Access patch points (on different processor) in parallel

December 5, 2011, 12:57 — **Access patch points (on different processor) in parallel** — #1

Arnoldinho
Senior Member

Arne Stahlmann
Join Date: Nov 2009
Location: Hanover, Germany
Posts: 208
Rep Power: 7

Hi all,

I have made an implementation to modify the z values of my mesh points within my solver according to some distance function during runtime. This all works well in serial mode. Now I have to make the functionality working in parallel:

I'm looping over all mesh points, and inside, I'm looping over the mesh points on a bottom boundary patch, like

Code:

```
forAll (mesh.points(), pointi)
{
    forAll(mesh.boundaryMesh()[patchi].localPoints(), dhPointi)
    {
        -> calculating some dependencies between pointi and dhPointi
    }
}
```

Of course, when running in parallel, the processor which is not holding the bottom boundary patch does not know these points. So how can I make it available?

Within the calculation, the point to patch points dependencies are stored in a vector (std::vector<std::vector<std::pair<label, double> > >) at the beginning of the simulation. During the simulation, all processors then have to have access to this vector (or at least to their 'part' containing their processor points).

Unfortunately I'm not familiar with parallel processes, so I need your help in this case!

Arne

December 6, 2011, 03:46 — #2

romant
Senior Member

Roman Thiele
Join Date: Aug 2009
Location: Stockholm, Sweden
Posts: 344
Rep Power: 10

Did you run it in parallel? I thought OF handles these things automatically.

_____
~roman

Quote

December 6, 2011, 07:35 #3

Arnoldinho
Senior Member

Arne Stahlmann
Join Date: Nov 2009
Location: Hanover, Germany
Posts: 208
Rep Power: 7

Yes, I ran it in parallel (using two cores). In my forAll loop, I'm actually not looping over all mesh points, but just a certain area/volume. If all points within that volume are on the mesh of one processor, it's all fine, even in parallel mode. Enlarging the search volume so that some mesh points are on the second processor, I get a Segmentation fault.

I could not fully get the actual position of the solver stopping to work, so it was my guess that it must come from the loop over the boundaryMesh points.

I could image two ways solving this:
1. pass the points on the second processor to the first one holding the boundary patch, and do all calculations there
2. pass the points on the boundaryMesh from first processor to the second one as well, so that calculations could be done independently.

Any hints?

Arne

Quote

December 6, 2011, 09:28 #4

Fransje
Senior Member

Francois
Join Date: Jun 2010
Location: Netherlands
Posts: 101
Rep Power: 6

Dear Arne,

It is not very clear for me what you are exactly trying to do. But I will try to give you some hints on dealing with multi-processor problems.

First of all here are a few things you have to know (if somebody thinks I'm wrong, please feel free to correct me!):

- When computing in parallel, each processor is unaware (in a computational way) of what's happening on the other processors around him, so it is also unaware of the existence and size of the variables on those processors. All it knows originally is that it has it's own physical boundary conditions (if any, as a function of how the domain is decomposed) and it has processor patches. The the communication between patches is done "independently" of the solver. (ie, the solver doesn't know anything about how its done. It only solves local aerodynamic problems...) There are, however, ways of making sure all the patches of your original domain will be seen on all processor (albeit, with size zero (if I'm not mistaken) when they don't exisit on the processor under consideration).
- When you are programming code for OpenFOAM, it will be run identically on all processors, unless you program contingencies to have it run under certain conditions. For example, you could do:

Code:

```
if( Pstream::master() == true ) // This will run only on the master processor
{
   //do something
}
```

to run code on the master processor. Or do:

Code:

```
if( Pstream::myProcNo() == 4 ) // This will run on processor 5
{
   //do something
}
```

to run something specifically on processor number 5 (index starts at 0)..
It is a good idea to check your variables conditionally if you are calling a function, or doing something which cannot cope with variables of size zero..
So have a look at the Pstream and UPstream class in the doxygen documentation.

The realization that things where done in this way in OpenFOAM was already an eye-opener for me. From here, there are a few ways to try to solve your problem:

- I would say that if you can, just decompose your domain in a way that ensure your whole bottom boundary patch AND the points you need are on the same processor. This will be the fastest walk around, involving the least amount of programming, and avoiding a "costly" inter-processor synchronization and communication. Of course, this is also the least-general solution..
- Now, if you want to go down the road of inter-processor communication, it's going to get a bit more complicated... But I'll try to give you a few pointers.. You could try doing something like this:
  1. You start by making a labelList of the size of the number of processors you have, and filled with zeros.
  2. In this variable, at the index of the local processor (remember, the code is run on all processors), you will then enter the size of the pointField you are looking at. So on each processor, the variable will be filled with zeros, except at the index of the processor, where it will contain the size of the local pointField of the boundary condition it contains.
  3. You then do a reduce operation (synchronization) over all processors, doing a sumOp(). (I will explain this later) You now have, on every processor, the same variable, containing per processor index the size of the local pointField.
  4. By doing a sum(your_label_list) you will know what the total (absolute) size of your pointField is.
  5. You now create a new pointField variable of size sum, filled with zeros (or, in your case, vector::zero)
  6. **Here comes the "difficult" part**.
     You now have to fill this variable with the local values of your b.c. pointField, in a way that first the values of the master processor are filled in, then those of the second processor, and so on. So if the size of the local pointField on the first processor is 3, on the second processor 1, and on the third processor 2, then your new variable will be filled locally as:
     On the 1st proc: (pnt_1.1, pnt_1.2, pnt_1.3, 0, 0, 0)
     On the 2nd proc: (0, 0, 0, pnt_2.1, 0, 0)
     On the 3nd proc: (0, 0, 0, 0, pnt_3.1, pnt_3.2)
  7. Then you do a reduce operation again, with a sumOp(), and you'll obtain a synchronized variable filled with: (pnt_1.1, pnt_1.2, pnt_1.3, pnt_2.1, pnt_3.1, pnt_3.2)
     Tada! You now have your total point field available on all processors!

  A (very) short introduction to reduce() operations can be found in Reduce operation for an array

- A code skeleton to obtain the pointField size per processor could be something like:

  Code:

```
unsigned int thisPntFieldSize = loc_pntField.size();

labelList pntFieldSizePerProc( Pstream::nProcs(), 0.0 );
label thisProcNb = Pstream::myProcNo();
pntFieldSizePerProc[thisProcNb] = thisPntFieldSize;

reduce( pntFieldSizePerProc, sumOp<labelList>() );
```

- And the second part could be something like:

  Code:

```
int totalSize = sum( pntFieldSizePerProc );
pointField global_pntField( totalSize, vector::zero );

// do your trick so that you fill-up your global_pnField correctly per processor
// something like:
if( thisPntFieldSize > 0 )
{
    // Sum the number of point on the processors before
    // the current processor and store it in pntsBefore
    for( int i = 0; i < thisPntFieldSize; i++)
    {
        global_pntField[ pntsBefore + i ] = loc_pntField[ i ];
     }
}
```

```
// and finish with

reduce( global_pntField, sumOp<pointField>() );
```

I hope this helps!

Kind regards,

Francois.

Arnoldinho, GDTech, konneym and 1 others like this.

Quote

December 6, 2011, 10:08 #5

Arnoldinho
Senior Member

Arne Stahlmann
Join Date: Nov 2009
Location: Hanover, Germany
Posts: 208
Rep Power: 7

Hi Francois,

Wow. Thanks a lot for taking the time to write such a detailed answer! This really helped me!

I have already started reading a few things about PStream and reduce operations, but haven't yet got really familiar with it.

Just for explanation what I am trying to achieve: For every mesh point (in search volume) I'm storing the dependency/label of the nearest points on the bottom boundary patch in xy plane incl. inverse dist. functions. This information is later used to move the inner mesh points (in z direction) depending on a distance function, which based on a bottom boundary patch (z) movement. Therefore, for every mesh point (and therefore every processor), the bottom patch point information are needed at the start of the simulation, to store the dependencies (point labels and inv. dist. weights). This vector is later, during the whole simulation, needed to calculate the new point positions in z direction. But I have to rethink if this can work at all in parallel without too much effort.

Another problem of going the "difficult" way could be that I have duplicate points, as some are of course lying on the processor patches.

Anyway, I also thought about decomposing the mesh in a way that all points and the boundary are used at the master processor only. Doing a rough estimate, this might still be well balanced up to a 16-cores node, which is OK. So I guess I will go this way first.
Nevertheless, here comes another question, maybe you have an idea: For decomposition, I have to use manual decomp., as otherwise above mentioned restriction can not be fulfilled (or is at least not efficient, e.g. when using simple decomp.). Do you know how to make a manual decomposition file, e.g. using setFields? I was also thinking of modifying the scotch decomposition method to ensure what I need...

Thanks again,

Arne

Quote

December 6, 2011, 12:54 #6

Fransje
Senior Member

Francois
Join Date: Jun 2010
Location: Netherlands
Posts: 101
Rep Power: 6

Good evening Arne,

No problem for the help. I remember when I was trying to figure out why some code was not working in parallel, and I lost a lot of time with it, so I'm just hoping I might be able to help some people avoid the same problem!

Anyways. For what you are trying to do, it is maybe not easier to work one mesh-level higher than with points? I mean with cells and faces?
Because then, not only can you easily find to which cell a boundary face belongs to (you can use the faceOwner() function, with the local patch face ID added to the patch.start()), but I think that from there, knowing the cells on the boundary, you should also quite easily be able to find the points forming the cell (from the primitiveMesh object I think?). Just an idea.

Indeed, it might be more difficult to deal with the cells on the processor boundaries, because you will have to synchronize the values of the points on both side of the processor patch. Unfortunately I don't have any experience with dynamic meshes.. :-(

I also haven't use the manual decomposition option before, so no luck there either.. But could the simple decomposition option in the decomposeParDict file maybe do the trick for you? You can specify the number of processors you want to use in (x, y, z) direction.

Kind regards,

Francois.

Quote

December 6, 2011, 13:03 #7

Arnoldinho
Senior Member

Arne Stahlmann
Join Date: Nov 2009
Location: Hanover, Germany
Posts: 208
Rep Power: 7

Hi Francois,

I thought of dealing with cells instead of faces, but for the mesh motion I'm performing, computing point locations is more robust. Using cells, there has to be some averaging when interpolating new positions to the points, which might cause some trouble.

Simple decomposition already works when using only few processors. But for some cases, I can then only decompose in z direction, which leads to a lot of processor communication/shared faces. So this not not really efficient. I'm having a look at how meshes are decomposed right now, and will hopefully fix this tomorrow.

Greetings,
Arne

Quote

September 11, 2012, 05:55 **Similar problem** #8

FabienF
New Member

Fabien Farella
Join Date: Jan 2012
Posts: 7
Rep Power: 4

Hi,

I want to perform a simple post-processing task (in parallel, using meshsearch).
Basically:

Quote:

- each processor loops over its interior field

-> current_pt=mesh.C()[cellI];
-> U_current[cellI]=U[cellI];

- if the value of field X (here mesh.C().component(2)) is above a threshold, it looks for an offset point:

-> sup_pt=current_pt+point(0.0 , 0.0, offset);
-> idx_sup = searchEngine.findCell(sup_pt)
-> U_sup[cellI]= Uint.interpolate(sup_pt,idx_sup)

- the new field Y gets the value Y[cellI]=U_curr[cellI]-U_sup[cellI]

Here is my implementation:

Quote:

```
meshSearch searchEngine(mesh);
interpolationCellPoint<vector> UInt(U);

vector U_sup=vector::zero;
vector U_curr=vector::zero;
point current_pt=point::zero;
point pt_sup=point::zero;

scalar offset=30.0;

forAll(mesh.C(), cellI)
{

        current_pt=mesh.C()[cellI];
```

```
        U_curr=U[cellI];

        if ( current_pt[2]>offset )
        {
        pt_sup=current_pt+point(0.0,0.0,offset);

        Pout << " pt :x "<< current_pt << endl;
        Pout << " pt sup: "<< pt_sup << endl;

        List<label> idx_sup(Pstream::nProcs(),0);
        List<scalar> dist_sup(Pstream::nProcs(),GREAT);
        List<vector> Usup(Pstream::nProcs(),vector::zero);

        idx_sup[Pstream::myProcNo()] = searchEngine.findNearestCell(pt_sup,cellI);
        reduce(idx_sup,maxOp<List<label> >());

        searcherSup=1000000.0;

        for (int i=0; i<Pstream::nProcs() ;i++)
        {

                dist_sup[i]=mag(mesh.C()[idx_sup[i]]-pt_sup);
                Pout << " pt sup found: "<< i << " "<< mesh.C()[idx_sup[i]] << " " <<
                dist_sup[i] <<endl;
                if (dist_sup[i]<searcherSup)
                {

                        searcherSup=dist_sup[i];
                        found_sup=i;

                }

        }
        Pout << " pt sup kept: "<< mesh.C()[idx_sup[found_sup]] << endl << endl;

        if (Pstream::myProcNo()==found_sup)
        {

                Usup[Pstream::myProcNo()]=UInt.interpolate(pt_sup,idx_sup[found_sup]);

        }

        reduce(Usup,maxOp<List<vector> >());

        U_sup = Usup[found_sup];
        Y[cellI] = U_curr-U_sup;

        Info << " U : "<< U_curr << endl;
        Info << " U sup: "<< U_sup << endl;
        }

}
Pout << "Finished! << endl;
```

It seems that the meshSearch finds the right points, but the problem is that processor2 (due to my decomposition and the offset value) finishes its task first, and the other processors seem to get stuck has soon as this happens.

(By the way, it is a simplified version of my problem. I am not trying to find offset points outside my mesh, as this sample code would suggest)

Do you have an idea why? I have been struggling for quiet a while!!!

Thanks,

Fabien

Quote

Post Reply

« Previous Thread | Next Thread »

Posting Rules

You **may not** post new threads
You **may not** post replies
You **may not** post attachments
You **may not** edit your posts

BB code is **On**
Smilies are **On**
[IMG] code is **On**
HTML code is **Off**
Trackbacks are **On**
Pingbacks are **On**
Refbacks are **On**

Forum Rules

Similar Threads

Thread	Thread Starter	Forum	Replies	Last Post
Fluent3DMeshToFoam	simvun	OpenFOAM Other Meshers: ICEM, Star, Ansys, Pointwise, GridPro, Ansa, ...	48	May 14, 2012 05:20
How to probe all points on a patch?	strikeraj	OpenFOAM Running, Solving & CFD	10	April 24, 2012 13:45
mapFields : internal edges	Gearb0x	OpenFOAM Running, Solving & CFD	3	April 19, 2010 09:02
BlockMeshmergePatchPairs	hjasak	OpenFOAM Native Meshers: blockMesh	11	August 15, 2008 07:36
Problem with rhoSimpleFoam	matteo_gautero	OpenFOAM Running, Solving & CFD	0	February 28, 2008 06:51

All times are GMT -4. The time now is 08:37.

Contact Us - CFD Online - Top