Data Partitioning for Multi-threaded Applications A case study using Median Filter and Pthreads

Petrus Kambala† and Thato Semoko‡
EEE4084F Class of 2018
University of Cape Town
South Africa
†KMBPET001 ‡SMKTHA004

***Abstract*—This paper investigates the efficiency of Multi-threaded programs by considering a design of a 2D median filter and Pthreads to remove noise data. The design of the median filter includes a choice between various sorting algorithms and the overral size of the filter.**

I. Introduction

Data partitioning is a process of dividing data into blocks among multiple threads. Each thread execute the same instructions on its designated block of data which in theory should results in a speed up of some sort. This investigation therefore aims to explore this approach using pthreads and median filter- which utilize statistics to remove extraneous values(noise) from data sets. The data set can be of arbitrary dimensions, however in this case only images will be used.

II. Methodology

To investigate the speedup attained by partitioning data for multi-threaded applications the following approach was taken. A 9 by 9 median filter will be used. The codes used for the filter is attached to this report.

A. Golden Measure

A Golden Measure is a solution that is not optimized and may therefore run very slow however, it ideally produces excellent results. For each pixel the median filter will sort the values covered by its window and pick the median value. A non-optimized sorting technique, bubble-sort with a worst-case and average complexity of $O(n^2)$ algorithm, will be used.

The filtering was run 5 times to allow for "cache warm up" before the time taken to filter is recorded. The output image produced was then stored for use as reference.

```
void bubbleSort(vector<JSAMPLE> &arr) {
    for (int i = 0; i < arr.size(); i++) {
        for (int j = 1; j < (arr.size() - i); j++) {
            if (arr[j - 1] > arr[j])
                                swap(arr[j-1], arr[j]);
        }
    }
}

//-------------------------------------------------------------------------
void swap(JSAMPLE &t, JSAMPLE &a) {
    JSAMPLE val = t;
    t = a;
    a = val;
}
```

Listing 1.1 the c++ codes for the bubble sort function

B. Multi-Threaded Version

In this approach, Pthreads was used. Pthreads is a standardized model that facilitates the *divide and conquer* technique for parallel programming [2]. We divide the data such that we only spawn two threads then the 1st thread works on the top half and the 2nd thread on the bottom half and recursively spawn other two threads for each half array each thread is given. The maximum number of threads used is always a power of 2 and is less than the height of the image to be filtered.

For sorting the data, the quickSort algorithm was used, the pivot selection and partitioning steps was done with the Hoare partition scheme [1] which performs well compared to other partition schemes.

Similar to the Golden Measure, the filter was run 5 times before recording the total time taken by all the threads to filter the image.

```
void quickSort(vector<JSAMPLE> &arr, int lo, int hi) {
    JSAMPLE pivot = arr[lo + (hi - lo) / 2];
    int i = lo, j = hi;
    while (i <= j) {
        while (arr[i] < pivot)//move in from left
            i++;
        while (arr[j] > pivot)//move in from right
            j--;
        if (i <= j) {
            swap(i, j);
            i++;
            j--;
        }
    }
    if (lo < j)quickSort(arr, lo, j);
    if (hi > i)quickSort(arr, i, hi);
}
```

Listing 1.2 Codes used for the quickSort algorithm

III. Results

After carrying out the experiments the following results were obtained:

A. Golden Measure

The average time of execution and the output image subsequent to filtering is shown below;

Time taken to filter "small.jpg": 11751.1 ms.

As expected, the golden measure, in this case, time of execution for a serial median filter to run on as a single

Fig. 1. Golden measure Output image

process. The output image as seen from figure 1 has been accuretely filtered.

B. Multi-Threaded Version

The multi-threaded program aims to optimize the Golden Measure results and uses the quicksort algorithm as mentioned in subsection II-B

C. Serial vs Threaded

The table below shows how the threaded filtering compares to the golden measure by also showing the speed up.

Number of threads	Time (ms)	Speed Up
1	2243.22	5.1708
2	1161.08	9.99
4	1004.63	11.545
8	996.979	11.6344
16	989.761	11.72
32	962.678	12.05
64	845.539	13.7
128	853.736	13.6

Fig. 2. Comparison of serial median filtering and threaded filtering

Figure 3 shows how the speed up with using multiple threads with the median filter compares to the golden measure. The speed up grows very fast between 2 and 8 threads, and then starts to grow slower from there until with 64 threads where diminishing returns start to appear.

However, this approach performs much better that the serial version of the algorithm, we used only one partitioning scheme across threads, letting thread 1 handle rows 1 to 8, thread 2 rows 9 to 16 etc. **Why?** Also, the RAM access stage for each thread need not to go into the critical section because the ...

The correctness of this algorithm however, matches that of the golden measure, we used the correlation function from Octave to ensure that the filtered pixels are accurate. Correlation gave a coefficient of 1.

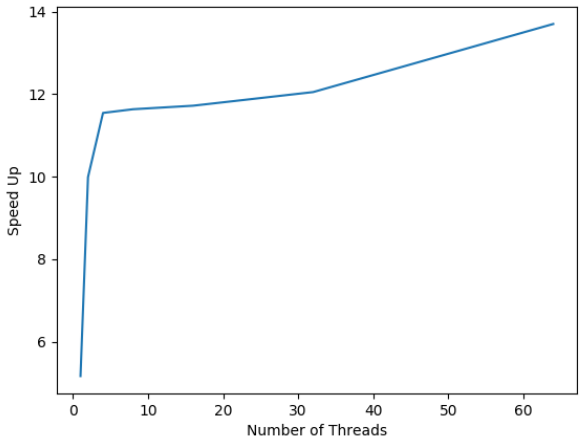

Fig. 3. Speed up with partitioning the data equally amongst threads (partitioning the rows sequentially)

IV. Conclusion

The conclusion should provide a summary of your findings. Many people only read the introduction and conclusion of a paper. They sometimes scan the tables and figures. If the conclusion hints at interesting findings, only then will they bother to read the whole paper.

You can also include work that you intend to do in future, such as ideas for further improvements, or to make the solution more accessible to the general user-base, etc.

Publishers often charge "overlength article charges" [2], so keep within the page limit. In EEE4084F we will simulate overlength fees by means of a mark reduction at 10% per page. Late submissions will be charged at 10% per day, or part thereof.

References

[1] "Quick sort algorithm using hoare's partitioning scheme," 2018.
[2] "Voluntary Page and Overlength Article Charges," http://www.ieee.org/advertisement/2012vpcopc.pdf, 2014.