MATLAB

- How to run algorithms developed in MATLAB
  - Basics about using MATLAB on Beagle2: why running compiled MATLAB code
  - MATLAB functions that can be compiled and run on Beagle2
  - How to run MATLAB executables via the Batch system with an example
  - Examples on how to run multiple MATLAB executables at the same time
  - Tutorial

Basics about using MATLAB on Beagle2: why running compiled MATLAB code

- The MATLAB GUI is not supported and most likely will not be in the future because according to our experience MATLAB is not very effective in exploiting massively parallel supercomputers such as Beagle2 (feel free to write to us should you feel like you have reasons to believe otherwise).
- However, compiled executables from MATLAB can be run on Beagle2. Once MATLAB programs are developed, they should be compiled using `mcc` and run as command line executables with MCR; details on how to do that can be found later in this section. In general MATLAB compiled executables do not use Beagle2 very efficiently (both in terms of CPU and memory) and this should be considered carefully when planning large calculations.
- The MCR compiler is loaded with the command `module load matlab`
- For more information on `mcc` - the MATLAB compiler
- The MATLAB compiler produces executables that in order to run require the MATLAB Compiler Runtime (MCR), a set of shared libraries that enables the execution of MATLAB files without an installed version of MATLAB. (To be precise, the Beagle2 compute nodes won't have MATLAB installed, or at least it won't work properly there, but they will have MCR available, to use MCR you do not need to load the MATLAB module.) Currently MCR is available under either `/soft/matlab/7.13/` or `/soft/mcr/v714/` (if you require other versions let us know).
- We are grateful for help from Chun-Wai Chan, Ingrid Reiser and Andrew Jamieson in developing this section.
- The following are approaches that have been tested and shown to work on Beagle; other approaches might also work or even work better. We welcome difference opinions (if possible with supporting facts) from our users (send to beagle-support@lists.uchicago.edu, mention MATLAB in the subject).

MATLAB functions that can be compiled and run on Beagle2

- In the following, we will be discussing how to turn MATLAB code into command line executables only.
- Any MATLAB function of the form `function foofunc(x1,x2,... ,xn)` can be turned into an executable using the MATLAB compiler (`mcc` after typing `module load matlab`). The calling parameters become arguments for the executable. However, those arguments will be considered as strings and will need to be edited as:

```
if (isdeployed)
    x1 = str2num(x1);
    x2 = str2num(x2);
    ...
    xn = str2num(xn);
end
```

Note that this should not affect the behavior of those functions when not compiled.

- Matrices will have to be passed as “[1 2 3]” “[4 5 6]”
- MATLAB functions can call other MATLAB functions from other files, usually leaving them in the compilation directory will be sufficient.
- Other approaches are possible, see the MATLAB mcc guide (URL provided later in this section).
- At this stage, it does not appear that there is a way to control how MATLAB creates threads or that it can run a multi-threaded program efficiently on a 32-core Cray XE6 node (MATLAB checks directly `/proc/cpuinfo` to determine how many cores are available for a calculation and uses all of them, independently from the instructions given by the `aprun` command). We recommend users to compile with the flag `-R -singleCompThread` as in `mcc -R -singleCompThread -R -nojvm -R -nodisplay -mv myapp.m -o my_app -v` where the `-v` option (verbose) displays all the the compilation steps -- e.g., it helps identify which third-party compiler is used and what environment variables are referenced.
- `mcc` generates a standalone application
- `-v` option (verbose) displays all the compilation steps -- e.g., it helps identify which third-party compiler is used and what environment variables are referenced
- `-R` specifies run-time options for MCR
  - `-R -nojvm` disables the java virtual machine
  - `-R -nodisplay` eliminates functions that would produce a display
  - `-R -singleCompThread` runs MCR single threaded

The compilation can be performed on any machine with a version of MATLAB installed, it does not need to be performed on Beagle2. However, it is important that the MCR, the runtime environment, used on Beagle2 is the same as the MATLAB version used to compile the program, you or we can install other versions of MCR for you if necessary (for example because you are compiling...
MATLAB with packages that we do not have on Beagle and are therefore using your own version of `mcc` on your machine -- all you will need to do is provide us with your version of MCRInstaller.bin -- see below for how to locate it.

- Other command line options for `mcc` can be found on the Mathworks web site: mcc quick reference guide
- The compilation can also be performed from the MATLAB GUI, but we will not discuss this option in the following.

## How to run MATLAB executables via the Batch system with an example

Built on Beagle2 with a lot of help from Chun-Wei Chan (first it was developed on SIRAF, then the tree was moved to Beagle2).

Here are a series of steps we used to run the “magic square” example from the MATLAB web site ([http://www.mathworks.com/help/toolbox/ compiler/bsl9eor.html](http://www.mathworks.com/help/toolbox/compiler/bsl9eor.html)).

First you will need to compile `magicsquare.m`, which you can find in your own installation of MATLAB (on Beagle2 is in `/soft/matlab/7.13/extern/examples/compiler/`, on other systems it will be usually something like `/usr/local/packages/matlab/extern/examples/compiler`, ask your system administrator for further details).

First you need to type `module load matlab`.

Then to compile, as explained above, we recommend the following set of options:

```bash
mcc -R -singleCompThread -R -nojvm -R -nodisplay -mv magicsquare.m -o magicsquare
```

After the compilation, a number of files will be generated:

- `mccExcludedFiles.log`: log file with the MATLAB toolboxes that are not included in MCR and therefore can’t be used in the application
- `my_app` (in this case `magicsquare`): the executable you will need to copy to Beagle
- `readme.txt`: contains information, for example *where is the version of MCRInstaller.bin for your specific MATLAB, which you will need if different from the ones available on Beagle2*
- `run_my_app.sh` (in this case `run_magicsquare.sh`): a shell script that can be used to run my_app. We recommend that you use it too avoid having to take care of too many variables in your PBS scripts. However, you will need to modify those scripts when using them on Beagle. (Previously we recommended to not use this script on Beagle, the main reason for the change is that using the script makes easier to run many copies of the same executable, which is usually what is done on Beagle.)

The following steps can be followed to run computations on Beagle:

- Then `magicsquare` and `run_magicsquare.sh` need to be copied to Beagle2 to the directory from where you will be submitting your script.
- `run_magicsquare.sh` need to be modified, specifically we suggest that you do the following (note that uniform solutions will make it easier for us to help you too):
- To prevent the various scripts from blocking each other, add the following lines at the beginning of the script, right after the initial comments (series of lines starting with `#`)

```bash
# Added to run on Beagle after August 2011 -- TMP has to be defined by the calling PBS script
tmp=`mktemp -d $TMP/matlabcachedir.XXXXXXXXXX`
echo $tmp
export MCR_CACHE_ROOT=$tmp;
# end added

# In order to remove the temporary cache directories, after the line eval
"${exe_dir}/magicsquare $args"
# add

# Added to run on Beagle after August 2011
rm -rf $tmp
# end added
```

The following script works on Beagle2 (at least it did on August 15th 2011). Note that this example is running single-threaded (because usually MATLAB does poorly when multithreaded) without MPI (-n 1) **because it is an example**. In general, it is better to make use of all the cores available on a node (pending memory constraints)! In the case of MATLAB compiled code, to run multiple instances of the same node, `-d <n_instances>` should be used instead of `-n <n_instances>`. A description of PBS directives and commands can be found at the relevant section of this wiki.
Examples on how to run multiple MATLAB executables at the same time

- Note that this can also be done using other scripting languages (e.g., other shells, swift or Python). Some of these possibilities will be covered in the near future.
- The scripts need to be modified to fit your needs; this is just an example. Feel free to contact us at beagle-support@lists.uchicago.edu if you want any help in developing these scripts.
- This scripts is taken as is from a set of simulations done on the Beagle2 supercomputer (with minimal modifications to anonymize it).
- This is a shell script that prepares and submits PBS scripts. The minimum and maximum values of n for the external loop should be provided as argument (e.g., script.sh 1 67)
- Note that the blocks send less than 32 computations per node. This is done because most of these jobs required more than the 2 GB available on each core when a Beagle2 node is used fully with 32 processes (when more memory than the 64 GB per node is used, an OOM or out of memory condition will manifest and all the jobs in the PBS script will be killed).
- The execution requires three scripts and one executable: the PBS script (the first), the script that allows to pack jobs on a node (the second), plus the wrapper for the MATLAB executable (discussed before) and the MATLAB executable.
#!/bin/bash

# name of input file
infile="whatever.mat"

# NOTE submission dir is where calculations are run
walltime="8:00:00"

# How many ht terms to run on each node
# first the lower bound, then the upper bound
# array needs to have even number of terms, obviously
blk=(1 10 11 15 16 18 19 20)
nblk=$((${#blk[@]}/2))

for ((t=0;t<$nblk;t++));do
    tmin=${blk[$(($t*2))]}  
tmax=${blk[$(($t*2+1))]}  
filename="Exe"${n}""_"${tmin}"to"${tmax}".pbs"
    runname="Exe"${n}""_"${tmin}"to"${tmax}" 
    echo "#!/bin/bash
    #PBS -N $runname
    #PBS -l walltime=$walltime
    #PBS -l mppwidth=32
    #PBS -j oe
    # Load modules and set for dynamic environment
    . /opt/modules/3.2.6.7/init/bash
    export CRAY_ROOTFS=DSL
    # echo for the e/o file from PBS the current definition of CRAY_ROOTFS
    echo "CRAY_ROOTFS is "CRAY_ROOTFS"
    # Set the paths to the MCR RUNTIME environment
    export MCRROOT=/soft/mcr/v714
    export MCRROOT=/soft/matlab/7.13/
    cd /lustre/beagle2/`whoami`/bin
    cp $infile \$PBS_O_WORKDIR
    cp run_t_real.sh t_real real_tmin_tmax.sh \$PBS_O_WORKDIR
    cd \$PBS_O_WORKDIR
    # Set up TMP and a cache root dir for MCR, it won't work if it isn't set
    mkdir -p tmp
    export TMP.existsSync/LUSTRETMP
    export TMPDIR.existsSync/LUSTRETMP
    aprun -d $((tmax-tmin+1)) -n 1 real_tmin_tmax.sh \$MCRROOT $infile \$n $tmintemp $tmax 2>&1
    qsub $filename
    done
    done

Second script

#!/bin/bash
MCRROOT=$1
infile=$2
N=$3
tmin=$4
tmax=$5
for ((t=$tmin;t<=$tmax;t++));do
    (./run_t_real.sh MCRROOT $N $t $infile ) &> $N$t$t.log &
done
wait