Deep Learning - lab 4

Hardware acceleration, keras, hyperparametrization

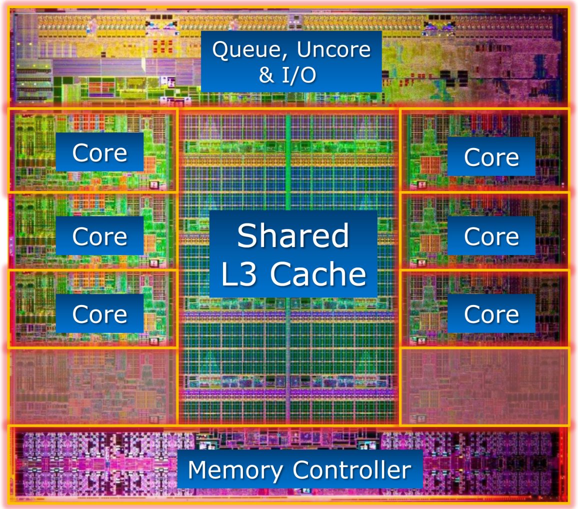
Prof. Stefano Carrazza

University of Milan and INFN Milan

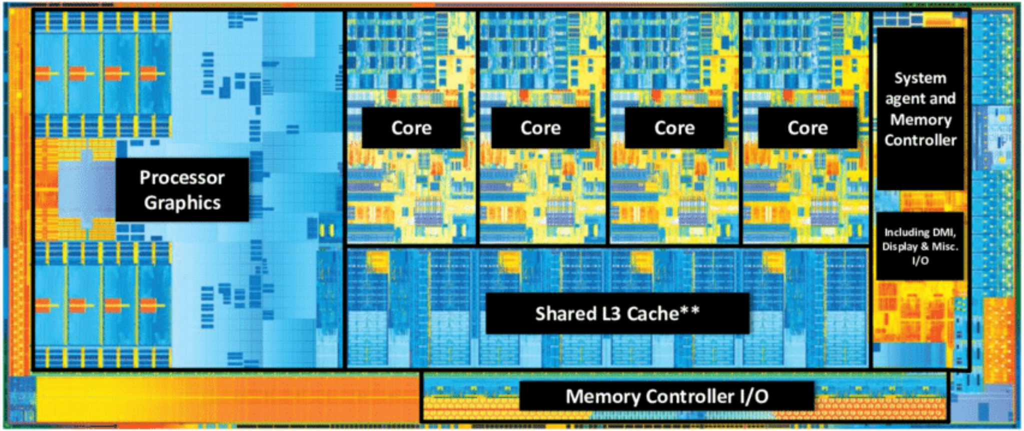
Hardware acceleration

CPU threading

CPU threading



CPU threading



Each CPU core (in a Multi-core CPU) can have multi-thread support (usually 2 threads per core) using hyper-threading technology.

CPU threading

Modern CPUs support three types of parallelism:

- instruction-level parallelism (ILP), done automatically by CPU.
- single instruction, multiple data (SIMD) instructions (SSE, AVX2, etc.)
- thread-level parallelism (TLP) $\rightarrow$ using compilers/libraries

CPU threading

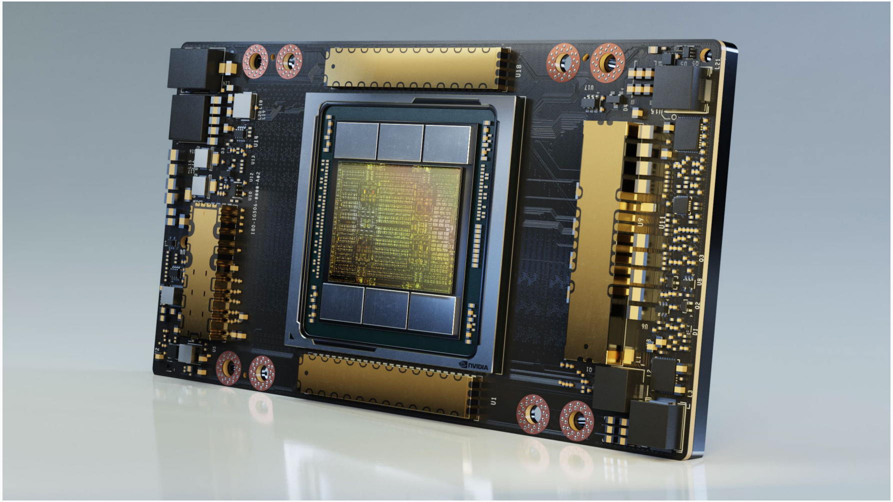
Single-threaded code:

```
const int N = 100000;
double sum = 0, a[N];
for (int i = 0; i < N; i++)
  a[i] = 2 * i;
```

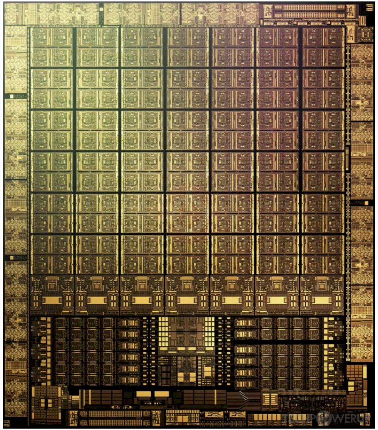
Multi-threading code (using openMP):

```
const int N = 100000;
double sum = 0, a[N];
#pragma omp parallel for
for (int i = 0; i < N; i++)
  a[i] = 2 * i;
```

GPU architecture



GPU architecture



- Larger number of threads when compared to CPU
- High power consumption
- Slower clock speed in comparison to CPU
- Limited RAM memory, e.g. 80GB maximum
- Requires special compilers and SDK, e.g. CUDA (NVIDIA) or ROCm (AMD)

Keras built-in methods

Keras

```python
inputs = keras.Input(shape=(784,), name="digits")
x = layers.Dense(64, activation="relu", name="dense_1")(inputs)
x = layers.Dense(64, activation="relu", name="dense_2")(x)
outputs = layers.Dense(10, activation="softmax", name="predictions")(x)

model = keras.Model(inputs=inputs, outputs=outputs)

model.compile(
  optimizer=keras.optimizers.RMSprop(),  # Optimizer
  # Loss function to minimize
  loss=keras.losses.SparseCategoricalCrossentropy(),
  # List of metrics to monitor
  metrics=[keras.metrics.SparseCategoricalAccuracy()],
)
```

Keras

```
inputs = keras.Input(shape=(784,), name="digits")
x = layers.Dense(64, activation="relu", name="dense_1")(inputs)
x = layers.Dense(64, activation="relu", name="dense_2")(x)
outputs = layers.Dense(10, activation="softmax", name="predictions")(x)

model = keras.Model(inputs=inputs, outputs=outputs)

model.compile(
  optimizer="rmsprop",
  loss="sparse_categorical_crossentropy",
  metrics=["sparse_categorical_accuracy"],
)
```

Keras

```
history = model.fit(
    x_train,
    y_train,
    batch_size=64,
    epochs=2,
    # We pass some validation for monitoring validation loss and metrics
    # at the end of each epoch
    validation_data=(x_val, y_val),
)
print(history.history)
"""
{'loss': [0.3386789858341217, 0.1543138176202774],
 'sparse_categorical_accuracy': [0.9050400257110596, 0.9548400044441223],
 'val_loss': [0.19569723308086395, 0.14253544807434082],
 'val_sparse_categorical_accuracy': [0.9426000118255615, 0.9592999815940857]}
"""
```

Keras

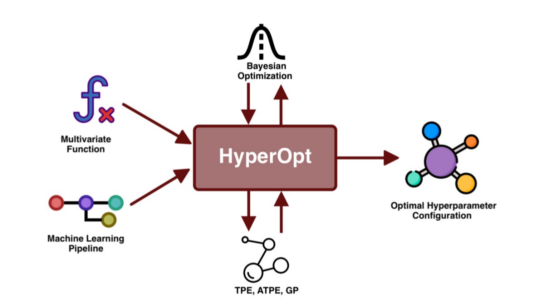
```python
# Evaluate the model on the test data using `evaluate`
print("Evaluate on test data")
results = model.evaluate(x_test, y_test, batch_size=128)
print("test loss, test acc:", results)

# Generate predictions (probabilities -- the output of the last layer)
# on new data using `predict`
print("Generate predictions for 3 samples")
predictions = model.predict(x_test[:3])
print("predictions shape:", predictions.shape)
```

Hyperparametrization

Hyperopt

Hyperopt is a simple python package which provides:

- Random Search
- Tree of Parzen Estimators (TPE)
- Adaptive TPE



`https://github.com/hyperopt`

Hyperopt expressions

```python
from hyperopt import hp

# define search space
a = hp.uniform('a', -10, 10)
b = hp.choice('b', [1, 2, 3, 4])
c = hp.loguniform('c', -5, 0)
# randint, normal, lognormal, ...
```

Hyperopt basics

```python
from hyperopt import fmin, tpe, hp

# define search space
space = hp.uniform('x', -10, 10)

# function to be minimized
def objective(x):
  return x ** 2

best = fmin(objective, space, algo=tpe.suggest, max_evals=100)

print(best)
"""{'x': 0.038665882143389} """
```

Hyperopt basics

```
from hyperopt import fmin, tpe, hp, space_eval

# define search space
space = hp.uniform('x', -10, 10)

# function to be minimized
def objective(x):
  return x ** 2

best = fmin(objective, space, algo=tpe.suggest, max_evals=100)

print(best)
"""{'x': 0.038665882143389 45}"""
print(space_eval(space, best))
"""0.038665882143389 45"""
```

Hyperopt attaching extra information

```
from hyperopt import fmin, tpe, hp, STATUS_OK

# define search space
space = hp.uniform('x', -10, 10)

# function to be minimized
def objective(x):
  return {'loss': x ** 2, 'status': STATUS_OK}

best = fmin(objective, space, algo=tpe.suggest, max_evals=100)

print(best)
"""{'x': 0.038665882143894 5}"""
```

Hyperopt the trials object

```
import time
from hyperopt import fmin, tpe, hp, STATUS_OK, Trials

def objective(x):
  return {'loss': x ** 2, 'status': STATUS_OK, 'eval_time': time.time()}

trials = Trials() # objecting collecting sequential trials
best = fmin(objective, space=hp.uniform('x', -10, 10),
            algo=tpe.suggest, max_evals=100,
            trials=trials)

# trials.trials     : list of dicts representing everything about the search
# trials.results    : list of dicts returned by objective during the search
# trials.losses()   : list of losses (matching ok)
# trials.statuses(): list of status strings
```