AUTOMATYKA • 2009 • Tom 13 • Zeszyt 3

Zbigniew Bubliński*

Wykorzystanie instrukcji SSE w przetwarzaniu obrazów**

1. Wprowadzenie

Przetwarzanie obrazów cyfrowych jest czynnością dosyć czasochłonną – głównie ze względu na dużą ilość danych do obróbki. Wykorzystanie możliwości współczesnego procesora i zastosowanie specjalizowanych instrukcji MMX i/lub SSE pozwala na znaczne skrócenie czasu przetwarzania poprzez równoległe wykonywanie kilku operacji. W dalszej części niniejszej publikacji przedstawiono przykład wykorzystania rozkazów MMX i SSE do szybkiej binaryzacji obrazu cyfrowego.

2. Instrukcje MMX I SSE

Dodanie do procesora z rodziny x86 rozszerzeń architektury SIMD (*Single Instruction, Multiple Data*) wiązało się z pojawieniem się nowych instrukcji wzbogacających jego możliwości o wykonywanie pewnych instrukcji jednocześnie na większej liczbie danych. Ilość przetwarzanych danych i rodzaj operacji zależy od poziomu ewolucyjnego procesora. Już w procesorze Pentium pojawiła się możliwość wykonywania kilku niezbyt złożonych instrukcji na danych będących liczbami całkowitymi (tzw. rozkazy MMX (*MultiMedia eXtension albo MatrixMathematics eXtension* [4, 5, 6]). Nowsze procesory z rodziny x86 potrafiły już wykonywać operacje na liczbach zmiennoprzecinkowych. Ogólna nazwa nowych rozszerzeń to tzw. SSE (*Streaming SIMD Extension*), a dodany numer identyfikuje poziom rozszerzeń, a co za tym idzie, udostępniane możliwości – w najnowszych procesorach występują warianty SSE4.1 i SSE4.2.

Instrukcje MMX pozwalają na przetwarzanie danych o maksymalnej wielkości 64 bitów (taki jest rozmiar wykorzystywanych rejestrów), a więc zakładając, że przetwarzane dane są jednobajtowe, mamy możliwość wykonania pewnych operacji jednocześnie na ośmiu danych (np. pikselach obrazu). Rejestry wprowadzone w rozszerzeniach SSE mają już długość 128 bajtów, co umożliwia jednoczesne przetwarzanie 16 pikseli.

* Katedra Automatyki, Akademia Górniczo-Hutnicza w Krakowie
** Praca wykonana w ramach badań własnych – umowa AGH nr 10.10.120.783

Wynika z tego, że użycie instrukcji MMX i SSE pozwoliłoby w teorii na ośmio- lub szesnastokrotne skrócenie czasu przetwarzania, ale istnieje kilka przeszkód znacząco utrudniających ich wykorzystanie. Najważniejsze z nich to:

- ograniczona lista operacji możliwych do zrealizowania w sposób równoległy,
- konieczność odpowiedniego przygotowania danych (upakowanie danych z pamięci w rejestrach i rozpakowanie danych z rejestrów przy zapisie do pamięci),
- konieczność pisania istotnych funkcji i procedur lub ich fragmentów w asemblerze.

W następnym rozdziale przedstawiono operację binaryzacji stałoprogowej obrazu cyfrowego, którą zaimplementowano w asemblerze wykorzystując rozkazy MMX oraz SSE2.

3. Binaryzacja stałoprogowa obrazu cyfrowego

Binaryzacja stałoprogowa jest prostą, ale często stosowaną metodą przetwarzania obrazów cyfrowych [1, 2]. Stałoprogowość oznacza, że w trakcie przetwarzania obrazu próg binaryzacji ma cały czas taką samą wartość. Istnieją też wersje zmiennoprogowe, czyli takie, w których wartość progu jest wyznaczana lokalnie dla każdego piksela albo dla pewnych fragmentów obrazu, ale nie nadają się one do przyspieszania poprzez zrównoleglanie operacji, gdyż zwykle w ich przypadku większym problemem jest wyznaczenie lokalnego progu, a nie sama binaryzacja.

Operacja binaryzacji stałoprogowej sprowadza się do porównania wartości każdego piksela (poziomu jasności/szarości dla klasycznych obrazów cyfrowych [3]) z wartością progu i wstawieniu do obrazu wynikowego wartości określającej wynik porównania – najczęściej jest to 0 dla relacji piksel < próg i 255 w przeciwnym przypadku. Można taką operację zapisać w postaci następującego pseudokodu:

```
if obraz_we[i,j] >= próg then
   obraz_wy[i,j] = 255
else
   obraz_wy[i,j] = 0
```

Taka operacja dobrze nadaje się do zrównoleglenia – jednocześnie można porównywać i modyfikować kilka lub kilkanaście punktów obrazu. W celu zbadania jakie przyspieszenia można uzyskać poprzez wykorzystanie rozszerzeń SIMD, zaimplementowano binaryzację stałoprogową w postaci czterech funkcji. Ich opis i uzyskane wyniki przedstawiono w następnym rozdziale.

4. Wyniki testów

Szczegółowe badania prowadzono z użyciem czterech wersji binaryzacji stałoprogowej. Pierwsza z nich stanowi niejako definicyjne podejście do zagadnienia i jest zapisem przedstawionego wcześniej pseudokodu bez wykorzystania jakichkolwiek sposobów

optymalizacji – zarówno w samym kodzie, jak i w trakcie jego kompilacji. Druga wersja powstała w asemblerze i jest zoptymalizowaną modyfikacją pierwszej – użycie asemblera pozwoliło wyeliminować nadmiarowe operacje wynikające z niedoskonałości kompilatora. Pozostałe dwie wersje zostały napisane w asemblerze z użyciem instrukcji MMX (pierwsza) i SSE2 (druga), co pozwoliło na jednoczesne przetwarzanie odpowiednio 8 i 16 punktów obrazu. Wszystkie funkcje były użyte do binaryzacji obrazów o rozmiarach 512×512 pikseli, przy czym zawartość obrazów była generowana losowo. W celu zminimalizowania błędów wynikających z wykonywania przez system operacyjny innych czynności sama binaryzacja była powtarzana 10 000 razy i łączny czas tylu powtórzeń binaryzacji był brany jako wynik pomiaru. Aby dodatkowo wyeliminować pewne fluktuacje rezultatów wynikające z losowej zawartości obrazu, pomiary powtórzono dziesięciokrotnie i jako wynik końcowy brano średnią arytmetyczną ze zmierzonych czasów.

Dla porównania uzyskanych rezultatów badania zostały przeprowadzone na dwóch komputerach. Komputer A był wyposażony w procesor czterordzeniowy Intel Q6600 (2,4 GHz, FSB 1066 MHz), zaś komputer B w procesor Intel P8600 (2,4 GHz, FSB 1066 GHz) – ciekawostką jest fakt, że pomimo identycznych parametrów wyniki różnych testów dotyczących aplikacji jednowątkowych i opublikowanych w Internecie świadczą o większej efektywności procesora P8600, co zresztą zostało potwierdzone w trakcie przeprowadzonych badań. Szczegółowe wyniki pomiarów czasu i uzyskane przyspieszenia zawarto w tabeli 1.

Tabela 1
Średnie czasy binaryzacji i przyspieszenia – porównanie metod

		Kod w C	Kod w ASM	Kod w ASM (MMX)	Kod w ASM (SSE2)
Procesor A	Czas [ms]	2251,709	1790,476	96,270	48,337
	Przyspieszenie względem kodu w C [x]	1,000	1,258	23,390	46,584
	Przyspieszenie względem kodu w ASM [x]	0,795	1,000	18,599	37,042
Procesor B	Czas [ms]	2205,950	1723,111	91,311	45,628
	Przyspieszenie względem kodu w C [x]	1,000	1,280	24,159	48,347
	Przyspieszenie względem kodu w ASM [x]	0,781	1,000	18,871	37,764

Przedstawione wyniki wyraźnie wskazują, że użycie rozkazów MMX w kodzie napisanym w asemblerze pozwala na ponad 18-krotne przyspieszenie operacji binaryzacji (w stosunku do kodu w asemblerze – czasy dla kodu napisanego w C nie są dobrym układem odniesienia, gdyż zbytnio zależą od samego kompilatora i jego możliwości optymalizacyjnych). Jeszcze lepsze rezultaty można uzyskać, stosując rozkazy SSE2 – możliwe jest wówczas około 37-krotne skrócenie czasu binaryzacji.

Rezultaty uzyskane na procesorach o zbliżonych parametrach wykazują tylko subtelne różnice i dodatkowo potwierdzają istniejące już wyniki porównań efektywności różnych procesorów uzyskane w rozmaitych testach. Warto też zwrócić uwagę na to, że te różnice dotyczą tylko samych czasów, natomiast przyspieszenia są w zasadzie identyczne.

5. Wnioski

Użycie w kodzie instrukcji MMX i SSE umożliwiających wykorzystanie rozszerzeń SIMD zawartych w architekturach współczesnych procesorów pozwala w znaczący sposób przyspieszyć realizację takich metod przetwarzania obrazu cyfrowego jak binaryzacja stałoprogowa i chociaż wiąże się z tym konieczność pisania istotnych czasowo fragmentów programu w asemblerze, to jednak uzyskane efekty w pełni potwierdzają celowość takiego postępowania.

Literatura

[1] Parker J.R., *Algorithms for Image Processing and Computer Vision*. Wiley Computer Publishing, New York, 1997.
[2] Pratt W.K., *Digital Image Processing*. John Wiley & Sons, Inc., New York, 1991.
[3] Skarbek W., *Metody reprezentacji obrazów cyfrowych*. Akademicka Oficyna Wydawnicza PLJ, Warszawa, 1993.

Źródła internetowe:

[4] AMD Inc. http://www.amd.com.
[5] [INTEL Corp. http://www.intel.com.
[6] MMX Primer http://www.tommesani.com/MMXPrimer.html.