Geosci. Model Dev. Discuss.,
https://doi.org/10.5194/gmd-2020-91-RC1, 2020

Geoscientific Model Development Discussions
Open Access
EGU

Interactive comment on "A new distributed algorithm for routing network generation in model coupling and its evaluation based on C-Coupler2" *by* Hao Yu et al.

Moritz Hanke (Referee)

hanke@dkrz.de

Received and published: 24 April 2020

Summary

This manuscript addresses the issue of the generation of a routing table, which is used to handle the data redistribution between two sets of processes. In climate modelling this occurs for example in the transfer of data between two component models.

The authors of the manuscript assume that the majority of existing coupling solutions use an inefficient algorithm to generate these routing tables and they attribute the inefficiency mainly to the use of collective communication.

Printer-friendly version

Discussion paper

The majority of the manuscript concentrates on presenting an algorithm for the generation of routing tables and evaluation of its performance. Even though the presented algorithm itself is well suited for this task, it is not completely new and is already being used by different coupling solutions and communication libraries.

Overall, the manuscript is well structured and sufficiently well written. However, due to the lack of substantial contribution to modelling science, I would not recommend this manuscript for publication in the Geoscientific Model Development.

General

The presented algorithm is basically a rendezvous algorithm, which uses a distributed directory. This was first introduced in [1]. I assume that you did not know about this paper and therefore did not reference it. This algorithm works by distributing data among the processes in a globally known decomposition. This is called “regular intermediate distribution” in the manuscript. Using this intermediate decomposition, accessing data without knowing the original decomposition and without gathering all data on a single process is easily possible. The use of distributed directories in a similar context is mentioned in the following references [2], [3], and [4].

To generate the distributed directory, the authors propose to use a parallel sort, which uses the global indices of the cells as sorting keys. Such algorithms are commonly known and are no new inventions.

The remainder of the presented algorithm does not contain any significant scientific contributions. Therefore, I do not think that this manuscript contains a substantial contribution to modelling science.

Specific comments

The manuscript describes the router network generation based on two predefined decompositions from two component models as being a fundamental functionality of a coupler. However, this routing table can also be a by-product, in case both compo-

nents have different grids and the coupler generates interpolation weights online.

Line 30: "there is almost no evidence of scalable initialization of a coupler" [...] "(Craig et al., 2017; Liu et al., 2018)"
Both papers mentioned in this context contain figures with initialisation cost measurements (see figure 2 and figure 8 respectively). Scaling behaviour in these figures is indeed sub-optimal. However, the cause for this scaling behaviour is not explicitly attributed by either paper to the router generation. Your manuscript indicates that in fact this is the case without providing evidence. By recreating figure 8 from the second paper with your new router generation implementation, you could have confirmed (at least for the C-Coupler) this.
In Hanke et al., 2016 (cited by the manuscript) figure 3 (b) shows good scalability of the overall coupler initialisation for up to 3072 processes per component.

Line 56: "almost all existing couplers use the following 4 steps for generating a routing network"
This is a very strong claim, which I would not support (see [2], [3], and [4]).

Paragraph 3.2 and figure 2:
This is the description of a basic parallel sorting algorithm. A shorter paragraph and a reference to a respective paper would have been enough.

Line 167-168: "it only utilizes point-to-point communications and does not rely on any collective communication"
The parallel sorting algorithm has a complexity of O(log(n)) and a similar communication pattern as the MPI implementations of various collective communication operations[5]. Therefore, you could argue that you actually did use collective communication, which you implemented using point-to-point communication.

Paragraph 4:
This paragraph does not describe how these measurements where generated. Did you do a single run, average of multiple runs or average over multiple executions of the

algorithm within a single run? It is possible, that especially the first execution of this algorithm produces for some MPI implementations a much higher run time than the following ones.

Figure 3:
I would have preferred to have absolute runtimes instead of speedups for the evaluation of the individual algorithms and speedups only for the direct comparison between the two.

Tables 1 to 6:
In my opinion, these tables add no significant value to the understanding of the algorithm.

Table 7:
I assume the 1600 cores mean, that you used two toy components with 1600 cores each. I could not find an explicit description of this.
The core counts are rather odd. You mentioned that your nodes have 24 processors each. Therefore, I would have assume, that the core counts in your tests are multiples of 24.

Remarks

Intel MPI library (3.2.2)
Why did you use such an old version?
In my experiments Intel MPI often performed very poorly. Maybe give the most recent OpenMPI a try.

You propose to use a sorting algorithm with complexity O(log(n)) to generate the distributed directory. In some tests, I have seen that some MPI implementations introduce significant delays with such communication pattern, when being used for the first time in a run.
Alternatively, each process could compute for all its local points the destination rank

in the distributed directory. Using alltoall, you can exchange the number of points that need to be sent, in order to get directly from the original to the intermediate decomposition. Afterwards, alltoallv can redistribute the data in a single communication call. Depending on the MPI implementation alltoall can also have a complexity of O(log(n)). However, very little data is exchanged and it can be highly optimised within the MPI. The communication matrix for the alltoallv is probably very sparse. Hence, it can be implemented by the user using point-to-point communication. In my tests, this approach delivers really good results.

The code provided for this manuscript already uses some parts of ESMF for time management. Maybe have a look at ESMCI_MeshRedist.C and Zoltan/dr_dd.c in the original ESMF repository, these should contain algorithms very similar to the one proposed in this manuscript.

1: A. Pinar and B. Hendrickson, Communication Support for Adaptive Computation in Proc. SIAM Conf. on Parallel Processing for Scientific Computing, 2001.
2: https://www.earthsystemcog.org/site_media/projects/esmf/pres_0812_board_gerhard.pdf
3: http://www.cs.sandia.gov/Zoltan/ug_html/ug_util_dd.html
4: Hanke, M., Redler, R., Holfeld, T., and Yastremsky, M.: YAC 1.2.0: new aspects for coupling software in Earth system modelling, Geosci. Model Dev., 9, 2755–2769, https://doi.org/10.5194/gmd-9-2755-2016, 2016
5: Thakur, R., Rabenseifner, R., Gropp, W. (2005). Optimization of Collective Communication Operations in MPICH. The International Journal of High Performance Computing Applications, 19(1), 49–66. https://doi.org/10.1177/1094342005051521

Interactive comment on Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2020-91, 2020.

Printer-friendly version

Discussion paper