과제 #1 채점기준

4190.414A 멀티코어 컴퓨팅(001)

September 27, 2022

1 문제 1: 컴퓨터에서의 수 표현 (20점)

다음은 `long double` 포맷에 대해 조교가 작성한 프로그램 예시이다. 이 외에도 다양한 방법이 있다.

```
#include <stdio.h>

#define TYPE long double
#define FORMAT "%Lf"

void print_byte(char x) {
  for (int i = 7; i >= 0; --i) {
    printf("%d", x & (1 << i) ? 1 : 0);
  }
}

int main() {
  TYPE x;
  scanf(FORMAT, &x);
  printf(FORMAT, x);
  printf("\n");
  for (int i = sizeof(x) - 1; i >= 0; --i) {
    print_byte(*((char*)&x + i));
  }
  printf("\n");
  return 0;
}
```

동작 (10점) 타입 별로 올바르게 작동하면 각 2점. `long double` 타입의 경우, 환경마다 다르게 구현되어있음을 고려해서 답이 다르더라도 조교가 보았을 때 적절한 구현이면 점수 부여.

보고서 (10점) 과제에서 제시한 질문에 대한 답이 적절히 있으면 각 5점.

2 문제 2: 부동소수점 연산 성능 (40점)

Single core에서 single precision 덧셈 연산을 한다고 가정하자. 이 때 theoretical peak FLOPS는 다음과 같다.

$$2(\mathrm{OP/cycle}) * 3.9(\mathrm{GHz}) = 7.8(\mathrm{GFLOPS})$$

실습 서버의 Xeon Gold 6230 은 active core 가 한 개일 때 turbo frequency 인 3.9 GHz 로 작동함을 유의하자. 또한, Skylake microachitecture 에서 FP 연산은 사이클당 2개를 실행할 수 있다. (링크의 Scheduler Ports & Execution Units 항목 참고)

가장 naive 한 구현 방식은 다음과 같다.

```
$ cat prob2_naive.c
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>

double get_time() {
  struct timeval tv;
  gettimeofday(&tv, 0);
  return tv.tv_sec + tv.tv_usec * 1e-6;
}

int main(int argc, char **argv) {
  long n = atol(argv[1]);
  double st = get_time();
  float a = 1, b = 2, c;
  for (long i = 0; i < n; ++i) {
    c = a + b;
  }
  double et = get_time();
  printf("N: %ld\n", n);
  printf("Elapsed time: %f sec\n", et - st);
  printf("Throughput: %f GFLOPS\n", n / (et - st) / 1e9);
  return 0;
}
$ gcc -g -o prob2_naive prob2_naive.c
$ ./a.out 10000000000
N: 10000000000
Elapsed time: 25.011827 sec
Throughput: 0.399811 GFLOPS
```

0.40 GFLOPS 정도로 이론 성능에 택도 없이 못 미치는 것을 알 수 있다. 원인을 파악하기 위해 어셈블리 코드를 살펴보자.

```
$ objdump -S -d prob2_naive
(...)
126e:        f3 0f 10 45 d4              movss  -0x2c(%rbp),%xmm0
1273:        f3 0f 58 45 d8              addss  -0x28(%rbp),%xmm0
1278:        f3 0f 11 45 dc              movss  %xmm0,-0x24(%rbp)
127d:        48 83 45 e0 01              addq   $0x1,-0x20(%rbp)
1282:        48 8b 45 e0                 mov    -0x20(%rbp),%rax
1286:        48 3b 45 e8                 cmp    -0x18(%rbp),%rax
128a:        7c e2                       jl     126e <main+0x61>
```

실제 덧셈 instruction인 addss 외에 xmm 레지스터로의 load/store, iteration count, 비교 연산, 점프 등 다른 instruction으로 인하여 제 성능을 내지 못하는 것을 볼 수 있다. 이론 성능에 가깝게 하려면 어떻게 해야할까? 다음과 같이 루프를 수정하고 어셈블리를 확인해보자.

```
$ cat prob2_improved.c
(...)
  float s0, s1, s2, s3, s4, s5, s6, s7;
  float x0, x1, x2, x3, x4, x5, x6, x7;
  s0 = s1 = s2 = s3 = s4 = s5 = s6 = s7 = 0;
  scanf("%f%f%f%f%f%f%f%f", &x0, &x1, &x2, &x3, &x4, &x5, &x6, &x7);
  double st = get_time();
  for (long i = 0; i < n / 8; ++i) {
    s0 += x0; s1 += x1; s2 += x2; s3 += x3;
    s4 += x4; s5 += x5; s6 += x6; s7 += x7;
  }
  double et = get_time();
  // 최적화를 막기 위해 결과를 출력
  printf("N: %ld, %f, %f, %f, %f, %f, %f, %f, %f\n",
    n, s0, s1, s2, s3, s4, s5, s6, s7);
(...)
$ gcc -g -O2 -o prob2_improved prob2_improved.c
$ ./prob2_improved 10000000000
(...)
Elapsed time: 1.352955 sec
Throughput: 7.391228 GFLOPS
$ objdump -S -d prob2_improved
(...)
11f0:        48 83 c0 01                 add    $0x1,%rax
11f4:        f3 41 0f 58 c7              addss  %xmm15,%xmm0
11f9:        f3 41 0f 58 ce              addss  %xmm14,%xmm1
11fe:        f3 41 0f 58 d5              addss  %xmm13,%xmm2
1203:        f3 41 0f 58 dc              addss  %xmm12,%xmm3
1208:        f3 41 0f 58 e3              addss  %xmm11,%xmm4
120d:        f3 41 0f 58 ea              addss  %xmm10,%xmm5
1212:        f3 41 0f 58 f1              addss  %xmm9,%xmm6
1217:        f3 41 0f 58 f8              addss  %xmm8,%xmm7
121c:        48 39 d0                    cmp    %rdx,%rax
121f:        7c cf                       jl     11f0 <main+0x110>
(...)
```

Unrolling을 통해 addss가 차지하는 비중을 높인 결과, 7.4 GFLOPS로 이론 성능(7.8 GFLOPS)에 가까운 성능을 얻을 수 있다. **이런 식으로 이론 성능과 차이가 나는 이유를 근거와 함께 제시하거나 이론 성능에 근접하도록 하면 좋은 점수를 받을 수 있다.**

채점 기준은 다음과 같다.

실험 (20점) 실험을 적절하게 수행했으면 20점 (성능 무관). 컴파일러 최적화 등으로 `add, mul` 연산이 사라졌으면 -10점. 실수 연산을 함수로 wrapping 한다든지 해서 연산과 무관한 코드가 너무 많이 성능 측정에 포함될 경우 -3점.

보고서 (20점) 10점을 기준으로, 논의가 잘 되어 있으면 +5 또는 +10, 내용이 너무 부실한 경우 -5 또는 -10. 논의의 예시로, 실험 결과에 대한 분석 및 원인 설명이 잘 되어 있으면 +5점, 개선 방안을 잘 설명했으면 +10점을 받을 수 있다. 물론 다른 방향의 논의도 가능하다.

3 문제 3: Vector Instruction (40점)

Single core에서 single precision 덧셈 연산을 AVX2를 이용해서 한다고 가정하자. 이 때 theoretical peak FLOPS는 다음과 같다.

$$2(\texttt{OP/cycle}) * 8(\texttt{float/OP}) * 3.8(\texttt{GHz}) = 60.8(\texttt{GFLOPS})$$

Frequency가 `3.8GHz` 임에 주의하자. Intel CPU는 AVX 사용 여부와 몇개의 코어에서 AVX를 사용하고 있는지에 따라 frequency가 달라진다. (링크의 Frequencies table 참고.) 또한, Skylake microarchitecture 에서 AVX2 instruction throughput 은 2 OP / cycle 이다. (링크 참고.)

아래와 같이 vector instruction을 사용해보자.

```
$ cat prob3_naive.c
(...)
  __m256 s = _mm256_set_ps(1, 2, 3, 4, 5, 6, 7, 8);
  __m256 x = _mm256_set_ps(1, 2, 3, 4, 5, 6, 7, 8);
  double st = get_time();
  for (long i = 0; i < n / 8; ++i) {
    s = _mm256_add_ps(s, x);
  }
  double et = get_time();
  // 최적화를 막기 위해 결과를 출력
  printf("N: %ld, %f\n", n, _mm256_cvtss_f32(s));
(...)
$ gcc -g -mavx2 -O2 prob3_naive.c
$ ./a.out 100000000000
(...)
Elapsed time: 12.839930 sec
Throughput: 7.788205 GFLOPS
$ objdump -S -d a.out
(...)
1160:        48 83 c0 01                 add    $0x1,%rax
1164:        c5 fc 58 c2                 vaddps %ymm2,%ymm0,%ymm0
1168:        48 39 d0                    cmp    %rdx,%rax
116b:        7c f3                       jl     1160 <main+0xa0>
(...)
```

성능은 약 7.8 GFLOPS로 vector instruction을 사용하지 않았을 때보다 별 이득이 없다. `vaddps` 외 다른 instruction 이 많고, `%ymm1` 레지스터의 dependence로 인해 pipeline이 100% 활용되지 않기 때문이다. 이를 해결하기 위해 문제 2에서 했던 것처럼 unrolling을 해보자.

```
$ cat prob3_improved.c
(...)
  float y0, y1, y2, y3, y4, y5, y6, y7;
```

```
  scanf("%f%f%f%f", &y0, &y1, &y2, &y3);
  scanf("%f%f%f%f", &y4, &y5, &y6, &y7);
  __m256 s0, s1, s2, s3, s4, s5, s6, s7;
  __m256 x0, x1, x2, x3, x4, x5, x6, x7;
  s0 = s1 = s2 = s3 = _mm256_set_ps(0, 0, 0, 0, 0, 0, 0, 0);
  s4 = s5 = s6 = s7 = _mm256_set_ps(0, 0, 0, 0, 0, 0, 0, 0);
  x0 = _mm256_set_ps(y0, y0, y0, y0, y0, y0, y0, y0);
  x1 = _mm256_set_ps(y1, y1, y1, y1, y1, y1, y1, y1);
  x2 = _mm256_set_ps(y2, y2, y2, y2, y2, y2, y2, y2);
  x3 = _mm256_set_ps(y3, y3, y3, y3, y3, y3, y3, y3);
  x4 = _mm256_set_ps(y4, y4, y4, y4, y4, y4, y4, y4);
  x5 = _mm256_set_ps(y5, y5, y5, y5, y5, y5, y5, y5);
  x6 = _mm256_set_ps(y6, y6, y6, y6, y6, y6, y6, y6);
  x7 = _mm256_set_ps(y7, y7, y7, y7, y7, y7, y7, y7);
  double st = get_time();
  for (long i = 0; i < n / 8 / 8; ++i) {
    s0 = _mm256_add_ps(s0, x0);
    s1 = _mm256_add_ps(s1, x1);
    s2 = _mm256_add_ps(s2, x2);
    s3 = _mm256_add_ps(s3, x3);
    s4 = _mm256_add_ps(s4, x4);
    s5 = _mm256_add_ps(s5, x5);
    s6 = _mm256_add_ps(s6, x6);
    s7 = _mm256_add_ps(s7, x7);
  }
  double et = get_time();
  // 최적화를 막기 위해 결과를 출력
  printf("%f %f %f %f\n",
    _mm256_cvtss_f32(s0), _mm256_cvtss_f32(s1),
    _mm256_cvtss_f32(s2), _mm256_cvtss_f32(s3));
  printf("%f %f %f %f\n",
    _mm256_cvtss_f32(s4), _mm256_cvtss_f32(s5),
    _mm256_cvtss_f32(s6), _mm256_cvtss_f32(s7));
(...)
$ gcc -g -mavx2 -O2 prob3_improved.c
$ ./a.out 100000000000
(...)
Elapsed time: 1.708480 sec
Throughput: 58.531564 GFLOPS
$ objdump -S -d a.out
(...)
1280:        48 83 c0 01                 add    $0x1,%rax
1284:        c4 c1 7c 58 c0              vaddps %ymm8,%ymm0,%ymm0
1289:        c4 c1 74 58 c9              vaddps %ymm9,%ymm1,%ymm1
128e:        c4 c1 6c 58 d2              vaddps %ymm10,%ymm2,%ymm2
1293:        c4 c1 64 58 db              vaddps %ymm11,%ymm3,%ymm3
1298:        c4 c1 5c 58 e4              vaddps %ymm12,%ymm4,%ymm4
```

```
129d:       c4 c1 54 58 ed              vaddps %ymm13,%ymm5,%ymm5
12a2:       c4 c1 4c 58 f6              vaddps %ymm14,%ymm6,%ymm6
12a7:       c4 c1 44 58 ff              vaddps %ymm15,%ymm7,%ymm7
12ac:       48 39 d0                    cmp    %rdx,%rax
12af:       7c cf                       jl     1280 <main+0x1a0>
(...)
```

성능은 약 58.5 GFLOPS으로 이론 성능(60.8 GFLOPS)에 근접해진다. 언롤링을 충분히 하지 않은 경우 더 낮은 성능이 나올 수 있음에 유의하자.

채점 기준은 다음과 같다.

동작 (20점) Vector instruction 을 잘 사용해서 실험을 수행했으면 20점. (성능 무관)

보고서 (20점) 10점을 기준으로, 논의가 잘 되어 있으면 +5 또는 +10. 내용이 부실한 경우 -5 또는 -10.