Response letter

We would like to thank the reviewers for the constructive comments which are addressed in the revised manuscript. In this response letter, we summarize the changes.

Reviewer: 1

Summary: this paper propose a generalized embedded block (EB) architecture for low precision DNN, called ML-Blocks. MLBlocks includes several MACs units connected via a flexible routing, where each configuration performs a few parallel dot-products in a systolic array fashion. To maximize MACs utilization with minimal routing, they propose a tool that can generate an MLBlock that supports a set of subset of loop unrollings that best satisfy the area/performance trade-offs. The main purpose of MLBlocks is to ensure maximum utilisation of these MAC units over a set of benchmark set with minimal routing overhead.

The need for efficient low precision MAC units is the important topic for DNN hardware accelerator, as research shows little lost in performance when running DNN inference.

- As the result in the paper might sound promising, it would have been great to actually evaluate layer-wise mapping of DNN such as VGG or ResNet as in [1]. As for the benchmark used, they description is not clear. Looking at the reference [2] for the Baidu kernels, but not clear explanation is provided on the web site. In that case, the authors provide a clear description of the benchmark for a better evaluation of their paper. For instance, regarding the GEMM operation, it should be wise to provide the dimension of the inputs as on the website to keep the two coherent. Similar to convolution and RNN (hidden units).

  We added a new section (Section 5.6), which compares our work with previous works. In this section, we evaluate different architectures using all of the convolution and fully-connected layers from both VGG16 and ResNet18 for an input batch size of 2. Also, we updated Table 1 to present the details of the selected kernels from DeepBench.

- The pseudo code in Algorithm 2, is it for classic 2D or 3D convolution.?

  Algorithm 2 is pseudo-code for classic batched standard 2D convolution layer. To make this clear, we have updated the caption.

- What is the resources utilization in terms of BELs?

  Unfortunately, we did not understand the meaning of the term "BELs". Our guess is that it is a typo for the term "CLBs".

  With this in mind, we added Subsection 5.4 to report the implementation results for MLBlocks as overlays implemented with LUTs. We report CLB and register utilization, as well as the maximum working frequency. Please note the Verilog models are not efficient for FPGA resources. The results show 1) the resource overheads and 2) performance efficiency for different variations of MLBlocks compared with instantiating the equivalent numbers of 8×8 multiply and 32-bit accumulator units.

- What does F stands for in the paper? It is not explain in any picture. The authors should provide more explanation about the acronyms used in the paper. The definition of the different parameters are explained later on (page 13). I think it should be done before the paragraph unroll to make the reading more comprehensive.

  Thanks for noticing our omission. In Section 3.2, which is placed just before the "Unroll" Section, We clarified the definitions of the parameters used in Algorithm 2 (which presents a pseudo-code for a batched standard 2D convolution). X, Y, and B defines the input height, width, and batch size of the convolution. K defines the number of filters with the height, width and depth of F_X and F_Y and C.

  Furthermore, we highlighted their bidirectional mapping to parameters in Algorithm 1 with comments in Algorithm 2 and corresponding text in Section 3.2.

- In Figure 7.b What Verilog model is generated is it for the whole architecture, or only MLBlocks? that paragraph is not very clear.

  In this case, for each N (from T) configurations, the tool generates the corresponding Verilog model for an MLBlock instance and synthesizes it. We clarified this point in the manuscript in Section 3.5. Also, we updated Figure 7 with two flowcharts for both approaches.

- In Table 1, what does "# of cases" stand for?

  In the previous version of Table 1, "# of cases" refers to the number of kernels we picked for each benchmark category, i.e., there are 19, 12, and 8 kernel cases respectively in GEMM, CNN, and RNN groups.

We have now modified Table 1 to present the computation of each kernel in detail. We describe the computation in terms of the iteration limit and stride for each of the variable types (batching, expansion and reduction). This should provide the reader greater insight into the benchmarks for which our work is optimised over. We have aldo modified the corresponding text at the start of Section 5

- Beside the power and area, what is a latency of EBs compare to DSP. When it comes to ML, especially inference, the latency is a critical factor. Also, additional experiments regarding the routing overhead will help proving the claims of the paper.

  To deal with the IO constraints, the suggested architecture has a port for one of the inputs (W). This means MLBlock-M requires M cycles to be (re)loaded. Also, due to the pipeline registers in MAC units, any arrangement of them requires a certain number of cycles to generate the first output. This latency may vary between the configurations of an MLBlock. In Section 5.5, our performance analyzer tool optimizes the implementation of kernels considering the total number of required clock cycles. This includes both (re)loading and the computation pipeline latencies. Actually, these latencies are the primary reason behind the performance reduction in Figures 14 and 16 where the low budget for the number of EBs causes more frequent costly (re)loadings. Consequently, MLBlocks could not deliver their best performance. To clarify this, we commented on these two latencies and their involvement in our performance analyzer in Section 5.5.

  Furthermore, in the newly added Section 5.4, we evaluate the MLBlocks as FPGA overlays. Considering the resource utilization of a single-cycle signed 8×8 multiply and 32-bit accumulation unit, our MLBlock models (without high precision support) have 37% resource overhead on average due to the additional routing. Resource overhead for an MLBlock-M is calculated by $\frac{\text{CLB}_{\text{MLBlock-}M} - M \times \text{CLB}_{8\times8\text{MAC}}}{M \times \text{CLB}_{8\times8\text{MAC}}}$, where $\text{CLB}_{\text{MLBlock-}M}$ and $\text{CLB}_{8\times8\text{MAC}}$ are the CLB costs for MLBlock-M and the single-cycle signed 8×8 MAC unit, respectively.

Reviewer: 2

- Parts of the description are hard to follow, particularly around the projections. As one example, the discussion around Algorithm 2 talks about r0,r1,r2, but these don't show up in Algorithm 2; I think this requires following (matching?) variables and mappings between Algorithm 1 and Algorithm 2. Maybe the reader needs more help seeing how to following the mapping.

  Thank you for this suggestion, we clarified the variable mapping from Algorithm 2 to Algorithm 1 in the last paragraph of Section 3.2. We also explicitly added this mapping as comments in Algorithm 2.

- The paper starts out claiming that this work is superior to related work because it comprehensively exposes and explores the design space rather than making manual choices. However, when it gets to Section 4 it selects a few points of the design from the start (e.g. W stationary) also section 3.4.3 limits exploration to windowing in only one dimension. So, maybe introductory claims are overstated?

  We believe that the proposed methodology in Section 3 is general and covers a wide range of possible architectures. However, we agree that the suggested architecture and constraints in Section 4 could be relaxed to cover a larger design space.

  Nevertheless, we use the suggested architecture as a case study to show how this generalized methodology delivers automation for finding a DSP-like replacement block for such benchmarks with higher performance and efficiency in comparison to the previously expert-designed architectures.

  We have added an extra bullet to the contributions to make this clear.

- The paper talks about related work, but does not provide any quantitative placement of the related work.

  We added a new section (Section 5.6) that compares MLBlocks with some of the previous works from both Xilinx and Intel architectures (from industry and academia). We also added a very recent DSP architecture, Xilinx DSP58, to the previous works.

- there's a claim that the Intel architecture [22] is a special case of your design. Which point is it? How well does it work? Could you mark it in Figure 11? include it in Figure 12 and 13?

  We apologise for the confusion. The AI-tensor implementation is different to MLBlocks. For instance, AI-tensor uses a pipelined adder-tree structure while MLBlocks are based on systolic arrays. AI-tensor includes a double buffering system to cover reloading latencies, which is not present in our architecture. As such, there is no data entry regarding this block in Figures 11, 12, or 13. This has been clarified in the last part of Section 5.6.2.

(Note Figures 12 and 13 compare MLBlock instances generated with the same the design constraints as the Xilinx architecture. Figure 11 shows the search space for MLBlock-12 instance only and compares the Greedy and 3-Config techniques.)

However, the AI-tensor can compute three 10-element dot-products (one input is shared). Focusing on the computation, we can describe this by a couple of projections: 1) <(1,-,-),10,3,1,1>(one shared 10-element input I and three sets of 10-element weights), and 2) <(1,-,-),10,1,3,1>(10-element shared input W and three batches of 10-element of I). If the number of MAC units is 30, our tool automatically considers these two projections along with other choices such as <(1,-,-),3,10,1>and <(3,1,1),10,1,1>.

Therefore, the MLBlock methodology does cover the underlying computation of AI-tensor, exploring it based on a systolic array architecture. Extending support for alternative architectures is a valid suggestion, again we have added this to our future work.

- can you place other related works?

  We noticed the most recent Xilinx DSP architecture, DSP58, which is a superset of DSP48E2, offers native 3-element dot product operation for 9×8-bit signed fixed-point numbers. As well as adding this design into the previous works, we compared its performance with MLBlocks in the newly added Section 5.6.

- Your focus here is on revising the fixed block, and the big complaint is the the Xilinx DSP48E2 is inefficient on these lower precision operations. The paper doesn't establish that the low-precision calculations cannot be adequately addressed in the fined-grained LUT logic. It looks like your own work ([16]) and Betz's work ([17,18]) might be speaking to narrowing the gap in the fine-grained logic rather than in the hard block. How does that compare?

  Thanks for this suggestion. Due to the lack of accurate information, especially about the CLBs and routing footprint, and our limited tools, it was not possible for us to provide a reasonable model to measure the area cost of the same computation using logic elements. However, 8×8 LUT-based multipliers will not be efficient as embedded multipliers, and yet increased support for these operations is critical for block floating point [3], inference [4] and in particular training tasks [5]. The importance of these solutions for compute-bound DNN applications is likely to grow in importance in the future. We have modified the introduction to highlight this.

  In addition, to get an alternative view of these trade-offs, we also synthesized our MLBlock models as overlays to measure their resource utilization, presented in the newly added section 5.3.

  Nevertheless, we have added this suggestion to our future works, as well as denoting it at the end of Section 6.

- Which of the results are averages over the large benchmarks set? and which are for specific points? Table 1 suggests the benchmark set considers a wide range of values for some of the parameters. I think Tables 2 and 3 and Figure 11 are averages. Maybe Figure 13 is specific sizes. I'm less clear about Figure 12. Could you show distributes/ranges across the benchmark set for the cases where you are aggregating results across many benchmarks?

  We clarified this by elaborating more on the benchmark description in the second paragraph of Section 5. Briefly, our benchmark includes three categories of kernels, GEMM, CNN, and LSTMs, which have 19, 12, and 8 instances, respectively. The results are reported by averaging over all 39 (=19+12+8) instances, unless the three categories are mentioned (then, results are averaged over the instances of each group), or the instance names are used individually.

  It follows that you are correct: Table 2, Table 3, and Figure 11 all average over 39 cases. However, Figures 12 and 13 report the average for each individual category, as well as the total average. In contrast, Figure 15 reports the result for each individual benchmark case. Please note, Figure 13 differentiates the results for high and low precision kernels by adding 8 and 16 as post-fix showing kernel input data types.

  To make this clear, we updated Table 1, to present the details for each selected kernel. Finally, we added details onto the caption of each Figure/Table to make this clear.

- p. 111:2 "...supports a set of subset of loop unrolling...". Is "set of" an editing typo that should be removed? Otherwise, I have trouble parsing what this is supposed to be saying.

- p111:11 "...where the steamed data is delayed.." maybe "steamed" $\rightarrow$ "streamed"?

  Thanks, we addressed these two typos.

Reviewer: 3

Dear authors, Thanks for your work on this manuscript that highlights the benefits of computation efficient logic blocks towards emerging workloads like ML. The work is well motivated and presents the novel primitive block MLBlock and a systolic architecture that promises high gains in compute density per area compared to an industrial logic block featured in the Xilinx Ultrascale+ FPGAs. However, please allow me to share some of my concerns for which I would like to recommend a major revision for your manuscript.

- Firstly, in 3.4.2 it is not clear how you define the I/O constraints. Isn't it the case that the bandwidth metric should include the notion of time? How the unique computation projections take into consideration the time perspective so that the bandwidth can be quantified in a common used metric, instead of a normalized one ?

  In this work, we made the assumption that IOs deliver their values in parallel each clock cycle to the MAC units without having any special buffering, except for 1D windowing is a limitation of our models. This is a limitation of our models; currently, we are working on generalizing IO, considering both serialization and double buffering, as our future works.

  We added an explanation for this limitation to clarify this assumption (Section 3.4.2). We also added this point as our future works in Section 6.

- Secondly, it would be helpful if the selection (3.5) process can be better explained through an algorithmic description. It is unclear how the processes of "find the best utilization" and "find the best compute density" are actually delivering the claimed sets. What is the input/output of those processes and how the potential forms of windowing are selected at every step (incremental, step-wise) ? Of course the selected heuristic is definitely a sub-optimal process since it can deliver solutions with up to two order of magnitude difference in the experimentation results (util./area). While the authors state that an automated design-space exploration is needed, it would be helpful if they can cite related works in this perspective that motivate this future direction.

  We clarified the descriptions for the selection processes in the manuscript. Also, we updated Figures 7a and 7b with new flowcharts. The input to both flows are 1) benchmarks and 2) design constraints. The output is the Verilog model for the MLBlock unit. Any size of 1D windowing as a part of projection definition is always considered. The greedy approach is incremental, while the N-Config is an exhaustive search.

  Regarding comparing other works, we added Section 5.6, which compares MLBlocks with different solutions for Xilinx and Intel architectures from both industry and academia. This section shows how our methodology finds better solutions compared to the expert-designed approaches.

- Thirdly, I would like to highlight the absence of absolute metrics in the experimental results. Since the objective of this work is to propose MLBlock as an alternative to Xilinx DSP48E2 for ML workloads, using the STMicro 28nm technology node, why there are reported only normalized results for the compute density and the performance speedup? If the scale of the results does not allow the drawings to capture the complete scale, it would really help the reader quantify the results, if absolute values are referred in the manuscript, i.e. GOPS for performance and Watt for power. The area metric reported in um^2 is derived only from RTL synthesis or after PnR ? How exactly do you calculate this metric after synthesis (e.g. you get the kilogates and estimate the absolute value in um^2) ? How you get the same metrics from Xilinx? While it might be straightforward for the tools you are using, a detailed elaboration of the experimentation setup would improve the reproducability of the results of the manuscript.

  The main intention of using ratios was to make the comparisons easier. However, we totally agree with your suggestion. We are reporting post-synthesis results in um^2 directly from the tool's reports. To provide a better sense, we added the absolute area and power values for our DSP48E2 model, as it is our baseline. Other values can be deduced. This can be seen in Table 3.

  Furthermore, we have introduced additional area results for different MLBlock instances in the newly added Section 5.6. We used the Xilinx DSP48E2 Verilog model, provided by [6]. We will also make all the frameworks and scripts publicly available for easy reproduction.

- In Table 1 the details of the selected cases of DeepBench do not help the reader quantify the breadth of the compute kernels that MLBlocks can provide a better alternative over DSP48s. For example, what are the dimensions of the 19 selected GEMM cases? Is there a specific id associated with those GEMM cases?

  We have clarified the benchmark cases by updating Table 1, which now provides the details for each kernel.

  I would expect that there are high deviations on the results obtained for the different cases and the geomean values of Fig. 12-15 cannot capture such deviations. Maybe the authors can consider to add more figures with detailed results for every case.

Yes, you are right about the deviation among the results. However, there is clear supremacy (in low-precision arithmetic) for MLBlocks comparing to other counterparts. We elaborated more on this point by adding Figure 15, showing the performance speed up results for each benchmark case in a situation where only 4800 EBs are available.

In this figure, for CNNs, the speedups are consistently close to the computation density ratios. However, for GEMMs, this is not always the case as some kernels have very low reuse factors or extremely narrow dimensions, e.g. GEMM-9, 10, 12, 16, 18. In contrast, due to the very large matrix multiplications in RNNs, there are some cases (e.g. RNN-1, 3, 6, 7) in which MLBlocks surpass the nominal speedup rates. This is because of their larger memories to save and reuse weights, leading to a higher utilization rate while having more MAC units.

- In Table2, why the 4/5-configs do not report area/obj. values ? If it is due to excessive time of the experiments, how you can obtain results for those cases?

  Yes, you are correct. In the case of 4 or more configurations, the number of iterations increases rapidly. Thus, synthesizing all possible cases requires excessive time. However, calculating the utilization rate is relatively easy.

  So while we have only reported the utilization rate for $N \geq 4$, it is not significantly improved by adding more configurations. Thus, in practice, we suggest up to 3-Config. Using the Greedy approach, a reasonably good solution in a couple of minutes is achievable, which we used this approach to generate MLBlock instances. We clarified this points by revising Section 5.1.

- Can you also indicate the time step (e.g. t, t+1, etc.) in figures 4 and 5?

  We note that the reviewer was confused by our previous figures 4 and 5. The main intention is to show how well a given n-dimension algorithm can be tiled using a specific tile unit, where this specific tile unit is an unrolling instance. We did not intend to depict a specific implementation such as a pipeline. In the revised manuscript, we removed the arrows and hope this clarifies the explanation.

- How can an MLBlock for low precision can be compared with an identical implementation in the selected Xilinx device using LUTs and FFs (and not DSP48s) ? Xilinx default synthesis strategy will select LUTs/FFs to implement low-precision arithmetic functions, since for those functions the DSP48s might not provide better compute density, i.e. a metric that the authors are using to quantify their results. Is it possible to provide some representative comparison for GEMM/LSTM/CNN 8/6 ?

  To provide better insight, we synthesized our DSP48E2 and MLBlock models as FPGA overlays. We reported post-implementation resource utilization and timing results in the newly added Section 5.4. This shows how many CLB and Registers are encapsulated by each model. Please also note, the mentioned models are not optimized for LUT-based implementations, and the calculated performance per CLB could improve for all cases.

- When the bit-serial multiplier is selected, what is the impact in the latency ? (considering that some cycles are needed to fill the pipeline of those units). Again, can the authors provide an example with latency cycles?

  Using the suggested serial multiplier, each MAC unit can do 8×8, 8×16, 8×16, or 16×16 multiplication accompanied by 32-bit accumulation in 1, 2, 2, or 4 cycles respectively. This means the MLBlock's computation pipeline delivers new output in 1, 2, 2, or 4 cycles. Each MAC unit has a sequencer that orchestras this process. On the other hand, as using serial multipliers requires loading more weights (both upper and lower parts of the 16bit weights), it also increases the loading latency by a factor of 2. We added the abovementioned explanations to the manuscript in Section 4.1.

Minor comments / typos:

- Abstract, last sentence, for 16-bit only the 2x higher performance is mentioned, but not gains/loss in delay?

  Thanks for your suggestion, we elaborate on that.

- p3, 2nd par., 1st line, please add reference for block floating point.

  Yes, you are right. We cited [3] as it is one of the earliest and solid works on using block floating-point arithmetic in CNNs.

- p3, 2nd par., last line. Can the sentence of ”it is challenging to do so in a systemic fashion” be elaborated or cited?

  We elaborated on the sentence by denoting the absence of two essential modelling components 1) a generalized problem formulation accompanied with 2) an efficient mapping to configurable architectures in the context of EB architecture exploration.

- p17, Section 4.2, par.4, 1st line, ”assume the a MAC” , erase the.
- p21, Sec. 5.4, par.2, 2nd line, ”nested loops according Algorithm 1”, add ”to” after ”according”
- p22, line 7, ”separate logical EBs”, delete the ”s” (refers to projection, not plural)

  Thanks for your careful reading, we addressed these three typos.

References

[1] D. T. Kwadjo, J. M. Mbongue, and C. Bobda, “Exploring a layer-based pre-implemented flow for mapping cnn on fpga,” in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2021, pp. 116–123.

[2] Baidu, “DeepBench,” 2016, https://github.com/baidu-research/DeepBench.

[3] Z. Song, Z. Liu, and D. Wang, “Computation error analysis of block floating point arithmetic oriented convolution neural network accelerator design,” in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, S. A. McIlraith and K. Q. Weinberger, Eds. AAAI Press, 2018, pp. 816–823. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16057

[4] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Quantized neural networks: Training neural networks with low precision weights and activations,” *J. Mach. Learn. Res.*, vol. 18, no. 1, p. 6869–6898, Jan. 2017.

[5] G. Yang, T. Zhang, P. Kirichenko, J. Bai, A. G. Wilson, and C. D. Sa, “SWALP : Stochastic weight averaging in low precision training,” in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 2019, pp. 7015–7024. [Online]. Available: http://proceedings.mlr.press/v97/yang19d.html

[6] S. Rasoulinezhad, H. Zhou, L. Wang, and P. H. Leong, “Pir-dsp: An fpga dsp block architecture for multi-precision deep neural networks,” in *27th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2019, San Diego, CA, USA, April 28 – May 1 2019*, 2019.