chemengineering

Article

Fortran Coarray Implementation of Semi-Lagrangian Convected Air Particles within an Atmospheric Model

Soren Rasmussen [1,*], Ethan D. Gutmann [2], Irene Moulitsas [1] and Salvatore Filippone [3]

[1] Centre for Computational Engineering Sciences, Cranfield University, Bedford MK43 0AL, UK; i.moulitsas@cranfield.ac.uk
[2] National Center for Atmospheric Research, Boulder, CO 80305, USA; gutmann@ucar.edu
[3] Department of Civil and Computer Engineering, Università di Roma "Tor Vergata", 32249 Rome, Italy; salvatore.filippone@uniroma2.it
* Correspondence: s.rasmussen@cranfield.ac.uk

Abstract: This work added semi-Lagrangian convected air particles to the Intermediate Complexity Atmospheric Research (ICAR) model. The ICAR model is a simplified atmospheric model using quasi-dynamical downscaling to gain performance over more traditional atmospheric models. The ICAR model uses Fortran coarrays to split the domain amongst images and handle the halo region communication of the image's boundary regions. The newly implemented convected air particles use trilinear interpolation to compute initial properties from the Eulerian domain and calculate humidity and buoyancy forces as the model runs. This paper investigated the performance cost and scaling attributes of executing unsaturated and saturated air particles versus the original particle-less model. An in-depth analysis was done on the communication patterns and performance of the semi-Lagrangian air particles, as well as the performance cost of a variety of initial conditions such as wind speed and saturation mixing ratios. This study found that given a linear increase in the number of particles communicated, there is an initial decrease in performance, but that it then levels out, indicating that over the runtime of the model, there is an initial cost of particle communication, but that the computational benefits quickly offset it. The study provided insight into the number of processors required to amortize the additional computational cost of the air particles.

Keywords: convection; air parcels; atmospheric model; high-performance computing; Coarray Fortran; PGAS

Citation: Rasmussen, S.; Gutmann, E.D.; Moulitsas, I.; Filippone, S. Fortran Coarray Implementation of Semi-Lagrangian Convected Air Particles within an Atmospheric Model. *ChemEngineering* **2021**, *5*, 21. https://doi.org/10.3390/chemengineering5020021

Academic Editor: Alessio Alexiadis

Received: 16 January 2021
Accepted: 19 April 2021
Published: 6 May 2021

1. Introduction

The Intermediate Complexity Atmospheric Research (ICAR) model is an atmospheric model that balances the complexity of more accurate, high-order modeling schemes with the resources required to model larger scale scenarios [1]. It is currently in use at the National Center for Atmospheric Research (NCAR) in the United States and has been used for simulations such as a year-long precipitation simulation of a 169×132 km section of the Rocky Mountains and precipitation patterns in the European Alps and the South Island of New Zealand [2–4]. ICAR has a three-dimensional model that handles a three-dimensional wind field, advection of heat and moisture. The ICAR model initializes the domain and tracks the changes of variables such as temperature, humidity, and pressure. The model is able to emulate realistic terrain by handling hills and mountains within the environment.

This paper aimed to examine the implications of adding semi-Lagrangian convected parcels that move through the existing three-dimensional Cartesian grid of the ICAR application. The ICAR model was implemented using Coarray Fortran, a partitioned global address space (PGAS) parallel programming model, to handle the communication required for the domain decomposition of the problem [5–7]. The PGAS model functions as a global memory address space, such that each process or thread has local memory that is accessible by all other processes or threads [8]. Coarrays offer high-level syntax to easily provide the benefits of the PGAS model and will be examined more later (Section 1.1).

This work performed an in-depth analysis of the performance and communication implications of adding semi-Lagrangian convected air parcels on top of a Cartesian atmospheric model. Within the atmospheric community, air parcel is the term used to refer to pockets of air, but due to the nature of the parcels used here and for sake of unity, parcels are referred to as particles.

Other work in the field has been done examining aspects of using Fortran coarrays and the performance cost of various semi-Lagrangian schemes. The Integrated Forecasting System (IFS), which is used by the European Centre for Medium Range Weather Forecasts (ECMWF), has examined the performance benefits of using coarrays and OpenMP within their large-scale model [9,10]. Similar work has been done within the Yi-He Global Spectral Model (YHGSM), which examined the benefits of changing the communication of the semi-Lagrangian scheme from two-sided to one-sided Message Passing Interface (MPI) communication [11]. MPI and two-sided and one-sided communication are discussed in Section 1.1.

A framework called Moist Parcel-in-Cell (MPIC) has been developed for convection of Lagrangian particles [12,13]. These papers focused on the convection physics and accuracy of the MPIC modeling. MPIC uses OpenMP for parallelization, and the research group has worked to integrate it with the Met-Office/NERC cloud model (MONC) infrastructure, which uses the MPI for communication [14]. Work that has closely looked at the results of scaling coarrays and Message Passing Interface (MPI) have done so looking at the Cellular Automata (CA) problem [15,16]. The ICAR model has also previously been used to examine the performance and ease-of-programmability comparisons between the MPI and coarrays [17]. This study showed that re-coding coarrays to an asynchronous two-sided MPI implementation, while having little impact on additional lines for the physics files, required 226 extra lines of code in the module that controlled the communication. This equated to ∼91% additional lines.

The use of particles in atmospheric science is more commonly applied to trajectory analysis using offline models such as the Hybrid Single-Particle Lagrangian Integrated Trajectory (HYSPLIT) model [18]. In the current implementation, the particles are directly coupled within the model analogous to the approach taken by [19] for chemical transport studies. While it is beyond the scope of the current study, future work could use this particle implementation to add a simple atmospheric chemistry module to ICAR to permit the study of atmospheric chemistry in this computationally efficient model.

The novelty of the work presented here is the in-depth analysis of the performance and communication cost of adding semi-Lagrangian air particles to the coarray ICAR model. We analyzed a variety of parameters, such as wind speed and moisture, to understand their effects on performance. We also examined the use of Fortran coarrays with two different communication back-ends, OpenCoarrays with one-sided MPI puts, and Cray's SHMEM-based communication. This helps showcase the performance of HPC clusters from two different manufactures, as well as the performance of a Free and Open-Source Software (FOSS) library with OpenCoarrays and the proprietary Cray SHMEM communication backend.

1.1. Fortran Coarrays

The performance of code has traditionally increased due to advancements in hardware or algorithms, but as time passed and the physical limits of hardware were being approached, this meant the end of Moore's law and Dennard scaling [20]. The move to parallel programming with multicore processors and off-loading to devices such as GPUs has meant that performance gains could still be achieved. Computing with MPI has been a traditional way to use multiple processors locally and communicate data across nodes. The MPI functions on the Single-Program Multiple-Data (SPMD) paradigm, where the user writes a single program that gets run by every processing element, called rank within the MPI, at the same time. Similar to the MPI, coarrays also follow the SPMD paradigm, but refer to the processing elements as images, instead of ranks. In the SPMDP paradigm, data are often split up amongst images, as in Figure 1, so that each processor can work on a chunk of the problem before communicating the results back to a main node.

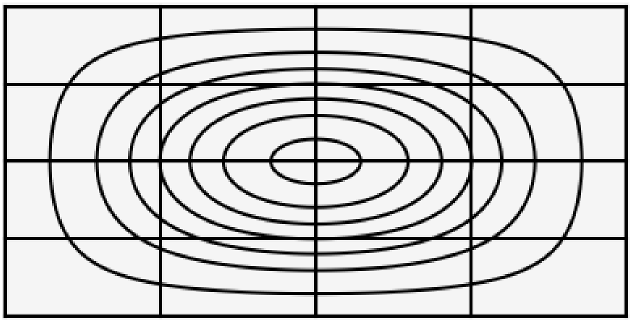


Figure 1. Overhead of the hill contour, distributed across 16 images.

Fortran coarrays started out as a language extension and were formally added to the Fortran Standard in 2008 [5,6]. Coarrays are similar in concept to the MPI, but offer the advantage of a simplified syntax with square brackets [17]. A variable without square brackets will always refer to the current image's copy of a coarray variable. Adding square brackets, such as `var[i]`, will allow any image to access or change the *i*th image's `var` variable data. For instance, the following code allows the first image to send the local data in the `cloud_moisture` variable to the second image's `rain` variable Figure 2.

```
real, allocatable :: rain[:]
...
allocate(rain[*])
if (this_image() .eq. 1) then
  rain[2] = cloud_moisture
end if
sync all
```

Figure 2. Coarray example of variable rain.

In this instance, the coarray variables are scalars, but they can also be arrays and even derived types. Coarrays allow blocking synchronization with the `sync all` command and offer the typical range of collective communication, sum, reduce, etc.

To perform the same action in the MPI, one could use two-sided or one-sided communication. Two-sided communication is communication where a sender and a receiver both need to make a function call for a message to be communicated. Two-sided communication can be done using blocking `MPI_Send` and `MPI_Recv` or non-blocking `MPI_Isend` and `MPI_Irecv`. One-sided communication is a Remote Memory Access (RMA) method to allow a single rank to perform communication between multiple ranks. For example, either a sender calls a `put` to send a message or a receiver calls a `get` to retrieve the information. This MPI method of calling `puts` and `gets` is equivalent to the PGAS of coarrays.

The example above could be done with an `MPI_Put`. A code snippet showing the equivalent method with one-sided MPI communication is shown below Figure 3.

```
use mpi_f08
type(MPI_Win) :: window
integer(kind=MPI_ADDRESS_KIND) :: wsize
integer :: disp_unit, ierr
real :: rain, cloud_moisture

...
call MPI_Win_create(rain, wsize, disp_unit, MPI_INFO_NULL, &
                    MPI_COMM_WORLD, window, ierr)
call MPI_Win_fence(0, window, ierr)
if (rank .eq. 0) then
  call MPI_Put(cloud_moisture, 1, MPI_REAL, 1, disp_unit, 1, &
               MPI_REAL, window, ierr)
end if
call MPI_Win_fence(0, window, ierr)
...
```

Figure 3. One-sided MPI example.

The ICAR model uses coarrays to handle domain decomposition; see Figure 1. The ICAR mini-app used here has two different topologies, one that contains a single hill, which is split among the available images, and one that is a flat plain. Figure 4 shows a side view of the hill with particles, and Figure 1 shows a top-down view of the hill's contour lines and how the distribution of the domain across 16 images would occur. The physics in the ICAR model has a variety of parameters that need to be updated using numerical analysis and stencils. The stencils have halo regions of depth one that are contained in variables of the *exchangeable_t* type. The type *exchangeable_t* has been created based on object-oriented principles. Each variable of type *exchangeable_t* has a local three-dimensional array that represents parameters such as water vapor, cloud water mass, or potential temperature. It contains halo regions for the north, south, east, and west directions and has procedures, such as `put_north` and `retrieve_south_halo`, to help facilitate the halo communication. The ICAR model uses coarrays to update halo regions; see Figure 5. These halo regions are coarrays that perform "puts" during the update phase. The update phase occurs between the initialization step and the start of the computation and after each computational time step.

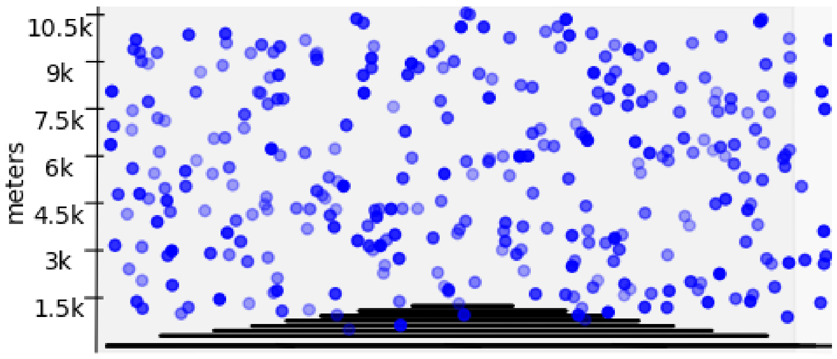


Figure 4. Side view of hill and air particles.

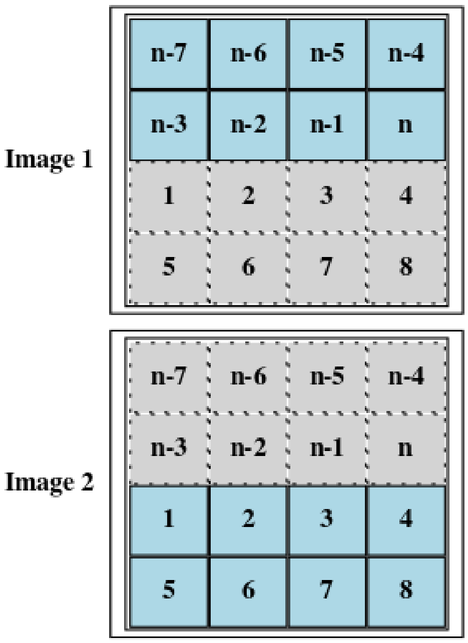


Figure 5. Halo regions to deal with trilinear interpolation.

To handle the movement of the particles, a number of additions needed to be made to the ICAR application. A new type `convection_exchangeable_t` was created that was based on the `exchangeable_t` type. The new type `convection_exchangeable_t` changed the coarray halo arrays to coarray buffers. These are buffers into which the neighboring images will directly put particles that need to be transferred across the boundary. When a particle's horizontal x or y values are greater than the image's local grid, they are loaded into an "input" coarray buffer of the neighboring region. Then, at the end of the particle computation phase, the buffer holding the particles is iterated over and unloaded into the image's array of particles. This is a key difference from the original ICAR model, where there is an explicit compute phase and then a communication phase of the halo regions. With the particles, there is an explicit compute phase and unloading phase.

A complication that occurred from adding semi-Lagrangian particles to the code is that a particle could exist between the boundaries with a halo region of one. Figure 6 indicates how an air particle would look as it exists between the boundaries of Image 1 and 2. A halo region of one would solve this issue because the particle could access the data points 2,3,n-1,n-2.The complication occurs because horizontal forces are applied to the particle, and further calculations for its pressure, saturated mixing ratio, etc., are needed. This sometimes results in a particle in Image 2 needing the points n-1, n-2, n-5, and n-6. This is why the boundary regions within the `convection_exchangeable_t` type were all changed from one to two. While the original code only had halo regions in the north, south, east, and west directions, four additional coarray buffers were needed to handle the diagonal movement of the particles. Since the u and v wind fields work simultaneously, they can move the particle diagonally. If a particle is near the "x" and "y" boundary at the same time, there is a possibility that the particle will jump to a diagonal neighbor.

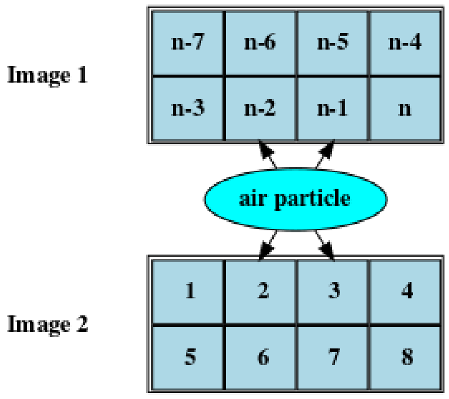


Figure 6. Lagrangian air particle existing on the boundary.

1.2. Convection

As the introduced semi-Lagrangian convected air particles warm, they rise and cool until their momentum brings them to an area where the surrounding region is cooler and they start to fall. As they fall, the particles warm until they reach an area where they are warmer than the surrounding environment, and they begin to rise. This oscillation causes a number of transformations within the air particle. As the air particle is rising, as long as its relative humidity is below 100%, it is considered a dry air particle and will cool at the adiabatic lapse rate with a constant mixing ratio and potential temperature [21]. Potential temperature is defined as the temperature a particle of air would be at if it were brought to a standard reference pressure P_0, defined as 1000 hPa. The equation used for calculating the potential temperature is $\theta = T(\frac{P_0}{P})^{R/c_p}$, where T is the absolute temperature of the air particle with pressure P. $R/c_p = 0.286$, where R is the gas constant of air and c_p is the specific heat of air at a constant pressure [22,23].

When the relative humidity increases over 100%, the air particle is saturated, and the water vapor within the particle will be converted into liquid cloud water. This phase change causes a change in temperature and potential temperature while pressure stays constant. As the particle falls, this process is reversed.

The air particles are implemented in a semi-Lagrangian manner. They are kept within an unordered buffer and operate essentially independently of the environment. This is allowable because as air particles rise, they do so in an adiabatic manner. It is standard practice to assume that the convected air particle's heat exchange with the surrounding environment is so small that it can be neglected [23].

2. Materials and Methods

In accordance with MDPI research data policies, the source code used and data sets created for this paper are publicly available [24,25].

2.1. Air Particle Physics

The framework for handling the physics of the air particle is roughly broken into the steps depicted in Figure 7. An overview of the process will be discussed before going over the equations of state. The first calculations, shown in the Move Particle Section, are to find the buoyancy force in the z-direction and the horizontal forces from the wind. This calculates the trajectory of the air particle and where it should be placed. Once the particle is moved, the changes in the physical properties of the particle are calculated. This process is depicted in the lapse rate section of the figure. First, the dry lapse rate is calculated, and if the relative humidity stays below 100%, the process continues. The particle is checked to see if it has moved beyond the image's boundaries and communicated if so. If the relative humidity goes above 100%, then an iterative process is started to find the correct amount of phase and lapse rate change.

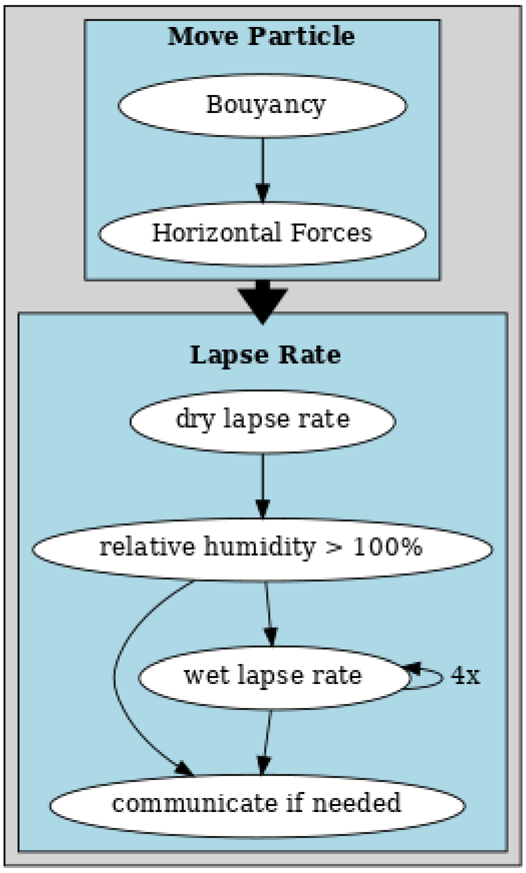


Figure 7. Overview of the computational phase of a particle.

The two equations of state used during this process are presented here. The equation of state used to formulate important properties of the air particle is the ideal gas law, where P is pressure, V is volume, n is the amount of moles, R is the ideal gas constant, and T is temperature.

$$PV = nRT$$

The other equation of state is the first law of thermodynamics and is used such that the change of internal energy within a closed system is equal to the quantity of energy from heat, minus the amount of thermodynamic work done on its surroundings [26], where U is the internal energy of a closed system, Q is the heat, and W is the thermodynamic work.

$$\Delta U = Q - W$$

From these equations of state, the following formulas have been derived and are commonly used within the meteorological community. The subsequent equation is how the buoyancy force for movement of the air particle is calculated. The downward force on the air particle is $\rho g V$, where ρ is the particle's density, g is gravity, and V is the volume [23]. When an air particle rises, it displaces an equal volume of ambient air, so the downward force on the displaced air is $\rho' g V$, where ρ' is the displaced air's density. The upward force is the same for both the air particle and the displaced air and equal to $-V(\delta p/\delta z)$. For the air particles, this is the only force in the z-direction, and so, the particle's equation of motion is:

$$\frac{d^2}{dt^2} = F_b = g\left(\frac{T - T'}{T}\right)$$

where T is the particle's temperature and T' is the environment's.

The force is applied at a constant rate through the time period, where d is displacement, $a = F_b$ is acceleration, and v_0 is the previous time step's velocity.

$$d = v_o t + \frac{1}{2} a t^2$$

2.1.1. Lapse Rate

After the buoyancy force has been applied and the air particle has been moved to its new elevation, the change in temperature is calculated. The following equation is applied to calculate the new pressure:

$$p = p_0 - z_d \times \frac{g \times p_0}{287.05 \times T_0}$$

where the gravity variable is $g = 9.81$, T_0 and p_0 are the original temperature and pressure, and z_d is the distance the particle is displaced in the z-direction.

For the change in temperature of a dry air particle, the Exner (Π) function is used. The Exner function is defined as the following, where p is pressure, R_d is the gas constant of dry air, c_p is the heat capacity of dry air, T is temperature, and θ is the potential temperature:

$$\Pi = \left(\frac{p}{p_0}\right)^{R_d/c_p} = \frac{T}{\theta}$$

Rearranging, the new temperature is obtained from $T = \frac{\Pi}{\theta}$.

The next step is to check if the relative humidity of the air particle has exceeded 100%. This saturated mixing ratio is obtained from a function within the ICAR code that calculates it from temperature and pressure arguments. The relative humidity is calculated by dividing the particle's current water vapor by the saturated mixing ratio and multiplying by 100 to get the percentage. If the relative humidity is above 100%, then the extra humidity is taken and changed to cloud water. If the relative humidity is below 100% and there is cloud water in the particle, the cloud water will be converted back into water vapor so as to keep the relative humidity as close to 100% as possible. The change in temperature is calculated as follows, where q is heat, C_p is the specific heat of air at constant pressure, T is temperature, and P is pressure. C_p is calculated from $C_p = (1004 * (1 + 1.84 * v))$, where v is the mass of water vapor divided by the mass of dry air [22].

$$\Delta q = C_p \times \Delta T - \frac{\Delta P}{\rho}$$

For this, a specific latent heat of 2.5×10^6 J kg^{-1} is used for the particle. There is no change in pressure during the moist adiabatic process, so the $\frac{\Delta P}{\rho}$ expression is zero. The potential temperature does not change during the dry adiabatic process, but for the moist adiabatic process, the Exner function is used to calculate the new potential temperature.

2.1.2. Orographic Lifting

In addition to forces applied in the z-direction, a wind field is applied for movement in the x- and y-direction. As the air particle moves through the environment, orographic lifting will occur when the particle hits a barrier and has to rise to flow over the barrier. The ICAR application keeps track of the altitude of the ground, and when an air particle is moving in the x- and y-direction due to wind flow, the air particles will rise or fall accordingly. Bilinear interpolation is used in the x- and y-direction to estimate the altitude of the ground. The orographic lifting is simplistic in that the altitude of the air particle is changed the exact amount as the change in altitude of the ground floor.

2.2. Methodology

Within this section, a detailed account of how the air particles are handled is given. It is the purpose of this study to implement and add the fundamental components of functioning semi-Lagrangian air particles to the ICAR mini-app. Every air particle is initialized by randomly choosing a position within a coarray image's subdomain. The program uses Fortran's intrinsic random number generator with a seed of -1. The intrinsic `random_init` is called with both values equal to true.This is to ensure that each image has distinct random values when calling the function and that each time the program is run as a whole, the random numbers will be the same to ensure reproducibility between runs.

At initialization, an air particle of type *convection_particle* is created using the values of the environment at its current position. The *convection_particle* type has variables to keep track of the particle's x, y, z position within the image's grid, the height of the particle in meters, the u, v, w values of the wind field, the velocity, pressure, temperature, potential temperature, water vapor, and relative humidity. For orographic lifting, the height of the ground below the particle is also kept. After the initial x, y, z position is randomly generated, trilinear interpolation is used to gather the values of *convection_particle*'s types. See Figure 8 for an example of trilinear interpolation. For example, the initial x, y, z position would be the red C in Figure 8. Calculating C requires eight regular grid points from the surrounding particle's area, and five of those points are listed in the figure as $c_{000}, c_{001}, c_{100}, c_{110}, c_{111}$. Note, when using trilinear interpolation, one needs to be sure that if $\exists i \in \{x, y, z\}$, such that $floor(i) = ceiling(i)$, then bilinear interpolation is used. For example, if c_{000} and c_{001} were equal, then $floor(x) = ceiling(x)$ would be true. In that case, the trilinear interpolation to calculate the value in a cube would have to be switched to a bilinear interpolation, which finds the value in a square. Likewise, if $\exists i \in \{x, y\}$ such that $floor(i) = ceiling(i)$, then bilinear interpolation needs to be substituted for linear interpolation. If in the rare case that $\forall i \in \{x, y\}$, it is true that $floor(i) = ceiling(i)$, there is no need for interpolation. Finding the right level of interpolation is done to avoid divide by zero errors from $floor(i) - ceiling(i) = 0$.

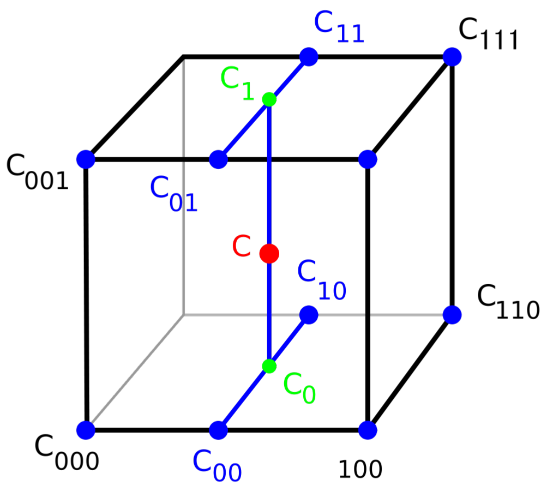


Figure 8. Trilinear interpolation [27].

When running performance tests, to keep the communication costs of each image as similar as possible, any particle that goes beyond the original domain is wrapped around to the other side. In Figure 1, a birds-eye view of the domain is shown and how it would be split up amongst 16 images. If a particle were to leave the top of that image heading north, it would continue to head north from the corresponding bottommost image. This allows for the particle count over the entire domain to remain constant. Likewise, if convection causes a particle to go off the top or bottom of the z-axis, a replacement particle is created.

There were two different types of initialization of atmospheric and environment variables that were tested in our model. One was an ideal case, where the potential temperature θ was set at 300 K for the entire model. The other setup was done by using a real atmospheric sounding of Seattle. The soundings were extracted from the University of Wyoming's weather website; a site with atmospheric soundings from locations around the world [28]. This was done using Python libraries BeautifulSoup, to extract the sounding data, and Pandas to handle the data manipulation and preprocessing. The atmospheric sounding was interpolated to produce data at every meter of elevation and written to a file for later use. The atmospheric sounding of Seattle gave the real-world potential temperature at elevations above 1600 m. This was beyond our model's upper limit, so the extra data were unused in this case, but could be used if the domain of the model were expanded. Potential temperatures below the sounding's lowest elevation were set at 300 K.

2.3. Hardware and Software

It has been noted in the literature that different hardware platforms, coupled with different software environments and compiler vendors, can result in performance variation [29]. For our experiments, we used two different HPC systems and employed two different compilers, thus covering both open-source and proprietary software implementations.

Cheyenne was one of two HPC clusters used. Cheyenne is a 5.34 petaflop SGI machine run by the National Center for Atmospheric Research (NCAR). The SGI ICE XA cluster has 145,152 Intel Xeon cores. Each node is dual socket with two Intel Xeon E5-2697V4 (Broadwell) processors, and a Mellanox EDR InfiniBand interconnect is used. The project uses the `caf` compiler wrapper provided by the Sourcery Institute with the OpenCoarrays library. The OpenCoarrays library is an open-source project that provides the coarray implementation for GNU gfortran [30]. The underlying compiler that `caf` wraps is GNU gfortran 10.0.0 with flags `-fcoarray=lib` and includes paths to the OpenCoarray library. `caf` was used with flags `-cpp -O3` for preprocessing and optimization. The project used OpenCoarrays 2.9.0, which itself was built with MPICH 3.3.2. OpenCoarrays handles the communication requirements of coarrays with a number of possible communication backends. This project uses the MPI backend with ones-sided RMA puts and gets, though it is possible to use OpenSHMEM.

A Cray cluster was used with nodes of two Intel Xeon Broadwell chips for a total of 44 cores. The Cray Fortran compiler Version 11.0.0 was used with flags `-e F -h caf -O3` for preprocessing, coarrays, and optimization. Cray's underlying communication layer for Fortran coarrays is the SHMEM library, standing for Symmetric Hierarchical MEMory [31]. SHMEM was created to implement one-sided PGAS routines. When accessing intranode coarray memory, the program quickly ran out of the default allocation of memory. This led to `lib-4205 : UNRECOVERABLE library error: The program was unable to request more memory space`. This issue was fixed with the help of an environment variable that influences the amount of available memory. By setting `PGAS_MEMINFO_DISPLAY` to one, the system report information on the PGAS library was displayed. It showed that the symmetric heap was equal to 65 MB per process. By increasing the memory by setting `XT_SYMMETRIC_HEAP_SIZE` to `500M`, the program was able to procure the memory it needed and run successfully.

When compiling with Cray, the flags `-Rbcps` were initially used. They allowed for run-time checks of array bounds (b), array conformance checking (c), check array allocation status and other items (p), and character substring bounds (s). After initial development, these flags were removed. Flags that were also used included `-e F`, which turns on the preprocessor expansion of macros in Fortran source lines. This was required to handle macros for assertions, changing Cray output to files, and to delay PGAS synchronization.

If the `_CRAYFTN` macro was defined, the following code would be called:

```
call assign ('' assign -S on -y on p:\%.txt'', ierr)
```

This would ensure repeat data values, such as "8 8 8", would be output with spaces between them. This is not the normal Cray behavior, which would output those values as "3*8", presumably for the sake of saving file space. While helpful for memory and time saving, it does make final data analysis more difficult. The additional flag `-h caf` was used, which allows the compiler to recognize coarray syntax. This is a default flag, but when it is defined on the command line, the macro `_CRAY_COARRAY` gets set to one [32]. An important directive to use when using coarrays is `!DIR$ PGAS DEFER_SYNC`. This ensures that the synchronization of data is delayed until a synchronization point [33,34]. This can be verified using Cray's profiling tools to see if the PGAS library calls are the non-blocking routine `__pgas_put_nbi`.

2.3.1. Profiling Tools

Basic performance measurements were done using hand-coded `system_clock` timers. When more sophisticated profiling and analysis was needed, HPE Cray's CrayPat performance analysis tool and the Tuning and Analysis Utility (TAU) were used. TAU was developed by the University of Oregon Performance Research Lab, the LANL Advanced Computing Laboratory, and The Research Center Julich at ZAM, Germany [35]. TAU Commander is TAU's tool that packages all the capabilities of TAU in a simple package for users. Depending on the user's needs, TAU Commander builds the required tools and allows for flexibility in choosing the combination of profiling, tracing, and other counters required. This project, with its use of CMake and the OpenCoarray Library, required a few tweaks to get everything working.

TAU's Fortran compiler wrapper `taucaf` was used as the Fortran compiler when building the ICAR application. The wrapper was missing the OpenCoarray library though, so the system paths needed to be added, and the `libcaf` library had to be loaded. A simple way to achieve this is to call "`caf -show`" from the terminal. The `caf` compiler is itself a wrapper to call `gfortran`, so this option will expand the full command needed to use the OpenCoarray Library. By copying that whole output, without the `${@}`, and then adding it to the `CMAKE_Fortran_FLAGS` Cmake variable, the project will be able to load and link everything correctly. The `caf` compiler is a bash script to implement the wrapper correctly, and the `${@}` is a special parameter that expands the positional arguments, every argument after the script's name [36]. The users can inform TAU of which information that they want in a few ways. One is using environment variables, such as `TAU_COMM_MATRIX`, to profile the communication matrix or running `tau_exec` with command line arguments. This was used in conjunction with the other default values, which turn on sampling and generate a performance profile.

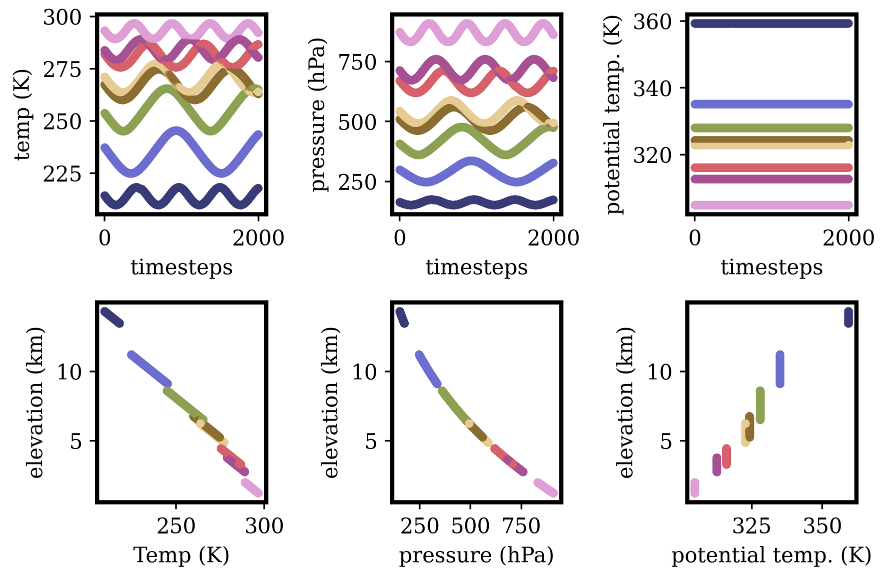
Craypat was designed by Cray to allow the instrumentation of an application without recompilation, only through linking [37]. Using the original binary, `pat_build` was used with the `-g caf` flag to create a new executable that would produce profiling data when run. There are many additional profiling flags that can be passed; this project would sometimes add `heap` for a closer look at the memory behavior. The new binary can then be run using `aprun` in the same usual fashion "`aprun -n $num_procs ./test-ideal+pat`".

3. Results and Discussions

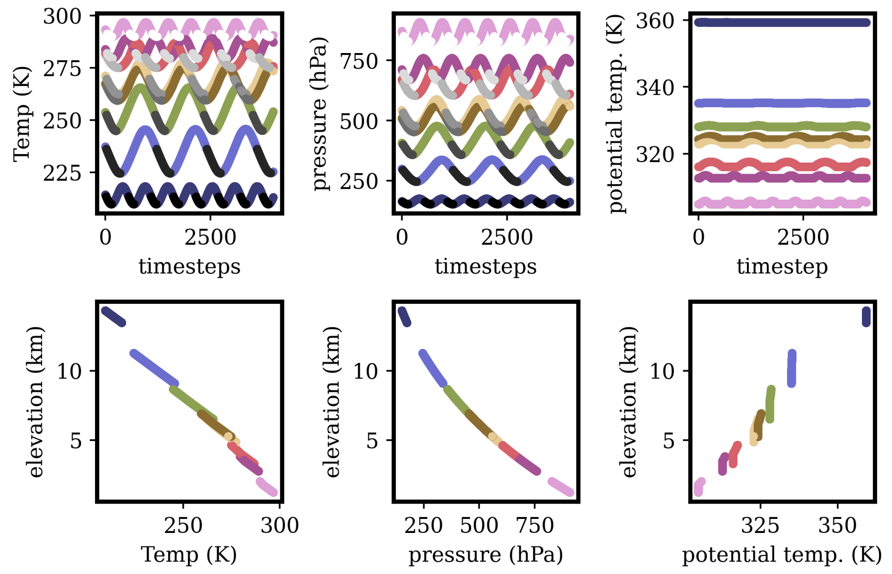
3.1. Validation

Validation of the model was done by ensuring that the air particles showed the same properties as real-world data and other modeling equations. The domain size used for the validation runs had dimensions $nx = 20, ny = 20, nz = 30$. Each cell corresponds to a length, width, and height of 500 m, and this is consistent throughout the modeling. To test the dry air particles, the hill was removed from the model, and the atmospheric sounding of Seattle was used. During the dry particle runs, the water vapor in the environment and particles was then set to zero to remove all moisture. For the wet particle runs, the water vapor in the particles was set to the saturated mixing ratio of the environment where its coordinates are. This means that if the particle stayed at the current temperature and pressure, the relative humidity would be 100%. Given that particles were initialized with an initial upward velocity of 5 m/s, the wet particles would become saturated immediately. The runs were done for 200 time steps of 20 s each for the ICAR model and time steps of one every second for the particles, totaling 4000 steps for the particles.

The results of the dry and wet runs are shown in Figure 9a,b, respectively. Both figures show a behavior that would be expected for a working model. The potential temperature of a particle experiencing dry adiabatic lifting is constant, which is what occurs in Figure 9a. The potential temperature only changes during the moist adiabatic lapse rate, during the process of the phase shift from water vapor to rain water, or vice versa. The third graph in Figure 9b confirms this behavior. Additionally, the trajectories of the temperature and pressure of the saturated and unsaturated particles follow what would be expected [28].



(**a**)



(**b**)

Figure 9. Particle attributes over time and varying elevation for: (**a**) eight dry air particles (**b**) eight saturated air particles; note that the grayscale indicates the particles are saturated.

3.2. Halo Depth

The addition of air particles to the ICAR model required the increase of the halo depth from one to two, and this study wanted to quantify the effects of increasing halo depth size.

Increasing the halo depth size results in creating more data that need to be communicated during the halo exchange. Another benefit of understanding the performance cost is that different numerical methods, possibly used for greater precision, could require larger stencils that would also increase the halo depth size. In Figure 10, the performance is shown as the depth of the halo region is increased using grid dimensions of size 500 × 500 × 30 and 2000 × 2000 × 30. For those two sizes, this graph shows the scaling of one node using 44 cores and two nodes using 22 cores each. The reason for increasing the node count while keeping the work per image constant is to summarize the cost of intranode communication with halo depth variance. Figure 10 shows that while increased halo depth affects run-time, splitting the work across nodes in this case had minimal effect on performance.

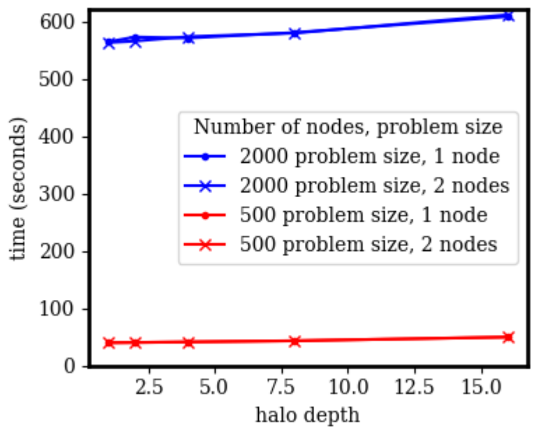


Figure 10. Effects of changing halo depth using a Cray machine.

3.3. Particle Count and Wind Speed

In addition to halo depth, the wind speed and particle count are variables that could possibly affect performance. As wind speed increases the amount of particles to communicate across, image boundaries will naturally increase. The particles are moving faster and will cross an image boundary in a shorter time frame. A larger number of particles would also translate to a larger amount of particles being transferred across image boundaries. Additionally, as the number of particles increases, it is intuitive that the computation cost to extract environmental information for the particle physics would also grow. The graph in Figure 11 attempts to isolate the effects of adding particles to the model by looking at the performance cost of increasing the particle count of two types of particles. Additionally, the graphs in Figure 12 attempt to examine the effects of wind on the number of particles communicated, how that communication looks, and the performance.

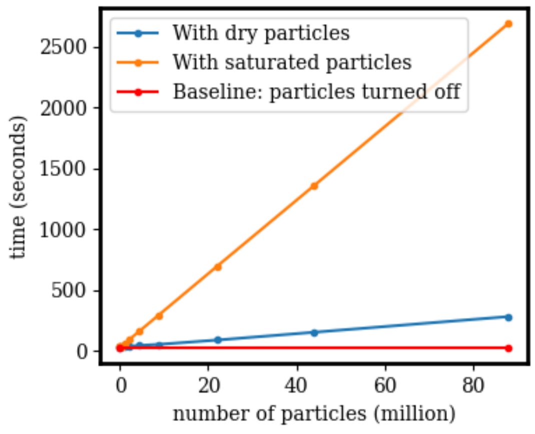
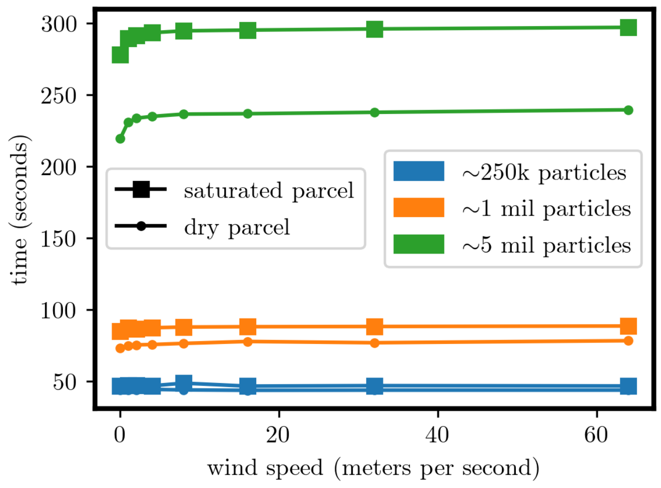
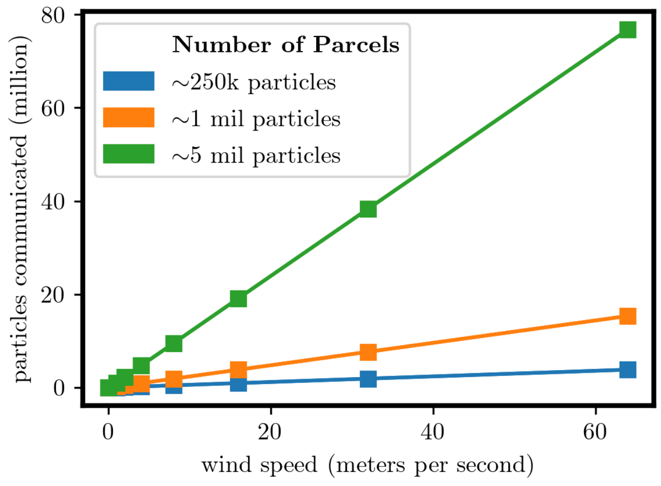


Figure 11. Performance effects of increasing the particle count using a Cray machine.

Figure 11 shows the computation costs of saturated and unsaturated particles as the number of particles increases. For these runs, wind speed was changed to zero, so no communication cost was measured. This figure was run on a single node of the Cray machine with 44 images. The number of particles per image ranged from one to two million, for a total of 88 million particles. The domain was 200 × 200 × 30, and the problem was run for 200 time steps, with 19 additional time steps of 1 s for the particles. There was clearly linear growth for both dry and saturated air particles. The cost of saturated particles grew at a faster rate, with the time cost of 88 million particles being ∼9.5 times the dry particles. One-point-one million saturated particles took ∼1.7 times, and the dry particles and 2.2 million saturated particles took ∼2.5 times. Therefore, around 1.5 million particles, the cost of saturated particles doubled the cost of dry particles. The additional performance cost of the saturated air particles was intuitive since the anytime a particle's relative humidity was over 100%, an iterative process was used to calculate the exact change in temperature and water mass.



(**a**)



(**b**)

Figure 12. On a Cray machine. (**a**) The effects of wind speed with varying particle count using dry and saturated particles. (**b**) The number of total particles communicated with the change in wind.

To examine the effects of wind speed on the model, we first turn to Figure 12a. This figure was created using a problem domain of 500 × 500 × 30 over 200 time steps. The runs were done with initially dry or saturated particles and a total particle count of around 250 thousand, 1 million, and 5 million particles. The 250 thousand particles were chosen as the starting point since that would equate to a particle per horizontal grid cell. In all of the varying particle count, the results showed the expected behavior of saturated particles taking more time than unsaturated. As the particle count grew, so did the difference between dry and saturated. This was also an intuitive result since each saturated particle had to perform an iterative process to correctly convert water vapor and cloud water. What was more unexpected was that after an initial performance hit from increased communication, there was negligible cost as the wind speed increased. This was most clearly shown in the five million particle run, where from 0 m/s to 1 m/s, there was a 4.03% performance cost, but from 4 to 8 m/s, there was a 0.45% increase.

Examining Figure 12b shows a linear increase in the number of particles communicated when the wind speed was increased in the model. For five million particles, a change of 0 to 1 m/s in wind speed resulted in zero total particles communicated to 1,048,073. That initial cost of one million particles communicated took an extra 4.03% of the initial time. Compare that with the 0.45% cost when moving from 4 to 8 m/s, even though 4,819,692 additional particles were communicated. This indicated that there was a cost to the communication phase of the particles, but that it did not matter how often they were communicated.

All the previous runs were done on one node, meaning that the communication would all be inter-node. This choice was made due to the exponential growth of runs that would have to be done if this were scaled out to include more nodes. To get a feel for the intra- vs. inter-node communication though, additional runs were done holding the image count constant, but adding two and four nodes. The runs were done over 200 time steps with a wind speed of 8 m/s and with dry air particles. Table 1 shows that running a total of 44 images with one or two nodes leads to approximately the same performance while the jump to four nodes results in a 29.5% performance increase.

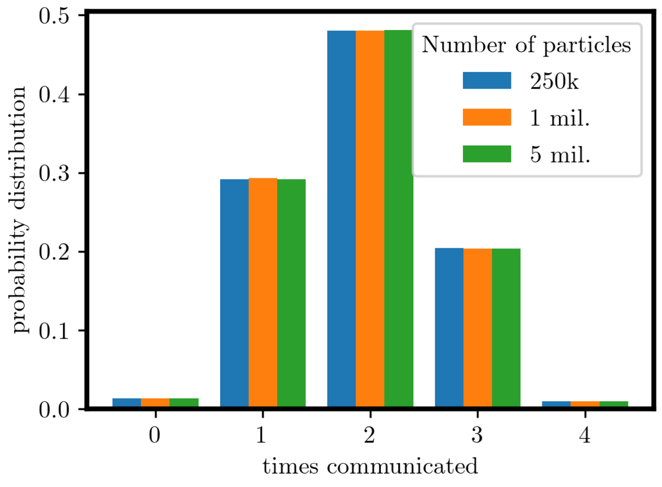
Table 1. Inter- vs. intra-node communication.

Nodes	**Images**	**Time**	**Average Time (s)**	**Percent of Time**	**Average Time (s)**	**Percent of Time**	**Average Maximum**
		(s.)	**PGAS**	**PGAS**	**__pgas_sync_all**	**__pgas_sync_all**	**Memory Usage (MBs)**
1	44	235.58	12.37	5.0%	6.47	2.6%	100
2	44	234.93	—	—	—	—	—
4	44	165.55	24.49	13.0%	10.88	5.8%	468.8
2	88	117.61	—	—	—	—	—
4	176	59.6	—	—	—	—	—

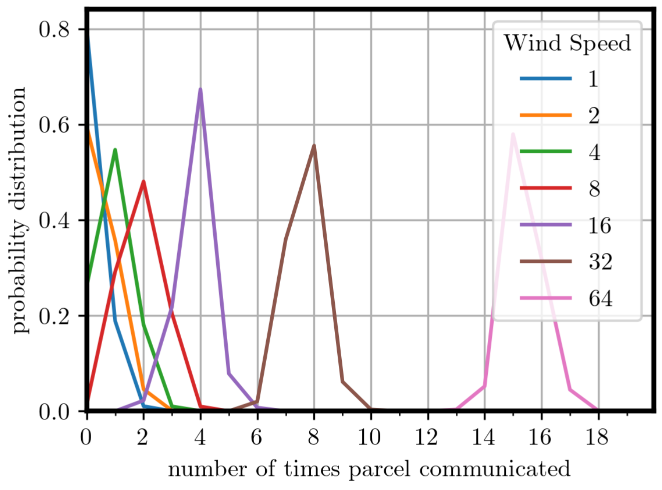
This jump in performance was a surprising result and required further investigation. Cray's Performance Tools were used to look closely at the data to understand what was occurring. The percent of time spent using PGAS functions was as expected much higher for the four node version, 13.0% vs. 5.0%. While the largest PGAS function in the single-node version was `sync all`, at 2.6%, for the four-node version it was `co_broadcast` at 6.0%. The `co_broadcast` was used in the initial setup phase of the model to communicate physics calculations to all images. The table conveys that `sync all` within the `retrieve` function used by the particles took up most of the wait time, but the actual communication of the data did not. The `__pgas_put_nbi`, used by the particle communication function, took 0.0% and 0.1% of the time for the single- and four-node runs. From testing, it appeared that in `__pgas_put_nbi`, `nbi` likely stood for the non-blocking interface, since it was called when a non-blocking put was specifically requested with the `!DIR$ PGAS DEFER_SYNC` directive. Two things should be noted from this part of the analysis. First, the particle communication in this method took almost no time, and it was the syncs during the unloading of the communication buffers that took the time. Second, despite PGAS taking much longer on the four-node version, it still ran 29.5% faster. It could possibly be from the larger amount of memory each image used, but further investigation would be needed. This would fall in line with what has been noted before, that increasing memory per core will increase performance before reaching saturation [38].

Analysis was done on the distribution of the number of particles communicated. Figure 13a was created using a base wind speed of 8.0 m/s on a 500 × 500 × 30 domain. This figure exhibits how with varying particle counts, the distribution of how many times a particle is communicated will stay relatively constant. The consistency in the distribution suggests we can fix the particle count, without loss of generality, to perform the runs done in Figure 13b. Figure 13b was made with a particle count of 250 thousand. Figure 13b shows the probability distribution of the number of times a particle is communicated as the wind speed changes. Again, this confirmed that particles were being communicated as expected. These figures and Figure 12a,b express that despite linear growth in the number of particles communicated with an increase in wind speed, only the initial increase had a noticeable effect on the runtime.

Particle Communication Distribution



(**a**)



(**b**)

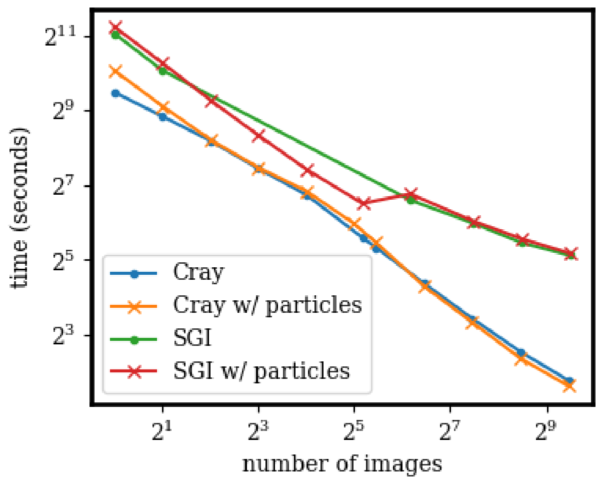
Figure 13. (**a**) Distribution of particles communicated with the change in particle count. (**b**) Distribution of particles communicated with the change in wind speed.

3.4. Scaling and Speedup

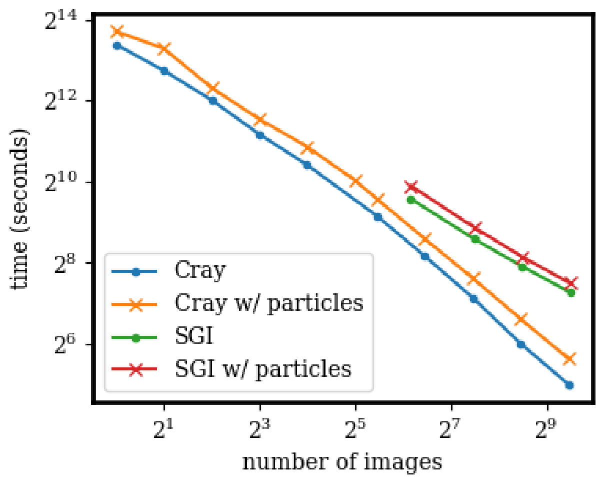
As is common practice, we present both strong and weak scaling results to evaluate the performance. Strong scaling suggests that the overall problem size is fixed, and performance is assessed by increasing the number of images. In weak scaling, the size of the problem is fixed per image; therefore, the overall problem size is scaled along with the number of images. The following strong scaling runs were done on a 500 × 500 × 30 and 2000 × 2000 × 30 domain size on both the CRAYand SGI HPC systems. The weak scaling runs were done using 12 thousand and 768 thousand grid points per process. Both strong and weak scaling were run for 200 time steps with each time step being 20 s. The convected air particles' time steps were run every second.

The strong scaling experiments, Figure 14a,b, were created using one image and doubling until 36 images. After that, the full processor count of the nodes was used, which for the Cray cluster was 44 processes times the node count and for the SGI cluster 36 processes times the node count. The number of nodes was doubled every time, up to 16 nodes with a total of 704 processes.

Strong Scaling



(**a**)

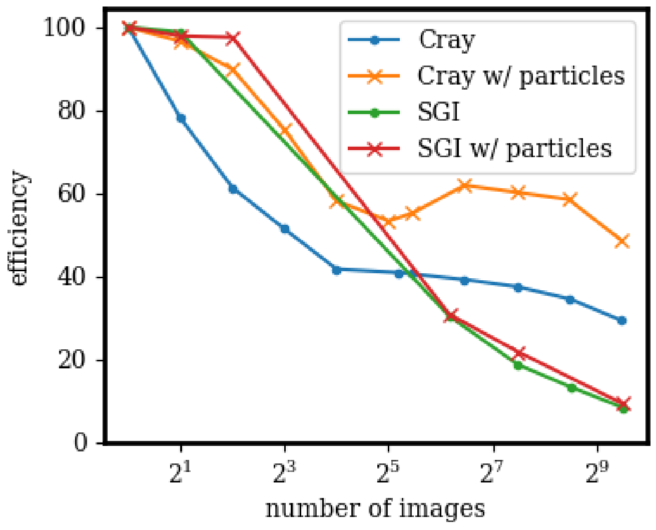


(**b**)

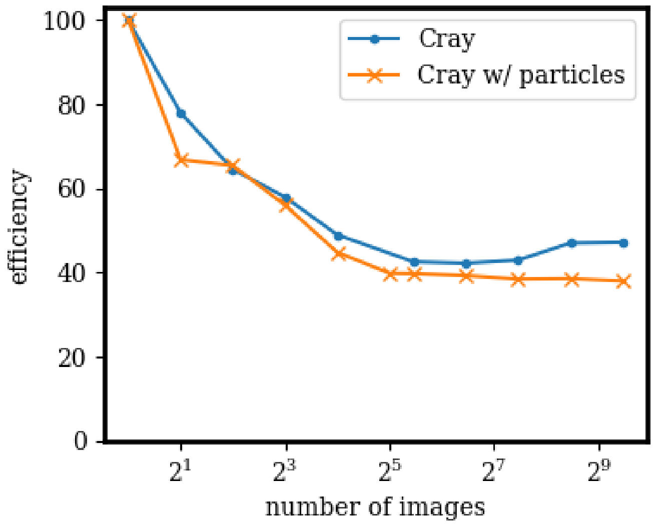
Figure 14. Log-log graphs of the domains of (**a**) grid dimensions 500 × 500 × 30 and (**b**) grid dimensions 2000 × 2000 × 30.

In Figure 14a, the air particles are one per horizontal grid cell. This graph shows that the addition of particles did not really affect the runtime and scaled at the same pace as the particle-less runs. Figure 14b shows that the addition of particles had a consistent cost that did not disappear. However, it also shows that the addition of particles did not affect the scaling on the larger domain size. The strong scaling efficiency, defined as $\frac{T_1}{n \times t_n} \times 100$, is shown in Figure 15a,b. T_1 is the time to complete the work using one processor; n is the number of processors used; and t_n is the amount of time n processors took to complete. Figure 15a shows that the strong scaling efficiency for the SGI machine did not seem to be affected by the addition of particles. On the Cray machine, the efficiency of the run with particles actually increased. This actually switched around with the larger problem domain, where the addition of particles made the efficiency slightly worse. This seemed to indicate that as the problem size grew, the efficiency cost of the addition of particles would be less and less.

Strong Scaling Efficiency



(**a**)

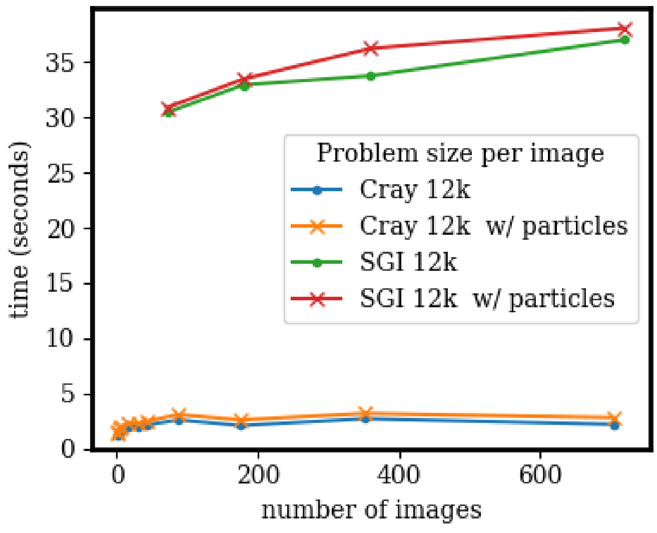


(**b**)

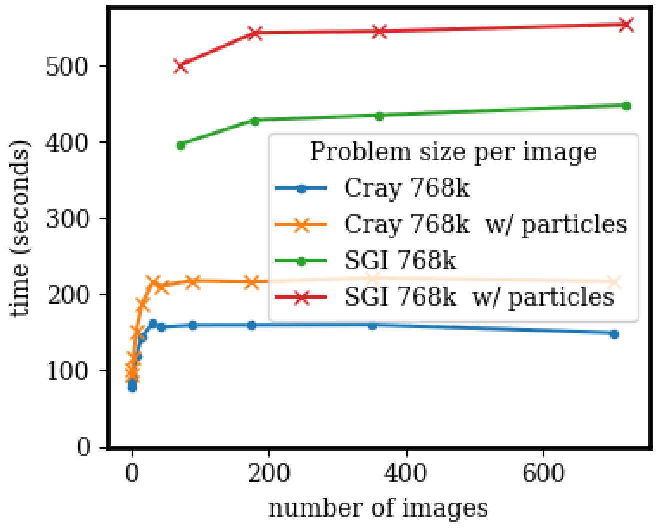
Figure 15. (**a**) Grid dimensions 500 × 500 × 30. (**b**) Grid dimensions 2000 × 2000 × 30.

In Figure 16a,b, the weak scaling results are shown. When the runs were done with particles, a particle per cell was used. Figure 16a starts at the size 20 × 20 × 30, and Figure 16b starts at 160 × 160 × 30. In both figures, the runtime increases until the node processes are all used, at which point the scaling generally levels out. This first increase indicated better cache use and lower communication cost of a single processor. Figure 16a's runtimes are rather low, ranging from 1.15 to 3.2 s. This might be the reason the weak scaling curve never flattened, but oscillated up and down. Figure 16b on the other hand has a weak scaling curve that flattens as all of the first node's 36 or 44 processors are being used and stays flat. Thus, it can be said that linear scaling was achieved and that the ICAR application with and without particles should scale well with larger processor counts. This matched the strong scaling results as well.

Weak Scaling



(**a**)



(**b**)

Figure 16. (**a**) Twelve-thousand grid cells per process; (**b**) 768 k grid cells per process.

4. Conclusions

The strong and weak scaling graphs showed that the ICAR model scales well and that the addition of the semi-Lagrangian convected air particles does not affect the scaling. The weak scaling indicated that the framework of communicating the convected air particles immediately upon leaving an image's boundary is an effective way to take advantage of scaling. The strong scaling efficiency suggested that there is better efficiency as the problem size grows. This would need to be tested on larger problem sizes, but the scaling for 2000 × 2000 × 30 was better than 500 × 500 × 30. Furthermore, at 2000 × 2000 × 30, Figure 15b suggested that the scaling efficiency levels off for runs with and without particles, which is desirable.

Investigating the particle communication showed that there was an initial performance hit from communicating the air particles. After that original hit though, increasing the number of times a particle was communicated across boundaries had a minimal effect on the performance. The cost of the communication of the air particles really comes from the cost of the synchronization points. The synchronization time of the air particles took 2.6% and 5.8% of the time for one and four nodes. Communicating the particles across the boundaries took 0.0% and 0.1% percent of the time; note this percentage and the rounding were done by the performance tool.

This work showed that there was a performance cost of adding air particles, but that it did not affect the HPC scalability of the original program. Scaling to a large number of images is a very important feature of any scientific code, even more so in our case where the underlying problem is that of downscaling atmospheric models. Hence, if the introduction of semi-Lagrangian convected air particles did not scale well, our approach would conflict with the primary goal of the underlying model.

Within the ICAR model, it indicates the benefit of spending more time in the future to make some of the processing more efficient. Currently, the air particle physics are calculated every second, and comparing that with the environment time step being 20 s, there is room for improvement. If additional features of air particles are deemed helpful for the ICAR model, it shows that they should not affect scaling; for example, the extra computation from calculating the coriolis force due to the Earth's rotation. Additionally, it would be beneficial to have the particles interact more with the surrounding environment. Currently, the environment is used to calculate buoyancy forces and saturation ratios, but once cloud water is converted, the particle never "rains".

Author Contributions: Conceptualization, S.R., E.D.G., I.M., and S.F.; data curation, S.R.; formal analysis, S.R. and E.D.G.; investigation, S.R. and E.D.G.; methodology, S.R., E.D.G., I.M. and S.F.; software, S.R. and E.D.G.; supervision, E.D.G., I.M. and S.F.; validation, S.R. and E.D.G.; visualization, S.R.; writing—original draft, S.R.; writing—review and editing, S.R., E.D.G., I.M. and S.F. All authors read and agreed to the published version of the manuscript.

Funding: The National Center for Atmospheric Research is sponsored by the National Science Foundation. Funding support provided by the U.S. Army Corps of Engineers. The author Soren Rasmussen is funded by a fellowship provided by Sourcery Institute.

Institutional Review Board Statement: Not applicable.

Data Availability Statement: Data available in a publicly accessible repository. The data presented in this study are openly available in scrasmussen/icar_data at 10.5281/zenodo.4734314.

Acknowledgments: We would like to acknowledge the high-performance computing support from Cheyenne (doi:10.5065/D6RX99HX) provided by NCAR's Computational and Information Systems Laboratory, sponsored by the National Science Foundation. We would also like to acknowledge the high-performance computing support from Swan provided by Cray Inc.

Conflicts of Interest: The author Soren Rasmussen is funded by a fellowship provided by Sourcery Institute. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; nor in the decision to publish the results. Cray Inc. had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; nor in the decision to publish the results.

References

1. Gutmann, E.; Barstad, I.; Clark, M.; Arnold, J.; Rasmussen, R. The intermediate complexity atmospheric research model (ICAR). *J. Hydrometeorol.* **2016**, *17*, 957–973. [CrossRef]
2. Bernhardt, M.; Härer, S.; Feigl, M.; Schulz, K. Der Wert Alpiner Forschungseinzugsgebiete im Bereich der Fernerkundung, der Schneedeckenmodellierung und der lokalen Klimamodellierung. *ÖSterreichische-Wasser-Und Abfallwirtsch.* **2018**, *70*, 515–528. [CrossRef]
3. Horak, J.; Hofer, M.; Maussion, F.; Gutmann, E.; Gohm, A.; Rotach, M.W. Assessing the added value of the Intermediate Complexity Atmospheric Research (ICAR) model for precipitation in complex topography. *Hydrol. Earth Syst. Sci.* **2019**, *23*, 2715–2734. [CrossRef]
4. Horak, J.; Hofer, M.; Gutmann, E.; Gohm, A.; Rotach, M.W. A process-based evaluation of the Intermediate Complexity Atmospheric Research Model (ICAR) 1.0. 1. *Geosci. Model Dev.* **2021**, *14*, 1657–1680. [CrossRef]
5. Numrich, R.W.; Reid, J. Co-Array Fortran for parallel programming. In *ACM Sigplan Fortran Forum*; ACM: New York, NY, USA, 1998; Volume 17:2, pp. 1–31.
6. ISO/IEC. *Fortran Standard 2008*; Technical report, J3; ISO/IEC: Geneva, Switzerland, 2010.
7. Coarfa, C.; Dotsenko, Y.; Mellor-Crummey, J.; Cantonnet, F.; El-Ghazawi, T.; Mohanti, A.; Yao, Y.; Chavarría-Miranda, D. An evaluation of global address space languages: Co-array fortran and unified parallel C. In Proceedings of the tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Chicago, IL, USA, 15–17 June 2005; pp. 36–47.
8. Stitt, T. *An Introduction to the Partitioned Global Address Space (PGAS) Programming Model*; Connexions, Rice University: Houston, TX, USA, 2009.
9. Mozdzynski, G.; Hamrud, M.; Wedi, N. A partitioned global address space implementation of the European centre for medium range weather forecasts integrated forecasting system. *Int. J. High Perform. Comput. Appl.* **2015**, *29*, 261–273. [CrossRef]
10. Simmons, A.; Burridge, D.; Jarraud, M.; Girard, C.; Wergen, W. The ECMWF medium-range prediction models development of the numerical formulations and the impact of increased resolution. *Meteorol. Atmos. Phys.* **1989**, *40*, 28–60. [CrossRef]
11. Jiang, T.; Guo, P.; Wu, J. One-sided on-demand communication technology for the semi-Lagrange scheme in the YHGSM. *Concurr. Comput. Pract. Exp.* **2020**, *32*, e5586. [CrossRef]
12. Dritschel, D.G.; Böing, S.J.; Parker, D.J.; Blyth, A.M. The moist parcel-in-cell method for modelling moist convection. *Q. J. R. Meteorol. Soc.* **2018**, *144*, 1695–1718. [CrossRef]
13. Böing, S.J.; Dritschel, D.G.; Parker, D.J.; Blyth, A.M. Comparison of the Moist Parcel-in-Cell (MPIC) model with large-eddy simulation for an idealized cloud. *Q. J. R. Meteorol. Soc.* **2019**, *145*, 1865–1881. [CrossRef]
14. Brown, N.; Weiland, M.; Hill, A.; Shipway, B.; Maynard, C.; Allen, T.; Rezny, M. A highly scalable met office nerc cloud model. *arXiv* **2020**, arXiv:2009.12849.
15. Shterenlikht, A.; Cebamanos, L. Cellular automata beyond 100k cores: MPI vs. Fortran coarrays. In Proceedings of the 25th European MPI Users' Group Meeting, Barcelona, Spain, 23 September 2018; pp. 1–10.
16. Shterenlikht, A.; Cebamanos, L. MPI vs Fortran coarrays beyond 100k cores: 3D cellular automata. *Parallel Comput.* **2019**, *84*, 37–49. [CrossRef]
17. Rasmussen, S.; Gutmann, E.D.; Friesen, B.; Rouson, D.; Filippone, S.; Moulitsas, I. Development and Performance Comparison of MPI and Fortran Coarrays within an Atmospheric Research Model. Presented at the Workshop 2018 IEEE/ACM Parallel Applications Workshop, Alternatives To MPI (PAW-ATM), Dallas, TX, USA, 16 November 2018.
18. Stein, A.; Draxler, R.R.; Rolph, G.D.; Stunder, B.J.; Cohen, M.; Ngan, F. NOAA's HYSPLIT atmospheric transport and dispersion modeling system. *Bull. Am. Meteorol. Soc.* **2015**, *96*, 2059–2077. [CrossRef]
19. Ngan, F.; Stein, A.; Finn, D.; Eckman, R. Dispersion simulations using HYSPLIT for the Sagebrush Tracer Experiment. *Atmos. Environ.* **2018**, *186*, 18–31. [CrossRef]
20. Esmaeilzadeh, H.; Blem, E.; Amant, R.S.; Sankaralingam, K.; Burger, D. Dark silicon and the end of multicore scaling. In Proceedings of the 2011 38th Annual International Symposium on Computer Architecture (ISCA), San Jose, CA, USA, 4–8 June 2011; pp. 365–376.
21. Moisseeva, N.; Stull, R. A noniterative approach to modelling moist thermodynamics. *Atmos. Chem. Phys.* **2017**, *17*. [CrossRef]
22. Stull, R.B. *Practical Meteorology: An Algebra-Based Survey of Atmospheric Science*; University of British Columbia: Vancouver, BC, Canada, 2018.
23. Yau, M.K.; Rogers, R.R. *A Short Course in Cloud Physics*; Elsevier: Amsterdam, The Netherlands, 1996.
24. Rasmussen, S.; Gutmann, E. Coarray ICAR Fork. [Code]. Available online: github.com/scrasmussen/coarray_icar/releases/tag/v0.1 (accessed on 14 January 2021).
25. Rasmussen, S.; Gutmann, E. ICAR Data. [Dataset]. Available online: github.com/scrasmussen/icar_data/releases/tag/v0.0.1 (accessed on 14 January 2021).
26. Mandl, F. *Statistical Physics*; Wiley: Hoboken, NJ, USA, 1971.
27. Marmelad. 3D Interpolation. Available online: https://en.wikipedia.org/wiki/Trilinear_interpolation#/media/File:3D_interpolation2.svg (access on 14 January 2021).

28. University of Wyoming. Upper Air Soundings. Available online: weather.uwyo.edu/upperair/sounding.html (accessed on 10 November 2020).
29. Sharma, A.; Moulitsas, I. MPI to Coarray Fortran: Experiences with a CFD Solver for Unstructured Meshes. *Sci. Program.* **2017**, *2017*, 3409647. [CrossRef]
30. Fanfarillo, A.; Burnus, T.; Cardellini, V.; Filippone, S.; Nagle, D.; Rouson, D. OpenCoarrays: Open-source transport layers supporting coarray Fortran compilers. In Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models, Eugene, OR, USA, 6–10 October 2014; pp. 1–11.
31. Feind, K. Shared memory access (SHMEM) routines. *Cray Res.* **1995**, *53*, 303–308.
32. HPE Cray. *Cray Fortran Reference Manual*; Technical Report; Cray Inc.: Seattle, DC, USA, 2018.
33. Shan, H.; Wright, N.J.; Shalf, J.; Yelick, K.; Wagner, M.; Wichmann, N. A preliminary evaluation of the hardware acceleration of the Cray Gemini interconnect for PGAS languages and comparison with MPI. *ACM Sigmetrics Perform. Eval. Rev.* **2012**, *40*, 92–98. [CrossRef]
34. Shan, H.; Austin, B.; Wright, N.J.; Strohmaier, E.; Shalf, J.; Yelick, K. Accelerating applications at scale using one-sided communication. In Proceedings of the Conference on Partitioned Global Address Space Programming Models (PGAS'12), Santa Barbara, CA, USA, 10–12 October 2012.
35. Shende, S.S.; Malony, A.D. The TAU parallel performance system. *Int. J. High Perform. Comput. Appl.* **2006**, *20*, 287–311. [CrossRef]
36. Ramey, C.; Fox, B. Bash 5.0 Reference Manual. 2019. Available online: gnu.org/software/bash/manual/ (accessed on 12 May 2020).
37. Kaufmann, S.; Homer, B. *Craypat-Cray X1 Performance Analysis Tool*; Cray User Group: Seattle, DC, USA, 2003; pp. 1–32.
38. Zivanovic, D.; Pavlovic, M.; Radulovic, M.; Shin, H.; Son, J.; Mckee, S.A.; Carpenter, P.M.; Radojković, P.; Ayguadé, E. Main memory in HPC: Do we need more or could we live with less? *ACM Trans. Archit. Code Optim. (TACO)* **2017**, *14*, 1–26. [CrossRef]