고급소프트웨어실습1

Lecture 10 과제

반 : 4

학번 : 20171669

이름 : 이재영

1. 실습
2. 실습1
3. 행렬 계산의 속도 향상을 위해 적용한 방법

2번의 경우는 전치행렬을 사용해서 행렬 계산을 사용했고 3,4번은 Loop unrolling 기법을 사용해서 행렬 계산의 속도를 향상했다.

1. 어떠한 근거로 자신이 이 방법이 더 효율적인지 정했는가

C++에서는 row major로 행렬을 접근하는데 기존의 1번의 경우, column major로 행렬을 접근하기 때문에 여러 번 block에 접근하기 때문에 시간이 많이 느렸지만 2번의 경우는 전치행렬을 사용해 row major로 사용하므로 더 효율적이다.

3,4번의 경우는 loop unrolling을 사용하기 때문에 비교 과정에서의 시간낭비를 줄이기 때문에 더더욱 효율적인 것을 확인할 수 있다.

1. 어떤 m 값에 대해 loop unrolling 방법이 효과적이었는지 요약

3번에서는 m을 8번, 4번에서는 m을 32번 한 결과 다음과 같이 결과가 나옴을 확인할 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 실습2

다음 결과와 같이 Horner’s rule을 사용하면 다항식의 계산을 효율적으로 할 수 있다. Horner’s rule을 사용하면 실행시간이 적게 걸리는 이유는 곱셈의 연산 횟수가 줄어들기 때문이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 실습3

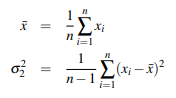
X의 값이 음수이면 테일러 급수식으로 보면 양수, 음수를 번갈아 가면서 더하게 된다. 이 때, 항을 계산하면 할수록 소수점 아래 자리수는 늘어나는데 유효숫자의 제한으로 정확하게 나타내지 못하므로 double을 사용하면 유효숫자의 범위가 늘어나 정확한 값을 얻을 수 있다.

다음 결과 값들을 확인해보면 기존의 식을 사용한 Taylor\_series를 사용하면 값의 차이가 발생하는 반면 Taylor\_series\_ex로 음수인 경우는 양수로 바꿔서 계산한 후 역수 값을 취하는데 이 경우 결과값이 완전히 동일 하다는 것을 알 수 있다.

텍스트, 시계이(가) 표시된 사진

자동 생성된 설명

1. 숙제
2. 숙제1

<1><2>텍스트이(가) 표시된 사진

자동 생성된 설명

1. 두 분산 값 계산 방법의 결과가 상당히 차이가 나게 해주는 샘플 데이터를 생성한 후, 계산 결과를 비교 분석하라. 분산을 어떻게 계산한 것이 더 정확한 것으로 판단되는가? Double 타입의 연산이 필요하면 이유를 기술한다.

<1>의 식을 사용하는 경우 값과 평균의 차이를 구하는데 이 때 값이 비슷해서 부동 소수점 연산의 오차가 발생할 수 있는 경우가 많이 발생한다. <2>의 식은 두 수의 뺄셈의 방법이 <1>의 식보다는 현저히 적게 나오므로 <1>의 식이 더 정확하다. 또한 <2>의 식은 뒤의 값이 큰 경우 분산은 음수가 나올 수 없는데 분산이 음수가 나오는 경우가 있을 수 있기 때문에 위험하다.

double연산이 필요한 경우는 float의 범위를 벗어나는 경우 double이 그 범위를 cover할 수 있기 때문에 필요하다.

1. 충분히 큰 n에 대하여 두 방법 중 어떤 방법이 더 빠르게 분산값을 계산하는가?

연산의 수행 결과를 비교해보면 <2>의 식을 사용하는 경우 <1>의 식보다 더 빠르게 분산 값을 구한다는 것을 확인 가능하다.

텍스트이(가) 표시된 사진

자동 생성된 설명

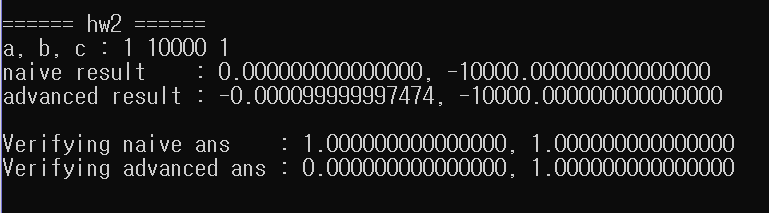
N이 1000일 경우 2의 식이 1의 식보다 더 빠르다는 것을 확인할 수 있다.

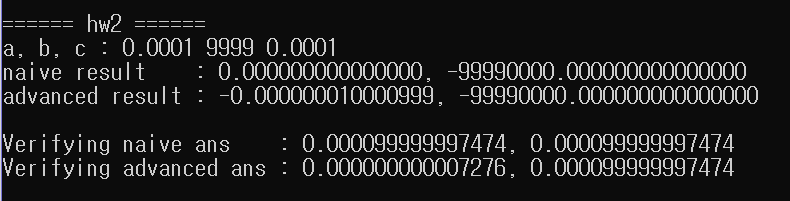
텍스트이(가) 표시된 사진

자동 생성된 설명

다음의 경우와 같이 n이 100000인 경우 실행에 오류가 난 것을 확인할 수 있다.

1. 숙제2
2. 위 프로그램이 심각한 문제를 야기하는 상황을 세 가지 발생시켜라. 즉 그런 문제를 일으킬 a, b, 그리고 c값을 적절히 설정한 후, 위에서 구한 두 근을 다시 f(x)에 대입하여 0이 나오는지 확임함으로서 심각한 문제가 발생하였다는 것을 증명하라.





텍스트이(가) 표시된 사진

자동 생성된 설명

위의 결과들은 기존의 근의 공식으로 사용해서 심각한 문제가 발생하는 3가지의 경우를 나타낸다. 기존의 근의 공식의 경우는 분자에서 a와 c가 매우 작아서 루트의 값이 b의 값과 매우 근사하게 된다면 비슷한 수의 뺄셈을 수행하게 되므로 부동 소수점 연산의 오차가 발생한다.

1. 다음 그러한 문제를 완화시킬 수 있는 방법을 사용하여 위의 2차 방정식을 풀어주는 함수를 새롭게 구현한 후, 위의 문제와 동일한 과정를 거쳐 (즉 자신이 구한 근에 대해 f(x) 함수 값을 구하여), 위에서 심각한 문제를 야기한 세 경우 각각에 대해 자신의 두 번째 함수가 안정적으로 근을 구했음을 밝혀라

* 기존의 근의 공식에서는 부동 소수점의 오차가 발생하기 때문에 다음과 같은 식으로 유리화 변환하여 근의 공식을 계산한다. 이 공식을 사용한 결과 기존의 결과보다 오차를 감소한다는 것을 확인할 수 있으며 이는 비슷한 수의 뺄셈은 고려하기 않기 때문에 근이 더 정확하다는 것을 의미한다.

1. 숙제3
2. Strength Reduction 방법

텍스트이(가) 표시된 사진

자동 생성된 설명

<debug>

텍스트이(가) 표시된 사진

자동 생성된 설명

<release>

텍스트이(가) 표시된 사진

자동 생성된 설명

Strength reduction 방법은 비용이 많이 드는 연산자들을 비용이 적게 드는 연산자들로 바꿔서 수행하는 방법이다. 코드에서는 곱셈과 나눗셈 연산(\*, /)을 bit(<<,>>) 연산으로 바꿔서 수행시간을 감소시켰다. Release모드에서는 코드를 최적화해서 수행하는데 Strength Reduction을 사용한 경우가 훨씬 수행시간이 빠르다는 것을 볼 수 있다.

1. Loop Invariant Code Motion

텍스트이(가) 표시된 사진

자동 생성된 설명

<debug>

텍스트이(가) 표시된 사진

자동 생성된 설명

<release>

텍스트이(가) 표시된 사진

자동 생성된 설명

이는 loop에서 overhead를 발생하는 불변하는 계산들을 loop문 밖으로 빼서 상수 값으로 대체하는 방법으로 위의 결과와 같이 수행시간의 감소를 확인할 수 있다.

1. Use Local Instead of Global

텍스트이(가) 표시된 사진

자동 생성된 설명

<debug>

텍스트이(가) 표시된 사진

자동 생성된 설명

<release>

텍스트이(가) 표시된 사진

자동 생성된 설명

Global 변수를 사용하는 대신 local 변수를 사용한 경우로 이 방법은, global 변수가 register에 저장이 불가능하기 때문에 연산 속도가 느린 것을 보완하는 방법이다. Local 변수를 사용하는 경우 register에 저장이 가능하므로 연산 수행이 빠르며 이를 통해 수행시간을 줄이는 것을 확인할 수 있다. Release 모드의 결과에서는 극명한 차이를 확인할 수 있다.

1. Loop Inversion

텍스트이(가) 표시된 사진

자동 생성된 설명

<debug>

텍스트이(가) 표시된 사진

자동 생성된 설명

<release>

텍스트이(가) 표시된 사진

자동 생성된 설명

Loop inversion이라는 방법은 반복문 while을 조건문 if 안의 do while 반복문으로 바꿔주는 방법이다. 이 방법은 조건문을 확인할 때 jump 수행을 하지 않아도 되므로 연산 과정에서 파이프라인을 효과적으로 수행할 수 있도록 해준다. 위의 결과를 통해 수행시간이 줄었다는 것을 확인할 수 있다.

1. Use Eliminating Sub Expression

텍스트이(가) 표시된 사진

자동 생성된 설명

<debug>

텍스트이(가) 표시된 사진

자동 생성된 설명

<release>

텍스트이(가) 표시된 사진

자동 생성된 설명

Elimination sub expression 방법은 반복적으로 수행되는 함수의 결과 혹은 연산 과정을 상수로 취급해서 연산의 횟수를 줄이는 방법으로 코드에서는 pow함수의 연산 횟수를 줄이면서 연산의 overhead를 줄이면서 수행시간이 줄었다는 것을 확인할 수 있다.