Response to reviewers: asQ: parallel-in-time finite element simulations using ParaDiag for geoscientific models and beyond

January 2025

We thank the reviewers for their careful reading of the manuscript, and for their positive comments and useful suggestions. We will address each of the reviews in turn, with our responses listed below the reviewer's comment.

1 Review 1

This paper is a well-written overview of the ParaDiag-II and related parallel-in-time methods. Its primary focus is the implementation of ParaDiag-II in scalable Python and PETSc frameworks and the validation of this implementation using four example problems. I recommend the article be accepted with minor revisions. Following are some points to consider in revisions as well as some typos to fix.

1. "with a survey [of] previous research".

   Response: Corrected.

2. "in Schreiber et al. (2018); Schreiber and Loft (2019); Schreiber et al. (2019) used a related approach with coefficients": missing punctuation.

   Response: Corrected.

3. Missing period after eq. 25.

   Response: Corrected.

4. Comma rather than period after eq. 26.

   Response: Corrected.

5. It's a matter of taste, but one could imagine making Tc and Tb independent of kp, thus making Tc a measure of the communication cost per application of P and similarly for Tb. This construction seems to align better with the other

cost measures. The final line in equation 30 would still come out the same since kp would cancel. Feel free to reject this suggestion; I make it in case it had not occurred to you.

Response: k_p is the number of Krylov iterations per block solve, whereas m_p is the number of applications of P, so we will assume that the comment was intended to refer to m_p and not k_p. In which case, it is indeed possible to make T_b and T_c independent of m_p, and we agree that this formulation better reflects the influence of m_p, k_p, and T_c on the total time, particularly the fact that T_c/T_b is independent of m_p. We will change the following two definitions:

- T_b will be defined as the time taken *per* block solve, i.e. $T_b = k_p N_x^{q-1}$.
- T_c will be defined as the time taken for collective communications *per* preconditioner application.

Equations 29-31 will now read:

$$T_p \approx \frac{W_p}{P_p} + T_c m_p \sim \left(k_p N_x^{q-1} + T_c\right) m_p \tag{29}$$

$$\begin{aligned} \frac{T_s}{T_p} \approx S &= N_t \frac{\left(k_s N_x^{q-1}\right) m_s}{\left(k_p N_x^{q-1} + T_c\right) m_p} \\ &= \left(\frac{N_t}{\gamma\omega}\right) \frac{1}{1 + T_c/T_b} \end{aligned} \tag{30}$$

$$\gamma = \frac{k_p}{k_s}, \quad \omega = \frac{m_p}{m_s}, \quad T_b = k_p N_x^{q-1} \tag{31}$$

and the text will be updated accordingly.

6. "$Tc << Tb$": use
ll rather than <<.

   Response: Corrected.

7. "the convergence criteria is": "criterion", singular.

   Response: Corrected.

8. "ARCHER2 consists of 5860 nodes, each with 2 AMD EPYC 7742 CPUs": I suggest some terminology be clarified. I conclude there are $2 \times 5860 = 11720$ CPUs total on the machine. My understanding of usage is that "CPU" = "processor". But Fig. 3 shows number of "processors" out to $16384 > 11720$, and Fig. 6 shows 32768. Now I'm wondering if the horizontal axis in Fig. 3 is actually number of cores, not number of CPUs. If so, is my usage incorrect? By "processor" do most people mean "core" and not "CPU"?

   Response: You are correct, there is an inconsistency between how we refer to the hardware between the text and the figure labels. The figure labels that say

"processors" should say "cores", to match up with the description in the text. These labels will be corrected in the revised manuscript.

9. "A quadrilateral mesh with 128^2 elements is used resulting in ≈65kDoFs, which is small enough to fit on a single core for the serial-in-time method." I'm not understanding. Is only one core used for the serial-in-time method?

   Response: Yes, one core is used for the serial-in-time method for this example. We chose to make this problem serial-in-space because a) LU has very low parallel efficiency for such a small number of DoFs and b) this means that the first example is isolating just the time-parallelism, independently of spatial-parallelism which we introduce in the later examples.

   We will update the text to make this reasoning more explicit.

10. These seem in conflict (64 vs 32 cores/node):

    - "The best performance was obtained by underfilling each node by allocating only two cores per L3 cache. This strategy is used in all examples unless otherwise stated, giving a maximum of 64 cores per node."
    - "It was found that the best results were obtained by allocating only a single core per L3 cache up to a maximum of 32 cores per node."

    Or does this apparent discrepancy have something to do with the complex-valued blocks?

    Response: These two statements do conflict, due to an error in our wording. This should read along the lines of:

    "Due to the memory bound nature of FEM computations, the best performance was obtained by underfilling each node by allocating *fewer than four cores per L3 cache*. For the two shallow water equation examples and the compressible Euler equation example, we use two cores per L3 cache, giving a maximum of 64 cores per node. For the advection equation example, using a single core per L3 cache gave the best scaling due to the small size of the spatial domain and the use of LU for the block solves, giving a maximum of 32 cores per node for this example."

    The text will be updated in the revised manuscript.

11. Fig. 9: In the legend, I think "$N\Delta t$" should be "$N_t\Delta t$".

    Response: The legend should indeed be $N_t\Delta t$. This will be corrected in the revised manuscript.

12. Fig. 13: Minor grid lines are on, unlike in the other figures. I like minor grid lines, but only if they are a much lighter shade; these are a bit obtrusive. I suggest either adding minor grid lines to all the figures or removing them from this one.

If the former, consider using pyplot.grid options to make thin, light-gray lines that are very faint compared with the rest of the plot entities.

Response: Reducing the thickness/shade of the minor grid lines of Fig. 13, in combination with reducing the ylimits, does improve the figure. Several other figures have large gaps between major grid lines (e.g. Figs 8, 9, 14, 15). We will include the changes to Fig 13 in the revised manuscript and add minor grid lines in the same formatting to other figures for better consistency and clarity.

13. Fig. 13: Consider reducing the y limits, especially at the top of the plot.

    Response: See response to previous comment.

14. Fig. 14: The "1000" in the top-left corner seems to have a rendering problem.

    Response: The formatting will be fixed in the revised manuscript.

15. "methods have been recently been implemented": Fix typo.

    Response: Corrected.

16. The numerical experiments are well presented and very interesting. I have one request. It is unclear to me that the block equations are being solved at the practical strong-scaling limit, i.e., space parallelism has been saturated, following the terminology in the first sentence of Sect. 3.3. I believe nonspecialist readers, an I am one, would benefit from one additional plot, probably for the nonlinear shallow water case, that shows the serial-in-time simulation at each resolution strong-scaled out in number of processors until speedup is lost. That is, take mesh 7, for example, and run the serial-in-time simulation on a range of processor counts, possibly up through all available processors. That way, the reader can gain some intuition about the following question: If I have N processors available to me, how should I configure my simulation in terms of Nt vs parallelism in space? An alternative is to clarify for the reader that one of two situations holds in the experiments: either (1) space parallelism is indeed saturated in the serial-in-time runs or (2) for the sake of tractability of experiments, you're pretending it is.

    Response: We have run the strong scaling sweep for the spatial parallelism for the nonlinear shallow water equations test case on mesh 7, and show the results in Table 1. This mesh has 3.44×10^6 DoFs, so the scaling goes from 1.72×10^6DoFs/core on 2 cores (using one L3 cache), through the 53×10^3Dofs/core on 64 cores, as used in the existing results (one full node), to 6.7×10^3DoFs/cores on 512 cores (8 nodes).

    At $P = 64$, the efficiency of the spatial parallelism is already down to 54%, dropping to 19% at $P = 512$, so we are reasonably far along the strong scaling curve for the results in the manuscript. The parallel-in-time results are also included for comparison. With $P = 256$, the parallel-in-time method with $N_t = 2$

Table 1: Strong scaling results for the nonlinear shallow water equations example using spatial parallelism for the serial-in-time method: T, time in seconds; S, speedup vs $P = 2$; η, parallel efficiency in percentage vs. $P = 2$. Parallel-in-time results are shown for: N_t, the number of timesteps; S, the speedup vs the serial-in-time result with $P = 2$; η the parallel-efficiency vs the serial-in-time result with $P = 2$; S and η vs the serial-in-time result with $P = 64$ are shown in brackets.

	Serial-in-time			Parallel-in-time			
P	T	S	η	T	S	η	N_t
2	85.0	1	100	-	-	-	-
4	47.7	1.79	89	-	-	-	-
8	25.9	3.29	82	-	-	-	-
16	14.0	6.07	76	-	-	-	-
32	8.29	10.3	64	-	-	-	-
64	4.91	17.3	54	-	-	-	-
128	3.25	26.2	41	-	-	-	-
256	2.67	31.8	29	3.22	26.4 (1.53)	21 (38)	2
512	1.78	47.8	19	1.62	52.5 (3.03)	20 (38)	4
1024	-	-	-	1.21	70.2 (4.02)	14 (25)	8
2048	-	-	-	0.96	88.5 (5.11)	8.6 (16)	16
4096	-	-	-	0.56	152 (8.77)	7.5 (14)	32
8192	-	-	-	1.09	78.0 (4.50)	1.9 (3.5)	64

is slower than the serial-in-time method with the same number of cores by about 17%. However, from $P = 512$, the parallel-in-time method with $N_t \geq 4$ overtakes the serial-in-time method. Note that for the parallel-in-time method it would be possible to increase the number of cores used for spatial parallelism further, for a lower DoFs/core count, to increase the total speedup further.

We will include a figure with these results in the manuscript, along with a brief discussion of the implications for choosing the parallel partition in space vs. time.

17. There is a fair bit of python code in this manuscript. If GMD permits it, it would be nice to use the Latex listings package or an alternative rather than verbatim.

    Response: The listings package is included in the GMD template. The code is clearer with this formatting, and will be updated in the revised manuscript.

2 Review 2

This paper introduces the asQ library for implementing ParaDiag parallel-in-time methods, focusing on applications related to weather and climate. The paper is really well written, and I was impressed by both the asQ library design and numerical

results. The authors gave a nice intro on other relavent research with a comprehensive set of references. They also did a good job of describing the asQ library and positioning it relative to existing methods and software. The numerical examples are known to be difficult to solve parallel-in-time, yet excellent speedups were achieved in all cases. The authors are aware of the current limitations of ParaDiag, but have several ideas for future research to expand its applicability. I only have a few minor comments.

Minor comments:

1. Page 2, paragraph before 1.1. Change to "a survey of previous research".

   Response: Corrected.

2. Page 6, after (13). Change to "inverse $x = P^{-1}b$" or "inverse of $Px = b$"? This is what the algorithms seems to be computing.

   Response: Corrected.

3. Page 10, second sentence after (36). Change to "backward Euler" (no "s").

   Response: Corrected.

4. Page 10, "time_partition = [2, 2, 2, 2]". This description grows with P_t. This may not be a problem for a while (or ever, since roundoff grows with N_t), but I was curious. Is there a P_t independent way to describe the partition in asQ? Are there any other things about the interface that don't scale?

   Response: The `time_partition` has N_t elements, but we do not expect this to harm performance because we never loop over `time_partition` during the solves. We define the `time_partition` as a list because in the future we plan to experiment with a variable number of timesteps per ensemble comm (see the last paragraph of Section 5.2.1). However, allowing the user to pass just P_t and a constant number of timesteps per ensemble comm would be trivial to implement and would cover most use cases, so we will consider allowing this in future asQ versions.

   The only other part of the interface that could scale with N_t or P_t is the specification of the PETSc solver options for the blocks. The standard use is setting default options for all blocks with `'circulant_block':{<options>}`, which is independent of N_t and P_t. However we allow setting options for a specific block `i` with `'circulant_block_i':{<options}`. It is also possible to mix the two, setting defaults for most options but specifying block-dependent values only for a subset of options (e.g. using the same preconditioner but different convergence criteria for each block).

   If the user sets options for each block separately then this would scale with N_t, but it is possible to set these options only on the "local" communicator for each block (e.g. `if ensemble.ensemble_comm.rank == n:...`), which reduces the total footprint.

5. Page 16, description of strong scaling. I was really confused by this until I saw the subsequent comment about nonlinear problems and the "full timeseries of N_T timesteps" broken up into windows. I'm not sure why you don't describe the linear problems in the same way, especially given that you use the term "windows" when discussing the numerical results. Related to this, is the time given for the serial-in-time method in Figure 3 for 16,384 timesteps (which you could call N_T)? Also in Figure 3, are you plotting T_p/N_t or (T_p/N_t)*N_w (where T_p is the time to solve N_t timesteps in a window)? It looks like maybe you don't compute the full timeseries in the linear case, and instead compute only one window of size N_t and simply assume (reasonably) that the full N_T timesteps would be N_w times larger. If this is all true, I think it would be less confusing to present it this way because it lines up exactly with the standard definition of strong scaling. If I've misunderstood, then some additional detail is needed.

   Response: The reviewer's interpretation is broadly correct, and we agree that the explanation of how the timings are calculated could be much clearer. We will attempt to explain more logically here, and will update the explanation in the revised manuscript accordingly.

   - We want to solve a (large) number of timesteps N_T. However, ParaDiag may be more efficient when solving an all-at-once system with $N_t < N_T$ timesteps, so we split the timeseries of N_T timesteps into N_w windows of N_t timesteps each, with $N_w N_t = N_T$. We then use ParaDiag to solve the all-at-once system for each window sequentially.
   - For the linear equations, for a given α the number of iterations to solve each window is constant. Therefore, if α and N_t are fixed then the time taken to solve each window is also essentially constant. This means that we can solve a single window of a given N_t and extrapolate the time taken to solve N_T timesteps.
   - Likewise, the time taken to solve each timestep of the serial-in-time method is essentially constant for linear equations, so we can solve one timestep and again extrapolate.
   - For the linear equation examples, for each α and N_t combination we actually solved five windows and averaged the timings to account for network variability, although in practice we observed very little variation between solves.
   - For the nonlinear equations, for both the serial- and parallel-in-time methods, the number of iterations per timestep/window solve varies as the nonlinearity in the local solution changes. For these cases we calculated the entire timeseries of N_T timesteps for both the serial- and parallel-in-time methods, and the timings shown are for the entire calculation (i.e. the sum of T_s over N_T timesteps and of T_p over all N_w windows respectively).

- Following this explanation of the windowing approach, we hope that the definition of strong vs weak scaling in terms of resolution - rather than in terms of DoFs/core - will be clearer to understand: The solution we seek is N_T timesteps at a given resolution, so by increasing the window size N_t we are using more cores to obtain the same solution - i.e. we are strong scaling - even though the DoFs/core remains the same, which usually implies weak scaling.

In response to the question about the data in Fig. 3, the timings shown are the time taken *per timestep*, not time taken for the whole timeseries, i.e. we have plotted T_p/N_t where T_p is the time taken for a single window solve. However, plotting the time taken to solving all N_T timesteps would be identical, because all data points would simply be scaled by N_T.

6. Page 27, 5.2.1. Delete one of "been" in the second sentence.

   Response: Corrected.

3 Review 3

The paper introduces asQ, a new software library built on top of the Firedrake software, for rapid testing and development of the ParaDiag parallel-in-time algorithm. While tailored towards earth system modeling, solved equations can be specified via the Uniform Form Language (UFL), making the framework quite general. The sections of the paper provide a detailed introduction into ParaDiag, the numerical algorithm, including a comprehensive review of the relevant literature. Then, the asQ library is described including the used space-time parallelization paradigm. The reader is walked through an example how to solve the heat equation with asQ to illustrate the steps that are needed. Wallclock scaling is shown for different numerical examples. While speedups for linear problems are extremely good, the fact that an averaged Jacobian needs to be used in ParaDiag results in much more modest speedups for nonlinear problems which are, however, very much in line with what other PinT methods deliver.

Overall, I think this is a strong, timely and well-written paper. Providing high-quality software libraries is exactly what is required to push parallel-in-time integration methods to broader use and there are still only few codes that do this. Therefore, asQ and the paper clearly fill an important need. Explanations in the paper are very clear and should be understandable for non-PinT-experts. The numerical results are convincingly linked to theory while the shown speedups are well explained using the provided performance model. I have only a few fairly minor issues that should be addressed before publication. The used code is provided open-source and numerical experiments can be reproduced via Python scripts and a complete installation is available as a Singularity container.

Specific comments.

1. I am not entirely sure I understand why the transpose and thus all-to-all communication is needed. Is this a consequence of the way the method is implemented or is this intrinsic to ParaDiag?

   If I understand correctly, in Steps 1 and 3 on p. 6 only a (I)FFT in time is required for every spatial DoF, but the (I)FFT for each spatial DoF is independent from all others. Since the number of parallel copies of one spatial DoF (that is, N_t) is small(ish), is it not possible to store all the temporal copies of a given spatial DoF on the same node?

   Basically, let us say the spatial parallelization breaks down the spatial domain into four sub-domains Omega_1, Omega_2, Omega_3, Omega_4. Then say we use four parallel time steps. Cannot Node 1 hold the four temporal copies of Omega_1, Node 2 four copies of Omega_2 etc? This way, the (I)FFT in time can be computed without sending messages. This could even enable some "hybrid" space-time parallelization where the (I)FFT are parallelized with OpenMP or some other shared memory paradigm to avoid having to transfer full solutions in time. Is this possible in principle but not within the used framework, just very challenging to implement or is my understanding incorrect?

   I am not suggesting that the authors implement this but I would be interested in some more details on why the transpose is required and if, theoretically, there are ways around it.

   Response: Referring to the diagram of the communicator layout in Fig. 2, the reviewers suggestion would mean grouping all ranks of each temporal communicator (the vertical grey `ensemble_comm` lines) on the same node (i.e. contiguous ranks), whereas we have chosen to always group the spatial commmunicators (the horizontal blue lines) contiguously.

   This alternate rank layout is entirely possible, although it is not exposed in the current implementation of the Firedrake `Ensemble`. However, we did briefly experiment with grouping the ranks of the temporal communicators contiguously, and found it to give worse performance than grouping the ranks of the spatial communicators contiguously.

   Taking the example in the comment, the (I)FFTs could be carried out with no off-node data movement if the temporal communicators were contiguous. The profile in Fig. 4 shows that the FFT calculation time is negligible, so even if it were parallelised we would not expect a significant speedup gain, therefore the real win is eliminating the transposes (or at least carrying them out over shared memory, which could use OpenMP or MPI shared memory protocols). However when computing the block solves, each spatial domain is now distributed over four nodes, so every single halo swap requires off-node communication. This will severely degrade the performance of the block solves, especially if we assume that a) spatial parallelism has been saturated in the serial-in-time method before time-parallelism is used and b) the serial-in-time method uses all cores of a single

node. In which case, the spatial parallelism with only on-node halo swaps is already the lowest we are willing to accept, and will be even worse for off-node halo swaps.

If we then increase the number of timesteps to greater than the number of cores on each node, then it is no longer possible to hold all timesteps of one spatial subdomain on a single node. In which case, some off-node communication will still be required for the FFTs, and because the off-node message sizes will still be $\mathcal{O}(N_x/P_t)$ the Bruck algorithm is unlikely to be used (Sec. 3.3). This means that the FFT communication will have the same algorithmic scaling as if we had grouped the spatial communicators, but the block solves will be far less efficient due to the off-node halo swaps.

We do note that grouping the temporal communicators may be the more efficient choice for algorithms where the amount of computation requiring all timesteps of a single spatial subdomain, independently of the other spatial subdomains, is much higher. For example waveform relaxation methods which require solving the entire timeseries within each subdomain independently.

Currently the rank layout is not discussed in the manuscript but, given its importance, in the revised version we can add a sentence in Sec. 3.3 stating that we always group the spatial communicators to maximise the parallel efficiency of the block solves.

2. Eq. (24), I am not sure I understand what the Lip operator means. Is that the minimum L such that satisfies a Lipshitz condition for f(u) - Nabla f(u_tilde) * u ? But as what, as a function of u with fixed u_tilde? May be it is clearer to simply spell out the condition that the kappa needs to satisfy.

Response: The reviewer is correct in their interpretation of κ. Specifically, for a given $\hat{f}$, $\hat{u}$ and $\hat{t}$, we can define the function $g(u;t)$ for the "error" in the Jacobian $\nabla_{\hat{u}}\hat{f}\left(\hat{u},\hat{t}\right)$ as:

$$g(u;t) = f(u,t) - \left(\nabla_{\hat{u}}\hat{f}\left(\hat{u},\hat{t}\right)\right)u. \tag{1}$$

Then, κ is the minimum value satisfying the Lipschitz condition for g with respect to u over the time-domain of interest, $T = N_t\Delta t$.

$$\|g(v;t) - g(w;t)\|_\infty \leq \kappa\|v - w\|_\infty \;\forall\; t \in [0,T] \tag{2}$$

This is simply a restatement of Remark 3.8 on Theorem 3.6 of (Čaklović, 2023, ParaDiag and Collocation Methods: Theory and Implementation) in our own nomenclature and with the inclusion of t in f and $\hat{f}$.

We will include this definition in the revised manuscript, along with reference to the specific Remark and Theorem in (Čaklović, 2023).

3. Some of the figures are a bit hard to read in print. I would suggest slightly thicker lines and slightly larger markers. The yellow coloured lines are also sometimes difficult to see in print, maybe some darker colour would help.

   Response: In the revised manuscript, yellow will be replaced with a darker colour, the line thickness will be increased, and the marker size will be increased where this does not cause obscuring overlap between data.

4. There are a lot of parameters to keep track of $(N_x, N_t, k_s, k_p, m_s, m_p, ...)$ and I found myself going back and forth a lot to find their definitions. Having them all summarized in a concise table for quick reference would help a lot.

   Response: A nomenclature table will be included in an appendix in the revised manuscript.

5. The description of the main steps of ParaDiag in the paragraph after Eq. (27) could probably be summarised in pseudo-code.

   Response: Given how many nested levels are required to construct ParaDiag solvers, we agree that a pseudo-code or algorithm block would be useful for the reader. We will include one to summarise the explanation of these levels in the paragraph after Eq. (26). In the paragraph after Eq. (27) discussing the most time consuming components of the algorithm we will reference the relevant lines in the pseudo-code.

Technical corrections/minor comments.

1. The unit in which wallclock time is measured seems not to be stated and is missing from the axis labels in Figures.

   Response: The wallclock time is measured in seconds. This will be added to the figure labels in the revised manuscript.

2. There is a very recent preprint discussing integration of parallel-in-time functionality into the Nektar++ code. This effort probably deserves to be mentioned in the discussion of the (few) general-purpose software packages supporting PinT.

   Xing, Jacques Y. and Moxey, David and Cantwell, Chris D., Enhancing the Nektar++ Spectral/Hp Element Framework for Parallel-in-Time Simulations. Available at SSRN:

   https://ssrn.com/abstract=5010907 or http://dx.doi.org/10.2139/ssrn.5010907

   Response: Thank you for bringing this manuscript to our attention, we will include it in the introduction and in the comparison of asQ to other PinT software in section 3.1.

   It has also come to our attention that the "Paralpha" software used in (Čaklović et. al., 2023, A parallel-in-time collocation method using diagonalization: theory

and implementation for linear problems) should also be cited as another ParaDiag code that is both space-time parallel and has capability for general equations. We will include it in the relevant sections as well.