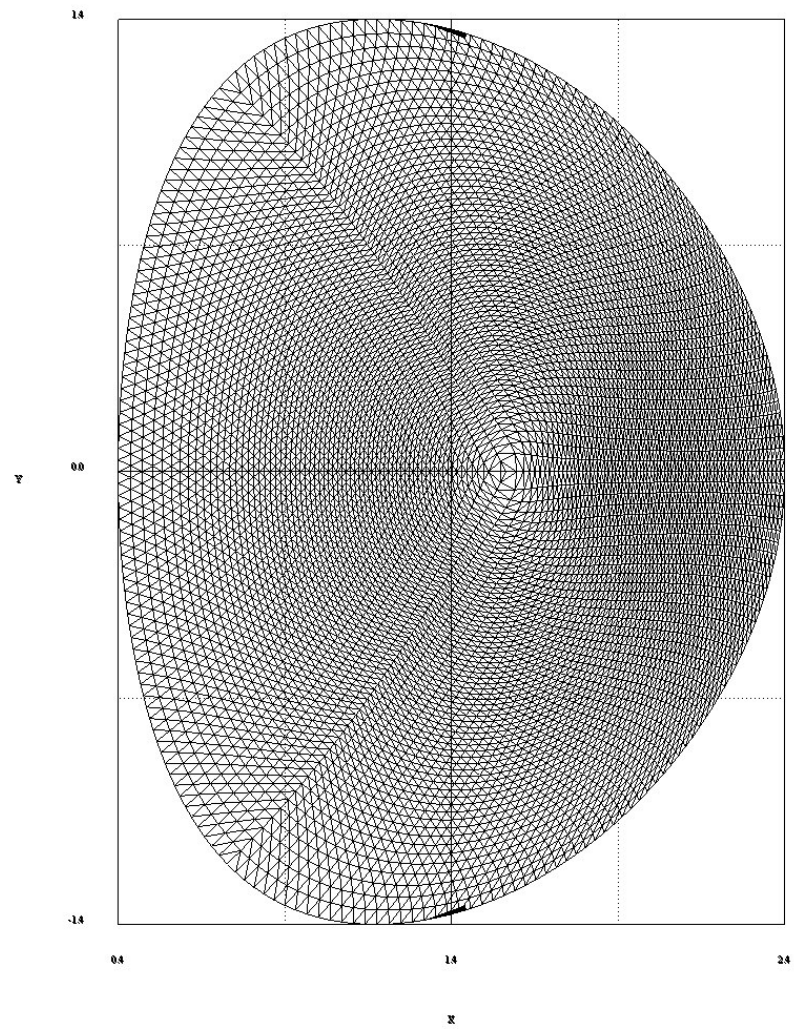
Meshing and Data Structure of M3D (Linear Basis Function Version)

Josh Breslau
PPPL

ORNL/PPPL X1/XT3 Workshop
Oak Ridge, TN
August 9, 2005

The M3D Mesh

- All calculations are done in real space (not spectral).
- Uses linear basis functions on **triangular finite elements** in each constant-ϕ plane.
- Mesh has same topology in all planes. In the tokamak case, it has the same geometry in all planes as well.
- Mesh is aligned with flux surfaces (from VMEC-generated input files) but does not follow field lines.
- Uses either finite differences or pseudo-spectral derivatives between planes.



Setting up a Mesh: the Configuration File

The mesh dimensions and domain decomposition for an M3D run are specified in the configuration file, given on the command line, e.g.

```
poe ./m3dp.x ... -configFile config.dat48 ...
```

where "config.dat48" is the name of a configuration file for 48 processors.

File format: six integers (one to a line) in the file determine sizes as follows:

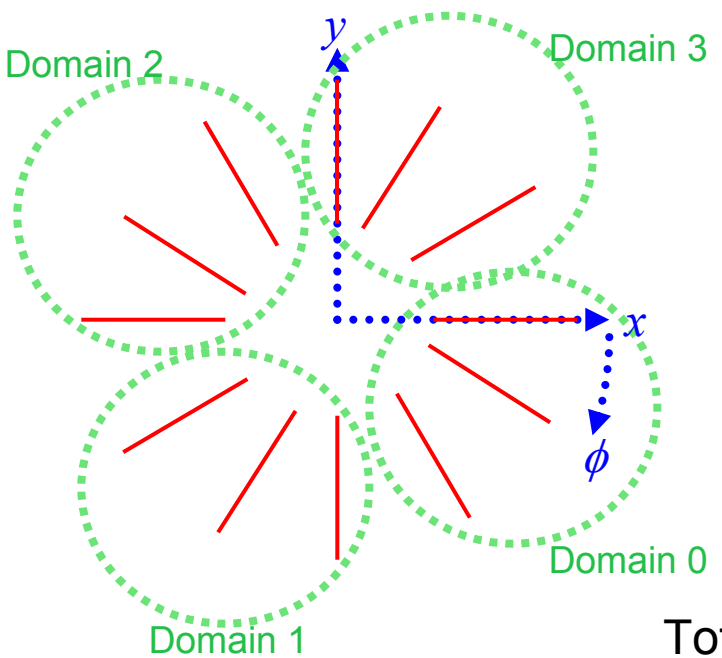
A: *Total number of planes (constant ϕ cross-sections).*
B: *Number of CPUs toroidally.* ***A*** *must be divisible by* ***B****.*
C: *Number of (minor) radial grids globally. Usually odd. For linear elements,* ***C****=31 would be small;* ***C****=61 is medium;* ***C****=91 fairly large;* ***C****=121 very large.*
D: *Number of radial CPUs. Should generally be one; use caution when exceeding.*
E: *Poloidal symmetry of mesh, a rough measure of θ resolution. At least 3. Could go as high as 5 or more for highly shaped stellarator cross-sections.*
F: *Number of θ CPUs. Can vary from 1 to about 6-8 depending on* ***C*** *&* ***D****.*

Quantities A,C, and E in red determine the mesh structure and global data size.
Quantities in B, D, and F in blue determine the domain decomposition and local data size.

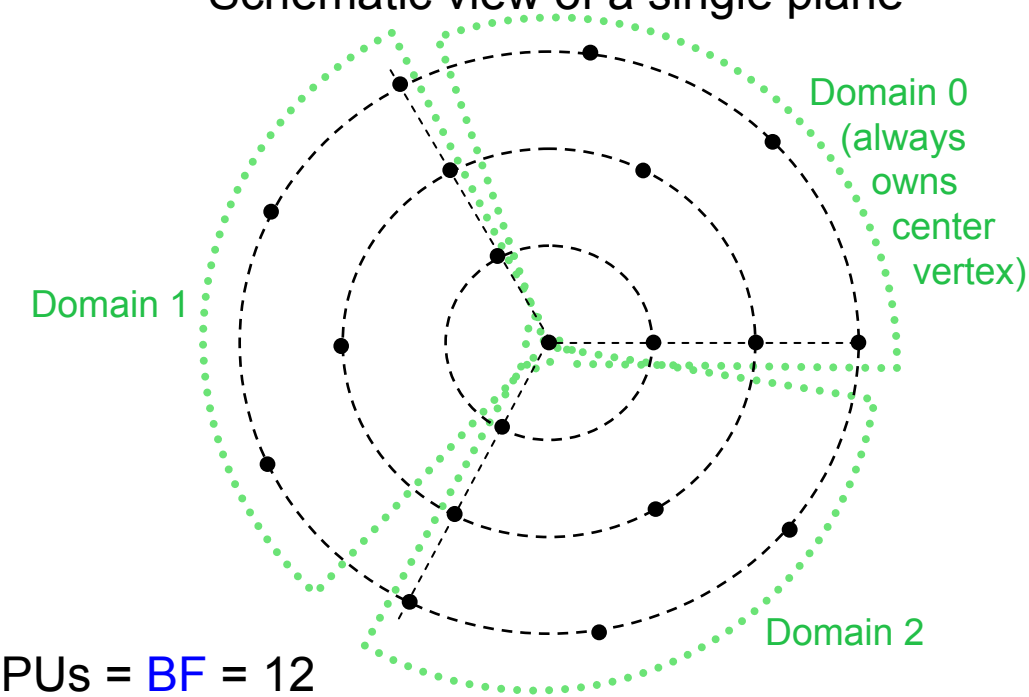
A Sample Configuration File

```
12        // A. Total number of planes in toroidal direction.
4         // B. Number of CPUs in toroidal direction (B<=A).
4         // C. Number of grids in minor radial direction.
1         // D. Number of CPUs in radial direction.
3         // E. Number of mesh partitions in theta direction (E>=3).
3         // F. Number of CPUs in theta direction.
4.71      // G. Not used.
```

Overhead view of planes



Schematic view of a single plane



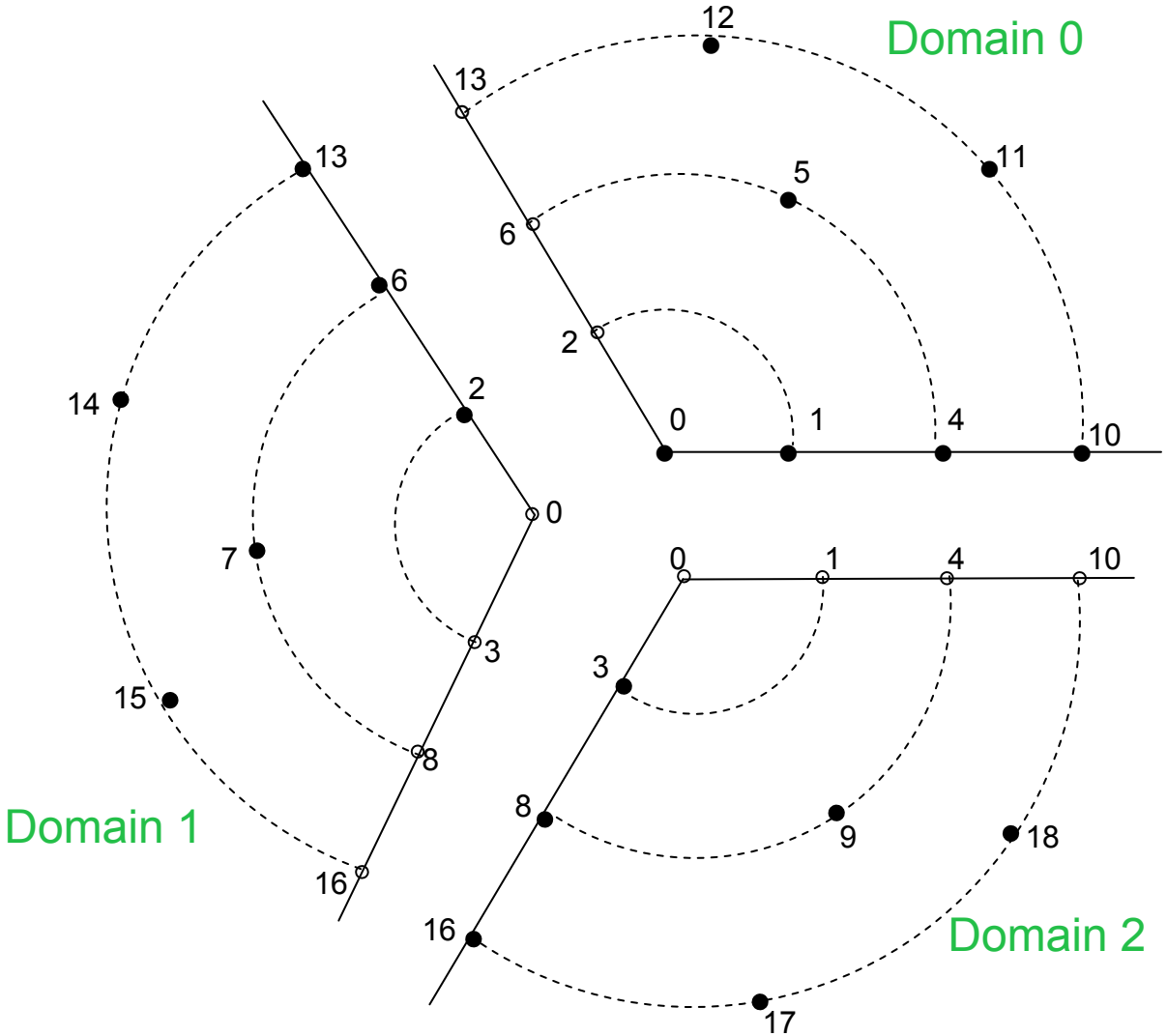
Total CPUs = BF = 12

Total Vertices N = A[1 + EC(C-1)/2] = 228

(must have `kz` ≥ A/B = 3 and `lz` ≥ N/(AF) = 6.33 in m3d/code/param1 header file.)

Domain Decomposition and Ghost Vertices

Vertex indices shown in "application" ordering.

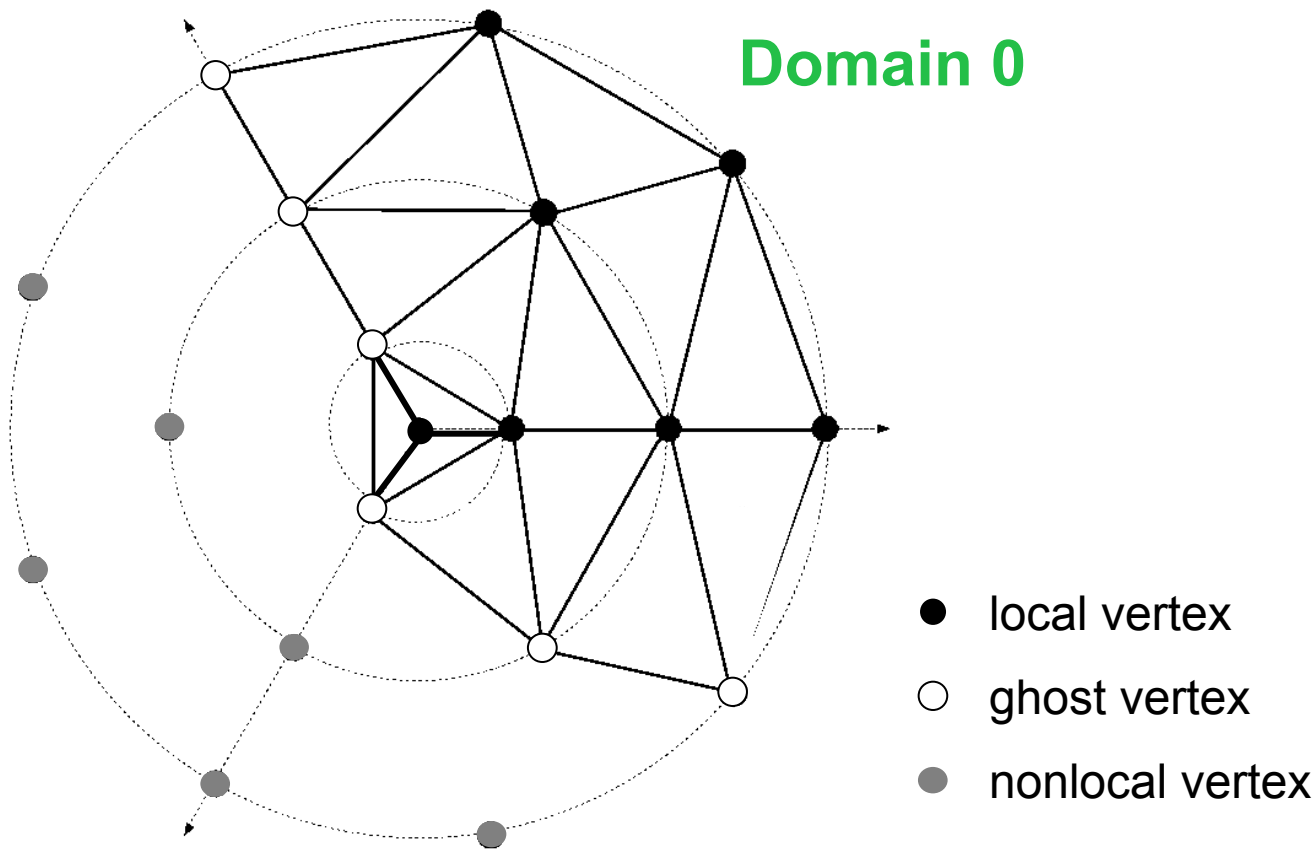


• local vertex

° ghost vertex

Triangulation

Triangulation is done wedge by wedge. All elements containing at least one local vertex are considered local.

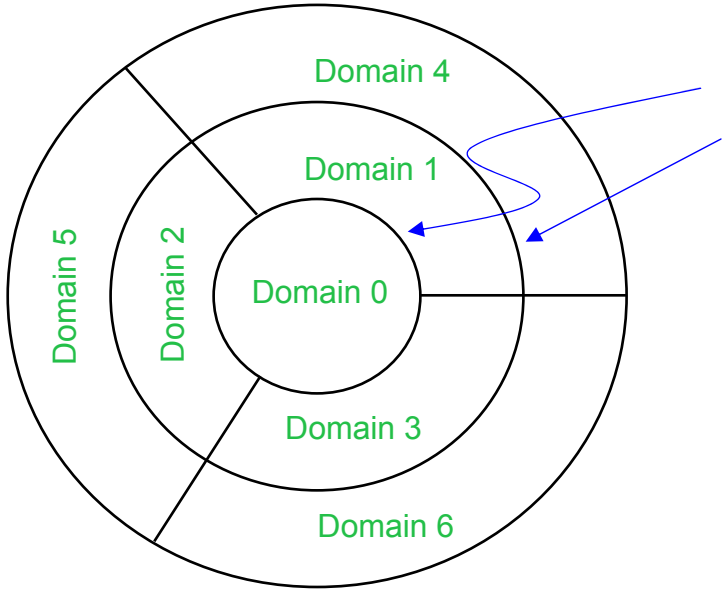


Total number of triangles in a plane = $E(C - 1)^2$.

Special Case: D > 1

```
48        // A. Total number of planes in toroidal direction.
24        // B. Number of CPUs in toroidal direction (B<=A).
100       // C. Number of grids in minor radial direction.
3         // D. Number of CPUs in radial direction.
4         // E. Number of mesh partitions in theta direction (E>=3).
3         // F. Number of CPUs in theta direction.
4.71      // G. Not used.
```

In-plane domain decomposition



Minor radial domain boundaries are computed automatically to optimize load balancing.

Total CPUs = B[F(D-1) + 1] = 168

Total Vertices N = A[1 + EC(C-1)/2] = 950,448 (same formula as D=1 case)

(must have `kz` ≥ A/B = 2 and

`lz` ≥ N/{A[F(D-1)+1]} = 2,828.71 in m3d/code/param1 header file.)

Packing the Mesh at a Flux Surface

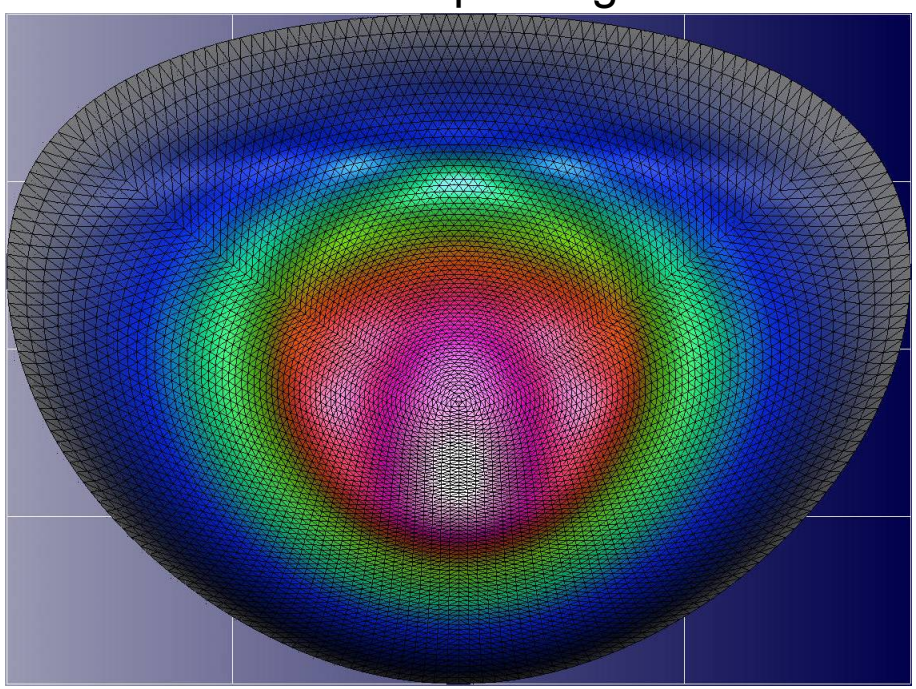
In order to resolve fine structures at a particular surface, the option exists to concentrate zones of the M3D mesh in about a given minor radius.

Command line options:

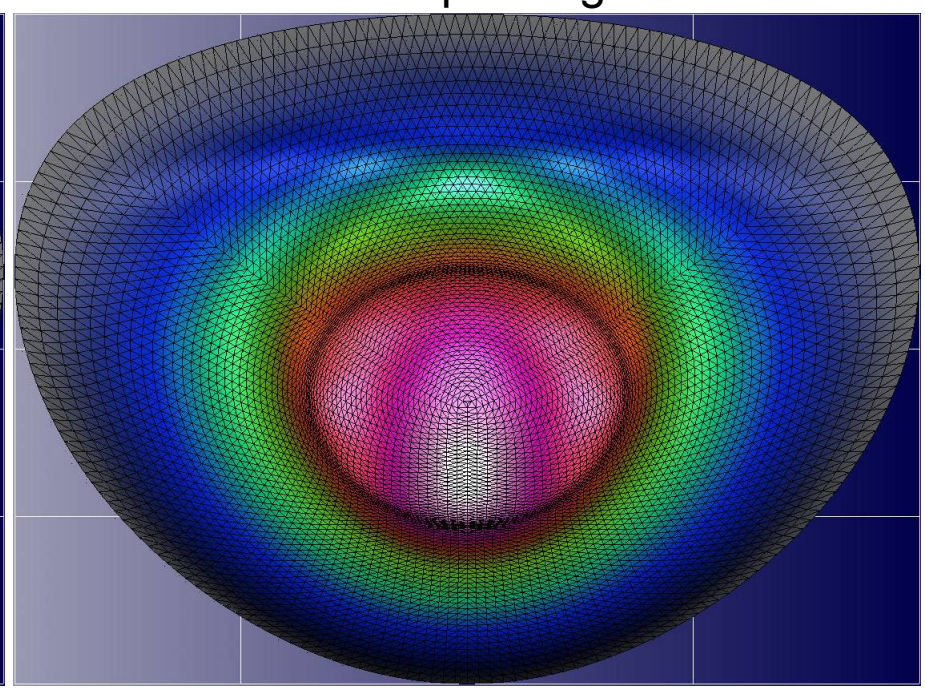
`-packingFactor` <*pf*> *Ratio of packed to unpacked mesh density.*
`-packingRadius` <x_0> *Relative position of packing surface (from 0 to 1).*
`-packingWidth` <*w*> *Relative width of peak packing area (on 0 to 1 scale).*

Example: pf=4.0; x_0=0.5; w=0.12:

Before packing



After packing



MPI Communicators

Each processor in a parallel run belongs to three separate MPI communicators:

- **Global**: for collective operations over all processors (e.g. volume integrals). Local rank: `phone.gRank`; Name: `MPI_COMM_WORLD`.

- **Poloidal**: all processors within a given range of toroidal angle ϕ. Used for operations that are collective over a constant-ϕ plane but not between different planes. Local rank: `phone.pRank`; Name: `phone.poloidalComm[phone.tRank]`.

- **Toroidal**: processors at all values of ϕ that contain the same segment of the domain-decomposed plane. Used for operations that are collective toroidally, not within planes (e.g., toroidal FFTs for pseudo-spectral runs). Local rank: `phone.tRank`; Name: `phone.toroidalComm[phone.pRank]`.

Data Storage in Fortran

Scalar physical variables in the Fortran part of M3D (primarily m1.F and mpar1.F for MHD) occupy static two-dimensional arrays.

For example, `u(1:ld, 1:ku)` holds the value of velocity stream function U at all local vertices (`1-ld`) in each local plane (`1-ku`).

`u(1:ldb,k)` are the local interior vertices in local plane `k`, while `u(ldb+1:ld,k)` are the boundary vertices.

Physical coordinates of vertices are available – the value of ϕ at plane `k` is stored in `zH(k)`, while `rmajor + xH(l,k)` and `yh(l,k)` are the values of major radius *R* and *z* respectively at vertex `l` in that plane – but the Fortran routines are otherwise unaware of the relative positions of local vertices within the global mesh or other details of parallelization.

Mathematical Operations on Fortran Data

First derivatives:

`dxdr(isymr,xx,aa)`: aa $\rightarrow \partial(xx)/\partial R$

`dxdz(isymr,xx,aa)`: aa $\rightarrow \partial(xx)/\partial z$

`dxdphi(x,isym,xx)`: xx $\rightarrow \partial(xx)/\partial\phi$, using either finite differences or an FFT-based derivative depending on compile-time options.

`div(a_r,a_z,cc)`: cc $\rightarrow \partial(a_r)/\partial R + \partial(a_z)/\partial z$

Laplacian:

`delsq(aa,bb,isy)`: bb $\rightarrow \partial^2(aa)/\partial R^2 + \partial^2(aa)/\partial z^2$

Volume Integral:

`cvol(a,b)`: b $\rightarrow \iiint a\, dV$ (b is a scalar)

Binary operators:

`gradsq(aa,bb,isy1,isy2,cc)`: cc $\rightarrow \nabla_\perp aa \cdot \nabla_\perp bb$

`gcro(a,b,isy1,isy2,cc)`: cc $\rightarrow \nabla_\perp aa \times \nabla_\perp bb \cdot \hat{\phi}$

For additional operations, see my earlier presentation.

Interpolation

The primary Fortran-callable routines for interpolation on the M3D mesh at an arbitrary point in 3D space are in **m3d/interface/interpRealSpace.c**:

```
get_weights2d(R, z, plane, pe, el, w)
```

and

```
interp3d(aa, phi, pe, el, w, val)
```

Normal usage: first call `get_weights2d()` once for each point you wish to interpolate; on input, `R` and `z` are the cylindrical coordinates and `plane` is an integer giving the index of the nearest local plane. On output, `pe` is an integer giving the rank of the processor containing the point within the local poloidal communicator; `el` is an integer giving the index of the triangle containing the point within that processor; and `w` is an array of three double-precision reals holding the appropriate weights for linear interpolation from the vertices of the triangle to the point. All three of these outputs should be stored for future reference. This routine is collective over each poloidal communicator.

Next, when interpolating a scalar variable `aa` at a particular point, `interp3d()` can be called. `phi` is the toroidal angle in radians; `pe`, `el`, and `w` are the saved values from the prior call to `get_weights2d()`; and on return `val` holds the interpolated value. Inter-plane interpolation is 3rd-order (based on the four nearest planes). This routine is globally collective, and so must be called by all processors at the same time for each point.

These routines are not guaranteed to give accurate results for stellarators.

Alternate Interpolation

The `get_weights2d()` routine performs a search through the mesh, which is robust but relatively slow, though it only must be done once per point. For interpolation within a plane, faster alternatives exist.

```
interp2d(aa,plane,pe,el,w,val)
```

works like `interp3d`, (and also requires a previous call to `get_weights2d`) but is four times as fast and is only collective over a poloidal communicator, so it may be done in parallel over several planes at once.

An alternative to all of these is `getvw(xv,yv,vert,weight)` in **m3d/interface/interpolate.c**, which is used by G. Fu's hybrid code. This routine is for in-plane interpolation; it finds elements directly rather than searching, and is not collective, avoiding interprocessor communication. On input, `xv` and `yv` are the normalized coordinates of the point relative to the M3D mesh – that is, they always refer to radial grid index `ir` = $(N_R-1)\sqrt{(xv^2 + yv^2)}$ where N_R is the number of radial grids. On output, `vert` is an array of three integers giving the indices of the vertices of the bounding triangle in the local ordering, or 0 if the vertices are nonlocal. `weight` is an array of three reals giving the corresponding weights for interpolation.

Weights for `getvw` are computed using flux coordinates rather than realspace coordinates and may be more suitable for interpolation in stellarator geometries.

Saving Data

Three methods currently exist to save M3D variable data to disk from Fortran. All have drawbacks, however, and new utilities may need to be developed to enable interprocessor communication by means of files.

1. **Saving checkpoint data:** M3D checkpoint files are written in native binary format (though a utility exists to convert them to HDF5 and back for restarts on different platforms). New variables may added to the checkpoint files simply by adding new calls to routine `parucheck(istep,ivar,var)` in `write_mpp` in **m3d/code/mpar1.F**., being careful to keep the `ivar` index sequential. A corresponding call to parrcheck must be added to the `read_mpp` routine to make the resulting file readable.

2. **Saving output in HDF5 format:** Adding data to HDF5 output is more cumbersome. Standard procedure at present is to copy the variable you are interested in to a "work" buffer such as `wa`, then replace some other variable with this one in the call to `parvars` preceding the calls to `setvname` and to `ucd3dc` in `outpp` in **mpar1.F**. It is also recommended that you relabel the new variable by altering the corresponding `setvname` call. HDF5 data is defined at the vertices at full resolution, but in single precision.

3. **Saving output in UCD format:** The procedure here is the same as for HDF5; you just need to turn on "UCD" instead of or in addition to "HDF" in the main Makefile. The format, however, is ASCII, and each processor writes a separate file, which must then be concatenated together at the end of the run. This option is not recommended.

Developing New Utilities

- There is probably room for improvement in the efficiency and accuracy of the routines described above.
- Much of the information in this talk will not apply directly to upcoming higher-order element versions of M3D.
- Interfacing with existing ORNL physics codes is likely to require the development of new routines targeted to them.
- Please let us know what you need. I and other M3D developers will be happy to work with you to make this collaboration successful.