The Parallelization of an Advancing-front, All-quadrilateral Meshing Algorithm for Adaptive analysis[1]

Randy R. Lober

Thermal & Fluid Engineering Department 1513

Sandia National Laboratories, Albuquerque, New Mexico 87185-0835, U.S.A.

Timothy J. Tautges

Computational Mechanics and Visualization Department 1425

Sandia National Laboratories, Albuquerque, New Mexico 87185-0441, U.S.A.

Rich A. Cairncross

Manufacturing and Environmental Fluid Dynamics Department 1511

Sandia National Laboratories, Albuquerque, New Mexico 87185-0827, U.S.A.

Abstract

The ability to perform effective adaptive analysis has become a critical issue in the area of physical simulation. Of the multiple technologies required to realize a parallel adaptive analysis capability, automatic mesh generation is an enabling technology, filling a critical need in the appropriate discretization of a problem domain. The paving [1] algorithm's unique ability to generate a function-following quadrilateral grid is a substantial advantage in Sandia's pursuit of a modified h-method adaptive capability. This characteristic combined with a strong transitioning ability allow the paving algorithm to place elements where an error function indicates more mesh resolution is needed. Other desirable characteristics of this algorithm include its boundary sensitivity and orientation insensitivity (elements near the boundary are of the highest quality and the spatial orientation of the geometry has no effect on the resulting mesh).

Although the original paving algorithm is highly serial, a two stage approach has been designed to parallelize the algorithm but also retain the nice qualities of the serial algorithm. Our approach also allows the subdomain decomposition used by the meshing code to be shared with the finite element physics code, eliminating the need for data transfer across the processors between the analysis and remeshing steps. In addition, the meshed subdomains are adjusted with a dynamic load balancer to improve the original decomposition and maintain load efficiency each time the mesh has been regenerated.

1. This work was performed at Sandia National Laboratories, which is operated for the U.S. Department of Energy under Contract No. DE-AC04-94AL85000.

This initial parallel implementation assumes an approach of restarting the physics problem from time zero at each iteration, with a refined mesh adapting to the previous iterations' objective function. The remeshing tools are being developed to enable 'real time' remeshing and geometry regeneration. Progress on the redesign of the paving algorithm for parallel operation is discussed including extensions allowing adaptive control and geometry regeneration.

Introduction

The traditional finite element method is an established and widely used modeling technique in the engineering analysis community. Newer adaptive finite element techniques iteratively redistribute the degrees of freedom (through remeshing) in a finite element simulation to minimize the error according to a prescribed error function, effectively increasing the accuracy of the solution. Since non-linear adaptive finite element analyses are computationally intensive, massively parallel computing is the most appropriate environment for these analyses. Although massively parallel machines have proven useful for traditional finite element calculations, adaptive analysis on these machines has largely been unexplored.

This paper will describe a framework of software components that together define an approach to achieve a modified h-method adaptive analysis in a parallel environment. The parallel capability is currently under construction and its components include the Cubit [2] meshing toolkit (which supplies the paving meshing algorithm with adaptive controls and geometry regeneration), Sandia's 'tiling' dynamic load balancing code [3], a finite element physics code (the PRONTO [4] code is employed for the parallel implementation), and methods to determine the error or objective function resulting from an analysis iteration of the physics code and map the solution data from the previous mesh to the newly created mesh [5]. This paper will focus on the parallelization aspects of converting the paving algorithm (work in progress) and the adaptive and geometric extensions which were made to the meshing toolkit.

Parallelization Approach

The paving algorithm [1] is a tightly controlled advancing front method which creates all-quadrilateral meshes in general regions. This technique necklaces rows of well-formed, boundary sensitive elements around each region boundary (region boundaries must include one external boundary or loop, and may include numerous internal loops, or boundaries that divide the area to be meshed from an internal void, or hole, in the region. Refer to Figure [1]) until the necklaces, or fronts, collide. When collisions or intersections are detected, angle criteria dictate where connections will be generated between the opposing fronts. In addition, as the necklaces progressed outward from an internal boundary or hole, elements

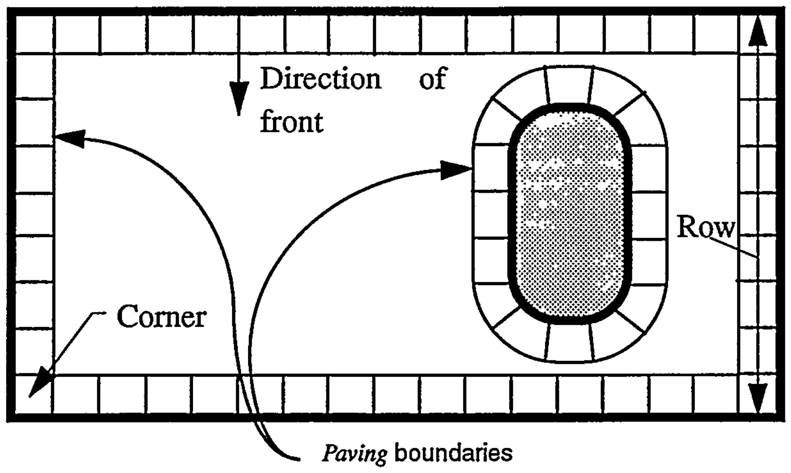


Figure [1] Paving terminology

are inserted when dictated by angle criteria to maintain element aspect ratios on the order of one. Similarly, when necklaces progress away from a concave boundary and begin to impinge on themselves, tucks and seams (element deletions) are performed to maintain element aspect ratio consistency. It has been further established that this technique is quite powerful in managing transitions from coarse element regions to fine regions, a capability that has been demonstrated with a previous adaptive refinement scheme [6]. Since the placement of each element depends on the outcome of the previous element placement, paving is a very serial technique. Two domain decomposition techniques were considered to support a parallel paving implementation.

The first technique considered was termed the moving-front approach. Intending to preserve two important paving advantages (boundary sensitivity and orientation insensitivity), the moving-front parallel paving technique was designed to provide each processor with a dedicated strip of the current meshing front to populate with paved elements until intersection contact occurred with neighbor processors on both sides of the dedicated strip. Since this approach is essentially an advancing front method, the nice paving characteristics are maintained. The limitations include 1) significant communication requirements to avoid neighboring mesh overlaps, and 2) the problem of detecting intersecting fronts from the opposite side without global communication from each processor. While these two issues can be resolved by distributing processor nodes throughout the meshing domain and assigning each a region of influence, this solution sounds very much like the second technique we considered, an area-based decomposition approach. Therefore in its initial form, moving-front contains serious communication bottlenecks, and with this one envisioned modification, it is really another variation of the area-based paving approach.

The area-based decomposition approach exhibits an inherent simplicity by employing paving in essentially the same form in which it operates serially, but contained within a subdomain. Thus once subdomains are defined for the problem regime, paving is initiated in each subdomain simultaneously, treating that subdomain as it would a normal, bounded surface. The main challenge left to solve is the negotiation of subdomain boundaries, since typical driving objective functions (like an error measure) will exhibit coarsening regions as well as refining ones. While this approach seems at first to be the most sure method to realize a parallel paving algorithm, thought must be given to what makes paving such a versatile and powerful algorithm in serial: first and foremost, must be its generality in meshing any region, regardless of boundary complexity or number of boundaries; second is its orientation insensitivity, and finally, its boundary sensitivity. With the area-based approach, generality is maintained, but boundary sensitivity and orientation influence may be affected quite adversely by laying a regular grid over a general geometry and paving the cells. This intrusion of a geometric artifact into the problem cannot be ignored - this similar problem affects all quad and octree discretization algorithms and is one of the primary criticisms of these methods for finite element mesh generation. The investigation into domain decomposition for parallel meshing literature yielded an interesting alternative [7], that of using a background coarse mesh as the initial decomposition for the problem. This approach was implemented and provided a rapid decomposition which was conveniently defined within our data structure, thus enabling work to progress immediately on the issue of parallel paving, or paving within a subdomain. The primary method of decomposition for most problems, however, is to generate a less coarse mesh serially (once prior to start up of the parallel adaptive problem), and use a proven domain decomposition tool (e.g., Chaco [8]) to construct the subdomains. Considering the advantages and disadvantages of the moving-front and the area-based restructured pavers, the area-based method was selected for further development.

The Cubit [2] meshing toolkit is providing the development environment for this effort, supplying the data structures for mesh and geometry entities. The Cubit design currently employs a one-to-one match between underlying solid model geometry definition and the overlying data structure that contains the mesh data and finite element model topology. This mirror-like topology structure allows the meshing tools to generate and store non-manifold mesh data on top of a standard manifold solid model definition. Following this approach, the initial area-based parallel paving implementation generated an individual geometric entity (a surface) and overlying data structure (a reference surface) for each subdomain to store meshing data and control information. It became apparent, however, that this one-to-one match with the underlying solid model definition may be quite inflexible when a finely discretized subdomain boundary needs to be coarsened during boundary negotiation. By the Cubit definition, mesh nodes representing discretization points along a curve must in fact lie on that curve, yet it is quite possible in this coarsening scenario that the curve itself needs to be redefined to allow for sharp corners or even kinking of the once smooth subdomain boundaries. These curve modifications would be unwieldy with typical cubic spline curves such as are commonly employed in Cubit's solid modeler.

To address this issue, a virtual geometry paradigm was conceived where the one-to-one solid model entity to data structure entity ratio is relaxed completely. This methodology is similar to the "structured block" technique which is used by several commercial grid generators for the computational fluid dynamics field. In this approach, mesh data and control instructions can span multiple physical geometric entities and vice-versa, since the one-to-one ratio can now be maintained with virtual geometry which can exist anywhere in the model. For the purposes of parallel paving, this paradigm will consist of physical geometry defining the underlying surface equation and bounding curves of the problem domain, but the internal subdomain boundaries are defined by ordered lists of mesh entities, i.e., connected mesh nodes that lie on a surface, but whose position within the surface can be manipulated as necessary by the two subdomains sharing the boundary and the nodes that define the boundary. The ability to manipulate the subdomain boundary topology without disturbing the underlying surface geometry is a nice advantage for the virtual-geometry based area decomposition method. In this method virtual curves can be defined from consecutive sequences of nodes where appropriate, constructing a series of connected straight line segments which together will be treated as a single curve. To modify these curves, nodes can be added or removed from the defining sequence order. Now the sharp corners can be resolved readily, and elements can be swapped between adjacent subdomains (which will be typical during dynamic load balancing) without the need to modify the underlying solid model definition.

Virtual curves will also be modified by boundary negotiation between neighboring subdomains. Boundary negotiation will occur at each iteration of the overall parallel adaptive analysis loop as the mesh relocates to better resolve the emerging gradients of interest. The significant steps of this data flow loop are as follows:

Serial portion (first iteration only):

1. Start up (generate coarse mesh, initial mesh decomposition (subdomain construction), data loading

Parallel portion (each iteration):

2. Subdomain boundary discretization and negotiation
3. Adaptive meshing via sizing function
4. Re-map variables from old mesh to current (beginning with second iteration)
5. Dynamic load balancing (beginning with second iteration)
6. Finite element analysis

7. Convergence test (met accuracy criteria? if yes, exit)
8. Generate new spatial objective function over problem domain (error metric)
9. Go to step 2

A controlling driver code is being constructed to manage this process loop and enable efficient diskless data exchange between the various code components. Since the density of the subdomain boundary discretization listed in the second step above is controlled by the objective function from step 8, the virtual curves which define the subdomain boundaries will adjust as the number of nodes supplied to model the virtual curve increases or decreases. The topology of the subdomain geometry will remain constant even though the underlying geometry can change at each iteration. As some subdomains grow in number of elements contained and others diminish, Sandia's "tiling" load balancing code [3] will balance the resulting mesh at each iteration to maintain analysis efficiency. Figure

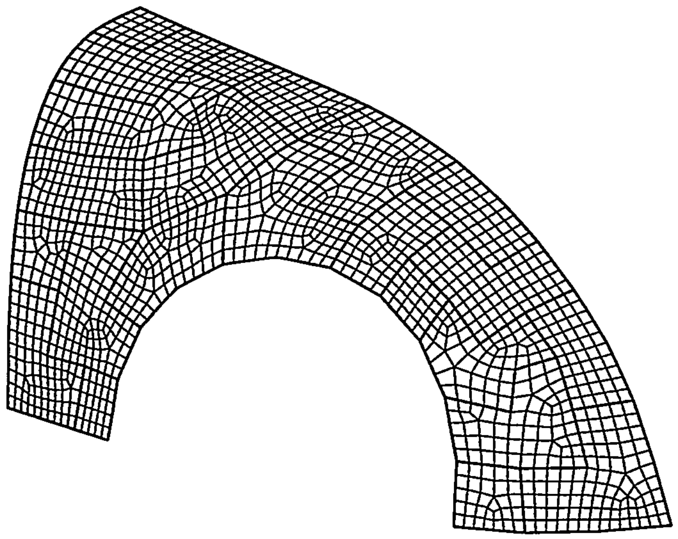


Figure [2] NURB surface mesh containing 24 simple subdomains

[2] shows the result of paving the interior of 24 subdomains of an automotive-like fender (NURB surface), and Figure [3] displays the same 24 subdomains after load balancing has occurred. Note the significant element swapping that occurs between adjacent subdomains: this result improves parallel performance but will require constant change to the subdomain boundaries. This tendency further justifies the need of representing the interior subdomain boundaries with the more flexible virtual geometry paradigm.

Adaptive Meshing Capability

To begin the generation of an adaptive mesh, a function to define the mesh density must be developed. This function will typically be an error function which will guide the paving algorithm to resolve high error regions with greater mesh detail. One error measure has

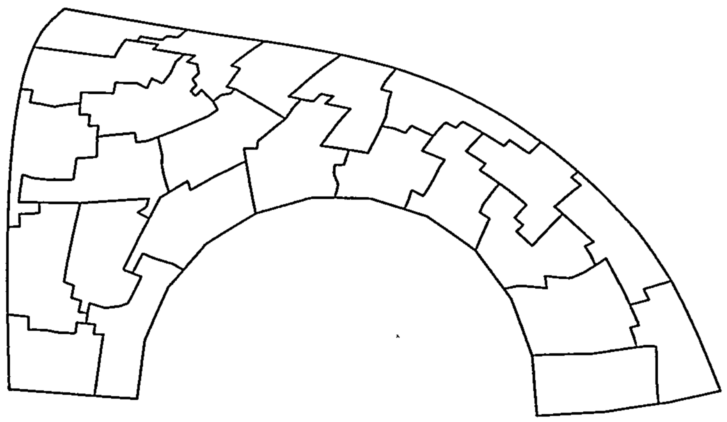


Figure [3] NURB surface (mesh hidden) with 24 subdomains after load balancing

already been tested [9] in this scenario, and more error measures are planned to be added as needed. The issue of resolving this background objective function has significant impact on the parallel capability of this code. Two types of analysis output variables can be used to provide sizing information: nodal and element based. Nodal variables (temperature or displacement) contain discreet values at each node point of the previous mesh, and new point evaluations along a boundary between two subdomains can be determined explicitly with only the two neighboring subdomains. This characteristic of nodal variables is desirable, since the communication needs of these evaluations for a parallel computation are limited to the two neighboring domains. Element based variables must be extended to the node locations via a standard Laplacian interpolation. The value of the function at a node, n, is simply the summation of all the contributions from its surrounding elements divided by the number of surrounding elements. Due to this interpolation, this type of variable requires more information in the neighborhood of a vertex (to produce an equivalent field of nodal function values) at which more than two subdomains meet (see Figure [4]). Since the function value at the vertex requires contributions from each of the

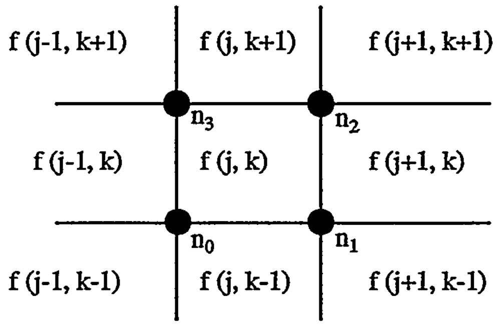


Figure [4] Interpolating element function values to nodal locations

neighboring elements, each adjacent subdomain must communicate its corner element's contribution at this location. It is then important to minimize the number of subdomains which meet at a vertex for analyses which use element based variable functions. Once the nodal function values are available, function size requests are computed using a bilinear interpolation technique from the nodal values residing on the previous mesh nodal locations. This interpolation works well [6] for the resolution range of the paving algorithm, even though the gradient of the interpolated function changes discontinuously at the boundaries of each element of the previous mesh. Other higher order interpolations could be pursued (such as bicubic interpolation) which would alleviate the discontinuity, but at a higher computational cost.

Paving displays significant strength in its role as a versatile quadrilateral meshing algorithm for general geometry. The transitioning capability of paving is what allows it to effectively generate new mesh which follows a provided function. This extension of following a function introduces some additional constraints, however. Paving operates in several stages: 1) mesh filling, then 2) mesh cleanup. Mesh filling is intended predominately to manage the coverage of the problem domain with quadrilaterals, even if some of the resulting element connections are less than ideal. The mesh filling process is controlled by numerous heuristic constraints which produce very good quadrilateral (nearly square) elements to begin with. Mesh cleanup improves the overall mesh by looking for undesirable node topology and replacing them with near-regular mesh configurations. The extension of paving to follow an objective function dynamically adjusts some of these heuristics, guiding the mesh filling phase to create elements smaller or larger as indicated by the local objective function values. While the mesh filling phase tends to model the gradients of the objective function quite well, the subsequent mesh cleanup and overall smoothing operations (grid relaxation) tend to muddy the gradients out again, since the gradients are normally achieved in part by allowing slightly elevated element deformation in the local neighborhood of the gradient. In short, the objective of capturing an external function within a mesh is many times in contrast with the objective of maintaining pristine element quality.

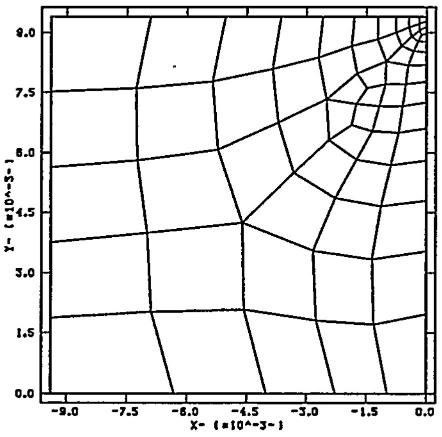
Meshing controls were designed to provide the user with some external capability to buffer the severity with which the objective function behaves. Functions with extreme gradients (orders of magnitude) in a very small area are more difficult to accurately represent in a mesh than are functions with more gentle fluctuations. The current controls allow users to set maximum and minimum mesh size values, which effectively scale the range over which the objective function can request, allowing the paving algorithm more flexibility to insure an appropriate mesh is constructed. Quality metrics were also added to assess element quality and actual size versus requested size by the driving objective function.

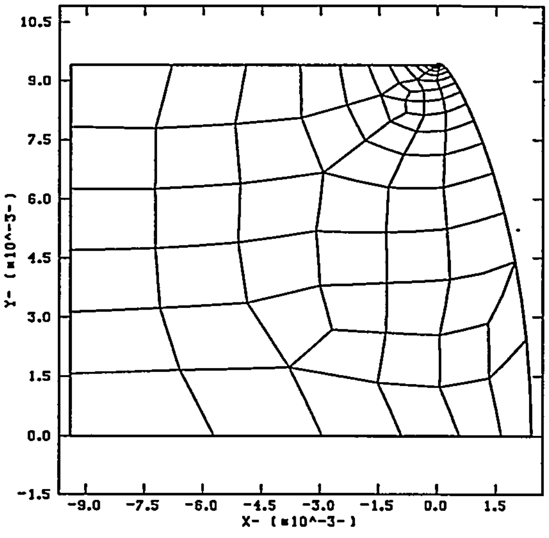
Once a satisfactory new mesh has been built, the physical simulation is started again. The initial implementation will support a remesh/restart methodology in which the physics code is restarted with the improved mesh at time equals zero. The tools to perform a true adaptive solution (in which remeshing is performed at multiple intervals throughout the analysis) are currently being developed within this same framework. In this latter scenario, the analysis data must be transferred from the previous mesh to the new one at each remeshing interval. An isoparametric finite element data remapping code, MERLIN II [5], is currently being tested with several cases for the nodal and element variable cases. MERLIN II employs an isoparametric representation of the finite elements, and uses Newton's method to iteratively determine interpolated quantities.

Geometry Regeneration

Another important capability to support adaptive analysis is the geometry regeneration task, that of reconstructing surface and solid model data from deformed tessellated models (finite element mesh models). Since finite element mesh models are typically non-manifold, constructing a manifold solid model from the non-manifold mesh information only can be challenging. By definition, a non-manifold edge (or curve) is being used to bound more than two faces (or surfaces), and a non-manifold vertex has geometric elements that can only be connected topologically through that vertex (such as two cones which meet at their apexes). To help with this construction difficulty, topology records were added to the finite element model data which Cubit [2] generates to be used by the analysis code. By retaining this geometric topology information with the mesh, Cubit can reconstruct new solid geometry (currently in two dimensions) from deformed mesh and use the new geometry to generate a new adaptive mesh.

A fluid mechanics example demonstrates the generation of new two dimensional geometry from a deformed mesh. This problem simulates the filling of a mold, specifically the manufacturing of rocket propellant. GOMA [10], a full-Newton finite element program for free and moving boundary problems with coupled momentum, energy, mass, and chemical species transport, has been used to model the fluid behavior during the filling of the mold. The fluid motion begins flowing to the right, as shown in Figure [5]. As the material flows past the upper right corner of the boundary, it expands upward resulting in significant and rapid mesh deformation at this location. Figure [6] shows in detail this deformation, which prevents the GOMA solver (a fully coupled Newton-Raphson iteration) from converging. The severe kinking in the corner element of Figure [6] is possible since the chosen element type was a nine-node quadrilateral, however the finite element solution at this element is meaningless as a result of the kinking. Elements which undergo distortion resulting in internal angles greater than 180° will not yield useful results, yet the advantage of remeshing allows one to proceed beyond the extensive distortion by replacing the deformed





Initial mesh

First deformed mesh

Figure [5] Mold filling: initial and first deformed mesh

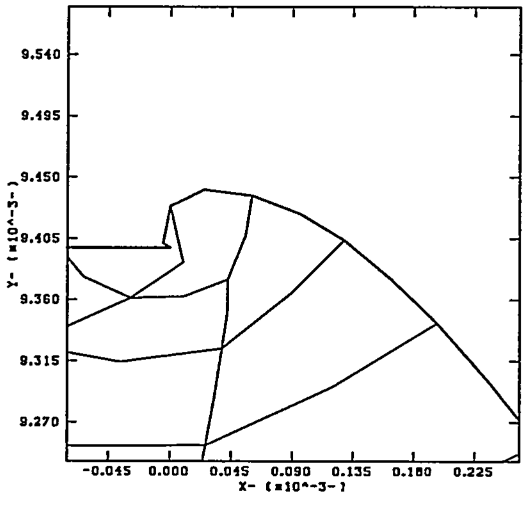
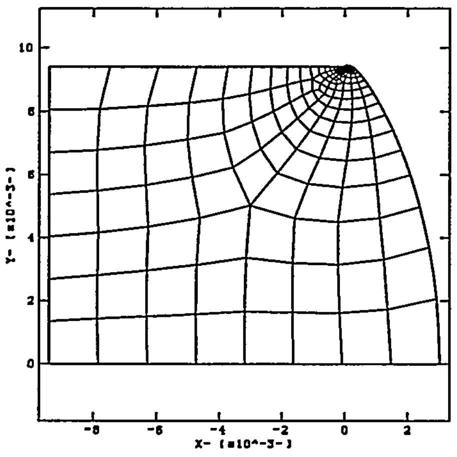
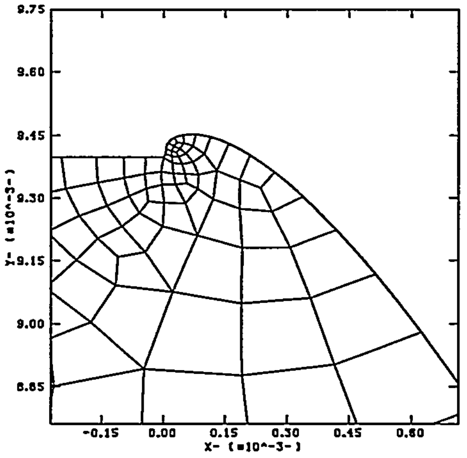


Figure [6] Mold filling: close view of upper right corner, first deformed mesh

mesh with newly generated elements. By remeshing in intervals whenever deformation begins to deteriorate the efficiency of the GOMA solver, the problem can progress through time effectively. At each remeshing interval, Cubit was used to create new geometry from the previous deformed mesh and generate a new mesh, which MERLIN II populated with the appropriate variable values from the previous iteration. Figure [7] and Figure [8] display the resulting new mesh following the third and fourth remeshing intervals from the mold filling simulation. A capability to perform geometry regeneration for multiple



Remesh after third interval



Close view of upper right corner

Figure [7] Mold filling: new mesh after third remesh interval

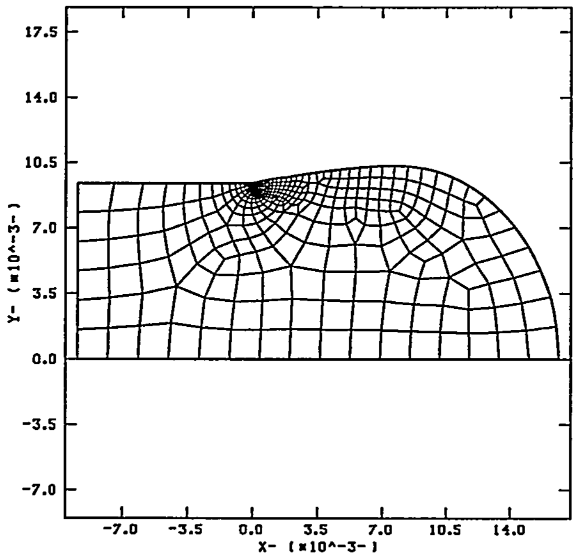


Figure [8] Mold filling: new mesh after fourth remesh interval

material models is also being developed.

Future Directions

Future effort will focus first on completing the driver code to control the data exchange between code modules. The area-based decomposition paver with virtual geometry will then be tested (in parallel on a network) with the parallel driver code, using external files initially for data exchange. Then an efficient diskless data exchange mechanism will be designed and implemented within the driver code to avoid excessive disk access during loop iterations. The issue of subdomain boundary negotiation will be resolved with one of the two methods being considered: 1) a single processor will "own" each subdomain boundary and will assume the responsibility of meshing it, or 2) processors sharing a subdomain boundary will each generate their own local mesh for it using a deterministic algorithm. Next, the global smoothing stage of the paving process will be investigated for reformulation as a fully parallel operation that is still adherent to the objective function, and the cleanup stage (element deletion/insertion to improve a mesh containing poorly shaped elements) will be redesigned to reduce connectivity irregularities along the interior subdomain boundaries. The meshing code module memory requirements will be reduced to allow operation within the typical memory space of a single node of a massively parallel machine. At this point, the ability to operate a remesh/restart adaptive analysis loop in parallel will be tested using a problem which exhibits a coarsening and refining objective function (on a network), followed by a large problem to be run on a massively parallel machine. Additional work will also be performed in the improvement of the adaptive meshing and the geometry regeneration capability. The ultimate goal of this adaptive work is the achievement of tools to perform adaptive analysis on general two and three dimensional geometry in serial and in parallel environments.

References

1. Blacker, T. D., and Stephenson, M. B., "Paving: A New Approach to Automated Quadrilateral Mesh Generation," *Int. J. Num. Meth. Engr.*, 32, 811-847, 1991.
2. Blacker, et al., "CUBIT Mesh Generation Environment, Volume 1: User's Manual", SAND94-1100, Sandia National Laboratories, Albuquerque, New Mexico, 1994.
3. Devine, K. D., Flaherty, J. E., Wheat, S. R., and Maccabe, A. B., "A Massively Parallel Adaptive Finite Element Method with Dynamic Load Balancing", Proceedings of Supercomputing '93, IEEE Computer Society Press, pp. 2-11, 1993.
4. Taylor, L. M. and Flanagan, D. P., "PRONTO 3D A Three-dimensional Transient Solid Dynamics Program", SAND87-1912, Sandia National Laboratories, Albuquerque, New Mexico, 1992.
5. Gartling, D. K., "MERLIN II - A Computer Program to Transfer Solution Data Between Finite Element Meshes", SAND89-2989, Sandia National Laboratories, Albuquerque, New Mexico, 1991.
6. Blacker, T. D., Jung, J., and Witkowski, W. R., "An adaptive finite element technique using element equilibrium and paving", ASME Paper No. 90-WA/CIE-2, November, (1990).
7. Wu, P. and Houstis, E. N., "Parallel Adaptive Mesh Generation and Decomposition", to be published in *Engineering with Computers.*
8. Hendrickson, B. and Leland, R, "The Chaco User's Guide Version 1.0", SAND93-2339, Sandia National Laboratories, Albuquerque, New Mexico, 1993.
9. Tautges, T. J., Lober, R. R., Vaughan, C. T., and Jung, J., "The Design of a Parallel Adaptive Paving All-Quadrilateral Meshing Algorithm", accepted for the 6th International Symposium on Computational Fluid Dynamics, September 4-8, 1995, Lake Tahoe, USA.
10. Sackinger, P. A., Schunk, P. R., and Rao, R. R., "A Newton-Raphson Pseudo-Solid Domain Mapping Technique for Free and Moving Boundary Problems: A Finite Element Implementation", submitted to *Journal of Computational Physics*, 1995.