porting EFIT-AI to new HPC platforms using GPU-offloading directives, such as OpenACC and OpenMP on NVIDIA, AMD and Intel GPUs.

OpenACC and OpenMP offer a straightforward way to accelerate Fortran do-loops and handle data management with unified memory that gives us the flexibility to achieve speedup and performance portability without changing the code entirely or keeping maintenance for codes written in different languages. The porting process mainly consists of identifying the main CPU bottlenecks, changing portions of the original code to expose more parallelism in terms of do-loops, and thoroughly analyzing code performance

to avoid overheads related to data movement due to some code parts that will still be operating on the CPU.

The main goals of this effort are to review two of the most widely used GPU-offloading techniques and the caveats that we need to introduce to achieve performance on the new HPC platforms (Perlmutter-NERSC [20], Frontier-OLCF [25], and Sunspot-ALCF [1]). Moreover, we analyze the performance with same or similar pragmas for NVIDIA, AMD and Intel GPUs, ensuring performance portability.

This paper is organized as follows: Section 2 explains the main EFIT-AI core solver and describes the fitting and equilibrium reconstruction algorithm. Related work about OpenACC or OpenMP on several GPU architectures and applications is described in Section 3. Section 4 presents the GPU targets and compilers, programming models and environmental variables. Section 5 introduces the GPU-offloading directives used in this work with OpenACC and OpenMP. Section 6 reports baseline CPU timings and analyzes GPU performance for NVIDIA, AMD and Intel platforms. Finally, Section 7 draws some conclusions and recommendations for porting Fortran code to the new HPC architectures.

2 EFIT CORE SOLVER

EFIT's core solver computes the poloidal magnetic flux function ψ by solving the equilibrium Grad–Shafranov equation [6, 29] while approximately conserving the available experimental data and the imposed physics constraints:

$$\Delta^*\psi(R,Z) = -\mu_0 R J_\varphi(R,\psi), \tag{1}$$

$$J_\varphi(R,\psi) = RP'(\psi) + \frac{\mu_0 FF'(\psi)}{4\pi^2 R} \tag{2}$$

where, ψ is the poloidal magnetic flux per radian of the toroidal angle φ enclosed by a magnetic surface and $\Delta^* = R^2\nabla\times(\nabla/R^2)$. J_φ is the toroidal current density, P is pressure and $F = 2\pi RB_\varphi/\mu_0$ is the poloidal current stream function with B_φ as the toroidal magnetic field at the major radius R.

EFIT reconstructs the current source flowing within the plasma based on the external magnetic measurements and the spectroscopic and kinetic profile measurements inside the plasma by distributing the current density J_φ among grid points in a rectangular spatial (R,Z) mesh. The Grad–Shafranov equation is solved by transforming the non-linear optimization problem into a sequence of linear optimization problems using the Picard iteration scheme[15–17] where convergence is achieved by efficiently finding the solution vector that best fits the available measurements and the imposed physics and free-boundary magnetohydrodynamic equilibrium constraints. The iteration process continues until the maximum ψ residual between subsequent iterations over the grid is less than the tolerance ϵ.

EFIT is a computationally intensive code for real-time equilibrium reconstructions. On between-shot or post-shot analysis of discharges for experimental support, the fitting procedure for a time slice could converge on hundreds or thousands of iterations; thus, time per iteration becomes critical to decrease time-to-solution. Low spatial resolution grids (65×65, 129×129) are used to overcome the lack of code performance. At the same time, high-resolution grids (257×257, 513×513) are required to get more accurate information for plasma control allowing exploring new computational methods that exploit HPC systems to improve the performance of equilibrium reconstruction codes at high spatial resolution. Inside EFIT, the equilibrium reconstruction is done by the subroutine `fit_`, which consists of several function calls, such as setting up the appropriate response functions (`green_`), computation of the current density (`current_`), calculation of the poloidal fluxes (`pflux_`), computation of the dimensionless poloidal fluxes and the location of the plasma boundary (`steps_`). In EFIT-AI, depending on the shot to be solved, `fit_` could take between ten or hundreds of iterations, making `fit_` to be around 50% to 90% of time spent on a single EFIT-AI run.

We measured the time per invocation, in seconds, for the fitting call (`fit_`) on one time slice from the DIII-D shot #186610. All CPU timings for `fit_` and different grid sizes are reported in Table 1. CPU timings were collected using one CPU core on AMD EPYC 7763 (Milan) on Perlmutter, an AMD EPYC 7A53 Optimized 3rd Gen EPYC on Frontier, and an Intel Xeon "Sapphire Rapids" CPU on Sunspot.

Table 1: EFIT core solver timing per invocation for the fitting call `fit_` using one CPU.

Baseline CPU. Wall-clock time per call in seconds for `fit_` subroutine.				
	65×65	129×129	257×257	513×513
Perlmutter	0.004	0.024	0.17	1.15
Frontier	0.004	0.023	0.16	1.15
Sunspot	0.003	0.02	0.21	1.34

To accelerate `fit_`, we profiled it to pinpoint the most time-consuming inner subroutines. We used the Cray Performance Measurement and Analysis Tools [8] to identify bottlenecks in Cray systems, such as Perlmutter and Frontier. Once the main calls inside `fit_` were identified, wall clock timings were collected using `omp_get_wtime()` that is available on Perlmutter, Frontier and Sunspot. Figure 1 is a pie chart for the relative timings of different subroutines inside `fit_` for a 513×513 grid to highlight the most time consuming call for the highest resolution grid on all the machines. Close to 90% of the time in `fit_` is about computing the poloidal fluxes through `pflux_`, making this subroutine a very good candidate to be accelerated with OpenACC or OpenMP directives since most of the subroutine is based on do-loops as described in Section 5.

Table 2 also shows the percentage of time that `pflux_` represents in `fit_` where it is clear that `pflux_` dominates the fitting procedure across different grid sizes. Moreover, Table 2 presents the time per invocation for `pflux_` on the host. As the time per call is generally milliseconds to seconds, there are opportunities to accelerate dozens of loop nests. However, 10us of latency will impede acceleration of the smaller loops.

3 RELATED WORK

Prior work on GPU-acceleration of EFIT [9] leveraged CUDA to accelerate several key routines by optimizing matrix operations to solve Grad-Shafranov equation. Although broadly successful, adoption of a new programming model and language effectively forked

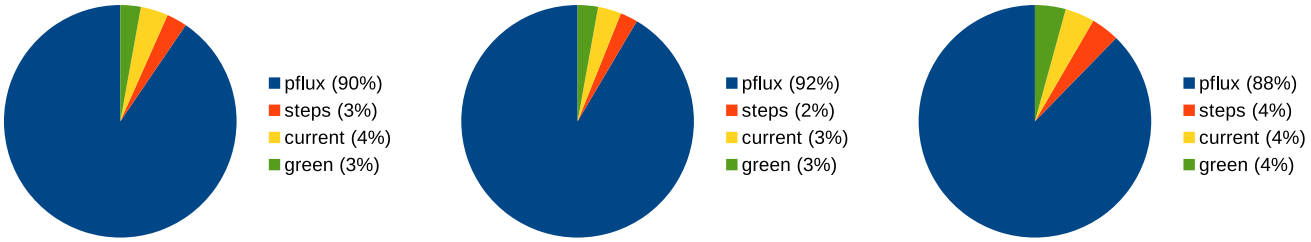

Figure 1: Breakdown of timings, in percentage, for the most expensive calls inside the `fit_` subroutine for grid size 513×513 on (left) an AMD EPYC 7763 CPU on Perlmutter, (middle) an AMD "Optimized 3rd Gen EPYC" CPU on Frontier and (right) an Intel Xeon CPU on Sunspot. As we can observe, `pflux_` dominates the fitting subroutine with a 90% of the time across all architectures.

Table 2: Time per call in seconds for `pflux_` with different grid sizes and the percentage of time that `pflux_` represents in `fit_`.

Baseline CPU. Wall-clock time per call in seconds.				
	65×65	129×129	257×257	513×513
Perlmutter time	2.4e-3	1.6e-2	1.4e-1	1.04e+0
% of `fit_`	57%	72%	84%	90%
Frontier time	2.2e-3	1.7e-2	1.4e-1	1.05e+0
% of `fit_`	61%	75%	85%	92%
Sunspot time	1.5e-3	1.2e-2	1.8e-1	1.18e+0
% of `fit_`	47%	61%	84%	88%

the code base demanding a substantial increase in software engineering efforts to keep them in sync. New parallelization strategies have been implemented in different parts of both the Fortran and CUDA branches. As such, neither is currently the ideal implementation. Ultimately, an easily maintainable, single source implementation capable of targeting CPUs or GPUs in a high-performance portable manner is required.

For a smaller code or a larger software development team, a complete rewrite in C++ to leverage executors [7] or Kokkos [4] would be a viable and attractive option. However, scope, funding, and time necessitate a more expedient solution.

Many research technologies promise performance portability across platforms. However, operators of production fusion devices will ultimately demand production software solutions that will be supported for decades (e.g. there are still Fortran 77 compilers). As such, we are focused on leveraging OpenMP and OpenACC programming models due to their support for Fortran, support across multiple vendor compilers, and robust standardization efforts.

Using GPU-offloading directive models require optimizations depending on the architectures and the devices used and is still an area of research as heterogeneous systems form part of the current and new HPC architectures [14].

GPU acceleration of a Fortran code related to quantum many-body application [5] showed several optimizations and suggestions about acceleration on NVIDIA and AMD GPUs using OpenACC or OpenMP, where no one method for reductions was best in all situations. A proxy for LAMMPS, was accelerated using OpenMP and several optimizations were evaluated using the roofline model on earlier GPUs [19]. As most of the kernels studied were intended to be compute bound, the roofline model showed that the kernels were memory bound as data reuse and locality become imperative on GPUs. Ibrahim et al. [10] investigated the performance portability of sparse block diagonal matrix times multiple vectors (SpMM) using CUDA, HIP, OpenACC and Kokkos implementations on NVIDIA and AMD GPUs, where native implementations as CUDA and HIP achieved the highest speedup. However, OpenACC presented a competitive performance on NVIDIA A100 GPU but a poor performance on AMD MI250X GPU. Compared to prior studies applied to gpu-offloading using OpenACC and/or OpenMP, the work in this paper is distinguished by looking at recent GPUs and providing suggestions and the experiences on performance analysis and optimizations for performance portable GPU directives for a fusion plasma application with a code written in Fortran.

4 EXPERIMENTAL SETUP

In this paper, we evaluate our performance portability efforts using three GPU-based systems — NVIDIA A100 GPU in NERSC's Perlmutter [20], AMD MI250X GPU in OLCF's Frontier system [25] and Intel PVC GPU in ALCF's Sunspot test system [1]. Perlmutter is the new HPE Cray EX supercomputer at NERSC. The GPU partition includes over 1500 GPU-accelerated nodes and 35PB of all-flash storage. Each GPU node consists of a single socket of an AMD EPYC 7763 (Milan) processor and four NVIDIA Ampere A100 GPUs. Frontier is the new OLCF exascale system. Frontier computer nodes consist of one 64-core AMD EPYC 7A53 (an optimized 3rd Gen EPYC) CPU and four AMD MI250X GPUs. Sunspot is a test system with identical hardware to the new Aurora exascale system. Sunspot computer nodes consist of a pair of Intel Xeon CPUs based on Sapphire Rapids architecture and six Intel Xe GPUs based on Ponte Vecchio (PVC) architecture.

In all of our experiments, we use only one GPU as EFIT's typical usage will MPI parallelize multiple time steps across multiple cores (or GPUs in an accelerated framework). As Perlmutter and Frontier machines use a similar node architecture with one AMD EPYC 64-core CPU connected to four GPUs (technically, there are 8 Graphical Compute Devices on Frontier), the nominal threshold for positive acceleration is 16× (64 cores / 4 GPUs) on Perlmutter and 8× (64 cores / 8 GCDs) on Frontier. On Sunspot, one node contains two Intel CPU Xeon, each CPU with 54 cores, and 6 PVC GPUs, each one containing two stacks, which means a nominal threshold of

acceleration of about 8.7×. These speedups represent the minimum for performance optimization.

4.1 GPU Architectures

NVIDIA A100: is NVIDIA's latest GPU [21]. It instantiates 108 streaming multiprocessors (SM) each with four warp schedulers of 16 integer units and 8 double-precision floating point units. Although there is also a double-precision tensor core, it is unlikely to be exploited in EFIT. As such, the GPU provides a peak performance of about 9.7 TFLOP/s in double-precision. The SMs each include a 192KB shared memory/data cache and share a 40MB L2 cache and 40GB of HBM accessible at 1.5TB/s. Although the four GPUs are interconnected via NVLINK, they are individually connected to the CPU with a PCIe 4.0 ×16 link providing at most about 32GB/s. The high discrepancy between HBM bandwidth and PCIe bandwidth demands at least 50× reuse of data lest the GPU become PCIe-bound.

AMD MI250X: is AMD's latest compute oriented GPU [2]. It instantiates two Graphical Compute Dies (GCDs) each of 110 compute units (CU). Each CU includes four 16-wide 64b SIMD units to execute either integer or floating-point instructions and a small L1 cache. Each GCD also includes a 8MB L2 cache, provides a peak FP64 performance of about 24TFLOP/s, and is connected to 4 HBM stacks of 64GB providing 1.6TB/s. Each GCD is connected to the host CPU via AMD's infinity fabric and is subject to the same requirements on data locality. As the GCD forms the nominal programmable device, we use it as the basis of our performance analysis.

Intel Xe PVC: is Intel's latest compute oriented GPU [12]. It contains two tiles/stacks with a total of 1024 execution units (EUs). Each EU has a 512b SIMD width to execute either integer or floating-point instructions. EUs are grouped together into a Xe-core with a shared cache. 16x Xe-core form a Slice and 64x Xe-core or 4x Slice form a Stack, providing 15TFLOP/s peak FP64 performance and 1.3TB/s HBM bandwidth. As one stack forms the nominal programmable device, we use it as the basis of our performance analysis. Thus, compared to A100 and one GCD MI250X, a Intel PVC "stack" provides 1.5× more peak FLOP rate than A100 and 0.6× less FLOP rate than one GCD MI250X, comparable bandwidth, and comparable host connectivity.

As a good practice, running one process per GCD MI250X or one PVC stack is recommended. In this paper, our tests presented use either one A100 GPU on Perlmutter or one MI250X GCD on Frontier or one PVC GPU stack on Sunspot.

4.2 Programming Models and Compilers

At the core of this paper is an effort to assess the performance portability and ultimate potential of GPU acceleration using two different directive-based programming models: OpenACC and OpenMP target. For OpenACC or OpenMP, different compiler flags have been added to the EFIT building system to consider the architectures requirements and enable unified memory for data management if it is possible for the architecture. Table 3 enumerates the compiler, version, flags and environment variables we used on our respective test systems.

Perlmutter features the programming environment NVIDIA HPC SDK (Software Development Kit) [22] to support diverse parallel programming models such as OpenMP and OpenACC for our Fortran code. On the other hand, the Frontier system provides Cray compilers through modules and Cray Programming Environment [24]. On Sunspot, Intel oneAPI [13] provides the libraries and compilers required to do GPU-offloading on Intel GPUs for OpenMP and Fortran codes.

GPU data movement presented in Section 6 was collected using GPU profilers as NVIDIA Nsight Compute[23] on NVIDIA GPUs, AMD rocProf[3] on AMD GPUs, and Intel Advisor[11] on Intel GPUs. More details about command lines and formulas to compute GPU data movement are in the Appendix A.

Finally, GPU porting efforts in this paper are based on using Unified Memory. Since NVIDIA and AMD GPUs have this feature available trough compiler flags(`-gpu=managed`) or environmental variables (`CRAY_ACC_USE_UNIFIED_MEM,HSA_XNACK`), as described in Table 3, data transfer between host and device are opportunistically managed by the accelerator library. However, on Intel this feature is not available yet, which means the user must ensure the most efficient way to handle data transfers according to their own application.

5 PORTABLE GPU-ACCELERATION

In this section, we will describe the implementations done for OpenACC and OpenMP to accelerate do-loops that have been found in the code. Most of the GPU code lines introduced are related to directives for loops that go over the grid to do array computations on the inner or the boundary elements. Most of the do-loops found in the fitting function are of $O(N^2)$ complexity, where N is the number of total elements(e.g., 65, 129, 257, 513), and $O(N^3)$ for specific nested loops that operate over boundary elements that require information of inner elements over the grid, making $O(N^3)$ loops the most time consuming operations in the code.

5.1 OpenACC Implementation

Following OpenACC best practices manual and examples [26] to accelerate codes using GPUs, we have added OpenACC directives on most of the do-loops inside the `pflux_` subroutine call. Figure 2 shows an example of the directives introduced to accelerate a loop of $O(N^3)$ complexity that is one of the most expensive calls. As shown in this example, the proposed directives should make reductions and be optimized for vector operations on the GPU; thus, `num_workers` and `vector_length` are required to get the most performant timings for this kernel. Depending on the OpenACC specification, some features will be available for a GPU vendor; as such, we will focus on OpenACC 2.7 specification for NVIDIA GPUs and OpenACC 2.0 specification for AMD GPUs. See [27] for more details about the specifications.

In Table 4, we show all the OpenACC directives and the number of code lines introduced to accelerate the code and its percentage compared to the total number of code lines for the subroutine being accelerated, showing the potential of accelerating legacy code without over manipulating or totally changing it and with a minimal number of new code lines.

Table 3: Compiler versions, flags and environment variables used for OpenMP and OpenACC on Perlmutter-NERSC, Frontier-OLCF and Sunspot-ALCF.

HPC System	GPU offload	Compiler version	Compiler flags	Env. variables
Perlmutter-NERSC	OpenMP	NVHPC 22.7, CUDAToolkit 11.7, PrgEnv-nvidia/8.3.3	-mp=gpu -gpu=cc80,managed	
	OpenACC	NVHPC 22.7 CUDAToolkit 11.7, PrgEnv-nvidia/8.3.3	-acc -gpu=cc80,managed	
Frontier-OLCF	OpenMP	HPE/Cray PrgEnv 8.3.3, CCE 14.0.2, ROCm 5.2.0	-h omp -hsystem_alloc	CRAY_ACC_USE_UNIFIED_MEM=1, HSA_XNACK=1, CRAY_MALLOPT_OFF=1
	OpenACC	HPE/Cray PrgEnv 8.3.3 CCE 14.0.2, ROCm 5.2.0	-h acc -hsystem_alloc	CRAY_ACC_USE_UNIFIED_MEM=1, HSA_XNACK=1, CRAY_MALLOPT_OFF=1,
Sunspot-ALCF	OpenMP	oneapi/eng-compiler/ 2023.05.15.003	-fiopenmp -fopenmp-targets=spir64	

```
!$acc parallel loop gang worker num_workers(
  numWork) vector_length(vecLen)
do j=1,nh
  kk=(nw-1)*nh+j
  tempsum1=0.
  tempsum2=0.
  !$acc loop vector reduction(+:tempsum1,
  tempsum2)
  do ii=1,nw
   do jj=1,nh
    kkkk=(ii-1)*nh+jj
    mj=abs(j-jj)+1
    mk=(nw-1)*nh+mj
    tempsum1=tempsum1-gridpc(mj,ii)*pcurrt(kkkk)
    tempsum2=tempsum2-gridpc(mk,ii)*pcurrt(kkkk)
   enddo
  enddo
  psi(j)=tempsum1
  psi(kk)=tempsum2
enddo
```

Figure 2: OpenACC implementation of a $O(N^3)$ loop to compute a poloidal flux at the boundary elements.

Table 4: OpenACC directives and number of times that the directive was used over `pflux_`.

OpenACC directives	Number of lines in the code (% of the total code lines in the subroutine)
!$acc kernel	4 (1.0%)
!$acc end kernel	4 (1.0%)
!$acc parallel loop gang worker	2 (0.5%)
!$acc loop vector reduction	2 (0.5%)

5.2 OpenMP Implementation

For OpenMP, we use OpenMP 4.5 and some directives available from OpenMP 5.0 [28] for NVIDIA, AMD and Intel GPUs.

Figure 3 shows an example of the same piece of code that was accelerated using OpenACC, in the last section, but now with OpenMP directives. One can observe similarly parallelized loops. However, OpenMP requires defining the reduction variables on both directives, since the reduction should happen across threads and GPU blocks. Another difference is that OpenMP requires the use of "`target teams distribute`" as a compiler indication to gpu-offload the loop, instead of a simple "`parallel do`" that is used to parallelize the loop over CPUs threads. Furthermore, on OpenMP implementations, we are using the `collapse` clause, collapsing two loops to enable parallelization of N×N iterations that is critical for good utilization of GPUs.

Table 5 summarizes the OpenMP lines used for GPU acceleration that map precisely to the OpenACC lines in Table 1. In this way, it is easier for the reader to compare and have a code translation from OpenACC to OpenMP and vice-versa.

Table 5: OpenMP directives and number of times that the directive was used over `pflux_`.

OpenMP directives	Number of lines in the code (% of the total code lines in the subroutine)
!$omp target teams distribute parallel do collapse(2)	4 (1.0%)
!$omp target team distribute reduction	2 (0.5%)
!$omp parallel do reduction collapse(2)	2 (0.5%)

6 PERFORMANCE PORTABILITY RESULTS

EFIT is an equilibrium fitting code that provides a complete equilibrium description of a plasma. Several input parameters are needed

```
!$omp target teams distribute reduction(+:
  tempsum1,tempsum2)
do j=1,nh
  kk=(nw-1)*nh+j
  tempsum1=0.
  tempsum2=0.
  !$omp parallel do reduction(+:tempsum1,
  tempsum2) collapse(2)
  do ii=1,nw
   do jj=1,nh
    kkkk=(ii-1)*nh+jj
    mj=abs(j-jj)+1
    mk=(nw-1)*nh+mj
    tempsum1=tempsum1-gridpc(mj,ii)*pcurrt(kkkk)
    tempsum2=tempsum2-gridpc(mk,ii)*pcurrt(kkkk)
   enddo
  enddo
  psi(j)=tempsum1
  psi(kk)=tempsum2
enddo
```

Figure 3: OpenMP implementation of a $O(N^3)$ loop to compute a poloidal flux at the boundary elements.

to run an equilibrium fitting test, and are described as the measurement values for diagnostics being fit corresponding to the chosen shot and time. The tests in this section are looking at shot #186610, at 2.4s into the discharge, from a DIII-D experiment and to measure performance, we run the code with four grid sizes (65×65, 129×129, 257×257 and 513×513) until the fitting is good enough that ϵ is less than 1e-5, as described in Section 2. Baseline CPU timings are measured with the original code, as we start to implement GPU-offload directives, general optimizations are applied to code. Optimization as avoiding array sections on Fortran "(:)" and doing reductions on scalar variables instead of array reductions, if possible. By doing reductions on scalar variables, required for better GPU performance, improved the performance on only CPU by 3×. For consistency, the results and speed-up presented in this section are computed against Baseline CPU (original code) and for completeness, baseline CPU, optimized CPU and GPU implementations are plotted in Figure 7 for Perlmutter, Frontier and Sunspot.

6.1 Performance and Portability of OpenACC

Table 6 shows the time and benefit (acceleration vs. a single CPU core) for `pflux_` when using OpenACC and the associated code modifications needed for reductions on NVIDIA and AMD GPUs. As expected from loop nests with O(N^2) and O(N^3), larger grids see increased acceleration. However, AMD sees acceleration saturate around 257×257 grids, whereas NVIDIA continues to see increased acceleration (see Figure 7 for speed-up achieved by OpenACC). In fact, the nearly 4× and nearly 8× increase in run times when doubling the grid dimension suggests that the NVIDIA architecture is still dominated by the O(N^2) while the AMD architecture is dominated by the O(N^3) loop nests — possibly extracting insufficient parallelism from the OpenACC pragmas.

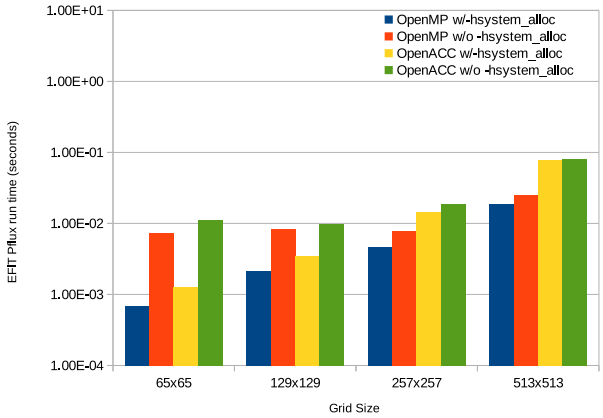

Figure 4: Timing in seconds for `pflux_` using OpenACC or OpenMP with and without `-hsystem_alloc` on AMD MI250X GPU (single GCD). We can observe that using the Cray compiler flag to use system configurations for memory allocations improves the GPU performance for small size problems up to 10×.

OpenACC on AMD underperformed compared to Perlmutter for small problem size as the data management for small size problems becomes expensive. As some parts of `fit_` still execute on the CPU (necessitating copies between the host and the device), the poor performance of this routine was exacerbated on small problems and thus penalized overall run time and acceleration. Here, it is important to note that the best performance achieved on AMD was with CRAY_MALLOPT_OFF=1 and the flag *–hsystem_alloc*, where the run-time for small size problems got between 10× to 2× faster by using the system default mallopt for OpenACC. We can observe the boost on speedup in Figure 4, where it is clear that the use of *–hsystem_alloc* improves the performance on AMD, using Cray compilers on almost all grid sizes, mainly on small size problems, and independently if is OpenACC or OpenMP. For completeness' sake, the flag *–hsystem_alloc* sets the system default mallopt parameters instead of the compiler default parameters that control the behavior of the memory-allocation functions.

For the finest grids, we can take a closer look to the O(N^3) loop nest by analyzing GPU data movement on the 513×513 grid. Figure 5 shows that OpenACC on AMD moves close to 3.7× more data from GPU HBM compared to OpenACC implementations on NVIDIA, indicating a lack of data reuse from the OpenACC directives compared to the same code on NVIDIA GPUs. OpenACC data on Intel GPUs are not available since there are no OpenACC compilers supporting Intel GPUs.

Further optimization could be realized by explicitly setting the vector length and the number of workers. However, the optimal values for these parameters are accelerator-specific and thus defy our performance portability goals. For the results presented in this paper, `num_workers` was set to 4 and `vector_length` was set to 32 (the warp size for the NVIDIA A100). For OpenACC tests on Frontier, we kept `num_workers` equal to four and modified the `vector_length` to 64 to match the warp size for AMD GPUs.

Table 6: Time per call and speedup vs. a single CPU core for `pflux_` with OpenACC acceleration directives on NVIDIA and AMD GPUs. Observe AMD acceleration with OpenACC saturates at 257×257 grids and generally underperforms NVIDIA.

	Grid Size			
GPU	65×65	129×129	257×257	513×513
NVIDIA time (s)	9.10e-4	1.80e-3	4.45e-3	1.63e-2
speedup	2.4×	10×	31×	65×
AMD time (s)	1.6e-3	3.4e-3	1.2e-2	8.4e-2
speedup	1.4×	5×	12×	13×

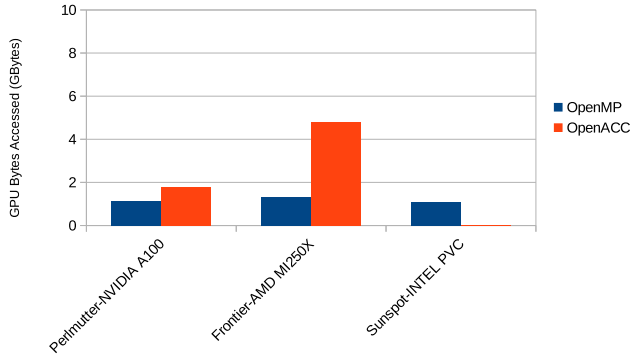

Figure 5: GPU data movement of the most expensive kernel, loop nests with O(N^3) for the 513 × 513 test, in `pflux_` using OpenMP or OpenACC on NVIDIA A100 GPU, OpenMP or OpenACC on AMD MI250X GPU (single GCD), and OpenMP on Intel PVC GPU (single stack). Note that OpenMP is the most efficient gpu-offload directive related to GPU data movement across all GPU architectures. OpenACC moves 1.6× more data than OpenMP on NVIDIA and 3.7× more data than OpenMP on AMD GPUs.

From Figure 1, Amdhal's law nominally limits `fit_` to a 16× speedup (infinite speedup of `pflux_`). From Table 6, we can infer that after acceleration, `pflux_` has been reduced to 10% of `fit_`'s run time on NVIDIA GPUs and closer to 50% on AMD GPUs. Clearly, continued acceleration of EFIT on NVIDIA GPUs must focus on the other routines in `fit_` as further acceleration of `pflux_` will only provide asymptotic benefits. Conversely, `pflux_` will require more adaptations or compiler updates that will optimize data movement for OpenACC on AMD GPUs.

6.2 Performance and Portability of OpenMP

Table 7 shows the time and benefit (acceleration vs. a single CPU core) for `pflux_` when using OpenMP and the associated code modifications needed for reductions on NVIDIA, AMD and Intel GPUs. As with OpenACC, acceleration increases with grid size. Concurrently, run time trends towards O(N^2) on NVIDIA, AMD and Intel GPUs. This suggests the OpenMP compiler for AMD much more effectively exploits parallelism in the O(N^3) loop nests ensuring they do not dominate the run time. We can revisit the GPU data movement for the O(N^3) loop nest on the 513×513 test in Figure 5. OpenMP is moving a similar amount of data from HBM on NVIDIA, AMD and Intel, providing better performance portability across architectures.

NVIDIA OpenMP run time nearly perfectly matches NVIDIA OpenACC run time. Conversely, AMD OpenMP performance is substantially faster than AMD OpenACC performance — nearly 4× for the largest grid. Nevertheless, AMD GCD performance still lags behind NVIDIA A100 performance by about 1.5× for larger grids despite superior compute and comparable bandwidth and host connectivity. It is important to notice that OpenMP implementation on AMD GPUs gets the benefit from using "`!$omp loop`" directives on O(N^3) loop nests, and also by using *–hsystem_alloc* as in Figure 4, where speedup on the smallest grids are nearly 10× faster with the system allocation flags with the Cray compiler.

Intel OpenMP required additional clauses to achieve the performance reported in Table 7. As on Intel GPU, explicit clauses for data movement between host and GPU are not required to ensure that the application runs and gives the right answer. The same GPU-offload directives used for NVIDIA or AMD are applicable on Intel GPUs. However, the compiler creates an executable that incurs on continue copies of data from host to GPUs and vice-versa, even if loop nests are one after the other and data should not be moved continuously from GPU to the host. To avoid unnecessary copies, the directive "`!$omp target data map (to:) (from:)`" is used to ensure the compiler instructs the copies at that directive and at the end with "`!$omp end target data`". In Table 7, Intel OpenMP underperforms on all grid sizes compared to NVIDIA or AMD, which clearly indicates that more optimizations are required to avoid unnecessary copies from host to GPUs and vice-versa.

Ultimately, on a system like Perlmutter, one must attain an acceleration of 16× (GPU vs. single core) for the 4 GPUs to be faster than the 64 host cores. We see that OpenMP reaches this threshold for the 257×257 and 513×513 grids. Conversely, Frontier needs only an 8× acceleration for the 8 GCDs to deliver greater throughput than the 64-core host. We see the AMD GPU attain this at 129×129 and larger grids. Sunspot, with 6 GPUs and two stacks per each GPU, needs 8.7× acceleration to be faster than the 104 total host cores, which is reached for 257×257 and 513×513, same as Perlmutter. As such, the overall throughput of a Frontier node is higher than that of a Perlmutter or a Sunspot node despite the lower per-device performance compared to Perlmutter. Although, Frontier has a higher per-device performance compared to Sunspot.

Figure 6 shows that the fraction of `fit_` time spent in `pflux_` has been reduced to 16% on NVIDIA GPU, 27% on AMD GPU and 44% on Intel GPU using OpenMP. As such, further acceleration of `fit_` can only be realized through the acceleration of the other three principle routines on NVIDIA and AMD. However, further optimizations on Intel GPUs are required to minimize data movement from the host to the device.

7 DISCUSSION AND CONCLUSIONS

In this paper, we evaluated the suitability of OpenACC and OpenMP for providing performance portability when accelerating reconstruction of tokamak plasma fusion equilibria. To that end, we GPU-accelerated the most expensive function in the core solver of EFIT, a Fortran-based equilibrium reconstruction code, in both

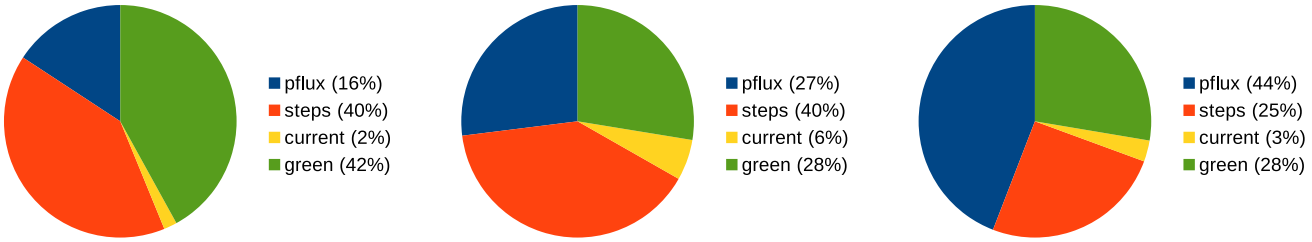

Figure 6: Pie-chart about relative timings for `fit_` subroutine with OpenMP GPU-accelerated directives on (left) NVIDIA A100 GPU, (middle) single GCD on AMD MI250X GPU and (right) single stack on Intel PVC GPU for grid size 513×513. Notice that GPU-offload acceleration provides a significant speedup for `pflux_`, reducing its contribution from 90% to under 50% on all architectures.

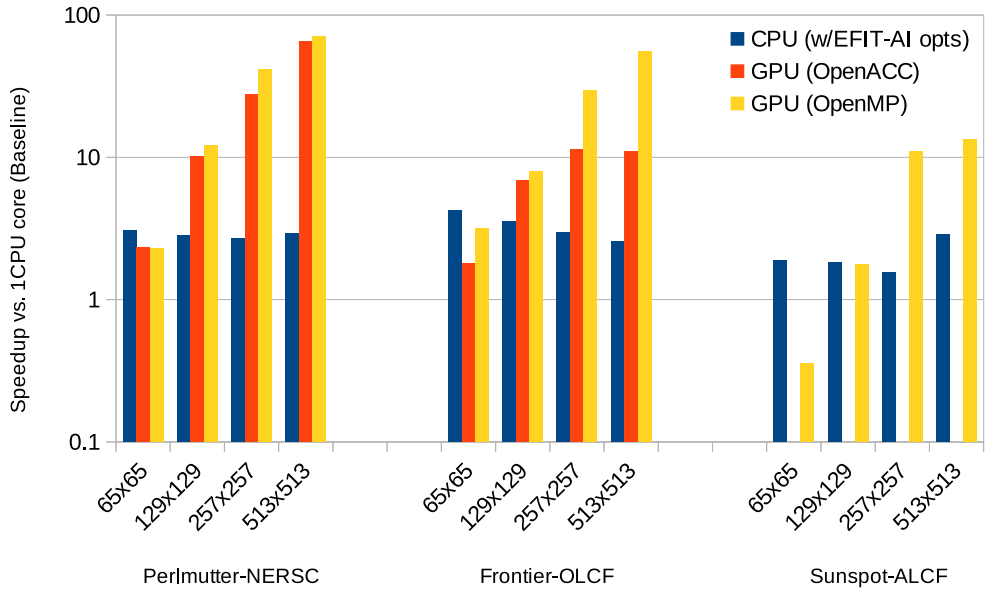

Figure 7: Speedup for `pflux_` compared with a single CPU core (Baseline) with all the optimizations and OpenMP or OpenACC on Perlmutter with AMD CPUs and NVIDIA A100 GPUs, Frontier with AMD CPUs and AMD MI250X GPUs (single GCD), and Sunspot with Intel CPUs and Intel PVC GPUs (single stack). Note that OpenMP achieves the highest speed-up across all architectures, where GPU acceleration becomes significant on the finest grid size tests.

OpenACC and OpenMP and evaluated performance as a function of grid size on NVIDIA A100, AMD MI250X and Intel PVC GPUs. Figure 7 shows a final summary about the speed-up achieved on all GPU architectures and grid sizes. Acceleration, relative to an AMD EPYC CPU, ranged from 2× to 70× on the NVIDIA GPU for both programming models, indicating both the importance of coarse-grained offloading and the support for NVIDIA GPUs across multiple programming models. Conversely, AMD GPU acceleration topped out at 13× and 56× for OpenACC and OpenMP respectively — a clear indication of momentum behind OpenMP compilers.

Nevertheless, AMD GCDs slightly underperformed NVIDIA GPUs on large problems despite their advantage in computing power and parity in bandwidth and host connectivity. This, coupled with the nearly identical NVIDIA OpenMP and OpenACC performance, suggests compiler and runtime development work is still needed if Frontier is to live up to its potential.

On Sunspot, acceleration relative to their Intel CPU, was achieved at most 13× using OpenMP. Indicating that more optimization work is required from the user to avoid unnecessary copies from host to device and vice-versa. Nevertheless, OpenMP still holds with significant performance for high resolution grids, at least for 257×257 and 513×513, independently of the architecture.

In terms of productive performance portability, we observe that modification of only eight lines of source code (roughly 2% of the routine) allowed close to 70× reduction in run time of the dominate subroutine within the fitting routine on NVIDIA and AMD, and close to 13× on Intel architectures. As a result, the `pflux_` routine went from 90% of `fit_` time to 16% on NVIDIA, 27% on AMD

Table 7: Time per call and speedup vs. a single CPU core for `pflux_` with OpenMP acceleration directives on NVIDIA and AMD GPUs. Unlike OpenACC, AMD acceleration continues to increase on larger grids attaining over 70% of the performance of the NVIDIA GPU. Intel acceleration underperforms as further optimization is required to minimize data movement from host to device.

	Grid Size			
GPU	65×65	129×129	257×257	513×513
NVIDIA time (s)	1.05e-3	1.39e-3	3.42e-3	1.48e-2
speedup	2×	11×	41×	70×
AMD time (s)	6.9e-4	2.16e-3	4.6e-3	1.89e-2
speedup	3×	8×	30×	56×
INTEL time (s)	4.2e-3	6.73e-3	1.6e-2	8.84e-2
speedup	0.35×	2×	11×	13×

and 44% on Intel. Further GPU acceleration of EFIT will require similar optimization of the other routines in `fit_`, implementing new algorithms that are more suitable for GPUs, and more user-defined optimizations on Intel GPUs.

ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Fusion Energy Sciences, using the DIII-D National Fusion Facility, a DOE Office of Science user facility, under Award(s) DE-FC02-04ER54698, DE-AC02-05CH11231 and DE-SC0021203. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231, resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725, and resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

DISCLAIMER

A GPU PROFILING AND DATA COLLECTION

The paper focuses on the performance-portable GPU acceleration of an equilibrium reconstruction code for burning plasma on NVIDIA A100, AMD MI250X and Intel PVC GPUs, as well as, different GPU-offload directives models such as OpenACC and OpenMP. Here, we describe the hardware and software used to GPU accelerate the most time consuming loops and the methodology to extract the data needed for GPU data movement included in the paper.

Machines: Results presented in this paper were obtained on a NVIDIA A100 GPU on Perlmutter at NERSC, AMD MI250X GPU on Frontier at OLCF and Intel PVC GPU on Sunspot at ALCF. In all experiments, we use only a single process running on one A100 GPU, one GDC on MI250X and one stack on Intel PVC GPU.

In this section we describe the command lines given to the profiler to gather GPU metrics for the GPU data movement on the figures described in the paper.

A.1 Perlmutter-NERSC

On Perlmutter-NERSC, NVIDIA Nsight Compute [23] command line to gather GPU metrics for double precision is depicted below:

```
nv-nsight-cu-cli -k <kernel_name> --metrics
"dram__bytes.sum" <exe> <params>
```

The metric `dram__bytes.sum` gives the value for GPU data movement presented in the paper. Another NVIDIA profiler to get the roofline plots, and GPU data movement is NVIDIA Nsight Compute by using the next command line:

```
srun -n 1 ncu -o output_file --set full <exe> <params>
```

where results in the *output_file* can be visualized using the NVIDIA Nsight Compute API on your local machine. In the report, GPU data movement can be found in the `Memory Workload Analysis`.

A.2 Frontier-OLCF

AMD-rocProf [3] is the profiler available on Frontier-OLCF to collect GPU metrics. The command line used is:

```
rocprof -i  input_file.txt --timestamp on
-o my_output.csv <exe> <params>
```

AMD ROCm Profiler needs an input file with the kernel name and the metrics to be collected. An example of the input file is showed below:

```
kernel: <kernel_name>
pmc : SQ_INSTS_VALU_ADD_F16 SQ_INSTS_VALU_MUL_F16
SQ_INSTS_VALU_FMA_F16 SQ_INSTS_VALU_TRANS_F16
pmc : SQ_INSTS_VALU_ADD_F32 SQ_INSTS_VALU_MUL_F32
SQ_INSTS_VALU_FMA_F32 SQ_INSTS_VALU_TRANS_F32
pmc : SQ_INSTS_VALU_ADD_F64 SQ_INSTS_VALU_MUL_F64
SQ_INSTS_VALU_FMA_F64 SQ_INSTS_VALU_TRANS_F64
pmc : SQ_INSTS_VALU_MFMA_MOPS_F16
SQ_INSTS_VALU_MFMA_MOPS_BF16
SQ_INSTS_VALU_MFMA_MOPS_F32
SQ_INSTS_VALU_MFMA_MOPS_F64
pmc : TCC_EA_RDREQ_32B_sum TCC_EA_RDREQ_sum
TCC_EA_WRREQ_sum TCC_EA_WRREQ_64B_sum
gpu: 0
```

To compute GPU data movement, we use the `rocprof` metrics:

`TCC_EA_WRREQ_64B, TCC_EA_WRREQ_sum, TCC_EA_RDREQ_32B, and TCC_EA_RDREQ_sum.`

So we compute GPU data movement with the next formula:

```
GPU Bytes Moved = 64* TCC_EA_WRREQ_64B +
32*(TCC_EA_WRREQ_sum - TCC_EA_WRREQ_64B)
+ 32*TCC_EA_RDREQ_32B +
64*(TCC_EA_RDREQ_sum - TCC_EA_RDREQ_32B).
```

More information can be found here: https://docs.olcf.ornl.gov/systems/frontier_user_guide.html#getting-started-with-the-rocm-profiler.

A.3 Sunspot-ALCF

Intel provides several tools for GPU profiling. In this work, we have used Intel Advisor [11] as the GPU metrics collector for GPU data movement. To profile our application with Intel Advisor, we used the following command line:

```
ZE_AFFINITY_MASK=0.0 advisor --collect=roofline
--profile-gpu -- <exec> <params>
```

where *ZE_AFFINITY_MASK* = 0.0, is at the beginning of the line to run our executable with one stack. Intel Advisor will collect the information in a directory that will be needed to create a html report with the next line:

```
advisor --report=all --project-dir=.
--report-output=roofline.html
```

The *html* file can be opened with a web browser and it contains general information as data movement across the memory hierarchy, FLOP count, instructions executed, etc. For a specific kernel GPU data movement can be found in the tab `GPU memory`.

REFERENCES

[1] ALCF. 2023. ALCF: Sunspot GPU Nodes. https://www.alcf.anl.gov/support-center/aurorasunspot/getting-started-sunspot

[2] AMD. 2022. AMD CDNA 2 ARCHITECTURE. https://www.amd.com/system/files/documents/amd-cdna2-white-paper.pdf.

[3] AMD. 2023. AMD rocProf Documentation. https://docs.amd.com/projects/rocprofiler/en/docs-5.1.0/rocprof.html.

[4] H. Carter Edwards, Christian R. Trott, and Daniel Sunderland. 2014. Kokkos. *J. Parallel Distrib. Comput.* 74, 12 (dec 2014), 3202–3216. https://doi.org/10.1016/j.jpdc.2014.07.003

[5] Brandon Cook, Patrick J. Fasano, Pieter Maris, Chao Yang, and Dossay Oryspayev. 2022. Accelerating Quantum Many-Body Configuration Interaction with Directives. In *Accelerator Programming Using Directives*, Sridutt Bhalachandra, Christopher Daley, and Verónica Melesse Vergara (Eds.). Springer International Publishing, Cham, 112–132.

[6] H. Grad and H. Rubin. 1958. Hydromagnetic equilibria and force-free fields. *Proc. 2nd UN Conf. on the Peaceful Uses of Atomic Energy* 31 (1958), 190.

[7] J. Hoberock, M. Garland, C. Kohlhoff, C. Mysen, C. Edwards, G. Brown, D. Hollman, L. Howes, K. Shoop, L. Baker, and E. Niebler. 2020. P0443r10: A unified executors proposal for C++ (Jan 2019). https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2020/p0443r12.html#proposed-wording.

[8] HPE. 2023. Cray Performance Measurement and Analysis Tools User Guide 6.5.0 S-2376. https://support.hpe.com/hpesc/public/docDisplay?docId=a00113916en_us&page=index.html

[9] Y. Huang, Z.P. Luo, B.J. Xiao, L.L. Lao, A. Mele, A. Pironti, M. Mattei, G. Ambrosino, Q.P. Yuan, Y.H. Wang, and N.N. Bao. 2020. GPU-optimized fast plasma equilibrium reconstruction in fine grids for real-time control and data analysis. *Nuclear Fusion* 60, 7 (jun 2020), 076023. https://doi.org/10.1088/1741-4326/ab91f8

[10] Khaled Z. Ibrahim, Chao Yang, and Pieter Maris. 2022. Performance Portability of Sparse Block Diagonal Matrix Multiple Vector Multiplications on GPUs. In *2022 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*. 58–67. https://doi.org/10.1109/P3HPC56579.2022.00011

[11] INTEL. 2023. INTEL Advisor tool on Sunspot-ALCF. https://www.intel.com/content/www/us/en/developer/tools/oneapi/advisor.html.

[12] INTEL. 2023. INTEL IRIS XE GPU ARCHITECTURE. https://www.intel.com/content/www/us/en/docs/oneapi/optimization-guide-gpu/2023-0/intel-iris-xe-gpu-architecture.html.

[13] INTEL. 2023. INTEL oneAPI on Sunspot-ALCF. https://software.intel.com/ONEAPI.

[14] Jacob Lambert, Seyong Lee, Jeffrey S. Vetter, and Allen D. Malony. 2020. CCAMP: An Integrated Translation and optimization Framework for OpenACC and OpenMP. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–14. https://doi.org/10.1109/SC41405.2020.00102

[15] L.L. Lao, J.R. Ferron, R.J. Groebner, W. Howl, H. St. John, E.J. Strait, and T.S. Taylor. 1990. Equilibrium analysis of current profiles in tokamaks. *Nuclear Fusion* 30, 6 (jun 1990), 1035–1049. https://doi.org/10.1088/0029-5515/30/6/006

[16] L.L. Lao, H. St. John, R.D. Stambaugh, A.G. Kellman, and W. Pfeiffer. 1985. Reconstruction of current profile parameters and plasma shapes in tokamaks. *Nuclear Fusion* 25, 11 (nov 1985), 1611–1622. https://doi.org/10.1088/0029-5515/25/11/007

[17] L. L. Lao, H. E. St. John, Q. Peng, J. R. Ferron, E. J. Strait, T. S. Taylor, W. H. Meyer, C. Zhang, and K. I. You. 2005. MHD Equilibrium Reconstruction in the DIII-D Tokamak. *Fusion Science and Technology* 48, 2 (2005), 968–977. https://doi.org/10.13182/FST48-968 arXiv:https://doi.org/10.13182/FST48-968

[18] L L Lao, S Kruger, C Akcay, P Balaprakash, T A Bechtel, E Howell, J Koo, J Leddy, M Leinhauser, Y Q Liu, S Madireddy, J McClenaghan, D Orozco, A Pankin, D Schissel, S Smith, X Sun, and S Williams. 2022. Application of machine learning and artificial intelligence to extend EFIT equilibrium reconstruction. *Plasma Physics and Controlled Fusion* 64, 7 (jun 2022), 074001. https://doi.org/10.1088/1361-6587/ac6fff

[19] Neil A. Mehta, Rahulkumar Gayatri, Yasaman Ghadar, Christopher Knight, and Jack Deslippe. 2021. Evaluating Performance Portability of OpenMP for SNAP on NVIDIA, Intel, and AMD GPUs Using the Roofline Methodology. In *Accelerator Programming Using Directives*, Sridutt Bhalachandra, Sandra Wienke, Sunita Chandrasekaran, and Guido Juckeland (Eds.). Springer International Publishing, Cham, 3–24.

[20] NERSC. 2022. NERSC: Perlmutter GPU Nodes. https://docs.nersc.gov/systems/perlmutter/

[21] NVIDIA. 2020. NVIDIA A100 GPU ARCHITECTURE. https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf.

[22] NVIDIA. 2022. NVHPC Documentation. https://docs.nvidia.com/hpc-sdk/archive/22.7/index.html.

[23] NVIDIA. 2023. NVIDIA Nsight Compute CLI Documentation. https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html.

[24] OLCF. 2023. Cray Compilers on Frontier-OLCF. https://docs.olcf.ornl.gov/systems/frontier_user_guide.html#compiling

[25] OLCF. 2023. OLCF: Frontier GPU Nodes. https://docs.olcf.ornl.gov/systems/frontier_user_guide.html

[26] OpenACC. 2021. OpenACC Programming and Best Practices Guide. https://www.openacc.org/sites/default/files/inline-files/OpenACC_Programming_Guide_0_0.pdf

[27] OpenACC. 2022. OpenACC Application Program Interface. https://www.openacc.org/

[28] OpenMP. 2023. OpenMP Application Program Interface. https://www.openmp.org/

[29] V. D. Shafranov. 1966. Plasma equilibrium in a magnetic field. *Reviews of Plasma Physics* 2 (1966), 103.

[30] TOP 500. 2023. TOP 500 website. https://www.top500.org/.