Architektury systemów komputerowych

Pracownia 3: „Optymalizacja kodu”

31 maja 2020

Wprowadzenie

Celem tej pracowni jest zdobycie praktycznej wiedzy na temat optymalizacji programów pod kątem użycia pamięci podręcznych i wykorzystania właściwości mikroarchitektury procesora, tj. superskalarnego przetwarzania instrukcji i predykcji skoków warunkowych. Oddawanie rozwiązań będzie zrealizowane przy użyciu systemu GitHub Classroom. Rozwiązanie każdego z zadań składa się z kilku etapów, które opisano poniżej.

Najpierw należy uzupełnić procedury w wyznaczonym pliku źródłowym. Rozwiązanie musi spełniać kryteria podane w pliku «`GNUmakefile`», np. maksymalny procent liczby chybień w pamięć podręczną lub liczby źle przewidzianych skoków. Rozwiązanie będzie testowane automatycznie w symulowanym środowisku przy pomocy narzędzia profilującego `callgrind`[1] będącego częścią programu programu `valgrind`[2]. Przeprowadzenie symulacji jest możliwe na komputerze studenta – wystarczy wykonać polecenie «`make sim`».

Następnie należy wyznaczyć parametry systemu, na którym student rozwiązuje zadanie, co najmniej nazwę kodową mikroarchitektury (np. Skylake) i konfigurację podsystemu pamięci podręcznej. Część z tych parametrów można odczytać przy pomocy narzędzia «`x86info`». Możliwe, że przyda się również wiedza na temat opóźnień przetwarzania instrukcji lub koszt źle przewidzianego skoku. Szczegółowe informacje na temat danej realizacji mikroarchitektury procesora można znaleźć w dokumencie: „The microarchitecture of Intel, AMD and VIA CPUs: An optimization guide for assembly programmers and compiler makers”[3].

Po zebraniu informacji na temat systemu należy przeprowadzić szereg eksperymentów. Polegają one na uruchomieniu optymalizowanego program na swoim komputerze i zebraniu wyników z liczników sprzętowych[4] wbudowanych w procesor. Należy zauważyć, że monitor maszyn wirtualnych (np. `VirtualBox`) nie daje dostępu do tychże liczników. Innymi słowy, eksperymenty trzeba wykonywać pod kontrolą systemu `Linux` zainstalowanego na fizycznej maszynie. Program testowy, skonsolidowany z biblioteką `PAPI`[5], przyjmuje opcję «`-p`», przy pomocy której można wybrać do odczytu określony zestaw liczników sprzętowych. Następnie należy próbować obniżyć czas uruchomienia programu poprzez (w nawiasie podano jak odczytać liczniki):

- zmniejszenie liczby chybień w pamięć podręczną i `TLB` (`-p memory`),
- zmniejszenie liczby źle przewidzianych skoków (`-p branch`),
- zwiększenie liczby instrukcji przetworzonych w ciągu jednego cyklu zegarowego procesora (`-p ipc`).

UWAGA! Pamiętaj, że właściwy sposób mierzenia czasu wykonania programu polega na wielokrotnym jego uruchomieniu, odrzuceniu skrajnych pomiarów i uśrednieniu reszty wyników. Należy zadbać o niską wariancję wyników pomiaru czasu wykonywania programu na procesorze. Najłatwiej osiągnąć to minimalizując obciążenie systemu, tj. ograniczając liczbę procesów.

Po zakończeniu eksperymentów należy napisać sprawozdanie z wykonania zadania. Do każdego zadania dołączono listę pytań, na które należy odpowiedzieć. Przedstawiona w raporcie teza musi być solidnie podparta danymi zebranymi z eksperymentów. Raporty, które będą zawierać mętne wyjaśnienia niepodparte danymi i wiedzą na temat architektury systemów komputerowych, nie zostaną dobrze ocenione. Nie ulegaj pokusie naginania wyników – oceniający na pewno to wychwyci. Ciężar skonstruowania przekonującej argumentacji spoczywa na Was!

[1] `https://valgrind.org/docs/manual/cl-manual.html`
[2] `https://valgrind.org/docs/manual`
[3] `https://www.agner.org/optimize/microarchitecture.pdf`
[4] `https://en.wikipedia.org/wiki/Hardware_performance_counter`
[5] `https://icl.utk.edu/papi/`

Rozwiązania

Rozwiązania zadań i raporty należy umieścić przed upływem terminu w prywatnym repozytorium utworzonym przy pomocy serwisu GitHub Classroom. Można modyfikować wyłącznie te pliki, które zostały wskazane w treści zadania. Należy zadbać o to by programy kompilowały się bez ostrzeżeń pod systemem Debian 10 i Ubuntu 18.04 dla architektury x86-64.

Raport należy zawrzeć w pliku «raport.md» w formacie markdown. Dla wygody sprawdzającego w pliku «raport.url» można zawrzeć odnośnik do dokumentu w serwisie hackmd.io[6]. Rezultaty uruchomień programu zbierz do pliku tekstowego i utwórz z nich wykres. Tworzenie wykresów z danych numerycznych przy pomocy narzędzia gnuplot[7] przystępnie wyjaśniono na stronie gnuplot: not so Frequently Asked Questions[8]. W repozytorium można umieszczać wyłącznie pliki tekstowe.

Rozwiązania będą zbierane i oceniane przy pomocy systemu GitHub Classroom. Dla każdego zadania prowadzący dostarczy odnośnik do wygenerowania prywatnej wersji repozytorium. Rozwiązanie należy umieścić w gałęzi «master». W gałęzi «feedback» musi się znajdować **niezmodyfikowana** wersja gałęzi «master» głównego repozytorium zadania np. opt-matmult[9].

Autorzy zadań dokładają wszelkich starań, by były one dobrze przetestowane. Niestety nie są w stanie z góry przewidzieć wszystkich różnic między szerokim spektrum maszyn, na których zadanie będzie rozwiązywane. W związku z tym czasami będzie zachodziła potrzeba zsynchronizowania repozytorium rozwiązania studenta z repozytorium głównym zadania. W katalogu swojego repozytorium należy wykonać poniższy szereg poleceń, będąc przygotowanym na rozwiązywanie konfliktów:

```
$ git checkout feedback
$ git pull https://github.com/ii-ask/NAZWA-ZADANIA.git
  «tu należy rozwiązać potencjalne konflikty»
$ git push
$ git checkout master
$ git merge feedback
```

Prowadzący będzie oceniał zmiany plików w prośbie o połączenie (ang. *pull request*) o nazwie «Feedback». Inne prośby będą przez sprawdzającego ignorowane. W zakładce „Files changed” mają być widać **wyłącznie** zmiany, które są częścią rozwiązania i zostały przygotowane przez studenta. Rozwiązania z nadmiarowymi plikami lub zbędnym kodem będą kierowane do poprawki.

[6] https://hackmd.io
[7] http://www.gnuplot.info/
[8] http://lowrank.net/gnuplot/datafile2-e.html
[9] https://github.com/ii-ask/opt-matmult

Zadania

Punktacja: W nawiasach przy zadaniu podano dwie liczby, które należy rozumieć następująco: pierwsza to punkty do zdobycia za poprawne zakończenie symulacji w automatycznej ocenie zadania, druga to punkty do uzyskania za sprawozdanie.

Zadanie 1 (1+2). Na slajdach do wykładu pt. „Cache Memories" zaprezentowano różne podejścia do implementacji mnożenia dwóch macierzy. Na slajdzie 47[10] podano trzy rozwiązania o różnej kolejności przeglądania elementów tablicy. Na slajdzie 53 widnieje rozwiązanie wykorzystujące technikę kafelkowania. Należy uzupełnić ciało procedur «`matmult0`» ... «`matmult3`» w pliku źródłowym «`matmult.c`». Powtórz eksperyment ze slajdu 48 dla rosnących wartości n rozmiaru boku macierzy.

W pliku źródłowym «`matmult.h`» zdefiniowano wartości «`A_OFFSET`», «`B_OFFSET`», «`C_OFFSET`». Dobrane wartości wymuszają, aby macierze nie zaczynały się pod takimi samymi adresami wirtualnymi modulo rozmiar strony. Jeśli po ustawieniu definicji tych wartości na 0 obserwujesz spadek wydajności w kafelkowanej wersji mnożenia macierzy postaraj się wyjaśnić ten fenomen.

Wskazówka: Obserwowany efekt najprawdopodobniej wynika z generowania konfliktów w obrębie zbiorów.

Sprawozdanie: Czy uzyskane wyniki różnią się od tych uzyskanych na slajdzie? Z czego wynika rozbieżność między wynikami dla poszczególnych wersji mnożenia macierzy? Jaki wpływ ma rozmiar kafelka na wydajność «`matmult3`»? Dla jakich wartości n obserwujesz znaczny spadek wydajności? Czy rozmiar kafelka ma znaczenie? Czy inny wybór wartości domyślnych «`*_OFFSET`» daje poprawę wydajności?

Zadanie 2 (1+1). Poniżej podano funkcję transponującą macierz kwadratową o rozmiarze n. Niestety jej kod charakteryzuje się niską lokalnością przestrzenną dla tablicy «`dst`». Używając metody kafelkowania zoptymalizuj poniższą funkcję pod kątem lepszego wykorzystania pamięci podręcznej.

```
1 void transpose0(int *dst, int *src, int n) {
2   for (int i = 0; i < n; i++)
3     for (int j = 0; j < n; j++)
4       dst[j * n + i] = src[i * n + j];
5 }
```

Należy uzupełnić ciało procedury «`transpose1`» w pliku źródłowym «`transpose.c`».

Sprawozdanie: Jaki wpływ na wydajność «`transpose1`» ma rozmiar kafelka? Czy czas wykonania programu z różnymi rozmiarami macierzy identyfikuje rozmiary poszczególnych poziomów pamięci podręcznej?

[10] Chodzi o numer w prawym dolnym rogu slajdu.