Infusing High-Performance Computing and Machine Learning in Mechanical Engineering Education

Christy Dunlap[1], Jeff Pummill[2], Han Hu[1]
[1]Department of Mechanical Engineering, University of Arkansas, Fayetteville, AR 72701
[2]Arkansas High Performance Computing Center, University of Arkansas, Fayetteville, AR 72701

Abstract

This paper presents the integration of parallel computing into engineering education. This paper will discuss the role of scientific computing in molecular dynamics simulations, density functional theory simulations, and deep learning model training. It is becoming more critical for students to have the capability of running simulations or train machine learning models and when needed to improve computational times with the aid of supercomputers. This paper will cover current applications of scientific computing in research areas, upscaling on supercomputers (with an emphasis on machine learning models), how to use supercomputers, and ideas on integrating the topics into the mechanical engineering curriculum.

Keywords

Supercomputing, Wafer-Scale Engine, Machine Learning, Numerical Simulation

Introduction

Scientific computing applications are emerging in several fields, in the form of simulations, machine learning model training, etc. Simulations are used in heat transfer, for example, Sheikholeslami et al. used finite element method to simulate the solidification of NEPCM.[1] They ran this simulation to determine the effect of metallic fins and nanoparticles on the performance of the system. They are also used to model fluids, Malki et al. used computational fluid dynamics and BEM in simulations for understanding tidal stream devices.[2] They simulated different turbine array layouts to predict their performances. López et al. used computational fluid dynamics with Ansys and OpenFOAM to study erosion.[3] Molecular dynamics and DFT are widely used in materials. Wei et al. combine both machine learning and molecular dynamics to improve accuracy for a relatively low computational cost.[4] They used a GAP machine learning model to predict the potentials used in the simulation.

Machine learning is probably the broadest method and rapidly growing as the number of databases increases. Wang et al. used a neural network to predict the instability of a slope to prevent landslides.[5] Our lab, in particular, uses machine learning in analyzing two-phase cooling. We have done some work developing an acoustic regression model. This model is to be used in predicting the heat flux of a pool boiling experiment from hydrophone data. Another application in the scope of our lab is using high-speed images of boiling experiments to predict heat flux or segment the images into boiling regimes. This work is done to better understand the instabilities (i.e. critical heat flux) that limit two-phase cooling.

The mechanical engineering undergraduate curriculum at the University of Arkansas develops the student's understanding of the governing equations behind concepts such as heat transfer and fluid dynamics. Students learn how to hand solve the Navier-Stokes equations for extremely simplified examples or how to use Heisler charts in heat transfer. It is important to understand the equations and how one would go about solving them but, the general curriculum lacks expansion on computational methods. The required course, computational methods, introduce some techniques for approximating equations of simple examples (e.g., spring-mass systems) and they are relatively cheap computationally to just introduce the concepts.

Due to this, at the end of the curriculum, students understand different computational techniques as well as different general equations but there is a gap on how to connect the two and understand how to use relevant software available, such as LAMMPS, Python libraries, or COMSOL. Molecular dynamics, density functional theorem (DFT), and machine learning are becoming more important to several fields of research. Machine learning in particular is being incorporated into essentially every field. It is important for engineers to have some knowledge of machine learning to keep up with the change and leverage the capability of machine learning as a tool. There is a new machine learning elective offered at the University of Arkansas for mechanical engineering, but it would also be good to see other main courses integrate some examples of machine learning into their curriculum.

The mechanical engineering curriculum also lacks training for students in using large datasets or large simulations. Most of the work is done on smaller subsets of data that can be easily run on their personal computer or they are just given the governing equations. This structure allows students to learn the methods, but there is a critical piece of information missing, how to run and upscale their code. Once in the field, there will be a need to process larger datasets or run larger/ longer simulations. This disconnect can be resolved through the integration of supercomputing and more computational methods within the courses. This can be through the actual use of supercomputers or even by incorporating examples explaining how the code would be run on a supercomputer. Molecular dynamic simulations, although much faster than DFT can take a significant amount of time to run depending on the number of atoms, equations used, and length of the simulation. DFT simulations will take even longer but generate more accurate results. Molecular dynamic simulation, DFT, and training machine learning models can all benefit from using supercomputers for speedups.

Programming skills are becoming an essential part of most jobs. The mechanical engineering curriculum introduces MATLAB and Visual Basic in Excel, but MATLAB can be expensive to purchase outside of school and Excel cannot handle larger datasets as well as other programming languages. Python would be a good introductory language for all scientists and engineers because of its convenient IDEs (e.g., Jupyter Notebook) and simple syntax which makes it much more "readable" for students that are new to programming. Python is now a very mature language and while not as fast as compiled codes like C++ and Fortran, it does possess all of the necessary functions and libraries for a wide array of tasks. It would also be good to introduce open source packages for running simulations such as OpenFOAM.

Supercomputers

Supercomputers provide essential speed ups for code that would ordinarily take a significant amount of time. Supercomputers make it possible to run code over multiple CPUs to improve performance and also provide massive pools of memory for large data problems that require them. GPUs have been shown to improve the speed of machine learning models, especially CNNs. Our lab has access to two supercomputers, bridges2 and Neocortex. Neocortex is a new generation of supercomputers, known as wafer-scale engines used specifically for machine learning model training. To show the benefit of supercomputing several cases were run for machine learning, DFT simulations, and molecular dynamics simulations on a few different supercomputers.

Table 1: Supercomputers, their location, and type.

Super Computer	Location	Type
Pinnacle	AHPCC	CPU
Bridges2 RM	PSC	CPU
Bridges2 GPU	PSC	GPU Nvidia Tesla V100
Neocortex	PSC	GPU (wafer-scale engine) Cerebras-2

Machine learning consists of two primary types; supervised and unsupervised learning. In supervised learning, a model is trained with both input and output data. The model is fit to best match the expected output so it requires both input and labels. While in unsupervised learning, no labels are necessary. Within supervised learning, there are many model architectures. The simplest is a multilayer perceptron (MLP). A mlp is composed of layers of "neurons". Where each neuron output is defined as: $\boldsymbol{out} = \boldsymbol{\sigma}((\sum \boldsymbol{input_i} * \boldsymbol{W_i}) + \boldsymbol{b})$ and is commonly represented as shown in figure 1. Where σ is a specified activation function. Once the structure is defined, the model is fit to the data by updating the weights and biases.

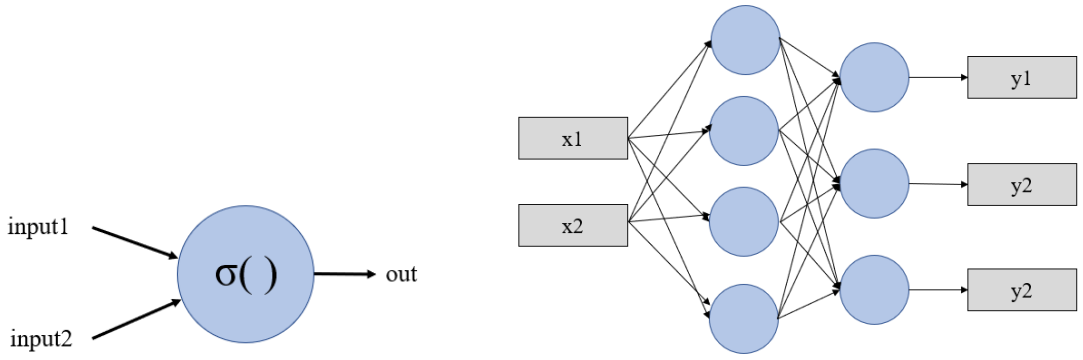

Figure 1: Left: Representation of neuron in machine learning. Right: Example figure of a multilayer perceptron. It is constructed by combining neurons. This particular model has 2 layers and the first layer has 4 neurons and the 2nd layer has 3 neurons.

Another type of machine learning model is a convolutional neural network. This architecture is commonly used in image processing because it is good at extracting features.

Three machine learning cases were run as shown in table 2. ML1 and ML2 are the same multilayer perceptron code just on two different supercomputers. ML3 is a convolutional neural network made with TensorFlow and ran on bridges2 for different numbers of GPUs and is shown in figure 2.[6]

Table 2: Machine learning cases ran on different supercomputers.

Case ID	ML Case	Dataset	Cluster
ML1	Simple MLP	Images	Neocortex
ML2	Simple MLP	Images	Bridges2 GPU
ML3	CNN	Images	Bridges2 GPU

```
import numpy as np
import argparse
import time
import tensorflow as tf

#Define number of gpus variable
ap=argparse.ArgumentParser()
ap.add_argument("-n","--ngpus", type=int, default=1, help='Number of GPUS')
args=vars(ap.parse_args())

ngpus=args["ngpus"]
epochs=10

#Load Data return as tensorflow dataset
(train_images,  train_labels),(test_images, test_labels)=tf.keras.datasets.cifar10.load_data()
train_images, test_images=train_images/255.0, test_images/255.0


#Define Model
def buildCNN():
    model=tf.keras.models.Sequential()
    model.add(tf.keras.layers.Conv2D(550, (3,3), activation='relu', input_shape=(32,32,3)))
    model.add(tf.keras.layers.MaxPooling2D((2,2)))
    model.add(tf.keras.layers.Conv2D(450, (3,3), activation='relu'))
    model.add(tf.keras.layers.MaxPooling2D((2,2)))
    model.add(tf.keras.layers.Conv2D(300, (3,3), activation='relu'))
    model.add(tf.keras.layers.MaxPooling2D((2,2)))
    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dense(64, activation='relu'))
    model.add(tf.keras.layers.Dense(10))
    return model

device_type='GPU'
devices=tf.config.experimental.list_physical_devices(device_type)
devices_names=[d.name.split('e:')[1] for d in devices]
strategy=tf.distribute.MirroredStrategy(devices=devices_names[:ngpus])

start=time.perf_counter()

with strategy.scope():
    model = buildCNN()
    model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), optimizer='adam')
model.fit(train_images, train_labels,batch_size=250, epochs=epochs, verbose=2)

elapsed=time.perf_counter()-start
print('time', elapsed)
```

Figure 2: Example machine learning model code for running on supercomputers with multiple GPUs.

The model defined in figure 2 was ran for different batch sizes on different numbers of GPUs using bridges2. The speed comparison results from this test are shown in figure3.

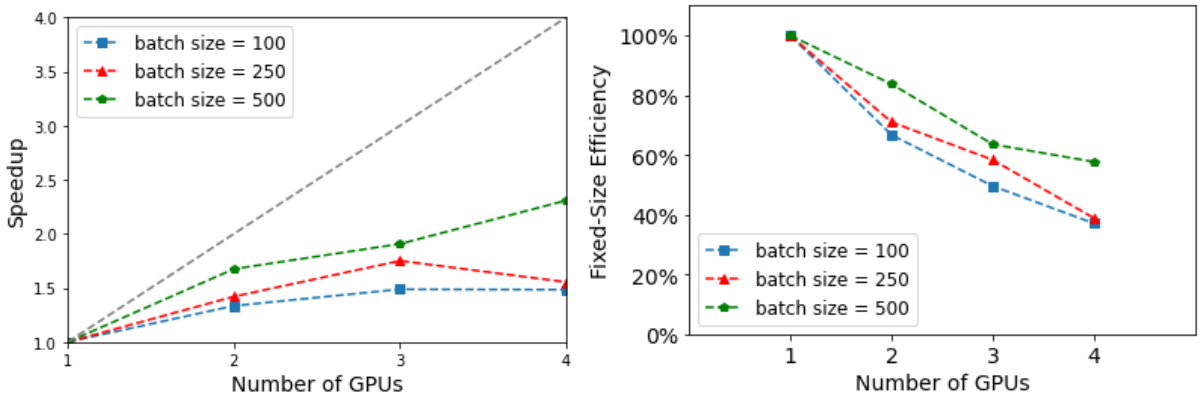

Figure 3: Left: Speed up comparison for CNN training on multiple GPUs. Right: Fixed-size parallel efficiency vs number of GPUs.

Figure 4 shows the speed up for training a simple multi-layer perceptron on a personal computer and two supercomputers. It was found that the Neocortex was slightly slower than CPU model different types of models will be tested to attempt to fully leverage the Neocortex supercomputer.

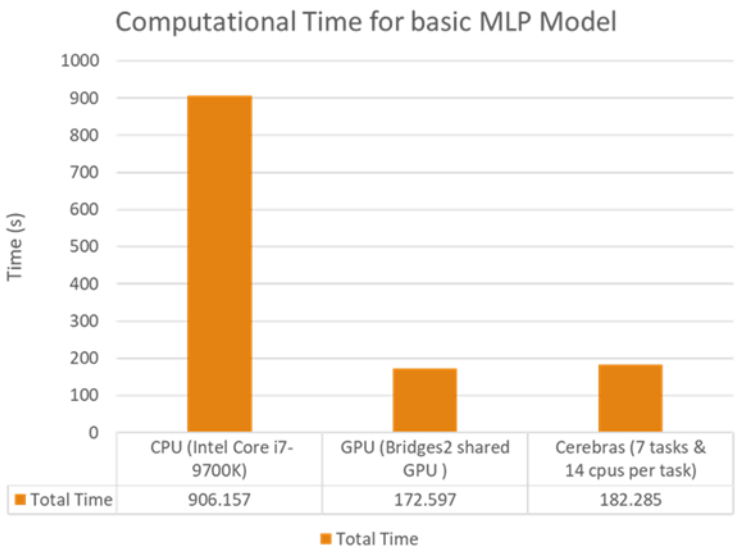

Figure 4: Plot showing speed comparison between CPU, Bridges2 GPU, and Cerebras for a basic MLP model. This is for case ID ML1 and ML2.

One density functional theory simulation (DFT1) was ran on Bridges2 CPU supercomputer with varying numbers of CPUs.

Table 3: DFT case.

Case ID	DFT Case	Pseudopotential	Number of atoms	Cluster
DFT1	FeNi	Fe/Ni.pbe-n-kjpaw_ps1.1.0.0.UPF	108	Bridges2 RM

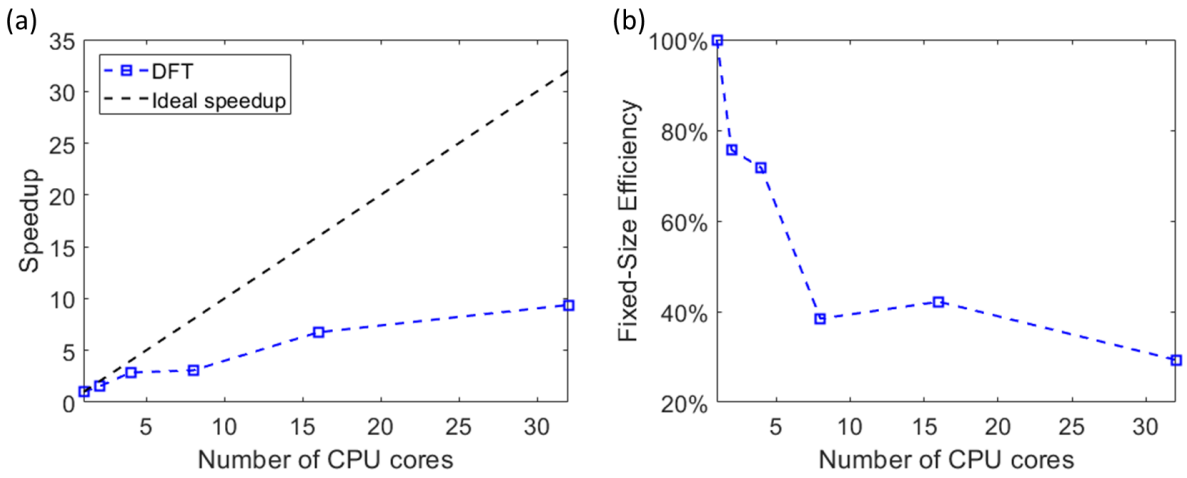

Figure 5: Case DFT1 showing speed up and b.) fixed-size parallel efficiency vs. CPUs.

Three molecular dynamics simulations were ran on Bridges2 CPU supercomputer for varying numbers of CPUs using LAMMPS.

Table 4: Molecular Dynamic Simulations ran using LAMMPS.

Case ID	MD Case	Force Field	Number of atoms	Cluster
MD1	HAP deformation at 300 K	IFF	120,736 – 2,816,000	Bridges2 RM
MD2				
MD3	FeNi diffusion	MEAM	80,000 – 1,280,000	Bridges2 RM

Figures 6 and 7 show the speed-ups seen from increasing the number of CPU cores for various code cases along with the fixed size efficiency.

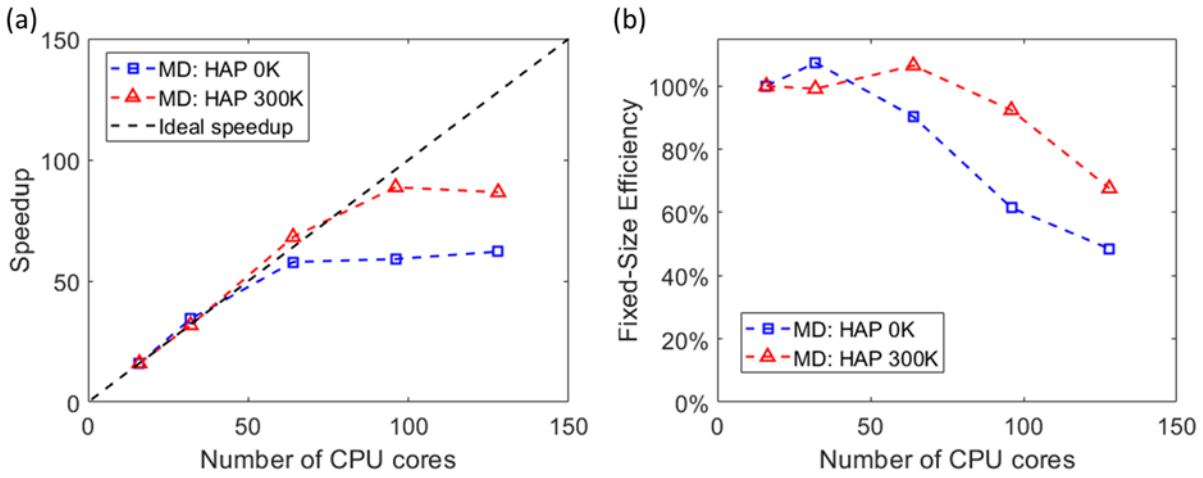

Figure 6: Case MD1 and MD2 showing speed up and b.) fixed-size parallel efficiency vs. CPUs.

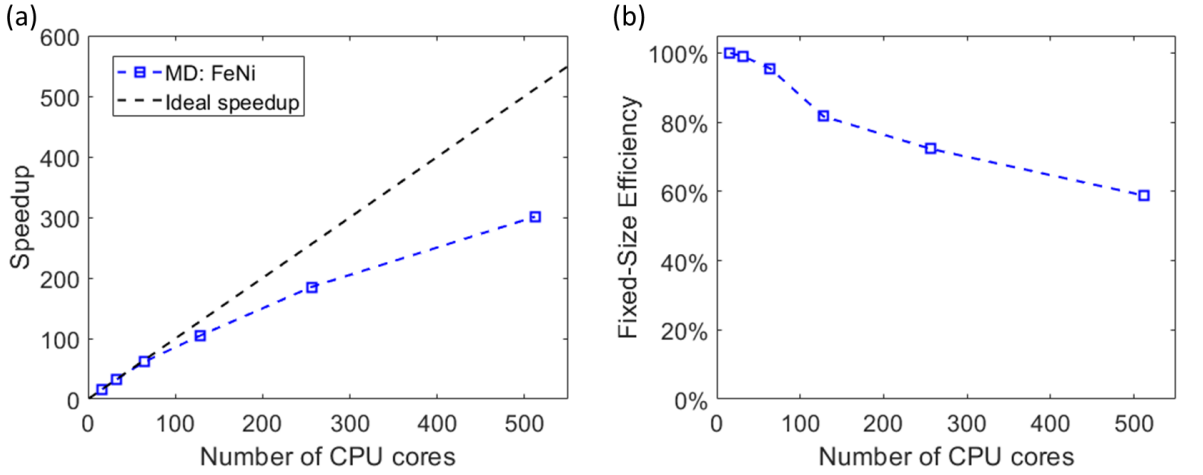

Figure 7: Case MD2 showing speedup and fixed size parallel efficiency against the number of CPU cores.

Supercomputing

The first hurdle in running codes on a supercomputer is connecting to one and transferring files. For this example, the supercomputer Bridges2 is used but other supercomputers may be slightly different.

Transferring files

The first step in using the supercomputer is transferring files, there needs to be a way to get the code and data to the supercomputer and get the results back to the personal computer. This process varies between a mac or Linux and a windows computer. For a mac or Linux, everything can be done in the terminal. There are a few different protocols that can be used such as scp or sftp.

To use sftp, first navigate to the folder in the computer terminal where the code will be moved from or to using “cd” command. Then use “sftp username@bridges2.psc.edu” to connect to the supercomputer. Once the command line shows “sftp>”, commands “get” and “put” can be used to transfer files. Use “cd” command again to switch to the correct folder on the supercomputer where code should be moved to or results should be pulled from.

Command	
get *filename*	Move file from remote computer to personal computer
put *filename*	Move file from personal computer to remote computer
cd path	Change directory to
get -r *foldername*	Used to move a folder from remote computer to personal computer
put -r *foldername*	Used to move a folder from personal computer to the remote computer

Table 5: Common Linux commands used in Supercomputing

Once the files are transferred, to close the session, type “exit”.

For a windows computer, a SFTP client should be installed such as WinSCP. WinSCP can be used for sftp or scp file transfer protocols. To use WinSCP select the file protocol, fill out the host name, username, and password then login. Once connected, files or folders can be dragged and dropped between the right and left sides to transfer.

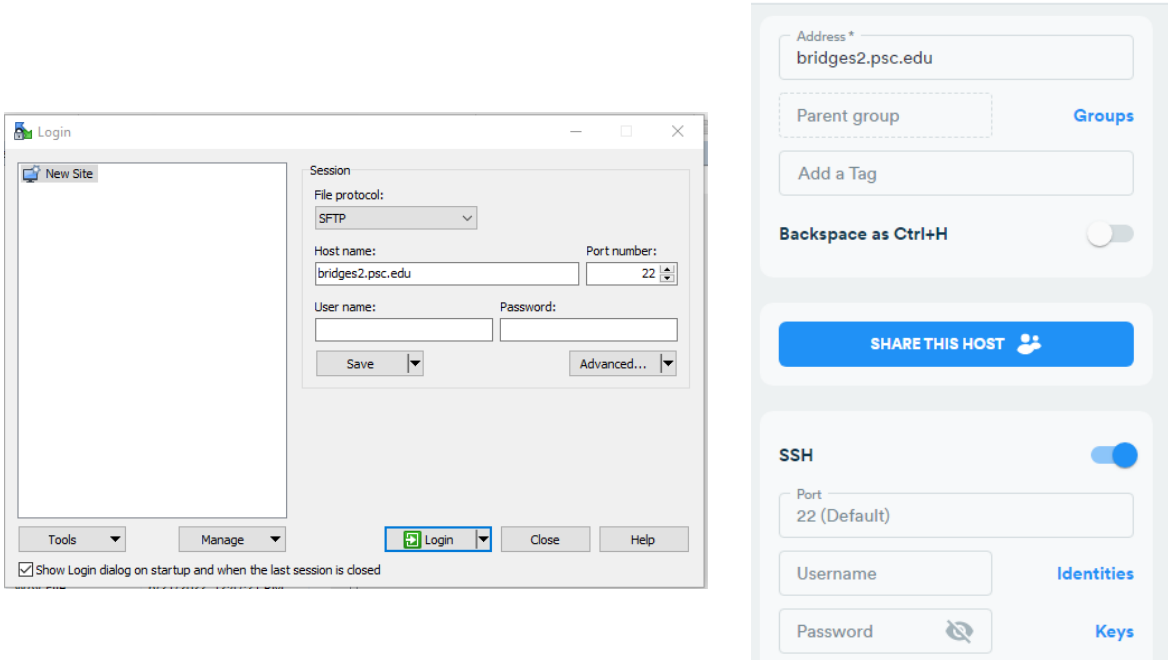

Figure 8: WinSCP login menu screen. To access files on Supercomputer, choose file protocol, enter hostname, username, and password. Termius add new host menu.

Connecting to the supercomputer

On a Linux or mac system, the terminal can be used for all commands. The ssh command in the terminal can be used to connect. For example, to connect to bridges2 the command “ssh username@bridges2.psc.edu ” is used. When prompted, type your password, and then you have access to the supercomputers. For a windows system, software should be installed to connect to the supercomputer. For example, Termius with WinSCP or putty can be used. In termius, a new host can be set up as so. Open the host and now you are connected to the supercomputer.

Using the supercomputer

For bridges2, there are two ways to run code; interactive and batch jobs. Interactive sessions are good for debugging. They allow the user to run codes in real time while connected to the computer. Batch jobs are submitted using SLURM or an equivalent batch scheduling system such as PBS or LSF and will be sent to a queue to be ran. Batch jobs allow for longer codes to be ran and can be submitted alongside other codes. The user does not have to be actively connected to the supercomputer after submitting them. A file will be generated showing the output of the code once it is completed running for the user to view later.

Interactive Session

1. Request an interactive session
2. Load desired modules
   a. This may include anaconda or Cuda modules. There are commands for viewing what modules are available to be loaded on Bridges2.
3. Run the code

a. Once all necessary modules are loaded run the code.

Table 6: Common commands used when running python code on Bridges2 in an interactive session.

Command		Example
interact	Used to start an interactive session.	interact -p GPU-shared --gres=gpu:1 -t 1:00:00
module spider	Lists all available modules, this is done in an interactive session	module spider anaconda
module load	Loads specified module	module load anaconda3-tf2.2020.11
conda create	Creates an anaconda environment, used with loaded anaconda module	conda create --prefix $PROJECT/env --clone $AI_ENV
conda activate	Activates already created anaconda environment	conda activate $PROJECT/env
ipython	Used to run jupyter notebook (.ipynb) files. Used with loaded anaconda module	ipython code.ipynb
python3	Used for running a python file (.py) with anaconda module loaded.	python3 code.py

Batch Job

1. Prepare a batch script

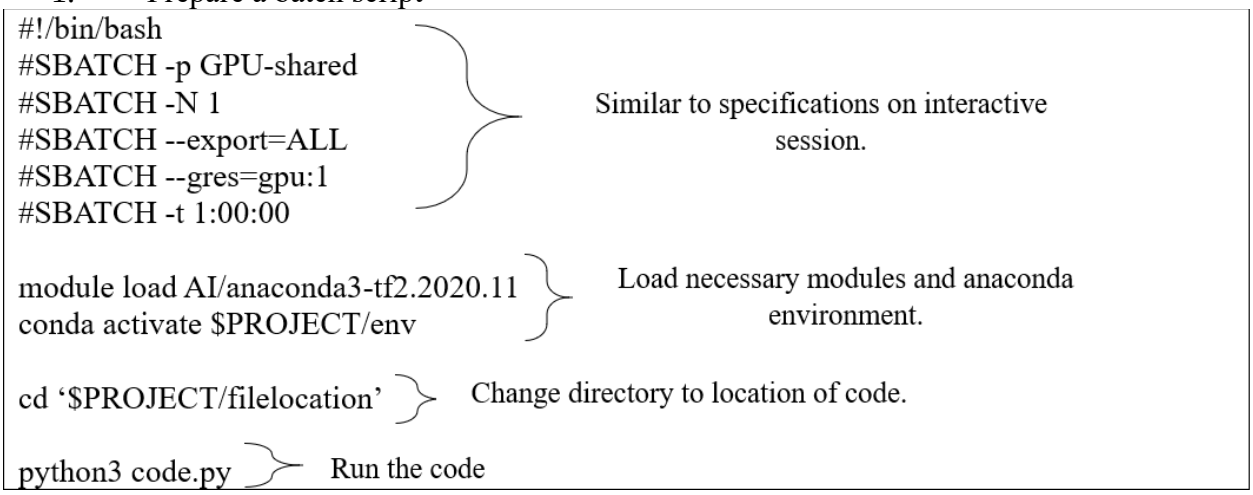

Figure 9: Example job script for batch job.

2. Submit batch script

Command	Function	Example

sbatch	Submits script to be ran	sbatch script.slurm
squeue	Used for showing what the status of submitted jobs is.	squeue -u username
scancel	Cancels previously submitted job. (Uses jobid which can be found using command squeue.)	scancel jobid

3. View results

Once the code is finished running it will output a txt file with the output from running.

Command	Function	Example
more	Shows the contents of file in command line.	more output.out
vi	Opens an interactive editor where the file can be viewed and edited. Within this editor: *i* (insert), can be pressed to edit the text. This portion can be closed by pressing the "esc" key. *:wq* (write and quit), will exit the editor.	vi output.out

Another new type of supercomputer is a wafer scale engine. For example, Neocortex is designed to speed up machine learning codes which use TensorFlow or PyTorch. TensorFlow is focused on here because it currently has the most documentation and usable layers/ loss functions.

To use Neocortex, code must first be converted to the correct structure. The model needs to use the TensorFlow estimator structure. The code should include 2 main functions.

1. Input_fn
   a. This function is used to load the data for the model.
   b. Inputs:
      i. The only input is "params" or parameters that can be defined by user.
   c. Outputs:
      i. Should output a tensorflow dataset.
2. Model_fn
   a. This function is used to define the model, optimizer, train_op, and loss.
   b. Inside this function the dtype should be set to float16.

c. Inputs:
    i. Features: this is the input to the model.
    ii. Labels: this is the correct output of the model.
    iii. Mode=tf.estimator.ModeKeys.TRAIN
    iv. Params is once again variables from user input.
d. Output:
    i. Returns spec=tf.estimator.Estimator.Spec(mode=mode, loss=loss, train_op=train_op, eval_metric_ops=None)

To run the code on Cerebras, it first must be compiled and then it can be run.

Integration To Classroom

There are two ways scientific computing should be incorporated into the classroom. First, each of the primary courses could add a bit more information on how to implement these methods in practice. By just introducing these topics, it will prepare students for an easier transition to running simulations. For example, in the mechanical engineering fluid curriculum, it might be good to touch on common computation packages that are used for fluid simulations such as OpenFOAM or COMSOL.

The other way these topics could be included in the curriculum is by creating a new elective that would allow students to use an available package to run a sample simulation or train a model and then expand on this by teaching how to upscale their model. This might mean using more atoms or increasing the time in a simulation or using more data for training a supervised machine learning mode. The course could include a scaling assignment to emphasize the speed up for increasing CPU nodes or GPUs on a supercomputer for large-scale problems. It would be important for the scaling to be at a large scale because as the processor clock speed is often slower on supercomputer nodes than on a desktop system, one often won't see a speedup at all unless the problem is of sufficient size to either utilize the large memory footprint, use all of the cores efficiently, or both. GPU codes would show speedup on most any problem.

Conclusion

In this paper, the role scientific computing plays in the engineering field is shown. It also shows how running code on supercomputers can lead to faster computational times. This is demonstrated through a few case studies including machine learning models and simulations including DFT and molecular dynamics. The process of moving code to and running code on supercomputers (Bridges2 and Neocortex) is also discussed. Lastly, ways that scientific computing can be integrated into the mechanical engineering curriculum are presented.

Acknowledgments

This work was supported by the Arkansas EPSCoR Data Analytics that are Robust & Trusted (DART) through seed grant number 22-EPS4-0028, under NSF grant number OIA- 1946391. This work used the Extreme Science and Engineering Discovery Environment (XSEDE) Bridges2 RM and Bridges2 GPU at Pittsburgh Supercomputing Center through allocation TG-MCH200010 supported by NSF grant number ACI-1548562, and Neocortex CS-1 supported by NSF grant number OAC-2005597.

References

The preferred reference style is IEEE. See the *Citing Sources and the References* section in the Paper Formatting Guidelines below for more instructions.

[1] M. Sheikholeslami, R. ul Haq, A. Shafee, Z. Li, Y. G. Elaraki, and I. Tlili, "Heat transfer simulation of heat storage unit with nanoparticles and fins through a heat exchanger," *International Journal of Heat and Mass Transfer*, vol. 135, pp. 470–478, Jun. 2019, doi: 10.1016/j.ijheatmasstransfer.2019.02.003.

[2] R. Malki, I. Masters, A. J. Williams, and T. Nick Croft, "Planning tidal stream turbine array layouts using a coupled blade element momentum - computational fluid dynamics model," *Renewable Energy*, vol. 63, pp. 46–54, Mar. 2014, doi: 10.1016/j.renene.2013.08.039.

[3] A. López, W. Nicholls, M. T. Stickland, and W. M. Dempster, "CFD study of Jet Impingement Test erosion using Ansys Fluent® and OpenFOAM®," *Computer Physics Communications*, vol. 197, pp. 88–95, Dec. 2015, doi: 10.1016/j.cpc.2015.07.016.

[4] Z. Wei, C. Zhang, Y. Kan, Y. Zhang, and Y. Chen, "Developing machine learning potential for classical molecular dynamics simulation with superior phonon properties," *Computational Materials Science*, vol. 202, no. November 2021, p. 111012, 2022, doi: 10.1016/j.commatsci.2021.111012.

[5] H. B. Wang, W. Y. Xu, and R. C. Xu, "Slope stability evaluation using Back Propagation Neural Networks," *Engineering Geology*, vol. 80, no. 3–4, pp. 302–315, Aug. 2005, doi: 10.1016/j.enggeo.2005.06.005.

[6] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, ´ M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wat- ´ tenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

Christy Dunlap

Christy Dunlap is a Ph.D. student in the Department of Mechanical Engineering at the University of Arkansas. Christy obtained her B.S. in Mechanical Engineering and B.S. in Mathematics with Applied Concentration from the University of Arkansas in 2021. Her research covers system design, DNA sequencing, thermal data analytics, and multimodal fusion. Christy is proficient in programming using Python, MATLAB, C++, and Arduino, machine learning packages including TensorFlow and scikit-learn, operating system and software maintenance on Linux systems.

Jeff Pummill

Mr. Pummill joined the University of Arkansas in 2005 as the Senior Systems Administrator for the newly formed high performance computing center to manage the 379th fastest supercomputer in the world followed in 2007 by a new system that ranked 339th in the world to support an increasingly diverse scientific workload at the UofA campus. In 2015, he negotiated acquisition and relocation of a 3 year old $2.8M supercomputer from San Diego Supercomputer Center to meet the increasing demands of research community on campus. In an adjunct capacity in the Fulbright College Department of Biological Sciences, he delivers lectures and workshops on scientific computing. As co-Director of AHPCC, he enjoys working with research faculty and graduate students and has co-authored a number of publications as part of his collaborative efforts to advance both AHPCC and scientific computing on campus. Jeff has been the recipient of nearly $500,000 in NSF funding as part of his work with the National research computing effort.

Han Hu

Han Hu is an Assistant Professor in the Department of Mechanical Engineering at the University of Arkansas. He leads the Nano Energy and Data-Driven Discovery (NED^3) Laboratory and his research interests cover experimental characterization and multi-scale modeling of two-phase heat transfer enhancement on micro-/nano-structured surfaces, immersion cooling of power electronics, diffusion kinetics in high-entropy alloys, and multimodal data fusion.