Neural, Parallel, and Scientific Computations 29(2021), No.1, 21 - 38

A PARALLEL IMPLEMENTATION OF THE ROTATIONAL COMPENSATOR PHASE UNWRAPPING ALGORITHM

FATMA GAMAL[1], MOSTAFA SOLIMAN[2], AND SAMIA HESHMAT[3]

[1] [2] Computer and Systems Section -[3]Communication and Electronic Section,
Electrical Engineering Department, Aswan University, Aswan, Egypt.
[2]Computer Science and Engineering Department,
Egypt-Japan University of Science and Technology, Egypt.

ABSTRACT. Phase unwrapping algorithms are essential in image processing for optical measurements. Many research fields used phase unwrapping algorithms to recover phase data in a 2D phase data map. The rotational compensator phase unwrapping algorithm (RC) has high accuracy compared to other algorithms. However, RC takes a long time to execute. In this paper, parallel processing techniques such as SIMD and multithreading are exploited on multicore processors to accelerate the RC algorithm. Exploiting SIMD instructions inside multiple threads on a machine with Intel Xeon gold 6140 processor accelerates the RC algorithm by 3.11. Moreover, implementing a mathematical approximation of arctangent beside the parallel RC algorithm increases the speedup by 4.36 times over the original algorithm.

AMS (MOS) Subject Classification. 39A10.

1. INTRODUCTION

Two-dimensional (2D) phase unwrapping is a key data-processing chains in various applications used in many research fields, such as optics, magnetic resonance imaging, and synthetic aperture radar interferometry [1–7]. The phase unwrapping process has a direct influence on the accuracy of final results; this is why many algorithms have been proposed [7–19]. The rotational compensator phase unwrapping algorithm (RC) has high accuracy compared to other phase unwrapping algorithms [1, 6]. However, it takes a long time to execute. The time is an important factor in the applications that use the phase unwrapping algorithms.

Parallel processing plays a critical role in reducing execution time in multimedia applications by processing multiple instructions simultaneously. Single instruction multiple data (SIMD), multithreading, and blocking are substantial parallel processing techniques that run on multicore processors to accelerate applications on it [20].

Received January 17, 2021 1061-5369 $15.00 © Dynamic Publishers, Inc.
www.dynamicpublishers.org; www.dynamicpublishers.com;
https://doi.org/10.46719/npsc20202912.

SIMD is used to accelerate applications that have an enormous amount of data parallelisms such as multimedia applications [21–28]. The goal of the multithreading technique is to improve the utilization of a single core or a single processor by using thread-level parallelism [29]. Blocking depends on space loops iterations into smaller chunks or blocks to reduce cache miss overhead, and reuse the data [30].

This paper analyzes and profiles the RC algorithm to exploit multicore systems for accelerating RC algorithm as follows:

- Develop and implement an approximation mathematical equation of arctan function
- Implement the SIMD version of the RC algorithm by using AVX2 and AVX512 instructions
- Exploit multicore processors by developing the multithreading version of RC algorithm
- Exploit memory hierarchy by dividing the data of the RC algorithm into blocks to fit into cache memory
- Put all these techniques together to get a speedup close to the ideal value

We have got that the parallel RC algorithm speedup is 4.36 times on average over the original RC algorithm. Where, the time of processing 1000x1000 image is reduced from 8248 seconds (2.29 hours) to 1842 seconds (0.51 hour). The remainder of this paper is organized as follows: Section 2 presents related work. The phase unwrapping, the problem that the RC algorithm solves, and the profiling of the RC algorithm are described in details in Section 3. Section 4 presents the parallel implementation of the RC phase unwrapping and the arctan function approximation used. Section 5 represents the results. Finally, Section 6 summarizes this paper.

2. **RELATED WORK**

Phase unwrapping is a mathematical problem-solving technique increasingly used in synthetic aperture radar (SAR) interferometry, optical interferometry, adaptive optics, and medical imaging [31]. López et al. method used multithreading to accelerate Goldstein phase unwrapping algorithm. A machine with a four-core Intel corei7 processor at 3.4 GHz working frequency and 11.7 GB of RAM is used [32]. This method can be speeded up the execution time to 12 times over the sequential algorithm for 7202x7202 image size. Huang el al. method used multithreading with blocking to accelerate Goldstein phase unwrapping algorithm [33]. In this method a machine has an HPC cluster with two 2.67-GHz quad-core Intel Xeon 5550 processors, and 12 GB of memory is used. The results can be generated by this method with more than 25 times speedup over the serial implementation with the use of 64 processors. Meanwhile, in Barabadi et al. method multithreading is used to accelerate the dual-stage phase unwrapping algorithm based on the branch-cut method [34]. A machine with an Intel Core i7-8700 CPU 3.2 GHz with six physical cores and 32 GB of memory is used. The speedup for this method is about 6.5 times than the original algorithm.

Furthermore, Karasev et al. method is used the GPU to accelerate the weighted least-square phase unwrapping algorithm [35]. In this method a machine with an NVIDIA Geforce 8800GTX video card having 768 MB video RAM and 1 GB of system RAM is used. This algorithm is run for data grids ranging in size from 256x256 to 2048x2048 corresponds to GPU speedup of 5.3 to 34.5 relative to the CPU. The method by Mistry et al. is used the GPU to accelerate the minimum L^P norm phase unwrapping algorithm on a machine with an NVIDIA GeForce 8800GTX GPU with 16 multiprocessors and 128 scalar processors running on 1500 MHz [36]. The speedup of this method is 7.3 over the CPU. In the method by Wu et al. the GPU is used to accelerate Goldstein's phase unwrapping algorithm on a machine with an Intel Xeon X5550 CPU and an NVIDIA Tesla C2050 GPU [37]. The GPU has 14 multiprocessors with 448 cores; each of the multiprocessors has 64KB cache/shared memory. The speedup of this method is more than 780 times over the CPU.

The methods of Karasev et al. and Mistry et al. [35, 36] used the least-squares algorithm with discrete cosine transforms. Moreover, the methods [32–34, 37] used the Goldstein and the branch-cut algorithms. The performance evaluation of the RC phase unwrapping algorithm among Goldstein and Least-Squares algorithms shows that RC has higher accuracy [6]; though it has high time cost than them. However, the primary selection criterion for picking an algorithm is the quality of the unwrapped results for the available data. Hence, we choose the RC phase unwrapping algorithm to be the first contribution accelerating its process. Furthermore, the methods implemented by López et al., Huang el al. and Barabadi et al. use machines have a CPU to accelerate the phase unwrapping algorithms [32–34]. Meanwhile, the used machines in the methods implemented by Karasev et al., Mistry et al. and Wu et al. [35–37] have a GPU to accelerate the phase unwrapping algorithms. Therefore, multicore systems have become more commercially prevalent; we prefer using CPU in our work.

3. **PROFILING ROTATIONAL COMPENSATOR PHASE UNWRAPPING PROCESS**

Due to the importance of the phase unwrapping process in multimedia applications, the need of accurate and robust unwrapping methods are essentials. In these applications, the phase carries information about physical quantities. However, phase mapping is ambiguous because the extracted phase returned in a form that suffers from 2π phase jumps. In this case, the phase data is unwrapped to fit for usage. Furthermore, the presence of noise in the measured data has many problems to produce accurate unwrapped results; since many singular points (SPs) are found. This noise has an essential effect on the computational time of any unwrapping method.

3.1. **Mathematical Background.** Phase unwrapping algorithms are based on one assumption that the true unwrapping phase data Φ_i is less than one-half cycle, which progressively various enough to make the phase difference between the neighbour's values be within the one-half cycle (π rad) of each other, as shown in Eq.(3.1) [1].

$$|\Delta\Phi^i| = |\phi_{i+1} - \phi_i| < \pi \tag{3.1}$$

Where $|\Delta\Phi^i|$ is the difference of the true phases. If this assumption is true everywhere the unwrapping process can be applied by integrating wrapped phase differences or gradient throw any path from one pixel to another one in the data; to generate the unwrapped phase. The equation of the wrapped gradient phase difference between two pixels is as shown in Eqs.(3.2) and (3.3), respectively [1].

$$\nabla\Psi^i = \Delta\Psi^i - Int[\frac{\Delta\Psi^i}{2\pi}]2\pi \tag{3.2}$$

$$\Delta\Psi^i = \Psi_{i+1} - \Psi_i \tag{3.3}$$

Where, Ψ_i is the wrapped phase at pixel i, and $\nabla\Psi^i$ is the wrapped gradient phase difference.

In the absence of discontinuity sources, the unwrapped result is independent of the unwrapping path. Therefore, the unwrapped phase map is consistent. Considering that a path consists of M points, the points are numbered from 0 to $M-1$. If the difference of the true unwrapped phases satisfies this relation $|\Delta\Phi^i| < \pi$ [1], where the wrapped gradient is identical to the difference of the true phases $\nabla\Psi^i$=$\Delta\Phi^i$, we can retrieve the true unwrapped phase as follows:

$$\Phi_M = \Phi_0 + \sum_{i=0}^{M-1} \nabla\Psi^i \tag{3.4}$$

In the 2D phase unwrapping, there are paths with a loop, which means that the last point can be considered the first point. In the case where point M is identical to point 0, if the relation $|\Delta\Phi^i| < \pi$ is satisfied, the summation of $\nabla\Psi^i$ for all points (from 0 to $M-1$) must equal to zero. See [1] for more details.

Due to discontinuity, the path of integration becomes dependent. It isn't possible to choose a path randomly. If Eq.(3.4) used to retrieve the unwrapped phase, it suffers from multiples of 2π addition or subtraction fault that affect all over the phase map.

Restrictions are applied to the unwrapping path in the corrupted areas, which result in the path being dependent. To avoid this situation; corrupted areas or SPs must be identified, balanced, and isolated from the rest of the non-singular pixels using barriers (branch cuts) in the phase map. Once SPs are isolated, phase unwrapping process will take an independent path avoiding these branch cuts; therefore, it retrieves the true phase [1].

The phase unwrapping process has a direct influence on the accuracy of final results; therefore, many algorithms have been proposed. The RC is a phase unwrapping approach for noisy wrapped phase maps of continuous objects, the performance evaluation of the RC phase unwrapping algorithm among other algorithms shows that RC has higher accuracy [1,6].

3.2. **Rotational Compensator.** The idea of the compensator is proposed to compensate and cancel the singularity effect. One of the methods is the RC method. This method computes the compensator by superposing the impact of each SP. The RC can cancel the singularity of each SP by adding an integral of isotropic singular function along with any loops. When a closed-loop includes SP, the integral along the loop will have a value of $2\pi S$, where S is the residue of the SP, as shown in the following equation [1]:

$$\sum_{i=0}^{3} \nabla\Psi^i = 2\pi S \tag{3.5}$$

Representing an integral of segment i, which is a member of the loop comprising N segments, as C^i, we can reduce Eq.(3.5) to:

$$\sum_{i=0}^{3} (\nabla\Psi^i + C^i) = 0 \tag{3.6}$$

These suggest that the singularity of Ψ^i regularized by compensator C^i, and phase unwrapping becomes an independent path. The RC for the i^{th} segment, which is a path from r_i to r_{i+1} that cancel the singularity of the SPs, ${}^{R}C^i_j$ represented as follows:

$$ {}^{R}C^i_j = -S_j(\Theta_{i+1j} - \Theta ij) \tag{3.7}$$

Where S_j denotes the residue of the j^{th} SP, and Θ_{i+1j} with Θij are azimuthal angles of both ends of the i^{th} segment, where the origin is located at the j^{th} SP. When the measured data contains several SPs [1], the total compensator of the i^{th} segment is estimated as the summation of the ${}^{R}C^i_j$ for j [1]:

$$ {}^{R}C^{i} = \sum_{j=1}^{N} {}^{R}C_{j}^{\ i} \tag{3.8} $$

Consequently, we can retrieve the true unwrapped phase data by summing the phase differences between the adjoining pixels and the total compensator as follows:sss

$$ \Phi_M = \Phi_0 + \sum_{i=0}^{M-1} (\nabla\Psi^i + C^i) \tag{3.9} $$

Where C^i is equal ${}^{R}C_j^i$ and the path is composed from 0 to $M-1$ segment [1].

3.3. **Profiling RC Method.** The RC method is divided into three stages as shown in Figure 1. In the singular point computations stage, the position of SPs in the image is computed to detect the places of each SP in the data. Meanwhile, SP positioning and pairing stage, the effect of the compensator is confined to a smaller region and to determine the dipole pairs. However, the RC stage can remove the inconsistency by canceling the singularity effect [1]. Therefore, to produce the unwrapped image, the wrapped image is entered the system and goes through the stages.

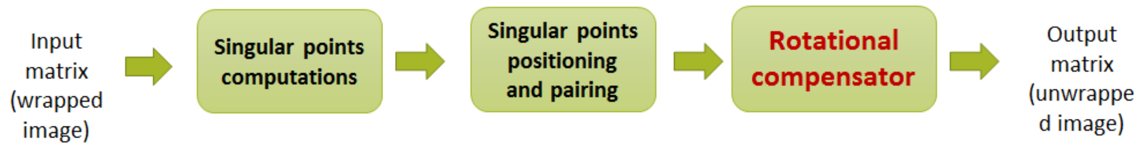
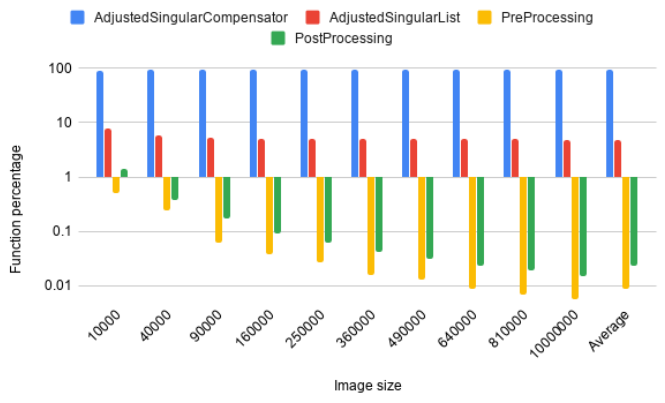

Input matrix (wrapped image)
Singular points computations
Singular points positioning and pairing
Rotational compensator
Output matrix (unwrapped image)

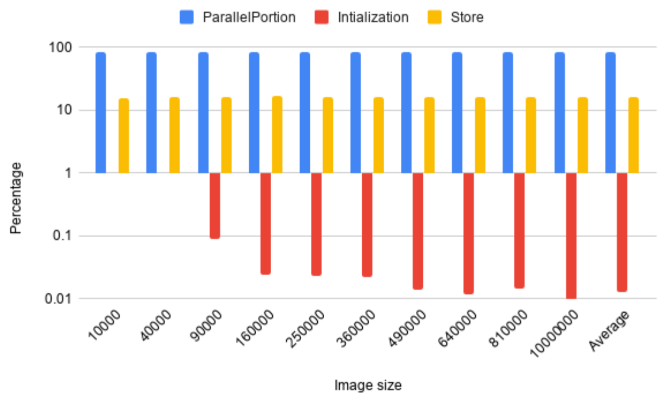
Figure 1. RC Stages

By analyzing and profiling the stages of the RC method we found that the computation time is proportional to KN^2, N denotes one-dimensional area size in pixels. From Eq.(3.8), the cost of time to compute the RC for all segments equals the product of SPs number and segments number of the compensated path. Since both are proportional to the area size ($\propto N^2$) [1]. The number of SPs isn't exactly equaled the number of rows multiply columns. Therefore we consider the number of SPs as K, and we can represent the complexity of the RC method as $O(KN^2)$. By profiling the RC functions we found that the function AdjustedSingularCompensator (ASC) takes about 95.06% of the overall time. According to the principle "make the common case fast" [26], we focus on the ASC function which is the bottleneck of the RC method. Figure 2(a) shows the profiling of the most time-consuming functions of the RC method on a logarithmic scale.

The profiling of ASC function before performs parallel processing technique shows that the function performs approximately $2N^2$ atan2() calculation, $2N^2$ for subtraction, division, multiplication and addition calculations, and $7N^2$ load/store operation.



(a) RC Profiling



(b) ASC Function Profiling

FIGURE 2. Profiling RC Method

It is observed that the function is memory intensive with little computational intensive.

The complexity of the ASC function is $O(KN^2)$. It depends on the number of rows multiplied by the number of columns. Assuming the number of rows is equal to the number of columns (rows=cols=N). While the number of SPs isn't constant, varying from image to another and increasing with image size (assuming it K). Therefore, the total complexity of the adjusted singular compensator function is $O(KN^2)$. From profiling and analyzing ASC function it can be found that we can run horizontal compensator calculation in parallel with vertical compensator calculation which we call parallel portion. The parallel portion takes 83.64% from ASC function time and 79.51% from RC time. Figure 2(b) shows ASC function profiling on a logarithmic scale.

According to Amdahl's law [38], if P is the proportion of a system or program that can run parallel, and $1 - P$ is the proportion that remains serial, then the maximum speedup achieved using N number of processors is:

$$Speedup(N) = \frac{1}{1 - P + \frac{P}{N}} \tag{3.10}$$

If N tends to infinity, then the maximum speedup tends to 1/(1-P). Therefore the maximum speedup that the RC method can reach is 4.88.

4. PARALLEL IMPLEMENTATION OF RC PHASE UNWRAPPING

This paper aims to accelerate the RC without affecting the accuracy of the algorithm exploiting parallel processing techniques on multicore processors. Investing multicore processors has many challenges such as implement the SIMD version of the RC algorithm, develop the multithreading version of the RC algorithm, exploit memory hierarchy by dividing the data of the RC algorithm into blocks to fit into cache memory, and put all these techniques together to get a speedup close to the ideal value. Algorithm 1 shows the parallel implementation of the RC phase unwrapping.

```
1  Function ASC
       input  : $Image$, $Singular$
       output: $Compensate$
2      $Compensate \longleftarrow$ Allocate with size of $Image$
3      $Comp \longleftarrow$ Allocate with size of $Image$
4      $dh \longleftarrow$ Allocate with size of $NumOfCols$
5      $dv \longleftarrow$ Allocate with size of $NumOfRows$
6      $Theta0 \longleftarrow 0.0$
7      $Theta1 \longleftarrow 0.0$
8      $Sing \longleftarrow$ user define data type
9      //$Compensate$ is user define data type
10     //$Image$ contains $NumOfCols$ and $NumOfRows$
11     for $Col = 0$ to $NumOfCols - 1$ do
12         for $Row = 0$ to $NumOfRows - 1$ do
13             $Compensate[Col][Row].Horizontal = 0$
14             $Compensate[Col][Row].Vertical = 0$
15     for $i = 0$ to $Singular.Length - 1$ do
16         $Sing = SingularData[i]$
17         //horizontal and vertical compensator can run in parallel
```

Algorithm 1: Parallel RC Algorithm

```
18 thread begin 0 to $N-1$
19     //horizontal compensator
20     for $Col = 0$ to $NumOfCols - 1$ do
21         do in parallel
22             Load $n$ elements in $dv$
23         $Theta0[0] = atan2(-dv[0], dh[0])$
               ⋮
24         $Theta0[n-1] = atan2(-dv[n-1], dh[0])$
25         for $Row = 0$ to $NumOfRows - 1$ do
26             do in parallel
27                 Load $n$ elements in $dh$
28             $Theta1[0] = atan2(-dv[Col], dh[0])$
                   ⋮
29             $Theta1[n-1] = atan2(-dv[Col], dh[n-1])$
30             do in parallel
31                 $Comp[Col][Row] = (Theta1 - Theta0)/2\pi$
32             Repeat from line 28 to 31 for $k$ elements
33     //vertical compensator
34     for $Row = 0$ to $NumOfRows - 1$ do
35         do in parallel
36             Load $n$ elements in $dh$
37         $Theta0[0] = atan2(-dv[Col], dh[Row])$
               ⋮
38         $Theta0[n-1] = atan2(-dv[Col], dh[Row + n - 1])$
39         for $Col = 0$ to $NumOfCols - 1$ do
40             do in parallel
41                 Load $n$ elements in $dv$
42             $Theta1[0] = atan2(-dv[0], dh[Row])$
                   ⋮
43             $Theta1[n-1] = atan2(-dv[n-1], dh[Row])$
44             do in parallel
45                 $Comp[Col][Row] = (Theta1 - Theta0)/2\pi$
46             Repeat from line 42 to 45 for $k$ elements
```

Algorithm 1: Parallel RC Algorithm (Continued)

4.1. **Arctangent Approximation.** As shown in Algorithm 1 the ASC function performs approximately $2N^2$ atan2(), the evaluation of the arctangent function (atan2) is commonly encountered in real-time multimedia applications. The most direct solution is based on the Taylor series. However, this series converges slowly for arguments close to one and hence is inefficient [39]. The atan2 function in the math library helps to find the trigonometric arctangent of multiple parameters. It is used to calculate the trigonometric arctangent of y/x and returns the angle in radius from the x-axis to the specified point (y,x). In terms of the standard arctan function, whose range is $(-\pi, \pi)$ rad, it can be expressed as follows:

$$atan2(y,x) = \begin{cases} arctan(y/x), & \text{if } x > 0 \\ arctan(y/x) + \pi, & \text{if } x < 0 \ \& \ y \geq 0 \\ arctan(y/x) - \pi, & \text{if } x < 0 \ \& \ y < 0 \\ \pi/2, & \text{if } x = 0 \ \& \ y > 0 \\ -\pi/2, & \text{if } x = 0 \ \& \ y < 0 \\ undefined, & \text{if } x = 0 \ \& \ y = 0 \end{cases} \tag{4.1}$$

Where the arctan function is the inverse of the tangent function, it returns the angle whose tangent is a given number, the built-in function based on the Taylor series expansion, and it takes time to calculate the angle [39]. To avoid the overhead of atan2 implementing an approximate version of the function is the appropriate solution. There is lot of mathematical atan2 approximation, implementing the following approximation with a maximum absolute error of 0.0038 rad (0.22 degree) [39]:

$$arctan(x) \approx 4\pi x + 0.273x(1 - |x|), \qquad -1 \leq x \leq 1 \tag{4.2}$$

The arctan in Eq.(4.2) applicable for angles in the range of $-\pi/4$ to $\pi/4$. , we implement the arctan and extends the angular range to $-\pi$ to π to calculate the four-quadrant using Eq.(4.1) (see [39] for more details about the mathematical approximation of arctan function).

The speedup of atan2 approximation is 1.54 over the built-in function and the overall speedup of the RC algorithm is 1.43, Figure 3 shows the speedup of the atan2 approximation over the built-in function with various sizes of images. After

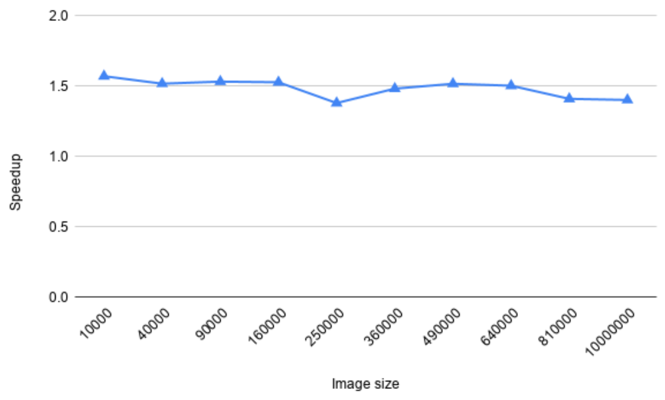


FIGURE 3. RC Speedup by Atan2 Approximation

replacing the built-in arctan function with the approximation one the accuracy of RC is decreased by an acceptable factor.

4.2. **Technical Environment.** The environment used to execute the algorithm is Unix FreeBSD 64 bit operating system, using c language, on Intel Xeon gold 6140 machine with 2.3 GHz frequency, 18 core with 36 Threads, 32 GB RAM, 32 KB level 1 cache, 1 MB level 2 cache, and 24.75 MB level 3 cache. We choose using Pthread library for multi-threading than others to get more flexibility and using SIMD instructions inside it.

5. PERFORMANCE EVALUATION OF THE RC ALGORITHM ON MULTI-CORE PROCESSORS

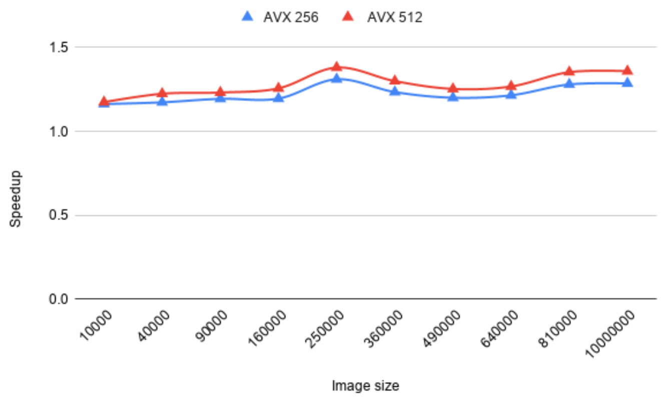
In this paper, we use AVX and AVX2 instructions to accelerate the RC algorithm. We apply multithreading technique with 2, 4, 8, 16, and 32 threads to get the benefits of the hyper-threading technology [29]. Merging SIMD with multithreading accelerate the algorithm more than using one of them separately. The use of the blocking technique also accelerates the algorithm due to the reduction of cache miss overhead and reusing the data. We embed all together to get an efficient acceleration talking the benefits of each technique.

The ASC function considers as 95.06% from the overall algorithm. However, the portion of the function that had been executed in parallel is 83.64% of the function. Therefore we can execute 79.51% of the RC algorithm in parallel. We use Amdahl's law in Eq.(3.10) to predict the possible speedup. The expected speedup of the RC algorithm according to the specifications of the machine of Intel Xeon gold 6140 should be 2.48x, 3.29x, 1.66x, 2.48x, 3.29x, 3.93x, and 4.35x for AVX256, AVX512, multithreading with 2 thread count, 4 thread, 8 thread, 16 thread, and 32 thread, respectively. Symbol x represents the original time of the RC algorithm. The ideal speedup isn't reachable due to lot of factors; simply using twice as many cores doesn't immediately generate twice the performance. there is a portion of the algorithm that doesn't execute in parallel. The overall speedup of an algorithm is always limited by its sequential part that can't be parallelized, which is 20.49% in the RC algorithm [38].

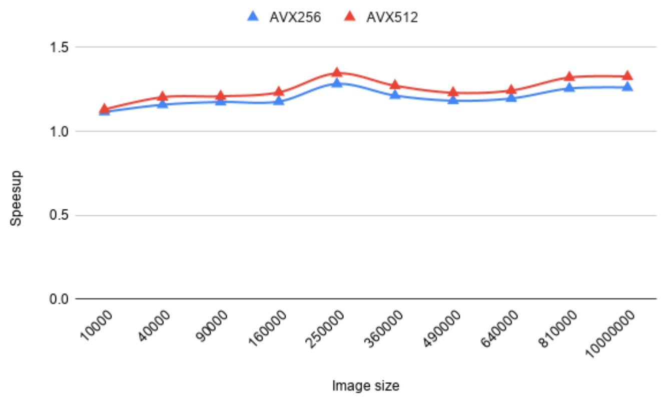
5.1. **Exploiting SIMD.** Multicore processors have many architecture capabilities that can accelerate applications. SIMD architectures have significant advantages over the other systems described by Flynn's classification scheme. SIMD exploits a significant level of data-parallelism. It benefits enterprise applications in data mining and multimedia applications, as well as the applications in computational science and engineering using linear algebra [40]. Apply and exploit SIMD on the RC phase unwrapping algorithm give speedup over sequential algorithm.

Applying AVX256 instructions, which execute 4 double-precision floating-point operations in parallel on the target algorithm yields to accelerate ASC function by 1.26, which yields to speed RC algorithm by 1.24, putting AVX256 with arctan function approximation gives a speedup of 1.8 for ASC function and 1.74 for RC. All the below results represent the parallel processing techniques speedup without adding arctan approximation speedup.

Using AVX512 which executes 8 double precision floating-point operations in parallel accelerates ASC function by 1.33 and speed RC algorithm by 1.3, as shown in Table 1. Figure 4 shows the speedup of ASC function and RC using AVX256 and AVX512.



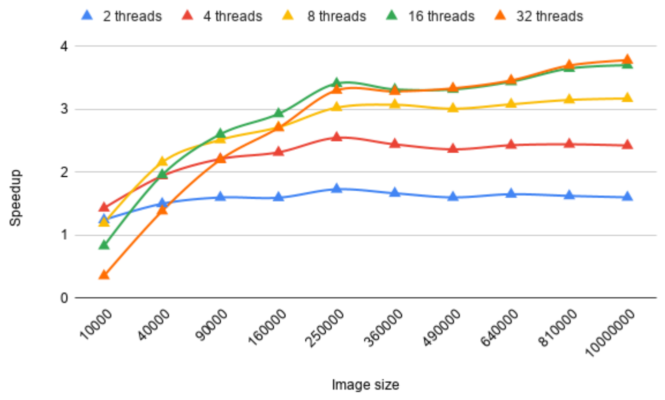
(a) ASC Function



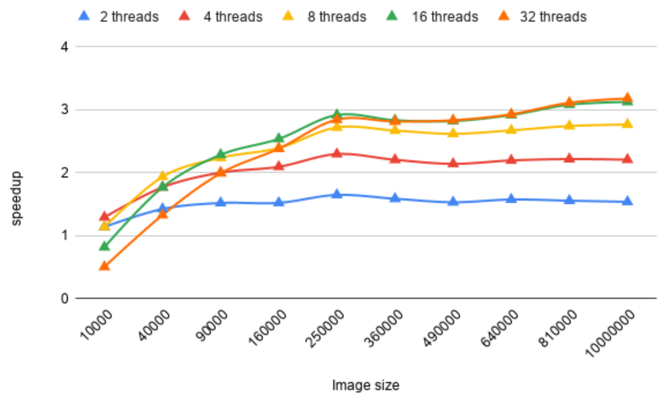
(b) RC

Figure 4. Speedup of AVX256 and AVX512

5.2. **Exploiting Multithreading.** Multithreading is a mandatory software technology for taking full advantage of the capabilities of modern computing platforms and it offers significant efficiency improvements to the RC algorithm. We use multithreading to exploit data level parallelism by dividing the same task with different data among multiple cores. As shown in Figure 5 and Table 1, creating two, four, eight, and 16 threads give a speedup close to the ideal. However, increasing the threads to 32 gives a speedup less than expected. The number of physical cores is 18, therefore in case of creating 32 give results less than ideal. Beside, the overhead of creating 32 reduce the speedup.



(a) ASC Function



(b) RC

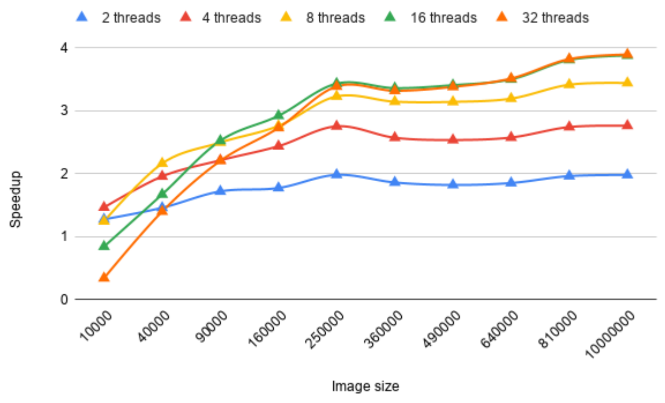
FIGURE 5. Speedup of Multithreading with Various Thread Count

5.3. **Exploiting SIMD With Multithreading.** Put AVX instructions with multithreading together exploits data level parallelism, therefore SIMD-Thrd accelerates the RC algorithm more than using each method separately. The using of SIMD-Thrd algorithm increase the Speedup of SIMD RC algorithm about 1.36 to 2.5 times for various thread count. the highest speedup was achieved by creating 32 thread. Table 1 shows ASC function and RC speedup for all SIMD with multithreading with a various thread count over the sequential one. Figure 6 Shows ASC function and RC speedup using SIMD with multithreading over the sequential.

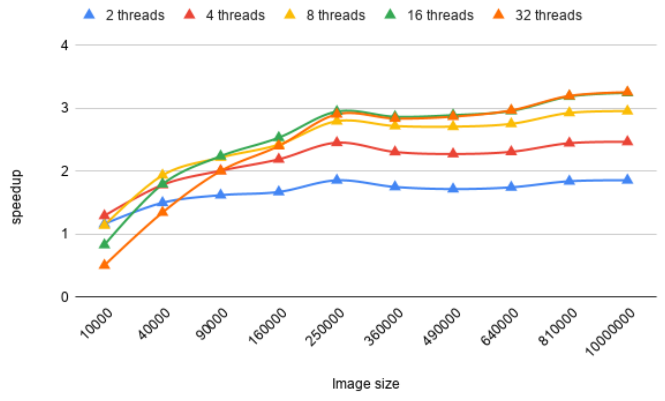
Adding more cores or parallel processing techniques will speed RC algorithm with a small speedup because we can't speed this algorithm more than the maximum speedup which is 4.88. Applying parallel processing techniques on an Intel Core i5

	function speedup	**RC speedup**
AVX256	1.26	1.24
AVX512	1.33	1.30
Thrd2	1.62	1.55
Thrd4	2.43	2.20
Thrd8	3.12	2.72
Thrd16	3.56	3.02
Thrd32	3.59	3.04
SIMD-Thrd2	1.93	1.62
SIMD-Thrd4	2.69	2.41
SIMD-Thrd8	3.33	2.87
SIMD-Thrd16	3.65	3.09
SIMD-Thrd32	3.69	3.11
Blocking	1.21	1.22
SIMD-Block	1.29	1.26

Table 1. ASC Function and RC Speedup



(a) ASC Function

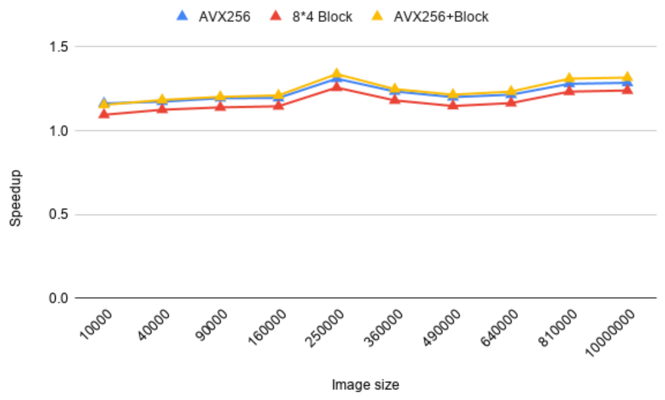


(b) RC

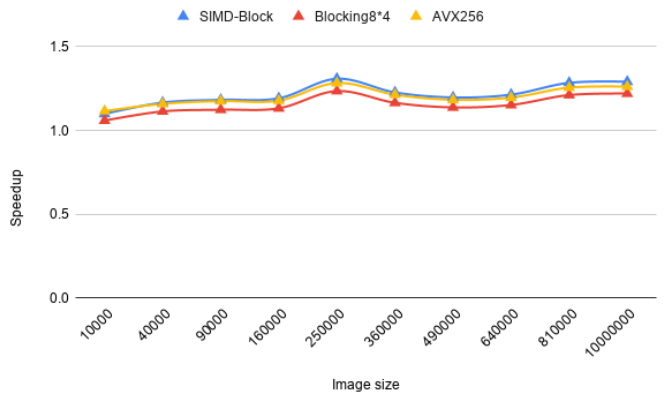
Figure 6. Speedup of SIMD-Thread with Various Thread Count

with two cores machine, the RC speedup when using Approximation of arctan function was 1.54, SIMD-Thrd with four threads speedup was 1.67 Putting them together we get a maximum speedup of 2.57. By using the same machine with windows 10 operating system and Cygwin compiler we get a speedup of 1.25 for SIMD-Thrd with four threads.

5.4. **Exploiting Blocking.** The increasing gap between memory latency and processor speed is a critical bottleneck in achieving high performance. Exploiting memory hierarchy by blocking is one of the most critical techniques that can hide memory latency, Applying blocking technique with various block size reduces the overhead of memory and accelerates the RC algorithm over the original one, we try block sizes of 4*4, 8*8 , 8*4, and 16*4. The speedup of each block size are 1.19, 1.21, 1.21, 1.22 , respectively. According to the results of each block size, we chose the 8*4 block size that gives a suitable speedup and apply SIMD technique together with it. Figure 7 shows the speedup of ASC function and RC using SIMD, blocking and SIMD with blocking.



(a) ASC Function



(b) RC

FIGURE 7. Speedup of SIMD, Blocking, and SIMD-Blocking

As shown in Figure 7 the SIMD with blocking gives a speedup greater than each method alone.

6. CONCLUSION

This paper improves the speedup of the RC algorithm which is a mathematical problem-solving technique by using SIMD instructions inside multiple threads to exploit data level parallelism by partitioning data among cores and perform the same operations on it. Furthermore, implementing an approximation of the arctan function yields to appropriate speedup. The speedup of the parallel RC algorithm is 4.36 times over the sequential algorithm, this speedup is closed to the ideal which is 4.88. The parallel RC algorithm balances between high accuracy and high speedup which are important factors in image processing applications. In the future work we will increase the accuracy of the parallel algorithm to equal the original algorithm and implement the parallel RC algorithm as a general library to be used in various phase unwrapping applications.

ACKNOWLEDGMENT

We thank Prof. Satoshi Tomioka (Graduate School of Engineering, Hokkaido University) for providing us the source code of the RC algorithm.

REFERENCES

[1] Tomioka, S., Heshmat, S., Miyamoto, N., & Nishiyama, S. (2010). Phase unwrapping for noisy phase maps using rotational compensator with virtual singular points. Applied Optics, 49(25), 4735. DOI: 10.136/ao.49.004735

[2] Aiello, L., Riccio, D., Ferraro, P., Grilli, S., Sansone, L., & Coppola, G. et al. (2007). Green's formulation for robust phase unwrapping in digital holography. Optics And Lasers In Engineering, 45(6), 750-755. DOI: 10.1016/j.optlaseng.2006.10.002

[3] Heshmat, S., Tomioka, S., and Nishiyama, S., "Rotational and direct compensation for digital hologram phase unwrapping," Proc. SPIE 8413, Speckle 2012: V International Conference on Speckle Metrology, 84130S (2012).

[4] Heshmat, S., Tomioka, S., & Nishiyama, S. (2012). Phase unwrapping algorithm based on singularity compensation for three-dimensional shape measurement. Optical Review, 19(6), 444-450. DOI: 10.1007/s10043-012-0076-9

[5] Heshmat, S., Tomioka, S., & Nishiyama, S. (2013). Phase extraction and unwrapping using rotational and direct compensators for the digital hologram. Optical Engineering, 52(10), 101910. DOI: 10.1117/1.oe.52.10.101910

[6] Heshmat, S., Tomioka, S., & Nishiyama, S. (2014). Performance Evaluation of Phase Unwrapping Algorithms for Noisy Phase Measurements. International Journal Of Opto mechatronics, 8(4), 260-274. DOI: 10.1080/15599612.2014.942927

[7] Goldstein, R., Zebker, H., & Werner, C. (1988). Satellite radar interferometry: Two-dimensional phase unwrapping. Radio Science, 23(4), 713-720. DOI: 10.1029/rs023i004p00713

[8] Huntley, J. M., “Noise-immune phase unwrapping algorithm,” J. Appl. Opt. 28, 3268 – 3270 (1989).

[9] Cusack, R., Huntley, J. M., and Goldrein, H. T., “Improved noise immune phase-unwrapping algorithm,” J. Appl. Opt. 34, 781 – 789 (1995).

[10] Flynn, T. J. (1997). Two-dimensional phase unwrapping with minimum weighted discontinuity. JOSA A, 14(10), 2692-2701.

[11] Karout, S. A., Gdeisat, M. A., Burton, D. R., & Lalor, M. J. (2007). Two-dimensional phase unwrapping using a hybrid genetic algorithm. Applied Optics, 46(5), 730-743.

[12] Itoh, K. (1982). Analysis of the phase unwrapping algorithm. Applied optics, 21(14), 2470-2470.

[13] Judge, T. R., & Bryanston-Cross, P. J. (1994). A review of phase unwrapping techniques in fringe analysis. Optics and Lasers in Engineering, 21(4), 199-239.

[14] Buckland, J. R., Huntley, J. M., & Turner, S. R. E. (1995). Unwrapping noisy phase maps by use of a minimum-cost-matching algorithm. Applied Optics, 34(23), 5100-5108.

[15] Cusack, R., Huntley, J. M., & Goldrein, H. T. (1995). Improved noise-immune phase-unwrapping algorithm. Applied Optics, 34(5), 781-789.

[16] Hunt, B. R. (1979). Matrix formulation of the reconstruction of phase values from phase differences. JOSA, 69(3), 393-399.

[17] Takajo, H., & Takahashi, T. (1988). Least-squares phase estimation from the phase difference. JOSA A, 5(3), 416-425.

[18] Ghiglia, D. C., & Romero, L. A. (1989). Direct phase estimation from phase differences using fast elliptic partial differential equation solvers. Optics letters, 14(20), 1107-1109.

[19] Ghiglia, D. C., & Romero, L. A. (1994). Robust two-dimensional weighted and unweighted phase unwrapping that uses fast transforms and iterative methods. JOSA A, 11(1), 107-117.

[20] Rouse, Margaret (March 27, 2007). "Definition: multi-core processor". TechTarget. Archived from the original on August 5, 2010. Retrieved March 6, 2013.

[21] Barney, B. (2010). Introduction to parallel computing. Lawrence Livermore National Laboratory, 6(13), 10.

[22] S. Lantz, K. McDermott, M. Reid, D. Riley, P. Wittich, S. Berkman, et al., "Speeding up Particle Track Reconstruction using a Parallel Kalman Filter Algorithm," 2020.

[23] E. Limonova, A. Terekhin, D. Nikolaev, and V. Arlazarov, "Fast implementation of morphological filtering using ARM NEON extension," 2020.

[24] S. Berkman, G. Cerati, B. Gravelle, B. Norris, A. R. Hall, and M. Wang, "Reconstruction for Liquid Argon TPC Neutrino Detectors Using Parallel Architectures," 2020.

[25] D. Jodlbauer, U. Langer, and T. Wick, "Parallel matrix-free higher-order finite element solvers for phase-field fracture problems," 2020.

[26] Patterson, D. A., & Hennessy, J. L. (2012). Computer organization and design: the hardware/software interface.

[27] Cockshott, P., & Renfrew, K. (2013). SIMD programming manual for Linux and Windows. Springer Science & Business Media.

[28] Conte, G.; Tommesani, S.; Zanichelli, F. (2000). "The long and winding road to high-performance image processing with MMX/SSE". Proc. Fifth IEEE Int’l Workshop on Computer Architectures for Machine Perception.

[29] Intel Hyper-Threading Technology, Technical User’s Guide.

[30] Park, N., Hong, B., & Prasanna, V. K. (2003). Tiling, block data layout, and memory hierarchy performance. IEEE Transactions on Parallel and Distributed Systems, 14(7), 640-654.

[31] Ghiglia, D. C., & Pritt, M. D. (1998). Two-dimensional phase unwrapping: theory, algorithms, and software. Wiely-Interscience, first ed.(April 1998).

[32] López-Ocaña, A., Cruz-Santos, W., García-Arellano, A., & Rueda-Paz, J. A partition strategy to speedup Goldstein's phase unwrapping algorithm on a multi-core architecture.

[33] Huang, Q., Zhou, H., Dong, S., & Xu, S. (2015). Parallel branch-cut algorithm based on simulated annealing for large-scale phase unwrapping. IEEE Transactions on Geoscience and Remote Sensing, 53(7), 3833-3846.

[34] Barabadi, B., Gara, M., Jooya, A., Baniasadi, A., & Dimopoulos, N. (2019, October). Dual-Stage Phase Unwrapping. In 2019 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC) (pp. 1-7). IEEE.

[35] Karasev, P. A., Campbell, D. P., & Richards, M. A. (2007, April). Obtaining a 35x speedup in 2d phase unwrapping using commodity graphics processors. In 2007 IEEE Radar Conference (pp. 574-578). IEEE.

[36] Mistry, P., Braganza, S., Kaeli, D.& Leeser, M. (2009, March). Accelerating phase unwrapping and affine transformations for optical quadrature microscopy using CUDA. In Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units (pp. 28-37).

[37] Wu, Z., Ma, W., Long, G., Li, Y., Tang, Q., & Wang, Z. (2014, May). High performance two-dimensional phase unwrapping on GPUs. In Proceedings of the 11th ACM Conference on Computing Frontiers (pp. 1-10).

[38] Rodgers, D. P. (1985). Improvements in multiprocessor system design. ACM SIGARCH Computer Architecture News, 13(3), 225-231.

[39] Rajan, S., Wang, S., Inkol, R., & Joyal, A. (2012). Efficient Approximations for the Arctangent Function. Streamlining Digital Signal Processing, 6, 265.

[40] Marinescu, D. C. (2017). Cloud computing: theory and practice. Morgan Kaufmann.