Warum ist Java so langsam?

Man sagt den Entwicklern der Sprache aber nach, dass man immer immer den "besseren" statt den "schnelleren" Weg wähle.

== Java ist langsam ==

Hier nun also ein paar Kenndaten:

- Der Array-Zugriff ist viel langsamer im Vergleich zu C, was an den vielen Überlaufkontrollen liegt. Hier hat aber bisher jede Java Version massive Verbesserungen bereitgestellt.
- Man kann nicht auf beliebige Speicherbereiche zugreifen. Dadurch können einige Bitoperationen wie Kompression und Dekompression sehr langsam sein. Diese Restriktionen setzen aber die Hochsprachen aus Sicherheitsgründen um.
- Java verwendet viel mehr Speicher als C. Wenn die Applikation selbst viel Speicher verbraucht und der verfügbare Speicher aufgebraucht ist, macht es das System langsamer. Allerdings ist die Zuweisung / Freigabe von Speicher durch die Garbage Collection sehr schnell.
- Streamsbasiertes Arbeiten ist langsam, weil Synchronisation beim Stream Zugriff genutzt wird. Seit Java 5 gibt es jedoch neue - nicht synchronisierte - I/O-Bibliotheken (java.nio.*).
- Java bietet keine gleiche Low-Level-Funktionalität, wie es bei C der Fall ist, so kann man keine schmutzigen Tricks verwenden um einige Operationen schneller zu machen. So können aber zu gunsten der Portabilität weder Inline-Assembler noch clevere x86 Tricks verwendet werden.
- String-Operationen sind langsam. Java verwendet unveränderliche, UTF-16 kodierte Strings. Das heißt, Sie brauchen mehr Speicher, mehr Speicherzugriffe und manche Vorgänge sind komplizierter als mit ASCII. Es ist die richtige Entscheidung für Portabilität, büßt dadurch aber Leistung ein.
- Startzeiten sind noch immer langsam, weil viele ungenutzter Balast geladen wird

== Java ist schnell ==

In aller Fairness, gibt es mehrere Anwendungen, bei denen Java gar schneller als die meisten anderen Sprachen:

- Speicherzuweisung und Freigaben sind schnell und billig. In einigen Fällen ist es besser Speicher neu zu allokieren als zwischengespeicherten wiederzuverwenden.
- Object Instanziierung und objektorientierte Features sind schnell zu verwenden (schneller als C++ in vielen Fällen?)
- Methodenaufrufe sind grundsätzlich kostenfrei. Der HotSpot Compiler wird diese von alleine inlinen, was für saubereren Code und manchmal für bessere Ergebnisse als beim manuellen Inlining sorgt. Verglichen dazu sind C / C++ Methodenaufrufe die nicht als inline deklariert sind, sehr teuer.
- Synchronisation und Multi-Threading sind einfach und effizient, was bei aktuellen Mehr-Kern-CPUs meist einen Geschwindigkeitsgewinn bringt. In C ist die Komplexität beim Entwickeln mit Multi-Threading viel fehleranfälliger.
- Die String-Klasse ist intelligent konstruiert. Teilstrings können Verweise auf das übergeordnete String Charakter Array haben (Speichereinsparung), so dass viele Operationen (Verkettung, substring, Länge) sehr schnell laufen.
- Collections sind schnell. Teilweise schlägt Standard Java das Standard C++, was etwas mit der Objekt Behandlung zu tun haben könnte.
- Das "Array kopieren" ist stark optimiert. Es wird von Hand abgestimmt Assembler code genutzt der in einigen Benchmarks das Äquivalent in C schlägt.

== Quellen ==

- stackoverflow.com/questions/21…ion-of-being-slow/2163570