Das Handbuch zu KCachegrind

Ursprünglicher Autor der Dokumentation: Josef Weidendorfer
Aktualisierung und Korrekturen: Federico Zenith
Deutsche Übersetzung: Thomas Reitelbach

Das Handbuch zu KCachegrind

Inhaltsverzeichnis

Zusammenfassung

KCachegrind ist ein Werkzeug zur Visualisierung von Profildaten auf der Basis von KDE Frameworks 5.

Kapitel 1

Einleitung

KCachegrind ermöglicht das Durchsehen der von einem Profiler erzeugten Profildaten. In diesem Kapitel wird erklärt, warum man Profildaten erhebt und wie man sie erzeugt. Außerdem werden einige verfügbare Profiler-Werkzeuge vorgestellt.

1.1 Profiling

Bei der Entwicklung einer Anwendung gehört es zu den abschließenden Schritten, es in der Geschwindigkeit zu optimieren. Es ist natürlich nicht sinnvoll, die Funktionen zu optimieren, die selten aufgerufen werden. Daher muss der Programmautor wissen, in welchem der Programmabschnitt die meiste Zeit verbraucht wird.

Für sequentiell ablaufenden Code reicht es in der Regel aus, statistische Daten über die Laufzeiteigenschaften des Programms zu ermitteln. Dazu gehört die in bestimmten Funktionen verbrachte Zeit und die zugehörigen Codezeilen. Das Erheben dieser Daten nennt man Profiling. Dabei läuft das Programm in einer kontrollierten Laufzeitumgebung des Profiling-Werkzeugs ab, welches am Ende des Durchlaufs eine Zusammenfassung der gesammelten Daten ausgibt. Im Gegensatz dazu gibt es auch parallel ablaufenden Code. Geschwindigkeitsprobleme resultieren hier häufig daraus, dass ein Prozessor auf Daten eines anderen Prozessors warten muss. Diese Wartezeit lässt sich nicht ohne Weiteres ermitteln und zuordnen; in einem solchen Fall müssen Ereignisse mit Hilfe von Zeitstempeln rückverfolgt werden. Diese Art Daten kann KCachegrind nicht anzeigen.

Nach der Analyse der erzeugten Profiling-Daten sollte es ein Leichtes sein, den möglichen Flaschenhals im Code aufzufinden. Beispielsweise kann man den Code in besonders häufig aufgerufenen Funktionen optimieren. Danach sollte man den Erfolg der Optimierungen mit einem erneuten Profiler-Durchlauf überprüfen.

1.2 Profiling-Methoden

Um verbrauchte Zeit zu messen oder eingetretene Ereignisse in einem bestimmten Code-Bereich aufzuzeichnen (z. B. einer Funktion) muss zusätzlicher Code vor und hinter dem fraglichen Bereich einprogrammiert werden. Dieser Code misst dann die Zeit oder liest einen globalen Ereigniszähler aus und berechnet die Unterschiede. Diese Veränderung des Original-Codes nennt man Instrumentieren. Die Instrumentierung kann der Programmierer, der Compiler oder auch die Laufzeitumgebung einrichten. Die fraglichen Code-Bereiche sind zumeist verschachtelt und der Aufwand für die Messung dieser Bereiche beeinflusst immer auch die Messergebnisse. Daher

sollte die Methode der Instrumentierung nur in ausgewählten Bereichen angewandt und die Ergebnisse mit Bedacht ausgewertet werden. Dieser Umstand macht eine wirklich exakte Messung zu einer äußerst komplexen Aufgabe.

Eine exakte Messung ist dennoch möglich, nämlich mit Hilfe von Hardware-Zählwerken, so wie sie in den heutigen modernen Prozessoren vorkommen. Diese Zähler inkrementieren mit jedem auftretenden Ereignis. Wir möchten auftretende Ereignisse bestimmten Code-Bereichen zuordnen können. Ohne Hardware-Zähler müssten wir jedes Ereignis selbst mit Hilfe von Software zählen - was natürlich nicht möglich ist. Unter der Annahme, dass die Ereignisverteilung im Quell-Code gleichmäßig ist, wurde jedoch eine Methode entwickelt, die nur jedes n-te Ereignis untersucht - wodurch die „Kosten" der Messung beeinflussbar sind: Sampling. Zeitbasiertes Sampling (Time Based Sampling, TBS) untersucht in regelmäßigen Zeitabständen den Programmzähler und erzeugt ein Histogramm über den Programmcode. Das ereignisbasierte Sampling (Event Based Sampling, EBS) untersucht im Gegensatz dazu den Hardware-Zähler moderner Prozessoren. Dazu wird eine Interrupt-Leitung verwendet, die bei einem Zähler-Niedrigstand auslöst; daraufhin wird ein Histogramm der zugehörigen Ereignisverteilung erzeugt: Innerhalb dieser Routine wird der Ereigniszähler wieder auf das Element n der Sampling-Methode zurückgesetzt. Der Vorteil des Sampling ist, dass der Programmcode nicht verändert werden muss. Aber es handelt sich immer noch um eine Kompromisslösung: Die Methode misst genauer, wenn n möglichst klein ist. Mit schrumpfendem n steigt jedoch auch die Auslastung des Interrupt-Handlers.

Eine weitere Messmethode ist es, bestimmte Ereignisse im Rechner zu simulieren, während der fragliche Code läuft; eine geführte Simulation. Eine solche Simulation wird von einem mehr oder weniger genauen Rechnermodell abgeleitet. Mit exakten Rechnermodellen sind ziemlich genaue Rückschlüsse auf die Realität möglich - jedoch ist der für solche Tests nötige Zeitaufwand in der Praxis oft inakzeptabel. Der Vorteil einer Simulation ist, dass im Grunde beliebig komplexer Messungs-Code in den Quell-Code eingefügt werden kann, ohne die Messergebnisse zu beeinflussen. Dies direkt vor der Ausführung (Laufzeit-Instrumentierung) mit der Original-Binärdatei durchgeführt ist für den Benutzer sehr komfortabel: Es ist kein erneutes Kompilieren erforderlich. Eine Simulation ist dann sinnvoll, wenn nur Teile einer Maschine mit einem einfachen Modell simuliert werden sollen. Ein weiterer Vorteil ist, dass die Messergebnisse aus Simulationen einfacher Rechnermodelle auch leichter zu interpretieren sind: oftmals ist das Problem mit echter Hardware, dass sich Ereignisse verschiedener Rechnermodule überschneiden.

1.3 Profiling-Werkzeuge

Am bekanntesten ist der GCC-Profiler gprof: Man muss das Programm mit der Option `-pg` kompilieren. Bei der Programmausführung wird dann eine Datei namens `gmon.out` erzeugt, die mittels **gprof** in ein leserliches Format umgewandelt werden kann. Ein Nachteil ist natürlich das notwendige Rekompilieren des Programms, welches außerdem statisch gelinkt werden muss. In diesem Fall wird die compilergenerierte Instrumentierung angewandt - alle Aufrufe rund um Funktionen und die zugehörigen Funktionsaufrufe werden gemessen. Zusammen mit zeitbasiertem Sampling (TBS) ergibt sich daraus ein Histogramm der Verteilung der verbrauchten Zeit im Code. Aus den beiden Zeiten lässt sich die Zeit errechnen, die ein Programm in bestimmten Funktionen und den von dort aufgerufenen Funktionen verbringt.

Es gibt auch Bibliotheken zur exakten Messung aufgetretener Ereignisse. Diese sind in der Lage, Hardware-Zeitgeber auszulesen. Am bekanntesten sind der Patch PerfCtr für Linux® und die architekturunabhängigen Bibliotheken PAPI und PCL. Dennoch ist zur exakten Messung eine Instrumentierung des Codes nötig. Man kann dazu entweder die Bibliotheken selbst oder automatische Systeme wie ADAPTOR (für FORTRAN-Quellcode) oder DynaProf (Code-Einschleusung über DynInst) verwenden.

OProfile ist ein systemweites Profiling-Werkzeug für Linux®. Es verwendet Sampling.

Eine sehr komfortable Möglichkeit des Profiling ist es jedoch, Cachegrind oder Callgrind zu verwenden. Dabei handelt es sich um Simulatoren, die auf das Instrumentierungs-Grundgerüst von Valgrind aufsetzen. Diese sind eine gute Alternative zu den anderen Profiling-Werkzeugen, weil

keine Hardware-Zähler verwendet werden (oftmals ist das mit heutigen Linux®-Installationen schwierig) und die Programmdateien nicht verändert werden müssen. Der Nachteil einer Simulation ist natürlich die Geschwindigkeitseinbuße. Diese kann allerdings reduziert werden, indem man die Simulation auf die interessanten Programmabschnitte beschränkt. Ohne Messung bzw. Simulation der Instrumentierung reduziert Valgrind die Geschwindigkeit um den Faktor 3 bis 5. Wenn nur das Aufruf-Diagramm und die Anzahl der Funktionsaufrufe von Interesse sind, kann man den Cache-Simulator ganz ausschalten.

Die Cache-Simulation ist der erste Schritt zur Abschätzung der echten Zeiten, da die Laufzeit von Programmen stark abhängig ist von der Ausnutzung des sogenannten Aufruf-*Caches* (kleine und sehr schnelle Puffer, die erneuten Zugriff auf die gleichen Speicherbereiche erheblich beschleunigen). Cachegrind führt eine Cache-Simulation durch, in der die Speicheradressen abgefangen werden. Die erhobenen Daten enthalten auch die Anzahl der Instruktionen, Zugriffe auf Daten-Bereiche im Cache sowie fehlgeschlagene Zugriffe auf den Cache der ersten und zweiten Stufe. Diese Informationen werden den entsprechenden Code-Zeilen und Funktionen des Programms zugeordnet. Durch eine geschickte Kombination der erhaltenen Daten lässt sich eine gute Abschätzung der benötigten Zeit ermitteln.

Callgrind ist eine Erweiterung für Cachegrind, die das Aufruf-Diagramm eines Programms zur Laufzeit erzeugt. Ein Aufruf-Diagramm gibt Auskunft darüber, welche Funktionen die anderen Funktionen aufrufen und wieviele Ereignisse in einem Funktionsdurchlauf abgearbeitet werden. Des Weiteren können die zu sammelnden Profiler-Daten nach Threads und Aufrufketten-Kontext unterteilt werden. Mit Callgrind ist es möglich, Profiling-Daten auf Instruktions-Ebene zu erhalten und damit auch disassemblierten Code zu untersuchen.

1.4 Darstellung

Profiling-Werkzeuge erzeugen für gewöhnlich eine sehr große Datenmenge. Durch diese sollte man möglichst einfach im Aufruf-Diagramm auf und ab navigieren können. Außerdem kann schnell und komfortabel zwischen verschiedenen Sortier-Modi für Funktionen und verschiedenen Ansichten für Ereignistypen umgeschaltet werden. Es bietet sich daher an, diese Aufgabe einem Programm mit grafischer Oberfläche zu überlassen.

KCachegrind ist eine grafisches Visualisierungswerkzeug für Profiling-Daten und erfüllt all diese Wünsche. Obwohl das Programm ursprünglich programmiert wurde, um die Daten von Cachegrind und Calltree anzuzeigen, gibt es auch Konverter, mit denen sich Daten anderer Profiling-Werkzeuge anzeigen lassen. Im Anhang finden Sie auch eine Beschreibung des Datenformats von Cachegrind/Callgrind.

Neben einer Liste der Funktionen sortiert nach exklusiven und inklusiven Kosten, optional gruppiert nach Quelldatei, Bibliothek oder C++-Klasse, unterstützt KCachegrind mehrere Ansichten einer ausgewählten Funktion. Im Einzelnen sind dies:

- eine Aufruf-Diagramm-Ansicht, die den Bereich des Aufruf-Diagramms um die gewählte Funktion darstellt,
- eine Baumansicht, die die Beziehungen eingebetteter Aufrufe zueinander darstellt, (mit den inklusiven Kosten zur schnellen Ermittlung problematischer Funktionen),
- eine Quellcode- und Disassembler-Ansicht, die Details zu den Kosten einzelner Quelltextzeilen und Assembler-Anweisungen darstellt.

Kapitel 2

KCachegrind verwenden

2.1 Visualisierungsdaten erheben

Als erstes müssen Geschwindigkeitsdaten erzeugt werden. Dafür misst man bestimmte Laufzeit-Eigenschaften eines Programms mit einem Profiling-Werkzeug. KCachegrind selbst enthält kein Profiling-Werkzeug, kann aber gut mit Callgrind zusammenarbeiten. Wenn man einen Konverter einsetzt, können auch die von OProfile erzeugten Profiler-Daten visualisiert werden. Eine Erklärung der genauen Bedienung dieser Programme würde den Rahmen dieses Handbuchs sprengen; im folgrenden Kapitel erhalten Sie dennoch eine kurze Schnellstart-Einführung.

2.1.1 Callgrind

Callgrind ist ein Bestandteil von Valgrind. Früher hieß das Programm Calltree, der Name war jedoch sehr irreführend.

Am häufigsten wird das Programm benutzt, indem **`valgrind --tool=callgrind`** dem Befehlszeilenaufruf der Anwendung vorangestellt wird. Zum Beispiel:

```
valgrind --tool=callgrind meinprogramm meineargumente
```

Beim Programmende wird die Datei `callgrind.out.pid` erzeugt, die Sie dann in KCachegrind laden können.

Eine fortgeschrittenere Methode ist es, Profiling-Daten immer nur dann zu erzeugen, wenn eine bestimmte Funktion Ihres Programms aufgerufen wird. Wenn Sie beispielsweise für Konqueror die Profiling-Daten zum Rendern einer Webseite erzeugen möchten, könnten Sie Daten erheben, wenn der Menüeintrag Ansicht/Aktualisieren benutzt wird (dies ist gleichzusetzen mit einem Aufruf von **Ansicht** → **Erneut Laden**). Dies entspricht dem Aufruf von `KonqMainWindow::slotReload`. Benutzen Sie dazu den folgenden Aufruf:

```
valgrind --tool=callgrind --dump-before=KonqMainWindow::slotReload konqueror
```

Dadurch werden mehrere Profiler-Dateien erzeugt, jede mit einer eindeutigen, aufsteigenden Nummer im Dateinamen. Es wird auch eine Datei ohne aufsteigende Nummer erzeugt. Diese können Sie in KCachegrind laden; die Folgedateien werden automatisch auch geladen und können in der **Komponenten-Übersicht** und **Komponenten**-Liste betrachtet werden.

2.1.2 OProfile

OProfile finden Sie unter http://oprofile.sf.net. Folgen Sie der Installationsanleitung auf der Webseite. Bevor Sie dies allerdings tun, prüfen Sie, ob das Programm bereits von Ihrer Distribution mitgeliefert wird (so z. B. bei SuSE®).

Systemweites Profiling ist nur dem Systemverwalter erlaubt, da so alle Aktionen im System überwacht werden können. Die folgenden Schritte müssen Sie also als Systemverwalter ausführen. Richten Sie zuerst den Profiling-Prozess ein; mit der GUI **oprof_start** oder mit dem Befehlszeilenprogramm **opcontrol**. Der Timer-Modus (TBS, siehe Einführung) sollte voreingestellt sein. Um die Messung zu starten, führen Sie **`opcontrol -s`** aus. Starten Sie nun die Anwendung an deren Werten Sie interessiert sind und führen Sie anschließend **`opcontrol -d`** aus. Die Messergebnisse werden in Dateien im Ordner `/var/lib/oprofile/samples/` gespeichert. Um die Daten zu betrachten, führen Sie

```
opreport -gdf | op2callgrind
```

in einem leeren Ordner aus. Es wird für jedes auf dem Systen laufende Programm eine Datei angelegt, die für sich in KCachegrind geöffnet werden kann.

2.2 Grundlagen der Benutzeroberfläche

Wenn Sie KCachegrind mit einer Profildaten-Datei als Argument starten, oder nachdem Sie über **Datei** → **Öffnen** eine Datei geöffnet haben, sehen Sie eine Navigationsleiste mit der Liste der Funktionen links und dem Hauptbereich rechts, in dem die ausgewählte Funktion dargestellt wird. Dieser Bereich kann so eingerichtet werden, dass er mehrere Ansichten anzeigt.

Beim ersten Start ist dieser Bereich in zwei übereinander angeordnete Ansichten aufgeteilt, jede Ansicht kann auswählbare Unterfenster enthalten. Benutzen Sie das Kontextmenü der Unterfensterreiter oder den Trennbalken, um diese Ansichten anzupassen, Um schnell zwischen den verschiedenen Ansichten zu wechseln, benutzen Sie **Ansicht** → **Layout** → **Nächstes Layout** (**Strg+Pfeil rechts**) und **Ansicht** → **Layout** → **Vorheriges Layout** (**Strg+Pfeil links**).

Der aktuelle Ereignistyp ist wichtig für die Darstellung: Für Callgrind ist das zum Beispiel „Cache Misses" oder „Cycle Estimation"; für OProfile im einfachsten Fall „Timer". Sie können den Ereignistyp mit dem Kombinationsfeld in der Werkzeugleiste oder in der **Ereignistyp**-Ansicht ändern. Wenn Sie die Funktion `main` wählen und einen Blick auf den Aufruf-Graphen werfen, sehen Sie einen ersten Überblick über die Laufzeit-Charakteristik. Sie sehen dort die Aufrufe, die von Ihrem Programm getätigt wurden. Beachten Sie, dass die Aufruf-Graph-Ansicht nur Funktionen anzeigt, die eine hohe Anzahl Ereignisse aufweisen. Wenn Sie im Aufruf-Graphen auf eine Funktion klicken, werden die aufgerufenen Funktionen um die ausgewählte herum angezeigt.

Um die GUI besser kennen zu lernen, lesen Sie zusätzlich zu diesem Handbuch auch den Dokumentationsabschnitt auf der Webseite. Zudem hat jedes Element in KCachegrind eine „Was ist das"-Hilfe.

Kapitel 3

Grundlegende Konzepte

Dieses Kapitel erläutert einige in KCachegrind verwendete Konzepte und Begriffe aus der Bedienungsoberfläche.

3.1 Das Datenmodell der Profildaten

3.1.1 Kosteneinheiten

Kosten von Ereignistypen (wie L2-Misses) sind Kostenpunkten zugeordnet, was Elemente mit Bezugspunkten im Quelltext oder in Datenstrukturen eines gegebenen Programms sind. Kostenpunkte können nicht nur einfache Positionen im Quelltext oder in Daten sein, sondern auch Positions-Tupel. Ein Aufruf, zum Beispiel, hat eine Quelle und ein Ziel, und eine Datenadresse kann einen Datentyp und eine Adresse im Quelltext haben, an der er alloziert wird.

KCachegrind sind die wie folgt genannten Kostenpunkte bekannt. Einfache Positionen:

Instruktion
: Eine Assembler-Instruktion an einer bestimmten Adresse.

Quelltextzeile einer Funktion
: Alle Instruktionen, die der Compiler (über Debug-Informationen) einer Quelltextzeile zuordnet (Dateiname und Zeilennummer) und die im Kontext einer Funktion ausgeführt werden. Letzteres ist wichtig, da eine Zeile innerhalb einer Inline-Funktion im Kontext verschiedener Funktionen auftauchen kann. Instruktionen ohne Zuordnung erscheinen mit Zeilennummer 0 in Datei `???`.

Funktion
: Alle Quelltextzeilen einer Funktion. Eine Funktion wird über ihren Namen und, sofern vorhanden, ihre Position in einem Binärobjekt angegeben. Letzteres ist wichtig, da jedes Binärobjekt eines Programms Funktionen mit demselben Namen enthalten kann (auf diese kann z. B. mit `dlopen` oder `dlsym` zugegriffen werden; der Laufzeit-Linker löst Funktionen in verwendeten Binärobjekten in einer bestimmten Suchreihenfolge auf). Wenn ein Profiling-Werkzeug den Symbolnamen einer Funktion nicht bestimmen kann, z. B. weil keine Debug-Informationen verfügbar sind, wird entweder die Adresse der zuerst ausgeführten Instruktion, oder `???` angegeben.

Binärobjekt
: Alle Funktionen, die innerhalb der Grenzen eines gegebenen Binärobjekts liegen. Dies gilt für die primäre Programmdatei, sowie für Shared-Libraries.

Quelltextdatei

Alle Funktionen, deren erste Instruktion einer Zeile der gegebenen Quelltextdatei zugeordnet ist.

Klasse

Symbolnamen von Funktionen sind normalerweise hierarchisch in Namensräumen geordnet, z. B. C++-Namensräume, oder Klassen in objektorientierten Sprachen; daher kann eine Klasse selbst Funktionen der Klasse und eingebetteter Klassen enthalten.

Profilabschnitt

Ein Zeitabschnitt eines Profiler-Durchlaufs mit einem gegebenen Thread-ID, Prozess-ID und ausgeführter Befehlszeile.

Wie Sie an der Liste erkennen können, definiert ein Satz Kostenpunkte oft einen neuen Kostenpunkt; daher entsteht eine Einrechnungshierarchie von Kostenpunkten, die durch die obige Beschreibung ersichtlich sein sollte.

Positions-Tupel:

- Aufruf von Instruktionsadresse auf Zielfunktion.
- Aufruf von Quelltextzeile auf Zielfunktion.
- Aufruf von Quellfunktion auf Zielfunktion.
- (Un)bedingter Sprung von Quellinstruktion auf Zielinstruktion.
- (Un)bedingter Sprung von Quellzeile auf Zielzeile.

Sprünge zwischen Funktionen werden nicht unterstützt, da diese in einem Aufrufgraphen nicht sinnvoll sind; daher müssen Konstruktionen wie Ausnahmen (Exceptions) und Long-Jumps in C so verändert werden, dass sie, sofern nötig, als Pop-Instruktion auf den Aufrufstapel ausgeführt werden.

3.1.2 Ereignistypen

In den Profildaten können weitere Ereignistypen angegeben werden, indem ihnen ein Name zugeordnet wird. Ihre Kosten bezogen auf einen Kostenpunkt ist ein 64-Bit-Integer.

Ereignistypen, deren Kosten in einer Profildaten-Datei angegeben sind, heißen „echte Ereignisse". Zusätzlich können Sie Formeln für Ereignistypen definieren, die aus echten Ereignissen errechnet werden. Diese heißen abgeleitete Ereignisse.

3.2 Darstellungsstatus

Der Darstellungsstatus eines KCachegrind-Fensters beinhaltet:

- den anzuzeigenden primären und sekundären Ereignistyp,
- die Funktionsgruppierung (für die **Funktionsprofil**-Liste und die Einfärbung der Elemente),
- die Abschnitte des Profils, die in die Darstellung eingehen
- ein aktiver Kostenpunkt (z. B. eine Funktion, die aus der Funktionsprofil-Ansicht ausgewählt wurde),
- ein ausgewählter Kostenpunkt.

Dieser Status beeinflusst die Darstellung.

Ansichten werden immer nur für einen Kostenpunkt (den aktiven) dargestellt. Ist eine Darstellung nicht angemessen für einen Kostenpunkt, wird sie ausgeblendet (z. B. wenn ein ELF-Objekt aus der Gruppenliste ausgewählt wird, sind Anmerkungen am Quelltext nicht sinnvoll).

Zum Beispiel zeigt die Aufgerufene-Liste einer aktiven Funktion alle Funktionen, die auf der aktiven heraus aufgerufen wurden. Sie können eine dieser Funktionen markieren, ohne sie zu aktivieren. Wird der Aufruf-Graph daneben angezeigt, wird hier automatisch die gleiche Funktion ausgewählt.

3.3 Bestandteile der Benutzeroberfläche

3.3.1 Navigationsbereich

Docks sind Seitenfenster, die an jeder Seite des KCachegrind-Fensters angebracht werden können. Sie enthalten jeweils eine Liste mit Kostenpunkten in einer bestimmten Ordnung.

- Das **Funktionsprofil** ist eine Liste von Funktionen, in der die Inklusiv- und Exklusivkosten, die Anzahl der Aufrufe, der Name und die Position der Funktionen angezeigt werden.
- **Übersicht der Profilabschnitte**
- **Aufrufstapel**

3.3.2 Ansichtsbereiche

Der Anzeigebereich, gewöhnlich der rechte Bereich im KCachegrind-Hauptfenster, ist aus einem (Standard), oder mehreren Unterfenstern aufgebaut, die entweder waagerecht oder senkrecht angeordnet sind. Jedes Unterfenster enthält verschiedene Ansichten, die jeweils immer einen Kostenpunkt anzeigen. Der Name dieses Kostenpunkts wird oben im Unterfenster angezeigt. Sind mehrere Unterfenster vorhanden, ist immer nur eines von ihnen aktiv. Der Name des Kostenpunkts wird im aktiven Unterfenster fett dargestellt und bestimmt den aktiven Kostenpunkt im KCachegrind-Fenster.

3.3.3 Bereiche eines Unterfensters

Jedes Unterfenster kann bis zu vier Bereiche enthalten: oben, rechts, links und unten. Zudem kann jeder Bereich mehrere Ansichten enthalten. Der sichtbare Teil eines Bereichs wird über die Unterfensterleiste ausgewählt. Die Unterfensterleisten der oberen und rechten Bereiche befindet sich oben; die der linken und unteren Bereiche befinden sich unten. Sie können festlegen, welche Darstellungsart in welchem Bereich angezeigt wird, indem Sie das Kontextmenü des Unterfensters verwenden.

3.3.4 Abgleich von Ansichten über ein ausgewähltes Element in einem Unterfenster

Neben einem aktiven Element hat jedes Unterfenster ein ausgewähltes Element. Da die meisten Darstellungsarten mehrere Elemente anzeigen, wobei das aktive irgendwie zentriert wird, können Sie das gewählte Element ändern, indem Sie in der Ansicht navigieren (mit Mausklick oder der Tastatur). Ausgewählte Elemente werden hervorgehoben. Wenn Sie die Auswahl in einer Ansicht eines Unterfensters ändern, wird die Hervorhebung in allen anderen Ansichten ebenfalls geändert.

3.3.5 Abgleich zwischen Unterfenstern

Sind mehrere Unterfenster vorhanden, ändert eine Auswahländerung in einem Unterfenster eine Aktivitätsänderung im nächsten Unterfenster rechts oder unter dem ersten Unterfenster. Diese Verknüpfung ermöglicht eine schnelle Navigation in Aufruf-Graphen.

3.3.6 Layouts

Das Layout aller Unterfensteransichten eines Fensters kann gespeichert werden (siehe Menüpunkt **Ansicht** → **Layout**). Nach dem Kopieren des aktuellen Layouts (**Ansicht** → **Layout** → **Kopieren** (**Strg++**)) und Ändern einiger Größen oder Verschieben einer Darstellungsansicht in einen anderen Bereich einer Unterfensteransicht können Sie mit **Strg**+**Pfeil links** bzw. **Strg**+**Pfeil rechts** zwischen dem alten und dem neuen Layout umschalten. Die Layouts werden über KCachegrind-Sitzungen mit dem gleichen Profiler-Befehl hinweg gespeichert. Sie können die aktuellen Layouts als Voreinstellung für neue Sitzungen speichern, oder das Standard-Layout wiederherstellen.

3.4 Navigationsbereich

3.4.1 Flaches Profil

Das **flache Profil** enthält eine Gruppenliste und eine Funktionsliste. Die Gruppenliste zeigt alle Gruppen, in denen Kosten verursacht werden, abhängig vom gewählten Gruppentyp. Die Gruppenliste ist ausgeblendet, wenn die Gruppierung ausgeschaltet ist.

Die Funktionsliste enthält die Funktionen der gewählten Gruppe (oder alle Funktionen, wenn die Gruppierung abgeschaltet ist), sortiert nach Spalten, z. B. Inklusiv- oder Exklusivkosten. Die Anzahl der Elemente in der Liste ist begrenzt, und kann unter **Einstellungen** → **KCachegrind einrichten ...** angepasst werden.

3.4.2 Übersicht der Profilabschnitte

Während eines Profiler-Durchlaufs können mehrere Profildaten-Dateien erzeugt werden, die alle zusammen in KCachegrind geladen werden können. Das Dock **Übersicht der Profilabschnitte** zeigt diese an, waagerecht angeordnet nach Erstellungszeit. Die Größen der Rechtecke sind proportional zu den Kosten der Abschnitte. Sie können einen oder mehrere Abschnitte auswählen um die Anzeige der Kosten in den anderen KCachegrind-Ansichten auf diese Abschnitte zu beschränken.

Die Abschnitte sind weiter unterteilt in einen Partitions- und einen Inklusivkosten-Modus:

Anzeigemodus für Details

Die Aufteilung der Dateien eines Profildaten-Abschnitts wird in Gruppen angezeigt, je nachdem, welcher Gruppentyp ausgewählt ist. Sind zum Beispiel ELF-Objektgruppen ausgewählt, sehen Sie farbige Rechtecke für jedes benutzte ELF-Objekt (Shared-Library oder Programmdatei), deren Größe durch die verursachten Kosten bestimmt wird.

Diagramm-Modus

Es wird ein Rechteck mit den Inklusivkosten der aktiven Funktion angezeigt. Dieses ist wiederum in die Inklusivkosten der aufgerufenen Funktionen unterteilt.

3.4.3 Aufrufstapel

Dieser Aufrufstapel ist nicht echt sondern ein „sehr wahrscheinliches" Abbild. Er wird von der aktiven Funktion her aufgebaut und enthält alle Aufrufer und Aufgerufenen absteigend mit den höchsten Kosten oben.

Die Spalten **Kosten** und **Aufrufe** zeigen die Kosten aller Aufrufe der Funktion eine Zeile höher.

3.5 Ansichten

3.5.1 Ereignistyp

Die **Ereignistyp**-Liste zeigt alle verfügbaren Kostentypen und die dazugehörigen Exklusiv- und Inklusivkosten der derzeit aktiven Funktion für den Ereignistyp.

Wenn Sie einen Ereignistyp wählen, ändern Sie den Kostentyp, der überall in KCachegrind angezeigt wird.

3.5.2 Aufruflisten

Diese Listen zeigen Aufrufe auf und von der derzeit aktiven Funktion. **Alle Aufrufer** und **Alle Aufgerufenen** bedeutet, dass diese Funktionen in Richtung vom Aufrufer zum Aufgerufenen erreichbar sind, selbst wenn andere Funktionen dazwischen stehen.

Die Ansicht von Aufruflisten enthält:

- Direkte **Aufrufer**
- Direkt **Aufgerufene**
- **Alle Aufrufer**
- **Alle Aufgerufenen**

3.5.3 Karten

Eine Baumstruktur-Darstellung des primären Ereignistyps, auf- oder abwärts in der Aufrufhierarchie. Jedes farbige Rechteck stellt eine Funktion dar; die Größe ist (abgesehen von zeichnerischen Ungenauigkeiten) proportional zu den verursachten Kosten, während die aktive Funktion ausgeführt wird.

In der **Aufruferkarte** zeigt der Graph die eingebettete Hierarchie aller Aufrufer der aktiven Funktion; in der **Aufrufkarte** werden alle aufgerufenen Funktionen angezeigt.

Einstellungen zum Erscheinungsbild finden Sie im Kontextmenü. Wählen Sie **Unpassende Ränder nicht zeichnen** um genaue Größen-Proportionen zu bekommen. Da dieser Modus sehr zeitintensiv sein kann, ist es ratsam, die maximale Verschachtelungstiefe zu begrenzen. **Beste Teilungsrichtung pro Ebene** bestimmt die Teilungsrichtung anhand des Seitenverhältnisses des übergeordneten Elements. **Immer beste Teilungsrichtung** entscheidet anhand des verbleibenden Platzes für jedes Geschwister. **Proportionen ignorieren** nutzt den Platz zuerst für die Funktionsbeschriftung. Beachten Sie, dass das Größenverhältnis stark abweichen kann.

Tastatursteuerung ist mit der linken und rechten Pfeiltaste möglich, um die Geschwister zu durchlaufen; und mit den Tasten Pfeil hoch/runter, um Verschachtelungebenen hoch bzw. runter zu gehen. Die Eingabetaste aktiviert das aktuelle Element.

3.5.4 Aufrufgraph

Diese Ansicht zeigt den Aufrufgraph um die aktive Funktion herum an. Die angezeigten Kosten sind nur die Kosten, die bei der Ausführung der Funktion angefallen sind; d. h. die Kosten für `main()`, sofern angezeigt, sollten den Kosten der aktiven Funktion entsprechen, da diese ein Teil der Inklusivkosten von `main()` sind, die bei der Ausführung der aktiven Funktion angefallen sind.

Für Zyklen wird ein blauer Pfeil angezeigt, der angibt, dass es sich nicht um einen echten Aufruf handelt; er wird nie ausgeführt, sondern nur zum richtigen Zeichnen benötigt.

Ist der Graph größer als der Zeichenbereich, wird in einer Ecke eine Übersicht angezeigt. Es gibt ähnliche Darstellungen für die Aufruf-Baumansicht; die ausgewählte Funktion ist hervorgehoben.

3.5.5 Code-Anmerkungen

Die Quelltext/Assembler-Listen mit Anmerkungen zeigen die Quelltextzeilen bzw. disassemblierten Instruktionen der aktiven Funktion zusammen mit den (Exklusiv)-Kosten, die beim Ausführen der Quelltextzeile/Instruktion verursacht wurden. Im Falle eines Aufrufs werden Zeilen mit Details zum Aufruf in den Quelltext eingefügt: (Inklusiv)-Kosten innerhalb des Aufrufs, Anzahl der Aufrufe und Aufrufziel.

Wählen Sie so eine Aufrufinformationslinie, um das Aufrufziel zu aktivieren.

Kapitel 4

Befehlsreferenz

4.1 Das Hauptfenster von KCachegrind

4.1.1 Das Menü Datei

Datei → Neu (Strg-N)

Öffnet ein leeres Hauptfenster, in das Sie Profildaten laden können. Diese Aktion wird eigentlich nicht benötigt, da **Datei → Öffnen ...** ein neues Hauptfenster öffnet, wenn im aktuellen Fenster bereits Daten angezeigt werden.

Datei → Öffnen ... (Strg-O)

Öffnet den ˝Datei öffnen˝-Dialog zur Auswahl der zu ladenden Profildaten-Datei. Wenn im aktuellen Fenster bereits Daten angezeigt werden, wird ein neues Hauptfenster geöffnet. Zum Hinzufügen zusätzlicher Profildaten im aktuellen Fenster, verwenden Sie **Datei → Hinzufügen ...**.

Der Name von Profildaten-Dateien endet normalerweise auf `pid.part-threadID`, wobei *`part`* und *`threadID`* optional sind. *`pid`* und *`part`* werden für mehrere Profildaten-Dateien eines einzelnen Programmdurchlaufs verwendet. Wenn Sie eine Datei mit der Erweiterung `pid` öffnen, werden vorhandene Daten-Dateien desselben Durchlaufs mit zusätzlichen Erweiterungen automatisch geladen.

Beispiel: Existieren die Profildaten-Dateien `cachegrind.out.123` und `cachegrind.out.123.1`, wird beim Laden der ersten Datei automatisch die zweite Datei mit geladen.

Datei → Hinzufügen ...

Fügt im aktuellen Fenster eine Profildaten-Datei hinzu. Hiermit können Sie das Laden mehrerer Daten-Dateien in einem Hauptfenster erzwingen, obwohl diese nicht aus demselben Durchlauf stammen, also nicht den Namenskonventionen für Profildaten-Dateien entsprechen. Das ist vor allem zum Direktvergleich einsetzbar.

Datei → Neu laden (F5)

Profildaten neu laden. Dies ist vor allem dann sinnvoll, wenn eine neue Profildaten-Datei für eine bereits geladene Anwendung erzeugt worden ist.

Datei → Beenden (Strg+Q)

Beendet KCachegrind

Kapitel 5

Fragen und Antworten

1. *Wozu ist KCachegrind gut?*

   KCachegrind ist nützlich für die späte Software-Entwicklungs-Phase Profiling. Wenn man keine Anwendungen entwickelt, dann braucht man KCachegrind nicht.

2. *Was ist der Unterschied zwischen **Incl.** und **Self**?*

   Dies sind Kosteneigenschaften für Funktionen hinsichtlich einiger Ereignistypen. Da sich Funktionen gegenseitig aufrufen können, ist es sinvoll, zwischen Kosten zu unterscheiden, die die Funktion selbst verursacht („Exklusivkosten") und solchen, die alle aufgerufenen Funktionen berücksichtigt („Inklusivkosten").

   Also werden Sie für `main()` immer Inklusivkosten von nahezu 100 % haben, wogegen die Exlusivkosten vernachlässigbar sind, wenn die eigentliche Arbeit in einer anderen Funktion erledigt wird.

3. *Wenn ich in der **Aufrufgraph**-Ansicht auf eine Funktion klicke, werden für `main()` die gleichen Kosten angezeigt, wie für die gewählte Funktion. Sollten diese nicht immer 100 % betragen?*

   Sie haben eine Funktion unterhalb von `main()` gewählt, die geringere Kosten als `main()` selbst hat. Für jede Funktion wird nur der Teil der Kosten angezeigt, der von der *aktivierten* Funktion verursacht wird. Daher können die Kosten für eine Funktion nie höher sein als die der aktivierten Funktion.

Kapitel 6

Glossar

Kostenpunkt

Abstraktes Element mit Bezug zu Stellen im Quelltext denen sich Ereigniszahlen zuordnen lassen. Kostenpunkte werden in den Größen Position im Quelltext (z. B. Zeile, Funktion), Position der Daten (z. B. aufgerufene Datentypen, Datenobjekt), Ort der Ausführung (z. B. Thread, Prozess) und Tupel oder Tripel der genannten Positionen (z. B. Aufrufe, Zugriff auf Objekte, aus dem Cache entfernte Daten) angegeben.

Ereigniskosten

Summe der Ereignisse eines Ereignistyps, der während der Ausführung aufgetreten ist und zu einem bestimmten Kostenpunkt gehört. Die Kosten werden dem Punkt zugeordnet.

Ereignistyp

Art eines Ereignisses, dessen Kosten einem Kostenpunkt zugeordnet werden können. Es gibt echte und abgeleitete Ereignistypen.

Abgeleiteter Ereignistyp

Virtueller Ereignistyp, der nur in Darstellungen angezeigt wird, die mit einer Formel aus einem echten Ereignistyp errechnet werden.

Profildaten-Datei

Eine Datei mit Daten, die Messwerte eines Profil-Experiments (oder eines Abschnitts davon) enthält oder beim Nachverarbeiten von Profildaten erzeugt wurde. Die Datenmenge verhält normalerweise linear zur Größe des Quelltextes.

Profildaten-Abschnitt

Daten aus einer Profildaten-Datei.

Profil-Experiment

Ein Programmablauf, der von einem Profiling-Werkzeug überwacht wird und eine oder mehrere Profildaten-Dateien aus Abschnitten und/oder Threads des Ablaufs erzeugt.

Profil-Projekt

Eine Konfiguration für Profil-Experimente, die für ein zu überwachendes Programm, vielleicht in mehreren Versionen, verwendet wird. Profildaten zu vergleichen, ist nur dann sinnvoll, wenn sie in Experimenten eines Profil-Projekts erzeugt wurden.

Profiling

Der Prozess des Sammelns statistischer Informationen über die Laufzeit-Eigenschaften eines Programmablaufs.

Echte Ereignistypen

Ereignistyp, der mit einem entsprechenden Werkzeug gemessen werden kann. Es ist ein Sensor für den gegebenen Ereignistyp erforderlich.

Profildaten

Eine Abfolge zeitlich markierter Ereignisse, die beim Überwachen eines Programmablaufs aufgetreten sind. Die Datenmenge verhält sich normalerweise linear zu Laufzeit des Programms.

Profildaten-Abschnitt

Siehe "Profildaten-Abschnitt".

Tracing

Der Prozess des Überwachens eines Programmaufrufs. Dabei werden auftretende Ereignisse nach Zeitstempeln sortiert in eine Datei geschrieben, die Profildaten-Datei.

Kapitel 7

Danksagungen und Lizenz

Dank an Julian Seward für seine einzigartige Anwendung Valgrind und Nicholas Nethercote für den Cachegrind-Zusatz. Ohne diese Programme würde KCachegrind nicht existieren. Viele Ideen für die Benutzeroberfläche stammen ebenfalls von ihnen.

Danke für die vielen Fehlerberichte und Vorschläge von verschiedenen Benutzern.

Übersetzung Thomas Reitelbach tr@erdfunkstelle.de