VisuAlgo Analiza

Prakični seminarski rad u okviru kursa
Verifikacija softvera
Matematički fakultet

Nikola Subotić, 1029/2023
mi231029@alas.matf.bg.ac.rs

7. januar 2024.

Sažetak

U ovom izveštaju će biti pružen detaljan pregled alata koji su korišćeni za analizu projekta VisuAlgo, zajedno sa rezultatima dobijenim njihovim izvršavanjem. Pored toga, istaći ćemo potencijalne izazove i predložiti određene izmene u izvornom projektu gde god je to moguće. Konačno, pružićemo opšti dojam o projektu i izložiti zaključke.

Sadržaj

1 Clang-tidy

Clang-tidy je moćan alat za C++ programiranje, baziran na Clang kompajleru, čiji je osnovni zadatak detektovanje i ispravljanje uobičajenih grešaka u kodiranju. To uključuje kršenje stila, netačno korišćenje interfejsa i otklanjanje bagova kroz statičku analizu. On funkcioniše kao modularni i proširiv okvir, pružajući programerima mogućnost pisanja prilagođenih provera.

Alat clan-tidy je pokrenut nad *.cpp* i *.h* fajlovima u korenom direktorijumu projekta. Za pokretanje alata korišćena je pomoćna skripta, tj. alat *run-clang-tidy*, koji pokreće clang-tidy nad svim fajlovima koji su navedeni u bazi prevodjenja (eng. *compilation database*).

Alat je uspeo da pronađe više od pet nedoslednosti u kodu. Konkretno u klasi *Node.cpp* u linijama 8 i 9, naglasio nam je da promenljiva koja je deklarisana sa ključnom rečju extern je inicijaliozvana, što ne bi prebalo. Takođe još jedna od nedoslednosti koja se javlja u analizi jeste pogrešan redosled navođenja polja u listi inicijalizacije, gde npr. u klasi *Walker.cpp* u liniji 12, polje 'heuristic' će biti inicijalizovano posle polja 'gridMatrix', što znaci da redosled inicijalizacije se ne poklapa sa redosledom deklaracije u konsturktoru. Još jedan minimalan propust je postojanje nekorišćenje promenljive 'value' u klasi *Widget.cpp*. Da bi prevazišli prvu nedoslednost, dovoljno je da unutar *Node.hpp* fajla samo deklarišemo promenljive, a unutar *Node.cpp* ih definišemo. Problem ne poklapanja redosleda inicijalizacije i deklaracije, možemo rešiti tako što uskladimo redosled. Poslednje, ali ne i ne preterano bitno, možemo samo ukloniti promenljivu value jer se ne koristi.

2 Memcheck

Valgrind pruža niz alata za ispravke i analizu performansi koji pomažu pri ubrzanju i poboljšanju tačnosti programa. Najpoznatiji među tim alatima je Memcheck. Alat Memcheck može otkriti greške povezane s memorijom koje su česte u C i C++ programima, a koje mogu uzrokovati prestanak rada programa i nepredvidivo ponašanje.

U ovoj analizi alat Memcheck je pokrenut četiri puta i izlazi su sačuvani, a zatim na osnovu izlaza su generisani finalni izveštaji. Naime, ispotavlja se da je broj memorijskih blokova koji su izgubljeni direktno relativno mali, u proseku svega 4000 bajtova, dok u proseku broj indirektno izubljenih bajtova je veći, negde oko 16000. Površnom analizom, zaključeno je da problem direktnog gubitka blokova nije vezan za korisnički kod, već se problem dešava negde u sistemskim biblitoekama.

3 Perf i FlameGraph

Perf je alat za profiliranje namenjen Linux sistemima verzije 2.6 i novijim, koji apstrahuje razlike u hardveru CPU-a prilikom merenja performansi na Linuxu i pruža jednostavno komandno korisničko okruženje. Alat perf nudi bogat skup komandi za prikupljanje i analizu podataka o performansama i tragovima. Plamen grafikon (eng. *FlameGraph*) je vizualizacija hijerarhijskih podataka, kreirana da prikaže stek tragove profilisanog softvera kako bi se najčešći putevi koda brzo i precizno identifikovali.

Korisćenje perf rada u ovoj analizi, bilo je vezano za izvršavanje dva algortima i to: A* algoritma i DFS algoritma. Izlaz iz perfa je prosleđen

Plamen grafikonu koji je za nas kreirao slikovitiji prikaz u vidu hijeharhije. Na grafikonu se može primeniti da je vremene izvršavanja algoritma A* znatno brže nego izvršavanje algortima DFS, na konkretnom primeru i to za različite verzije lavirinta. Ovaj postupak možemo ponoviti i za druge algoritme koji su predstavljeni, različite grafove i lavirtine, kako bi smo utvrdili kako se implementirani algoritmi ponašaju i nad kojim primerima su efikasniji, a nad kojim imaju problem da pronađu optimalno rešenje.

4 Flawfinder

Flawfinder je alat autora David Wheeler (eng. *David A. Wheeler*). To je program koji pregleda izvorni kod u jezicima C/C++ i prijavljuje moguće sigurnosne propuste. Može biti koristan alat za analizu softvera u potrazi za ranjivostima i takođe može poslužiti kao jednostavan uvod u alate za statičku analizu izvornog koda.

Flawfinder je pregledao sve *.cpp* i *.h* fajlove i dao sledeće rezultate:

Naglasio je da u fajlv *GridMatrix.cpp* u liniji 270, kao i u fajlu *widget.cpp* u liniji 355 postoji opastonst prilikom otvaranja datoteke i da li napadač može preusmeriti ili time prisiliti na otvaranje drugog tipa datoteke. Ovo analizom zaključili smo da je potrebno uvesti dodatne bezbenosne provere prilikom otvaranja datoteka, ako želimo da budemo sigurni da je naša aplikacija visokog sigurnosog nivoa.

5 Cachegrind

Još jedan Valgrindov alata koji je obrađen u okviru seminarskog rada je Cachegrind. Cachegrind se ističe kao alat za profajliranje od visoke preciznosti koji pažljivo broji tačan broj instrukcija koje izvršava program. Za razliku od drugih profilera koji se oslanjaju na aproksimacije kroz tehnike uzorkovanja i ograničavaju se na merenje na nivou funkcija, Cachegrind pruža detaljne podatke na nivou fajlova, funkcija i linija koda. Iako radi sporije, ova namerna metoda osigurava prikupljanje preciznih i reproduktivnih podataka o profiliranju, dodatno poboljšanih sposobnošću spajanja i poređenja informacija iz različitih pokretanja programa.

Za pokretanje alata bilo je potrebno prevesti program u debug modu sa simbolom -g. Nakon toga pokrenut je alat Cachegrind i dodata je opcija –log-file, kako bi sačuvali izlaz iz alata Cachegrind. Tako dobijen rezultat nije čitljiv, pa zbog toga je potrebno bilo iskoristiti program cg_annotate koji nam generiše novi fajl , sa propratnim infomacijama, što pruža detaljan opis.

Metrike	**Vrednosti**
I broj instrukcija	3,031,747,109
I1 promašaji	12,062,998
LLi promašaji	178,322
I1 procenat promašaja	0.40%
LLi procenat promašaja	0.01%

Tabela 1: Metrike keša instrukcija

Statistike ukazuju na niske stope promašaja instrukcijskog keša (I1) i keša podataka (D1), što pokazuje efikasnu upotrebu keša. Osim toga, niske

Metrika	Vrednosti
D broj instrukcija	1,288,322,702
D1 promašaji	13,453,254
LLd promašaji	1,292,192
D1 procenat promašaja	1.0%
LLd procenat promašaja	0.1%

Tabela 2: Metrike keša podataka

Metrike	Vrednosti
LL broj instrukcija	25,516,252
LL promašaji	1,470,514
LL procenat promašaja	0.0%

Tabela 3: Drugi nivo keš memorije (Last Level)

stope promašaja poslednjeg nivoa keša (LL) sugerišu na efikasne obrasce pristupa memoriji, doprinoseći ukupno optimizovanoj performansi.

6 Zaključak

Uvođenje alata za statičku i dinamičku analizu, kao i profiliranje, pružilo je detaljan uvid u performanse i sigurnost projekta VisuAlgo. Clang-tidy je otkrio nekoliko stilskih i logičkih nedoslednosti koje su uspešno otklonjene, a Memcheck je identifikovao i analizirao potencijalne memorijske propuste. Korišćenje Perf i FlameGraph omogućilo je efikasno upoređivanje performansi različitih algoritama. Flawfinder je identifikovao sigurnosne ranjivosti koje zahtevaju dodatne provere prilikom otvaranja datoteka. Cachegrind je pružio dubok uvid u keš memoriju, što je doprinelo optimizaciji pristupa memoriji. Korišćenjem ovih alata stekli smo uvid u verifikaciju i analizu softvera, dok integracija ovih alata doprinosi pobolšanom kvalitetu, kao i performansama ovog projekta.