Rebuttal of pap141s2

Thanks for your careful and valuable comments. We find some common concerns and we will explain them in the beginning, and then explain other concerns point by point.

Common concerns

1. *Most of reviewers are concerned about the generality/portability of the parallel strategy mentioned in the paper.*

   We are sorry about that the writing structure of the paper makes such a misunderstanding. The UNAT (Unstructured Acceleration Toolkit) can be separated into two parts: **framework** (*Section 2: Design and API*) and **kernel** (*Section 3: Parallelization Strategy*). The **framework** is in charge of the management of Domain Specific Language (DSL), construction of topology and packaging of data arrays, and the **kernel** is responsible for the parallel implementation of specific hardware or architecture. Therefore, the **framework** has compatibility across many-cores/multi-cores architectures, while the parallel implementation has to be specialized and tuned for the specific architecture. As mentioned in the paper, OP2 is a comparable counterpart developed in Oxford University, while it also adopts different parallel strategies for Intel CPUs and NVIDIA GPUs. Due to time constraints, only the parallel implementation on SW26010 is performed in this paper.

2. *Some reviewers are concerned about the significance of this work because of the better choice of hand-optimized performance or the combination of many but not novel techniques.*

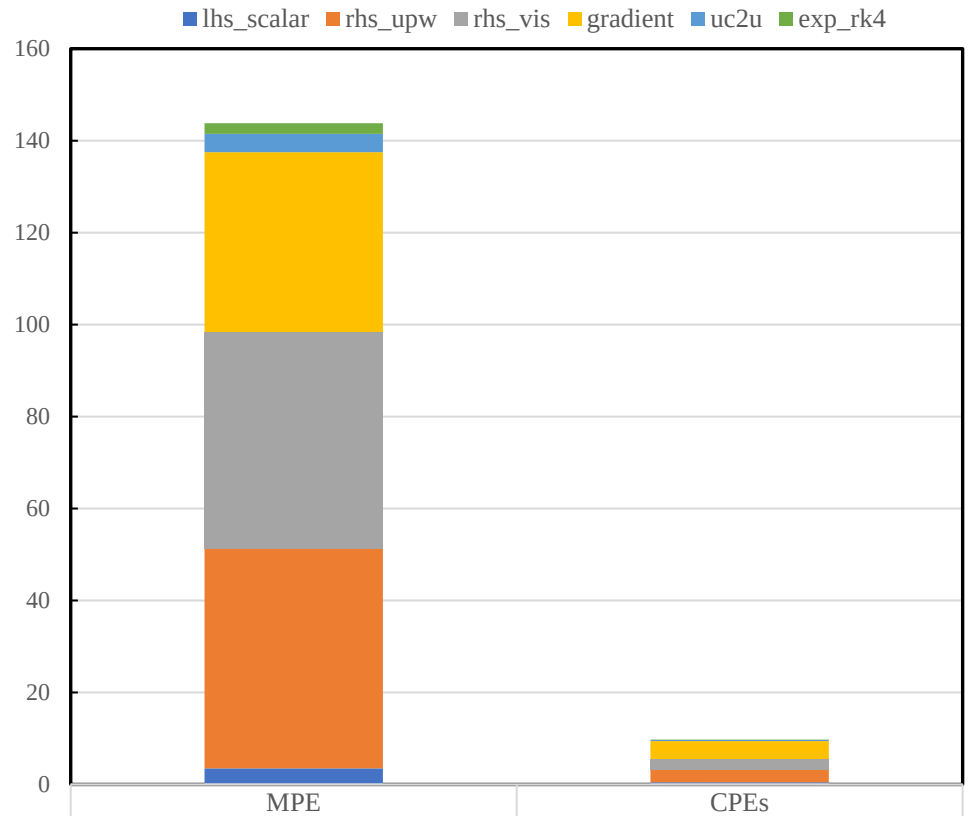
   Firstly, we acknowledge that the performance of our parallel implementation is a little worse than hand-optimized codes. But considering for its generality, the performance is considerable. For the compatibility across different computing kernels, we have to give up some common but useful techniques such as vectorization, instruction optimization, et al. But we actually adopt some novel techniques such as the Register-Level Communication (RLC), which is deleted because of the limitation of paper length. In this rebuttal, we add it to the end of the rebuttal for reference.

   The significance of this work is the auto parallelization for engineering computations. Engineers write their serial codes as usual and pack them with our implemented DSL, then they can obtain nearly hand-optimized performance. In spite of SpMV, we have other test cases to prove the performance of UNAT. We adopt UNAT to accelerate a well-tuned in-house cell-centered flow equation solver, the original time-consuming kernels are shown in the following table:

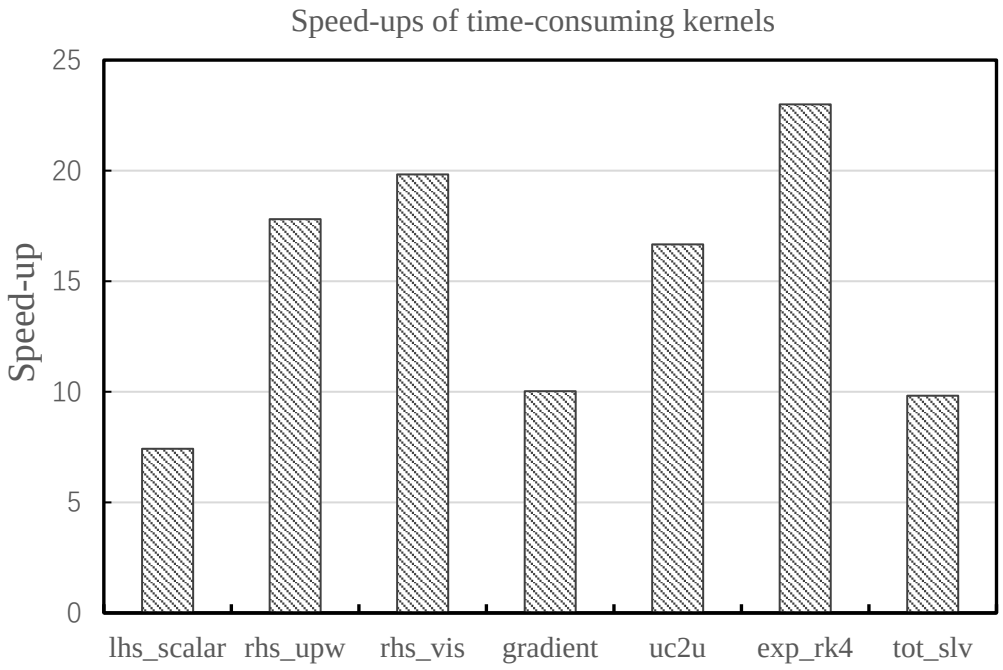
kernels	lhs_scalar	rhs_upw	rhs_vis	gradient	uc2u	exp_rk4	tot_slv
time	3.49	47.9	47	39.1	4	2.3	144.4
ratio	2.4%	33.2%	32.5%	27.1%	2.8%	1.6%	100%

   There are 6 main kernels in this solver, and the acceleration work was performed **within one week** by the Row-Subsection (RSS) strategy in UNAT. The following figure illustrates the comparison of time-consuming kernels before and after acceleration, running on Management Processing Element (MPE) and Computing Processing Elements (CPEs) separately.

Comparison of time-consuming kernels running on MPE and CPEs



	MPE	CPEs
exp_rk4	2.3	0.1
uc2u	4	0.24
gradient	39.1	3.9
rhs_vis	47	2.37
rhs_upw	47.9	2.69
lhs_scalar	3.49	0.47


Speed-ups of time-consuming kernels
Speed-up
25
20
15
10
5
0
lhs_scalar
rhs_upw
rhs_vis
gradient
uc2u
exp_rk4
tot_slv

Response for Reviewer #1:

Q1: *While the paper itself manifests an effort of intellectual merit, the significance of its achievements can be put into question.*

A1: As we mentioned in ***Common concerns #1***, we sacrifice some performance for the generality, and the scope of UNAT is unstructured mesh computation, and is not limited to SpMV.

Q2: *the manuscript was let down by a considerable deficiency in the quality of writing.*

A2: We are apologized for the grammatical and typing mistakes, and we will examine the paper carefully to check the mistakes and missing references.

Response for Reviewer #2:

Q1: *although I can easily follow the paper, it's unclear to me how this work can advance the state of the art*

A1: This question can refer to ***Common concerns #2***. Although what we adopted are some common techniques other than Register-Level Communication (RLC), the application and combination of these algorithms to a specific architecture are still an interesting issue. There are many challenges for SW26010 such as the extremely limited bandwidth and the absent of L2 cache. So we developed a data-reuse mechanism and RLC pattern to alleviate the memory bandwidth bottleneck.

Response for Reviewer #3:

Q1: *The paper includes a description of the reasoning for UNAT, and the design, but does not include enough information to understand how some of the examples are supposed to be transformed into a functional operator on a target.*

A1: Thanks for your careful review and you are correct, the function operators are indeed a macro and we use *define_ops(func_name)* to hide the function name and formal parameter. We apologize for the omissions of the statement and will add it to the revised paper.

Q2: *given that the baseline is a single-core version, it's somewhat unclear how well the parallelization is actually working here in comparison to another method.*

A2: The baseline is a single-core version, and we have two parallel strategies in UNAT. The two strategies have their own features and application scenarios, so the comparison of performance between them is meaningless. For the specific strategy, we compared the performance under different matrix formats (LDU and COO) and data layouts (AoS and SoA). In addition, we proposed a method to analyze the utilization ratio of SW26010 as shown in Figure 18.

Q3: *the paper seems like a reasonable start toward a Programming Models track paper, but an inappropriate fit for the Experiments track.*

A3: Well, actually this paper was contributed to the Algorithms track and fit for one of the topic areas: *domain-specific parallel and distributed algorithms*. We have no idea why this paper was contributed to the Experiment track. However, UNAT makes some innovations in support of large-scale infrastructures and facilities, which also fit for one of the topic areas of Experiment track.

Q4: *Comparing with a serial version might be reasonable when it comes to a complex system that had only one implementation at the start, but for SPMV there are enough implementations around that comparing with even a basic OpenMP/OpenACC version would have been trivial, and it makes one wonder as a reviewer if there's a reason that comparison wasn't made.*

A4: The comparing of complex system can refer to ***Common concerns #2.*** As for the comparison of SPMV, we kindly accept your advice and will add it to the revised paper. When we wrote the performance section, we thought the comparison with the experiments on the same platform may be more meaningful, because it can explain that we can achieve nearly hand-optimized performance with UNAT.

Response for Reviewer #4:

Q1: *My largest concern is that the presented experiment is not reproducible for the reader.*

A1: The source code can be founded on github: *https://github.com/lhb8125/UNAT*. Actually this paper was contributed to the Algorithms track and we have no idea why this paper was contributed to the Experiment track. However, UNAT makes some innovations in support of large-scale infrastructures and facilities, which also fit for one of the topic areas of Experiment track.

Q2: *A second concern is the tranferability of the presented approach to other platforms.*

A2: This question can refer to ***Common concerns #1.*** The framework of the toolkit can be ported from one platform to another platform, but the parallel strategies must be different and have to be tuned for different architectures.

Q3: *how large was the deviation between multiple runs.*

A3: Thanks for your careful review and advice. We will add it to the revised paper.

Q4: *The comparison of UNAT to hand-optimized code on the basis of the achieved FLOPs per second can be misleading.*

A4: The comparison of absolute runtime is fair and visualized, but the absolute runtime cannot be found in the literature. The amount of FLOPs of SPMV is calculated by the following equation, and we will explain that in the revised paper.

$$\text{FLOPs} = \frac{N_{edge} * 2 + N_{node}}{T}$$

where N_{edge} and N_{node} indicate the count of edges and nodes respectively and T indicates the absolute runtime.

Q5: *the paper is unfortunately very rough to read due to the authors obviously writing in a language foreign to them.*

A5: Thanks for your careful correctness. We will re-check the paper and use an internal proof-reading process to improve the quality of our scientific work.

Q6: *Why did you not use the same test cases for both iterators?.*

A6: The start point of the design is to show the test cases as many as possible, not limited to OpenFOAM or Goodwin. In addition, the two strategies have their own features and the comparison between them is not the key point what we focus. Therefore, we adopt two different cases for them.

Appendix

On-chip Point-to-Point Transfer

The SW26010 processor provides 8GB global memory for each CG, which is shared by the CPEs of same CG through DMA. However, memory bandwidth between global memory and the LDM is relatively low and varies with the size of continuous memory access block of CPE. We prepare a mini program to measure the effective DMA bandwidth under various sizes of data block and the results are presented in Table 1. The effective DMA bandwidth varies from 1 GB/s to 31 GB/s. While in unstructured finite volume scheme, there are abundant irregular and discrete memory access, which means the memory bandwidth is frequently around 1 GB/s.

Table 1: The DMA bandwidth between global memory and LDM

Size(Byte)	DMA Bandwidth(GB/s)
8	0.97
32	4.82
64	12.84
128	18.46
512	23.27
1024	28.71
2048	29.53
4096	31.56

However, an advance feature of SW26010 architecture improving this situation is register communication technique. The CPEs of one CG are designed as 8*8 array and can transfer data with others on the same row/column. The access latency comparison between register communication and discrete memory access instructions *gload*, *gstore* is shown in Table 2**错误!未找到引用源。**. But in spite of the low latency, the limitation of this technique is also evident. The irregularity and discreteness of memory access determined that the source and destination in register communication is uncertain and the CPEs may transfer data to any cores. Therefore, we develop a novel Register-Level Communication (RLC) mechanism to support transforming data to all other cores. We can perform the data transfer over the CPEs domain rather than point-to-point communication through RLC.

Table 2: The latency of different memory access pattern

Source	Mode	Cycles
global memory	gload	177
	gload&gstore	278
register	point-to-point	10
	BCast (row)	14
	BCast (column)	14

In our RLC implementation, the prepared data is packed with headers containing *src_id*, *dst_id* and the number of effective data. To avoid the congestion, the communication over the domain is splited into four directions: Top-Left to Down-Right, Top-right to Down-Left, Down-Right to Top-Left and Down-Left to Top-Right. Take the Top-Left to Down-Right direction as example, the data flow is illustrated in Figure 1.

For the specific CPE **P**, firstly it receives the data from CPEs on the same column by *GetC()*, and then transmits the data from cores on the same row to cores on the same column. Finally, it sends its own data to other cores on the same row. It is clear that the receptive field and influence field of CPE **P** is Top-Left cores and Down-Right cores respectively as the coloured cells in Figure 1. The cells with slash

and the blanked cells indicates the transmitting field and irrelevant field respectively. Hence we can obtain the fields for CPE **P** in four directions as shown in Table 3. where i_p and j_p indicates row index and column index of CPE **P**. The ideal degree of parallelism is 8 and the measured bandwidth is 7.02 GB/s.

Table 3: The influence and receptive fields for CPE P in four register communication directions

Direction	Receptive field	Influence field
Top-Left--Down-Right	$i < i_p, j \leq j_p$	$i > i_p, j \geq j_p$
Top-right--Down-Left	$i \leq i_p, j > j_p$	$i \geq i_p, j < j_p$
Down-Right--Top-Left	$i > i_p, j \geq j_p$	$i < i_p, j \leq j_p$
Down-Left--Top-Right	$i \geq i_p, j < j_p$	$i \leq i_p, j > j_p$

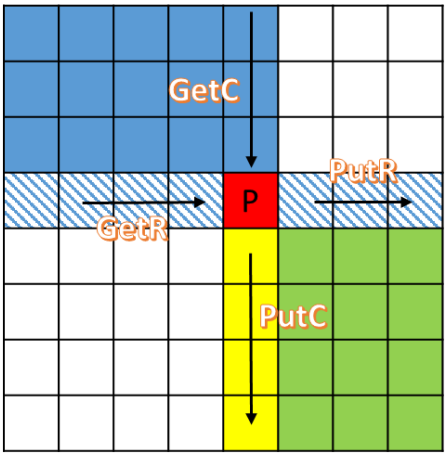


Figure 1: The pattern of Register-Level-Communication