집중교육2- ISPC Performance Compare (2차과제 보고서)

201521032 한태희

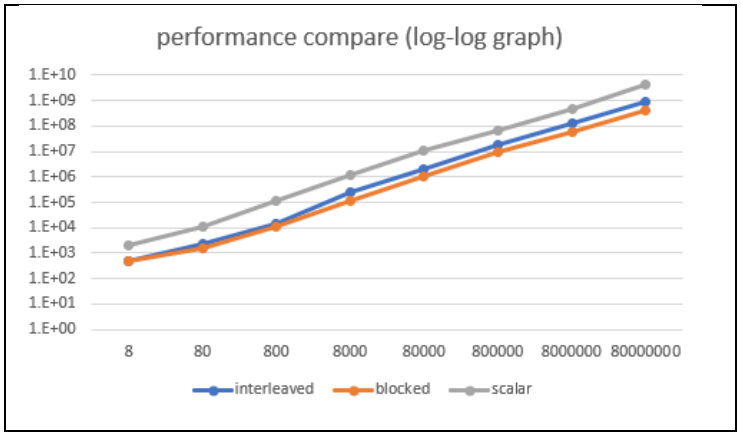
1. 성능 측정 결과

두번째 실습 과제의 주제는 인텔에서 제공하는 ISPC 컴파일러로 구현한 interleaved vector 방식과 blocked vector 방식, 그리고 단순한 scalar 방식으로 구현한 sin(x) 연산의 성능을 비교하고 분석하는 것이다. 실습 시간에 주어진 interleaved 방식의 벡터 연산을 이용해서 blocked 방식의 sin(x) 연산을 구현하였다.

3가지 종류의 함수를 임의의 크기에 배열로 실행시키면서, __rdtsc() 함수를 이용해 sin(x)함수가 실행되는데 소요된 사이클 횟수를 측정하였다..

	8.E+00	8.E+01	8.E+02	8.E+03	8.E+04	8.E+05	8.E+06	8.E+07
interleaved	490	2303	15008	261807	1904196	19318943	1.28E+08	9.05E+08
blocked	462	1470	10990	107660	1091587	9976277	61949137	4.33E+08
scalar	2114	11522	111720	1113791	11194855	63421643	4.77E+08	4.49E+09

위의 표에서 1행은 함수의 종류, 1열은 입력된 배열의 크기를 나타내며, 나머지 값은 해당 독립변수일 때 소요된 사이클 횟수를 나타낸다. 해당 결과 표를 배열의 log-log scale의 그래프로 나타내면 아래와 같다.



회귀 분석 결과, 입력 배열을 N, 연산이 진행된 사이클 타임을 CT라고 한다면 3가지 경우에 아래와 같은 관계식을 가지고 있음을 알 수 있다. 측정한 N의 크기가 매우 차이 나므로 상수는 크게 의미가 없고, N에 곱해지는 계수를 의미 있게 활용할 수 있다.

(interleaved vector) CT = 6561452.647 + (11.26996736* N)

(blocked vector) CT = 3374868.648 + (5.394598319* N)

(scalar) CT = 7538449.024 + (56.02092044* N)

2. scalar 연산 방식과 vector 연산 방식의 성능 비교

scalar 연산과 vector 연산을 비교하면, 모든 경우의 수의 측정에서 vector 연산이 훨씬 짧은 클럭 사이클에 끝나는 것을 확인할 수 있었다. scalar 연산과의 사이클 횟수의 비율을 구해보면, interleaved vector 연산은 n= $8*10^6$ 인 경우에 최소로 약 3.7 배, n= $8*10^2$ 인 경우에 최대로 약 7.4 배의 속도로 실행되었다 blocked vector 연산은 n= $8*10^1$ 인 경우에 최소로 약 4.6 배, n= $8*10^3$ 인 경우에 최대로 약 10.1 배의 속도로 실행되었다.

실습에서 제공된 컴퓨터는 벡터 방식으로 float 데이터를 처리할 경우 동시 8 개의 ALU 가 연산을 분할해서 실행한다. 실습으로 제공된 환경은 타인과 리소스를 공유하고, 운영체제 등에서도 리소스를 사용할 수 있기 때문에 항상 균일하게 vector 연산이 scalar 연산보다 8 배 좋은 성능을 보여주진 않았다. 하지만 vector 연산이 scalar 연산보다 상수 배율만큼 향상된 성능으로 실행된다는 것은 확인할 수 있었다.

3. interleaved vector와 blocked vector의 성능 비교

blocked vector 연산으로 구현한 sinx 연산은 interleaved vector로 구현한 연산보다 항상 빠른 결과를 보여주었다. 같은 vector 연산이더라도 스레드가 참조하는 메모리 구역을 나누는 방식에 따라 2배의 성능 차이가 발생했다. 그래프를 보면 확인할 수 있듯이, 800개의 배열을 연산할 때까지는 속도의 차이가 크게 발생하지 않았지만, 8000개 이상의 배열을 연산할 경우부터 2배의 성능 차이가 있다. 이론에서 배웠던 내용으로는 원인이 두 알고리즘의 메모리 참조 방식이 다르기 때문에 성능 차이가 발생한다고 설명이 되어있다. interleaved 방식은 각 스레드의 캐시가 분산되어 넓게 분포되어 있는 구간을 커버해야 하지만, blocked 방식은 하나의 스레드 캐시가 집중된 한 구간만 저장하면 되기 때문이다. 실제로 이러한 현상이 발생하는지 개인 컴퓨터에 valgrind 어플리케이션을 설치하여, cachegrind 기능을 이용해 두 경우에 메모리 읽기 쓰기가 진행될 때, 캐시 미스가 얼마나 발생하는지 측정해 보았다. (데이터 크기 800000의 경우)

```
==21235== D   refs:       1,087,094  (711,345 rd   + 375,749 wr)
==21235== D1  misses:       116,747  ( 64,321 rd   +  52,426 wr)
==21235== LLd misses:       109,758  ( 58,251 rd   +  51,507 wr)
==21235== D1  miss rate:       10.7% (    9.0%     +    14.0%  )
==21235== LLd miss rate:       10.1% (    8.2%     +    13.7%  )
```

<interleaved vector 연산의 경우 데이터 참조 횟수와 캐시 미스 횟수>

```
==21237== D   refs:       2,287,097  (1,311,345 rd   + 975,752 wr)
==21237== D1  misses:       116,753  (   64,324 rd   +  52,429 wr)
==21237== LLd misses:       109,764  (   58,254 rd   +  51,510 wr)
==21237== D1  miss rate:        5.1% (      4.9%     +     5.4%  )
==21237== LLd miss rate:        4.8% (      4.4%     +     5.3%  )
```

<blocked 연산의 경우 데이터 참조 횟수와 캐시 미스 횟수>

실습으로 제공된 컴퓨터와 달리 개인 컴퓨터는 Vector 연산에 물리적 ALU가 많이 지원되지 않기 때문에, 실행 사이클은 실습 컴퓨터보다 효율의 폭이 적게 나왔다. 하지만, 데이터 참조 캐시 미스 횟수는 예상대로 interleaved 방식이 blocked 횟수보다 cache miss rate가 더 높게 나왔다.

정확하게는 캐시 미스 횟수는 비슷하지만 데이터를 참조한 횟수가 크게 차이가 나는데, blocked 방식은 한번 데이터 구간을 한번에 캐시로 읽어오면 (prefetch) 이후 새롭게 읽지 않아도 이미 읽어온 구간에서 연속적으로 데이터 처리가 가능하지만, interleaved 방식은 스레드 캐시로 읽어온 주소 구간을 모두 알뜰하게 사용하지 않고 현재 병렬 중인 SIMD ALU의 개수만큼 격차를 두어 그 다음을 사용해 상대적으로 캐시 효율이 떨어지기 때문이다.