QLS-612 2024 Module 5

Inés Gonzalez Pepe

Numpy & Scipy

Numpy

Computing the L_2 norm of a 1,000-dimensional vector

C version:

```c
double squared_norm = 0.0;
for (int i = 0; i != size; ++i) {
  squared_norm += values[i] * values[i];
}
double norm = sqrt(squared_norm);
```

Duration: ~ **1.3 µs**

Python version:

```python
squared_norm = 0.0
for val in values:
    squared_norm += val**2
norm = math.sqrt(squared_norm)
```

Duration: ~ **220 µs**

Why is the C program so much faster?

C pipeline

```
double squared_norm = 0.0;
for (int i = 0; i != size; ++i) {
  squared_norm += values[i] * values[i];
}
double norm = sqrt(squared_norm);
```

Compile ahead of time

```
f20f59c1    mulsd  %xmm1,%xmm0
f20f104df0  movsd  -0x10(%rbp),%xmm1
f20f58c1    addsd  %xmm1,%xmm0
f20f1145f0  movsd  %xmm0,-0x10(%rbp)
...
```

Run on hardware

compute_norm

CPU

Python pipeline

```
squared_norm = 0.0
for val in values:
    squared_norm += val**2
norm = math.sqrt(squared_norm)
```

Compile at runtime

```
LOAD_FAST
LOAD_CONST
BINARY_POWER
INPLACE_ADD
STORE_FAST
...
```

Run on CPython VM

compute_norm.py

python

CPU

The C program spends most of its time performing numerical operations.

The Python program spends most of its time running the interpreter's machinery, inspecting types and looking up functions, allocating and freeing memory, and waiting for memory read operations.

For a fast computation we need

- **static typing** & compiling to optimized **machine code**
- data locality – an **efficient data structure** like a C array

The CPython interpreter is written in C.
It can call C functions and manipulate C data structures.
It provides an API to link our own C code into the Python runtime

numpy provides:

- many routines implemented in C and exposed in Python modules
- a powerful and optimized data structure: the numpy array

C: ~ 1.3 µs

```c
double squared_norm = 0.0;
for (int i = 0; i != size; ++i) {
  squared_norm += values[i] * values[i];
}
double norm = sqrt(squared_norm);
```

Python: ~ 220 µs

```python
squared_norm = 0.0
for val in values:
    squared_norm += val**2
norm = math.sqrt(squared_norm)
```

Python with numpy: ~ 3 µs

```python
norm = np.linalg.norm(values)
```

Getting started with NumPy

```
pip install -U numpy scipy
```

Keeping your friends:

```
export MKL_NUM_THREADS=4
export NUMEXPR_NUM_THREADS=4
export OMP_NUM_THREADS=4
```

```
import numpy as np

my_array = np.array([1, 2, 3])
```

Array creation

From a Python sequence: `np.array([1, 2, 3])`
From a range: `np.arange(10)`
With a fill value: `np.zeros((5, 3))`, `np.ones(2, dtype="int")`
Diagonal matrices: `np.eye(3)`, `np.diag([1, 2, 3])`
With random values: `np.random.random(3)`

Mathematical operations

Element-wise operators: `+, -, *, /, **, %, ==, <, ...`
Matrix multiplication: `@, np.ndarray.dot(), np.dot()`
Transpose: `.T, .transpose()`
Aggregates: `.sum(), .mean(), .max(), .var(), ...`
"universal functions": `np.exp(), np.sqrt(), np.logical_or(), np.maximum(), ...`

Let’s checkout the Jupyter Notebook