Geosci. Model Dev. Discuss.,
https://doi.org/10.5194/gmd-2020-257-AC2, 2021

Geoscientific
Model Development
Discussions
Open Access

Interactive comment on "Parallelizing a serial code: open–source module, EZ Parallel 1.0, and geophysics examples" *by* Jason Louis Turner and Samuel N. Stechmann

Jason Louis Turner and Samuel N. Stechmann

jlturner5@wisc.edu

Received and published: 8 January 2021

Dear Anonymous Referee,

Thank you for reading our paper and providing these helpful comments. Below are detailed responses to your comments. *Your comments are shown in black italics*, the authors' response is in blue, and the authors' changes in the manuscript are in green.

This paper addresses the issue of parallelizing a serial code used in geophysical applications. An open-source module named EZ Parallel is produced for this purpose to automatically with minimum invasion and knowledge of the user parallelize a typical

Printer-friendly version

Discussion paper

coding setup of geophysical code. The authors demonstrate a very good speed-up of two example codes relative to the amount of user work invested in parallelization. The authors used MPI to achieve this goal.

Mayor point:

In algorithmic and modelling terms, the idea for this paper is not a new thing, but this paper with the supplemented open-source module does bring a merit for giving us a somewhat simpler way of speeding up a user code on distributed systems, at least for very simple grids. I do like the fact how more advanced usage of MPI becomes hidden in the user code. However, I am concerned about the fact that the overhead of implementing MPI on the user side for simple grids is not that big compared to using EZ Parallel in the code. I would like to be explained more why using EZ Parallel is much easier than MPI for simple grids which you demonstrated. I think this paper would benefit more in explaining the tedious overhead of using MPI over EZ Parallel, in terms of additional lines of code and specific knowledge of MPI needed. Also assume we have a user who just coded two-level quasi-geostropic equation and has a pretty good idea how to use MPI to parallelise it. Why EZ Parallel is much better for him?

A new paragraph has been added to explain how using EZP can be easier than directly writing out the MPI commands. Other new description, in addition to this paragraph, is also aimed at explaining why EZP could be useful for a user who already has experience with MPI.

One benefit to such a user would be the included parallel fast Fourier transforms that are included in EZ Parallel, as they utilize an algorithm introduced by Mortensen et. al. in their 2019 paper “Fast Parallel Multidimensional FFT Using Advanced MPI” (https://doi.org/10.1016/j.jpdc.2019.02.006). The EZ Parallel fast Fourier transform can handle 1-D and 2-D transforms of both double precision and double precision complex data types, using approximately 350 lines of code (including initializing the transforms and error handling). The EZ Parallel fast Fourier transform requires only two subroutine calls by the user, a call to the transform initialization and a call to the execution of the transformation. Additionally, the subroutines of EZ Parallel have been thoroughly tested and debugged, and will notify the user of many different types of errors. MPI itself has a relatively poor repertoire of easily-accessible and easily-usable debugging tools, so they could use this library with confidence that it would not readily introduce errors into their code.

(LINES 44 - 57)

The goal of EZP is to break the parallelization process into a few high-level tasks. For the scenarios of interest described above, where the setup involves PDEs on a 2D or 3D domain, the high-level tasks include, for instance, a domain decomposition to break the domain into several partitions (see Section 2). For each of the high-level tasks, the goal of EZP is to provide a corresponding subroutine. Then the subroutine can be called to accomplish the task, and the user does not need to worry about the parallel programming details. For instance, a simple command `CALL DOMAIN_DECOMPOSITION` could be inserted into a code, and the domain decomposition will be accomplished without the user needing to write their own parallel programming algorithms. The EZ Parallel subroutine that accomplishes the domain decomposition contains approximately 150 lines of code, including defining MPI derived datatypes to be used in some inter-core communications and error handling. As an analogous situation that is already in widespread use, consider the calculation of Fourier transforms. Most users do not write their own Fourier transform algorithms, in parallel or in serial. Instead, a user would typically take advantage of the many established Fourier transform codes that have been written, which allow the user to type a simple command such as `y = FFT(x)` and obtain the Fourier transform. No knowledge is needed of the underlying details of the Fourier transform code or algorithm. The goal of EZP is to provide a similar type of high-level functionality for some parallel programming tasks, in order to ease the process of parallelizing a serial code.

(LINES 98 - 100)

Also, we hope that even experienced users of MPI will find the EZP module useful, since EZP could speed up and simplify the code-writing process (by providing high-level functionality via the grouping together of MPI commands).

General questions:

1. Have you investigated using asynchronous MPI? Is there any problem adding support for OpenMP through EZ Parallel?

We have indeed considered using asynchronous MPI, and it would undoubtedly allow the user to further optimize the performance of their code. However, we would like to strike a balance between providing the user a variety of tools and simplifying the process of parallelizing their serial code. Since we would like EZ Parallel to be easy for users who have relatively little experience with parallel computing, for simplicity's sake we have utilized only blocking sends/receives. In future versions of the EZ Parallel, we hope to find a simplistic way of implementing subroutines with asynchronous MPI.

We also did not utilize OpenMP for a similar reason, and hope to provide functionality with OpenMP in future versions.

(LINES 313 - 314)

It would also be interesting to explore asynchronous MPI, or to pursue similar software involving CUDA or OpenMP, or other languages such as Julia or Python.

2. After parallelising the code and its output, one is faced with multiple output_procID which need to merged and visualized. Can EZ Parallel provide a tool to that, or is merging left to the user?

The EZ Parallel examples come with example Matlab scripts that we used for merging output files and visualizations.

(LINES 108 - 109)

In the examples of parallelized codes we have provided example Matlab scripts that we used for merging output files (see Subsection 2.4) and visualizations.

Minor remarks:

line 78: "with modification in red" but I do not see red text in Algorithm 2

Thank you – this has been corrected.

The text “with modifications in red” has been changed to “with modifications in boldface”.

line 195: andf $->$ and

Thank you – this typo has been corrected.

line 211: Did you test scaling with I/O?

I/O was disabled during the scaling tests, following somewhat standard practice.

(LINES 249 - 250)

File output was disabled for all tests following common practice (e.g., Arabas et. al., 2015), and we analyzed the average execution time per time-step for consistency between runs.

line 257: psuedo $->$ pseudo

Thank you – this typo has been corrected.

Algorithm 4 after INITIALIZE_GRID: paralllelization $->$ parallelization

Thank you – this typo has been corrected.

Fig 6 & Fig 7 take a large portion but are not directly relevant, they could be grouped together in Fig 6a 6b.

It is true that these two figures take up a large amount of space. If we were to shrink the size of the figures, though, then the serial-simulation domain (shown in the inset) would become so small that it would be difficult to discern. We would like to maintain our original intent for these figures, which was to highlight the much bigger domain size that can be utilized in parallel simulations versus serial simulations, and we do feel that a figure highlights this nicely. To keep this intent and also save space, we have taken the following approach.

Given that the intent can be accomplished with only Figure 6 by itself, we have removed Figure 7.

We hope that these responses are satisfactory and would like to again thank you for your efforts and constructive feedback. The authors welcome any further constructive comments.

Sincerely,

Jason Torchinsky [Formerly “Turner"] (jlturner5@wisc.edu)

Samuel Stechmann

Interactive comment on Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2020-257, 2020.