Python Tutorial: Map-Reduce

Philip Leong
School of Electrical and Information Engineering
The University of Sydney

THE UNIVERSITY OF SYDNEY

Simple Loop

```
print("Example 1 - simple loop")
def sq(x):
    return x * x

for i in range(10):
    print(sq(i), end=' ')
print()
```

```
Example 1 - simple loop
0 1 4 9 16 25 36 49 64 81
```

Output

```
Example 1 – simple loop
0 1 4 9 16 25 36 49 64 81
Example 2 – list comprehension
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
Example 3 – using map function (actually map returns a list)
<map object at 0x103ba8668>
0 1 4 9 16 25 36 49 64 81
Example 4 – anonymous function
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
Example 5 – reduce
285
285
Example 6 – parallel map
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

List Comprehension (1)

```
print("Example 2 - list comprehension")
print([i * i for i in range(10)])
```

```
Example 2 - list comprehension
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

THE UNIVERSITY OF SYDNEY

List Comprehension (2)

A list comprehension consists of brackets containing an expression followed by a `for` clause, then zero or more `for` or `if` clauses. The result will be a new list resulting from evaluating the expression in the context of the `for` and `if` clauses which follow it. For example, this listcomp combines the elements of two lists if they are not equal:

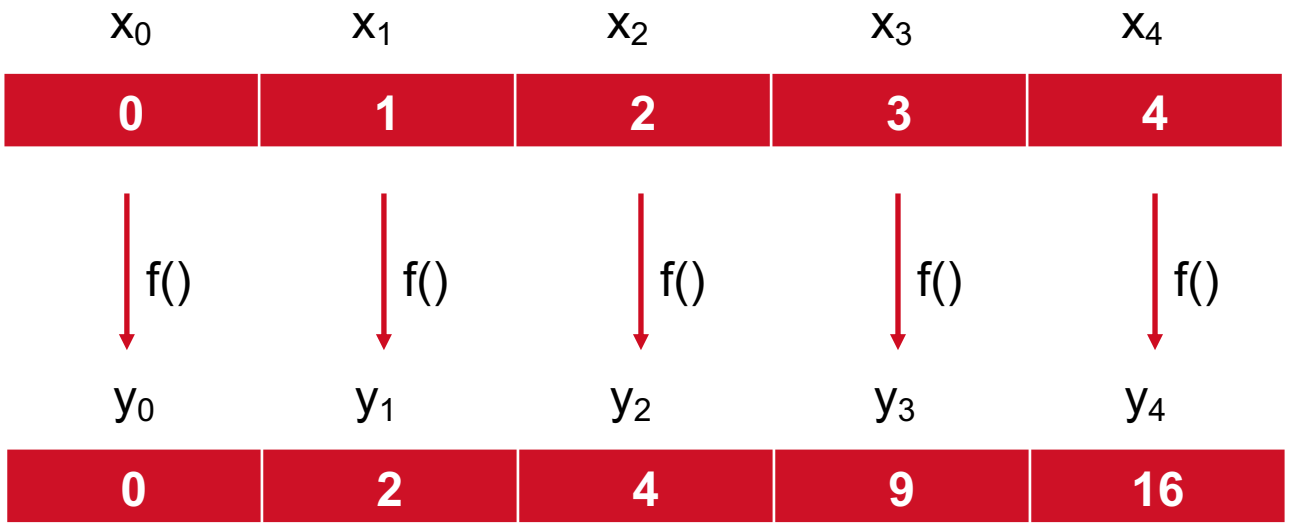
```
>>> [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

and it's equivalent to:

```
>>> combs = []
>>> for x in [1,2,3]:
...     for y in [3,1,4]:
...         if x != y:
...             combs.append((x, y))
...
>>> combs
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

Map Function (1)

- Y=map(f, X)
- Called higher-order function as it takes a function as a argument



Map Function (2)

```
print("Example 3 - using map function (actually map returns a map object)")
l = map(sq, range(10))
print(l)
print(*l)
```

```
Example 3 - using map function (actually map returns a list)
<map object at 0x103ba8668>
0 1 4 9 16 25 36 49 64 81
```

map(*function*, *iterable*, ...)

Return an iterator that applies *function* to every item of *iterable*, yielding the results. If additional *iterable* arguments are passed, *function* must take that many arguments and is applied to the items from all iterables in parallel. With multiple iterables, the iterator stops when the shortest it erable is exhausted. For cases where the function inputs are already arranged into argument tuples, see `itertools.starmap()`.

Anonymous Function

```
print("Example 4 - anonymous function")
print(list(map(lambda x: x * x, range(10))))
```

```
Example 4 - anonymous function
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Reduce

› Also called fold, recombines the results from recursively processing its components (like a list)

```
print("Example 5 - reduce")
from functools import reduce
import operator
l = [i * i for i in range(10)]
print(reduce(operator.add, l))
print(sum(l))
```

```
Example 5 - reduce
285
285
```

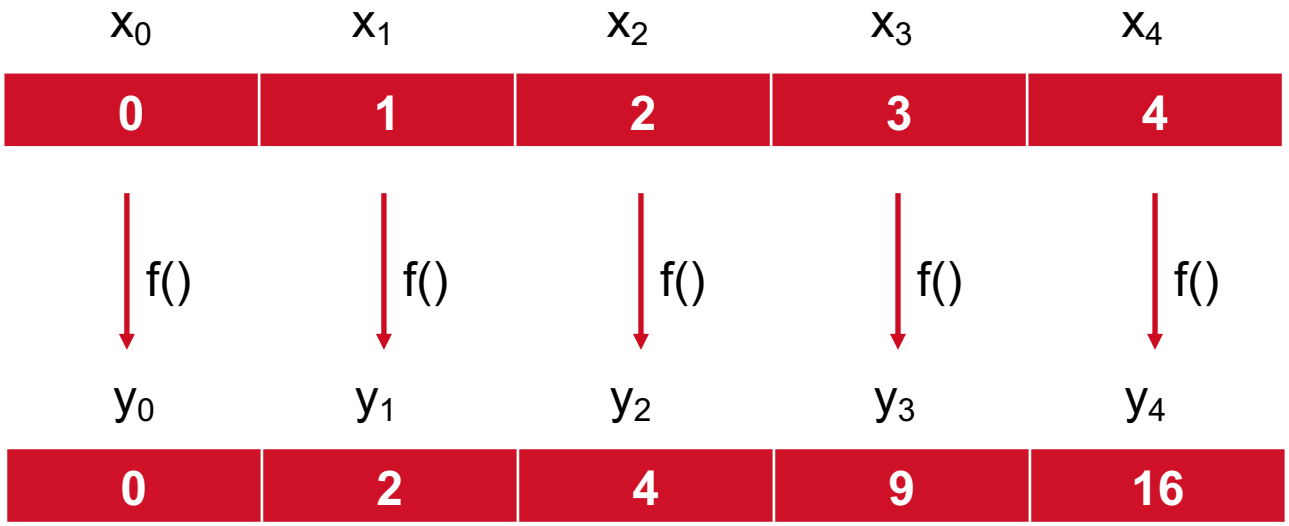
Map-Reduce as a Way to Express Parallelism

[PDF] Google › research › archive › mapreduce-osdi04

MapReduce: Simplified Data Processing on Large ... - Research

by J Dean - 2004 - Cited by 12289 - Related articles

MapReduce: Simplified Data Processing on Large Clusters. **Jeffrey Dean** and Sanjay Ghemawat jeff@google.com, sanjay@google.com. Google, Inc. Abstract.



Parallel Map

```
print("Example 6 - parallel map")
import multiprocessing
pool = multiprocessing.Pool()

def sq(x):
    return x * x

print(pool.map(sq, range(10)))
pool.close()
```

```
Example 6 - parallel map
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

THE UNIVERSITY OF SYDNEY

A Simple Library

```
import collections
import itertools
import multiprocessing

class SimpleMapReduce(object):

    def __init__(self, map_func, reduce_func, num_workers=None):
        self.map_func = map_func
        self.reduce_func = reduce_func
        self.pool = multiprocessing.Pool(num_workers)

    def partition(self, mapped_values):
        partitioned_data = collections.defaultdict(list)
        for key, value in mapped_values:
            partitioned_data[key].append(value)
        return partitioned_data.items()

    def __call__(self, inputs, chunksize=1):
        map_responses = self.pool.map(self.map_func, inputs, chunksize=chunksize)
        partitioned_data = self.partition(itertools.chain(*map_responses))
        reduced_values = self.pool.map(self.reduce_func, partitioned_data)
        return reduced_values
```

Parallel Word Count (1)

```
import multiprocessing
import string
import sys

from simplemapreduce import SimpleMapReduce

def file_to_words(filename):
    """Read a file and return a sequence of (word, occurances) values.
    """

    print(multiprocessing.current_process().name, 'reading', filename)
    output = []

    with open(filename, 'rt') as f:
        for line in f:
            for word in line.split():
                word = word.lower()
                if word.isalpha():
                    output.append( (word, 1) )
    return output


def count_words(item):
    """Convert the partitioned data for a word to a
    tuple containing the word and the number of occurances.
    """
    word, occurances = item
    return (word, sum(occurances))
```

THE UNIVERSITY OF SYDNEY

Parallel Word Count (2)

```
def count_words(item):
    """Convert the partitioned data for a word to a
    tuple containing the word and the number of occurances.
    """
    word, occurances = item
    return (word, sum(occurances))


if __name__ == '__main__':
    import operator
    import glob

    input_files = sys.argv

    mapper = SimpleMapReduce(file_to_words, count_words)
    word_counts = mapper(input_files)
    word_counts.sort(key=operator.itemgetter(1))
    word_counts.reverse()

    print('\nTOP 10 WORDS BY FREQUENCY\n')
    top10 = word_counts[:10]
    print(top10)
    longest = max(len(word) for word, count in top10)
    for word, count in top10:
        print('{}: {}'.format(count, word))
```

Output

```
ForkPoolWorker-1 reading parwc.py
ForkPoolWorker-3 reading mrexamples.py
ForkPoolWorker-2 reading parwc.py
ForkPoolWorker-4 reading simplemapreduce.py

TOP 10 WORDS BY FREQUENCY

[('import', 18), ('for', 14), ('in', 12), ('return', 10), ('a', 9), ('de
, ('i', 5), ('x', 5)]
18: import
14: for
12: in
10: return
9: a
9: def
8: word
6: the
5: i
5: x
```

Applications to ML and FPGAs

[PDF] Neural Information Processing Systems › cc › papers › paper › 315 ...

Map-Reduce for Machine Learning on Multicore

by C Chu - 2007 - Cited by 1527 - Related articles

Map-Reduce for Machine Learning on Multicore. Cheng-Tao Chu. * chengtao@stanford.edu. Sang Kyun Kim. * skkim38@stanford.edu. Yi-An Lin. *.

FPMR: **MapReduce** framework on **FPGA**

Y Shan, B Wang, J Yan, Y Wang, N Xu… - Proceedings of the 18th …, 2010 - dl.acm.org

… **MapReduce** is a parallel programming model proposed by Google [7] for the ease of massive data processing and has been successfully applied to many applications [7, 8, 9, 10]. This model provides two primitives, **map** and **reduce** …

★ Cited by 164 Related articles All 10 versions

Map-reduce as a programming model for custom computing machines

JHC Yeung, CC Tsang, KH Tsoi… - … Symposium on Field …, 2008 - ieeexplore.ieee.org

… on the **map-reduce** higher order functions common in func- tional languages is presented which supports both field pro- grammable gate array (**FPGA**) and graphics … The ap- proximation is then computed as 4h N . **Map-reduce** is used in Google's "**MapReduce**" library to …

☆ Cited by 89 Related articles All 10 versions »

THE UNIVERSITY OF SYDNEY

Summary

- Map – applies a function to elements of an iterator
- List comprehension

```
map(f, iterable)
```

is basically equivalent to:

```
[f(x) for x in iterable]
```

- Reduce – applies operator recursively to a data structure
- Good for removing loops and parallel python code (see Apache Hadoop)
- Multiprocessing library good for parallel programming via processes