

Vergleich von Pascal- Compilern

Michael Kaufmann

Projekt für die Vorlesung
„Computer Systems Performance Analysis and
Benchmarking“ von Dr. Christian Kurmann

Inhaltsverzeichnis

1 Ziel dieses Projekts	1
2 Getestete Pascal-Compiler	1
2.1 Delphi	1
2.2 Free Pascal	1
2.3 Virtual Pascal	2
2.4 GNU Pascal	2
2.5 TMT Pascal	3
2.6 Weitere Compiler.....	3
3 Konfiguration der Compiler	4
3.1 Datentyp-Unterschiede.....	4
3.1.1 „Real“	4
3.1.2 „LongInt“.....	4
4 Design und Messverfahren.....	5
5 Die Testprogramme (Workloads)	5
5.1 Ackermann-Funktion	5
5.2 Springertour	6
5.3 Dhystone.....	7
5.4 Whetstone	7
5.5 Matrixmultiplikation (Integer)	8
5.6 Matrixmultiplikation (Floating-Point)	9
5.7 QuickSort	10
5.8 ZipInfo (Zip-Datei testen).....	11
6 Messresultate.....	12
7 Analyse der Messresultate	13
7.1 Berechnung der Effekte.....	14
7.2 Interaktionen der Compiler mit den Testprogrammen	15
7.3 Verursacher der Variation	15
7.4 Analyse der Varianz (ANOVA)	15
7.5 Vertrauensintervalle für die Effekte	16
8 Schlussfolgerungen	16
Anhang: Turbo Pascal	18
Literaturverzeichnis.....	19

1 Ziel dieses Projekts

Das Ziel dieses Projekts ist, verschiedene Pascal-Compiler auf die Qualität des erzeugten Codes zu testen. Dazu wurden einige Testprogramme mit allen vorhandenen Compilern kompiliert und die Laufzeiten gemessen.

2 Getestete Pascal-Compiler

Bei „Pascal“ als Programmiersprache fällt den meisten wahrscheinlich Delphi (oder Niklaus Wirth als Erfinder der Sprache Pascal) ein. Ausser Delphi gibt es aber noch weitere Pascal-Compiler, die eine nähere Betrachtung verdienen.

Ich habe einige Tests auch mit Turbo Pascal durchgeführt, aber Turbo Pascal habe ich nicht in die nachfolgende Analyse miteinbezogen. Die Gründe dafür stehen im Anhang, der Turbo Pascal gewidmet ist.

Alle beteiligten Compiler werden im folgenden kurz vorgestellt.

2.1 Delphi

Delphi ist *die* Pascal-Entwicklungsumgebung schlechthin – ohne Delphi würde man heute kaum noch in Pascal programmieren.

Der Compiler optimiert nicht für Pentium-, sondern nur für 386er-Prozessoren. Am Test nahmen gleich zwei Versionen teil, um zu sehen, wie sich der Compiler über die Jahre entwickelt hat.

Kylix (das „Delphi für Linux“) nahm am Test nicht teil, da der Kylix-Compiler kaum grosse Unterschiede zum Delphi-Compiler aufweisen dürfte.

<i>Getestete Versionen:</i>	2.0 (vom 22.2.1996) 7.0 (vom 23.8.2001)
<i>Hersteller:</i>	Borland
<i>Lizenz:</i>	Kommerziell (Von Delphi 6 gab es eine kostenlose „Personal Edition“ für den privaten Gebrauch)
<i>Unterstützte Plattformen:</i>	Windows
<i>Prozessor-Optimierung:</i>	386er
<i>Sprachumfang:</i>	Mit objektorientierten Erweiterungen (Object Pascal) gegenüber dem Turbo-Pascal-Sprachumfang
<i>Webseite:</i>	http://www.borland.com/delphi/

2.2 Free Pascal

Free Pascal ist ein Open-Source-Compiler, der auch heute noch emsig weiterentwickelt wird. Er ist kompatibel zu Turbo Pascal 7 (die Entwicklungsumgebungen

gleichen sich aufs Haar) und daher ideal geeignet, um alte Turbo Pascal-Programme auf Windows oder Linux zu retten.

Der Compiler ist selbst komplett in Pascal geschrieben und beherrscht Optimierungen für aktuelle Prozessoren.

<i>Getestete Version:</i>	1.0.6 (vom 23.4.2002) für Windows
<i>Hersteller:</i>	Carl Eric Codère, Daniël Mantione, Florian Klämpfl u.a.
<i>Lizenz:</i>	GPL
<i>Unterstützte Plattformen:</i>	Windows, DOS (32 Bit ¹), Linux, Amiga und viele weitere
<i>Prozessor-Optimierung:</i>	80386 bis Pentium II, Motorola 680x0
<i>Sprachumfang:</i>	Weitgehend kompatibel zu Delphi 5, mit eigenen Erweiterungen
<i>Webseite:</i>	http://www.freepascal.org/

2.3 Virtual Pascal

Virtual Pascal ist von der Oberfläche her sehr ähnlich zu Free Pascal und Turbo Pascal. Auch Virtual Pascal eignet sich gut zum Retten von alten Turbo Pascal-Programmen. Der Quelltext von Virtual Pascal ist aber nur teilweise erhältlich – vielleicht ein Grund, dass die Weiterentwicklung von Virtual Pascal etwas ins Stocken geraten ist.

<i>Getestete Version:</i>	2.1.243 (vom 5.11.2002) für Windows
<i>Hersteller:</i>	Allan Mertner, Vitaly Miryanov u.a.
<i>Lizenz:</i>	Freeware, teilweise mit Quelltext
<i>Unterstützte Plattformen:</i>	Windows, OS/2, Linux (Beta-Version)
<i>Prozessor-Optimierung:</i>	80386 bis Pentium
<i>Sprachumfang:</i>	Weitgehend kompatibel zu Delphi 2
<i>Webseite:</i>	http://www.vpascal.com/

2.4 GNU Pascal

Der GNU Pascal Compiler setzt auf dem GNU C Compiler auf und profitiert davon einerseits durch dessen gute Optimierung der erzeugten Programme und andererseits durch die Verfügbarkeit auf praktisch allen Plattformen. Der GNU Pascal Compiler unterstützt Pascal-Sprachstandards (z.B. Extended Pascal), um die sich Bor-

¹ DOS ist eigentlich ein 16 Bit-Betriebssystem. Es gibt aber 32 Bit-Programme für DOS! Am Anfang der ausführbaren Datei eines solchen Programms steht ein kleines Hilfsprogramm („Extender“), der den Prozessor in den 32 Bit-Modus schaltet, das Hauptprogramm lädt und dem Programm Routinen bereitstellt, z.B. zur Speicherverwaltung. Solche 32 Bit-Programme sind ein bisschen langsamer als echte Windows-Programme.

land nicht gekümmert hat und daher heute eine geringe Bedeutung haben. GNU Pascal ist ein reines Kommandozeilen-Programm.

<i>Getestete Version:</i>	2.1 (vom 10.5.2002) für Windows (MinGW-Version)
<i>Hersteller:</i>	Jukka Virtanen, Peter Gerwinski, u.v.a.
<i>Lizenz:</i>	GPL
<i>Unterstützte Plattformen:</i>	Alle vom GNU C Compiler unterstützten Plattformen, z.B. DOS (32 Bit), Windows, Linux
<i>Prozessor-Optimierung:</i>	Alle vom GNU C Compiler unterstützten Prozessoren
<i>Sprachumfang:</i>	Weitgehend kompatibel zu Turbo Pascal und zum Sprachkern von Delphi, mit eigenen Erweiterungen
<i>Webseite:</i>	http://www.gnu-pascal.de/

2.5 TMT Pascal

TMT Pascal ist ein Compiler ohne objektorientierte Erweiterungen. Eine Entwicklungsumgebung für Windows wird mitgeliefert.

<i>Getestete Version:</i>	3.90 Lite (vom 24.4.2002) für DOS (32 Bit)
<i>Hersteller:</i>	TMT Development
<i>Lizenz:</i>	Kommerziell, die Version für DOS (Lite) ist kostenlos für den privaten Gebrauch
<i>Unterstützte Plattformen:</i>	Windows, DOS (32 Bit), OS/2
<i>Prozessor-Optimierung:</i>	80386
<i>Sprachumfang:</i>	Weitgehend kompatibel zu Turbo Pascal
<i>Webseite:</i>	http://www.tmt.com/

2.6 Weitere Compiler

Es gibt noch einige weitere Pascal-Compiler, die aber nicht am Vergleich teilnehmen konnten, weil ihr Sprachumfang zu klein ist oder weil sie schon an einfachen Programmen gescheitert sind.

Zwei Beispiele davon sind *Pascal Pro* [2], das Intel-Assemblercode erzeugt und einfache Pascal-Programme compilieren kann und *DPas* [3], das schon an der Springertour mit einem Absturz scheiterte.

Interessant ist *PtoC* [4]: Es übersetzt Pascal-Programme nach C. Die erzeugten C-Programme sind erstaunlich gut zu lesen und sogar die Kommentare werden übernommen. Der Sprachumfang ist aber klein: Nicht mal die „break“-Anweisung kannte PtoC. Daher ist es am besten für Leute geeignet, die komplett von Pascal nach C umsteigen wollen.

3 Konfiguration der Compiler

Die Compiler wurden so konfiguriert, dass der bestmögliche Code erzeugt werden konnte. Alle Testprogramme wurden folgendermassen compiliert:

- Keine Bereichsprüfung (bei Arrays)
- Keine I/O-Prüfung
- Keine Überlaufprüfung (bei Variablen)
- Keine Stackprüfung
- Keine vollständige Auswertung von Booleschen Ausdrücken
- Bestmögliche Optimierung (Variablen in Registern, Codegenerierung für den jüngsten unterstützten Prozessor)

3.1 Datentyp-Unterschiede

3.1.1 „Real“

Ich hatte bei einigen Tests sehr auffällige „Ausreisser“. Diese konnte ich aber schliesslich darauf zurückführen, dass die Compiler „veraltete“ Datentypen anders behandeln. Der Gleitkomma-Datentyp „Real“ ist so ein Fall: Er hat definitionsgemäss 6 Bytes, was sich aber heute auf keinen Prozessor mehr vernünftig abbilden lässt.

Einige Compiler emulieren das mühsam, andere machen aus dem „Real“ einfach ein „Double“ und sind so viel schneller (aber nicht mehr kompatibel). Ich habe alle Testprogramme so abgeändert, dass sie kein „Real“ mehr verwenden.

3.1.2 „LongInt“

Das zweite Datentyp-Problem kennt man in der C-Welt nur zu gut: Die Grösse eines „Integer“ ist nicht genau festgelegt. In der 16 Bit-Welt hat ein Integer bei Pascal 16 Bit. In der 32 Bit-Welt kann man nicht so sicher sein, einige haben den Integer-Bereich auf 32 Bit erweitert, andere sind bei 16 Bit geblieben.

Bisher konnte man auf einen gemeinsamen Nenner kommen, indem man ein „LongInt“ verwendete, was alle Compiler als 32 Bit-Wert verstanden. Aber mit GNU Pascal gibt es jetzt eine Ausnahme: GNU Pascal definiert „LongInt“ als 64 Bit-Wert, was natürlich auf einem heutigen Prozessor zu Geschwindigkeitseinbussen führt. Ich musste daher in alle betroffenen Testprogramme Spezialbehandlungen für GNU Pascal einbauen.

4 Design und Messverfahren

Es wurde ein „Two-Factor Full Factorial Design with Replications“ mit multiplikativem Modell gewählt (siehe Kapitel 22 von [1]). Das heisst, dass alle Testprogramme mit allen Compilern compiliert wurden und die Laufzeit jedes so erzeugten Programms mehrmals gemessen wurde. Dies geschah mit einem kleinen Hilfsprogramm, das die Laufzeit in Millisekunden misst.

Als Testsystem diente ein PC mit 900 MHz Pentium III-Prozessor, 32 KB L1-Cache, 256 KB L2-Cache, 256 MB RAM und Windows 2000.

Die im folgenden Abschnitt graphisch dargestellten Laufzeiten sind die Mittelwerte der wiederholten Messungen.

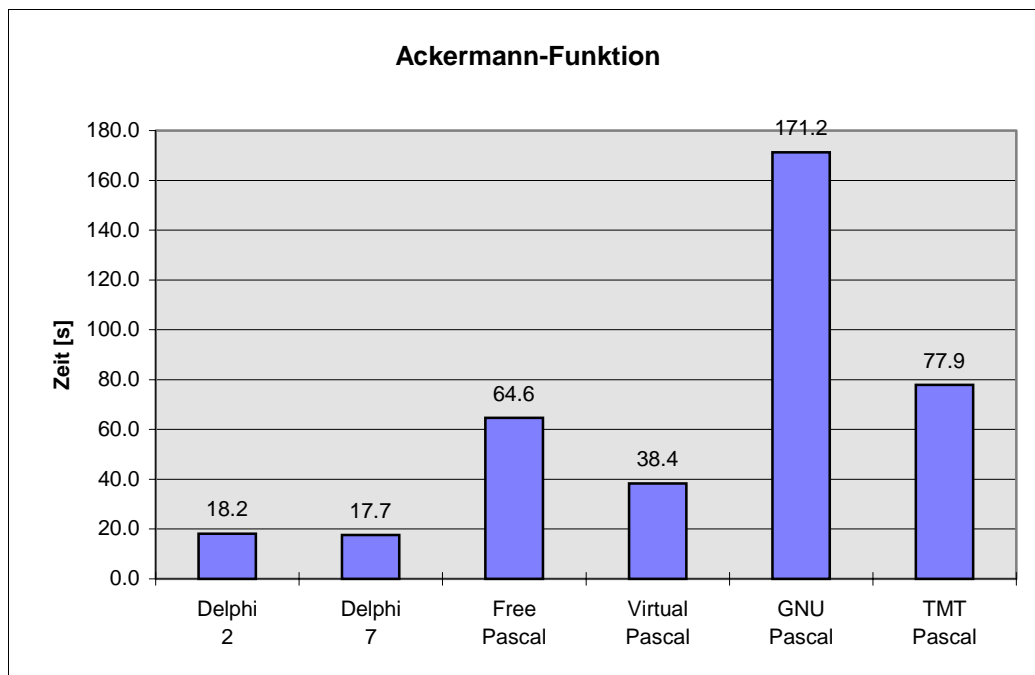
5 Die Testprogramme (Workloads)

5.1 Ackermann-Funktion

Beschreibung des Programms

Die Ackermann-Funktion ist eine stark rekursive Funktion, mit der man den Overhead der Compiler bei einem Funktionsaufruf gut messen kann. Die Aufgabe war, *Ackermann(3,12)* zu bestimmen.

Messresultate



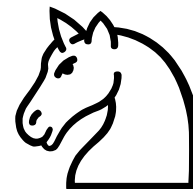
GNU Pascal scheint einen riesigen Overhead bei Funktionsaufrufen zu haben². Ohne dieses schlechte Resultat hätte GNU Pascal gesamthaft gesehen bedeutend besser abgeschnitten.

Die beiden Delphis haben diese Aufgabe ausgezeichnet gemeistert: Sie sind doppelt so schnell wie die beste Konkurrenz.

5.2 Springertour

Beschreibung des Programms

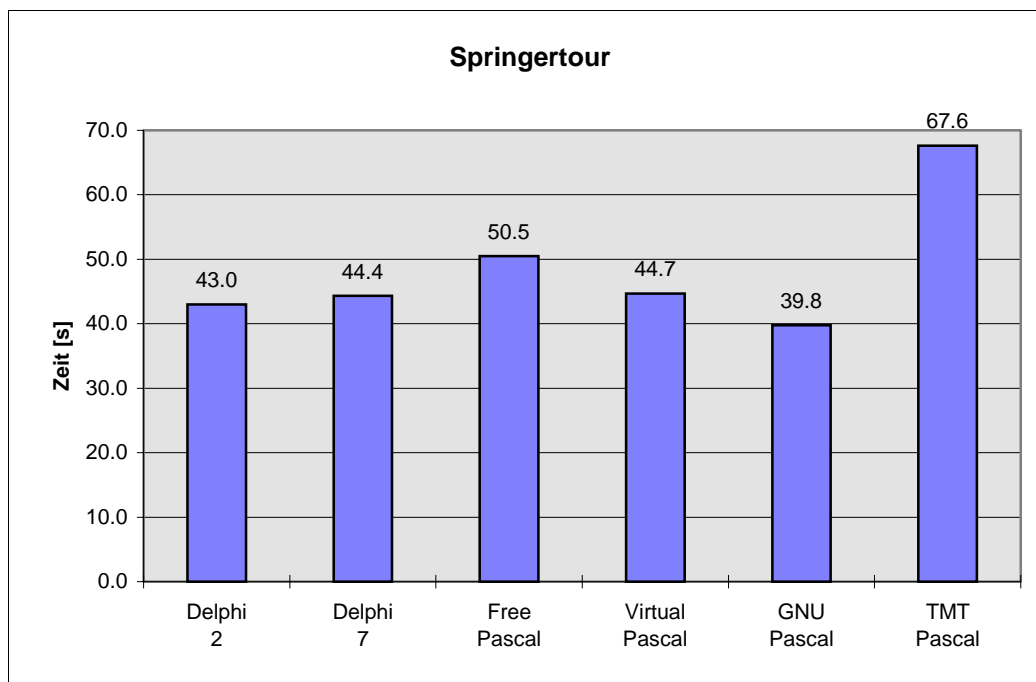
Eine „Springertour“ ist eine Zugfolge eines Springers auf einem Schachbrett, in der die Figur jedes Feld einmal besucht und am Schluss wieder auf dem Ausgangsfeld steht.



Das Programm setzt eine gute Heuristik ein, sodass ich eine anspruchsvolle Aufgabe wählen konnte: 100'000 Springertouren auf einem 10x10-Brett mussten gefunden werden.

Mit dem Springertour-Programm kann man die Optimierung von rekursiven Prozeduren und des Array-Zugriffs testen.

Messresultate



² Natürlich nicht der GNU Pascal Compiler selbst, sondern das von ihm erzeugte Programm. Der Einfachheit halber habe ich darauf verzichtet, immer explizit zu schreiben, dass die Laufzeiten der generierten Programme gemeint sind und *nicht* die Laufzeiten der Compiler beim Compilieren der Programme.

Hier liegen die Compiler erstaunlicherweise recht nah beisammen. GNU Pascal ist sogar am schnellsten, obwohl es bei der Ackermann-Funktion sehr langsam war.

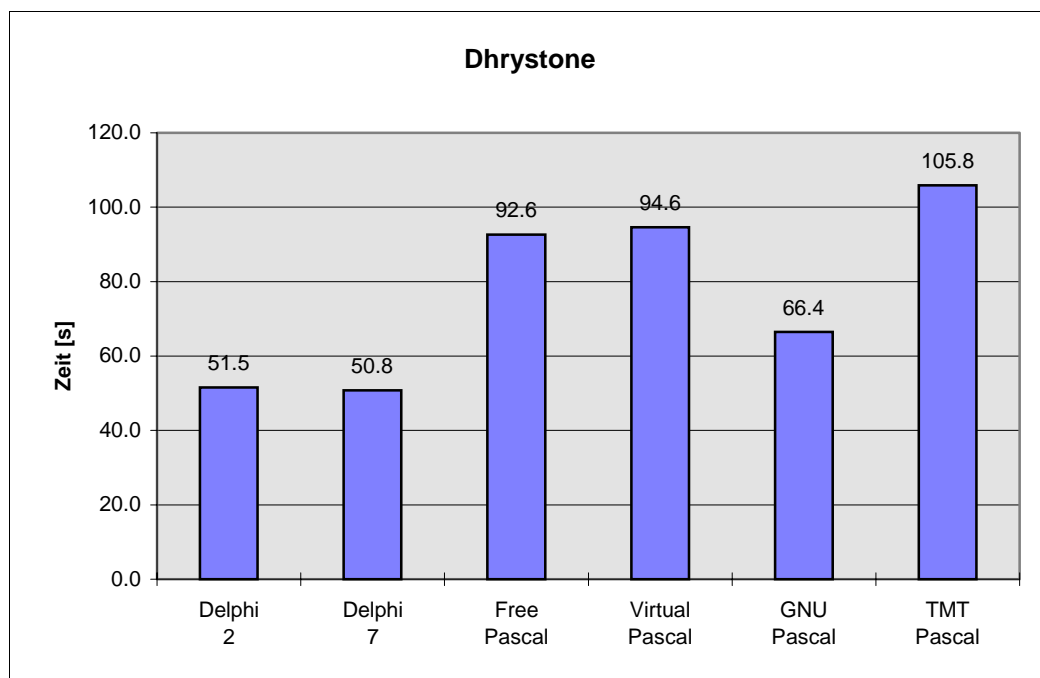
5.3 Dhrystone

Beschreibung des Programms

Dhrystone [5] ist ein klassischer Integer- und Zeichenketten-Benchmark von Reinhold Weicker mit Entstehungsjahr 1984. Er wurde vor allem gebraucht, um unterschiedliche Computersysteme zu vergleichen.

Die Aufgabe war, den Dhrystone-Benchmark 100 Millionen mal zu durchlaufen.

Messresultate



Der klare Testsieger ist Delphi, der sich gegen GNU Pascal durchsetzen konnte. Free Pascal und Virtual Pascal liegen fast gleichauf.

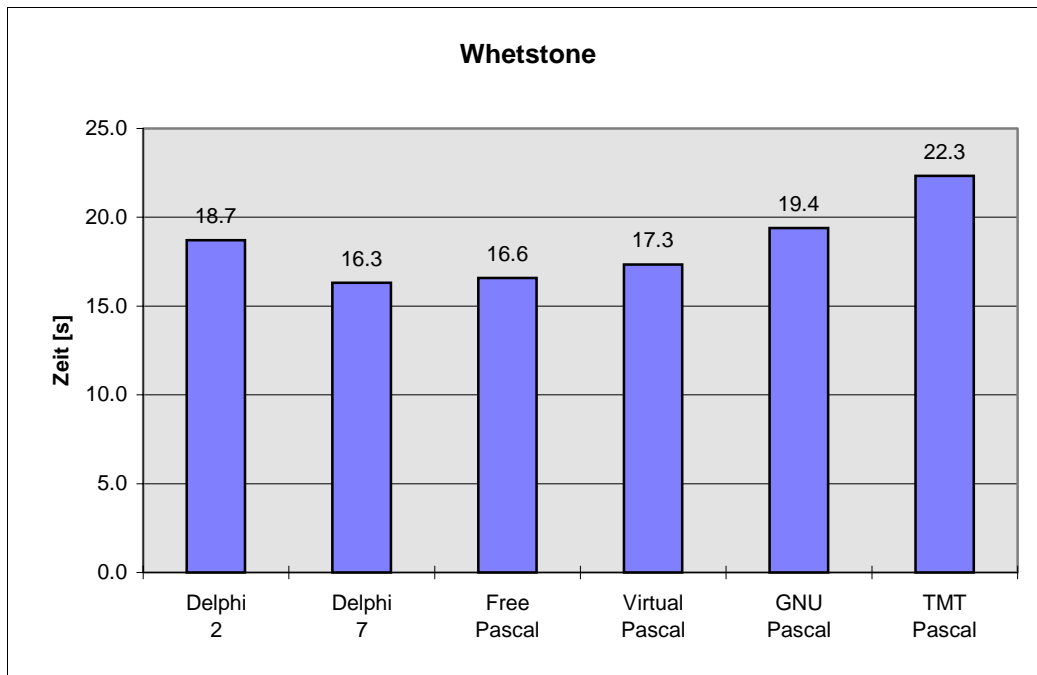
5.4 Whetstone

Beschreibung des Programms

Der Whetstone-Benchmark [6] ist sogar noch älter als der Dhrystone-Benchmark – nämlich aus dem Jahr 1972. Er beinhaltet sowohl Ganzzahl- als auch Gleitkomma-Arithmetik.

Die Aufgabe bestand darin, 10 Milliarden „Whetstones“ zu berechnen. („Whetstones“ ist die Einheit, die vom Whetstone-Benchmark benutzt wird. Das entspricht keinesfalls 10 Milliarden Durchgängen.)

Messresultate



Zum ersten mal wird ein grösserer Unterschied zwischen Delphi 2 und 7 sichtbar. GNU Pascal ist hier schlechter als Delphi, dafür sind Free Pascal und Virtual Pascal gut platziert.

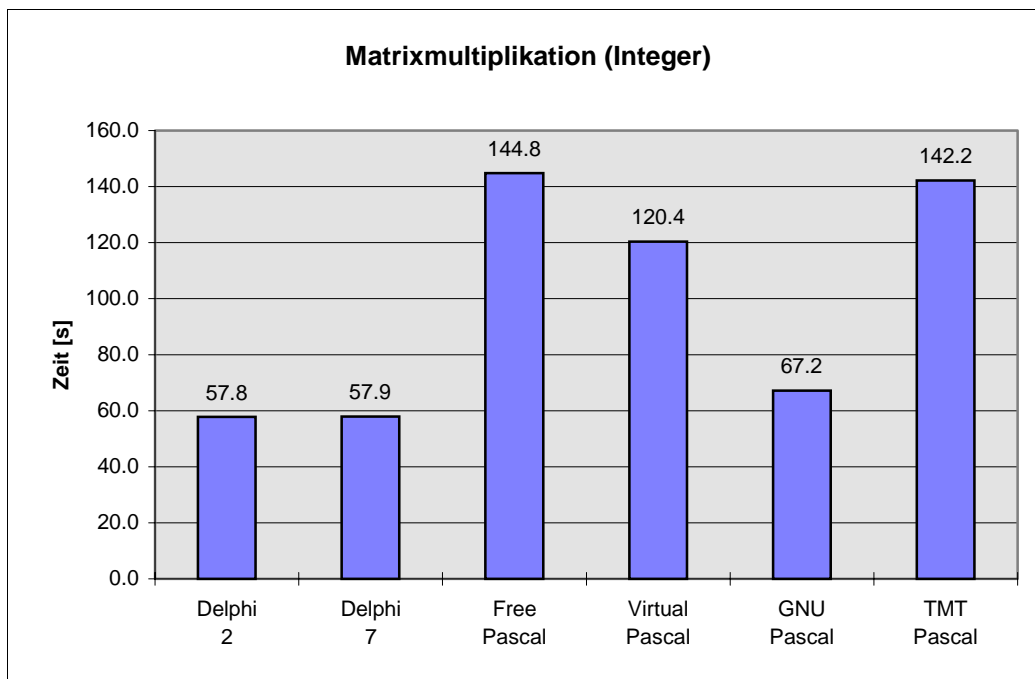
5.5 Matrixmultiplikation (Integer)

Beschreibung des Programms

Zwei 1500 x 1500-Matrizen mussten multipliziert werden. Neben der Integer-Arithmetik testet dieser Workload vor allem den Speicherzugriff und die Schleifenoptimierung.

Die Matrizen wurden mit einem Zufallszahlengenerator gefüllt, den ich in das Programm eingebaut habe. Das Füllen der Matrizen dauerte nur ein Bruchteil der gesamten Laufzeit und wurde vernachlässigt.

Messresultate



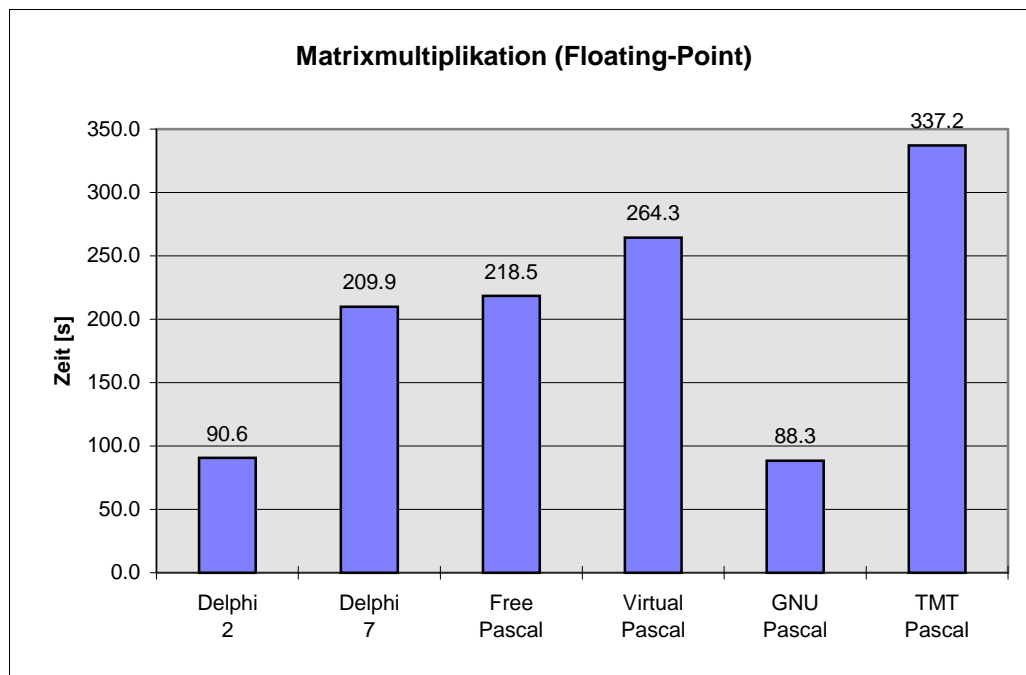
Wieder stehen die Delphis an der Spitze. Free Pascal fällt diesmal hinter TMT Pascal zurück. GNU Pascal hat ein solides Ergebnis.

5.6 Matrixmultiplikation (Floating-Point)

Beschreibung des Programms

Dasselbe nochmal, aber mit Floating-Point-Arithmetik (genauer gesagt mit „Double“-Arithmetik).

Messresultate



Das Ergebnis von Delphi 7 ist äusserst enttäuschend. Dass sich Delphi 7 von seinem Urahn Delphi 2 derart deklassieren lässt, ist erstaunlich. Dieses schlechte Resultat hat Delphi 7 gesamthaft stark zurückgeworfen.

Ein kleiner Test mit Delphi 6, dem Vorgänger von Delphi 7, zeigt auf, dass Delphi 6 noch mithalten konnte und erst Delphi 7 so langsam ist.

Sieger ist GNU Pascal, dem Floating Point-Berechnungen offenbar sehr entgegenkommen. Auch Free Pascal konnte sich gegenüber der Integer-Variante verbessern. TMT Pascal ist zurückgefallen.

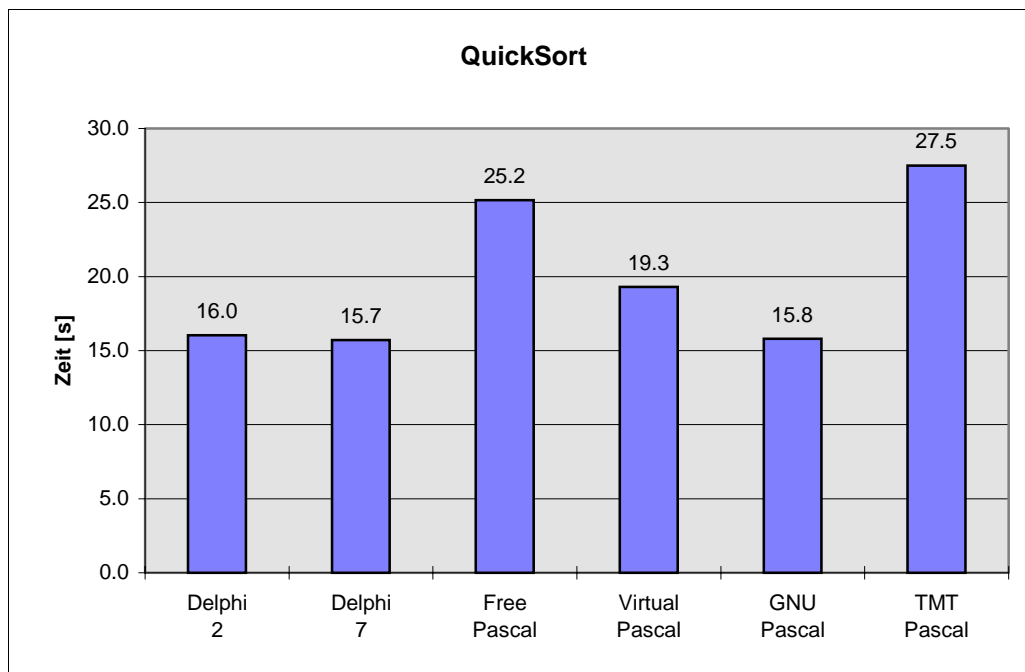
5.7 QuickSort

Beschreibung des Programms

Ein Array mit 30 Millionen Integer-Zahlen musste sortiert werden. Es wurde wieder mit einem in das Programm eingebauten Zufallszahlengenerator gefüllt.

Bei diesem Test sind Speicherzugriff und Rekursion die kritischen Punkte.

Messresultate



Gleich drei Compiler liegen praktisch gleichauf, mit leichten Vorteilen für Delphi 7. TMT Pascal kommt auch hier nicht an die anderen heran.

5.8 ZipInfo (Zip-Datei testen)

Beschreibung des Programms

Diese Aufgabe für die Compiler ist am praxisnahsten: Das Testprogramm entpackt eine Zip-Datei. Damit wird vieles getestet: Datei- und Speicherzugriff, Bit-Operationen, Integer-Arithmetik, Schleifen, Bäume und wahrscheinlich noch viel mehr.

Als Grundlage diente „Chief's Unzip“ [7], eine Pascal-Portierung des bekannten „unzip“-Programms von InfoZip [8]. Chief's Unzip enthält Entpackroutinen, die man entweder direkt oder als Bibliothek (DLL-Datei) in ein eigenes Programm einbinden kann. Im Test wurde die erste Variante verwendet.

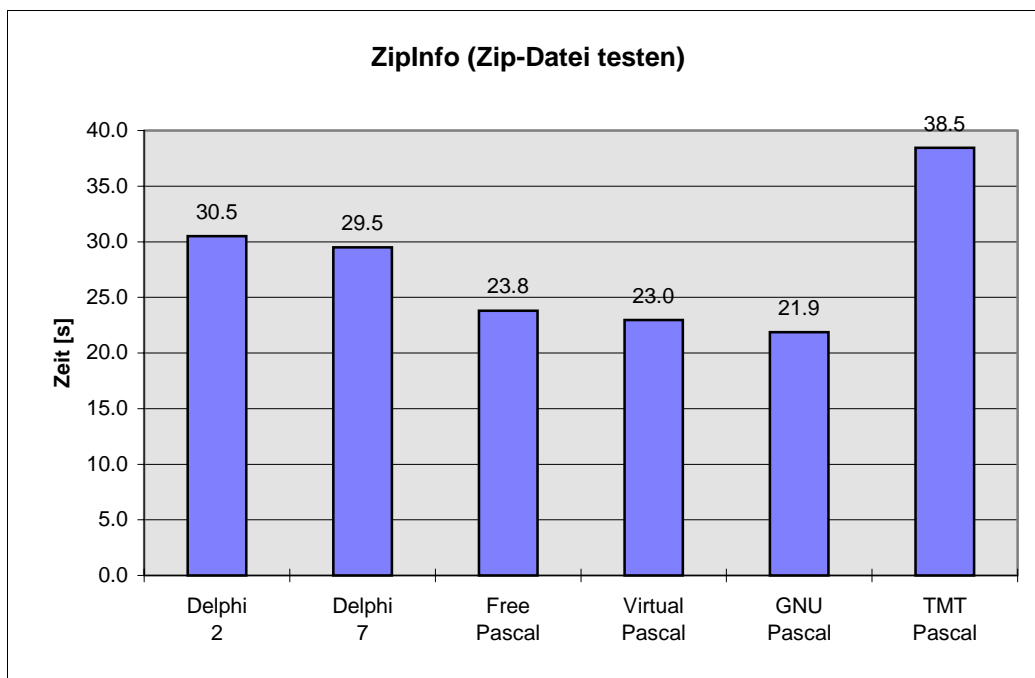
Damit die Festplatte beim Testen keinen Engpass bildet, werden die entpackten Daten nicht auf die Festplatte geschrieben. Das Testprogramm entpackt die Daten daher nur im Arbeitsspeicher (wobei es natürlich laufend alte Daten mit neuen überschreibt). Die Prüfsumme (CRC) wird mitberechnet.

Die Compilierung des Testprogramms war kein leichtes Unterfangen: Zum Zeitpunkt, als der Support für TMT Pascal und GNU Pascal in „Chief's Unzip“ einprogrammiert wurde, hatten diese beiden Compiler offensichtlich Mühe mit gewissen Pascal-Konstrukten. Mittels bedingter Compilierung (`{ $ifdef }`, `{ $endif }`) hat man eine Extrawurst für diese eingebaut. Mittlerweile haben sie aber dazugelernt – und prompt

verhinderten jetzt diese „Hacks“, dass das Programm bei diesen Compilern fehlerfrei lief.

Die Zip-Datei, die zu testen war, enthält 304 MB an Daten und ist gepackt 216 MB gross. Sie enthält 257 Dateien, davon ist eine Datei mit 57 MB sehr gross, 5 Dateien sind zwischen 5 und 10 MB, und der Rest zwischen 0,5 und 1,5 MB.

Messresultate



GNU Pascal hat sich auch in diesem Praxistest bewährt. Delphi ist nicht so schnell wie sonst und muss dem Duo Free Pascal/Virtual Pascal den zweiten und dritten Platz überlassen. Das Schlusslicht ist wieder TMT Pascal.

6 Messresultate

In der folgenden Tabelle sind zunächst einmal alle Messresultate aufgeführt, die nachher noch genauer analysiert werden. Alle Laufzeiten wurden vier mal gemessen und sind in Millisekunden angegeben.

7 Analyse der Messresultate

	Delphi 2	Delphi 7	Free Pascal	Virtual Pascal	GNU Pascal	TMT Pascal
Ackermann	18'276	17'755	64'413	38'075	171'016	76'661
	18'136	17'705	64'793	38'676	170'996	78'783
	18'126	17'696	64'984	38'786	171'036	78'583
	18'086	17'505	64'393	37'915	171'857	77'402
Springertour	43'272	44'383	50'483	44'714	39'808	67'627
	42'872	44'353	50'473	44'694	39'757	67'627
	42'902	44'343	50'462	44'684	39'737	67'617
	42'901	44'344	50'462	44'695	39'748	67'627
Dhystone	51'604	50'793	92'663	94'636	66'506	105'842
	51'514	50'733	92'593	94'616	66'445	105'812
	51'514	50'743	92'613	94'606	66'416	105'903
	51'544	50'733	92'613	94'626	66'406	105'842
Whetstone	19'048	16'343	16'544	17'335	19'448	22'342
	18'587	16'304	16'814	17'325	19'348	22'322
	18'597	16'303	16'483	17'345	19'368	22'332
	18'577	16'313	16'494	17'325	19'378	22'322
Matrixmult. Integer	57'863	57'913	144'798	120'403	67'257	142'285
	57'803	57'844	144'748	120'343	67'286	142'184
	57'783	57'833	144'778	120'323	67'176	142'174
	57'783	57'953	144'728	120'353	67'177	142'185
Matrixmult. Float	90'771	209'912	218'374	264'390	88'357	337'305
	90'590	210'132	218'634	264'300	88'237	337'215
	90'591	209'811	218'484	264'280	88'347	337'235
	90'580	209'892	218'314	264'341	88'247	337'225
QuickSort	16'244	15'793	25'187	19'308	15'873	27'500
	15'933	15'693	25'146	19'308	15'782	27'480
	15'933	15'683	25'146	19'287	15'772	27'479
	16'013	15'682	25'147	19'307	15'782	27'490
ZipInfo	31'465	29'462	23'734	22'893	21'832	38'435
	30'203	29'423	23'914	23'073	21'921	38'495
	30'133	29'532	23'734	22'983	21'902	38'435
	30'234	29'562	23'825	22'903	21'861	38'445

Die Mittelwerte der Messresultate (die schon in den einzelnen Diagrammen zu sehen waren) sind:

	Delphi 2	Delphi 7	Free Pascal	Virtual Pascal	GNU Pascal	TMT Pascal
Ackermann	18'156	17'665	64'646	38'363	171'226	77'857
Springertour	42'987	44'356	50'470	44'697	39'763	67'625
Dhystone	51'544	50'751	92'621	94'621	66'443	105'850
Whetstone	18'702	16'316	16'584	17'333	19'386	22'330
Matrixmult. Integer	57'808	57'886	144'763	120'356	67'224	142'207
Matrixmult. Float	90'633	209'937	218'452	264'328	88'297	337'245
QuickSort	16'031	15'713	25'157	19'303	15'802	27'487
ZipInfo	30'509	29'495	23'802	22'963	21'879	38'453

7 Analyse der Messresultate

In der Analyse benutzt man üblicherweise ein additives Modell, in dem sich die einzelnen Effekte aufaddieren. Das ist aber hier nicht der Fall: Die Laufzeit Y eines Programms W ist nämlich

$$Y = C \cdot W,$$

wobei

7 Analyse der Messresultate

C = „Güte“ des Compilers = Anzahl Operationen pro Sekunde und

W = Workload = Anzahl Operationen eines Testprogramms.

Man sieht, dass sich die Effekte multiplizieren statt addieren. Um trotzdem ein additives Modell verwenden zu können, transformiert man die Messdaten und nimmt den Logarithmus davon. Das funktioniert, weil der Logarithmus einer Multiplikation eine Addition ist. (Mehr dazu im Buch [1], Kapitel 18.8)

Logarithmische Transformation der Mittelwertstabelle (10er-Logarithmus):

	Delphi 2	Delphi 7	Free Pascal	Virtual Pascal	GNU Pascal	TMT Pascal
Ackermann	4.259	4.247	4.811	4.584	5.234	4.891
Springertour	4.633	4.647	4.703	4.650	4.599	4.830
Dhystone	4.712	4.705	4.967	4.976	4.822	5.025
Whetstone	4.272	4.213	4.220	4.239	4.287	4.349
Matrixmult. Integer	4.762	4.763	5.161	5.080	4.828	5.153
Matrixmult. Float	4.957	5.322	5.339	5.422	4.946	5.528
QuickSort	4.205	4.196	4.401	4.286	4.199	4.439
ZipInfo	4.484	4.470	4.377	4.361	4.340	4.585

7.1 Berechnung der Effekte

Im folgenden gehe ich nach dem Buch vor. Als erstes berechnet man, welchen Effekt die Compiler und Workloads haben:

	Delphi 2	Delphi 7	Free Pascal	Virtual Pascal	GNU Pascal	TMT Pascal	Zeilen-summe	Zeilen-mittel	Zeilen-effekt
Ackermann	4.259	4.247	4.811	4.584	5.234	4.891	112.102	4.671	-0.006
Springertour	4.633	4.647	4.703	4.650	4.599	4.830	112.253	4.677	0.001
Dhystone	4.712	4.705	4.967	4.976	4.822	5.025	116.830	4.868	0.191
Whetstone	4.272	4.213	4.220	4.239	4.287	4.349	102.318	4.263	-0.413
Matrixm. Integer	4.762	4.763	5.161	5.080	4.828	5.153	118.985	4.958	0.281
Matrixm. Float	4.957	5.322	5.339	5.422	4.946	5.528	126.059	5.252	0.576
QuickSort	4.205	4.196	4.401	4.286	4.199	4.439	102.901	4.288	-0.389
ZipInfo	4.484	4.470	4.377	4.361	4.340	4.585	106.467	4.436	-0.241
<i>Spaltensumme</i>	36.285	36.563	37.977	37.598	37.255	38.800	224.478		
<i>Spaltenmittel</i>	4.536	4.570	4.747	4.700	4.657	4.850		4.677	
<i>Spalteneffekt</i>	-0.1410	-0.1063	0.0705	0.0232	-0.0197	0.1734			

Beispiele zur Interpretation dieser Tabelle:

Delphi 2 braucht 0.141 „logarithmische Zeit“ weniger (27,7% weniger) als ein mittlerer Compiler.

Die Ackermann-Funktion braucht 0.006 „logarithmische Zeit“ weniger (1,4% weniger) als ein mittlerer Workload.

Mit diesen Daten kann man die **Rangliste** machen: Gewonnen hat Delphi 2. Zweiter ist Delphi 7, gefolgt von GNU Pascal. Virtual Pascal ist besser als Free Pascal und TMT Pascal landet auf dem letzten Platz.

7.2 Interaktionen der Compiler mit den Testprogrammen

Die folgende Tabelle zeigt auf, wieweit Compiler und Testprogramme „interagieren“. Diese Werte berechnet man, indem man die Differenz zwischen tatsächlich gemessenem Wert und vorausgesagtem Wert berechnet:

$$\text{Interaktion}(i, j) = \text{Wert}(i, j) - (\underbrace{\text{Mittelwert}}_{4.677} + \text{Zeileneffekt}(i) + \text{Spalteneffekt}(j))$$

	Delphi 2	Delphi 7	Free Pascal	Virtual Pascal	GNU Pascal	TMT Pascal
Ackermann	-0.271	-0.318	0.069	-0.110	0.582	0.047
Springertour	0.097	0.076	-0.045	-0.050	-0.058	-0.020
Dhystone	-0.015	-0.056	0.028	0.085	-0.026	-0.017
Whetstone	0.150	0.056	-0.114	-0.048	0.044	-0.088
Matrixmult. Integer	-0.055	-0.089	0.132	0.100	-0.110	0.022
Matrixmult. Float	-0.154	0.176	0.016	0.147	-0.287	0.102
QuickSort	0.058	0.015	0.043	-0.025	-0.069	-0.022
ZiplInfo	0.189	0.140	-0.130	-0.098	-0.076	-0.025

Beispiel zur Interpretation dieser Tabelle:

Die Ackermann-Funktion auf Delphi 2 braucht 0.271 „logarithmische Zeit“ weniger als ein mittlerer Workload auf Delphi 2 *und* die Ackermann-Funktion auf Delphi 2 braucht 0.271 „logarithmische Zeit“ weniger als die Ackermann-Funktion auf einem mittleren Compiler.

7.3 Verursacher der Variation

Variation begründet durch den Compiler: 8.34%

Variation begründet durch den Workload: 77.08%

Variation begründet durch die Interaktionen: 14.57%

Variation begründet durch die Fehler: 0.003%

(Kontrollwerte: SST = 25.759, SSA = 2.148, SSB = 19.856, SSAB = 3.754, SSE = 0.000691)

7.4 Analyse der Varianz (ANOVA)

(Kontrollwerte: MSA = 0.4296, MSB = 2.837, MSAB = 0.1073, MSE = 0.000004798)

8 Schlussfolgerungen

	<i>Freiheitsgrade</i>	<i>Berechneter F-Wert</i>	<i>Tabellierter F-Wert ($\alpha = 0.01$)</i>
Compiler	5	89'536.0	≈ 3
Workloads	7	591'140.9	≈ 3
Interaktionen	35	22'354.2	≈ 2

Standardabweichung der Fehler: $s_e = \sqrt{\text{MSE}} = 0.00219$

Aus dieser Tabelle kann man ablesen, dass die Effekte statistisch signifikant sind. Das ist der Fall, weil die berechneten F-Werte grösser als die tabellierten F-Werte sind.

7.5 Vertrauensintervalle für die Effekte

<i>Parameter</i>	<i>Mittlerer Effekt</i>	<i>Standardabweichung</i>	<i>95%-Vertrauensintervall</i>
Mittelwert	4.6766	0.0001581	(4.6763, 4.6769)
Delphi 2	-0.1410	0.0003535	(-0.1469, -0.1403)
Delphi 7	-0.1063	0.0003535	(-0.1070, -0.1056)
Free Pascal	0.0705	0.0003535	(0.0698, 0.0712)
Virtual Pascal	0.0232	0.0003535	(0.0225, 0.0239)
GNU Pascal	-0.0197	0.0003535	(-0.0204, -0.0190)
TMT Pascal	0.1734	0.0003535	(0.1727, 0.1741)

Die Vertrauensintervalle gehen dank der 4 Wiederholungen der Messungen über einen sehr kleinen Bereich. Alle Effekte sind signifikant, weil die Null in keinem Vertrauensintervall enthalten ist.

8 Schlussfolgerungen

Delphi 2 ist der beste der getesteten Pascal-Compiler. Delphi 7 und GNU Pascal haben beide durch je einen Patzer ein schlechteres Resultat eingefahren.

GNU Pascal hat überrascht: Der GNU Pascal Compiler mit dem C-Compiler im Hintergrund birgt einiges an Potential und liefert Resultate, die nah an jene von Delphi herankommen. Und es gibt noch bessere C-Compiler als der GNU C Compiler, folglich müsste man mit dem Umweg über einen C-Compiler sogar noch bessere Resultate erreichen können.

Delphi 7 konnte sich nicht gross von Delphi 2 absetzen. Offensichtlich war der Delphi-Compiler schon in der Version 2 ausgezeichnet, aber es hat sich seither nicht mehr allzuviel getan.

Free Pascal und Virtual Pascal haben im Test schlechter abgeschnitten als Delphi und GNU Pascal, aber dass das nicht unbedingt für die Praxis zutreffen muss, zeigt der letzte Test mit dem Entpacken der Zip-Datei: Hier liegen die beiden vorne.

8 Schlussfolgerungen

Von TMT Pascal muss man leider abraten: Es war immer letzter oder zweitletzter. Statt TMT Pascal setzt man besser einen der Gratis-Compiler ein.

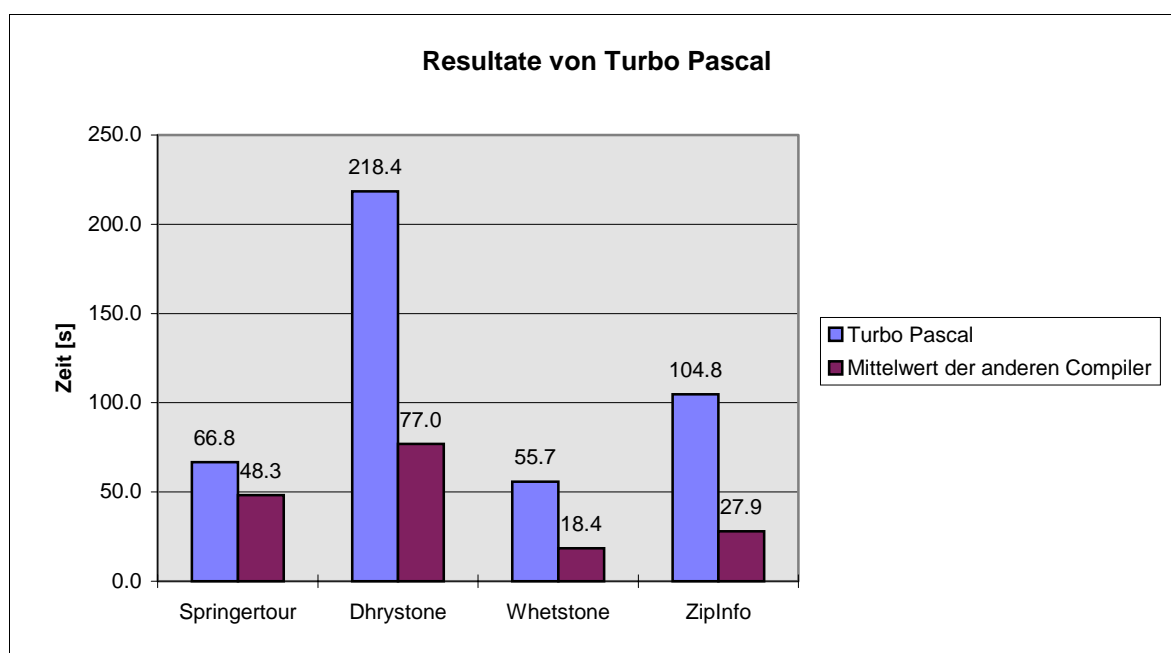
Die Analyse der Messdaten hat gezeigt, dass die Fehler sehr klein und die Resultate signifikant sind. Zwei Wiederholungen der Messungen statt vier hätten höchstwahrscheinlich auch gereicht.

Anhang: Turbo Pascal

Der Urahn aller Pascal-Compiler ist Turbo Pascal. Er ist wohl der Grund dafür, dass sich die Sprache Pascal überhaupt als eine der „grossen Programmiersprachen“ durchsetzen konnte. Heute ist Turbo Pascal überholt und man sollte stattdessen Delphi oder Free Pascal/Virtual Pascal einsetzen.

<i>Getestete Version:</i>	7.01 (vom 11.3.1993)
<i>Hersteller:</i>	Borland
<i>Lizenz:</i>	Kommerziell, heute aber gratis erhältlich
<i>Unterstützte Plattformen:</i>	DOS (16 Bit)
<i>Prozessor-Optimierung:</i>	8086 bis 80286
<i>Sprachumfang:</i>	Grundlegendes Pascal mit kleinen Erweiterungen
<i>Webseite:</i>	Englische Versionen bis 5.5: http://bdn.borland.com/museum/ Französische Version 7.01: http://www.inprise.fr/download/compileurs/

Wie gesagt, konnte Turbo Pascal bei diesem Vergleich nicht „offiziell“ teilnehmen. Das aus dem Grund, dass es bei einigen Tests gar nicht möglich war, diese mit Turbo Pascal durchzuführen, da bei ihm Arrays auf 64 KB und der ganze Speicher auf 640 KB beschränkt sind (diese Beschränkungen gelten für alle alten DOS-Programme). Zudem ist Turbo Pascal als 16 Bit-Compiler massiv benachteiligt: Mehrere Maschinenbefehle werden benötigt, um zwei 32 Bit-Zahlen zu multiplizieren.



Literaturverzeichnis

- [1] Raj Jain: „The Art of Computer Systems Performance Analysis“, John Wiley & Sons, 1991
- [2] Homepage von „Pascal Pro“:
<http://www.fortunecity.com/skyscraper/sql/39/>
- [3] „DPas“ ist ein Programm von Dieter Pawelczak:
<http://people.freenet.de/dieterp/>
- [4] „PtoC“ ist ein Programm von Konstantin Knizhnik:
<http://www.garret.ru/~knizhnik/>
- [5] Den Dhrystone-Benchmark kann man hier herunterladen:
<http://www.netlib.org/benchmark/dhry-pascal>
- [6] Den Whetstone-Benchmark kann man hier herunterladen:
<http://www.programmersheaven.com/zone24/cat31/3980.htm>
- [7] „Chief's Unzip“ ist erhältlich auf der Homepage von Abimbola Olowofoyeku:
http://website.lineone.net/~african_chief/
Die eigentliche Portierung auf Pascal wurde von Christian Ghisler durchgeführt:
<http://www.ghisler.com/>
- [8] Homepage der InfoZip-Gruppe:
<http://www.info-zip.org/>