Parallel FESOM2 Climate Data Analysis

Guglielmi Matteo, 232088, Alessandro Zinni, 229709
Department of Information Engineering and Computer Science
University of Trento, Italy
matteo.guglielmi@studenti.unitn.it, alessandro.zinni@studenti.unitn.it

***Abstract*— Nowadays, lots of applications require collecting a huge amount of data to fulfil specific tasks. One of these applications is `Climate Data Analysis` which many research programs focused on in recent years. With this work, our objective is to `devise a parallel pipeline` to speed up the processingof FESOM2 data acquired on a global scale. More precisely, the final goal is be `to reduce`: (1) the `number of data` at out disposal performing some form of `data reduction` (in this specific case, the latter will be base on a maximum operation) (2) the `time consumption` to shrink the data.**

***Index Terms*—FESOM2, netCDF4, mesh, vnod, unod, MPI**

I. Introduction

THE technological progress over the last decades allowed scientists all over the world to carry out research programs implying a huge amount of data. During this period, new means started to emerge, leading to the very recent super-computing infrastructures. The need of such computational structures arises to comply with the complexity of many applications. Indeed, nowadays even the simplest sensors need to acquire a consistent amount of data in order to obtain refined and reliable results.

Obviously, resources imply a cost and putting together heterogeneous hardware is not always trivial and possible. Luckily, there are some software based techniques that allow to parallelize the execution of such processing algorithms (e.g. MPI) bringing further improvements to the expected performances.

II. Adopted procedure

The procedure for developing this work was devised as follow:

1) **serial** implementation of **maximum depth reduction**;
2) **parallel** implementation of **maximum depth reduction** using `MPI API` [1] commands trying to minimize execution time;
3) perform **benchmarking** plotting trend of different performance measures.

III. The data : FESOM2

Climate models are becoming increasingly important to a wider range of users. They provide **projections of anthropogenic climate change** and they are **used to understand the functioning of the climate system** [1]. Despite progresses have been made in modeling climate changes, still they appear to be **sub-optimal** with substantial shortcomings. In light of that, it was recognized that these shortcomings were **due to** the **lack of spatial resolution** to capture processes that are too small in scale and, therefore, structured state-of-the-art climate models were not able to characterize it. As a consequence, a new generation of **unstructured-mesh methods** has emerged.

A. Why an unstructured mesh?

UGrid based methods allow to have a varying spatial resolution across the region under investigation (e.g. the whole globe, a continent, etc.) and this translated into new opportunities to develop more precise climate models.

B. FESOM

With the intent of extending the usage of UGrid model to global scale problems, FESOM reveals itself to be the perfect candidate. The latter has been developed over the last twenty years andit turned out to be the ideal candidate for climate research applications [1].

Brief history of FESOM: The first FESOM version (version 1.1) was documented by Danilov et al. (2004) [2]. Overall, the approach revealed to be too slow for climate scale simulations and this pushed the model developers to devise a new solution leading to version 1.2 [3]. This new model was about 10 times faster then the one described in [2].
Further improvements were achieved with the newer version Fesom 1.3 [4] that consisted in concatenating the FESOM model upon the ocean. Subsequently, [3] and [4] features were combined leading to version 1.4 [1].
More recently, the 2nd version of the unstructured-mesh Finite-Element Sea ice-Ocean circulation model (FESOM) was released. The latter is build on top of the previous model FESOM 1.4 [1] but differs by its discretization structure which has been defined over volumes instead of points 1), and was formulated such that it further improves model flexibility. The new version improves the numerical efficiency of FESOM in terms of CPU time by at least 3 times while retaining its fidelity in simulating sea ice and the ocean. In this section, we did not enter in details concerning the features of each version. Indeed, this digression were meant to transmit a simple concept: the **field is advancing** so **rapidly** that details of the FESOM implementation might change across very short times, as the research is very active.

C. Used data

In this work, we had at our disposal three different **`netCDF4 files`**:

- **`vnod.fesom.2010.nc`**: this contains the meridional (**`vertical`**) **`velocity`** component for every point of the mesh across 12 timesteps and 69 depths;

[1] https://www.open-mpi.org/doc/

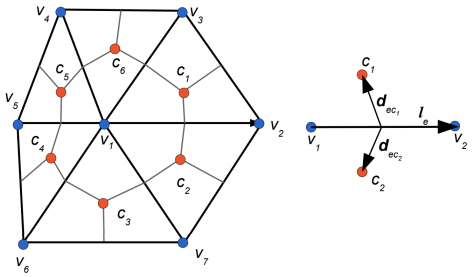

Fig. 1. Schematic of cell–vertex discretization (left) and the edge-based structure (right). The horizontal velocities are located at cell (triangle) centers (red circles) and scalar quantities are at vertices (blue circles). the vertical velocities are at scalar locations too.

- **`unod.fesom.2010.nc`**: this contains the **`horizontal velocity`** component for every point of the mesh across 12 timesteps and 69 depths;
- **`fesom.mesh.diag.nc`**: contains the **`spatial reference`** of each node (i.e. each point of the mesh has an associated latitude and longitude coordinate).

More precisely, vnod and unod files have 3 dimensions: `nz1` (i.e. depth), `time`, and `nod2` (i.e. mesh nodes). The velocity variable is expressed in dependency of the three aforementioned dimensions.
The mesh file presents only one dimension (`NODE2`) and two variables, `lat` (i.e. latitude) and `long` (i.e. longitude) which combined encode the spatial location of each node.

It is worth mentioning that during the development we **assume** that **each node**, across the different files, is **ordered in the same way**, i.e. the dimension nod2, consisting in $\sim 8M$ points, is equally sorted across the different FESOM data.

IV. NETCDF4 MODEL

The **`netCDF Data Model`** can be intended as a way of thinking and **`arranging data`** into dimensions, variables and attributes. These three components together compose the so called *The Classic Model* which was later expanded into *The Enhanced Data Model*.

A. The Classic Model

The classic netCDF data model (Figure 2) consists of variables, dimensions, and attributes. More precisely, each building blocks represent :

TABLE I
CLASSIC MODEL COMPONENTS.

Variables	N-dimensional arrays of data.
Dimensions	Axes of the data arrays.
Attributes	Supplementary metadata.

B. The Enhanced Model

The Enhanced Model is the extension of the netCDF data classic model in a backwards compatible way. It contains the variables, dimensions, and attributes of the classic data model, but adds:

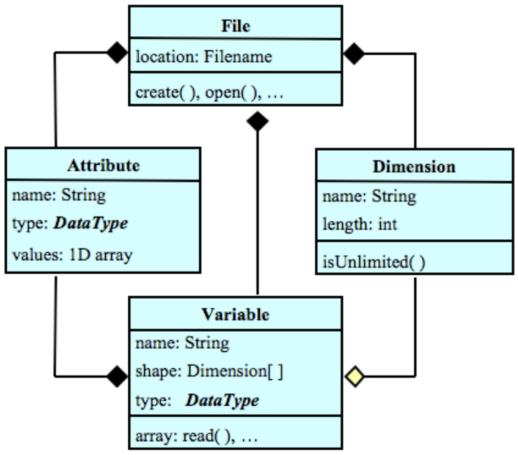

Fig. 2. UML of "The Classic Model".

- groups - A way of hierarchically organizing data, similar to directories in a Unix file system.
- user-defined types - The user can now define compound types (like C structures), enumeration types, variable length arrays, and opaque types.

V. THE IMPLEMENTATION

This work consists in **performing a comparison between a serial implementation**, which is the "canonical way" in developing code, **and a parallel implementation**, subdividing the arithmetic operations into processes, to investigate the differences in performances using different approaches.
At the end, benchmarks, have been performed to **address** the **speed-up and efficiency** brought by the parallel implementation in function of the number of processes used.

A. Serial workflow

In this section, the serial implementation is briefly discussed, investigating the salient steps. We're going to refer to **Algorithm 1** to alleviate the discussion. The netCDF library works assigning different ids to files and variables to be retrieved. Once the variables are extracted, the data at our disposal is a 3D matrix organized as `matrix[TIME][DEPTH][NODE2]` where `TIME=12`, `DEPTH=69` and `NODE2=8852366`. Through **Algorithm 1** the reduced matrix is obtained.

VI. PARALLEL IMPLEMENTATION

As briefly mentioned in Section I and Section II, to implement the parallel version we exploited the capabilities of `MPI`. The latter allows us to split an arbitrary data among a variable number of processes which are quantified using the `PBS` [2] interface.
MPI provides lots of functions and methods implemented for C programming language. Among these, we made use of:

- **`MPI_Barrier`**[3] to align all the processes in specfic points;

[2]https://2021.help.altair.com/2021.1/PBSProfessional/PBS2021.1.pdf
[3]https://www.mpich.org/static/docs/v3.2/www3/MPI_Barrier.html

Algorithm 1: Function to compute reduction.

```
Input: var_idx, matrix, dimension
Output: max
/* var_idx: variable index along
   which to reduce,                    */
/* matrix: data to reduce in form of
   3D matrix,                          */
/* dimension: support variable to
   print results                       */

function COMPUTE_MAXIMUM(var_idx, matrix,
 dimension):
    max ← NULL
    if var_idx is 0 then
        dimension ← TIME
        max ← ALLOC_VECTOR_OF_POINTERS
        for i from 0 to TIME do
            max[i] ← ALLOC_SPACE4FLOAT
            max[i][0] ← matrix[i][0][0]
            for j from 0 to DEPTH do
                for k from 0 to NODE2 do
                    if max[i][k] < matrix[i][j][k] then
                        max[i][k] ← matrix[i][j][k]
                    end
                end
            end
        end
    else
        /* same for var_idx = 0 but
           switching the two outer fors
           */
    end
```

- **`MPI_Scatterv`**[4] allows to split buffers among different processes;
- **`MPI_Gatherv`**[5] allows reconciling into specified location data coming from different processes.

The main reason behind using the "v" variant of `MPI_Scatter`[6] and `MPI_Gather`[7] is that these variants allow dealing **only** with **contiguous data and chunks uniform in size** on the sender side. On the other side, the "v" variants allow to define a displacement in order to jump uniformly over memory locations. This provides extra capabilities such as irregular message sizes are allowed, data can be distributed/gathered in any order among processes. Moreover, **`MPI_Scatterv` accepts memory gaps** between messages in source data and **`MPI_Gatherv` allows a variable count of data from each process**. This is a **`key aspect`** since we want to perform **data reduction along** the **columns** of a 3D matrix. In fact, in **C programming language** multidimensional arrays are stored in **row-major ordered** and zero-based indexing meaning that the way multidimensional arrays are flattened and made contiguous in memory follows

[4]https://www.mpich.org/static/docs/v3.1/www3/MPI_Scatterv.html
[5]https://www.mpich.org/static/docs/v3.1/www3/MPI_Gatherv.html
[6]https://www.mpich.org/static/docs/v3.1/www3/MPI_Scatter.html
[7]https://www.mpich.org/static/docs/v3.3/www3/MPI_Gather.html

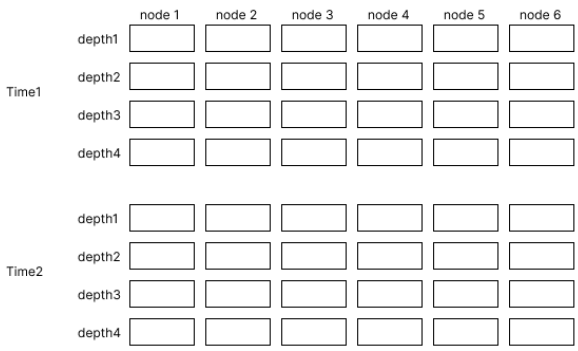

Fig. 3. Sketch on how data is arranged in memory

the rows (i.e. consecutive elements of a row reside next to each other). In light of this, you can now understand the importance conveyed by using Scatterv and Gatherv with respect to the classic variants.

How does all of this apply to our implementation? In order to scatter and gather data correctly, it is important to know how data is arranged in memory Another important aspect is the **definition of custom** types (using `MPI_Type_vector`), in particular:

- **one** custom vector type has been defined **on** the "**scatter side**" to scatter the different columns. Indeed, the strided vector has been defined with `TIME*DEPTH` blocks each of them containing just one float with a stride of `NODE2` elements between the start of two consecutive blocks. When scattering, each process is going to receive $\frac{\text{n°nodes}}{\text{n°processes}}$ of this vector type, with the last process taking all the remaining data if the number of nodes (i.e. `NODE2`) is not divisible by the number of used processes;
- **one** custom vector type has been defined at the moment of **gathering the drafted data** from the different processes. Indeed, since the depths are reduced just to one, the strided datatype has been defined with a number of blocks equal to `TIME`, containing just one float and `NODE2` as a stride between two blocks.

VII. Results

A. Benchmarks

To address the performances, we exploited the `MPI_Wtime`[8] provided by `MPI` itself which acts as a chronometer. In particular, the elapsed **time** has been **measured between before** the whole **splitting** procedure **and after the** drafted data has been **gather**ed from the several processes.
It is worth mentioning that in the **initial trials** we experienced a **huge overhead** caused by the communication between processes **and** a lot of **noise** probably due to other processes running on the cluster. To further address these issues, we tried an alternative simpler but heavier computation to verify whether our hypothesis was truthful or not. It turned out that

[8]https://www.mpich.org/static/docs/v3.1/www3/MPI_Wtime.html

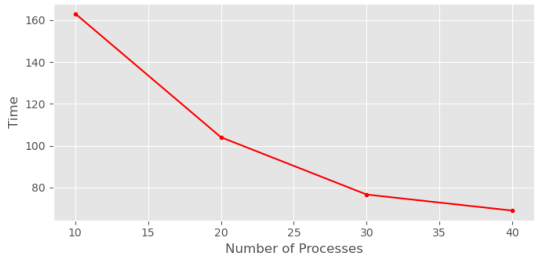

Fig. 4. Benchmark performed to investigate time gain as function of the number of processes used.

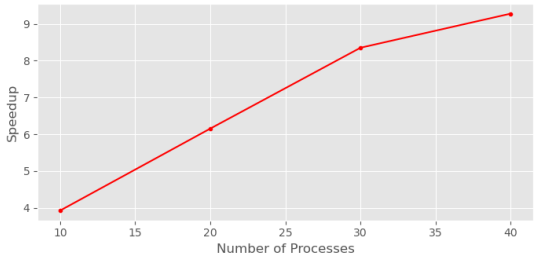

Fig. 5. Benchmark performed to investigate speedup as function of the number of processes used.

in the first measurements, what we were measuring was only noise since the parallel computation of the maximum lasted very few seconds and the communication/allocation times were orders of magnitude bigger. In light of this observation, we devised a **wasting time function**, that wastes an amount of time proportional to the data that each process receives in order **to scale the computation times** to the additional time due to this overhead. The deployed trick allowed us to properly address parallel maximum computation time without them being drowned in noise.

B. Metrics

Time:

The **first performance metric** we used to address whether our parallel implementation bring some improvements is simply observing the **computation times** across different runs and with different number of PBS processes. In particular, after having collected 5 $^{\text{runs}}/_{\text{proc}}$ and averaged the relative times, we've built a graphical trend to better and easily appreciate the gain in time in function of the number of processes used. As showcased in (**Figure 4**), increasing the number of processes brings an improvement in execution times.

It is worth mentioning that, as expected, the **betterment is not linear** and it **will reach a saturation point** where the dispalyed trend inverts its direction. Unfortunately, we couldn't properly test and find this saturation point due to the cluster payload and limitations in resources allocation.

Speedup:

The **Speedup** is a performance measure that quantifies the gain in time using a parallel application with respect to a serial one. Mathematically, this quantity is defined as:

$$\text{Speedup} = \frac{\text{T}_{\text{serial}}}{\text{T}_{\text{parallel}}}$$

The corresponding plot is displayed in **Figure 5** and, as expected, the shown function follows an **increasing trend** since the denominator, i.e. the seconds taken to carry out the parallel reduction, express a monothonically decreasing tendency.

Efficiency:

Unlike the previous quantity, the **Efficiency** takes also into account the number of processes used to reach a certain speedup. Indeed, this measure is defined as:

$$\text{Efficiency} = \frac{\text{S}}{\text{p}} = \frac{\left(\frac{\text{T}_{\text{serial}}}{\text{T}_{\text{parallel}}}\right)}{\text{p}} = \frac{\text{T}_{\text{serial}}}{\text{p} \cdot \text{T}_{\text{parallel}}}$$

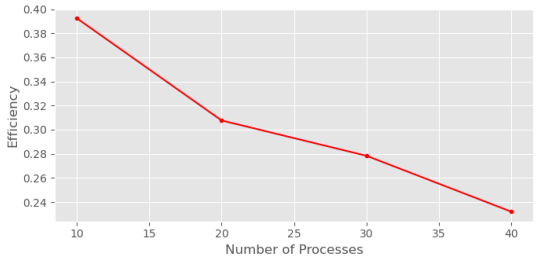

Fig. 6. Benchmark performed to investigate efficiency as function of the number of processes used.

The corresponding plot is displayed in **Figure 6** and, as expected, the shown function **follows a decreasing trend**. This follows our theoretical expectations since the speedup is not linear. Moreover, the efficiency measure possible values range between $0 \to 1$ where:

- 0 is achieve when whether the number of processes become very large or the execution time becomes several orders of magnitude bigger than the serial one;
- 1 is achieved in case of linear speedup. In fact, point-wise, the improvements obtained by progressively introducing additional processes should be proportional to the number of processes itself. As a consequence, the following relation should hold:

$$\begin{aligned}\text{Efficiency} &= \frac{\text{T}_{\text{serial}}}{\text{p} \cdot \text{T}_{\text{parallel}}} = \frac{\text{T}_{\text{serial}}}{\text{p} \cdot \dfrac{\text{T}_{\text{serial}}}{\text{p}}} \\ &= \frac{\text{T}_{\text{serial}}}{\cancel{\text{p}} \cdot \dfrac{\text{T}_{\text{serial}}}{\cancel{\text{p}}}} = \frac{\text{T}_{\text{serial}}}{\text{T}_{\text{serial}}} = 1\end{aligned}$$

Since the last condition is not respected, adding processes reduces the efficiency while still reducing the execution times.

VIII. Conclusion

To summarise, in this work we've presented a possible parallel implementation (refer to **Section VI**) to perform data reduction of FESOM2 data along the depth dimension.

As pointed out during the report, we made use of the MPI APIs to divide mesh patches across different processes and to take time measurements. The latter has been used to build benchmarks trends (**Section VII-A**) showcasing different performance measurements, i.e. efficiency and speedup.

Furthermore, problems and limitations has been discussed in order to better contextualize and address the reached performances. As final remarks, I would like to point out that the

results obtained in this work are not intended as optimal, but they are a good starting point for further improvements.

REFERENCES

[1] Q. Wang, S. Danilov, D. Sidorenko, R. Timmermann, C. Wekerle, X. Wang, J. T., and J. Schröter, "The finite element sea ice-ocean model (fesom) v.1.4: formulation of an ocean general circulation model," *Geosci. Model Dev. Discuss.*, April 2014.

[2] S. Danilov, Q. Wang, M. Losch, D. Sidorenko, and J. Schröter, "Modeling ocean circulation on unstructured meshes: comparison of two horizontal discretizations," *Ocean Dynam.*, vol. 58, pp. 365–374, 2008.

[3] Q. Wang, S. Danilov, M. Losch, and J. Schröter, "Finite element ocean circulation model based on triangular prismatic elements, with application in studying the effect of vertical discretization," *J. Geophys. Res.*, vol. C05015, no. 113, pp. 365–374, 2008.

[4] R. Timmermann, S. Danilov, S. J., C. Böning, D. Sidorenko, and K. Rollenhagen, "Ocean circulation and sea ice distribution in a finite element global sea ice-ocean model," *Ocean Model.*, no. 27, pp. 115–129, 2009.