한국자원공학회지 J. Korean Soc. Miner. Energy Resour. Eng.
Vol. 51, No. 5 (2014) pp. 705-714, http://dx.doi.org/10.12972/ksmer.2014.51.5.705
ISSN 2288-0291(print)
ISSN 2288-2790(online)

기술보고

파이썬을 이용한 생산적인 소규모 자료처리

하완수[1)]*

Productive Small-scale Data Processing using Python

Wansoo Ha*

(*Received 29 July 2014; Final version Received 18 September 2014; Accepted 16 October 2014*)

Abstract : Python is an object oriented programming language, similar to Perl and Ruby. It supports a dynamic typing and operates using a python interpreter. Static typing languages such as Fortran and C emphasize efficient execution program. On the other hand, Python's design philosophy values productivity of programmers. In general, a Python version of an algorithm is shorter and slower than its Fortran or C versions. Recent development of numerical libraries shortened the runtime of Python numerical programs and many scientists and engineers adopt Python as their programming language. In this study, I compared Python and Fortran programming languages by developing functions of simple algorithms used in seismic data processing. I tried to make Python programs efficient by exploiting numerical libraries. Benchmarking examples show that small-scale data processing programming using Python and Numpy library can be more efficient than that using Fortran language.

Key words : Data processing, Python, Productivity, Efficiency

요 약 : 파이썬은 펄, 루비 등과 같은 객체지향 프로그래밍 언어로, 해석기를 통해 작동하며 동적 자료형을 지원한다. 계산 속도를 중시하여 정적 자료형을 이용하는 포트란 프로그래밍에서와 달리 파이썬 프로그래밍에서는 개발자의 생산성을 더 중시한다. 일반적인 파이썬 프로그램의 경우 동일한 기능의 포트란 프로그램에 비해 개발 시간은 짧지만 실행 시간은 길다. 최근 수치 계산 라이브러리들이 발전되면서 파이썬 프로그램의 실행 시간이 짧아지고 있고, 과학 및 공학 계산에 파이썬을 이용하는 사례가 증가하고 있다. 본 연구에서는 탄성파 자료처리에 사용되는 간단한 알고리즘들을 대상으로 수치 라이브러리들을 통해 파이썬 프로그램의 실행 속도를 향상시키는 방법을 살펴본다. 예제들을 통해 프로그램 작성 시간을 고려하면 소규모 자료처리의 경우 포트란 대신 파이썬과 Numpy 라이브러리를 이용하는 것이 더 효율적임을 알 수 있다.

주요어 : 자료처리, 파이썬, 생산성, 효율성

서 론

수치해석 프로그래밍 분야에서 계산의 정확도가 같거나 유사하다면 그 다음으로는 보통 프로그램의 실행 속도에 관심을 갖는다. 따라서 수치해석 분야에서는 일반적으로 포트란, C/C++이나 자바와 같은 컴파일 언어를 선호하고, 각종 수치해석 라이브러리들도 이러한 언어들로 많이 개발되어 있다(Press *et al.,* 1996). 탄성파 자료처리 분야에서도 파동 전파 모델링, 역시간 구조보정이나 완전파형역산과 같이 상당히 많은 계산이 필요하고 계산에 수 일 또는 수 주가 걸리는 분야가 많기 때문에 위와 같이 계산이 효율적인 프로그래밍 언어를 많이 사용한다. 오픈소스 탄성파 자료처리 패키지인 Seismic Unix나 Madagascar에서도 대부분의 프로그램이 C 또는 포트란 언어로 작성되어 있다(Seismic Unix, 2014; Madagascar, 2014).

그러나 모든 탄성파 자료처리 분야에서 계산량이 많은 것은 아니다. 직접 프로그래밍을 하며 연구를 하다 보면 속도모델을 수정한다거나, 공통송신원모음 몇 개를 수정하는 등 특정 자료에만 사용할 수 있는 자료처리 프로그램들도 작성하게 된다. 이러한 자료처리의 경우 또는 자료처리 결과 분석시에는 비교적 간단한 알고리즘을 사용하게 되고, 프로그램의 실행 횟수도 적으며, 실행 시간도 초 단위 또는 그 이하로 짧아지게 된다. 기존의 프로그램

1) 부경대학교 에너지자원공학과
*Corresponding Author(하완수)
E-mail; wansooha@pknu.ac.kr
Address; Dept. of Energy Resources Engineering, Pukyong National University, Busan, Korea

에서 지원하지 않는 새로운 자료처리 알고리즘을 개발하는 경우에도 프로그램을 빈번히 수정하며 실행하는 경우가 생기게 된다. 이때에는 프로그램의 실행 시간보다 개발 시간, 즉 개발자의 생산성이 중요해진다(Weinberg, 1998). 대규모 처리에 비해 소규모 자료 처리의 최적화에 큰 노력을 기울일 필요가 없기 때문에 프로그램을 빠르게 작성할 수 있는 프로그래밍 언어를 사용하는 것이 바람직하다고 생각할 수도 있다. 예를 들면, 10회 정도 사용할 프로그램을 작성할 때 다른 프로그래밍 언어를 사용하여 계산 시간이 1초에서 10초로 길어지더라도 프로그램 개발 시간이 10분에서 5분으로 단축된다면 다른 언어를 사용할 가치가 있는 것이다. 수치해석 연구자의 생산성을 높이기 위한 언어로는 매트랩(Matlab, 2014), 줄리아(Julia Programming Language, 2014), 통계 분석을 위한 R(The R project, 2014)과 같은 언어들이 있다. 본 연구에서는 최근 수치해석 분야에서 사용자가 많아지고 있는 파이썬 언어를 대상으로, 소규모 자료처리에 효율적으로 사용하기 위한 방법을 살펴본다.

파이썬 언어는 해석기를 통해 작동하는 범용 오픈소스 프로그래밍 언어로, 객체지향 프로그래밍과 동적 자료형을 지원한다(Beazley, 2009). 다양한 운영체제에서 사용가능하고, 해석기를 통해 작동하기 때문에 별도의 컴파일 과정이 필요 없다. 범용 언어로 수치 해석 외에도 사무용 프로그래밍, 웹프로그래밍, 게임 등 다양한 분야에 사용된다. 조사 방법에 따라 차이가 있지만, 2014년 가장 많이 사용되는 프로그래밍 언어 4위에 오르기도 했다(IEEE Spectrum, 2014). 수치해석과 관련해서는 생명공학, 통계 및 기계 학습 분야, 경제학을 포함한 사회 과학 분야에서 먼저 주목을 받기 시작했다(Janert, 2010). 오픈소스 언어로, 사용자는 원할 경우 파이썬 프로그래밍 언어의 소스코드에 쉽게 접근할 수 있다. 언어 설계시부터 간결함을 중시하며 언어 문법에서 들여쓰기를 강제한다. 파이썬은 C나 포트란 코드와의 통합이 쉽기 때문에 기존의 빠른 코드들을 붙이기 위한 풀언어로도 자주 사용되고, 알고리즘 테스트를 위한 프로토타이핑에 많이 사용된다. 자체적으로 정규표현식을 지원하기 때문에 문자열을 쉽게 다룰 수 있다. 이 외에도 다양한 특징들이 있으나 본 연구에서 다루고자 하는 생산성 및 효율성의 관점에서 본다면 동적 자료형 지원이 가장 중요한 특징 중 하나라 할 수 있다.

동적 자료형은 정적 자료형과 대비되는 개념으로, 정적 자료형을 지원하는 언어에서는 컴파일시 정수, 실수와 같은 변수의 자료형을 검사하지만, 동적 자료형을 지원하는 언어에서는 프로그램 실행시 자료형을 검사한다. 정적 자료형 언어에서는 하나의 변수가 하나의 자료형만 가지게 되지만, 동적 자료형 언어에서는 하나의 변수가 여러 가지 자료형으로 사용될 수 있다. 포트란이나 C와 같은 정적 자료형 언어와 비교했을 때 동적 자료형을 지원하는 언어에서는 정수, 실수 등의 변수 선언부가 없고, 하나의 함수로 여러 가지 자료형의 인자를 지원하는 다형성을 구현하기가 용이하다. 따라서 동일한 프로그램 작성시 일반적으로 정적 자료형 언어보다 적은 양의 코드로 빠르게 작성할 수 있다(Beazley, 2009). 반면에, 프로그램의 실행 속도는 정적 자료형 언어에 비해 느려지게 된다. 이는 변수에 값을 할당할 때 내부적으로 파이썬 객체 번호와 자료형 및 변수값 정보를 저장하며, 연산시 변수 내에 저장된 자료형을 바탕으로 알맞는 연산 함수를 찾아야 하기 때문이다. 또한 파이썬 리스트를 이용하여 여러 개의 변수를 저장할 경우, 각각의 변수 내에 객체의 번호와 자료형 정보를 가지고 있기 때문에 변수의 값이 메모리상에 연속적으로 분포하지 않게 된다는 문제도 있다. 리스트 내의 값들을 참조하기 위하여 반복문을 사용할 경우, 인덱스의 값이 리스트의 범위를 넘어가는지 판단하는 과정 및 각종 예외 처리 등의 과정도 빠른 연산에 방해가 된다(The Python Language Reference, 2014). 이에 더하여 포트란이나 C와 같은 컴파일 언어와 달리 프로그램을 한 줄씩 읽어 해석하는 해석기로 실행하기 때문에 실행 시간이 길어지게 된다(Beazley, 2009).

그러나 수년에 걸친 발전으로 최근 파이썬 언어의 실행 속도가 많이 빨라졌고, 과학 및 공학 분야에서도 파이썬 언어의 사용이 증가하고 있다(McKinney, 2012; Lanaro, 2013). 파이썬 해석기의 실행 속도가 빨라진 영향도 있지만(Python, 2014; PyPy, 2014), 그보다는 정적 자료형 언어를 이용하여 개발한 각종 파이썬 라이브러리들의 영향이 크다. 대표적인 것이 C언어로 개발된 파이썬 배열 연산 라이브러리인 Numpy이다. Numpy 자체는 C로 만들었지만, C를 몰라도 파이썬 문법을 통해 쉽게 사용할 수 있다. Numpy에서는 동일한 자료형의 값들이 메모리상에 연속적으로 위치하도록 배열을 생성하며, 벡터 연산을 지원한다(McKinney, 2012). Numba는 LLVM(LLVM, 2014) 컴파일러를 이용하여 파이썬 코드를 기계어로 실시간 컴파일하는 방식으로 파이썬 함수를 가속한다. 파이썬 함수를 수정할 필요 없이 간단히 빠른 함수를 만들 수 있다는 것이 Numba 라이브러리의 큰 장점이다(Numba, 2014). Cython은 파이썬 함수에서 사용하는 변수에 자료형을 지정해주면 자동으로 C 코드를 생성한 후 컴파일하여 파이썬에서 사용할 수 있도록 만들어주는 라이브러리이다. C언어를 직접 사용할 필요 없이 파이썬 코드와 일부 추가된 명령어만으로 실행 속도가 빠른 코드를 생성할 수 있다(Lanaro, 2013). 이 외에

도 캐시 최적화를 통해 Numpy 벡터 계산 속도를 향상시키는 Numexpr, C 코드를 파이썬 함수로 만들어주는 Ctypes나 Weave 라이브러리, 포트란 코드를 파이썬 함수로 만들어주는 F2py와 같은 다양한 라이브러리들이 개발되어 있다(Numexpr, 2014; Numpy, 2014; Python, 2014; SciPy, 2014). 실행 속도 향상 외에도 수치해석과 관련하여 각종 수치해석 계산 라이브러리를 포함한 SciPy, 과학적 가시화를 위한 Matplotlib 및 Mayavi, 대화식 개발을 위한 IPython 등 많은 파이썬 라이브러리들이 존재한다(McKinney, 2012; SciPy, 2014; Vaingast, 2009). 대규모 계산에서 파이썬 언어를 이용할 경우에는 계산 시간을 많이 차지하는 병목 부분만 C나 포트란으로 작성하여 파이썬에서 불러오는 방식으로 접근한다(Lanaro, 2013). 그러나 이렇게 두 개 이상의 언어를 사용하는 방식은 본 연구에서 다루고자 하는 소규모 자료처리에서는 오히려 개발 시간이 증가하게 되는 원인이 된다.

본 연구에서는 간단한 자료처리 프로그램 작성시의 생산성과 효율성을 비교한다. 효율성은 함수 실행 시간으로 비교할 수 있지만, 생산성은 정량화하기 어려우므로 프로그램 코드의 길이를 비교한다(Prechelt, 2000). 비교를 위해 파이썬 언어와 라이브러리들을 탄성파 탐사에서 사용하는 몇 가지 간단한 알고리즘에 적용하여 포트란90으로 작성한 경우와 실행 시간 및 코드 길이를 비교하였다. 포트란 언어는 최적화된 수치해석을 위해 개발된 언어로, 실행 속도가 빠르며 배열의 벡터 연산을 지원하여 C언어를 사용할 때보다 코드 길이를 줄이기에 용이하기 때문에 파이썬 언어의 비교 대상으로 삼았다. 연구에는 파이썬 수치 계산 분야에서 많이 사용되는 Numpy, Numba 및 Cython 라이브러리들을 사용하였다.

연구 방법

프로그램 비교시 각각의 라이브러리들을 사용하여 자료처리 함수를 작성하고 함수의 실행 시간을 비교하였다. 라이브러들의 비교를 위해 탄성파 자료 처리에서 사용하는 알고리즘들 중 비교적 간단한 표준화, 상수곱, 이득 제어, 트레이스 중합, 이동평균, 이산 푸리에 변환, 행렬곱 및 1차원 파동방정식 모델링 알고리즘을 사용하였다.

표준화 예제에서는 500×500 크기의 2차원 배열을 배열의 최대 절대값으로 나누어 새로운 2차원 배열을 만드는 프로그램을 작성하였다. 상수곱 문제에서는 5395×956 크기의 BP 속도모델(Billette and Brandsberg-Dahl, 2005)을 km/s 단위에서 m/s 단위로 바꾸는 프로그램을 작성하였다(Fig. 1). 이득 제어 예제에서는 샘플수가 10000, 샘플 간격이 1 ms인 779개의 트레이스에 지수함수를 곱하였다(Fig. 2). 중합 예제에서는 샘플수가 10001인 1000개의 트레이스를 중합하였다(Fig. 3). 이동 평균 문제에서는 Marmousi 속도모델(Versteeg, 1994)을 평활화하였으며(Fig. 4), 이산 푸리에 변환 예제에서는 샘플

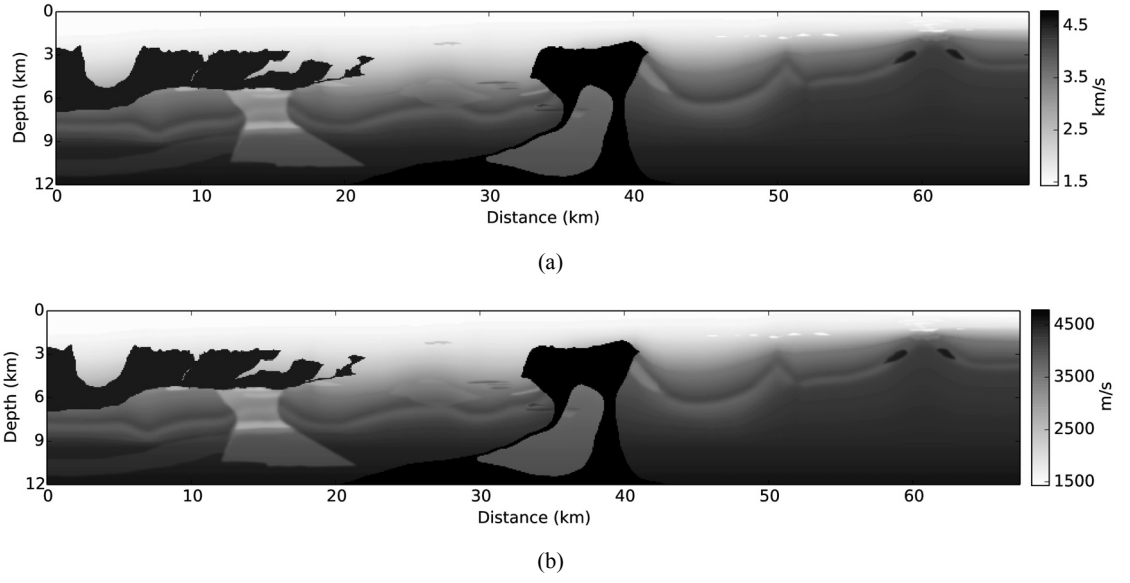
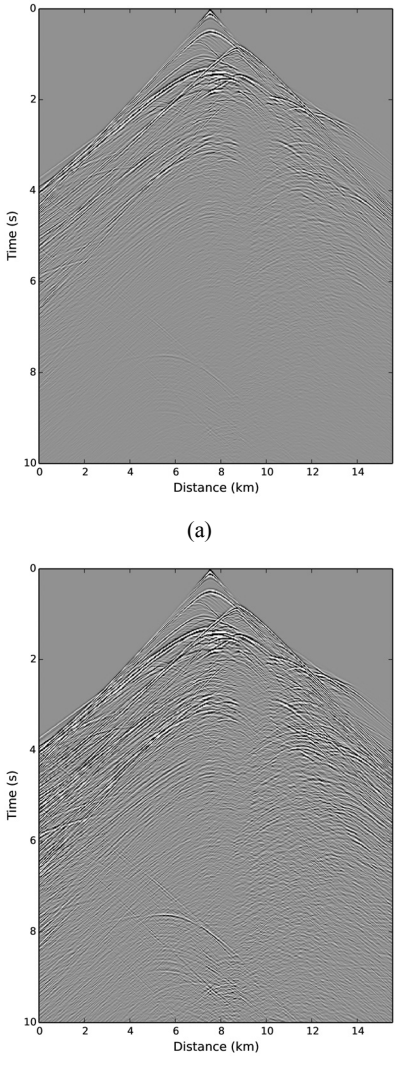


Fig. 1. (a) The BP velocity model (Billette and Brandsberg-Dahl, 2005) in km/s and (b) the velocity model in m/s.

수가 10000, 샘플 간격이 1 ms인 파형요소에 대해 100 Hz까지 주파수 성분 구하였다(Fig. 5). 행렬곱 예제에서는 500×500 크기의 행렬을 이용하여 행렬곱을 계산하였고, 1차원 파동방정식 모델링의 경우 길이가 25 km인 공간에서 5 m 간격의 격자를 만들고 1 ms 간격으로 5초간 모델링을 수행하였다(Fig. 6).

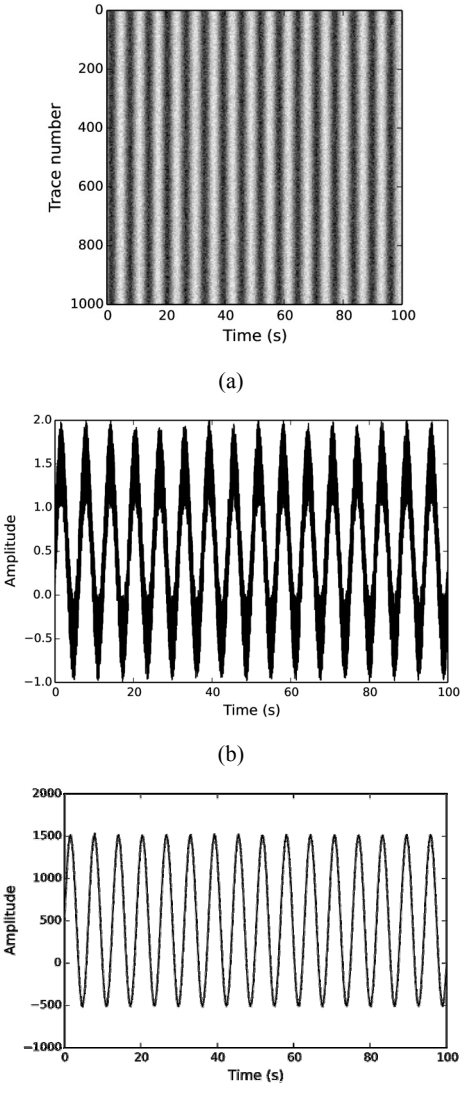
이중 대표적으로 2차원 배열을 최대 절대값으로 나누어주는 표준화 알고리즘의 코드를 Tables 1～5에 제시하였다. 오류 처리와 주석 등 실제 계산에 포함되지 않는 부분은 생략하였다.

파이썬 코드들에서 라이브러리 모듈을 읽어 들이는 부분은 함수의 줄 수에 포함시키지 않았다. 포트란 코드의 경우 정적 자료형 언어의 특성을 살리기 위해 “implicit none”과 intent문을 사용하였으나 “implicit none”은 서브루틴의 줄 수에 포함시키지 않았다. 또한 Cython이나 포트란 코드의 컴파일 시간은 실행 속도에 포함시키지 않았다. Numpy와 포트란 코드에서 반복문을 제거하는 벡터 연산이 가능할 때에는 벡터 연산을 이용하였고, 내장 함수가 있을 때에는 내장 함수를 이용하였다. 포트란



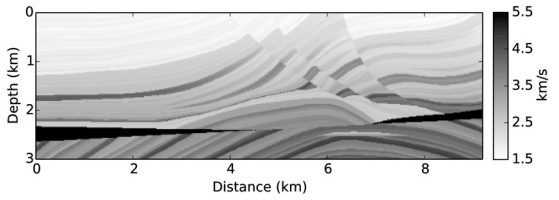
(a) (b)

Fig. 2. (a) A shot gather and (b) the shot gather after applying the gain function to it.

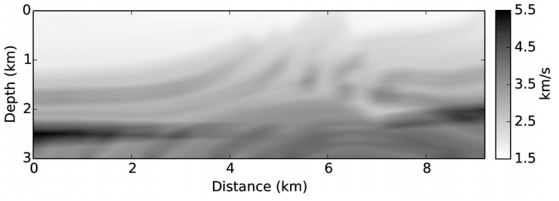


(a) (b) (c)

Fig. 3. (a) A gather with 1000 noisy traces, (b) the 500th trace extracted from the gather, and (c) the stacked trace.



(a)

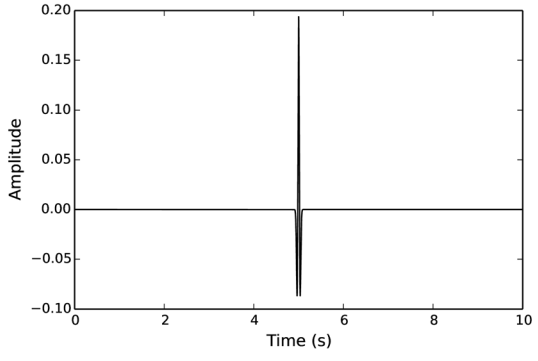


(b)

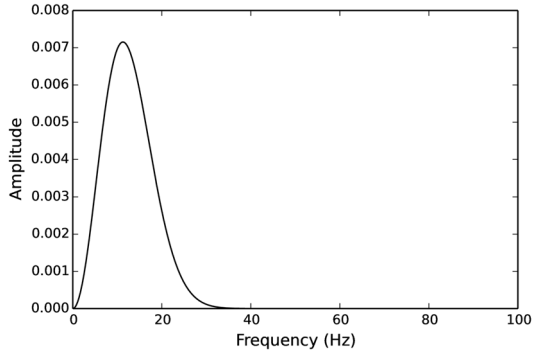
Fig. 4. (a) The Marmousi velocity model (Versteeg, 1994) and (b) the velocity model after smoothing.

의 경우에는 벡터 연산을 이용한 코드와 이용하지 않은 코드를 모두 작성하였고, 내장함수를 사용할 수 있을 때에는 내장함수를 사용한 코드와 사용하지 않은 코드를 모두 작성하여 각각의 실행시간 차이가 10% 이하일 때에는 짧은 코드를, 실행시간 차이가 10% 보다 클 때에는 빠른 코드를 선택하여 비교하였다.

함수의 실행 시간은 알고리즘에 따라 함수를 1～10 회 반복하는 루프를 만들고, 루프를 3～5 회 실행하여 가장 빠른 루프의 계산 시간을 반복 횟수로 나누어 구하였다. 함수는 모두 파이썬 메인 프로그램에서 호출하며 시간을 측정하였다. 포트란도 서브루틴만 별도로 컴파일한 후 파이썬에서 호출하는 방식으로 실행하였다. 파이썬은 버전 2.7.6, Numpy는 버전 1.8.1, Numba는 버전 0.13.1, Cython은 버전 0.20.1을 사용하였다. 포트란은 gfortran 버전 4.9.0으로 컴파일했으며, 최적화와 관련해서는 -O3 -funroll-loops 옵션을 사용하였다. 비교를 위해 Intel Core i5 3.2 GHz CPU와 Mac OS X 운영체제를 사용하였다.

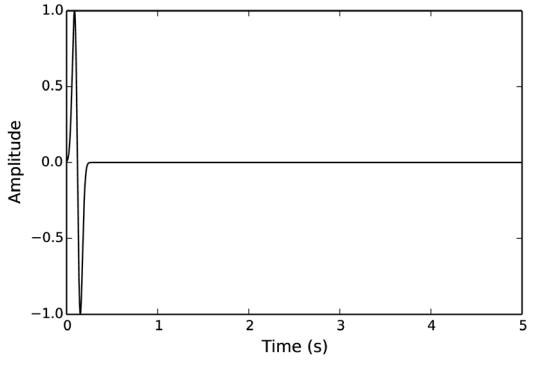


(a)

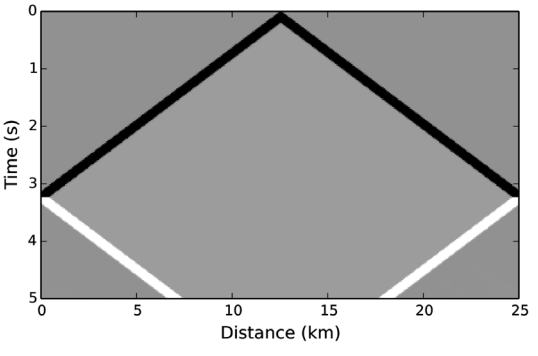


(b)

Fig. 5. (a) A time-domain ricker wavelet and (b) its frequency spectrum.



(a)



(b)

Fig. 6. (a) A first-derivative Gaussian source wavelet used for the modeling and (b) the resultant seismogram.

Table 1. A python code for the normalization without the Numpy vectorization

```
def normal_py(arr, n1, n2):
    mxabs=0.
    for i1 in xrange(n1):
        for i2 in xrange(n2):
            val=abs(arr[i1, i2])
            if val > mxabs: mxabs=val
    for i1 in xrange(n1):
        for i2 in xrange(n2):
            arr[i1, i2]/=mxabs
    return arr
```

Table 2. A python code for the normalization using the Numpy vectorization

```
def normal_np(arr):
    return arr/abs(arr).max()
```

Table 3. A Numba version of the normalization code using the python code in Table 1

```
normal_nb=autojit(normal_py)
```

Table 4. A Cython version of the normalization code

```
def normal_cy(np.ndarray[double,ndim=2] arr, int n1, int n2):
    cdef double val, mxabs=0.
    cdef int i1, i2
    for i1 in xrange(n1):
        for i2 in xrange(n2):
            val=abs(arr[i1, i2])
            if val > mxabs: mxabs=val
    for i1 in xrange(n1):
        for i2 in xrange(n2):
            arr[i1, i2]/=mxabs
    return arr
```

Table 5. A fortran version of the normalization code

```
subroutine normal_v(arr_out, arr, n1, n2)
integer,intent(in):: n1, n2
real(kind=8),intent(in):: arr(n1, n2)
real(kind=8),intent(out):: arr_out(n1, n2)
arr_out=arr/maxval(abs(arr))
end subroutine
```

결과 및 고찰

함수의 실행 시간을 Fig. 7에 나타내었다. Numpy 벡터 연산 없이 파이썬 루프를 사용하여 작성한 기본 함수에 대한 상대적인 가속 정도는 Fig. 8에 제시하였다.

8가지 알고리즘들 중 상대적으로 간단한 표준화, 상수곱, 이득 제어 및 중합에 대해 Numpy, Numba, Cython 및 포트란 프로그램을 사용할 경우 모두 기본 파이썬 프로그램에 비해 200～1000 배 이상에 이르는 속도 향상을 얻을 수 있음을 알 수 있다. 상대적으로 복잡한 이동평균, 이산 푸리에 변환, 행렬곱 및 1차원 파동 방정식 모델링에 대해서는 각각의 라이브러리 별로 편차가 크게 나타났다.

이차원 이동 평균법은 반복문 네 개가 중첩되어 Numpy

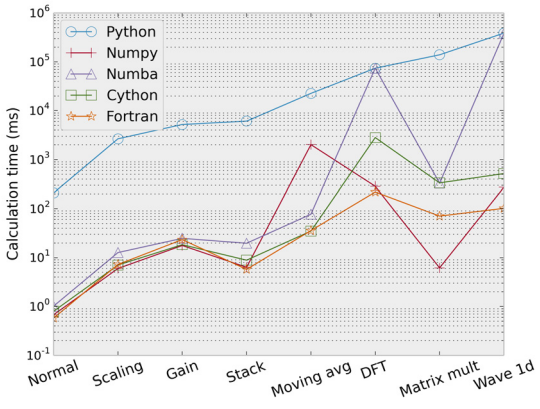


Fig. 7. Calculation times of each application with respect to the processing algorithms.

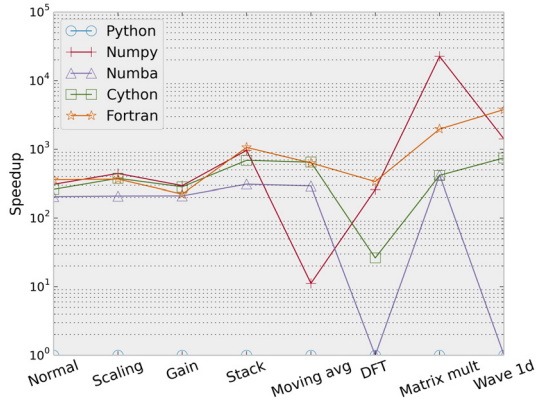


Fig. 8. Speedups of each application with respect to the processing algorithms.

나 포트란에서 벡터 연산을 사용하는데 한계가 있다. 이로 인해 Numpy 프로그램의 속도 향상이 11 배로 감소하였다. 포트란 프로그램의 경우 컴파일러 최적화로 인해 벡터 연산을 사용한 경우와 사용하지 않은 경우의 시간차가 크지 않았다. Numba와 Cython 프로그램에서는 파이썬 반복문을 기계어 및 C 언어의 루프로 변환하기 때문에 290～650 배에 달하는 속도 향상을 얻을 수 있었다.

행렬곱 예제에서는 Numpy 프로그램이 포트란 프로그램보다 빠른 실행 결과를 나타냈다. 파이썬, Numba 및 Cython 프로그램에서는 기본적인 행렬 곱셈식을 사용하였으나(Kreyszig, 2011), Numpy와 포트란의 경우 내장 함수를 사용하였다. 따라서 내부적인 계산 알고리즘에 차이가 있다. Numpy의 행렬곱 함수는 실제로 LAPACK의 행렬곱 함수를 사용하고(Netlib, 2014), 그로 인해 포트란의 내장함수보다 빠른 결과를 얻을 수 있었다. Numpy의 20000 배, 포트란의 2000 배에는 미치지 못하지만 Numba와 Cython 함수도 400 배의 속도 향상을 보였다.

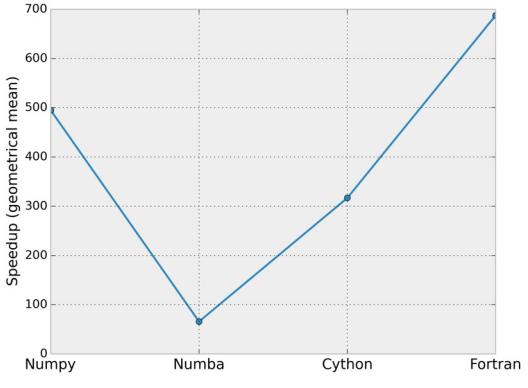


Fig. 9. Geometrical mean speedups of each application.

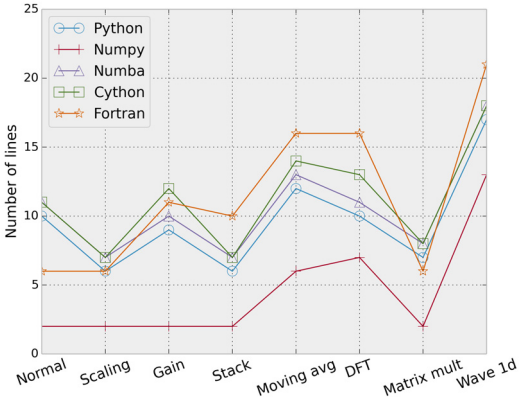


Fig. 10. Number of code lines in each application with respect to the processing algorithms.

이산 푸리에 변환과 1차원 파동 방정식 모델링의 경우 Numba 프로그램으로는 속도 향상을 얻을 수 없었다. 각각의 경우에 대해 포트란 프로그램의 실행 시간이 가장 짧았고, 그 다음이 Numpy, Cython 프로그램 순이었다.

여덟 가지 알고리즘에 대한 속도 향상 정도의 기하평균을 Fig. 9에 제시하였다(Fleming and Wallace, 1986). 평균적으로 포트란 프로그램에 비해 Numpy 프로그램은 1.4 배, Cython 프로그램은 2.2 배, Numba 프로그램은 10.4 배 느림을 알 수 있다.

프로그램의 실행 시간과 더불어 프로그램 작성시의 생산성 비교를 위해 프로그램의 길이를 비교하였다(Fig. 10). Numba는 파이썬 함수를 수정하는 방식으로 작동하기 때문에 파이썬 기본 프로그램보다 한 줄 길어지게 된다(Table 3). Cython의 경우도 파이썬 함수에 자료형을 추가하는 방식으로 작동하므로 파이썬 함수보다 길어진다(Table 4). Numpy의 경우 루프의 벡터 연산과 다양한 내장 함수들로 인해 간단한 알고리즘의 경우 두 줄로 작성이 가능했다(Table 2). 좀 더 복잡한 1차원 파동 방정식 모델링 알고리즘의 경우에도 기본 파이썬 프로그램보다 20% 이상 짧게 작성할 수 있었다. 포트란의 경우도 벡터 연산을 지원하고 여러 가지 내장 함수들이 있지만 자료형 선언부로 인해 줄 수가 길어지게 된다.

위의 결과들을 종합해볼 때, Numpy의 벡터 연산과 내장 함수들을 적극적으로 사용하는 것이 생산성 및 효율성 향상에 중요함을 알 수 있다. Numpy의 벡터 연산을 사용하면 내부적으로 파이썬 루프가 아닌 C 루프를 사용하게 되므로 속도가 향상된다. 또한 Numpy의 선형대수 관련 내장함수들은 BLAS(Basic Linear Algebra Subprograms), LAPACK(Linear Algebra Package)등의 라이브러리를 이용하기 때문에 간단하게 작성한 포트란 프로그램의 경우보다 빠르게 결과를 얻을 수도 있다(BLAS, 2014; Numpy, 2014; LAPACK, 2014). 전체적으로 Numpy 프로그램이 포트란 프로그램에 비해 1.4 배 느렸지만, 코드의 길이는 2.8 배 짧았다(Fig 10). 위의 예제들과 같이 간단한 자료처리의 경우 프로그램의 실행 시간보다 개발 시간이 훨씬 길어지게 되므로 파이썬과 Numpy를 사용하면 포트란 언어를 사용하는 것보다 빠르게 결과를 얻을 수 있다.

이동 평균 예제에서와 같이 함수 내의 루프 구조가 복잡해서 Numpy의 벡터 연산을 효율적으로 적용하기 곤란할 때에는 Numba나 Cython을 이용하는 것도 도움이 되었다. Numpy와 비교하지 않는다면, Numba와 Cython

라이브러리를 통해서도 상당한 속도 향상을 얻을 수 있음을 알 수 있다(Fig. 9). 특히 Cython의 경우 Numba와 달리 전체 알고리즘에서 비교적 고른 속도 향상을 나타냈다. 단, Cython의 경우 Numba와 비교했을 때 변수의 자료형을 직접 지정해주어야 하기 때문에 생산성이 떨어진다는 단점이 있다. Numba와 Cython 라이브러리 사용 시 Numpy 벡터 연산을 이용한 코드를 바탕으로 수정할 수도 있다. 그러나 그럴 경우 코드는 짧아지지만 파이썬 반복문을 빠르게 실행하도록 바꿔주는 라이브러리의 목적을 달성할 수 없게 되고, 테스트 결과 효율성도 떨어졌다. 비교에 사용한 Numba의 버전이 0.13.1, Cython의 버전은 0.20.1임을 볼 때 앞으로 라이브러리의 최적화를 통해 더 좋은 결과를 얻을 수 있을 것으로 예상된다.

처리하는 자료의 크기는 앞에서 살펴본 상대적인 속도에 크게 영향을 미치지 않는 것으로 나타났다(Fig. 11). 이득 제어의 경우 전체 트레이스에서 샘플 길이를 100, 1000, 10000으로 바꿔가며 처리하였고, 이산 푸리에 변환의 경우에도 샘플 길이를 100, 1000, 10000으로 바꿔가며 알고리즘을 적용하였다. 행렬곱의 경우 곱하는 두 행렬의 크기를 10×10, 100×100, 1000×1000으로 바꿔가며 계산하였다. 결과적으로 각각의 처리 알고리즘에 대해 자료의 크기에 따라 약간의 변동은 있으나 라이브러리별 상대적인 계산 시간은 앞에서 살펴본 그래프에서 크게 변하지 않았다(Fig. 7).

본 연구에서는 편의를 위해 포트란과 Cython의 컴파일 시간을 무시하였고, 프로그램의 줄 수와 생산성이 반비례한다고 가정하였다. 또한 사용자가 포트란과 파이썬 라이브러리들을 모두 사용할 줄 안다고 가정하였다. 컴파일 시간을 포함할 경우 별도의 컴파일이 필요 없는 파이썬, Numpy 및 Numba의 효율성이 상대적으로 크게 증가한다. 프로그램 개발 시간은 사용자의 숙련도와 밀접하게 관련된 것으로, 포트란에 익숙하고 파이썬에 초보인 사용자는 두 줄의 Numpy 벡터 연산 코드보다 여섯 줄의 포트란 코드를 더 빠르게 작성할 수도 있다(Table 2, 5). 프로그래밍의 생산성을 측정하는 방법으로는 코드의 줄 수를 세는 방법 외에도, 숙련된 프로그래머들이 코드를 작성하는 시간을 직접 측정하는 방법이 있다. 그러나, 프로그래밍은 사람의 행동이 개입되는 일로, 각 사람의 경험, 지능, 동기, 훈련, 환경 등에 영향을 받게 된다. 또한 불확정성 원리의 일종으로, 생산성을 측정하는 행위 자체가 관찰 대상의 생산성에 영향을 미치는 호손 효과가 발생한다(Weinberg, 1998). 이는 산업심리학 연구의 한계로, 본 연구에서 비교한 코드 줄 수 또한 프로그래머의 영향을 받게 된다는 점에 주의해야 한다.

일반적으로 수치해석 연구 분야에서는 상용 프로그램

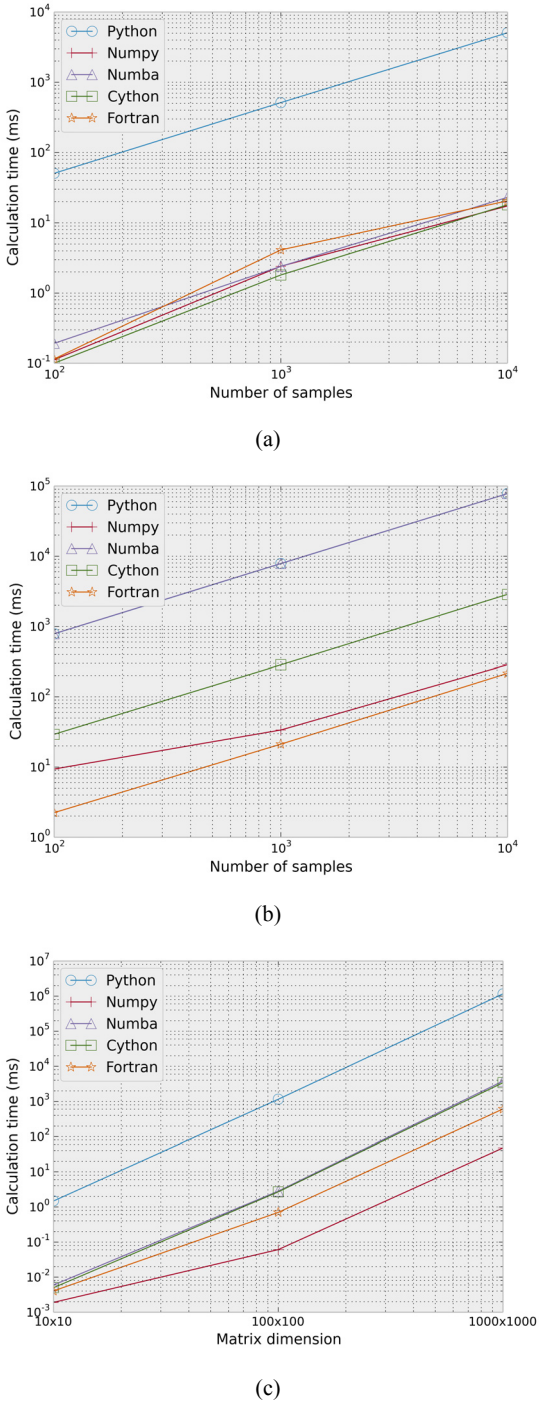


Fig. 11. Calculation times with respect to the size of data for the (a) gain, (b) discrete Fourier transform, and (c) matrix multiplication examples.

개발이나 웹 프로그래밍과 같은 다른 프로그래밍 분야에 비해 프로그램 작성자의 생산성에 관한 논의가 드물다. 여기에는 여러 가지 원인이 있을 수 있는데, 생산성이 측정하기 어렵다는 점 외에도, 효율적인 알고리즘을 통한 계산 효율 향상은 훌륭한 연구 주제가 되지만 연구자의 생산성 향상은 수치해석 분야의 연구 주제가 되기 어렵다는 문제가 있다. 또한 대학이나 연구소 등에서 수치해석 연구용 프로그램을 개발하는 학생이나 연구자는 상용 프로그램 개발자에 비해 프로그램을 개발할 시간적 여유가 많다는 점도 영향을 미칠 수 있다. 그러나 연구자의 생산성이 높아질 경우 추가 연구나 분석을 수행할 여지가 생길 수 있고, 새로운 알고리즘도 보다 쉽게 시도해볼 수 있게 된다. 이러한 생산성은 삶의 수준과도 연관되는 것으로(Mankiw, 2011), 생산성이 높은 언어를 익혀두는 것이 연구자에게 도움이 될 것이다. 개발 속도 외에도 파이썬을 이용하면 다양한 종류의 오픈소스 프로그램들에 쉽게 접근할 수 있다. 본 논문의 모든 그림도 파이썬 Matplotlib 라이브러리를 이용하여 그린 것이다(Vaingast, 2009). 본 연구에 사용한 코드들은 https://github.com/pkgpl/PythonProcessing에서 확인할 수 있다.

결 론

수많은 파이썬 사용자들이 노력하여 파이썬 언어와 과학 계산용 라이브러리들을 발전시켜왔다. 본 연구에서는 이러한 파이썬 언어를 사용하여 간단한 자료처리에서 생산성과 효율성을 향상시키는 방안을 살펴보았다. 실행 시간이 짧은 프로그램일수록 생산성이 중요해지지만 효율성이 지나치게 떨어진다면 생산성이 높더라도 동적 자료형 언어를 사용할 수 없을 것이다. 파이썬에서 Numpy, Numba 및 Cython 라이브러리를 이용하여 여덟 개의 자료처리 관련 알고리즘들의 실행 시간과 코드 길이를 비교한 결과 Numpy의 벡터 연산과 내장함수를 사용하는 것이 생산성과 효율성을 향상시키는 가장 좋은 방법임을 알 수 있었다. Numpy 프로그램을 정적 자료형을 사용하는 언어인 포트란으로 개발한 프로그램과 비교했을 때 평균적으로 실행 시간은 1.4배 느려졌지만 프로그램을 2.8배 짧은 코드로 개발할 수 있었다.

사 사

이 논문은 부경대학교 자율창의학술연구비(2013년)에 의하여 연구되었음.

References

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

Billette, F.J. and Brandsberg-Dahl, S., 2005, "The 2004 BP velocity benchmark," *67th Annual Meeting*, EAGE, Madrid, Spain, June 13-16, p. 8035.

BLAS, 2014.8.28., http://www.netlib.org/blas.

Fleming, P.J. and Wallace, J.J., 1986, "How not to lie with statistics: The correct way to summarize benchmark results," *Communications of the ACM*, Vol. 29, No. 3, pp. 218-221.

IEEE Spectrum, 2014.8.28., http://spectrum.ieee.org/static/interactive-the-top-programming-languages.

IPython, 2014.8.28., http://ipython.org.

Janert, P.K., 2010, *Data analysis with open source tools*, O'Reilly, p. 439.

Julia Programming Language, 2014.7.22, http://julialang.org.

Kreyszig, E., 2011, *Advanced engineering mathematics, 10th ed*., John Wiley & Sons, Inc, p. 263.

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

LAPACK, 2014.8.28., http://www.netlib.org/lapack.

LLVM, 2014.7.22, http://llvm.org.

Madagascar, 2014.7.22, http://www.ahay.org.

Mankiw, N.G., 2011, *Principles of economics, 6th ed*., Cengage Learning, p. 12.

Matlab, 2014.7.22, http://www.mathworks.co.kr/products/matlab.

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

Netlib, 2014.7.22, http://www.netlib.org.

Numba, 2014.7.22, http://numba.pydata.org.

Numexpr, 2014.7.22, http://github.com/pydata/numexpr.

Numpy, 2014.7.22, 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.

PyLab, 2014.8.28., http://wiki.scipy.org/PyLab.

PyPy, 2014.7.22, http://pypy.org.

Python, 2014.7.22, http://www.python.org.

SciPy, 2014.7.22, http://www.scipy.org.

Seismic Unix, 2014.7.22, http://www.cwp.mines.edu/cwpcodes.

The Python Language Reference, 2014.7.22, http://docs.python.org.

The R project, 2014.7.22, http://www.r-project.org.

Vaingast, S., 2009, *Beginning python visualization*, Apress, pp. 249-284.
Versteeg, R., 1994, "The Marmousi experience: velocity model determination on a synthetic complex data set," *Leading Edge*, Vol. 13, pp. 927-936.
Weinberg, G.M., 1998, The Psychology of computer programming: Silver anniversary edition, Weinberg & Weinberg, p. 63.

부록. 라이브러리 요약

독자들의 편의를 위해 본문에 언급한 다양한 라이브러리들에 대한 간략한 설명을 추가한다.

BLAS(Basic Linear Algebra Subprograms) 라이브러리는 포트란으로 작성된 효율적인 저수준 선형대수 연산을 위한 라이브러리로, 벡터 합, 내적, 행렬과 벡터의 곱, 행렬 곱 등의 연산을 지원한다(BLAS, 2014).

LAPACK(Linear Algebra Package)은 BLAS에 기초한 고수준 선형대수 라이브러리로, 고유치 문제, 특이값 분해, 행렬 분해 등을 위한 함수들을 포함한다. 포트란으로 작성되어 있다(LAPACK, 2014).

LLVM은 Low Level Virtual Machine이란 이름으로 시작하였으나, 현재는 LLVM을 이름으로 하는 컴파일러의 기반구조이다. 프로그램을 작성 언어에 무관하게 쉽게 최적화하도록 만들기 위한 라이브러이다. C++로 작성되어 있다(LLVM, 2014).

Numpy는 파이썬에서 균일한 자료형으로 이루어진 배열의 수치 연산을 위한 라이브러리로, 루프 연산의 벡터화를 지원한다. 내부적으로는 C언어로 작성되어 있기 때문에 배열의 벡터 연산을 수행할 경우 C언어로 작성한 프로그램의 성능을 나타낸다. 선형 대수 연산에서 BLAS와 LAPACK 등을 사용한다. 대부분의 파이썬 수치 계산 라이브러리들은 Numpy를 기반으로 한다(Numpy, 2014).

Numexpr은 중앙 처리 장치의 캐시 메모리 사용 최적화를 통해 Numpy 배열 계산 속도를 향상시키기 위한 라이브러리이다(Numexpr, 2014). 배열의 크기가 캐시 메모리의 크기보다 클 때 효과적이다.

Numba는 LLVM 라이브러리를 사용하여 실시간으로 파이썬 코드를 기계어로 변환하여 컴파일한다. 따라서 간단히 파이썬 프로그램의 실행 시간을 단축시킬 수 있다(Numba, 2014).

Cython은 파이썬 코드로부터 C 코드를 생성하고 컴파일하여 프로그램의 실행속도를 향상시키기 위한 라이브러이다(Lanaro, 2013).

Ctypes와 Weave는 C 언어로 작성된 함수나 코드의 일부분을 파이썬에서 쉽게 불러 쓸 수 있도록 만들어주는 라이브러리이다. Ctypes는 파이썬 표준 라이브러리에 포함되어 있고, Weave는 SciPy에 포함되어 있다(Python, 2014; SciPy, 2014).

F2py는 포트란 서브루틴이나 함수를 파이썬에서 불러쓰기 쉽게 만들어주는 라이브러리로, Numpy에 포함되어 있다(Numpy, 2014).

SciPy는 각종 수치해석을 위한 함수들을 포함하는 파이썬 라이브러리로, 최적화, 특수 함수, 적분, 푸리에 변환, 신호 처리, 선형 대수, 통계 등의 다양한 함수들을 지원한다. 이러한 함수들은 대부분 C 또는 포트란과 같이 실행 속도가 빠른 언어로 작성되어 있다(SciPy, 2014).

Matplotlib은 다양한 종류의 출판 품질 2차원 그래프를 그리기 위한 파이썬 라이브러리이다. 일부 3차원 그래프도 지원한다(Vaingast, 2009).

MayaVi는 3차원 영상화를 위한 라이브러리이다(Vaingast, 2009).

IPython은 파이썬 기능을 확장하여 대화식 프로그래밍을 편리하게 하기 위한 라이브러리이다(IPython, 2014). PyLab 프로젝트는 Python, IPython, Numpy, SciPy 및 Matplotlib을 합하여 Matlab보다 우수한 계산 환경을 만드는 것을 목표로 한다(PyLab, 2014).

하 완 수

2006년 서울대학교 공과대학 지구환경시스템공학부, 공학사
2008년 서울대학교 공과대학 에너지시스템공학부, 공학석사
2011년 서울대학교 공과대학 에너지시스템공학부, 공학박사

현재 부경대학교 에너지자원공학과 조교수
(E-mail; wansooha@pknu.ac.kr)