Doświadczalne określenie długości bloku pamięci podręcznej

Bartosz Bednarczyk

22 stycznia 2016

Krótki wstęp teoretyczny

Pamięć podręczna procesora to pamięć statyczna o krótkim czasie dostępu. Zlokalizowana jest często bezpośrednio w jądrze procesora. Ma ona wielopoziomową budowę, dzięki czemu zapewnia złudzenie posiadania szybciej i pojemnej pamięci głównej. Zmniejsza również średni czas dostępu do pamięci głównej. Nas interesować będzie pamięć tzw. pierwszego poziomu, którą dalej będę nazywał w skrócie pamięcią $L1$. Charakteryzuje się ona najszybszym dostępem wśród innych poziomów kaszu.

Ciekawe fakty o pamięci $L1$

Spośród wielu informacji o kaszu uznałem, że interesujące są następujące fakty:

- L1 jest zawsze zintegrowana z rdzeniem procesora, a więc taktowana tą samą częstotliwością, działająca z tą samą szybkością;
- L1 ma decydujący wpływ na wydajność systemu;
- Dzięki pamięci $L1$ znacznie rzadziej musi się odwoływać do wolnej pamięci RAM.

Jak sprawdzić informacje na temat pamięci na swoim komputerze?

Nic prostszego. Wystarczy wpisać polecenie **sysctl hw**. Część wydruku z mojego komputera można zaobserwować poniżej. Szczególną uwagę należy zwrócić na fragment zaznaczony niebieskim kolorem.

```
hw.ncpu: 4
hw.byteorder: 1234
hw.memsize: 4294967296
hw.activecpu: 4
hw.targettype:
hw.physicalcpu: 2
hw.physicalcpu_max: 2
hw.logicalcpu: 4
hw.logicalcpu_max: 4
hw.cputype: 7
hw.cpusubtype: 8
hw.cpu64bit_capable: 1
hw.cpufamily: 280134364
hw.cacheconfig: 4 2 2 4 0 0 0 0 0 0
hw.cachesize: 4294967296 32768 262144 3145728 0 0 0 0 0 0
hw.pagesize: 4096
hw.pagesize32: 4096
hw.busfrequency: 100000000
hw.busfrequency_min: 100000000
hw.busfrequency_max: 100000000
hw.cpufrequency: 1400000000
hw.cpufrequency_min: 1400000000
hw.cpufrequency_max: 1400000000
hw.cachelinesize: 64
hw.l1icachesize: 32768
hw.l1dcachesize: 32768
hw.l2cachesize: 262144
hw.l3cachesize: 3145728
hw.tbfrequency: 1000000000
```

Zrób to sam!

Ciekawym pomysłem jest to, by samemu pobawić się w mierzenie długości bloku kaszu. Napisałem w tym celu program komputerowy (w języku C), który przez wielokrotne testy oraz ich uśrednianie, sugerował nam żądany rozmiar. Przedstawię najpierw z jakich narzędzi będę korzystał w kodzie.

- **Wędrowanie po tablicy (array walker)**. Tworzymy jednowymiarową tablicę 32-bitowych słów o N elementach, nazywamy ją „array" . Tablica zaczyna się pod adresami podzielnymi przez rozmiar strony i ma rozmiar wielokrotności rozmiaru strony. Mamy zbiór S wszystkich indeksów tej tablicy. Chcemy wygenerować pewne szczególne permutacje zbioru S. Po zakodowaniu permutacji, uruchamiamy procedurę array_walker. Przechodzi ona po kolejnych elementach tablicy i generuje odczyty pamięci (i potencjalne chybienia). Procedura kończy się po osiągnięciu ostatniego elementu ciągu lub po przejrzeniu N elementów, gdy permutacja była cyklem.
- **Zatruwanie pamięci podręcznej**. W niektórych przypadkach należy zapełnić pamięć podręczną pewnymi danymi w taki sposób, aby każdy kolejny odczyt innych danych generował chybienie. Dzięki temu, w pewnym sensie „opróżniamy" pamięć podręczną. Określa się to mianem „zatruwania pamięci podręcznej". Zadanie to realizuje procedura „poison_cache".
- **Pobieranie czasu (timer_*)**. Chcemy mierzyć czas z dokładnością do milisekund. Służy do tego zestaw procedur timer_init, timer_start, timer_stop, timer_print.

Algorytm programu wygląda następująco:

1. Utwórz dostatecznie dużą tablicę (rozmiar musi być potęgą dwójki).
2. Zatruj kasz.
3. Następnie dla $n \in \{1, 2, 3, \ldots, N^1\}$ powtarzaj:
   (a) Zatruj kasz.
   (b) Zacznij odmierzać czas.
   (c) Wielokrotnie wędruj po elementach tablicy, zapełniając je przykładowymi wartościami (np. jedynką), „skacząc" co n elementów.
   (d) Zatrzymaj czas i podaj ile czasu upłynęło.
4. Zakończ program.

Kiedy wykonamy program wielokrotnie i uśrednimy otrzymane wyniki, to możemy spróbować wywnioskować z nich rozmiar bloku pamięci podręcznej. Wynik moich eksperymentów poniżej.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0.90	0.884	0.882	0.87	0.93	1.55	1.81	2.03	2.31	2.56	2,79	3.01	3.24	3.53	3.79	**4,16**	4,17
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
4,10	4,14	4,21	4,17	4,14	4,18	4,16	4,13	4,19	4,14	4,15	4,23	4,14	4,19	4,30	4,21	4,20

Z przedstawionej tabeli można wyczytać, że początkowo czas przechodzenia po tablicy był stosunkowo niewielki. Rośnie on wraz ze wzrostem N. W pewnym momencie czas się stabilizuje i wyniki obliczeń są w miarę podobne. Od czego to zależy?

[1] Liczba testów kontrolnych - u mnie jest to 37.

Okazuje się, że czas przechodzenia po tablicy ma ścisły związek z rozmiarem kaszu. Od podanego N zależy, czy wszystkie elementy, po których przechodzimy, mieszczą się w pamięci podręcznej czy nie. Jeżeli nie, to program musi doczytywać elementy na bieżąco i wyrzucać stare dane z pamięci, które za chwilę znowu będą zmieniane. Przy N mniejszych od rozmiaru długości bloku kaszu, taka sytuacja nie występuje.

Możemy więc wywnioskować, że idealnym kandydatem na „zbilansowanie" wyników byłoby $N = 16$, gdyż dla $N > 16$ wyniki były w miarę podobnie, a w poprzednich iteracjach czasy gwałtownie rosły. Obliczając formułę, $16 * sizeof(int) * sizeof(int)$ dostajemy liczbę 64, co zgadza się z wyświetlonym komunikatem systemowym z pierwszej strony sprawozdania (patrz: hw.cachelinesize).

Udało nam się zatem policzyć długość bloku kaszu bez używania specjalistycznych narzędzi.

Dodatek. Kod źródłowy funkcji „measure"

Do sprawozdania załączam również kod źródłowy funkcji „measure" napisany w języku C, który realizuje algorytm opisany na poprzedniej stronie.

```
void measure() {
  struct timeval tval_before, tval_after, tval_result;
  array_init(1<<16); poison_init((1<<16)/getpagesize());

  for(int n = 1; n < 37; n++) {
    printf("%d", n); poison_cache();
    gettimeofday(&tval_before, NULL);

    for(int j = 1; j < 1000000; j++) {
      int counter = 0, i = 0;
      while(counter < 1700) {
        array.data[i] = 1; i+= n; counter++;
      }
    }

    gettimeofday(&tval_after, NULL);
    timersub(&tval_after, &tval_before, &tval_result);
    printf("␣%ld.%06ld\n",
      (long int)tval_result.tv_sec, (long int)tval_result.tv_usec);
}}
```