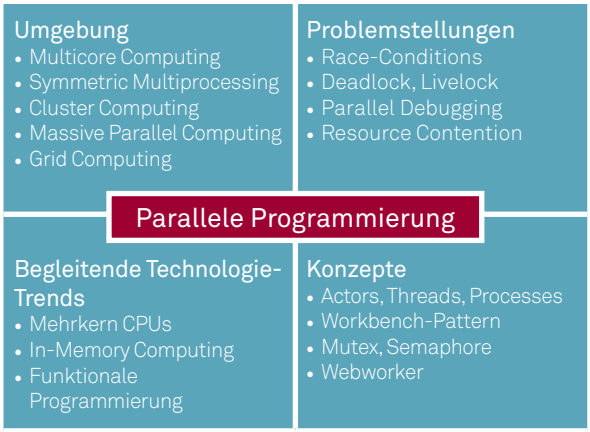
Parallele Programmierung

Trends und Technologien

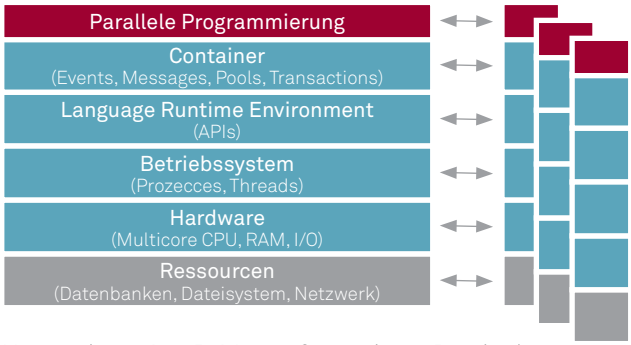
Um das Jahr 2005 startete Intel mit der Auslieferung erster Dual-Core CPUs für den Consumer-Markt. Seitdem hält der Mehrkerntrend unvermindert an, der mittlerweile auch den Mobilgerätemarkt erreicht hat. Gleichzeitig forcieren Trends wie Business-Intelligence die Verarbeitung von extrem großen Datenmengen – Um hier einen Wettbewerbsvorteil erzielen zu können, müssen mehr Daten, genauer und vor allem schneller analysiert werden können.

Definition

Die parallele Programmierung beschäftigt sich mit der Fragestellung, wie Algorithmen entworfen werden können, die verteilte Rechen-Ressourcen durch Nebenläufigkeit optimal ausnützen. Verteilte Rechen-Ressourcen sind z.B. auf Chip-Ebene Mehrkern-CPUs und auf Systemebene Cluster von „rack-mounted" PCs in Rechenzentren. Die parallele Programierung steht im Gegensatz zur klassischen sequentiellen Programmierung. Ziel ist eine Effizienzsteigerung bzgl. der verbrauchten Zeit zur Lösung eines Problems.

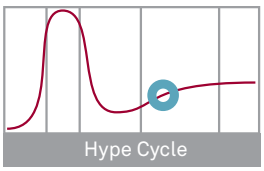


Eine der Herausforderungen, der sich die parallele Programmierung stellen muss, ist der konkurrierende Zugriff paralleler Verarbeitungsstränge auf gemeinsame Ressourcen, z.B. Speicher. Hier muss durch den Einsatz bestimmter Programmierkonstrukte sichergestellt werden, dass der Zugriff auf gemeinsame Ressourcen stets zu einem konsistenten Gesamtzustand des Systems führt. Der Einsatz dieser



Konstrukte, wie z.B. Mutex, Semaphore, Reader/Writer-Locks, verhindert typische Defekte, wie Race Conditions. Gemeinsam ist den Konstrukten, dass sie den parallelen Zugriff bewusst einschränken bzw. verhindern. Diese Synchronisations-Mechanismen selbst führen bei naivem Einsatz zu spezifischen Problemen wie Resource Contention, Deadlocks bzw. Livelocks, die ein System unbenutzbar machen können. Dem wird mit sogenannten Resource-Aquisition-Protokollen begegnet, die den Zugriff regeln. Für den Entwickler bedeuten diese spezifischen Anforderungen eine erhöhte Komplexität in allen Phasen des Entwicklungszyklus.

Die Theorie versucht daher Entwurfsmuster aufzuzeigen, die den Parallelitätsgrad maximiert und dabei die Zahl der Synchronisationspunkte und die damit einhergehenden Gefahren minimiert: Solche Verfahren versuchen ein Problem schnittmengenfrei zu partitionieren, so dass die Teilprobleme keine gemeinsamen Ressourcen benötigen. Der Ablauf gliedert sich in drei Phasen. Er beginnt mit einer sequentiellen Phase zur Partitionierung, gefolgt von einer



parallelen Berechnungsphase. Sind alle Teilprobleme berechnet, erfolgt wieder eine sequentielle Phase, die die einzelnen Teilergebnisse konsolidiert. Ein Beispiel ist das von Google vorgestellte „Map/Reduce"-Verfahren. Neuere Ansätze wie das Actor/Messaging/Event-Model bzw. die Rückbesinnung auf die seiteneffektfreie funktionale Programmierung folgen ebenfalls dem Paradigma „Vermeide gemeinsame Ressourcen" – Sie bieten im ersten Fall dem Entwickler APIs, die es weitgehend unnötig machen, sich explizit mit Locking-Konstrukten auseinanderzusetzen, oder die parallele Verarbeitung wird direkt durch die Eigenschaften der Programmiersprache begünstigt.

Reifegrad

Die parallele Programmierung ist im wissenschaftlichen Umfeld bzw. in rechenintensiven Industrie-Anwendungen (z.B. Automotive-Crash-Simulation) längst gut verstanden. Neuere Trends, wie einleitend genannt die Consumer Mehrkern-CPUs, aber auch Entwicklungen in Plattformen wie die HTML5-WebWorker oder Hadoop setzen sich gerade erst am Markt durch und rücken daher das Thema stärker in den Fokus der betrieblichen Informationsverarbeitung.

Marktübersicht

Programmiersprachen wie Java fokussieren auf die Abstraktion auf Prozess und Thread-Ebene mit typischen Locking-Datenstrukturen und speziellen Datentypen, die für den Einsatz in der parallelen Programmierung geeignet sind. Sprachen wie Erlang und Scala hingegen setzen auf das Actor/Messaging-Konzept.

Alternativen

Eine wirkliche Alternative existiert nicht. Die Beschleunigung von sequentiellen Berechnungen wird zwar auch durch den Einsatz schnellerer Hardware oder Verfahren wie In-Memory-Computing erreicht, allerdings ist hier kein vergleichbar hohes relatives Optimierungspotential mehr zu erwarten. Verschiebt man den Blickwinkel auf die Nutzerseite, so sind Plattformen wie Java Enterprise Edition (JEE) auf das Handling von vielen Benutzern parallel ausgelegt, wobei die Anfragen eines Benutzers im Prinzip sequentiell bearbeitet werden, aber viele Benutzer das System gleichzeitig verwenden können. Die Parallelität ist in diesem Fall in der Plattform verborgen und wirkt sich nur bedingt auf die Programmierung aus.

Referenzszenario

Im Umfeld der betrieblichen Informationsverarbeitung existieren typischerweise Probleme in der Massendatenverarbeitung. Datenbestände sind oft gut fachlich partitionierbar und eignen sich daher gut für Divide-and-Conquer Algorithmen wie Map/Reduce. Hier können z.B. BI-Daten aus den Laufzeitdaten möglichst effizient verdichtet werden, wobei die Ergebnisse schneller zur Verfügung stehen.

Business Impact

Die hochgradig parallele Verarbeitung von sehr großen Datenmengen ermöglicht neue Anwendungsfälle. In Verbindung mit In-Memory-Computing werden plötzlich Realtime-Business-Intelligence-Szenarien denkbar und damit die ad-hoc Analyse von Geschäftskennzahlen.

Pro	Contra
Wirtschaftliche und effiziente Ausnutzung von Rechen-Ressourcen	Programmierung ist komplexer bei Algorithmen, Entwicklung, Test, Wartung, etc. Das bedeutet höheren Aufwand.
Neue Anwendungsfälle möglich	Erfordert erfahrene Entwickler

msg systems ag
Robert-Bürkle-Straße 1 | 85737 Ismaning/München
Telefon: +49 89 96101-0 | Fax: +49 89 96101-1113
www.msg-systems.com | info@msg-systems.com

Stand: März 2013
http://www.msg-systems.com/techrefresh
XT-250-1303-1