Thank you for reviewing the paper and for your comments. I have adjusted the revised version of the paper accordingly, and here I provide the answers (in black) to your comments (in blue).

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".

I have rephrased the sentence to “component concurrency is a function parallel technique, it decomposes the algorithmic space, while these parallelization methods are data parallel techniques, they decompose the data space.”

I understand that the ‘function (or functional) parallelism’ term is a bit old-fashion, and ‘task parallelism’ is more commonly used nowadays. Nevertheless, it is a valid term, and I prefer it for expressing the general concept, as ‘task parallelism’ in some cases is associated with shared memory parallelization paradigms, for example in OpenMP.

- 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?

This whole paragraph refers to the work done in this paper, to further underline it I start the paragraph with “In this work we study the characteristics...”. The phrase “traditional parallelization techniques...” indeed needs clarification: I changed this to “data parallelization techniques (domain decomposition and loop-level shared memory parallelization)...”. The scaling limitations of these techniques lie on the size of the grid, not the software structure. This is further analyzed in the paper.

Software structure and engineering is another large subject, which also has implications on multi-level parallelism implementation. This would require a study on its own, as it is rather a complex issue.

- 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.

This is a very large and complex issue. I can only offer a short comment here. In some cases, the quick and dirty approach is much easier. For example, it's easier to add a few lines of code in the ICON-O surface fluxes module in order to add the HAMOCC surface fluxes, accessing both the ocean and the HAMOCC information, instead of doing so in HAMOCC. Modular design and interfacing is harder, but in the long run probably more robust and advantageous in code-development. A very interesting subject which I would like to discuss in the future, this is a study on its own.

- 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.

Here I use the term communication in a strict sense as direct communication between parallel processes. I have added a footnote: "Here by communication it is meant direct communication between the parallel tasks. The cost of it can be significant when it takes place through the network." While there are cases that some communication may occur in shared memory parallel regions (for example reduction, or memory locking) these are not the typical case in our codes. Some communication cost occurs when for example flushing the cache after a parallel region, but I put them under the "overhead" umbrella.

- 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?

Indeed, the "high-level implementation" needs further discussion. I have rephrased it as follows: "The other aspect of these two options is the implementation. The distributed memory case is independent of the architecture, and can even be applied in hybrid mode. While in principal the same functionality can be achieved using shared memory parallelization, no such standard, to the authors' knowledge, is currently mature enough to be implemented across multiple architectures." For example, the same concurrent implementation can be used the run ICON-O on CPUs and HAMOCC on GPUS (which is one of our current projects).

- 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 subscript p is used here to indicate the parallel workload in a parallel region, in contrast to the total workload. This is a bit tricky since a parallel region is defined as any parallel region (shared or distributed) enclosed between two syncs, in any component, independently of sequential or concurrent setup. I have added subscripts A,B,AB to identify the cases of concurrent and sequential runs of A and B.

I attempted to clarify here why N_B = lambda N_A is taken, and to indicate the concurrency cost, but then I would only repeat what follows. On the other hand, one of the advantages of not including this information in this paragraph is to demonstrate that increasing the parallel workload (and thus concurrency) is only useful in the context of scaling concerns.

- The "at most linear" scaling seems to actually mean "monotonic", since F'(N)<=0.

The sentence is rephrased as: “We assume that the parallel efficiency is a non increasing function of N, that is F'(N) <= 0.”. This is equivalent to S'(N) <= S(N)/N, where S is the parallel speed-up. In this sense I used the term “at most linear scaling”, which I dropped, since this is a bit of a vague term.

- 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.

The three runs were not meant to provide some statistical confidence, but to avoid accounting a worst case run. In these worst cases the runs can be up to two times slower (due to hardware/system malfunction), and would result a clear outlier. A few of the runs were such cases, but they did not affect the overall results, as they were only one of the three runs.

- In Table 1 and 2 the lambdas are greater that 1, while in the mathematical analysis it is assumed to be less than 1.

Yes, in this case L(N, lambda)=F(N(1+lambda))/F(N lambda). I have added a clarification on this in the experiments section. I have taken lambda < 1 in the analysis because it makes it simpler. On the other hand, I had in mind the scenario that we have a model A (ICON-O) and we add model B (HAMOCC), so I kept this convention, resulting lambda >1 in the experiments. The conclusions do not change.

- 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))?

Indeed, additional coarse-grained concurrency can be implemented within ICON-O, to the sea-ice module for example. Interestingly, this also probably would be better off with a distributed memory implementation, since the sea-ice is a 2D model, running in an effectively smaller grid than the ocean, but on the other hand it has a lot of global reduction operations, which incur high communication cost for large number of MPI processes.

Shared memory task parallelization can also be further applied. For example in the tracer transport, parallelizing over the number of tracers. In paragraph at line 306 we provide a short discussion. Another candidate for shared-memory concurrency is the diagnostics, the cost of which could be significant depending on the setup. Other processes, such as calculating the tides' potential, may provide opportunities for middle-level shared-memory task parallelism.

- Line 259: I would use "assume" instead of "accept"

Done