Analiza wydajności aplikacji w środowisku MS Windows 98/NT/2000

Artur Miktus[1]

STRESZCZENIE: W pracy przedstawiono przegląd sposobów definiowania, pomiaru i uogólniania wskaźników wydajnościowych komputerów. Zaprezentowano przykłady oceny wpływu ulepszeń wybranych fragmentów systemu komputerowego na pracę całego systemu. Omówiono zastosowanie aplikacji VTune[2] firmy Intel do analizy wydajności aplikacji w środowisku MS Windows 98/NT/2000.

1. Pomiary wydajności

Zagadnienie pomiarów wydajności systemów komputerowych pojawia się często w trakcie ich projektowania, udoskonalania i użytkowania. Wydajność systemu komputerowego może być różnie rozumiana i, na przykład, dla jednego użytkownika, pracującego na pojedynczym komputerze, zwiększenie wydajności oznacza, iż na tym komputerze dany program wykona się w krótszym czasie, a dla innego użytkownika, na przykład administratora ośrodka obliczeniowego, że w ciągu dnia pracy ośrodek ten wykona więcej zadań (transakcji). Użytkownicy mogą więc być zainteresowani zmniejszeniem czasu odpowiedzi[3] (ang. response time), czyli czasu od uruchomienia programu do zakończenia wykonywanego zadania, lub zwiększaniem przepustowości (ang. throughput), czyli ilości zadań całkowicie wykonanych w danej jednostce czasu. Niezależnie jednak od tego czy badamy czas odpowiedzi, czy przepustowość, istotą zagadnienia jest pomiar czasu wykonania jednego bądź wielu zadań.

Niestety, nie zawsze czas wykonania wspomnianych zadań jest wartością publikowaną przy porównywaniu wydajności komputerów. W dodatku może on być różnie definiowany.

---

[1] Zakład Systemów Komputerowych, Instytut Automatyki i Robotyki WAT, ul. Kaliskiego 2,00-908 Warszawa.

[2] VTune to znak zastrzeżony firmy Intel. MS Windows to znak zastrzeżony firmy Microsoft. Inne występujące w tekście znaki są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli. Znaki te zostały użyte jedynie w celu identyfikacji odpowiednich produktów.

Zgodnie z naturalnym pojmowaniem *czasu wykonania zadania* przedmiotem pomiaru jest odcinek czasu między uruchomieniem a zakończeniem zadania, włączając w to także czas odczytu danych z dysku czy czas potrzebny na wykonanie w tle zadań systemu operacyjnego. Węższe znaczeniowo jest pojęcie *czasu procesora*, który nie obejmuje na przykład oczekiwania na zakończenie operacji wejścia/ wyjścia. Czas procesora może dalej być podzielony na *czas procesora dla użytkownika* (ang. user CPU time), gdy procesor wykonuje program użytkownika i *systemowy czas procesora* (ang. system CPU time), gdy procesor wykonuje zadania systemu operacyjnego, w tym również usługi zlecone przez program użytkowy. Rozróżnienie to można zauważyć w UNIX-owej komendzie *time*, która mierzy czas wykonania polecenia, podając jako wynik trzy liczby: real, user, sys. Na podstawie czasu odpowiedzi uzyskujemy charakterystyki wydajności systemu, a rozważając czas użytkowy procesora badamy łączną wydajność procesora i podsystemu pamięci.

2. Rodzaje benchmarków

Najlepszym badaczem wydajności systemu komputerowego zarówno w sensie czasu wykonania jak i przepustowości jest operator, który w dłuższym okresie czasu wykonuje te same codzienne, rutynowe zadania – uruchamia mieszankę swoich stałych programów i komend systemu operacyjnego i mierzy czas ich wykonania. Niestety, nie zawsze można posłużyć się tą metodą, choćby ze względu na kłopot w określeniu składników mieszanki pomiarowej. Wynika stąd potrzeba wyróżnienia specjalnych klas programów, służących do testowania wydajności komputerów (ang. benchmark). Można w związku z tym wyróżnić co najmniej 4 klasy takich programów [1]:

1. Rzeczywiste programy, używane do realizacji konkretnych zadań (np. kompilatory, programy do generacji scen 3D), mające konkretne dane wejściowe i parametry. Przykład: gra Quake. Pomiar ilości ramek obrazu, wyświetlonych w ciągu sekundy „fps" (ang. frames per second) określa jakość systemu, w tym procesora i pamięci komputera, procesora i pamięci karty grafiki, sterowników programowych karty grafiki do współpracy z konkretnym systemem operacyjnym i sprzętem (http://www.quake3arena.com/).

[3] Stosowana jest także nazwa czas wykonania (ang. execution time).

2. Programy typu „rdzeń" bądź „jądro" (ang. kernel), czyli fragmenty rzeczywistych programów, wyodrębnione dla celów określania wydajności wyizolowanych elementów komputera. Przykład: Linpack [7].
3. Programy „zabawkowe" (ang. toy benchmark), czyli niewielkie programy, rzędu 100 linii kodu, których wyniki są znane użytkownikowi jeszcze przed uruchomieniem programu. Są małe, łatwe w użyciu i można je uruchomić na praktycznie każdym komputerze. Przykład: sito Erastotenesa, QuickSort.
4. Benchmarki syntetyczne, podobne w filozofii działania do kerneli, próbują uchwycić średnią częstotliwość instrukcji i dostępów do danych na podstawie analizy dużych zbiorów programów. Przykład: Whetstone [8] i Dhrystone [9].

Należy przy tym zauważyć, że wyniki wszystkich testów zależą nie tylko od rodzaju i szybkości wykonującego je procesora, ale też od organizacji i parametrów czasowych systemu pamięci oraz jakości i parametrów optymalizacyjnych kompilatorów, użytych do otrzymania testów.

Ponieważ często sukcesy rynkowe firm komputerowych zależą od publikowanych wskaźników „cena/ wydajność", firmy te podejmują działania, mające na celu wykazanie, za pomocą benchmarków, dużej wydajności produkowanych przez siebie komputerów. Jest oczywiste, że rzeczywisty wzrost wydajności, odczuwany przez użytkownika w aplikacjach, nie musi korespondować ze wzrostem wydajności odnotowanym dla benchmarków. W celu zwiększenia wiarygodności i wszechstronności testów, projektanci benchmarków wprowadzają całe zestawy programów, mając nadzieję, że ewentualna słabość jednego ze składników zestawu zostanie uzupełniona silną stroną innego. Może to być prawdą, szczególnie gdy o wyniku testu, w przeciwieństwie do efektów fragmentarycznych, decyduje łączny czas wykonania całego zestawu. Przykładami takich zestawów są:

a) SPEC CPU2000, składający się z następujących grup programów:
   - 12 programów, operujących na liczbach całkowitych (zestaw CINT 2000, http://www.specbench.org/osg/cpu2000/CINT2000), w tym między innymi programy realizujące zadania kompresji 5 plików, kompilacji 5 plików, znajdowania optymalnego ruchu dla 5 rozkładów na szachownicy, modelowania trójwymiarowego obrazu o rozmiarze 150 na 150 pikseli czy symulacji pracy obiektowej bazy danych.

- 14 programów, operujących na liczbach zmiennoprzecinkowych (zestaw CFP 2000, http://www.specbench.org/osg/cpu2000/CFP2000), w tym między innymi programy rozpoznawania obiektów, bazujące na sieci neuronowej, symulacji propagacji fal sejsmicznych czy badania dynamiki obiektów w zderzeniach niesprężystych.

b) 3Dmark 2000 i Video 2000 firmy MadOnion (http://www.madonion.com), wykonujące fragmenty aplikacji graficznych, testujące podsystem procesora, pamięci i karty grafiki.

3. Definiowanie, porównywanie i uogólnianie wskaźników wydajnościowych[4]

Na pozór wystarczy uzgodnić zestaw programów, środowisko badań i definicję, *co to znaczy szybciej*, aby bez kłopotów badać wydajność komputerów. Rozważmy przykład, przedstawiony w tab.1 [1]:

Tab.1. Przykładowe czasy wykonania programów P1 i P2 na komputerach A, B i C

	Komputer A	Komputer B	Komputer C
Czas wykonania P1	1	10	20
Czas wykonania P2	1000	100	20
Łączny czas wykonania P1+P2	1001	110	40

Porównując czasy wykonania pojedynczych testów uzyskamy następujące stwierdzenia:

a) A jest 10 razy szybszy od B dla P1;
b) B jest 10 razy szybszy od A dla P2;
c) A jest 20 razy szybszy od C dla P1;
d) C jest 50 razy szybszy od A dla P2;
e) B jest 2 razy szybszy od C dla P1;
f) C jest 5 razy szybszy od B dla P2;

[4] W Internecie można znaleźć pewną wypowiedź, która w przekonaniu autora dobrze oddaje sytuację, panującą w dziedzinie porównywania wyników pomiarów wydajności komputerów: „A benchmark is like sex. Everybody wants it, everybody is sure of how to do it, but nobody can agree on how to compare performance"

Pojedyncze stwierdzenia mogą być użyteczne, ale zbiorcze porównanie może być kłopotliwe – który komputer jest najbardziej wydajny? Porównajmy zatem łączny czas wykonania programów P1 i P2:

g) B jest 9.1 razy szybszy niż A dla P1+P2;

h) C jest 25 razy szybszy niż A dla P1+P2;

i) C jest 2.75 razy szybszy niż B dla P1+P2;

Czyżby więc najszybszy był komputer C? Próba odpowiedzi na to pytanie wymaga przeprowadzenia klasyfikacji sposobów uogólniania wyników pomiarów.

Ze względu na metodę uzyskiwania wskaźników wydajnościowych można wyróżnić:

a) pomiary bezwzględne;

b) wskaźniki względne.

Ze względu na metodę uporządkowania wyników uogólnionych przy porównywaniu wielokryterialnym można wyróżnić:

a) uzyskanie porządku metodą wyznaczania dla składowych ocen:

- średniej arytmetycznej;
- średniej geometrycznej (czasy pomiarów są dodatnie niezerowe);

b) uzyskanie porządku na podstawie rozkładu wyrównującego.

Wymienione metody porządkowania można stosować zarówno do wskaźników bezwzględnych, jak i względnych.

Jeśli obciążenie (ang. workload) składa się z wykonywanych z tą samą częstością programów P1 i P2, wyżej przedstawione stwierdzenia (g – i) pozwalają określić względny czas wykonania zadania na poszczególnych komputerach. Można też wtedy do porównania wydajności komputerów zastosować średnią arytmetyczną czasów wykonania poszczególnych programów zestawu. Jeśli natomiast programy, składające się na rzeczywiste obciążenie systemu, nie wykonują się z tą samą częstością, można każdemu programowi przypisać wagę, określającą jego procentową częstość występowania w obciążeniu rzeczywistym i obliczyć *ważony czas wykonania zadania*

(1) $$CW = \sum_{i=1}^{n} (w_i * czas_i)$$

gdzie: CW – ważony czas wykonania zadania;

w_i – waga, przyporządkowana i-temu składnikowi zadania;

$czas_i$ – czas wykonania i-tego składnika zadania;

Metoda wyrównywania obciążeń

Rozważmy przykład trzech różnych rozkładów, odpowiadających częstości występowania programów P1 i P2 w obciążeniach dla komputerów A, B i C z tab.1. W celu wyrównania czasu wykonania poszczególnych składowych wagi dla danego rozkładu są uzyskane jako liczby odwrotnie proporcjonalne do czasu wykonania programów P1 i P2 na danym komputerze. Obliczmy dla przykładu wagę dla programu P1, wykonywanego na komputerze B:

(2) $$W1(B) = \frac{T(P2)}{T(P1) + T(P2)}$$

Rozkład RB wyrównuje obciążenia komputera B, bo ważone czasy wykonania zadań P1 (0.909 *1 = 0.909) i P2 (0.091 * 100= 0.910) są w przybliżeniu równe. Uzyskane w ten sposób rozkłady dla wszystkich komputerów są przedstawione w tab.2.

Tab.2. Przykład rozkładów RA, RB, RC dla komputerów A, B i C z tab.1

	Rozkład RA	Rozkład RB	Rozkład RC
Waga dla P1	0.999	0.909	0.50
Waga dla P2	0.001	0.091	0.50

Tab.3. Ważone czasy wykonania zadania CW dla komputerów A, B i C względem rozkładów RA, RB i RC

	Komputer A	Komputer B	Komputer C
CW względem rozkładu RA	2.00	10.09	20.00
CW względem rozkładu RB	91.91	18.19	20.00
CW względem rozkładu RC	500.50	55.00	20.00

Ważony czas wykonania zadania o rozkładzie R, wyrównującym czasy wykonania programów P1 i P2 na danym komputerze można obliczyć jako:

(3) CW(A) = (0.999*1) + (0.001 * 1000) = 0.999 + 1 = 2;

(4) CW(B) = (0.909*10) + (0.091 * 100) = 9.09 + 9.10 = 18.19;

(5) CW(C) = (0.50*20) + (0.50 * 20) = 10 + 10 = 20;

Podobnie można obliczyć pozostałe czasy wykonania zadań dla przyjętych rozkładów. Zacienione pola w tabeli 3 odpowiadają obliczonym powyżej ważonym czasom wykonania zadania dla rozkładu wyrównującego względem macierzystego komputera. Wykonanie na komputerze A programów P1 i P2 zgodnie z rozkładem wyrównującym:

1) dla komputera macierzystego A daje ważony czas wykonania zadania równy 2.00;
2) dla komputera B daje ważony czas wykonania zadania równy 91.91;
3) dla komputera C daje ważony czas wykonania zadania równy już 500.5.

Wobec tego porównywanie danych w poszczególnych wierszach, czyli czasów wykonania zadania, składającego się z programów P1 i P2, wykonujących się zgodnie z przyjętym rozkładem, daje dla każdego wiersza różne wyniki. Według danych z wiersza dla rozkładu RA, komputer A jest 10 razy szybszy od komputera C, a według wiersza dla rozkładu RC komputer C jest ok. 25 razy szybszy od komputera A. Konieczne jest zatem przyjęcie za wzorcowe nie tylko programów, wchodzących w skład zadania testowego, ale i ustalonego rozkładu częstości wykonywania poszczególnych programów testowych.

Metoda wyrównywania obciążeń względem maszyny wzorcowej

Podejście zbliżone do powyższego, polegające na wyrównaniu (normalizacji) czasu wykonania względem maszyny wzorcowej wykorzystano przy projektowaniu wczesnej wersji benchmarku SPEC (System Performance Evaluation Corporation, http://www.specbench.org/), gdzie za wzorcowy przyjęto czas wykonania zadań składowych na komputerze VAX - 11/780. Aktualnie (SPEC CPU2000) komputerem wzorcowym jest UltraSPARC 10 [11]. Badania na testowanym komputerze wykonuje się za pomocą ustalonego zbioru programów składowych, wykonywanych zgodnie z rozkładem wyrównującym, obliczonym względem maszyny wzorcowej. Interpretacja wyników testów polegająca na tym, że czas wykonania programu na komputerze testowanym można oszacować jako iloczyn wskaźnika SPEC danej maszyny i czasu wykonania na maszynie wzorcowej, może niestety okazać się myląca (jak pokazano w tab. 3).

Przy porównywaniu wydajności różnych komputerów można, na przykład, skorzystać ze wspomnianej wcześniej średniej arytmetycznej, bądź też średniej geometrycznej czasów wykonania poszczególnych składowych testu, bezwzględnych bądź znormalizowanych. Analizę zachowania wybranych średnich przeprowadzimy, kontynuując poprzednie dane [1].

Tab.4. Czasy wykonania programów z tab.1, znormalizowane względem każdego komputera

	Normalizacja do A			Normalizacja do B			Normalizacja do C		
	A	B	C	A	B	C	A	B	C
Znormalizowany czas wykonania P1	1.0	10.0	20.0	0.1	1.0	2.0	0.05	0.5	1.0
Znormalizowany czas wykonania P2	1.0	0.1	0.02	10.0	1.0	0.2	50.0	5.0	1.0
Średnia arytmetyczna czasów znormalizowanych	1.0	5.05	10.01	5.05	1.0	1.1	25.03	2.75	1.0
Średnia geometryczna czasów znormalizowanych	1.0	1.0	0.63	1.0	1.0	0.63	1.58	1.58	1.0

Czasy wykonania z tabeli 1 znormalizowano do każdej maszyny. Jak widać, średnia arytmetyczna czasów znormalizowanych zmienia się w zależności od wyboru maszyny wzorcowej (referencyjnej) – w kolumnach zacienionych dla normalizacji względem komputera A średnia ta jest dla B 5 razy większa niż A, podczas gdy w kolumnach zacienionych dla normalizacji dla komputera B jest odwrotnie. Przy normalizacji względem komputera A komputer C jest najwolniejszy, ale przy normalizacji względem C najszybszy. Średnia geometryczna czasów znormalizowanych zachowuje się bardziej konsekwentnie, niezależnie od wyboru maszyny referencyjnej – A i B mają tę samą wydajność, a C jest 1.58 razy szybszy (1 / 0.63 = 1.58).

Podsumowując niedostatki metody, bazującej na normalizowanej średniej arytmetycznej, należy zauważyć, że ponieważ wagi są tu obliczone proporcjonalnie do czasu wykonania programów składowych na maszynie wzorcowej, zależą one nie tylko od częstotliwości występowania poszczególnych składowych w rozkładzie obciążenia, ale też i właściwości wzorcowej maszyny. Niedogodności tej jest pozbawiona, jak widać na podstawie tabeli 4, średnia geometryczna. Niestety, metoda średniej geometrycznej jest obarczona inną, poważną wadą: za jej pomocą nie można wprost przewidzieć czasu wykonania danego zadania – średnie z tabeli sugerują, że wydajność maszyn A i B jest

dla programów P1 i P2 taka sama, a jest to prawda tylko dla bardzo konkretnego rozkładu obciążenia tzn. sto wykonań programu P1 na jedno wykonanie programu P2:

(6) W(100:1) = [100,1];

(7) $CW(100:1)_A$ = (100*1) + (1*1000) = 1100;

(8) $CW(100:1)_B$ = (100*10) + (1*100) = 1100;

(9) $CW(100:1)_C$ = (100*20) + (1*20) = 2200;

Ważone średnie arytmetyczne wskazują, że maszyny A i B są około dwukrotnie szybsze od C, a średnie geometryczne, że C jest o około 58% szybsza od A i B:

(10) GA = sqrt (1 * 1000) = sqrt (1000) = 31.62

(11) GB = sqrt (10 * 100) = sqrt (1000) = 31.62

(12) GC = sqrt (20 * 20) = sqrt (400) = 20

(13) GA/ GC = GB/GC = 31.62 / 20 = 1.58

Przy projektowaniu nowych testów należałoby dążyć do uzyskania takiego wskaźnika, który umożliwiałby nie tylko porównanie wydajności różnych komputerów, ale też przewidywanie czasu wykonania danego obciążenia na podstawie tego wskaźnika i czasu wykonania na maszynie wzorcowej.

Dodatkowo cechą zastosowania metody średniej geometrycznej (użytej między innymi w testach SPEC) jako metody uogólniania wydajności dla zestawu benchmarków jest to, iż dopinguje ona projektantów do skupienia uwagi na tych elementach systemu, dla których zwiększanie wydajności jest najłatwiejsze, a nie na tych, które są na danej maszynie najwolniejsze. Na przykład, jeśli jakieś ulepszenie może skrócić czas wykonania jednego z testów z 2 sekund do 1, ocena, uzyskana za pomocą średniej geometrycznej będzie dla tego przypadku tak samo dobra jak dla skrócenia czasu wykonania innego testu z 10000 do 5000s,

gdyż oczywiście:

(14) $$\sqrt{\frac{2}{1}} = \sqrt{\frac{10000}{5000}} = 1.41.$$

Małe programy są znacznie bardziej podatne na udoskonalenia dające znaczący, lecz nie reprezentatywny wzrost wydajności. W takich przypadkach bardziej obiektywne są miary, uwzględniające całkowity czas wykonania.

4. Ocena ulepszeń

Po omówieniu sposobów definiowania, pomiaru i uogólniania wskaźników wydajnościowych, czas na kilka uwag o ocenie ulepszeń w systemie, zarówno dotyczących architektury środowiska jak i usuwania wąskich gardeł w programach, na których najszybszym wykonaniu nam zależy. Pojawia się pytanie: na ile ulepszenie, przyśpieszające pracę pewnego fragmentu wykonywanego programu, niezależnie od przyczyny, może poprawić wydajność całego badanego obiektu?

O wpływie tego ulepszenia, zwiększającego szybkość działania fragmentu programu na przyspieszenie (ang. speedup) całości mówi *prawo Amdahla* [1],[10], używające dwóch pojęć:

1) Procentowego zakresu ulepszenia, czyli części czasu wykonania programu pierwotnego, która może skorzystać z ulepszenia. Np. jeśli cały proces trwa 60s, a część, którą można ulepszyć 20s, to współczynnik ten, oznaczony roboczo *pzu* wynosi 1/3; pzu < 1;
2) Przyspieszenia (ang. speedup) fragmentu *pf*, równego ilorazowi czasu wykonania fragmentu pierwotnego i fragmentu po ulepszeniu (w tym miejscu milcząco zakładamy ulepszenie programu, ale program może oczywiście pozostać niezmieniony, a ulepszeniu ulec środowisko: procesor, pamięć itd.).

Prawo Amdahla mówi, że możliwe do osiągnięcia przyspieszenie działania całego obiektu, uzyskane dzięki ulepszeniu fragmentu tego obiektu, jest ograniczone czasem, w którym ulepszenie to może zostać użyte.Można to zapisać następującymi zależnościami:

$$(15)\quad Czas\_wykonania\_nowy = Czas\_wykonania\_stary * \left((1 - pzu) + \frac{pzu}{pf} \right)$$

$$(16)\quad Zmiana\_wydajnosci\_calosci = \frac{1}{(1 - pzu) + \left(\frac{pzu}{pf} \right)}$$

Przykład (ocena wpływu ulepszenia) [1]:

Udało się dziesięciokrotnie przyspieszyć fragment programu, wykonujący się początkowo przez 40 % czasu wykonania całego programu. Jakie uzyskano przyspieszenie? Ponieważ pzu = 0.40, pf = 10, zatem otrzymujemy

(17) Przyspieszenie_całości = 1 / [(1 – 0.4) + (0.4/10)] = 1 / [1 – 0.4 + 0.04] = 1 / 0.64 = = 1.56

Gdyby udało się skrócić do zera czas wykonania tej części (w efekcie nieskończonego przyspieszenia fragmentu), największe możliwe przyspieszenie wynosiłoby :

(18) Przyspieszenie_całości_max = 1 / [(1 – 0.4) + (0.4/∞)] = 1 / 0.6 = 1.67

Przykład (ocena rozwiązań alternatywnych) [1]:

Załóżmy, że realizacja instrukcji pierwiastka kwadratowego FPSQRT jest odpowiedzialna za 20% czasu wykonania benchmarku ma pewnej maszynie. Jedną z opcji zwiększenia wydajności jest dodanie sprzętu, który będzie wykonywał tę operację 10-krotnie szybciej. Drugą opcją jest dwukrotne przyspieszenie wykonania wszystkich instrukcji zmiennoprzecinkowych, które są odpowiedzialne za 50% czasu wykonania benchmarku. Które rozwiązanie da większe przyspieszenie? W wyniku obliczeń otrzymujemy:

(19) Przyspieszenie (FRSQRT) = 1 / [1 – 0.2 + (0.2 / 10)] = 1 / 0.82 = 1.22

(20) Przyspieszenie (ZMP) = 1 / [1 – 0.5 + (0.5 / 2)] = 1 / 0.75 = 1.33

Jeśli założenia projektowe były realne, lepsze wyniki uzyskamy inwestując w niewielkie ulepszenie częściej występujących w teście instrukcji zmiennoprzecinkowych.

5. Określenie wymagań na narzędzie do wspomagania badania wydajności

Jak oszacować częstotliwość występowania danej instrukcji w programie, jak znaleźć wąskie gardło systemu? Jednym ze sposobów poza, na przykład, zastosowaniem drogich monitorów sprzętowych, jest zastosowanie programu typu profiler, umozliwiającego wykonanie następujących operacji:

1) generację wymuszeń, sterujących praca programu,
2) pomiar i zapamiętanie badanych parametrów,
3) przetwarzanie ich w celu odrzucenia informacji nadmiarowych,
4) prezentację w czytelny sposób wyników.

Przy tym profiler taki powinien ponadto mieć następujące właściwości:

1) nie obciążać badanego systemu,
2) być dokładny,
3) rejestrować zdarzenia na różnym poziomie abstrakcji,
4) mieć szeroki zakres zastosowań,
5) być zgodny z szeroka gamą badanych produktów (procesorów, systemów),
6) integrować się z procesem projektowania, dostarczając niezbędnych wskazówek dla wykrycia i usunięcia zatorów,
7) być łatwy w użyciu,
8) oczywiście na dodatek być tani.

6. Przykład narzędzia, pomocnego przy badaniu wydajności

Przykładem produktu, który częściowo spełnia powyższe kryteria jest program VTUNE Performance Analyzer ver.4.5 firmy Intel [5]. Pozwala on na zbieranie, analizowanie i zobrazowanie informacji dotyczących wydajności programów, pracujących na procesorach, wyprodukowanych przez Intel, od i486 wzwyż, pod kontrolą systemów operacyjnych Microsoft Windows 95/98/NT (http://developer.intel.com/vtune/).

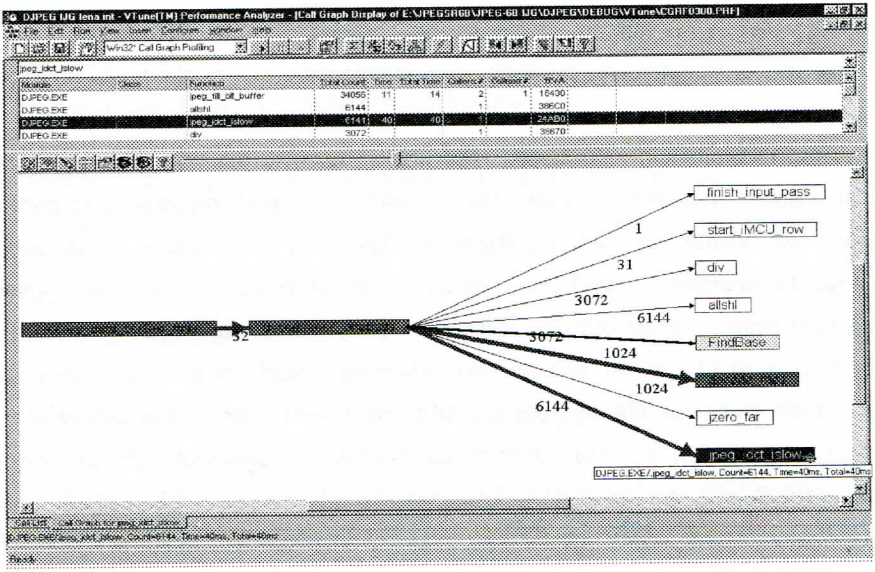
Badaniu poddawany jest program wykonywalny (np. plik typu .exe lub .dll), przy czym jeśli dysponujemy kodem źródłowym tego programu (w językach C, Java, Fortran, asembler), dla disasemblingu potrzebna jest kompilacja do postaci z informacjami dla debuggera. Można badać sam program wykonywalny bez dostępnego kodu źródłowego, jednak czytelność prezentowanych w czasie analizy fragmentów programu jest wtedy mniejsza, gdyż uzyskujemy tylko kod asemblerowy dla docelowego procesora. Program VTune wykonuje próbkowanie (ang. sampling) czasu wykonania badanego programu wykonywalnego przy przyjętych parametrach uruchomienia, rejestrując adresy wykonywanych w czasie próbkowania instrukcji. Próbkowaniu podlegają nie tylko instrukcje, należące do badanego kodu, ale też wszystkie zadania aktywne w tle, w tym zadania systemowe. Należy zauważyć, że możemy dzięki temu badać program wykonywalny w jego naturalnym środowisku, bez przyjmowania często fałszywych założeń co do stanu obciążenia komputera zadaniami systemowymi. Sampling może być wykonany jako time-based i event-based czyli próbkowanie wyzwalanie jest zegarem lub określonymi zdarzeniami (np. chybieniem pamięci cache). Według danych firmy Intel narzut wnoszony przez zbieranie informacji o adresach punktów wykonania jest rzędu 1 % czasu trwania samplingu. Po uzyskaniu adresów program umożliwia analizę punktów

krytycznych (ang. hotspotów) badanej aplikacji. Ponieważ uzyskane na podstawie próbkowania wykresy są bardzo użyteczne do zgrubnej analizy zachowania badanej aplikacji czy też jej fragmentów, przedstawimy przykładowe zrzuty ekranowe, uzyskane w czasie analizy rzeczywistej aplikacji. Badanym obiektem jest program, służący do dekompresji pliku typu „.jpg" do postaci typu „.bmp". Jako wzorcowy, przyjęto program „djpeg", autorstwa Thomasa G. Lane[5]. Tekst źródłowy został skompilowany za pomocą kompilatora MS Visual C++ v.6 i poddany analizie przy dekompresji (powszechnie uznawanego za wzorcowy w tej klasie badań) obrazu „lena.jpg". Na podstawie analizy częstości wywołania procedur (funkcja „callgraph" programu VTune - rys.1) stwierdzono, że procedura jpeg_idct_islow, wykonująca odwrotną transformatę kosinusową, jest wywoływana dla badanego obrazu „lena.jpg" 6144 razy. Jest to jedna z dwóch najliczniej wykonywanych procedur, jest więc dobrym kandydatem do ulepszeń (rys. 2). Badana procedura, znajdująca się w pliku *jidctint.c* spowodowała ok. 20% próbek czasu wykonania, zarejestrowanych w czasie samplingu. Po zastosowaniu zmian w procedurze, zasugerowanych w [3,4], a polegających na wykorzystaniu do obliczeń IDCT instrukcji MMX, uzyskano znaczącą zmianę w zachowaniu programu, przedstawioną na rys. 3. Zmodyfikowana procedura spowodowała już tylko ok. 6% zarejestrowanych próbek, co oznacza około trzykrotne przyspieszenie działania badanej procedury.

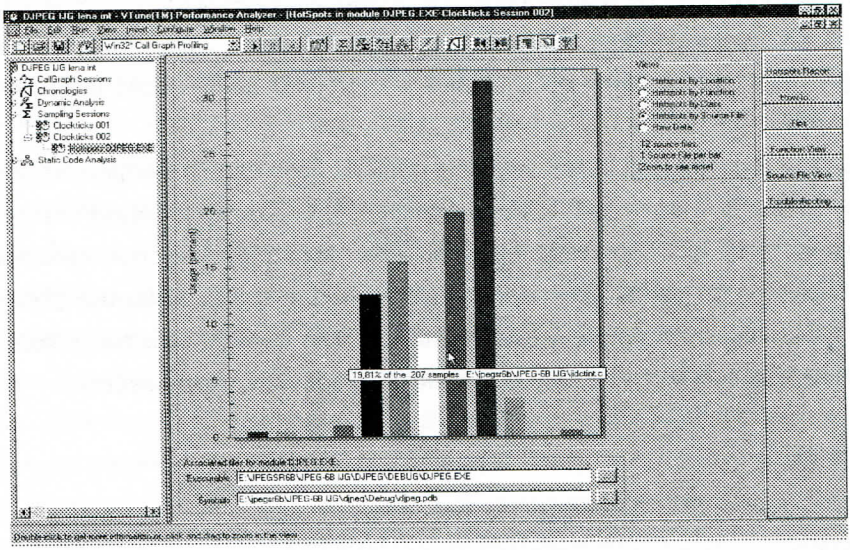
Wyniki te, pokazujące skuteczność przeprowadzonych zmian (przyspieszenie badanego fragmentu programu około trzykrotnie), zostały potwierdzone inną metodą, przedstawioną w [6].

Program VTune dysponuje również funkcją „code coach", umożliwiającą uzyskanie podpowiedzi co do rodzaju instrukcji czy konstrukcji programowej, dającej szansę na uzyskanie większej wydajności badanego kodu. Jest też możliwa statyczna i dynamiczna analiza badanego programu, dzięki czemu można między innymi analizować położenie kodu w poszczególnych liniach pamięci cache, badać dane o wykonaniu kodu na starszych typach procesorów bez zmian w konfiguracji sprzętowej komputera.

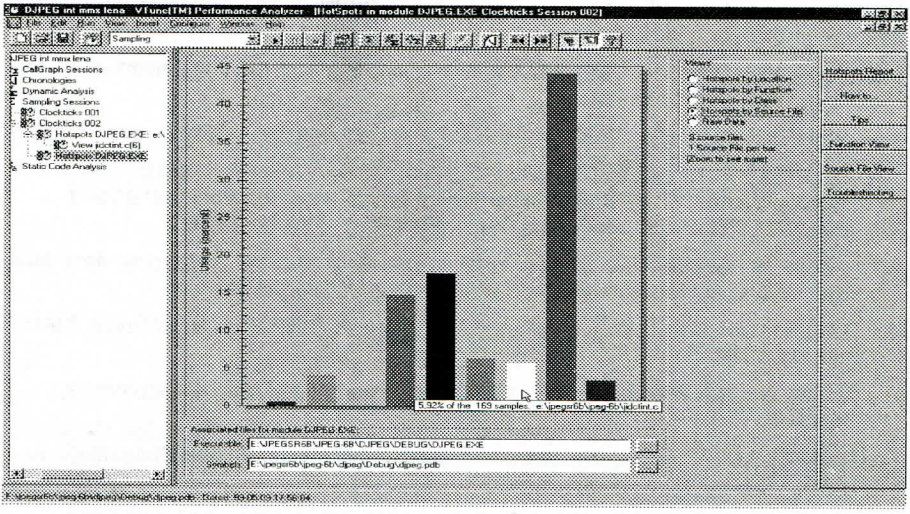
[5] Linki do tekstów źródłowych programu można znaleźć na stronie organizacji Indepedent JPEG Group (http://www.ijg.org).



Rys.1 Graficzne przedstawienie zależności między procedurami badanego programu



Rys.2. Rozkład obciążenia, generowanego przez wykonywany program, jako funkcji adresu w kodzie źródłowym programu



Rys. 3. Rozkład obciążenia, generowanego przez wykonywany program, jako funkcji adresu w kodzie źródłowym programu po modyfikacji

7. Podsumowanie

Na podstawie zaprezentowanych rozważań można zauważyć trudności, występujące przy próbie określenia obiektywnej miary wydajności komputerów. Należy rozwiązać problem wyboru wskaźników jakości, jak też problem wyboru metod, pozwalających na porównywanie różnych komputerów. W artykule omówiono słabe strony wspomnianych metod, wskazując na metodę ważonej średniej geometrycznej jako pozbawionej części niedostatków innych metod. Omówiono również zastosowanie programu profilującego VTune firmy Intel przy badaniu wydajności przykładowej aplikacji, wykonywanej na wybranych komputerach. W przeprowadzonym eksperymencie potwierdzono tezę o możliwości ułatwienia procesu podejmowania decyzji projektowych przy tworzeniu własnych benchmarków dzięki narzędziom tego typu.

Literatura

[1] David A. Patterson, John L. Hennessy, *Computer Architecture. A Quantitative Approach. Second Edition*, Morgan Kaufmann Publishers Inc. San Francisco, California 1996.

[2] Piotr Metzger, *Diagnostyka i optymalizacja komputerów*, Wydawnictwo Helion, Gliwice 2000.

[3] Intel, Application Note 528, Using MMX™ Instructions in a Fast iDCT Algorithm for MPEG Decoding, 1997. Intel Corporation.

[4] Intel, Application Note 886, JPEG Inverse DCT and Dequantization Optimized for Pentium® II Processor, 1997. Intel Corporation.

[5] Intel, Getting Started VTune Intel's Visual Tuning Environment for Windows* 95 and Windows NT* Developers, 1997. Intel Corporation.

[6] Artur Miktus, *Metoda pomiaru czasu wykonania sekwencji kodu dla procesorów firmy Intel*, Biuletyn Instytutu Automatyki I Robotyki WAT nr 11/2000 Warszawa.

[7] J.J. Dongarra, J.R. Bunch, C.B. Moler, and G.W. Stewart, LINPACK *Users' Guide*, SIAM, Philadelphia, 1979.

[8] H.J.Curnow and B.A.Wichmann, *A Synthetic Benchmark*. The Computer Journal 19,1 (1976), 43-49.

[9] Reinhold Weicker, A detailed look at some popular benchmarks, Parallel Computing, No. 17 (1991), 1153-1172.

[10] Amdahl, G., The Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities, AFIPS Conf. Proc. 30, pp. 483--485, 1967.

[11] Michael Riepe, *New Version of the SPEC test suite: CPU2000* http://www.heise.de/ix/artikel/E/2000/05/127/

Recenzent: dr hab. inż. Roman Kulesza, prof. WAT
Praca wpłynęła do redakcji 30.06.2000