



UPM
UNIVERSITI PUTRA MALAYSIA
BERILMU BERBAKTI

PUTRA
PERTANIAN UNTUK RAKYAT

**INSTITUT
PENYELIDIKAN
MATEMATIK**
INSTITUTE FOR MATHEMATICAL RESEARCH
اينستيتوت قېليديقن ماتماتيك

**INSTITUTE FOR MATHEMATICAL RESEARCH (INSPEM),
UNIVERSITI PUTRA MALAYSIA**

SLURM COMMAND



With Knowledge We Serve Agriculture • Innovation • Life



CREATING SLURM SCRIPT

- Create a job script called `myjob.sh` with below content (sample script)

```
#!/bin/bash
#SBATCH --job-name="my job"
#SBATCH --time=00:10:00
#SBATCH --mem-per-cpu=1G
# Your code below this line
```

SUBMITTING JOB

- Submit job using `sbatch` command: `$ sbatch myjob.sh`
- Some of the options for `sbatch`
 - `--job-name:` name of the job
 - `--cpus-per-task:` how many cpu per task
 - `--mem-per-cpu:` how much ram allocation per cpu used
 - `--test-only:` validate batch script and return estimated start time
 - `--time:` expected runtime of the job in `dd-hh:mm:ss` format
- Commonly used options
 - `--user:` name of the owner
 - `--jobs:` job id
 - `--states:` state of the job

MONITORING JOBS



UPM
UNIVERSITI PUTRA MALAYSIA
BERILMU BERBAKTI

PUTRA
PERTANIAN UNTUK RAKYAT

- **queue job states:**

- **PD - Pending, waiting resource allocation**

- **R - Running**

- **S - Suspended, resource released for other job**

- **CA - Cancelled, by the user or system administrator**

- **CG - Completing, nearly completion**

- **CD - Completed, finished with exit code zero F - Failed, terminated with nonzero exit code**

- **To view running jobs: `$ queue`**

- **To get a more detailed information about a job, use `scontrol`: `$ scontrol show job <JOBID>`**

- **To delete a job, use below command.**

- **Get the jobid from queue : `$scancel <JOBID>`**

- **To delete all running job : `$scancel --state=R`**

- **Delete all jobs belonging to a user : `$scancel --user <USERNAME>`**

- **Delete a job with jobname : `$scancel -jobname "myjobname"`**



UPM
UNIVERSITI PUTRA MALAYSIA
BERILMU BERBAKTI

PUTRA
PERTANIAN UNTUK RAKYAT

EXAMPLE USING MULTIPLE CPU CORES

```
#!/bin/bash
#SBATCH --job-name=ml_training
#SBATCH --output=ml_training_%j.out
#SBATCH --error=ml_training_%j.err
#SBATCH --time=01:00:00
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1 # Adjust this to 1, 2, 4, etc. to compare
#SBATCH --mem=4G
# Run the Python script python train.py
```



EXAMPLE (TRAINING PYTHON CODE) – train.py

```
import numpy as np
import time

from sklearn.datasets import make_classification
from sklearn.model_selection import
train_test_split

from sklearn.linear_model import
LogisticRegression

from sklearn.metrics import accuracy_score

# Generate a larger synthetic dataset
X, y = make_classification (n_samples=2000000,
n_features=20, n_informative=15, n_classes=3,
random_state=42)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2,
random_state=42)
```

```
# Create a logistic regression model
model = LogisticRegression(max_iter=500) # Measure the
time taken to train the model

start_time = time.time()

model.fit(X_train, y_train)

end_time = time.time()

# Calculate training time

training_time = end_time - start_time

# Make predictions

y_pred = model.predict(X_test) # Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
print(f'Training Time: {training_time:.2f} seconds')
```



UPM
UNIVERSITI PUTRA MALAYSIA
BERILMU BERBAKTI

PUTRA
PERTANIAN UNTUK RAKYAT

PYTHON AND HPC



Practical Coding Examples:



Writing Python scripts for HPC.



Data processing with multiple cores.



Training a machine learning model using GPU acceleration.



UPM
UNIVERSITI PUTRA MALAYSIA
BERILMU BERBAKTI

PUTRA
PERTANIAN UNTUK RAKYAT

DATA PROCESSING WITH MULTIPLE CORE OF CPU

- ***Python's Multi-Processing Module:***
- ***Key features:***
 - ***Process class: Create and manage separate processes***
 - ***Pool class: Manage a pool of worker processes***
 - ***Queues and Pipes: Mechanisms for inter-process communication***

USING THE PROCESS CLASS

- **The Process class allows you to create and manage separate processes.**

Code Example(E4): Creating and Starting a Process

```
from multiprocessing import Process # Import the
Process class from the multiprocessing module

def worker function():
    print("Worker function is running") # Define a
    simple function to be executed by the process

if __name__ == "__main__":
    process = Process(target=worker_function) #
    Create a Process object with the target
    function
    process.start() # Start the process
    process.join() # Wait for the process to
    complete
```

MULTIPROCESSING (E5)

```
import time

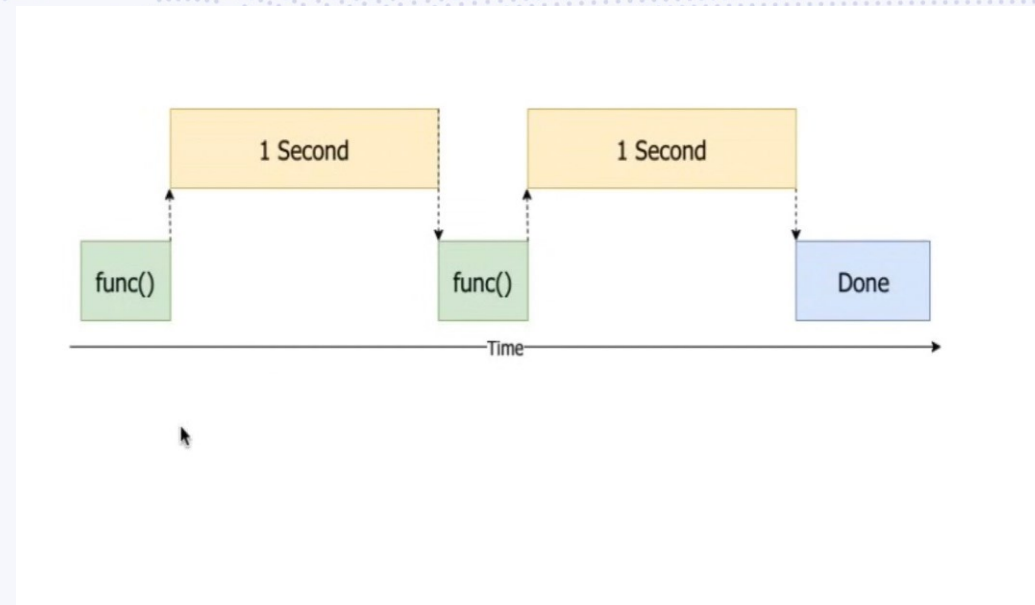
start = time.perf_counter()

def do_something():
    print('sleeping 1 second...')
    time.sleep(1)
    print('done Sleeping...')

do_something()
do_something()

finish = time.perf_counter()

print(f'Finished in {finish-start , 2} second(s)')
```



MULTIPROCESSING (E6)

```
import multiprocessing
import time

start = time.perf_counter()

def do_something():
    print('sleeping 1 second...')
    time.sleep(1)
    print('done Sleeping...')

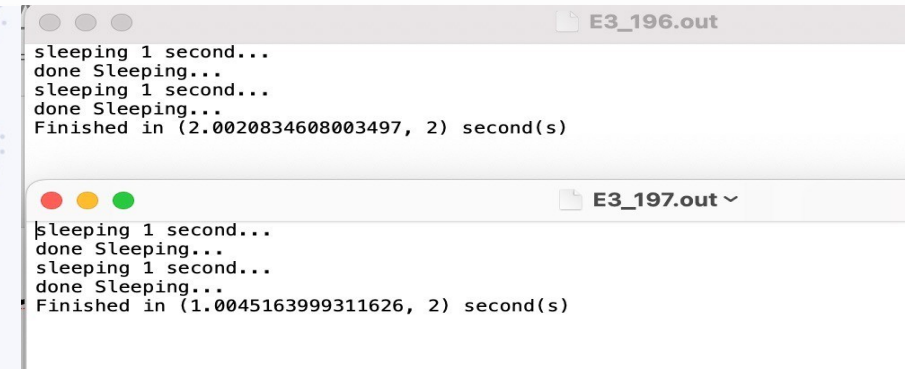
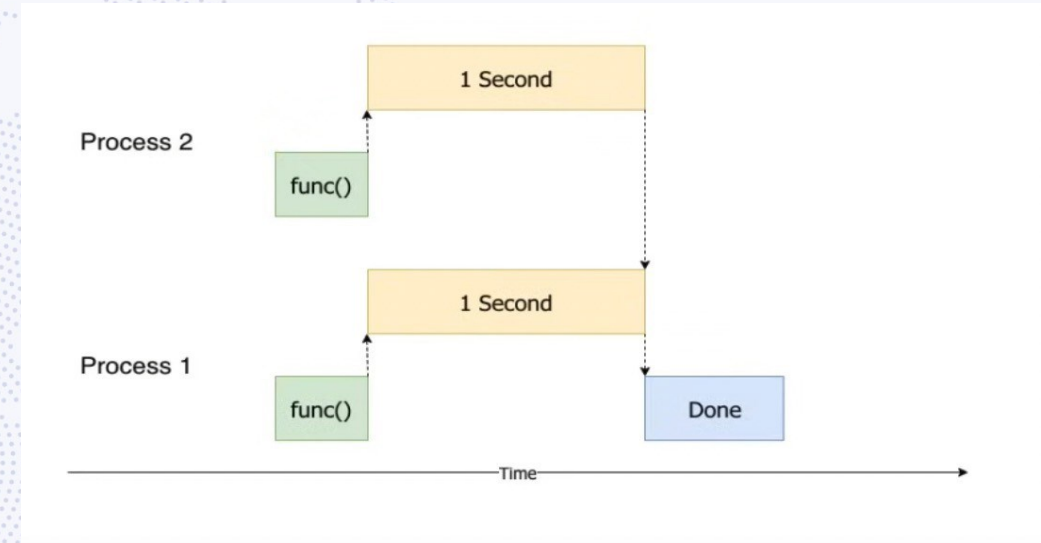
p1 = multiprocessing.Process(target=do_something)
p2 = multiprocessing.Process(target=do_something)

p1.start()
p2.start()

p1.join()
p2.join()

finish = time.perf_counter()

print(f'Finished in {finish-start , 2} second(s)')
```





MULTIPROCESSING (E7)

```
import multiprocessing
import time
start = time.perf_counter()

def do_something(seconds):
    print ('sleeping {seconds} second...')
    time.sleep(seconds)
    print ('done Sleeping...')

processes = []
for _ in range(10):
    p = multiprocessing.Process(target=do_something ,args=[1.5])
    p.start()
    processes.append(p)

for process in processes:
    process.join()

finish = time.perf_counter()
print (f'Finished in {finish-start , 2} second(s)')
```

USING THE POOL CLASS

The Pool class provides a convenient means of parallelizing the execution of a function across multiple input values.

Code Example(E8): Distributing Tasks Among Process

```
from multiprocessing import Pool # Import the Pool class
from the multiprocessing module

def worker_function(x):
    return x * x # Define a simple function to be executed
                # by the pool

if __name__ == "__main__":
    with Pool(4) as p: # Create a pool of 4 worker
                      # processes

        results = p.map(worker_function, [1, 2, 3, 4, 5])
        # Apply the worker function to the list [1, 2, 3,
        # 4, 5]

        print(results) # Print the results
```



PROCESSING

- **Scenario: Parallelizing data processing in a machine learning pipeline.**

```
from multiprocessing import Pool # Import the Pool class from the multiprocessing module
import random

def preprocess (data_chunk):
    return [x * 0.1 for x in data_chunk] # Example preprocessing: Scaling the data

if __name__ == "__main__":
    data = [random.randint(0, 100) for _ in range(1000)] # Create a list of random integers
    data_chunks = [data[i:i + 100] for i in range(0, len(data), 100)] # Split the data into chunks

    with Pool(4) as p: # Create a pool of 4 worker processes
        processed_chunks = p.map(preprocess, data_chunks) # Apply the preprocess function to each chunk

    processed_data = [item for sublist in processed_chunks for item in sublist] # Flatten the list of lists
    print(processed_data) # Print the processed data
```

CONCURRENT FUTURE MODULE (E10)



UPM
UNIVERSITI PUTRA MALAYSIA
BERILMU BERBAKTI

PUTRA
PERTANIAN UNTUK RAKYAT

```
import concurrent.futures
import time
start = time.perf_counter()

def do_something(seconds):

    print(f'Sleeping {seconds} second(s)...')
    time.sleep(seconds)

    return f'Done Sleeping...{seconds}'

with concurrent.futures.ProcessPoolExecutor() as executor:
    secs = [5, 4, 3, 2, 1]

    results = executor.map(do_something, secs)
    for result in results:
        print(result)

finish = time.perf_counter()
print(f'Finished in {round(finish-start, 2)} second(s)')
```



BEST PRACTICES AND TIPS

- ***Profile your code to identify bottlenecks.***
- ***Avoid global variables to prevent conflicts and ensure thread-safety.***
- ***Use appropriate chunk sizes to balance the load across processes.***
- ***Monitor resource usage to avoid overloading the system.***
- ***Deadlocks: Ensure proper handling of process synchronization.***
- ***Overhead of process creation: Consider the overhead when creating a large number of processes.***
- ***Global Interpreter Lock (GIL): Be aware that multi-processing avoids GIL, unlike multi-threading.***



ACCELERATING MACHINE LEARNING WORKFLOWS WITH GPU COMPUTING:

Why Use GPUs?

- Definition and advantages of GPU computing:
 - **Parallel processing capabilities**
 - **Higher throughput compared to CPUs**
- Use cases in machine learning:
 - **Training deep neural networks**
 - **Handling large-scale data**

- **Overview of CUDA:**
 - **Compute Unified Device Architecture (CUDA) by NVIDIA**
 - **Allows direct programming of NVIDIA GPUs**
- **Key features:**
 - **High performance**
 - **Scalability**
 - **Integration with popular ML frameworks**



SETTING UP CUDA WITH PYTHON (E11)

```
import torch
print(torch.cuda.is_available()) # Check if CUDA is
available
print(torch.cuda.device_count()) # Check the number of
available GPUs
```



USING A SINGLE GPU (E12)

Setting up a simple computation on a single GPU using PyTorch

```
import torch

# Create a tensor
x = torch.tensor([1.0, 2.0, 3.0])

# Move tensor to GPU
x = x.cuda()

# Perform a computation
y = x ** 2

# Move result back to CPU
y = y.cpu()

print(y) # Print the result
```



Accelerating Machine Learning with Multi-CPU

- Utilizes multiple CPU cores
- Suitable for tasks that can be easily parallelized
- Commonly used for data preprocessing, model training, and hyperparameter tuning.
- **Advantages:**
 - Easy to implement with Python's multiprocessing module
 - Scales well with the number of CPU cores
- **Challenges:**
 - Limited by the number of CPU cores
 - Overhead of process creation and inter-process communication.



Accelerating Machine Learning with GPU

- Utilizes thousands of smaller, efficient cores
- Ideal for parallel tasks like matrix operations and deep learning
- Supported by libraries like TensorFlow and PyTorch
- **Advantages:**
 - Significant speedup for large-scale computations
 - Essential for deep learning and complex models
- **Challenges:**
 - Requires knowledge of GPU programming
 - Limited by GPU memory
 - May involve higher costs



USING MULTIPLE CPU CORES WITH MULTI-PROCESSING(E13)

```
import multiprocessing as mp
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

def train_model(X_train, y_train, X_test, y_test, q):
    clf = RandomForestClassifier(n_estimators=100)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    q.put(accuracy)
```



USING MULTIPLE CPU CORES WITH MULTI-PROCESSING 2 (E13)

```
if __name__ == "__main__":  
    # Print the number of available CPU cores  
    print(f"Number of available CPU cores:  
    {mp.cpu_count()}")  
  
    # Generate synthetic dataset  
    X, y = make_classification(n_samples=1000,  
    n_features=20, random_state=42)  
    X_train, X_test, y_train, y_test =  
    train_test_split(X, y, test_size=0.2,  
    random_state=42)  
  
    processes = []  
    queue = mp.Queue()  
  
    # Start the timer  
    start = time.time()
```

```
# Create and start processes  
for _ in range(mp.cpu_count()):  
    p = mp.Process(target=train_model, args=(X_train, y_train, X_test, y_test,  
    queue))  
    processes.append(p)  
    p.start()  
  
# Join processes  
for p in processes:  
    p.join()  
  
# Gather results  
accuracies = [queue.get() for _ in range(len(processes))]  
print(f"Accuracies from all processes: {accuracies} ")  
print(f"Average accuracy: {sum(accuracies) / len(accuracies)} ")  
print(f"Run duration Time: {time.time() - start}")
```

USING MULTIPLE CPU CORES WITH MULTI-PROCESSING 2 (E13)

```
if __name__ == "__main__":  
  
# Print the number of available CPU cores  
print(f"Number of available CPU cores:  
{mp.cpu_count()}")  
  
# Generate synthetic dataset  
X, y = make_classification(n_samples=1000,  
n_features=20, random_state=42)  
  
X_train, X_test, y_train, y_test =  
train_test_split(X, y, test_size=0.2,  
random_state=42)  
  
processes = []  
  
queue = mp.Queue()  
  
# Start the timer  
start = time.time()
```

```
# Create and start processes  
for _ in range(mp.cpu_count()):  
    p = mp.Process(target=train_model, args=(X_train, y_train, X_test, y_test,  
queue))  
    processes.append(p)  
    p.start()  
  
# Join processes  
for p in processes:  
    p.join()  
  
# Gather results  
accuracies = [queue.get() for _ in range(len(processes))]  
  
print(f"Accuracies from all processes: {accuracies} ")  
print(f"Average accuracy: {sum(accuracies) / len(accuracies)} ")  
print(f"Run duration Time: {time.time() - start}")
```

EXAMPLE GPU ACCELERATION (E14)



UPM
UNIVERSITI PUTRA MALAYSIA
BERILMU BERBAKTI

PUTRA
PERTANIAN UNTUK RAKYAT

```
import torch

from torch import nn, optim

from torch.utils.data import DataLoader, TensorDataset

from sklearn.datasets import make_classification

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

import time
# Check if GPU is available

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

print(f"Using device: {device}")

# Define a simple neural network

class SimpleNN(nn.Module):

    def __init__(self, input_dim):

        super(SimpleNN, self).__init__()

        self.fc1 = nn.Linear(input_dim, 100)

        self.fc2 = nn.Linear(100, 1)

    def forward(self, x):

        x = torch.relu(self.fc1(x))

        x = torch.sigmoid(self.fc2(x))

        return x
```

EXAMPLE GPU ACCELERATION 2 (E14)

```
# Training function
def train_model(X_train, y_train, X_test, y_test):
    input_dim = X_train.shape[1]
    model = SimpleNN(input_dim).to(device)
    criterion = nn.BCELoss()
    optimizer = optim.Adam(model.parameters(),
                             lr=0.001)
    train_dataset = TensorDataset(torch.tensor(X_train,
                                                dtype=torch.float32), torch.tensor(y_train,
                                                dtype=torch.float32))
    train_loader = DataLoader(train_dataset,
                              batch_size=32, shuffle=True)
```

```
# Train the model
model.train()
for epoch in range(20): # Adjust the number of epochs as needed
    for X_batch, y_batch in train_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device).unsqueeze(1)
        optimizer.zero_grad()
        outputs = model(X_batch)
        loss = criterion(outputs, y_batch)
        loss.backward()
        optimizer.step()
# Evaluate the model
model.eval()
with torch.no_grad():
    X_test_tensor = torch.tensor(X_test, dtype=torch.float32).to(device)
    y_test_tensor = torch.tensor(y_test, dtype=torch.float32).to(device).unsqueeze(1)
    outputs = model(X_test_tensor)
    predictions = (outputs > 0.5).float()
    accuracy = accuracy_score(y_test_tensor.cpu(), predictions.cpu())
return accuracy
```



EXAMPLE GPU ACCELERATION 3 (E14)

```
if __name__ == "__main__":  
    # Generate synthetic dataset  
  
    X, y = make_classification(n_samples=1000, n_features=20, random_state=42)  
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)  
  
    # Start the timer  
    start = time.time()  
  
    # Train the model and calculate accuracy  
  
    accuracy = train_model(X_train, y_train, X_test, y_test)  
    # Print the accuracy and running time  
    print(f"Accuracy: {accuracy}")  
  
    print(f"Run duration Time: {time.time() - start}")
```



UPM
UNIVERSITI PUTRA MALAYSIA
BERILMU BERBAKTI

PUTRA
PERTANIAN UNTUK RAKYAT

CHOOSING BETWEEN MULTI-CPU AND GPU

- **Multi-CPU:**
 - Suitable for simpler, smaller tasks
 - Lower cost, easier to implement for non-deep learning tasks
- **GPU:**
 - Ideal for deep learning and tasks with heavy parallelism
 - Provides substantial speedup for large datasets and complex models

CONCLUSION

- Both Multi-CPU and GPU have their place in machine learning
- Multi-CPU is accessible and suitable for general parallel tasks
- GPU acceleration is powerful for deep learning and large-scale problems
- Choose based on task complexity, dataset size, and resource availability



UPM
UNIVERSITI PUTRA MALAYSIA
BERILMU BERBAKTI

PUTRA
PERTANIAN UNTUK RAKYAT

INTERACTIVE USER TESTING

- ***Real-time Code Testing and Debugging:***
- ***Participants test their code on HPC systems.***
- ***Live debugging and troubleshooting with instructor assistance.***
- ***Common issues and how to resolve them.***



Agriculture • Innovation • Life

With Knowledge We Serve

Thank you