Optymalizacja programów Open-Source

Profilery niskiego poziomu
część 1

Krzysztof Lichota
lichota@mimuw.edu.pl

Profiling niskiego poziomu

Profiling niskiego poziomu

- Profiling wysokiego poziomu pozwala wykryć, które funkcje zabierają dużo czasu
- Ale co dalej?

Sposoby profilowania niskiego poziomu

- Analiza kodu źródłowego (!)
  - Skupienie się od razu na kodzie maszynowym może spowodować przeoczenie możliwych optymalizacji w algorytmie
- Profiling per linia kodu źródłowego/funkcja/adres – za pomocą poznanych już narzędzi
  - Narzędzia profilujące za pomocą próbkowania mają dość małą dokładność
- Narzędzia do analizy niskopoziomowej na poziomie procesora

Valgrind

Valgrind

- Narzędzie do dynamicznej translacji i instrumentacji kodu maszynowego na różne sposoby
- Możliwe różne instrumentacje:
  - memcheck – sprawdza użycie niezainicjalizowanych zmiennych, wycieki pamięci, błędy alokacji pamięci
  - cachegrind/callgrind – analizuje zużycie cache CPU i drzewo wywołań
  - massif – analiza zużycia pamięci
  - helgrind – błędy w programach współbieżnych

Valgrind

- Bardzo duży narzut – program działa 20x do 100x wolniej (zależnie od instrumentacji)
- Ale pozwala na wykrywanie bardzo trudnych błędów i bardzo skomplikowaną analizę
- Niektóre programy działają z problemami pod Valgrindem

cachegrind/callgrind

cachegrind

- Symuluje użycie cache CPU poziomu 1 i 2
- **Nie** symuluje kosztu instrukcji, branch prediction, itp.
- Przydatny do optymalizacji kodu, który intensywnie korzysta z pamięci

cachegrind - wywołanie

- Sposób wywołania: valgrind --tool=cachegrind <polecenie>

Przydatne opcje:

- Wyłączenie symulacji cache: --cache-sim=no
- Wyłączenie branch prediction: --branch-sim=no
- Opcje określające wielkość i rodzaje cache

callgrind

- Sposób wywołania: valgrind --tool=callgrind --dump-instr=yes --trace-jump=yes --simulate-cache=yes <polecenie>

Przydatne opcje:

- Włączenie pełnej symulacji cache, a nie tylko odczytów: --simulate-cache=yes
- Zbieranie informacji o skokach warunkowych: --collect-jumps=yes

Przydatne opcje callgrind

- Włączenie od określonego miejsca:
  - Uruchomić z opcją: --instr-atstart=no
  - W odpowiednim momencie włączyć za pomocą: callgrind_control -i on (w tym samym katalogu, co uruchomiony program)
- Profilowanie tylko określonej funkcji: --toggle-collect=<funkcja>

Wynik działania callgrind

- Surowy wynik działania callgrind nie daje się czytać
- Istnieje narzędzie do przetworzenia wyników na postać tekstową: callgrind_annotate
- Ale istnieje też dużo lepsze narzędzie graficzne: kcachegrind :)

kcachegrind

kcachegrind/callgrind

- Narzędzie do wizualizacji wyników callgrind/cachegrind (ale można też wizualizować wyniki innych profilerów za pomocą skryptów konwertujących)
- Pokazuje bardzo szczegółowo co się dzieje instrukcja po instrukcji (np. pojedyncze wywołania dynamicznego linkera)
- Pokazuje ścieżki wykonania w kodzie i która gałąź była wybrana ile razy
- Pokazuje drzewo wywołań (podobnie jak Google Profiler)

kcachegrind/callgrind

- Pokazuje obrazowo drzewo wywołań z obszarem proporcjonalnym do kosztu (treemap)
- Może pokazywać oszacowanie liczby cykli procesora, liczbę nietrafień w cache a nawet własną funkcję kosztu

Jak optymalizować

Jak optymalizować?

- Zamienić funkcję na inline
- Wymusić użycie rejestrów przez kompilator
- Zmienić konstrukcje pętli na efektywniejszą (np. nie wyliczać indeksu jako mnożenia tylko dodawanie)
- Zamienić mnożenia na przesunięcia bitowe
- Operować na większych blokach danych naraz (w szczególności operacje wektorowe: MMX, SSE oraz operacje na ciągach: SCAS, STOS)
- Użyć specyficznych instrukcji procesora (np. XLAT)
- Czasem trzeba zrobić kilka wariantów funkcji na różne procesory (np. RAID5 w Linuksie) i sprawdzać, która jest najlepsza na żywo.

Jak optymalizować? (2)

- Optymalizować użycie cache (procesor jest kilkakrotnie szybszy od pamięci!)
  - Wyrównać struktury danych w pamięci
  - Robić ręczny prefetching
  - Unikać cache pollution
- Optymalizować skoki (branch prediction)
- ...
- Ogólnie: trzeba bardzo dobrze znać budowę kompilatorów, architekturę komputera, używanego procesora, itd.

Bibliografia

- http://valgrind.org/
- http://rac.uits.iu.edu/rats/research/docs/valgrind/cg_techo
- http://kcachegrind.sourceforge.net/
- http://kcachegrind.sourceforge.net/docs/slides-akademy.