Benchmarking

Hoang Oanh Pham

Outline

- Introduce benchmarking.
- Why we use benchmarking?
- Test with examples:
  - Fashion-MNIST
  - CPU-only vs GPU – Matlab
  - BERT – for GPU memory usage
  - LAMMPS

What is benchmarking?

- Process of measuring the performance.
- Identify internal opportunities for improvement.

How can I make my code run faster?

- Depends on:
  - Code features/quality
  - Dataset being used
  - Network and disk usage over life of job

Why benchmark?

- To understand performance of OWN CODE on OWN DATA
- To understand how hardware resources requested affects run time

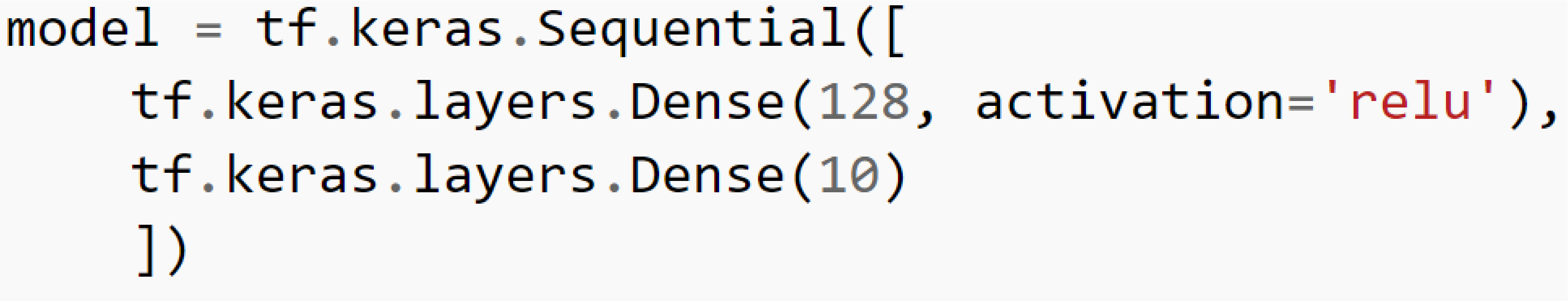
Let’s test!

- Fashion-MNIST
- BERT – for GPU memory usage
- CPU-only vs GPU – Matlab
- LAMMPS

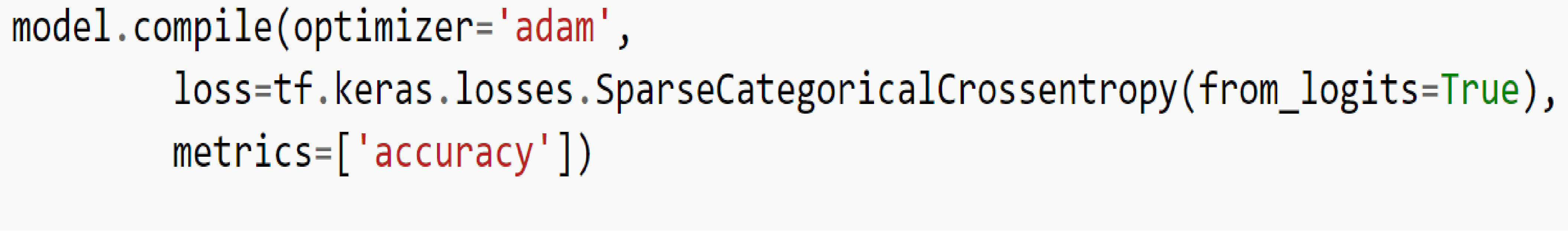
Fashion-MNIST

- Performs image classification on images of clothing from the Fashion-MNIST dataset
- Fashion-MNIST dataset:
  - images with the same format as MNIST data
  - allowing for a drop-in replacement of the MNIST dataset
  - 28 x 28 grayscale images
  - concatenated into single files which are then compressed
- Reading: https://proteusmaster.urcf.drexel.edu/urcfwiki/index.php/Slurm_-_Job_Script_Example_05a_TensorFlow_With_Anaconda_Python#Note_on_Performance_Tuning_for_Intel_CPUs

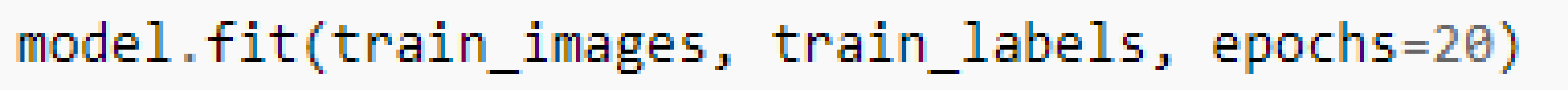
Fashion-MNIST: Code



```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10)
    ])
```



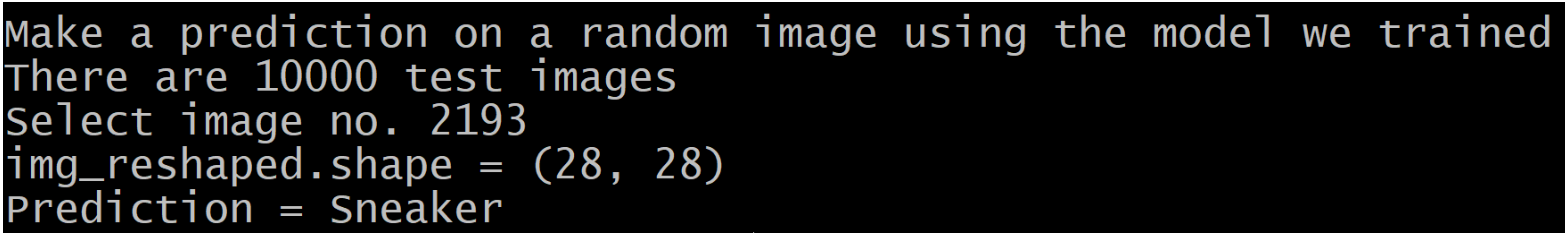
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```



```
model.fit(train_images, train_labels, epochs=20)
```

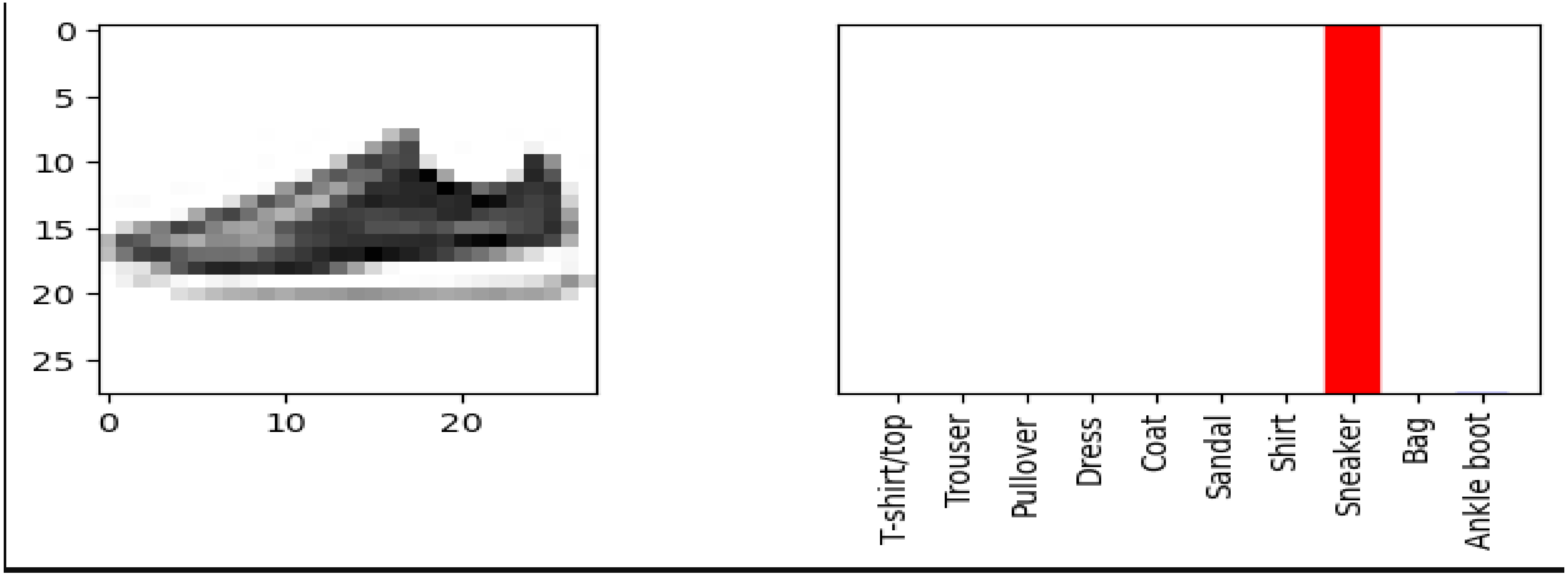
Fashion-MNIST

- Training time is 33 seconds

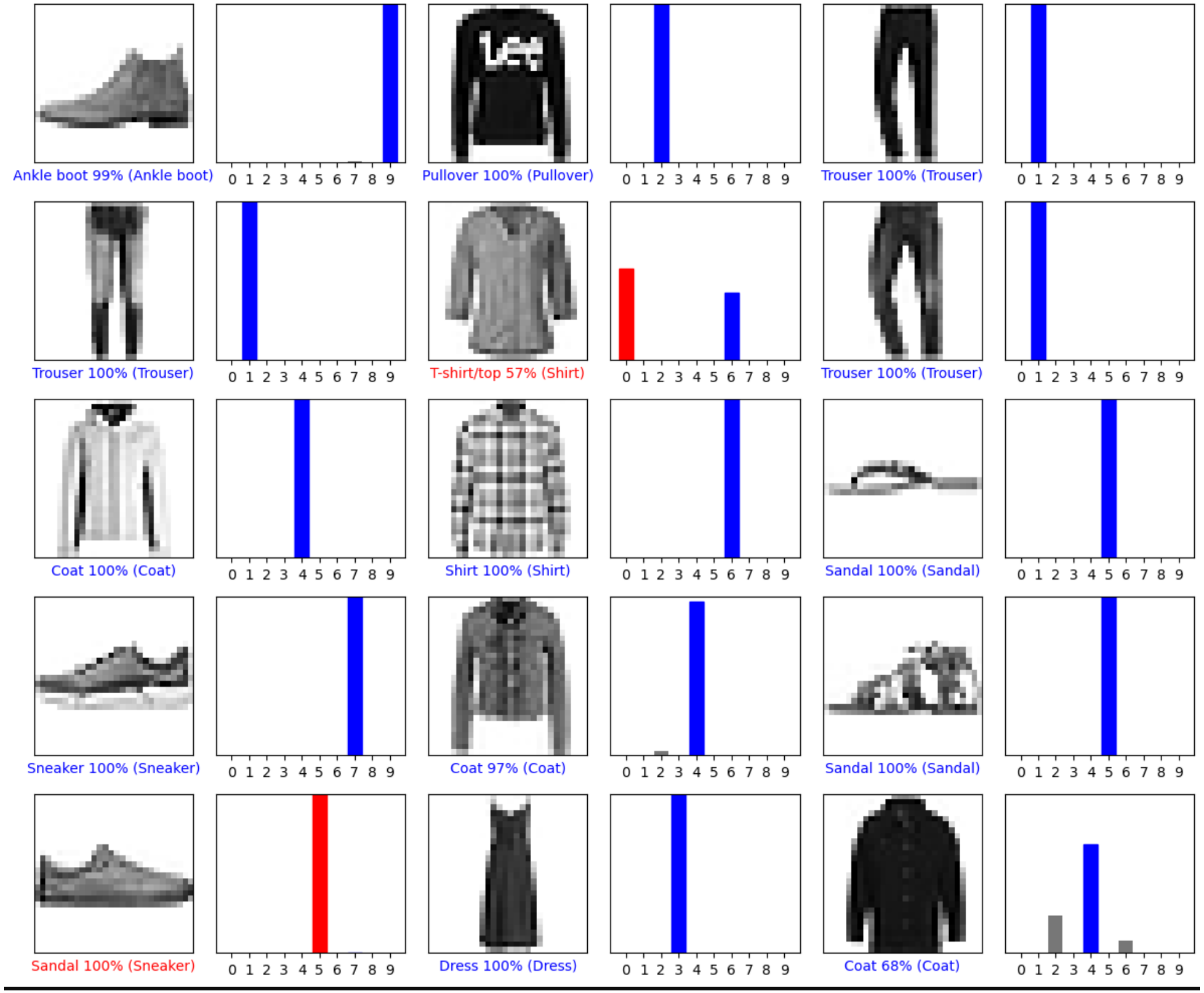


```
Make a prediction on a random image using the model we trained
There are 10000 test images
Select image no. 2193
img_reshaped.shape = (28, 28)
Prediction = Sneaker
```

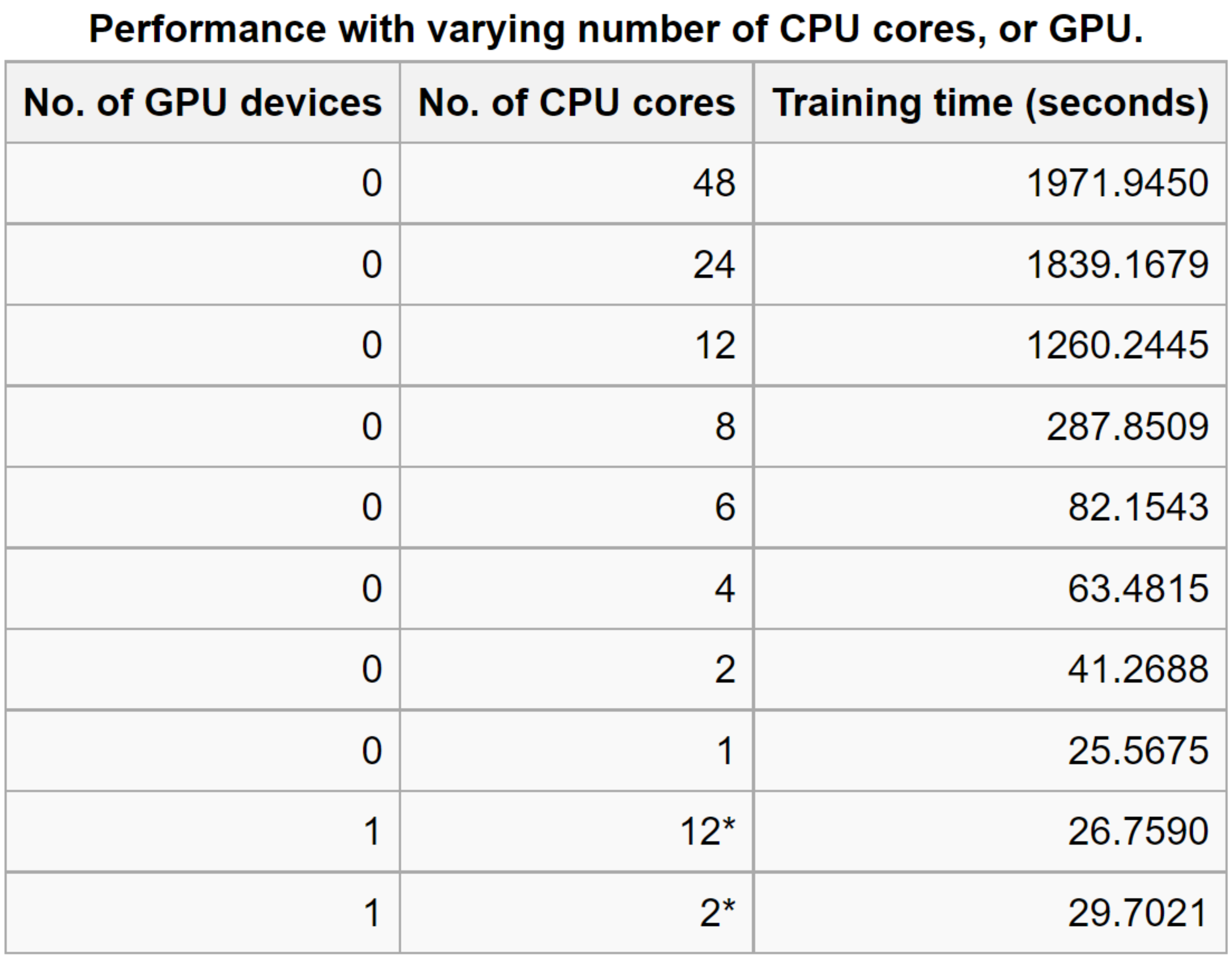
Fashion-MNIST: Prediction



Fashion-MNIST: Verification



Fashion-MNIST



Performance with varying number of CPU cores, or GPU.

No. of GPU devices	No. of CPU cores	Training time (seconds)
0	48	1971.9450
0	24	1839.1679
0	12	1260.2445
0	8	287.8509
0	6	82.1543
0	4	63.4815
0	2	41.2688
0	1	25.5675
1	12*	26.7590
1	2*	29.7021

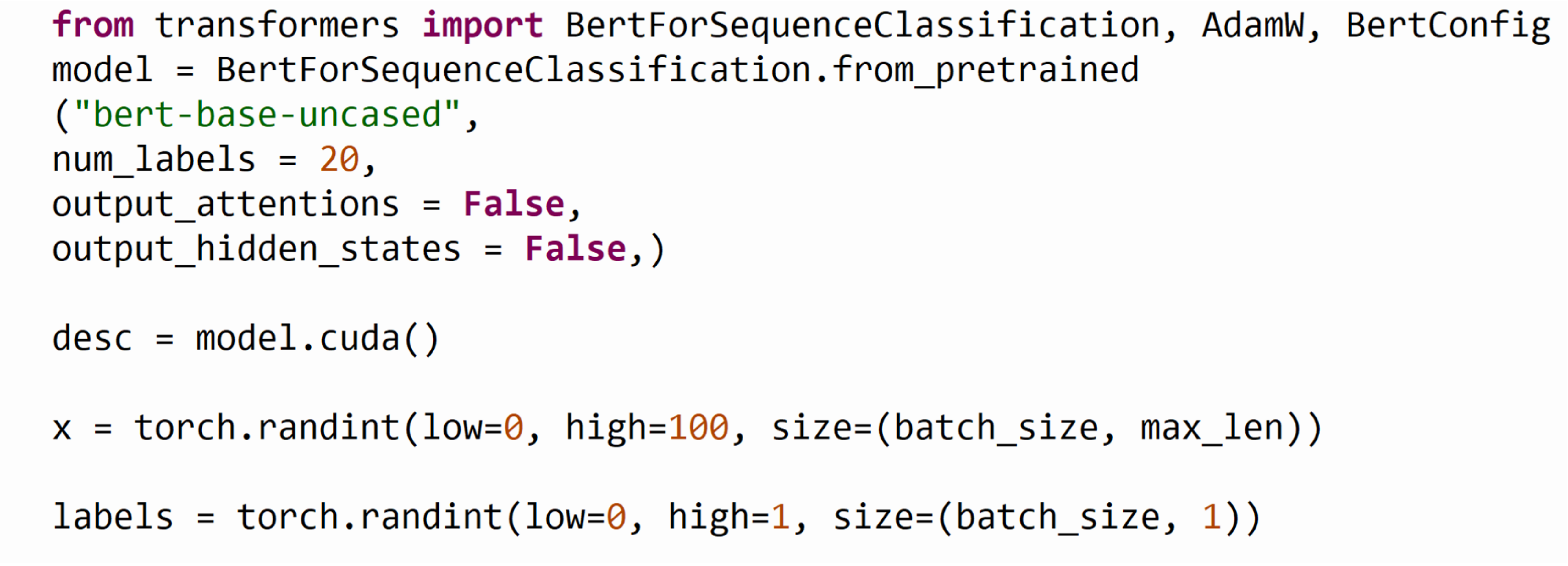
Fashion-MNIST: Conclusion

- Performance of training this simple model on this small dataset cannot be extrapolated to other models and datasets
- It is not true that "more cores means faster computation"
- Performance can vary depending on how threads are distributed
- Performance strongly depends on the dataset used

Bert: how the data set effects the memory

- Training batch size affects the memory usage.
- Batch size: a hyperparameter that defines the number of samples to work through before updating the internal model parameters.
- Reading: https://proteusmaster.urcf.drexel.edu/urcfwiki/index.php/GPU_Memory_Limits_for_BERT#Code

Bert: Code



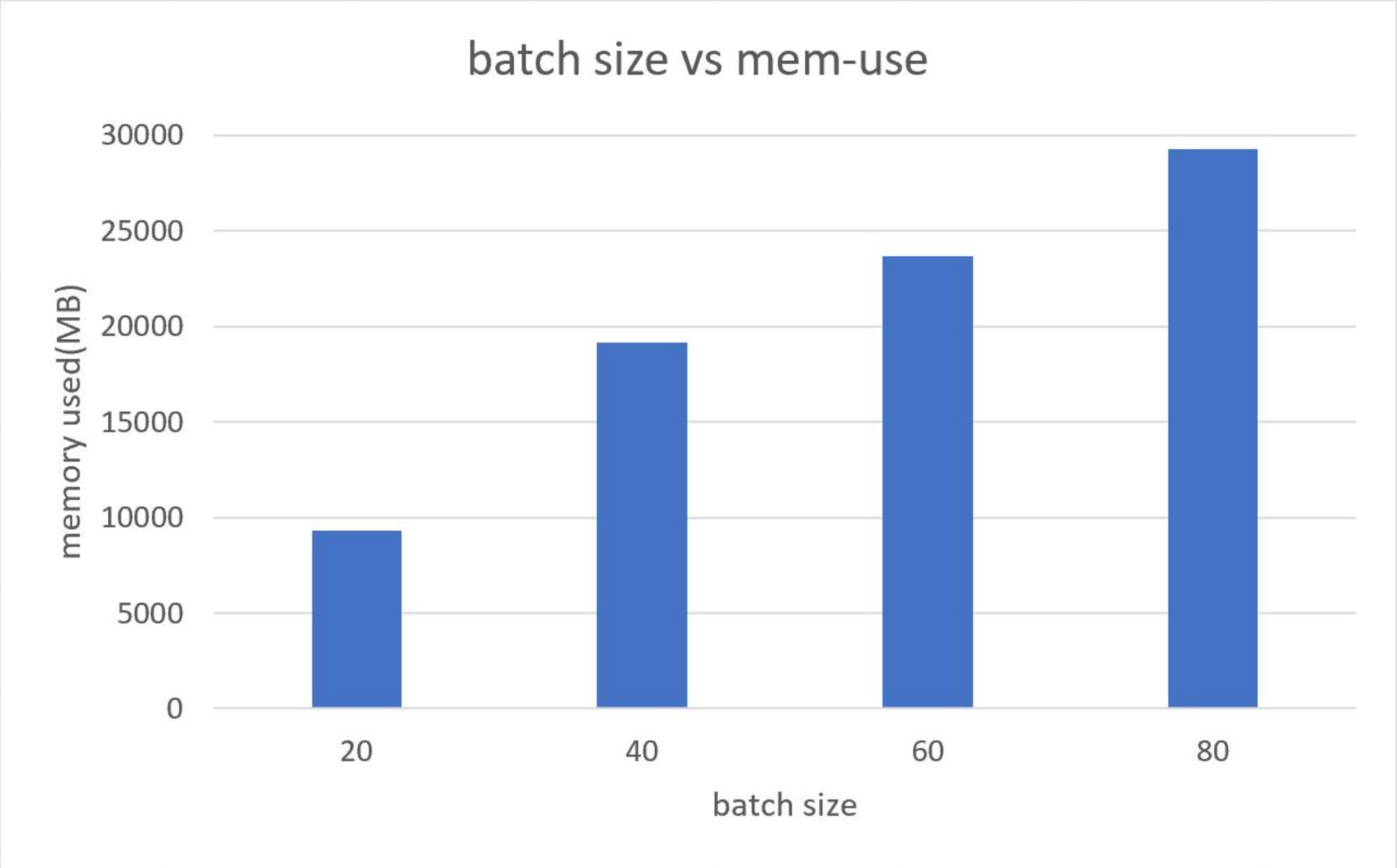
```
from transformers import BertForSequenceClassification, AdamW, BertConfig
model = BertForSequenceClassification.from_pretrained
("bert-base-uncased",
num_labels = 20,
output_attentions = False,
output_hidden_states = False,)

desc = model.cuda()

x = torch.randint(low=0, high=100, size=(batch_size, max_len))

labels = torch.randint(low=0, high=1, size=(batch_size, 1))
```

Bert:


batch size vs mem-use
memory used(MB)
30000
25000
20000
15000
10000
5000
0
20
40
60
80
batch size

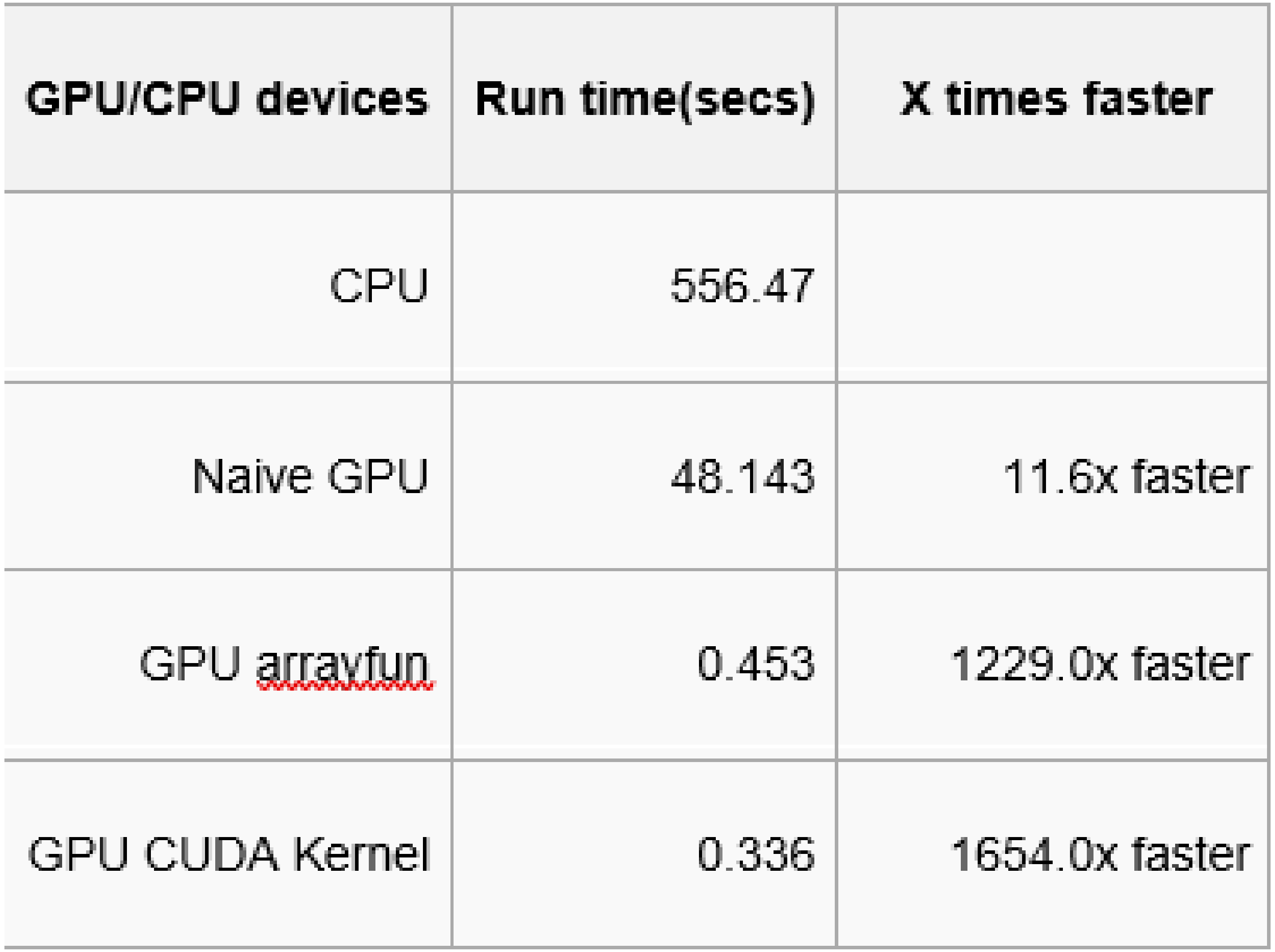
Bert: Conclusion

- The bigger the value of training batch size, the bigger the amount of memory used.
- When the batch size reaches 100, it produces error: out of memory

Matlab

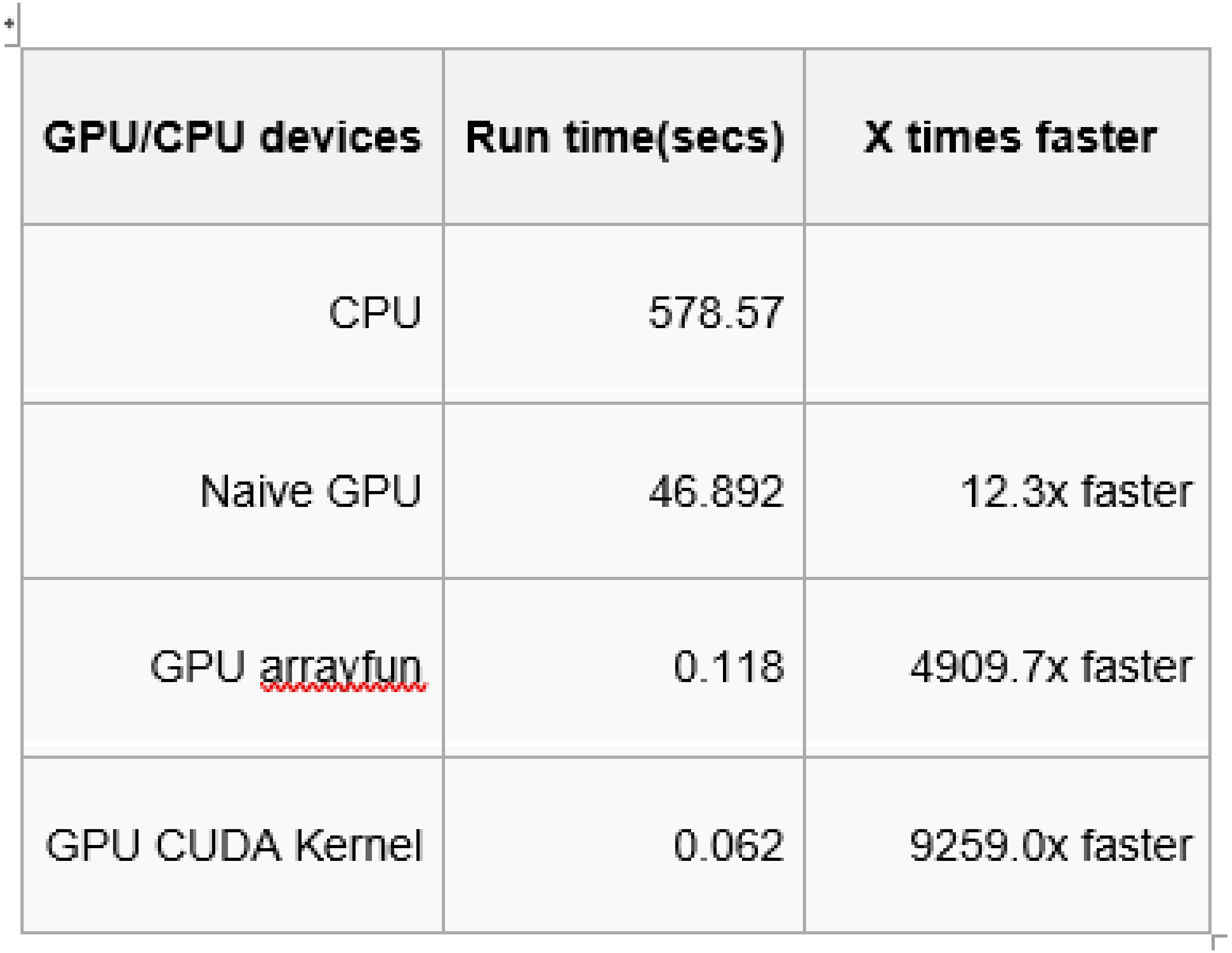
- Make use of GPU hardware in three ways:
  - Using the existing algorithm but with GPU data as input
  - Using arrayfun to perform the algorithm on each element independently
  - Using the MATLAB/CUDA interface to run some existing CUDA/C++ Kernel code
- V100 has 5120 CUDA cores
- Reading:
  - https://proteusmaster.urcf.drexel.edu/urcfwiki/index.php/MATLAB#GPU_Example
  - https://www.mathworks.com/help/parallel-computing/illustrating-three-approaches-to-gpu-computing-the-mandelbrot-set.html;jsessionid=ed7b2cafab333a097e00141bf10f

Matlab with Anaconda



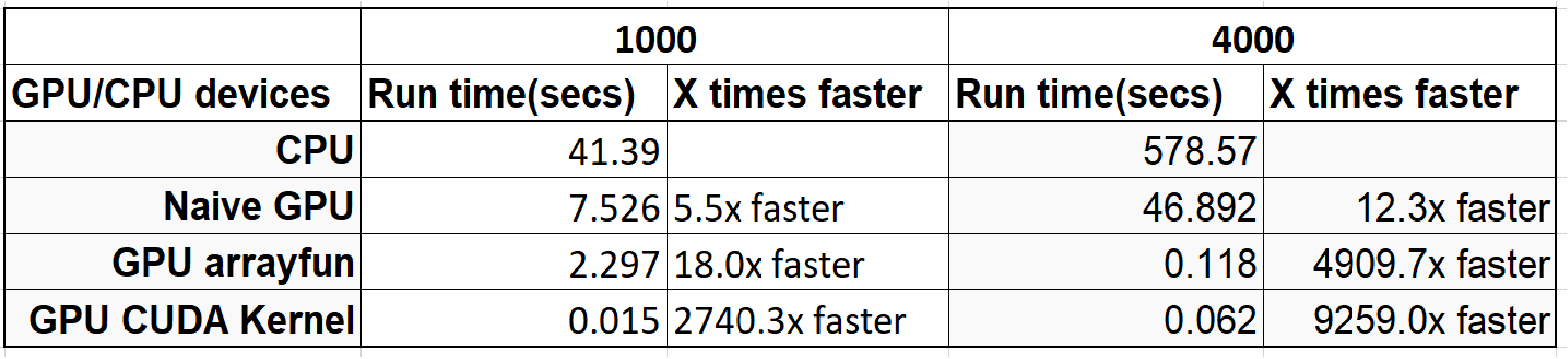
GPU/CPU devices	Run time(secs)	X times faster
CPU	556.47	
Naive GPU	48.143	11.6x faster
GPU arrayfun	0.453	1229.0x faster
GPU CUDA Kernel	0.336	1654.0x faster

Matlab without Anaconda



GPU/CPU devices	Run time(secs)	X times faster
CPU	578.57	
Naive GPU	46.892	12.3x faster
GPU arrayfun	0.118	4909.7x faster
GPU CUDA Kernel	0.062	9259.0x faster

Different gridSize vs run time:



	1000		4000	
GPU/CPU devices	Run time(secs)	X times faster	Run time(secs)	X times faster
CPU	41.39		578.57	
Naive GPU	7.526	5.5x faster	46.892	12.3x faster
GPU arrayfun	2.297	18.0x faster	0.118	4909.7x faster
GPU CUDA Kernel	0.015	2740.3x faster	0.062	9259.0x faster

Matlab: Conclusion

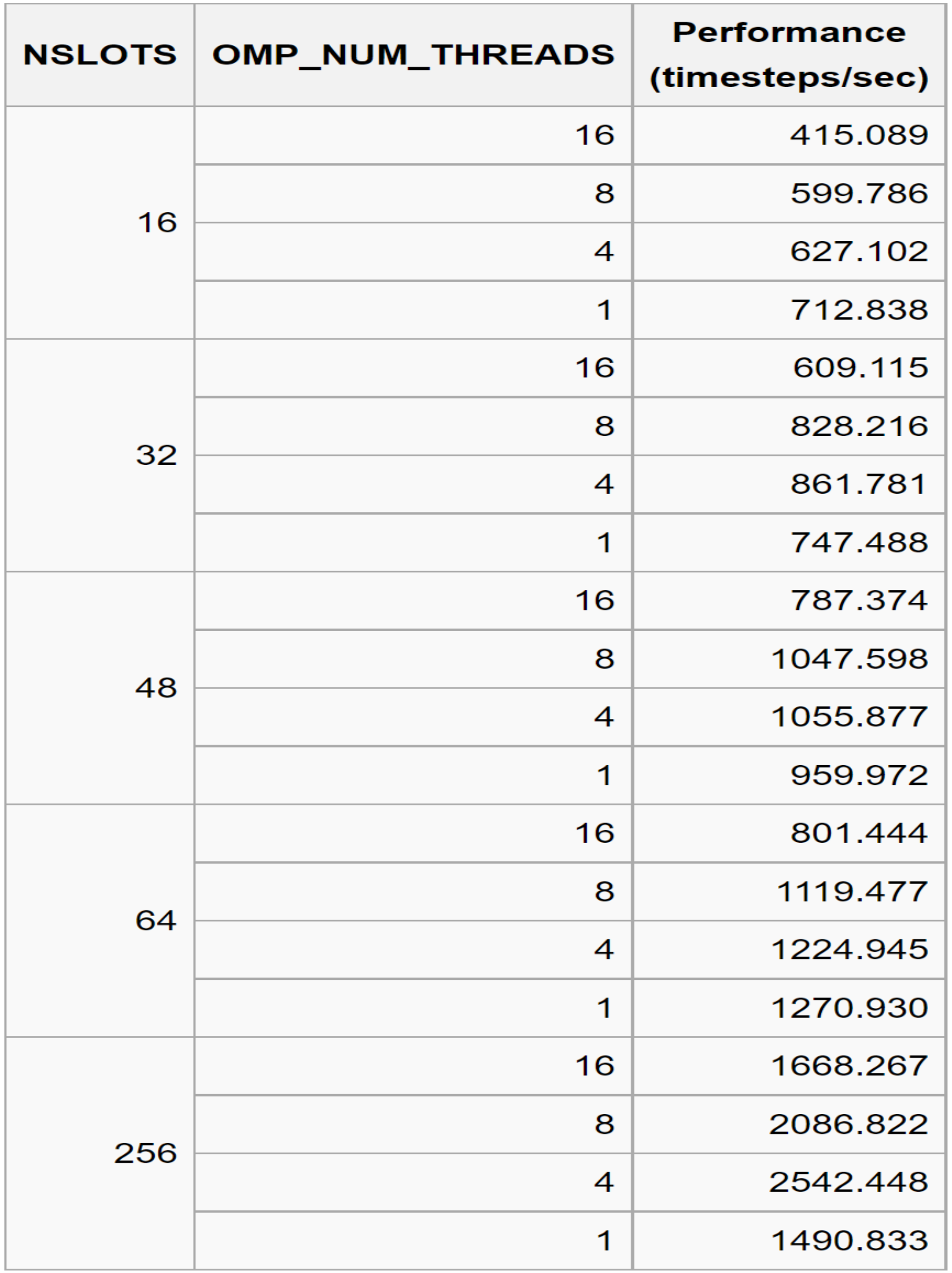
- If you want to use CUDA code, make sure you don't have Anaconda set up
- GPU version is fast because it doesn’t use for-loops – it runs a parallel computation on the GPU

LAMMPS:

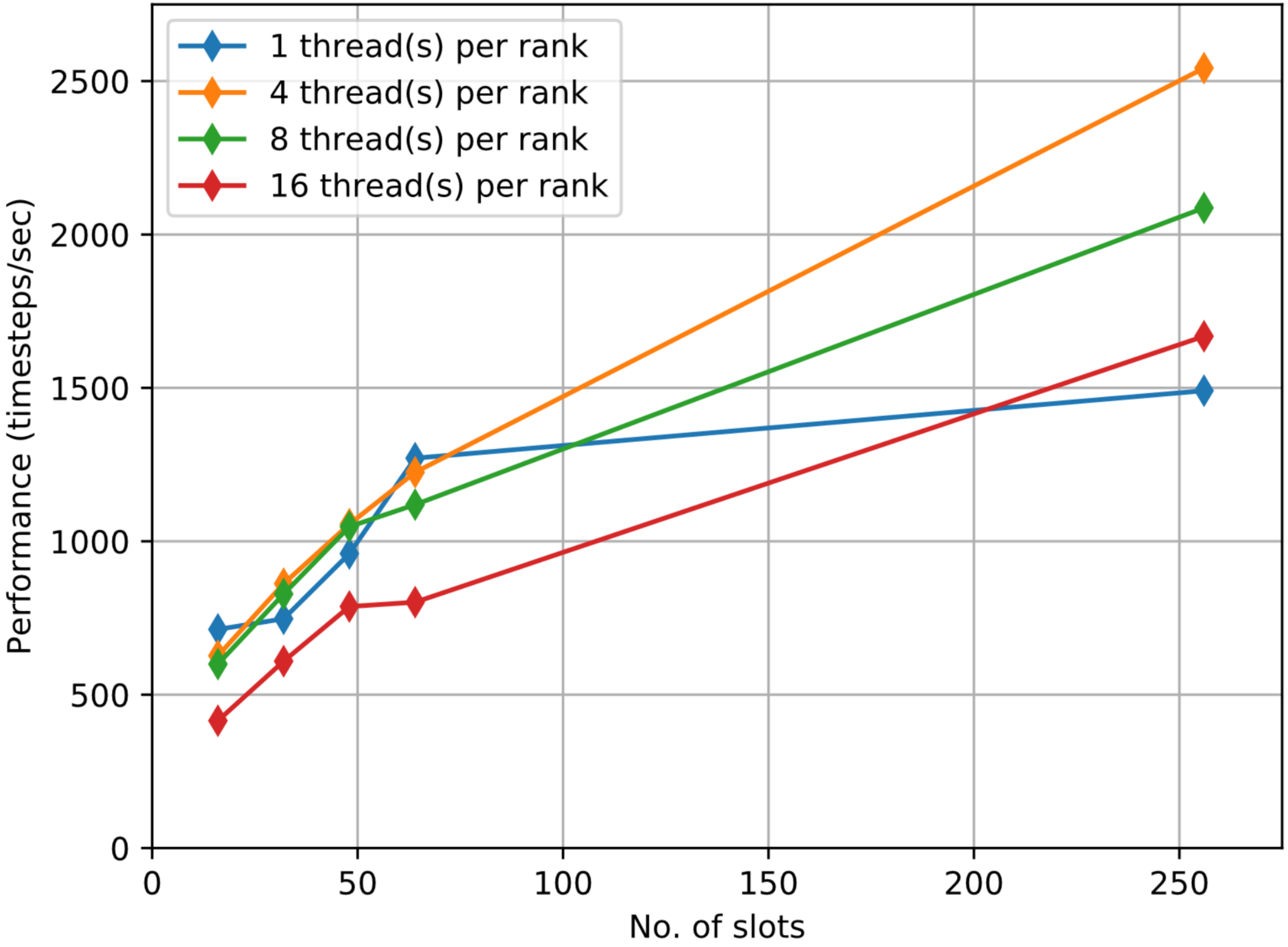
- LAMMPS: Molecular dynamics code.
- Hybrid MPI-OpenMP:
  - MPI – individual “ranks” which are distinct processes; ranks talk to each other for parallel computation; each rank is serial
  - OpenMP – multithreading
  - Hybrid MPI-OpenMP – each rank is multithreaded
- Reading: https://proteusmaster.urcf.drexel.edu/urcfwiki/index.php/LAMMPS#Benchmark_Results_with_Different_Slot_Distributions

LAMMPS

- OMP_NUM_THREADS == 1 => traditional MPI, each rank is serial
- OMP_NUM_THREADS > 1 => hybrid, each rank is multithreaded
- The more timestep/second, the faster performance.



NSLOTS	OMP_NUM_THREADS	Performance (timesteps/sec)
16	16	415.089
	8	599.786
	4	627.102
	1	712.838
32	16	609.115
	8	828.216
	4	861.781
	1	747.488
48	16	787.374
	8	1047.598
	4	1055.877
	1	959.972
64	16	801.444
	8	1119.477
	4	1224.945
	1	1270.930
256	16	1668.267
	8	2086.822
	4	2542.448
	1	1490.833


1 thread(s) per rank
4 thread(s) per rank
8 thread(s) per rank
16 thread(s) per rank
Performance (timesteps/sec)
0
500
1000
1500
2000
2500
0
50
100
150
200
250
No. of slots

LAMMPS: Conclusion

- Performance strongly depends on the type of problem
- For this example, best performance is 4 threads/MPI task.
- If you do not know hardware, try to do like lammps example, find out the best parameters for your work. Change number of slots, num threads.

Question?

Thank you!

References

- https://proteusmaster.urcf.drexel.edu/urcfwiki/index.php/Slurm_-_Job_Script_Example_05a_TensorFlow_With_Anaconda_Python#Note_on_Performance_Tuning_for_Intel_CPUs
- https://proteusmaster.urcf.drexel.edu/urcfwiki/index.php/MATLAB#GPU_Example
- https://proteusmaster.urcf.drexel.edu/urcfwiki/index.php/Slurm_-_Job_Script_Example_06_Matlab
- https://proteusmaster.urcf.drexel.edu/urcfwiki/index.php/GPU_Memory_Limits_for_BERT#Code
- https://en.wikipedia.org/wiki/MNIST_database
- https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/
- https://proteusmaster.urcf.drexel.edu/urcfwiki/index.php/LAMMPS#Benchmark_Results_with_Different_Slot_Distributions
- https://www.mathworks.com/help/parallel-computing/illustrating-three-approaches-to-gpu-computing-the-mandelbrot-set.html;jsessionid=74f67b2220b9dbf1c20b98be5bd4