SSE Exam Solutions

1 2018 Q1

1.1 Part (b)

1.1.1 Code Segment 1

- This code cannot be vectorised using SSE.
- The code requires a dependency in both it's current and previous every iteration, The relience on the previous iteration is what's holding back the iteration, as we will not be able to access the previous updated iteration if we are computing four at a time.

1.1.2 Code Segment 2

- This code can be vectorised using SSE.
- The reason it can be is that the only dependency is the current value found in array[i]. This means that if we work on this 4 times at a time, there will not be a dependency issue there as they are mutually exclusive operations.
- Vectorising this requires us to change the starting position, since we will do four steps at a time but from the top of the array to the bottom, we must decrement by 3 at the start of the loop.

```
[a, b, c, d, 4, 5, 6, 7]
             ^^^^^^^^^^ 1st iteration
[0, 1, 2, 3, e, f, g, h]
 ^^^^^^^^^^ 2nd iteration
```

- As we know, we must also have a postloop for when the array is not divisible by 4. This post loop is the same as our initial loop given, we'll just need to keep track of the i variable so that we can finish the last elements, so the i variable will be put in scope of the two loops.
- An optimisation is to vectorise the constants outside of the loop, so that we do not recompute the same multiplier and SIZE for a vector each iteration.

```
void compute(float* array, int SIZE, float multiplier) {
    int i;
    __m128 v4multipler = _mm_set1_ps(multiplier);
    __m128 v4SIZE = _mm_set1_ps(SIZE);
    for (i = SIZE - 4; i >= 0; i-=4) {
        __m128 v4array = _mm_loadu_ps(&(array[i]));
        __m128 v4array_mul_v4multiplier = _mm_mul_ps(v4array, v4multipler);
        __m128 v4result = _mm_div_ps(v4array_mul_v4multiplier,v4SIZE);
        _mm_storeu_ps(&(array[i]), v4result);
    }
     for (/* i = i */; i >= 0; i--) {
        array[i] = (array[i] * multiplier) / SIZE;
    }
}
```

1.1.3 Code Segment 3

- This code can be vectorised using SSE.
- The assumptions needed to be made here, is that the array contains non negative integers.
- This code requires intrinsics for integers.
- Calculate max 4 at a time calculating of the array each time, at the end of the vectored loop select the maximum element horziontally out of the 4 lanes. And have a post loop to do the remaining elements not divisible by 4.

```
int compute(int* array, int SIZE) {
    // array is 16-byte aligned address
    __m128i v4max = _mm_setzero_si128(); // set all four lanes of vector to 0
    int i;
    for(i = 0; i < SIZE - 3; i+=4) {
        __m128i v4array = _mm_load_si128((__m128i*)&(array[i]));
        v4max = _mm_max_epi32(v4array, v4max);
    }
    // select the maximum value which is in one of the four lanes in v4max
    v4max = _mm_max_epi32(v4max, _mm_shuffle_epi32(v4max, _MM_SHUFFLE(0, 0, 3, 2)));
    v4max = _mm_max_epi32(v4max, _mm_shuffle_epi32(v4max, _MM_SHUFFLE(0, 0, 0, 1)));
    int max = _mm_cvtsi128_si32(v4max);
    // postloop for when the array is not divsible by 4
    for(/* i = i */; i < SIZE; i++) {
        if (array[i] > max) { max = array[i]; }
    }
    return max;
}
```

1.1.4 Code Segment 4

- Exam paper has a mistake with the second parameter with function, I will assume it is strcmp from standard C.
- This code can be vectorised using SSE.
- The code requires the use of string comparison intrinsics. We will load 16 bytes at a time because we are restricted to loading 128 bits with SSE. We do a check to see if there's a match, then we check where the match is. If there's no initial match, it'll check to see if there's a null character somewhere therefore false.
- The postloop ensures that if a string isn't divisible by 16, then it can do a postloop that was the same as our initial programme unvectorised programme.

```
bool strcmp_sse(char* string1, char* string2, int strlength) {
    int i;
    for(i = 0; i < strlength - 15; i+=16) {
        __m128i v16string1 = _mm_load_si128((__m128i*)&(string1[i]));
        __m128i v16string2 = _mm_load_si128((__m128i*)&(string2[i]));
        if (_mm_cmpistrc(v16string1,v16string2,_SIDD_UBYTE_OPS |
                                               _SIDD_CMP_EQUAL_EACH |
                                               _SIDD_NEGATIVE_POLARITY |
                                               _SIDD_LEAST_SIGNIFICANT)) {
            int index = _mm_cmpistri(v16string1,v16string2, _SIDD_UBYTE_OPS |
                                                            _SIDD_CMP_EQUAL_EACH |
                                                            _SIDD_NEGATIVE_POLARITY |
                                                            _SIDD_LEAST_SIGNIFICANT);
            if (string1[index] != string2[index])
                return false;
        } else if (_mm_cmpistrz(v16string1,v16string2,_SIDD_UBYTE_OPS |
                                                      _SIDD_CMP_EQUAL_EACH |
                                                      _SIDD_NEGATIVE_POLARITY |
                                                      _SIDD_LEAST_SIGNIFICANT)) {
            return false;
        }
    }
    // postloop
    // i = i;
    for (; i < strlength; i++) {
        if (string1[i] != string2[i])
            return false;
    }
    return true;
}
```

2 2017 Q1

2.1 Part (b)

TODO

3 2016 Q1

3.1 Part (c)

TODO