Synchronous Deterministic Parallel Programming for Multicores with ForeC

(manuscript TOPLAS-00030-2016)

Responses to the reviewers

Eugene Yip   Partha S. Roop   Alain Girault   Morteza Biglari-Abhari

June 18, 2017

We thank the anonymous Reviewers for their comments and suggestions, which have helped us to improve the quality of our paper. Please find below our responses to Reviewers 1 and 2.

1 Responses to Reviewer 1

Comment: *The presentation is definitely effective, and my only suggestion for improvement is related to the structure: the discussion of related work is scattered over the article (introduction, Section "Related Work", other subsections in the following). Maybe all these parts could be rearranged in a more consistent way, e.g. combining all the parts of Sections 1, 2, and 3 in a single section "Related Work" with appropriate subsection, and then simply keeping the current "Discussion" subsections (or renaming to "Related Work" if new related work is rather introduced).*

Response: A response.

Comment: *Applicability/Restrictions. (1) From my point of view, this is a very weak point of the presented approach. I am rather sceptical that the presented model of parallelism could be really used in practice. Although the authors claim that it is perfect for parallel execution, there is still the single-level barrier synchronization, which enforces that all the threads have to meet at the end of a global tick. This means that even the threads, which do not communicate at all, have to synchronize all the time. An acceptable performance for a real embedded software application having a size to use a multi-core processor, would be out of reach from my point of view. The industrial example I know from e.g avionics of the automotive domain are completely out of reach for the presented approach.*

Response: A response.

Comment: *(2) Furthermore, the same single level of synchronization exists is also there in the model. Mixing different tasks of different timing granularity (as it could be done in Lustre) is impossible. Thus, the given approach is only feasible for data-intensive applications having a single level of time granularity.*

Response: A response.

Comment: *(3) It should be also stated that - compared to Esterel and Lustre, ForeC is on the implementation level - a lower level of abstraction. The threads in the program are also "threads" in the implementation (either pieces of atomically executed code or real threads), which is different to Esterel, where the aim is to model parallel threads and executing them sequentially. This is a true limitation since the restrictions that parallel threads could not synchronize in the course of a global tick gives the division into threads a real semantic difference. Thus, the program author already has to think how to parallelize the program (a design decision). Of course, the better performance is due to this point and - naturally - the restriction that parallel threads could not synchronize in the course of a global tick.*

Response: A response.

Comment: *However, before a possible publication, the article should be carefully revised. With respect to the technical content, I do not any significant points. However, the authors could either clearly state the limitations of their approach so that people not very familiar with the area do not have wrong expectations. Of course, I would even prefer that the authors continue to work on their language - and maybe see the possibility to address my concerns. Furthermore, as said above, the authors could consider to give a better structure of the related work parts.*

Response: A response.

Comment: *What, exactly, can your language do and not do?*

Response: A response.

Comment: *What, exactly, were the challenges in designing and implementing such a language and how did you overcome them?*

Response: A response.

Comment: *How, exactly, does your work differ from others?*

Response: A response.

Comment: *A really fundamental question I couldn't figure out is exactly which C features you support and which you don't. For example, a lot of the analysis you seem to do appears to rely on the absence of pointer-induced aliasing, arrays, and various other C mainstays. Exactly what types do you support? Do you support arrays and other aggregate types? How are they dealt with w.r.t. your mechanisms for preventing race conditions on access to shared structures?*

Response: A response.

Comment: *Do you support recursive functions? If so, how? If not, exactly what constraints do you put on the call graph of your program? Do you support function pointers, for example?*

Response: A response.

Comment: *Do you support recursive data types (e.g., trees implemented with structs and pointers)?*

Response: A response.

Comment: *You appear to insist every loop have a bound placed on it (your # notation). How do you, if at all, check that these bounds are respected? Are these only for loops whose bodies don't include pause statements, or do they apply to every loop?*

Response: A response.

Comment: *Your claims that your technique can produce code that's more efficient than Esterel and OpenMP need to be backed up more and made more clear. What things were you measuring? How were the Esterel and OpenMP programs compiled? What made your solution better? How does your solution scale with an increasing number of available cores?*

Response: A response.

Comment: *A central technical contribution, which I wasn't able to understand fully, was how you implemented your language on multi-core processors. Really basic things, such as whether the generated code relied on a particular number of threads or would create more at runtime, weren't obvious.*

Response: A response.

Comment: *Exactly how are you implementing your "copy on fork; combine on join" semantics for local variables? It seems like this could easily be extremely costly and wasteful (e.g., if in fact only one thread ever touches certain variables). I didn't see how you were implementing this at all, let alone efficiently. This seems fundamental to your solution; please explain how you handled it.*

Response: A response.

Comment: *Do you map each ForeC thread to a unique pthread, or do you map multiple logical threads onto a single operating-system thread? I'm sure I could probably eventually figure this out from your discussion, but it should have been stated much more clearly right at the beginning.*

Response: A response.

Comment: *This is one paper that would greatly benefit from having the discussion of related work moved later. At the moment, it takes you something like 12 pages to get to actually discussing any contribution. Why you do indeed have to explain how your technique relates to the functional languages and related work, it would greatly benefit your presentation to move that later. I really want to learn how you solved problems rather than get a history lesson on all the other techniques in this space.*

Response: A response.

Comment: *Your formal presentation of semantics is encouraging, but I'm not really sure how much it adds. Given that others have argued the formal semantics and determinism of, say, Esterel, wouldn't it be enough to explain how your semantics differ, if any, from those other works? Nothing dramatically different (i.e., constructs that demand a greatly different treatment) jumped out at me, but perhaps I missed it. In either case, either stress how these semantics are somehow critical or novel, or explain how they're essentially the same as others.*

Response: A response.