High-Speed-Software zum Nulltarif dank Multicore?

Karl Nieratschker, SKT Nieratschker

Da sich die CPU-Taktfrequenz nicht mehr beliebig erhöhen lässt, wird auch die darauf laufende Software nicht mehr automatisch schneller. Stattdessen versucht man heute, die Leistung durch eine größere Zahl von CPUs (Multicore-System) zu steigern. Dieser Beitrag beschäftigt sich mit der Frage, ob existierende Singlecore-Software unmodifiziert vom Einsatz der Multicore-Technologie profitieren kann, oder ob dafür spezielle Anpassungen nötig sind.

Um diese Frage beantworten zu können, muss zwischen einigen Fällen unterschieden werden.

Systeme ohne Betriebssystemunterstützung

In diesem Fall ist der Anwendungsprogrammierer nicht nur für seine Applikation im Allgemeinen zuständig, sondern auch dafür, dass die CPU zum richtigen Zeitpunkt den richtigen Programmteil seiner Applikation ausführt. Wird von einem Singlecore- auf ein Multicore-System gewechselt, dann muss die Applikation auch so geändert werden, dass der zweite Core ebenfalls genutzt wird.

Systeme mit Betriebssystemunterstützung

Wurde auf dem Singlecore-System bereits ein Betriebssystem eingesetzt, das auch auf dem ersten Core eines neuen Dualcore-Systems laufen soll, dann gibt es für die Softwareimplementierung des zweiten Cores verschiedene Möglichkeiten. Soll auf dem zweiten Core ein anderes oder gar kein Betriebssystem laufen, dann handelt es sich um asymmetrisches Multiprocessing (AMP). In diesem Fall muss ein Teil der Applikation auf den anderen Core übertragen werden. Enthält dieser Applikationsteil Systemaufrufe, dann muss dafür Ersatz gefunden werden. Falls kein Betriebssystem auf dem zweiten Core existiert, muss auch für die CPU-Verteilung gesorgt werden. Müssen die Applikationsteile auf den verschiedenen Cores Daten austauschen, dann muss auch eine entsprechende Kommunikationslösung implementiert werden.

Soll auf dem Core 2 dasselbe Betriebssystem wie auf Core 1 laufen (Symmetrisches Multiprocessing, SMP), dann wird nur ein einziges Betriebssystem benötigt (sofern das Betriebssystem dies unterstützt), und die Applikation muss somit nicht aufgeteilt werden. Die beiden Cores und alle anderen Ressourcen werden vom Betriebssystem nach Bedarf zugeteilt. Diese Konfiguration erfordert zunächst keine Änderungen der Applikation und soll deshalb weiter untersucht werden. (**Bild 1**)

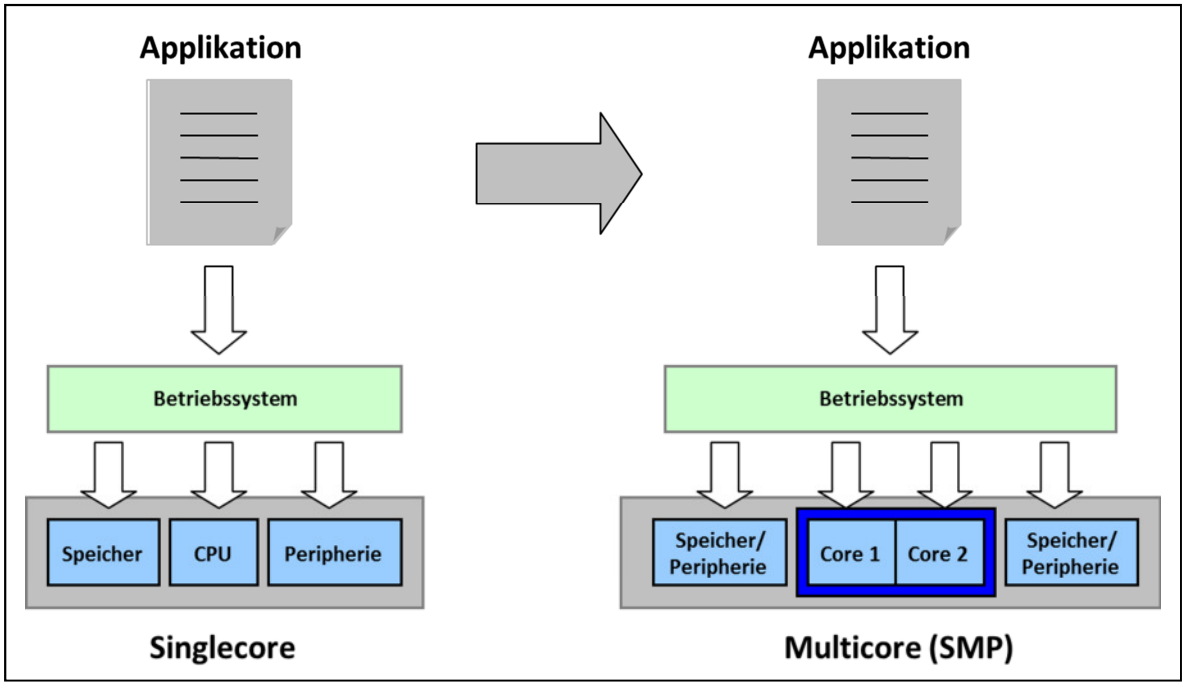

Bild 1: Übergang von Singlecore- auf Symmetrisches Multicore-System

Multithread-Design auf Singlecore-Systemen (Nebenläufigkeit)

Der Wechsel von einem Singlecore- auf ein Multicore-System auf der Basis desselben Betriebssystems ist für die Applikation prinzipiell transparent, so dass es keine Notwendigkeit für eine Änderung der Applikation gibt. Aufgrund des zusätzlichen Cores können jetzt mehrere Threads wirklich gleichzeitig laufen, so dass es zu einer tatsächlichen Parallelität kommt. Neben der dadurch möglichen Leistungssteigerung hat diese Tatsache aber auch Konsequenzen, die dazu führen können, dass bisher problemlos funktionierende Singlecore-Anwendungen plötzlich ein fehlerhaftes Verhalten zeigen oder sogar abstürzen. Dies kann z.B. damit zu tun haben, dass die zeitliche Ausführungsreihenfolge der Threads anders ist als vorher. Eventuell vorhandene Fehler bei der Zugriffssteuerung von Threads auf gemeinsame Daten können sich dadurch zum ersten Mal bemerkbar machen. Ein anderer klassischer Problemfall ist der Versuch, die Ablaufreihenfolge von Threads mithilfe von Prioritäten zu regeln. Eine Applikation, die sich z.B. darauf verlässt, dass Threads mit hoher Priorität von Threads niedrigerer Priorität nicht gestört werden können, läuft auf einem Singlecore-System, das Prioritäten nicht verändert, wie erwartet. In einem Multicore-System können Threads mit unterschiedlicher Priorität aber gleichzeitig ausgeführt werden, so dass es nun zu sporadisch auftretenden, und damit schwer lokalisierbaren Problemen kommen kann. Dabei ist zu beachten, dass es sich hier nicht um Probleme der Multicore-Technologie, sondern um Fehler im Design der Singlecore-Applikation handelt, die durch die Anwendung in einer Multicore-Umgebung auftreten.

Multithread-Design auf Multicore-Systemen (Parallelität)

Eine Singlecore-Applikation kann theoretisch immer dann von der zusätzlichen Leistung weiterer Cores profitieren, wenn es gleichzeitig mehrere Threads gibt, die ausgeführt werden können (**Bild 2**). In der Praxis ist es allerdings häufig so, dass die Threads hauptsächlich auf Ereignisse warten müssen, so dass sich der mögliche Vorteil gar nicht auswirken kann. Auch wenn beim Entwurf der Singlecore-Applikation die zeitscheibenbasierende Quasiparallelität vermieden wurde, kann ein eventuell arbeitsloser Core nicht genutzt werden.

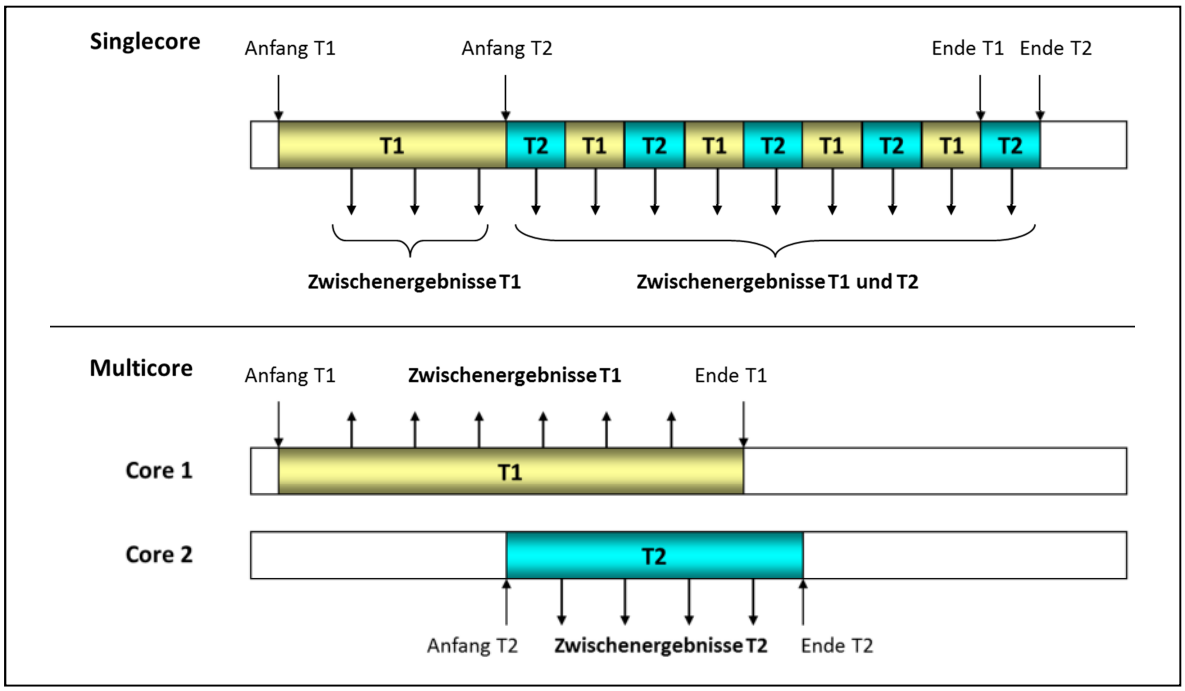

Bild 2: Lange Operationen mit Zwischenergebnisausgabe

Grundsätzlich muss ein gutes Multicore-Softwaredesign dafür sorgen, dass es immer genügend ausführbereite Threads gibt, um die Cores auslasten zu können. Eine Möglichkeit besteht darin, bisher sequenziell ausgeführte, aber parallelisierbare Operationen von unabhängigen Threads erledigen zu lassen. Darüber hinaus kann es aber auch nötig sein, lang laufende Operationen so aufzuteilen, dass mehrere gleichzeitig laufende Threads das Ergebnis in kürzerer Zeit ermitteln können. Grundsätzlich ist bei diesen Parallelisierungsmaßnahmen immer zu beachten, dass das Erzeugen und Löschen, sowie die Synchronisation der Threads einen erheblichen Zeitaufwand kosten kann, der im Verhältnis zur erledigenden Arbeit stehen muss. Aus diesem Grund dürfen die durch die Zerlegung entstehenden Teile nicht zu klein werden. Entstehen durch die Zerlegung mehr Threads als es verfügbare Cores gibt, dann kann dies zu einem unerwünschten Timeslicing-Effekt wie bei Singlecore-Systemen führen.

Beispiel: Parallelisierung einer Schleife

Eine häufig genutzte Möglichkeit, mehr Parallelität zu erreichen, besteht darin, eine lang laufende Operation, wie z.B. eine große Schleife, so zu zerlegen, dass ihr Code von mehreren parallel arbeitenden Threads ausgeführt werden kann (**Bild 3**).

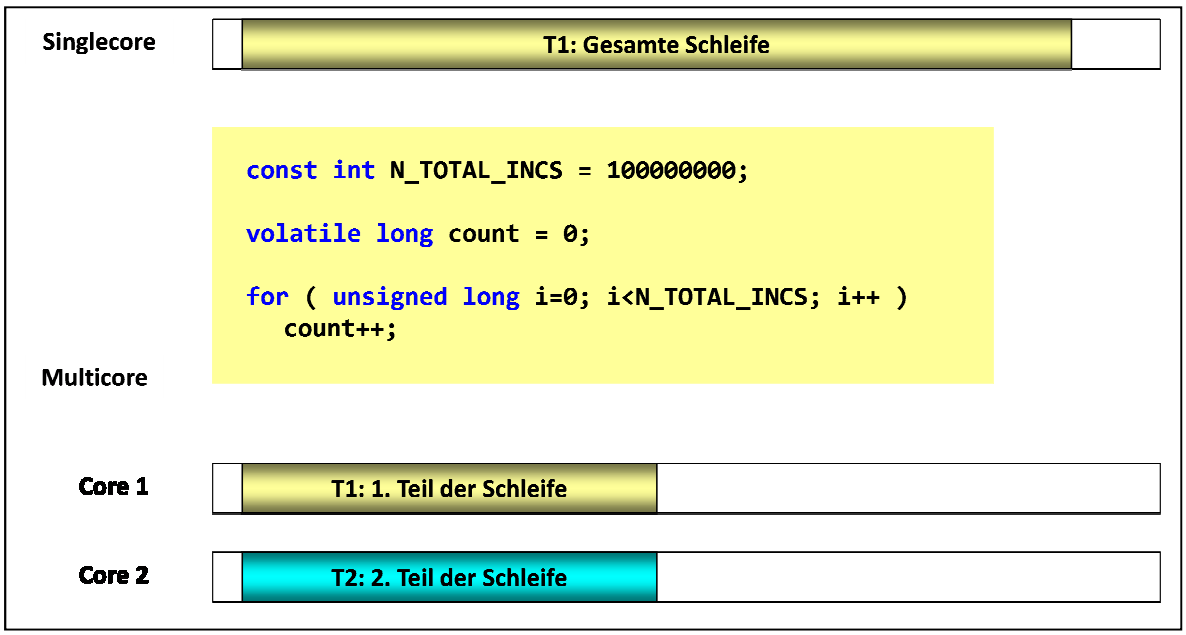

Bild 3: **Leistungssteigerung durch Aufteilung einer Schleife auf zwei Threads**

Dies führt normalerweise dazu, dass nun mehrere Threads auf dieselben Daten zugreifen müssen. Dabei kann das Problem auftreten, dass die Ausführung der dafür nötigen Synchronisationsmechanismen länger dauert, als der Code, der dadurch geschützt werden soll. Dies kann dazu führen, dass der Code auf Multicore-Systemen langsamer läuft, als dies vor dem Optimierungsversuch der Fall war. In manchen Fällen kann dieses Problem dadurch gelöst werden, dass jeder Thread ein Teilergebnis in einem privaten Speicher ermittelt und die Teilergebnisse zum Schluss von einem übergeordneten Thread zusammengefasst werden. Dadurch entfällt die Notwendigkeit einer Threadsynchronisation, so dass die zusätzliche CPU-Leistung voll zur Geltung kommen sollte. Trotzdem kann die Applikation immer noch schlechtere Laufzeitergebnisse liefern als in der Singlecore-Umgebung. Der Grund für dieses Verhalten kann die Arbeitsweise des CPU-Caches sein, denn beim Lesen eines Wertes wird normalerweise nicht nur der Wert selbst, sondern eine ganze Cachezeile (z.B. 64 Bytes) geladen. Verändert der Thread nun den Wert und liest dann ein *anderer* Thread auf einem *anderen* Core einen *anderen* Wert *derselben* Cachezeile, dann wird die gesamte Cachezeile aktualisiert, obwohl dies eigentlich nicht nötig wäre. Um dieses sog. False Sharing zu vermeiden, muss also dafür gesorgt werden, dass die Speicherbereiche von Threads, die von verschiedenen Cores ausgeführt werden, nicht innerhalb derselben Cachezeile liegen. (**Bild 4**).

	Cores	Threads	Loops	Zeit in ms	Kommentar
1.	1	1	100 000 000	484 - 500	
2.	1	2	100 000 000	484 - 500	Sync-Fehler!
3.	1	2	10 000 000	12109 - 12187	Mutex
4.	1	2	10 000 000	578 - 594	Critical Section
5.	1	2	10 000 000	279 - 313	Interlocked
6.	1	2	100 000 000	484 - 500	Private Variable
7.	1	4	100 000 000	484 - 516	Private Variable
8.	2	2	100 000 000	843 - 860	Private Variable
9.	2	2	100 000 000	250 - 266	Cache-Korrektur

Bild 4: Laufzeitergebnisse der Aufteilung einer Schleife auf mehrere Threads

Generell ist die Leistungsoptimierung auf Threadebene sehr kompliziert, weshalb Produkte wie z.B. OpenMP, Threading Building Blocks (TBB) von Intel® oder Thread Parallel Library (TPL) von Microsoft® immer mehr an Bedeutung gewinnen. Mit ihrer Hilfe können derartige Probleme auf höherer Ebene ohne explizite Threadprogrammierung gelöst werden.

Zusammenfassung

Singlecore-Anwendungen sollten prinzipiell unmodifiziert auch auf SMP-Multicore-Systemen laufen. Eventuell auftretende Fehler sind in der Regel auf ein fehlerhaftes Design der Applikation zurückzuführen. Das Leistungspotenzial des Multicore-Systems wird i.A. nur teilweise genutzt, in manchen Fällen ist das Laufzeitverhalten sogar schlechter als vorher. Um die Möglichkeiten des Systems optimal nutzen zu können, müssen Applikationen deshalb in der Regel speziell optimiert werden. Dies muss nicht unbedingt mit komplizierter Threadprogrammierung realisiert werden, da zunehmend mehr Produkte zur Verfügung stehen, die diese Probleme auf höherer Ebene lösen.

Autor

Dipl. Inf. Karl Nieratschker verfügt über eine langjährige Erfahrung in den Bereichen Softwareentwicklung, Support, Beratung und Training, überwiegend für Embedded-/Realtime- und Windows-Systeme. Seit 1999 ist er als freiberuflicher Trainer, Softwareberater und Coach tätig. Einer seiner Schwerpunkte ist Multithreading- und Multicore-Programmierung.

Internet: www.skt-nieratschker.de
Email: office@skt-nieratschker.de