Editor # 1: Dr. Lutz Gross

I am happy to inform you that your paper has been accepted subject to minor revisions. Please provide a revised manuscript that

1. address the commends of reviewer #2 - in particular about a consistent reference to the "ShellSet v1.1.0" program code.

2. includes the feedback from reviewer#1: The discussion of the performance of dgbsv is oversimplifying the situation as the performance is sensitive to many factors including memory bandwidth (as mentioned by reviewer #2), memory hierarchy, cashe size, thread placement (hyperthreading), matrix size, matrix bandwidth and competing applications (as mentioned in the manuscript). One needs to be careful to draw conclusions from the a single test case.

We have made the changes as requested by the reviewers.

I would like to ask you to modify the manuscripts in the following way:

2.1 better clarify how the timings were obtained (single user? single model? averaged?)

Each of the performance tests in section 5 used the same fixed model and are averaged over 3 iterations of each test. The MKL test in section 5.1 used only a single MPI process with Shells iterated twice, the MPI test in section 5.2.2 also iterated Shells twice while the test in section 5.2.1 iterated Shells 5 times (in line with Bird et al. (2008)). Where necessary we have clarified this in the text.

2.2 remove "this could be because the problem size is not large enough or, since we are working on a banded array, the problem is simply not complex enough." - this can not be justified by the test (meaning of 'complex'?). As using more than 16 threads is crossing CPU boundaries a possible reason could be slower cross-socket memory access and slower cross-socket thread barriers -# of thread barriers tend to increase with a smaller problem size per thread.

We have removed the text and added a note about the slowdown possibly being due to using cores which span multiple CPUs.

2.3 remove the paragraph after line 368: 16 independent models on 16 cores may uncoordinately compete over memory bandwidth and caches so there is no guarantee that this runs faster than a single model on 16 cores.

We have removed this paragraph.

Referee #1: Dr. Lavinia Tunini

Minor comments (# of lines referred to cleaned manuscript):

• Misleading reference. The authors, in the abstract, mention the link https://zenodo.org/records/7986808, but, in another part of the manuscript – the end of the Introduction – they cite May et al. (2023), which connects to the same Zenodo link mentioned in the abstract. In order to avoid confusions, I suggest to cite in both places, as well as in the Code and Data Availability section, the ShellSet software source as: https://zenodo.org/records/7986808, May et al. (2023).

We have changed all references to match the suggestion.

• The authors should specify in the manuscript (may be in the Code and Data Availability section) that May et al. (2023) is the main source for the ShellSet program v.1.1.0, and which are the main differences with respect to the version at https://github.com/JonBMay even if these differences are detailed at the github link.

We have clarified the differences between the two repositories within the Code and Data Availability section.

• Lines 43-44. Please, rephrase the sentence as "In this work we present ShellSet, which is composed by (i) a single program ...; (ii) and OrbScore2, a scoring program. . . ." , as well as eliminate "which we call ShellSet" at line 50.

We have rephrased the sentence and eliminated the text at line 50.

• Line 53. Missing dot.

Added.

• Line 83. Extra ")"

Removed.

• Line 241: "[. . .], there are two new Python programs .."

Done.

Referee #2: Dr. Rene Gassmoeller

I recommend this article for publication, but have one remaining comment that I think would be useful to consider in the performance discussion of the final article:

The MKL thread and MPI performance tests now look much better and clearly show the speedup users can expect. For MPI you see very reasonable speedup that is almost linear up to 4-8 cores on the desktop and up to 16-32 cores on an HPC Node (let's say 6 cores on the desktop and 24 cores on the HPC node). You justify the reducing computational efficiency for larger core numbers in turn with a too small model size (for MKL threads), possible background load on the desktop (MPI desktop), or not sufficient model numbers to utilize the full HPC node (MPI HPC node). However, at least for the MPI tests, what I think more likely is that you see the transition from a compute bound to a memory-bandwidth bound algorithm. Every system has a maximum memory bandwidth that has to be shared across all MPI workers (or all MKL threads). When you increase the number of participating compute workers (threads or MPI processes) you increase the load on the memory bandwidth until the model time is no longer limited by the available compute resources, but by the speed with which new data can be delivered to the CPU cores. This is a hard limit that will be the same for different parallelization options (MPI/threads), but varies between different compute architectures (desktop vs HPC). This would also fit the observation that Core 12900 and Xeon Gold 5218 have similar maximum memory bandwidth (google search result: up to 75 GB/s for 12900, and up to 128 GB/s for 5218), but 4 Xeon Gold would have 4x the memory bandwidth (therefore increasing your scaling from $\tilde{6}$ cores to $\tilde{2}4$ cores). All of this is just a back of the envelope calculation of course since I dont know your actual memory bandwidth. However, many finite element codes are memory bandwidth limited, because they rely on sparse matrix - vector products, and short of rewriting the algorithm there is not much you can do about it (see e.g. the introduction of Kronbichler & Kormann, 2012, https://www.sciencedirect.com/science/article/pii/S0045793012001429 for a description of the problem, or Clevenger & Heister, 2021, https://onlinelibrary.wiley.com/doi/full/10.1002/nla.2375 for an implementation of a matrix-free algorithm for the Stokes equation that circumvents the memory bandwidth limit).

• I think it would be useful to mention the possibility of a memory bandwidth limitation somewhere in the performance discussion, because it gives a uniform explanation of the scalability limit you see. It also means that your scalability is not inherently limited to the speedup you observe, because if you increase the number of models further and distribute them across several HPC nodes you will see further speedup (each node increases the available memory bandwidth), therefore making the case that ShellSet can indeed be used for much larger studies (say ¿1,000 models) as long as sufficient HPC resources are available.

We have noted that memory bandwidth may be a cause of reduced performance increase, and stabilising, within the performance section. We have also added reference to this inside the conclusion.