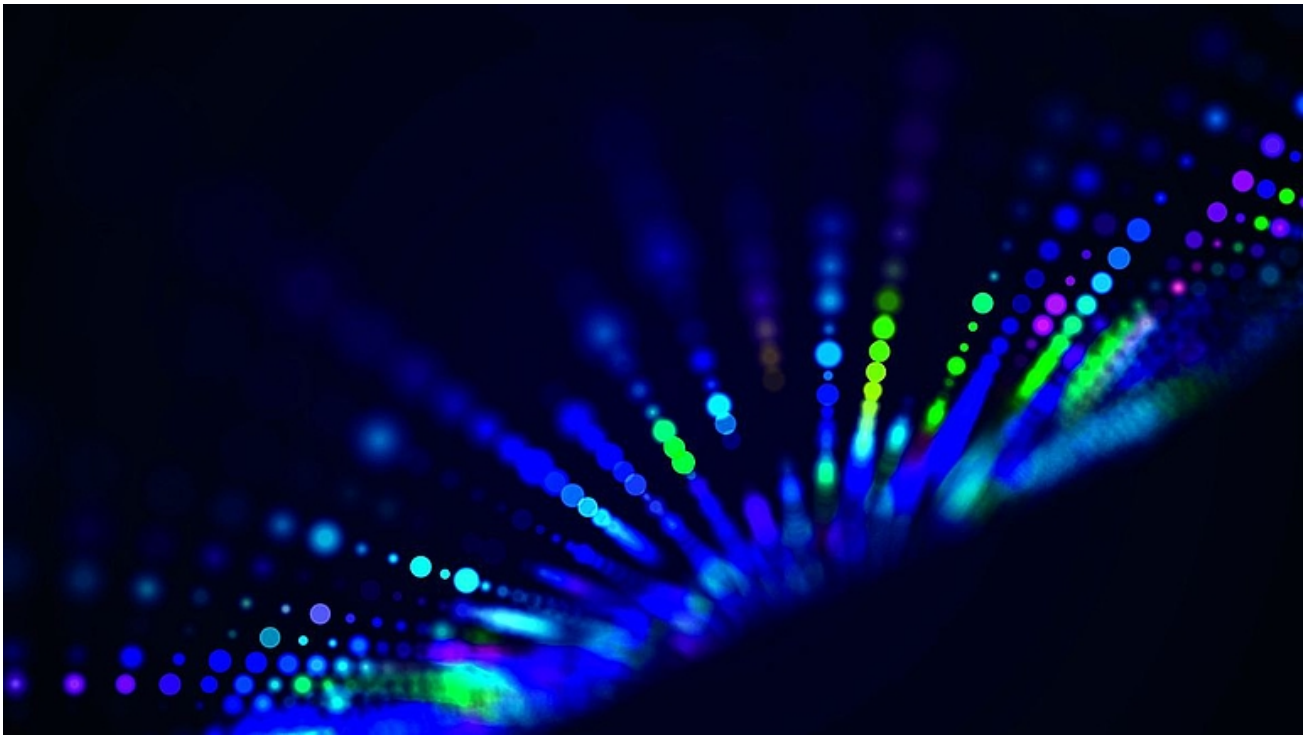
Praxis

8 Regeln für die Parallelprogrammierung von Multicores

Die Herausforderungen der Parallelität lösen und das Potenzial von Multicore ausschöpfen

von James Reinders



Dieser Artikel erschien vor mehr als elf Jahren in der ersten Ausgabe der Zeitschrift The Parallel Universe. Was im April 2009 richtig war, stimmt heute mehr denn je, zumal die Parallelität in den Rechnerarchitekturen immer weiter an Bedeutung gewonnen hat. Deshalb veröffentlichen wir hier eine übersetzte Fassung der Ratschläge von „Mr. Parallel“ James Reinders.

Die Programmierung von Multicore-Prozessoren bringt ganz neue Herausforderungen mit sich. Die folgenden acht Regeln für die Multicore-Programmierung sollen dabei helfen, diese Herausforderungen zu meistern.

Regel 1: Denken Sie parallel

Nähern Sie sich jedem Problem mit Blick auf den parallelen Anteil. Verstehen Sie, wo Parallelverarbeitung möglich ist, und organisieren Sie Ihr Denken so, dass Sie diesen Anteil richtig beschreiben. Entscheiden Sie sich für den besten Parallelisierungsansatz, bevor sie die anderen Design- oder Implementierungsentscheidungen treffen. Lernen Sie, „parallel zu denken“.

Regel 2: Programmieren Sie mit Abstraktion

Konzentrieren Sie sich darauf, mit ihrem Code Parallelverarbeitung auszudrücken. Vermeiden Sie dabei aber, Code zur Verwaltung von Threads oder Prozessorkernen zu schreiben. Nutzen sie Abstraktionen wie Bibliotheken, OpenMP und Threading Building Blocks (TBB). Verwenden Sie keine nativen Threads (P-Threads, Windows-Threads, Boost-Threads und dergleichen). Native Thread-Bibliotheken sind die Assembler-Sprachen für Parallelismen. Sie bieten zwar ein Maximum an Flexibilität, kosten aber zu viel Zeit beim Schreiben, Debuggen und Warten. Ihre Programmierung sollte daher so abstrakt sein, dass sich Ihr Code um Ihr Problem dreht und nicht um die Thread- oder Kernverwaltung.

Regel 3: Programmieren Sie Tasks, nicht Threads und Cores

Überlassen Sie die Zuordnung von Tasks zu Threads oder Prozessorkernen Ihrem Programm, derjenigen Abstraktion, die Sie für die Thread- und Kernverwaltung verwenden. Erstellen Sie in Ihrem Programm entweder eine Fülle von Tasks oder aber einen Task, der automatisch auf mehrere Prozessorkerne verteilt werden kann, wie zum Beispiel eine Programmschleife in OpenMP. Während der Erstellung von Tasks steht es Ihnen frei, so viele wie möglich zu erstellen, ohne sich über eine Überlastung der Hardware Gedanken machen zu müssen.

Regel 4: Bauen Sie eine Option zum Abschalten der Parallelität ein

Um das Debugging zu vereinfachen, erstellen Sie Programme am besten so, dass sie auch ohne die parallele Ausführung von Tasks getestet werden können. So können Sie die Programme zuerst mit und später ohne Concurrency ausführen, um zu sehen, ob beide Varianten funktionieren – oder auch nicht. Das Testen allgemeiner Probleme ist einfacher, wenn das Programm ohne parallele Tasks auskommt, da uns diese Funktionsweise vertrauter ist und sie besser von den gängigen Test-Tools unterstützt wird. Tritt ein Fehler ausschließlich beim Testen des parallelen Programmes auf, gibt das erste Hinweise auf die Art des zu suchenden Fehlers. Wer diese Regel ignoriert und das Programm nicht zwingen kann, in nur einem Thread zu laufen, wird zu viel Zeit mit der Fehlersuche vertun. Weil das Programm ausschließlich für Testzwecke sequenziell ablaufen können soll, muss dieser einzige Thread nicht wirklich performant sein. Es gilt einzig und allein, keine Programme zu erstellen, die Parallelverarbeitung zwingend benötigen, um korrekt zu funktionieren, wie es zum Beispiel bei vielen Producer/Consumer-Modellen der Fall ist.

Regel 5: Vermeiden Sie die Verwendung von Sperren

Sagen Sie einfach „nein“ zu Locks. Solche Sperren bremsen die Programme aus, schwächen die Skalierbarkeit und sind die Quelle vieler Fehler in der Parallelprogrammierung. Verwenden Sie möglichst implizite Synchronisation für Ihr Programm. Ist doch einmal die explizite Synchronisierung nötig, verwenden Sie atomare Operationen. Nutzen Sie Locks nur als allerletzte Option. Tun Sie ihr Bestes, Programme zu entwerfen, die völlig ohne Locks auskommen.

Regel 6: Verwenden Sie Tools und Bibliotheken, um Concurrency zu implementieren

Beharren Sie nicht hartnäckig auf altbekannten Tools. Bleiben Sie kritisch gegenüber der Tool-Unterstützung mit Blick darauf, wie die Tools die Parallelverarbeitung präsentieren und mit ihr umgehen. Die meisten Tools sind noch nicht reif für die

Parallelverarbeitung. Suchen Sie nach thread-sicheren Bibliotheken. Idealerweise sind das solche, die darauf ausgelegt sind, selbst parallel zu Arbeiten.

Regel 7: Verwenden Sie eine skalierbare Speicherzuweisung

Auf Threads verteilte Programme brauchen unbedingt eine skalierbare Speicherzuweisung. Punkt! Es gibt dafür eine ganze Reihe von Lösungen – und vermutlich ist jede besser als malloc(). Die Verwendung einer skalierbaren Speicherzuweisung beschleunigt die Anwendungen, weil sie globale Engpässe eliminiert, Caches durch die Wiederverwendung von Speicher innerhalb von Threads besser ausnutzt und durch eine saubere Partitionierung Cache-Line-Sharing vermeidet.

Regel 8: Entwerfen Sie Skalierbarkeit durch erhöhte Arbeitslasten

Die Workloads, die Ihr Programm bewältigen muss, werden mit der Zeit wachsen. Planen Sie das ein. Entworfen mit Skalierung mit Blick, wird das Programm mit zunehmender Anzahl der Prozessorkerne auch mehr Last bewältigen können. Jedes Jahr übertragen wir unseren Computern mehr Arbeit. Daher sollten die Programmdesigns mehr Parallelismus zulassen, was bei der Bewältigung größerer Workloads in der Zukunft Vorteile bringt.

Das Beste aus Multicore-Prozessoren herausholen

Ich habe diese Regeln mit der impliziten Erwähnung des Threading vielfach beschrieben. Einzig und allein Regel 7 ist spezifisch auf das Threading bezogen. Threading ist nicht der einzige Weg, um Mehrwert aus Multicore herauszuholen. Oft reicht es, mehrere Programme oder Prozesse parallel auszuführen, wie es insbesondere für Serveranwendungen typisch ist.

Diese Regeln helfen Ihnen, das Beste aus Multicore-Prozessoren herauszuholen. Einige dieser Regeln werden in den nächsten zehn Jahren an Bedeutung gewinnen, da die Anzahl der Prozessorkerne zunimmt und wir eine Zunahme der Vielfalt der Kerne sehen. Beispielsweise wird mit dem Aufkommen heterogener Prozessoren und NUMA die Regel 3 immer wichtiger.

Sie sollten ALLE acht Regeln verstehen und sich zu Herzen nehmen!

„Mr. Parallel“ James Reinders

Chief Evangelist bei Intel und Gründungsherausgeber der Zeitschrift The Parallel Universe.

Bildnachweise

Intel

AI Trendletter

Impressum
|
Kontakt & Anfrage