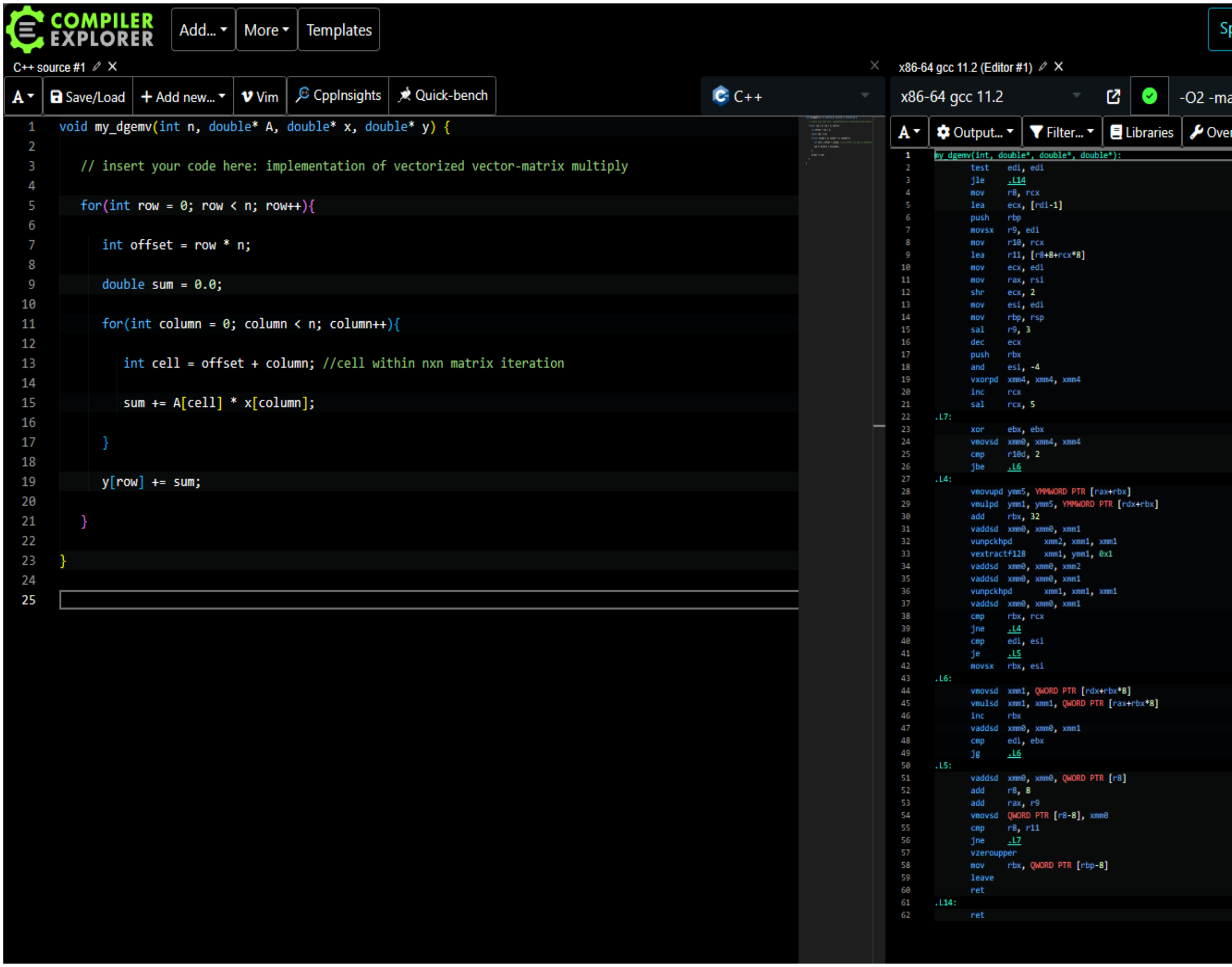
CSC656 CP3 writeup

Runtime (s)							
Problem Size	blas	basic	vectorized	omp-1	omp-4	omp-16	omp-64
1024	0.00016	0.0009	0.00024	0.0009	0.00047	0.00028	0.00065
2048	0.00051	0.00358	0.0011	0.00357	0.00184	0.00167	0.00179
4096	0.00410	0.01455	0.00476	0.01454	0.00432	0.00333	0.00332
8192	0.01838	0.0583	0.01963	0.05819	0.01544	0.01304	0.01341
16384	0.06815	0.23697	0.07996	0.23310	0.06004	0.05635	0.05827

MFLOP/s							
Problem Size	blas	basic	vectorized	omp-1	omp-4	omp-16	omp-64
1024	13051.57	2321.365	8758.753	2342.598	4420.333	7601.902	3205.512
2048	16353.47	2342.743	7600.29	2351.923	4546.888	5035.478	4694.002
4096	8187.42	2306.812	7054.885	2307.514	7768.228	10080.596	10108.252
8192	7304.32	2302.219	6838.273	2306.389	8691.11	10290.996	10006.699
16384	7878.1	2265.551	6714.636	2303.132	8941.44	9528.232	9213.231

% memory bandwidth utilized							
Problem Size	blas	basic	vectorized	omp-1	omp-4	omp-16	omp-64
1024	0.02489	0.00443	0.01671	0.00447	0.00843	0.01450	0.00611
2048	0.01560	0.00223	0.00725	0.00224	0.00434	0.0048	0.00448
4096	0.0039	0.0011	0.00336	0.0.0011	0.0037	0.00481	0.00482
8192	0.00174	0.00055	0.00163	0.00055	0.00207	0.00245	0.00239
16384	0.00094	0.00027	0.0008	0.00027	0.00107	0.00114	0.0011

Screenshot of dgemv_vectorized.cpp turned into assembly in godbolt.org:



```cpp
void my_dgemv(int n, double* A, double* x, double* y) {

  // insert your code here: implementation of vectorized vector-matrix multiply

  for(int row = 0; row < n; row++){

    int offset = row * n;

    double sum = 0.0;

    for(int column = 0; column < n; column++){

      int cell = offset + column; //cell within nxn matrix iteration

      sum += A[cell] * x[column];

    }

    y[row] += sum;

  }

}

```

```asm
my_dgemv(int, double*, double*, double*):
        test    edi, edi
        jle     .L14
        mov     r8, rcx
        lea     ecx, [rdi-1]
        push    rbp
        movsx   r9, edi
        mov     r10, rcx
        lea     r11, [r8+8+rcx*8]
        mov     ecx, edi
        mov     rax, rsi
        shr     ecx, 2
        mov     esi, edi
        mov     rbp, rsp
        sal     r9, 3
        dec     ecx
        push    rbx
        and     esi, -4
        vxorpd  xmm4, xmm4, xmm4
        inc     rcx
        sal     rcx, 5
.L7:
        xor     ebx, ebx
        vmovsd  xmm0, xmm4, xmm4
        cmp     r10d, 2
        jbe     .L6
.L4:
        vmovupd ymm5, YMMWORD PTR [rax+rbx]
        vmulpd  ymm1, ymm5, YMMWORD PTR [rdx+rbx]
        add     rbx, 32
        vaddsd  xmm0, xmm0, xmm1
        vunpckhpd       xmm2, xmm1, xmm1
        vextractf128    xmm1, ymm1, 0x1
        vaddsd  xmm0, xmm0, xmm2
        vaddsd  xmm0, xmm0, xmm1
        vunpckhpd       xmm1, xmm1, xmm1
        vaddsd  xmm0, xmm0, xmm1
        cmp     rbx, rcx
        jne     .L4
        cmp     edi, esi
        je      .L5
        movsx   rbx, esi
.L6:
        vmovsd  xmm1, QWORD PTR [rdx+rbx*8]
        vmulsd  xmm1, xmm1, QWORD PTR [rax+rbx*8]
        inc     rbx
        vaddsd  xmm0, xmm0, xmm1
        cmp     edi, ebx
        jg      .L6
.L5:
        vaddsd  xmm0, xmm0, QWORD PTR [r8]
        add     r8, 8
        add     rax, r9
        vmovsd  QWORD PTR [r8-8], xmm0
        cmp     r8, r11
        jne     .L7
        vzeroupper
        mov     rbx, QWORD PTR [rbp-8]
        leave
        ret
.L14:
        ret
```

Question answers:

Q1.

At n=16384 between basic and vectorized, in terms of MFLOP/s, vectorized has a much better performance. The difference is 6714.636(vect) - 2265.551(basic) = 4449.085

In terms of memory utilization, vectorized was also better using 0.0008% vs 0.0027%. A difference of 0.0019%

Q2.

At n=16384 between basic and openMP-4, in terms of MFLOP/s, omp-4 has the better performance. The difference is 8941.44(omp-4) - 2265.551(basic) = 6675.889

In terms of memory utilization, basic had the better performance better using 0.0027% vs 0.00107%. A difference of 0.0080%

Q3.

From 1 -> 4 threads, the speedup is:
0.23310 / 0.06004 = 3.8824

From 1 -> 16 threads, the speedup is:
0.23310 / 0.05635 = 4.1366

From 1 -> 64 threads, the speedup is:
0.23310 / 0.05827 = 4.0003