

# AlignTest:

Ein Testprogramm zur prozessorspezifischen X86-Codeoptimierung unter Free Pascal / Lazarus

**Systemvoraussetzungen:**

- **x86-kompatibler Prozessor (32- oder 64 Bit)**
- **Windows / Linux / Mac**
- **Installiertes FPC 2.6.2 / Lazarus 1.2.0**

Dieses Programm entstand aus der Arbeit an der Optimierung kurzer Funktionen, die als "*Arbeitspferde*" für die Bearbeitung großer (Short-) String-Datensätze dienen. Also vor allem mit den *Compare*-Methoden, die man zum Sortieren braucht, und den *Pos*-Funktionen, mit denen man Texte durchsuchen kann. Im Zuge dieser Arbeit begann ich schließlich damit, Entwürfe hierfür in Assembler zu schreiben und deren Laufzeitverhalten zu testen.

Dabei bin ich darauf gestoßen, dass es unheimlich schwierig ist, vorherzusagen, welche Lösungen richtig schnell funktionieren und welche nicht. Die Zeiten, als man den Zeitbedarf von Assemblercode anhand von Handbuchtabeln einfach addieren konnte, sind längst vorbei: Moderne Prozessoren versuchen, den weiteren Programmablauf *vorherzusagen* und im Hintergrund Vorarbeit zu leisten, indem sie z.B. später erst anstehende Anweisungen schon mal vorsorglich nebenher abarbeiten, um schon fertige Ergebnisse parat zu haben, wenn diese dann gebraucht werden. Das Ganze ist ein Spiel mit *Wahrscheinlichkeiten*, denn solange die Vorhersagen, welchem Entscheidungspfad das Programm demnächst wohl folgen würde, zutreffen, funktioniert das Ganze wunderbar – sobald diese *Predictions* aber in die falsche Richtung gehen, war der ganze Hintergrundprozeß für die Katz und muss neu aufgebaut werden. Das kostet dann wiederum richtig viel der zuvor mühsam eingesparten Laufzeit.

Um nun optimalen, schnellen Code zu schreiben, muss man als Programmierer nun seinerseits vorhersehen, wie der Prozessor an dieser oder jener Stelle wohl „ticken“ würde. Das hat sich als außerordentlich schwierig herausgestellt, und sehr viel von dem, was man darüber im Internet nachlesen kann, hat sich dabei leider auch als durchaus fragwürdig erwiesen. Manchmal nützt es z.B., sinnlos scheinende *Verzögerungen* (sogenannte NOPs, No-Operation-Anweisungen) in den Code einzubauen. Im Internet wird seit einiger Zeit das seltsame Phänomen diskutiert, dass sogar *wahllos eingestreute NOP's* Programme beschleunigen können – bislang konnte m.W. aber noch niemand eine schlüssige Erklärung für diesen Effekt vorbringen.

Ich selbst bin in meiner kleinen Welt der *Compare*- und *Pos*-Funktionen von einer Überraschung in die nächste gefallen, z.B. als ich den Auswirkungen des sogenannten Alignments nachging, der Anordnung von Prozessoranweisungen und Sprungstellen relativ zu 16-, 32- und 64-Byte-Blöcken innerhalb des Speichers bzw. Prozessor-Caches. Man könnte das alles als Kleinkrämerei abtun, aber die Erfahrung zeigt, dass es hier um *Laufzeitunterschiede bis zu einer Größenordnung von 1:3* geht – für eine Compare-Funktion, die beim Sortieren von Datensätzen millionenfach durchlaufen wird, sind das Welten, die der Anwender sehr wohl zu spüren bekommt. Sich mit diesem Thema zu beschäftigen lohnt sich also durchaus!

Und es gab noch mehr Überraschungen – zu den genauen technischen Details sei an dieser Stelle auf die Datei *Technische\_Details.pdf* verwiesen. Hier nur soviel: Ich habe daher beschlossen, sozusagen eine weitere Vergrößerungslinse an mein Mikroskop zu schrauben und eine denkbar einfache Funktion (die einen ShortString nach der Position eines Chars durchsucht - also das, was die Free-Pascal-Funktion *System.Pos* in der Variante (Char; ShortString) macht) in einer Reihe von Varianten durchzutesten. Dabei ist das hier präsentierte Testprogramm entstanden.

Es hat sich dann gezeigt, dass keineswegs alles, was auf meinem AMD-64-Desktop gut funktioniert, auch auf meinem 32-bit Laptop (mit einem Intel Celeron) richtig die geweckten Erwartungen erfüllt. Manches dann aber schon. Damit bin ich nun an einem Punkt angelangt, **an dem ich Euch um Eure (möglichst zahlreiche!) Mithilfe bitten möchte!**

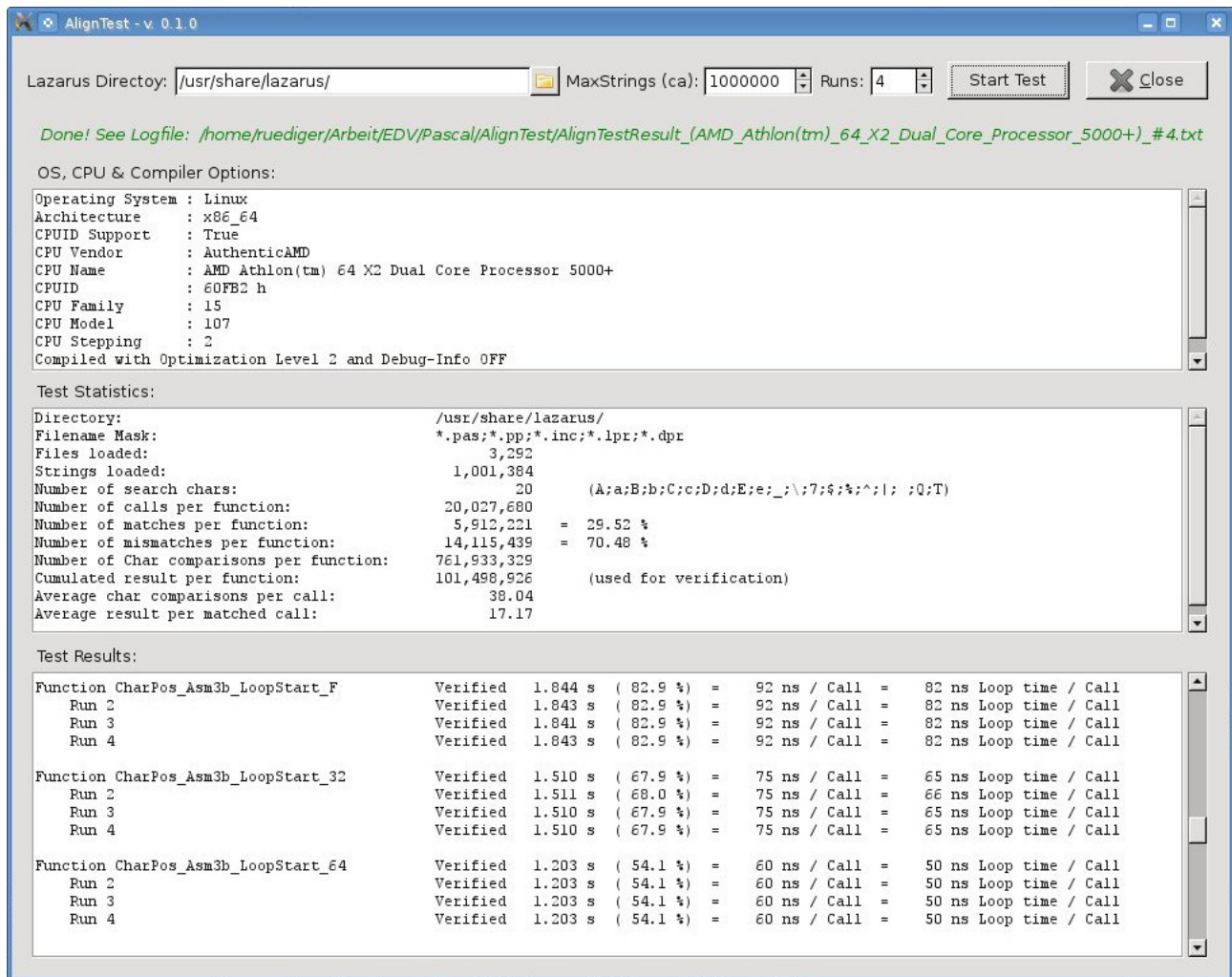
Die Frage ist nämlich, ob es genügend *Gemeinsamkeiten* im Mikro-Laufzeitverhalten unterschiedlichster x86-Prozessoren gibt, aus denen sich Erkenntnisse in Bezug auf Code-Optimierung gewinnen lassen, oder ob die *Unterschiede* überwiegen.

**Das Ziel dieses Testprogramms ist daher, Ergebnisdaten von *möglichst vielen verschiedenen Prozessoren* zu sammeln und diese miteinander zu *vergleichen*.**

Dabei geht es nicht um Geschwindigkeitsunterschiede zwischen einzelnen Prozessoren insgesamt, sondern darum, zu untersuchen, *welche alternativen Programmiermethoden bzw. Codesegmente unter welchen Prozessoren relativ (!) schnell laufen und welche Prozessoren von ihnen ausgebremst werden*. Die Frage ist also nicht: Welches ist der beste Prozessor, sondern: *Welches ist der beste Code* - für diese spezielle CPU oder vielleicht auch für die meisten bzw. alle CPU's? Gibt es denn soetwas wie den „*besten Code*“ für alle geläufigen x86-Prozessoren überhaupt?

Da einzelne Prozessorfamilien und -typen ebenfalls ständig weiterentwickelt werden (in verschiedene „*Modelle*“ unterteilt und unterhalb der Modellebene in das sogenannte „*Stepping*“), interessieren hier übrigens nicht nur *CPU-Typen*, sondern auch deren *Baureihen und Revisionen*.

Nun zum Testprogramm AlignTest selbst:



AlignTest vermisst und protokolliert die jeweilige Laufzeit einer bewusst äußerst einfach gehaltenen Pos(Char; ShortString)-Funktion, die einen ShortString nach dem ersten Vorkommen eines Chars durchsucht. Der Kern dieser Funktion besteht aus lediglich 6 Assembler-Zeilen. AlignTest vermisst nun das Zeitverhalten von mehr als hundert jeweils geringfügig veränderten Variationen dieser Funktion.

Als Datenbasis verwendet das Programm dabei das Lazarus-Verzeichnis (dieses muss ggfls. noch ausgewählt werden) und lädt dann von dort etwa eine Million Quelltextzeilen aus den Source-Dateien. (Da dieses Programm sich an Lazarus-Benutzer wendet, kann davon ausgegangen werden, dass diese immer vorhanden sind. Hierzu gab es Einwände, diese werden unten diskutiert). Diese Strings werden dann jeweils nach 20 verschiedenen Buchstaben durchsucht. Wichtig ist dabei: Das Programm misst den *gesamten Durchlauf* dieser >20.000.000 Aufrufe pro Funktion, es nimmt also *keine* (in der Tat unsinnigen) *Mikromessungen* vor, die Streuung der Ergebnisse ist in der Regel dementsprechend gering (< ± 1% bei gleichmäßiger Belastung des Computers).

Der gesamte Test dauert ca. 12 Minuten (natürlich stark abhängig vom jeweiligen Rechner). Wem das zu lange dauert, der hat die Möglichkeit, die Anzahl eingelesener Strings zu reduzieren (voreingestellt ist 1.000.000) oder kann die Anzahl der Runs

heruntersetzen – die sind auf 4 voreingestellt, um auch durchschnittliche Abweichungen, also gewissermaßen das „Betriebssystem-Rauschen“, sowie Ausreißer erfassen zu können. Im Zweifelsfall aber bitte lieber die Anzahl der Strings als die der Runs reduzieren!

Gut wäre übrigens, dann eine Kaffeepause einzulegen und den Computer während des Tests unbelastet von anderen rechenintensiven Aktivitäten laufen zu lassen (wobei auch das Gegenteil dessen mal ganz interessant zu sehen ist, bloß dass das eben nicht Gegenstand des Tests ist).

Anschließend generiert das Programm ein Logfile namens "*AlignTestResult(Prozessorname).txt*", das im selben Verzeichnis wie die Quelltext-Dateien abgelegt wird (zwei meiner Ergebnisfiles sind im Ordner „*Documentation*“ enthalten).

Worum ich Euch nun bitten möchte, ist Folgendes:

1. Die Programmdateien herunterzuladen und in einen Ordner zu entzippen (Wer diesen Text liest, hat das schon getan).
2. Das Programm (Datei *AlignTest.lpr*) in Lazarus zu laden, es (*unverändert!*) zu kompilieren und zu starten. Voreingestellt ist das kompilieren ohne Debugger mit Optimization Level 2. Es spielt übrigens für die Messungen keine Rolle, ob man das Programm innerhalb der Lazarus-IDE laufen lässt oder außerhalb als Executable aufruft.
3. Ggfls. muss jetzt noch das Lazarus-Verzeichnis eingegeben werden, falls die Vorgabe (*/usr/share/lazarus/* für Linux bzw. *c:\lazarus* für Windows) bei Euch nicht zutrifft.
4. Den Test zu starten (das dauert wie gesagt etwa 12 Minuten).
5. Und schließlich die Ergebnisdatei ("*AlignTestResult(Prozessorname).txt*") entweder hier zu posten oder mir per EMail zuzusenden (mailto: [rue.walter@web.de](mailto:rue.walter@web.de)).

Ich habe mich bewusst dagegen entschieden, fertig kompilierte Executables zu veröffentlichen, da ich selbst äußerst zögerlich wäre, eine solche Fremddatei unbesehen auf meinem Rechner laufen zu lassen. Da sich das Projekt ausschließlich an Lazarus-Programmierer wendet, die mit Quelltext-Projekten umzugehen wissen, sehe ich darin auch kein Problem – der Mehraufwand besteht aus gerade mal 2 Mausklicks.

Das Programm steht – wie FPC / Lazarus selbst – unter einer freien Lizenz (LGPL). Für manche Nutzer wird es daher interessant sein, sich in den Quelltext einzulesen und ihn als Ausgangsbasis für eigene Versuche auch zu modifizieren (was ich ausdrücklich begrüße!). Andererseits baut der Test natürlich darauf, dass wir nicht am Ende Äpfel mit Birnen vergleichen und die übersandten Ergebnisse den jeweiligen Routinen im Originalzustand entsprechen. AlignTest protokolliert daher auch die MD5-Summen aller Test-relevanten Units zum Zeitpunkt der Kompilation – Ergebnislisten, aus denen hervorgeht, dass die Dateien

- TestFunctions.pas
- CharPosAsmFuncs\_X86\_64.inc
- CharPosAsmFuncs\_i386.inc

verändert wurden (uns sei es auch nur ein zusätzliches Leerzeichen), können daher nicht ausgewertet werden. Also bitte entweder erst mal testen, dann ändern, oder im Zweifelsfall die Zipdateien noch einmal in ein neues Verzeichnis entpacken und den Test dort starten.

---

Was könnte bei dieser Erhebung herauskommen? Naja, wenn man das vorher immer schon wüsste, bräuchte man es nicht zu machen. Der *Worst Case* wäre: Die Ergebnisse sind derart „*gewürfelt*“, dass sich daraus *keinerlei Gemeinsamkeiten* ableiten lassen. Der Versuch, Codeoptimierung auf unterster Assemblerebene zu betreiben, wäre dann eine *Sackgasse*, verlorene Liebesmüh. Aber auch das wäre dann ja eine Erkenntnis!

Im *besten Fall* dagegen könnten wir aber schon einiges darüber lernen, wie man zeitkritische *Hotspot*-Routinen (für die ist das hier vorrangig von Interesse) erheblich verbessern kann. Diese Erkenntnisse könnten – im allerbesten Fall – vielleicht sogar in die weitere Entwicklung von Free Pascal einfließen.

Jedenfalls bin ich bei meinen Internetrecherchen zu diesem Thema nirgends auf einen solchen Versuch gestoßen, das Mikro-Laufzeitverhalten prozessorübergreifend empirisch zu untersuchen. Klar: Hier wird nur ein *winziger Ausschnitt* des Prozessorverhaltens betrachtet. Aber allein dieser Ausschnitt kann *Laufzeitunterschiede von 1:3*, z.B. beim Sortieren, bewirken.

Dieser Test selbst wird die (Pascal-) Welt nicht umkrempeln. Er ist bewusst extrem fokussiert auf einen winzigen Ausschnitt, der mit hoher Vergrößerung betrachtet wird. Meine Hoffnung ist vielmehr, dass die Ergebnisse dieses Tests interessant genug sein werden, um weitere, umfangreichere Untersuchungen zum Micro-Prozessorverhalten anzustoßen.

Ob ich hier Neuland betrete oder bloß im nebligen Morast herumstapfe – ganz ehrlich: Ich weiß es nicht. Aber ich denke, einen Versuch ist es wert. Es kann auch eine Bauchlandung erste Klasse werden, aber das muss man einfach mal riskieren. Und selbst daraus kann man ja Erkenntnisse gewinnen. Die eigentlich interessanteste Frage ist also: *Gibt es da ein fruchtbares Land zu entdecken – oder bloß eine Wüste?*

Selbstverständlich werde ich auch die Ergebnisse (wie immer sie ausfallen mögen) veröffentlichen.

Jedenfalls: Je mehr Testergebnisse ich auswerten kann, desto wertvoller und damit umso eher von allgemeinem Interesse könnten etwaige Erkenntnisse daraus sein.

---

## Nachtrag zu einigen Einwänden:

Nach der Erstveröffentlichung von AlignTest wurden einige Einwände formuliert, deren wichtigste ich im Folgenden kurz darstellen und diskutieren möchte.

### 1) Unterschiedliche Performance der verglichenen Computer

Es ist wichtig zu verstehen, dass die *Gesamtpformance* des jeweiligen Computer aus der Betrachtung der Ergebnisse *herausgekürzt* wird und nur die *relative* Performance der einzelnen Codeblöcke *im Vergleich zueinander* betrachtet wird. An dieser Stelle sei auf die Erklärung in der Datei *ReadMe.pdf* verwiesen.

### 2) Störeinflüsse infolge unterschiedlicher RAM-Zugriffszeiten

AlignTest übergibt bei jedem einzelnen Funktionsaufruf einen String aus dem RAM an die jeweilige Routine. Je nachdem, ob dieser String nun im Cache vorgehalten wird oder nicht, ergeben sich daraus unterschiedliche Zugriffszeiten, die das Messergebnis verfälschen können.

Um diese möglichen Verfälschungen in den Griff zu bekommen, verfolgt AlignTest mehrere Strategien:

- a) Die String-Datenmenge ist erheblich *größer* als der (normalerweise) vorhandene Cache, > 50 MB.
- b) Jeder Testdurchlauf greift auf jeden String je einmal zu.
- c) Die Strings werden *gemischt*, indem sie zu Testbeginn sortiert werden. Sie werden also *nicht in der Reihenfolge ihrer Erstellung* ausgelesen.
- d) Jeder Testdurchlauf findet dieselben Bedingungen vor. Ändern sich diese rechnerintern auf ungleichmäßige Weise, müsste dies Auswirkungen auf die Streuung der Ergebnisse haben. Deshalb wird die komplette Abarbeitung eines jeden Funktionstests mehrfach wiederholt (voreingestellt sind 4 Runs). Anhand der *Streuungen* dieser Runs lässt sich das Ausmaß möglicher Verfälschungen abschätzen. Dasselbe gilt übrigens für schwankende Betriebssystemlasten.

Die bisherigen Erfahrungen waren insofern ermutigend, als die Streuungen im Allgemeinen im Vergleich zu den gemessenen Signalen *relativ gering* ausfallen (um nahezu zwei Größenordnungen kleiner). Es gab allerdings Einzelfälle mit exorbitanten Streuungen, diese (nicht auswertbaren) Fälle sind aber klar erkennbar.

### 3) Inhomogenität der Datenbasis

Bemängelt wurde ferner, dass die Tests von Rechner zu Rechner mit zwar *ähnlichen*, aber *nicht gleichen*, aus Quelltextdateien gewonnen Datensätzen durchgeführt werden, anstatt mit einem künstlich generierten und in seiner Charakteristik besser kontrollierbaren Stringensemble. Dies geriet insbesondere durch den Umstand in Blickfeld, dass infolge der anderen Verzeichnisstruktur unter Windows (für mich als Linux-Monogamisten unerwartet) auch die FPC-Sourcen eingelesen wurden und die Windows-Nutzer plötzlich mit > 2 Millionen Strings testeten anstelle der anvisierten 1 Million, wie sie für Linux typisch sind. (Das war der Grund, warum AlignTest ein zusätzliches „*MaxStrings*“-Edit erhielt).

Die bei AlignTest gewählte Vorgehensweise hat den Vorteil, realistische, „*lebensechte*“ Bedingungen zu schaffen, und es kann davon ausgegangen werden, dass die genutzten Dateien bei jedem Lazarus-User (an die sich der Test ausschließlich wendet) vorhanden sind. Damit nimmt AlignTest allerdings in der Tat in Kauf, dass die Testbedingungen auf jedem Rechner unterschiedlich sind.

Eine vorläufige Auswertung der Ergebnisse hat jedoch ergeben, dass die Charakteristiken der Datenensembles einander sehr ähnlich sind (selbst bei 2 Mio. Windows-Strings verglichen mit 1 Mio. Linux-Strings): 30 % Trefferwahrscheinlichkeit, durchschnittliche Lauflänge der Funktion von 38 Chars, durchschnittliche Lauflänge bei Treffern 17.

Das wichtigere Argument ist allerdings, dass es auf Datenhomogenität an dieser Stelle gar nicht ankommt – jedenfalls, solange die Datenstrukturen einander einigermaßen ähnlich sehen. Der Grund dafür ist, dass die Messwerte der einzelnen Funktionen ohnehin nur *relativ zueinander verglichen* werden. Auf dem jeweiligen Rechner aber findet jede Funktion die exakt gleichen Strings vor.

Wenn also (um die Argumentation in der *ReadMe.pdf* noch einmal aufzugreifen) Funktion A doppelt so schnell ist wie die (lediglich intern anders gestaltete) Funktion B, dann wird sie *in hinreichender Näherung* sowohl dann doppelt so schnell sein, wenn sie die Strings durchschnittlich 37 Zeichen weit zu durchlaufen hat, wie auch dann, wenn die durchschnittliche Lauflänge bei 39 liegt. Vielleicht ändert das etwas an der zweiten Nachkommastelle, aber die beobachteten Signale selbst liegen in der Größenordnung von 100 – 300 %. *Die durch Dateninhomogenität entstehende Unschärfe liegt also schlimmstenfalls im Promillebereich.*

März 2015  
Rüdiger Walter