論文 J. of The Korean Society for Aeronautical and Space Sciences 46(6), 503-512(2018)
DOI:https://doi.org/10.5139/JKSAS.2018.46.6.503
ISSN 1225-1348(print), 2287-6871(online)

고해상도 SAR 영상처리 고속화를 위한 병렬 성능 최적화 기법 연구

이규범*, 김규빈*, 안솔보름*, 조진연*, 임병균**, 김동현**, 김정호***

A Study on Parallel Performance Optimization Method for Acceleration of High Resolution SAR Image Processing

Kyu Beom Lee*, Gyu Bin Kim*, Sol Bo Reum An*, Jin Yeon Cho*,
Byoung-Gyun Lim**, Dong-Hyun Kim** and Jeong Ho Kim***
Department of Aerospace Engineering, Inha University*,***
Korea Aerospace Research Institute**

ABSTRACT

SAR(Synthetic Aperture Radar) is a technology to acquire images by processing signals obtained from radar, and there is an increasing demand for utilization of high-resolution SAR images. In this paper, for high-speed processing of high-resolution SAR image data, a study for SAR image processing algorithms to achieve optimal performance in multi-core based computer architecture is performed. The performance deterioration due to a large amount of input/output data for high resolution images is reduced by maximizing the memory utilization, and the parallelization ratio of the code is increased by using dynamic scheduling and nested parallelism of OpenMP. As a result, not only the total computation time is reduced, but also the upper bound of parallel performance is increased and the actual parallel performance on a multi-core system with 10 cores is improved by more than 8 times. The result of this study is expected to be used effectively in the development of high-resolution SAR image processing software for multi-core systems with large memory.

초 록

SAR(Synthetic Aperture Radar)는 레이더를 이용하여 얻은 신호를 처리해 영상을 획득하는 기술로서, SAR 영상의 활용도와 고해상도 영상에 대한 요구가 증가하고 있는 상황이다. 따라서 본 연구에서는 고해상도 영상 데이터의 고속 처리를 위해 SAR 영상처리 알고리즘을 다중코어 기반의 컴퓨터 구조에서 최적의 성능을 낼 수 있도록 구현하기 위한 연구를 수행했다. 고해상도 영상에 따른 방대한 양의 입출력에 의한 성능 저하를 개선시키기 위해 메모리를 최대한 활용하는 성능 최적화 기법을 적용하고 OpenMP의 동적 스케쥴링 기법과 중첩 병렬성(nested parallelism)을 사용해 코드의 병렬화 비율을 높였다. 그 결과 전체 계산 시간을 줄일 뿐만 아니라 병렬 성능의 최대 한계치를 크게 높일 수 있었으며, 제안된 기법을 10개 코어를 가진 다중코어 시스템에 적용한 결과 기존 대비 8배 이상의 성능 향상이 있었다. 본 연구 결과는 대용량 메모리를 가진 다중코어 시스템을 대상으로 하는 고해상도 SAR 영상처리 소프트웨어 개발에 효과적으로 활용될 수 있을 것으로 기대된다.

† Received : January 25, 2018 Revised : March 19, 2018 Accepted : May 15, 2018
*** Corresponding author, E-mail : JeongHoKim@inha.ac.kr

Key Words : Synthetic Aperture Radar(합성개구레이더), Image Processing(영상처리), Parallel Programing(병렬 프로그래밍), Code Optimization(코드 최적화)

Ⅰ. 서 론

다목적실용위성 5호에 국내 최초로 탑재된 전천후 영상레이더인 SAR(Synthetic Aperture Radar)는 마이크로파를 지상 목표물에 방사한 후 반사되어 돌아온 신호를 합성하여 영상을 만드는 장치로서 다양한 플랫폼에 탑재되어 원하는 관심 지역을 주·야간 및 악천후에도 전천후로 상시 관측 및 정찰을 할 수 있다. 특히 최근 반도체 및 신호처리 기술의 비약적인 발전으로 1m 이내인 고해상도의 SAR 영상 생성이 가능해졌고, 표적 영상 형성과 표적 식별 기술이 발전함에 따라 군사 목적으로 국경 감시나 군사 시설 및 테러 위협에 대한 필수적인 감시 정찰 수단으로 활용도가 높아지고 있다[1].

이러한 SAR를 이용하여 영상을 생성하기 위해서는 SAR에서 수집된 데이터로부터 복잡한 처리 절차 및 FFT 기법 등과 같은 수치 알고리즘이 적용되며 이로 인해 많은 계산시간이 소요된다. 특히 고해상도급 SAR의 경우는 처리해야 하는 데이터가 수십 GB단위로 급속히 증가하기 때문에 영상처리를 위해 소요되는 시간이 상당히 길어지게 되고 위성에서 전송된 방대한 양의 데이터를 짧은 시간 내에 영상으로 확인하는 것이 불가능하게 되어 그만큼 영상자료의 효용성이 떨어지게 된다.

SAR 영상자료가 효용성을 얻기 위해서는 참고문헌[2-4]와 같이 효과적인 영상처리 알고리즘 개발이 필요하기도 하지만, 방대한 양의 계산을 요구하는 고해상도 SAR 영상처리 기술의 특성상 컴퓨터의 성능을 최대한으로 활용해 빠른 처리 능력을 갖는 소프트웨어 개발도 필수적이다. 특히 2005년 전후로 해서 하드웨어 개발사들은 클럭 속도나 주파수를 향상시키기 보다는 집적도를 높여 하나의 통합기판 위에 많은 프로세서를 배치함으로써 시스템의 전체 작업처리 능력을 향상시키는 쪽으로 프로세서를 개발해 왔다. 그래서 병렬처리 기능이 없는 순차적(serial)인 소프트웨어나 효과적으로 병렬화 되지 못한 소프트웨어는 병렬 처리 기반으로 개발되어온 최신 하드웨어의 성능 상의 혜택을 효과적으로 반영 할 수 없어 하드웨어의 성능발전에 관계없이 소프트웨어 성능은 정체 될 수밖에 없다.

따라서 컴퓨터공학 분야에서는 병렬처리가 주목을 받고 있으며, 최근 병렬 계산에 효과적이라고 알려져 있는 고성능 GPU(Graphics Processing Unit)와 같은 극단적인 병렬처리 기반의 고성능 하드웨어 가속기를 다중코어로 구성된 CPU와 함께 활용하여 고속으로 영상 처리를 수행하려는 연구가 국외[5-7]와 국내[8]에서 있었다. 하지만 수십 GB의 방대한 양의 고해상도 SAR 영상 데이터 때문에 계산을 어느 정도 병렬화 하더라도 컴퓨터 구조를 고려한 성능 최적화가 제대로 이루어지지 않고 구현 가능한 병렬성을 최대한으로 끌어내지 못한다면, 이러한 최신의 고성능 하드웨어를 사용하더라도 준실시간의 고성능 영상처리는 불가능하다.

따라서 본 논문에서는 영상자료의 효용성과 업무의 효율성을 높일 수 있도록, 병렬성이 높은 SAR 영상처리 알고리즘을 탑재하고 있는 소프트웨어에 최신 컴퓨터 구조를 고려한 병렬 최적화 기법을 적용하는 연구를 수행한다. 이 연구를 통해 준실시간으로 고해상도 SAR 영상처리를 위한 고성능 소프트웨어 개발 시에 활용 될 수 있는 성능 최적화 기법 및 병렬 프로그래밍 기법의 기틀을 마련하고자 한다.

Ⅱ. 본 론

2.1 SAR 영상처리 알고리즘

SAR 신호 계측은 기본적으로 SAR가 장착된 항공기나 위성이 이동하면서 이루어지며, 시간에 따라 계측된 SAR 영상 데이터를 처리할 때에는 계산의 효율을 위해 FFT를 사용하여 주파수 영역으로 바꾸어 계산한다. 이때 SAR가 탑재된 플랫폼이 이동하기 때문에 만일 적분 시간이 길면, 상대 운동에 의해 시간에 따라 다른 도플러 주파수가 달라지고 거리 단위로 수신되는 반사 신호의 거리 위치가 이동하게 된다. 이를 range cell migration(RCM) 현상이라고 하며 고해상도 영상처리에서는 이를 반드시 고려해야 한다. 그리고 레이다 신호에 대한 해상도를 높이기 위해 신호 압축을 수행한다. 이러한 여러 가지 신호 처리 작업은 여러 단계를 나누어 수행되며, 각각 단계에서 많은 연산이 필요하다.

SAR 영상 처리를 위해 최초로 고안된 알고리즘은 range Doppler algorithm(RDA)인데, 이 알

고리즘은 간단한 것에 비해 정확도가 높아 효과적이다. 그러나 상대 운동 때문에 발생하는 오차를 줄이기 위해 수행되는 range cell migration correction(RCMC) 작업과 해상도를 높이기 위해 수행되는 secondary range compression(SRC) 작업 과정에서 효율성이 떨어지게 된다는 단점이 있다[9]. 이를 보완하기 위해 chirp scaling algorithm(CSA)이 개발되었는데, 여기서 Chirp라는 것은 시간에 따라 주파수가 빨라지거나 느려지는 신호를 의미한다. 이 알고리즘은 chirp scaling 원리[10]를 통해 RCMC를 위해 시간 영역에서 보간 하지 않고 위상(phase)을 곱하는 것만으로 효과적으로 처리 할 수 있다. 또한 SRC가 azimuth 주파수만으로 처리될 수 있어 range 방향 데이터와 azimuth 방향 데이터가 서로 독립적이 된다. 바로 이점이 SAR 영상의 병렬 처리를 가능하게 한다. 개발된 소프트웨어에서 대상으로 하는 알고리즘은 CSA에서 RCM이 정확한 신호라 가정하고 azimuth 신호 처리에 초점을 맞추어 baseband azimuth scaling(BAS)를 도입하여 획득한 데이터를 azimuth 블록으로 나눈다. 이 블록을 subaperture라 부르며, SAR 안테나가 조정될 때 총 azimuth 신호 대역폭이 여러 펄스 반복 주파수 간격에 걸쳐져 샘플링이 부족할 수 있다는 문제를 해결할 수 있는 대안이 된다[2]. 이렇게 나뉜 subaperture에서 영상처리를 수행하며, 모든 처리 후에 하나의 이미지를 생성하기 위해 subaperture를 조립할 때 각 subaperture들이 부드럽게 연결되도록 subaperture의 경계가 약간 겹치게 구성된다.

SAR의 작동 방식에는 여러 가지 모드가 있는데 Fig. 1에서 볼 수 있는 Terrain Observation by Progressive Scans(TOPS)는 센서의 뒤편에

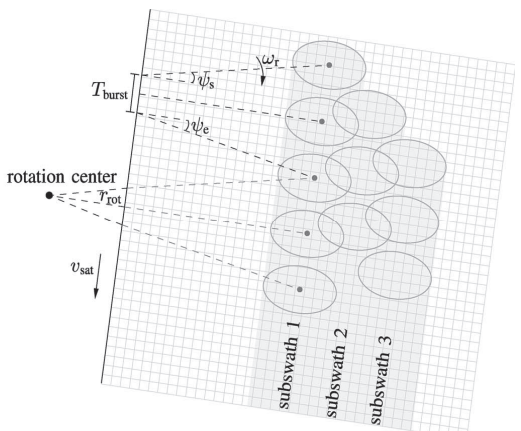

Fig. 1. TOPS mode[2]

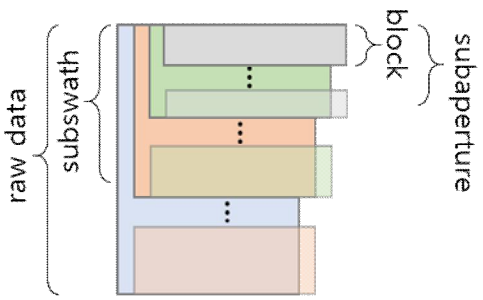

Fig. 2. Data composition

회전 중심을 두고 회전하면서 계측하며, 여러 subswath에 대하여 샘플링하게 된다. 이에 따라 많은 양의 원시 데이터를 생성하게 되고 엄청난 연산 시간이 필요하게 되므로 본 연구에서는 TOPS 모드로 생성되는 원시 데이터를 연구 대상으로 결정했다.

개발된 소프트웨어는 상당히 큰 SAR 원시 데이터를 메모리에 모두 올리지 않고 swath에 따라 읽고 처리하게 된다. 그리고 subaperture에서 처리 시에는 병렬 작업 개수에 따라 블록으로 나누어 계산하게 된다. 이를 도식화하면 Fig. 2와 같다. 만일 하나의 subaperture의 데이터가 M×N 개이고 스레드 개수가 P일 경우에 블록의 크기는 [M/P]×N개가 된다. [k]는 k보다 크고 가장 가까운 정수를 의미한다.

2.2 컴퓨터 구조와 병렬 처리

2.2.1 컴퓨터 구조와 성능

앞서 언급한대로 근래의 컴퓨터 하드웨어의 발전 방향이 클럭 속도의 향상 보다는 집적도의 향상을 통한 하드웨어의 병렬성을 향상시키는 쪽으로 발전하게 됨에 따라 성능 향상을 위한 가장 중요한 방법은 병렬 처리의 효율을 최대한 높이는 것이다.

병렬 처리 시에는 전체 작업량을 얼마만큼 병렬로 처리하는지가 중요하다. 어떤 프로그램이든 순차적으로 실행되는 부분은 존재 할 수밖에 없는데, 전체 프로그램 실행 시간 대비 순차적으로 실행되는 시간 비율을 통해 병렬 처리 시 얻을 수 있는 성능 이득 한계를 다음의 식을 통해 이론적으로 계산할 수 있으며 이를 Amdahl의 법칙[11]이라 한다.

$$S(N)=\frac{1}{(1-p)+p/N} \quad (1)$$

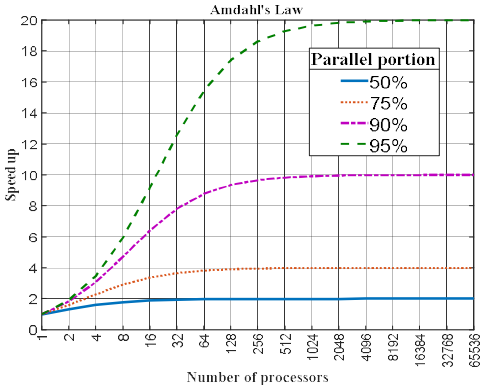

Fig. 3. Amdahl's law

위 식에서 S는 프로그램 실행 시 이론적인 최대 병렬 성능향상률(speedup)이며 p는 코드 상에서 병렬화를 할 수 있는 작업량의 비율이고, N은 시스템의 CPU 코어 개수이다. 만일 전체 코드를 병렬화 할 수 있다면 프로세서의 개수에 비례하여 성능이 향상될 것이다. 하지만 코드 상에서 순차적인 부분이 0이 될 수는 없기 때문에 병렬 성능 이득에는 한계가 필연적으로 발생한다. Fig. 3은 위 식을 그래프로 도시한 것인데 순차적인 부분의 존재로 인해 CPU가 무한히 많더라도 병렬 성능 이득에는 한계가 발생하는 것을 확인 할 수 있다. 따라서 코드를 분석 후 병렬화 비율을 높이는 것이 병렬 처리에 대한 이득을 높이는 가장 효과적인 방법이다. 그러므로 본 연구에서는 순차적인 부분의 비율을 줄이고 병렬 계산의 비율을 높임으로써 전체 병렬 성능을 향상시키고자 하였다.

프로그램의 성능 향상을 위해서는 병렬 처리 외에 개별 프로세서를 활용한 계산의 효율을 높이는 것 또한 고려해야 한다. 현재의 컴퓨터 시스템은 프로세서의 처리 성능에 비해 상대적으로 느린 메모리의 데이터 전송속도로 인한 병목 현상을 극복하기 위해 Fig. 4와 같은 다층 메모리 구조를 채택하고 있다. 이러한 구조는 프로세서에서 다음 연산에 필요로 하는 데이터가 메모리 구조의 상위에 있을수록 더 빠른 연산이 가능하다는 것을 의미한다. 따라서 본 연구에서는 하드디스크보다는 메인메모리를, 그리고 메인메모리보다는 캐시 메모리를 더 많이 활용할 수 있도록 코드를 수정함으로써 성능 향상을 이루고자 하였다.

2.2.2 병렬 처리의 분류

병렬 처리는 어플리케이션의 관점에서는 데이터 병렬 처리(data parallelism), 작업 병렬 처리(task parallelism)로 나눌 수 있다[13].

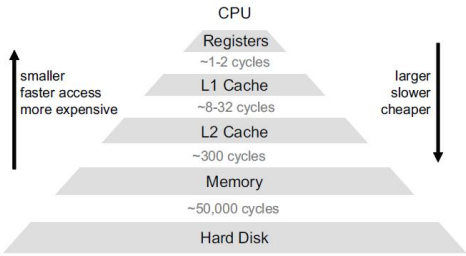

Fig. 4. Memory hierarchy[12]

데이터 병렬 처리는 많은 양의 데이터를 여러 프로세서에서 나누어 동시에 같은 연산을 처리하는 것을 의미한다. 이는 후술할 벡터 연산과 OpenMP를 통한 루프문의 병렬화를 통해 이루어질 수 있다. 같은 연산을 수행하기 때문에 작업량은 모든 프로세서에 거의 같게 배분되며 많은 양의 데이터를 처리하기 때문에 데이터의 지역성을 통한 캐시메모리 최적화가 특히 중요하다.

작업 병렬 처리는 병렬 컴퓨팅 환경에서 다중 프로세서에 작업을 기준으로 병렬화가 된 형태를 의미한다. 데이터 병렬 처리와는 달리 프로세서가 각자 다른 작업을 수행할 수도 있으며 따라서 프로세서마다 연산량이 다를 수 있어 하나의 프로세서의 작업이 끝나고 다른 프로세서의 작업이 끝날 때까지 기다리지 않도록 작업량을 효율적으로 분배하는 것이 중요하다. 이를 위해 동적 스케줄링(dynamic scheduling) 기법을 이용할 수 있다.

2.2.3 OpenMP

OpenMP[14]는 컴파일러 지시자 형태로 제공되는 병렬 프로그래밍 도구로 C, C++, Fortran 언어에서 사용할 수 있는 표준화된 API이다. 표준화에 의해 멀티 플랫폼을 지원한다는 것이 큰 장점이고 가장 널리 사용되고 있는 공유메모리 병렬 프로그래밍 API로서 단일 시스템의 메모리를 공유하는 다중 코어를 가진 시스템에서 쉽게 병렬 프로그램을 구현할 수 있도록 해준다. OpenMP는 표준 병렬 프로그래밍 API로서 대부분의 컴파일러에서 지원이 된다.

본 연구에서 사용하는 OpenMP의 프로그래밍 모델은 Fork-Join모델이라고 하는 것으로 Fig. 5와 같이 나타낼 수 있다.

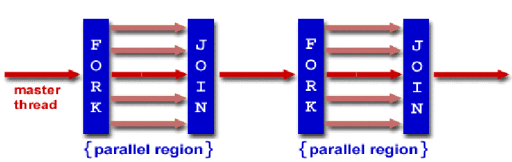

Fig. 5. Multi-threading using fork-join model

마스터 스레드가 실행을 하다가 for 루프 등의 병렬 처리가 가능한 부분을 만나게 되면 여러 개의 멀티 스레드로 나뉘어져 실행을 하게 되고, 루프가 끝나면 다시 마스터 스레드로 합쳐지게 된다. 이러한 형태를 전체 프로그램에 대해서 반복하는 구조를 갖는다. OpenMP의 전형적인 사용방법을 보면 이는 보통 루프를 병렬 처리하기 위해서 사용하는데, 먼저 가장 시간이 오래 걸리는 루프를 찾아 그 부분을 여러 스레드로 나누어 실행을 하게 한다. 일반적인 순차적 프로그램에 OpenMP 디렉티브 부분만을 추가하면 컴파일러가 병렬 처리가 가능한 실행 파일을 생성하게 된다.

2.3 SAR 영상처리 순차성능 최적화

2.3.1 I/O 최적화

고해상도 SAR 영상처리는 특성상 방대한 양의 데이터를 다루기 때문에 I/O(Input/Output)에 따라 소프트웨어 성능이 좌우되는 I/O bound가 있을 가능성이 매우 크며 이에 대한 최적화가 필수적이다.

이러한 데이터 I/O 부분은 일반적으로 순차적인 실행 부분으로서 Amdahl의 법칙에 따라 병렬 성능을 급격히 저하시킨다. 따라서 이를 방지하기 위해서 디스크를 통한 데이터 입출력을 최소화하고 가능한 데이터를 모두 한 번에 메모리(RAM)에 올려 계산하는 것이 가장 이상적이며, 만일 메모리의 용량이 부족해 입력 데이터를 한 번에 모두 메모리에 저장하지 못한다면 여러 번 나누어 읽더라도 디스크와 메모리에서 순차적인 데이터 전송으로 인한 대기시간은 최소화해야 한다.

이를 위해 본 연구에서는 먼저 입력 데이터 전처리 단계에서의 파일을 통한 여러 단계의 데이터 변환 과정을 프로그램의 큰 수정 없이 메모리를 최대한 활용해서 수행할 수 있도록 파일 캐시(cache)의 개념을 구현하였다. 캐시는 본래 자주 쓰는 데이터를 저장하는 접근이 빠른 저장소라는 의미인데, Fig. 6과 같은 알고리즘을 적용하여 메모리를 디스크에 대한 캐시로 사용함으로써 파일 입출력 시의 디스크 I/O를 상당 부분 줄일 수 있었다.

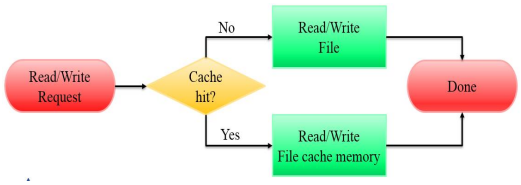

Fig. 6. Block diagram of file cache

Table 1. Pseudo-code of program

```
1  Program SAR image processing
2  For i=1 to the number of subswaths
3    HDF5read() : read raw data
4    Make sub-apertures
5    For j=1 to the number of subapertures
6      TakeSubapertureBreakIntoBlockFiles()
7        : Split sub-apertures into blocks
8      AzimuthFFTUsingThreads()
9        : Azimuth FFT
10     RangeCompressionUsingThreads()
11       : Range compression
12     AzimuthCompressionUsingThreads()
13       : Azimuth compression
14     RaBlockingOfSubapBlockFiles()
15       : Range blocking for recombination
16   Next j
17   SubapCombDerotACUsingThreads()
18     : Recombination of sub-apertures
19   Print Subswath data
20 Next i
21 Combine Subwaths data
22 End Program
```

이와 더불어 Table 1의 3~4줄의 전처리 작업 후 Table 1의 5번에서 시작하는 안쪽 루프내에서의 디스크 I/O 최소화를 위해 Fig. 7과 같이 SAR 영상 처리 시 매 단계가 끝나고 디스크에 결과 데이터를 쓰고 다음 단계에서 그 데이터를 읽고 처리하는 방식을 메모리 용량이 충분한 경우 Fig. 8과 같이 데이터를 메모리에 상주시키고 계산하는 방식으로 수정하였다.

그림에서 CBFM은 전체 영상처리 데이터를 메모리 용량에 맞춰서 블록 단위로 분리해서 관리하는 클래스로서 현재 처리 중인 블록에 해당하는 배열 데이터를 멤버 변수로 가지고 있으며,

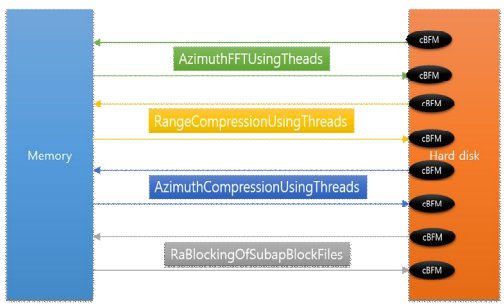

Fig. 7. Data transfer diagram

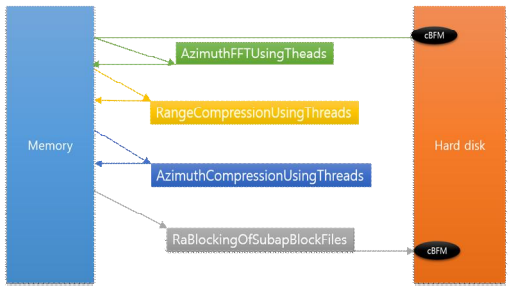

Fig. 8. Modified data transfer diagram

메모리로 처리 가능한 용량을 넘어가는 대용량 영상처리 데이터도 처리할 수 있도록 하기 위한 것이다. 그런데 Fig. 7의 방식에서는 전체 영상처리 데이터가 하나의 블록으로 메모리상에서 처리가 가능한 경우에도 여러 개의 블록을 사용해야 할 때와 마찬가지로 매 단계마다 디스크 I/O를 거치게 되어 있다. 따라서 메모리가 충분한 경우에도 이로 인해 상당한 성능저하가 발생하게 되는데 반해 Fig. 8의 방식에서는 메모리 용량이 충분해서 하나의 블록으로 처리 가능한 경우에는 다음 단계에서 이미 메모리에 로드된 블록과 현재 처리하고자 하는 블록과 같게 되므로 디스크 I/O를 거치지 않고 메모리에 있는 데이터를 바로 사용할 수 있게 된다.

따라서 이상과 같은 I/O 최적화 과정을 통해 전체 계산시간 중 순차적인 실행이 필요한 부분의 시간을 대폭 줄일 수 있게 되고, 이에 따라 Table 2에서 확인할 수 있듯이 순차 수행시간이 대폭 감소했을 뿐 아니라 가용 스레드를 최대로 사용한 병렬 수행시간은 그보다 훨씬 더 큰 비율로 감소하였다. 여기서 'Org'는 수정 전 코드를 의미하며 'Mod1'은 I/O 최적화된 코드를 의미한다. Table 3은 성능 비교에 사용된 컴퓨터의 사양을 나타낸다.

Table 2. Performance analysis

# threads	Org	Mod1	Performance Ratio
1	7,536 sec	4,830 sec	1.56
10	7,104 sec	1,200 sec	5.92
Parallel Speedup	1.06	4.02	-

Table 3. System specification

CPU	Intel(R) i7-6950X 10 physical cores
RAM	128GB

2.3.2 CPU 최적화

최근의 CPU 구조는 벡터 연산을 지원하도록 구성되어 있으며, 이를 활용하면 Fig. 9에 도시된 것처럼 한 번의 싸이클에 여러 개의 데이터에 대한 계산을 할 수 있게 하는 SIMD(Single Instruction Multiple Data) 연산이 가능하다. 따라서 이 SIMD 연산을 이용하면 같은 양의 벡터 데이터를 처리하는데 필요한 싸이클 수가 줄어들게 되어 CPU 시간을 감소시킬 수 있다. 이 SIMD연산은 최신 하드웨어에서 이론적으로 Single precision의 경우에는 64배, Double precision의 경우에는 32배까지 성능이 향상될 수 있다[15]. 따라서 이 SIMD 연산을 효과적으로 사용하면 소프트웨어의 성능이 향상된다.

SIMD 연산을 사용하기 위해서는 루프가 효율적으로 벡터화 되어야한다. 루프의 벡터화는 일반적으로 컴파일러가 판단해서 자동으로 SIMD 연산을 사용할 수 있도록 벡터화를 해주지만 다음과 같은 경우 프로그래머가 판단하기에는 벡터화 되어야 하는 루프임에도 컴파일러가 의존성이 있다고 판단하여 벡터화를 하지 않을 수도 있다.

1. Class의 멤버 변수를 루프문 내에서 값을 수정한 경우
2. for문의 반복 조건에 대한 변수가 상수가 아닌 경우
3. for문 중간에 break이 있는 경우

이와 같은 이유로 컴파일러에 의해 벡터화가 되지 않은 루프문에 대해 프로그래머가 의존성이 없어 벡터화가 가능하다고 판단하는 경우, OpenMP에서 지원하는 '#pragma omp simd'와 같은 지시자를 사용하여 강제적으로 컴파일러가 루프를 벡터화 하도록 할 수 있다.

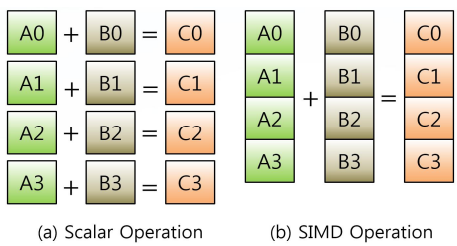

Fig. 9. SIMD parallelism

Table 4. Results of main loop vectorization

Location	Self CPU Time (Function/loop)	
	scalar loop	vectorized loop
Func1	250.6 sec	153.3 sec
Func2	145.9 sec	116.9 sec
	128.34 sec	87.7 sec
	57.6 sec	49.7 sec
Func3	105.8 sec	93.0 sec

Table 4에는 'hot spot'으로 판단되는 주요 루프를 포함한 함수에서 위와 같은 루프에 대해 강제적으로 루프 벡터화를 수행했을 때 얻어진 성능 이득을 도시하였고, 효율이 증가하는 것을 확인했다. 루프 최적화 과정에서 벡터화 뿐만 아니라 계산의 효율을 위한 중복 계산 제거 등 기본적인 scalar tuning도 수행하였으나 성능에 미치는 영향은 벡터화에 비하면 크지 않았다.

추가적인 CPU 최적화를 통한 성능향상을 위해 지수함수 및 삼각함수에 대한 성능 최적화를 수행하였다. SAR 영상처리의 복잡한 계산 특성상 x^p와 같은 지수함수 계산이 상당히 많이 사용되는데 이때 지수가 정수인 경우나 1/2인 경우는 일반적인 pow라는 내부 함수를 쓰는 것보다는 곱의 형태로 코드를 작성하거나 sqrt라는 내부 함수를 쓰는 것이 훨씬 효과적이다. 해당 계산을 반복수행해서 비교한 결과를 나타낸 Table 5에서 보는 바와 같이 매우 큰 성능차이를 확인할 수 있다. 실제로 이와 같은 수정을 본 연구에서 SAR 영상처리 코드에 적용한 결과 전체CPU 시간이 기존 대비 85%로 감소한 것을 확인할 수 있었다.

Table 5. Comparison of exponent computation

CPU time (sec) for repeated computation of X^r			
r	code	data type	
		float	double
r = 3 (x^3)	pow(x,3.0)	21.3	23.6
	pow(x,3)	3.20	3.15
	x * x * x	1.06	1.08
r = 0.5 ($x^{0.5}$)	pow(x,0.5)	20.1	22.8
	sqrt(x)	1.83	1.26

2.4 SAR 영상처리 병렬화

고해상도 SAR 영상처리 시 입력 데이터가 상당히 크기 때문에 앞서 언급한대로 일반적으로 데이터를 메모리에 모두 올리지 않고 swath 단위로 읽어서 처리한다. 이러한 경우 Table 1의 2번 줄에 해당하는 루프에서 보듯이 루프의 시작 부분인 3번째 줄의 HDF5read()함수를 통해 입력 데이터를 순차적으로 읽어서 처리하는 방식으로 프로그램이 실행되게 된다. 이를 그림으로 도식화 하면 다음 Fig. 10과 같이 병렬 처리 영역 외에 중간에 입력 데이터를 읽고 처리하기 위한 순차적인 부분이 발생하게 된다. 이 때문에 코드의 병렬화 비율이 작아져 병렬 성능이 저하되며 또한 병렬 성능 한계치도 매우 낮은 수준에서 형성되어 아무리 코어가 많은 시스템에서 프로그램을 실행해도 코어 수가 증가하는 만큼의 성능을 내지 못하게 된다.

따라서 병렬 성능 및 그 한계치를 높이기 위해서는 반드시 이러한 I/O 등에 의해 발생하는 순차적인 부분을 제거해야 한다. 그러나 I/O는 불가피하게 수행되어야 하는 작업이므로 이 부분을 제거하는 대신 이 부분이 다른 종류의 작업과 동시에 수행될 수 있도록 작업 병렬 처리(task parallelism)를 구현하였다. 즉, 현재 subswath에 대한 연산이 수행되는 동안에 다음 subswath에 필요한 데이터에 대한 읽기 등의 준비 과정을 미리 시작함으로써 연산과 읽기를 동시에 수행하게 되는 I/O overlapping을 구현하여 순차적 부분으로 인해 불가피하게 발생하는 성능 한계를 극복하고자 했다. 이 I/O overlapping을 구현하기 위해서는 병렬 연산용으로 사용하는 스레드 외에 입력 데이터 처리를 위한 추가적인 스레드가 필요한데, 이는 OpenMP에서 제공하는 nested parallelism을 사용하여 구현이 가능했다. 이와 더불어 효율적인 I/O overlapping의 구현을 위해서 입력 데이터 처리에 사용되는 스레드와 병렬 연산에 사용되는 스레드 간의 작업 균형(load balancing)이 이루어져야 하는데 이 경우 OpenMP에서 기본적으로 적용되는 정적 스케쥴링(static scheduling)으로는 다

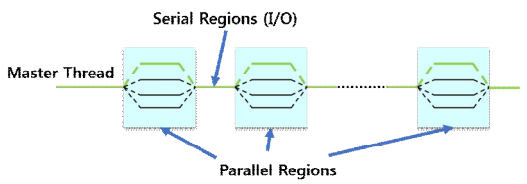

Fig. 10. Intermediate I/O time between parallel region

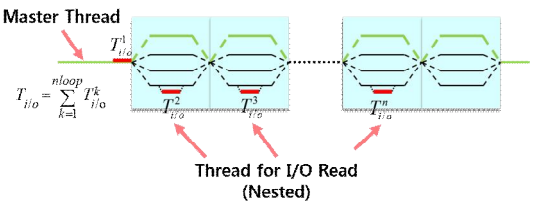

Fig. 11. I/O overlapping by nested parallelism

른 종류의 작업을 수행하는 스레드 때문에 각 코어 간의 작업량이 균형 있게 배분되지 못해서 병렬 성능의 저하가 발생하게 된다. 따라서 본 연구에서는 병렬 연산을 좀 더 잘게 쪼개서 충분한 개수의 작업을 생성하여 입력 데이터 처리 작업과 함께 동적 스케쥴링(dynamic scheduling)을 통해 효율적인 작업 병렬 처리가 구현되도록 했다. 이 때 개별 작업의 크기는 동적 스케쥴링으로 인한 오버헤드가 상대적으로 너무 커지지 않도록 충분한 크기를 갖도록 조절해줘야 한다. 본 연구에서 사용된 데이터의 크기가 매우 크기 때문에 충분한 개수의 작업을 생성하는 데는 큰 문제가 없었다.

이와 같은 과정을 통해 Fig. 10에 있는 병렬 연산 사이의 순차적인 입력 데이터 처리 부분이 Fig. 11과 같이 기존의 병렬 연산 부분과 함께 병렬로 수행이 되게 된다. Fig. 11에서 첫 번째 연산을 하는 루프에서 필요한 데이터는 주 스레드에서 읽고 연산에 사용되는 객체 A에 데이터를 파싱한다. 그리고 객체 A의 데이터를 가지고 루프를 병렬 처리 할 때 추가적인 스레드를 사용해 다음 루프에서 필요한 입력 에디터를 미리 읽는다. 객체 A의 데이터를 이용하여 계산을 수행하는 동안 미리 읽은 입력 데이터는 다른 객체 B에 저장한 다음, 다음 루프 계산 직전에 객체 A의 내부 데이터의 주소와 객체 B의 내부 데이터의 주소를 교환함으로써 객체 복사 시간을 단축하고 다음 루프 계산에서도 객체 A를 가지고 연산을 진행할 수 있게 한다.

프로그램이 실행되는 시간은 루프 계산에 필요한 데이터를 읽고 준비하는데 걸리는 I/O 시간 $T_{i/o}$, 병렬화 가능한 루프 내의 연산 시간 T_{comp}, 기타 순차 처리 부분에 의해 걸리는 시간 T_{serial}로 나눌 수가 있다. 이때 I/O overlapping을 적용하지 않은 병렬 프로그램의 N개의 코어를 사용한 이상적인 실행 시간은 다음과 같이 표현할 수 있다.

$$T_{total} = T^1_{i/o} + \sum_{k=2}^{n} T^k_{i/o} + \frac{T_{comp}}{N} + T_{serial} \quad (2)$$

여기서 I/O 시간은 디스크 I/O 대기 시간 $T^k_{i/o,disk}$ 외에 데이터 재배열 등 CPU 시간을 요구하는 작업들로 인한 시간 $T^k_{i/o,CPU}$도 포함하고 있다.

$$T^k_{i/o} = T^k_{i/o,disk} + T^k_{i/o,CPU} \quad (3)$$

I/O overlapping을 적용한 프로그램에 대하여 병렬 처리 시 이상적인 실행 시간은 다음과 같이 구할 수 있는데 여기서 코어 수로 나누어지는 부분의 비율 자체가 커졌다는 것을 수식적으로 확인 할 수 있다. 뿐만 아니라 루프 두 번째 반복부터는 $T^k_{i/o,disk}$로 표현되는 디스크 I/O 대기 시간이 사라진 것을 알 수 있는데 이는 I/O overlapping을 통해 디스크 I/O 대기 시간에 다른 스레드가 코어를 점유해서 계산을 수행할 수 있게 되기 때문이다.

$$T_{parallel\_total} = T^1_{i/o} + \frac{\sum_{k=2}^{n} T^k_{i/o,CPU} + T_{comp}}{N} + T_{serial} \quad (4)$$

이와 같이 동적 스케쥴링에 의한 작업 병렬 처리를 통해 I/O overlapping을 구현함으로써 코드의 전반적인 병렬성이 커지게 되어 병렬 효율을 크게 향상시킬 수 있었다. 본 연구에서 사용된 SAR 영상처리 코드에서 I/O overlapping을 적용하기 전후의 이론적인 최대 병렬 성능향상률을 다음 Table 6에서 비교하였다. 이때의 이론적 최대 병렬 성능향상률은 식 (1)을 사용해 구했으며 무한개의 코어가 있다고 가정했을 때의 이상적인 값으로 병렬 성능향상률의 상한계(Upper bound)이다. 여기서 식 (1)의 p에 해당하는 병렬화 가능한 작업량의 비율은 Intel사의 프로파일링 툴인 VTune[16]을 이용한 코드 분석을 통해 전체 실행 시간과 OpenMP fork 함수로부터 파생된 실행 시간의 비율을 이용해서 구하였다.

Table 6에서 보는 바와 같이 I/O overlapping을 통해 병렬 성능의 한계가 순차 성능 대비 8.33배에서 40배로 크게 향상되었다. 이는 매우 중요한 의미를 갖는 것으로서 고사양 하드웨어를 통해 병렬 계산 자원의 충분한 확보가 가능하다면 I/O

Table 6. Maximum parallel performance

I/O overlapping	Parallel portion	Ideal speedup
Unused	88.80%	8.33
Used	97.45%	40.00

Table 7. Performance analysis

# threads	Org	Mod2	Performance Ratio
1	7,536 sec	3,771 sec	2.00
10	7,104 sec	868 sec	8.18
Parallel Speedup	1.06	4.34	-

overlapping을 적용한 코드가 그렇지 않은 코드보다 최대 4.8배 정도까지의 우수한 성능을 낼 수 있는 잠재력을 확보했다는 뜻이다. 즉, I/O overlapping을 적용한 코드는 하드웨어 사양이 높을수록 그에 따른 성능 향상을 더 크게 기대할 수 있다.

이상에서 적용한 모든 성능 최적화 기법의 종합적인 효과를 확인하기 위해 최종적으로 개선된 코드의 성능을 최적화 이전의 기존 코드의 성능과 비교해서 다음 Table 7에 나타내었다. Table 3의 시스템에서 1개의 스레드를 사용했을 때와 최대 스레드(10개)를 사용했을 때의 실제 계산 시간을 비교하였다. 'Mod2'로 표시된 것이 I/O overlapping을 적용한 최종 코드의 결과이다. 10개의 스레드를 사용했을 때 Table 6의 성능 한계치에는 미치지 못하지만 최적화 이전 코드에 비하여 상당한 성능 향상을 확인할 수 있다. 특히 순차성능, 즉 1개의 스레드를 사용했을 때의 성능 최적화로 인한 성능 향상은 2배 정도에 그치지만 10개의 스레드를 사용한 병렬 성능의 경우는 기존 코드에 비하여 8배가 넘는 성능 향상을 이루게 된 것을 확인할 수 있다. 또한 병렬 성능향상률에 있어서도 기존 코드는 10개의 스레드를 사용하더라도 1개를 사용했을 때에 비해서 거의 성능 향상이 없는 반면, 수정된 코드는 10개의 스레드를 사용하는 경우 1개를 사용하는 경우에 비해서 4.34배 정도 빠르게 계산을 수행할 수 있는 것을 확인할 수 있다. 그리고 본 연구를 통해 병렬화 가능한 작업량의 비율이 크게 향상됨에 따라 고사양 하드웨어를 통해 더 많은 코어 수 및 GPU 등의 추가적인 병렬 계산 자원의 확보가 가능하다면 Table 6의 결과에 근접해 가는 더 큰 성능 향상이 가능할 것으로 예상된다.

Ⅲ. 결 론

본 연구에서는 고해상도 SAR 영상처리를 위한 소프트웨어가 공유 메모리 시스템에서 보다 효율적인 영상처리를 수행할 수 있기 위한 성능 최적화 기법을 연구했다. 순차성능 최적화 방안으로는 최신 하드웨어의 구조를 고려하여 메모리 및 I/O 최적화, SIMD 연산을 위한 벡터화 등 여러 성능 향상 기법을 적용했다. 또한 고성능 SAR 영상처리에서 성능 저하의 가장 큰 요인이 될 수 있는 반복적인 I/O에 대해서 동적 스케쥴링에 의한 작업 병렬화의 일환으로 I/O overlapping을 구현함으로써 코드의 병렬화 비율을 높여 성능 향상 한계를 크게 높일 수 있었다.

본 연구에서 적용된 성능 최적화 기법들은 많은 계산량이 요구되는 다양한 수치해석 기법들에도 일반적으로 적용될 수 있는 효율적인 기법들로서 본 연구를 통해 이러한 기법들을 SAR 영상처리 소프트웨어 성능 개선에 적용했을 때의 효용성을 확인할 수 있었다.

본 연구를 통해 기존에 개발했던 SAR 영상처리 소프트웨어의 성능을 비약적으로 향상시켜 전체 영상 처리 시간이 대폭 감소했을 뿐만 아니라, 병렬 처리를 통한 성능 향상의 여지 또한 크게 증가했다. 따라서 본 연구를 통해 병렬 최적화된 프로그램이 수많은 코어를 가지고 있는 Intel Xeon Phi나 고성능 GPU와 같은 하드웨어 가속기를 장착한 보다 더 고사양의 하드웨어에 적용된다면 더 큰 비율의 성능 향상을 달성할 수 있을 것으로 기대된다.

후 기

본 연구는 2017년도 정부의 지원으로 한국연구재단의 지원을 받아 수행된 다목적실용위성6호 개발사업의 일환으로 수행되었습니다. (NRF-2017M1A3A4A04018688)

References

1) Lee, W. K., and Kwak, Y. K., "SAR Image Processing Technique," *The Proceedings of the Korea Electromagnetic Engineering Society*. Vol. 22, No. 6, 2011, pp.40~54.

2) Prats, P., Scheiber, R., Mittermayer, J., Meta, A., and Moreira, A., "Processing of Sliding Spotlight and TOPS SAR Data Using Baseband Azimuth Scaling," *Institute of Electrical and Electronics Engineers*

TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING. Vol. 48, No. 2, 2010, pp.770~780.

3) Bhogendra Rao, P. V. R. R., and Shashank, S. S., "Parallelization of Synthetic Aperture Radar (SAR) Image Formation Algorithm," *Proceedings of the First International Conference on Computational Intelligence and Informatics*, Springer, 2016, pp.713~722.

4) Pandya, B., and Gajjar, N., "Parallization of Synthetic Aperture Radar (SAR) Imaging Algorithms on GPU," *International Journal of Computer Science & Communication*. Vol. 5, No. 1, 2014, pp.143~146.

5) Peternier, A., Deflippiy, M., Pasqualiy, P., Cantone, A., Krause, R., Vitulli, R., Ogushi, R., and Meroni, A., "Performance Analysis of GPU-based SAR and Interferometric SAR Image Processing," *2013 Asia-Pacific Conference on Synthetic Aperture Radar*, September 2013, pp.277~280.

6) Fasih, A., and Hartley, T., "GPU-accelerated Synthetic Aperture Radar Back-projection in CUDA," *2010 Institute of Electrical and Electronics Engineers International Radar Conference*, May 2010, pp. 1408~1413.

7) Zhang, F., Li, G., Li, W., Hu, W., and Hu, Y., "Accelerating Spaceborne SAR Imaging Using Multiple CPU/GPU Deep Collaborative Computing," *Journal of Sensors*, Vol. 16, No. 4, 2016, pp.494~512.

8) Park W. S., Tak, K. H., and Bang H. C., "Acceleration of SAR Image Preprocessing using Graphic Processing Unit," *Proceeding of The Korean Society for Aeronautical and Space Sciences Fall Conference*. November 2015, pp.1342~1345.

9) Cumming, I. G., and Wong, F. H., *Digital Processing of Synthetic Aperture Radar Data*, Artech House INC., Norwood, MA, USA, 2005, pp.283~322.

10) Papulis, A., *Systems and Transforms with Applications in Optics*, McGraw-Hill, New York, 1968.

11) Hennessy, J. L., and Patterson, D. A., *Computer Architecture: A Quantitative Approach*, 5th Ed., Elsevier, MA, USA, 2012, pp.46~48.

12) Chellappa, S., Franchetti, F., and Puschel, M., "How to Write Fast Numerical Code: A Small Introduction," *Generative and Transformational Techniques in Software Engineering II : International Summer School, GTTSE 2007, Braga, Portual, July 2-7. 2007, Revised Papers*, Springer, 2007, pp.196~259.

13)Hennesy, J. L., and Patterson, D. A., *Computer Architecture : A Quantitative Approach*, 5th Ed., Elsevier, Waltham, MA, USA, 2012.

14) http://www.openmp.org

15) Bramas, B., "Fast Sorting Algorithms using AVX-512 on Intel Knights Landing," arXiv preprint arXiv:1704.08579, 2017.

16) https://software.intel.com/en-us/intel-vtune-amplifier-xe