MMX i SSE

Zbigniew Koza

Wydział Fizyki i Astronomii
Uniwersytet Wrocławski

Wrocław, 15 marca 2011

Spis treści

Spis treści

Część 1

Co to są MMX, SSE i AVX?

- MMX (***M**ulti**M**edia e**X**tensions*) to rozszerzenie zestawu instrukcji procesorów SISD o instrukcje wektorowe na 64-bitowych rejestrach
- SSE (*Streaming SIMD Extension*) to rozszerzenie zestawu instrukcji procesorów SISD o instrukcje wektorowe na 128-bitowych rejestrach
- AVX (*Advanced Vector Extensions*) to rozszerzenie SSE na rejestry 256-bitowe.
- MMX, SSE i AVX umożliwiają jednoczesne wykonywanie 2, 4, 8 lub 16 (SSE) instrukcji.
- SSE to „komputer 128-bitowy”
- Pierwotnie pomyślane do przyspieszania operacji graficznych, mogą być wykorzystywane do przyspieszania „zwykłych” obliczeń i kryptografii.

Zalety i wady

- Zalety:
  - Umożliwiają stopniowe zrównoleglanie istniejącego kodu szeregowego
  - Wykorzystanie MMX/SSE zwykle przyspiesza programy
  - Są łatwe do nauki, łatwe do stosowania
  - Nie wymagają zewnętrznych bibliotek
  - Posiadają dobrą dokumentację
- Wady:
  - Dostępne wyłącznie na procesorach zgodnych ze standardem i386 (kod jest nieprzenośny)
  - Możliwość zrównoleglania kodu jest ograniczona sprzętowo (tę barierę usiłuje przekroczyć AVX)
  - Brak wsparcia dla mnożenia i dzielenia liczb całkowitych 32-bitowych (MMX, SSE)

Różnice

- Rejestry
  - MMX operuje na 8 rejestrach 64-bitowych (mapowane na koprocesor)
  - SSE operuje na osobnych 8 rejestrach 128-bitowych
  - AVX operuje na osobnych 16 rejestrach 256-bitowych
- Możliwości
  - MMX: 64-bitowe operacje na liczbach całkowitych ($+$, $-$, $*$, $/$, bitowe, przesunięcia, porównania, przypisania, konwersje)
  - SSE: 128-bitowe operacje na typie *float* ($+$, $-$, $*$, $/$, porównania, przypisania, konwersje) i 64-bitowe rozszerzenie MMX (min, max, inne).
  - SSE2: SSE + 128-bitowe operacje na typie *double* oraz na liczbach całkowitych
  - AVX: Jak SSE, ale na rejestrach 256-bitowych + instrukcje typu FMAD
- Sprzęt
  - MMX – Pentium MMX, AMD K6
  - SSE – Pentium III
  - SSE2 – Pentium 4
  - AVX – Intel Sandy Bridge / AMD Bulldozer (od 2011)

Kolejne wersje

- MMX – 64-bitowa wektoryzacja arytmetyki liczb całkowitych
- SSE – 128-bitowa wektoryzacja arytmetyki krótkich liczb zmiennopozycyjnych *float*
- SSE2 – 128-bitowa wektoryzacja arytmetyki długich liczb zmiennopozycyjnych *double* oraz liczb całkowitych (od *char* do *unsigned int*)
- SSE3, SSE4, SSE4 – nie mają typowych zastosowań w obliczeniach numerycznych
- SSE5 – projekt AMD niezrealizowany z powodu ogłoszenia projektu AVX, zastąpiony przez XOP, FMA4, CVT16
- AVX – od 2011 r, dalsza wektoryzacja arytmetyki zmiennopozycyjnej (rejestry 256-bitowe, operacje 3-argumentowe, FMAD, IMAD), gcc 4.4 lub Intel Compiler 11.1, linux kernel 2.6.30 lub Windows 7 SP1

Część 2

Nowe typy danych

Typ danych	MMX	SSE	SSE2	uwagi
__m64	tak	tak	tak	
__m128	–	tak	tak	4 × float
__m128d	–	–	tak	2 × double
__m128i	–	–	tak	16 × char ...2 × long long

- Adresy danych typu __m128 muszą być wielokrotnością 16

```
float* tab   = (float*) _mm_malloc( N * sizeof(float), 16 );
...
__m128 asse = _mm_load_pd( &tab[4] ); // indeks tab = 4*k
```

Konwencja nazewnicza

```
_mm_<intrin_op>_<suffix>
```

intrin_op	podstawowa operacja, np. add, mul
suffix	s = skalar, p = *packed*, ep = *extended packed*. Następnie: • s = *float* • d = *double* • i128 = *signed 128-bit integer* • i64 = *signed 64-bit integer* • u64 = *unsigned 64-bit integer* ... • i8 = *signed 8-bit integer* • u8 = *unsigned 8-bit integer*

```
__m64 _mm_add_pi8(__m64 m1, __m64 m2); // MMX
__m128 _mm_add_ss(__m128 a, __m128 b); // SSE
__mm128i _mm_add_epi8(__m128i a, __m128i b); // SSE2
```

Operacje na danych skalarnych i upakowanych

- Operacja skalarna wykonywana jest na pierwszej „składowej” danych, pozostałe są przepisywane z 1. argumentu.

```
__m128 a, __m128 b;
__m128 r = _mm_min_ss(a, b);
```

jest równoważne

```
r0 = min(a0, b0);
r1 = a1; r2 = a2; r3 = a3;
```

- Operacja „upakowana” – równocześnie na wszystkich „składowych”.

```
__m128 r = _mm_min_ps(a, b);
```

jest równoważne

```
r0 = min(a0, b0); r1 = min(a1, b1);
r2 = min(a2, b2); r3 = min(a3, b3);
```

Użycie w C/C++

1. Włączenie właściwego nagłówka:

```
#include <mmintrin.h> // MMX
#include <mm3dnow.h>  // MMX 3D Now!
#include <xmmintrin.h> // SSE
#include <emmintrin.h> // SSE2
#include <pmmintrin.h> // SSE3
#include <tmmintrin.h> // SSSE3
#include <smmintrin.h> // SSE4
#include <avxintrin.h> // AVX
#include <immintrin.h> // wszystko, co obsluguje kompilator
```

2. Zastosowanie odpowiedniej flagi kompilatora

```
-msse
-msse2
...
-msse4
-msse2avx
```

Przykład kodu

```
#include <emmintrin.h>
...
void calc_sse (double* wynik, double* p1, double* p2, int size)
{
  __m128d asse, bsse;
  const __m128d sse_pol = _mm_set_pd1(0.5);
  for (int i = 0; i < size; i += 2)
  {
    asse = _mm_load_pd( &p1[i] );
    bsse = _mm_load_pd( &p2[i] );
    asse = _mm_mul_pd( asse, asse );
    bsse = _mm_mul_pd( bsse, bsse );
    asse = _mm_add_pd( asse, bsse);
    asse = _mm_sqrt_pd( asse );
    asse = _mm_add_pd(asse, sse_pol);
    _mm_store_pd( &wynik[i], asse);
  }
}
```

Czy to się opłaca?

- $w_i = \sqrt{a_i^2 + b_i^2} + 0.5, i = 1, \ldots, 10^6$.

Procesor	przyspieszenie		$t_{\min}$ [ms]	
	float	double	float	double
AMD Phenom II X6 1055T 2.8GHz	3.07	1.51	31	69
AMD Athlon 64 X2 Dual Core 6000+	2.37	1.22	55	129
AMD Athlon 64 X2 Dual Core 4000+	2.28	2.25	96	213
Intel Pentium Dual-Core E5200 2.50GHz	1.36	1.15	51	102
Intel Core 2 Duo T8100 2.1 GHz;	2.08	1.08	65	152
Intel Atom CPU N450 1.66GHz	3.22	1.20	207	602

Alternatywa: optymalizacja automatyczna

- `-mfpmath=sse`

Procesor	przyspieszenie	
	float	double
AMD Phenom II X6 1055T 2.8GHz	1.20	1.24
AMD Athlon 64 X2 Dual Core 6000+	1.22	1.22
AMD Athlon 64 X2 Dual Core 4000+	1.00	1.00
Intel Pentium Dual-Core E5200 2.50GHz	0.99	1.08
Intel Core 2 Duo T8100 2.1 GHz	1.66	1.15
Intel Atom CPU N450 1.66GHz	1.04	1.19

Literatura

1. *Intel C++ Compiler for Linux Systems User's Guide*