Link to this document's Jupyter Notebook (./0211-PROJECT_Proposal.ipynb)

CMSE401 Project Proposal

In this milestone project you will write a project proposal submit it using a git repository that you create and share with your instructor. This repository will be used for the remainder of the course to turn in all project related files.

---

1. Project overview

Before you begin you need to do some research about the software and problems you want to explore. Your instructor has a lot of experience helping students find projects that are fun/interesting and will meet all of the class requirements. Please schedule an appointment with your instructor either during office hours or by appointment.

Class projects will include Three key milestones:

- **Milestone 1: Project Proposals** - (This assignment) Overview of what you would like to do for your projects. What software will you explore, what science problem you will solve and what are the expected outcomes. Also the creation of a git repository.
- **Milestone 2: Software Exploration** - You will explore software NOT covered in this course. Software may include a library, programming language, hardware, scientific code, etc). The project will require you to learn about the software, get it installed and working on the HPC (or equivalent) and write up a short tutorial with examples with enough details so that your peers can get it up and running). See 0325-PROJECT_Part1 (0325-PROJECT_Part1.ipynb) for more information.
- **Milestone 3: Benchmark and Optimization** - You will identify an existing scientific/engineering problem that currently runs slow. You will either write a new program form scratch or modify an existing code to run in parallel. You may also modify an existing parallel code to run in parallel faster. You may, but are not required to use the software you explored in milestone 2 of the project. See 0415-PROJECT_Part2 (0415-PROJECT_Part2.ipynb) for more information.

Software ideas for Milestone 1

```
OpenFlow
Blast
Petsci
OpenMM
MKL
BLAS
FFTW
Kokkos
Charm++
Python Numba
FPGA
Tensorflow
GPU
OpenACC
OpenCV
MakeFlow
Hadoop
Tensorflows
Torch
Twister
Condor
XSEDE
```

---

2. Project Proposal

Please write a 1-page proposal about what you plan to do for your project. If you do not have any ideas you should go talk with your instructor. The proposal should consist of the following:

- Project title
- Motivating picture
- Abstract - summary about why you picked. your project and what it is going to cover
- Schedule - Weekly schedule of what you plan on doing (so you don't fall behind)
- Software Exploration - Which application did you pick out for your software exploration
- Benchmark and optimization - What problem are you trying to make go faster

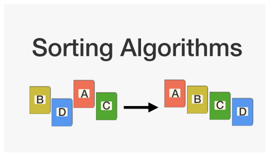
The following is a basic template you can use for your project proposal. Your instructor highly recommends using either markdown or jupyternotebooks when writing your proposal.

---- START TEMPLATE ----

Parallelizing Sorting Algorithms

By Michael El-Hage

✅ Replace the following with a picture that "defines" your project. This could be software logo, an expected outcome of your project, or a graphical representation of the research area.



Abstract

✅ Provide a short paragraph about the domain you are going to cover in your project (research area, sports, economics, etc). Explain why you picked this domain (i.e. what is your motivation). Explain how computation is used I this domain. Give a short description that describes the software are you want to explore, what hardware are you going to use during this exploration, what you hope to benchmark, what you hope to speed up and and what will be a successful outcome.

In this project I will be exploring the domain of sorting algorithms. My motivation for choosing this domain is to gain a deeper understanding of parallel programming techniques and to evaluate the performance of parallelizing sorting algorithms. Computation plays a significant role in sorting algorithms, as there is often an issue of sorting large amounts of data in a short amount of time. Parallelization of these sorting algorithms can be useful in speeding up these time consuming processes. The software I will be explorering will be a variety of sorting algorithms such as quick sort, mergesort, bubble sort, etc. I will be using the hpcc to help me conduct this study as well as my own personal computer to compare the time differences. A successful outcome for this project would be a significant speedup in sorting algorithms given a large dataset.

Schedule

✅ provide a schedule for you to complete all of your milestones on time. Include what you plan to work on each week from now until when the projects are due:

- Thursday February 11 - Project Proposal Milestone Due
- Week of February 15 - Get more familiar with parallel programming and technical requirements for the project
- Week of February 22 - Choose a selection of sorting algorithms I plan to make run faster
- Week of March 1 - Implement sorting algorithms that I want to parallelize
- Week of March 8 - contiue implementing different sorting algorithms
- Week of March 15 - begin implementing parallel versions of algorithms
- Week of March 21 - continue implementing communicating sorting algorithm
- Week of March 22 - testing and debugging code, finalizing code
- Thursday March 25 - Project Part 1 Due
- Week of March 28 - conduct time study on scripts (different languages, different versions, etc.)
- Week of March 29 - continue time studies
- Week of April 5 - begin writing up report and creating presentation
- Week of April 12 - finish up report and presentation
- Thursday April 15 - Final Project due

Part 1 Software Exploration

✅ Provide a more detailed paragraph about the software you are going to review for the first part of the assignment. Again this will vary based on project but provide any links (references) and information on where you plan to start. Also descrbe what you expect as the outcome of this step. Will you have a list of instructions for running the code on the HPC or maybe a folder in a format compatible with `getexample` ?

I will be reviewing existing sorting algorithms and be parallelizing them to run faster. Some specific algorithms I will be reviewing will be quick sort, mergesort, and bubble sort. OpeMP and MPI will be very helpful with providing support for my project. Some resources I will be using are the OpenMP specifications, and the MPI specifcations pages. I will have a list of instructions for running the code in a readme file on the HPC.

Part 2 Benchmark and Optimization

✅ Explain what code you are going to benchmark. Do you know if it already can be run in parallel? If so what types of things are you going to look for to optimize the though put of the software. How will you evaluate the software to know if the project is successful? What do you hope/expect to deliver and a successful outcome in your final report? For example, you plan to implement `openmp` and demonstrate 20 times speedup in the code or I plan to identify a bottleneck in the code and demonstrate a 4x speed up the slow.

The code that I will be benchmarking will be code used for existing sorting algorithms. I will be implementing the algorithms in a non parallel version and then create a parallel version after, evaluating the time required to run the code given a data set. I will be looking for ways to optimize my software through evaluating code structure, data distribution, etc. I would evaluate the performance of the code by measuring the execution time of each script and algorithm I make and comparing it to the parallel version of the algorithm. My project would be sucessful if the parallel version of my sorting algorithms are substantially faster than the original version. I am hoping and expecting that through creating parallel versions of the sorting algorithms, I will create a faster execution time of the algorithms.

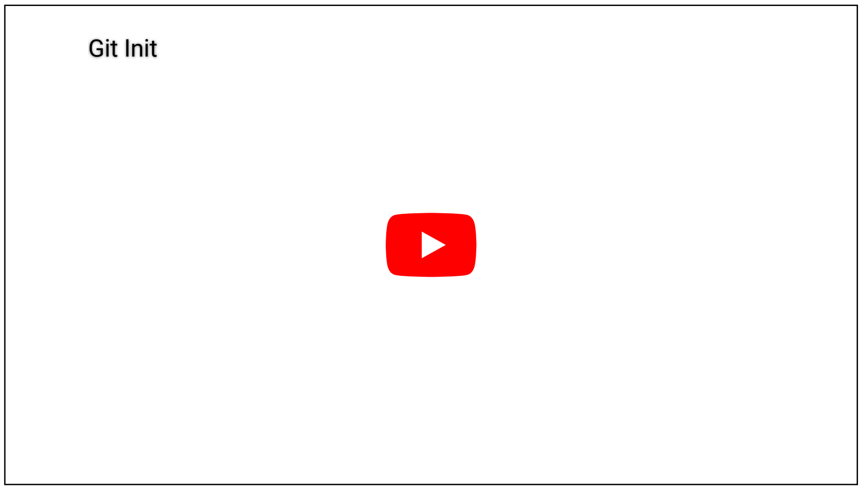
---

3. Setting up your GIT repository

To submit your proposal you are going to create a project folder, commit your proposal to the folder and share your git repository with your instructor. The following videos from the Getting to know git (Tutorial) (https://msu-cmse-courses.github.io/cmse401-S23-student/0000--Jupyter-Getting-Started-Guide.html) and may be helpful setting up your repository.

In [1]:
```
# Git init intorduction https://www.youtube.com/playlist?list=PLqPfbT7gwVP_AlE6HeDQUJsG4nUbGyeh3
from IPython.display import YouTubeVideo
YouTubeVideo("IAAv4DjYYUA",width=640,height=360, cc_load_policy=True)
```

Out[1]:



The following video are instructions specifically for how to use the MSU Gitlab

In [ ]:
```
#Inicializing Repository using Gitlab https://www.youtube.com/playlist?list=PLqPfbT7gwVP_AlE6HeDQUJsG4nUbGyeh3
from IPython.display import YouTubeVideo
YouTubeVideo("6_cegMFG0Pw",width=640,height=360, cc_load_policy=True)
```

The following are instructions for how to use the more general Github:

In [ ]:
```
#Inicializing Repository using Github
from IPython.display import YouTubeVideo
YouTubeVideo("dpeHlFm8SYU",width=640,height=360, cc_load_policy=True)
```

As you update and change the files in your repository you will need to push those changes to gitlab or github. The following instructions walk you though this process:

In [ ]:
```
#Git Add Committ https://www.youtube.com/playlist?list=PLqPfbT7gwVP_AlE6HeDQUJsG4nUbGyeh3

from IPython.display import YouTubeVideo
YouTubeVideo("GTM-h5xX2Lk",width=640,height=360, cc_load_policy=True)
```

What not to include (building a .gitignore file)

First thing we want to teach is is that not everything should go into a git repository. i.e. we do not want to bloat our repository with unwanted files. The git repository works best with Text files that represent "source" code and not compiled or generated code. Here are some basic guidelines of what not to include:

- `.ipynb_checkpoint` - These folders are generated when you run jupyter notebooks. They are "temporary" compiled folders that will change each time you run your notebook and should not be included in your repository.
- `__pychache__` - Similar to .ipynb_checkpoint folders these folders are often generated when running python scripts and should not be included in your repository.
- ***Other "Temporary" files*** - Temporary files are generated by all types of software and often start with a special characters such as the dot (.) or the tilde (~). For example many text editors generate temporary files to save a document in case of a program crash. Do not include temporary files in your repository.
- ***Compiled Code*** - Programs such as C and FORTRAN must compile their code to an executable in order to run on your computer. These compiled codes are not editable and should be left out of your repository. Instead it is better to include instructions for compiling the source code as part of your repository.
- ***Program Output*** - Do not include any program output in your repository (unless for very specific reasons such as documentation, testing, or figures in your final report). Assume that any output that can be generated by the source code should not be included with the source code (it is redundant).

A good rule of thumb is that if you did not generate the file and/or do not know what it is you probably do NOT want to include it in your repository.

WARNING do not blindly add all files to your repository with the * (star) syntax. This is bad practice. For example do NOT do the following:

```
git add * #THIS IS BAD!!!!
```

Other files to avoid

In addition to the above files it is good to avoid any type of "Binary" file (with a few exceptions). As stated early, git works best with text files so it can easily track changes. Some example binary files to avoid include:

- ***Large Data files*** Although it is good to include a few example inputs to your software, avoid using entire datasets. It is best to store these files someplace else.
- ***Non-Text formats*** such as Word, Excel or PowerPoint documents should be avoided. These tend to change each time they are opened even if the core text does not change. it is better to use an alternative text example.

Note: one exception to the above rules are image files (ex jpg or png) that are used to help markdown or in the documentation. It is typically okay to include these since they tend to get included only once and do not change much as the project evolves.

The `.gitignore` (typically read "dot git ignore") is aa text file that contains a list of regular expressions (we will learn more about these later) that specify names of files we do not want to include in a git repository.

.gitignore file

The `.gitignore` (read "dot git ignore") file is used to help keep unwanted files out of your project. Each line `.gitignore` file are filenames you want git to ignore. For example, based on what we said above, a good place to start on your `.gitignore` file would be the following two lines:

```
.ipynbcheckpoint
__pychache__
```

What should go into a .gitignore depends a lot on the type of project. However, you don't need to invent these from scratch. For example, you could just copy the .gitignore file from the course repository or find one on the internet. If you are using github I think it can also automatically create a .gitignore file for you if you specify your project as a python project.

In [ ]:

```python
#Hidden files https://www.youtube.com/playlist?list=PLqPfbT7gwVP_AlE6HeDQUJsG4nUbGyeh3

from IPython.display import YouTubeVideo
YouTubeVideo("kzI-mPSY8y4",width=640,height=360, cc_load_policy=True)
```

Avoid Spaces in file names

When you name all of the files and folders inside of a repository, it is important that your names ***DO NOT include spaces***. Although all modern computer's have ways to accept names with spaces do not use them. Instead use underscores (_) or `CamleCase` (No spaces and capital letters at the beginning of each word in the name). Avoiding spaces in your names will ***ALWAYS*** save time in the long run.

Always Use Relative Paths

In your code there are two basic ways to determine the location of a folder inside your computer; Relative Paths and Absolute Paths. A relative path is a path starting from your current directory and an absolute path is is a path starting from your computer's "root" directory.

- Relative paths typically start with a single dot (.), representing the currecnt directory, or two double dots (..) representing the current directories parent folder.
- Absolute paths typically start at the global root directory (/) on a Linux or Mac machine or with a drive label (ex C:) on a windows machine.

ALWAYS use relative paths in your git repository. This ensures that others will be able to use your software if they download it onto their computer. For example:

```
Good: ./data/  or ../data/ is a relative path to a child directory or sibling directory called data.
Bad (not acceptable): C:/research/data or /mnt/home/data are absolute paths to a data directory
```

Jupyter notebook files in git repositories

Turns out that Jupyter notebook files and git repositories work very poorly together. Jupyter notebook files are a unique combination of source and program generated information. So, everytime you run a jupyter file it can add output cells which make git think you you changed something important. In many cases it is just a few numbers or some output text. When you run the `git status` command it always looks like jupyter notebook files have changed even when they have not changed.

A good rule of thumb is to clear all of the output files before committing any changes to jupyter notebook files.

- Open the jupyter notebook file
- Select "Reset Kernel and clear output" from the menu
- Save the notebook file.
- Do your "git add" and "git commit" commands

The following video goes though why we have to treat jupyter notebooks this way:

Direct Link (https://www.youtube.com/embed/79hW_TzLos8)

In [ ]:
```python
# Jupyter vs. Git -- https://www.youtube.com/playlist?list=PLqPfbT7gwVP_AlE6HeDQUJsG4nUbGyeh3
from IPython.display import YouTubeVideo
YouTubeVideo("79hW_TzLos8",width=640,height=360, cc_load_policy=True)
```

4. Turning in your Project

In order to turn in your GIT repository you just need to give the instructors and classmates the permissions to clone the repository and provide the full git command. Please use the following form to submit this information.

Or if you prefer, you can use the Direct Link (https://docs.google.com/forms/d/e/1FAIpQLSe7E2CNUuAsRf-UyEROPB2ESA_f9OyARExk4gnnCZiPR-lQDQ/viewform)

In [ ]:
```python
from IPython.display import HTML
HTML(
"""
<iframe
    src="https://docs.google.com/forms/d/e/1FAIpQLSe7E2CNUuAsRf-UyEROPB2ESA_f9OyARExk4gnnCZiPR-lQDQ/viewform"
    width="100%"
    height="1100px"
    frameborder="0"
    marginheight="0"
    marginwidth="0">
    Loading...
</iframe>
"""
)
```

4. Rubric

I am not a big fan of giving student's detailed rubrics like this one because i think it is important for students to be able to determine on their own what is "good" work and what is not. However, not having a rubric is also not really fair because it can be difficult to know what an instructor will expect from you; this is especially true the first time something is turned in. Please use the following as a guild and I reserve the right to change my grading slightly based on the submissions and effort:

```
(50 points) Proposal
    - (5 points) Is there a project title?
    - (5 points) Is there a motivating picture included and referenced?
    - (10 points) Is the abstract well written and easy to understand?
    - (5 points) Are there due dates and milestones in the schedule?
    - (10 points) Are the goals for Part1 clear from the proposal, including links and references?
    - (10 points) Are the goals for Part2 clear from the proposal, including links and references?
    - (5 points) Is the proposal written in ipynb or md file?

(50 points) Correctly setting up your git repository
    - (10 points) Was everything turned in on time?
    - (10 points) Is there a gitignore file (does it work)?
    - (10 points) Are the permissions to the git repository set up correctly?
    - (5 points) Does the project use correct Filenames?
    - (5 points) Does the project have any temporary or hidden files that should not be included?
    - (10 points) Were all directions followed?

100 points total
```

Congratulations, you are done!

Your instructor can download your git report using the link you provided in the above Google form.

Written by Dr. Dirk Colbry, Michigan State University

(http://creativecommons.org/licenses/by-nc/4.0/)
This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License (http://creativecommons.org/licenses/by-nc/4.0/).
