

Inhaltsverzeichnis

1	Einleitung und Motivation	1
2	Debugging und Codeoptimierung	7
2.1	Debuggingtechniken	7
2.1.1	Strategien zur Auswertung von Beobachtungsdaten	8
2.1.1.1	Off-line-Debugging	8
2.1.1.2	On-line-Debugging	9
2.1.1.3	Post-mortem-Debugging	9
2.1.2	Mögliche Abstraktionsebenen	9
2.1.2.1	Anwendungsebene	10
2.1.2.2	Hochsprachebene	10
2.1.2.3	Maschinensprachebene	11
2.1.2.4	Hardwareebene	11
2.1.3	Instrumentierungstechniken	11
2.1.3.1	Quellcode-Instrumentierung	12
2.1.3.2	Objektcode-Instrumentierung	13
2.1.3.3	Hardware-Instrumentierung	15
2.1.4	Festlegung der betrachteten Debuggingtechnik	15
2.2	Techniken zur Codeoptimierung	16
2.2.1	Datenflußanalyse	16
2.2.1.1	Programmdarstellungen für die Datenflußanalyse	17
2.2.1.2	Skalare Datenflußanalyse	20
2.2.1.3	Datenflußanalyse für Felder und Abhängigkeitsanalyse	24
2.2.1.4	Interprozedurale Analyse	25
2.2.2	Code-verbessernde Transformationen	25
2.2.2.1	Optimale Nutzung des Befehlssatzes	26
2.2.2.2	Elimination von Redundanz und Overhead	26
2.2.2.3	Nutzung von Speicherhierarchien	28
2.2.2.4	Nutzung von Befehlspipelines	29
2.2.2.5	Erhöhung der Parallelität auf Anweisungs- und Schleifenebene	29
2.3	Auswirkung von Codetransformationen auf die Beobachtbarkeit von Programmen	31
2.3.1	Einfluß von Optimierungen auf die Abbildung zwischen Quell- und Zielprogramm	34
2.3.2	Debuggingprobleme durch Programmtransformationen	36
2.3.2.1	Codeprobleme	36
2.3.2.2	Ablaufprobleme	38

2.3.2.3	Das Variablenzuordnungsproblem	39
2.3.2.4	Datenprobleme	39
2.4	Anforderungen an Debugger für optimierte Programme	41
3	Methoden und Werkzeuge zum Debugging optimierter Programme	43
3.1	Klassifikation der Methoden	43
3.1.1	Unterdrückung von Transformationen	43
3.1.2	Verwendung debugger-unterstützender Transformationen	45
3.1.3	Behandlung der angewandten Transformationen	45
3.2	Effiziente Quellcode-Instrumentierung	46
3.2.1	Trace-Scheduling von quellcode-instrumentierten Programmen	46
3.2.2	Debugging-Unterstützung durch inkrementelle Optimierung	49
3.2.2.1	Debugging mit Hilfe inkrementeller Übersetzung	49
3.2.2.2	Debugging optimierter Programme mit Hilfe inkrementeller Optimierung	49
3.2.2.3	Inkrementelle Optimierung von Programmen mit Debug-Anforderungen	50
3.3	Festlegung von Beobachtungspunkten	53
3.3.1	Der Debugger für Typed Smalltalk	53
3.3.1.1	Modell des Programmzustands	53
3.3.1.2	Debug-Information	54
3.3.1.3	Implementierung des Debuggers	55
3.3.2	Der SELF-Debugger: Dynamische Deoptimierung	56
3.3.2.1	Debug-Information	57
3.3.2.2	Implementierung des Debuggers	57
3.4	Eine debugger-unterstützende Optimierungstechnik	58
3.4.1	Globale Umbenennung	59
3.4.2	Wiederverwendung von Variablen	61
3.4.3	Laufzeitsystem	62
3.4.4	Ergebnisse	62
3.5	Tabellengesteuerte Behandlung von Optimierungen	63
3.5.1	Navigator	63
3.5.1.1	Prozedureinbau	63
3.5.1.2	Cross-jumping	65
3.5.2	DOC	66
3.5.3	CXdb	69
3.5.4	Beschreibung der Effekte von Optimierungen durch Termersetzungssysteme	71
3.6	Analyse der Transformationen durch den Debugger	71
3.6.1	Berechnung nichtaktueller Variablen in lokal optimiertem Code	71
3.6.1.1	Programmdarstellung	72
3.6.1.2	Algorithmen zur Berechnung nichtaktueller Variablen	72
3.6.1.3	Recovery	73
3.6.1.4	Globale Optimierungen	74
3.6.2	Berechnung von nichtresidenten Variablen durch Datenflußanalyse	74

3.6.3	Berechnung und Recovery von nichtaktuellen Variablen bei lokalem Scheduling	76
3.6.3.1	Programmdarstellung	76
3.6.3.2	Berechnung von nichtaktuellen Variablen	77
3.6.3.3	Recovery	77
3.6.3.4	Ergebnisse	78
3.6.4	Berechnung nichtaktueller Variablen durch Datenflußanalyse	79
3.6.4.1	Definition der Aktualität	79
3.6.4.2	Programmdarstellung	80
3.6.4.3	Bestimmung der Aktualität durch Vergleich erreichender Definitionen	81
3.6.4.4	Bestimmung der Aktualität durch gekoppelte Datenflußanalyse	82
3.6.4.5	Behandlung indirekter Zuweisungen	84
3.6.5	Debugging parallelisierter Programme mit Hilfe virtueller Zeit	86
3.6.5.1	Parallelisierung durch Prozeßaufteilung	86
3.6.5.2	Virtuelle Zeit als Mittel zum Debugging parallelisierter Programme	86
3.7	Vergleich und Bewertung	89
3.7.1	Zielsetzung der Methoden	89
3.7.2	Anwendungsbereiche der Methoden	91
3.7.3	Implementierungsbezogene Eigenschaften	94
3.7.4	Implementierungs- und Wartungsaufwand	95
3.7.5	Realisierung	96
3.7.6	Zusammenfassung	96
4	Debugging-Unterstützung durch Analyse von Programmtransformationen	99
4.1	Abgrenzung und Ziele	99
4.1.1	Ziele des Verfahrens	99
4.1.1.1	Behandlung von Optimierungen	99
4.1.1.2	Interaktives Debugging	100
4.1.1.3	Weitgehende Unabhängigkeit vom Optimierer	101
4.1.1.4	Transparentes Debugerverhalten	102
4.1.1.5	Beliebige Haltepunktabbildung	104
4.1.1.6	Implementierung in realistischer Umgebung	105
4.1.1.7	Formales Modell	105
4.1.2	Grundzüge des Verfahrens	105
4.1.2.1	Debug-Information	106
4.1.2.2	Analyse im Debugger	106
4.1.2.3	Bewertung	108
4.2	Ein Modell zur Beschreibung von Abläufen optimierter Programme	109
4.2.1	Beschreibung von Programmabläufen mit Hilfe abstrakter Maschinen	109
4.2.2	Abläufe von kompilierten Programmen	110
4.2.3	Grundlegende Funktionen eines Debuggers	112
4.2.3.1	Bestimmung der aktuellen Anweisung	113
4.2.3.2	Ermittlung von Variablenwerten	113
4.2.3.3	Setzen von Haltepunkten	113

	4.2.3.4 Einzelschrittabarbeitung	114
	4.2.3.5 Aufzeichnung von Ablaufspuren	114
	4.2.3.6 Bestimmung von Prozeduraufrufketten	115
4.3	Schnittstelle zwischen Compiler und Debugger	115
4.3.1	Symbolinformation	115
4.3.2	Repräsentationen von Quell- und Zielprogramm	116
4.3.3	Abbildungen zwischen Quell- und Zielcode	118
	4.3.3.1 Abbildung zur Generierung von Pfadpaaren	121
	4.3.3.2 Semantische Äquivalenz	126
	4.3.3.3 Hilfsabbildungen	130
4.4	Ein Verfahren zur Bestimmung von Variablenwerten	131
4.4.1	Allgemeines	131
	4.4.1.1 Begriffsdefinition	131
	4.4.1.2 Datenfluß-Charakterisierung der Aktualität	132
4.4.2	Bestimmung von aktuellen Variablen in Abwesenheit von Schleifen- transformationen	135
	4.4.2.1 Bestimmung relevanter Zielcodevariablen	135
	4.4.2.2 Berechnung der Aktualität durch Datenflußanalyse	137
	4.4.2.3 Algorithmus	151
	4.4.2.4 Beweis der Korrektheit	157
	4.4.2.5 Behandlung indirekter Zuweisungen	165
	4.4.2.6 Komplexitätsbetrachtung	169
4.4.3	Bestimmung von aktuellen Variablen in schleifentransformierten Pro- grammen	171
	4.4.3.1 Bestimmung der erreichenden Definitionen im Quellprogramm 172	
	4.4.3.2 Einschränkung der Pfade im Zielprogramm	176
	4.4.3.3 Bestimmung der Verfügbarkeitsrelation im Zielprogramm	179
	4.4.3.4 Aktualität von Variablen	181
	4.4.3.5 Implementierungsaspekte	181
4.4.4	Erweiterungen	183
	4.4.4.1 Erweiterungen der Algorithmen	183
	4.4.4.2 Erweiterungen und Grenzen des Modells	191
4.4.5	Behandlung nichtaktueller Variablen	196
	4.4.5.1 Recovery	196
	4.4.5.2 Aufzeichnung von Variablenwerten	198
	4.4.5.3 Beschreibung der Effekte von Transformationen	199
4.5	Zur Bestimmung von Haltepunktabbildungen	200
4.5.1	Optimale Haltepunkte bezüglich des Programmzustands	202
4.5.2	Transparente Haltepunktabbildung bezüglich des Programmflusses	203
	4.5.2.1 Haltepunktprädikate	203
	4.5.2.2 Forderungen an die Transparenz der Haltepunktabbildung	204
	4.5.2.3 Ein Algorithmus zur Bestimmung syntaktischer Haltepunkt- abbildungen	207
4.5.3	Optimale Haltepunktabbildung	217
4.6	Weitere Debuggerfunktionen	219
4.6.1	Einzelschrittabarbeitung	219

4.6.2	Aufzeichnung von Ablaufspuren	220
4.6.3	Melden von Unterbrechungen	222
4.6.4	Bestimmung von Prozeduraufzuketten	223
4.6.5	Modifikation von Variablen	224
4.6.6	Funktionen zur Unterstützung der Fehlerlokalisierung	226
5	Implementierung und Ergebnisse	229
5.1	Der modifizierte SUN C-Compiler	229
5.1.1	Aufbau des Compilers	229
5.1.1.1	Ablauf des Compilers ohne Codeoptimierung	229
5.1.1.2	Ablauf des Compilers bei eingeschaltetem Optimierer	231
5.1.2	Aufbau des Optimierers und unterstützte Transformationen	232
5.1.2.1	Aufbau der Zwischendarstellung	232
5.1.2.2	Der Zwischencodeoptimierer <code>iropt</code>	233
5.1.2.3	Optimierung im Assembler	237
5.1.3	Erzeugung der Debug-Information	238
5.1.3.1	Quellcodebezogene Information	238
5.1.3.2	Flußgraph des Quellprogramms	238
5.1.3.3	Flußgraph des Zielprogramms	239
5.1.3.4	Abbildungen zwischen Quell- und Zielcode	240
5.2	Werkzeuge zum Debugging optimierter Programme	245
5.2.1	Das Datenflußanalysemodul <code>FLOW</code>	245
5.2.1.1	Aufbau des Datenflußanalysemoduls	245
5.2.1.2	Abbildungen zwischen Quell- bzw. Zielprogramm und den Programmdarstellungen	247
5.2.1.3	Resultat der Analyse	249
5.2.2	Der Debugger <code>DETOP/X</code>	250
5.2.3	Das Werkzeug <code>VISOPT</code> zur Visualisierung von Optimierungen	252
5.3	Ergebnisse	254
5.3.1	Aufwand zur Erzeugung der Debug-Information	254
5.3.2	Resultate der Untersuchungen mit dem Analysemodul <code>FLOW</code>	256
5.3.2.1	Laufzeit der Analyse	257
5.3.2.2	Häufigkeit nichtaktueller Variablen	258
5.3.2.3	Recovery	262
5.3.2.4	Haltepunkte mit transparentem Debuggerverhalten	263
5.3.2.5	Laufzeitinformation	265
5.3.2.6	Optimale Haltepunktabbildung	268
6	Zusammenfassung und Ausblick	269
A	Detaillierte Ergebnisse der Untersuchungen mit <code>FLOW</code>	273
	Literaturverzeichnis	277
	Index	291