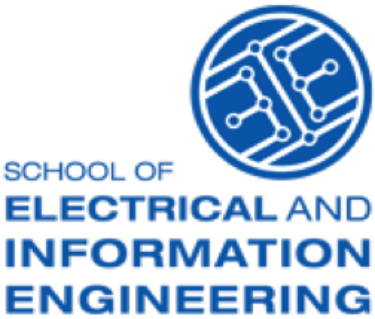

University of the Witwatersrand
School of Electrical and Information Engineering

ELEN4020: Data Intensive Computing

Laboratory Exercise 1

Authors:

Kayla-Jade Butkow
714227

Jared Ping
704447

Lara Timm
704157

Matthew van Rooyen
706692

Date Handed In: 23rd February, 2018

1. Problem Solution

In order to avoid repetition in calculating the capacity of the matrix, a `struct` was created which holds the one-dimensional array, the capacity of the array and the number of dimensions in the required matrix. This `struct` acts as the input to procedures one and two. Procedure three take the `struct` as an input as well as an array containing the dimensions of the original matrix.

For each of the test matrices, each of the functions containing the procedures is called. After the three functions have been called on a matrix, the memory is cleared prior to creating the next matrix.

1.1 Matrix Generation

In order to bypass the problem of requiring a `for` loop for each dimension of the array, the K-dimensional array with bounds N_0, N_1,...N_{k-1} was converted into a one-dimensional (1D) array, with the number of elements (N) determined by Equation 1. The memory for the array was allocated on the heap using `malloc()`. The memory for the array was created such that it could hold N integer values.

Algorithm 1 contains the pseudo-code for the matrix generation.

$$N = \prod_{n=0}^{k-1} N_n \tag{1}$$

1.2 Procedure 1

Since the K-dimensional array was converted into a 1D array, a simple `for` loop was used to set the values in the array to zero. The loop spans from 0 to N (as defined in Equation 1), and sets the value of the array at the corresponding index to zero.

The pseudo-code for this procedure can be found in Algorithm 2.

1.3 Procedure 2

In this procedure, the value of 10% of the elements in the array is set uniformly to 1. The assumption made was that starting at the first element of the 1D array, every tenth element is set to 1. This means that there is a uniform spacing of 9 elements between every two entries with a value of 1. This was achieved using a `for` loop.

The pseudo code for this procedure can be found in Algorithm 3.

1.4 Procedure 3

Since 5% of the elements must be selected, it is necessary to calculate the number of data points to be printed. In order to do this, N is divided by 20.

A random number is then generated between 0 and N-1. This random number acts as the index in the 1D array of the value that should be printed. This index is then added to an array containing all the randomly generated indices. In order to ensure that one element is not printed multiple times, as each random index is generated, it is compared to each other index. If the two indices are the same, a new random number is generated and the process is repeated for the number of data points required.

Once the index is calculated, the set of coordinates corresponding to the index must be calculated. A column-major indexing approach was used, as it simplified the implementation [1]. This choice incurs no performance loss, since the input array to the function is one-dimensional (as created in the matrix generation procedure). However, if operations were required on a multidimensional array that had not been converted into a one-dimensional array, a row-major indexing scheme should be used in C, as the use thereof allows for stride-one access between entries in the array [1]. This drastically improves performance for large matrices [1]. Using column major indexing, the left-hand coordinate is the least significant coordinate, and the right hand coordinate is the most significant. Based on the column major indexing, the coordinates were calculated and displayed as follows:

1. Calculate the maximum amount of elements per dimension
2. Calculate the coordinates from most significant to least significant using the maximum number of elements per dimension
3. Print the coordinates from least significant to most significant

This process is expanded upon using the pseudo-code in Algorithm 4.

REFERENCES

[1] G. Hager and G. Wellein. *Introduction to High Performance Computing for Scientists and Engineers*. CRC Press, 2011.

[2] D. Buttlar et al. *PThreads Programming: A POSIX Standard for Better Multiprocessing*. O'Reilly Media, Inc, Sep 1996.

[3] A. Binstock. "Threading Models for High-Performance Computing: Pthreads or OpenMP?", Oct 2010. URL `https://software.intel.com/en-us/articles/threading-models-for-high-performance/-computing-pthreads-or-openmp`. Last ac-

cessed: 20/02/2018.
[4] B. Kuhn, P. Petersen, and E. O'Toole. "OpenMP versus threading in C/C++." vol. 12, no. 12, pp. 1165–1176, 2000.

Appendix

A Pseudo-code

Algorithm 1 Array Generation

```
function GENERATEARRAY(dimensions, dimension_length)
    capacity = 1
    srand(time(NULL))
    for i = 0 to dimension_length do
        capacity = capacity * dimensions[i]
    created_array = malloc(capacity * sizeof(int))
    if created_array is equal to null then
        catch error
    for i = 0 to capacity do
        created_array = rand() % 10
    return arrayInfo
```

Algorithm 2 Procedure 1

```
function INITIALIZEZERO(arrayContainer, arrayInfo)
    for i = 0 to arrayInfo.array_capacity do
        arrayInfo._array_ptr[i] ← 0
return arrayInfo
```

Algorithm 3 Procedure 2

```
function UNIFORMONE(arrayContainer, arrayInfo)
    amount_to_set ← arrayInfo._array_capacity / 10
    spacing = 10
    for i = 0 to amount_to_set do
        arrayInfo.array_ptr[i * spacing] ← 1
return arrayInfo
```

B PThread and OpenMP Libraries

In complying with the outcomes for the laboratory, the following research regarding the PThread and OpenMP libraries was done.

B1 PThread Library

PThreads is a standardized model for dividing a program into parallel tasks [2]. PThreads was defined by the IEEE POSIX operating system interface standards [2]. The PThreads library specifies the interface to manage the actions required by threads [3].

B1.1 Disadvantages In order to use the PThreads library, the code must be written specifically for the library [3]. This involves the use of PThread specific functions and data structures. The implication of this is that once the library has been used, the application is inherently casted as a threaded model [3]. Furthermore, since the API is very low level, large amounts of complex code is required to perform simple threading tasks [3]. The developer is also required to control all stages of the threading process, such as implicitly stating the number of threads and terminating them after the parallel block has executed [3].

B1.2 Advantages Owing to the inherently low-level API of the PThreads library, extensive control over all aspects of the program is provided [3]. This is advantageous in applications which require low-level control.

Algorithm 4 Procedure 3

```
function isValueInArray(array[], value, array_size)
    for i = 0 to array_size do
        if array[i] is equal to value then
            return True
        return False

function uniformRandom(arrayContainer, arrayInfo)
    amount_to_print ← arrayInfo.array_capacity / 20
    random_indices[amount_to_print];
    spacing = 20
    random_index ← rand() % dimensions[0]
    dimension_capacity_array ← malloc(arrayInfo.array_capacity * sizeof(int))
    array_coordinates ← malloc(arrayInfo.array_capacity * sizeof(int))
    dimension_capacity = 1
    for i =0 to amount_to_print do
        while isValueInArray(random_indices, random_index, i) is true do
            random_index ← rand() % arrayInfo.array_capacity
            random_indices[i] ← random_index;
    for i = 0 to arrayinfo.number_of_dimensions do
        dimension_capacity_array[i] ← dimension_capacity * dimensions[i]
        dimension_capacity ← dimension_capacity * dimensions[i]
    function memset(array_coordinates, 0, sizeof arrayInfo.number_of_dimensions)
    print Format = [coords] : [value]
    for i = 0 to amount_to_print do
        target_index ← random_index + i * spacing
        for j = arrayInfo.number_of_dimensions to 0 do
            if target_index/dimension_capacity_array[j-1] > 0 then
                array_coordinates[j] ← target_index/dimension_capacity_array[j-1]
                target_index ← target_index - (array_coordinates[j] × _dimension_capacity_array[j-1])
            array_coordinates[0] ← target_index;
        for k = 0 to arrayInfo.number_of_dimensions do
            print array_coordinates[k]
    return arrayInfo
```

B2 OpenMP Library

The OpenMP library was developed to provide an API which could run the same code base, equally well, across a range of operating systems [3]. The OpenMP library takes the form of a set of compiler directives (pragmas) which are operating system-specific [3]. Through the sensible use of OpenMP pragmas, a single threaded program can be parallelised while maintaining its serial structure [3, 4].

B2.1 Advantages If a compiler does not recognise an OpenMP pragma it is ignored [1]. As a result, a level of security exists where code containing these library-specific directives can be compiled by a different toolset, without concerns of it breaking [3]. Additionally, by disabling support for OpenMP, the code can be compiled as a single-threaded program and debugged with relative ease [3].

Unlike PThreads, OpenMP does not lock the software into a set number of threads for a particular application. In this way the number of threads is determined at run-time and can be scaled according to hardware availability [3].

B2.2 Disadvantages If a finer level of control is required, developers can access OpenMP's threading API. At this level, the functionality provided by the OpenMP API is a small in comparison to that of PThreads [3]. A far smaller range of primitive functions exist which are needed to achieve a fine-grained level of control [3].

Algorithm 5 Main Function

```
function MAIN
    dimensions[] == [6,6,6]
    arrayContainer generated_array ← generateArray(dimensions, sizeof(dimensions)/sizeof(dimensions[0]));
    generated_array ← initializeZero(generated_array);
    generated_array ← uniformOne(generated_array);
    generated_array ← uniformRandomOne(generated_array, dimensions);
    free(generated_array.array_ptr);
```