

# HPC-II Spring 2009 – Homework 1

Robert van Engelen

Due date: April 7, 2009

For a detailed SSE intrinsics overview, see

[http://www.cs.fsu.edu/~engelen/courses/HPC-adv/intref\\_cls.pdf](http://www.cs.fsu.edu/~engelen/courses/HPC-adv/intref_cls.pdf)

1. Convert the following code into vectorized form using the SSE intrinsics and 128bit SSE registers as discussed in class:

```
float a[N], b[N];
for (i = 0; i < N; i++)
    b[i] = a[2*i]+a[2*i+1];
```

Note: you may assume that  $N$  is a multiple of 4 and  $a$  and  $b$  are aligned to 16-byte addresses.

2. Convert the following code into vectorized form using the SSE intrinsics and 128bit SSE registers as discussed in class:

```
double a[2*N+1], b[2*N+1], c[N];
for (i = 0; i < N; i++)
    c[i] = sqrt(a[2*i]*b[2*i]+a[2*i+1]*b[2*i+1]);
```

3. Challenge question for bonus: Convert the following code into vectorized form using the SSE intrinsics and 128bit SSE registers as discussed in class:

```
char a[N], short s = 0;
for (i = 0; i < N; i++)
    if (a[i] > 32)
        s++;
```

## Solutions

- Use the horizontal add

```
_mm_hadd_ps(x, y) = |x0+x1|x2+x3|y0+y1|y2+y3|
```

to implement:

```
float a[N], b[N];
__m128 *av = (__m128*)a;
__m128 *bv = (__m128*)b;
for (i = 0; i < N/4; i++)
    bv[i] = _mm_hadd_ps(av[2*i], av[2*i+1]);
```

- Also use the horizontal add here, but after packed multiplication of a with b:

```
double a[2*N+1], b[2*N+1], c[N];
__m128d *av = (__m128d*)a;
__m128d *bv = (__m128d*)b;
__m128d *cv = (__m128d*)c;
for (i = 0; i < N/2; i++)
    cv[i] = _mm_sqrt_pd(_mm_hadd_pd(_mm_mul_pd(av[2*i], bv[2*i]),
                                     _mm_mul_pd(av[2*i+1], bv[2*i+1])));
```

- Encode control flow using logic to update multiple sums, finally add the sums together:

```
char a[N], short s;
short sv[8];
__m128i av = (_m128i*)a;
__m128i cv = _mm_set1_epi8(32);
__m128i zv = _mm_setzero_si128();
__m128i sv1 = zv;
__m128i sv2 = zv;
for (i = 0; i < N/16; i++)
{
    __m128i gv, gv1, gv2;
    gv = _mm_cmpgt_epi8(av[i], cv);
    gv = _mm_sub_epi8(zv, gv); // or use _mm_and_epi8 with vector of ones
    gv1 = _mm_unpackhi_epi8(zv, gv);
    gv2 = _mm_unpacklo_epi8(zv, gv);
    sv1 = _mm_add_epi16(sv1, gv1);
    sv2 = _mm_add_epi16(sv2, gv2);
}
```

```

_mm_store_si128(sv, sv1);
s = (short)sv[0] + (short)sv[1] + (short)sv[2] + (short)sv[3]
  + (short)sv[4] + (short)sv[5] + (short)sv[6] + (short)sv[7];
_mm_store_si128(sv, sv2);
s += (short)sv[0] + (short)sv[1] + (short)sv[2] + (short)sv[3]
  + (short)sv[4] + (short)sv[5] + (short)sv[6] + (short)sv[7];

```

The following alternative works and is correct, but the overall strategy is not advised. Similar solutions may suffer overflow in the horizontal adds when summing (in this case we just sum 1's, so its OK).

```

char a[N], short s = 0;
__m128i av = (_m128i*)a;
__m128i cv = _mm_set1_epi8(32);
__m128i z1 = _mm_set1_epi8(1);
for (i = 0; i < N/16; i++)
{ __m128i gv, tv;
  char t[16];
  gv = _mm_cmpgt_epi8(av[i], cv);
  gv = _mm_and_epi8(z1, gv);
  tv = _mm_hadd_epi8(tv, tv);
  tv = _mm_hadd_epi8(tv, tv);
  tv = _mm_hadd_epi8(tv, tv);
  tv = _mm_hadd_epi8(tv, tv);
  _mm_store_si128(t, tv);
  s += (short)t[0];
}

```