A Framework for Parallel Unstructured Grid Generation for Practical Aerodynamic Simulations

George Zagaris*†
College of William and Mary, Williamsburg, VA 23185
Shahyar Z. Pirzadeh‡
NASA Langley Research Center, Hampton, Virginia 23681
Nikos Chrisochoides§
College of William and Mary, Williamsburg, VA 23185

A framework for parallel unstructured grid generation targeting both shared memory multi-processors and distributed memory architectures is presented. The two fundamental building-blocks of the framework consist of: (1) the Advancing-Partition (AP) method used for domain decomposition and (2) the Advancing Front (AF) method used for mesh generation. Starting from the surface mesh of the computational domain, the AP method is applied recursively to generate a set of sub-domains. Next, the sub-domains are meshed in parallel using the AF method. The recursive nature of domain decomposition naturally maps to a divide-and-conquer algorithm which exhibits inherent parallelism. For the parallel implementation, the Master/Worker pattern is employed to dynamically balance the varying workloads of each task on the set of available CPUs. Performance results by this approach are presented and discussed in detail as well as future work and improvements.

I. Introduction

Parallel unstructured grid generation is not a new area in the field of applied Computational Fluid Dynamics(CFD). Although modern flow-solvers are parallelized and extensively utilized for routine CFD analysis, grid generation, a pre-processing step, still remains serial for most production environments. There is vast literature on techniques and algorithms, as well as, the issues and challenges related to parallel unstructured grid generation. In practice however, parallel grid generation, for end-to-end parallel CFD simulations, is still an open problem to date. For an in depth review, the interested reader is referred to a recent survey[1] on parallel mesh generation methods.

Lohner et al. proposed a scheme for Parallel Advancing front[2] that is based on the subdivision of the background grid. This scheme was determined by the authors to be inadequate for a production environment[3] and a new scheme was proposed. The new scheme employs the oct-tree[4,5] as an auxiliary data structure to parallelize grid generation at each front. A more recent publication[6] demonstrates the extension of this approach to Reynolds-Average Navier-Stokes (RANS) parallel unstructured grid generation and applicability of this approach in aerodynamic simulations. However, further improvements are necessary for this approach to exhibit better scalability. As the authors note, small imbalances between workloads incur heavy CPU penalty and decrease in performance.[3]

De Cougny and Shephard propose an oct-tree based technique[7] for parallel mesh generation. They build an oct-tree and decompose the entire domain. Octants that are interior to the domain are meshed using

*Graduate Student, College of William and Mary, Williamsburg, VA 23185, USA.
†Graduate Student Research Program Fellow, Configuration and Aerodynamics Branch, NASA Langley Research Center, Hampton, VA 23681-2199, USA.
‡Senior Research Engineer, Configuration Aerodynamics Branch, NASA Langley Research Center, Hampton, VA, 23681-2199, USA.
§Associate Professor, Computer Science Department, College of William and Mary, Williamsburg, VA 23185, USA

template subdivisions and the octants that intersect the domain boundaries are meshed using the advancing front. However, a shortcoming of this approach is that meshing the interface can create difficulties and a re-partitioning strategy is required[7] . Furthermore, the point distribution and thus size and shape of the mesh elements created by this approach is constrained by the oct-tree vertices and the template subdivisions. Consequently, this approach is not practical for generating anisotropically stretched grids which are desired for aerodynamic simulations.

Several methods based on partitioning a coarse mesh[8–11] have also been presented. In these methods, a coarse mesh is decomposed into several sub-domains. Next, the boundaries of the sub-domains are refined and then each sub-domain is meshed in parallel using a conventional off-the-shelf mesh generation method. The benefit of this approach is that it circumvents the difficulty of decomposition of an empty domain. Instead, the coarse mesh is partitioned using conventional graph partitioning methods such as the METIS[12] graph partitioning library. However, a shortcoming of this approach is that: (1) the method introduces artifacts at the partition interfaces which are not desirable for solving Partial Differential Equations (PDEs) and (2) this approach is subject to under-refinement and over-refinement. Furthermore, the method is not suited for Reynolds-Average Navier-Stokes (RANS) grid generation schemes that are essential in state of the art aerodynamic simulations.

A domain decomposition method based on the medial axis[13–15] was presented for 2D isotropic grid generation. While this approach seems promising, its extension to 3D is non-trivial primarily due to the computational complexity of the method in 3D. Furthermore, in the context of complex, real-world aerodynamic simulations substantial work is required in order to extend this work for (1) anisotropically "stretched" grids and (2) RANS grid generation methods.

Galtier and George[16] presented a method of pre-partitioning a surface mesh by triangulating calculated surfaces that intersect the boundaries of the domain. A similar method has also been presented by Ivanov et. al[17] and Larwood et. al.[18] However, the applicability of these methods to anisotropic and RANS grids has not been studied.

The focus of this paper is the design and implementation of a framework for parallel unstructured grid generation using VGRID:[19,20] NASA's unstructured grid generator. The framework presented in this paper is based on two fundamental building blocks: (1) the Advancing Partition (AP) method[21,22] and (2) the Advancing Front (AF) grid generation algorithm.[23,24] Both techniques are implemented within NASA's unstructured grid generator VGRID and integrated in the present framework for parallel unstructured grid generation. The AP method creates sub-domains by inserting interfaces that physically divide the domain which are part of the final grid. Next, the AF algorithm is applied in parallel to mesh the sub-domains. Importantly, the AP method utilizes the AF method to create the cells at the interface based on the same sizing function and quality metrics as the grid generation process. Consequently, the cells created at the interface contain no artifacts and are, by construction, compatible with the sizing function used for grid generation. Furthermore, the quality of the final grid generated in parallel is equivalent to the quality of the corresponding grid obtained by the serial code. This paper serves as a progress report on the current state of the framework and future directions for improvement and further research.

II. Parallel Unstructured Grid Generation Framework

The framework can be logically viewed as a system-of-systems consisting of five basic components. Figure 1 shows the basic component architecture of the framework.

At the top level, the *parallel application layer* implements a Master/Worker algorithm[25] which drives the entire process. The parallel application layer utilizes the *mobile task support layer*, which is built on top of the *Message Passing Interface (MPI)*, to send and receive tasks between the master process and the worker processes.

The *VGRID layer* provides functionality for the two fundamental operations: domain decomposition and grid generation. The implementation consists of two algorithms:

1. *The Advancing Front (AF) Algorithm*: The AF algorithm generates a volume mesh in a domain defined by a boundary triangulation (surface mesh). The initial front consists of the boundary faces. Then, the front advances iteratively towards the interior of the domain by adding new cells (tetrahedra) in the field and redefining the current front. The front advances until there are no more faces left on the front. For the details of the AF algorithm the interested reader is referred to references [23, 24].

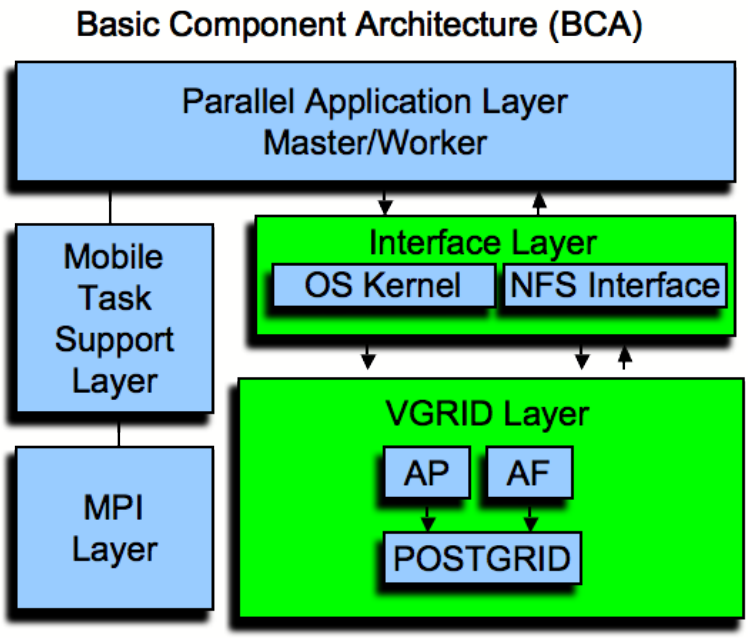


Figure 1. Basic component architecture of the parallel framework

2. *The Advancing Partition (AP) Algorithm*: The AP algorithm is based on the AF method. AP decomposes a domain defined by a boundary triangulation in two sub-domains. This process involves generating an *interface* or *separator* which physically decouples the domain. The interface is defined by a layer of cells (tetrahedra) along an imaginary partition plane located at the center of mesh density. The cells at the interface are generated using the AF algorithm described earlier. However, in this case, the faces on the front are restricted only to the faces intersecting the partition plane. For the details of the AP algorithm the interested reader is referred to references [21, 22].

The three main support processes for the two building blocks of the framework are: (1) AP for domain decomposition, (2) AF for mesh generation and (3) POSTGRID for post-processing the grid and completing any remaining unmeshed regions. Two types of interfaces between the *parallel application layer* and the *VGRID Layer* are supported in the current implementation of the framework: (a) system calls to the *Operating System (OS) kernel* for creating a new address space for the corresponding VGRID process, and (b) the Network File System *(NFS) interface*, for communicating input and output to and from the corresponding VGRID process.

There are four design considerations for implementation of a parallel grid generation framework:

1. **Stability**: The size and shape of the cells at the interface must be compatible with the sizing function used by the grid generation process.

2. **Scalability**: Serial parts of the code must be efficient or parallelized such that the parallel performance is not deteriorated.

3. **Code re-use**: The framework must enable the integration of off-the-shelf components.

4. **Load Balancing**: The computation time must be equalized on each CPU.

The present implementation of the framework addresses the stability and code-re-use considerations and, in part, provides acceptable load balancing and scalability. In particular, the stability consideration is satisfied a priori by the AP method since the metrics for quality, size and shape used to generate cells at the interface are the same as the metrics used by the AF algorithm during the grid generation process. Code re-use is achieved by integrating off-the-shelf VGRID components, and Load Balancing is partially addressed by the Master/Worker algorithm in addition to AP's built-in load estimation/balancing component. In the current implementation the scalability consideration is not fully addressed for two main reasons: (1) not all

serial parts of the code are parallelized and (2) the amount of exploitable concurrency varies over the life of the process. The bottlenecks and potential improvements in the current implementation are discussed in detail in the results section.

II.A. Implementation

From the implementation perspective, the framework can be viewed as an integrated, pipelined, system of modules. Modules were written using the C++[26] and Fortran90 programming languages, and MPI[27] was used for parallel programming. Figure 2 shows a functional flow block diagram of the framework.

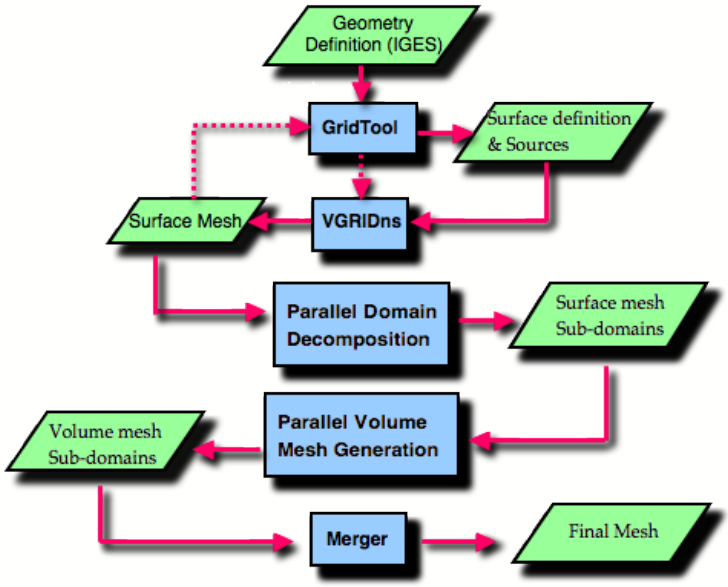


Figure 2. Functional flow block diagram of the parallel framework

The geometry model is given in the Initial Graphics Exchange Specification (IGES) format which is widely used within the CFD community. The pre-processing utility code GridTool[28] is used to bridge the gap between CAD and the input to the underlying grid generator system VGRID. This is a user-interactive step by which the user defines the surfaces (patches) of the geometry to be meshed and the *source elements*[29,30] which determine the grid distribution. After the surfaces and source elements are defined, a surface mesh is generated using VGRID sequentially.

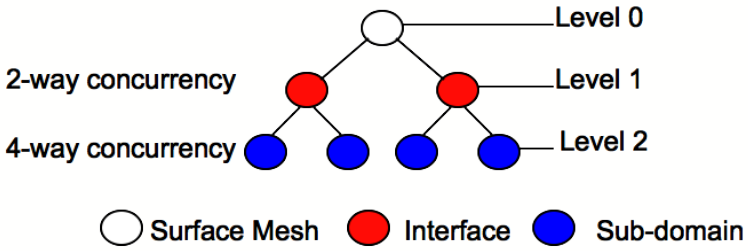
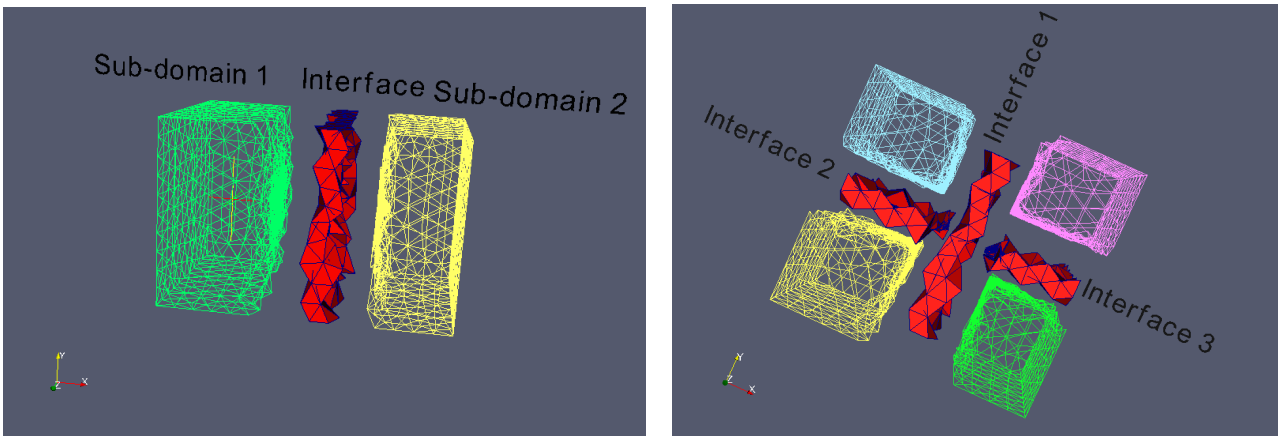


Figure 3. Logical hierarchical view of the domain decomposition

Next, the computational domain defined by the surface mesh is decomposed in two sub-domains by the AP method. The AP method is applied recursively to generate more sub-domains. Figure 3 shows a logical hierarchical view of the domain decomposition. At each level $l \in \{0, 1, 2, ..., N\}$ there are 2^l sub-domains and $2^l - 1$ interfaces, where l is level of decomposition. The desired level of decomposition is prescribed by the user as input. Figures 4(a) and 4(b) demonstrate the principle idea using a simple box configuration.

The recursive strategy for domain decomposition naturally maps to a divide-and-conquer algorithm, i.e., each decomposition step breaks the problem into a smaller sub-problem which can be solved independently.



(a) (b)

Figure 4. Example domain decomposition using the AP method on a simple box configuration (a) decomposition at level 1 (b) decomposition at level 2

Furthermore, the potential concurrency in this strategy is obvious. Since domain decomposition and mesh generation require only local operations at the sub-domain level, sub-domains can be processed concurrently as indicated in Figure 3. At level zero, the domain Ω defined by the surface mesh is decomposed into two sub-domains Ω_1, Ω_2 utilizing one CPU. At level one, the sub-domains Ω_1 and Ω_2 are further decomposed concurrently utilizing two CPUs. Notably, the amount of available concurrency doubles at each level of decomposition. After a domain is decomposed to the desired level, the sub-domains are meshed independently and in parallel. The final mesh is then composed by merging the sub-grids and interfaces.

For the parallel implementation, the Master/Worker pattern is selected in light of its programming ease, dynamic load-balancing and applicability to both distributed memory and shared-memory hardware architectures. At a high level, the Master/Worker consists of two logical processes: (a) the Master process which creates a set of tasks and dynamically assigns them to the worker processes in a First-Come-First-Serve (FCFS) fashion and (b) the Worker process which receives tasks and performs the computations. The *mobile task support layer* is implemented on top of MPI as a programming abstraction to facilitate sending and receiving of the tasks. In the present implementation, there are four types of tasks: (1) domain decomposition task, (2) mesh generation task, (3) acknowledgment task and (4) termination task.

The Master process includes two main data structures: (a) The *task-queue* which contains the list of tasks (mesh generation or domain decomposition tasks) to be sent to a corresponding worker process and (b) the *cpu-pool* which contains the list of CPU IDs corresponding to the available (currently idle) CPUs designated as workers. These data structures are being updated throughout the parallel grid generation process.

The communication protocol between the Master and Worker processes consists of two asynchronous messages. First, the Master initiates the communication by sending a task to a Worker. The task is either a mesh generation task or a domain decomposition task. Second, the receiving Worker sends a signal to the Master process to acknowledge completion of the corresponding task and gets back on the *cpu-pool* for later assignment. For termination detection, a single termination task is sent from the Master process to each Worker when all tasks are complete and all processes terminate.

In this paper, two distinct implementations of the framework are presented: (1) *PVGRID1*, which performs the decomposition serially and mesh generation in parallel and (2) *PVGRID2*, which performs the decomposition and mesh generation in parallel and asynchronously.

In *PVGRID1*, the Master process starts by performing the domain decomposition serially during which, the worker processes remain idle. After all the sub-domains are generated, the Master process creates a queue of mesh generation tasks corresponding to each sub-domain. Tasks are scattered on the set of available CPUs, and then the Master process waits for an acknowledgment. As soon as an acknowledgment signal is received, the corresponding process becomes available and the next task in the *task-queue* is assigned. This process is repeated until the *task-queue* is empty. Lastly, the Master process sends a Termination task to each of the worker processes to exit. A shortcoming of this implementation is that the potential concurrency during the domain decomposition is not exploited.

In contrast, *PVGRID2* further exploits concurrency during domain decomposition. The *task-queue* is initialized with domain decomposition at level zero and the *cpu-pool* contains the CPU IDs of all the worker processes. The Master process assigns the next task in the *task-queue* to the next available Worker from the *cpu-pool*. While the *task-queue* is empty, the Master process waits for acknowledgment. An acknowledgment for completion of a domain decomposition task triggers the addition of new tasks in the *task-queue*. This is an important distinction from *PVGRID1* where no new tasks are added to the *task-queue* during the parallel execution. There are two conditions for adding new tasks in the task-queue:

1. The sub-domains are at the desired level of decomposition prescribed by the user. In this case, two new Mesh Generation tasks are added to the task-queue for mesh generation.

2. Otherwise, two new Domain Decomposition tasks are added to the task-queue to schedule the sub-domains for decomposition.

Another significant difference between *PVGRID1* and *PVGRID2* is the criteria used for termination detection. *PVGRID1* terminates when the *task-queue* is empty. However, in *PVGRID2*, the *task-queue* being empty does not imply that all the work has been done. Recall, an acknowledgment of a domain decomposition task triggers the addition of new tasks in the *task-queue*. From the hierarchical process view, shown in Figure 3, it is easy to see that the total number of tasks at a desired level of decomposition $\mathcal{L}$ is the sum of the nodes in the hierarchy tree, given by $\sum_{l=0}^{\mathcal{L}} 2^l$. Hence, to detect termination, the master process keeps a task counter denoted by $\mathcal{C}$ of all complete tasks. Then, the process described above is repeated until the predicate $\mathcal{C} = \sum_{l=0}^{\mathcal{L}} 2^l$ is satisfied, i.e., all tasks are complete. Lastly, the Master process sends a Termination task to each of the worker processes to exit. As the results demonstrate in Section III, further exploiting concurrency in *PVGRID2* provides substantial improvements in the performance.

III. Results

Four sample configurations were employed for testing the present parallel grid generation framework and its performance: (1) a Simple Box configuration, (2) a Sphere-in-box configuration, (3) a transport wing/fuselage (DLR-F6) configuration and (4) a generic business jet configuration. For the performance evaluation of parallel computations, we measure and compare the speedup defined as, $\frac{T_s}{T_p}$ where T_s is the sequential execution time and T_p is the parallel execution time. The speedups are compared for two implementations, *PVGRID1* and *PVGRID2*. The Mercury cluster on TeraGrid,[31] supported by the NCSA[32] at the University of Illinois at Urbana-Champaign, was chosen as the target platform for this evaluation. Mercury is an IBM IA-64 Linux cluster equipped with 887 IBM dual-core Itanium 2 (@1.3/1.5GHz) processors with 4GB or 12GB of memory per node.

III.A. Simple Box Configuration

This geometry is defined by a cube and contains a single point source for defining the grid length-scales such that the grid distribution is uniform. A uniform mesh, consisting of 23 million elements, was generated for this configuration. The domain defined by the surface mesh was decomposed into 128 sub-domains, and the experiments were conducted by varying the number of processors from 2 to 64. Sample partitions obtained from this configuration are illustrated in Figure 5. Figures 6 (a) and 6 (b) show a comparison of the speedups and execution times obtained from *PVGRID1* and *PVGRID2*. Furthermore, Table 1 summarizes the quantitative performance results. The maximum speedup, with respect to the sequential execution time of *VGRID*, obtained from *PVGRID1* is 7 and 20 for *PVGRID2* using 64 CPUs. A notable observation from the speedup results in Table 1 is that *PVGRID2* scales better as the number of processors increases. In particular, beyond 16 processors, *PVGRID2* is about three times faster than *PVGRID1*. As discussed earlier, *PVGRID2* exploits more concurrency in the domain decomposition which substantially improved the scalability and performance. However, the amount of work, i.e., the number of sub-domains that can be processed concurrently, varies during the parallel domain decomposition phase. Consequently, while the number of sub-domains is smaller than the total number of CPUs, some CPUs remain idle. This is one of the main reasons why the scalability of the current implementation is sub-linear even for the ideal scenario where the geometry is a cube and the mesh is uniform, and thus the load is balanced.

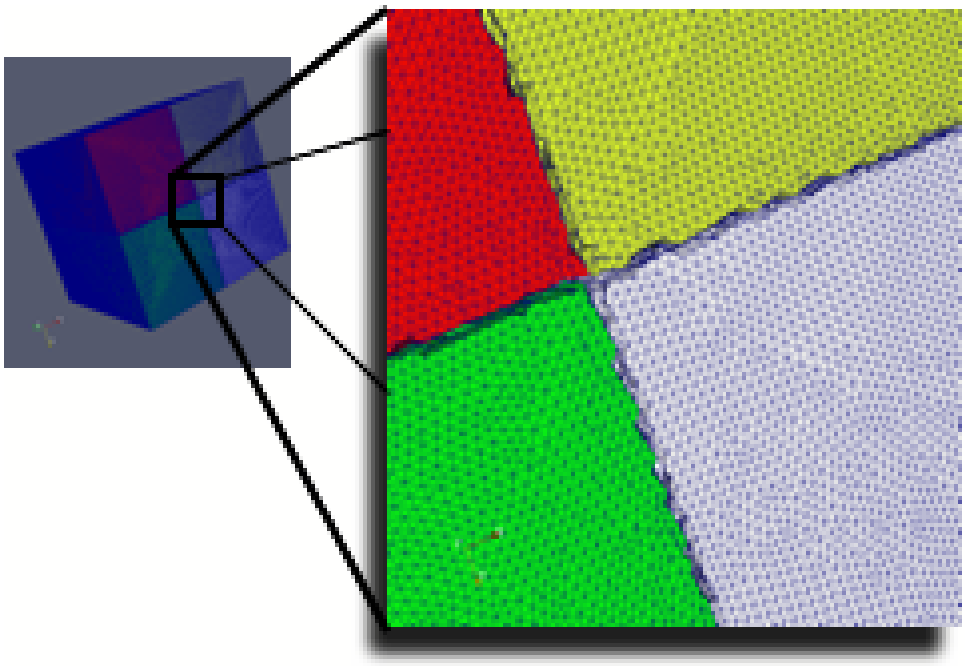
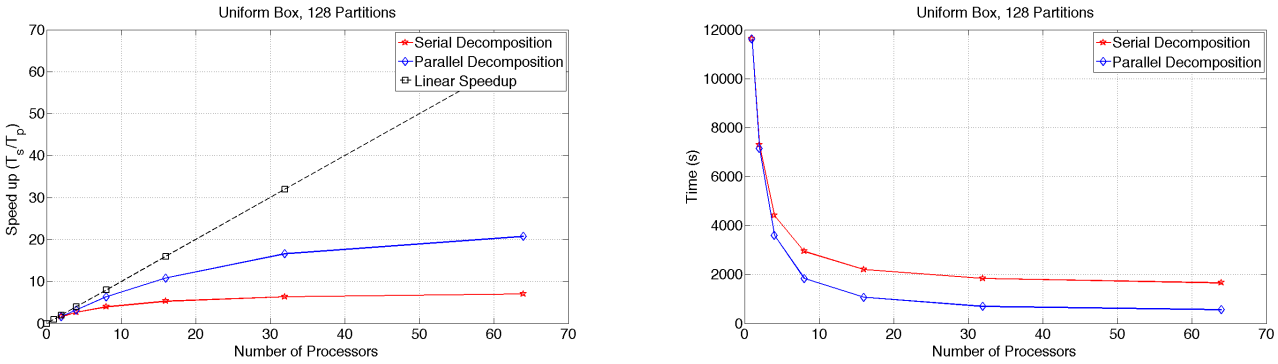


Figure 5. Sample partitioning of the simple box configuration



(a) (b)

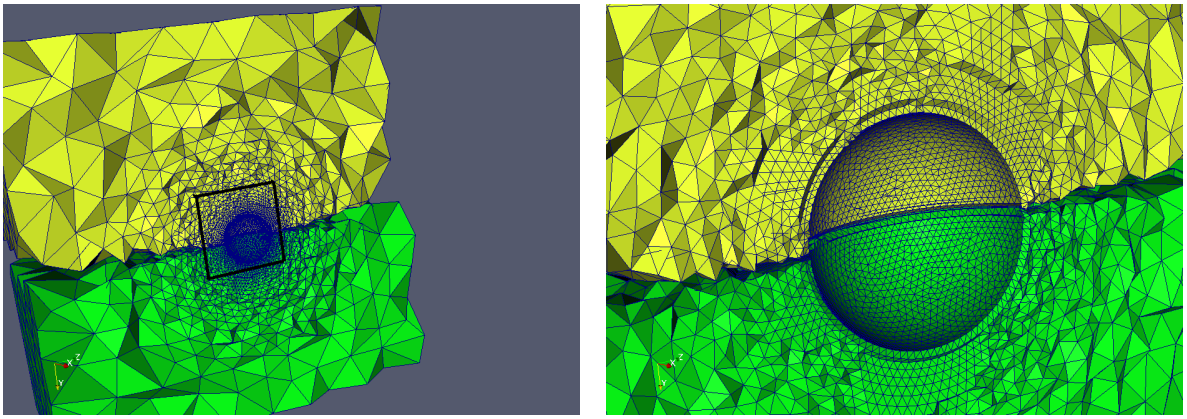
Figure 6. Performance plots for the Simple Box configuration (a) Comparative speedup plot of PVGRID1 (Serial Decomposition) and PVGRID2 (Parallel Decomposition) using 2-64 processors. (b) Comparative plot of the execution times of PVGRID1 (Serial Decomposition) and PVGRID2 (Parallel Decomposition) using 2-64 processors.

III.B. Sphere-in-box Configuration

This configuration consists of a spherical geometry positioned at the center of a cubical domain. A single sphere source is used to define non-uniform grid distribution in the field. The domain defined by the surface mesh was partitioned into 128 sub-domains and the performance measurements were made for up to 64 CPUs. In contrast to the Simple Box configuration, the final mesh is non-uniform and consists of 12 million elements. Figures 7(a) and 7(b) show sample partitions obtained for this configuration. Performance results are illustrated in Figures 8(a) and 8(b). Table 2 summarizes the quantitative performance measurements obtained using this configuration. For this case, the maximum speedup obtained was 13.76 using *PVGRID2* for 64 CPUs. However, in comparison to the Simple Box configuration, the speedup of *PVGRID2* is much smaller. This result is attributed mainly to two reasons: (1) The mesh is non-uniform and hence there are imbalances and (2) the problem size for this case, i.e. the number of elements in the mesh, is much smaller in comparison to the Simple Box case. As it is demonstrated in the Discussion section, scalability is proportional to the problem size. Generally, larger problems exhibit more scalability.

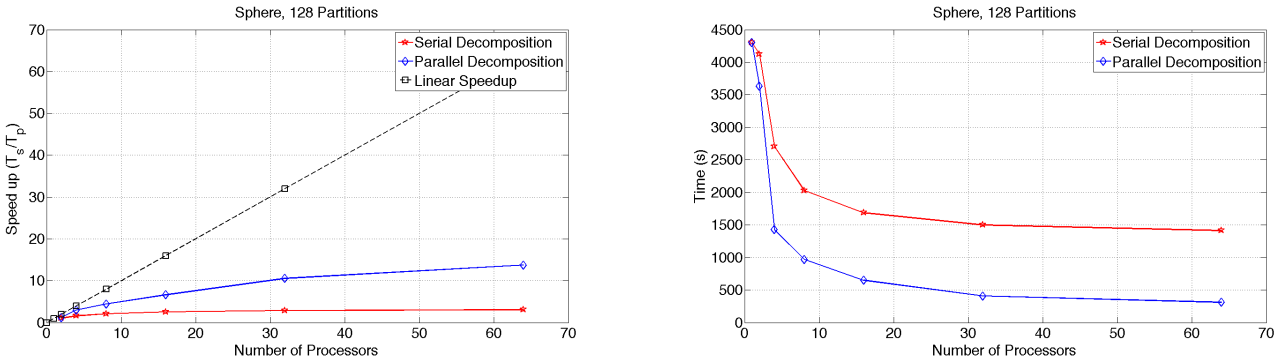
Table 1. Quantitative Performance Results for the Simple Box Configuration using PVGRID1 (Serial Decomposition) and PVGRID2 (Parallel Decomposition) in comparison with the serial version of the code VGRID.

CPUs	VGRID (s)	PVGRID1 (s)	PVGRID2 (s)	PVGRID1 Speedup (T_s/T_p)	PVGRID2 Speedup (T_s/T_p)
1	11627.01	N/A	N/A	N/A	N/A
2	N/A	7287.02	7145.44	1.59	1.63
4	N/A	4403.88	3593.22	2.64	3.24
8	N/A	2942.73	1834.08	3.95	6.34
16	N/A	2207.84	1072.66	5.27	10.84
32	N/A	1834.02	699.193	6.33	16.63
64	N/A	1652.42	560.572	7.04	20.74



(a) (b)

Figure 7. Sample partitioning of the Sphere-in-box configuration (a) Top-left and bottom-left partitions (b) Close-up view of the sample partitions



(a) (b)

Figure 8. Performance plots of the Sphere-in-Box configuration (a) Plot of the speed-up obtained for 2-64 processors from PVGRID1 (Serial Decomposition) and PVGRID2 (Parallel Decomposition) (b) Comparative plot of the execution times of PVGRID1 and PVGRID2 using 2-64 processors.

III.C. Wing/Fuselage DLR-F6 Transport

The DLR-F6 is a generic transport geometry model consisting of a wing and fuselage. Due to geometric symmetry, only half the configuration is modeled. A total of 74 line and point sources are used to define the grid distribution in the field. The domain defined by the surface mesh was decomposed into 128 sub-domains

Table 2. Quantitative Performance Results for the Sphere-in-Box configuration using PVGRID2 (Parallel Decomposition) in comparison with the serial version of the code VGRID.

CPUs	VGRID (s)	PVGRID2 (s)	PVGRID2 Speedup ($\frac{T_s}{T_p}$)
1	4300.5	N/A	N/A
2	N/A	3630.84	1.18
4	N/A	1425.9	3.02
8	N/A	968.97	4.44
16	N/A	649.74	6.62
32	N/A	407.515	10.55
64	N/A	312.555	13.76

and the final mesh consisted of 32 million elements. Figure 9 shows the surface mesh and sample partitions for the DLR-F6 wing/fuselage configuration (Note, the wing is not shown in Figure 9 for visual clarity). Performance measurements for up to 64 processors were obtained as presented in Figures 10 (a) and 10 (b). Furthermore, quantitative performance results, obtained for this configuration, are summarized in Table 3.

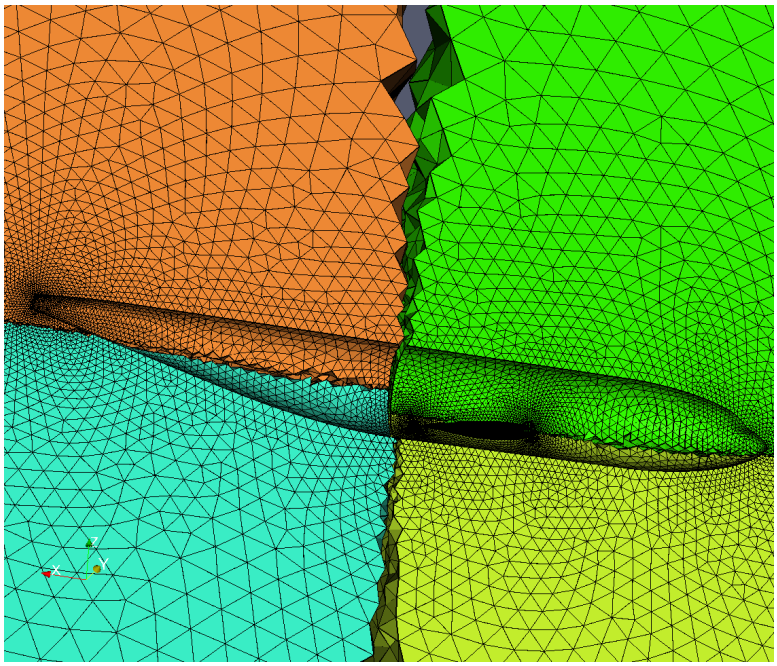
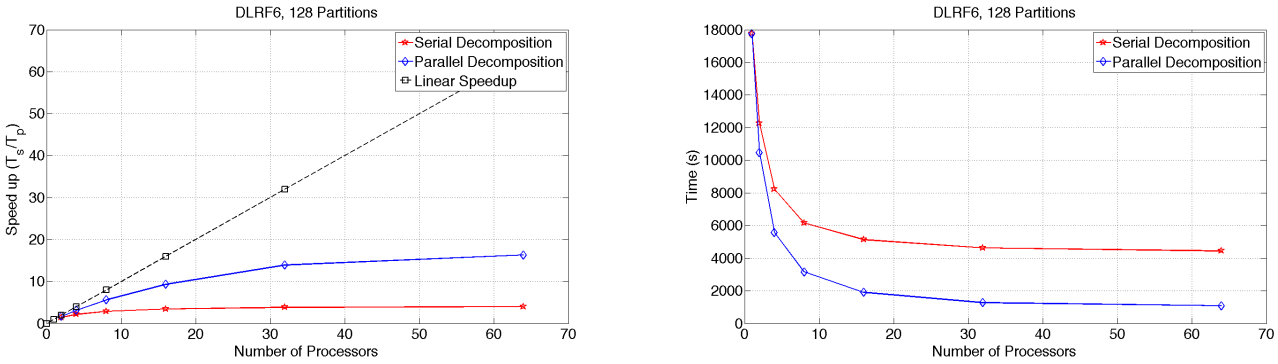


Figure 9. Sample partitions of the DLR-F6 wing/fuselage configuration.

III.D. Generic Business Jet

This geometry includes a wing and a fuselage along with a pylon and a flow-through nacelle. As with in the DLR-F6 case, only half the geometry is modeled due to symmetry. A total of 134 line and point sources are used to define the grid distribution for this example. This case is employed to explore the feasibility of the present framework on more complex geometries than the previous cases. A mesh consisting of 25 million elements was generated for the Generic Business Jet configuration. The domain defined by the surface mesh was partitioned into 128 sub-domains and the performance measurements were produced by varying the number of processors from 2 to 64. Sample partitions using this configuration are illustrated in Figures 11 (a) and 11 (b). For visualization purposes, a coarser mesh was used to produce these figures. Performance results are presented in Figures 12 (a) and 12 (b). A summary of the quantitative performance results is presented in Table 4.

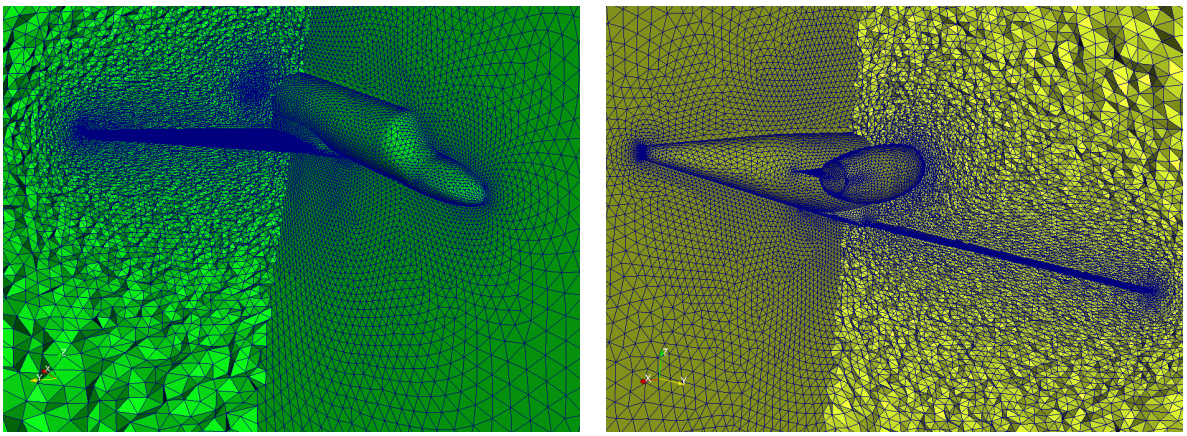


(a) (b)

Figure 10. Performance plots using the DLR-F6 wing/fuselage configuration (a) Comparative speedup plots using PVGRID1 (Serial Decomposition) and PVGRID2 (Parallel Decomposition) (b) Comparative plots of the execution times for PVGRID1 and PVGRID2.

Table 3. Quantitative Performance Results for the DLR-F6 wing/fuselage configuration PVGRID2 (Parallel Decomposition) in comparison to the serial version of the code VGRID.

CPUs	VGRID (s)	PVGRID2 (s)	PVGRID2 Speedup ($\frac{T_s}{T_p}$)
1	17745.61	N/A	N/A
2	N/A	10471	1.69
4	N/A	5569.33	3.19
8	N/A	3155.44	5.62
16	N/A	1904.48	9.32
32	N/A	1275.42	13.91
64	N/A	1087.04	16.32

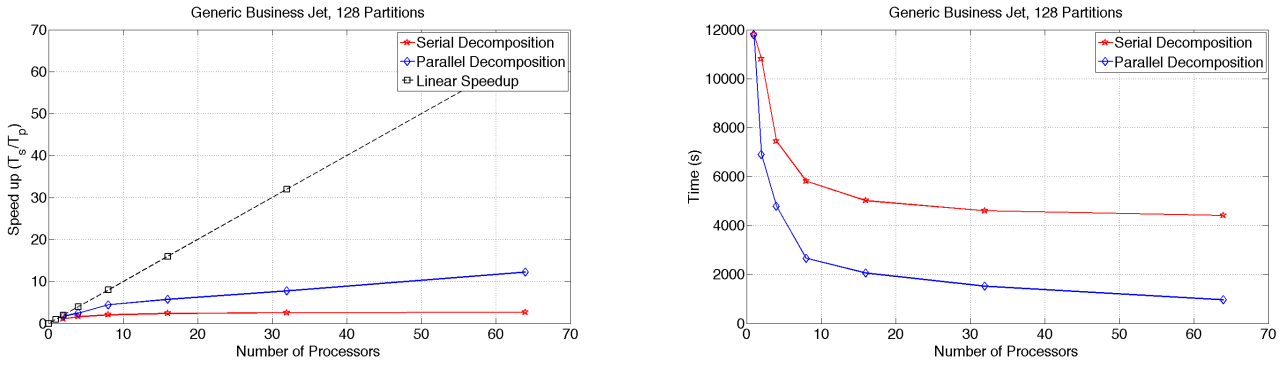


(a) (b)

Figure 11. Sample partitions of the Generic Business Jet configuration (a) Partition consisting of the wing and front part of the fuselage (b) Partition consisting of the flow-through nacelle and tail of the fuselage

IV. Discussion

The results presented earlier demonstrated the applicability of the current framework for parallel unstructured grid generation. However, a notable characteristic of the performance measurements obtained using the current implementation of the framework is that the speedup lacks the desired level of scalability and performance. This section presents the performance bottlenecks and shortcomings identified in the current



(a) (b)

Figure 12. Performance plots using the Generic Business Jet configuration (a) Comparative speedup plots using PVGRID1 (Serial Decomposition) and PVGRID2 (Parallel Decomposition) (b) Comparative plot of the execution times of PVGRID1 and PVGRID2.

Table 4. Quantitative Performance Results for the Generic Business Jet configuration using PVGRID2 (Parallel Decomposition) with respect to the serial code VGRID.

CPUs	VGRID (s)	PVGRID2 (s)	PVGRID2 Speedup ($\frac{T_s}{T_p}$)
1	11797.7	N/A	N/A
2	N/A	6894.7	1.71
4	N/A	4782.08	2.47
8	N/A	2662.36	4.43
16	N/A	2062.2	5.72
32	N/A	1521.15	7.76
64	N/A	962.35	12.26

implementation as well as the degree to which the four parallel design considerations, i.e., *Stability*, *Code re-use*, *Scalability*, and *Load Balancing*, are satisfied.

IV.A. Stability and Code Re-use

The Stability and Code re-use design considerations are interrelated and are the two most important driving parameters for the design phase. Intuitively, the biggest benefit of code re-use is to reduce the implementation time and cost by integrating existing components. However, in our experience code re-use is also the means by which the stability consideration is satisfied. Re-using the existing functionality of the AF kernel in the AP method, used for domain decomposition, enabled the generation of *interfaces* or *separators* that are compatible with the same metrics for quality, size and shape used for grid generation by the AF method. Hence, the stability and code re-use considerations are satisfied in the current implementation of the parallel framework.

IV.B. Load Balancing

Load Balancing is addressed, in part, by the Master/Worker model and by a load estimation algorithm built in the AP method for positioning partition interfaces. Figure 13 shows the grid element distribution among sub-grids obtained for the examples presented. As illustrated, the number of points and elements on each sub-domain vary greatly which can negatively affect load balancing and, thus, the speedup performance. Partitioning an empty (unmeshed) domain such that the load on each sub-domain is balanced is a very difficult problem. Further investigation and extension of the present framework is planned for future work.

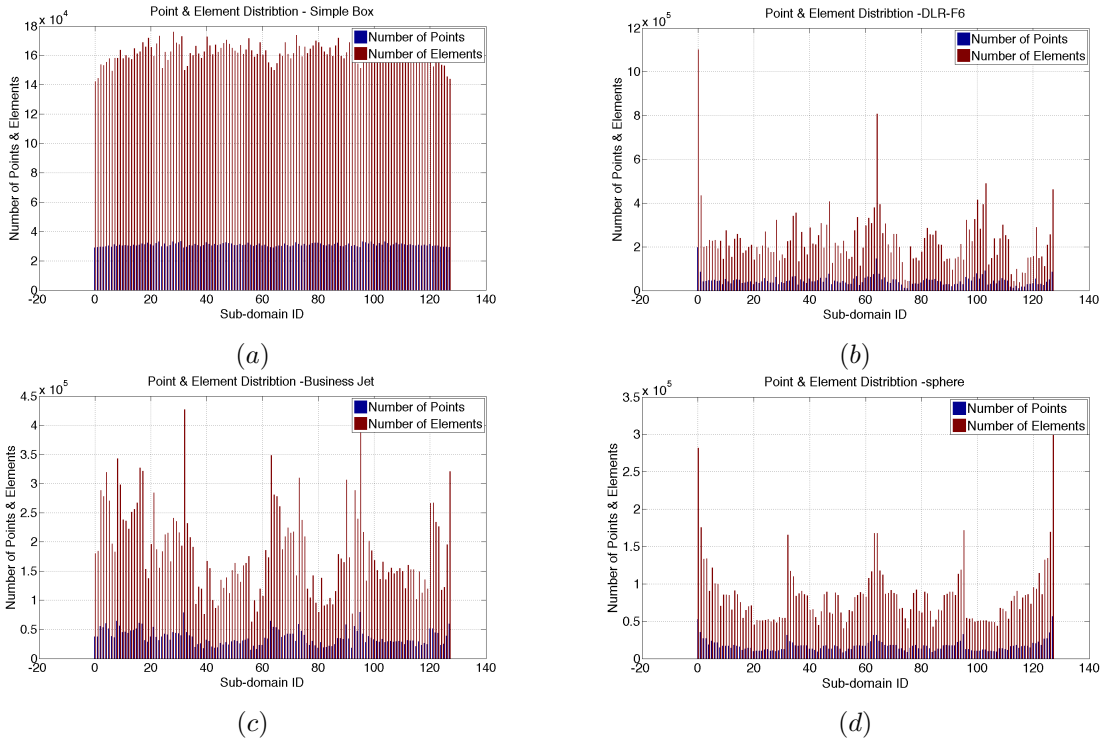


Figure 13. Distribution of grid elements among partitions (a) Simple box configuration (b) DLR-F6 configuration (c) Business jet configuration (d) Sphere-in-Box configuration

IV.C. Scalability

In *PVGRID2*, scalability is partially addressed by exploiting more concurrency in the domain decomposition phase. However, a notable characteristic of the performance measurements on the examples presented earlier is that the speedup is sub-linear and beyond 8 CPUs there are little practical performance benefits.

The following paragraphs present the identified shortcomings and potential improvements in the current implementation of the framework.

IV.C.1. Concurrency of the Domain Decomposition

As discussed earlier, *PVGRID2* exploits more concurrency in the domain decomposition phase. However, an inherent performance bottleneck is that the exploitable concurrency for domain decomposition varies at each level of decomposition. As indicated in Figure 3, there is little or no exploitable concurrency at the topmost levels. Concurrency doubles at each level of decomposition. Hence, at the topmost level utilization of resources is low, i.e., most processes are idle waiting for work. Utilization increases as the amount of concurrency increases at later levels.

Furthermore, decomposing a (single) sub-domain is performed serially. The complexity of generating an interface is the same as with the mesh generation process (since the AP method utilizes the AF method for generating the interface). By Amdahl's law[25] it is known that the serial parts of the program can significantly constrain the speedup that can be achieved by adding more CPUs. The serial decomposition part constrains how fast resources can be utilized and the speedup obtained as the number of CPUs increases.

There are two potential improvements to exploit more concurrency for the Domain Decomposition:

1. Decomposing a domain into N sub-domains instead of 2 (at each level). By this approach, N-way concurrency is attained at each level of decomposition which enables resources to be utilized faster.

2. Exploit shared-memory parallelism, i.e., using threads, for domain decomposition. In contrast to the current approach where cells at the interface(s) are introduced one-by-one, exploiting shared-memory parallelism enables multiple cells to be introduced concurrently. Consequently, the execution time for generating the interface(s) is reduced and resources can be utilized faster.

IV.C.2. Interpolation of Background Sources

A shortcoming of the off-the-shelf approach of parallelization employed in the present framework is that the serial code is not fully parallelized. At a high level, the serial code for volume mesh generation works as follows: *For every face on the front, interpolate the grid spacing from the background sources.* The algorithm to implement this operation consists of two nested loops: (a) an outer-loop over all the faces on the current front and (b) an inner-loop over all the sources. Note, that the number of faces on the front changes and the number of sources is constant throughout the process. Further, the domain decomposition described in this work essentially reduces the number of faces on the front, i.e., the number of iterations of the outer-loop. However, the number of iterations in the inner-loop is constant for all the sub-domains. Hence, the serial code is not fully parallelized. Since, the execution time of the inner-loop is serial and constant regardless of the decomposition, the serial parts of the code deteriorate the performance gained by adding more CPUs. Thus, the number of sources, i.e., the number of iterations of the inner-loop, has an effect in the overall performance. Table 5 summarizes the speedup obtained from a variety of configurations and the corresponding number of sources. As illustrated, a large number of sources can greatly affect the speedup of the parallel application. Since sources can be interpolated independently, the inner-loop can be easily parallelized in order to eliminate the serial parts of the algorithm and further improve the performance.

Table 5. Summary of speedups for a variety of configurations in relation to the number of sources

Configuration	Number of Sources	Maximum Speedup
Simple Box	1	20.74
Sphere-in-box	1	13.76
DLR-F6	74	16.32
Business Jet	134	12.26

IV.C.3. VGRID Interface

In the present implementation, the parallel framework creates a VGRID process using a system kernel call. Moreover, input and output to VGRID is accomplished via the Network File System (NFS) as indicated in Figure 1. This approach introduces the following overheads:

1. creation of a new address space for the VGRID process.
2. reading files from NFS and initializing data-structures.
3. writing files to NFS.

Better interfacing with VGRID eliminates these performance impediments. An Application Programming Interface (API) for VGRID resolves these issues by:

1. integrating VGRID's functionality within a single address space, and
2. allowing data communication through function calls instead of the file system.

Furthermore, appropriate error handling can easily be enabled via an API with appropriate return codes from the function calls.

IV.C.4. Problem Size

Another parameter that has an effect on the performance of parallel computations is the size of the problem, i.e., the mesh size in terms of the number of cells and points in the mesh. Generally, larger problems exhibit more scalability. In other words, grids of larger sizes have more performance gains when using more CPUs. Figure 14 demonstrates this observation using the Simple Box configuration. The experiment on this configuration, presented earlier, gave a speedup of 20 using 64 CPUs for a grid of 23 million cells. Increasing the grid size to 100 million cells with the same number of processors and partitions, produced a speedup of 30 (see Figure 14). Decomposition of larger problems yields sub-domains with more work to compensate

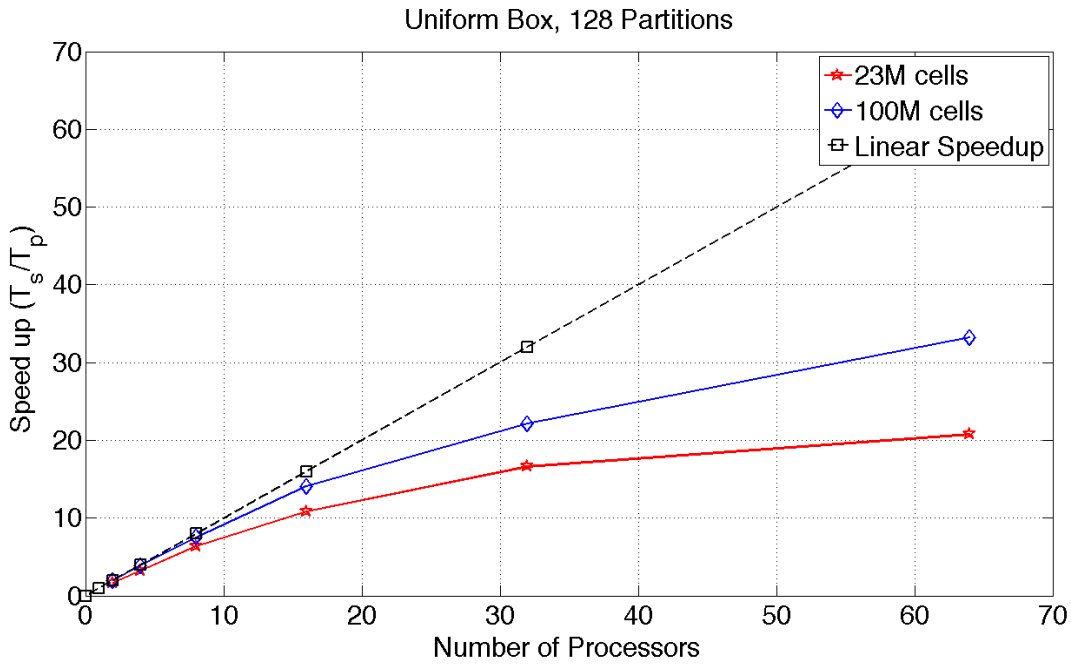


Figure 14. Comparative plot of the speedup obtained for generating a uniform grid of different sizes for the simple box configuration.

for the under utilization of resources at earlier stages of decomposition (due to the insufficient amount of exploitable concurrency which was discussed earlier).

For a given problem size, there is an upper-bound on the number of CPUs that can be effectively utilized. In most practical CFD settings, there is no need for an excessive number of processors for grid generation purposes. In practice, a few partitions and processors (on the order of 10) are usually sufficient. The performance of the present parallel framework, for the example configurations presented in this paper, is reasonable in the range of 2 to 10 processors.

V. Concluding Remarks

In this preliminary report, the design and implementation of a framework for parallel unstructured grid generation is presented. While the applicability of the present approach for CFD simulations is demonstrated on a number of sample grids, the scalability performance of the current implementation require further improvements. The performance of the current implementation in practice, where 2 to 10 processors are utilized for grid generation purposes, is reasonable. The *Scalability* of the current implementation however, beyond 16 processors, may be low depending on the problem size. The most important performance drawbacks identified in the current implementation are:

1. Workload imbalances between sub-domains which deteriorate the performance.
2. Inherently insufficient exploitable concurrency during the early stages of domain decomposition which leads to under utilization of resources.
3. Parts of the code that remain serial, in particular source interpolation.
4. Overhead for process creation, file I/O and data-structure initialization due to inefficient coupling between the parallel application layer and the VGRID layer.

Future work is focused on further improving the scalability of the current implementation by addressing the above issues.

Acknowledgments

This is a joint work by the College of William and Mary in Williamsburg, VA and NASA Langley Center (LaRC) in Hampton, VA. This work is partially funded by the NASA Graduate Student Research Program (GSRP) grants NL06AA02H (2006) and NNX07AM10H (2007,2008), the NASA Fundamental Aeronautics Program (FAP), Subsonic Fixed Wing project and by the National Science Foundation (NSF) through TeraGrid resources provided by the National Center for Supercomputing Applications (NCSA) at the university of Illinois at Urbana-Champaign. Additionally, the authors would like to thank Andrey Chernikov, Andriy Fedorov and Andriy Kot from the College of William and Mary for their useful comments and helpful discussions.

References

[1]Chrisochoides, N., *Numerical Solution of Partial Differential Equations on Parallel Computers*, chap. 7 Parallel Mesh Generation, Springer-Verlag, 2005, pp. 237–255.

[2]Lohner, R., Camberos, J., and Merriam, M., "Parallel Unstructured Grid Generation," *Computer Methods in Applied Mechanics and Engineering*, Vol. 95, 1992, pp. 343–357.

[3]Lohner, R. and Cebral, J. R., "Parallel Advancing Front Grid Generation," *Proceedings of the Eighth International Meshing Roundtable*, October 1999, pp. 67–74, South Lake, Tahoe, CA, USA.

[4]Samet, H., *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, 1990.

[5]Samet, H., *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*, Addison-Wesley, 1995.

[6]Sharov, D., Luo, H., Baum, J. D., and Lohner, R., "Unstructured Navier-Stokes grid generation at corners and ridges," *International Journal for Numerical Methods in Fluids*, Vol. 43, 2003, pp. 717–728.

[7]de Cougny, H. L. and Shephard, M. S., "Parallel Volume meshing using face removals and hierarchical repartitioning," *Computer Methods in applied mechanics and engineering*, Vol. 174, 1999, pp. 275–298.

[8]Said, R., Weatherill, N. P., Morgan, K., and Verhoeven, N. A., "Distributed Parallel Delaunay Mesh Generation," *Computer Method in Applied Mechanics and Engineering*, Vol. 177, 1999, pp. 109–125.

[9]Chernikov, A., Chrisochoides, N., and Barker, K., "Parallel Programming Environment for Mesh Generation," *Proceedings of the Eighth International Conference on Numerical Grid Generation in Computational Field Simulations*, June 2002, pp. 805–814, Honolulu, HI, USA.

[10]Ito, Y., Shih, A. M., Erukala, A. K., Soni, B. K., Chernikov, A., Chrisochoides, N. P., and Nakahashi, K., "Parallel Unstructured Mesh Generation by an Advancing Front," *Mathematics and Computers in Simulation*, Vol. 75, January 2007, pp. 200–209.

[11]Zagaris, G., Pirzadeh, S., Chernikov, A., and Chrisochoides, N., "Parallel Advancing Front for CFD Simulations of Real-World Aerodynamic Problems," *9th US National Congress on Computational Mechanics*, July 2007, San Francisco, CA, USA.

[12]"Metis, Family of Multilevel Partitioning Algorithms," http://glaros.dtc.umn.edu/gkhome/views/metis.

[13]Linardakis, L. and Chrisochoides, N., "Delaunay Decoupling Method for Parallel Guaranteed Quality Planar Mesh Refinement," *SIAM Sci. Comput.*, Vol. 27, No. 4, 2006, pp. 1394–1423.

[14]Linardakis, L. and Chrisochoides, N., "A Static Medial Axis Domain Decomposition for 2D Geometries," *ACM Transactions on Mathematical Software*, Vol. 34, No. 1, November 2005, pp. 1–19.

[15]Linardakis, L., *Decoupling Method for Parallel Delaunay 2D Mesh Generation*, Ph.D. thesis, College of William and Mary, Williamsburg, Virginia, July 2007.

[16]Galtier, J. and George, P. L., "Prepartitioning as a way to mesh subdomains in parallel," *Proceedings of the Fifth International Meshing Roundtable*, 1996, pp. 107–121.

[17]Ivanov, E., Andra, H., and Kudryavtsev, A. N., "Domain Decomposition approach for Automatic Parallel Generation of Tetrahedral Grids," *Computational Mehods in Applied Mathematics*, Vol. 6, No. 2, 2006, pp. 178–193.

[18]Larwood, B. G., Weatherill, N. P., Hassan, O., and Morgan, K., "Domain Decompositionn approach for Parallel Unstructured Mesh Generation," *International Journal for Numerical Methods in Engineering*, Vol. 58, 2003, pp. 177–188.

[19]Pirzadeh, S., "Recent Progress in Unstructured Grid Generation," *American Institute of Aeronautics and Astronautics (AIAA)*, 1992.

[20]Pirzadeh, S., "Three-Dimensional Unstructured Viscous Grids by the Advancing-Layers Method," *American Institute of Aeronautics and Astronautics (AIAA)*, Vol. 34, No. 1, January 1996, pp. 43–49.

[21]Pirzadeh, S., "Domain Decomposition by the Advancing-Partition Method," Tech. Rep. NASA/TM-2008-215350, NASA Langley Research Center, September 2008.

[22]Pirzadeh, S. and Zagaris, G., "Domain Decomposition by the Advancing-Partition Method for Parallel Unstructured Grid Generation," *Proceedings of the 47th American Institute of Aeronautics and Astronautics (AIAA) Aerospace Sciences Meeting*, January 2009, Orlando, Florida (submitted for publication).

[23]Lohner, R. and Parikh, P., "Generation of Three-Dimensional Unstructured Grids by the Advancing Front Method," *Proceedings of the 26th American Institute of Aeronautics and Astronautics (AIAA) Aerospace Sciences Meeting*, 1988, Reno, NV.

[24]Parikh, P., Pirzadeh, S., and Lohner, R., "A package for 3-D unstructured grid generation, finite-element flow and flow field visualization," NASA Technical Report NASA-CR-182090, September 1990.

[25]Mattson, T. G., Sanders, B. A., and Massingill, B. L., *Patterns for Parallel Programming*, Addison-Wesley, 2004.

[26]Stroustrup, B., *The C++ Programming Language*, Addison-Wesley, 1997.

[27]Gropp, W., Lusk, E., and Skjellum, A., *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, MIT press, 1999.

[28]Samareh, J., "GridTool: A surface modelling and grid generation tool," *Proceedings of the Workshop on Surface Modelling, Grid Generation and Related Issues in CFD Solutions*, 9-11 May 1995.

[29]Pirzadeh, S., "Structured Background Grids for Generation of Unstructured Grids by Advancing Front Method," *Americal Institute of Aeronautics and Astronautics (AIAA)*, Vol. 31, No. 2, 1993, pp. 257–265.

[30]Pirzadeh, S., "Advanced Unstructured Grid Generation for Complex Aerodynamics Applications," *Proceedings of the 26th American Institute of Aeronautics and Astronautics (AIAA) Applied Aerodynamics Meeting*, August 2008.

[31] "TeraGrid Cluster," http://www.teragrid.org.

[32] "National Center for Supercomputing Applications," http://www.ncsa.uiuc.edu/.