[Lab-2] ISPC programming

소프트웨어학과 201720707 나용성

1. Run the example code

- Make a performance comparison

1) Between the scalar and vector versions

```
./sinx_interleaved
Elapsed time with SISD : 108239
Elapsed time with SIMD : 18716
```

Scalar version에서의 실행시간은 108239, Interleaved assignment방식의 vector version에서의 실행시간은 18176으로 vector version에서 약 6배 가량 실행속도가 빨라졌다. 한번에 programCount개의 x배열 원소에 대해 y를 계산해내고 memory locality도 만족하는 interleaved assignment 방식이므로 실행속도가 많이 빨라졌다.

2) Between the interleaved and blocked assignments of program instances

```
./sinx_blocked
Elapsed time with SISD : 143885
Elapsed time with SIMD : 36855
```

Blocked assignment방식의 vector version에서의 실행시간이 36855로 위의 18176에서의 실행시간보다 2배정도 느리다.

- 원인분석

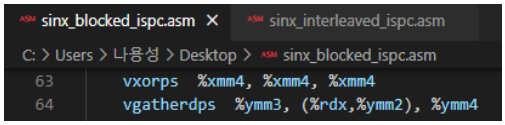
sinx_blocked_ispc.asm

```
565     cltq
566     vextractps  $3, %xmm2, (%rcx,%rax)
567     incl    %esi
568     cmpl    %esi, %r8d
569     jne .LBB1_7
570 .LBB1_8:
571     popq    %rbx
572     vzeroupper
573     retq
574 .Lfunc_end1:
575     .size   sinx, .Lfunc_end1-sinx
576
577     .ident  "clang version 12.0.1 (/usr
578     .section    ".note.GNU-stack","",@p
579
```

sinx_interleaved_ispc.asm

```
213     popq    %rbp
214     vzeroupper
215     retq
216 .Lfunc_end1:
217     .size   sinx, .Lfunc_end1-sinx
218                                     # --
219     .ident  "clang version 12.0.1 (/usr/local
220     .section    ".note.GNU-stack","",@progbit
221
```

그 원인은 interleaved assignment에선 memory locality가 있고 blocked assignment에선 memory locality가 없다는 이유도 있고 어셈블리 코드를 봤을 때 blocked assignment에서의 어셈블리 코드가 578줄로 interleaved assignment에서 220줄보다 두배에서 세배 정도 길기 때문에 실제로 실행하는 instruction이 많아 더 오래 걸리는 이유도 있을 것이다.



그리고 blocked assignment 에선 위에서 처럼 gather, scatter 가 필요하므로 실행시간이 interleaved assignment 에서보다 blocked assignment 에서 더 길어지게 된다.

2. Write ISPC code for finding a min and max value in a given array

- 실행결과

```
./min_max
min = 4, max = 1997
```

최솟값과 최대값을 탐색할 array x에는 0~1999사이의 int형 변수 1024개가 랜덤하게 생성되어 저장되는데 실행 결과 최솟값이 4, 최댓값이 1997이므로 제대로 최솟값 최댓값을 제대로 탐색했다고 볼 수 있다.

- 코드리뷰

```
export void min_ispc(
        uniform int N,
        uniform int x[],
        uniform int min_x[])
{
        // assume N % programCount = 0
        for (uniform int i=0; i<N; i+=programCount)
        {
                int idx = i + programIndex;
                uniform int min_temp = reduce_min(x[idx]);
                *min_x = *min_x < min_temp ? *min_x : min_temp;

        }
}
export void max_ispc(
        uniform int N,
        uniform int x[],
        uniform int max_x[])
{
        // assume N % programCount = 0
        for (uniform int i=0; i<N; i+=programCount)
        {
                int idx = i + programIndex;
                uniform int max_temp = reduce_max(x[idx]);
                *max_x = *max_x > max_temp ? *max_x : max_temp;
        }
}
```

SIMD를 이용하여 빠르게 최솟값, 최댓값을 탐색하고자 할 땐 ispc API인 reduce_min(int32 a), reduce_max(int32 a)를 사용하면 된다. reduce_min과 reduce_max는 현재 실행중인 program instance들 사이에서 각각 최솟값, 최댓값인 a를 계산해낸다. 그러므로 interleaved assignment에서 각각의 program instance들이 갖는 idx가 서로 다르므로 API의 parameter로 x[idx]를 전달하면 한번의 iteration에서 programCount개의 x 배열 원소 사이에서의 최솟값, 최댓값을 계산해낼 수 있고 계산된 최솟값, 최댓값과 지금까지 진행에서의 최솟값, 최댓값과 비교하여 조건에 맞을 시 저장하도록 하여 SIMD를 이용한 배열에서의 최솟값, 최댓값 계산이 가능하다.