KCachegrindi käsiraamat

Dokumentatsiooni algne autor: Josef Weidendorfer
Uuendused ja parandused: FedericoŽenith
Tõlge eesti keelde: Marek Laane

Sisukord

Kokkuvõte

KCachegrind on KDE töökeskkonna tarbeks loodud profileerimisandmete kuvamise tööriist.

Peatükk 1

Sissejuhatus

KCachegrind on profileerimistööriistade tekitatud andmete brauser. Käesolevas peatükis selgitame, mida tähendab üldse profileerimine, kuidas seda tehakse ning räägime veidi olemasolevatest profileerimistööriistadest.

1.1 Profileerimine

Programmi luues soovid kindlasti lõpuks muuta selle ka võimalikult kiireks (aga samas ikka töökindlaks!). Aeg on mõistagi kallis väärtus ja optimeerimisfunktsioone kasutatakse harva. Nii on sul vaja teada saada, milline osa programmist kulutab kõige rohkem aega.

Jadakoodi puhul piisab tavaliselt programmide käituskarakteristika, näiteks funktsioonides ja koodiridades veedetud ajahulga statistiliste andmete kogumisest. Seda nimetatakse profileerimiseks. Vastav programm töötab profileerimistööriista kontrolli all, mis programmi täitmise lõppedes väljastab kokkuvõtte. Paralleelkoodi puhul on aga asjad teisiti - peamiselt tekitab jõudlusprobleeme see, kui üks protsessor ootab teiselt andmeid. Et sellist ooteaega ei ole tavaliselt võimalik väga lihtsalt muuta, on siin mõttekas genereerida ajatempliga sündmuste jäljed. Selliseid andmeid KCachegrind ei kuva.

Profileerimisandmete analüüsimisel peaks olema üsna lihtne näha koodi nii-öelda pudelikaelu ning seejärel parandada näiteks väljakutsete esitamise viisi ja optimeerida koodi erinevaid piirkondi. Pärast seda saab uue profileerimisega kontrollida, kui edukas optimeerimine oli.

1.2 Profileerimismeetodid

Koodi piirkonna (nt. funktsiooni) täitmisele kuluva aja täpseks mõõtmiseks või selle käigus toimuvate sündmuste jäädvustamiseks tuleb antud piirkonna ette ja järele lisada mõõtmiskood. See arvestab aega või sündmusi ning arvutab erinevusi. See tähendab, et enne täitmist on vaja muuta originaalkoodi. Seda nimetatakse instrumentatsiooniks. Instrumentatsiooniga võib tegelda nii programmeerija ise, kompilaator või käitussüsteem. Et huvipakkuvad piirkonnad on enamasti pesastatud, mõjutab mõõtmise teostamine otseselt ka mõõtmistulemusi. Seepärast tuleb instrumentatsiooniga olla väga hoolas ning mõõtmise tulemusi enne järelduste tegemist põhjalikult uurida. Pole midagi parata - jõudlusanalüüs on täppismõõtmise korral keeruline ja aeganõudev ettevõtmine.

Täppismõõtmise muudavad võimalikuks tänapäevaste protsessorite pakutavad riistvaralised loendurid (sealhulgas ajalist kasvu arvestavad loendurid), mis arvestavad iga sündmuse aega. Ilma loendurita tuleks meil sündmuste omistamisel koodi piirkondadele tegelda eraldi iga sündmusega, suurendades ise loenduri näitu antud koodi piirkonnas. Tarkvaraliselt ei ole see

mõistagi võimalik, kuid eeldades, et sündmuste jaotus koodis on ühetaoline, kui me uurime iga sündmuse asemel ainult iga n-ndat sündmust, siis selle tarbeks on loodud häälestatav mõõtmismeetod, mida nimetatakse diskreetimiseks. Ajapõhine diskreetimine (TBS, Time Based Sampling) kasutab taimerit ja uurib regulaarselt programmi loendurit programmikoodi histogrammi loomiseks. Sündmusepõhine diskreetimine (EBS, Event Based Sampling) kasutab ära tänapäevaste protsessorite riistvaralisi loendureid ning töötab režiimis, milles loenduri alatäite korral kutsutakse välja katkestusetöötleja vastava sündmusjaotuse histogrammi tekitamiseks; töötlejas taaslähtestatakse sündmuseloendur alati diskreetimismeetodi n-väärtusele. Diskreetimise eeliseks on see, et koodi ei tule muuta, kuid see on siiski omamoodi kompromiss: mainitud eeldus võib olla korrektne, kui n on väike, aga mida väiksem on n, seda suurem on katkestusetöötleja üldkulu.

Teine mõõtmisviis on selle matkimine, mis juhtub arvutis antud koodi täitmisel, st. täitmise simulatsioon. Simulatsioon arvestab alati enam-vähem täpse masinamudeliga, kuid väga konkreetsete nõuete korral, kus on vaja äärmiselt realistlikku täpsust, võib simulatsiooniaeg olla tegelikkuses talumatult suur. Simulatsiooni eeliseks on see, et koodi võib ilma häirivaid tulemusi kartmata lisada suvalist, ka keerukat mõõte/simulatsioonikoodi. Lisamine otse enne täitmist (niinimetatud käitusinstrumentatsioon) originaal-binaarfaili kasutades on kasutajale äärmiselt mugav, sest puudub vajadus taaskompileerimise järele. Simulatsioon on mõttekas siis, kui simuleeritakse ainult teatud masina osi lihtsa mudeliga. Teine positiivne joon on see, et lihtsa mudeli tekitatud tulemusi on enamasti suhteliselt lihtsam mõista - sageli ongi tegeliku riistvara probleemiks see, et tulemustesse satuvad masina erinevate osade omavahel kattuvad efektid.

1.3 Profileerimistööriistad

Tuntuim sellistest tööriistadest on GCC profileerimistööriist gprof. Selle kasutamiseks tuleb programm kompileerida võtmega `-pg`, misjärel programmi käivitamine tekitab faili `gmon.out`, mille käsuga **gprof** saab muuta inimsilmale arusaadavale kujule. Puuduseks on aga vajadus kasutada kompileerimist, et ette valmistada käivitatav fail, mis peab olema staatiliselt lingitud. Meie kasutame kompilaatori genereeritud instrumentatsiooni, mis mõõdab funktsioonides esinevaid väljakutseteid ja arvestab vastavalt väljakutseid, koos TBS-iga, mis tagab koodi ajajaotuse histogrammi. Nende kahe infokogumi põhjal saab heuristiliselt arvutada funktsioonide kumulatiivset aega, st. aega, mis veedetakse funktsioonis koos kõigi sellest välja kutsutud funktsioonidega.

Sündmuste toimumise täppismõõtmiseks on olemas selliste funktsioonidega teegid, mis suudavad lugeda riistvaraliste loendurite andmeid. Tuntumad neist on PerfCtr Linux® jaoks ja arhitektuurist sõltumatud PAPI ning PCL teegid. Täppismõõtmine vajab siiski koodi instrumentatsiooni, nagu juba eespool märgitud. Teekide puhta kasutamise asemel võib kasutada ka automaatse instrumentatsiooni süsteeme, nagu ADAPTOR (FORTRAN-i lähtekoodi instrumentatsioon) või DynaProf (koodi sisestamine DynInst'i vahendusel).

OProfile on diskreetimist kasutav Linux® süsteemse profileerimise tööriist.

Mitmes mõttes on profileerimiseks väga mugav kasutada Cachegrindi või Callgrindi, mis on käitusinstrumentatsiooni raamistikku Valgrind kasutavad simulaatorid. Et nende puhul pole vaja ligipääsu riistvaralistele loenduritele (see on tänapäeva Linux® distributsioonide puhul sageli keeruline) ning profileeritavad binaarfailid saab jätta muutmata, on need heaks alternatiiviks muudele profileerimistööriistadele. Simulatsiooni miinuspoole - aegluse - saab osaliselt kompenseerida simulatsiooni ainult huvipakkuvates programmide osades ning isegi võib-olla ainult silmuste mõningates kordustes ette võttes. Mõõtmis/simulatsiooniinstrumentatsioonita on Valgrindi aegluskoefitsient kõigest vahemikus 3 kuni 5. Pealegi võib juhul, kui sulle pakub huvi ainult väljakutsete graafik ja väljakutsete loendamine, lülitada välja vahemälu simulaatori.

Vahemälu simulatsioon on esimene samm reaalaja hindamisel, kuivõrd tänapäevaste süsteemide korral on käitus äärmiselt tundlik niinimetatud *vahemälu* (väikesed ja kiired puhvrid, mis kiirendavad korduvaid pöördumisi ühtede ja samade mälupesade poole) kasutamisele. Cachegrind simuleerib vahemälu mällupöördumisi jälitades. Saadud andmed hõlmavad instruktsioonide ja andmete mällupöördumiste ning 1. ja 2. taseme vahemälu vajakute arvu ja seondavad need käivitatava programmi lähteridade ja funktsioonidega. Tüüpiliste protsessorite vajakute latentsusaega kasutades saab vajakuid kombineerides hinnata kulunud aega.

Callgrind on Cachegrindi laiend, mis loob käigult programmi väljakutsete graafiku, st. näitab, millised funktsioonid milliseid välja kutsuvad ja kui palju sündmusi funktsiooni töö ajal ette tuleb. Lisaks saab profileerimisandmeid koondada eraldi lõimedesse ja väljakutsete ahelatesse. See pakub profileerimisandmeid instruktsiooni tasemel, mis lubab dissambleeritud koodi annoteerimist.

1.4 Kuvamine

Profileerimistööriistad loovad tüüpiliselt päris palju andmeid. Soov võimalikult vähese vaevaga väljakutsete graafikus liikuda ning kiiresti lülituda funktsioonide sorteerimise režiimide ja erinevate sündmusetüüpide näitamise vahel ongi motiveerinud selleks kõige paremini sobivate graafiliste kasutajaliideste loomist.

KCachegrind on nende soovide täitja, võimaldades kuvada profileerimisandmeid. Kuigi algselt loodi see Cachegrindi ja Calltree andmete võimalikult hõlpsat sirvimist silmas pidades, on olemas konverterid, mis suudavad näidata ka muude tööriistade profileerimisandmeid. Lisas kirjeldame ka Cachegrindi/Callgrindi failivormingut.

Lisaks funktsioonide nimekirjale, mida saab sorteerida kas tavalise või kumulatiivse kulu järgi ning rühmitada ka lähtefaili, jagatud teegi või C++-klassi järgi, pakub KCachegrind valitud funktsiooni korral veel mitmeid vaateid, nimelt

- väljakutsete graafiku vaade, mis näitab väljakutsete graafiku osa valitud funktsiooni piirkonnas,
- puukaardivaade, mis näitab pesastatud väljakutsete seoseid, samuti kumulatiivset kulu, mis võimaldab kiiresti visuaalselt tuvastada problemaatilised funktsioonid,
- lähtekoodi ja disassembleri annotatsioonivaade, mis võimaldab näha kuludetaile lähtekoodi ridade ja assembleri instruktsioonide suhtes.

Peatükk 2

KCachegrindi kasutamine

2.1 Kuvatavate andmete tekitamine

Kõik, kes jõudlust hinnata soovivad, tahavad mõistagi näha vastavaid andmeid, mõõtes profileerimistööriistaga rakenduse käituskarakteristika aspekte. KCachegrind ei paku omalt poolt ühtki profileerimistööriista, kuid tuleb sellega ometi hästi toime koostöös Callgrindiga ning suudab konverteri vahendusel kuvada ka OProfile'i loodud andmeid. Kuigi käesolev käsiraamat ei ole pühendatud nende tööriistade põhjalikule tutvustamisele, räägime järgnevalt siiski neist lühidalt.

2.1.1 Callgrind

Callgrind kuulub Valgrindi koosseisu. Pane tähele, et varem nimetati seda Calltree'ks, aga see nimi oli eksitav.

Kõige levinum on selle kasutamiseks käivitada rakendus käsurealt käsuga **`valgrind --tool=callgrind`** , näiteks

```
valgrind --tool=callgrind myprogram myargs
```

Programmi töö lõppemisel luuakse fail `callgrind.out.pid`, mille saab laadida KCachegrindi.

Veidi täiustatum viis on salvestada välja profileerimisandmed rakenduste määratud funktsiooni väljakutsumisel. Näiteks Konquerori korral ainult veebilehekülje renderdamise profileerimisandmete nägemiseks võib lasta andmed salvestada ainult siis, kui valitakse menüükäsk **Vaade** → **Laadi uuesti**. See vastab väljakutsele `KonqMainWindow::slotReload`, milleks kasuta käsku

```
valgrind --tool=callgrind --dump-before=KonqMainWindow::slotRe
load konqueror
```

See loob rea profileerimisandmete faile, mille nimele lisatakse lõppu järjestikku kasvavad numbrid. Luuakse ka ilma sellise numbrita fail, mille nime lõpus on ainult protsessi PID - selle faili avamisel KCachegrindis avatakse ka teised ning neid võib näha **osade ülevaates** ja **osade** nimekirjas.

2.1.2 OProfile

OProfile'i leiab selle koduleheküljelt. Järgi paigaldamiseks veebileheküljel toodud juhiseid, aga enne seda võiksid kontrollida, ega sinu distributsioon juba seda valmispakituna ei paku (nagu teeb näiteks SuSE®).

Süsteemne profileerimine on lubatud ainult administraatorile (root), kes ainsana saab jälgida kogu süsteemis toimuvat. Seepärast tuleb kõike järgnevat sooritada administraatori õigustes. Esmalt seadista profileerimisprotsess, kasutades graafilist kasutajaliidest **oprof_start** või käsureatööriista **opcontrol**. Standardseadistus peaks olema taimerirežiim (TBS, vaata selle kohta sissejuhatust). Mõõtmise käivitamiseks anna käsk **`opcontrol -s`**. Seejärel käivita vajalik rakendus ning anna hiljem käsk **`opcontrol -d`**. See kirjutab mõõtmistulemused failidena kataloogi `/var/lib/oprofile/samples/` . Andmete kuvamiseks KCachegrindis anna tühjas kataloogis käsk:

```
opreport -gdf | op2callgrind
```

See tekitab hulga faile - üks iga süsteemis töötava programmi kohta. Kõik need saab ükshaaval KCachegrindis avada.

2.2 Kasutajaliidese põhitõed

Kui käivitada KCachegrind käsurealt profiiliandmetega või avada need menüükäsuga **Fail** → **Ava...**, näed vasakul külgriba funktsioonide nimekirjaga ning paremal, põhiosas, kuvatakse valitud funktsiooni. Vaateala on jagatud mitmeks, et näidata korraga mitut erinevat vaadet.

Algul on ala jagatud ülemiseks ja alumiseks pooleks, mõlemas omad sakkidega valitavad vaated. Vaadete liigutamiseks kasuta sakkide kontekstimenüüd ning kohenda vaadetevahelisi eraldajaid. Kiireks lülitumiseks vaadete vahel kasuta **Vaade** → **Paigutus** → **Liigu järgmisele** (**Ctrl+**→) ja **Vaade** → **Paigutus** → **Liigu eelmisele** (**Ctrl+**←).

Kuvamisel on oluline aktiivne sündmuse tüüp: Callgrindi korral on need näiteks vahemälu vajakud või tsükli hinnang, OProfile'i korral lihtsaimal juhul 'taimer'. Sündmuse tüüpi saab muuta liitkastis tööriistaribal või **sündmuse tüübbi** vaates. Esimese ülevaate käituskarakteristikast saab vasakul nimekirjas funktsiooni main valides ning väljakutsete graafiku kuva uurides: nii näed, mis sinu programmis tegelikult toimub. Pane tähele, et väljakutsete graafiku vaates näidatakse ainult suurte sündmuste arvuga funktsioone. Graafikus mõnel funktsioonil topeltklõpsu tehes see muutub ning näitab väljakutsutud funktsioone valitud funktsiooni lähikonnas.

Graafilise kasutajaliidese põhjalikumaks tundmaõppimiseks tasuks lisaks käesolevale käsiraamatule tutvuda dokumentatsiooniga veebileheküljel. Lisaks sellele pakub iga KCachegrindi element abivõimalust 'Mis see on?'

Peatükk 3

Peamised kontseptsioonid

Selles peatükis seletame mõningaid KCachegrindi kontseptsioone ja tutvustame liideses kasutatavaid mõisteid.

3.1 Profileerimisandmete andmemudel

3.1.1 Kuluolemid

Sündmuste tüüpide (näiteks L2 vajakute) kuluarvestus omistatakse kuluolemitele, mis on seotud antud programmi lähtekoodi või andmestruktuuridega. Kuluolemid ei pruugi olla lihtsalt koodi- või andmepositsioonid, need võivad olla ka positsioonijärjendid. Nii on näiteks väljakutsel allikas ja sihtmärk, andmeaadressil aga andmetüüp ja koodipositsioon, kus eraldus aset leiab.

KCachegrind tunneb ja kasutab järgmisi kuluolemeid. Lihtsad positsioonid:

Instruktsioon
: Assembleri instruktsioon määratud addressiga.

Funktsiooni lähtekoodi rida
: Kõik instruktsioonid, mida kompilaator (silumisinfo vahendusel) seob antud lähtekoodi reaga, mis on määratud lähtekoodi faili nime ja reanumbriga ja mis käivitatakse mingi funktsiooni kontekstis. Viimast on vaja seepärast, et inline-funktsiooni sees esinev lähtekoodi rida võib esineda mitme funktsiooni kontekstis. Instruktsioonid, mis ei ole seotud reaalse lähtekoodi reaga, seotakse reaga 0 failis `???`.

Funktsioon
: Funktsioon koosneb antud fuktsiooni kõigist lähtekoodi ridadest. Funktsiooni määrab selle nimi ja asukoht mingis binaarobjektis, kui see on saadaval. Viimast on vaja seepärast, et ühe programmi binaarobjektid võivad sisaldada samanimelisi funktsioone (neile pääseb ligi näiteks `dlopen` või `dlsym` abil; käituslinkur lahendab funktsioonid kasutatava binaarobjekti etteantud otsingujärjekorras). Kui profileerimistööriist ei suuda tuvastada funktsiooni sümbolnime (kui näiteks pole saadaval silumisinfot), kasutatakse tavaliselt esimesena käivitatud instruktsiooni aadressi või ˝`???`.

Binaarobjekt
: Kõik funktsioonid, mille kood asub antud binaarobjekti vahemikus kas peamises käivitatavas failis või jagatud teegis.

Lähtekoodi fail
: Kõik funktsioonid, mille esimene instruktsioon on seotud antud lähtekoodi faili reaga.

Klass
: Funktsioonide sümbolnimed on tavaliselt hierarhiliselt korraldatud nimeruumidesse, nt. C++ nimeruumidesse, või objektorienteeritud keele klassidesse. Nii võib klass sisaldada klassi funktsioone või ka põimitud klasse.

Profiili osa
: Profileerimise mingi ajalõik antud lõime ID-ga, protsessi ID-ga ja käivitatud käsureaga.

Nagu nimekirjast näha, võib üks kuluolemite kogum sageli defineerida mõne muu kuluolemi, mistõttu on olemas kuluolemite kumulatiivne hierarhia.

Positsioonijärjendid:

- Väljakutse instruktsiooniaadressilt sihtfunktsioonile.
- Väljakutse lähtekoodi realt sihtfunktsioonile.
- Väljakutse lähtekoodi funktsioonilt sihtfunktsioonile.
- Tingimuslik/tingimusteta hüpe lähtekoodist sihtinstruktsioonile.
- Tingimuslik/tingimusteta hüpe lähtekoodist sihtreale.

Hüpped funktsioonide vahel ei ole lubatud, sest neil ei ole ka väljakutsete graafikul mingit mõtet. Sestap tuleb sellised konstruktsioonid, nagu eranditöötlus ja C-keelele omased pikad hüpped vajaduse korral transleerida väljakutsete pinus.

3.1.2 Sündmuste tüübid

Profileerimisandmetes saab määrata igasuguseid sündmuste tüüpe neile nime andes. Nende kulu kuluolemi suhtes on 64-bitine täisarv.

Sündmuste tüüpe, mille kulu on määratletud profileerimisandmete failis, nimetatakse reaalseteks sündmusteks. Lisaks võib määrata viisi, kuidas tuletatakse reaalsetest sündmustest muid sündmuste tüüpe. Viimaseid nimetatakse järglassündmusteks.

3.2 Kuvamisolek

KCachegrindi akna kuvamisolek näitab:

- esmast ja sekundaarset sündmuse tüüpi,
- funktsiooni rühmitust (kasutusel **funktsiooni profiili** nimekirjas ja olemi värvimisel),
- profiili osasid, mille kulu kuva arvestab,
- aktiivset kuluolemit (nt. funktsiooni profiili külgdokilt valitud funktsiooni),
- valitud kuluolemit.

See olek mõjutab kuva.

Vaade näitab alati üht, aktiivset kuluolemit. Kui kuluolemi korral ei ole kuvamine võimalik, siis seda ka ei näidata (nt. kui valid rühmituse nimekirjas topeltklõpsuga ELF-objekti, ei ole ELF-objekti korral kuidagi võimalik lähtekoodi annotatsioon).

Nii näitab aktiivse funktsiooni korral väljakutsutute nimekiri kõiki funktsioone, mida aktiivne funktsioon välja kutsub. Nende seast võib valida mõne seda aktiivseks muutmata. Kui väljakutsete graafikut näidatakse selle kõrval, valib see automaatselt sama funktsiooni.

3.3 Graafilise kasutajaliidese komponendid

3.3.1 Külgdokid

Külgdokid ehk külgribad on aknad, mida võib seada mis tahes KCachegrindi peaakna serva. Need näitavad alati mingi reegli kohaselt sorteeritud kuluolemite nimekirja.

- **Funktsiooni profiil** on funktsioonide nimekiri, mis näitab kumulatiivset ja omakulu, väljakutsete arvu, funktsioonide nime ja asukohta.
- **Osade ülevaade**
- **Väljakutsete pinu**

3.3.2 Vaateala

Vaateala, mis tavaliselt moodustab KCachegrindi peaakna parempoolse osa, koosneb ühest (vaikimisi) või enamast kaardist, mis on korraldatud kas rõhtsalt või püstiselt. Iga kaart näitab korraga ainult ühe kuluolemi vaadet. Olemi nime näeb kaardi ülaosas. Kui kaarte on palju, saab korraga ometi olla neist aktiivne ainult üks. Aktiivse kaardi olemi nime näidatakse rasvases kirjas ning see määrab ka KCachegrindi akna aktiivse kuluolemi.

3.3.3 Kaardi alad

Igal kaardil võib olla kuni neli vaateala, nimelt üleval, paremal, vasakul ja all. Igas alas võib olla mitu kaartidele eraldatud vaadet. Nähtava vaate saab valida kaardirealt. Ülemise ja parempoolse ala kaardiribad asuvad üleval, vasakpoolse ja alumise ala omad all. Selle, milline vaade on millisel alal, saab määrata kaartide kontekstimenüüst.

3.3.4 Vaate sünkroniseerimine kaardil olemi valimisega

Aktiivse olemi kõrval on igal kaardil ka valitud olem. Kuna enamik vaatetüüpe näitab mitmeid olemeid, seades mingil moel keskmesse aktiivse, saab valitud elementi vahetada vaate sees liikudes (hiireklõpsuga või klaviatuuri abil). Tavaliselt on valitud elemendid ka esile tõstetud. Valitud olemi muutmisel mõnes kaardi vaates tõstavad sama kaardi muud vaated samuti automaatselt esile uue valitud olemi.

3.3.5 Sünkroniseerimine kaartide vahel

Mitme kaardi korral toob valiku muutmine ühel kaardil kaasa muutuse järgmisel kaardil (suunaga paremale/alla). Selline sidumine võimaldab näiteks kiiresti lehitseda väljakutsete graafikuid.

3.3.6 Paigutused

Akna kõigi kaartide paigutuse saab salvestada (**Vaade** → **Paigutus**). Aktiivse paigutuse dubleerimisel (**Vaade** → **Paigutus** → **Klooni** (**Ctrl++**)) ming mõne suuruse muutmiseks või kuva liigutamisel kaardivaates kuhugi mujale saab endise ja uue paigutuse vahel lülituda kiirklahviga **Ctrl**+← ja **Ctrl**+→. Paigutusi saab KCachegrindis salvestada ning need jäetakse seansside vahel meelde. Parajasti aktiivse paigutuse saab muuta kõigile uutele KCachegrindi seanssidele vaikepaigutuseks või siis taastada vaikepaigutuse.

3.4 Külgdokid

3.4.1 Lameprofiil

Lameprofiil koosneb gruppide ja funktsioonide nimekirjast. Gruppide nimekirja kuuluvad sõltuvalt valitud grupi tüübist kõik grupid, kus esineb kulu. Gruppide nimekiri on peidetud, kui rühmitamine ei ole sisse lülitatud.

Funktsioonide nimekiri sisaldab valitud grupi funktsioone või kõiki funktsioone, kui rühmitamine ei ole sisse lülitatud. Funktsioonid on seatud järjekorda mõne veeru, näiteks kumulatiivse või omakulu järgi. Seda, kui palju funktsioone nimekirjas maksimaalselt näidatakse, saab määrata seadistustedialoogis (**Seadistused** → **KCachegrindi seadistamine**).

3.4.2 Osade ülevaade

Profileerimisel on võimalik luua mitu profiiliandmete faili, mida saab üheskoos KCachegrindis laadida. **Osade ülevaate** dokis näidatakse neid vastavalt loomisajale rõhtsalt korraldatuna, kusjuures ristkülikute suurus vastab antud osa kulule. Üht või mõnda osa valides saab piirata kulu, mida KCachegrind näitab muudes vaadetes, ainult valitud osaga või osadega.

Osad on aga täiendavalt jagatud kas partitsioneerimis- või kumulatiivses režiimis:

Osadeks jagamise režiim
: Partitsioneerimine: profiiliandmete osa on jagatud rühmadeks vastavalt valitud rühma tüübile. Kui näiteks valida ELF-objektide rühmad, näed iga kasutatud ELF-objekti (jagatud teegi või käivitatava faili) kohta värvilist ristkülikut, mille suurus sõltub objekti kulust.

Skeemirežiim
: Selle puhul näidatakse parajasti aktiivse funktsiooni kumulatiivset kulu osas ristkülikuna. See omakorda on täiendavalt jagatud, näidates oma väljakutsutute kumulatiivset kulu.

3.4.3 Väljakutsete pinu

See on täiesti fiktiivne 'kõige tõenäolisem' väljakutsete pinu. Seda alustatakse parajasti aktiivsest funktsioonist ning üles ja alla lisatake suurima kuluga väljakutsujad ja väljakutsutud.

Veerud **Kulu** ja **Väljakutsed** näitavad kõigi mainitud rea väljakutsete kulu.

3.5 Vaated

3.5.1 Sündmuse tüüp

Sündmuse tüübi nimekiri näitab kõiki antud sündmuse tüübi parajasti aktiivse funktsiooni saadaolevaid kulutüüpe ning vastavat kumulatiivset ja omakulu.

Nimekirjast sündmuse tüüpi valides saab muuta kõikjal KCachegrindis näidatavat kulutüüpi.

3.5.2 Väljakutsete nimekirjad

Need nimekirjad näitavaid väljakutseid nii parajasti aktiivsest funktsioonist kui parajasti aktiivsele funktsioonile. **Kõik väljakutsujad** ja **kõik väljakutsutud** tähendavad siin funktsioone, mida on võimalik väljakutsuja/väljakutsutu puhul jälgida isegi juhul, kui vahele jääb muid funktsioone.

Väljakutsete nimekirjad on järgmised:

- Otse**väljakutsujad**
- Otse**väljakutsutud**
- **Kõik väljakutsujad**
- **Kõik väljakutsutud**

3.5.3 Kaardid

Esmase sündmuse tüübi puukujuline vaade piki väljakutsete hierarhiat üles või alla. Iga värviline ristkülik tähistab funktsiooni, selle suurus on aga võimalikult hästi proportsioonis kuluga ajal, mis aktiivne funktsioon töötab (proportsionaalsuse täpsusel on küll omad piirangud).

Väljakutsujate kaardi korral näitab graafik kõigi parajasti aktiivse funktsiooni väljakutsujate pesastatud hierarhiat. **Väljakutsutute kaardi** korral näitab graafik kõigi parajasti aktiivse funktsiooni väljakutsutute pesastatud hierarhiat.

Välimuse valikud leiab kontekstimenüüst. Täpse suuruse näitamiseks vali **Jäta vigased piirded vahele**. Kuna see võib võtta tublisti aega, tasuks eelnevalt piirata kuvamise maksimaalne sügavus. **Parim** määratleb järglaste poolitamise suuna eellaste proportsiooni põhjal. **Alati parim** määratleb järelejäänud ruumi igale järglasele. **Ignoreeri proportsioone** jätab enne järglaste joonistamist ruumi funktsiooni nimele. Arvesta, et suuruse proportsioonid võivad eriti halval juhul olla tugevasti moonutatud.

Järglaste vahel saab liikuda ka klaviatuuri abil, kasutades vasak- ja paremnoole klahve. Üles- ja allanoole klahvidega saab hierarhias taseme võrra üles- või allapoole liikuda. Klahv **Enter** aktiveerib elemendi, millel parajasti viibid.

3.5.4 Väljakutsegraafik

See näitab väljakutsete graafikut aktiivse funktsiooni piirkonnas. Näidatakse ainult kulu aktiivse funktsiooni tegeliku töötamise ajal, st. `main()` puhul, kui see muidugi on näha, näidatakse kulu, mis on tegelikult sama aktiivse funktsiooni kuluga, kuna see on osa `main()` kumulatiivsest kulust aktiivse funktsiooni töötamise ajal.

Tsüklite puhul näitavad sinised väljakutsenooled, et tegemist on kunstliku väljakutsega, mis on lisatud ainult graafiku korrigeerimiseks ja mida tegelikult pole kunagi esinenud.

Kui graafik ei mahu näitamisalasse ära, näidatakse küljel tillukest eelvaatlust. See sarnaneb kõiges täpselt väljakutsepuule ning valitud funktsioon on seal esile tõstetud.

3.5.5 Annotatsioonid

Annoteeritud lähtekoodi ja assembleri nimekiri näitab parajasti aktiivse funktsiooni lähtekoodiridu/disassembleeritud instruktsioone koos (oma)kuluga lähterea/instruktsiooni koodi täitmisel. Väljakutse korral lisatakse lähtekoodi read väljakutse detailidega: väljakutse (kumulatiivne) kulu, väljakutsete arv ja väljakutse sihtmärk.

Väljakutse sihtmärgi aktiveerimiseks vali selline väljakutse infoga rida.

Peatükk 4

Käskude seletused

4.1 KCachegrindi peaaken

4.1.1 Menüü Fail

Fail → Uus (Ctrl-N)

Avab tühja tipptaseme akna, millesse saab laadida profileerimisandmed. Õigupoolest pole seda väga vajagi, sest **Fail → Ava...** tekitab samuti uue tipptaseme akna, mis näitab juba valitud andmeid.

Fail → Ava (Ctrl-O)

Avab KDE failidialoogi, milles saab valida laaditava profileerimisandmete faili. Kui parajasti avatud tipptaseme aknas on juba mingid andmed, avatakse uus aken. Kui soovid avada täiendavad profileerimisandmed aktiivses aknas, kasuta käsku **Fail → Lisa**.

Profileerimisandmete failinime lõpus seisab tavaliselt `.pid.part-threadID`, kus *part* ja *threadID* tähistavad ühe ja sama rakenduse profileerimise erinevaid andmefaile. Kui laadida fail, mille lõpus seisab ainult `pid`, laetakse ühtlasi ka kõik antud profileerimise muude failinime lõppudega andmefailid.

Kui sul on profileerimisandmete failid `cachegrind.out.123` ja `cachegrind.out.123.1`, siis esimest laadides laetakse automaatselt ka teine.

Fail → Lisa

Lisab profileerimisandmete faili aktiivsesse aknasse. Selle käsuga saab mitu andmefaili laadida ühes ja samas tipptaseme aknas isegi siis, kui need ei ole pärit ühest ja samast profileerimisest (sellele osutavad failinimed). Seda saab kasutada näiteks võrdlemiseks.

Fail → Laadi uuesti (F5)

Laadib profileerimisandmed uuesti. See on tõenäoliselt kõige huvipakkuvam siis, kui juba avatud rakenduse profiili jaoks on loodud uus profileerimisandmete fail.

Fail → Välju (Ctrl-Q)

Lõpetab KCachegrindi töö

Peatükk 5

Küsimused ja vastused

Seda dokumenti on võibolla juba uuendatud. Värskeima versiooni leiad aadressilt http://docs.kde.org/current/kdesdk/ .

1. *Milleks see KCachegrind üldse hea on?*

   KCachegrindist on kasu tarkvara arendamise viimases järgus, mida nimetatakse profileerimiseks. Kui sa ei tööta ise tarkvara välja, ei ole sul ka KCachegrindiga midagi tarka peale hakata.

2. *Mis on **Kumul.** ja **Oma** erinevus?*

   Need on teatud sündmuse tüübi korral funktsiooni kulu atribuudid. Kui üks funktsioon teist välja kutsub, on mõttekas eristada funktsiooni enda kulu ('omakulu') ja nii selle kui kõigi väljakutsutud funktsioonide kulu ('kumulatiivne kulu').

   Nii näiteks on `main()` korral kumulatiivne kulu alati peaaegu 100%, samas kui selle omakulu on tavaliselt tühine, sest tegeliku töö teevad ära muud funktsioonid.

3. *Minu KCachegrindi tööriista- ja menüüriba näeb välja üsna tühi. Kas see peabki nii olema?*

   Ilmselt on KCachegrindi paigaldamisel midagi valesti läinud. Soovitatav on rakendus kompileerida paigalduspref iksiga, milleks on sinu süsteemi KDE baaskataloog, näiteks **`configure --prefix= /opt/kde4; make install`**. Kui valid mõne muu kataloogi, näiteks `$HOME /kde`, peab panema keskkonnamuutuja `KDEDIR` enne KCachegrindi käivitamist sellele kataloogile osutama.

4. *Kui ma teen **väljakutsegraafikus** funktsioonil topeltklõpsu, näitab see funktsiooni `main()` jaoks valitud funktsiooniga sama kulu. Kas see ei peaks olema üsna konstantselt 100%?*

   Sa aktiveerisid funktsiooni `main()` all, mille kulu on väiksem kui funktsioonil `main()`. Iga funktsiooni korral näidatakse ainult funktsiooni täieliku kulu osa ehk siis kulu *aktiveeritud* funktsiooni töötamise ajal, mis tähendab, et suvalise funktsiooni näidatav kulu ei saa kunagi olla suurem kui aktiveeritud funktsiooni kulu.

Peatükk 6

Sõnastik

Kuluolem
: Abstraktne element, mis on seotud lähtekoodiga, mille sündmusi arvestatakse. Kuluolemi mõõtmeteks on koodi asukoht (nt. lähterida, funktsioon), andmete asukoht (nt. kasutatud andmete tüüp, andmeobjekt), täitmise asukoht (nt. lõim, protsess) ning eelmainitud asukohtade järjendid (nt. väljakutsed, objekti kasutamine lausest, vahemälust väljatõrjutud andmed).

Sündmuste kulu
: Teatud sündmuse tüübi sündmuste summa täitmise ajal, mis on seotud teatud kuluolemiga. Kulu on seotud olemiga.

Sündmuse tüüp
: Sündmuse liik, mille kulu saab siduda kuluolemiga. Need jagunevad reaalseteks sündmuse tüüpideks ja päritud sündmuse tüüpideks.

Päritud sündmuse tüüp
: Virtuaalne sündmuse tüüp, mida näitab ainult vaade ja mis on määratud reaalsete sündmuste tüüpide põhjal teatud valemiga.

Profiiliandmete fail
: Fail, mis sisaldab profileerimiseksperimendi või selle osa käigus mõõdetud või jälituse järeltöötlusel loodud andmeid. Selle suurus on tavaliselt lineaarses sõltuvuses programmi koodi suurusest.

Profiiliandmete osa
: Profiiliandmete faili andmed.

Profileerimisekperiment
: Programmi töö profileerimistööriista kontrolli all, mis võib anda tulemuseks palju programmi osade või lõimede põhjal loodud profileerimisandmete faile.

Profileerimisprojekt
: Profileerimiseksperimentide seadistus mingi programmi jaoks, mida soovitakse profileerida (võib-olla ka selle mitmeid versioone). Profileerimisandmete võrdlemisel on tavaliselt mõtet ainult ühe profileerimisprojekti eksperimentidega loodud erinevate profileerimisandmete korral.

Profileerimine
: Statistilise info kogumine töötava programmi käituskarakteristika kohta.

Reaalne sündmuse tüüp

Sündmuse tüüp, mida saab mõõta tööriistaga. See vajab sensori olemasolu antud sündmuse tüübile.

Jälitus

Ajatempliga sündmuste jada, mis toimusid jälitatava programmi töötamise ajal. Selle suurus on tavaliselt lineaarses sõltuvuses programmi töötamise ajast.

Jälituse osa

Vt. "Profiiliandmete osa".

Jälitamine

Programmi töötamise jälgimine ja toimuvate sündmuste salvestamine koos ajatemplitega väljundfaili ehk jälitusfaili.

Peatükk 7

Autorid ja litsents

Tänud Julian Sewardile suurepärase Valgrindi ja Nicholas Nethercote'ile Cachegrindi lisanduse eest. Ilma nende programmideta ei oleks KCachegrindi lihtsalt olemas. Neilt on pärit ka hulk häid mõtteid GUI kohta.

Tänud ka kõigile kasutajatele, kes saatsid vearaporteid ja ettepanekuid.

Tõlge eesti keelde: Marek Laane bald@starman.ee

Lisa A

Paigaldamine

A.1 KCachegrindi hankimine

KCachegrind kuulub KDE paketi kdesdk koosseisu. Vahepealseid vähemtoetatud väljalaskeid, Callgrindi ja täiendava dokumentatsiooni leiab veebilehekülgelt. Sealt leiab täpsemaid juhiseid paigaldamise ja kompileerimise kohta.

A.2 Nõuded

KCachegrindi kasutamiseks on vajalik KDE 4.x. Profileerimisandmete tekitamiseks on soovitatav kasutada Cachegrindi või Calltree/Callgrindi.

A.3 Kompileerimine ja paigaldamine

Et KCachegrind oma süsteemis kompileerida ja paigaldada, anna KCachegrind baaskataloogis järgmised käsud:

```
% ./configure
% make
% make install
```

Kuna KCachegrind kasutab **autoconf**'i ja **automake**'i, ei tohiks kompileerimisel probleeme esineda. Kui neid siiski ette tuleb, anna sellest palun teada KDE meililistides.

A.4 Seadistamine

Kõik seadistamisvõimalused leiab seadistustedialoogis või vaadete kontekstimenüüdes.