Hindawi
Modelling and Simulation in Engineering
Volume 2019, Article ID 5231839, 10 pages
https://doi.org/10.1155/2019/5231839

Research Article

High-Performance Parallel Simulation of Airflow for Complex Terrain Surface

Kenji Ono[1] **and Takanori Uchida**[2]

[1]*Research Institute for Information Technology, Kyushu University, Fukuoka 819-0395, Japan*
[2]*Research Institute for Applied Mechanics, Kyushu University, Fukuoka 816-8580, Japan*

Correspondence should be addressed to Kenji Ono; keno@cc.kyushu-u.ac.jp

Received 3 September 2018; Accepted 18 December 2018; Published 3 February 2019

Academic Editor: Gaetano Sequenzia

It is important to develop a reliable and high-throughput simulation method for predicting airflows in the installation planning phase of windmill power plants. This study proposes a two-stage mesh generation approach to reduce the meshing cost and introduces a hybrid parallelization scheme for atmospheric fluid simulations. The meshing approach splits mesh generation into two stages: in the first stage, the meshing parameters that uniquely determine the mesh distribution are extracted, and in the second stage, a mesh system is generated in parallel via an in situ approach using the parameters obtained in the initialization phase of the simulation. The proposed two-stage approach is flexible since an arbitrary number of processes can be selected at run time. An efficient OpenMP-MPI hybrid parallelization scheme using a middleware that provides a framework of parallel codes based on the domain decomposition method is also developed. The preliminary results of the meshing and computing performance show excellent scalability in the strong scaling test.

1. Introduction

Power generation using natural energy sources has gradually replaced that from traditional thermal and atomic power sources due to concerns regarding the environment and resource sustainability. In particular, wind and solar have potential to provide low-emission power generation. Recently, large windmill power plants have achieved high efficiency in power generation. The selection of installation sites for windmill power plants directly impacts the entire life cycle and costs, including those of design, installation, operation, and removal.

It is thus important to develop a method for predicting detailed wind conditions utilizing data from observations and simulations. Many factors affect prediction results, and thus, numerous calculations with high-resolution meshes are usually required to obtain reliable results of wind flow over complex terrains. Reliable prediction of wind power generation requires taking into account various factors, including wind direction, turbulence, land features, atmospheric stratification, and periodicity. A simulation of wind conditions requires at least a 10-min time integration, which is very computationally intensive. To reduce wind flow simulation time, the speeding up and parallelization of the simulation algorithms is essential. In addition, the cost of mesh generation is one of the main issues in computational fluid dynamics (CFD) because meshing is extremely demanding and time-consuming for operators and sometimes requires user skill.

The present study proposes a two-stage mesh generation approach that greatly reduces the meshing cost and introduces a hybrid parallelization scheme for atmospheric fluid simulation. The meshing approach splits mesh generation into two stages: in the first stage, the parameters that uniquely determine the mesh distribution are extracted, and in the second stage, a mesh system is generated in parallel via an in situ approach based on the parameters obtained in the initialization phase of the simulation. To facilitate the extraction of meshing parameters, an easy-to-use graphical application that allows the user to interactively explore the optimal parameters is developed. The proposed two-stage approach is flexible since an arbitrary number of processes can be selected at run time. An efficient OpenMP-MPI hybrid parallelization scheme using middleware that provides a framework of parallelism based on the domain decomposition method is also developed. The preliminary results of the meshing and computing performance show excellent scalability.

2. Grid Generation

2.1. Grid System. Mesh generation for atmospheric fluid simulation with a complex terrain shape must reflect the complex configuration of the ground in the mesh system and appropriately resolve the boundary layer. Figure 1 shows an example of the mesh system used in the present research. A uniform Cartesian mesh is used for the XY-plane (horizontal direction), and a nonuniform distribution is used in the Z (vertical) direction, similar to the sigma coordinate system [1]. Equation (1) expresses the mapping relation between the physical and computational coordinate spaces:

$$\begin{cases} x = x(\xi), \\ y = y(\eta), \\ z = z(\xi,\eta,\zeta), \end{cases} \leftrightarrow \begin{cases} \xi = \xi(x), \\ \eta = \eta(y), \\ \zeta = \zeta(x,y,z). \end{cases} \quad (1)$$

2.2. Two-Stage Approach. Mesh generation for complex configurations is typically conducted using commercial software suites, e.g., [2], which have practical functions and user-friendly graphical user interfaces (GUIs). However, when such applications are run on commodity desktop PCs, limited memory becomes a critical constraint for generating large-scale mesh systems. In addition, the file size of the generated mesh is large, making the transfer of mesh data from a PC to computing servers time-consuming. For fast, large-scale mesh generation, the present study proposes a scalable, in situ two-stage meshing approach that can avoid time-consuming file access [3]. This approach splits mesh generation into two stages: in the first stage, the parameters that uniquely determine the mesh distribution are extracted using a GUI application, and in the second stage, a mesh system is generated in parallel using the parameters obtained in the initialization phase of the simulation. An overview of this procedure is illustrated in Figure 2.

2.3. Graphical Application. In the first stage, the parameters that uniquely determine the desired mesh system are found. Generally, terrain data are provided from satellite or geographic information system (GIS) services [4] and are usually in DEM or STL format, as shown in Figure 3. In this example, the STL format is used. A graphical application called *FXgen* is used to help determine the optimal parameters, which are used in the second stage. The essential parameters that determine the mesh distribution are the size of the computational region, the number of points, and the mesh spacing for the two ends of the line segments in the Z direction. These parameters can be interactively explored via a user-friendly GUI application, as illustrated in Figure 4. Although this process is heuristic, it is rather quick since the interface is efficient. Users can

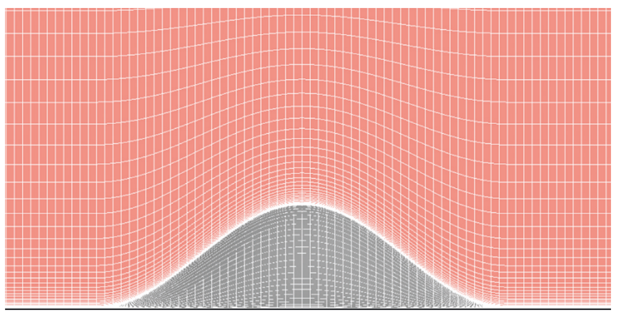
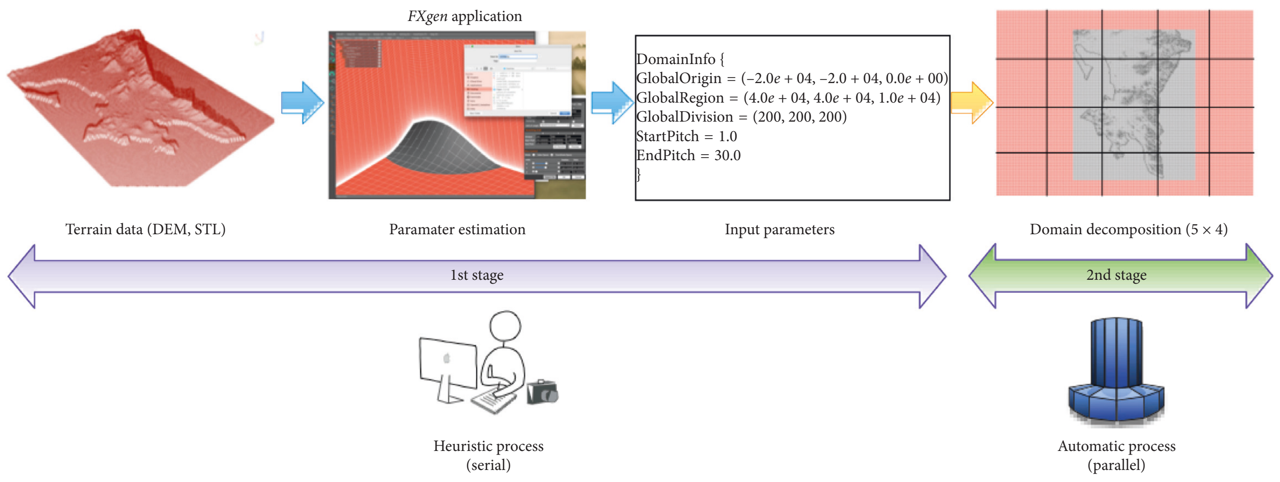


FIGURE 1: Cross section of the mesh system for an isolated peak configuration. A uniform Cartesian mesh is used in the horizontal direction, and nonuniform stretching is used in the vertical direction.

find the optimal parameters using the interface shown in Figure 4(b). For instance, users can specify the number of divisions in the X and Y (horizontal) directions, and the mesh distribution results will immediately be displayed from the top view, as shown in Figure 5(a). To determine the mesh distribution in the Z direction, the commonly used Vinokur's stretching function is used [5], where the number of divisions in the Z direction and the mesh spacing for the two ends of the line segments are specified. The result is a nonuniform stretched mesh in the Z direction, as depicted in Figure 5(b). Figure 5(c) shows the mesh distribution around the surface. Although the mesh distribution around a steep cliff is notoriously difficult to correctly obtain in mesh generation, the mesh obtained using the proposed method was well generated since the region just above the surface is filled by high-resolution fine meshes. Finally, the obtained parameters are output in JSON format, as shown in Figure 6.

2.4. Handling of Geometry Data. In a practical simulation, complex geometry shapes, given in file formats such as STL, OBJ, or iGES, are often encountered. In order to handle such geometry files in parallel, several libraries can be used. For instance, OpenFOAM provides the *distributedTriSurfaceMesh* class to load and redistribute mesh data [6], and Ray et al. presented a parallel mesh framework called *MOAB* (Mesh-Oriented dAtaBase) for building parallel mesh generators [7]. One of the present authors developed a polygon handling library called *Polylib* to manage polygonal elements in parallel [8], which provides functions such as load, save, redistribution, manipulation, grouping, and migration.

2.5. On-the-Fly Meshing. The obtained parameters are used to generate the same mesh generated in the first stage. In the second stage, the entire computational domain is divided by the number of processes in the control file or using command line arguments during the invocation of the parallel simulation program. The domain decomposition process is described using application programming interfaces (APIs) provided by the CBrick library [9], which are designed to facilitate the construction of message passing



```
DomainInfo {
GlobalOrigin = (−2.0e + 04, −2.0 + 04, 0.0e + 00)
GlobalRegion = (4.0e + 04, 4.0e + 04, 1.0e + 04)
GlobalDivision = (200, 200, 200)
StartPitch = 1.0
EndPitch = 30.0
}
```

Figure 2: Overview of two-stage meshing process. The first stage is processed serially and heuristically, and the second stage is processed in parallel and automatically.

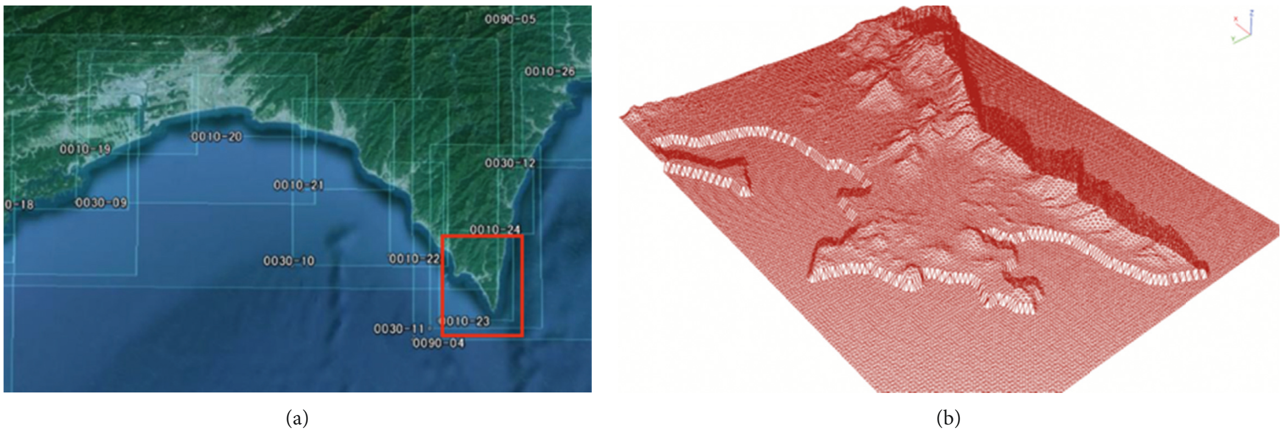
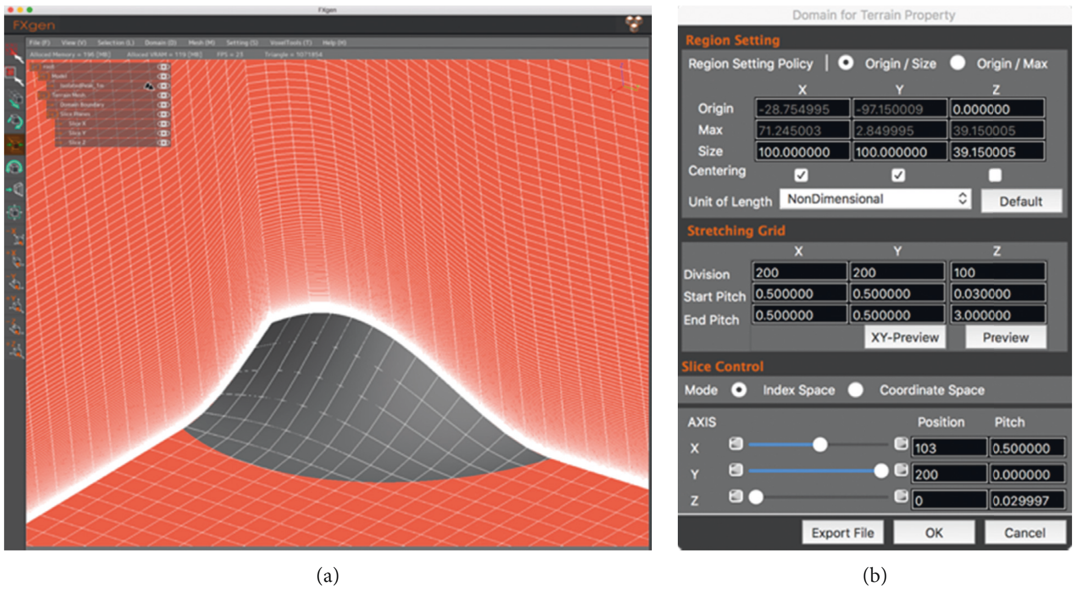


Figure 3: Basic rocket ship design. The rocket ship is propelled with three thrusters and features a single viewing window. The nose cone is detachable upon impact. (a) Specification of a region from GIS repository. (b) Polygonized terrain data corresponding to red region in (a).



Domain for Terrain Property

Region Setting

Region Setting Policy | [✓ Origin / Size] Origin / Max

	X	Y	Z
Origin	-28.754995	-97.150009	0.000000
Max	71.245003	2.849995	39.150005
Size	100.000000	100.000000	39.150005
Centering	✓	✓	☐

Unit of Length: NonDimensional

Default

Stretching Grid

	X	Y	Z
Division	200	200	100
Start Pitch	0.500000	0.500000	0.030000
End Pitch	0.500000	0.500000	3.000000

XY-Preview Preview

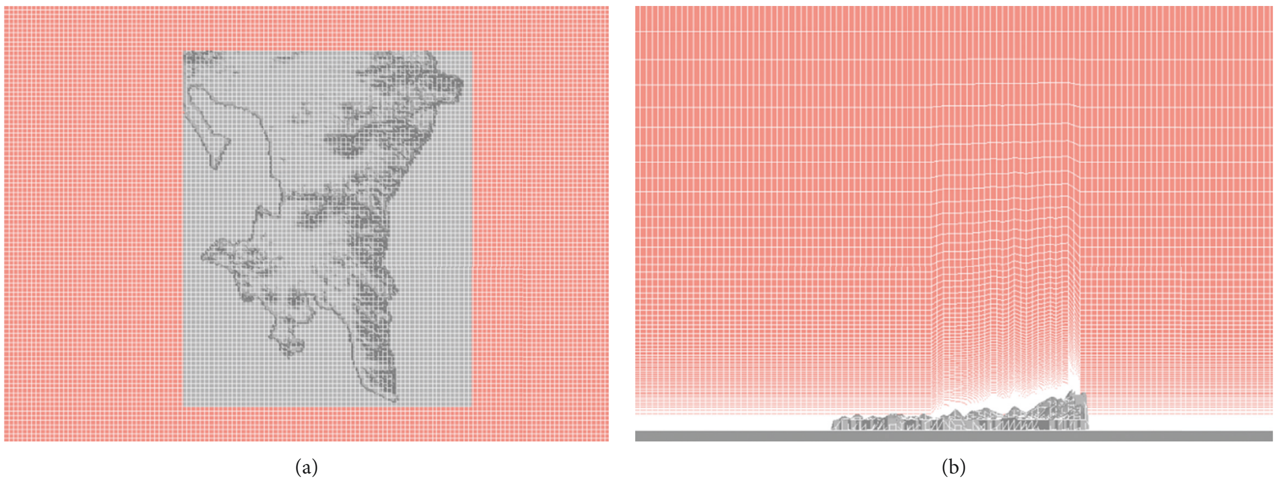
Slice Control

Mode [✓ Index Space] Coordinate Space

AXIS	Position	Pitch
X	103	0.500000
Y	200	0.000000
Z	0	0.029997

Export File OK Cancel

Figure 4: Exploration of meshing parameters with GUI application. (a) Screenshot of *FXgen* application. (b) Dialogue for parameter input.



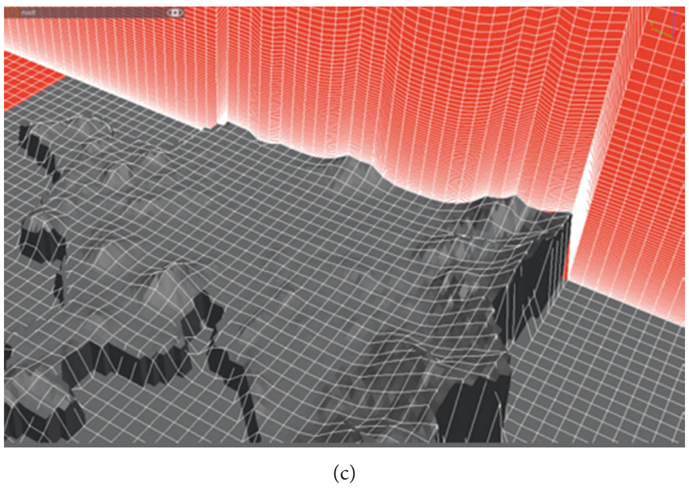


Figure 5: Meshing process in *FXgen*. The GUI application allows the desired parameters to be explored with confirmation of the mesh distribution. (a) Uniform mesh in the *XY* direction. (b) Stretched mesh in the *Z* direction. (c) Closeup view near the surface.

```
DomainInfo {
  GlobalOrigin   = (-2.000000e+04, -2.000000e+04, 0.000000e+00)
  GlobalRegion   = (4.000000e+04, 4.000000e+04, 1.000000e+04)
  GlobalDivision = (200, 200, 200)
  GlobalDivision = (  1,   1,   1)
  StartPitch     = 1.000000e+00
  EndPitch       = 3.000000e+01
}
```

Figure 6: Extracted mesh parameters. *GlobalOrigin* and *GlobalRegion* give information on the computational region, *GlobalDivision* is the number of divisions in the *X*, *Y*, and *Z* directions, and *StartPitch* and *EndPitch* represent the spacings of the two ends of line segments, respectively.

interface (MPI) applications and the optimization of their performance. A domain decomposition pattern is automatically calculated by the algorithm. By applying the same algorithm for mesh generation in both stages, the desired meshes can be generated using the same parameters. The proposed two-stage approach is flexible since an arbitrary number of processes can be selected at run time, thanks to on-the-fly meshing. A more detailed description can be found elsewhere [3].

3. Parallel Application

In this section, the performance enhancement for the wind simulator RIAM-COMPACT [10] is described.

3.1. Governing Equation and Solution Method. RIAM-COMPACT can be used for large-scale simulations of turbulence to predict the local wind flow over complex terrains. It uses collocated grids in the boundary-fitted coordinate system. Equations (2) and (3) show the non-dimensional governing equations of the flow, the filtered continuity equation for incompressible fluid, and the filtered Navier–Stokes equations, respectively.

$$\frac{\partial \overline{u}}{\partial x_i} = 0, \tag{2}$$

$$\frac{\partial \overline{u}_i}{\partial t} + \overline{u}_j \frac{\partial \overline{u}_i}{\partial x_j} = -\frac{\partial \overline{p}}{\partial x_i} + \frac{1}{Re}\left(1 + \frac{\nu_e'}{\nu'}\right)\frac{\partial}{\partial x_j}\frac{\partial \overline{u}_i}{\partial x_j} + 2\overline{S}_{ij}\frac{\partial \nu_e}{\partial x_j}, \tag{3}$$

$$\nu_e = \frac{\nu_e'}{U_0 L} = \left(C_s f_s \Delta\right)^2 |\overline{S}|, \tag{4}$$

$$\overline{S}_{ij}' = \frac{1}{2}\left(\frac{\partial \overline{u}_i'}{\partial x_j'} + \frac{\partial \overline{u}_j'}{\partial x_i'}\right), \tag{5}$$

where u_i is the velocity component, and p, ν, ν_e, and S_{ij} denote the pressure, fluid viscosity, eddy viscosity of the Smagorinsky model, and strain tensor, respectively. Note that the prime symbol indicates dimensional values.

3.2. Parallel Mesh Generation at Initialization. RIAM-COMPACT is parallelized using OpenMP and MPI. Figure 7 presents pseudocode of the initialization phase of the simulation, including domain decomposition, polygonal data loading/distribution, and mesh generation. Object *D* is an instance of the *CBrick* class, which provides APIs for domain management. Note that the argument "*XY*" of the method *findOptimalDivision()* sets the division of the domain to be only on the *XY*-plane. This is done because the stretching function is applied to the *Z* direction, and thus domain decomposition must be avoided in the *Z* direction. Object *PL* shows the use of polygon management class *MPIPolylib* to load the STL file at rank 0 and to distribute the partitioned polygon data to other ranks. Inside the **for** loop, object *sf* calculates the coordinate values at a given line segment in the *Z* direction and stores the values in the work array *zx*. The calculated coordinate values in array *zx* are copied to array *Z*. The object's method *sf- > distribution(zx)* means that the mesh distribution of a line segment between *pos_z* and *max_z* is calculated using Vinokur's stretching function, where the spacings of the two ends are *ds_st* and *ds_ed*, respectively, and the number of divisions is *NK*. For performance, the outer **for** loop is parallelized using OpenMP and the function *Zcopy()* is vectorized.

3.3. Parallelization. RIAM-COMPACT is written in Fortran90 and C/C++. Fortran90 is used for the main algorithms, and C/C++ is used for the main function, allocating memory, utilities, and bridging other C++ class libraries. For MPI-based parallelization, the APIs provided by the MPI library are used to describe the parallel code. For frequently used communication patterns, such as Allreduce and neighbor (point-to-point) communications, the *CBrick* library provides convenient APIs. OpenMP is used for threading **do/for** loops.

3.4. Performance Monitoring. It is very useful to assess the performance of applications in the production run and the tuning phase, which include various computing platforms. The performance monitoring library called *PMlib* [11] was used here to measure the parallel performance of RIAM-COMPACT at run time. *PMlib* provides simple APIs for instrumenting code and creating useful reports. Figure 8 shows pseudocode of instrumentation using *PMlib*. In this example, the user function *mykernel()* is being measured by calling the *PM.start()* and *PM.stop()* methods just before and after the *mykernel()* call. Figure 9 shows a report in simple format. A detailed report includes the performance of each process.

4. Related Work

In order to construct a high-throughput simulation method for atmospheric fluid flow, an efficient meshing method and scalable parallelization for kernel codes are essential. The literature on mesh generation was reviewed. Various mesh implementations have been proposed, such as a partitioning-based unstructured mesh [12], an octree or hierarchical Cartesian mesh with a space-filling curve [13], and a structured mesh with simple domain decomposition [14]. These methods are categorized as in situ meshing approaches, i.e., the mesh is automatically generated in the initialization phase of the CFD simulation. Among them, the Cartesian-based approach is probably the fastest and most powerful for meshing complex configurations. However, a naive implementation of this approach can have some difficulty creating smooth meshes that appropriately resolve the boundary layer near the surface just above the ground level. To overcome this problem, Yamazaki et al. used a Cartesian mesh with block-structured mesh refinement to concentrate the mesh around the surface, and introduced the cut-cell technique to generate a terrain-following mesh. Their method captures the boundary layer, but its scalability was only confirmed up to 16 threads [15].

Another issue of meshing is the automatic generation of a mesh system with complex terrain geometry for atmospheric simulation. Gargallo-Peiró et al. proposed an automatic procedure for generating hybrid meshes to simulate turbulent flows for wind farms [16]. They developed a mesh system that captures the topographic features of the ground and turbine area. Their mesh generation process is separated in two steps: in the first step, a background mesh is generated; in the second step, the mesh system around the turbine is inserted. They generated a mesh system for a wind farm with an element count on the order of 10 million in about 300 seconds using a meshing application on a PC.

Evetts reported that the Glyph script helps to automate specific meshing processes for wind farms [17]. The Glyph

```
int G_size[3];     // Number of division for each direction (global)
float G_region[3]; // Length of the entire computational region
int size[3];       // Number of division (local)
float pitch[3];    // Mesh width for each direction
float origin[3];   // Original points
int gc;            // Size of guide cell
int numProc;       // Number of processes
int myRank;        // Rank number of each subdomain

load_parameters("filename_of_control_parameter");

// Domain Decomposition
CBrick D;
D.setSubDomain(G_size, gc, numProc, myRank);
D.findOptimalDivision("XY");
D.createRankTable();

allocateArray();

// Polygon management
MPIPolylib* PL = MPIPolylib::get_instance();
PL->init_parallel_info(origin, size, gc, pitch);
PL->load_only_in_rank0( "filename_of_STL" );
PL->distribute_from_rank0();

int NI = size[0], NJ = size[1], NK = size[2];

float max_z = origin[2] + G_region[2];
float dz_st = spacing_at_start_of_line_segment;
float dz_ed = spacing_at_end_of_line_segment;
float* zx = Alloc_Real_S3D(NK);

// Intersection, Stretching
#pragma omp parallel for collapse(2)
for( int j=0; j<NJ; j++ ) {
  for( int i=0; i<NI; i++ ) {
    float pos_x = origin[0] + i * pitch[0];
    float pos_y = origin[1] + j * pitch[1];
    float pos_z = calc_intersection();

    Stretch* sf = new Stretch( NK, pos_z, max_z, dz_st, dz_ed );
    if( !sf->distribution(zx) ) printf("## ERROR\n");
    delete sf;
    // copy from zx to Z(i,j,*), this function is vectorized
    Zcopy(Z, i, j, zx);
  }
}
```

Figure 7: Pseudocode of the initialization phase of RIAM-COMPACT.

script has the following steps: import terrain, refine the area of interest, create a surface mesh, create a volume mesh, smooth and analyze the volume mesh, and export the file to a CFD solver. The script can make a grid system in a few minutes. This procedure is similar to our approach but it generates a grid file.

```
#include <PerfMonitor.h>
using namespace pm_lib;
PerfMonitor PM;

int main(int argc, char *argv[]) {
  PM.initialize();
  PM.setProperties("specified_label", PM.CALC);
  PM.start("specified_label");
  mykernel();
  PM.stop ("specified_label");
  PM.print(stdout);
  PM.printDetail(stdout);
  return 0;
}
```

Figure 8: Example of instrumentation using *PMlib*.

```
# PMlib Basic Report -------------------------------------------------------

Timing Statistics Report from PMlib version 5.8.5
Linked PMlib supports: MPI, no-OpenMP, no-HWPC, no-OTF
Parallel Mode:   FlatMPI (8192 processes)
Total time (PMlib enabled elapsed time) = 2.752032e+01 [sec]

Section             | call  |        accumulated time[sec]           | user defined argument values
Label               |       |      avr   avr[%]     sdv    avr/call  | user.value    sdv    user.perf
--------------------+-------+----------------------------------------+---------------------------
File_Output         :     1   5.258e+00  19.11  8.27e+00  5.258e+00    0.000e+00  0.00e+00   0.00 Mflops
Comm_Res_Poisson    :  1901   5.079e+00  18.46  8.01e-01  2.672e-03    4.983e+08  0.00e+00  98.12 MB/sec
Poisson_2_SOR_MAF   :  1901   4.993e+00  18.14  3.66e-01  2.627e-03    3.422e+10  1.95e+09   6.85 Gflops
Grid_gen            :     1   2.063e-01   0.75  4.33e-03  2.063e-01    0.000e+00  0.00e+00   0.00 Mflops
--------------------+-------+----------------------------------------+---------------------------
Sections per process          1.079e+01     -Exclusive COMM sections-  1.422e+09          131.81 MB/sec
Sections per process          1.100e+01     -Exclusive CALC sections-  3.637e+10            3.31 Gflops
--------------------+-------+----------------------------------------+---------------------------
Sections total job            1.079e+01     -Exclusive COMM sections-  1.165e+13            1.08 TB/sec
Sections total job            1.100e+01     -Exclusive CALC sections-  2.980e+14           27.10 Tflops
```

Figure 9: Example report generated by *PMlib*.

5. Simulation Results

5.1. Evaluation Environment. The supercomputer ITO at Kyushu University [18] was used to evaluate the performance of the mesh generation process and computing process. The specifications of ITO subsystem A are shown in Table 1. The evaluation was carried out for Flat-MPI mode and hybrid parallel (OpenMP + MPI) mode using up to 256 nodes, i.e., 8,192 processes for Flat-MPI mode and 512 processes for hybrid mode.

The computation time was measured for 20 steps of time integration from a given initial condition, and the performance was measured using the function provided by *PMlib*.

Table 1: Specifications of ITO subsystem A.

Architecture	Intel Skylake-SP
CPU and clock	Intel Xeon Gold 6154 processor, 3.0 GHz
Number of nodes	2,000
CPUs per node	2
Cores per CPU	18
Memory per node	192 GB
Compilers	Fujitsu compilers for C/C++ and Fortran

5.2. Performance of Flat-MPI Mode. Figure 10 shows the performance results for Flat-MPI mode. The generated mesh size was $2001 \times 721 \times 721$ (approximately 1 billion points), and the required memory for meshing was 66 GB. In this

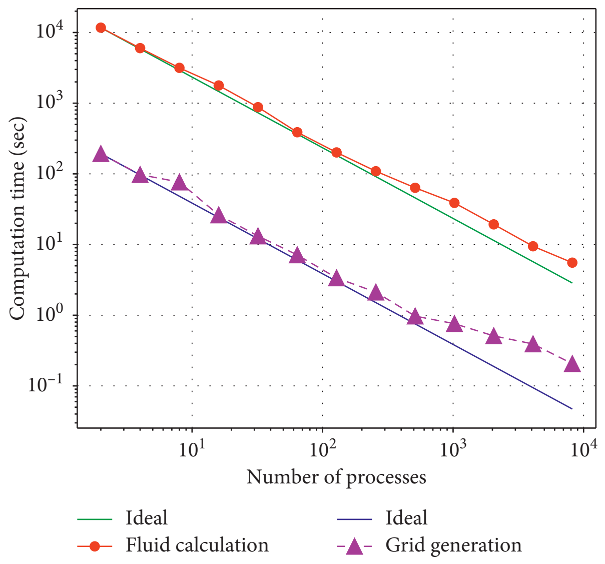


Figure 10: Performance of grid generation and fluid calculation for Flat-MPI mode.

case, the number of processes ranged from 2 to 8192. The obtained meshing performance was excellent with 512 processes but became worse with over 1024 processes. The main reason for this performance degradation is load imbalance, as the given mesh size is not an aliquot part, especially with over 512 processes. The performance of the fluid calculation exhibits the same behaviour. As shown in Figure 10, the cost of meshing is two orders of magnitude lower than that of the fluid calculation.

5.3. Performance of Hybrid Mode. Figure 11 shows the performance results for the hybrid parallel mode. The generated mesh size was $4001 \times 721 \times 721$ (approximately 2 billion points), and the required memory for meshing was 132 GB. The performance was measured for up to 512 processes, with 18 threads used per process. The performance of the fluid calculation was excellent compared to that for the Flat-MPI mode because the degree of freedom to divide the domain in the X direction was higher and thus load balancing improved. Although meshing performance dropped for the case with over 128 processes, the ratio of meshing time to flow calculation time was under 0.01; meshing was thus a negligible part of the entire computation time. Note that since the time integration was only 20 steps for this test case, the meshing time in a practical case that requires many time steps for integration can be ignored.

5.4. Computed Flow Field. Figure 12 shows that the fine mesh surrounding the surface was sufficient to correctly capture the boundary layer. This confirms that the structure of the flow separation at the ridge is well reproduced.

6. Conclusions

A two-stage in situ mesh generation approach was proposed for reducing the computational cost of the meshing process

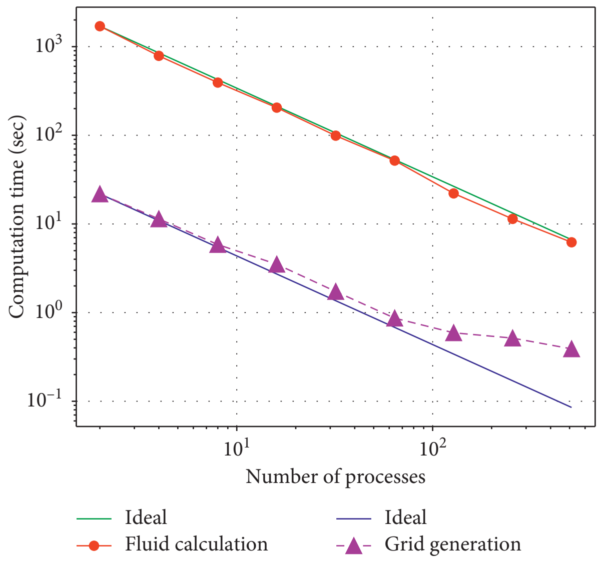


Figure 11: Performance of grid generation and fluid calculation for hybrid mode.

for large-scale parallel atmospheric fluid simulation with complex terrain. The proposed mesh generation process has two steps: in the first step, the parameters that uniquely determine the mesh distribution are extracted using a heuristic approach; in the second step, a mesh is automatically generated by a parallel meshing algorithm based on the obtained parameters in the initialization phase of the simulation. The proposed two-stage approach is flexible since an arbitrary number of processes can be selected at run time, thanks to the in situ meshing.

An efficient implementation of RIAM-COMPACT using OpenMP-MPI hybrid parallelization was also developed. RIAM-COMPACT was constructed using middleware that provides a framework of parallel codes based on the domain decomposition method. The preliminary performance results for the meshing and computing

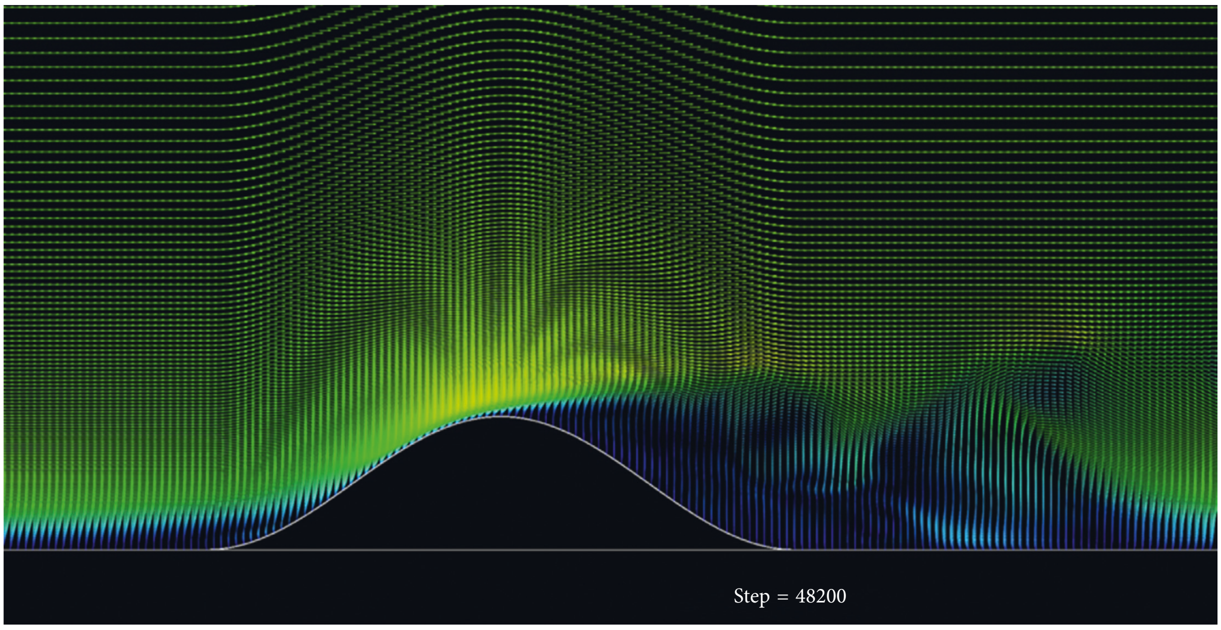


Figure 12: Computed vector field at the center line of the isolated peak example (Figure 1). The size of the mesh system was set to $801 \times 181 \times 181$, and the Reynolds number based on the height of the peak was 10^4.

performance show excellent scalability on the supercomputer ITO.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research used some computational resources of the K computer at the RIKEN Center for Computational Science in Kobe, Japan, and was carried out using the computer resources offered under the category of a general project by the Research Institute for Information Technology, Kyushu University. This research was partially supported by the Japanese Ministry of Education, Culture, Sports, Science and Technology (MEXT) as a social and scientific priority issue to be tackled using the post-K computer (Project ID: hp170238 for "Development of Innovative Design and Production Processes that Lead the Way for the Manufacturing Industry in the Near Future" and hp170270 for Accelerated Development of Innovative Clean Energy Systems).

References

[1] P. C. Chu and C. Fan, "Hydrostatic correction for sigma coordinate ocean models," *Journal of Geophysical Research*, vol. 108, no. 6, p. 3206, 2003.

[2] Pointwise, 2018, http://www.pointwise.com.

[3] K. Ono and Y. Uchida, "Two-stage in situ parallel meshing for large-scale Atmospheric fluid simulation over complex topography," in *Proceedings of 27th International Meshing Roundtable*, Eugene, Oregon, USA, October 2018.

[4] Geospatial Information Authority of Japan, 2018, http://cyberjapandata.gsi.go.jp/3d/sample.html.

[5] M. Vinokur, "On one-dimensional stretching functions for finite-difference calculations," *Journal of Computational Physics*, vol. 50, no. 2, pp. 215–234, 1983.

[6] D. Martineau, J. Gould, and J. Papper, "Towards an efficient distributed geometry for parallel mesh generation," in *Proceedings of 22nd International Meshing Roundtable*, Orlando, FL, USA, October 2013.

[7] N. Ray, I. Grindeanu, X. Zhao, V. Mahadevan, and X. Jiao, "Array-based hierarchical mesh generation in parallel," *Procedia Engineering*, vol. 124, pp. 291–303, 2015.

[8] Polygon management library, 2018, http://avr-aics-riken.github.io/Polylib.

[9] CBrick library, 2018, https://github.com/RIIT-KyushuUniv/CBrick.

[10] T. Uchida, "CFD Prediction of the airflow at a large-scale wind farm above a steep, three-dimensional escarpment," *Energy and Power Engineering*, vol. 9, no. 13, pp. 829–842, 2017.

[11] Performance monitor library, 2018, http://avr-aics-riken.github.io/PMlib.

[12] J. Chen, Y. Zheng, and X. Ning, "Scalable parallel quadrilateral mesh generation coupled with mesh partitioning," in *Proceedings of Sixth International Conference on Parallel and Distributed Computing Applications and Technologies (PDCAT'05)*, pp. 966–970, Dalian, China, December 2005.

[13] A. Lintermann, S. Schlimpert, J. H. Grimmen, C. Günther, M. Meinke, and W. Schröder, "Massively parallel grid generation on HPC systems," *Computer Methods in Applied Mechanics and Engineering*, vol. 277, pp. 131–153, 2014.

[14] V. D. Liseikin, *Grid Generation Methods*, Springer, Netherlands, 2010.

[15] H. Yamazaki, T. Satomura, and N. Nikiforakis, "3D cut-cell modelling for high-resolution atmospheric simula-tions,"

Quarterly Journal of the Royal Meteorological Society, vol. 142, 2015.

[16] A. Gargallo-Peiró, M. Avila, H. Owen, L. Prieto, and A. Folch, "Mesh generation for atmospheric boundary layer simulation in wind farm design and management," *Procedia Engineering*, vol. 124, pp. 239–251, 2015.

[17] S. Evetts, "Glyph scripting automates terrain meshing for wind turbine siting," 2018, http://www.pointwise.com/theconnector/2014-September/Glyph-Scripting-Automates-Terrain-Meshing-Wind-Turbine-Siting.html.

[18] Research Institute for Information Technology, Kyushu University, "The supercomputer ITO," 2018, https://www.cc.kyushu-u.ac.jp/scp/eng/system/01_into.html.