Sławomir Zborowski

Mierzenie wydajności kodu C++

Fundamentalnym wymaganiem stawianym przed każdą aplikacją, abstrahując od użytych języków programowania i technologii, jest jej poprawne działanie – zgodnie z ustaloną specyfikacją. Jednym z kolejnych kryteriów, które powinny być brane pod uwagę przez twórców aplikacji, jest jej wydajność. Odwrotna kolejność (nazywana również przedwczesną optymalizacją) może doprowadzić zarówno do wydajnych, jak i niewydajnych programów, niekoniecznie zachowujących się poprawnie – coś, z czego spłodzenia żaden inżynier oprogramowania nie byłby zapewne zadowolony.

Nierzadko jednak szybkość działania aplikacji, lub któregoś z jej podsystemów, jest jednym z wymogów jawnie wpisanym w specyfikację produktu. W ekstremalnych przypadkach niewystarczająco szybkie działanie może być powodem, dla którego użytkownicy rezygnują z jej korzystania. Dobrym przykładem z polskiego podwórka był portal nasza-klasa.pl, który w swoich początkowych latach zniechęcił część użytkowników swoim powolnym, łagodnie rzecz ujmując, działaniem.

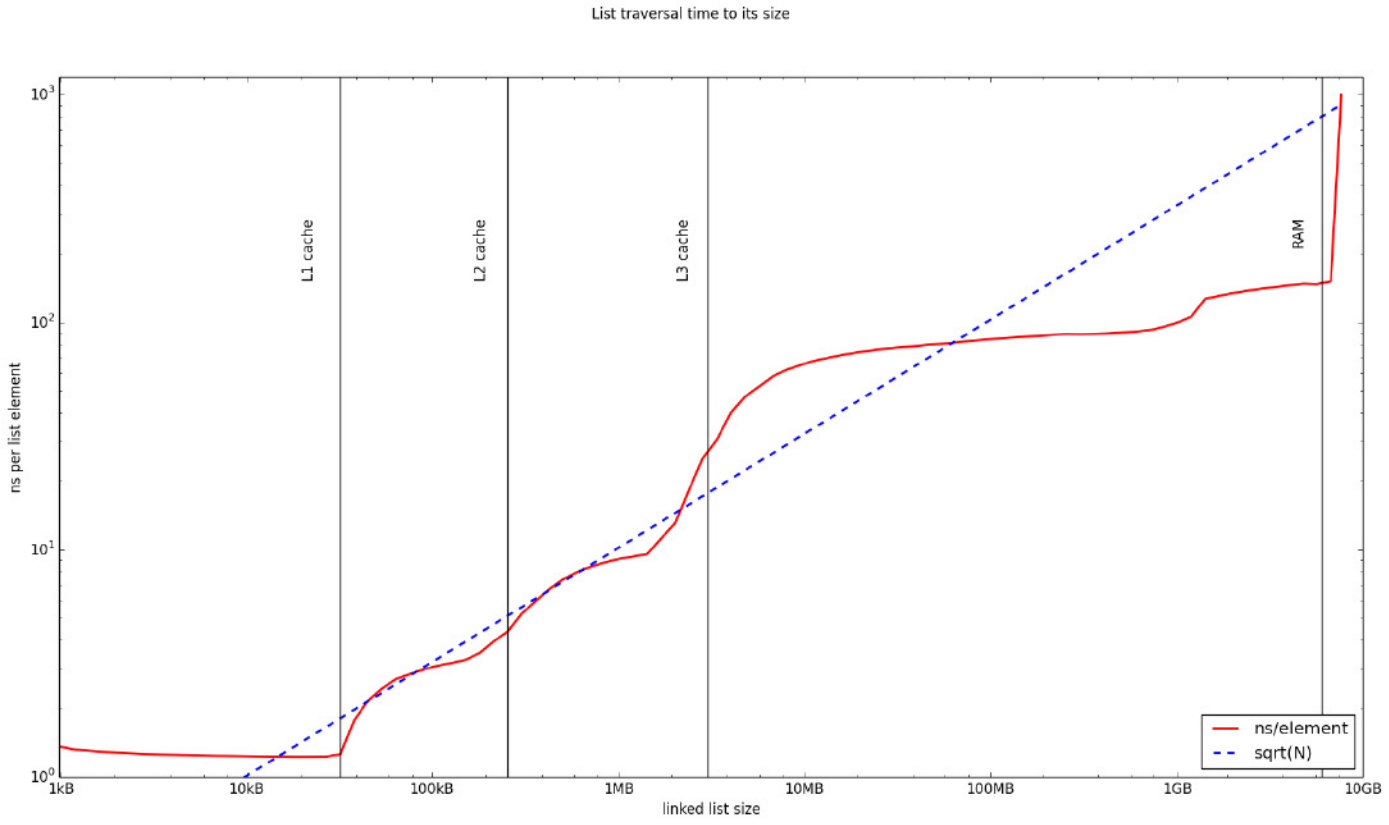
Poprawność działania programu lub jego fragmentu sprawdzić można, tworząc stowarzyszone z kodem źródłowym testy – np. jednostkowe. Zadanie to znacznie ułatwiają liczne frameworki do testowania takie jak m.in. GoogleTest, Boost.Test itp. Jak jednak przedstawia się sytuacja w przypadku badania wydajności aplikacji? Celem artykułu jest odpowiedzenie czytelnikowi na te i inne pytania.

BIG-O VS RZECZYWISTOŚĆ

Tradycyjną metodą określania złożoności obliczeniowej i pamięciowej algorytmów, która bezpośrednio przekłada się na wydajność, jest jej teoretyczne wyznaczanie, korzystając z notacji dużego O. Takie podejście sprawdza się jednak tylko do algorytmów, które przeważnie są jedynie częścią badanej aplikacji. Sytuacja może się jeszcze bardziej skomplikować – np. w momencie, gdy w grę wchodzą zewnętrzne biblioteki, do których źródeł nie ma dostępu.

Dodatkowo każdy program uruchomiony jest w pewnym środowisku, które może mieć znaczący wpływ na funkcjonowanie aplikacji. W przypadku oprogramowania może to być np. system operacyjny wraz ze swoimi optymalizacjami dostępu do zasobów. Nie bez znaczenia jest również sprzęt – poczynając od pamięci cache i systemów przewidywania skoków wbudowanych w procesor, a kończąc na buforach znajdujących się wewnątrz dysków twardych.

Dobrym przykładem ilustrującym poziom skomplikowania dzisiejszych systemów i niemożność ocenienia ad hoc wydajności aplikacji jest przechodzenie po jednokierunkowej liście. Jak można się spodziewać, teoretyczna złożoność takiej operacji to O(N). Po przeprowadzeniu prostego doświadczenia[0] okazuje się jednak, że lepszym przybliżeniem byłoby O(√N). Przedstawia to Rysunek 1 ilustrujący średni czas dostępu do elementów w zwiększającej się liście. Zaznaczone zostały na nim rozmiary pamięci cache i RAM właściwe dla sprzętu, na którym przeprowadzone zostały testy.



Rysunek 1. Wykres przedstawiający średni czas dostępu do elementów zwiększającej się listy jednokierunkowej. Na wykresie pionowymi liniami oznaczono rozmiary pamięci L1, L2, L3 cache procesora oraz dostępnej pamięci RAM