Introduction to SSE

A wide range of software applications, including graphics, MPEG video, music synthesis and more, show many common fundamental characteristics:

- Small data types (e.g., 8-bit pixels, 16-bit audio samples)
- Small, highly repetitive loops
- Computationally-intensive algorithms
- Frequent multiplies and accumulates
- Highly parallel operations

The Intel **Streaming SIMD Extensions (SSE)** comprise a set of extensions to the Intel x86 architecture that is designed to greatly enhance the performance of advanced media and communication applications.

In class, you saw actual Intel SSE *assembly instructions*; here we'll demonstrate the use of compiler *builtins* that map one-to-one to SSE assembly.

Example: Inner Product

Recall that the *inner product* of two vectors $\mathbf{x} = (x_1, x_2, ..., x_k)$ and $\mathbf{y} = (y_1, y_2, ..., y_k)$ is defined as follows: $\mathbf{x} \bullet \mathbf{y} = x_1y_1 + x_2y_2 + ... + x_ky_k$. Normally, we could compute the inner product as follows:

```
float x[k];  float y[k];        // operand vectors of length k
float inner_product = 0.0;      // accumulator

for (int i = 0; i < k; i++)
        inner_product += x[i] * y[i];
```

To take advantage of SSE operations, we can rewrite this code using the SSE intrinsics:

```
// floating point vector type
typedef float v4sf __attribute__ ((vector_size (4*sizeof(float))));

float x[k];  float y[k];        // operand vectors of length k
float inner_product = 0.0, temp[4];
v4sf acc, X, Y;                 // 4x32-bit float registers

acc = __builtin_ia32_xorps(acc, acc); // zero the accumulator

for (int i = 0; i < (k - 3); i += 4) {
        X = __builtin_ia32_loadups(&x[i]); // load groups of four floats
        Y = __builtin_ia32_loadups(&y[i]);
        acc = __builtin_ia32_addps(acc, __builtin_ia32_mulps(X, Y));
}

__builtin_ia32_storeups(temp, acc); // add the accumulated values
inner_product = temp[0] + temp[1] + temp[2] + temp[3];

for (; i < k; i++)              // add up the remaining floats
        inner_product += x[i] * y[i];
```

Problems

1. **Matrix-Vector Multiplication**

   Write a function **mv_multiply** that multiplies a matrix and a vector. Recall that if A is a $k \times k$ matrix, B is a k-vector, and $A * B = C$, then C is a k-vector where $C_i = \Sigma_{j=1}^{k} A_{i,j} * B_j$.

   Code *without* SSE intrinsics is on the web site (under section 13). The original should have a run-time of around 9 seconds, while a basic solution using the SSE intrinsics should run in about 4 seconds.

2. **Mandelbrot Sets**

   Write a function that determines whether a series of points in a complex plane are inside the Mandelbrot set. Let $f_c(z) = z^2 + c$. Let $f_c^n(z)$ be the results of composing $f_c(z)$ with itself n times. (So $f_c^n(z) = f_c^{n-1}(f_c(z))$ and $f_c^1(z) = f_c(z)$.) Then, a point (x, y) is considered to be in the Mandelbrot set, if for a complex number $c = x + yi$, $f_c^n(0)$ does not diverge to infinity as n approaches infinity.

   Code *without* SSE intrinsics is on the web site (under section 13). The original should have a run-time of around 10 seconds, while a basic solution using the SSE intrinsics should run in about 5 seconds.

   The code makes a simplifying assumption that if $|f_c^{200}(0)| < 4$, then it does not diverge to infinity.

Useful Intrinsics

```
v4sf __builtin_ia32_loadups(float *)
void __builtin_ia32_storeups(float *, v4sf)
v4sf __builtin_ia32_addps(v4sf, v4sf) // parallel arithmetic ops
v4sf __builtin_ia32_subps(v4sf, v4sf)
v4sf __builtin_ia32_mulps(v4sf, v4sf)
v4sf __builtin_ia32_divps(v4sf, v4sf)
v4sf __builtin_ia32_xorps(v4sf, v4sf) // 128-bit XOR
                                      // Can be used to quickly generate 0s
```

Notes

1. Compile your code with the command: **gcc -Wall -O2 -o execfile -msse filename.c**
2. Run your program using: **./execfile**
3. For further details about SSE intrinsics and the gcc compiler's implementation of them, visit:

   `http://www.intel80386.com/simd/mmx2-doc.html`
   `http://gcc.gnu.org/onlinedocs/gcc-3.4.0/gcc/X86-Built-in-Functions.html`
   `http://ds9a.nl/gcc-simd/example.html`