Profiling – eine Übersicht

Name	*Funktion*	*Kommandozeile*	*Compilerflags*	*Einsatz*
GProf (GNU Profiler)	• Statistisches Profiling • flaches Profil (Liste von Funktionen annotiert mit ihrer Laufzeit) • Aufrufgraph	gprof [params] <programm> nützliche Parameter: --annotated source --no-graph (falls beim Übersetzen einige Module nicht mit -pg übersetzt wurden)	**-pg:** als Compilerflag wird Code instrumentiert zur Erstellung des Aufrufgraphen, als Bindeoption wird (statistisches) Profiling aktiviert **-g:** Debuginformationen generieren **-O0**: Optimierung ausschalten (Programm entspricht eher dem Quellcode)	• Einfaches oberflächliches Profil eines Programms erstellen • grober Überblick • zum Profiling von Bibliotheken spezielle Profiling Version dieser Bibliotheken benötigt
GCov (GNU Coverage)	• Überdeckungsanalyse • zeilenweise Anzahl der Ausführungen • Verzweiguns-wahrscheinichkeiten	gcov [params] <programm.c> nützliche Parameter: -b (gibt zusätzlich Verzwei-gungsstatistik nach jedem bedingtem Sprung aus)	**-fprofile-arcs:** Verzweigungs-informationen zur Laufzeit sammeln **-ftest-coverage:** Überdeckungs-analyse durchführen	• Zeilenweise genaues Profil • Optimierung von Verzweigungen • Test von Testdaten (gute Überdeckung?)
valgrind / callgrind 1)	• Speicheranalyse • Cache-Profiling • Aufrufgraph	valgrind –tool=callgrind <programm> -> kcachegrind callgrind.out.*	Keine gesonderte Übersetzung nötig	• Bedürfnis nach genauen Daten, Vorliebe für hübsche GUIs und viel Zeit • Profiling der benutzten Bibliotheken
oprofile	• Statistisches Profiling	Nur als root: • modprobe oprofile • oprof_start (grafisches Konfigurationstool) • opcontrol -s/-d • opreport/opgprof	Keine gesonderte Übersetzung nötig	• Systemweites Profiling • minimalinvasives Profiling • Kernel Profiling

1) um valgrind, callgrind und KCachegrind im IPD Pool nutzen zu können, erstmal mit “module add kcachegrind” einbinden. Am besten diesen Befehl gleich in ~/.bashrc einbauen.