High Performance Computing
Homework 11

Dai Nam Nguyen and Sara Restrepo Velasquez
April 28th , 2024

For this assignment the Fortran module **searchutils.f90** was imported to Python using the **f2py** feature in the **Makefile**, as shown in Figure 1. Once this file is complied, the module can be imported and used in Python.

```
all: f2py

f2py: searchutils.f90
	python3.11 -m numpy.f2py -c searchutils.f90 -m searchUtilsTeam07
```

Figure 1. Makefile.

The documentation that instructs the proper use of the module can be obtained by importing the module into **iPython**, as shown in Figure 2.

```
In [4]: print(search.searchutils.__doc__) > searchUtilsTeam07_documentation.txt
idx = linearsearch(arr,x,[n])

Wrapper for ``linearsearch``.

Parameters
----------
arr : input rank-1 array('d') with bounds (n)
x : input float

Other Parameters
----------------
n : input int, optional
    Default: shape(arr, 0)

Returns
-------
idx : int
idx = binarysearch(arr,x,[n])

Wrapper for ``binarysearch``.

Parameters
----------
arr : input rank-1 array('d') with bounds (n)
x : input float

Other Parameters
----------------
n : input int, optional
    Default: shape(arr, 0)

Returns
-------
idx : int
```

Figure 2. Documentation obtained from iPython.

Two scripts were created to test the Fortran module. The first script, **test_module.py** consists of small arrays, sorted and unsorted, that check that the Fortran search functions are working properly. The second script, **performance_eval.py** compares the times the performance of the Fortran algorithms against Numpy's **searchsorted** and **where** functions. It is important to note that since Fortran is 1-indexed and Python is 0-indexed, the resulting indices differ by 1, but this is only to show that all algorithms are working correctly. The results from both scripts can be found in the **results_test_module.txt** and **results_performance_eval.txt** files.

The timing results may be observed in Table 1. The overall fast method was **Fortran's Binary Search**, with a timing of 3.576E-6 s. The slowest method was **Fortran's Linear Search**. The fastest search algorithm to use with a sorted array is **Fortran's Binary Search**, whereas the fastest search algorithm to use with an unsorted array is **Numpy's where**.

Table 1. Timing results of performance_eval.py.

Array	Method	Time [s]
Sorted	Fortran Linear Search	0.009648561477661133
	Fortran Binary Search	3.5762786865234375e-06
	Numpy SearchSorted	3.647804260253906e-05
	Numpy Where	0.009321928024291992
Unsorted	Fortran Linear Search	0.009444475173950195
	Numpy Where	0.008392333984375

For the case of **Binary Search**, the fact that it discards part of the array as it operates, greatly influences the timings that it obtains. The other algorithms need to iterate through the entire array in order to find the desired number, which increases CPU time.

For the case of unsorted arrays, both **Fortran's Linear Search** and **Numpy's Where** obtain very similar timings. It is likely that **Numpy's Where** has been optimized, and that's why it obtains slightly better timings.