Getting started: performing basic operations on Beagle2

- Basics about the system
  - Basics about programming environment
- Modules and Programming Environment (PrgEnv)
- How to work on the filesystem
  - Description of the filesystem
  - HIPAA
  - Lustre
    - Useful commands on lustre
  - Striping
    - Useful commands for striping
  - How to move data to and from Beagle
- How to submit jobs
  - Projects
  - Basics about job submission on Beagle2
  - Job Submission Best Practices
  - Batch jobs
    - Commands for submitting and inquiring about jobs
    - PBS (batch) scripts
  - Aprun
  - Memory usage
- Running Swift on Beagle2
- Additional resources:
  - In case you need help/support

Note: All policies and approaches are subject to changes. While we will do our best to keep users informed of such changes, but it is not always possible to do so.

Basics about programming environment

The operating system on Beagle2 is the native Cray Linux Environment (CLE)

On login nodes: Is very similar to a conventional Linux environment.

On compute nodes is available as:

- CLE Static (which only allows the utilization of statically linked software, and it is the basic OS used for large simulations in the “Extreme Scalability Model” (ESM))
- CLE with Dynamic Shared Objects and Libraries (DSL) — see How to develop/port programs for/to Beagle

`xtnodestat` command shows:

- Current configuration of Beagle2’s nodes: which blades are compute which are service and where they are located in the machine.
- It will also provide information about the current workload of the machine.

Type `man xtnodestat` for more details. Please note that: free nodes as seen through xtnodestat does not always mean they are available for your use.

Modules and Programming Environment (PrgEnv)

Programming environments support the creation, modification, execution and debugging of programs. Programming Environments available on Beagle2 are: Cray Programming Environment and the GNU programming environment. The programming environment is managed by the `module` command. To learn more about modules see this page.

When working with the Cray Linux Environment, you will usually have to load a "module", see Environment User's Guide

Module is a "package" on a Cray system that enables you to dynamically modify the user environment by installing or uninstalling "modulefiles". Module contains commands to configure the shell environment for a particular compiler or library. It allows multiple versions of software to be installed simultaneously; the user can choose which version to use while compiling code or running their jobs.

Default compiler on Beagle is PrgEnv-cray, if you want to switch to PrgEnv-gnu:

```
module swap PrgEnv-cray PrgEnv-gnu
```

The `module` command provides a number of capabilities to the user including:
module load load a module
module unload unload a module
module swap unload a module file and load another (module switch produce the same effect)
module list listing which module files are currently loaded
module avail determining which module files can be loaded; lists all available modules on the system
module use(dir) to preprend a directory dir to the MODULEPATH environment variable. If you want to add a directory to the list where the module command looks for new modules.
module use --append dir will append the directory to MODULEPATH.
module unuse(dir) will remove directory dir from the MODULEPATH environment variable.

Note: in situations when a new compiler has to be utilized -- module swap might be a more appropriate strategy.

The modules that a user has loaded are persistent as long as you're logged in.

To add modules permanently to your environment you can add module commands to a file in your home directory called .modules. For example if you want to always use the GNU programming environment you would add:

ams@login1:~> cat ~/.modules
module unload PrgEnv-cray
module load PrgEnv-gnu

How to work on the filesystem

Description of the filesystem

Beagle now mount the following filesystems:

/home : CI home directories (read-only on compute nodes, will soon be removed)
  • Reliable for small storage of data like source code, shell scripts, etc.
  • Slow. It is not tuned for high performance parallel jobs.
  • Should not be used for calculations on Beagle!
  • 10 GB quotas and they are enforced!
  • Referenced by the environment variable $HOME

/lustre/beagle2: local Lustre filesystem (this is where batch jobs should do most of their I/O)
  • It's a parallel distributed file system.
  • Fast. High-performance.
  • Scratch filesystem. NO BACKUP.
  • Files in Lustre are subjected to purging. It is the users' responsibility to protect themselves from data loss!
  • Referenced by the environment variable $LUSTREDIR
  • 450TB of usable space
  • While there are currently no restrictions in terms of usage and capacity, these conditions will likely change.
  • Allows users to control the striping parameters when storing data on the filesystem. Tuning these parameters correctly can lead to better computation performance--see bellow.

/soft : local Cray software repository (read-only)

NOTE: Home directories are not mounted on the compute nodes (for performance reasons), so you'll always want to be working out of the Lustre scratch filesystem (/lustre/beagle2/your_user_name>). Make sure to copy everything you're working on out of your home directory to your Lustre directory and work out of that Lustre directory whenever you're on Beagle

/ufs : internal filesystem for ALPS scheduler (read-write)

/tmp, /var, /opt, /dev and so on are in general read only from any node and usually more restricted from the compute node.

NOTE: The CI Systems Group reserves the right to rebuild the Lustre filesystem at any time. While best efforts are always made to recover data, the primary focus will be to return the filesystem to availability as quickly as possible. Advance notice will be give as early as possible. Aside from unexpected disaster recovery, all attempts will be made to limit outages to necessary maintenance, reconfiguration, and reliability testing.

If you encounter undesirable behavior with the Lustre filesystem, please contact beagle-support@ci.uchicago.edu . (It is assumed that the filesystem will need some tuning as its use and activity increase.)

Research and HIPAA Privacy Protections
HIPAA’s Regulatory Scope

HIPAA’s protections focus on “individually identifiable health information,” which HIPAA defines as information in “any form or medium” that “[r]elates to the past, present, or future physical or mental health or condition of an individual; the provision of healthcare to an individual; or the past, present, or future payment for the provision of health care to an individual” (Security and Privacy 2013).

HIPAA’s protections reach only a subset of individually identifiable health information -- formally called protected health information or simply “PHI” -- created in or by what HIPAA calls covered entities. Covered entities include individual healthcare providers, healthcare provider organizations, health plans, and health information clearinghouses that engage in electronic healthcare transactions (see Health and Human Services Covered Entity Decision Charts). HIPAA’s protections for PHI extend to non-U.S. citizens’ data as well.

Some identifiable health information used for research originates outside of covered entities, and so may not be covered by HIPAA. However, you must check with your organization’s privacy authorities before assuming your situation falls outside HIPAA’s scope.

What Kinds of Users and Uses Are Covered?

HIPAA regulations set requirements for use and disclosure of PHI by covered entities, and by extension on all members of a covered entity’s workforce that have contact with PHI. HIPAA’s data protection requirements also apply “in the same manner” to business associates (and by extension to the workforce of such business associates) that perform functions using PHI on a covered entity’s behalf.

Researchers may be part of the workforce of a covered entity, or may be covered entities themselves if they are also healthcare providers. If so, they are directly affected by the HIPAA’s research rules. Researchers who meet neither of these conditions are still indirectly affected by HIPAA rules if a covered entity is the source of their data and those data meet the definition of PHI.

HIPAA’s rules on use and disclosure are generally “purpose-based” -- that is, the intended use sets the rules more than the type of data itself. The research rules discussed here are different than those for, say, treatment or treatment-related payments (relatively liberal), or for marketing or fundraising (relatively strict). A few types of data, such as psychotherapy notes do receive special protection under HIPAA. State laws also often have many categories of data with special protections, with which you should be familiar (or be in contact with an organizational official who has that knowledge).

What Constitutes "Research"?

Like the Common Rule, HIPAA defines research as a “systematic investigation, including research development, testing, and evaluation, designed to develop and contribute to generalizable knowledge” (Protection of Human Subjects 2009; Security and Privacy 2013). Note that some kinds of investigative activities that use patient data are excluded in this definition. For example:

1. Quality assessment and improvement, including outcomes evaluation and development of clinical guidelines or protocols, fall under the category of healthcare operations under HIPAA -- provided the primary aim is not obtaining generalizable knowledge.

2. Activities that aim primarily for generalizable knowledge of population health can fall into the category of public health activity under HIPAA.

The regulations are complex. So, as with the covered entity status, a determination by an organization’s IRB, designated privacy official(s), or legal counsel is usually required to assure that an activity is “not research” and therefore subject to different HIPAA rules.
Who Enforces the HIPAA Research Protections?

A covered entity may choose to rely on an IRB to assess compliance with both the FDA and Common Rule requirements and HIPAA research requirements. Alternatively, HIPAA provides that covered entities may create a Privacy Board to handle some research-related issues, notably determinations about eligibility for waivers, alterations, and exemptions from authorization processes. A covered entity may also leave some decisions about compliance with the research provisions of HIPAA to its designated privacy officer. It is critical that you understand the allocation of responsibilities at your organization.

Research subjects, like patients generally, have recourse to both your organization’s authorities and to federal and state agencies in the event they wish to file complaints about or have questions regarding an organization’s protective efforts.

As with any other planned activity related to protected health information, research must be mentioned in a privacy notice that HIPAA requires be provided by covered entities to their patients/customers. The privacy notice must include the ways in which data subjects may register complaints and report problems, either locally or with federal authorities. Every researcher should be familiar with their organization’s privacy notice, particularly the persons or departments it identifies as enforcement authorities for the organization.

HIPAA Research-Related Rules

If the data in question meet the definition of PHI and are being used for purposes that fall within HIPAA’s definition of research, HIPAA generally requires explicit written authorization (consent) from the data subject for research uses.

However, HIPAA allows for research-related access to individuals’ identifiable health data without authorization under certain circumstances:

1. The research involves only minimal risk.
2. The research is used solely for activities preparatory to research.
3. Only deceased individual’s information is used.
4. It is “grandfathered” research where all legal permissions were in place before HIPAA took effect.

Data that do not identify individuals can be used for research without specific authorization if:

1. Only fully de-identified data are used.
2. A “limited data set” is used, under an approved “data use agreement.”

Each of these conditions is described in the sections below.

Waivers of Alterations of Authorization Requirement Due to Minimal Risk

An organization’s IRB or Privacy Board (and in some organizations a designated privacy official) may determine that a waiver or alteration of the authorization requirement is appropriate. The conditions are modeled on the criteria for a waiver of informed consent in the Common Rule.

Use or disclosure of the PHI must involve no more than minimal risk to the privacy of the research subjects, and include the following elements:

- An adequate plan to protect any data identifiers from improper use and disclosure.
- An adequate plan to destroy data identifiers at the earliest opportunity consistent with conduct of the research (unless there is a health or research justification for retaining the identifiers, or such retention is otherwise required by law).
- Adequate written assurances that the PHI will not be reused or disclosed to any other individual or entity, except as required by law for authorized oversight of the research project, or for other research for which the use or disclosure of PHI would be permitted by HIPAA.
- The research could not practicably be conducted without access to and use of the PHI.
- The research could not practicably be conducted without the waiver or alteration to the authorization.

More about what counts as a data identifier is provided in the sections below on de-identified data and limited data sets.

Activities Preparatory to Research; Decedents’ Information Exceptions

HIPAA provides for two more exceptions to the authorization requirement for identifiable data:

- Where the PHI will be used solely for reviews preparatory to research (for example, for protocol development or identifying potential subjects) and will not leave the covered entity.
- Where the PHI refers solely to deceased individuals (the covered entity may ask for documentation of death of all data subjects).

In each case, the researcher must make a written or oral representation to the covered entity’s designated officials that such access is necessary for the research purposes -- someone from the IRB, the Privacy Board, or a privacy officer / designee -- who would then determine the appropriateness of the request.
Grandfathered Research

If all informed consents and other legal permissions required at the time were in place before HIPAA took effect (April 2003 in most cases), and have not changed since, a new HIPAA authorization is not required even for identified data. Obviously, this is no longer a commonly used pathway to bypass authorizations.

De-identified Data

A researcher may use fully de-identified health data without any authorization from individual data subjects. As the name implies, de-identified information must have all direct and indirect identifiers removed, to eliminate (or at least make highly improbable) re-identification using statistical techniques. De-identified information is no longer considered PHI, because by definition it is no longer individually identifiable.

HHS issued its Guidance Regarding Methods for De-identification of Protected Health Information in 2012. This guidance provides a detailed description of alternative methods, and should be considered required reading for anyone contemplating a de-identification strategy.

Under the HIPAA regulations, successful de-identification may be based on an “Expert Determination” by an “individual with appropriate knowledge” of statistical techniques who has analyzed the data set and can attest that the risk of re-identification is “very small.” (Very small is not defined in the regulations.) Alternatively, covered entities may use the “Safe Harbor” method of removing 18 types of identifying elements specified in the HIPAA regulations. In either case, the covered entity must have no actual knowledge that re-identification is possible or likely, for example by linking to other known data sets.

Limited Data Sets and Data Use Agreements

De-identification trades privacy protection for research productivity. Sometimes the trade-off is too steep, and a fully de-identified data set will not meet a research need. As an alternative, a covered entity may disclose PHI in a limited data set (LDS) to a researcher who has entered into an appropriate data use agreement. A LDS must have all direct identifiers removed; however, it may still include information that could “indirectly” identify the subject using statistical methods. That is, the disclosure risk is greater than “very small.”

The data use agreement for an LDS must:

- Delineate the permitted uses and disclosures of such information by the recipient, consistent with the purposes of research;
- Limit the individuals that can use or receive the data; and
- Require the recipient to agree not to re-identify the data or contact the individuals.

Minimum Necessary Uses and Disclosures

Uses and disclosures of data for research that are allowed to bypass the authorization requirement are still subject to the minimum necessary standard -- that is, the uses/disclosures must be no more than the minimum required for the described research purpose. A covered entity may rely on a researcher’s documentation -- or the assessment of an IRB or Privacy Board -- that the information requested is the minimum necessary for the research purpose.

By contrast, research information obtained using an authorization is not bound by the minimum necessary standard -- on the theory that the data subject has given explicit permission in accordance with the signed authorization. However, be aware that while HIPAA may not require a minimum necessary justification at all times, an IRB’s evaluation of risks and burdens on human research subjects arguably does.

Disclosure Accounting
Individuals whose health information is covered by HIPAA have the right to an “accounting of disclosures” of their PHI. In this context, a “disclosure” occurs when PHI is communicated to an outside individual or entity, including another covered entity. Access within the covered entity -- for example, by members of a research team who are all part of the same organization’s workforce -- is considered a “use” not a disclosure. There is no accounting requirement for these internal uses for research.

In addition to being limited to external disclosures, disclosure accounting is not required for:

- Disclosures made under authority of a consent/authorization, on the theory that individuals are aware of what they have expressly permitted for that research.
- Disclosures to the individual directly about him/herself.
- Limited data set disclosures subject to a data use agreement.
- De-identified information that no longer qualifies as PHI.

When an accounting is required, it must include disclosures during the six years prior to the data subject’s request, and include certain types of information depending on the size of the protocol.

While HIPAA may not require it, many organizations will require that researchers maintain logs of all disclosures from research data collections as a security measure, including transfers to other individuals within the covered entity. Electronic data storage will increasingly offer this capability cheaply and automatically; older collections will require manual logging.

Characteristics of Authorizations

If a research activity meets none of the bypassing criteria above, an authorization (consent) is required. When they are required, authorizations must be:

- In “plain language” so that individuals can understand the information contained in the form, and therefore are able to make an informed decision.
- Executed in writing, and signed by the research subject (or an authorized personal representative).

Authorizations must include a specific description of the PHI to be used or disclosed, the name(s) or other identification of individuals involved in the research, and description of each purpose of the requested use or disclosure.

HIPAA authorizations are normally required to have an explicit expiration date. In the context of research, it is sufficient to specify an expiration “event” -- such as “the end of the study.” A research authorization can also have no expiration date at all, as would be the case for a research database or repository, or other future use, though this absence must be clearly indicated.

HIPAA authorizations cannot normally be combined with other types of documents (such as a privacy notice). However, HIPAA research authorizations can be combined with any other legal permission related to the study, including an informed consent that meets Common Rule or FDA regulations or another type of authorization.

As with any informed consent document, researchers are strongly urged to rely on standard models rather than creating their own authorization forms, lest they make a critical error in format or content. Most organizations will already have standard documents available; check with your IRB, Privacy Board, or privacy officer.

If there are multiple documents that limit information use or disclosure, the most restrictive one applies. Whether in a single instrument or several, the core requirement is to provide enough information for the data subject to make an informed choice.

Revocations of Authorizations

Like other kinds of HIPAA authorizations, those for research may be revoked by the subject at any time, provided that the revocation is in writing. Revocation of an authorization is not valid to the extent that the covered entity has taken actions relying on it, such as in the provision of prior treatment. Such revocations may be limited “as necessary to maintain the integrity of the research study.”

Recruiting into Research
It is still permissible under HIPAA to discuss recruitment into research with patients for whom such involvement might be appropriate. This common practice is considered to fall within the definition of treatment, at least when the conversation is undertaken by one of the patient's healthcare providers.

Remember, however, that a data subject's information cannot generally be disclosed to a third party -- even another care provider -- for a research use without an authorization from the individual or an approved waiver, alteration, or exception to authorization.

HHS guidance on HIPAA has affirmed that recruitment efforts can qualify as a “preparatory to research” activity that would allow a researcher to identify potential research participants, and even contact them for purposes of seeking their authorization (HHS 2004). However, such efforts must be approved, and the PHI used for this purpose cannot leave the covered entity during this activity.

"Retrospective" Research

As electronic health data collections grow in scale and scope it is an increasingly common practice to “browse” them, looking for interesting patterns that could translate into research possibilities. Indeed, bio-repositories of tissue and data created just for this purpose are increasingly common, and the scope and scale of such repositories grow daily. (Retrospective analysis of paper charts hasn’t gone away either.)

Use or disclosure of PHI for retrospective research studies may be done only with patient authorization -- or with a waiver, alteration, or exception determination from an IRB or Privacy Board. It should not be difficult to meet one of the criteria for the latter for such exploratory efforts. Alternatively, the data collection itself may have been created with an explicit authorization from subjects for future research. However, remember that you generally cannot proceed on your own without some approval from an IRB, Privacy Board, or other designated governing entity.

Security Rule

Efforts to meet the Common Rule, FDA, and HIPAA regulations’ privacy requirements are only part of the researcher’s task. HIPAA also has a Security Rule that complements its Privacy Rule. The Security Rule requires that PHI collections receive appropriate information security protections for as long as they exist. If you do not know how to do that, find a resource at your organization that does. In addition to a privacy officer, HIPAA requires designation of a security official, who should be able to help assure appropriate data protection.

It is important to note that HIPAA’s requirements include reporting of security breaches and data exposures. In addition to notifying affected individuals, HHS must be notified of exposures of PHI; in addition to potentially triggering an investigation, exposures involving more than 500 persons are posted on the HHS “Breach Portal” website for all the world to see. State laws may also include breach-reporting requirements.

Conclusion

Although the specifics are lengthy, the net administrative burden that HIPAA adds to existing Common Rule and FDA regulations is generally not a large one. Compared to protocol approval generally -- and the details of informed consent particularly -- a HIPAA authorization is relatively easy. Additionally, as noted, there are several pathways around the authorization requirement.

To approve a study under the Common Rule and FDA requirements, IRBs have long been required to determine that there are adequate provisions to protect the privacy of subjects and to maintain the confidentiality of data. Where researchers are meeting those requirements, HIPAA should change very little beyond the additional “paperwork.”

As noted, HIPAA applies to covered entities and their business associates, and to the PHI that originates in or by them. Research conducted by organizations that do not qualify as such, using data that does not derive from any covered entity source, is not reached by HIPAA. In such cases, the requirements of the Common Rule and FDA remain as protections for human subjects’ privacy and other interests. The issue then is not "PHI" but what the Common Rule defines as identifiable "private information."

Here are the key points:

1. HIPAA privacy protections supplement those of other federal regulations (viz., the Common Rule and FDA), state law, and certification/accreditation requirements.
2. HIPAA protects identifiable health information (PHI) originating or held in covered entities or their business associates. De-identified data is not protected, and not all identifiable health information is considered PHI either.
3. Under HIPAA, research activity using PHI generally requires authorization. However, there are several alternatives that allow bypassing the authorization requirement.
4. Minimum necessary standards, disclosure accounting requirements, and the characteristics of authorizations (when required) must be understood by researchers when HIPAA applies.
5. Privacy protection includes a commitment to data security throughout the lifecycle of your data.
6. If you are unsure about the particulars at your organization or have questions, consult with your organization’s IRB, Privacy Board, or privacy official. For data security issues, consult with your organization’s security official.

Acknowledgements
The author would like to thank the following individuals for their editorial and content review of this and prior versions: Jaime Arango, Evelyne Bital, Helenemarie Blake, Joey Casanova, Anita Cava, Amanda Cotter-Rojas, Ken Goodman, Karen Hansen, Margaret Rankovic, Daniel Smith, and Sally Mann.

References


Additional Resources


Lustre

Useful commands on lustre:

- lfs df system configuration information
- lfs find[directory | file name] find a file or directory
- lfs quota -u$LOGNAME /login/beagle display quota

Striping

Useful commands for striping:

- lfs setstripe create a file or directory with a specific striping pattern
- lfs getstripe display file striping patterns
  - To find more about it use: man lfs
  - The default striping is 2: each file created is split across 2 OSTs (potentially double read/write bandwidth)
  - Usually good values are between one and four.
  - Striping can be set either on file or directory level.
  - Cannot change the stripe pattern on an existing file.
  - Can change the stripe pattern on a directory.
  - Striping must be set on a directory before files in it are created.
  - New files inherit the striping of the parent directory.

NOTE: Striping over too many OSTs will cause unnecessary overhead and lead to a loss in performance! We do NOT recommend changing striping settings unless you absolutely know what you are doing. Striping config is already set to Cray recommendations for a volume of that size.

How to move data to and from Beagle

Beagle is not HIPAA-compliant — do not put PHI (Protected Health Information) data on Beagle2 !!!

Make sure that you are properly handling PHI data, the consequences of mishandling could be considerable both for your and for the institutions you work for.

Factors for choosing a data movement tool:
Make sure you have permission to move such data from its source to its target if you are not the owner or the sole owner.

Consider carefully the structure of Beagle's filesystem before deciding where you move your data:
- **Relatively small files** (say < 1 GB) that should be considered permanent: `/home/<username>` (disk quota 10 GB).
- **Larger data to be used for calculations**, but which does not need to be backed up locally: `/lustre/beagle2` (currently there is no disk quota).

**Recommended data movement tools:**

- **scp/sftp**
  - quick to initiate but
  - slow and not scalable.

- **Globus Online**
  - Provides high-performance and is easy to use from either a command line or web browser.
  - Provides fault tolerant, fire-and-forget transfers.
  - For moving larger data.
  - When scp is too slow/unreliable
  - **Globus Online** See also **Globus Tools and Grid Services**

**Globus Online** addresses the challenges faced by researchers in moving, sharing, and archiving large volumes of data among distributed sites. With Globus Online, you hand-off data movement tasks to a hosted service that manages the entire operation, monitoring performance and errors, retrying failed transfers, correcting problems automatically whenever possible, and reporting status to keep you informed so that you can focus on your research. Command line and web-based interfaces are available. The command line interface, which requires only ssh to be installed on the client, is the method of choice for script-based workflows. Globus Online also has a REST-style transfer API.

After you register, simply use the **Beagle2 endpoint** "ci#beagle" as well as other sources or destinations. The Beagle2 endpoints server nodes are tuned especially for WAN data movement tasks. With a growing collection of Globus Online endpoints you'll be using the highest performing WAN-tuned systems with simplicity.

By default any file transfer command will be initiated on the service/login in node. The user can also bundle commands into a batch script and submit it to the scheduler. Users can also build multiple batch scripts with job dependency to move data to the machine using a few processors, run the jobs with a lot of processors, and then move the results off the machine. Here's an example of a batch script.

```bash
#!/bin/bash
JOB1=`qsub -lmppwidth=1 copy_input.pbs`
JOB2=`qsub -lmppwidth=128 -W depend=afterok:$JOB1 run.pbs`
JOB3=`qsub -lmppwidth=1 -W depend=afterok:$JOB2 copy_results.pbs`
```

**How to submit jobs**
Projects

A valid HPC project is required to submit jobs.

To join an HPC project visit http://www.ci.uchicago.edu/hpc/projects

`projects` to check whether or not you're a member of a project, to see what projects you're a member of (do this when you login on Beagle).

`projects --available` will tell you which are the projects that are available for your use.

`projects --set my_project_code:` to set one of the projects that are available to you as your default project.

Basics about job submission on Beagle2

To run a batch job on Beagle2:

1. Prepare a PBS script that specifies the application you want to run and the resources it will require.
   
   *Note:* Your application’s executable line must start with one of the application launch commands (aprun for ESM jobs; ccmrun for CCM jobs).

2. Submit your job PBS script using the TORQUE `qsub` command.

3. Monitor your job’s progress using the TORQUE `qstat` command, the Moab `showq` command.
When jobs are executed, they are allocated at least one node. Each node has 32 cores on Beagle2.
If a user wants to run a different computation on each of the cores of a node, Swift, a scripting language should be used. PBS scripts consist of PBS directives, comments, and executable statements (aprun).

Every executable needs to be initiated by the aprun command. It is necessary to properly match your aprun parameters with your PBS parameters.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.

If a user wants to run a different computation on each of the cores of a node, a specific scripting language should be used, such as Swift. Examples of such languages include Swift, which can run a different computation on each of the cores.
From the directory that contains the script file, type:

```
qsub myjob.pbs
```

NOTE: Scripts submitted via qsub use default bash shells, so you need to make sure you load modules or set any environmental variables you use in the submit script.

**PBS (batch) scripts**

A PBS job script is a text file you prepare that specifies which application to run and the resources required to run it. A detailed FAQ about PBS scripts is available from [HPC Job Management](#) where users can learn the basics of building their scripts. **Note:** The TORQUE directives in your PBS script **must precede your executable lines** (lines that begin with one of the application launch commands `aprun` for ESM jobs; `ccmrund` for CCM jobs, or `module load` commands); if directives occur on subsequent lines, they will be ignored. More specifically to Beagle these are some of the instructions that can be given:

```
#PBS -A my_project_code to set the project to which this run will be charged
#PBS -N job_name
#PBS -l mppwidth=<nodes*cores_per_node> is the number of processing elements (instance of an executable) requested and corresponds to the number of MPI or executable tasks. Default is one.
#PBS -l mppdepth=<threads_per_MPI_task>. Default is one. Use for OpenMP. The number cannot be larger than the number of cores per node (32). In some situations multiple threads can be run on same core, see Cray Doc: `aprun` or type `man aprun` for details.

**NOTE:** It is necessary to add `setenv OMP_NUM_THREADS=<number_of_threads>` in the PBS script before the `aprun` flags `openMP_program` if using OpenMP.

#PBS -l mppnppn= Number of processing elements (or MPI tasks) per node. PE is one instance of an executable propagated by the Application Level Placement Scheduler.

Using a smaller mppnppn number will result in fewer MPI tasks or executables (to run multiple executables per node scripts are necessary) being scheduled per node. That will give each core/PE more memory, but leave cores unused on the node, or allowing for mixed MPI/openMP executables (multiple openMP threads on cores per MPI task).

**NOTE:** We recommend you not to use the directive `mppnppn` in the batch script. If you want less than the default 32 MPI tasks per node or use OpenMP you should request all 32 cores on the desired number of nodes with the mppwidth parameter and with `aprun -N node` you will specify the number of cores per node.

#PBS -l walltime=hh:mm:ss, i.e., in hours, minutes and seconds. Be mindful that specific queues might not allow all job-time lengths.

#PBS -q queue_name, to submit a job to a specific queue (use `qstat -q` to find which are the available queues). Batch is the default queue.

#PBS -o job_output_file_name to connect as specific file to the output of the PBS script

#PBS -j oe join output and error file.

#PBS -l advres=<res_id>, if a user is running a job that requires a reservation. In order to send computations that run on it it is necessary to add this line to the PBS script.

#PBS -V Please don't use this option! This can propagate large numbers of environment variable settings from the submitting shell into a job.

ug: In the script, these instructions can be followed by other instructions and in the end by the `aprun` command, to run the executable. Otherwise you would be attempting to run your calculations on a login node and not on the reserved compute nodes!

For pure MPI scripts running a single program the **total number of nodes** requested is the number of PEs requested divided the number of nodes and rounded up.

For MPI/OpenMP tasks the **total number of nodes** will be ceiling (mppdepth*mpwidth/32). Type `man aprun` for details.

**NOTE:** Since Moab assigns entire nodes to jobs, the **total number of cores requested should be a multiple of 32.** If it is smaller, Moab will effectively round it up to the closest multiple of 32 in the sense of locking up those resources.

Type `man pbs_resources` when logged into Beagle for more information/more option.

**Example of PBS script:**
#!/bin/bash
#PBS -N myjob
#PBS -l walltime=10:00:00
#PBS -l mppwidth=544 ## ceiling ((100 (tasks)/6(tasks per node))*32(total cores per node)=544
#PBS -j oe ## join standard output and standard error-recommended!
. /opt/modules/default/init/bash

cd $PBS_O_WORKDIR
aprun -n 100 -N 6 ./myexecutable

# Job directive lines begin with #PBS. These directives tell the batch system how many nodes to reserve for your job and how long to reserve those nodes.
# $PBS_O_WORKDIR holds the path to the directory from which you submitted your job. While not required, most batch scripts have "cd $PBS_O_WORKDIR" as the first command after the directives.
# The aprun command is used to start execution of your code on Beagle2's compute nodes.

NOTE: All options may be specified as either (1) qsub command-line options (see below) or (2) as directives in the batch script as #PBS options (for batch jobs). We recommend putting your directives (options) in the script instead. Then you will have a record of the directives you used, which is useful for record-keeping as well as debugging should something go wrong.

Aprun

All codes that execute on Beagle2's compute nodes must be started with the "aprun" command. Without the "aprun" command, the code will run (if it runs at all) on the shared MOM node that executes your batch job commands.

To run aprun similar instructions should be used as given to the PBS script for qsub. Here are the equivalent aprun options.

<table>
<thead>
<tr>
<th>aprun Option</th>
<th>qsub -l Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-n 100 -N 6</td>
<td>-l mppwidth=nodes*cores_per_node</td>
<td>Width (number of PEs). Number of MPI tasks. There is 32 cores per node on Beagle2.</td>
</tr>
<tr>
<td>-d 32</td>
<td>-l mppdepth=threads_per_MPI_task</td>
<td>Depth (The number of threads to run for each PE). Number of OpenMP threads per MPI task. For OpenMP job you must also set the environment variable OMP_NUM_THREADS to this same value. Make sure that this value multiplied by the value for -N does not exceed 32.</td>
</tr>
<tr>
<td>-N 6</td>
<td>-l mppnpn=MPI_tasks_per_node</td>
<td>Number of PEs per node. Number of MPI tasks to run on each node.</td>
</tr>
<tr>
<td>-B</td>
<td>-l mppnpn=MPI_tasks_per_node</td>
<td>Reuse the width, depth, nppn and memory specified with qsub: no need to specify aprun options -n, -d, -N, and -m; aprun will exit with an error if the user specifies these with the -B optio</td>
</tr>
<tr>
<td>-S</td>
<td>Specifies the number of PEs to allocate per NUMA node. You'll get better performance if you distribute your MPI tasks among the 4 NUMA nodes (each NUMA node has 8 cores). Value can be 1-8. Default is 8</td>
<td></td>
</tr>
</tbody>
</table>

Example of batch script for running an MPI/OpenMP code using 6 nodes:

#!/bin/bash
#PBS -l mppwidth=256
#PBS -l walltime=1:00:00
. /opt/modules/default/init/bash
Memory usage

Our compute nodes have 64 GB of physical memory (2GB per core), but not all the memory is available to user programs. “System overhead” requires memory to run the node, and message passing library buffers all consume memory, as does loading the executable into the memory. Thus the precise memory available to an application varies. So if you are using all 32 cores per node, you will get a bit less than 2 GB per MPI task on average. If you see an error message, “OOM killer terminated this process.” in your job output, it means that your code has exhausted the memory available on the node (OOM stands for “out of memory”). One simple thing you can try when your code runs into an OOM error is to use more nodes and fewer cores per node. You can choose to launch fewer than 32 tasks per node to increase the memory available for each MPI task. Note that your account will be charged for all 32 cores per node, regardless of how many cores you actually use.

For aprun options refer to our wiki page or man page.

https://wiki.uchicago.edu/display/Beagle/Getting+started%3A+performing+basic+operations+on+Beagle2
https://wiki.uchicago.edu/display/Beagle/Examples+of+PBS+scripts

For example if you would like to run 64 MPI tasks and use only 16 cores per compute node:

```bash
#PBS -l mppwidth=128
aprun -n 64 -N 16 -S 3 ./a.out
```

This example uses #PBS -l mppwidth=128 because 128 cores are required and this number must be multiple of 32 (64 MPI tasks / 16 tasks used per compute node X 32 cores per compute node). Use the -S 3 option to place the 16 MPI tasks per compute node on cores from all four NUMA nodes to ensure best performance and access to all compute node memory. We need this option because the default is for aprun to pack the NUMA nodes, meaning 16 tasks on just two NUMA nodes.

Where -S Specifies the number of PEs to allocate per NUMA node. Each NUMA node has 8 cores. Value for S can be 1-8. Default is 8.

If you are using OpenMP please refer to this page:
https://wiki.uchicago.edu/display/Beagle/Examples+of+PBS+scripts

For more information see the CrayDoc page http://docs.cray.com/cgi-bin/craydoc.cgi?mode=Show&q=f=man/alpsm/31/cat1/aprun.1.html or type man aprun.

Running Swift on Beagle2

Swift is now installed on Beagle2 as a module. Swift supports a many-task computing environment for Beagle2. In this model, Swift scripts and the Swift runtime are used to submit and manage large numbers of small process executions on Beagle2’s massive number of cores. Swift is able to do this without overloading the Beagle2 scheduler by using a user space scheduler called Coasters.

- The Swift web site is here.
- Swift documentation is here.
- To get started with Swift on Beagle2 follow the steps outlined here.

Additional resources:

- Workload Management and Application Placement for the Cray Linux Environment from CrayDoc
- HPC Scheduling and HPC Job Management on job management

In case you need help/support

- please email beagle-support@lists.uchicago.edu. This will create a ticket in our ticketing system so that we can best track and resolve your issues.