SIMD in DSP-Algorithmen

Tim Blechmann
TimBlechmann@gmx.net

December 19, 2005

Compiler und Sprachen

- Zur Zeit kann kein mir bekannter Compiler guten SIMD-Code generieren.
  - gcc-4 hat eine (einfache) Autovektorisierung
  - icc schafft unter gewissen Umständen vektorisierten Code
- Algorithmen müssen für SIMD eventuell anders foruliert werden, um effizient ablaufen zu können.
- Sprachen unterstützen nicht alle Features der Hardware.

Streaming SIMD Extensions

- SSE Instruktionen arbeiten auf 32bit floats
- 8 128-bit registers: xmm0-xmm7
- Operationen für einfache Skalare (Xss, 1 float) und für parallele Skalare (Xps, 4 floats)

Optimierbare Algorithmen

- Lineare Funktionen
- Linearisierbare Funktionen: nicht linear geschrieben, aber linearisierbar
- Akkumulierende Funktionen: akkumulieren Daten eines Vektors
- Spezielle Instruktionen: Einsatz einer bestimmten Instruktion, die die Sprache nicht kennt

Testsystem

- Pentium-M 750, 1.86 GHz, 2MB cache
- Linux 2.6.12-rc5-RT-V0.7.47-12
- oprofile: Linux system profiler (http://oprofile.sourceforge.net/), CPU_CLK_UNHALTED
- gcc-3.4 (-msse -march=pentium-m -mfpmath=sse -funroll-loops -O2)
- float[64], 64 Samples entsprechen 1.3 ms bei 44100 kHz

Lineare Funktionen: addvec

```
void addvec(float * dest, float * src, unsigned int n)
{
    while (n--)
        *dest++ += *src++;
}

void addvec_8(float * dest, float * src, unsigned int n)
{
    for (unsigned int loops = n>>3; loops--;
         dest+=8, src+=8)
    {
        dest[0] += src[0], dest[1] += src[1],
        dest[2] += src[2], dest[3] += src[3],
        dest[4] += src[4], dest[5] += src[5],
        dest[6] += src[6], dest[7] += src[7];
    }
}
```

Lineare Funktionen: addvec_simd

```
void addvec_simd(float *dst, const float *src, unsigned int n)
{
    asm(
        "shr        $4, %0                  \n"
        /* loop: *dst += *src */
        "1:                                 \n"
        "movaps     (%2,%3),%%xmm0          \n"
        "addps      (%1,%3),%%xmm0          \n"
        "movaps     %%xmm0,(%2,%3)          \n"

        "movaps     16(%2,%3),%%xmm1        \n"
        "addps      16(%1,%3),%%xmm1        \n"
        "movaps     %%xmm1,16(%2,%3)        \n"

        "movaps     32(%2,%3),%%xmm2        \n"
        "addps      32(%1,%3),%%xmm2        \n"
        "movaps     %%xmm2,32(%2,%3)        \n"

        "movaps     48(%2,%3),%%xmm3        \n"
        "addps      48(%1,%3),%%xmm3        \n"
        "movaps     %%xmm3,48(%2,%3)        \n"
        "addl       $64,%3                  \n"
        "loop       1b                      \n"
        :
        : "c"(n), "r"(src), "r"(dst), "r"(0)
        : "%xmm0","%xmm1","%xmm2","%xmm3"  );
}
```

Lineare Funktionen: Benchmarks

- zerovec:

```
10074     2.2526  dsp_tests       zerovec(float*, unsigned int)
7517      1.6808  dsp_tests       zerovec_8(float*, unsigned int)
4364      0.9758  dsp_tests       zerovec_simd(float*, unsigned int)
```

- setvec:

```
7816      1.7477  dsp_tests       setvec(float*, float, unsigned int)
6715      1.5015  dsp_tests       setvec_8(float*, float, unsigned int)
5376      1.2021  dsp_tests       setvec_simd(float*, float, unsigned int)
```

- addvec:

```
19348     4.3263  dsp_tests       addvec(float*, float*, unsigned int)
16021     3.5824  dsp_tests       addvec_8(float*, float*, unsigned int)
7752      1.7334  dsp_tests       addvec_simd(float*, float const*, unsigned int)
```

- times_vec_scalar:

```
20130     4.5012  dsp_tests       times_vec_scalar(float*, float*, float, unsigned int)
17826     3.9860  dsp_tests       times_vec_scalar_8(float*, float*, float, unsigned int)
6010      1.3439  dsp_tests       times_vec_scalar_simd(float*, float*, float, unsigned int)
```

Linearisierbare Funktionen: clipvec

```
template <class T>
inline T clip(T t, T low, T hi)
{
    if (t < low) return low;
    if (t > hi) return hi;
    return t;
}
void clip_vec(float * dest, float * src, float * low, float * hi, int n)
{
    while (n--)
        *dest++ = clip(*src++, *low++, *hi++);
}
void clip_vec_8(float * dest, float * src, float * low, float * hi, int n)
{
    for (unsigned int loops = n>>3; loops--; dest+=8, src+=8, low+=8, hi+=8)
    {
        dest[0] = clip(src[0], low[0], hi[0]);
        dest[1] = clip(src[1], low[1], hi[1]);
        dest[2] = clip(src[2], low[2], hi[2]);
        dest[3] = clip(src[3], low[3], hi[3]);
        dest[4] = clip(src[4], low[4], hi[4]);
        dest[5] = clip(src[5], low[5], hi[5]);
        dest[6] = clip(src[6], low[6], hi[6]);
        dest[7] = clip(src[7], low[7], hi[7]);
    }
}
```

Linearisierbare Funktionen: clipvec_simd

```
void clip_vec_simd(float * dest, float * src, float * low, float * hi, int n)
{
    asm(
        "push     %%ebx                    \n" /* clobber ebx */
        "xorl     %%ebx,%%ebx              \n"
        "shrl     $4, %4                   \n"

        /* loop: *dest = min ( max (lo, *src), hi )*/
        "1:                                \n"
        "movaps   (%0,%%ebx), %%xmm0       \n"
        "maxps    (%3,%%ebx), %%xmm0       \n"
        "minps    (%2,%%ebx), %%xmm0       \n"
        "movaps   %%xmm0, (%1,%%ebx)       \n"

        "movaps   16(%0,%%ebx), %%xmm1     \n"
        "maxps    16(%3,%%ebx), %%xmm1     \n"
        "minps    16(%2,%%ebx), %%xmm1     \n"
        "movaps   %%xmm1, 16(%1,%%ebx)     \n"
```

Linearisierbare Funktionen: clipvec_simd

```
        "movaps     32(%0,%%ebx), %%xmm2 \n"
        "maxps      32(%3,%%ebx), %%xmm2 \n"
        "minps      32(%2,%%ebx), %%xmm2 \n"
        "movaps     %%xmm2, 32(%1,%%ebx) \n"

        "movaps     48(%0,%%ebx), %%xmm3 \n"
        "maxps      48(%3,%%ebx), %%xmm3 \n"
        "minps      48(%2,%%ebx), %%xmm3 \n"
        "movaps     %%xmm3, 48(%1,%%ebx) \n"

        "addl       $64, %%ebx           \n"
        "loop       1b                   \n"

        "pop        %%ebx                \n"
        :
        :"S"(src), "D"(dest), "d"(hi), "a"(low), "c"(n)
        :"%xmm0", "%xmm1", "%xmm2", "%xmm3"
        );
```

Linearisierbare Funktionen: sgn_vec

```
void sgn_vec(float * dest, float * src, unsigned int n)
{
    while (n--)
    {
        float sample = *src++;
        if (sample > 0)
            *dest++ = 1.f;
        else if (sample < 0)
            *dest++ = -1.f;
        else
            *dest++ = 0;
    };
}
```

Linearisierbare Funktionen: sgn_vec_simd

```
void sgn_vec_simd(float * dest, float * src, unsigned int n)
{
    asm(
        ".section   .rodata                 \n"
        ".align 16                          \n"
        "2:                                 \n"
        ".long  0x3F800000                  \n" /* bitmask */
        ".long  0x3F800000                  \n" /* 1.0 */
        ".long  0x3F800000                  \n"
        ".long  0x3F800000                  \n"

        "3:                                 \n"
        ".align 16                          \n"
        ".long  2147483648                  \n" /* bitmask */
        ".long  2147483648                  \n" /* 0x80000000 */
        ".long  2147483648                  \n"
        ".long  2147483648                  \n"

        ".text                              \n"

        "movaps     (2b), %%xmm6            \n" /* xmm6 = 1.0 */
        "movaps     (3b), %%xmm5            \n" /* xmm5 = bitmask for sign */
        "xorps      %%xmm7,%%xmm7           \n" /* xmm7 = 0 */
        "shrl       $3, %2                  \n"
```

Linearisierbare Funktionen: sgn_vec_simd

```
/* loop: *dest = sgn(*src) */
"1:                                    \n"
"movaps     (%0,%3), %%xmm0            \n"
"movaps     %%xmm0, %%xmm1             \n"
"cmpneqps   %%xmm7, %%xmm0             \n"  /* xmm0 = (xmm0 != 0) */
"andps      %%xmm5, %%xmm1             \n"  /* xmm1 = sign bitmask */
"andps      %%xmm6, %%xmm0             \n"  /* xmm0 = abs bitmask */
"orps       %%xmm1, %%xmm0             \n"
"movaps     %%xmm0,(%1,%3)             \n"

"movaps     16(%0,%3), %%xmm2          \n"
"movaps     %%xmm2, %%xmm3             \n"
"cmpneqps   %%xmm7, %%xmm2             \n"
"andps      %%xmm5, %%xmm3             \n"
"andps      %%xmm6, %%xmm2             \n"
"orps       %%xmm3, %%xmm2             \n"
"movaps     %%xmm2,16(%1,%3)           \n"

"addl       $32, %3                    \n"
"loop       1b                         \n"
: :"r"(src), "r"(dest), "c"(n), "r"(0)
:"%xmm1", "%xmm2", "%xmm3", "%xmm4",
"%xmm5", "%xmm6", "%xmm7", "%xmm0"
);
```

Linearisierbare Funktionen: testcopyvec

```
inline float undenormalize(float f)
{
    if ((((*(unsigned int*)&(f))&0x60000000) == 0) ||
        (((*(unsigned int*)&(f))&0x60000000) == 0x60000000))
        return 0;
    else
        return f;
}

void testcopyvec(float * dest, float * src, unsigned int n)
{
    while (n--)
        *dest++ = undenormalize(*src++);
}
```

Linearisierbare Funktionen: testcopyvec_simd

```
void testcopyvec_simd(float *dst, const float *src, unsigned int n)
{
    asm(
        ".section    .rodata                \n"
        ".align 16                          \n"
        "2:                                 \n"
        ".long   1610612736                 \n" /* bitmask */
        ".long   1610612736                 \n" /* 0x60000000 */
        ".long   1610612736                 \n"
        ".long   1610612736                 \n"

        ".text                              \n"
        "movaps     (2b), %%xmm0            \n" /* xmm0 = bitmask */
        "xorps      %%xmm1, %%xmm1          \n" /* xmm1 = 0x0     */
        "shr        $4, %0                  \n"

        "1:                                 \n"
        "movaps     (%1,%3), %%xmm2         \n"
        "movaps     %%xmm2, %%xmm3          \n"
        "andps      %%xmm0, %%xmm3          \n"
        "movaps     %%xmm3, %%xmm4          \n"
        "cmpneqps   %%xmm0, %%xmm3          \n"
        "cmpneqps   %%xmm1, %%xmm4          \n"
        "andps      %%xmm4, %%xmm3          \n"
        "andps      %%xmm3, %%xmm2          \n"
        "movaps     %%xmm2, (%2,%3)         \n"
```

Linearisierbare Funktionen: testcopyvec_simd

```
"movaps     16(%1,%3), %%xmm2       \n"
"movaps     %%xmm2, %%xmm3          \n"
"andps      %%xmm0, %%xmm3          \n"
"movaps     %%xmm3, %%xmm4          \n"
"cmpneqps   %%xmm0, %%xmm3          \n"
"cmpneqps   %%xmm1, %%xmm4          \n"
"andps      %%xmm4, %%xmm3          \n"
"andps      %%xmm3, %%xmm2          \n"
"movaps     %%xmm2, 16(%2,%3)       \n"

"movaps     32(%1,%3), %%xmm2       \n"
"movaps     %%xmm2, %%xmm3          \n"
"andps      %%xmm0, %%xmm3          \n"
"movaps     %%xmm3, %%xmm4          \n"
"cmpneqps   %%xmm0, %%xmm3          \n"
"cmpneqps   %%xmm1, %%xmm4          \n"
"andps      %%xmm4, %%xmm3          \n"
"andps      %%xmm3, %%xmm2          \n"
"movaps     %%xmm2, 32(%2,%3)       \n"
```

Linearisierbare Funktionen: testcopyvec_simd

```
        "movaps     48(%1,%3), %%xmm2       \n"
        "movaps     %%xmm2, %%xmm3          \n"
        "andps      %%xmm0, %%xmm3          \n"
        "movaps     %%xmm3, %%xmm4          \n"
        "cmpneqps   %%xmm0, %%xmm3          \n"
        "cmpneqps   %%xmm1, %%xmm4          \n"
        "andps      %%xmm4, %%xmm3          \n"
        "andps      %%xmm3, %%xmm2          \n"
        "movaps     %%xmm2, 48(%2,%3)       \n"

        "addl       $64,%3                  \n"
        "decl       %0                      \n"
        "jne        1b                      \n"
        :
        :"c"(n), "r"(src), "r"(dst), "r"(0)
        :"%xmm0", "%xmm1", "%xmm2", "%xmm3", "%xmm4");
}
```

Linearisierbare Funktionen: Benchmarks

- clip_vec:

```
55689    12.4523  dsp_tests       clip_vec(float*, float*, float*, float*, unsigned int)
40583     9.0745  dsp_tests       clip_vec_8(float*, float*, float*, float*, unsigned int)
10367     2.3181  dsp_tests       clip_vec_simd(float*, float*, float*, float*, unsigned int)
```

- testcopyvec:

```
40496     9.0551  dsp_tests       testcopyvec(float*, float*, unsigned int)
13851     3.0971  dsp_tests       testcopyvec_simd(float*, float const*, unsigned int)
```

- sgn_vec:

```
46913    10.4900  dsp_tests       sgn_vec(float*, float*, unsigned int)
15473     3.4598  dsp_tests       sgn_vec_simd(float*, float*, unsigned int)
```

Akkumulierende Funktionen: peak_rms

```
void peak_rms(const float * src, unsigned int n, float & peak, float & rms)
{
    float l_sum = 0;
    while (n--)
    {
        float f = fabsf(*src++);
        l_sum += f*f;
        if (f < peak)
            peak = f;
    }
    rms += l_sum;
}
```

Analog: peak_rms_8

Linearisierbare Funktionen: peak_rms_simd

```
void peak_rms_simd(float * src, unsigned int n, float & peak, float & sum)
{
    asm(
        ".section    .rodata                \n"
        ".align 16                          \n"
        "2:                                 \n"
        ".long   2147483647                 \n" /* bitmask for abs */
        ".long   2147483647                 \n" /* 0x7fffffff */
        ".long   2147483647                 \n"
        ".long   2147483647                 \n"

        ".text                              \n"

        "movaps     (2b),%%xmm7             \n" /* load abs mask */
        "xorps      %%xmm6,%%xmm6           \n" /* set peak to 0 */
        "movss      (%1),%%xmm6             \n" /* load current peak */

        "shrl       $4,%3                   \n"
        "xorps      %%xmm0,%%xmm0           \n" /* set l_sum to 0 */
        "movss      (%0),%%xmm0             \n" /* set l_sum to sum */
```

Linearisierbare Funktionen: peak_rms_simd

```
"1:                                     \n"
"movaps     (%2),%%xmm1                 \n"
"andps      %%xmm7,%%xmm1               \n"  /* absolute value */
"maxps      %%xmm1,%%xmm6               \n"  /* update maximum */
"mulps      %%xmm1,%%xmm1               \n"  /* square */
"addps      %%xmm1,%%xmm0               \n"  /* add square to l_sum */

"movaps     16(%2),%%xmm2               \n"
"andps      %%xmm7,%%xmm2               \n"
"maxps      %%xmm2,%%xmm6               \n"
"mulps      %%xmm2,%%xmm2               \n"
"addps      %%xmm2,%%xmm0               \n"

"movaps     32(%2),%%xmm3               \n"
"andps      %%xmm7,%%xmm3               \n"
"maxps      %%xmm3,%%xmm6               \n"
"mulps      %%xmm3,%%xmm3               \n"
"addps      %%xmm3,%%xmm0               \n"

"movaps     48(%2),%%xmm4               \n"
"andps      %%xmm7,%%xmm4               \n"
"maxps      %%xmm4,%%xmm6               \n"
"mulps      %%xmm4,%%xmm4               \n"
"addps      %%xmm4,%%xmm0               \n"

"addl       $64, %1                     \n"
"loop       1b                          \n"
```

Linearisierbare Funktionen: peak_rms_simd

```
/* sum up %%xmm0 */
"movhlps    %%xmm0,%%xmm2          \n"
"movaps     %%xmm0,%%xmm3          \n"
"movaps     %%xmm2,%%xmm4          \n"
"shufps     $81,%%xmm3,%%xmm3      \n"
"shufps     $81,%%xmm4,%%xmm4      \n"

"addss      %%xmm0,%%xmm2          \n"
"addss      %%xmm2,%%xmm3          \n"
"addss      %%xmm3,%%xmm4          \n"

/* update sum */
"movss      %%xmm4,(%0)            \n"
```

Linearisierbare Funktionen: peak_rms_simd

```
        /* find maximum of %xmm6 */
        "movhlps    %%xmm6,%%xmm2       \n"
        "movaps     %%xmm6,%%xmm3       \n"
        "movaps     %%xmm2,%%xmm4       \n"
        "shufps     $81,%%xmm3,%%xmm3   \n"
        "shufps     $81,%%xmm4,%%xmm4   \n"

        "maxss      %%xmm2,%%xmm3       \n"
        "maxss      %%xmm3,%%xmm4       \n"
        "maxss      %%xmm4,%%xmm6       \n"

        /* update peak */
        "movss      %%xmm6,(%1)         \n"

        :
        :"r"(&sum), "r"(&peak), "r"(src), "c"(n)
        :"%xmm0", "%xmm1", "%xmm2", "%xmm3",
        "%xmm4", "%xmm5", "%xmm6", "xmm7");
}
```

Akkumulierende Funktionen: Benchmarks

- peak_rms:

```
35185    11.8784  dsp_tests    peak_rms(float const*, unsigned int, float&, float&)
28345     9.5692  dsp_tests    peak_rms_8(float const*, unsigned int, float&, float&)
9401      3.1738  dsp_tests    peak_rms_simd(float*, unsigned int, float&, float&)
```

- abspeak:

```
22708     3.4812  dsp_tests    abspeak(float const*, unsigned int, float&)
19768     3.0305  dsp_tests    abspeak_8(float const*, unsigned int, float&)
6729      1.0316  dsp_tests    abspeak_simd(float*, unsigned int, float&)
```

- rms:

```
18687     2.8648  dsp_tests    rms(float const*, unsigned int, float&)
17138     2.6273  dsp_tests    rms_8(float const*, unsigned int, float&)
6782      1.0397  dsp_tests    rms_simd(float*, unsigned int, float&)
```

Spezielle Instruktionen: rsqrt_vec

```
void rsqrt_vec(float * dest, float * src, unsigned int n)
{
    while (n--)
    {
        *dest++ = 1.f / sqrtf(*src++);
    };
}

void rsqrt_vec_8(float * dest, float * src, unsigned int n)
{
    for (unsigned int loops = n>>3; loops--; dest+=8, src+=8)
    {
        dest[0] = 1.f / sqrtf(src[0]);
        dest[1] = 1.f / sqrtf(src[1]);
        dest[2] = 1.f / sqrtf(src[2]);
        dest[3] = 1.f / sqrtf(src[3]);
        dest[4] = 1.f / sqrtf(src[4]);
        dest[5] = 1.f / sqrtf(src[5]);
        dest[6] = 1.f / sqrtf(src[6]);
        dest[7] = 1.f / sqrtf(src[7]);
    }
}
```

Spezielle Instruktionen: rsqrt_vec_simd

```
void rsqrt_vec_simd(float * dest, float * src, unsigned int n)
{
    asm(
        "shrl       $4, %2                  \n"

        /* loop: *dest = 1 / sqrt(*src) */
        "1:                                 \n"
        "rsqrtps    (%0,%3), %%xmm1         \n"
        "movaps     %%xmm1, (%1,%3)         \n"

        "rsqrtps    16(%0,%3), %%xmm2       \n"
        "movaps     %%xmm2, 16(%1,%3)       \n"

        "rsqrtps    32(%0,%3), %%xmm3       \n"
        "movaps     %%xmm3, 32(%1,%3)       \n"

        "rsqrtps    48(%0,%3), %%xmm4       \n"
        "movaps     %%xmm4, 48(%1,%3)       \n"

        "addl       $64, %3                 \n"
        "loop       1b                      \n"
        :
        :"r"(src), "r"(dest), "c"(n), "r"(0)
        :"%xmm1", "%xmm2", "%xmm3", "%xmm4"
    );
}
```

Spezielle Instruktionen: Benchmarks

- rsqrt_vec:

```
87327     13.9904  dsp_tests        rsqrt_vec(float*, float*, unsigned int)
87102     13.9543  dsp_tests        rsqrt_vec_8(float*, float*, unsigned int)
8050       1.2897  dsp_tests        rsqrt_vec_simd(float*, float*, unsigned int)
```

- rcp_vec:

```
47831     12.2925  dsp_tests        rcp_vec_8(float*, float*, unsigned int)
46722     12.0075  dsp_tests        rcp_vec(float*, float*, unsigned int)
6192       1.5913  dsp_tests        rcp_vec_simd(float*, float*, unsigned int)
```

Nicht vektorisierbare Algorithmen

- Filter: Feedback/Feedforward Systeme
- Potenzen / Logarithmen
- Trigonometrische / Hyperbolische Funktionen
- Table Lookups

Simd vs. compiler (-msse -march=pentium-m -mfpmath=sse -funroll-loops -O2)

55689	12.4523	dsp_tests	clip_vec(float*, float*, float*, float*, unsigned int)
40583	9.0745	dsp_tests	clip_vec_8(float*, float*, float*, float*, unsigned int)
10367	2.3181	dsp_tests	clip_vec_simd(float*, float*, float*, float*, unsigned int)
46913	10.4900	dsp_tests	sgn_vec(float*, float*, unsigned int)
15473	3.4598	dsp_tests	sgn_vec_simd(float*, float*, unsigned int)
40496	9.0551	dsp_tests	testcopyvec(float*, float*, unsigned int)
13851	3.0971	dsp_tests	testcopyvec_simd(float*, float const*, unsigned int)
39187	8.7624	dsp_tests	peak_rms(float const*, unsigned int, float&, float&)
30250	6.7640	dsp_tests	peak_rms_8(float const*, unsigned int, float&, float&)
15433	3.4509	dsp_tests	peak_rms_simd(float*, unsigned int, float&, float&)
20130	4.5012	dsp_tests	times_vec_scalar(float*, float*, float, unsigned int)
17826	3.9860	dsp_tests	times_vec_scalar_8(float*, float*, float, unsigned int)
6010	1.3439	dsp_tests	times_vec_scalar_simd(float*, float*, float, unsigned int)
19348	4.3263	dsp_tests	addvec(float*, float*, unsigned int)
16021	3.5824	dsp_tests	addvec_8(float*, float*, unsigned int)
7752	1.7334	dsp_tests	addvec_simd(float*, float const*, unsigned int)
10074	2.2526	dsp_tests	zerovec(float*, unsigned int)
7517	1.6808	dsp_tests	zerovec_8(float*, unsigned int)
4364	0.9758	dsp_tests	zerovec_simd(float*, unsigned int)
7816	1.7477	dsp_tests	setvec(float*, float, unsigned int)
6715	1.5015	dsp_tests	setvec_8(float*, float, unsigned int)
5376	1.2021	dsp_tests	setvec_simd(float*, float, unsigned int)

Simd vs. compiler (-funroll-loops -O2)

74758	14.9049	dsp_tests	clip_vec(float*, float*, float*, float*, unsigned int)
57295	11.4232	dsp_tests	clip_vec_8(float*, float*, float*, float*, unsigned int)
8969	1.7882	dsp_tests	clip_vec_simd(float*, float*, float*, float*, unsigned int)
59556	11.8740	dsp_tests	sgn_vec(float*, float*, unsigned int)
14379	2.8668	dsp_tests	sgn_vec_simd(float*, float*, unsigned int)
52447	10.4566	dsp_tests	peak_rms(float const*, unsigned int, float&, float&)
41620	8.2980	dsp_tests	peak_rms_8(float const*, unsigned int, float&, float&)
12543	2.5008	dsp_tests	peak_rms_simd(float*, unsigned int, float&, float&)
48382	9.6461	dsp_tests	testcopyvec(float*, float*, unsigned int)
15546	3.0995	dsp_tests	testcopyvec_simd(float*, float const*, unsigned int)
16242	3.2382	dsp_tests	addvec(float*, float*, unsigned int)
11737	2.3401	dsp_tests	addvec_8(float*, float*, unsigned int)
7844	1.5639	dsp_tests	addvec_simd(float*, float const*, unsigned int)
12203	2.4330	dsp_tests	times_vec_scalar_8(float*, float*, float, unsigned int)
11835	2.3596	dsp_tests	times_vec_scalar(float*, float*, float, unsigned int)
5853	1.1669	dsp_tests	times_vec_scalar_simd(float*, float*, float, unsigned int)
8574	1.7094	dsp_tests	setvec(float*, float, unsigned int)
7633	1.5218	dsp_tests	setvec_8(float*, float, unsigned int)
4142	0.8258	dsp_tests	setvec_simd(float*, float, unsigned int)
6939	1.3835	dsp_tests	zerovec_8(float*, unsigned int)
6032	1.2026	dsp_tests	zerovec(float*, unsigned int)
4570	0.9111	dsp_tests	zerovec_simd(float*, unsigned int)

Biquad filter

- with -msse -funroll-loops -march=pentium-m -mfpmath=sse -O2

```
19363    28.3383  filterbiquad_8(float*, float*, unsigned int, biquadData*)
18436    26.9816  filterbiquad(float*, float*, unsigned int, biquadData*)
17083    25.0015  filterbiquad_direct(float*, float*, unsigned int, biquadData*)
12710    18.6015  filterbiquad_8_relaxed(float*, float*, unsigned int, biquadData*)
```

- with -msse -funroll-loops -mtune=pentium-m -mfpmath=sse -O2

```
19042    27.5771  filterbiquad_8_relaxed(float*, float*, unsigned int, biquadData*)
16837    24.3838  filterbiquad_direct(float*, float*, unsigned int, biquadData*)
16577    24.0072  filterbiquad_8(float*, float*, unsigned int, biquadData*)
16504    23.9015  filterbiquad(float*, float*, unsigned int, biquadData*)
```

- with -msse -mfpmath=sse -O2

```
24412    31.0870  filterbiquad_8_relaxed(float*, float*, unsigned int, biquadData*)
18445    23.4884  filterbiquad_8(float*, float*, unsigned int, biquadData*)
18232    23.2172  filterbiquad_direct(float*, float*, unsigned int, biquadData*)
17301    22.0316  filterbiquad(float*, float*, unsigned int, biquadData*)
```

- with -O2

```
27892    26.4339  filterbiquad_direct(float*, float*, unsigned int, biquadData*)
26456    25.0730  filterbiquad(float*, float*, unsigned int, biquadData*)
25587    24.2494  filterbiquad_8_relaxed(float*, float*, unsigned int, biquadData*)
25522    24.1878  filterbiquad_8(float*, float*, unsigned int, biquadData*)
```