**Informal guide for using the Fetch 2 deterministic radiation transport code**

This guidance has been produced after experience running Fetch 2 on a Desktop computer, through Ubuntu14 on a VMware virtual machine with 6 cores and 8 gigabytes of ram. Many of the issues mentioned below may not be issues on a different set up. Many issues encountered on a different set up may not have been issues on this set up. The following guide is supposed to give an idea of how to set up and run problems, and give some suggestions of how to fix issues for new users, should they arise. It is by no means supposed to be a formal manual. There are some technical points about running Fetch 2 in parallel in section 5.

Matthew Staniforth

[ms825@bath.ac.uk](mailto:ms825@bath.ac.uk)

01/09/2017

--------------------------------------------------------------------------------------------------------------------------------------

**Contents**

1. Geometry and meshing

2. Diamond Schema and .feml files

3. Running the code

4. Miscellaneous

5. Possible issues and their fixes

6. Known ongoing issues

**1. Geometry and meshing**

--------------------------------------------------------------------------------------------------------------------------------------

**i) Geometry**

**ii) Meshing**

--------------------------------------------------------------------------------------------------------------------------------------

**i) Geometry**

There are various examples of geometries, built in Gmesh, in the /tests directory of fetch. If there are any queries which are not answered here, it is advised to look there, and also in the /benchmarks and /examples directories.

This guidance is for running Gmesh on Ubuntu14, although everything seems to work the same if running Gmsh on Windows instead. There appears to be a strange bug running Gmesh on Ubuntu14 where sometimes (inexplicably) the points which are in the geometry become practically invisible (very, very small - and do not respond to changes in the variable which is supposed to dictate how big points look). If this happens I just revert to building geometries in Gmesh in windows and then move those back over to Ubuntu. A small point about this is that if you do that, you need to remove the line of text which appears at the top of the .geo file when you produce a geometry using Gmesh on windows. Otherwise, that seems fine to do.

Rule #1, don't use any of the fancy features. In particular, do not use extrude. The first thing to remember about Gmesh is that it's rubbish. It appears to be quite willing to build meshes riddled with issues (duplicate elements, etc), and tell you that everything is fine. Building geometries in the following way seems to dodge any issues:

Define points

Define lines

Define surfaces

Define volumes

Define Physical Surfaces

Define Physical Volumes.

This seems intuitively obvious, but there are many features which it may be tempting to use in order to speed the process up (IE extrude). Don’t do that.

The first four are relatively self-explanatory. Don't do anything obviously dodgy and everything's usually fine. Don't define one surface in the same place as/ over the top of a surface that already exists. The same goes for lines and volumes. That would cause the mesh to generate duplicate elements in those spots, and Gmesh can't handle that (in my experience).

This makes altering a complicated mesh once it's already built a bit of a pain, because you can’t simply (for example) add in a source volume where there wasn’t one before, or change a surface so that it’s now in two segments instead of just one in order to impose different conditions on those two surfaces. However, that’s life. A good piece of advice is probably to be pretty certain what your geometry is actually going to be before you build it – as opposed to building a geometry and then trying to change it.

The last two are important. In the .feml file later you will need to define boundary conditions on all the outer surfaces - these are the surfaces which you should define as "Physical Surfaces" in Gmesh (in the .geo file - it will say something like "Physical Surface(x) = {y} and you need the number x). This is something which you have to do as well as defining every surface normally.

I so far have not had to define any internal surface as a Physical surface for any reason - I can't think of why you ever would. You only need to define the outer surfaces as physical surfaces in order to impose boundary conditions on them. You have to define all the volumes present as "Physical Volumes" in gmsh. That's because you have to assign a material to them later on. Pay attention to what order you define these in, and then take note of the region IDs (in the .geo file - it will say something like "Physical Volume(x) = {y} and you need the number x). You'll need this later (when defining materials), and it can be a pain to work backwards to find out which volume is which.

--------------------------------------------------------------------------------------------------------------------------------------

**ii) Meshing**

Once you've done all that you can mesh the geometry. For first examples it's worth going for a mesh with only about 30000 elements. Work upwards from there once things are working. If your mesh is too fine, you can change the global mesh size factor by right clicking. (Higher number = coarser mesh). The mesh will always conform to the geometry (as far as I know).

If you want the mesh to be finer in some areas than others (ie in the beamline vs the shielding), you can change the local mesh factor at each point. In the .geo file, each point actually has 4 numbers. The first three are coordinates, the last one is the mesh factor at that point. Make that smaller for all the points containing a volume and you get a finer mesh in that volume.

I've had a bug if I mesh in 3d using Delaunay. To fix this: Before meshing, right click, go on mesh options, change the 3d option to "Frontal". Delaunay seems to have a bug on Ubuntu14.

--------------------------------------------------------------------------------------------------------------------------------------

**2. Diamond Schema and .feml files**

--------------------------------------------------------------------------------------------------------------------------------------

**i) Kobayashi problem 3i example**

**ii) Housekeeping**

**iii) The .feml file**

--------------------------------------------------------------------------------------------------------------------------------------

**i) Kobayashi problem 3i example**

Things are significantly eased in the .feml file if you borrow one that's already set up. I have previously used one of the kobayashi examples from the /benchmarks directory. It seems reasonable to copy the problem3i .feml file and edit that, as that is closest to what we want.

There are a few things to take note of in this example. Firstly, if we open the .geo file in gmsh we see a beamline with a source cube at one end. If we open the .geo file in a text editor, there is a well commented geometry which was created in gmsh in exactly the manner described above. Take particular note of the Physical Surfaces and Physical Volumes.

--------------------------------------------------------------------------------------------------------------------------------------

**ii) Housekeeping**

The first thing to make sure of is that you have created a directory in which to place the .msh file and .feml file, along with the bugle96 library. Copy the .feml file from kobayashi problem 3i into this new directory. Take care to rename the .feml file as something starting with "rad\_", otherwise it doesn't work.

The other thing necessary is the file containing the bugle96 library. Do a grep on the /tests directory of fetch, and the bugle96 library will be in one of those. Copy that into the directory for your example.

--------------------------------------------------------------------------------------------------------------------------------------

**iii) The .feml file**

cd into the directory you have created containing the .msh file (the .geo file isn't at all necessary but perhaps it's nice to also have that in the same place), and run the following:

diamond -s $FETCH\_RADIATION\_RNG rad\_example.feml

where rad\_example.feml is whatever you've named the .feml file. It does necessarily need to be rad\_something.feml, otherwise it’ll break. The schema should open and currently be filled in with options from kobayashi problem 3i. We're going to change the necessary ones, but most of them can be left alone because they're defaults. The ones necessary to change are listed below, in order of where they are in the schema. If any options are not mentioned below (many are not), it's because their defaults seem fine. Ensure that at every point you click "store data" otherwise what you just entered will vanish. Hit enter after you enter a string, or that'll vanish also.

>geometry

>>mesh (CoordinateMesh)

>>>from\_file

Change the file name on the right to whatever your mesh is called (you don't need the .msh extension).

>>mesh (RadParticleCurrentMesh)

Turn this on if you're calculating surface current anywhere. If not then leave it. Similarly with the other available options. At the time of writing (270717) there is a bug with surface current calculations (higher memory usage than it should be), so better sticking with outputting average flux in a volume (this comes later).

>io

Leave everything in here alone unless you have a particular reason to change time period between each output solution.

>timestepping

If not running adapt, leave this alone. If running adapt, set this to as many adaptive steps as you want. The angle order adapt increases one angle order per timestep, so we usually set the number of timesteps to be the max angle order. More details on this under the information about angle adapt.

>particle (Neutron)

>>method (Deterministic)

>>>equation (TimeIndependent)

>>>>group\_iteration

>>>>>maximum

set this back to 1000, or just delete the node (click on it and press delete). This doesn't make the node disappear altogether; it just goes back to its default (which is 1000).

>>>group\_set(All)

Basically all the important options are under this node, and are detailed belo

>>>>number\_of\_groups

This corresponds to the number of energy groups which are being considered in the problem. In the kobayashi example, it's just one. The bugle96 library contains 47 energy groups in total. At the time of writing, if considering any energy group, you must also consider all the energy groups below it. For example, if solving a problem where the source is emitting neutrons in the 23rd energy group (where we label the energy groups 1st through 47th with 1st being high energy), you must consider also all the energy groups below that. So, your number of groups would be 25, because that is the total number of energy groups 23-47. If you just put 1, you will be only considering the lowest energy group. If you put 2, the bottom 2, etcetera. The kobayashi example in question actually uses a .radmats file instead of the bugle library, but we come back to this later. The logic is the same. Change the number of groups to however many you're using.

>>>>angular\_discretisation (RADIANT)

>>>>>radiant\_angular\_discretisation (PN)

change this to radiant\_angular\_discretisation (SN) for problems where we consider the beamline to be a vacuum and you are not doing an adapt. If you are doing an adapt, change it to (HAAR\_WAVELETS).

>>>>>>angle\_order [if using (SN)]

Change this to the order of the angular expansion. For first examples we recommend something like 8 just to check everything's working.

>>>>>>angle\_order\_polar & angle\_order\_azimuthal [if using (HAAR\_WAVELETS)]

Change this to the order of the angular expansion. These are very different to the SN order, e.g., S2 and H(1,1) both have 4 angles in 2D but they're very different above that. For each order you go up evenly in polar and azimuthal, the number of angles multiplies by 4. (IE H(2,2) is 16 angles in 2D).

>>>>>solver (PETSc)

This is the PETSc solver, and is fine for small examples (like simple beamlines). I recommend using the internal solver, because that's what the guys at Imperial said to use - both work on small examples though.

>>>>>>iterative\_method (gmres)

There's no reason to change this option unless you're using the lu preconditioner, in which case it is necessary that the iterative method is preonly.

>>>>>>>preconditioner (ilu)

There are various preconditioners available. I usually use ilu if I'm ever using the matrix based solver (that is, the PETSc solver) because it tends not to have any issues with preconditioning. Other preconditioners are also available. Try a few and see what happens. If using the sor preconditioner and when trying to run the example you're seeing "KSP\_DIVERGED\_ITS" or similar errors, try changing the preconditioner to ilu or even lu (but if you change to lu then change the iterative method to preonly).

>>>>>>>relative\_error

Playing around with this is educational. (Probably set it at 1e-7 at first and then play with it).

>>>>>solver (Internal)

If you're using this solver, you can probably now just leave it. If you're seeing a solver failure where there is a Segmentation fault, you could try changing the default coarsening\_method to a different one. Some examples using an angular adapt have also changed the angular\_stride for some reason.

>>>>>angular\_flux\_field

>>>>>>boundary\_conditions

What needs to be set here depends on the physics of the example.

If not using an incoming surface flux (this is NOT the same as having a source volume), and you want your boundary conditions to be vacuum everywhere outside (not any sort of reflective boundary), then delete both "boundary\_conditions (Vacuum)" and "boundary\_conditions (Reflect)" and then leave it. This resets the boundary conditions to their defaults, which are vacuum boundary conditions.

If using an incoming surface flux, you need to do two things. Firstly, delete both of these nodes (click on them and hit delete on the keyboard). Now open a new one. Set the surface\_ids. This (in Gmesh language) is the "Physical Surface(x)" which corresponds to the surface you want your incoming flux coming through in the .geo file. Now set up your incoming flux. That means set type(Incoming\_flux), and under distribution (Isotropic) input the scalar flux coming from each energy group. This is simply a number density per unit volume. If your mesh is in metres, that means putting a value of 10 gives you 10 neutrons per metre squared. If you have 5 energy groups, that means you have to set 5 values (separate by spaces, not commas). If you have one energy group, that means you set 1 value. If you have no neutrons coming from a particular energy group, just set that value to be zero. The energy groups are ordered high to low. Example: If number\_of\_groups is 5, and the numbers in distribution(Isotropic) are 10 5 0 0 0 that means I'm considering groups 43-47, and I have an intensity of 10 in group 43, 5 in group 44, and zero in 45, 46 and 47.

Secondly, boundary conditions on the other surfaces. This means opening another boundary\_conditions node, set the surface\_ids to be the surfaces you're applying that boundary condition to, and set type(Vacuum) or whichever type of boundary condition you're using.

If not using an incoming surface flux, you do as above except have no incoming flux boundary condition, and just set the other boundary conditions to be what you want.

In the kobayashi problem 3i example, we can see that they didn't have an incoming flux, they had a source volume (which was set under fixed\_source.

>>>>>>fixed\_source

Here we set the options if we have a source volume.

>>>>>>>source\_strength

Set the source strength as described in the case of the incoming surface flux above. Each value represents an isotropic source for each energy group in the group set.

>>>>>>>region\_ids

Set the region\_ids for your source/sources. This/These should be the one/oness corresponding to Physical Volume(x) in the .geo file, where x is the appropriate number for that volume. If this is unclear, check how this exact kobayashi problem does it (how the regions are tagged).

>>>>>angular\_adaptivity

Don't turn this on. You might think you want this if you're doing a goal based angular adapt, but you don't want this one. If you seriously want to do this (this just adapts the angle to regions of high flux - which is usually inappropriate for beamline-type problems) then go for it. The options under the node are self-explanatory.

>>>>>spatial\_adaptivity

Try this if you wish to lose both hair and many years of your life. Also recommended for people with low blood pressure.

>>>output\_quantities

In here you choose what your output quantities are – this is similar to a tally in MCNP. You pick regions in the geometry and get Fetch to output quantities in those regions. There are various options under the quantity option, and you can set more than one (as can be seen in this kobayashi example). In order to output an average flux in a region (volume), just set a quantity with quantity\_type (Average\_flux), choose which energy groups to calculate it for, and pick the region IDs which you're calculating it in. These are the region IDs as explained previously. The number you want is the number x in "Physical Volume(x)" in the .geo file, where the physical volume is the one associated with the region you're calculating it in.

>>>goal\_based\_adaptivity

If you're not doing an adapt at all, ignore this option.

If doing an angle adapt, come back to set this (and angular\_adaptivity\_options) last, after you've already set the material\_data\_set and material\_mapping options.

If you have set the material\_data\_set and the material\_mapping options (along with any other options you which you may also set before you set both this and the angular\_adaptivity\_options), then now:

-right click method (Deterministic) and click copy

-open another particle node and set particle(Adjoint)

-right click method (Deterministic) under this particle, and click paste

You have now copied all the settings for the neutron particle into the adjoint particle but without any of the adapt stuff. This is what you need to do for goal based adaptivity.

Now go back to the particle (Neutron) node.

>>>goal\_based\_adaptivity

>>>>adapt\_angle

If doing an angle adapt, turn this on.

>>>>goal

Here define the goal which you're doing the goal adapt on. This is often the end of the beamline, for example. Define this in the same way things have been defined before. Select which energy groups you're adapting for, and define the region ID (as defined in Gmesh. It's the number x in "Physical Volume(x)).

>>>>goal\_target\_errors

Set your target errors for this goal region. This is a number which scales as a function of your real error - but not linearly, quadratically, or anything else. The only thing that can be guaranteed is that if you increase this, your real error increases, and if you decrease it, your real error decreases. How they work as a function of one another is unclear at the time of my writing this. Exactly how to set this in order to make angle adapts work properly is discussed under the "Miscellaneous" section of this document, later (the penultimate section).

>>>angular\_adaptivity\_options

If you're not doing an adapt at all, ignore this option.

If you are doing an angle adapt, here you can set the max\_adaptive\_angle\_order. This is pretty important for smaller examples, as otherwise you'll run out of memory pretty fast. If you're doing Sn, this is simple what you'd expect. If doing Haar Wavelets, this is the max angle order for both polar and azimuthal angle. Accordingly, the number you set here for Haar is more like 5 or 6, as opposed so 16 or higher for Sn.

Once you have set this option (provided you followed the guidance on setting material data before you set the adapt options), you should be ready to rock and roll. See below (after the section on the .feml file) for actually running the example.

>>>spatial\_adaptivity\_options

See the section above on spatial\_adaptivity.

>>>material\_data\_set

Here we set the material data - we don't assign it to regions. That's under material\_mapping.

>>>>from\_file

Use this option if you seriously want to use a .radmats file for your material data. Highly recommended to use the bugle96 library instead. At the moment it seems like you can't have material data coming from both .radmats files and the bugle library in one example. That makes sense, because there's no reason you'd want to do that. If you want to define a vacuum using the bugle library, just pick any material and set it's mass density to zero.

>>>>from\_bugle

Here we define our materials using a library. Make sure at this point on the right to set particle\_type to Neutron, and file\_name to bugle96 (at this point bugle96 should be in the directory with your .feml file and your mesh, and it should remain named as bugle96).

>>>>>start\_group

You have to tell the code what energy group in bugle to start from. This has to correspond to the number\_of\_groups option that was set earlier. It seems counter intuitive, because we have no choice about this, but there it is. For example if doing a calculation for 5 energy groups, the start group should be 43.

>>>>>>physical\_material

Under this option, we define each material. Label each material on the right as you define them.

>>>>>>>isotopic\_composition (WeightPercent)

>>>>>>>>mass\_density

Input the mass\_density of this material, whatever it is. You can usually find this online. For example, room temperature iron is 7.85 grams per centimetre cubed. Ensure that you get the units right on this.

>>>>>>>>isotope

Input the isotope/isotopes and what percentage of this material it/they makes/make up. You have to put the element and mass number under "name" and then also the mass number under "mass\_number" again.

>>>>Convert\_cm\_to\_metres

Turn this option on if your mesh is in units of metres. Be sure that you have rescaled the source appropriately in this case.

>>>material\_mapping

>>>>region

>>>>>region\_ids

Here we define what materials go in which region. Delete the nodes already in there, and open a new one. After naming the region, you need to put the region IDs which are going to be in this region. These correspond to "Physical material(x)" in the .geo file, as in previous options.

>>>>>>physical\_material

Simply name this exactly how you named the material when you defined it in the material\_data\_set section. This is all you need to do - as long as you give it exactly the same name as how you defined it under material\_data\_set, the code does the rest. The regions you defined in the previous step will then all be mapped to by this material.

---------------------------------------

**3. Running the code**

--------------------------------------------------------------------------------------------------------------------------------------

Once you've finished the .feml file, and are ready to run the code, run

fetch -v2 rad\_example.feml -ksp\_monitor

The -v2 shows you various different outputs (useful for finding out what's wrong in the case of errors) and -ksp\_monitor shows you the convergence. There are various other switches and options, but I've never used them and I don't know what they are.

Once you've run the example, if successful (often it can be interesting to visualise the output even in the case of bad solves or solver failures after one or two adaptive steps), run paraview in the current directory (if you don't have paraview then get it), and then

click open file (top left)

select your .vtu file and open it

click apply

skip to the final timestep (the top bar)

click the button to make the legend appear (top left)

click the button to rescale the legend (two to the right of the button which makes the legend appear)

You should have something which looks nice

**4. Miscellaneous**

--------------------------------------------------------------------------------------------------------------------------------------

**1.) Making angle adapts work**

Making an angle adapt actually work can be tricky. If you set the goal\_target\_error too high, then it seems that the code won't do an adapt at all - because it doesn't need to. It's already achieved the accuracy you ask for without any adapting, because your target error is too high. Similarly, if you want for example a H(4,4) adapt, and your error is too high, it seems that the code could only do an H(3,3) adapt and then be done, because again it has attained the target error without needing the extra angular order.

If on the other hand you have for example asked for an H(4,4) adapt (by this we mean that you have set the maximum angular order to be H(4,4), and you have set the goal\_target\_error too small, then the adapt will not work properly at all, because no matter what the code does, it can't reduce the error enough. All it will do in that case is bin as many possible angles as it can into the problem practically isotropically, because that's the best it can do. In this case it would just distribute the 512 angles around the place - as opposed to in your goal region, which is where you want them.

You need to have the right recipe of goal\_target\_error vs max\_adaptive\_angle\_order in order to get the adapt to work. Basically that means that you need to tinker with the target error until something reasonable happens. You'll know if your error is too high because it won't do the adapt using all the available angles, and you'll know if your error is too small because it will distribute angles not in the manner that you'd expect given the goal region that you've defined.

**2.) Visualising angle adapts**

It's important then, given the above, to be able to visualise where Fetch is putting the angles when it does an adapt. To do this, open paraview in the current directory as before, but open the file which is called "example.Neutron.CG\_expansion\_size\_...". As before, flick through the timesteps after turning on the legend and you can see where Fetch has put the angles.

**3.) Running fetch in multiple directories/subdirectories.**

In the situation where you wish to run a script executing fetch in multiple directories (ie looping over subdirectories and executing fetch inside them), some care must be taken. If fetch fails inside a subdirectory inside a for loop, it seems that not only will fetch exit but also the iteration of the for loop you're inside may exit. The point is, if you're running fetch inside a loop, make sure it's the last command inside the loop, otherwise if fetch exits, the rest of the loop may exit too.

---------------------------------------

**5. Possible issues and their fixes**

--------------------------------------------------------------------------------------------------------------------------------------

1.) Unclear how to tag regions/volumes/surfaces

Fix: See how it's done in the various examples/benchmarks. The kobayashi benchmarks are an excellent demonstration of this.

2.) Segmentation fault

Fix: Firstly check that this is not a memory error. Sometimes the code will spit a memory error and then spit a Segmentation fault. Check above the Seg fault in the output to see if there wasn't a memory error first. If there wasn't, and you're using the internal solver, try changing your coarsening method. If that one doesn't work, change to a different one. Try all the coarsening methods before panicking. Method number 3 often works for me. If getting a seg fault when using another solver or the above doesn't work, make sure angular discretisation is RADIANT, not RADIANT\_SN. Remeshing and using a coarser mesh has worked for me as a last resort, although not ideal.

3.) No associated simplex mix

Fix: This can mean two things. Firstly, try remeshing the geometry and going again. For some reason, sometimes this just works. If not, make sure you have actually mapped a material to this region. If you have a region without a material defined in it then this error gets thrown. If you've checked both these things and it not that, then that's new.

4.) I don't know what the different coarsening methods are

Fix: Ask Steven Dargaville (nicely).

5.) I don't know what half the options in the schema are for nor what they do.

Fix: Ask Steven Dargaville (very nicely).

6.) Issues with distance units

Fix: The units of distance are centimetres. You can change this to meters if you select that option under material\_data\_set.

7.) KSP\_DIVERGED\_ITS or KSP\_DIVERGED\_DTOL

Fix: These are usually preconditioning errors. Try another preconditioner (if you're using the PETSc solver). The ilu preconditioner is pretty good, and the lu preconditioner is an alternative if ilu still doesn't work. If you're seeing these errors when using the internal (matrix-free) solver, then tell Steven Dargaville - this is completely not expected to happen. It also has not happened to me at any point, so this would be highly unusual. The preconditioner being used for the internal solver is strong enough that these issues should be avoided.

8.) Unknown incoming flux distribution

Fix: I had this error more than once under certain conditions when trying to use an incoming flux boundary condition with the PETSc solver. Changing to the internal solver fixed this.

9.) Adjoint flux does not have the same size as forward residual

Fix: If you've set up an Adjoint particle (for a goal based adapt), you should have set everything under group\_set identically there to how it is set for the Neutron particle (In fact, your Adjoint particle settings should be identical to the Neutron particle settings everywhere apart from for the adaptivity settings). The best way to do it is to just copy the group\_set node from your Neutron particle over to your Adjoint particle. This error has appeared when angle\_order is not the same for the Adjoint particle as it is for the Neutron particle.

10.) I produced a geometry in windows and it doesn't work when I move it to Ubuntu.

Fix: Yeah, this is odd. Some text is produced and appears at the top of a .geo file when you make one in windows, but Gmesh doesn't do that on Ubuntu14. I fixed this issue by removing that text. If errors persist, you could also then go ahead and produce the mesh on windows and move the Mesh over to Ubuntu, rather than produce the geometry, move that, and then mesh in Ubuntu. I've avoided trying that, because if there's issues with a .geo file produced in windows, there may well be issues with a .msh file produced in windows.

11.) Rescaling an example to be in metres hasn’t worked

Fix: Make sure you also rescale the source. If you have an example that is in centimetres and you want to change it to metres for whatever reason, you must do the following (also applies the other way round if changing from metres to centimetres, or in fact changing to whatever unit. Just use the appropriate numbers):

Rescale mesh to be in metres. Whether this means rescaling the geometry and re-meshing, or something else, the important thing is that the mesh is changed.

Turn on the “convert\_cm\_to\_metres” node under material\_data\_set.

Rescale the source intensity. If, for example, when your geometry was in centimetres you were putting a source intensity of 1, you now need to put a source intensity of 1e6, because there are 1e6 cubic centimetres in a cubic metre.

12.) Running an example in parallel isn’t working with an MPI failure before Fetch starts

Fix: Check that you have imposed proper boundary conditions on your geometry. If running locally, Fetch can cope with an example when you have not defined all the physical boundaries in your geometry as physical surfaces. It simply assumes vacuum boundary conditions. A careful check of the std out shows that fetch throws a warning in this case, and tells you that such examples won’t work in parallel.

True to its word, Fetch does not cope with this in parallel. This can lead to running an example locally, seeing that it works, scaling it up to run in parallel, and then an inexplicable failure. Best policy is to *always* define all the boundaries of your geometry as a physical boundary, but specifically in the case of running in parallel this is absolutely necessary.

---------------------------------------

**6. Known ongoing issues (without fixes)**

--------------------------------------------------------------------------------------------------------------------------------------

1.) Persistent segmentation faulting

This happens sometimes. Sometimes just changing the mesh to be even very slightly coarser has fixed things. Changing the coarsening method in the solver often also works. Sometimes that doesn't work, and seg faults persist. It's usually a signal that the problem is eating up too much memory. On the occasions when I've been having this problem, it's frequently when I'm getting close to what I know to be the memory limit on the desktop machine I'm using.

2.) Memory usage of incoming flux boundary condition

It is known to the developers that when using the incoming flux boundary condition (IE having a source surface instead of a source volume), memory usage is higher than it should be. At the current date (09/08/2017) this is known to the developers and whilst it is on the agenda, it isn't fixed yet. This of course is only a problem when you're actually approaching your memory usage limit. If you're not, then using an incoming flux is fine, because the extra memory usage won't be an issue. If you are approaching your memory limit, then there's no real DYI fix for this - defining an incredibly thin source volume is probably the best idea.

3.) Inability to have materials defined by a .radmats file and the bugle library on the same problem.

This is slightly redundant, because if you’re using the bugle library there is no reason to define materials with a .radmats file. This issue was encountered when trying to verify that setting the mass density of a material defined by the bugle library to zero is equivalent to having a vacuum. If defining shielding by the bugle library and trying to define a vacuum using a .radmats file, things appear to break down.

Matthew Staniforth 16/08/2017

Ms825@bath.ac.uk