AUTOMATYKA • 2008 • Tom 12 • Zeszyt 3

Zbigniew Bubliński*, Mirosław Jabłoński*

Rozszerzenia SIMD w przetwarzaniu obrazów**

1. Wprowadzenie

Przetwarzanie i analiza obrazów cyfrowych jest działalnością stosunkowo czasochłonną – głównymi przyczynami takiego stanu rzeczy są z jednej strony złożone i wyrafinowane algorytmy, z drugiej zaś wykorzystywana w nich ogromna ilość danych. O ile na samo skomplikowanie metod obróbki obrazów niewiele można poradzić, to obróbkę dużej ilości danych można w odpowiedni sposób usprawnić, co pozwoli albo na skrócenie czasu przetwarzania, albo na użycie lepszych, ale bardziej kosztownych czasowo procedur bez konieczności zbytniego wydłużania całego procesu. Takim usprawnieniem może być przykładowo realizacja całości analizy (lub jej wybranych fragmentów) na drodze sprzętowej. Jest to jednak mało elastyczne i kosztowne podejście – większość istniejących systemów przetwarzania i analizy obrazów cyfrowych ma charakter czysto programowy, więc usprawnień należałoby raczej szukać w lepszym dopasowaniu wykonywanego programu do możliwości samego procesora. W dalszej części niniejszej publikacji przedstawiono podstawowe informacje dotyczące rozszerzeń SIMD architektury procesora x86 oraz przykład wykorzystania rozkazów MMX i SSE do szybkiej modyfikacji jasności obrazu cyfrowego. Prace były prowadzone w środowisku „VirtualDub" [8], bowiem stanowi ono od dłuższego czasu jedno z najczęściej wykorzystywanych narzędzi w Laboratorium Biocybernetyki.

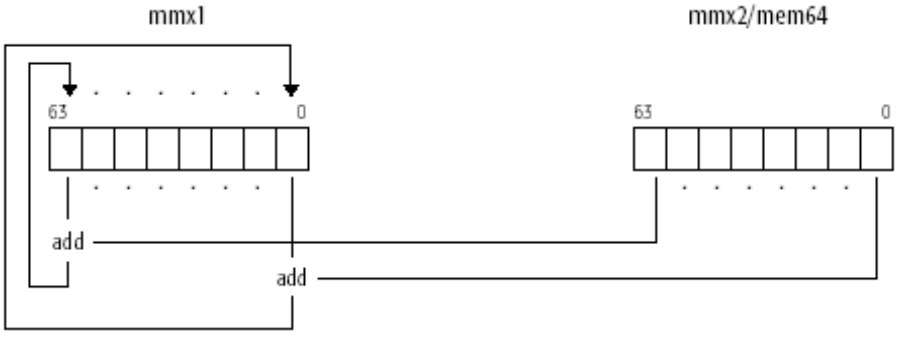
2. Rozszerzenia SIMD

Rozszerzenia SIMD (*Single Instruction, Multiple Data*) pojawiły się jako próba wzbogacenia możliwości klasycznego procesora o wykonywanie pewnych prostych instrukcji jednocześnie na kilku 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 – były to tzw. rozkazy MMX (MultiMedia eXtension albo MatrixMathematics eXtension [5]), a co

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

ciekawe – do ich realizacji wykorzystano zasoby udostępnione przez jednostkę zmiennoprzecinkową (koprocesor matematyczny). Następne procesory z rodziny x86 potrafiły już wykonywać operacje na liczbach zmiennoprzecinkowych, wykorzystując dedykowany do tego celu zasoby sprzętowe procesora. Ogólna nazwa nowych rozszerzeń to tzw. SSE (*Streaming SIMD Extension*), zaś dodany numer identyfikował poziom rozszerzeń, a co za tym idzie, udostępniane możliwości – w najnowszych procesorach występuje kilka wariantów SSE4.

Jak już wspomniano, rozkazy MMX mogą działać tylko na danych całkowitych [5, 7]. Wydaje się to znaczącym ograniczeniem, ale w wielu zastosowaniach operacje całkowitoliczbowe są zupełnie wystarczające. Przykładowo – przetwarzanie sygnałów multimedialnych (obraz, dźwięk) to jeden z takich obszarów. Rozszerzenie MMX pozwala na wykonywanie operacji na danych o maksymalnej wielkości 64 bitów, a więc zakładając że przetwarzane dane są jednobajtowe, mamy możliwość wykonania pewnych operacji jednocześnie na ośmiu danych (bajtach) – tak jak to pokazano na rysunku 1.



Rys. 1. Rozszerzenie MMX – reprezentacja danych i przykładowa operacja

Teoretycznie pozwoliłoby to na ośmiokrotne skrócenie czasu przetwarzania, ale jak wiadomo, praktyka zwykle odbiega od teorii – istnieje kilka poważnych przeszkód znacząco utrudniających wykorzystanie rozszerzeń SIMD. Po pierwsze, lista operacji możliwych do zrealizowania w sposób równoległy jest dosyć ograniczona, a więc nie każdy algorytm może być zaimplementowany w taki właśnie sposób. Po drugie, zwykle operacja SIMD wymaga odpowiedniego przygotowania danych – zanim zostanie zrealizowana, dane muszą być przeniesione z pamięci i właściwie upakowane w rejestrach, zaś po operacji trzeba dane rozpakować i ponownie umieścić w pamięci. Wiąże się z tym dodatkowy nakład pracy (czasu) i w pewnych sytuacjach może się okazać, że te dodatkowe czynności trwają dłużej, niż można zyskać poprzez równoległe przetwarzanie danych. Po trzecie – i to chyba największy problem – dostępne kompilatory języków wysokiego poziomu nie najlepiej sobie radzą z generacją kodu efektywnie wykorzystującego możliwości samego procesora. Chcąc więc uzyskać taki kod, konieczne jest napisanie całego programu lub jego krytycznych czasowo

fragmentów, w języku najniższego poziomu – w asemblerze. W następnym rozdziale przedstawiono operację zmiany jasności obrazu zrealizowaną programowo za pomocą filtrów działających w środowisku „VirtualDub", zaś istotne ze względu na czas ich wykonania części kodów napisano w asemblerze wykorzystując rozkazy MMX oraz SSE2.

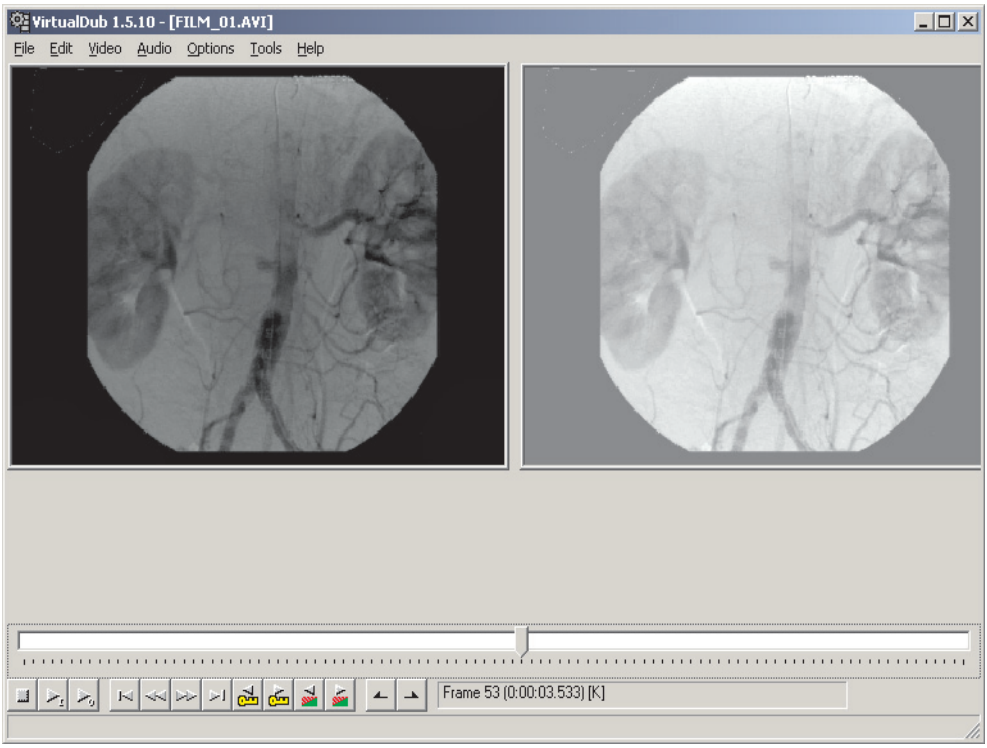
3. Zmiana jasności obrazu w środowisku „VirtualDub" – realizacja

Modyfikacja jasności monochromatycznego obrazu cyfrowego jest bardzo prostą operacją i sprowadza się do dodania pewnej stałej do wartości jasności poszczególnych punktów obrazu [2, 3]. Komplikacje pojawiają się w przypadku obrazów kolorowych – najczęściej spotykaną reprezentacją barw jest RGB, czyli kolor każdego punktu obrazu jest opisany trójką liczb określających „zawartość" poszczególnych składowych barwnych (*Red*, *Green*, *Blue*). Przy takiej reprezentacji zmiana jasności nie może już być dokonana poprzez dodanie tej samej stałej do wartości poszczególnych składowych, gdyż ze względu na różną charakterystykę czułości oka ludzkiego w różnych zakresach widmowych, taka modyfikacja skutkowałaby zmianą postrzeganej barwy punktu, a celem jest zmiana jasności, a nie koloru. Aby w sensowny sposób przeprowadzić taką operację, konieczne jest przetransformowanie obrazu z przestrzeni RGB w inną – taką, w której jedną ze składowych będzie jasność. Warunek ten spełnia kilka reprezentacji – przykładowo HLS, HSV, YUV. W praktyce najczęściej wykorzystywany jest jeden z wielu wariantów tej ostatniej, w której składowa Y to właśnie jasność (luminancja) a U i V to składowe chrominancyjne zawierające informację o kolorze punktu. Dysponując już reprezentacją YUV, modyfikacja jasności obrazu przebiega identycznie jak w przypadku obrazów monochromatycznych – zmieniana jest tylko składowa Y. Po dokonaniu modyfikacji konieczne jest przeprowadzenie transformacji odwrotnej, czyli powrót do reprezentacji RGB. Stosowne wzory transformacyjne można znaleźć w [1, 4, 6].

W środowisku „VirtualDub" wewnętrzną reprezentacją obrazów kolorowych jest RGB. Z tego względu konieczne było dodanie do implementowanego filtru kodu realizującego transfomaty RGB->YUV i YUV->RGB, ale w niniejszym opracowaniu skupiono się jedynie na samej modyfikacji jasności. Ponieważ w reprezentacji RGB każdy piksel obrazu opisany jest 32 bitami (po osiem bitów na każdą składową R, G, B i osiem bitów niewykorzystanych), przyjęto podobną reprezentację obrazu w przestrzeni YUV, ale poszczególne składowe przechowywane są oddzielnie, co pozwala zmniejszyć zapotrzebowanie na pamięć o 25%. Punktem wyjścia było napisanie stosownej funkcji w języku C, która realizuje modyfikację jasności obrazu o zadaną wartość, zgodnie z przedstawionymi wyżej założeniami. Funkcja ta i zmierzone czasy przetwarzania uzyskane przy jej użyciu stanowią układ odniesienia do kolejnych funkcji, napisanych w asemblerze – jednej, wykorzystującej rozkazy MMX do jednoczesnej modyfikacji ośmiu punktów obrazu, oraz drugiej, w której użyto rozkazów SSE2, co pozwoliło na jednoczesną modyfikację szesnastu punktów obrazu.

4. Wyniki testów

Opracowane filtry przetestowano w środowisku „VirtualDub", modyfikując jasność obrazów zawartych w trzech filmach w formacie AVI. Filmy różniły się liczbą klatek, rozmiarami klatek, a także zawartością. Każdy z filmów był przetwarzany kilkukrotnie, za każdym razem jego jasność była zmieniana w innym stopniu. Czas przetwarzania (samej modyfikacji jasności) był mierzony dla każdej klatki, a następnie czasy te zostały zsumowane, pozwalając uzyskać łączny czas przetwarzania filmu. W kolejnym kroku uzyskane wyniki zostały uśrednione dla każdego filmu. Na rysunku 2 pokazano środowisko „VirtualDub" w trakcie przetwarzania jednego z filmów.



Rys. 2. Wygląd ekranu w czasie testów

Testy przeprowadzono na dwóch komputerach typu PC – pierwszy (oznaczony literą A) był wyposażony w wiekowy już procesor AMD Athlon XP2000 (z zegarem 1667 MHz), drugi (oznaczony literą B) w procesor Intel Q6600 (z zegarem 2400 MHz). Taki dobór sprzętu miał z jednej strony pozwolić na ocenę przyspieszenia algorytmów poprzez wykorzystanie rozszerzeń SIMD, z drugiej zaś dostarczyć informacji o tym, jak unowocześnienie

architektury procesora wpływa na realne czasy przetwarzania. Uzyskane wyniki zestawiono w tabeli 1. Brak wyników dla procesora A i instrukcji SSE2 wynika z faktu, że rozszerzenie SSE2 pojawiło się dopiero w procesorze Pentium 4.

Tabela 1
Średnie czasy przetwarzania – porównanie metod

Film	Procesor	Filtr C [ms]	Filtr ASM+MMX [ms]	Przysp.	Filtr ASM+SSE2 [ms]	Przysp.
1	A	54,71	17,02	3,2 ×	–	–
	B	27,99	3,59	7,8 ×	1,86	15,0 ×
2	A	38,06	12,08	3,1 ×	–	–
	B	19,04	2,44	7,8 ×	1,27	16,0 ×
3	A	257,27	83,72	3,1×	–	–
	B	128,81	16,54	7,8 ×	8,60	15,0 ×

Z przedstawionych wyników wyraźnie widać, że samo użycie nowocześniejszego i szybszego procesora bez zmiany algorytmu (mechanizmu przetwarzania) niewiele daje. Wykorzystanie kodu napisanego w czystym C na lepszym procesorze wiąże się jedynie z mniej więcej dwukrotnym skróceniem czasu obliczeń (wyniki w trzeciej kolumnie tabeli). Zdecydowanie lepsze rezultaty można uzyskać, wykorzystując rozszerzenia SIMD – nawet na tym samym sprzęcie. Użycie rozkazów MMX w kodzie napisanym w asemblerze pozwala na skrócenie czasów przetwarzania od 3,1 raza dla procesora A aż do prawie osiem razy dla procesora B. Jeszcze lepsze wyniki można uzyskać, stosując rozkazy SSE – możliwe jest wówczas około piętnastokrotne skrócenie czasu obliczeń, choć konieczna jest do tego odpowiednia infrastruktura (w miarę nowoczesny sprzęt). Świadczy to o dużo lepszej efektywności rozszerzeń SIMD w stosunku do konwencjonalnych instrukcji współczesnych procesorów.

5. Podsumowanie

Wykorzystanie rozszerzeń SIMD zawartych w architekturze współczesnych procesorów pozwoliło w znaczący sposób przyspieszyć czas przetwarzania obrazu cyfrowego. Warto zwrócić uwagę, że wykorzystanie tych mechanizmów umożliwiło uzyskanie znacznie lepszych rezultatów niż w sytuacji, gdy użyto nowocześniejszego i szybszego procesora realizującego klasyczny algorytm. Otrzymanie takich wyników wymaga pewnej dodatkowej pracy, jest czasochłonne – zwłaszcza na etapie uruchamiania kodu oraz znacząco zmniejsza przenaszalność kodu, jednak skutecznie przyspiesza działanie całego systemu przetwarzania i analizy obrazów.

Literatura

[1] Keith J., *Video Demystified: A Handbook for the Digital Engineer*. 5th Edition, Elsevier 2005.
[2] Parker J.R., *Algorithms for Image Processing and Computer Vision*. New York, Wiley Computer Publishing 1997.
[3] Pratt W.K., *Digital Image Processing*. New York, John Wiley & Sons, Inc., 1991.
[4] Skarbek W., *Metody reprezentacji obrazów cyfrowych*. Warszawa, Akademicka Oficyna Wydawnicza PLJ 1993.

Źródła internetowe:

[5] INTEL Corp. www.intel.com
[6] Konwersja YUV-RGB http://www.fourcc.org/fccyvrgb.php
[7] MMX Primer http://www.tommesani.com/MMXPrimer.html
[8] VirtualDub http://www,virtualdub.org