Refactoring the EVP solver from the sea ice model CICE v6.5.1 for improved performance

Till Andreas Soya Rasmussen[1], Jacob Poulsen[2], Mads Hvid Ribergaard[1], Ruchira Sasanka[2], Anthony P. Craig[4], Elizabeth C. Hunke[3], and Stefan Rethmeier[1]

[1]Danish Meteorological Institute, Sankt Kjelds Plads 11, 2100 Copenhagen, Denmark
[2]Intel Corporation
[3]MS-B216, Los Alamos National Laboratory, Los Alamos, NM, 87545, USA
[4]Contractor to Science and Technology Corporation, Seattle, WA

Correspondence: Till Andreas Soya Rasmussen (tar@dmi.dk)

Abstract. This study focuses on the performance of the Elastic-Viscous-Plastic (EVP) dynamical solver within the sea ice model CICE v6.5.1. The study has been conducted in two steps. First, the standard EVP solver was extracted from CICE for experiments with refactored versions, which are used for performance testing. Second, one refactored version was integrated and tested in the full CICE model to demonstrate that the new algorithms do not significantly impact the physical results.

The study reveals two dominant bottlenecks, (1) the number of MPI and OpenMP synchronization points required for halo exchanges during each time-step combined with the irregular domain of active sea ice points, and (2) the lack of Single Instruction Multiple Data (SIMD) code generation.

The standard EVP solver has been refactored based on two generic patterns. The first pattern exposes how general finite-differences on masked multi-dimensional arrays can be expressed in order to produce significantly better code generation by changing the memory access pattern from random access to direct access. The second pattern takes an alternative approach to handle static grid properties.

The measured single core performance improvement is more than a factor of five compared to the standard implementation. The refactored implementation strong-scales on the Intel® Xeon® Scalable Processor Series node until the available bandwidth of the node is used. For the Intel® Xeon® CPU Max Series Series there is sufficient bandwidth to allow the strong-scaling to continue for all the cores on the node, resulting in a single node improvement factor of 35 over the standard implementation. This study also demonstrates improved performance on GPU processors.

1 Introduction

Numerical models of the Earth system and its components (e.g. ocean, atmosphere and sea ice) rely heavily on high performance computers (HPC, Lynch, 2006). When the first massively parallel computers emerged, steadily increasing CPU speeds improved performance sufficiently to support faster time-to-solution, higher resolution and improved physics. However, for the past decade improved performance has originated from an ever increasing number of cores, each supporting an increasing

number of SIMD lanes. Thus, for codes to run efficiently on today's hardware, they need to have excellent support of threads and efficient SIMD code generation.

Earth System Model (ESM) implementations often use static grid properties that are computed once and either carried from one subroutine to the next in data structures or accessed from global data structures. It makes sense from a logical perspective to not recompute the same thing over and over. Historically, floating point operations have been expensive and therefore this also made sense from a compute performance perspective. Modern hardware has changed and memory storage, particularly the bandwidth to memory, is now the scarce resource. Compared with floating point operations, it is not only scarce but also far more energy demanding. ESM components must be refactored to adapt to modern hardware features and limitations.

This study focuses on the dynamical solver of CICE (Hunke et al., 2024) a sea ice component in ESMs. In general, the same sea ice models are used both in climate models and in operational systems with different settings (Hunke et al., 2020). The dynamics solver in sea ice models is physically important – it calculates the momentum equation including internal sea ice stresses. The dynamics equations are usually based on the viscous-plastic (VP) model developed by Hibler (1979), which has a singularity that is difficult for numerical solvers to handle. Consequently, the Elastic-Viscous-Plastic approach was developed by Hunke and Dukowicz (1997), designed to solve the nonlinear VP equations using parallel computing architectures. In order to achieve a fast and nonsingular solution, elastic waves are added to the VP solution. The EVP solution ideally converges to the VP solution via hundreds of iterations, which dampen the elastic waves. Bouillon et al. (2013) and Kimmritz et al. (2016) showed that the number of iterations to convergence could be controlled and reduced, but the solver remains computationally expensive. Koldunov et al. (2019) reports that 550 iterations are needed in order to reach convergence with the traditional EVP solver in the finite element model FESOM2 at a resolution of 4.5km. According to Bouchat et al. (2022), the models of their inter-comparison used a wide range of number of EVP subcycles from 120 to 900. Thus, while several approaches have been proposed to reduce the number of iterations, some of these model systems likely do not iterate to convergence. The motivation for using fewer subcycles is often performance in terms of time-to-solution.

The number of subcycles needed to converge depends on the application, the configuration and the resolution. Unfortunately, the dynamics is one of the most computationally burdensome parts of the sea ice model. As an example, timings measured for standalone sea ice simulations using the operational model at the Danish Meteorological Institute (DMI) for the 1st of March 2020 show that the fraction of total runtime used by the dynamical core increases from approximately 15% to approximately 75% as the number of subcycles increases from 100 to 1000. Similar timings for other seasons, domains and/or a different strategy for allocation of memory will change the fractions, but there will still be a significant increase when the number of subcycles are increased. This motivates refactoring this part of the model system.

Another challenge for sea ice components in global Earth system models is load balancing across the domain, since sea ice covers only 12% and 7% of the ocean and Earth's surface, respectively (Weeks, 2010). In addition, the sea ice cover varies significantly in time and space, particularly with season. This adds additional complexity, especially for regional setups as the number of active sea ice points varies. Craig et al. (2015) discusses the inherent load imbalance issues and implements some advanced domain decompositions to improve the load balance in CICE.

This study demonstrates that the EVP model can be refactored to obtain a significant speedup, and that the method is useful for other parts of the sea ice model and for other ESM components. Section 2 presents the standard EVP solver, the refactorization, the standalone test and the experimental setup. Section 3 analyzes the results, and section 4 provides a discussion of the developments and next steps, including future work to improve CICE with the refactored EVP solver. The integration demonstrated in this study focus on correctness alone. Section 5 summarizes the conclusions.

2 The EVP solver

The aim of this study is to optimize the EVP solver of CICE by refactoring the code. This section analyzes the existing solver and describes improvements made in this study.

2.1 Standard implementation

The CICE grid is parallelized based on 2D blocks, including halos required for the finite difference calculation within the EVP solver. Communication between the blocks is based on MPI and/or OpenMP. EVP dynamics is calculated at every time step, and due to the nature of the finite differences, the velocities on the halo must be updated at every subcycling step. Input to the EVP dynamical core consists of external forcing from the ocean and atmosphere components and internal sea ice conditions. Stresses and velocities are output for use elsewhere in the code, such as to calculate advection.

Listing 1 provides a schematic overview of the standard EVP algorithm, which consists of an outer convergence loop, two inner stages (`stress` and `stepu`), and a halo swap of the velocities. Each of the inner stages carries an inner loop with its own trip-count based on subsets of the active points. The two inner stages within the subcycling exchange arrays used in the other stage. Therefore processes or threads must synchronize after each of the stages.

The inner stages operate on a subset of the grid points in 2D space. The grid points are classified as land points, water points or two types of active ice points, namely U cell points and T cell points, labeled according to the Arakawa definition of the B-grid (Arakawa and Lamb, 1977). T refers to the cell center and U refers to the velocity points at corners. Land and water points are static for a given configuration. Active points are defined by thresholds of the sea ice concentration and mass of sea ice plus snow. The result is that the number and location of active points may change at every time step, although they remain constant during the subcycling. Most grid cells have both T and U points active, but there are points that only belong to one of these subgroups. For instance, there may be ice in the cell center (T) but the U point lies along a coastline and is therefore inactive. The active sea ice points are always a subset of the water points, changing in time during the simulation due to the change of season and external forcing.

Listing 1. A schematic view of the subcycling of the EVP algorithm

```
1: do k = 1, ksub      ! ksub sub-cycles per model timestep
2:   ! stage 1 aka stress: use variables on T cells and velocities on U cells to
3:   ! define stress* on T cells and stage-interface vectors
4:   do i=1,nt         ! nt is number of active T cells at given timestep
```

```
5:      ! Finite-Difference computations here
6:      ...
7:    enddo
8:    ! stage 2 aka stepu, use variables on U cells and stage-interface
9:    ! to define new velocities and new variables on U cells
10:   do j=1,nu          ! nu is number of active U cells at given timestep
11:     ! Finite-Difference computations here
12:     ...
13:   enddo
14:   ! data dependencies: references in stage 1 are set in stage 2
15:   ! halo_swap with OpenMP and MPI neighbors
16: enddo
```

From a workload perspective, the arithmetic in the EVP algorithm confines itself to short-latency operations: `add`, `mult`, `div` and `sqrt`. The computation intensity is 0.3 FLOP/byte, which makes the workload highly bandwidth bound. Achieving a well-balanced workload is a huge challenge for any parallel algorithm working on these irregular sets, as was recognized in earlier studies that focus on the performance of CICE (Craig et al., 2015).

2.2 The refactored implementation

This section describes the refactored EVP solver, which was done in two steps. The first focused on improving the *core-level* parallelism and the second step focused on improving the *node-level* parallelism. The intention of the first step was to establish a solid, single-core baseline before diving into the thread parallelism of the solver.

2.2.1 Single core refactorization

An important part of the refactorization is changing memory access patterns to reduce the memory-bandwidth pressure at the cost of additional floating point operations. Listing 2 shows a snippet of the standard code (`v0`) before the first refactorization. the code fragment reveals a classical finite-difference pattern that is similar to the refactorization pattern shown in chapter 3 of Jeffers and Reinders (2015). The challenge is that compilers see the memory access pattern caused by indirect addressing as random memory access and will consequently refrain from using modern vector (SIMD) instructions in their code generation.

The refactored EVP code is shown in listing 3. The new solver introduces 1D structures in place of the original 2D structures, with additional indexing overhead to track the neighboring cells required by the finite-difference scheme. This change allows the compiler to see the memory access pattern as mostly direct addressing with some indirect addressing required for accessing states in neighboring cells. The compiler will consequently be able to generate SIMD instructions for the loop and will handle the remaining indirect addressing with SIMD gather instructions. The ratio of indirect to direct memory addressing is 10%-20%, depending on which of the two EVP loops is considered. For the Fortran programmer the two fragments look almost identical, but for the Fortran compiler the two fragments look very different, and the compiler will be able to convert the latter

fragment straight into an efficient ISA representation. The change of data structures from 2D to 1D is the only difference between `v0` and `v1`. The computational intensity of a loop iteration of `v0` and `v1` is identical.

Listing 2. Fragment showing Finite-Difference dependencies in the standard version of EVP (`v0`).

```
! VERSION: v0
subroutine stress(...,nx,ny,icellt,indxti,...)
  real (kind=dbl_kind), intent(in), dimension(nx,ny) :: cyp, ...
  ...
  do ij = 1, icellt
     i = indxti(ij)
     j = indxtj(ij)
     ! smaller Finite-Difference block with neighbor dependencies (i+-1,j+-1)
     divune    = cyp(i,j)*uvel(i  ,j  ) - dyt(i,j)*uvel(i-1,j  ) &
               + cxp(i,j)*vvel(i  ,j  ) - dxt(i,j)*vvel(i  ,j-1)
     ...
     ! larger block with no column dependencies
     stressp_1(i,j) = (stressp_1(i,j) + c1ne*(divune - Deltane)) &
                    * denom1
     ...
  enddo
end subroutine stress
```

Listing 3. Fragment showing Finite-Difference dependencies in the refactored version of EVP (`v1-simd`).

```
! VERSION: v1-simd
subroutine stress(...)
  real (kind=dbl_kind), dimension(:), intent(in), contiguous ::  uvel, ...
  ...
  do iw = il, iu
     tmp_uvel_ee = uvel(ee(iw))
     ...
     divune    = cyp(iw)*uvel(iw) - dyt(iw)*tmp_uvel_ee                    &
               + cxp(iw)*vvel(iw) - dxt(iw)*tmp_vvel_se
     ...
     stressp_1(iw) = (stressp_1(iw) + c1ne*(divune - Deltane)) * denom1
     ...
  enddo
end subroutine stress
```

CICE utilizes static grid properties which are computed once during initialization and then re-used in the rest of the simulation. As discussed in the introduction, a sensible strategy reduces bandwidth pressure by adding more pressure on the floating point engines while reducing memory storage. Our final refactored versions (`v2*`) have substituted 7 static grid arrays with 4 static base arrays plus some run time computations of local scalars, deriving all 7 original arrays as local scalars, cf. listing 4 that shows how the arrays `cxp` and `cyp` are derived. The `v2*` versions are further discussed in section 2.2.2.

Listing 4. Fragment showing Finite-Difference dependencies in the refactored version of EVP (`v2`).

```
! VERSION: v2-simd
subroutine stress(...)
  real (kind=dbl_kind), dimension(:), intent(in), contiguous ::  uvel, ...
  ...
  do iw = il, iu
     tmp_uvel_ee = uvel(ee(iw))
     tmp_cxp     = c1p5 * htn(iw) - p5 * htnm1(iw) ! derive cxp from htn, htnm1
     tmp_cyp     = c1p5 * hte(iw) - p5 * htem1(iw) ! derive cyp from htn, htnm1
     ...
     divune    = tmp_cyp*uvel(iw) - dyt(iw)*tmp_uvel_ee                    &
               + tmp_cxp*vvel(iw) - dxt(iw)*tmp_vvel_se
     ...
  enddo
end subroutine stress
```

2.2.2 Single node refactoring - OpenMP and OpenMP target

The existing standard parallelization operates on blocks of active points and elegantly uses the same parallelism for OpenMP and MPI. providing flexibility to run hybrid. It allows for a number of different methods to distribute the blocks onto the compute units to meet the complexity of the varying sea ice cover. There is no support for GPU offloading in the standard parallelization. The refactored EVP kernel demonstrates support for GPU offloading and showcases how this can be done in a portable fashion, confined to open standards. The underlying idea in the OpenMP parallelization is two-fold. First, we want to make the OpenMP synchronization points significantly more light-weight than the current standard implementation. The OpenMP synchronization points in the existing implementation involve explicit memory communication of data in the blocks used for parallelization. The proposed OpenMP synchronization only requires an OpenMP barrier to ensure cache coherency and no explicit data movement. Second, we want to increase the granularity of the parallelization unit from blocks of active points to single active points, which will allow better workload balance and scaling when the number of cores are increased.

Keeping the dependencies between the two stages in mind, we have two inner loops that can be parallelized. We must take into account that the T-cells and U-cells may not be identical sets, and we cannot even assume that one is included in the other. However, it is fair to assume that the difference between the T and U sets of active points is negligible, and instead of

treating the two sets as totally independent, we take advantage of their large overlap. Treating the two loops with the same trip count helps the performance tremendously both in terms of cache and in terms of None-Uniform Memory access (NUMA) placement. This requires some additional overhead in the code to skip the inactive points in the T and U loops.

The OpenMP standard published by OpenMP Architecture Review Board (2021) provides several options, and later Fortran standards (Fortran-2008 and Fortran-2018) also provide an opportunity to express this parallelism purely within Fortran. For OpenMP, we can do this either in an *outlined* fashion (see listing 5) or in an *inlined* fashion (see listing 6).

Listing 5. Fragment showing outlined OpenMP parallelization of EVP (`omp-outline`).

```
1:  ! VERSION: v2-omp-outline
2:  do i = 1, ndte
3:    !$omp parallel do schedule(runtime) private(iw)
4:    do iw = 1, union_tripcount
5:      call stress (iw,...)
6:    enddo
7:    !$omp end parallel do
8:    !$omp parallel do schedule(runtime) private(iw)
9:    do iw = 1, union_tripcount
10:     call stepu (iw,...)
11:   enddo
12:   !$omp end parallel do
13: enddo
```

Listing 6. Fragment showing traditional inlined (and OpenMP offloadling) OpenMP parallelization of EVP (`omp-inline`).

```
1:  ! VERSION: v2-omp-inline
2:  subroutine stress (...)
3:  ...
4: #ifdef _OPENMP_TARGET
5:    !$omp target teams distribute parallel do
6: #else
7:    !$omp parallel do schedule(runtime) default(none) shared(...) private(...)
8: #endif
9:    do iw = lb, ub
10:     if (skipt(iw)) cycle
11:     ...
12:   enddo
13: end subroutine stress
```

An alternative approach avoids explicit OpenMP runtime scheduling, instead using OpenMP as a short-hand notation for handling the spawning of the threads, then manually do the loop-splitting, as illustrated in listing 7. We refer to this approach

as the Single Program Multiple Data (SPMD) approach (Jeffers and Reinders, 2015; Levesque and Vose, 2017). In addition to its simplicity, this approach has the advantage that we can add balancing logic to the loop decomposition, accounting for the application itself and its data. For the sake of completeness, we also parallelize the EVP solver using the newer taskloop constructs available in OpenMP (see listing 8).

Listing 7. Fragment showing SPMD OpenMP parallelization of EVP (`omp-SPMD`).

```
1:  ! VERSION: v2-omp-SPMD
2:  !$omp parallel private(i)
3:  do i = 1, ndte
4:    call stress (union_tripcount, ...)
5:    !$omp barrier
6:    call stepu (union_tripcount, ...)
7:    !$omp barrier
8:  enddo
9:  !$omp end parallel
10: ....
11: subroutine stress(....)
12:   ...
13:   call domp_get_domain(lb,ub,il,iu) ! get local thread bounds il, iu
14:   do iw = il, iu
15:     if (skipme(iw)) cycle
16:     ....
17:   enddo
18: end subroutine stress
```

Listing 8. Fragment showing the newer OpenMP taskloop parallelization of EVP (`omp-taskloop`).

```
1:  ! VERSION: v2-omp-taskloop
2:  subroutine stress (...)
3:    ...
4:    !$omp parallel                                    &
5:    !$omp single                                      &
6:    !$omp taskloop simd                               &
7:    !$omp default(none) private(...) shared(...)      &
8:    do iw = lb, ub
9:      if (skipt(iw)) cycle
10:     ...
11:   enddo
12:   !$omp end taskloop simd
13:   !$omp end single
```

```
14:     !$omp end parallel
15:   end subroutine stress
```

A pure Fortran approach is available with the newer `do concurrent` (see listing 9) and here we may use compiler options to target either the CPU or GPU offloading.

Listing 9. Fragment showing pure Fortran-2018 approach to parallelism (`fortran-2018`).

```
1:   ! VERSION: v2-fortran-2018
2:   subroutine stress (...)
3:     ...
4:     do concurrent (iw=lb:ub) DEFAULT(NONE)                          &
5:                              SHARED(ee,ne,se,skipme,strength,uvel,vvel, &
6:                                     ...                              &
7:                              LOCAL(divune, divunw, divuse, divusw,    &
8:                                     ...                              &
9:     if (skipme(iw)) cycle
10:    ...
11:    enddo
12:  end subroutine stress
```

2.3 Test cases

Figure 1 shows the three domains used to test the refactorized EVP solver. Figure 1(a) is the operational sea ice model domain at DMI (Ponsoni et al., 2023), and figure 1(b) is the Regional Arctic System Model (RASM) domain (Brunke et al., 2018). Both domains cover the pan-Arctic area. These two systems are computationally expensive and are therefore used to analyze, demonstrate and test the performance of the new EVP solver. Tests are based on restart files from winter (March 1) and summer (September 1), where the ice extent is close to its maximum and minimum, respectively. All performance results shown in this manuscript originate from the RASM domain, which has the most grid points. Neither of these model systems include boundary conditions, which simplifies the problem.

The gx1 (1°) domain shown in figure 1(c and d) is included to test algorithm correctness and updates to the cyclic boundary conditions. In addition, gx1 is used for testing the numerical noise level for long runs using optimized flags within the CICE test suite. It is not used to evaluate the performance because MPI optimization is beyond the scope of this paper.

The number of grid points and their classification are listed in table 1. Figure 1 indicates and table 1 highlights that most grid points have both T and U points active, confirming our assumption in the design of the refactored EVP solver. However, there are cases where a grid point can have either a T or a U grid point active.

It is clear from figure 1 that the variation of active sea ice points is significant between winter and summer. Table 1 quantifies the variation of the different categories of grid cells. The number of active grid cells varies for the two pan-Arctic domains and are approximately half in summer compared to winter. The minimum and maximum sea ice extent is expected mid-September

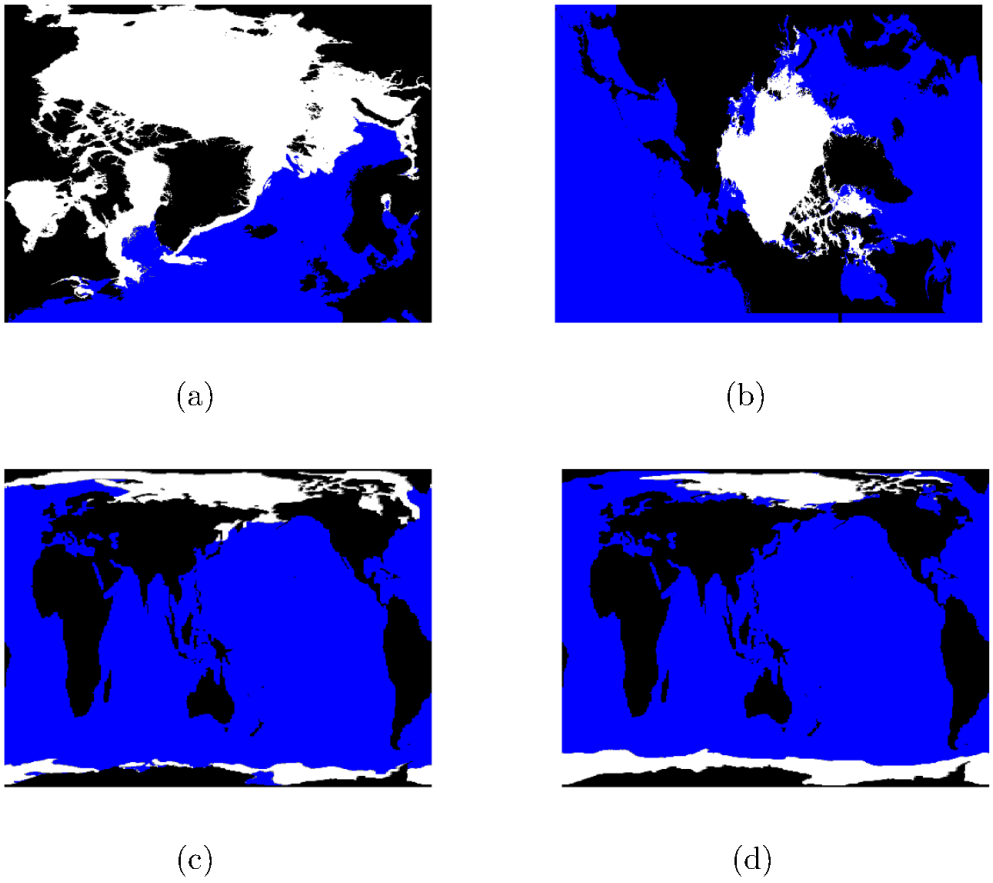


Figure 1. Three domains are used to test the new EVP solver and to verify its integration into CICE. Status of grid points from one restart file for each of the regional domains is shown and one winter and one summer extent are shown for the global domain. a) DMI domain on 1st of March 2020; b) RASM domain on 1st of September 2018; c) gx1 domain on 1st of March 2005; d) gx1 domain on 1st of September 2005. Black is land, blue is ocean, gray points have either active T or U points, and white points have both T and U active.

and mid-March, respectively. A sinusoidal variation is expected in the period between the minimum and the maximum for the two pan-Arctic domains. The variation is smaller in the global domain as Antarctica has the maximum number of active points when Arctic is at its minimum and vice versa. These variations can impact the strategy for allocation of memory as one goal is to reduce the memory usage.

Variation of the sea ice cover complicates load balancing when multiple blocks are used. The new implementation of the 1D solver automatically removes all land points and also allows the OpenMP implementation to share memory without synchronization between halos, as opposed to the original 2D OpenMP implementation within CICE.

Domain	total	water	ice winter		ice summer	
			$T \cap U$	$T \cup U$	$T \cap U$	$T \cup U$
DMI-NAAg	1,662,465	1,000,954	606,797	624,830	290,171	299,148
RASM	14,745,600	9,314,922	2,762,746	2,822,197	1,510,341	1,549,195
gx1	122,880	86,354	10,182	11,479	10,782	11,515

Table 1. Number of points for the four test domains: the total number of grid points excluding boundaries, the number of water points, the number of active T and U points ($\cap$), and the number of active T or U points ($\cup$) in winter and summer.

2.4 Test setup

The performance results in this study are based on a unit test that only includes the EVP solver and not the rest of CICE. Inputs are extracted from realistic CICE runs just before the subcycling of the EVP solver. Validation of the unit test is based on output extracted just after the EVP subcycling. The refactorizations have been tested in three stages (`v0`, `v1` and `v2`), as described below).

V0 Standard EVP solver.

V1 Single core refactoring of the memory access patterns used in the EVP solver

V2 Single node refactoring illustrating four different OpenMP approaches and one pure Fortran 2018 `do-concurrent` approach, including conversion from pre-computed grid arrays to scalars recomputed at every iteration.

Unit tests were conducted with different compiler flag optimizations ranging from very conservative to very aggressive. `V0` is the baseline and the performance of `v1` and `v2` has been compared to this. A weak-scaling feature has also been added to the standalone test in order to measure the performance at different resolutions / number of grid points and to allow for full node performance tests.

The refactorization and its impact on performance has been tested on four types of architectures (table 2), to demonstrate the effect of the bandwidth limitation and the performance enhancement on CPUs and GPUs. All CPU executables were built using the Intel® Classic Fortran compiler and all the GPU executables were built with the Intel® Fortran compiler from the oneAPI HPC toolkit 2023.0. All implementations use only open standards without proprietary extensions. Therefore, we expect that similar results can be achieved on hardware from other providers than Intel.

All performance numbers reported are the average time obtained for ten test repetitions using `omp_get_wtime()`. Timings do not include the conversion from 1D to 2D and vice versa. For the capacity measurements, 8 ensemble members run the same workload simultaneously, evenly distributed across the full node, device or set of devices. The timing of an ensemble run is the time of the slowest ensemble member; we repeat the ensemble runs ten times and report the average. All performance experiments on 4th Gen Intel® Xeon® Scalable Processor and Intel® Xeon® CPU Max Series are done in SNC4 mode and with HBM-only on Intel® Xeon® CPU Max Series.

Name	Type
3rd Gen Intel® Xeon® Scalable Processor	72core-CPU+DDR4 memory
4th Gen Intel® Xeon® Scalable Processor	112core-CPU+DDR5 memory
Intel® Xeon® CPU Max Series	112core-CPU+HBM memory
Intel® Data Center GPU Max Series	GPU+HBM memory

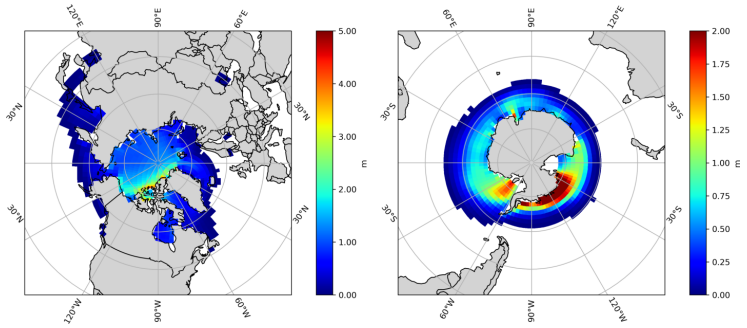
Table 2. Description of the CPUs and the GPU used in this study. Hardware listed with HBM includes high bandwidth memory. More information can be found on https://www.intel.com/content/www/us/en/products/overview.html.

The GPU results indicate only the compute part, not the usually time-consuming data traffic between the CPU and the GPU. Because the kernel constitutes a single model time step, most data traffic in this kernel is one-time initialization and hence would not contribute to the compute time in the $N-1$ remaining time-steps of a full simulation.

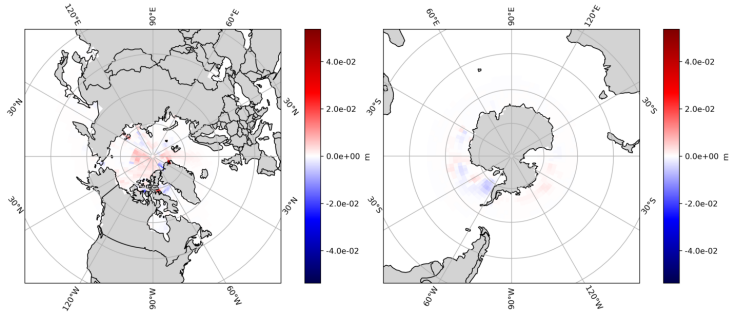
Finally, one of the `V2` refactored units was integrated back into CICE (Hunke et al., 2024). This initial integration is focused solely on correctness. Section 4 presents our proposal of a performance focused integration, which can be considered a refactorization at the *cluster-level*, on top of the refactoring at the *core-* and *node-levels* reported here.

The new method does not include any new physics. Therefore, it is important that the results remain the same. This is verified by checking that restart output files contain "bit-for-bit" identical results at the end of two parallel simulations, which requires identical *md5sum*s on non-optimized code. All tests verify this (not shown). It should be noted that the baseline `v0` cannot be run on a GPU, since the baseline code only supports building for CPU targets. Therefore it is not possible to cross-compare GPU baseline results with refactored GPU results. The GPU results were compared with the refactored CPU with no expectation of bit-for-bit identical results.

When the build of the binary executable uses more aggressive compiler optimization flags it may use operations that produce a different final round-off error. It may also do the exact same calculations in another order, which results in bit-for-bit differences due to the discrete representation of real numbers. For instance, a fused multiply-add operation has one rounding operation, whereas the same calculation can be represented by one multiplication followed by an addition that results in two rounding operations. Such a deviation does not originate from differences in the semantics dictated by the source code itself, which expresses the exact same set of computations. The difference originates from the ability of the compiler to choose from a larger set of instructions. For the optimized, non-bit-for-bit `V2` that is integrated into CICE it is verified that the numerical noise is at an acceptable level with a Quality Control (QC) module (Roberts et al., 2018) provided with the CICE software. The CICE QC test checks that two non-identical ice thickness results are statistically the same based on five year simulations on the gx1 domain. The result is shown in figure 2. The refactored code passes the QC test when integrated into CICE because the noise level is lower than the test's criterion.



(a)



(b)

Figure 2. a) Sea ice thickness on the northern hemisphere. b) Sea ice thickness on the southern hemispheres. c) Difference between CICE using the standard EVP solver (`v0`) and the refactored EVP solver (`v2`) implemented into CICE on the northern hemisphere. d) Difference between CICE using the standard EVP solver (`v0`) and the refactored EVP solver (`v2`) implemented into CICE on the northern hemisphere. All results represent the 1st of January 2009, after 5 years of simulation starting on the 1st of January 2005 and using the gx1 grid provided by the CICE Consortium.

3 Performance results

There are several ways to measure and evaluate compute performance. This section focuses on evaluating the EVP standalone kernel performance as measured by *time-to-solution* on different compute nodes. The evaluation is split into two steps, single-core performance and single-node performance.

3.1 Single core performance

The motivation for the single core refactorization described in section 2.2.1 is to allow the compiler to utilize vector instructions, also known as single instruction multiple data (SIMD) instructions, instead of confining itself to x86 scalar instructions for both memory accesses and math operations. The instruction sets formally known as AVX-2 (256 bit) and AVX-512 (512 bit) constitutes two newer generations of SIMD to the x86 instruction set architecture for microprocessors from Intel and AMD. Compiler options can be used to specify specific versions of SIMD instructions, but the compiler can only honor this request if

the code itself is SIMD vectorizable. One requirement for SIMD vectorization is direct memory rather than random memory access.

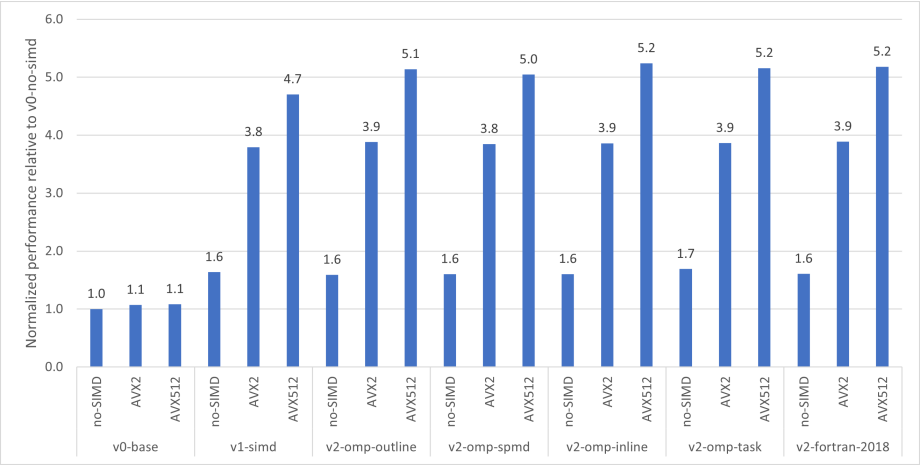


Figure 3. Single-core 3rd Gen Intel® Xeon® Scalable Processor (8360y) performance for the same algorithm (EVP) implemented via different approaches and with the build process requesting no SIMD or AVX2 and AVX512 code generation. The prefix versions `v0`, `v1` and `v2` are defined in section 2.4 and the baseline is the original implementation (`v0`) without SIMD generation. Each bar shows the improvement factor compared to the baseline.

Figure 3 shows the single core performance of the different EVP implementations described in listings 2-3 and listing 5-9 for the RASM domain described in table 1. The upstream implementation (`v0`) shows limited improvement when applying either AVX-2 or AVX-512 as described in section 2. The improvement factor for the single core refactorization is approximately 1.6 when SIMD instructions are not used for building `v1` and `v2*`. When SIMD is used, the improvement factor increases to 3.8 for AVX2 and 5.1 for AVX512 code generation. Moreover, the SIMD improvements are achieved across the different OpenMP versions. Although all the refactored versions show the same performance, this is not given a priori, since the intermediate code representation given to the compiler back-end is expected to be different for each of these representations. The refactorization from 2D to 1D changes the memory access from random to direct, allowing the compiler to use the SIMD instructions.

The results from the simulations on the single core show that the refactorization improves the code generation and associated performance. In addition, the one-dimensional compressed memory footprint is much more efficient that the standard two-dimensional block structure, since it reduces the memory footprint by the ratio of ice points to grid points. For the RASM case, it amounts to a factor of 5 reduction in winter and 10 in summer (see table 1). Importantly, all points in the 2D arrays used in the standard implementation must be allocated, whereas the refactored data structure only needs the active points allocated.

3.2 Single node performance

Efficient node performance requires that an implementation have both good single-core performance and good scaling properties. The main target of this section is to describe the performance results as measured by the *time-to-solution* on a given node

architecture. The performance diagrams found in this section show the performance outcome when all the cores available on the node or device are used. The refactored code is ported to GPUs using OpenMP target offloading. For the capacity scaling study this allow us to run on hosts that support multiple GPU devices, but we have confined the strong scaling study to classical OpenMP offloading, which currently only supports single device offloading, cf. Raul Torres and Teruel (2022).

In addition, single node performance is measured according to the relevant hardware metrics. Because the EVP implementation is memory bandwidth-bound, it is relevant to compare the sustained bandwidth performance of EVP with the well-established bandwidth benchmark STREAM triad, cf. McCalpin (1995) and figure 4. The STREAM triad benchmark delivers a main memory bandwidth number measured in Gb/s and is considered to be the practical limit sustainable on the system being measured.

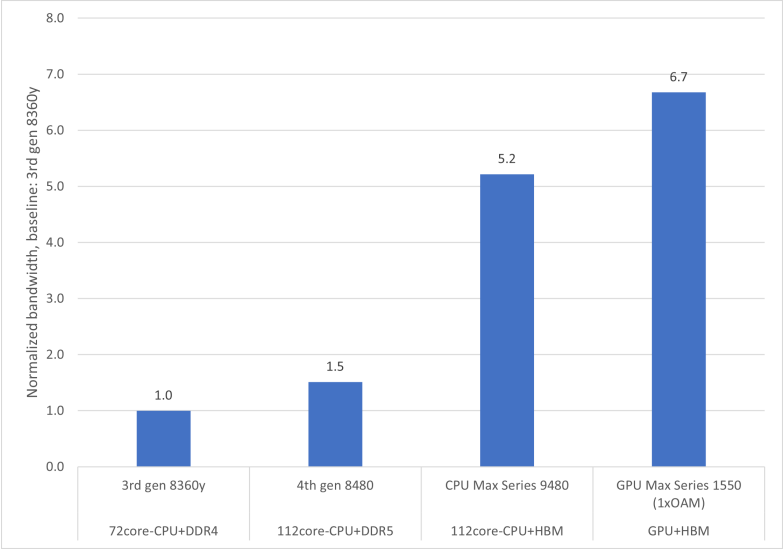


Figure 4. The figure shows the improvement factor for STREAM triad memory bandwidth benchmark on different hardware, indicating what is achievable at best for solely bandwidth-bound code. Left to right, the bars are the baselines based on 3rd Gen Intel® Xeon® Scalable Processor, 4th Gen Intel® Xeon® Scalable Processor, Intel® Xeon® CPU Max Series and Intel® Data Center GPU Max Series, cf. table 2.

Section 3.2.1 focuses on performance results for a strong scaling study whereas section 3.2.2 focuses on capacity scaling. Single-node performance is evaluated on the architectures described in table 2. The measured performance is compared to STREAM triad benchmarks. This indicates whether the algorithm utilizes the full bandwidth of the hardware. Numbers are compared for usage of the full node.

3.2.1 Strong scaling

Strong scaling is defined as the ability to run the same workload faster by using more resources. The ability to strong-scale any workload is described by Amdahls law (Amdahl, 1967).

Figure 5(a) shows the impact of the choice of architecture on the single-node performance results of the refactored EVP code for the four architectures described in table 2. Note that 3rd Gen Intel® Xeon® Scalable Processor only has 72 cores. The improvement factor between the CPUs with DDR-based memory coincides with the improvement factor obtained by STREAM triad (figure 4), which is considered the practical achievable limit of the hardware. The improvement factor for the two bandwidth-optimized architectures (Intel® Xeon® CPU Max Series and Intel® Data Center GPU Max Series) is less than the corresponding improvement factor obtained by STREAM triad. This indicates that bandwidth to memory is no longer the limiting performance factor. This finding will be discussed further in section 4.

Figure 5(b) shows the performance at different core counts for the two hardware types (4th Gen Intel® Xeon® Scalable Processor and Intel® Xeon® CPU Max Series) that are similar except for their bandwidth. The first observation is that the performance of the HBM-based CPU is better than that of the DDR-based CPU. The second observation is that the DDR-based hardware performance stops improving at approximately half the number of cores available on the node, which prevents further scaling on that memory system. With the HBM hardware, the code scales out to all the cores on the node. The improvement factor differs because the sustained bandwidth becomes saturated on the 4th Gen Intel® Xeon® Scalable Processor memory. This underlines the importance of the code refactorization to reduce the pressure on the bottleneck, which in this case is the bandwidth. It also illustrates that the hardware sets the limits for the potential optimization.

Strong scaling performance is also measured for bandwidth in absolute numbers in table 3. The absolute bandwidth measurements confirm that the DDR-based memory obtain the same bandwidth as STREAM triad, whereas the HBM based CPU's do not utilize the full bandwidth.

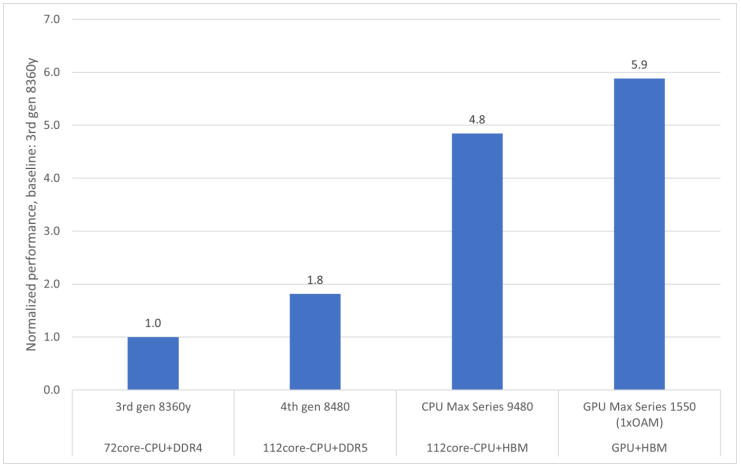
Hardware Name	Maximum bandwidth [Gb/s]	Average bandwidth [Gb/s]	Stream triad [Gb/s]
4th Gen Intel® Xeon® Scalable Processor	490	441	493
Intel® Xeon® CPU Max Series	1395	1221	1630

Table 3. Absolut measurements of memory bandwidth for strong scaling.

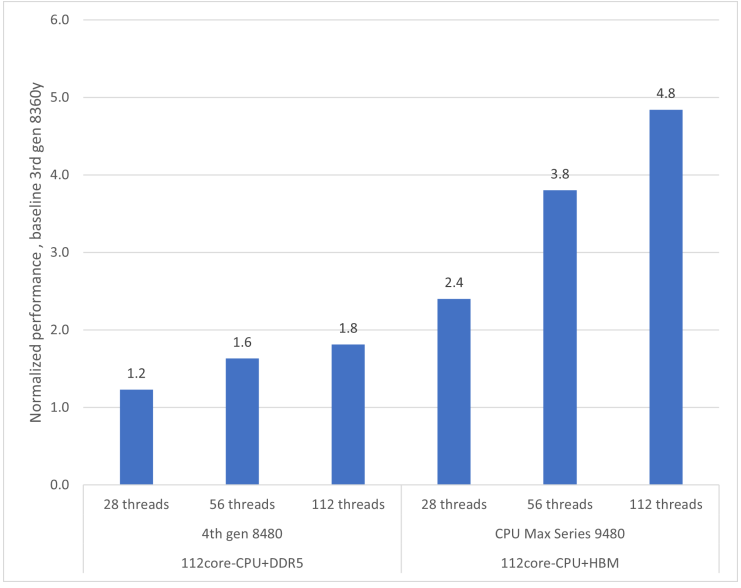
If both the standard and the new implementations of the EVP solver strongly scale equally well, then the node improvement factor should be the same as the single core improvement factor found in section 3.1. The observed improvement factors of refactored versus standard EVP on 4th Gen Intel® Xeon® Scalable Processor and Intel® Xeon® CPU Max Series are 13 and 35 (not shown), respectively, i.e. the new EVP solver also scales better than the original EVP implementation on both systems. The standard EVP code allows for multiple decompositions, which may affect the result, but the conclusions remain the same.

3.2.2 Capacity scaling

Capacity scaling is defined as the ability to run the same workload in multiple incarnations (called ensemble members) simultaneously on multiple compute resources. *Perfect capacity scaling* is achieved when we can run N ensemble members on N compute resources, with a performance degradation bounded by the *run-to-run-variance* measured when running one



(a)



(b)

Figure 5. Improvement factors for the refactored EVP code (`V2`). a) Strong scaling performance when using all available cores on one node for each of the four hardware types. Left to right: 3rd Gen Intel® Xeon® Scalable Processor, 4th Gen Intel® Xeon® Scalable Processor, Intel® Xeon® CPU Max Series and Intel® Data Center GPU Max Series, cf. table 2. b) Strong scaling performance at three different core counts for 4th Gen Intel® Xeon® Scalable Processorand Intel® Xeon® CPU Max Series . The baseline for a) and b) is the performance of 3rd Gen Intel® Xeon® Scalable Processor when using all cores.

ensemble member on 1 compute resource and leaving the rest of the compute resources idle. This performance metric indicates how sensitive the performance is to what is being executed on the neighboring compute resources.

Table 4 shows the absolute numbers and the output from the STREAM Triad test for capacity scaling. The capacity scaling is perfect on the DDR-based systems, i.e. the variance of the timings between individual ensemble members are similar to the variation in timings of repeated single member runs.

Hardware Name	Maximum [Gb/s]	Average [Gb/s]	Stream triad [Gb/s]
4th Gen Intel® Xeon® Scalable Processor	481	477	493
Intel® Xeon® CPU Max Series	1421	1280	1630

Table 4. Absolute measurements of memory bandwidth for capacity scaling.

Figure 6 summarizes the capacity scaling results, highlighting how well the different types of hardware perform compared to the best-known achievable bandwidth estimates, which are based on STREAM triad. The improvement factor between the two DDR-based systems again coincides with the improvement factor obtained by STREAM triad, which means that the performance is bandwidth bound. The improvement factor for the bandwidth optimized architectures is somewhat less than the corresponding improvement factor obtained by STREAM triad. This is discussed further in section 4.

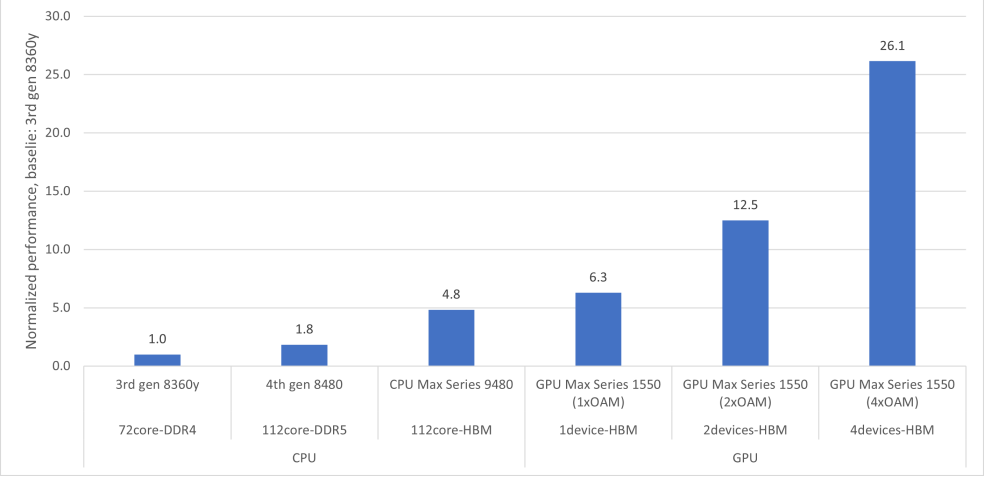


Figure 6. Capacity scaling performance of the refactored EVP unit test, normalized to the 3rd Gen Intel® Xeon® Scalable Processor (3rd gen 8360y) node baseline, which sustains the same bandwidth as STREAM triad. The figure also illustrates how much 1, 2 and 4 GPU devices per node matter to the collected node throughput.

HPC systems with GPUs typically host multiple devices per node, and this is the reason that we have conducted a multi-device experiment. The experiment shows that the performance continues to increase, but the energy required to drive a node with 4 GPU devices is significantly higher than the energy required to drive the dual-socket CPU that we cross-compare with. Our intent is not to compare the CPUs and GPUs directly because other elements such as energy or price should be considered.

4 Discussion

These results demonstrate a refactorization of the EVP solver within CICE that takes full advantage of modern CPUs and GPUs. All performance tests are based on an EVP solver unit test as described in section 2.4. The new implementation is easy to adapt to an unstructured grid, although it is implemented here on a structured grid. There were choices both in the refactorization of the EVP solver and in its integration into CICE, which are discussed in this section.

The new single-core SIMD parallel performance and the new single-node OpenMP parallel performance are evaluated in the previous sections. Based on comparisons with STREAM triad, the refactored code reaches the peak bandwidth performance of the two DDR-based memory systems, but the peak bandwidth of the HBM based system is not reached. On the Intel® Xeon® CPU Max Series with HBM running in HBM-only mode, we reached $\approx 80\%$ of the practical peak bandwidth. This result is consistent for both capacity scaling and strong scaling. Although the improvement factor is the same for both types of scaling, the reasons for the performance gap are quite different. The strong scaling gap for the Intel® Xeon® CPU Max Series CPU is solely due to limitations in the algorithm and data set, where there is an inherent imbalance.

The capacity scaling issue is that the hardware enforces a lower frequency (both core and uncore) to ensure that it does not overheat, when all NUMA domains operate simultaneously. A drop in the core frequency results in a slow down of the computations. In addition, the reduction of the uncore frequency causes higher memory latency. STREAM triad do not include the effects enforced by the hardware. For this reason the STREAM triad benchmark is too simple for the bandwidth optimized CPU.

The first integration step described in this study uses the current infrastructure within CICE and focuses solely on correctness, not on the performance, in order to establish a solid foundation for future work. For instance, the implementation utilizes existing gather/scatter methods to convert some of the arrays from 1D to 2D global to 3D block-structure (and vice versa), which is used for parallelization in the rest of the CICE code. It would be better to convert the 1D arrays directly to the block-structure (1D to 3D). The number of calls to gather/scatter methods could also be reduced. The ideal solution would be for all spatial quantities to only exist as 1D vectors; most EVP loops are already geared towards this as they loop in 1D space with pointers to the 2 indices used in the array allocation. The halo-swaps could be re-introduced directly on the new 1D data structures using `MPI-3 neighborhood collectives`.

A major performance challenge within the standard EVP solver is the set of halo updates required at every EVP subcycle, since each halo update introduces an MPI synchronization. Better convergence is achieved when the number of subcycles increases, but this also linearly increases the number of MPI synchronizations. The goal is to improve performance of the full model, therefore the number of synchronizations must be reduced.

The initial integration of the refactored EVP code into CICE only allows MPI synchronizations at the time step level. This prevents CICE from being split into sub components as is suggested below, and it requires the same number of threads for the refactored EVP as for the rest of CICE. The refactored EVP will consequently only be able to utilize a single MPI task, leaving the remaining MPI tasks idle. This is obviously a very inefficient integration that retains the observed scaling challenge at the *cluster*-level.

To improve the initial integration performance and to cope with the underlying challenges, an alternative approach is suggested that leverages MPMD parallelization (Mattson et al., 2005). This allows heterogeneous configurations, where the EVP solver could run on separate hardware resources and/or utilize different parallelization strategies (e.g. pure MPI, hybrid OpenMP-MPI, hybrid OpenMP-MPI running OpenMP offloading) relative to the rest of the model. If a time lag is implemented between the two components, they could run concurrently. This approach is also beneficial for the performance of the rest of the model, as it relieves the model from carrying EVP state variables and prevents flushing the cached EVP state at every time step. It would also allow runs of several EVP ensemble members on a single node, serving a set of model ensemble members each running on their own set of nodes. Also, it would be easier to integrate the new EVP component into other modeling systems, because it will have a pure MPI interface.

The MPMD pattern is generally used by ESM communities for load-balancing coupled model systems, e.g. where the ocean and the sea ice model run on different groups of the cores or nodes (e.g., Ponsoni et al., 2023; Craig et al., 2012). Sometimes MPMD is used internally within systems for I/O (e.g. the ocean model NEMO, Madec et al., 2023). To the best of our knowledge it is *not* common to use the MPMD pattern for model sub-components beyond I/O handling, nor is it common for it to support heterogeneous systems.

This new implementation of the EVP solver within CICE includes a strategy for how to allocate data. The strategy selected for this integration is to allocate all ocean points, then check whether or not they are active within the calculations. For the domains in this study, this strategy induces a large overhead, since there are many ocean points that are never active (see table 1). However, this behavior is domain specific and will be very different for different setups. A second strategy could be to reallocate all 1D vectors at every timestep, only allocating the active points. This induces an overhead for reallocating at every time step, but it reduces the memory usage. An alternative, intermediate method would be to only reallocate when the number of active points increases above what has been allocated. We propose this last strategy for the final MPMD-based MPI refactoring in CICE.

5 Conclusions

This study analyzed the performance of the EVP solver extracted from the sea ice model (CICE) and found performance challenges with the standard parallelization options at the *core-*, *node-level* and *cluster-level*. Evaluation of the refactorized solver demonstrates significant performance and memory footprint improvements.

The refactored EVP code improved performance by a factor of 5 compared to the original version when 1 core is used on 3rd Gen Intel® Xeon® Scalable Processor. This improvement is primarily the result of a change in the memory access patterns from random to direct, which allows the compiler to utilize vector instructions such as SIMD. When using 112 cores (full node) the improvement factor on the 4th Gen Intel® Xeon® Scalable Processor is 13 and on the Intel® Xeon® CPU Max Series is 35. The study showed that the limiting performance factor for EVP on traditional CPU's is the memory bandwidth. This is the main difference between the two types of hardware and the main reason for the difference in performance on a full node.

Table A1. Acronyms

Abbreviation	Full name
CICE	The Los Alamos sea ice model
DMI	Danish Meteorological Institute
DDR	Double Data Rate
ESM	Earth System Model
EVP	Elastic-Viscous-Plastic
HBM	High Bandwith Memory
MPMD	Multiple Program Multiple Data
NUMA	None-Uniform memory access
QC	Quality Control
RASM	Regional Arctic System Model
SIMD	Single Instruction Multiple Data
SPMD	Single Program Multiple Data
VP	Viscous-Plastic

The refactored version is capable of sustaining STREAM triad bandwidth (practical peak performance) on the CPUs within this study that are based on DDR-based memory. For strong scaling on the Intel® Xeon® CPU Max Series, only 80% of the bandwidth was used due to imbalances in the algorithm and the datasets. Finally, GPUs deliver higher memory bandwidth than CPUs, so we also ported the new implementation to nodes with GPU devices. All CPU and GPU performance was achieved solely by using open standards, OpenMP and oneAPI in particular.

The single-node improvements were integrated into the CICE model to check simulation correctness. Our next step will be to improve the integrated code, focusing on full model performance for both CPUs and GPUs.

Code and data availability. The source code for the standalone EVP units and test can be found on Rasmussen et al. (2024a). Inputdata for these are found at Rasmussen et al. (2024b). The CICE v6.5.1 code used for the QC test can be found at Hunke et al. (2024). Data sets for the QC runs can be found at https://github.com/CICE-Consortium/CICE/wiki/CICE-Input-Data with DOI numbers: 10.5281/zenodo.5208241, 10.5281/zenodo.8118062 and 10.5281/zenodo.3728599.

Appendix A: Abbreviations

Author contributions. JP contributed the main idea and effort of refactoring the EVP code, with input from TR, MR and RS. AC provided the RASM test configuration. TR integrated the refactored EVP with support from JP, MR AC, SR and EH. TR and JP wrote the manuscript with input from all others.

Competing interests. To the authors' knowledge, there are no competing interests.

Acknowledgements. The study is funded by the Danish State through the National Centre for Climate Research (NCKF) and the Nordic council of Ministers through the NOrdic CryOSphere Digital Twin (NOCOS DT) project with grant number 102642. Elizabeth Hunke was supported by the U.S. Department of Energy Office of Biological and Environmental Research, Earth System Model Development program. Anthony P. Craig was funded through a National Oceanic and Atmospheric Administration contract in support of the CICE Consortium.

References

References

Amdahl, G. M.: Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities, in: Proceedings of the April 18-20, 1967, Spring Joint Computer Conference, AFIPS '67 (Spring), p. 483–485, Association for Computing Machinery, New York, NY, USA, ISBN 9781450378956, https://doi.org/10.1145/1465482.1465560, 1967.

Arakawa, A. and Lamb, V. R.: Computational Design of the Basic Dynamical Processes of the UCLA General Circulation Model, in: General Circulation Models of the Atmosphere, edited by Chang, J., vol. 17 of *Methods in Computational Physics: Advances in Research and Applications*, pp. 173–265, Elsevier, https://doi.org/https://doi.org/10.1016/B978-0-12-460817-7.50009-4, 1977.

Bouchat, A., Hutter, N., Chanut, J., Dupont, F., Dukhovskoy, D., Garric, G., Lee, Y. J., Lemieux, J.-F., Lique, C., Losch, M., Maslowski, W., Myers, P. G., Ólason, E., Rampal, P., Rasmussen, T., Talandier, C., Tremblay, B., and Wang, Q.: Sea Ice Rheology Experiment (SIREx): 1. Scaling and Statistical Properties of Sea-Ice Deformation Fields, Journal of Geophysical Research: Oceans, 127, e2021JC017 667, https://doi.org/https://doi.org/10.1029/2021JC017667, e2021JC017667 2021JC017667, 2022.

Bouillon, S., Fichefet, T., Legat, V., and Madec, G.: The elastic–viscous–plastic method revisited, Ocean Modelling, 71, 2–12, https://doi.org/https://doi.org/10.1016/j.ocemod.2013.05.013, arctic Ocean, 2013.

Brunke, M. A., Cassano, J. J., Dawson, N., DuVivier, A. K., Gutowski Jr., W. J., Hamman, J., Maslowski, W., Nijssen, B., Reeves Eyre, J. E. J., Renteria, J. C., Roberts, A., and Zeng, X.: Evaluation of the atmosphere–land–ocean–sea ice interface processes in the Regional Arctic System Model version 1 (RASM1) using local and globally gridded observations, Geoscientific Model Development, 11, 4817–4841, https://doi.org/10.5194/gmd-11-4817-2018, 2018.

Craig, A. P., Vertenstein, M., and Jacob, R.: A new flexible coupler for earth system modeling developed for CCSM4 and CESM1, The International Journal of High Performance Computing Applications, 26, 31–42, https://doi.org/10.1177/1094342011428141, 2012.

Craig, A. P., Mickelson, S. A., Hunke, E. C., and Bailey, D. A.: Improved parallel performance of the CICE model in CESM1, The International Journal of High Performance Computing Applications, 29, 154–165, https://doi.org/10.1177/1094342014548771, 2015.

Hibler, W. D., I.: A Dynamic Thermodynamic Sea Ice Model, Journal of Physical Oceanography, 9, 815–846, https://doi.org/10.1175/1520-0485(1979)009<0815:ADTSIM>2.0.CO;2, 1979.

Hunke, E., Allard, R., Blain, P., Blockley, E., Feltham, D., Fichefet, T., Garric, G., Grumbine, R., Lemieux, J.-F., Rasmussen, T., Ribergaard, M., Roberts, A., Schweiger, A., Tietsche, S., Tremblay, B., Vancoppenolle, M., and Zhang, J.: Should Sea-Ice Modeling Tools Designed for Climate Research Be Used for Short-Term Forecasting?, Current Climate Change Reports 6, https://doi.org/10.1007/s40641-020-00162-y, 2020.

Hunke, E., Allard, R., Bailey, D. A., Blain, P., Craig, A., Dupont, F., DuVivier, A., Grumbine, R., Hebert, D., Holland, M., Jeffery, N., Lemieux, J.-F., Osinski, R., Poulsen, J., Steketee, A., Rasmussen, T., Ribergaard, M., Roach, L., Roberts, A., Turner, M., Winton, M., and Worthen, D.: CICE-Consortium/CICE: CICE Version 6.5.1, https://doi.org/10.5281/zenodo.11223920, 2024.

Hunke, E. C. and Dukowicz, J. K.: An elastic-viscous-plastic model for sea ice dynamics, J. Phys. Oceanogr., 27, 1849–1867, 1997.

Jeffers, J. and Reinders, J.: High performance parallelism pearls volume two: multicore and many-core programming approaches, Morgan Kaufmann, 2015.

Kimmritz, M., Danilov, S., and Losch, M.: The adaptive EVP method for solving the sea ice momentum equation, Ocean Modelling, 101, 59–67, https://doi.org/https://doi.org/10.1016/j.ocemod.2016.03.004, 2016.

Koldunov, N. V., Danilov, S., Sidorenko, D., Hutter, N., Losch, M., Goessling, H., Rakowsky, N., Scholz, P., Sein, D., Wang, Q., and Jung, T.: Fast EVP Solutions in a High-Resolution Sea Ice Model, Journal of Advances in Modeling Earth Systems, 11, 1269–1284, https://doi.org/https://doi.org/10.1029/2018MS001485, 2019.

Levesque, J. and Vose, A.: Programming for Hybrid Multi/Manycore MPP systems, CRC Press, Taylor and Francis Inc, ISBN 978-1-4398-7371-7, 2017.

Lynch, P.: The Emergence of Numerical Weather Prediction, Cambridge University Press, ISBN 0521857295 9780521857291, 2006.

Madec, G., Bell, M., Blaker, A., Bricaud, C., Bruciaferri, D., Castrillo, M., Calvert, D., Chanut, J., Clementi, E., Coward, A., Epicoco, I., Éthé, C., Ganderton, J., Harle, J., Hutchinson, K., Iovino, D., Lea, D., Lovato, T., Martin, M., Martin, N., Mele, F., Martins, D., Masson, S., Mathiot, P., Mele, F., Mocavero, S., Müller, S., Nurser, A. G., Paronuzzi, S., Peltier, M., Person, R., Rousset, C., Rynders, S., Samson, G., Téchené, S., Vancoppenolle, M., and Wilson, C.: NEMO Ocean Engine Reference Manual, https://doi.org/10.5281/zenodo.8167700, 2023.

Mattson, T. G., Sanders, B. A., and Massingill, B.: Patterns for parallel programming, Addison-Wesley, Boston, ISBN 0321228111 9780321228116, 2005.

McCalpin, J. D.: Memory Bandwidth and Machine Balance in Current High Performance Computers, IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter, pp. 19–25, 1995.

OpenMP Architecture Review Board: OpenMP Application Program Interface Version 5.2, https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-2.pdf, 2021.

Ponsoni, L., Ribergaard, M. H., Nielsen-Englyst, P., Wulf, T., Buus-Hinkler, J., Kreiner, M. B., and Rasmussen, T. A. S.: Greenlandic sea ice products with a focus on an updated operational forecast system, Frontiers in Marine Science, 10, https://doi.org/10.3389/fmars.2023.979782, 2023.

Rasmussen, T. A. S., Poulsen, J., Ribergaard, M. H., and Rethmeier, S.: dmidk/cice-evp1d: Unit test refactorization of EVP solver CICE, https://doi.org/10.5281/zenodo.10782548, 2024a.

Rasmussen, T. A. S., Poulsen, J., Ribergaard, M. H., and Rethmeier, S.: Input data for 1d EVP model, https://doi.org/10.5281/zenodo.11248366, 2024b.

Raul Torres, R. F. and Teruel, X.: A Novel Set of Directives for Multi-device Programming with OpenMP, https://doi.org/10.1109/IPDPSW55747.2022.00075, 2022.

Roberts, A. F., Hunke, E. C., Allard, R., Bailey, D. A., Craig, A. P., Lemieux, J.-F., and Turner, M. D.: Quality control for community-based sea-ice model development, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 376, 20170 344, https://doi.org/10.1098/rsta.2017.0344, 2018.

Weeks, W.: On sea ice, University of Alaska Press, 2010.