From CAD to Simulation Run

Automatic Grid Generation and Interfacing

Gamm Conference
25 – 28 March 2002, Augsburg, Germany

J. Häuser Dept. of HPCC, CLE, Salzgitter, Germany
P.R. Eiseman, Z. Cheng, PDC, White Plains, N.Y., U.S.A.
H.-G. Paap, HPC Consultants, Regensburg, Germany
Acknowledgments

This research was partly funded by the ministry of Science and Culture of the State of Lower Saxony, Germany and the European Commission under contract JavaPar 1997.262 and EXTV 1999.045.

The authors are grateful to T. Gollnick, T. Ludewig, Dept. of HPCC, CLE and Y. Xia, ESTEC-ESA, NL for providing material for this presentation.

Roy D. Williams, Center of Advanced Computational Research, Caltech, U.S.A. provided computing time.

The authors are obliged to Jean Muylaert, ESA, NL for geometry data and numerous stimulating discussions.
Presentation Overview

The following topics will be discussed with varying level of detail, each being itself a major research area

CAD Data Conversion
Grid Generation for Complex Geometries
Grid generation and HPCC
JavaGrid Concept
The need for accurate three-dimensional simulation in numerous fields of engineering and science for real processes requires the development of ever more sophisticated three-dimensional grids as well as powerful *High Performance Computing and Communications (HPCC)* resources.

These notes present the steps from the original CAD data to the final simulated solution. The Topology Input language for multi-block grids is explained along with the software *Automatic Zoning Manager (AZ)* for interactive topology design. Numerous utilities for grid clustering, merging, setting BCs for solvers are available, too. Furthermore, the strategy for parallelization of complex grids and, in particular, for Java as the language for High Performance Computing is presented. The concept of a *JavaGrid* is introduced, that promises to provide the combined power of networked computational resources for solving most complex scientific and engineering problems both in geometry and in physics.
Engineering computations deal with complex geometries.

CAD data is in general not directly usable, since patches may contain overlaps, intersections, or voids. There is no guarantee for a closed surface.
Importing Geometry

**Build-in surface types:** plane, ellipsoid, tube, linear, periodic boundary

**Digital surfaces:** Surfaces based on triangles or quads, e.g. Partran, Nastran, STL, Plot3d.

Currently under development: IGES (Tetin is similar to IGES)
Topology Input Language (TIL)

TIL code can be integrated into own software packages.

Example: CCAD from Concepts, ETI supports features for creating geometry for turbo-machinery and in addition generates TIL code automatically.

This clearly demonstrates the flexibility and strength of TIL code.
Closed Surface Construction

A novel algorithm that works globally
In order to construct a closed surface from the CAD patches, one needs to start from a closed surface, that is the CAD geometry (see next picture) is embedded within a closed surface. In order to construct a closed surface which is topologically similar to the original CAD geometry, the surrounding space is subdivided into uniform cubes. The size of a cube depends on the characteristic length scale of the geometry. Cubes containing data points, obtained from discretized CAD data, are marked. All other cubes are eliminated. Thus an initial closed surface is obtained as shown in the following picture. Finally, the idea is to shrink the surface like a balloon, and if close, to project. The simplest way of automatic shrinking is done by averaging neighboring data points. Finally, shrinking can be combined by projection, as shown in the final picture. To deal with multiply connected domains, a curvature dependent shrinking algorithm is needed.
Complex Grids and their Features

As an example we consider aerospace: In recent years there has been a slew of activities in designing advanced aerospace transportation vehicles both in the U.S. and Europe.

Related grid generation activities are in turbomachinery.
Space Activities and Spin-Offs
Two-Stage to Orbit Vehicle

Conceptual design of Two stage to orbit vehicle.

Courtesy of NASA Ames Research Center.
A grid is presented for an airplane configuration consisting of fuselage, wing, and engine.
Overlap Grids in GridPro
CFD Solution for Generic Missile
Blade Rows after Welding
Impeller with 3 Inducers

120 degrees periodic piece with one inducer and 3 splitters. The geometry for periodic boundaries is automatically determined along with the grid.
3D Laminar Vortex Breakdown

Apparatus of Sarpkaya/Leibovich (16 Vanes, 38 deg. Angle)

D. O. Snyder, R. E. Spall
3D Laminar Vortex Breakdown

Colors indicate the flow velocity.
The basic strategy for obtaining a complex (multiblock) grid is the separation of topology space and physical space.

Once the topology is specified, either interactively by, for instance, the GridPro AZ-manager or writing GridPro TIL code (Topology Input Language, see Ref. 1), surface and volume grids are generated completely automatic.

In the following a relatively simple grid example is presented. It will be seen that hundreds or thousands of blocks of different size are generated, necessitating a special parallelization strategy.
Design Variation (Initial Configuration)
Design Variation (Modified Configuration)
SET GRIDDEN 8
SET DISPLAY.SURF ON

COMPONENT main()
BEGIN
  INPUT 1 surf(sOUT (0..1));
  INPUT 2 corn(sIN (1:1..2),cIN (-16));
END

COMPONENT surf()
BEGIN
  s 0 -ellip (1 1 1) -o ;#TIL:1:1
  s 1 -ellip (3.33333333333 2.5 2) -t 0.2 0 0 ;#GI:0.2
END
The solution domain is defined between the torus and the sphere. A topology wireframe is constructed as shown.
The local wireframe topology is built outside the main wireframe structure and then linked to the main wireframe topology. The sphere gives the real position of the local topology.
Position the Local Topology

The local wireframe topology is placed in its real position.
Grid Topology and Final Grid

The final grid is generated based on the given topology.
TIL Code Ball in Torus Example

#TIL code Ball in Torus
SET GRIDDEN 8
SET DISPLAY.SURF ON

COMPONENT main()
BEGIN
    s 1 -implic "torus.h";  #analytical surface for torus
    INPUT 1 torus(sIN (1),cOUT(1:1..4, 2:1..4));
    INPUT 2 (0 0 1.5)*ball(cIN (1:1..8));
END
#surface description file for torus:
define FUNCUya=sqrt(y*y+z*z)-1.5, 1-(ya*ya+x*x)*4
COMPONENT ball(cIN c[1..8])
BEGIN
  s 1 -ellip(4 4 4);
  c 1 -0.35 0.35 -0.35 -s 1 -L c:1;
  c 2 -0.35 0.35 0.35 -s 1 -L c:2 1;
  ...  
  c 8 0.35 -0.35 -0.35 -s 1 -L c:8 4 7 5;
END
The Topology Builder Panel is used to interactively select surfaces and to construct the block topology, i.e. corners and edges. Surfaces can be imported using several different standards, but build-in surfaces can also be used.

The Grid Viewer Panel is used to view the grid in various ways and thereby check its quality by visual inspection. Furthermore, global grid quality can be checked by applying the utility qchk to the grid data file.
The miniCAD Panel is used to repair surfaces or modify surfaces for grid generation.

The Property Setter Panel is used for setting boundary conditions for the generated grid. Various flow solvers are already supported, but the list will be further increased, depending on the needs of our users.
Ggrid is the topology and grid generation engine. It uses the TIL code, i.e. the topology and the surface description, as input. First, the topology is parsed. Then, in the second stage, a multi-block grid is generated. Ggrid provides algorithms to optimize the grid quality with regard to smoothness and orthogonality throughout the entire grid, which translates into greater CFD accuracy and efficiency.

Usually, Ggrid is launched from the AZ-Manager after the topology has been designed and the surfaces have been prepared. However, it can be also launched from a command line, i.e. DOS prompt or UNIX shell. This gives, for instance, users the opportunity to integrate Ggrid into a design loop. After generating the grid, the flow solver is launched to compute the flow field, followed by a post optimizer that modifies the surfaces, which are then used to generate a modified grid. For moderate changes in the surface description the topology does not have to be changed. Thus, human interaction can be entirely removed from the design loop.
Cluster tool converts Euler- into Navier-Stokes grids.

Note: the optimized grid quality leads to a speedup of the convergence rate of 3 to 10, when compared with traditionally generated grids.
Cluster $\mu$ From Euler to Navier-Stokes

First, generate Euler grid, then apply cluster tool to convert it into a Navier-Stokes grid. Same Euler grid can be used to generate Navier-Stokes grids for different Reynolds numbers.

Variety of different strategies are implemented to steer cell heights, grid line distributions, etc.

Support for multi-grid technique.

Works also for non GridPro structured multi-block grids.
Utilities

- **chfmt**: Convert and translate data formats
- **cutg**: Extract cells from a grid bounded by planes
- **extconn**: Extends connectivity information for periodic and reflected boundary conditions
- **genconn**: Generate connectivity information based on a given multi-block grid
- **getg**: Extract blocks, surfaces, or sheets from given grid data file
- **grid2til**: Convert an elementary multi-block grid into a TIL code
- **hex2mb**: Reverse engineer a multi-block grid from an unstructured hexahedral grid
Utilities

- Convert IGES data into GridPro data format
- Convert GridPro grid into WIND input format
- Generate a ribbon from 3 path lines
- Post processing (smooth) overlap grid form mrgb -O
- Merge two grid files into one file and generate connectivity information
- Merge (equivalent) nodes and cells by tolerance
- Grid quality checker
Utilities

- **segb**: Segment general multi-block data into elementary block data
- **siz**: Check the physical size of data
- **syncb**: Synchronize or re-orient block boundaries or block indices
- **thin**: Remove nodes from triangular mesh while preserving geometry.
- **trf**: Transform grid data (translate, rotate); coarsening grid based on interpolation
- **xsec**: Compute planer cross section of digitized surface
Grid Generation and HPCC

Until recently, there were 3 basic parallelization strategies:

- do loop parallelization
- parallelization of numerical algorithm
- domain decomposition

With the advent of Java a new strategy emerged:

- thread based parallelization
Parallelization by Domain Decomposition

do loop and algorithm parallelization do NOT work for complex geometries, but Domain Decomposition does

Block topology for X-33 configuration
Task mapping for 462 block
Huygens space probe

- **Block to processor**
- **mapping for 8 processors**
  - (i) Heuristic method
  - (bin-packing)
  - (ii) Recursive bisection
JavaGrid Concept

how to build a computational grid using the

JavaGrid Concept
Java for HPCC

So far, software for computational science and engineering has been written mainly in Fortran, and in recent years the more advanced C programming language has been employed for visualization tasks. Unfortunately, these procedural languages force the programmer to think like a computer, breaking the problem down into a set of basic data types. **Object-oriented languages**, on the other hand, allow programmers not only to think more efficiently, but also to collaborate more effectively with others.

**Ten reasons for using Java as the language for HPCC**

1. Object-Oriented Programming
2. Robustness, Understandability
3. Computational Efficiency
4. Concurrent, distributed, parallel
5. Portability
6. Leveraging Business Investment
7. Multithreading, Dynamic linking
8. Database Connectivity
9. Remote Method Invocation, Internet
10. Security

*C++ is an obscure, hard to debug, unreadable and costly language*
Why Threads are good for HPCC

Threads allow straightforward implementation of macro- and microparallelism.

Threads provide the general parallelization strategy for HPC codes. Threads are allocated and handled by the OS not by the user. Processor allocation and scheduling is done by the OS.

Advanced numerical schemes, for instance, in CFD, i.e. GMRES, do not require the same computational work for each grid cell, i.e., load changes dynamically.

Without threads, highly sophisticated (dynamic) load balancing algorithms are needed for parallel efficiency.
In this figure a multiblock grid for the X-33 vehicle is shown. Each block is run in its own thread. Grids may have thousands of blocks, and thus the OS has to create the corresponding number of threads and is also responsible for starting and stopping all threads.
Client-Server communication through Java RMI (Remote Method Invocation). Each shared object has an interface, common to client and server sides, that defines what methods are available. The client can invoke methods on the object, but these are executed on the server where the object actually resides.
Every solver object contains the data of and the numerics for one block. The solver class is sent from the client to the server that is, different users may use different solvers.
The client controls remote objects by invoking methods on them. There is only one registered RMI object, the Master, which can spawn Sessions so that multiple users can work. Each Session can spawn a number of Node threads, each of which can dynamically load a Solver, which is responsible for computation in a single block of the grid.
Communication between blocks. Each block copies the first layer of internal points into the face buffer, and the flag is set to “ready” for that face, meaning that the neighbor block can read it. The neighbor reads from the buffer into its halo layer. The word “Transform” refers to the difficult problem (in 3D) of mapping the face array to one face of the block, or other protocol translation from block to block.
Parallel matrix multiplication is implemented by block matrices, as shown in the Figure. Matrices A and B are multiplied to produce C.

The multi-threaded matrix multiplication is performed by splitting matrix C into partitions. Each partition is then calculated by one thread, with the thread numbering as shown for matrix C. Concurrent access to the memory containing A and B is necessary: here we see the memory that thread 2 accesses.
Multi-threaded Matrix Multiplication

Megaflop rates for the pure Java multithreaded matrix-multiply benchmark.

<table>
<thead>
<tr>
<th>number of threads</th>
<th>HP using 16 CPUs</th>
<th>HP using 1 CPU</th>
<th>Sun using 4 CPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>30x30</td>
<td>300x300</td>
<td>30x30</td>
</tr>
<tr>
<td>1</td>
<td>7.01</td>
<td>8.64</td>
<td>6.51</td>
</tr>
<tr>
<td>4</td>
<td>11.38</td>
<td>33.49</td>
<td>3.86</td>
</tr>
<tr>
<td>9</td>
<td>6.33</td>
<td>72.53</td>
<td>2.40</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>118.68</td>
<td>8.69</td>
</tr>
<tr>
<td>25</td>
<td>2.62</td>
<td>112.97</td>
<td>1.04</td>
</tr>
<tr>
<td>36</td>
<td>1.83</td>
<td>110.66</td>
<td>0.75</td>
</tr>
<tr>
<td>100</td>
<td>0.64</td>
<td>109.53</td>
<td>0.29</td>
</tr>
</tbody>
</table>

On the HP architecture a maximal speedup of 13.74 using 16 processors for the 300x300 matrix example was measured.
## Java Parallel Euler Solver

Caltech HP V-Class times for 320 iterations

<table>
<thead>
<tr>
<th>number of blocks</th>
<th>number of cells</th>
<th>time in seconds</th>
<th>parallel speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>single processor</td>
<td>multi processor</td>
</tr>
<tr>
<td>16</td>
<td>121104</td>
<td>3246.73</td>
<td>541.13</td>
</tr>
<tr>
<td>16</td>
<td>200704</td>
<td>6908.88</td>
<td>1077.20</td>
</tr>
<tr>
<td>16</td>
<td>484416</td>
<td>12905.88</td>
<td>2720.48</td>
</tr>
<tr>
<td>48</td>
<td>118800</td>
<td>2980.93</td>
<td>225.76</td>
</tr>
<tr>
<td>48</td>
<td>202800</td>
<td>5190.54</td>
<td>436.09</td>
</tr>
<tr>
<td>48</td>
<td>480000</td>
<td>12663.30</td>
<td>1162.54</td>
</tr>
</tbody>
</table>
Conclusions

Both structured as well as unstructured hexahedral grids can be interactively constructed for very complex geometries providing extremely high grid quality.

In large scale computations a 3 to 10 fold speed up has been observed in comparison with traditionally generated grids.

Multiblock grids are straightforward to parallelize achieving perfect linear parallel scalability.

The Java thread concept has proved extremely well suited for large scale parallelization and for Grid computing.

Java with its OOP design and unique Internet and security features is the language for HPCC in science and engineering. Numerical performance rivals or exceeds C++.
Future Work

Grid generation
- automatic repair of CAD data
- automatic topology design
- Object-Oriented grid generation
- moving grids with variable topology
- automatic extension to hybrid grids

HPCC and Java
- handling of large number of threads (up to ten thousand)
- dynamic load balancing of the OS with regard to thread allocation
- extension to distributed parallel architectures (Java Spaces)

There is no doubt that Java is the most effective and efficient language for scientific and technical computing. *Java possesses, however, a steep learning curve.*


see also [www.cle.de/cfd](http://www.cle.de/cfd) for downloading additional information.