고급소프트웨어실습1 10주차 과제

20161565 권기윤

실습문제1

행렬 계산의 속도 향상을 위하여 (1) 자신이 적용한 방법과 (2) 어떠한 근거로 자신 이 적용한 방법이 더 효율적일지, 그리고 (3) 어떤 m 값에 대해 loop unrolling 방법이 가장 효과적이었는지를 요약하여 프로그램과 함께 제출할 것.

행렬 계산의 속도를 향상시키는 방법은 크게 보면 2가지가 있다. 첫번째는 행렬의 참조 순서를 캐시정책인 spatial locality를 고려하여 조정하여 Cache miss를 줄이는 것이다. 두번째 방법은 loop unrolling으로 반복문 상에서 수행되는 산술연산, 비교연산을 줄일 수 있다. Loop unrolling의 단점은 코드 길이가 길어져 가독성이 떨어진다는 것과 루프 길이를 지나치게 늘리면 레지스터의 재사용이 벌어져 일부가 spill out 될 수 있다. 그렇기 때문에 적절한 unrolling 횟수를 기반으로 구현하여야 한다. 적절한 m = 4 일 때, 가장 최고의 성능을 발휘하였다.

실습문제2

각 항목의 실험을 통하여 발견한 사실을 기술하여 프로그램과 함께 제출할 것

다항식의 계산에 대해서, Horner’s Rule을 사용하여 더 효율적인 계산이 가능했다. Horner’s Rule을 통해 기존의 다항식 계산법에서 강제되던 x의 곱연산과 달리 더 적은 산술연산을 통해 값을 얻을 수 있었다.

실습 문제3

각 항목의 실험을 통하여 발견한 사실을 기술하여 프로그램과 함께 제출할 것.

부동소수점 연산시에 비슷한 숫자의 뺄셈이 심각한 문제를 초래할 수 있다는 것과 이를 해결하는 방법을 알 수 있었다.

숙제1)

i) 우선 적절히 큰 n을 선택하여 난수를 발생시켜 적절한 구간의 샘플 데이터 xi (i = 1,2,···,n) 를 생성하라. 다음 각각 x¯, σ2, 그리고 σ1을 계산해주는 함수를 작성하라 (float 타입의 연산을 사용하되, 만약 실험중 double 타입의 연산이 필요하면 이유를 기술하고 사용할 것

텍스트이(가) 표시된 사진

자동 생성된 설명

N이 적절하게 큰 경우 각 함수는 float type을 사용하여도 무방하다.

텍스트이(가) 표시된 사진

자동 생성된 설명

< N=1000 >

하지만 N이 굉장히 큰 경우, 산술연산 과정에서 overflow가 발생하여 오류가 발생한다.

그렇기 때문에 double을 사용하여 산술연산을 수행하고, 결과를 float로 변환하는 과정을 거쳤다.

텍스트이(가) 표시된 사진

자동 생성된 설명

< N = 1000000 >

(ii) 두 분산 값 계산 방법의 결과가 상당히 차이가 나게 해주는 샘플 데이터를 생성한 후, 계산 결과를 비교분석하라. 분산을 어떻게 계산한 것이 더 정확한 것으로 판단되는가? (즉 수학적으로 동일한 두 식이 컴퓨터상에서는 얼마나 다를 수 있는지를 스스로 파악하는 것이 목적임. 일반적으로 후자의 방법이 더 위험하다고 알려져 있음. 참고로 분산 값은 음수일 수 없음

* init\_hw1(1) 의 parameter인 flag 값을 1로 주어 분산이 적은 data를 생성한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

< N = 100 >

텍스트이(가) 표시된 사진

자동 생성된 설명

< N = 1000 >

텍스트이(가) 표시된 사진

자동 생성된 설명

< N = 10000 >

텍스트이(가) 표시된 사진

자동 생성된 설명

< N = 100000 >

텍스트이(가) 표시된 사진

자동 생성된 설명

< N = 1000000 >

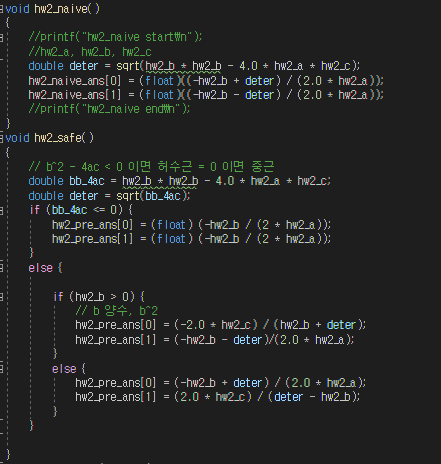
* hw1\_calc\_var2 은 hw1\_calc\_var2() 함수로 구현되었다. 데이터의 분산값이 매우 작아 계산 결과가 0으로 수렴하는 값이 나오는데, 이는 동 소수점의 비슷한 숫자의 뺄셈 오차가 발생하여도, 음수값이 발생하는 문제가 생기지 않는다. 계산 과정에서 제곱이 이루어지고 모든 계산은 양수의 덧셈으로 수행되기 때문이다. 하지만 이 방법 또한 정확하게 분산 값을 구하지 못하는데, double type에서 부동소수점을 표현가능한 자릿수에 한계가 있기 때문이다. 따라서 N의 크기가 크다면 부동소수점 표현의 한계로 정확한 계산이 이루어 지지 않고 결과가 모두 0으로 수렴한다.
* 두번째 방법은 개선된 식을 구현한 것으로 계산과정에서 뺄셈이 등장하여 이 과정에서 부동소수점의 비슷한 숫자의 뺄셈 오차가 발생한다. 때문에 수행 결과와 같이 분산 값으로는 나올 수 없는 음수 값이 나올 수 있다. 따라서 분산을 계산하는 경우 좀 더 느리지만 첫번째 방법이 정확하고 안전하다.

(iii) 충분히 큰 n에 대하여 두 방법 중 어떤 방법이 더 빠르게 분산 값을 계산하는가?

* : 이는 위의 계산 결과를 통해 확인 가능한데, N이 충분히 큰 경우 hw1\_calc\_var1()에 해당하는 두번째 방법이 더 빠르게 분산 값을 계산한다. 왜냐하면, 첫번째 방법의 경우 평균을 구하는 연산을 수행하고, 이후 다시 분산을 구하는 과정을 수행하고, 두번째 방법에 비해 더 많은 산술연산을 수행한다. 때문에 두번째 방법이 더 빠름을 확인 가능하다.

숙제2

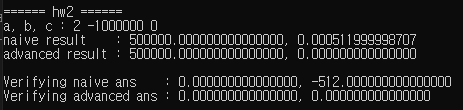
(iii) 다음 그러한 문제를 완화시킬 수 있는 방법을 사용하여 위의 2차 방정식을 풀어주는 함수를 새롭게 구현한 후, 위의 문제와 동일한 과정을 거쳐 (즉 자신이 구한 근에 대해 함수 값을 구하여), 위에서 심각한 문제를 야기한 세 경우 각각에 대해 자신의 두번째 함수가 안정적으로 근을 구했음을 밝혀라.



기존 방법 (naive)은 b > 0, b^2의 값과 ac의 값의 차이가 굉장히 큰 경우, 의 계산과정에서 비슷한 숫자의 부동소수점의 뺄셈 오차 문제가 발생한다. 이를 해결하기 위해서 hw2\_safe() 함수에서 각 문제에 대해 문제가 발생하는 식을 유리화 하여 뺄셈 과정을 덧셈으로 대체하게 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명



하지만 float로 표현 가능한 부동소수점이 부족해지게 된다면 여전히 문제가 발생한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이는 double로 계산한 경우 정확한 값을 도출해낸다.

숙제3

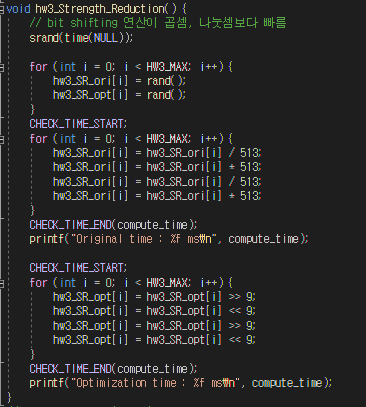
(i) 자료 조사를 통하여 일반적으로 널리 사용되고 있는 원시 언어 수준 (Source Code Level) 의 코드 최적화 기법에 대하여 공부를 한 후, (ii) Loop Unrolling과 Loop Fusion 기법을 제외한 나머지 기법 중 5개를 선정하여, (iii) 각 방법 별로 해당 방법의 효과를 최대한 보일 수 있는 C/C++ 코드를 작성한 후, (iv) 해당 기법 적용 전과 적용 후의 시간을 비교한 내용을 보고서에 명기한 후 그러한 결 과가 나온 이유를 상세히 설명하라

- release mode에서는 몇몇 실험의 경우 실험결과에 대한 확인이 힘들어 release mode에서 최적화를 off하여 실험을 진행하였다.



1) Constant Propagation

- 연산에서 사용하는 변수가 const한 값 일 경우, memory load 횟수를 줄이기 위해 메모리를 참조하는 변수를 사용하는 것보다 상수 값으로 사용하는 것이 효율적이다.

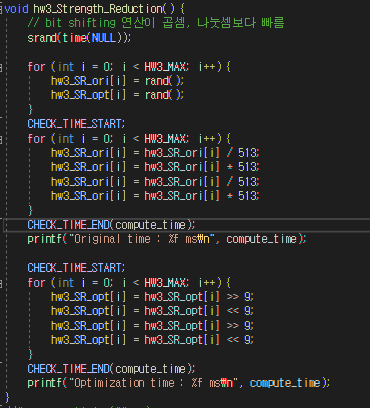


텍스트이(가) 표시된 사진

자동 생성된 설명

2) Strength Reduction

- 동일한 동작 수행에 있어서 곱셈, 나눗셈 연산보다 bit shifting 연산이 더 빠르다.



텍스트이(가) 표시된 사진

자동 생성된 설명

3) Loop Interchange

- 반복문에서 메모리 참조시에 cache에 적재되는 데이터의 구조를 이해하고 고려하여 접근하는 것이 더 효율적이다. C언어의 경우 row major식으로 행렬을 저장하는데, 한번에 같은 row가 cache에 적재된다. 따라서 column을 먼저 증가시키는 반복문이 효율적이다.

텍스트이(가) 표시된 사진

자동 생성된 설명



4) Loop Peeling

- 반복문 내부의 동작을 최소화하고, special case의 해결을 위해 반복문 확장을 하지 않는 것이 코드의 수행면에서 더 효율적이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

5) Loop Inversion

- for문의 비교연산은 while문의 비교연산보다 더 느리다. 따라서 동일한 동작이라면 while문이 더 빠르다.

텍스트이(가) 표시된 사진

자동 생성된 설명

