42ND AEROSPACE SCIENCES MEETING AND EXHIBIT
RENO NEVADA, 5-8 JANUARY 2004

AIAA-2004-1232

Applications of Space-Filling Curves to Cartesian Methods for CFD

M. J. Aftosmis[†]
Mail Stop T27B
NASA Ames Research Center
Moffett Field, CA 94404
aftosmis@nas.nasa.gov

M. J. Berger[‡]
Courant Institute
251 Mercer St.
New York, NY 10012
berger@cims.nyu.edu

S. M. Murman[¥]
ELORET
NASA Ames Research Center
Moffett Field, CA 94404
smurman@nas.nasa.gov

This paper presents a variety of novel uses of space-filling curves (SFCs) for Cartesian mesh methods in CFD. While these techniques will be demonstrated using non-body-fitted Cartesian meshes, many are applicable on general body-fitted meshes – both structured and unstructured. We demonstrate the use of single $\mathcal{O}(N \log N)$ SFC-based reordering to produce single-pass ($\mathcal{O}(N)$) algorithms for mesh partitioning, multigrid coarsening, and inter-mesh interpolation. The intermesh interpolation operator has many practical applications including "warm starts" on modified geometry, or as an inter-grid transfer operator on remeshed regions in moving-body simulations. Exploiting the compact construction of these operators, we further show that these algorithms are highly amenable to parallelization. Examples using the SFC-based mesh partitioner show nearly linear speedup to 640 CPUs even when using multigrid as a smoother. Partition statistics are presented showing that the SFC partitions are, on-average, within 15% of ideal even with only around 50,000 cells in each subdomain. The inter-mesh interpolation operator also has linear asymptotic complexity and can be used to map a solution with N unknowns to another mesh with M unknowns with $\mathcal{O}(M + N)$ operations. This capability is demonstrated both on moving-body simulations and in mapping solutions to perturbed meshes for control surface deflection or finite-difference-based gradient design methods.

1 Introduction

THE literature on numerical methods for Partial Differential Equations (PDEs) contains a wealth of diverse approaches for improving cache performance, performing domain decomposition, mesh coarsening and inter-mesh solution transfer. The infrastructure and algorithms supporting these operations are frequently unrelated and usually require different data structures and infrastructure to perform each task. For example, a high-performance, distributed-memory, unstructured mesh code may use Reverse Cuthill-McKee (RCM) ordering for improving cache-performance, recursive spectral bisection, or multi-level nested dissection for domain decomposition,[1][2] a graph-based mesh coarsener[4][3] and variety of fast spatial search data structures for inter-mesh interpolation, or solution transfer to re-gridded regions of a subdomain.

For each task, these techniques offer superb performance, and many of them are amenable to some degree of parallelization. Nevertheless, development and maintenance of such a diverse collection of tools requires considerable investment, and when a substantial overhaul is necessary (*e.g.* moving a code to distributed-memory parallelization) the level of effort required will be significant.

Space-filling curves (SFCs) offer a unifying data structure for all of these ends. Construction of these curves is extremely inexpensive as the SFC index for any voxel in space may be computed using only local information, and thus computation of indices may be performed in parallel. The asymptotic time complexity of constructing the curve is therefore bounded by the sort algorithm which orders the mesh along the curve. This sort can be performed with standard sorting routines such as quicksort which produces a method with $\mathcal{O}(N \log N)$ running time. After this initial sort, all subsequent coarsening, decomposition, and interpolation can may be performed with linear sweeps through the reordered mesh.

2 Space-Filling Curves

The central operation in using space-filling curves is a reordering of the mesh using one of the dozens of well-documented space-filling curves. In this work we consider both the Morton and Peano-Hilbert order[6]. Both orderings have been explored in scientific computing in a variety of roles, including the parallel solution of N-body problems in computational physics[7], algebraic multigrid[8], mesh generation,[9] in the solution of PDEs and dynamic repartitioning of adaptive methods[10]. Figure 1 shows both Peano-Hilbert and Morton SFCs constructed on Cartesian meshes at three levels of refinement. In two dimensions, the basic building block of the Hilbert curves is a "U" shaped line which visits each of 4 cells in a 2 x 2 block. Each subsequent level further divides the previous level's cells, creating subquadrants which are, themselves, visited by (rotated) U shaped curves as well. Similar properties exist for the Morton ordering which uses an "N" shaped curve as its basic building block.

In three spatial dimensions the curves follow the same basic construction rules, but the basic building block extends into the third dimension with additional U or N-shaped turns. Figure 2 illustrates this 3-D construction for the U-order showing: (a) the basic 2x2x2 building block (b) the same mesh after one uniform refinement, where each segment of the curve is replaced with an appropriately rotated basic building

[†] Research Scientist, Senior Member AIAA
[‡] Professor, Member AIAA
[¥] Senior Research Scientist, Member AIAA.

This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

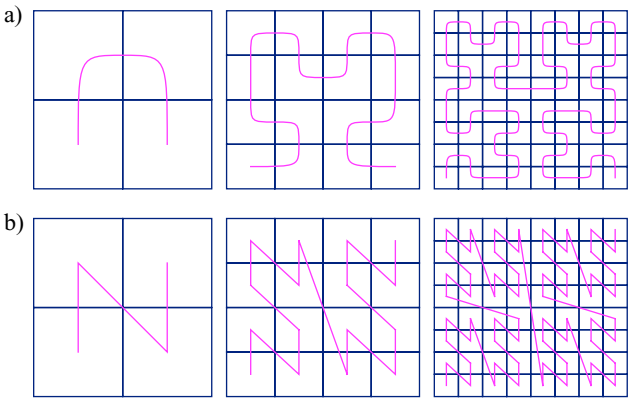


Figure 1: Space-filling curves used to order three Cartesian meshes in two spatial dimensions: a) Peano-Hilbert or "U-ordering", b) Morton or "N-ordering".

block, (c) the curve after additional adaptive refinement of cells near the back-south-west corner. Properties and d-dimensional construction rules for these space-filling curves are discussed extensively in refs. [11], [12] and [13].

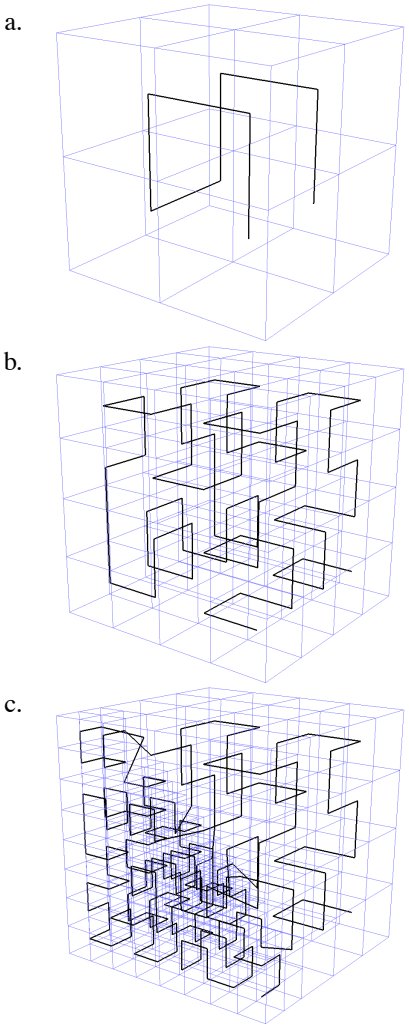


Figure 2: U-order in three dimensions for (a) a basic 2x2x2 block of cells, (b) the same block after uniform subdivision (c) cell order after refinement near the south-west-back cor-

Both orderings have locality properties which make them attractive as mesh partitioners[10][9]. For the present, we note only that such orderings have 3 important properties.

1. **Mapping** $M : \mathcal{R}^d \rightarrow \boldsymbol{U}$: The U and N orderings each provide unique mappings from the d-dimensional physical space of the problem domain $\mathcal{R}^d$ to a one-dimensional hyperspace, ***U,*** which one traverses following the curve.
2. **Locality**: In the U-order, each cell visited by the curve is directly connected to two face-neighboring cells which remain face-neighbors in the one dimensional hyperspace spanned by the curve. Locality in N-ordered domains is almost as good[6].
3. **Compact Construction**: Encoding and decoding the Hilbert or Morton order requires only local information. Following the integer indexing for Cartesian meshes outlined in ref. [5], a cell's 1-D index in ***U*** may be constructed using only that cell's integer coordinates in $\mathcal{R}^d$ and the maximum number of refinements that exist in the mesh. This aspect is in marked contrast to other partitioning schemes based on recursive spectral bisection or other multilevel decomposition approaches which require the entire connectivity matrix of the mesh in order to perform the partitioning.

To illustrate the compact construction rules for these orderings, consider the position of a cell i in the N-order. One way to construct this mapping would be from a global operation such as a recursive lexicographic ordering of all cells in the domain. Such a construction would not satisfy the property of *compactness*. Instead, the index of i in the N-order may be deduced solely by inspection of cell i's integer coordinates (x_i, y_i, z_i).

Assume (x_i, y_i, z_i) is the bitwise representation of the integer coordinates (x_i, y_i, z_i) using m-bit integers. The bit sequence $\{x_i^1 y_i^1 z_i^1\}$ denotes a 3-bit integer constructed by interleaving the first bit of x_i, y_i and z_i. One can then immediately compute cell i's location in ***U*** as the $3m$-bit integer $\{x_i^1 y_i^1 z_i^1 x_i^2 y_i^2 z_i^2 \ldots x_i^m y_i^m z_i^m\}$. Thus, simply by inspection of a cell's integer coordinates, we are able to directly calculate its

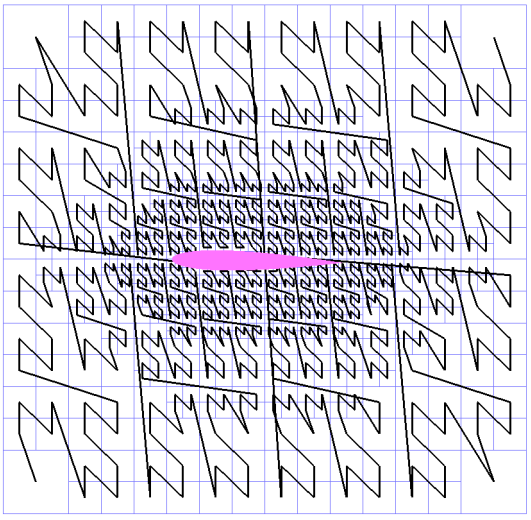


Figure 3: Morton order of an adaptively refined Cartesian mesh around a 2-D airfoil.

location, $M(i)$, in the one-dimensional space $\boldsymbol{U}$ without any additional information. Similarly compact construction rules exist for the U-order[13].

In an h-refined Cartesian mesh, the finest cell can be used to define the dimensions of a single voxel[1]. All coarser cells may then be reinterpreted as collections of this basic building block and are referred to by the index of their lowest constituent voxel. This interpretation leads to an integer indexing scheme which can be used to address all cells in the computational domain. With this integer indexing scheme, the construction rules in the previous paragraph can then be applied to generate the Morton index, $M(i)$, of all the cells in the mesh. Figure 3 shows an example of the N-ordering on an adaptively refined mesh around a 2-D airfoil.

Construction of the Peano-Hilbert index, $H(i)$, follows a similarly compact procedure. After computing the SFC indices $M(i)$ or $H(i)$ for all the cells in the mesh, one simply takes these indices as sort keys and applies any one of the standard sorting algorithms (we use the quicksort algorithm from the C standard library). Since all the other data required for construction is local, the sort establishes the asymptotic bound for runtime of the reordering and choosing quicksort gives typical runtimes of $\mathcal{O}(N \log N)$. In more concrete terms, computing $M(i)$ and $H(i)$ and then sorting cells and faces takes under 4 seconds per million cells on a 2Ghz Pentium 4.

3 Mesh Coarsening

The recursive nature of the SFC ordering makes it a natural choice for a mesh coarsener for h-refined meshes. In the same way that higher-order SFCs are generated by replacing segments with the basic U or N building block, the children produced by h-refinement replace their parent cell in the mesh. Since these children are sorted according to the local SFC they will all be nearest-neighbors on a contiguous segment of the SFC in the space of the curve U.

Figure 4 illustrates this observation in two-dimensions. Figure 4a shows the baseline multilevel mesh as it comes out of either the mesh generator or adaptation module. Frames (b) and (c) show the same mesh after two successive coarsenings. Cells in these meshes are ordered using the N-ordering, and their indices in the SFC order are shown. In fig. 4a cells 22, 23, 24, and 25 all lie adjacent to one another in the 1-dimensional hyperspace of the curve, U.

The coarsening algorithm proceeds with a single traversal of U with a running index i and testing if the cell at $U(i)$ is contained within the same parent as the cell at $U(i+1)$. When a contiguous group of cells are found that all coarsen into the same parent cell they are agglomerated into that parent. If any of the siblings in a contiguous set are further subdivided, then all siblings of the set are "not coarsenable" and get injected into the coarse mesh without modification.

The first coarsening of the mesh in fig. 4a produces the mesh in fig. 4b. Cells 22-25 are coarsened in this first pass to produce cell 10 on the mesh in frame (b). Note that while cells 20, 21 and 26 are all children of the same parent, they are not coarsenable since 22-25 are one level finer. The next coarsening pass produces the mesh in frame (c). Since cells 8-11 on mesh (b) were all siblings, they now coarsen to produce cell 5 on the mesh shown in frame (c).

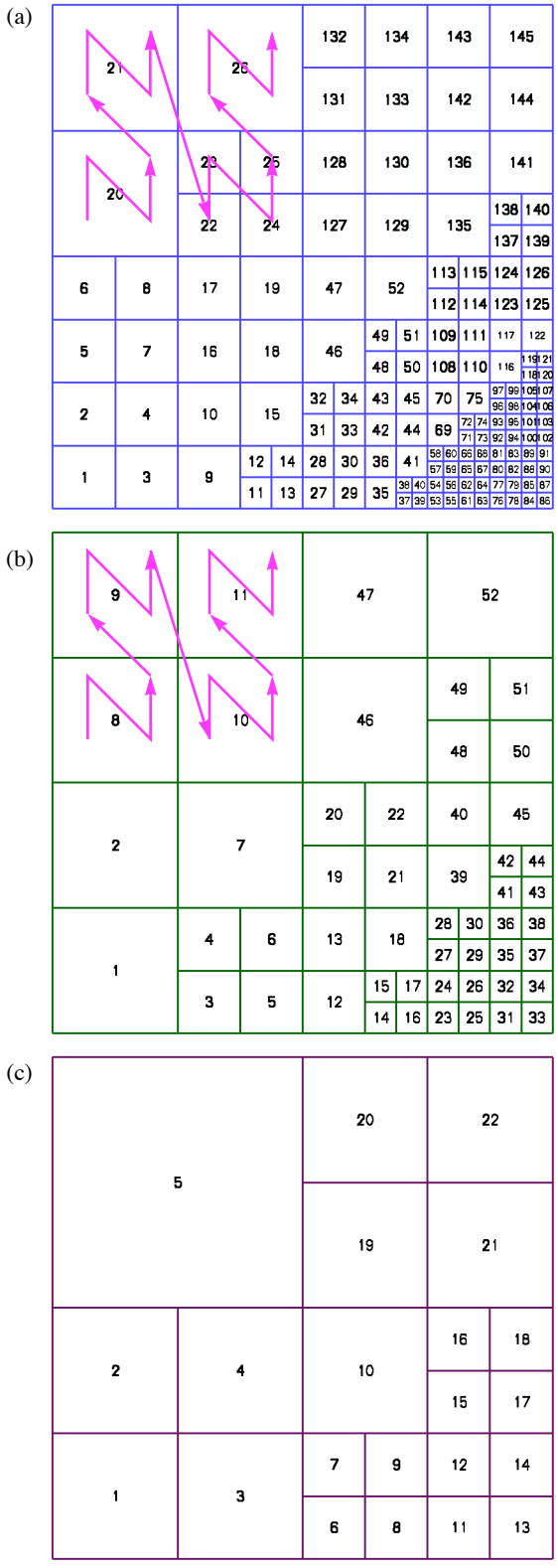


Figure 4: Mesh coarsening example using Morton ordering. The fine mesh in (a) is coarsened 2 times using the same SFC.

1. By analogy with "pixel", the smallest distinguishable box-shaped volume of 3-D space.

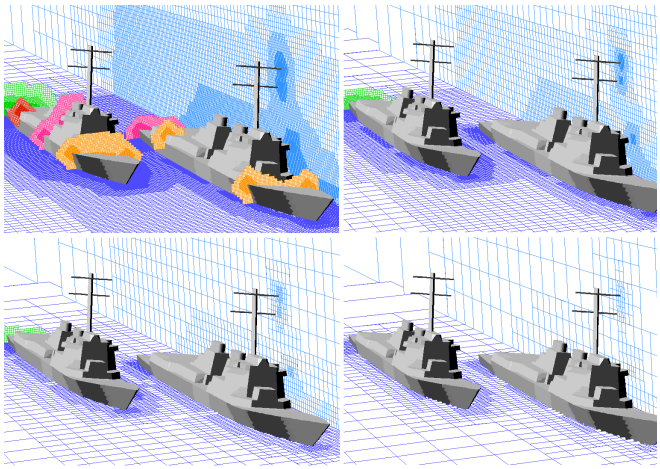


Figure 5: Mesh coarsening example in 3D using agglomeration along the SFC. The finest mesh contains 4.5M cells while the coarsest contains 4500 cells after 4 coarsenings.

Figure 5 shows an example in three dimensions. In this example, a 4.5M cell adaptively-refined mesh around two surface ships has undergone 4 cycles of coarsening by agglomeration along the SFC. The final mesh shown in the lower right frame of the figure contains 4500 cells. In practice, typical coarsening ratios for the algorithm are in excess of 7 on realistically complex problems.[16]

4 Domain decomposition

The mapping and locality properties that are exploited for the single-pass mesh coarsener described above make SFCs a natural choice for partitioners on hierarchical meshes.[8][9][15][16] Figure 6 illustrates these mapping and locality properties for an adapted two-dimensional Cartesian mesh, partitioned into three subdomains. The figure points out that for adapted Cartesian meshes, the hyperspace $\boldsymbol{U}$ may not be fully populated by cells in the mesh.

The quality of the partitioning resulting from U-ordered meshes have been examined in ref. [10] and were found to be

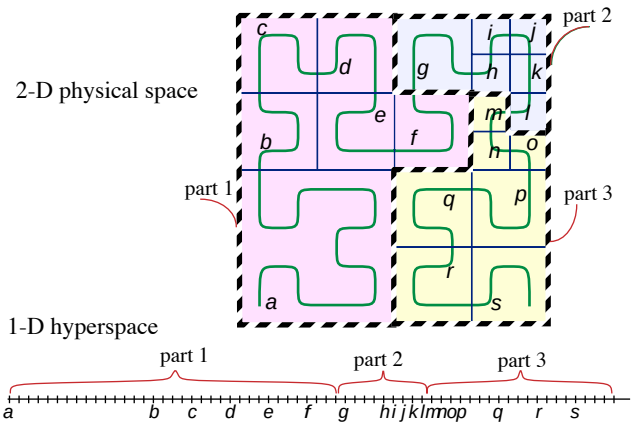


Figure 6: An adapted Cartesian mesh and associated space-filling curve based on the U-ordering of $M : \mathcal{R}^d \rightarrow \boldsymbol{U}$ with the U-ordering illustrating locality and mesh partitioning in two spatial dimensions. Partitions are indicated by the heavy dashed lines in the sketch.

competitive with respect to other popular partitioners. Weights can be assigned on a cell-by-cell basis. One advantage of using this partitioning strategy stems from the observation that mesh refinement or coarsening simply increases or decreases the population of $\boldsymbol{U}$ while leaving the relative order of elements away from the adaptation unchanged. Remapping the new mesh into new subdomains therefore only moves data at partition boundaries and avoids global remappings when cells adaptively refine during mesh adaptation. Recent experience with a variety of global repartitioners suggest that the communication required to conduct this remapping can be an order of magnitude more expensive than the repartitioning itself[14]. Additionally, since the partitioner just inserts breaks into the U-ordered cell list on-the-fly, the entire mesh may be stored as a single domain. At run-time, this single mesh may then be partitioned into any number of subdomains as it is read into the flow solver from mass storage. One benefit of this approach is that a simulation begun on some given number of CPUs may be restarted on a different number of CPUs. Alternatively, when performing parameter studies on a fixed mesh, each simulation may be run on a different number of CPUs - all sharing the same copy of the input mesh file. In a heterogeneous shared timesharing environment, where the number of available processors may not be known at the time of job submission, the value of such flexibility is obvious.

The SFC reordering pays additional dividends in cache-performance. The locality property of the SFC produces a connectivity matrix which is tightly clustered regardless of the number of subdomains. Our numerical experiments suggest that SFC reordered meshes have about the same data cache hit-rate as those reordered using reverse Cuthill-McKee.

Figure 7 shows an example of a three dimensional Cartesian mesh around the full Space Shuttle Launch Vehicle (SSLV) configuration. This complex configuration includes the orbiter, external tank, solid rocket boosters, and fore and aft attach hardware. The computational mesh contains about 4.7M cells at 14 levels of refinement, and is indicated by a single cutting plane passed through the mesh just behind the SSLV geometry. The coloring of gridlines in the mesh shows its partitioning into 16 subdomains using the U-order. Reordering this mesh with the algorithm of §2 required 20 sec. on a 2 Ghz Pentium 4, and preparing 4 levels of coarser meshes for multigrid required 15 sec. on the same machine. Partition boundaries are chosen to balance the load in each subdomain. In determining work loads, cut-cells were weighted 2.1x as compared to un-cut Cartesian hexahedra.

The partitioning in fig. 7 demonstrates how, even on adaptively-refined meshes, the partitioning tends to produce subdomains that are largely rectilinear. On a uniform mesh, an appropriately chosen number of partitions would result in a cubical decomposition of the computational domain. This "best case" establishes the minimum ratio of communication to computation (surface/volume ratio) for SFC partitioned meshes. For any given number of cells, one can conceive of an idealized cubical partitioner which would communicate across some number of faces, F_c given by:

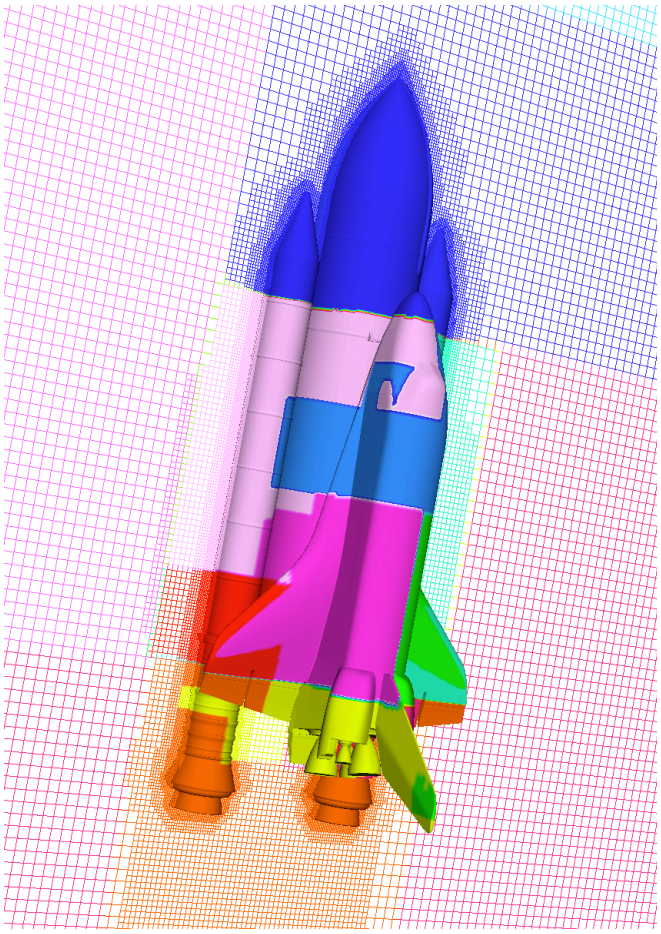


Figure 7: 4.7M cell mesh around full SSLV configuration including orbiter, external tank, solid rocket boosters, and fore and aft attach hardware. Mesh color indicates partitioning (16 partitions, U-ordering).

$$F_c = 6P\left(\frac{N}{P}\right)^{2/3} - 6N^{2/3} \quad (1)$$

where N is the total number of cells in the domain and P is the number of subdomains. The first term on the right is the surface area of all the subdomains, while the second term reduces this estimate by the surface area of the whole mesh since no data is exchanged there. Real hierarchical meshes coarsen rapidly away from geometry and have very few faces on the domain boundary, thus the estimate of F_c provided by eq. (1) is a reasonable lower bound for evaluating the quality of the partitioning provided by the SFC-based schemes.

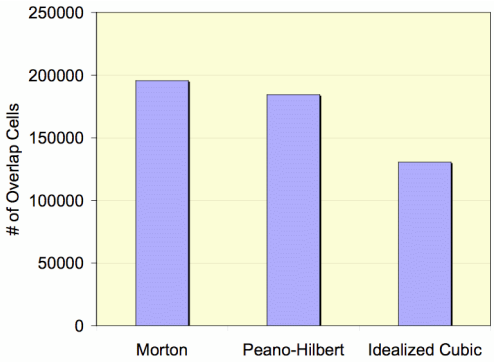


Figure 8: Partitioning statistics for a 1M cell adapted cartesian mesh decomposed into 32 subdomains with two different SFCs compared with the idealized cubical partitioner described in the text.

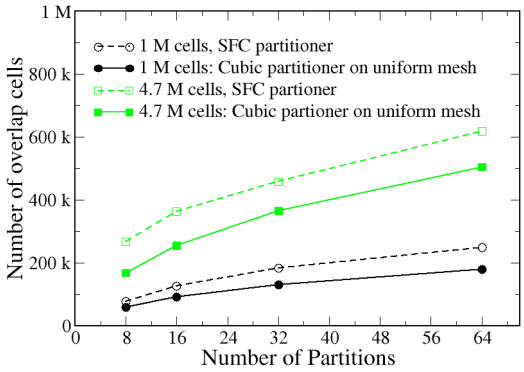


Figure 9: Comparison of subdomain overlap of two different adaptively refined meshes with an idealized cubic partitioner with varying numbers of subdomains.

Figure 8 examines the average partition statistics for a 1M cell adaptively-refined mesh decomposed into 32 subdomains. Even with this large number of subdomains on a relatively small mesh, this figure shows that both SFC partitioners perform well.

Figure 9 expands upon this example to show how the partition quality changes with mesh size and number of partitions. Results are shown for both the 1M cell mesh of fig. 8 and that of fig. 7 with 4.7M cells. In both cases the results from the actual Hilbert partitioning track the surface-to-volume of the idealized cubic partitioner reasonably well. Results in the following sections will show that this partitioning sufficiently minimizes communication to offer near ideal speedups on very large numbers of processors.

All meshes in the multigrid hierarchy are partitioned using the same approach. Since each coarse mesh is produced following the SFC on its finer counterpart, these are automatically generated sorted into SFC order. After partitioning the fine mesh the stack of coarse meshes are read in one-at-a-time and each is partitioned on-the-fly using the same load-balancing approach.

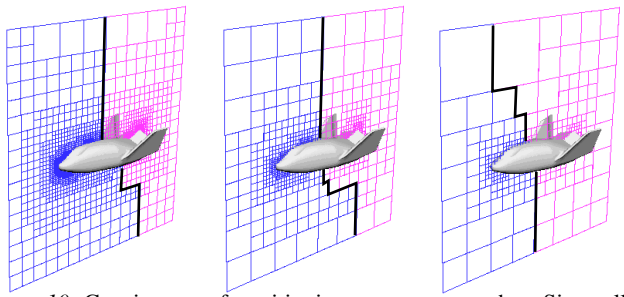


Figure 10: Consistency of partitioning on coarser meshes. Since all meshes in the multigrid hierarchy are partitioned with the same SFC, multigrid restriction and prolongation operators communicate primarily fine/coarse cells on their same processor.

# Partitions	*Avg. # fine cells*	*% Restrict to different partition*
8	593824	7.9%
16	296912	12.2%
32	148456	19.3%
64	74228	32.7%
128	37114	57.2%

Table 1: Partition overlap statistics for the SSLV mesh in fig. 7.

In most graph-based approaches to mesh partitioning, repartitioning the coarse mesh offers no guarantee of good overlap between coarse and fine partitions sitting on any given processor. Since the intergrid transfer operators in multigrid introduce communication between *every cell* in successive meshes in the hierarchy, good overlap limits the amount of off-processor bandwidth required for intergrid transfer. With SFC partitioned meshes, all meshes are being partitioned by exactly the same SFC, uniformly coarsened meshes would have maximal overlap with their finer counterparts. In practice, the coarsening is modified by refinement boundaries and coarsening rules, but uniform coarsening remains a good model. Figure 10 demonstrates this by showing the coarsening of a multilevel Cartesian mesh around a reentry vehicle. For simplicity, the mesh is shown partitioned into 2 subdomains. The subdomain boundaries in this example show good consistency of partitioning and 96% of the cells on the finest mesh restrict to coarser cells residing on the same processor. Only 4% of the communication for intergrid transfer needs to go between processors.

Table 1 extends these observations to larger numbers of processors. The table contains partition overlap statistics for the 4.7M cell mesh around the SSLV example in fig. 7. Overlap statistics are included for 8 to 128 partitions, and the data includes the average percentage of fine cells that restrict to different partitions. With 8 partitions on the mesh, under 8% of the cells need to communicate with different partitions. This percentage grows as the mesh is cut into ever smaller pieces, but at 64 processors nearly two thirds of the cells are still on the same processor as their coarser counterparts. At 128 processors this number has dropped-off somewhat and over half of the cells on the partition need to restrict to partners off-processor. Note, however that by this point, the partitions are *extremely* small. On average, each contains only 37000 cells and requires less than 25 Mb of storage on each processor.

4.1 Scalability and Performance

Scalability tests were conducted using the 4.7M cell mesh from fig. 7. Flow conditions for the test had the SSLV at $M_\infty = 2.6$, $\alpha = 2.09°$, $\beta = 0.8°$. Isobars of the discrete solution are shown in fig. 11. The inviscid multilevel solver from [15] and [16] was tested using both OpenMP and MPI communication libraries for both single and multigrid runs.

Figure 12 shows scalability results for this case on an SGI Origin 3000 (600Mhz, Mips R14000 CPUs) without multigrid. Results are presented using both OpenMP and MPI for off-processor communication. The MPI code was compiled using the SGI's native MPI library. Performance is linear across the entire experiment with parallel speedups of 599 and 486 for the OpenMP and MPI codes (respectively) on 640 CPUs. On this many CPUs the even a 4.7M cell mesh has only about 7350 cells per processor, thus the partitions are extremely small. Despite this very fine partitioning, results with both communication protocols are performing well, and the SFC partition quality (cf. fig. 8) is more than

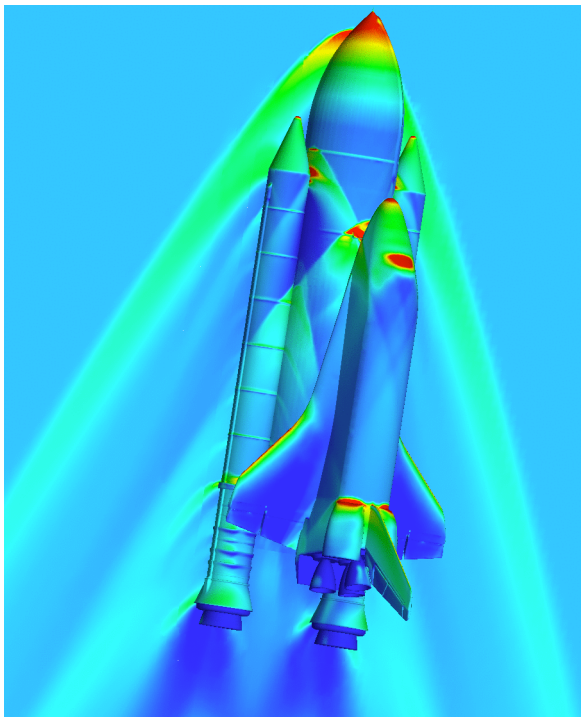


Figure 11: Isobars in discrete solution for SSLV configuration at $M_\infty = 2.6$, $\alpha = 2.09°$, $\beta = 0.8°$. This case was used for scalability results.

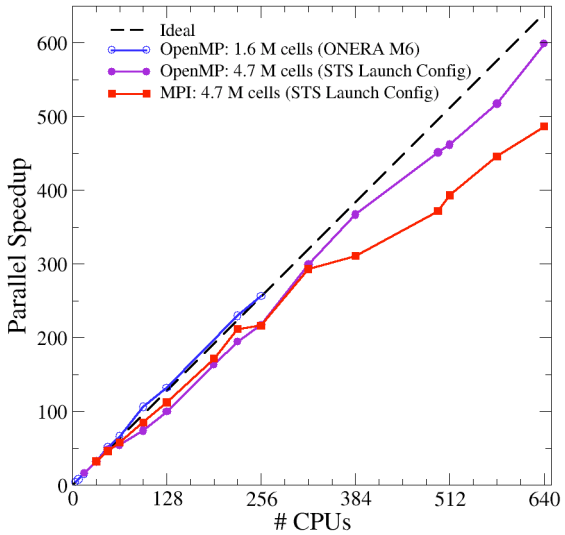


Figure 12: Parallel scalability of single grid scheme on 4.7M cell SSLV test case on SGI Origin 3000. Results are displayed using both OpenMP or MPI for communication

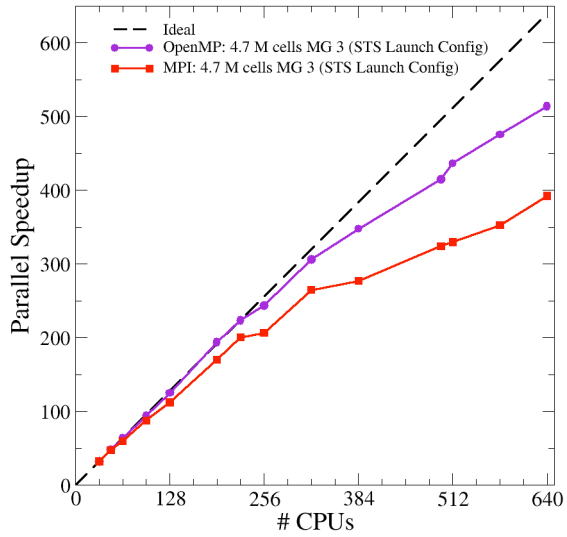


Figure 13: Parallel scalability of 3-level multigrid scheme on 4.7M cell SSLV test case on SGI Origin 3000. Results are displayed using both OpenMP or MPI for communication

adequate on this computing platform. To reiterate this point, the plot also shows results for a 1.6M cell mesh distributed onto as many as 256 processors (~6250 cells/partition). This curve lays on top of the "ideal" curve over its entirety. The slight performance advantage shown by OpenMP is probably due to the additional time required in the MPI code to explicitly pack and unpack the messages passed between partitions. All scalability runs were conducted with the machine in non-dedicated mode (normal queuing system), and its expected that some of the irregularity in the curves would improve under dedicated usage.

Figure 13 shows a similar scalability study using 3-level multigrid (W-cycles) using the same test cases. The additional communication load due to off-processor intergrid transfer pointed to by Table 1 starts to become apparent on large numbers of processors. Nevertheless results are still quite good. The OpenMP code achieves parallel speedups of about 514 on 640 CPUs while the MPI version shows about 392. The first and second coarse meshes in this grid hierarchy have 700000 and 105000 cells respectively. These translate into average cell counts of only 1100 and 180 cells per partition.

5 Inter-mesh Interpolation

The problem of efficient transfer of solutions or residuals from one mesh to another comes up frequently in vehicle analysis. In stability and control analysis, for example, one may wish to "warm-start" a solution with a deflected control surface from some undeflected baseline solution. In design using finite-difference-based gradients, it is common practice to compute a baseline configuration and then perturb the shape to establish the gradients with respect to the shape change. Since these perturbations are small, it makes sense to re-use the pre-computed baseline to warm-start the perturbed solutions. In moving-body simulations, when the body has moved, the geometry is no longer the same and we are again in a situation of needing to transfer the solution (and perhaps residual) to a new, but nearby, mesh.

In general, cells in the new mesh do not have a one-to-one correspondence with cells in the old mesh and some sort of three dimensional interpolation is required. At worst, finding interpolants may require searching over all the cells in the mesh, and such brute force algorithms are not practical. With N cells in the old mesh and M cells in the new mesh, any algorithm that runs in $\mathcal{O}(N\,M)$ time is sure to be expensive when both N and M are 10^7 or 10^8.

Even with parallelization and more sophisticated algorithms, inter-mesh transfer can be expensive. For example, Ref. [17] reports that finding interpolation coefficients on a 324000 cell mesh required 200 seconds on 16 CPUs. Other approaches use binning, hashing or a variety of spatial tree structures to limit the exhaustive searching needed for finding interpolation stencils[18][19]. Since these data structures are created for the interpolation, their construction cost (typically $\mathcal{O}(N \log N)$) must be included in the interpolation time.

SFCs offer an attractive alternative for performing this transfer. The key to this use is to recognize that for any Cartesian meshes with the same bounding box and number of coarse divisions, a single SFC describes all possible meshes covering the space. Essentially the SFC is playing the same role as a spatial data structure that spatial trees or coordinate bisection play in other approaches. An important difference is that rather than sorting into bins or traversing trees, the SFC sorts the meshes into a unique order visiting each cell only once in each mesh. With both meshes sorted, mapping from one to another becomes a simple task of walking down the SFC through both meshes in "lock step", marking time in one or the other as needed to accommodate nested refinement or boundaries.

Figure 14 provides a more concrete view of this using an example where we wish to map the data from the red mesh (left) to the blue mesh (right). Both meshes are visited by the same SFC. In the sketch, each cell is annotated with its SFC index showing the order in which the mapping algorithm sweeps the mesh. Cells 1-4$_{red}$ and 1-4$_{blue}$ are identical (same SFC index, and same size), so the current position counters for each of the two meshes will get incremented symmetrically as we progress through these cells. Entering cell 5, we note that 5$_{blue}$ is larger than 5$_{red}$ so the counter on the blue

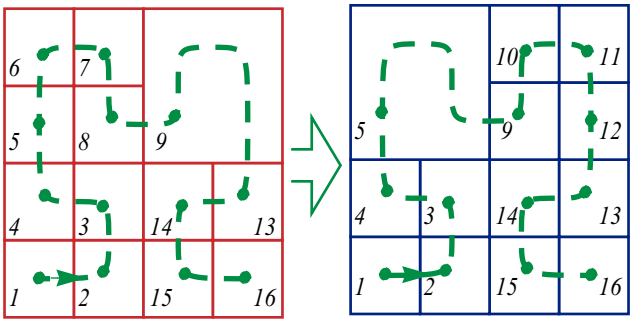


Figure 14: SFC used to map data from the red mesh (left) to the blue mesh (right).

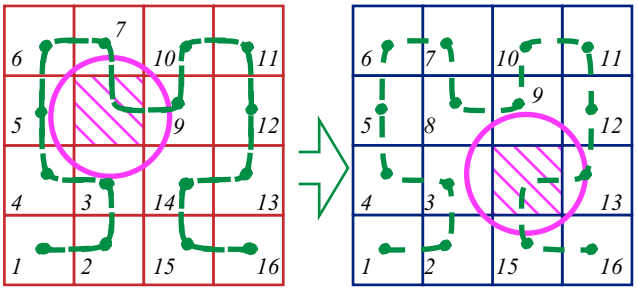


Figure 15: SFC used to map data from the red mesh (left) to blue mesh (right) with internal geometry.

mesh will not get incremented until it encounters the next cell on the red mesh that is not contained. In this way we simply mark time on the blue mesh until we encounter 9_{red} which is the first cell not contained by 5_{blue}. Data in $5\text{-}8_{red}$ all get mapped to 5_{blue}. 9_{red} is the first uncontained cell and we immediately notice that 9_{blue} is smaller than 9_{red}. This time its the red counter that gets suspended while cells $9\text{-}12_{blue}$ all receive a prolongation of the data in 9_{red}. Cells 13-16 match one-to-one in both meshes and are filled by direct injection.

From this sketch of the algorithm several things are clear: (1) Counters in both meshes increment monotonically. (2) At termination, both counters reach their respective maximums. (3) Every step of the algorithm increments counters in either the red, blue or both meshes. Thus if there are N cells in the red mesh, and there are M cells in the blue mesh, the algorithm has a run-time complexity of $\mathcal{O}(N+M)$, provided that the red and blue meshes are already in SFC order. If the meshes are not already sorted, the operation is bounded by the $\mathcal{O}(N \log N)$ complexity of the reordering.

Figure 15 outlines the situation when the red or blue meshes contain internal geometry. In this situation, cells that are completely internal to the geometry do not appear in the meshes. This leaves a gap in the SFC indexing on the mesh not covered by the extent of any cell. In the figure, cells 1-7 map one-to-one but there is a gap between 7_{red} and 9_{red} not cover by the size of 7_{red}. Thus no map exists for 8_{blue}. Cells 9-11 map one-to-one, but 12_{red} has no counterpart in the blue mesh. Since we are interested in generating a mapping from blue to red, 8_{blue} gets assigned a *nomap* flag, and 12_{red} simply gets skipped.

Figure 16 contains *Algorithm M*. with a detailed description of the full mapping algorithm handling both mesh refinement and changes in geometry. This algorithm is couched as a driver loop over the blue mesh which recursively calls a mapping function that increments counters in the red and blue mesh. Since it visits each cell in the red and blue meshes once each, *Alg. M.* has run-time complexity of $\mathcal{O}(N+M)$ which is linear in the total number of cells on the red and blue meshes - provided that both meshes are already sorted in SFC order. The algorithm makes use of the same prolongation and restriction operators as used by the multigrid scheme. Direct injection is used for prolongation of the flow state, while volume weighted averages are used for restriction. Since the majority of the cells map either 1:1 or, as a coarsening/refinement, the tail-recursion in *M4.1.1* is rarely exercised. The

driver loops on blue mesh

```
next_red = end_red = 0;
foreach hex in blue mesh{
  blue2red[this_blue] = getMap(next_red, end_red);
  next_red = end_red
}
getMap(this_red, end_red){
  end_red = this_red;

1. if sameHex(this_blue, this_red)              1:1 mapping
  1.1 map = INJECT from this_red to this_blue;
  1.2 increment end_red;

2. if (this_blue within this_red)      1:N mapping, several
  2.1 map = PROLONG from red to blueblues map to same red
  2.2 DO NOT increment end_red

3. if (this_red within this_blue)      N:1 mapping, several
                                              reds map to same blue
  3.1 scan forward in red:
      while (end_red within this_blue) increment end_red;
  3.2 map = RESTRICT from this_red to end_red into blue

4. Red and blue cells don't overlap, must be a gap in SFC on one
   mesh or other, compare SFC index of both cells

  4.1 if SFC(this_red) < SFC(this_blue)          gap in blue
    4.1.1 Increment start in red and call getMap
          getMap(this_red+1,this_blue)
  4.2 if SFC(this_red) > SFC(this_blue)          gap in red
    4.2.1 Corresponding cell didn't exist in red mesh
          map = NO_MAP
return map;
}
```

Figure 16: Algorithm *M*: *Given blue and red meshes presorted in SFC order, create blue-to-red mapping,*

current algorithm simplifies the implementation at a nominal run-time cost. When the blue mesh encounters a region for which there is no mapping (due to a hole, or "solid" region, in the red mesh) it gets a *NO_MAP* flag. When used for restarts, these *nomaps* are populated with freestream quantities. In the case of time-dependent moving-body simulations, *nomaps* indicate a cell that was uncovered by the motion of the geometry over the timestep. In this case, the state vector is set to zero, and the cell is filled by the space-time fluxes in the moving-body scheme.

Since it is a linear-time procedure, Algorithm *M* typically runs in seconds, even on meshes containing several million cells. Timing examples were run on a 2 Ghz Pentium 4 desktop machine using meshes similar to the 4.7M SSLV case used above. In these, tracing the space-filling curve with Algorithm *M* takes about 0.5 seconds, and actually transferring the state vector using this map takes about 1.5 seconds. Reordering cells and faces using the space-filling curve for such a mesh takes about 20 seconds, however this one-time cost was already paid during construction of the coarse meshes on the receiving grid.

The real utility of the intergrid transfer comes when the geometry is slightly modified and we seek a solution to a nearby problem. Since Cartesian meshes are ridged, they do not distort to follow the moving geometry, but instead result in a mesh with a different, but nearby, sets of cut, volume and interior cells. Examples of this exist in design, control surface deflection, and moving body simulations.

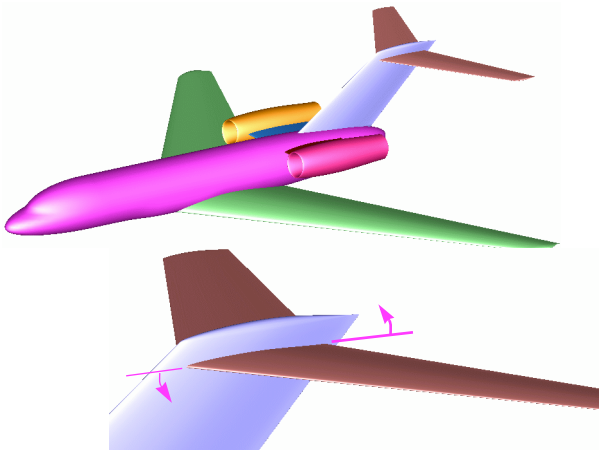


Figure 17: Transonic business jet example with T-tail, pylons and nacelles. Horizontal tail is shown in baseline position and direction of 2° deflection is indicated by the arrows.

The first example considers deflection of the T-tail on a transonic business jet. Figure 17 shows a generic business jet with T-tail, wings, pylons and nacelles. After computing the baseline configuration, the movable horizontal tail is deflected 2° to increase the nose-up pitching moment. The convergence plot in Figure 18 monitors forces and the L_1 norm of density in the discrete solution. At $M_\infty = 0.72$ and $\alpha = 2.8°$, the baseline configuration converges by 150 (3-level) multigrid cycles with about 6 orders of magnitude in the L_1 norm on a mesh with about 1.1M cells. This simulation was run using full-multigrid startup, so the coarse grid iterations are extremely fast. In this simulation, the lift vector is almost aligned with the *y*-axis, and inspection of this component of the integrated force vector shows that it stabilizes after 40-50 cycles on the fine mesh. Convergence continues until about 150 cycles.

The close-up of the tail in fig. 17 shows the deflection of the tail about its hinge. The tail was deflected 2° followed by a regeneration of a new volume mesh, a reordering and coarse

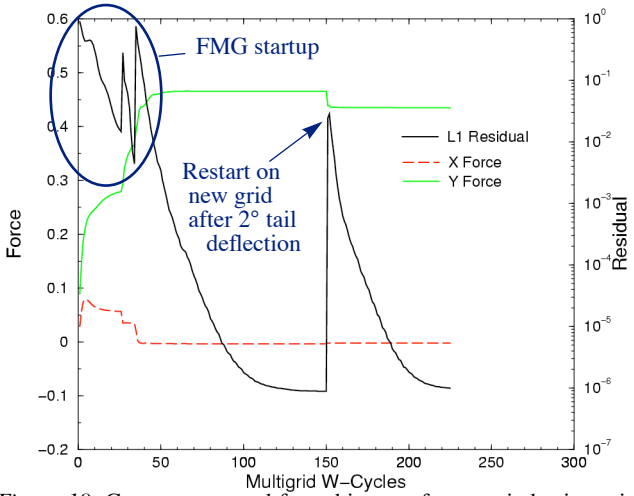


Figure 18: Convergence and force history of transonic business jet example at Mach 0.72, α 2.8°. The plot shows convergence of the baseline configuration as well as warm-start on the 2° deflected mesh after solution transfer.

mesh generation. Total time for this process was under 1 minute on a 2Ghz Pentium 4. The solution from the undeflected case was mapped to the new mesh and warm-started by the solver. Performance of this restart is tracked by the second half of the convergence plot (see annotation in fig. 18). Upon restart, the force vector changes to its new value by the end of the first multigrid W-cycle. Forces are converged to 3 digits after only 7 cycles and 4 digits after 40 cycles on the new mesh.

Despite the presence of substantial geometry in the flow, the baseline full multigrid solution for this case shows very rapid convergence. Nevertheless the solution transfer and subsequent warm start offers an improvement. Since Cartesian meshes of the baseline and deflected configurations are identical away from the geometry change and the meshes contain roughly the same characteristics, residual levels on the restarted mesh are directly comparable with those on the baseline grid. The warm-started solution reached 10^{-4} in one half the wall-clock time required by the baseline solution to reach this point (including the time for FMG startup). These results are typical for warm starts, and they usually offer savings on the order of 1.5-5 on configurations of realistic complexity.[20][21]

6 Moving Body Simulations

As shown in the preceding section, solution mapping is a convenience in design or in configuration studies. However in moving-body simulations, the mapping of the solution (and perhaps residual) at one time level to a new geometry and mesh at the new time level is a necessity since this is the only way the flow's history gets communicated through the simulation. The moving-boundary method of Ref. [22] requires that at each implicit timestep, the geometry be moved, and the Cartesian mesh be re-adapted and recut to the updated geometry. The simulations are then advanced over the next implicit iteration using a parallel multigrid scheme much like that discussed in the present work.[22] Such simulations use SFCs not only for the solution transfer, but also for the domain decomposition and coarse mesh generation. This strategy exercises all uses of the SFCs outlined in the present paper.

An example of much recent interest comes from the unfortunate loss of the STS-107 orbiter and crew on 1 Feb. 2003. In this case, foam debris from the forward attach hardware (the bipod ramp) was released at 81.7 seconds Mission Elapsed Time (MET) at an altitude of 65,600 feet. Technically, this case is interesting since an object measuring only inches across is being transported around 80 feet to its impact location. Conditions were Mach 2.46, $\alpha = 2.08°$, $\beta = -0.095°$. Figure 19 shows a composite image of isobars in this moving body simulation done with the method of [22].

The mesh is similar to the SSLV mesh shown earlier, but with additional refinement around the bipod, bipod ramps, and foam debris. At the start of the simulation this mesh contained about 4.6M cells. Figure 20 shows the mesh near the symmetry plane and STS-107 Launch Vehicle geometry col-

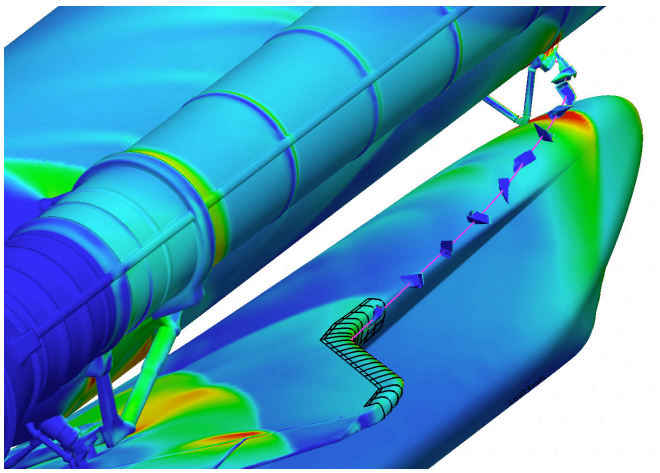


Figure 19: Composite (multiple exposure) of isobars in moving-body simulation of STS-107 debris event at 81.7 seconds MET from Ref.[23], using the Cartesian method of Ref.[22]

ored by mesh partition number. The figure includes frames from four time levels of the simulation. The back-to-back comparison of these provides some useful observations about the behavior of SFC based mesh partitioners.

As alluded to earlier, at each timestep in the moving-body simulation the mesh must be adapted to track the moving geometry. This mesh then undergoes a single reordering using the quicksort algorithm as described in §2. From there, three coarse meshes are produced with the linear-time algorithm of §3, and the solution gets transferred with the single-pass method of §5. Efficient solution with the multilevel method of [16] and [22] demands that the modified mesh be load-balanced, and this re-balancing uses the same ordering using the partitioning scheme of §4 for all meshes in the multigrid hierarchy. Over the course of the simulation, the mesh size varied from 4.6 to about 4.8M cells, and the entire simulation was run on 32 partitions. There were 136 timesteps in the simulation.

The partition coloring in fig. 20 reveals the characteristic "blockiness" of SFC-based partitioning in all the snapshots. Comparing any two frames shows that while the partitioning does change over the simulation due to the requirements of load-balancing a dynamically adapting mesh, the partitioning remains extremely consistent over the entire simulation. At no time does a processor that was working on one portion of the domain find itself integrating an entirely different set of cells at the next timestep. For dynamically partitioned grids, this observation implies that data residing in one node may undergo only minor modification when some distant region of the mesh is modified. The figure shows that even the partitions containing the debris motion undergo only minor modification over the course of the simulation. The layout of the problem on the machine is largely static, and while the partition boundaries do respond to mesh modifications, these changes involve only a subset of the cells near the partition boundaries. Since these boundaries are largely static, the moving debris passes through 4 mesh partitions over its trajectory (labeled *a*-*d* in the first frame of fig. 20).

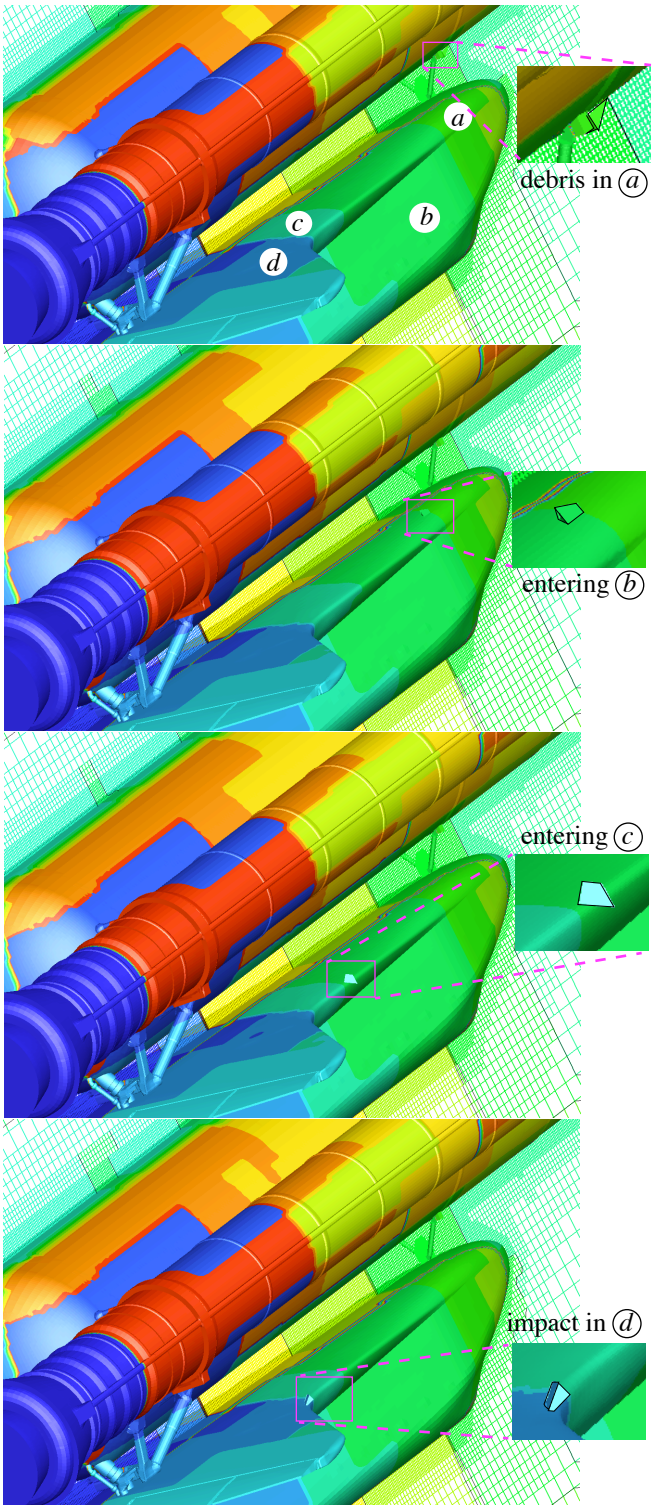


Figure 20: Snapshots of mesh, geometry and mesh partitioning of STS-107 debris case used in moving-body simulation of foam debris impacting orbiter leading edge [23]. The debris travels through several mesh partitions over its trajectory, while the partitioning stays relatively constant despite load-balancing at every timestep.

7 Summary

We have examined the use of space-filling curves in a variety of roles in CFD including mesh coarsening, domain decomposition, and inter-mesh interpolation. While these techniques were demonstrated using non-body-fitted Cartesian meshes, many are applicable on general body-fitted meshes. Algorithms for all of these uses were shown to have linear complexity after performing a single $\mathcal{O}(N \log N)$ reordering of the mesh. On current commodity desktop processors the reordering typically takes under 5 seconds per million cells, while coarsening, partitioning, or solution transfer are all even faster.

On adaptively-refined Cartesian meshes, the coarsening algorithm produces coarsening ratios of around 7 on practical problems, while the partitioner demonstrated linear scalability to well over 600 CPUs with as few as 7000 cells in each partition. The single-grid scheme posted speed-ups of 599 on 640 CPUs on real-world problems with complex geometry. Results were presented showing that in parallel multigrid applications, the partitioner consistently arranges subdomains on coarse and fine meshes with good overlap, thus minimizing the bandwidth required for prolongation and restriction. As a result, the parallel multigrid algorithm scales nearly as well as its single-grid counterpart.

The inter-mesh interpolation algorithm has many practical applications in CFD processes. These include warm-starting solutions after modifying geometry in a configuration study, obtaining Frechet derivatives in design, and as an intergrid transfer operator on remeshed regions in moving-body simulations. The algorithm also has linear asymptotic complexity and can be used to map a solution with N unknowns to another mesh with M unknowns with $\mathcal{O}(M+N)$ operations. These capabilities were demonstrated both on configuration studies examining control surface deflection and moving-body simulations examining debris transport through the flow around the full Space Shuttle launch vehicle during ascent.

8 Acknowledgements

The authors would like to thank G. Adomavicius for his contributions to the reordering tools. Additionally we are grateful to R. Gomez, D. Vicker (NASA JSC), S. Rogers and W. Chan (NASA ARC) for their work on geometry used in the SSLV simulations. Marsha Berger was supported by AFOSR grant F19620-00-0099 and by DOE grants DEFG02-00ER25053 and DE-FC02-01ER25472.

9 References

[1] Karypis, G., and Kumar, V., "METIS: A software package for partitioned unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices." University of Minn. Dept. of Comp. Sci., Minneapolis, MN., Nov. 1997

[2] Schloegel, K., Karypis, G., and Kumar, V., "Parallel Multilevel Diffusion Schemes for Repartitioning of Adaptive Meshes." *Tech. Rep. #97-014,* University of Minn. Dept. of Comp. Sci., 1997.

[3] Ollivier-Gooch, C., "Robust Coarsening of Unstructured Meshes for Multigrid Methods." Presented at the 14th AIAA Computational Fluid Dynamics Conference, Norfolk, Virginia, Jun. 1999.

[4] Venkatakrishnan, V and Mavriplis, D. J.,"Agglomeration multigrid for the three-dimensional Euler equations." *NASA/CR-191595*, 1995.

[5] Aftosmis, M.J., Berger, M.J., Melton, J.E.: "Robust and efficient Cartesian mesh generation for component-based geometry." *AIAA Paper 97-0196*, Jan. 1997.

[6] Samet, H., *The design and analysis of spatial data structures*. Addison-Wesley Series on Computer Science and Information Processing, Addison-Wesley, 1990.

[7] Salmon, J.K., Warren, M.S., and Winckelmans, G.S., "Fast parallel tree codes for gravitational and fluid dynamical N-body problems." *Intl. J. for Supercomp. Applic.* **8**(2), 1994.

[8] Griebel, M., Tilman, N., and Regler, H., "Algebraic multigrid methods for the solution of the Navier-Stokes equations in complicated geometries." *Intl. J. Numer. Methods for Heat and Fluid Flow* 26, pp. 281-301, 1998. Also SFB report 342/1/96A, Institut für Informatik, TU München, 1996.

[9] Behrens, J., and Zimmermann, J., "Parallelizing an unstructured grid generator with a space-filling curve approach, in Euro-Par 2000 Parallel Processing, 6th International Euro-Par Conference, Munich, Germany, August/September 2000, Proceedings, A. Bode, T. Ludwig, W. Karl, R. Wismüller (Eds.), Lecture Notes in Computer Science 1900, Springer-Verlag, 2000, 815-823.

[10] Pilkington, J.R., and Baden, S.B., "Dynamic partitioning of non-uniform structured workloads with spacefilling curves." *IEEE Trans. on Parallel and Distrib. Sys.* **7**(3), Mar. 1996.

[11] Sagan, H., *Space Filling Curves*. Springer-Verlag, ISBN 0387942653. Sep. 1994.

[12] Schrack, G., and Liu, X., "The spatial U-order and some of its mathematical characteristics." *Proceedings of the IEEE Pacific Rim Conf. on Communications, Computers and Signal Processing*. Victoia B.C, Canada, May 1995.

[13] Liu, X., and Schrack, G., "Encoding and decoding the Hilbert order." *Software - Practice and Experience,* **26**(12), pp. 1335-1346, Dec. 1996.

[14] Biswas, R., Oliker, L., "Experiments with repartitioning and load balancing adaptive meshes." NAS Technical Report NAS-97-021, NASA Ames Research Ctr., Moffett Field CA., Oct. 1997.

[15] Berger, M. J, Aftosmis, M. J., Adomavicius, G., "Parallel multigrid on Cartesian meshes with complex geometry"., Proceedings of the 8th International Conference on Parallel CFD, Trondheim Norway, Jun. 2000.

[16] Aftosmis, M. J., Berger, M. J, and Adomavicius, G., "A parallel multilevel method for adaptively refined Cartesian grids with embedded boundaries." *AIAA Paper 2000-0808*, Jan. 2000.

[17] Cliff, S.E., Thomas, S.D., Baker, T.J., Jameson, A., and Hicks, R.M., "Aerodynamic shape optimization using unstructured grid methods.", *AIAA 2002-5550*, 9th AIAA/ISSMO Symp. on Multidisciplinary Analysis and Optimization, Sep. 2002.

[18] Plimpton, S., Hendrickson, B., Stewart, J., "A parallel algorithm for interpolation between multiple grids." *Proc. of the 1998 ACM/ICCC Conf. on Supercomput.* San Jose CA., IEEE Washington DC, ISBN 0-89791-984-X 1998.

[19] Rogers, S. E., Suhs, N. E. and Dietz, W. E. ``PEGASUS 5: An Automated Pre-processor for Overset-Grid CFD,'' *AIAA Paper 2002-3186*, AIAA Fluid Dynamics Conference, June 24-27, 2002, St. Louis. Published in *AIAA J.* **41**(6), June 2003, pp. 1037-1045.

[20] Nemec, M., Aftosmis, M.J., and Pulliam, T.H. "CAD-based aerodynamic design of complex configurations using a Cartesian method." *AIAA 2004-0113*. Jan. 2004.

[21] Murman, S.M., Aftosmis, M.J., and Berger, M.J., "Simulations of 6-DOF motion with a Cartesian method." *AIAA Paper 2003-1246,* 41st AIAA Aerospace Sciences Meeting, Reno NV, Jan. 2003.

[22] Murman, S.M., Aftosmis, M.J., and Berger, M.J., "Implicit approaches for moving boundaries in a 3-D Cartesian method." *AIAA 2003-1119*. Jan. 2003.

[23] Gomez, R.J. III, Aftosmis, M.J., Vicker, D., Meakin, R.L., Stuart, P.C., Rogers, S.E., Greathouse, J.S., Murman, S.M., Chan, W.M., Lee, D.E., Condon, G.L., and Crain, T., "Debris transport analysis" Columbia Accident Investigation Board Report, Vol. II, Appendix D.8, U. S. Government Printing Office, Oct. 2003.