Parallel and Distributed computing: Lab Assignment One

Mustapha Tidoo Yussif and Jones Dari

January 2020

1 Introduction

The problem in question is to create a program that steps through the elements in a k-dimensional array with the rank of array varied at runtime. As a result of the dynamic nature of the dimension or the rank of the array, it is not possible to use a nested loop so we implemented an algorithm that uses only one loop to step through the multidimensional array once. Also, we have a method in the program that generates a dope vector (an array that matches an index of one-dimensional array to its corresponding cordinates in k-dimensional space). Creating the dope vector enable map indices in one-dimensional space to cordinates in K-dimensional space.

1.1 Detail Implementation

The three procedures in our program takes K-dimensional array as one of its input. We create one long one-dimensional array with size equal to the product of all the dimensions of the multidimensional correspondent. That is, we represented the multidimensional array as a linear array since its is difficults to nest arrays in C. The long 1-d together with other parameters are then passed to the procedures. How each procedure works is explain below.

1.2 Procedure one

The code for this routine initialises the elements of the array to zeroes using a while loop. This is shown in the psedocode below:

Algorithm 1: Initilizing array elements to zeroes

```
Input: arr, bounds
num_bounds ← sizeof(bounds)/sizeof(bounds[0]);
N ← 1;
i ← 0;
while i < num_bounds do
    N* = bounds[i];
    i++;
end
k ← 0;
while k < N do
    arr[k] ← 0;
    k++;
end
```

1.3 Procedure Two

The code for this procedure takes the array that is initialized to zeroes by the procedure two and set 10 percent of the array uniformly to 1s. It finds the 10 percent of the size of the array and put a one in every position of the array which is a multiple of the value got. For instance if the entire array has size 100. 10 percent of 100 is 10. Hence the code will put a 1 in indices 0, 10, 20, 30, 40, 50, 60, 70, 80, and 90.

Algorithm 2: set 10 percent of array elements to 1s

```
Input: arr, bounds
num_bounds ← sizeof(bounds)/sizeof(bounds[0]);
N ← 1;
i ← 0;
while i < num_bounds do
    N* = bounds[i];
    i++;
end
portion ← 0.1 * N;
k ← 0;
while k < N do
    arr[i] ← 0;
    if k%portion==0 then
        arr[k] ← 1;
    end
    k++;
end
```

1.4 Procedure Three

At this point, we randomly generate numbers. The count of the random numbers is 5 percent of the elements in the array. We then use the random numbers to print the values, the indices and the coordinates, generated by the method dedicated for it. This method is shown below:

Algorithm 3: Randomly print 5 percent of array elements

Input: arr, bounds
$num_bounds \leftarrow sizeof(bounds)/sizeof(bounds[0])$;
$N \leftarrow 1$;
$i \leftarrow 0$;
while $i < num_bounds$ **do**
    $N* = bounds[i]$;
    i++;
end
$portion \leftarrow 0.05 * N$;
$lower \leftarrow 0$;
$upper \leftarrow N$;
$k \leftarrow 0$;
$k \leftarrow 0$;
while $k < N$ **do**
    $index \leftarrow (rand()\%(upper - lower)) + lower$;
    printf("The value = %d, Index = %d", arr[index], index);
     $*cordinates = index\_to\_cordinate(index, bounds)$;
    $arr[i] \leftarrow 0$;
    $m \leftarrow 0$;
    while $m < num_bounds$ **do**
        print cordinates;
        m++;
    end
    k++;
end

1.5 Index to coordinate

This method takes an index of one-d array and the bounds of the corresponding multidimensional array and computer the coordiate that maps to the index in the multidimensional space. For example, say the index of the 1-d array is 1 and the bounds of a 3-d array are 2,3,4, the function coordinates will be calculated as follow:

$1 mod 4 = 1$ $1/4 = 0$
$0 mod 3 = 0$ $0/3 = 0$
$0 mod 2 = 0$ $0/2 = 0$
The coordinates (the modulos) are therefore $(0, 0, 1)$.