Haskell을 이용한 수치해석의 효율성 연구

오정한[O1], 변석우[2], 우균[3*]
[1]부산대학교 전기전자컴퓨터공학과, [2]경성대학교 컴퓨터공학과, [3]부산대학교 전기컴퓨터공학부
ojh1209@pusan.ac.kr, swbyun@ks.ac.kr, woogyun@pusan.ac.kr

A Study on Efficiency of Numerical Analysis by using Haskell

Junghan Oh[O1], Sugwoo Byun[2], Gyun Woo[3*]
[1]Department of Electrical and Computer Engineering, Pusan National University,
[2]School of Computer Science & Engineering, Kyungsung University,
[3]School of Electrical and Computer Engineering, Pusan National University

요 약

현대에는 수치계산의 대부분을 컴퓨터를 이용하므로 계산의 정밀도를 높이는 것은 중요한 일이다. 그래서 수치계산을 위한 좋은 프로그래밍 언어를 선택하는 것이 필요하다. 본 논문에서는 함수형 프로그래밍 언어인 Haskell이 수치해석에 적합한지에 대해 알아본다. 이를 위해 수치해석 분야에서 널리 사용되는 테일러 급수, 심프슨 1/3 적분법, 심프슨 3/8 적분법, 이분법, 뉴턴랩슨법을 Haskell과 C로 각각 구현하고 각 프로그램의 정밀도와 LOC를 측정하였다. 그 결과 Haskell 프로그램이 C 프로그램보다 정밀도가 우수함을 확인하였다. 그리고 Haskell 프로그램이 C 프로그램보다 LOC가 약 57.69%만큼 더 짧게 나타났기 때문에 Haskell을 이용해서 수치해석 프로그래밍하는 것이 더 좋음을 알 수 있다.

1. 서 론

현대에는 수치 계산 대부분을 컴퓨터에 맡기기 시작하면서 수치적 오차에 관해 민감한 분야가 많아졌다. 특히, 반도체 산업이나 위성항법 시스템 기반 산업, 정밀유도무기 제조업 등과 같은 분야들은 작은 오차에도 경제적인 손실이 발생하게 된다. 그래서 오차를 줄이고 정밀도를 높이는 것은 중요한 일이 되었다.

오차가 발생하는 원인은 여러 가지가 있다. 그 중 측정기의 불완전성 때문에 생기는 오차는 충분히 극복할 수 있는 오차다. 컴퓨터 계산상 발생하는 오차도 이 경우에 해당하는데 컴퓨터 계산을 정밀하게 하는 방법은 어떤 프로그램을 사용하느냐에 달려있다.

사용자의 입장에서 어떤 프로그램을 사용할 것인지는 오차를 줄이기 위해서 중요한 선택이다. 하지만 개발자의 입장에서 어떤 프로그래밍 언어로 프로그램을 개발하는지가 좀 더 본질적으로 중요하다. 그래서 본 논문은 좋은 정밀도를 가진 프로그램을 만들기 위해서 어떤 프로그래밍 언어가 좋은지를 비교하고자 한다.

본 논문에서 비교 대상으로 선택된 프로그래밍 언어는 Haskell과 C이고 본 논문은 Haskell이 C보다 계산정밀도 관점에서 더 뛰어남을 보이고자 한다. 그러기 앞서서 Haskell과 C의 특징을 비교하면 다음과 같다. 우선 C 같은 경우 뛰어난 효율성과 수많은 라이브러리, 익숙함 때문에 많은 사람에게 널리 쓰이고 있다. 하지만 C언어는 느긋한 계산을 지원하지 않는다[2]. 그리고 방대한 양의 수식을 계산해야 할 때, 함수형 언어의 패러다임보다 덜 효과적이다[2].

반면에 Haskell은 함수형 언어의 패러다임을 따르면서 더 효과적으로 계산하고, 느긋한 계산을 지원하므로 효율적인 계산을 진행한다[2]. 이런 장점과 더불어 Haskell이 C보다 수치를 정밀하게 다룬다면 Haskell은 C보다 효과적인 계산이 가능할 것이라 예상된다.

본 논문의 구성은 다음과 같다. 2장에서 관련 이론 및 관련 연구를 밝히고 3장에서 Haskell과 C의 정밀도와 소스코드를 비교하고자 한다. 마지막으로 4장에서 Haskell과 C의 비교 실험에 대해 결론을 내린다.

2. 관련 이론 및 관련 연구

본 장은 관련 이론 및 관련 연구를 소개한다. 본 장의 구성은 다음과 같다. 2.1장에서 테일러 급수 이론을 소개하고 2.2장에서 테일러 급수식을 변형한 식을 소개한다.

2.1 테일러 급수

테일러 급수 개념은 James Gregory에 의해 발견됐다. 그리고 1715년 Brook Taylor에 의해 공식적으로 소개됐다[3]. 이 테일러 급수는 복잡한 초월함수나 삼각함수를 다항 함수로 표현 가능하다는 특징이 있다. 그래서 미분이나 적분을 하기가 쉽다.

$$T_f(x) = \lim_{n\to\infty} \sum_{i=0}^{n} \frac{f^{(i)}(a)}{i!}(x-a)^i \qquad (1)$$

위 식(1)은 테일러 급수식이다. 일반적으로 식(1)은 유한한 항으로 함수를 근사화하는 것이 관행이다. 이렇게 한 함수에 대해 테일러 급수를 유한한 항으로 표현한 식을 테일러 다항식이라고 부른다. 이 테일러 다항식은 항이 많아질수록 근사치가 좋아진다는 특징이 있다[3].

2.2 테일러 급수의 변형된 식

기존 테일러 급수식은 무한하고 연속적인 것이 특징이

다[3]. 하지만 실험 환경적으로 시간, 메모리의 제한이 있다. 그러므로 유한하고 이산적으로 식을 고쳐야 한다. 아래 식(2)는 유한하고 이산적인 테일러 급수식이다[3].

$$T_f(x) = \sum_{i=0}^{n} \frac{(x-a)^i}{i!} \frac{\sum_{j=0}^{i} (-1)^{i-j} f(a+jh)\ {}_iC_j}{h^i} \qquad (2)$$

3. 실험 및 결과

본 장에서는 테일러 급수를 이용해서 Haskell 소스코드와 C 소스코드의 정밀도를 비교, 실험한다. 그리고 어떤 언어의 소스코드가 LOC가 나은지 확인한다. 이때 소스코드를 비교하기 위해 사용된 이론은 테일러 급수를 포함해서 심프슨 1/3 적분법, 심프슨 3/8 적분법, 이분법, 뉴턴랩슨법, 총 5개의 이론에 대해서 5개의 프로그램을 구현해 비교한다.

실험된 환경은 다음과 같다. 운영체제는 Microsoft Windows 7 Professional K, Haskell 컴파일 버전은 2014.2.0.0이고 C는 Visual Studio Community 2013에서 실험했다. C, Haskell 둘 다 소스코드 상 이용된 변수의 타입은 int형과 double형이다.

본 장의 구성은 다음과 같다. 우선 3.1장에서 테일러 급수를 이용해 오차율 비교, 실험한다. 다음 3.2장에서 Haskell과 C로 구현된 소스코드 LOC를 비교한다.

3.1 테일러 급수 오차율 비교 실험

아래 식(3)에서 $f(x)$는 내장함수로 만든 함수이고 $T_f(x)$는 $f(x)$를 이용한 테일러 급수의 값이다. 이 두 값의 차이를 이용해 오차율을 구한다. 오차율이 낮을수록 $T_f(x)$값이 $f(x)$값에 가까움을 의미한다. 그리고 이 오차율은 각 언어의 내장함수를 사용했기 때문에, 내장함수의 정밀도는 실험상 영향을 미치지 않는다. 이 실험은 테일러 급수의 계산과정 동안 얼마나 값 손실이 발생하지 않았는지가 중점이다.

$$\text{오차율}(\%) = \frac{|f(x) - T_f(x)|}{f(x)} \times 100 \qquad (3)$$

오차율을 Haskell, C 프로그램 각각에서 구한 후 비교한다. 그 후 어떤 언어로 구현한 프로그램이 더 정확한 결과를 내는지 판단한다. x값은 $[-0.5, 0.5]$ 범위에서 0.01의 차이를 주고 총 101개의 값으로 실험했다. 각 x값에 따른 오차율을 각각 구한 후 각 함수에 대한 오차율평균을 낸다. 표 1에서 $f(x)$는 실험에 사용된 함수이고 오차율 평균은 각 언어가 각 $f(x)$에 대해 나온 오차율의 평균을 낸 값이다.

<표 1> $f(x)$값에 따른 Haskell 프로그램과 C 프로그램의 오차율 비교

$f(x)$	오차율 평균(%)	
	Haskell	C
$sin(x)$	0.6321	1.2845
$cos(x)$	1.2481	1.2543
$tan(x)$	1.1579	2.4650
e^x	1.3142	1.3208
$sin(x)+cos(x)$	1.2719	1.2783
$sin(x)+tan(x)$	0.2892	0.6499
$sin(x)+e^x$	2.6871	2.7005
$cos(x)+tan(x)$	1.9091	1.9186
$cos(x)+e^x$	0.1671	0.1679
$tan(x)+e^x$	1.2704	1.2767

표 1의 결과를 보면, 다양한 $f(x)$에 대해, Haskell이 C보다 모든 경우에서 오차율의 평균이 낮은 것을 확인할 수 있다. 결국, 이번 실험의 결과는 다음과 같다. 대량의 수치계산을 할 때, Haskell이 C보다 더 정확하게 값을 계산해내는 경우가 더 많다는 것이다.

3.2 Haskell과 C의 테일러 급수의 LOC 비교

이번 장은 수치해석에서 유명한 이론을 이용해 Haskell과 C로 각 언어의 소스코드를 구현한다. 그리고 소스코드의 길이를 비교한다. 다음 그림 2와 그림 3은 Haskell과 C로 구현한 뉴턴랩슨법 소스코드이다[5].

```
#include <stdio.h>
#include <math.h>

double f(double);
double df(double);
double xpos(double);
double Newton(double);

double f(double x){
     return x*x*x - x*x + 2 * x - 5;
}

double df(double x){
     double h = 0.001;
     return (f(x + h) - f(x)) / h;
}

double xpos(double a){
     return -f(a) / df(a) + a;
}

double Newton(double x){
     if (abs(f(x)) < 0.001)
        return x;
     return Newton(xpos(x));
}

int main(){
     printf("%lf\n", Newton(5));
     return 0;
}
```

(그림 2) C의 뉴턴랩슨법 소스코드

```
main ::  IO()
main =     do putStrLn(show(newton(5)))

f :: Double -> Double
f x =      exp(x)

df :: Double -> Double
df x =     ((f (x + h)) - (f x))/h
           where h = 0.001

xpos :: Double -> Double
xpos a =   -(f a)/(df a) + a

newton x   | abs(f x) < 0.001   = x
           | otherwise       = newton (xpos x)
```

(그림 3) Haskell의 뉴턴랩슨법 소스코드

위 그림 2와 그림 3의 소스코드를 통해 LOC를 측정한 결과, Haskell이 C보다 57.69%만큼 좋다. 그리고 간결하다. 이 결과에 대한 이유로는 다음과 같다.

첫 번째로 조건문의 간결함이다. Haskell에서는 조건식 대신 보초선 등식으로 정의할 수 있다[2]. 이 보초선 등식은 기호 | 와 =로 if then 구문을 대체할 수 있다. 게다가 타입형을 따로 정의할 필요가 없기 때문에 인자에 타입형을 선언하는 C보다 소스코드가 짧아지다. 그래서 Haskell의 조건문을 C보다 더 간결하게 구현할 수 있다.

두 번째로 Haskell은 C처럼 함수의 선언을 하지 않는다. 그래서 만약 5개의 함수를 정의했다면, 5개의 선언문이 필요하다. 이로 인해 C는 Haskell보다 소스코드의 길이가 늘어나게 돼서 상대적으로 Haskell은 간결해진다.

그 외 요인으로는 Haskell은 괄호를 사용하지 않는다. 그리고 return 키워드가 없다. 이와 같은 이유로 Haskell은 C보다 간결한 소스코드의 작성이 가능하다.

다음은 뉴턴랩슨법을 비롯해 테일러 급수, 심프슨 1/3 수치 적분법, 심프슨 3/8 수치 적분법, 이분법까지 총 5개를 Haskell과 C로 소스코드를 각각 구현한다. 그리고 각각 5개의 LOC를 비교한다. 표 2는 정리된 결과이다.

<표 2> 수치해석 프로그램에 대한 Haskell과 C의 LOC 비교

수치해석 프로그램	Haskell (LOC)	C (LOC)
테일러 급수	20	58
심프슨 1/3 수치 적분법	14	31
심프슨 3/8 수치 적분법	15	34
이분법	9	26
뉴턴랩슨법	11	26

결과적으로 표 2를 보면 Haskell이 C보다 모든 프로그램에 대해서 LOC가 짧은 것으로 결과가 나온다. 수치로 계산하면 Haskell은 C보다 LOC가 60.20% 만큼 우수한 것으로 나온다. 이로써 Haskell로 수치해석 프로그램을 구현했을 때 C와 유사한 성능을 보이면서 간결한 코드 작성이 가능하다. 그러므로 Haskell을 수치해석분야에서 이용하는 것은 효과적임을 알 수 있다.

4. 결론 및 고찰

본 논문에서는 Haskell과 C의 정밀도를 비교하기 위해 테일러 급수를 사용했다. 테일러 급수로 모두 똑같은 알고리즘, 변수형, 내장함수로 프로그램을 각각 구현했을 때, Haskell의 프로그램이 C보다 좀 더 정밀한 것을 확인했다. 그리고 Haskell이 C보다 더 소스코드를 간결하게 작성되는 것을 뉴턴랩슨법 소스코드로 확인했다.

또한, 뉴턴랩슨법을 비롯해서 수치해석의 이론인 테일러 급수, 심프슨 1/3 수치 적분법, 심프슨 3/8 수치 적분법, 이분법을 구현한 소스코드로 Haskell의 간결함을 확인했다. 결국, 위 결과를 통해서 정밀도 관점에서 Haskell은 C만큼 수치해석에 적합하고 LOC 관점에서 코드 생산성이 뛰어남을 알 수 있었다.

그리고 본 논문 3.1절의 정밀도 비교실험에서 Haskell이 C보다 나은 결과가 나온 원인을 고찰해본 결과, Haskell과 C의 변수 메모리 크기 차이 때문이다. 우선 Haskell의 Double은 32bit 컴퓨터 시스템에서 12bytes의 메모리를 사용하고 64bit 시스템에서는 16bytes의 메모리를 사용한다[6]. 반면에 C의 메모리 크기는 System Vendor의 몫인데, 본 논문의 실험된 환경에서는 8bytes의 메모리를 사용했다. 그래서 Haskell이 좀 더 정밀하게 결과가 나올 수 있었다.

향후 연구 주제는 다음과 같다. Haskell과 C가 대량의 수치계산을 하게 될 때, 느긋한 계산법을 이용해 Haskell과 C의 실행시간을 측정 비교하는 것이다. 그래서 어떤 언어가 더 수치해석적으로 효과적이고 효율적으로 계산을 수행하는지 비교할 것이다.

ACKNOWLEDGMENT

본 연구는 미래창조과학부의 SW컴퓨팅산업원천기술개발사업의 일환으로 수행하였음(B0101-15-0644, 매니코어 기반 초고성능 스케일러블 OS 기초연구).

*교신 저자 : 우균(부산대학교, woogyun@pusan.ac.kr).

참고문헌

[1] TIOBE Index for April 2015, http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html.

[2] G. Hutton, *Haskell로 배우는 프로그래밍*, 도서출판 대림, 2009.

[3] Taylorseries, http://en.wikipedia.org/wiki/Taylor_series.

[4] Numerical_analysis, https://en.wikipedia.org/wiki/Numerical_analysis.

[5] 지영준, 김화준, 허정권, *C로 구현한 수치해석*, 높이깊이, 2010.

[6] GHC/Memory Footprint, https://wiki.haskell.org/GHC/Memory_Footprint.