Geoscientific Model Development Discussions

Geosci. Model Dev. Discuss., referee comment RC2
https://doi.org/10.5194/gmd-2022-214-RC2, 2022

Comment on gmd-2022-214

Mauro Bianco (Referee)

Referee comment on "Improving scalability of Earth system models through coarse-grained component concurrency – a case study with the ICON v2.6.5 modelling system" by Leonidas Linardakis et al., Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2022-214-RC2, 2022

The paper describes the effect of executing concurrently different components of an ESM in terms of performance: ICON-O + HAMMOC. Each of the components are implemented as distributed memory programs (MPI) with multithreaded inter-process execution (OpenMP). A mathematical model of the execution describes the trade-offs of running the components sequentially or in parallel, and gives an intuition on what are the constraints in the expected performance improvements. Performance evaluation is done on two model resolutions and two computer architectures, with an interesting analysis of the results that refers back to the model of execution. This gives the results a conceptual link that is too often neglected in the literature. The paper focuses on ICON-O and HAMMOC, that use the same grid. As a paper I have found it very informative and useful. The mathematical model is not very surprising at the end, but it offers a valuable baseline to reason about the results.

Some points for discussion:
- The abstract needs improvement in my opinion. Line 7 mention "function level parallelism". In the computer science literature the term used is typically "task parallelism", and it is opposed to "data parallelism".
- Would it be possible to explain in few words what are the limitations to scalability mentioned in in line 15 about "traditional parallelization techniques". I do not see the logical implication here, more so given that ICON-O and HAMOCC use the same grid. Is it a problem with software structure? That is, would an implementation with even less modularization (more monolithic) avoid this problem?
- As my main interest is in software architecture and engineering, I think it would be very interesting to me and useful to the community to expand on the implications that the software restructuring has on the code base. For instance it would be interesting to mention the (qualitative) effects of the ability to run sequentially or concurrently in terms of code maintainability and readability.
- I find the sentence in lines 214-215, about the fact that OpenMP does not incur in communication costs, not precise. It depends what communications we are considering. OpenMp can be quite costly in case of data to be access by different threads for instance. Maybe the sentence should specify that the communication cost refers to extra-node or

extra-process communication.
- I find the paragraph in line 227 not very convincing, since it is not clear to me what "a high-level implementation" means here. Is it just because it focuses on MPI to transfer data between MPI ranks?
- In Section 3.2 I think the treatment could be improved by removing the subscript "p" from "a_p" and use instead the letters "A" or "B" to indicate to which component the value refers to. Also the parameter lambda should be introduced more specifically, since it not immediately clear why the same lambda is used in W_B and N_B. It can be deduced, but it becomes clear later in the text. This could be explained earlier. Similarly, the cost of concurrency "C" could also be introduced with an example of what it may include.
- The "at most linear" scaling seems to actually mean "monotonic", since F'(N)<=0.
- In Section 5 the Authors mention that they ran three times each experiment. It could be useful to report on the variability of the execution times in those three runs to justify the use of such small number. If other limiting factors were in place maybe it is worth mentioning.
- In Table 1 and 2 the lambdas are greater that 1, while in the mathematical analysis it is assumed to be less than 1.
- The paper focuses on a simulation software with two components. Could a comment be made on the possibility, both in terms of software structure and performance benefit, of applying concurrency within the said components (I guess in "shared memory" style (see end of Section 3.1))?
- Line 259: I would use "assume" instead of "accept"