Proceedings of Global Power and Propulsion Society
ISSN-Nr: 2504-4400
Beijing Conference 2019
16th – 18th September, 2019
www.gpps.global

GPPS-BJ-2019-0148

Towards a Peta-scale Unstructured Computational Fluid Dynamics (CFD) Acceleration Toolkit based on Sunway architectures?

Hongbin Liu
National Supercomputing Center in Wuxi, China
lhb8125@gmail.com
Wuxi, China

Hu Ren
National Supercomputing Center in Wuxi, China
renhu@mail.nsccwx.cn
Wuxi, China

Chen Wang
AECC Comercial Aircraft Engine Co., Ltd
wangchen@acae.com.cn
Shanghai, China

Hanfeng Gu
National Supercomputing Center in Wuxi, China
hanfenggu@gmail.com
Wuxi, China

Fei Gao
National Supercomputing Center in Wuxi, China
gaofei@mail.nsccwx.cn
Wuxi, China

Guangwen Yang
Department of Computer Science and Technology, Tsinghua University, China
ygw@tsinghua.edu.cn
Wuxi, China

ABSTRACT

Adopting Large Eddy Simulation (LES) to simulate the complex flow is appropriate to overcome the limitation of current Reynolds-Averaged Navier-Stokes (RANS) modeling and it provides a deeper understanding of the complicated transitional and turbulent flow mechanism. However, the large computational cost limits its application in high Reynolds number flow. Though Sunway TaihuLight supercomputer has revealed its remarkable performance on atmospheric dynamics and earthquake simulation, there are still some unsolved problems for researchers: unstructured mesh and complicated programming pattern on SW26010. Therefore, we develop an ultra-scalable Unstructured Acceleration Toolkit (UNAT). In this toolkit, we derive a graph decomposition model of computing flow and reuse data through L1 cache to optimize the operator. Multi-level block method and register communication are developed to achieve the best performance of CPEs. CFD engineer can ignore the detail of parallel programming on SW26010 and utilize the operator interfaces offered by UNAT. Finally, we deploy UNAT to accelerate a combustion code and obtain 12-19x performance speed-up on specific operators, onto 65 cores within the chip compared with MPE.

INTRODUCTION

Computational Fluid Dynamics (CFD), which is considered as a feasible alternative, has become increasingly important for the development of next generation combustor. Owing to the development of CFD, manufactures have greatly reduced the number of real engine tests and design iterations. In the current design chain, CFD in gas turbine industry are mainly based on the Reynolds Averaged Navier-Stokes approaches, which model the entire turbulence and only resolve mean flow structures. The main defect of RANS Methods is the determination of model coefficients when new configurations are studied. The complexity of flows in modern gas turbines adds multiple constraints on RANS and limits their precision (Wang et al, 2017; Li et al, 2019). Turbulence resolved methods, including Large Eddy Simulation (LES), Direct Numerical Simulation (DNS) show more advantages in solving these unsteady flows. As the cost, those eddy-resolved methods demand high performance computation systems. With DNS all scales must be resolved and computing costs grow with the flow Reynolds number at about $O(Re^{9/4})$ and the Damkohler numbers at about *O(Da)* (Gicquel et al, 2012). The huge computational demands make the reactive flows in the realistic aero-engine combustor remain out of reach. LES which filters out all flow small scales put much less stringent limits for the computational size. In contrast to DNS, LES allow simulating turbulent flows with turbulent Reynolds numbers approximately 500 times larger with the same number of grid points (Gicquel et al, 2012).

With the development of Many Integrated-Chip, the computational capacity of supercomputer reaches a unprecedented level. The Sunway Taihulight is a Chinese home-grown supercomputing system with 40960 compute nodes, providing a peak performance of 125Pflops and a sustained Linpack performance of 93PFlops, and ever ranked No.1 on the Top-500 supercomputers in the past two years. There are abundant studies on this platform containing atmospheric dynamics, earthquake simulation, deep learning and molecular dynamics and two of them won the ACM Gorden Bell prizes in 2016 and 2017(Yang, 2016; Fu et al, 2017; Fang et al, 2017; Li et al, 2018; Duan, 2018).

Though some researchers have performed relevant CFD simulation on heterogeneous system (Vincent, 2016; Liu 2018), there are still some difficulties for the CFD code on Sunway platform: unstructured mesh and implicit method. As is well known, the irregular and discrete memory access introduced in unstructured mesh will extremely limit the performance of heterogeneous supercomputers. On the other hand, implicit solvers, with more challenging patterns in both compute and memory access, involve more difficulties to achieve a balanced design that can efficiently use the many-core architectures. Therefore, the CFD code need to be tuned carefully to run efficiently on Sunway platform, which is unfamiliar and unnecessary for CFD engineers and researchers.

To build the bridge between CFD codes and Sunway platform, we develop an ultra-scalable and auto-parallel Unstructured Acceleration Toolkit (UNAT). In UNAT, we encapsulate the detail of parallel program on SW26010 and expose the operator interface to CFD programmer. So CFD engineers can program their codes serially with parallel idea. To address the irregular memory access in unstructured mesh, multi-level block method and register communication are developed to achieve the best performance of SW26010.

UNAT is adopted to accelerate the combustion code of Shanghai AECC Commercial Aircraft Engine Co. Ltd. The computing flow is decomposed and then merged to coupled operators for data reuse, which results in 12-21x and 5x performance speedup separately on coupled operators and overall code, onto 65 cores within the chip compared with one master core.

METHODOLOGY

The SW26010 Processor

The SW26010 processor is designed by Shanghai High Performance IC Design Center. It is a many-core processor with 260 heterogeneous cores providing a peak performance of 3.06 TFlops and a performance-to-power ratio of 10 GFlops/Watt. As Figure 1 shows, the 260 cores are divided into four Core Groups (CGs). Each CG is composed of one management processing element (MPE), one intelligent memory processing element (IMPE) and 64 computing processing elements (CPEs) organized as 8 by 8 mesh. MPEs, IMPEs and CPEs are designed for different goals. The IMPE has single instruction fetch unit and multiple instruction buffer, mainly targeting at memory access operations. The MPE is a complete 64-bit RISC core with a frequency of 1.45 GHz, mainly targeting to handle the flow control of a program, I/O and communication functions. While CPEs adopt a more simplified microarchitecture to maximize the aggregated computing power. Each CPE has 16 KB L1 instruction cache and 64 KB local data memory (LDM) which can be configured as user-controlled fast buffer. A performance model based on the three-level (REG-LDM-MEM) memory hierarchy was proposed(Fang et al, 2017). The CPE can either directly access the global memory with a limited bandwidth of 8GB/s, or through a REG-LDM-MEM memory hierarchy to obtain much higher bandwidth. Each CPE has two pipelines to process instruction decoding and execution. A carefully orchestrated instruction reordering scheme can alleviate the dependences of instruction sequence, thus potentially improve the instruction execution efficiency. Inside each CPE cluster, a mesh network with 8 row communication buses and 8 column communication buses enables fast data communications and sharing at the level of registers, which allows efficient data reuse and communication among CPEs.

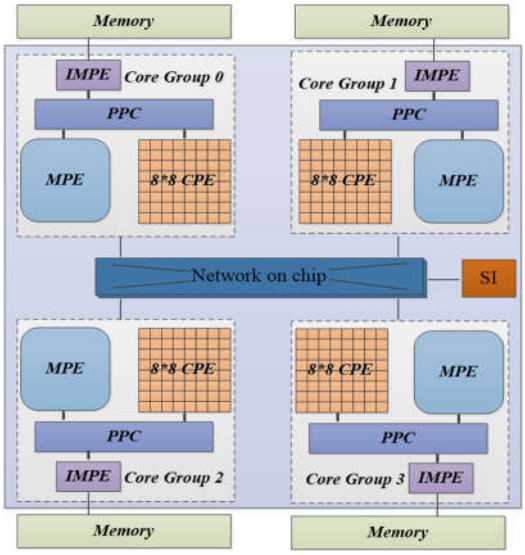

Figure 1: The architecture of SW26010

Performance Challenges for UNAT

The computation in CFD is substantially Finite Volume Method (FVM) computation based on unstructured mesh. Different from stencils on structured mesh, unstructured FVM discretization data structure is commonly described as an adjacency graph, with control volume cells as the vertices and faces between cells as the edges. Generally, the topology of the FVM graph can be represented as a sparse matrix, in different formats such as Compressed Coordinate (COO), Compressed Sparse Row (CSR), etc. While in this code, a more index-saving matrix format LDU is adopted, with half coordinates of COO to store a topology-symmetry matrix. LDU, as itself implies, stores coefficients of the sparse matrix separately in three arrays by upper triangle, lower triangle and diagonal. While CSR allows faster access to coefficients, LDU provides more freedom in coefficients ordering that is essential to the following work of data structure reformation. Generally, this kind of computation set up two main performance challenges on advanced HPC platform including TaihuLight: low arithmetic density and irregular memory access.

In industrial codes, numerical schemes with orders no more than 2 are most frequently used, which is more stable and reliable in complex-geometry combustor simulation. on the other hand, 2 order and lower schemes mean lower arithmetic density. When computing architecture evolution tends to achieve higher FLOPS to memory bandwidth, and this kind of simulation is becoming strong memory bandwidth limited.

According to the computation nature of unstructured FVM discretization, the data access to memory has to be irregular. As measured in real case, the topology matrix of a problem with 120K cells, the bandwidth (the largest distance of the coefficients from the diagonal) with minimizing reordering reach up to 7K. This value will become 100K in millions mesh size, or even larger when the geometry is very complex and the topology of cell connectivity is more irregular. Irregular memory access will increase cache miss and invalid memory bandwidth consuming. Problems will be alleviated on platform with large cache size, say 10 MB 3rd level cache on Intel multi-core processor, or on platform with large memory band, say 480GB/s of NVIDIA Tesla K80. While for SW26010 processor, the memory bandwidth is less than half of K80 in spite of higher FLOPS than it and only a single level cache, 64KB LDM per CPE is available. It is a heavy and challenging task to transplant and accelerate this code on the target platform.

In UNAT, we propose several solutions to address the above problem, which will be introduced in the next sections:

- Multi-level block reordering;
- On-chip point-to-point transfer;
- MPE-CPEs asynchronous parallel algorithm;
- Loop decomposition and fusion.

Multi-level Block Reordering

To solve the irregular memory access problem in unstructured mesh calculation, a multilevel block (MLB) data structure is proposed. Considering the configuration, the connected system can be seemed as three level: the first level connectivity is the network connecting CGs that transfer data with MPI; the second level implicitly hides in the mesh data segmentation caused by limited 64 B x 64 B LDM size in a single CG; the third level is the CPEs inside a CG that connected by register level data path. Correspondingly, the decomposition of the mesh is also applied in three levels.

As Figure 2 shows, MLB format has a no-uniform multi-block structure based on a LDU matrix. The coarsest blocks represent MPI level mesh decomposing. The coefficients on coarse diagonal block represent the internal faces of the mesh cells assigned to a process running on a single CG, and the off-diagonal coarse block is the MPI interface of cells that connected but distributed on different processes. The medium blocks represent the segmentation of mesh corresponding to the total cache size of a CG, which makes it possible that the relevant data in unstructured mesh calculation is lumped together. Finally, the finest level decomposing parts is confined to 64, corresponding to the amount of CPEs. The mesh data on diagonal block is "local" for the cells assigned to a single CPE and loaded into LDM by Direct Memory Access (DMA) directly, while the off-diagonal block contains mesh data to be transferred by on-chip register level communication. Because of the capability of random access provided by coordinate-LDU matrix, the faces, i.e. coefficients, can be reordered and grouped with the same roles together. As shown in Figure 2, the blue filling curve represents the sequence of faces in different blocks. Generally, the priority rule is that the faces located in finer level block is prior to coarser ones, and in the same level, row block is prior to column block. Inside the leaf block, face sequence is arbitrary. With MLB format, unstructured data is now "block-structured" and irregular data can be accessed continuously in a leaf block. Taking advantage of Metis, most faces or coefficients will be gathered in diagonal blocks in each level, and interfaces with "communication" will be minimized to a certain extent

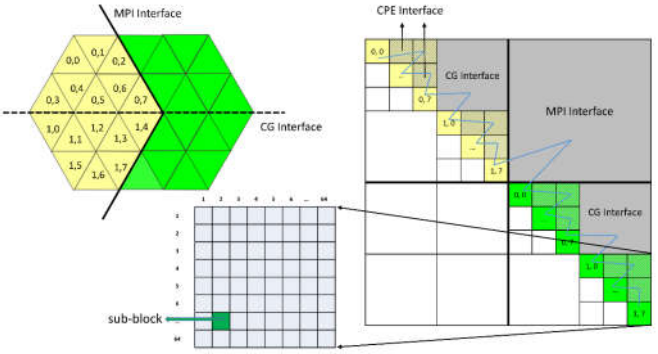

Figure 2: Multi-level Block of unstructured mesh. The unstructured mesh is divided and assigned to two process, colored with yellow and green respectively. Three levels are connected by three interface: MPI Interface, CG Interface and CPE Interface.

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.

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. 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 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

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 3.

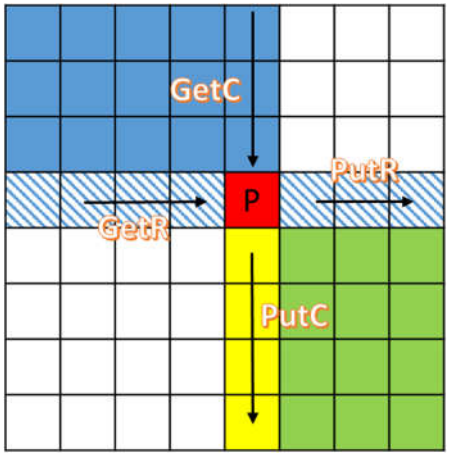

Figure 3: The pattern of Register-Level-Communication.

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 colored cells in Figure 3. 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	$i \geq i_p, j < j_p$	$i \leq i_p, j > j_p$

MPE-CPEs Asynchronous Parallel Algorithm

As mentioned above, the computational operation capability in finite volume is limited extremely by the indirect and irregular addressing. Therefore, we design a *MPE-CPEs asynchronous parallel algorithm with MLB*. The partition of the computational domain is illustrated in Figure 2, and one certain CG contains two levels. At the second level all row blocks are marked as diagonal blocks, colored cells, and off-diagonal blocks, gray cells marked as CG Interface. They are assigned to CPEs and MPE respectively with respect to the data density.

To address the read-write conflict problem between MPE and CPEs, we present an asynchronous parallel algorithm as shown in Table 4. Firstly, we describe the notations used in the algorithm. **row_id** and **row_id_m** denote the current index of row block for CPEs and MPE respectively, **levels** denotes the second decomposition level with MLB. The computation of diagonal block will be discussed later on, and for the **row_id**th row block, the computation of off-diagonal blocks in the (**row_id+1**)th row block are assigned to MPE. Specifically, we get the index of first edge, **edge_start_id**, and last edge, **edge_end_id**, in the current row block, and then traverse all edges to perform the computation.

Table 4: MPE-CPE asynchronous parallelism.

```
for row_id = 1->levels[0] do
    /* diagonal block */
    block(rowId, rowId) –> CPEs
    /* off-diagonal block */
    row_id_m = rowId+1
    if(row_id_m < levels)
        edge_start_id = block_start[row_id_m]
        edge_end_id = block_end[row_id_m]
        for iedge=edge_start_id->edge_end_id do
            /* computational operator */
        end for
    end if
end for
```

As for the diagonal block in CPEs, one CPE is in charge for the computation of one sub-block row. To explain the computational pattern of CPEs, we take the Sparse Matrix-vector Multiply (SpMV) as example. The coefficient matrix A is decomposed into three arrays, **lower**, **upper** and **diag** in LDU format, storing the lower triangle data, upper triangle data and diagonal data respectively. These data can be transfer into LDM completely through DMA because they are sparse. However, this pattern does not appropriate to **x** and **b** because they are dense. Figure 4 shows the computation process in CPEs and the amount of CPEs is cut down to 4 for the conciseness. In our implementation, the sub-block row is also divided into two parts: diagonal sub-block and off-diagonal

sub-block. Firstly, we fetch **x** and **b**, namely VertexData in Figure 4, in the diagonal sub-block through DMA. The

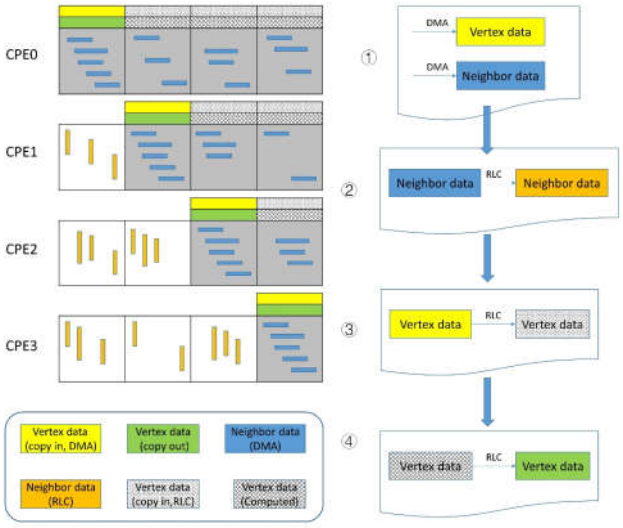

Figure 4: The computation process in CPEs: ① Fetch Vertex data and Neighbor data through DMA; ②Get the Neighbor data in the lower triangle through RLC; ③Get Vertex data (copy in) in the off-diagonal sub-block through RLC; ④Send Vertex data (copy out) in the off-diagonal sub-block to other CPEs through RLC.

column index of all edges in the sub-row block is also fetched through DMA, namely NeighborData in Figure 4(①). Then we transfer the Neighbor Data from the upper triangle edges to the lower triangle edges through Register-Level Communication (RLC), which is essential for the next step (②).

Table 5: Register-Level-Communication (VertexData)

```
#define CPE_NUMS 64
for i = 1->CPE_NUMS do
      bias[i] = 0
end for
for ipcg = 1->total_send_pcg do
      if (MYID> sPacks[ipcg].dst_id)
            edge_id = edge_start[sPacks[ipcg].dst_id]
                  + bias[sPacks[ipcg[.dst_id]
            sPacks[ipcg].data <- x[Neighbor[edge_id]]
            bias[sPacks[ipcg[.dst_id]]++
      end if
end for
reg_transfer_data()
for ipcg = 1->total_recv_pcg do
      if (MYID< rPacks[ipcg].src_id)
            x[length] <- rPacks[ipcg].data
            length++
      end if
end for
```

For the off-diagonal sub-block, though **x** cannot be loaded directly through DMA, they are stored in the LDM of other CPEs. Hence **x** can be padded completely through RLC, while this part is stored sparsely as well as edge data (③). The communication pattern is shown in Table 5. Here **CPE_NUMS** denotes the amount of CPEs in one CG and is defined as a constant. **total_send _pcg** and **total_recv_pcg** denote the count of send-packages and receive-packages for each CPE in RLC. **sPacks** and **rPacks** belong to **Packs** struct storing the package information in RLC including **src_id**, **dst_id** and **data**. Firstly, for every send-package, we can get the index of edge through array **bias** and **edge_start**, storing the starting index of edge of sub-blocks. Here the column index of edges, **Neighbor** is transferred from other CPEs in the last RLC (②). Apparently the **x** transferred through RLC is sparse and has the same length as edges in the off-diagonal sub-block. Function reg_transfer_data() performs the global register communication according to the struct **sPacks** and **rPacks**. In the last section, the existing data is padded with the **x** through RLC and then composes the completed **x**.

Now that the coefficient matrix **upper**, **lower**, **diag** and vector **x** is loaded into LDM, we can perform SpMV operation in the CPEs and get the output vector **b**. The storage pattern of b is identical to x. To avoid the write conflict between CPEs, b need to be transferred to the corresponding CPEs through RLC (④). This RLC algorithm is similar to x but the inverse direction. Finally, the output vector is transferred to MPE through DMA.

As mentioned above, the MLB decomposition **level** is estimated according to the LDM size by the next equation:

$$\text{levels} = \frac{(L_e \cdot C_e + L_v \cdot C_v) * sizeof(scalar) + 2 * sizeof(label)}{SPE_{NUM} * size_{ldm}}$$

Loop Decomposition and Fusion

In the combustion code, Flux calculation consumes most of the running time apart from linear solver. A calculation tree, as Figure 5 shows, can be used to represent the calculation

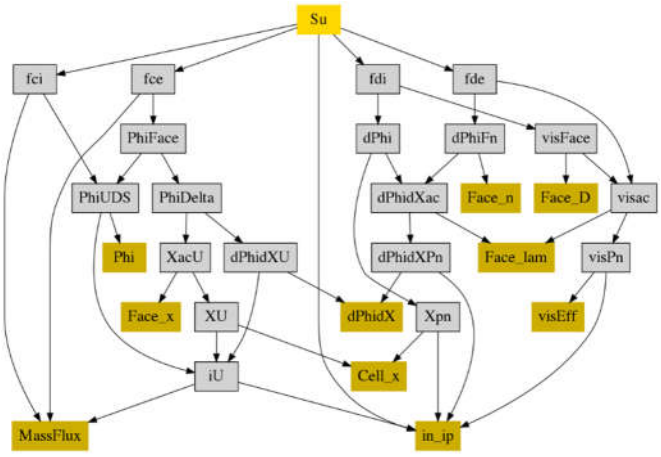

Figure 5: Flux loop decomposition and fusion.

procedure of the flux source when discretized the PDEs. Obviously, loop fusion is beneficial to entire calculation process by minimizing loop numbers and reducing intermediate variables. On the other hand, fusion too many variables together will cause heavy burden on LDM space, which calls for smaller decomposition of MLB leaf block. As

a result, the ratio of the data to be transferred via register communication and the CG interface needed to be calculate by MPE will increase.

To settle the trade-off, two fusion strategy are experimented. The first strategy partitions the calculation tree with certain number of variables and orders them in a sequence making sure the most reusable of data. The second strategy separates variables defined on faces and cell centres apart. The operation including these two kind of variables can be coupled, since most interpolation and integration operations have similar form and reusable data. Thus, interpolations and integrations can be simplified with least variables that allow the largest part decompose in MLB reordering, and then topology-free arrays operations will be merged entirely in a single operation. Experiments on 5-array and 4-array partition of the first strategy on a series of total mesh size on a single CG are performed, as well as the second separation strategy as shown in Table 6. The shortest run time is observed in 5-array and separation mode, while the mesh size varies from 10K to 1M cells. It can be concluded that, with comparable performance, the separation strategy has the more simplified algorithm and is easier to use, compared with the first one in this section. The performance will be discussed in the next chapter.

Table 6: Three different loop fusion strategy.

Strategy	Operator Number	Data size (DMA)	Reused data size	Total data
Merge5	17	19C+29F	1C+33F	20C+34F
Merge4	23	11C+47F	1C+28F	12C+55F
Separate	8	15C+38F	0C+14F	15C+42F

RESULTS AND DISCUSSION

Code Variation

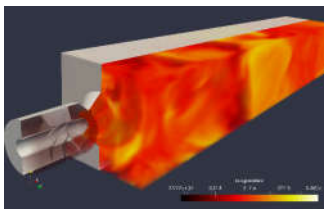

Figure 6: LDI flame with 6M mesh

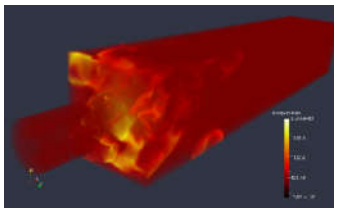

Figure 7: LDI flame with 50M mesh.

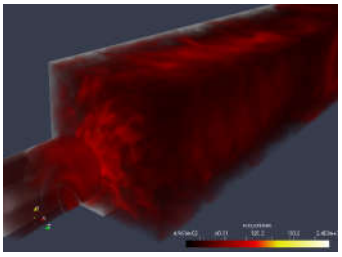

Figure 8: LDI flame with 400M mesh.

The transplanted and optimized code was validated to be correct and capable to simulate real combustion chamber in industrial scenario. The case used for variation is a standard Lean-Direct Injection (LDI) combustor, with spray, evaporation, ignition and other properties in real combustion turned on. Simulations on different mesh resolution from 6M, 50M to 400M, as shown in Figure 6-8. As one can observe, the flame structure becomes more complex and sharp while mesh size increases.

The parallel performance of the revised combustion code is illustrated in Figure 9. UNAT is adopted in general loop and linear algebraic solvers. The overall speed-up of combustion code is about 3 compared to the master.

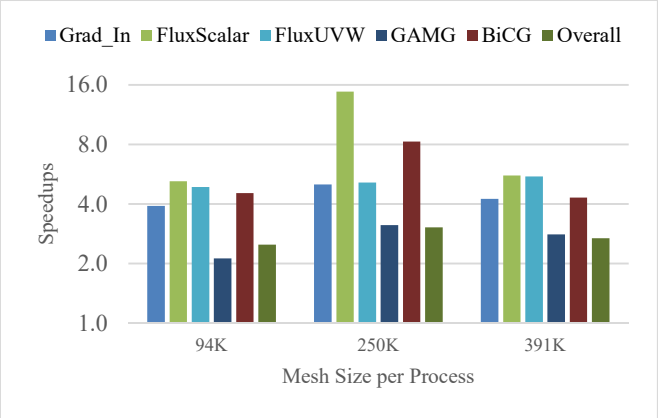

Figure 9: The speed-up of typical kernels and overall code

Performance of Specific Operator

In this section, Sparse Matrix-vector Multiply (SpMV) is adopted as the test operator because it plays an important role in linear algebra algorithm, and it is well known for the irregular computation and memory access pattern. We prepare several matrixes with different sizes as shown in Table 7. As shown in Figure 10, The speed-up increases with the mesh sizes and stabilises above 12.

Table 7: The count of cells and edges in different meshes.

Cells (K)	10	20	50	125	216	250	300
Edges(K)	19	49	139	367	637	735	884

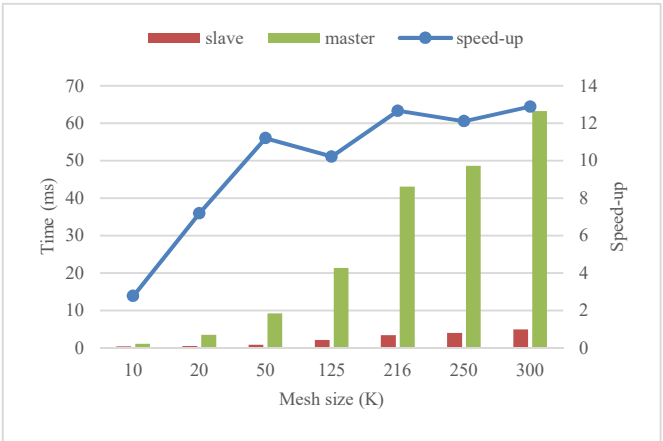

Figure 10: Running time and speed-up of UNAT on SpMV

Performance of Coupled Operator

To test the performance of loop fusion, we choose the three decomposition and fusion strategy listed in Table 6, and the test meshes is identical to the matrixes in Table 7. From Figure 11, "Separate" strategy has the least absolute running time in most meshes, while "Merge4" strategy has the highest speed-up because of the gain from the running time in MPE.

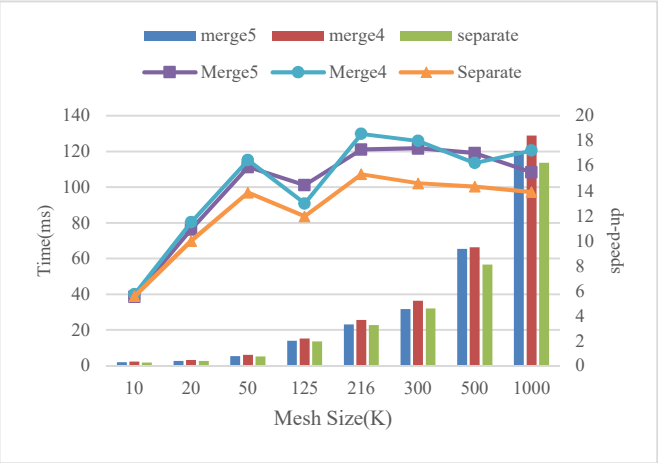

Figure 11: Running time (CPEs) and speed-up of the three decomposition strategy.

CONCLUSION

To address the memory access problem of unstructured mesh in parallel programming, we develop a toolkit, UNAT, to encapsulate the detail of parallel programming model and increase the user-friendliness. The main innovation we implemented are presented as follows:

1. We present the MLB reorder algorithm to minimize the influence of irregular memory access in unstructured grid. The basis for segmentation is determined by the LDM size to decrease the cache miss.
2. We designed an RLC pattern to perform the global communication between CPEs on chips. To avoid the congestion, the communication over the domain is splited into four directions. The ideal degree of parallelism is 8 and the measured bandwidth is 7.02 GB/s.
3. Inside a process, we design a *MPE-CPEs asynchronous parallel algorithm with MLB*. Diagonal blocks and off-diagonal blocks are assigned to CPEs and MPE separately with respect to the data density. For the diagonal blocks, RLC is adopted to reduce the data size loaded through DMA.
4. For the flux computation in combustion code, we decompose and fuse the computing flow to increase the reused data size. Three strategy are tested and separated strategy has the least running time mainly on CPEs, in which operator are divided into three parts: vector operations, interpolation and integration.

It is demonstrated that an established unstructured CFD code can be ported to modern high performance computing system at Peta-scale effortlessly and efficiently with UNAT. This imply the great potential of the future application for many established unstructured CFD codes existing in academic and industry.

References

Duan, X., Gao, P., Zhang, T., Zhang, M., Liu, W., Zhang, W., Xue, W., Fu, H., Gan, L., Chen, D. and Meng, X., 2018, November. Redesigning LAMMPS for peta-scale and hundred-billion-atom simulation on Sunway TaihuLight. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 148-159). IEEE.

Fang, J., Fu, H., Zhao, W., Chen, B., Zheng, W. and Yang, G., 2017, May. swdnn: A library for accelerating deep learning applications on sunway taihulight. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (pp. 615-624). IEEE.

Fu, H., He, C., Chen, B., Yin, Z., Zhang, Z., Zhang, W., Zhang, T., Xue, W., Liu, W., Yin, W. and Yang, G., 2017, November. 18.9-Pflops nonlinear earthquake simulation on Sunway TaihuLight: enabling depiction of 18-Hz and 8-meter scenarios. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (p. 2). ACM.

Gicquel, L.Y., Staffelbach, G. and Poinsot, T., 2012. Large eddy simulations of gaseous flames in gas turbine combustion chambers. *Progress in energy and combustion science*, *38*(6), pp.782-817.

Li, H. , Su, X. , & Yuan, X. . (2018). Entropy analysis of the flat tip leakage flow with delayed detached eddy simulation. Entropy, 21(1).

Li, L., Fang, J., Fu, H., Jiang, J., Zhao, W., He, C., You, X. and Yang, G., 2018, September. swCaffe: A Parallel Framework for Accelerating Deep Learning Applications on Sunway TaihuLight. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)* (pp. 413-422). IEEE.

Liu, H., Su, X. and Yuan, X., 2018. Accelerating unstructured large eddy simulation solver with GPU. *Engineering Computations*, *35*(5), pp.2025-2049.

Vincent, P., Witherden, F., Vermeire, B., Park, J.S. and Iyer, A., 2016, November. Towards green aviation with python at petascale. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (p. 1). IEEE Press.

Wang, H. , Lin, D. , Su, X. , & Yuan, X. . (2017). Entropy analysis of the interaction between the corner separation and wakes in a compressor cascade. Entropy, 19(7), 324.

Yang, C., Xue, W., Fu, H., You, H., Wang, X., Ao, Y., Liu, F., Gan, L., Xu, P., Wang, L. and Yang, G., 2016, November. 10M-core scalable fully-implicit solver for nonhydrostatic atmospheric dynamics. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (p. 6). IEEE Press.

ACKNOWLEDGMENTS

This work was supported by the National Key R&D Program of China under Project 2017YFB0203602.