공 학 석 사 학 위 논 문

고성능 파이썬을 이용한 3차원 시간 영역 파동 전파 모델링

2018년 2월

부경대학교 대학원

에너지자원공학과

조 상 훈

공 학 석 사 학 위 논 문

고성능 파이썬을 이용한 3차원 시간 영역 파동 전파 모델링

지도교수 하 완 수

이 논문을 공학석사 학위논문으로 제출함

2018년 2월

부경대학교 대학원

에너지자원공학과

조 상 훈

조상훈의 공학석사 학위논문을 인준함

2018년 2월

주 심 공학박사 최 요 순 (인)

위 원 공학박사 하 완 수 (인)

위 원 공학박사 엄 정 기 (인)

초 록

파형역산 또는 역시간 구조보정과 같은 3차원 탄성파 자료 처리는 방대한 양의 데이터가 필요하고 그에 따른 대량의 수치 계산이 필요하다. 본 연구에서는 3차원 시간 영역 파동전파 모델링을 이용하여 고성능 파이썬과 C언어의 성능 및 생산성을 비교하였다. 고성능 파이썬으로는 Numba와 Cython을 이용하였다. 모델 크기를 변경시켜가며 수치 모델링을 실시하였고 모델링을 수행한 결과, Numba와 Cython, C언어는 순수 파이썬에 비해 각각 약 1700배, 1800배, 1900배 빠른 성능을 얻을 수 있었다. 그리고 고성능 파이썬인 Numba와 Cython은 C언어에 비해 최대 각각 18.9%, 9.4% 가량 느렸지만, 코드 줄 수로 비교한 생산성은 38.1%, 19.6% 가량 높았다. 따라서 10~20% 정도의 성능이 중요하지 않다면, 생산성이 뛰어난 고성능 파이썬을 이용하는 것이 더욱 효율적일 것이다. 그러므로 고성능 파이썬은 대규모 자료처리 분야 및 고성능 컴퓨팅 분야, 수치모델링 분야 등에서 충분히 사용가능할 것으로 예상된다. 또한 고성능 파이썬은 C언어나 포트란 같은 정적 프로그래밍 언어의 생산적인 대안이 될 수 있을 것이라 기대된다. 향후 파형역산 및 역시간 구조보정에 적용해 보겠다.

주요어 : 3차원, 파동 전파 모델링, 고성능 파이썬

학 번 : 201655282

목 차

List of Figures

List of Tables

1. 서 론

탄성파 탐사는 고 분해능의 지하영상을 얻는 것이 목적이며, 초창기 석유 및 가스 탐사 분야에서는 천부 지층이 주 타깃이었기 때문에 탐사가 용이했다. 하지만 최근에는 심해 지역 등 탐사가 어려운 지형 등을 조사해야 하는 경우가 증가함에 따라 석유 및 가스 탐사 분야, 물리 탐사 분야에서는 3차원 탄성파 탐사를 이용하여 복잡한 지질구조를 규명하고 있다. 일반적으로 3차원 탄성파 탐사는 2차원 탄성파 탐사보다 탐사 범위가 방대하므로 더 많은 데이터를 획득하게 되며, 3차원 탄성파 탐사를 통해 얻은 방대한 데이터를 처리하기 위해서는 많은 노력이 필요하다. 또한 많은 시간을 소모하게 된다. 특히 완전 파형 역산이나 역시간 구조보정과 같은 3차원 탄성파 탐사 자료처리는 일반적인 2차원, 3차원 탄성파 탐사 자료처리에 비해 더 많은 데이터를 처리해야하므로 계산량이 증가하게 되고 소모되는 시간과 노력 또한 만만치 않다.

일반적으로 수치해석 분야에서는 정확도와 계산 성능을 중요하게 생각한다. 만약 정확도가 비슷하다면 프로그램 계산 속도에 초점을 맞추게 된다 (Ha, 2014). 그렇기 때문에 수치해석 분야에서는 계산 시간이 더 빠른 C/C++, 포트란과 같은 정적 자료형을 지원하는 프로그래밍 언어를 선호한다 (Press et al., 1996).

동적 프로그래밍 언어 중 하나인 파이썬 언어는 객체지향 프로그래밍을 지원하기 때문에 코드 작성이 수월하다는 장점을 지니고 있다 (Ha, 2014). 파이썬 언어의 특징으로는 여러 가지 운영체제에 쉽게 이용 가능하며, 컴파일 과정이 따로 필요하지 않으며 파이썬 언어는 오픈 소스이기 때문에 사용자의 접근이 용이하다. 또한 파이썬 언어는 다른 언어로 쓰여진 모듈들을 연결해주는 풀언어의 기능도 지닌다. 실제 파이썬 언어는 많

은 사용 프로그램에 스크립트 언어로 사용 중이며, 다양한 언어의 문자 처리에도 많이 이용되고 있다. 파이썬 언어의 가장 큰 장점은 코드 작성이 쉬운 동적 자료형을 지원하므로 비전문가들이 처음 학습하기에 가장 용이한 프로그래밍 언어이다. 현재 파이썬 언어는 수치해석 분야뿐만 아니라 웹 분야 및 게임 산업 분야 등 다양한 분야에서 사용이 증가하고 있다. 파이썬 언어의 여러 장점 중 동적 자료형을 지원하는 것이 생산성과 관련이 있기 때문에 가장 중요한 특징이라고 할 수 있다 (Ha, 2014).

자료형에는 동적 자료형과 정적 자료형이 존재하며, 동적 자료형과 정적 자료형은 서로 상반되는 관계이다. 정적 자료형을 지원하는 언어로는 C/C++, 포트란 등이 있으며, 컴파일 시에 자료형을 검사하게 된다. 정적 자료형 언어는 제한된 형태이므로 타입 에러 문제를 초기에 발견할 수 있어 안전성을 높일 수 있으며, 정적 자료형 언어는 프로그램 실행시 매번 자료형 검사를 반복할 필요가 없으므로, 실행 자료형 검사를 생략하고 다른 최적화를 수행할 수 있어 프로그램 실행속도가 증가하게 된다 (Wikipedia, 2017).

이와 반대되는 개념인 동적 자료형 언어는 런타임 시에 자료형을 검사하며, 동적 자료형을 지원하는 언어로는 줄리아, 매트랩, 파이썬 등이 있다. 동적 자료형 언어에서 변수는 자료형을 가지지 않으므로, 변수는 자료형에 상관없이 모든 값을 가질 수 있다 (Wikipedia, 2017). 또한 동적 자료형 언어는 변수 선언이 필요 없으므로 함수의 자료형을 바꿔가며 테스트를 수행할 수 있고, 변수 선언부를 생략할 수 있기 때문에 코드 줄 수를 줄일 수 있다. 하지만 동적 자료형 언어는 내부적으로 변수에 값을 할당하게 되며, 모델링 시 변수의 자료형을 찾아서 함수에 적용해야하기 때문에 정적 자료형 언어보다 계산 시간이 늘어나게 된다 (Ha, 2014). 또한 동적 자료형 언어는 한 줄씩 해석하기 때문에, 정적 자료형 언어보다 계산 시간이

증가하게 된다 (Beazley, 2009). 이와 더불어 실행 도중 변수에 예상하지 못한 타입이 들어와 타입 에러가 발생할 수도 있다.

일반적으로 정적 프로그래밍 언어보다 계산 속도가 느린 동적 프로그래밍 언어는 수치 모델링 분야와 탄성파 탐사 분야에서 선호되지 않았다. 따라서 동적 프로그래밍 중 하나인 파이썬은 주로 계산 시간이 크지 않은 소규모 자료처리에 사용되곤 했다. 예를 들어 특정 자료에만 이용할 수 있는 속도모델 및 공통 송신원모음 등을 수정하거나 간단한 알고리즘 등을 개발하는 경우에 사용되었다 (Ha, 2014). 이러한 자료처리는 상대적으로 많은 데이터를 사용하지 않기 때문에 모델링을 수행하는데 많은 시간을 소모할 필요가 없다. 하지만 최근 많은 연구와 발전을 통해 파이썬 언어의 계산 속도가 상당히 발전하였고, 공학 분야에서도 사용량이 증가하고 있다 (McKinney, 2012; Lanaro, 2013). 이는 많은 라이브러리들의 개발로 인한 결과로 대표적인 예로 Numpy 라이브러리가 있으며, Numpy 라이브러리 내부는 정적 프로그래밍 언어인 C언어나 포트란 언어를 사용하여 상당 부분 작성되어 있어 일반적인 파이썬 라이브러리에 비해 실행 속도가 빠르다. Numpy 라이브러리는 array를 생성하고 array를 바탕하고 연산 등을 수행하게 된다. 또한 Numpy 라이브러리는 난수생성, 푸리에 변환과 같은 함수들이 구현되어 있다. 일반적으로 파이썬을 이용하여 수치해석 및 모델링을 수행할 때 Numpy는 가장 기본이 되는 모듈이며 파이썬 사용자들이 가장 많이 사용하는 라이브러리다. 이와 더불어 파이썬의 성능을 극대화시키는 고성능 파이썬과 관련된 연구도 많이 진행되고 있다 (Numba, 2017; Cython, 2017).

파이썬의 계산 효율을 극대화 시키는 방법 중 하나는 Numba 장식자를 사용하는 것이다. Numba는 Continuum Analytics에서 후원하는 파이썬용 오픈소스로서, Numba의 가장 큰 장점은 파이썬 코드를 따로 수정할 필요

없이 간단한 장식자를 이용하여 사용할 수 있다는 것이다 (Numba, 2017). 파이썬의 계산 효율을 증가시키는 또 다른 방법은 Cython이다. Cython은 C언어와 같이 변수에 자료형 지정이 필요하다. 변수에 자료형을 지정하면 C언어로 작성된 코드를 자동으로 생성하고, 컴파일하게 되면 파이썬에 사용가능하도록 만들어준다. Cython의 장점은 C언어를 직접 사용할 필요 없이 파이썬 코드의 변수에 자료형을 지정하고 간단한 명령어 추가를 통해 속도 향상을 얻을 수 있다. 이외에도 파이썬 언어의 계산 효율을 증가시키는 여러 방법들이 존재하며, C 코드를 파이썬에서 불러서 사용할 수 있도록 해주는 Ctypes등이 있다 (Numpy, 2017; Python, 2017). 또한 계산 시간을 많이 차지하는 부분만 정적 자료형 언어를 사용하여 작성한 후 파이썬에서 불러와 사용하도록 도와주는 방식도 존재한다 (Lanaro, 2013).

파이썬의 계산 효율을 극대화 시키는 방법 중 하나는 Numba 장식자를 사용하는 것이다. Numba는 Continuum Analytics에서 후원하는 파이썬용 오픈소스로서, Numba의 가장 큰 장점은 파이썬 코드를 따로 수정할 필요 없이 간단한 장식자를 이용하여 사용할 수 있다는 것이다 (Numba, 2017). 파이썬의 계산 효율을 증가시키는 또 다른 방법은 Cython이다. Cython은 C언어와 같이 변수에 자료형 지정이 필요하다. 변수에 자료형을 지정하면 C언어로 작성된 코드를 자동으로 생성하고, 컴파일하게 되면 파이썬에 사용가능하도록 만들어준다. Cython의 장점은 C언어를 직접 사용할 필요 없이 파이썬 코드의 변수에 자료형을 지정하고 간단한 명령어 추가를 통해 속도 향상을 얻을 수 있다. 이외에도 파이썬 언어의 계산 효율을 증가시키는 여러 방법들이 존재하며, C 코드를 파이썬에서 불러서 사용할 수 있도록 해주는 Ctypes등이 있다 (Numpy, 2017; Python, 2017). 또한 계산 시간을 많이 차지하는 부분만 정적 자료형 언어를 사용하여 작성

한 후 파이썬에서 불러와 사용하도록 도와주는 방식도 존재한다 (Lanaro, 2013).

본 연구에서는 고성능 파이썬과 C언어를 3차원 시간 영역 파동 전파 모델링을 이용하여 성능 및 생산성을 비교하였다. 성능을 비교하기 위해 프로그램 실행시간을 이용하였고, 정량적으로 비교하기 어려운 생산성은 코드의 줄 수를 이용하여 비교하였다 (Prechelt, 2000). 동적 자료형을 지원하는 파이썬은 정적 자료형을 지원하는 C언어보다 계산 속도가 느리기 때문에 고성능 파이썬을 적용하였다. C언어는 파이썬 언어보다 계산 속도가 빠르고 수치해석 분야에서 가장 선호하는 프로그래밍 언어 중 하나이며, C 코드를 생성하여 컴파일하는 Cython과 유사하므로 직접적인 비교를 위해 고성능 파이썬의 비교대상으로 선택했다. 본 연구를 수행하기 위해 파이썬은 버전 3.5.3, Numpy는 버전 1.11.1, Numba는 버전 0.26, Cython은 버전 0.24 등의 라이브러리를 이용하여 고성능 파이썬을 구현하였다 (Ha, 2014). 그리고 gcc 컴파일러 버전 4.4.7을 이용하였으며 -O2 컴파일 옵션을 이용하였다.

2. 파동 전파 모델링 알고리즘

2.1 시간 영역 음향파 유한차분법 모델링

본 연구에서는 유한 차분법을 이용하여 3차원 시간 영역 음향파 파동 방정식 모델링을 구현하였다. 유한 차분법의 여러 방법 중 격자 기반의 유한 차분법을 사용하였다 (Kelly et al., 1976). 격자 기반의 유한 차분법은 기하학적인 영역을 생성하고 유한개의 격자점을 생성하게 되며, 서로 이웃하는 격자점들 사이에서 위치에 따른 변화를 이용해 미분방정식을 행렬방정식으로 전환시켜 계산하게 된다.

3차원 등방성 매질에 대한 시간 영역 음향파 파동 방정식은 다음과 같다.

$$\frac{1}{v^2}\frac{\partial^2 u}{\partial t^2}=\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2}+f \qquad (1)$$

위 식의 파동 방정식을 수치 해석적으로 풀기 위해서는 유한차분법을 사용하여 미분항을 전개한 후 근사시키며, 3개의 중첩된 루프로 이루어진 부분이 유한차분법에서 가장 핵심이 되는 부분이다 (Ha, 2014).

먼저 좌변항을 전개하면 다음과 같다.

$$\frac{1}{v^2}\frac{u^{k+1}-2u^k+u^{k-1}}{(\triangle t)^2}=\frac{\partial^2 u^k}{\partial x^2}+\frac{\partial^2 u^k}{\partial y^2}+\frac{\partial^2 u^k}{\partial z^2}+f^k \qquad (2)$$

한 차분식에서 3개의 중첩된 루프로 이루어진 곳이 핵심이 되는 부분이다

(Ha, 2014). 위 식에서 첨자 k는 시간 인덱스이며, Δt는 시간격자 간격을 의미한다. 시간 영역 음향파 파동 방정식 모델링은 매 시간마다 루프를 돌리는 시간 전진 기법을 이용하여 모델링을 수행할 수 있으며 (Kim, 2017), 미래의 파동장을 계산하기 위해서는 과거와 현재의 파동장을 알고 있어야한다. 식 (2)를 정리하여 미래의 파동장을 얻을 수 있다.

$$u^{k+1} = 2u^k - u^{k-1} + v^2(\triangle t)^2(\frac{\partial^2 u^k}{\partial x^2} + \frac{\partial^2 u^k}{\partial y^2} + \frac{\partial^2 u^k}{\partial z^2} + f^k) \tag{3}$$

식 (3)의 우변의 미분항을 전개하면 다음과 같다.

$$u_{i,j,k}^{k+1} = 2u_{i,j,k}^k - u_{i,j,k}^{k-1} + v_{i,j,k}^2(\triangle t)^2(\frac{u_{i-1,j,k}^k - 2u_{i,j,k}^k + u_{i+1,j,k}^k}{(\triangle x)^2} + \frac{u_{i,j-1,k}^k - 2u_{i,j,k}^k + u_{i,j+1,k}^k}{(\triangle y)^2} + \frac{u_{i,j,k-1}^k - 2u_{i,j,k}^k + u_{i,j,k+1}^k}{(\triangle z)^2} + f_{i,j,k}^k) \tag{4}$$

위 식에서 i, j, k는 x, y, z방향의 위치를 나타내는 공간 인덱스이며, Δx, Δy, Δz는 공간격자 간격을 나타낸다. 일반적으로 공간격자 간격이 동일하다고 가정하기 때문에 Δx, Δy, Δz는 h로 표현할 수 있으며, 위 식 (4)를 정리하면 다음과 같다.

$$u_{i,j,k}^{k+1} = 2u_{i,j,k}^k - u_{i,j,k}^{k-1} + v_{i,j,k}^2(\triangle t)^2(\frac{u_{i-1,j,k}^k + u_{i,j-1,k}^k + u_{i,j,k-1}^k - 2u_{i,j,k}^k + u_{i+1,j,k}^k + u_{i,j+1,k}^k + u_{i,j,k+1}^k}{h^2} + f_{i,j,k}^k) \tag{5}$$

위 식을 반복적으로 수행하면 파동 전파 모델링을 구현할 수 있다. 따라서

시간 영역 파동 전파 모델링은 현재와 과거의 파동장을 이용하여 미래의 파동장을 얻는 것이 주요하며, 이를 반복 수행함으로써 시간 영역 파동 전파 모델링을 수행할 수 있다.

2.2 송신원 추가

송신원은 음향파 파동 방정식을 유한 차분법을 사용하여 전개하면 계산이 가능하다. 유한 차분법을 이용하여 음향파 파동 방정식을 전개하여 정리한 식은 다음과 같다.

$$u_{i,j,k}^{k+1} = 2u_{i,j,k}^{k} - u_{i,j,k}^{k-1} + v_{i,j,k}^{2}(\triangle t)^{2}(\frac{u_{i-1,j,k}^{k} + u_{i,j-1,k}^{k} + u_{i,j,k-1}^{k} - 2u_{i,j,k}^{k} + u_{i+1,j,k}^{k} + u_{i,j+1,k}^{k} + u_{i,j,k+1}^{k}}{h^{2}} + f_{i,j,k}^{k}) \quad (6)$$

일반적으로 계산을 효율적으로 수행하기 위해 위 식(6)에서 송신원 부분은 따로 추출하여 정리한 후 파동 방정식과 별개로 수행하게 된다. 따라서 3차원 시간 영역 파동 전파 모델링을 구현하기 위해 위 식(6)에서 $v_{i,j,k}^{2}(\triangle t)^{2} * f_{i,j,k}^{k}$ 를 따로 추출하여 송신원을 추가하게 되며, 송신원을 제외한 나머지 항을 이용하여 파동 방정식을 풀게 된다.

2.3 Keys 경계 조건

실제 지하는 반무한 매질이기 때문에 반사파들이 발생하지 않는다. 하지만 유한 차분법이나 유한 요소법 등을 이용하는 수치 모델링 분야에서의 모델은 유한한 크기를 가정하므로 반사파가 발생하게 된다. 이러한 반사파는 탄성파 모델링 자료 처리 및 해석 시 혼동을 주거나, 기록된 파를 가릴 수 있기 때문에 적절한 방법을 이용해 반사파를 제거하는 것이 중요하다.

반사파를 제거하는 방법 중 주로 사용하는 방법은 적절한 경계 조건을 사용하는 것이다. 경계 조건에는 흡수 경계 조건과 투과 경계 조건이 존재한다. 흡수 경계 조건은 경계 영역을 모델 영역보다 좀 더 크게 바깥쪽에 적용하여 경계 영역으로 전파되는 파를 흡수하도록 만들어졌다. 쉽게 설명하면 외부로 전파되는 파는 점점 감쇠하게 되고 파가 경계 영역에 다가갈수록 파의 진폭은 현저히 감소하게 된다. 따라서 경계 영역에 도달하게 되면 파의 진폭은 모델 영역에 영향을 끼치지 않게 된다. 투과 경계 조건은 흡수 경계 조건과 유사하나 파를 모델링 영역 바깥쪽으로 흡수시키는 것이 아니라 전파시킨다는 차이가 존재한다.

본 연구에서는 투과 경계 조건 중 하나인 Keys 투과 경계 조건을 사용하였다. 파를 들어오는 파와 나가는 파로 분리시키는 Keys 투과 경계 조건은 파를 분리시킴으로써 평면파로부터 발생하는 반사파를 전파 방향에 따라서 반사파를 제거시키도록 만들어졌다 (Keys, 1985). 또한 유한 차분법을 이용하여 3차원 시간 영역 파동 방정식을 전개한 후, 송신원을 추가하였으며 Keys 경계 조건을 적용시켰다. 위 알고리즘을 파이썬 언어를 사용하여 모델링을 구현하였다. 파이썬을 이용하여 구현한 3차원 파동 전파 모델링 알고리즘을 Table 1에 나타내었다.

다음으로 고성능 파이썬을 구현하기 위해 Numba와 Cython의 라이브러리

를 이용하였으며, 파이썬으로 작성한 3차원 시간 영역 파동 전파 모델링 코드를 각 라이브러리에 맞게 수정한 후 적용하였다.

Table 1 The algorithm of 3D wave propagation modeling in time domain.

Algorithm
Variable initialize For i in range(time) Solve finite-difference equation Add the inject source Apply the boundary condition Apply time marching Return output

3. 고성능 파이썬

3.1 파이썬

프로그래밍 언어에는 많은 언어들이 존재하며, 주로 C/C++, 포트란, 자바, 파이썬 등의 프로그래밍 언어를 사용한다. 웹 분야에서는 객체 지향 프로그래밍을 지원하는 자바 언어를 선호하며, 수치 모델링 분야에서는 일반적으로 정적 자료형을 지원하는 언어를 많이 사용한다. 정적 자료형을 지원하는 언어로는 대표적으로 C/C++, 포트란 등이 존재한다. 정적 자료형을 지원하는 언어는 일반적으로 계산 성능이 뛰어나다는 장점을 지니고 있으나, 코드 작성에는 어려움이 존재한다. 이에 반해 동적 자료형을 지원하는 언어들은 정적 자료형 언어에 비해 계산 성능은 떨어지나 코드 작성에 용이하다는 장점을 지닌다. 이는 객체 지향 프로그래밍을 지원하기 때문이다 (Ha, 2014). 동적 자료형을 지원하는 대표적인 프로그래밍 언어로는 파이썬 자바 등이 있으며, 수치 모델링 분야에서는 파이썬을 주로 이용한다.

동적 자료형을 지원하는 파이썬 언어는 변수의 자료형을 찾아야하므로 변수의 자료형을 선언하는 정적 자료형 언어보다는 성능이 떨어지게 된다 (Beazley, 2009; Ha, 2014). 이와 더불어 실행 도중 변수에 예상하지 못한 타입이 들어와 타입 에러가 발생할 수도 있다. 하지만 변수 선언이 필요 없기 때문에 함수의 자료형을 바꿔가며 테스트를 수행할 수 있고, 변수 선언부가 생략되기 때문에 코드 줄 수를 줄여 생산성을 높일 수 있다는 장점이 존재한다.

또한 최근 프로그래밍을 비전공자들에게 알려주는 문화가 증가함에 따라 파이썬 언어는 더욱 각광받고 있다. 이는 객체 지향 프로그래밍을 지원하

기 때문에 파이썬 언어의 문법이 간결하기 때문이다. 따라서 초보자들이 코드 작성 및 이해하기 가장 용이하기 때문이다. 이와 더불어 고성능 파이썬 라이브러리에 관한 많은 연구와 발전을 통해 파이썬 언어의 계산 성능이 많이 증가하였다.

3.2 파이썬의 성능을 증가시키는 방법

일반적으로 계산 속도를 증가시키기 위해서는 명령어를 줄이는 것이 중요하다. 가장 간단하게 명령어를 줄이는 방법은 기계어로 코드를 컴파일하면 된다.

컴파일 하는 방식에는 AOT (Ahead Of Time)와 JIT (Just In time)가 존재한다. AOT 컴파일러는 컴파일을 미리 실행하는 것으로서, 정적 라이브러리들을 사용할 컴퓨터에 적합하도록 생성해준다. AOT 컴파일러는 사용하기 전에 컴파일을 실시하기 때문에 해당 라이브러리들을 코드에서 바로 사용할 수 있다는 장점이 있으며, Cython, Shed Skin, Pythran 등에서 사용되고 있다. JIT 컴파일러는 적시에 컴파일 하는 것을 뜻하며, Numba, PyPy 등에서 사용된다. 하지만 콜드 스타트 문제가 발생할 수도 있으며, 콜드 스타트란 최초 시점에 컴파일을 실시하게 되면 프로그램 실행 속도가 느린 것을 의미한다. AOT 컴파일러를 사용하는 것이 JIT 컴파일러를 사용하는 것보다 더 나은 속도 향상을 주지만 직접적인 노력이 필요하다는 단점이 존재하며, JIT 컴파일러는 큰 힘을 들이지 않고 쉽게 속도 향상을 이룰 수 있다는 장점이 있다.

3.3 고성능 파이썬의 종류 및 특징

현재 파이썬의 계산 성능을 증가시키는 고성능 파이썬으로 Cython, Shed Skin, Pythran, Numba, PyPy 등이 주로 이용되고 있다. Cython은 C언어와 비슷하게 파이썬 코드에 타입을 지정해줌으로서 파이썬 코드를 C 모듈로 컴파일해주는 컴파일러로서, 파이썬에서 import문을 활용하여 Cython을 이용할 수 있다.

Shed Skin 컴파일러는 파이썬 코드를 C++ 코드로 변환시켜 주며, 파이썬 함수의 변수를 자동으로 지정해주는 타입 추론을 사용한다. Shed Skin에서 지원하는 타입 추론은 이용자들이 함수 호출 방법만 제대로 알고 있다면 변수 타입을 지정하는 일은 Shed Skin이 알아서 실시하게 된다. 따라서 타입 추론은 굉장히 유용한 기능이라 할 수 있다. 하지만 파이썬에서 가장 널리 사용되는 Numpy 라이브러리는 지원하지 않는다.

Pythran 컴파일러는 파이썬 코드 일부를 C++ 코드로 변환시켜 준다. Pythran은 병렬화할 수 있는 부분을 찾아서 직접 병렬화를 시도하기 때문에 사용자가 직접 코드를 수정해야하는 번거로움이 없으며, 코드를 파이썬과 호환되도록 생성해준다. 하지만 아직 개발이 진행 중이기 때문에 종종 버그가 발생할 수도 있다.

Numba는 LLVM 컴파일러를 사용하여 컴파일 하며 JIT 컴파일러를 지원하기 때문에 적시에 컴파일을 실행하게 된다. Numba의 가장 큰 장점은 약간의 노력으로 큰 속도 향상을 이룰 수 있다는 점이다.

PyPy는 추적형 JIT 컴파일러를 사용하며, 코드의 큰 수정 없이도 속도 향상을 이룰 수 있다는 장점이 있다. 또한 모든 내장 모듈을 지원하고 있으며, 계속해서 발전을 해가는 중이다. 하지만 Shed Skin과 마찬가지로 가장 널리 사용하는 파이썬 라이브러리인 Numpy를 지원하지 않는다는 단점

이 존재한다.

본 연구에서는 고성능 파이썬을 적용하여 파이썬의 성능을 증가시켰다. 고성능 파이썬으로는 Cython과 Numba를 채택하여 사용하였다. Cython과 Numba를 고성능 파이썬으로 채택한 이유는 Cython은 AOT 컴파일러, Numba는 JIT 컴파일러를 사용하는 가장 대표적인 프로그램이자 현재 가장 널리 사용되는 방법들 중 하나이기 때문이다. 따라서 본 연구에서는 3차원 시간 영역 파동 전파 모델링에 Cython 및 Numba의 프로그램을 활용하여 파이썬의 성능을 증가시켰고, 성능 및 생산성을 C언어와 비교하였다.

3.4 고성능 파이썬 적용

3.4.1 Numba 적용

본 연구에서는 파이썬의 성능을 극대화시키기 위한 방법 중 하나로 Numba를 사용했다. Numba는 Continuum Analytics에서 후원하는 파이썬 오픈 소스이며, LLVM (Low Level Virtual Machine) 컴파일러 인프라를 사용하여 파이썬을 머신 코드 (Machine code)로 컴파일한다 (LLVM, 2017). LLVM 컴파일러는 프로그램을 컴파일 타임, 링크 타임, 런타임 상황에서 프로그램의 작성 언어에 상관없이 최적화를 쉽게 구현할 수 있도록 구성한 컴파일러이다 (LLVM, 2017). Numba의 단점은 툴체인으로서, LLVM 컴파일러를 사용하기 때문에 의존 관계가 복잡하므로 Intel Python이나 Anaconda에서 배포하는 프로그램을 이용하는 것이 유리하다.

Numba를 사용하는 방법은 간단하다. 컴파일을 적용시키고자 하는 함수 앞에 @jit라는 장식자를 추가한다. Jit는 just in time 컴파일러를 뜻하며 이는 적용 함수의 실행 코드를 적시에 컴파일하는 것을 의미한다. Numba 장식자를 이용한 고성능 파이썬은 기존에 작성한 파이썬 코드에 단지 장식자 한 줄을 추가 함으로서 기존의 파이썬 코드보다 월등한 속도 향상을 얻을 수 있다. 심지어 인자 타입이 같을 시, 함수를 다시 실행할 때는 컴파일을 한번 더 할 필요가 없기 때문에 좀 더 빠른 속도를 얻을 수 있다 (Numba, 2017).

Table 2에 Numba를 이용한 3차원 시간 영역 파동 전파 모델링 알고리즘을 나타내었으며, Fig. 1에 Numba를 적용한 파이썬 코드의 예시를 나타내었다. 일반적인 3차원 시간 영역 파동 전파 모델링 알고리즘에 Numba를 들여오고 Numba 장식자를 추가해주기만 하면 사용이 가능하다. 본 연구에

서는 파이썬을 사용하여 작성한 3차원 시간 영역 파동 전파 모델링 코드에 각 함수마다 @jit 장식자 한 줄만 추가하여 코드를 수정하였고 계산 시간 및 생산성을 측정하였다.

Table 2 The Python algorithm of 3D wave propagation modeling using Numba decorator in time domain.

Algorithm using Numba decorator

```
From numba import jit

@jit
Variable initialize
For i in range(time)
    Solve finite-difference equation
    Add the inject source
    Apply the boundary condition
    Apply time marching
Return output
```

```
import numpy as np
from numba import jit

@jit
def time3d(vel,dt,h,nt,nx,ny,nz,ne,srcx,srcy,srcz,dimx,dimy,dimz,dimxy):

    uo=np.zeros(dimxyz,dtype=np.float32)
    um=np.zeros(dimxyz,dtype=np.float32)
    up=np.zeros(dimxyz,dtype=np.float32)

    w=np.zeros(dimxyz,dtype=np.float32)

    seismo=np.zeros((nt,nx),dtype=np.float32)
```

Fig. 1. Example code of Python using Numba.

3.4.2 Cython 적용

본 연구에서는 파이썬의 계산 효율을 증대시키기 위한 또 다른 방법으로 Cython을 이용하였다. Cython은 CPython 모듈을 간단하게 생성할 수 있도록 만들어진 컴파일 언어이다. 파이썬 문법을 기반으로 만들어진 Cython은 C/C++ 함수를 불러와 외부 함수 인터페이스와의 실행 속도를 증가시키기 위해 정적 자료형이 지정 등이 추가된 형태를 가지고 있다 (Wikipedia, 2017). Cython은 파이썬 스크립트를 C언어로 컴파일할 수 있으며, 타입도 정적형 자료로 선언할 수 있는 기능도 제공한다. 또 다른 특징으로는 C언어로 작성된 함수를 파이썬 문법에서 부를 수 있다. C언어로 작성된 함수를 불러올 수 있는 여러 방법들이 존재하지만, Cython은 부가 코드를 따로 작성할 필요 없이 파이썬에서 바로 호출이 가능하다. 이러한 특징으로 인해 Cython은 생산성이 높은 파이썬의 장점을 유지하면서 외부 함수인 C 라이브러리와 쉽게 연동할 수 있으며, 실행 속도를 향상시킬 수 있다 (Wikipedia, 2017).

Cython은 Numba와 마찬가지로 파이썬 스크립트에서 Cython 함수를 들여와 사용할 수 있다. Cython은 기본적으로 파이썬 문법과 동일하게 사용할 수 있으며 속도 향상을 이루기 위해서는 자료형 선언을 해주어야 한다. 자료형 선언은 'cdef int i,j'와 같이 정의할 수 있다. Cython은 간단한 자료형 선언 및 간단한 명령어 추가를 통해 파이썬에 비해 월등한 속도 향상을 얻을 수 있다 (Lanaro, 2013).

Table 3에 Cython을 이용한 3차원 시간 영역 파동 전파 모델링 알고리즘을 나타내었으며, Fig. 2에 Cython을 적용한 파이썬 코드의 예시를 나타내었다. 일반적인 3차원 시간 영역 파동 전파 모델링 알고리즘에 Cython을 들여와 사용할 수 있다. 하지만 속도 향상을 이루기 위해서는 자료형 선언

이 필요하다. 본 연구에서는 파이썬으로 작성한 3차원 시간 영역 파동 전파 모델링 코드를 Cython코드에 맞게 코드 수정 및 자료형 선언을 하였고, 계산 시간 및 생산성을 측정하였다.

Table 3 The Python algorithm of 3D wave propagation modeling using Cython in time domain.

Algorithm using Cython

Import cython
cimport cython
 	
Variable initialize and Data type declaration
For i in range(time)
  Solve finite-difference equation
  Add the inject source
  Apply the boundary condition
  Apply time marching
Return output

```
import cython
%load_ext cython

import pyximport; pyximport.install()

%%cython --annotate
import numpy as np
cimport numpy as np
cimport cython

@cython.boundscheck(False)
@cython.wraparound(False)
@cython.nonecheck(False)
def time3d(np.ndarray[float,ndim=3] vel,float dt,float h,int nx,int ny,int nz,int ne,\
           int srcx,int srcy,int srcz,np.ndarray[float,ndim=1] w,int dimx,int dimy,int dimz):
    cdef int it,ix
    cdef int nt=int(len(w)/2)

    cdef np.ndarray uo=np.zeros((dimz,dimy,dimx),dtype=np.float32)
    cdef np.ndarray um=np.zeros((dimz,dimy,dimx),dtype=np.float32)
    cdef np.ndarray up=np.zeros((dimz,dimy,dimx),dtype=np.float32)
    cdef np.ndarray seismo=np.zeros((nt,nx),dtype=np.float32)
```

Fig. 2. Example code of Python using Cython.

4. 수치예제

4.1 모델 크기에 따른 순수 파이썬과 고성능 파이썬 성능 비교

본 연구에서는 C언어, 파이썬, 고성능 파이썬의 라이브러리를 사용하여 작성한 3차원 시간영역 파동 전파 모델링을 이용하여 각 언어의 성능 및 생산성을 비교하였다. P파 속도 모델은 반무한 매질이라고 가정한 후 2km/s의 상속도 모델을 적용하여 1ms 간격으로 총 3초 간 계산하였다. 경계 조건으로는 음향 파동 방정식을 들어오는 파와 나가는 파로 분리함으로써 전파방향에 따라 평면파의 반사를 제거할 수 있는 경계 조건인 Keys 투과 경계 조건을 적용하였다 (Keys, 1985).

먼저 순수 파이썬을 이용하여 구현한 3차원 시간 영역 파동 전파 모델링과 고성능 파이썬을 이용하여 구현한 3차원 시간 영역 파동 전파 모델링을 이용하여 모델 크기에 따른 성능을 비교하였다. 성능은 모델링의 계산 속도를 통해 비교하였다. 모델링을 수행하기 위해 Intel(R) Xeon(R) CPU E5-2670 v2 2.50GHz CPU 모델을 사용하였다. 모델링에 사용된 CPU는 CentOS 6.7 버전 및 256GB의 메모리를 지원한다. 순수 파이썬은 고성능 파이썬과 C언어에 비해 계산 속도가 월등히 느려 모델 크기를 50x50x50부터 50간격으로 150x150x150까지 크기를 늘려가며 계산하였으며, 2차 유한 차분식을 적용하였다. 계산 시간은 3번의 반복 실험 후 평균 시간으로 정의하였다. 순수 파이썬의 성능과 고성능 파이썬의 성능을 비교한 결과는 Fig. 3에 그래프로 나타내었고, 추가로 C언어를 사용하여 얻은 결과도 함께 나타내었다. 결과를 통해 알 수 있듯이 C언어가 파이썬에 비해 최대 1972배로 가장 뛰어난 성능을 나타낸다. 또한 Numba와 Cython 라이브러리를 이용한 고성능 파이썬들도 순수 파이썬에 비해 각각 최대 1775배,

1875배 이상 속도 향상을 이룰 수 있다는 것을 확인할 수 있었다. 따라서 고성능 파이썬을 사용하게 되면 순수 파이썬보다 월등한 속도 향상을 이룰 수 있으며 또한 C언어와 유사한 속도 향상을 얻을 수 있다.

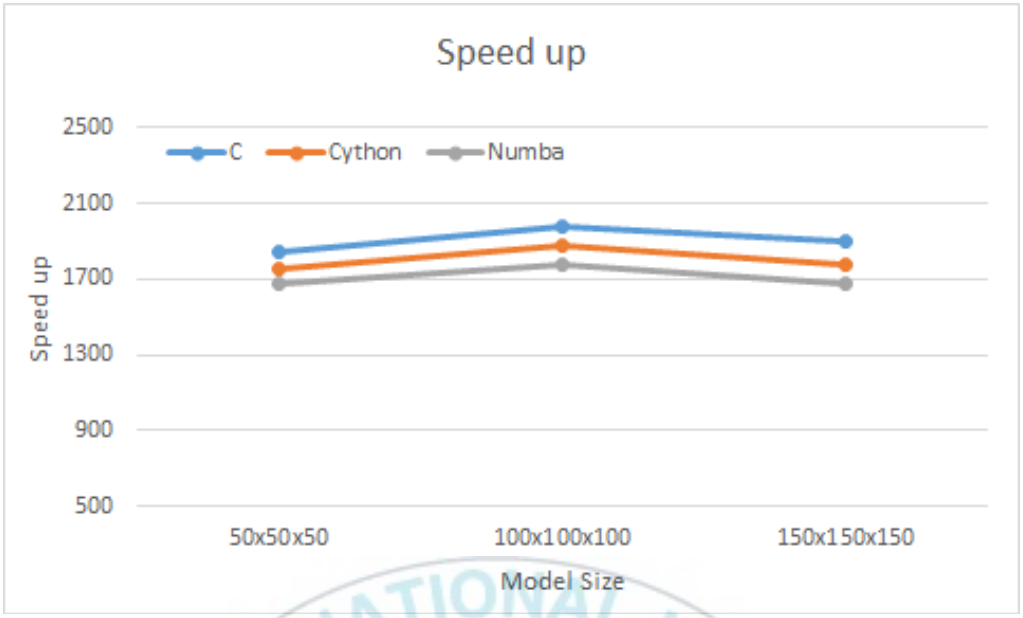

Fig. 3. In contrast to original Python, a comparison of each libraries speed up.

4.2 모델 크기에 따른 고성능 파이썬과 C언어의 성능 비교

본 연구에서는 순수 파이썬과 고성능 파이썬의 성능을 비교한 후, 모델 크기에 따라 고성능 파이썬과 C언어의 성능 및 생산성을 3차원 시간 영역 파동 전파 모델링을 이용하여 비교하였다. 성능은 계산 속도를 통해 비교하였고, 생산성은 코드의 줄 수를 통해 비교하였다 (Prechelt, 2000). 앞선 연구와 동일하게 P파 속도 모델은 반무한 매질이라고 가정한 후 2km/s의 상속도 모델을 적용하여 1ms 간격으로 총 3초 간 계산하였다.

고성능 파이썬과 C언어를 모델 크기에 따른 성능을 비교하기 위해 모델 크기를 100x100x100부터 100 간격으로 700x700x700까지 모델 크기를 증가시켜가며 모델링을 3번 반복 수행하였으며, 8차 유한 차분식을 적용하였다. 유한 차분식의 차수가 증가하게 되면 계산해야 되는 격자점의 수가 증가하게 된다. 따라서 총 모델링 계산 시간이 증가하지만 모델링의 정확도 또한 증가하게 된다. 계산 시간은 모델링을 3번 수행한 후 평균 계산 시간으로 정의하였다. 고성능 파이썬과 C언어의 성능을 비교한 결과는 Fig. 4에 나타내었다. 정적 자료형을 지원하는 C언어가 가장 성능이 뛰어나다는 것을 확인할 수 있었다. 고성능 파이썬인 Numba와 Cython은 C언어에 비해 각각 약 18.9%, 9.4% 계산 속도가 느렸다.

다음으로 고성능 파이썬과 C언어의 생산성을 비교하였다. 생산성은 코드 줄 수로 비교하였고, 이는 Table 4에 나타내었다. 먼저 순수 파이썬은 3차원 시간 영역 파동 전파 모델링 코드를 작성하는데 총 95줄을 소모하였으며, 고성능 파이썬인 Numba는 import문과 데코레이터만 추가하면 되므로 모델링을 작성하는데 총 97줄, 또 다른 고성능 파이썬인 Cython은 총 112줄을 소모하였다. 마지막으로 정적형 자료를 지원하는 C언어는 모델링 코

드를 작성하는데 총 134줄을 소모하였다. 코드 줄 수를 통해 생산성을 비교해본 결과, 고성능 파이썬인 Numba와 Cython은 C 언어에 각각 38.1%, 19.6% 생산성이 높았다. 이 결과를 통해 고성능 파이썬은 C언어보다 계산 속도는 조금 느리지만 그에 대응할만한 속력을 보여주며 오히려 생산성은 더 높은 것을 확인할 수 있다. 따라서 약 10~20% 정도의 성능이 중요하지 않은 경우라면 생산성이 더 높고 코드 작성이 좀 더 수월한 고성능 파이썬을 사용하는 것이 유리할 것이다.

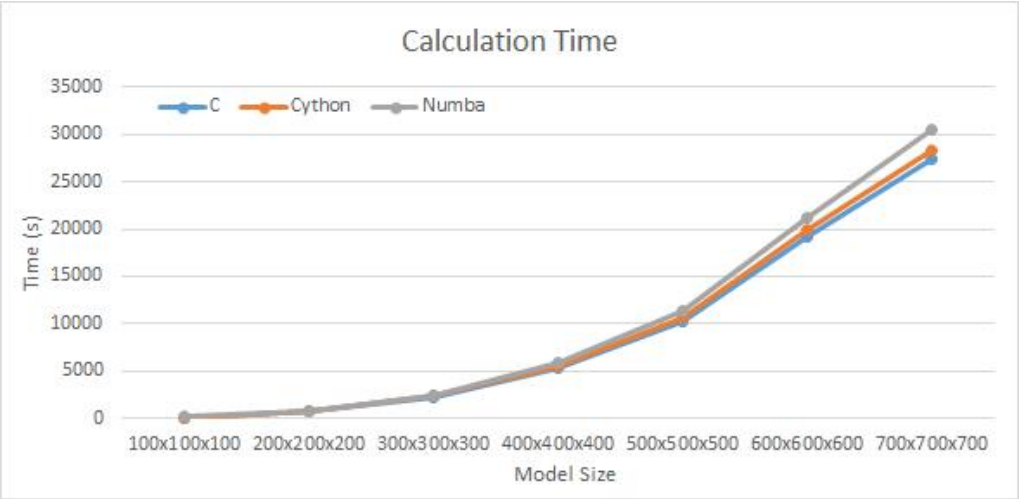

Fig. 4. Comparison of calculation time according to model size when 8^{th} order.

Table 4 Code lines of the program.

	Code line
Python	95
Numba	97
Cython	112
C	134

4.4 정확성 비교

마지막으로 Numba와 Cython, C언어에서 얻어진 계산 결과의 정확성을 트레이스를 추출하여 비교하였다. Fig. 5는 Numba와 C언어, Cython과 C언어의 트레이스를 1km 부근에서 추출하여 깊이 단면에 따라 정확도를 비교한 결과를 보여주며, 이는 모델링 1초가 지났을 때의 결과이다. Fig. 6은 각각 Numba와 C언어, Cython과 C언어의 트레이스를 3km 부근에서 추출하여 깊이 단면에 따라 비교한 결과를 보여준다. Fig. 6은 모델링 3초가 지난 후의 결과이다. 추출한 트레이스를 비교해보면 미세한 차이가 존재하지만 파동장이 오차범위 안에서 거의 일치하는 것을 확인할 수 있다.

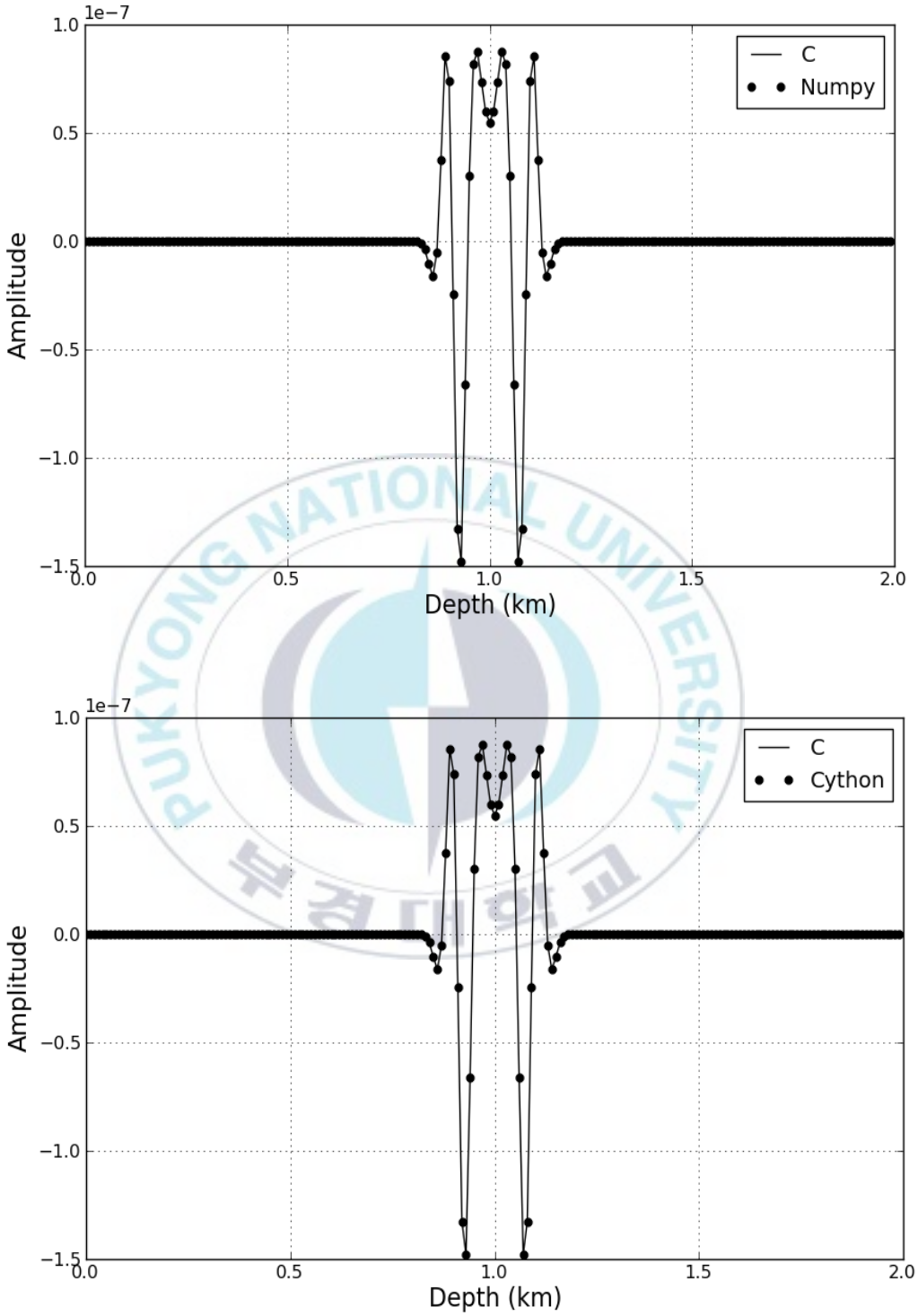

Fig. 5. (a) Profiles from the traces between C and Cython at 1km and (b) Profiles from the traces between C and Numba at lkm.

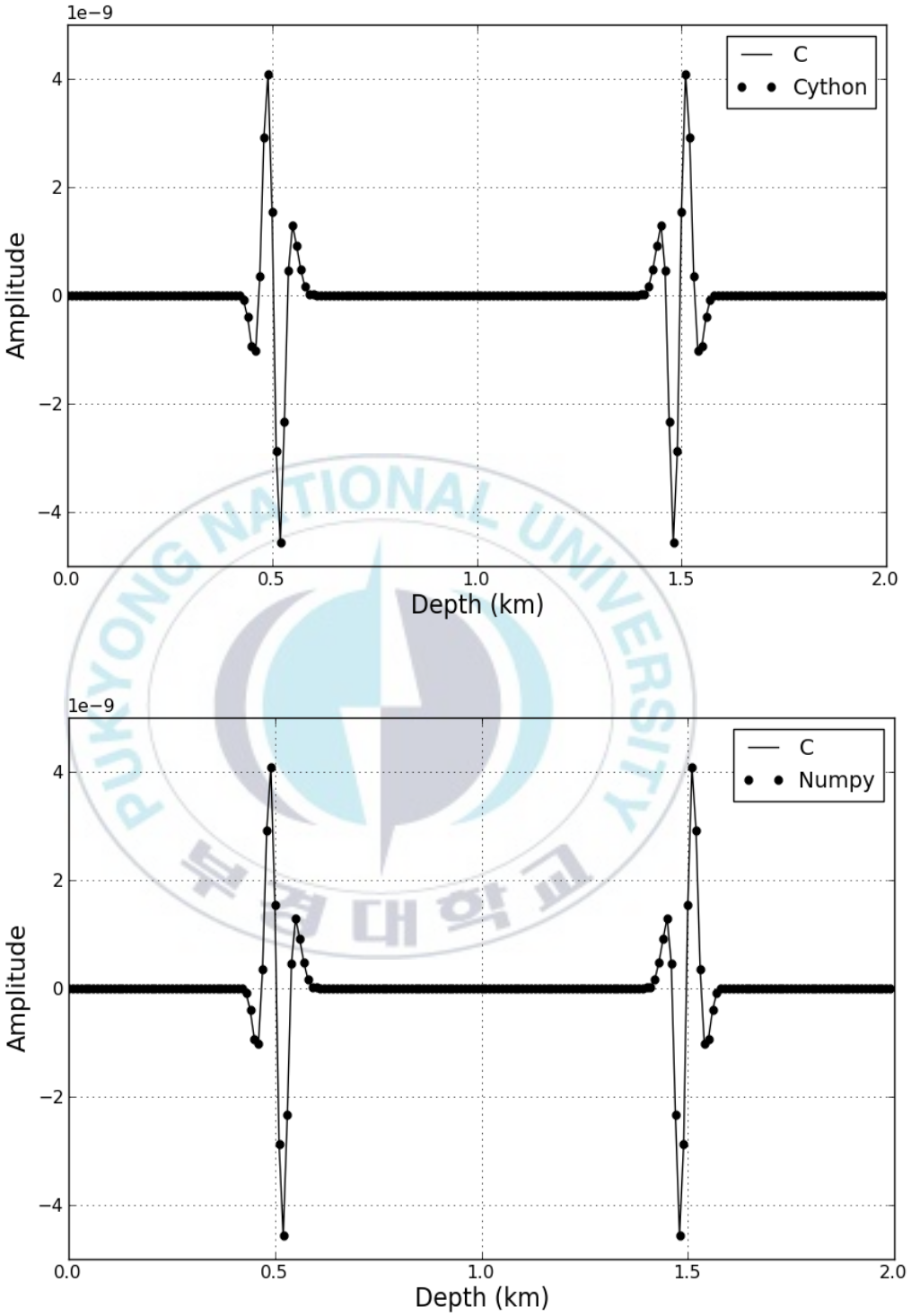

Fig. 6. (a) Profiles from the traces between C and Cython at 3km and (b) Profiles from the traces between C and Numba at 3km.

5. 토 의 및 결 론

본 연구에서는 고성능 파이썬 및 C언어를 사용하여 3차원 시간 영역 파동 전파 모델링을 구현하였고, 계산 시간 및 코드 줄 수를 통해 성능 및 생산성을 비교하였다. 계산 시간은 3번의 반복 실험 후 평균 시간으로 정의하였다. 3차원 시간 영역 파동 전파 모델링은 1ms 간격으로 총 3초간 수행하였고, P파 속도 모델은 반무한 매질의 2km/s인 상속도 모델을 사용하였다.

실험 방법으로는 먼저 순수 파이썬과 고성능 파이썬을 이용하여 구현한 3차원 시간 영역 파동 전파 모델링을 이용하여 성능을 비교하였다. 모델링을 수행하기 위해 Intel(R) Xeon(R) CPU E5-2670 v2 2.50GHz CPU 모델을 사용하였다. 순수 파이썬과 고성능 파이썬은 모델 크기를 50x50x50부터 150x 150x150까지 50간격으로 모델 크기를 늘려가며 계산하였으며, 2차 유한 차분식을 사용하였다. 실험 결과 고성능 파이썬인 Numba와 Cython, C언어는 순수 파이썬에 비해 각각 1775배, 1875배, 1972배 계산 속도가 향상되었다.

다음으로 고성능 파이썬과 C언어의 성능 및 생산성을 비교하였다. 성능 및 생산성은 계산 속도와 코드 줄 수를 이용하여 비교하였다. 모델링은 모델 크기를 100x100x100부터 700x700x700까지 100 간격으로 증가시켜가며 수행하였으며, 8차 유한 차분식을 적용하였다. 유한 차분식의 차수가 증가할수록 계산 시간은 증가하지만 모델링의 정확도도 향상된다. 실험 결과, 고성능 파이썬인 Numba와 Cython은 C언어에 비해 각각 약 18.9%, 9.4% 속도가 느렸지만, 코드 줄 수를 이용하여 비교한 생산성은 Numba와 Cython이 C언어보다 각각 약 38.1%, 19.6% 높았다. 따라서 약 10~20% 정도의 속도 향상이 크게 중요하지 않는 경우에는 생산성이 뛰어난 고성능

파이썬이 더 유리할 것이다. 그러므로 생산성이 뛰어난 고성능 파이썬은 정적 자료형 언어의 적절한 대안이 될 수 있을 것으로 기대된다. 추후 연구에는 역시간 구조보정이나 파형 역산 등에 적용해 볼 필요가 있으며, 고성능 컴퓨팅 분야 및 수치모델링 분야와 같은 여러 분야에 적절히 사용할 수 있을 것이라 예상된다.

참고문헌

Beazley, D.M., 2009, Python essential reference, 4th ed., Deceloper's Library, pp. 5-24.

Cython, 2017.9.25, http://www.cython.org.

Ha, W., 2014, Productive Small-scale Data Processing using Python, Ksmer, Vol. 51, No. 5, pp. 705-714.

IPython, 2017.9.25, http://www.ipython.org.

Keys, R.G., 7985, Absorbing boundary conditions for acoustic media, GEOPHYSICS, Vol. 50, No. 6, pp. 892-902.

Kelly, K.R., Ward, R.W, Treitel S. and Alford, R.M., 1976, SYNTHETIC SEISMOGRAMS: A FINITE-DIFFERENCE APPROACH, GEOPHYSICS, Vol. 41, No. 1, pp. 2-27.

Kim, A., Lee, J. and Ha, W., 2017, Three-dimensional Wave Propagation Modeling using OpenACC and GPU, Kseg, Vol. 20, No. 2, pp. 72-77.

Lanaro, G., 2013, Python high performance programming, Packt Publishing, pp. 49-70.

LLVM, 2017.9.25, http://llvm.org

McKinney, W., 2012, Python for data analysis, O'Reily, pp. 3-6

Micha, G. and Ian, Ozvald., 2016, High Performance Python, Hanbitmedia, Seoul, p. 424.

Min, D.-J., Pyun, S., Ha, W., Kwak, S., Jeong, W. and Shin, C., 2016, Geophysical numerical analysis, Cir Publishing Seoul, p. 270.

Numba, 2017.9.25, http:// numpy.org.

Numpy, 2017.9.25, http://www.numpy.org.

Prechelt, L., 2000, “An empirical comparison of seven programming languages” IEEE Computer, Vol. 33, No. 10, pp. 23-29.

Press, W.H., Teukolsky, S.A, Vetterling, W.T., Flannery, B.P. and Metcalf, M., 1996, Numerical Recipes in Fortran 90: Vol.2, Cambridge University Press, p. 1.

PyPy, 2017.9.25., http://pypy.org.

Python, 2017.9.25, http://www.python.org.

Scipy, 2017.9.25., http://www.scipy.org.

Ryu, D., Kim, A., and Ha, W., 2015, Expanding Domain Method for 3D Time-Laplace-Domain Hybrid Modeling, Geosystem Engineering, Vol 18, No5, 259-265.

The Python Language Reference, 2017.9.25, http://docs.python.org

Wikipedia, 2017.9.25, https://ko.wikipedia.org/wiki/%EC%82%AC%EC%9D%B4%EC%8D%AC%

Wikipedia, 2017.9.25, https://ko.wikipedia.org/wiki/%EC%9E%90%EB%A3 %8C%ED%98%95_%EC%B2%B4%EA%B3%84

Abstract

3D seismic data processing such as full waveform inversion or RTM requires large amount of numerical calculations. In this study, we compared performance and productivity of high-performance Python and C using 3D acoustic wave propagation modeling in time domain. We used Numba and Cython for high-performance Python. We conducted numerical modeling with changing model sizes. As a result of numerical modeling, Numba, Cython and C obtained respectively about 1700 times, 1800 times, 1900 times faster than original Python. And Numba and Cython of optimized high-performance Python is respectively about 18.9% and 9.4% slower than C. However, Productivity comparing code lines are 38.1% and 19.6% higher than C. If performance of 10~20% is not important, it is more efficient to use optimized high-performance Python than C. Therefore, optimized high-performance Python is expected to be sufficiently avaliable in the field of large-scale data processing, high-performance computing and numerical modeling. And optimized high-performance Python can be a productive alternative to the static programming languages like C and Fortran. We will apply to waveform inversion and reverse-time-migration in the future.

Keywords :3D, Wave Propagation Modeling, High-performance Python

Student Number : 201655282