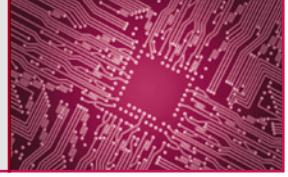


Mobiles Java, aber schnell

Entwickeln mobiler Java-Anwendungen

Michael Hofstätter, Stefan Feßl

Welche Einschränkungen und Besonderheiten sind auf eingebetteten oder mobilen Endgeräten beim Einsatz von Java zu beachten? Wie können oder sollen CPUs mit mehreren Kernen genutzt werden? Unter welchen Voraussetzungen ist die Nutzung von nativen Funktionen sinnvoll? Zur Beantwortung dieser Fragen werden anhand konkreter Beispiele Performanceunterschiede untersucht und konkrete Tipps gegeben, wie Einschränkungen vermieden werden können. Dazu gibt es Hinweise zu Tests und Messungen, um Schwachstellen zu erkennen. Für jene Anforderungen, in denen die Java-Performance dennoch nicht ausreicht, wird auch die Integration von Java und C mit Android NDK oder JNI sowie Multithreading betrachtet.

Unterschiede zwischen Desktops und mobilen Geräten

Seit der Einführung von Java gibt es einige Vorurteile, die der Programmiersprache nachhaltig anhaften. Am rufschädigsten dabei ist die angeblich mangelnde Performance. War das anfangs noch zutreffend, ist dieses Vorurteil – zumindest im Desktop- und Serverbereich – längst überholt. Doch wie verhält es sich im mobilen Bereich, wo auch die Hardwareumgebungen unterschiedlich sind? Viele werden sich noch an die ersten Apps für Handys erinnern: J2ME-Applikationen, deren Ladezeit alleine auch die geduldigsten Nutzer auf eine harte Probe stellten.

Wir möchten hier beleuchten, wie es derzeit um die Performance von Java-Applikationen auf mobilen Endgeräten bestellt ist und welche Punkte bei der Entwicklung von Apps zu beachten sind, um gute Ergebnisse zu erzielen.

Einer der wesentlichen Unterschiede zwischen mobilen Geräten und der Welt der Desktops und Notebooks ist die eingesetzte Prozessorarchitektur. Sind es am Desktop CISC-Prozessoren (Complex Instruction Set Computer) mit 80x86-Architektur, so kommen auf mobilen Devices hauptsächlich ARM-RISC-Prozessoren (Reduced Instruction Set Computer) zum Einsatz. Zwar verschwimmen auch hier langsam die Grenzen, da es bereits erste Handys mit einem Intel-Atom-Chip und erste Notebooks mit ARM-CPU am Markt gibt, doch die ARM-Architektur ist unbestritten die vorherrschende Plattform im mobilen Bereich. Sie basiert auf einem RISC-Design. Die besonderen Vorteile der ARM-Architektur sind der geringe Energieverbrauch und die geringe Größe des Chips, was auch der Hauptgrund für den Erfolg bei mobilen Geräten ist.

Im Zuge der unterschiedlichen ARM-Versionen gab es auch mehrere Entwicklungen, welche durchaus Auswirkungen auf die Performance haben können. Ein Feature von allen Cortex-A-Chips von ARM ist NEON. Dieser Befehlssatz (für Single Instruction, Multiple Data) beschleunigt Multimedia, Sprach-Processing und auch Bildberechnungen.

Von großer Bedeutung sind die Mehrkern-Designs von ARM, also Chips, die statt eines Rechenkerns zwei oder gar vier zur Verfügung stellen. Moderne High-End-Smartphones basieren auf ARMv7-Designs. Dazu gehören auch die CPU-Familien Cortex-A15, Cortex-A8 und Cortex-A7 der Cortex-A-Serie. Tabelle 1 enthält einen kurzen Vergleich mit der älteren ARM11-Version des ARMv6-Designs.

Im mobilen Bereich hat aber nicht nur die Hardware Einfluss auf die Leistung und Performance, sondern auch das Betriebssystem und welche Features im ARM-Design genutzt werden. Performanceverbesserungen gab es beispielsweise in der Android-Version 2.2 mit der Einführung des JIT-Compilers und mit 2.3 mit Optimierungen für Mehrkern-CPUs.

Neben Hard- und Software spielt aber natürlich auch der aktuelle Lastzustand des Endgerätes eine wesentliche Rolle. So reduziert zum Beispiel der Energiesparmodus auch die CPU-Leistung und die aktuell geladenen Apps reduzieren das verfügbare RAM. Und natürlich die Größe des RAM-Speichers, die auf älteren Handys und embedded Geräten stark eingeschränkt ist.

Nachdem wir uns nun mit den Grundlagen, Hardware und Betriebssystemumgebungen beschäftigt haben, vergleichen wir jetzt die Leistung der einzelnen Umgebungen, um die Bedeutung von optimierenden Maßnahmen während der Entwicklung zu untermauern.

Alle Messergebnisse, die hier beschrieben werden, beziehen sich auf real eingesetzte Geräte und sind damit nicht direkt mit Labor-Benchmarks zu vergleichen. Sie sind lediglich als Beispiele zu verstehen, um Prinzipien der Entwicklung zu untersuchen. Aus diesem Grund sollen hier auch nicht die Modellbezeichnungen angegeben werden, sondern lediglich die Version der CPU-Architektur, deren Kerne und Taktfrequenz.

Es werden fünf Handys mit Android (vom Single-Core mit 600 MHz bis zum Quad-Core mit 1,9 GHz), eine Entwicklerplatine und ein Notebook mit Intel i7CPU gegenübergestellt.

Grundlagen

Rechenintensive Algorithmen, wie der Alpha-Beta-Algorithmus sowie die Mandelbrot-Funktion, zeigen sehr deutlich die Grenzen und Unterschiede der CPU-Performance für Gleitkomma- und Integer-Berechnungen. Ergänzend wurde auch das String-Handling in Form einer einfachen String-Konkatenierung untersucht. Nicht betrachtet wird in diesem Artikel das weite Feld der Grafikperformance, ARM-Assemblerprogrammierung und NEON, da dies den Rahmen sprengen würde.

Wir haben die Algorithmen in Java implementiert und zur Ausführung in einer Android-App sowie einer Java-Consolen-Applikation verpackt. Die Programme wurden dabei für mobile Geräte optimiert.

Zur groben Einordnung werden wir zum Beginn den Performanceunterschied zwischen aktuellen Handys und einer derzeit leistungsmäßig wohl im Mittelfeld einzuordnenden Notebook-CPU betrachten. Ohne auf die genauen Zahlen einzugehen, zeigen die in Abbildung 1 gezeigten Ergebnis-

	Cortex-A15	Cortex-A9	Cortex-A8	Cortex-A7	ARM11
Takt	– 2,5 GHz	800 MHz – 2 GHz	600 MHz – 1 GHz	Bis 1 GHz	772 MHz
Rechenleistung DMIPS/MHz	3,4	2,5	2,0	1,9	1,25
Max. Anz. Kerne	4	4	1	4	1

Tabelle 1: Vergleich der ARM-Designs

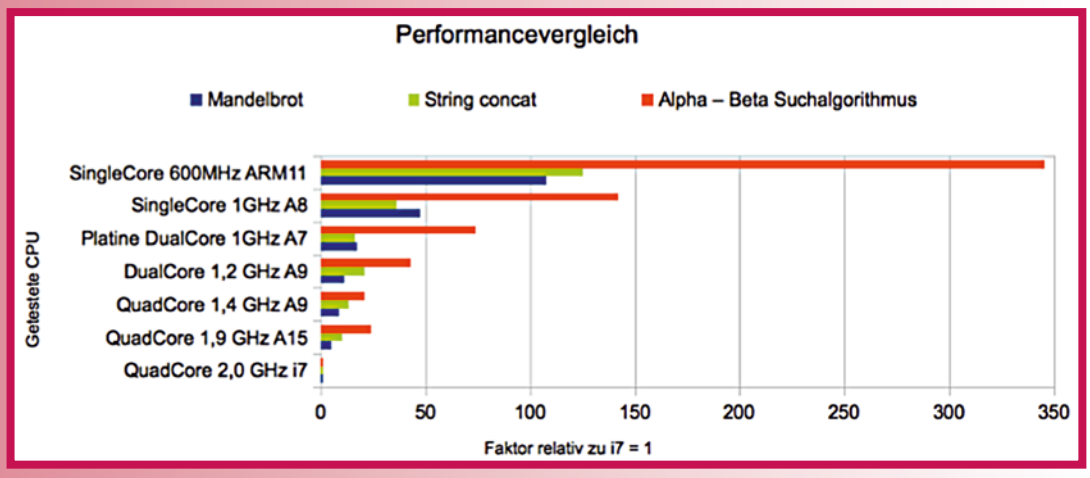


Abb. 1: Performance von Algorithmen für Single-Core

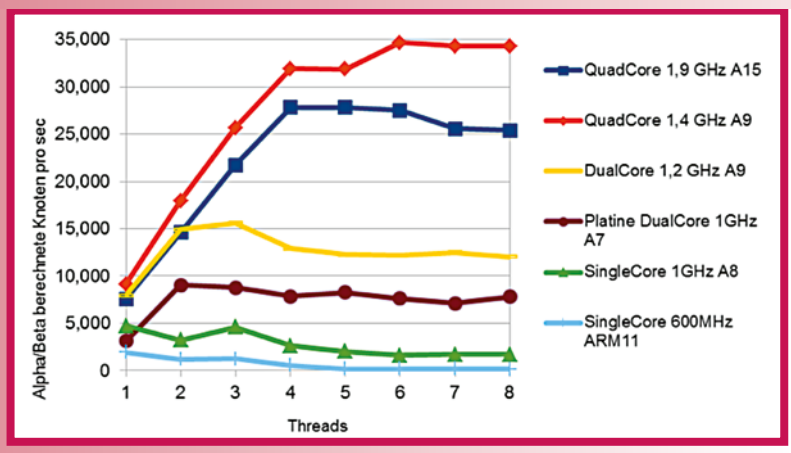


Abb. 2: Performancevergleich bei parallelisiertem Suchalgorithmus

se bereits deutlich, wie die Performance von Handy-CPUs trotz aller Leistungssteigerungen in den letzten Jahren einzuschätzen ist. Zwischen einer auf allen vier Kernen ausgelasteten ARM-CPU und dem i7 des Notebooks liegt ein Faktor 5 bis 20. Bei den Handys ist außerdem eine große Streuung der Werte zu erkennen. Zwischen den schnellsten und langsamsten hier getesteten Geräten wurde der Faktor 22 gemessen.

Wichtig ist also, bereits bei der Konzeption einer mobilen App mit sehr unterschiedlich leistungsfähigen Geräten zu rechnen. Zu sehen ist, dass die CPUs mit A9- und A15-Architektur bei diesen Tests jedenfalls wesentlich leistungsfähiger als ältere Versionen sind.

Multithreading

Viele moderne Handys besitzen bereits mehr als einen CPU-Kern. Ist es für Java-Programme sinnvoll, dies zu nutzen? Diese Grundfrage kann anhand von Abbildung 2 eindeutig mit Ja beantwortet werden.

Zu beachten sind die enormen Unterschiede in der gemessenen Performance der einzelnen Geräte. Dies zeigt sehr deutlich, dass es für Entwickler überaus wichtig ist, auf die verschiedenen Fähigkeiten der Hardware Rücksicht zu nehmen und die Potenziale der parallelen Abarbeitung zu nutzen.

FLoating-Point

Ist es schneller, den Datentyp `Float` statt `Double` zu verwenden? Die Apfelmännchen-Berechnung zeigt hier nur geringe Unterschiede und wiegt in diesem Anwendungsfall die geringere Präzision nicht auf. Im Durchschnitt ist die Berechnung mittels `Float` bei unseren Tests 12 Prozent schneller.

Einflussfaktor Speicher

Ein oft sehr unscheinbarer und vielfach unterschätzter Performance-Killer sind String-Operationen. Insbesondere das Anhängen von Strings führt zum Kopieren und Speicherallokationen. Relativ harmlos aussehende Programmzeilen führen so zu unnötig langen Laufzeiten. Vor allem bei Algorithmen in Schleifen kann das große Auswirkungen haben.

Als Programmbeispiel nutzen wir dies, um die Auswirkungen von Kopien im Speicher und Heap-Performance zu erahnen. Wie Tabelle 2 zeigt, empfiehlt es sich jedenfalls für mobile Applikationen besonders, kleine Häppchen zu verarbeiten. In wirklich performancekritischen Programmteilen sollten String-Operationen sowieso vermieden werden.

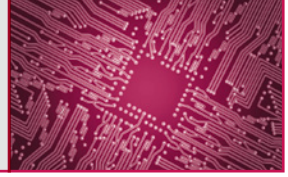
Device	Test 1: 1000 Strings mit 10 Zeichen zusammenhängen	Test 2: 10 mal 100 Strings mit 10 Zeichen zusammenhängen	Faktor Test1/Test2
Single-Core 600 MHz ARM11	1656	250	7
Single-Core 1 GHz A8	871	72	12
Dual-Core 1,2 GHz A9	469	42	11
Quad-Core 1,4 GHz A9	440	26	17
Quad-Core 1,9 GHz A15	145	20	7
Quad-Core 2,0 GHz i7	6	2	3

Tabelle 2: Vergleich von String-Konkatenierung

Native Funktionen

Java bietet die Möglichkeit, Programmteile in C mittels des „Java Native Interface“ (JNI) einzubinden. Für Android existiert dafür das sogenannte „Native Development Kit“ (NDK).

Lohnt sich der zusätzliche Aufwand, oder ist es besser, in Java zu imple-



Device	Java Floating Point	Native C Floating Point
Single-Core 600 MHz ARM11	20,65	5,64
Single-Core 1 GHz A8	7,8	4,64
Double-Core 1,2 GHz A9	1,88	0,93
Quad-Core 1,4 GHz A9	1,55	0,80
Quad-Core 1,9 GHz A15	0,71	0,52

Tabelle 3: Vergleich von Java und native Implementierung in C

mentieren? Wir haben die Funktion zum Berechnen eines Apfelmännchen-Bildes in C über das NDK in die App eingebunden. Die native Funktion war dabei nur beim ältesten Handy wesentlich schneller als die Java-Implementierung, ansonsten ist die Geschwindigkeitssteigerung eher moderat, wie Tabelle 3 zeigt.

Für dieses sehr einfache Beispiel mag die native Implementierung noch lohnend erscheinen. Die Wartungsaufwände sind gering, da der Code nur wenige Zeilen umfasst. Für komplexere Funktionen sind jedoch die höheren Aufwände für die native Entwicklung und Wartung zu beachten und dem doch bescheidenen Performancegewinn gegenüberzustellen.

Anwendungsdesign

Eine wichtige Maßnahme zur Sicherstellung der Wartbarkeit der Gesamtapplikation ist die Trennung der performancekritischen Teile von der restlichen Applikation. Für diese Teile empfiehlt es sich, unterschiedliche Regeln für die Programmierung zu definieren. Nachfolgend sollen einige dieser Regeln kurz genannt werden, die sich in Projekten bewährt haben:

- Unit-Tests sind das wichtigste Mittel zum Erhalten der Wartbarkeit des Codes, insbesondere da einerseits Abstriche an der Lesbarkeit zu erwarten sind und andererseits mehrmaliges Refactoring notwendig sein kann.
- Multithreading ist oft die effektivste Möglichkeit, eine Anwendung zu beschleunigen. Wichtig ist, auch die Einkern-Systeme mit älteren Android-Versionen zu berücksichtigen und nicht mit mehreren parallelen Threads zu belasten. Als Vereinfachung wird es meist ausreichen, zwischen der Ausführung in einem Single-Thread und vier parallelen Threads zu unterscheiden.
- Es empfiehlt sich, beim Start der App die Leistungsfähigkeit des Systems zu bestimmen und einen Schalter für die parallele Ausführung zu setzen.
- Speicherallokationen in oft ausgeführten Codeteilen sollten vermieden werden. Alle Strukturen und Bäume bis zur maximalen Tiefe sollten bereits vor der Ausführung angelegt werden. Hierbei gilt es zu beachten, dass der verfügbare RAM-Speicher oft knapp ist.
- Für schnelles Kopieren von Daten sollten eindimensionale Arrays angelegt und `System.arraycopy` genutzt werden.
- Vorsicht ist geboten bei String-Operationen, das oftmalige Kopieren und Anlegen im Heap ist häufig Grund für schlechte Performance.
- Die Einbindung von nativen C-Funktionen klappt leichter, wenn der zu optimierende Code zuerst in Java mit statischen Funktionen ohne Nutzung von Vererbung, Java-Frameworks und Bibliotheken implementiert wird. Es sollte also nur die Schnittmenge der Spracheigenschaften von Java und C genutzt werden. Dadurch ergibt sich ganz natürlich eine Trennung zwischen den performancekritischen und -unkritischen Programmteilen. Danach kann eine Funktion nach der anderen von Java in C portiert werden, bis die Performance ausreichend ist.

Zusammenfassung

Mobile Endgeräte und Entwicklerplatinen sind gut für den Einsatz von Java geeignet. Auch im Vergleich zur Programmierung in C ist die Performance gut und selbst für rechenintensive Aufgaben zumeist ausreichend. Die Unterschiede zwischen den Geräten sind allerdings groß, vor allem wenn die Applikationen auf mehreren Prozessorkernen laufen können.

Desktops und Notebooks spielen bezüglich Performance immer noch in einer andern Liga, die Unterschiede werden jedoch mit jeder Handygeneration kleiner.

Aus den durchgeführten Tests ist ersichtlich, dass der Speicherzugriff eine Schwäche der mobilen Geräte darstellt. Berechnungen mit Gleitkommazahlen und Multithreading zählen hingegen relativ gesehen zu deren Stärken. Daraus können mehrere Empfehlungen für performante Applikationen abgeleitet werden:

- Auf geringen Speicherverbrauch achten und möglichst wenig Bewegung am Heap verursachen; besonderes Augenmerk ist auf String-Manipulationen zu richten.
- Die Multithreading-Fähigkeit des Devices sollte genutzt werden, da es eine gut skalierende Leistungssteigerung verspricht.
- Wenig Beschleunigung bringt die Nutzung von kleineren Datentypen (wie `float` statt `double`, oder `short` statt `int`), auch die manuelle Optimierung der einzelnen Anweisungen bringt erfahrungsgemäß meist zu kleine Vorteile für den getätigten Aufwand.
- Native Entwicklung in C und Einbindung über das JNI kann eine Beschleunigung bringen, die aber in der Praxis leider oft geringer ausfällt, als erhofft.

Wenn Sie bei der Programmierung auf diese Punkte achten, dann bleiben Ihnen einige teure Umwege erspart und Ihre Applikation wird mit geringem Aufwand performant!

Dipl.-Ing. Michael Hofstätter, MSc ist Softwarearchitekt und Berater im Java-Bereich bei ANECON Software Design und Beratung GmbH. Vor acht Jahren begann er mit der Entwicklung für mobile Endgeräte auf Java ME und Android. Seine Schwerpunkte sind neben Entwicklungstätigkeiten Trainings für Vorgehensmethoden wie Kanban sowie Audits und Beratung zur Softwarequalität.
E-Mail: michael.hofstaetter@anecon.com

Ing. Stefan Feßl ist Kompetenzfeldleiter des Bereichs „Java-Entwicklung" bei ANECON Software Design und Beratung GmbH. Seine Laufbahn im IT-Umfeld begann er vor vierzehn Jahren als Java-Entwickler, seit acht Jahren ist er als Team- und Projektleiter im Bereich der Individualentwicklung von Software mit Java befasst. Im Zuge der Themenleitung ist einer seiner Schwerpunkte der Einsatz von Java in mobilen Endgeräten (Android).
E-Mail: stefan.fessl@anecon.com