BIULETYN INSTYTUTU AUTOMATYKI I ROBOTYKI
NR 17, 2002

Badanie szybkości kopiowania bloków pamięci operacyjnej

Artur Miktus

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

STRESZCZENIE: W pracy przedstawiono wyniki badania szybkości kopiowania bloków pamięci operacyjnej przy wykorzystaniu różnych systemów komputerowych klasy PC, pracujących pod kontrolą systemów operacyjnych rodziny Microsoft Windows. Na podstawie uzyskanych wyników dokonano wyboru najszybszej spośród badanych metod dla określonych grup systemów komputerowych

1. Wprowadzenie

W wielu programach komputerowych, stosowanych obecnie, jednym z kluczowych zadań jest kopiowanie znacznych obszarów pamięci operacyjnej. Kopiowanie to można wykonać, korzystając z wielu różnych metod, przy czym brak jest, jak dotąd analizy szybkości kopiowania w zależności od użytej metody i typu zastosowanego systemu komputerowego. W ramach prezentowanego artykułu przedstawionych zostanie 8 wybranych metod kopiowania dla bloków pamięci operacyjnej (PAO). Metody te zostały zaimplementowane w programie testowym, badającym szybkość kopiowania bloków PAO o wielkości od 1 MB do 256 MB. Badania zostały przeprowadzone przy wykorzystaniu 12 systemów komputerowych, wyposażonych w różne procesory (AMD K6 3D, AMD Athlon, Intel Celeron II, Intel Pentium III, Intel Pentium 4), różne wielkości pamięci operacyjnych (128 MB, 256 MB, 384 MB, 512 MB), różne typy pamięci operacyjnych (SDRAM, DDR, RDRAM) i pracujące pod kontrolą różnych systemów operacyjnych (Microsoft Windows w wersjach NT4, 98 SE, Me, 2000 Pro, XP HE, XP Pro). Dany system komputerowy był traktowany jako

obiekt badania nie podlegający zmianom konfiguracji. Dla każdego systemu komputerowego badania zostały przeprowadzone każdą z ośmiu metod, za wyjątkiem komputera z procesorem AMD K6 3D, który nie był w stanie wykonać rozkazów z grupy SSE. Kryterium wyboru wielkości bloków PAO było oparte o eksperyment początkowy, podczas którego badano, jaka wielkość bloku PAO nie powoduje wykorzystania przez system operacyjny mechanizmu pamięci wirtualnej. Zwykle były to bloki o wielkości od 1 MB do połowy wielkości zainstalowanej w systemie pamięci operacyjnej.

Inne parametry, mające wpływ na uzyskane wyniki, nie podlegające jednak celowym zmianom podczas przeprowadzenia badań, to:

1) rodzaj procesora, w tym lista obsługiwanych instrukcji;
2) szybkość taktowania procesora;
3) wielkość pamięci cache każdego występującego w badanym systemie poziomu szybkiej pamięci buforowej (cache);
4) rodzaj odwzorowania każdego występującego w badanym systemie poziomu szybkiej pamięci buforowej (cache);
5) strategia postępowania przy chybieniu przy zapisie do pamięci cache;
6) szybkość pamięci cache każdego występującego w badanym systemie poziomu szybkiej pamięci buforowej (cache);
7) wielkość linii pamięci cache każdego występującego w badanym systemie poziomu szybkiej pamięci buforowej (cache);
8) wielkość pamięci operacyjnej;
9) rodzaj pamięci operacyjnej (SDRAM, DDR, RDRAM);
10) parametry czasowe pamięci operacyjnej (liczba taktów opóźnienia RAS, liczba taktów opóźnienia CAS, częstotliwość taktowania magistrali pamięci);
11) typ chipsetu płyty głównej;
12) typ systemu operacyjnego (w tym mechanizmy zarządzania pamięcią: przydzielanie i zwalnianie bloków PAO, metoda przeprowadzania defragmentacji);
13) rodzaj kompilatora;

    rodzaj optymalizacji, użyty w czasie kompilacji kodu.

2. Algorytm badania

Celem przeprowadzonych badań było stwierdzenie, która z zaproponowanych metod okaże się najszybsza dla każdego z badanej grupy systemów komputerowych. Z dotychczas opublikowanych materiałów [2], [4], [5] wynikało, że metody kopiowania bloków PAO, stosujące tzw. hint „PREFETCH" uzyskują najlepsze wyniki dla niewielkich porcji danych, o wielkości do 2 kB. Dla aplikacji multimedialnych tego rodzaju wielkości nie są wystarczające, często występuje tu potrzeba kopiowania znacznie większych bloków, o wielkości mierzonej w megabajtach. W sytuacji, gdy kopiowane bloki nie mieszczą się w całości w pamięci podręcznej procesora (typowo 8-512 kB dla pamięci podręcznych pierwszego i drugiego poziomu), zastosowanie innych metod, szczególnie korzystających z rejestrów MMX i unikających „zanieczyszczania" pamięci podręcznej przy chybieniu podczas zapisu, może okazać się bardziej wydajne.

Przyjęto następujący algorytm badania szybkości kopiowania bloku PAO:

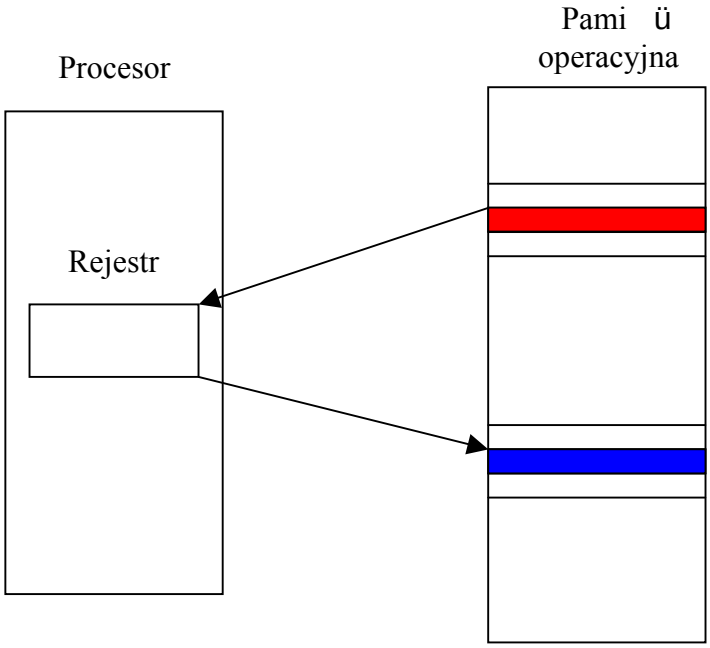
1. Alokacja na stercie dwóch bloków przyjętej wielkości (możliwe rozmiary pojedynczego bloku to 1 MB, 2 MB, 4 MB, 8 MB, 16 MB, 32 MB, 64 MB, 96 MB, 128 MB, 192 MB, 256 MB) – na tych blokach będą wykonywane kolejno operacje przygotowawcze i kopiowania różnymi metodami;
2. „Rozgrzewka buforów" – dwukrotne wykonanie kopiowania metodą „**movq**" (środkową w cyklu badań) dla niezainicjowanej zawartości bloku źródłowego;
3. Dla każdej spośród 8 badanych metod wykonanie następującej sekwencji działań:
    a) zapis do każdej z 4-bajtowych komórek bloku źródłowego PAO liczby całkowitej o wartości od „rozmiar bloku" do 1 za pomocą EAX;
    b) zerowanie bufora docelowego przez zapisywanie do każdego 4-bajtowego słowa bufora wyzerowanej uprzednio zawartości EAX;
    c) zainicjowanie pomiaru czasu wykonania procedury (QueryPerformanceCounter);
    d) wywołanie **badanej procedury** kopiowania bloku źródłowego do bloku docelowego;
    e) zakończenie pomiaru czasu wykonania po zakończeniu procedury kopiowania;
    f) zapisanie wyników pomiaru na ekran i do pliku dyskowego;

Dla zwiększenia wiarygodności uzyskanych wyników cykl badań dla każdej metody powtarzany był trzykrotnie, a za wynik pomiaru uznany został

najkrótszy z uzyskanych czasów kopiowania – przyjęto założenie, że wydłużenie zmierzonego czasu w stosunku do najkrótszego bierze się z wykonywania przez system w tle innych zadań. Do scharakteryzowania danej metody kopiowania przyjęto średnią arytmetyczną najlepszych czasów kopiowania dla wszystkich zastosowanych wielkości bloków PAO.

3. Badane metody kopiowania

Pierwsza z badanych metod wykorzystuje procedurę biblioteczną Microsoft Visual C++, o nazwie *memcpy*. Wyniki badania szybkości kopiowania bloków PAO tą metodą będą stanowiły ujednoczone punkt odniesienia do oceny innych, zastosowanych w badaniu, metod. Działanie procedury polega na kopiowaniu zadanej liczby bajtów spod adresu źródłowego pod adres docelowy w pamięci operacyjnej. Jej zastosowanie nie nakłada na programistę żadnych wymagań, dotyczących znajomości architektury stosowanego systemu komputerowego.



Rys. 1. Uproszczony model systemu komputerowego, wykorzystywany w metodach mem0klas, mem1klas, mem2klas i mem1

Kolejne metody, przedstawione poniżej, zakładają posiadanie przez programistę wiedzy na temat asemblera procesorów x86, rozszerzeń listy

rozkazów o nowe rozkazy MMX i SSE, a także zasad działania pamięci cache współczesnych procesorów.

Kolejne 3 metody, o nazwach odpowiednio **mem0klas**, **mem1klas** i **mem2klas** przyjmują uproszczony model systemu komputerowego, przedstawiony na rys.1.

Przesyłane będą kolejno w jednej iteracji:

1. dla metody mem0klas 1 bajt (rejestr al.);
2. dla metody mem1klas 4 bajty szeregowo (rejestr al);
3. dla metody mem2klas 4 bajty równolegle (rejestr eax).

Algorytm metody **mem0klas** przedstawiony został poniżej:

```
void mem0klas  (void *dst, void *src, int nbytes)
{
      _asm {
                  cld                  // kierunek rosnący
                  mov   esi, src             // esi: adres bloku źródłowego
                  mov   edi, dst             // edi: adres bloku docelowego
                  mov   ecx, nbytes  // liczba bajtów do skopiowania
      loop1:
                  mov   al,  byte ptr [esi]    // czytaj bajt z PAO do al
                  mov   byte ptr [edi],al      // zapisz bajt z al do PAO

                  add   esi, 1
                  add   edi, 1
                  dec   ecx
                  jnz   loop1
      }
}
```

Elementarna metoda mem0klas, kopiująca bajt po bajcie powinna, według przewidywań autora, dawać najgorsze wyniki spośród metod zastosowanych w badaniu.

Algorytm metody **mem1klas**, jako modyfikacja powyższego algorytmu, przedstawia się następująco:

```
void mem1klas  (void *dst, void *src, int nbytes)
{
      _asm {
                  cld                  // kierunek rosnący
                  mov   esi, src
                  mov   edi, dst
```

```
                mov   ecx, nbytes
                shr   ecx, 2          // 4 bajty na iterację

        loop1:
                mov   al,  byte ptr [esi+3] // czytaj
                mov   byte ptr [edi+3],al

                mov   al,  byte ptr [esi+2] // czytaj
                mov   byte ptr [edi+2],al

                mov   al,  byte ptr [esi+1] // czytaj
                mov   byte ptr [edi+1],al

                mov   al,  byte ptr [esi+0] // czytaj
                mov   byte ptr [edi+0],al

                add esi, 4
                add edi, 4
                dec ecx
                jnz loop1
        }
}
```

Tradycyjne rozwinięcie pętli powinno, według autora, poprawić wyniki dzięki czterokrotnie mniejszej w porównaniu z metodą mem0klas liczbie skoków warunkowych, wykonywanych w czasie realizacji procedury.

Inny sposób ulepszenia w stosunku do mem0klas zastosowano w metodzie mem2klas:

```
void mem2klas (void *dst, void *src, int nbytes)

{
        _asm {
                cld                   // kierunek rosnący
                mov   esi, src
                mov   edi, dst
                mov   ecx, nbytes
                shr   ecx, 2 // 4 bajty na iterację
        loop1:
                mov   eax, [esi] // czytaj czterobajtowe słowo PAO do eax
                mov   [edi],eax

                add   esi, 4
```

```
                add   edi, 4
                dec   ecx
                jnz   loop1
        }
}
```

W metodzie tej wykorzystany został 4-bajtowy rejestr EAX, co również w stosunku do mem0klas powinno, według autora, zmniejszyć liczbę wykonywanych skoków warunkowych, a co za tym idzie zwiększyć szybkość kopiowania bloku PAO.

Po wykonaniu wstępnych badań okazało się, że zwiększenie szerokości rejestru, przez który dokonywane jest kopiowanie, poprawia szybkość wykonywania operacji. Uzasadnione wobec tego wydaje się przypuszczenie, że wykorzystanie 8-bajtowych rejestrów MMX i instrukcji przesyłania danych 8-bajtowych **movq** pozwoli na dalsze zwiększenie szybkości kopiowania w procedurze **mem1**:

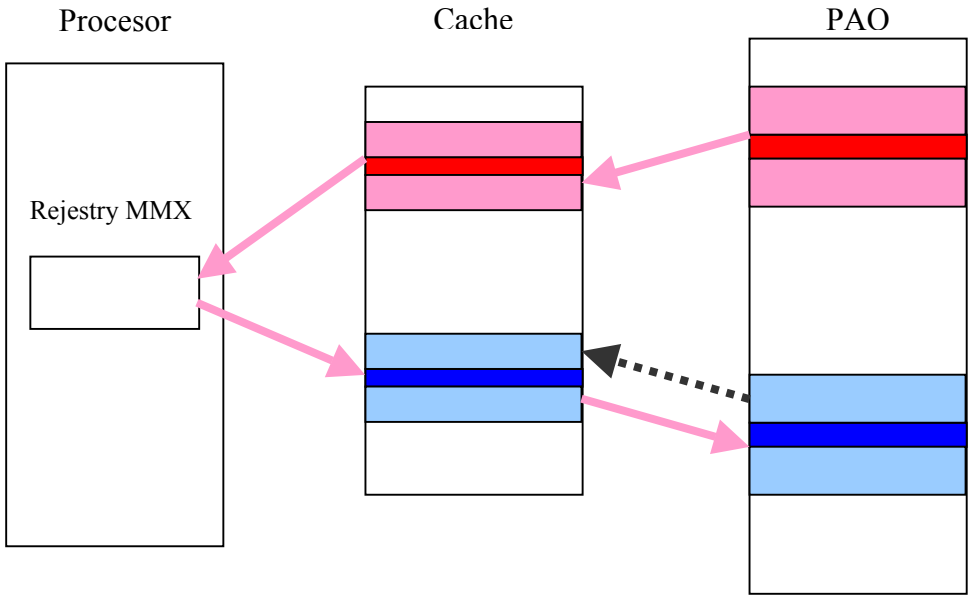
```
void mem1 (void *dst, void *src, int nbytes)
        {
            _asm {
                        mov esi, src
                        mov edi, dst
                        mov ecx, nbytes
                        shr ecx, 6 // 64 bajty na iterację
            loop1:
                        movq mm1,  0[ESI] // Czytaj
                        movq mm2,  8[ESI]
                        movq mm3, 16[ESI]
                        movq mm4, 24[ESI]
                        movq mm5, 32[ESI]
                        movq mm6, 40[ESI]
                        movq mm7, 48[ESI]
                        movq mm0, 56[ESI]

                        movq  0[EDI], mm1 // Pisz
                        movq  8[EDI], mm2
                        movq 16[EDI], mm3
                        movq 24[EDI], mm4
                        movq 32[EDI], mm5
                        movq 40[EDI], mm6
                        movq 48[EDI], mm7
                        movq 56[EDI], mm0

                        add esi, 64
                        add edi, 64
```

```
                    dec ecx
                    jnz loop1
                    emms
                }
}
```

Zastosowana instrukcja **movq** kopiuje 64-bitowe poczwórne słowo (ang. quadword) z miejsca źródłowego do docelowego, przy czym co najwyżej jeden z argumentów może znajdować się w pamięci operacyjnej.



Rys. 2. Uproszczony model systemu komputerowego, wykorzystywany w metodach mem2, mem3 i mem4

W kolejnych metodach zastosowano zmodyfikowany model badanego systemu komputerowego (rys. 2) – wykorzystano występowanie we współczesnych procesorach pamięci cache, której zadaniem jest przechowywanie danych i instrukcji bliżej procesora w hierarchii podsystemu pamięci, dzięki czemu czas dostępu do żądanej informacji ulega skróceniu.

Poprzednie metody oczywiście niejawnie wykorzystywały pamięć cache, która dotąd była „przezroczysta" dla programisty. Jednak typowo podczas zapisu do pamięci danych przy chybieniu jest stosowana strategia „**write-allocate**", która polega na transmisji z PAO do cache całej linii pamięci cache (32 lub 64 bajty) z nadzieją, że skoro przed chwilą dana ta była zapisywana, to za chwilę będziemy chcieli ją bądź jej otoczenie ponownie odczytać. Przy kopiowaniu bloków PAO to zachowanie sterownika pamięci cache nie jest korzystne, gdyż:

- **tracimy czas** na kopiowanie linii z PAO z bloku docelowego do cache, tworząc bufor docelowy, którego zawartość nie jest dalej potrzebna w cache;
- „zanieczyszczamy" cache danych niepotrzebną zawartością bloku docelowego, **tracąc miejsce** na ewentualne obszary, kopiowane z bloku źródłowego.

Istnieje możliwość uniknięcia tego niekorzystnego zjawiska dzięki zastosowaniu instrukcji SSE **movntq**, co przedstawiono w metodzie **mem2**:

```
void mem2 (void *dst, void *src, int nbytes)
{
      _asm {
                  mov   esi, src
            mov   edi, dst
            mov   ecx, nbytes
            shr     ecx, 6 // 64 bajty na iterację

        loop1:
            movq mm1,  0[ESI] // Czytaj
            movq mm2,  8[ESI]
            …

            movntq  0[EDI], mm1 // Non-temporal zapisz
            movntq  8[EDI], mm2
            …

            add esi, 64
            add edi, 64
            dec ecx
            jnz loop1

            emms
      }
}
```

Jeśli zamiast instrukcji **movq** zastosujemy w części kodu, kopiującej dane z rejestrów MMX do PAO instrukcje **movntq**, obszar jasnoszary w docelowym buforze w cache nie wystąpi, nie wystąpi też przesłanie powrotne z bloku docelowego PAO do cache, oznaczone ciemną, przerywaną strzałką (rys.2). Informacje z rejestrów zostaną przesłane **bezpośrednio** do pamięci operacyjnej.

Kolejna metoda **mem3,** kopiująca 64 bajty (8 ośmiobajtowych rejestrów MMX) na jedną iterację, wykorzystuje instrukcję SSE, służącą

do podpowiedzenia procesorowi (tzw. hint), iż programista uważa za pożyteczne sprowadzenie zawartości lokacji PAO do określonego poziomu pamięci cache: „**PREFETCH”**

```
void mem3 (void *dst, void *src, int nbytes)
{
_asm {
      mov esi, src
      mov edi, dst
      mov ecx, nbytes
      shr ecx, 6 // 64 bajty na iterację

loop1:
      prefetchnta 64[ESI] // pobierz wstępnie daną dla kolejnej pętli, non-
                          temporal
      prefetchnta 96[ESI]

      movq mm1,  0[ESI] // Czytaj dane źródłowe
      movq mm2,  8[ESI]
        …
      movntq  0[EDI], mm1 // Non-temporal zapisz
      movntq  8[EDI], mm2
        …
      add esi, 64
      add edi, 64
      dec ecx
      jnz loop1

      emms
        }
}
```

Instrukcja prefetch pobiera linię pamięci cache, zawierającą odczytywaną lokację pamięci do warstwy hierarchii pamięci cache, wskazywanej przez hint. W przypadku, gdy żądany obszar znajduje się w wyższej warstwie cache, nie jest podejmowane żadne działanie. Wśród kilku możliwych hintów występuje **prefetchnta** czyli „prefetch data into non-temporal cache structure”. Hinty te są zależne od implementacji logiki procesora i mogą być przez procesor zignorowane. Ilość pobieranych danych zależy od implementacji, minimum to 32 bajty (dlatego odczytywane są dwa obszary PAO: 64[esi] i 96[esi]). Dzięki występowaniu we współczesnych procesorach kilku kolejek **load/store**, mamy nadzieję na równoczesne pobranie do bufora

roboczego w cache linii, potrzebnych w następnej iteracji, w czasie przetwarzania iteracji bieżącej.

Ostatnia badana metoda kopiowania **mem4** zakłada, że być może kopiowanie wykona się jeszcze szybciej, jeśli zostanie rozbite na etapy, w czasie których jednorazowo będzie kopiowany obszar PAO, o wielkości pozwalającej na jego całkowite zmieszczenie się w pamięci cache L1. W zaimplementowanym przykładzie przyjęto wielkość takiego obszaru równą 2 kB. Wtedy pamięci cache L1 i L2 danych będą mogły działać z pełną szybkością, a w tle procesor zajmie się wykonywaniem „swojej" porcji przetwarzania.

Kopiowanie będzie odbywać się dwuetapowo, metodami już przedstawionymi, czyli wykorzystując **movq**, **movntq** i **prefetchnta**. Pierwszy etap to kopiowanie 2 KB porcji danych z PAO do cache z wykorzystaniem ośmiu 8-bajtowych rejestrów MMX (2048 B/ 64 B = 32 iteracje), za pomocą sekwencji instrukcji:

```
movq mm1, 0[esi]
 ...
movq 0[edi], mm1.
```

Ta 2 KB porcja danych zapisywana przez **movq** powinna się zmieścić w cache L1. Po wypełnieniu tego 2 KB bufora w cache L1, nastąpi ponowne przepisanie jego zawartości do bufora docelowego z wykorzystaniem poniższej sekwencji instrukcji:

```
movq mm1, 0[esi]
 ...
movntq 0[edi], mm1
```

Właśnie ta metoda jest wskazywana jako najszybsza w dostępnych publikacjach [4], [5].

4. Wyniki badań

Wyniki, uzyskane przy badaniu szybkości kopiowania bloków PAO różnej wielkości przedstawionymi metodami dla dostępnych systemów komputerowych przedstawiono w tabeli 1. Wyniki szczegółowe badania pojedynczego systemu komputerowego (na przykładzie AMD Athlon(tm) XP 1700+ (Palomino) 1474.3 MHz 256 MB DDR 268.0 MHz TOP W98 SE) przedstawiono w tabeli 2.

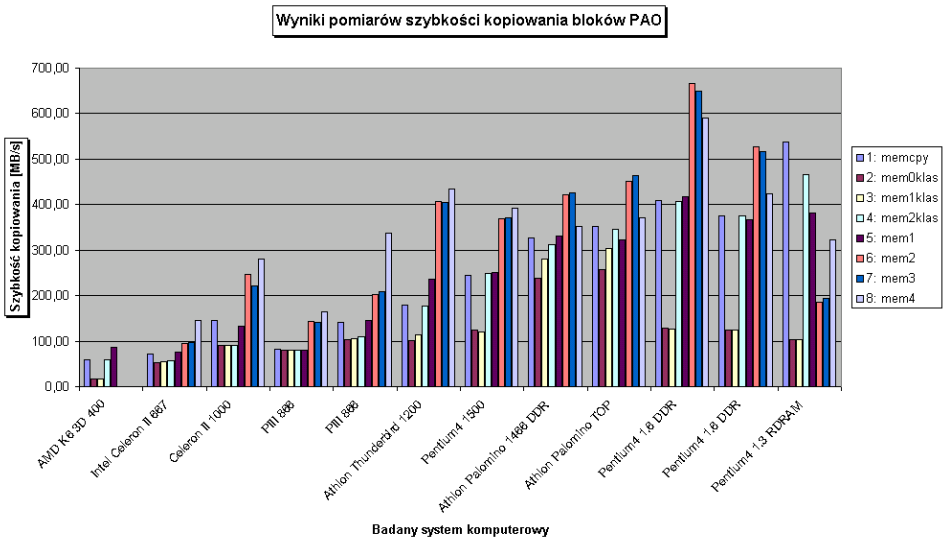
Tab.1. Wyniki badania szybkości kopiowania bloków PAO różnej wielkości przedstawionymi metodami dla 12 różnych systemów komputerowych.

Parametry badanego systemu komputerowego							Średnia z pomiarów bloków różnych wielkości							
Nr systemu	**Procesor**	**Zegar [MHz]**	**Typ PAO**	**Wielkość [MB]**	**Zegar PAO [MHz]**	***System operacyjny***	**memcpy**	**mem0klas**	**mem1klas**	**mem2klas**	**mem1**	**mem2**	**mem3**	**mem4**
1	K6 3D	400	SDRAM	128	100	Win NT4 SP6	58,41	17,63	17,75	59,09	86,0	brak	brak	brak
2	Celeron II	667	SDRAM	256	100	Win 98 SE	72	53	55	58	76	94	96	146
3	Celeron II	1000	SDRAM	256	100	Win 2K Pro	145	90	90	91	133	247	222	280
4	Pentium III	866	SDRAM	512	133	Win XP Pro	83	79	80	80	80	143	142	165
5	Pentium III	866	SDRAM	384	133	Win 2K Pro	141	103	105	110	145	202	208	337
6	Athlon Thunderbird	1200	SDRAM	256	133	Win XPPro	179	101	113	178	236	408	404	435
7	Pentium4	1500	SDRAM	256	100	Win 2K Pro	245	124	121	248	250	369	371	393
8	Athlon Palomino	1466	DDR	256	266	Win 98 SE	326	239	280	312	330	422	425	352
9	Athlon Palomino	1466	DDR	256	266	Win 98 SE	351	257	303	345	322	452	465	370
10	Pentium4	1600	DDR	128	266	Win XP HE	410	130	127	407	417	666	650	589
11	Pentium4	1600	DDR	256	266	Win Me	375	124	125	376	367	527	517	423
12	Pentium4	1300	RDRAM	128	400	Win 98 SE	538	104	104	465	382	185	194	323

Graficzną prezentację wyników badania szybkości kopiowania bloków PAO dla różnych systemów komputerowych przedstawia rys.3. Z rysunku tego wynika, że najszybciej operację kopiowania bloków PAO o wielkościach z przyjętego zakresu (1..256 MB) realizuje system komputerowy, wykorzystujący procesor Intel Pentium 4 i pamięci DDR, osiągając dla najszybszej metody **mem2** około 666 MB/s. Rywalizujący z nim system komputerowy, wykorzystujący procesor AMD Athlon XP 1700+ i pamięci DDR, osiągający najlepsze rezultaty dla metody **mem3** jest w stanie kopiować bloki PAO z szybkością około 465 MB/s.

Tab.2. Wyniki badania szybkości kopiowania bloków PAO dla systemu komputerowego nr 9 z tab.1.

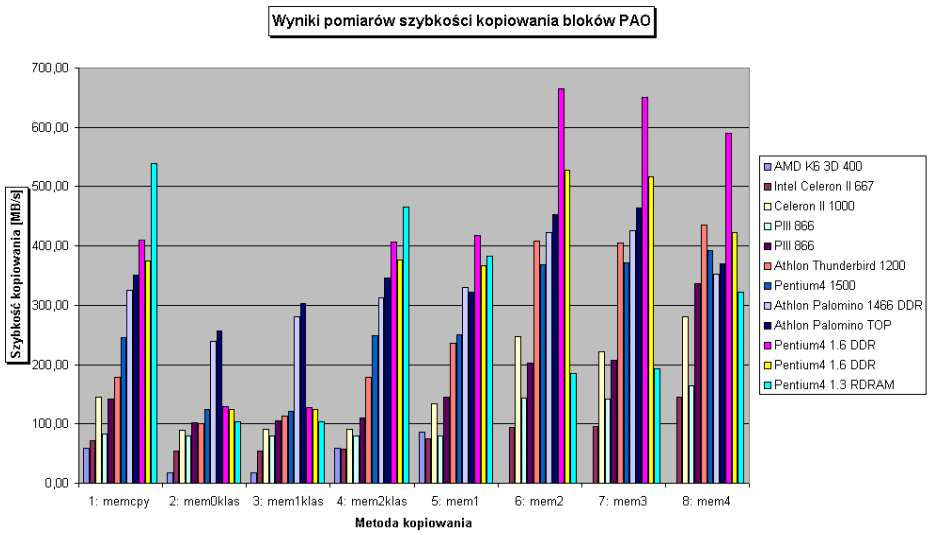
Metoda	Wielkość bloku						
	1 MB	2 MB	4 MB	8 MB	16 MB	32 MB	64 MB
memcpy	343,96	348,42	343,81	361,82	355,32	354,74	352,14
mem0klas	270,56	236,27	254,04	260,24	257,72	257,83	259,74
mem1klas	302,76	270,26	301,73	325,58	305,19	303,37	310,37
mem2klas	332,55	344,6	349,93	342,09	350,75	349,59	347,31
mem1	367,25	127,25	346,15	356,21	353,48	350,73	351,96
mem2	402,01	447,64	459,31	458,81	463,71	465,32	468,83
mem3	469,2	425,91	465	469,18	478,6	471,61	472,41
mem4	367,02	372,93	350,21	364,59	380,8	378,55	376,69



Rys. 3. Wyniki badania szybkości kopiowania dla różnych systemów komputerowych

Wyraźnie odstaje procesor Intel Pentium 4 z pamięcią RDRAM (typowy komputer „z supermarketu"), dla którego zamiast spodziewanego polepszenia szybkości przy kolejnych ulepszeniach metod kopiowania, zaobserwowano znaczący spadek tej szybkości w stosunku do metody **memcpy**. Procesory Intel Pentium III oraz Intel Celeron II, korzystające z pamięci SDRAM, charakteryzują się bardzo zbliżoną szybkością kopiowania bloków PAO.

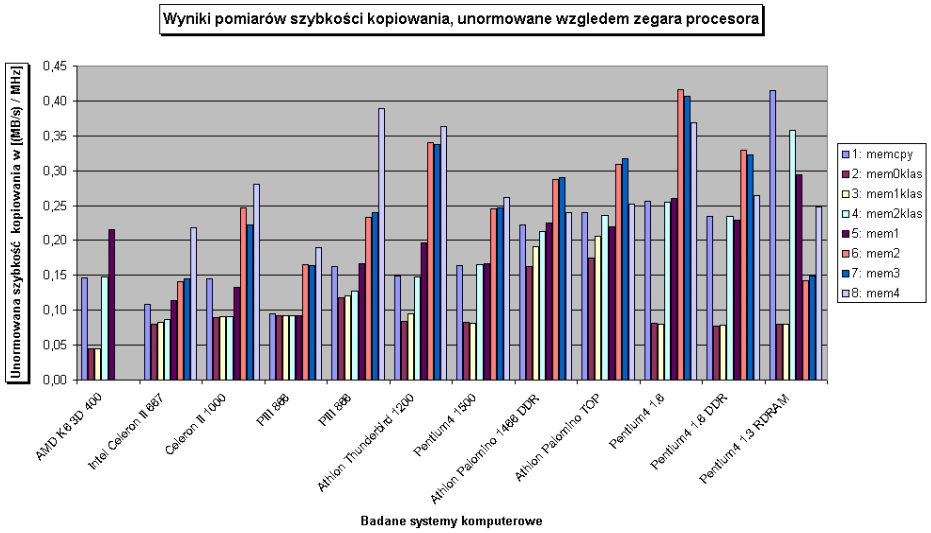
Graficzną prezentację wyników badania szybkości kopiowania bloków PAO dla różnych metod przedstawia rys.4.



Rys. 4. Wyniki badania szybkości kopiowania dla różnych metod

Najlepsze wyniki osiągają tu metody **mem2** i **mem3**, korzystające z instrukcji SSE **movntq**, zapobiegającej „zanieczyszczaniu" pamięci cache przy chybieniu przy zapisie. Warte zauważenia wydają się relacje między wynikami procedury bibliotecznej **memcpy** a wynikami zaawansowanych metod, stosujących instrukcje MMX i SSE. Można tu jak widać, osiągnąć od 30 do nawet 60 procent większą szybkość kopiowania w stosunku do metody wzorcowej, za jaką przyjęto procedurę biblioteczną memcpy. Metody klasyczne, wykorzystujące rejestr AL zgodnie z oczekiwaniami wykazują się najmniejszą szybkością kopiowania, ich stosowanie we własnych aplikacjach nie jest w związku z tym zalecane. Ponadto warto zauważyć, że dla przyjętej wielkości bloków PAO dla systemów nowoczesnych, wyposażonych w pamięci DDR najszybsze okazały się, zgodnie z przewidywaniem autora, metody **mem2** i **mem3**, natomiast dla systemów starszych, wyposażonych w pamięci SDRAM, najszybsza była, uznana za najszybszą w dostępnych publikacjach, metoda **mem4**. Ze względu na niedostępność dla autora w okresie tworzenia publikacji systemów komputerowych, wyposażonych w pamięci typu RDRAM (poza jednym modelem, który badanie dawało jednak wyniki o znacznym odchyleniu, a więc mało wiarygodne), wyciąganie wniosku co do wyższości metody **memcpy** nad pozostałymi dla tego typu pamięci nie wydaje się uzasadnione.

Ponieważ w badaniach zostały użyte systemy komputerowe z procesorami o różnych zegarach, interesujące wydaje się być porównanie szybkości kopiowania bloków PAO, znormalizowane względem częstotliwości zegara (wyniki, uzyskane z pomiarów podzielono przez częstotliwość taktowania procesora). Uzyskane rezultaty przedstawia rys.5.



Rys. 5. Wyniki pomiarów szybkości kopiowania, unormowane względem zegara procesora

Z przedstawionego rysunku wynika, że za wyjątkiem komputerów, zbudowanych w oparciu o procesor AMD Athlon z jądrem Palomino, pozostałe komputery dla metod **mem1klas** i **mem2klas** wykazują bardzo podobna wydajność. Można stąd wysnuć wniosek, że nawet w najnowocześniejszych procesorach Pentium 4 podsystem współpracy z pamięcią operacyjną pozostaje w znacznej części bez zmian w stosunku do Celerona II i Pentium III. Poważny wzrost wydajności dla wymienionych wcześniej metod w przypadku procesorów Athlon z jądrem Palomino świadczy o nowoczesności bloku współpracy z pamięcią operacyjną i jego dużym potencjale. Nie zmienia to jednak faktu, iż tak unormowana szybkość kopiowania przyjmuje wartości największe dla **mem2** i komputera z Pentium 4 i pamięciami DDR, natomiast bardzo niewiele ustępuje mu zaawansowana metoda **mem4** dla „leciwego" Pentium III z pamięciami SDRAM i biblioteczna procedura memcpy dla procesora Pentium 4 z pamięciami RDRAM. Duże różnice w szybkości kopiowania dla pozornie bliźniaczych systemów komputerowych, wyposażonych w procesory Intel Pentium 4 1,6 GHz i pamięci DDR można wyjaśnić wyższością podsystemu zarządzania pamięcią, zastosowanego w chipsecie Intel I845D na płycie głównej firmy Intel w stosunku do konkurującego chipsetu VIA P4X266, występującego na płycie głównej firmy MSI.

5. Podsumowanie

W artykule przedstawiono wyniki badania szybkości kopiowania bloków PAO o wielkości od 1 MB do 256 MB przy zastosowaniu 8 wybranych metod. Badania przeprowadzono dla 12 systemów komputerowych, pracujących pod kontrolą systemów operacyjnych rodziny Microsoft Windows. Na podstawie uzyskanych wyników potwierdzono, że uznawana w literaturze za najszybszą metoda mem4, wykorzystująca zaawansowane instrukcje SSE badanych procesorów, jest rzeczywiście najszybsza dla systemów komputerowych, wykorzystujących pamięci typu SDRAM. Jednak dla systemów komputerowych, wyposażonych w najnowocześniejsze procesory i pamięci typu DDR, szybsze okazały się inne badane metody, mem2 (dla Intel Pentium 4) i mem3 (dla AMD Athlon Palomino). Największy wzrost szybkości kopiowania w stosunku do przyjętej za metodę odniesienia memcpy uzyskano dzięki wykorzystaniu instrukcji movntq, modyfikującej działanie sterownika pamięci cache, zwykle stosującego przy chybieniu podczas zapisu strategię „write-allocate". Dla systemów komputerowych, wyposażonych w procesory Pentium 4 kopiowanie wykonuje się około 60%, a dla systemów komputerowych, wyposażonych w procesory Athlon około 30 % szybciej, niż dla metody memcpy.

Literatura

[1] Abel James, *Applications Tuning for Streaming SIMD Extensions,* Intel Technology Journal,Q2, 1999.

[2] AMD Code Samples for Microsoft Visual Studio, http://www.amd.com/us-en/Processors/DevelopWithAMD/0,,30_2252_2272_2274,00.html.

[3] Intel, *Intel Pentium 4 processor optimization guide*, Intel 2002.

[4] SGI, *Optimizing CPU to Memory Accesses on the SGI Visual Workstations 320 and 540*, http://www.sgi.com/developers/technology/irix/resources/asc_cpu.html.

[5] Wall Mike, *Optimizing Memory Bandwidth*, http://cdrom.amd.com/devconn/events/gdc_2002_amd.pdf.

[6] Wolf J., *Programming Methods for the Pentium® III Processor's Streaming SIMD Extensions using the VTune™ Performance Enhancement Environment*, Intel Technology Journal, Q2, 1999.

Recenzent: prof. dr hab. inż. Roman Kulesza

Praca wpłynęła do redakcji 5.10.2002