

Simulating Systolic Arrays

1. Systolic Sort-

Code:

```
1  import multiprocessing
2  from multiprocessing import Array
3
4  def compare_function(arr, buff, op,j):
5      mina=min(arr[j],op[j])
6      maxa=max(arr[j],op[j])
7      buff[j]=maxa
8      arr[j]=mina
9      op[j]=buff[j-1]
10
11 def systolic_sort(ip,n):
12     arr = Array('i', range(n))
13     buff = Array('i', range(n))
14     op = Array('i', range(n))
15
16     for i in range (0,n):
17         arr[i]=99999
18     for i in range (0,n):
19         buff[i]=99999
20     for i in range (0,n):
21         op[i]=99999
22
23     processes = []
24     iter=(2*n)-1
25     for i in range(0,iter):
26         op[0]=ip[i]
27         for j in range(0,n):
28             process = multiprocessing.Process(target=compare_function, args=(arr, buff, op,j))
29             processes.append(process)
30             process.start()
31
32         for process in processes:
33             process.join()
34         print("Cells after iteration ",i, " is ",arr[:])
35     print("Sorted List:", arr[:])
36     return
37
38 if __name__ == "__main__":
39     input_list = []
40     n = int(input("Enter number of elements : "))
41     for i in range(0, n):
42         ele = int(input())
43         # adding the element
44         input_list.append(ele)
45     print(input_list)
46     input_list.reverse()
47     for i in range(n-1):
48         input_list.append(99999)
49     systolic_sort(input_list,n)
```

Output-

The screenshot shows a VS Code editor with a Python file named `systolic_sort_f.py`. The code implements a systolic sorting algorithm using multiprocessing. It defines a `compare_function` that takes an array `arr`, a buffer `buff`, and an operation `op`. The `systolic_sort` function initializes `arr` and `buff` with large values (99999) and `op` with 1. It then runs a loop where it processes the array in parallel using `multiprocessing.Process`. The terminal output shows the execution of the script, starting with the prompt "Enter number of elements : 5". The output displays the state of the array after each iteration, showing the elements being sorted. The final sorted list is [2, 3, 5, 8, 9].

```
D: > AES > systolic_sort_f.py > systolic_sort
import multiprocessing
from multiprocessing import Array

def compare_function(arr, buff, op):
    min_val = min(arr[i], op[i])
    max_val = max(arr[i], op[i])
    buff[i] = max_val
    arr[i] = min_val
    op[i] = buff[i-1]

def systolic_sort(ip, n):
    arr = Array('i', range(n))
    buff = Array('i', range(n))
    op = Array('i', range(n))

    for i in range(0, n):
        arr[i] = 99999
        buff[i] = 99999
        op[i] = 99999

    processes = []
    iter = (2*n)-1
    for i in range(0, iter):
        op[0] = ip[i]
        for j in range(0, n):
            process = multiprocessing.Process(target=compare_function, args=(arr, buff, op, j))
            processes.append(process)
            process.start()

        for process in processes:
            process.join()

        print("Cells after iteration ", i, " is ", arr[i])
        print("Sorted List:", arr[i])
    return

if __name__ == "__main__":
    input_list = []
    n = int(input("Enter number of elements : "))
    for i in range(0, n):
        ele = int(input())
        # adding the element
        input_list.append(ele)
    print(input_list)
    input_list.reverse()
    for i in range(n-1):
        input_list.append(99999)
    systolic_sort(input_list, n)
```

OUTPUT TERMINAL ...

```
"d:\AES\systolic_sort_f.py"
Enter number of elements : 5
8
2
9
5
3
[8, 2, 9, 5, 3]
Cells after iteration 0 is [3, 99999, 99999, 99999, 99999]
Cells after iteration 1 is [3, 99999, 99999, 99999, 99999]
Cells after iteration 2 is [3, 5, 99999, 99999, 99999]
Cells after iteration 3 is [2, 5, 99999, 99999, 99999]
Cells after iteration 4 is [2, 3, 9, 99999, 99999]
Cells after iteration 5 is [2, 3, 5, 99999, 99999]
Cells after iteration 6 is [2, 3, 5, 9, 99999]
Cells after iteration 7 is [2, 3, 5, 8, 99999]
Cells after iteration 8 is [2, 3, 5, 8, 9]
Sorted List: [2, 3, 5, 8, 9]
PS C:\Users\Tanmay>
```

In this implementation of sorting using systolic arrays, we are using multiprocessing library of Python for parallel computation. We are keeping an array `arr` which gives us the output and two arrays `buff`, `op`. `buff` stores the results in the current iteration and `op` holds the result of previous iteration. I've made two different arrays because we don't want results of both the iterations to be mixed. In an iteration, a cell `arr[i]` takes an input from `op[i]` and compares with its present value. Greater of those values is passed to `buff[i]` and smaller one is stored in `arr[i]`. After an iteration, values of `buff` is passed to `op`. Arrays are initialized with a large value so that they don't interfere with comparison and are forwarded further.

2. Matrix Vector Multiplication

Code-

```

1  import multiprocessing
2  from multiprocessing import Array
3  from array import *
4
5  def worker(a,arr,x,i,j):
6      arr[j]+=x[i]*a[j][i]
7      return
8
9  def multiplication(a,x):
10     arr=Array('i',range(n))
11     op=Array('i',range(n))
12     for i in range (0,n):
13         arr[i]=0
14     for i in range (0,n):
15         op[i]=0
16     processes = []
17     for i in range(0,n):
18         for j in range(0,n):
19             process = multiprocessing.Process(target=worker, args=(a,arr,x,i,j))
20             processes.append(process)
21             process.start()
22         for process in processes:
23             process.join()
24         print("Output after iteration ",i," is ",arr[:])
25
26     print("OUTPUT MATRIX ",arr[:])
27     return
28
29 if __name__ == "__main__":
30     matrix=[]
31     n = int(input("Enter the matrix size:"))
32     for i in range(n):
33         ele =[]
34         for j in range(n):
35             ele.append(int(input()))
36         matrix.append(ele)
37     x=[]
38     print("Input the vector")
39     for i in range(0, n):
40         ele = int(input())
41         x.append(ele)
42     multiplication(matrix,x)

```

Output-

```
D: > AES > matvec_final.py > ...
1 import multiprocessing
2 from multiprocessing import Array
3 from array import *
4
5 def worker(a, arr, x, i, j):
6     arr[j] += a[i] * x[j][i]
7     return
8
9 def multiplication(a, x):
10    arr = Array('i', range(n))
11    op = Array('i', range(n))
12    for i in range(0, n):
13        arr[i] = 0
14        for j in range(0, n):
15            op[j] = 0
16        processes = []
17        for j in range(0, n):
18            process = multiprocessing.Process(target=worker, args=[a, arr, x, i, j])
19            processes.append(process)
20            process.start()
21        for process in processes:
22            process.join()
23        print("Output after iteration ", i, " is ", arr[:])
24
25    print("OUTPUT MATRIX ", arr[:])
26    return
27
28
29 if __name__ == "__main__":
30     matrix = []
31     n = int(input("Enter the matrix size:"))
32     for i in range(n):
33         ele = []
34         for j in range(n):
35             ele.append(int(input()))
36         matrix.append(ele)
37     x = []
38     print("Input the vector")
39     for i in range(0, n):
40         ele = int(input())
41         x.append(ele)
42     multiplication(matrix, x)
```

```
PS C:\Users\tanmay> python -u "d:\AES\matvec_final.py"
Enter the matrix size:3
1
2
3
4
5
6
7
8
9
Input the vector
1
2
3
Output after iteration 0 is [1, 4, 7]
Output after iteration 1 is [5, 14, 23]
Output after iteration 2 is [14, 32, 50]
OUTPUT MATRIX [14, 32, 50]
PS C:\Users\tanmay>
```

In this implementation of matrix-vector multiplication, we have calculated first terms of Y_i .

Let's say we have 4×4 matrix A and vector X.

$$Y_1 = A_{11}X_1 + A_{12}X_2 + A_{13}X_3 + A_{14}X_4$$

$$Y_2 = A_{21}X_1 + A_{22}X_2 + A_{23}X_3 + A_{24}X_4$$

.... and so on

We are computing terms $A_{i1}X_1$ (ith term) parallelly for all Y_i

3. 1-D Convolution

Code-

```

1  import multiprocessing
2  from multiprocessing import Array
3
4  def worker(arr,list,a,i,j):
5      arr[i]+=list[i+j]*a[j]
6      return
7
8  def conv(list,n,a,m):
9      arr = Array('i', range(n-m+1))
10     for i in range (n-m+1):
11         arr[i]=0
12     processes = []
13     for i in range(0,n-m+1):
14         for j in range(0,m):
15             process = multiprocessing.Process(target=worker, args=(arr,list,a,i,j))
16
17             processes.append(process)
18             process.start()
19
20     for process in processes:
21         process.join()
22     print("Output array after iteration ",i," is ",arr[:])
23     print("OUTPUT: ", arr[:])
24     return
25
26 if __name__ == "__main__":
27     input_list = []
28     n = int(input("Enter number of elements in first array : "))
29     for i in range(0, n):
30         ele = int(input())
31         # adding the element
32         input_list.append(ele)
33
34     list2=[]
35     m = int(input("Enter number of elements in second array : "))
36     for i in range(0, m):
37         ele = int(input())
38         # adding the element
39         list2.append(ele)
40
41     conv(input_list,n,list2,m)
42

```

Output-

```

D:\> AES > 1d_conv_final.py > ...
1  import multiprocessing
2  from multiprocessing import Array
3
4  def worker(arr,list,a,i,j):
5      arr[i]+=list[i+j]*a[j]
6      return
7
8  def conv(list,n,a,m):
9      arr = Array('i', range(n-m+1))
10     for i in range (n-m+1):
11         arr[i]=0
12     processes = []
13     for i in range(0,n-m+1):
14         for j in range(0,m):
15             process = multiprocessing.Process(target=worker,
16
17             processes.append(process)
18             process.start()
19
20     for process in processes:
21         process.join()
22     print("Output array after iteration ",i," is ",arr[:])
23     print("OUTPUT: ", arr[:])
24     return
25
26 if __name__ == "__main__":
27     input_list = []
28     n = int(input("Enter number of elements in first array : "))
29     for i in range(0, n):
30         ele = int(input())
31         # adding the element
32         input_list.append(ele)
33
34     list2=[]
35     m = int(input("Enter number of elements in second array : "))
36     for i in range(0, m):
37         ele = int(input())
38         # adding the element
39         list2.append(ele)
40
41     conv(input_list,n,list2,m)
42
P5 C:\Users\Tanmay> python -u "d:\AES\1d_conv_fina
Enter number of elements in first array : 12
9
7
2
4
8
7
3
1
5
8
4
Enter number of elements in second array : 3
1
3
6
Output array after iteration 0 is [42, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Output array after iteration 1 is [42, 37, 0, 0, 0, 0, 0, 0, 0, 0]
Output array after iteration 2 is [42, 37, 62, 0, 0, 0, 0, 0, 0, 0]
Output array after iteration 3 is [42, 37, 62, 70, 0, 0, 0, 0, 0, 0]
Output array after iteration 4 is [42, 37, 62, 70, 47, 0, 0, 0, 0, 0]
Output array after iteration 5 is [42, 37, 62, 70, 47, 22, 0, 0, 0, 0]
Output array after iteration 6 is [42, 37, 62, 70, 47, 22, 36, 0, 0, 0]
Output array after iteration 7 is [42, 37, 62, 70, 47, 22, 36, 70, 0, 0]
Output array after iteration 8 is [42, 37, 62, 70, 47, 22, 36, 70, 80, 0]
Output array after iteration 9 is [42, 37, 62, 70, 47, 22, 36, 70, 80, 57]
OUTPUT: [42, 37, 62, 70, 47, 22, 36, 70, 80, 57]
Ln 41, Col 31  Spaces: 4  UTF-8  CRLF  Python  3.7.3 32-bit  Go Live
Type here to search  13:44 17-11-2023

```

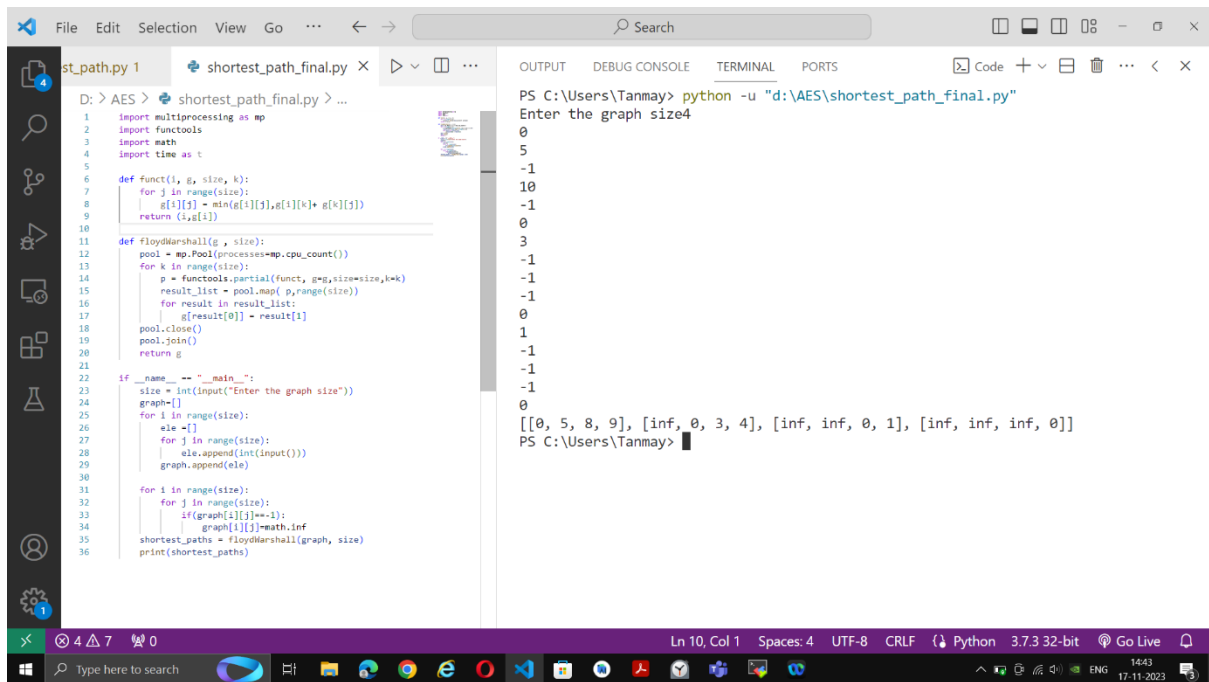
1D convolution is a mathematical operation that involves multiplying and adding an array with another array. The second array is called a kernel, filter, or weights. In one iteration, we are parallelly multiplying second array on a given index(es). We have used similar implementation of systolic arrays as previous programs.

4. Shortest Path

Code-

```
1  import multiprocessing as mp
2  import functools
3  import math
4  import time as t
5
6  def funct(i, g, size, k):
7      for j in range(size):
8          g[i][j] = min(g[i][j], g[i][k] + g[k][j])
9      return (i, g[i])
10
11 def floydWarshall(g, size):
12     pool = mp.Pool(processes=mp.cpu_count())
13     for k in range(size):
14         p = functools.partial(funct, g=g, size=size, k=k)
15         result_list = pool.map(p, range(size))
16         for result in result_list:
17             g[result[0]] = result[1]
18     pool.close()
19     pool.join()
20     return g
21
22 if __name__ == "__main__":
23     size = int(input("Enter the graph size"))
24     graph=[]
25     for i in range(size):
26         ele =[]
27         for j in range(size):
28             ele.append(int(input()))
29         graph.append(ele)
30
31     for i in range(size):
32         for j in range(size):
33             if(graph[i][j]==-1):
34                 graph[i][j]=math.inf
35     shortest_paths = floydWarshall(graph, size)
36     print(shortest_paths)
```

Output-



```
1 import multiprocessing as mp
2 import functools
3 import math
4 import time as t
5
6 def func(i, g, size, k):
7     for j in range(size):
8         g[i][j] = min(g[i][j], g[i][k] + g[k][j])
9     return (i, g[i])
10
11 def floydwarshall(g, size):
12     pool = mp.Pool(processes=mp.cpu_count())
13     for k in range(size):
14         p = functools.partial(func, g=g, size=size, k=k)
15         result_list = pool.map(p, range(size))
16         for result in result_list:
17             g[result[0]] = result[1]
18     pool.close()
19     pool.join()
20     return g
21
22 if __name__ == "__main__":
23     size = int(input("Enter the graph size"))
24     graph = []
25     for i in range(size):
26         ele = []
27         for j in range(size):
28             ele.append(int(input()))
29         graph.append(ele)
30
31     for i in range(size):
32         for j in range(size):
33             if (graph[i][j] == -1):
34                 graph[i][j] = math.inf
35     shortest_paths = floydwarshall(graph, size)
36     print(shortest_paths)
```

```
PS C:\Users\tanmay> python -u "d:\AES\shortest_path_final.py"
Enter the graph size4
5
-1
10
-1
0
0
3
-1
-1
-1
-1
0
1
-1
-1
-1
0
[[0, 5, 8, 9], [inf, 0, 3, 4], [inf, inf, 0, 1], [inf, inf, inf, 0]]
PS C:\Users\tanmay>
```

This is a parallel implementation of Floyd Warshall algorithm. Its single processor runtime is $O(n^3)$. Other implementations like All Pair Shortest path takes worse time. And it was not possible to parallelize Dijkstra's algorithm. In multiprocessing.Process, passing 2D array is not possible. Hence, I used multiprocessing.Pool. I was unaware of its implementation. Hence, used resources online for implementing Floyd Warshall and learning Pool multiprocessing.