Enterprise

Kloksnelheden blijven vrijwel gelijk, maar systemen worden met meerdere processors uitgerust. Om optimaal gebruik te maken van de rekenkracht in een systeem moet een programma worden verdeeld over verschillende processors. Omdat een programma bij het uitvoeren niet altijd gelijkmatig zal worden verdeeld over de verschillende processors, wordt het voor de ontwikkelaar een stuk lastiger om het verloop van een programma te voorspellen. Wat betekent dat voor de Java ontwikkelaar in een multi-core tijdperk?

Ontwikkelen in een multi-core tijdperk

Rino Kadijk
is Java-ontwikkelaar bij Capgemini.

Tien jaar geleden begonnen wij met een 350MHz processor. Ongeveer twee jaar later kochten mijn ouders een tweede PC met een 600MHz processor. Daarmee kochten zij een verdubbeling in snelheid voor ongeveer dezelfde prijs als twee jaar eerder. Die redenering komt behoorlijk overeen met de wet van Moore. Een programma dat een jaar eerder op de 350MHz processor werd uitgevoerd, kon bijna tweemaal zo snel worden uitgevoerd op de nieuwe PC. Helaas zijn processorfabrikanten tot de conclusie gekomen dat de kloksnelheid van een processor onder normale omstandigheden niet zomaar meer kan worden verhoogd. In plaats van het verhogen van de kloksnelheid worden systemen nu met meerdere processors uitgerust. Bij Azul Systems kun je nu al terecht voor Java-systemen met 864 processors (cores) op één machine. Maar ook de hedendaagse laptops zijn al uitgerust met meerdere processors. Volgens Azul Systems zal niet iedereen met parallel programmeren te maken krijgen. Maar, in de toekomst zal zo'n beetje elke programmeur wel op de hoogte moeten zijn van multi-core omgevingen, en zijn redeneren daarop aan moeten passen.

Het voordeel van een multi-core systeem is dat er meer rekencapaciteit beschikbaar is. Het grote nadeel is dat de ontwikkelaar niet gratis meer kan meeliften op de capaciteit in het systeem. Volgens Azul Systems worden veel van de problemen met schaalvergroting bepaald door de karakteristieken van de applicatie. Een multi-core systeem heeft meer rekenkracht, maar het gebruik ervan hangt af van de manier waarop de software is geschreven. Natuurlijk kunnen er wel meerdere programma's onafhankelijk van elkaar worden uitgevoerd, maar dat is niet altijd de snelheidswinst die de koper bij aanschaf van het systeem in gedachten had. Een programma dat meer capaciteit nodig heeft dan de maximale belasting van één processor, zal alleen sneller worden uitgevoerd op een multi-core systeem als de ontwikkelaar het programma daarvoor optimaliseert. De ontwikkelaar moet dus gaan coördineren hoe het programma moet worden verdeeld over meerdere processors. Volgens Gartner staat parallel programming op nummer twee als het gaat om de grootste uitdagingen in de software industrie.

De Wet van Amdahl

$$\frac{1}{(1 - P) + \frac{P}{S}}$$

Formule voor de Wet van Amdahl. 1 is de maximale snelheid per core. De nieuwe snelheid kan worden berekend door de oude snelheid (1 boven de streep) te delen door het gedeelte dat niet parallel kan worden uitgevoerd (1 – P) plus de oude snelheid gedeeld door de mate van versnelling door gebruik van meerdere cores (P/S).

De Wet van Amdahl (zie kader) beschrijft de maximale theoretische snelheidswinst door het gebruik van extra computerbronnen, gebaseerd op een sequentieel en een parallel uitvoerbaar gedeelte. In de praktijk blijkt deze formule lastig in te vullen. In theorie kan nog steeds een hoge snelheid per processor worden gehaald, maar in veel gevallen blijkt het erg complex om een sequentieel algoritme om te zetten naar een parallel algoritme. De snelheidswinst hangt samen met de mate waarin een sequentieel programma kan worden verdeeld over meerdere processors. Twee processors kunnen haast onmogelijk een verdubbeling in snelheid behalen

ten opzichte van één processor, omdat er ook een communicatieprotocol nodig is. Een praktisch voorbeeld is de methode Collections.sort in de JDK. De tijd die het algoritme nodig heeft op een multi-core systeem, schaalt niet lineair met het aantal elementen in de collectie.

Dit is te vergelijken met een team van medewerkers waarin twee identieke werknemers haast onmogelijk de snelheidsverdubbeling kunnen behalen ten opzichte van een team met één medewerker. Een projectteam moet eens in de zoveel tijd overleggen over de voortgang van het project en die tijd heeft een team van één medewerker niet nodig. Uit onderzoek blijkt dat een systeem met meer processors meestal een lagere snelheid per processor levert. Een systeem met één krachtige processor en meerdere kleine processors (asymmetric multicore chips) lijkt in theorie meer snelheid te kunnen leveren dan een systeem met alleen maar kleine processors. In deze vergelijking wordt de invloed van het geheugen niet meegenomen.

Garbage collection en onvoldoende heap memory zijn de eerste bottlenecks

Hoewel de Wet van Amdahl wel degelijk van toepassing is als het gaat om processors, blijkt in de praktijk dat garbage collection en voldoende heap geheugen de eerste twee bottlenecks zijn om op te lossen. Azul Systems gelooft dat het oplossen van garbage collection en object manipulatie de sleutel is tot het schalen op multi-core systemen. Om in de praktijk te schalen op moderne multi-core systemen, heeft Azul JVM zich gefocust op het elimineren van garbage collection en snelle object allocatie en manipulatie. Waarschijnlijk is dit een van de redenen waarom Sun Microsystems met een nieuw garbage collection algoritme (onder de naam G1) voor de HotSpot JVM komt.

Java.util.concurrent

Het is belangrijk dat kloksnelheden, compiler en cache optimalisaties worden verbeterd, want daar profiteren sequentiële en parallelle algoritmes direct van. Om deze veranderingen transparant te houden voor de ontwikkelaar is onder JSR-133 het Java-geheugenmodel verbeterd. Java had van het begin af aan al ondersteuning voor parallelle uitvoering van algoritmes om bijvoorbeeld de gebruikersinterface niet stil te laten vallen. Maar deze instructies werken op een erg laag niveau. Om de ontwikkelaar tegemoet te komen, zijn er onder JSR-166 een aantal objecten toegevoegd aan java.util.concurrent . Bijvoorbeeld ConcurrentHashMap is voor de ontwikkelaar bijna even intuïtief als de andere HashMap's. Maar ConcurrentHashMap presteert veel beter dan een synchronized wrapper in een multi-core omgeving. Een ander voorbeeld is het Executor Framework. Dit framework is geïntroduceerd om de ontwikkelaar meer in de richting van onafhankelijke taken te laten denken, waardoor algoritmes eenvoudiger parallel kunnen worden uitgevoerd.

Ondanks de uitgebreide bibliotheek worden er in Java 7 waarschijnlijk nog een aantal objecten (ForkJoin Framework, RecursiveTask, etcetera) toegevoegd aan java.util.concurrent. Op de vraag hoeveel van de ontwikkelaars java.util.concurrent zullen gebruiken antwoordde Shay Hassidim, Deputy CTO bij GigaSpaces Technologies: "Heel weinig. Alleen ontwikkelaars van applicatieservers zullen java.util.concurrent gaan gebruiken. De gemiddelde Java-ontwikkelaar zal aanroepen doen naar services die ervoor zorgen dat deze parallel worden uitgevoerd op een transparante manier." Waarschijnlijk komen er in de toekomst meer abstracte objecten die aanroepen op een transparante manier parallel uit kunnen voeren.

Ook de Servlet 3.0 API gaat richting een asynchroon model waardoor acties parallel kunnen worden uitgevoerd. Het is nu al zo dat de servlet container een thread per request start. Maar in de toekomst kan het zijn dat de logica binnen een servlet complexer wordt. Daarom zou het handig zijn als de servlet API een mogelijkheid biedt om de logica binnen een servlet op een transparante manier parallel uit te kunnen voeren.

Transactional Memory

Zelfs met de ondersteuning van de objecten uit java.util.concurrent is het lastig om een snel en correct parallel uitvoerbaar programma te ontwikkelen. Het redeneren over het verloop van een programma met een parallel uitvoerbaar gedeelte is veel lastiger vanwege de niet deterministische aard. Om die reden is een parallel uitvoerbaar programma ook lastiger te testen. In de praktijk blijken de meeste fouten in een parallel uitvoerbaar gedeelte pas naar voren te komen bij een hoge belasting van het systeem. Het zou eenvoudiger zijn als de programmeertaal een constructie zou hebben om aan te geven welke gedeeltes parallel uitgevoerd kunnen worden zonder dat de programmeur daar een hoop werk voor hoeft te doen. In de databasewereld worden transacties gebruikt om meerdere mutaties te groeperen. Als er een conflict ontstaat tussen twee transacties, kan één van de transacties ongedaan worden gemaakt en eventueel opnieuw worden uitgevoerd.

Er bestaan ideeën om eenzelfde (meer optimistisch) model ook in programmeertalen te ondersteunen door middel van Transactional Memory waardoor eenvoudiger parallelle programma's kunnen worden ontwikkeld. XSTM is een voorbeeld van een open-source bibliotheek die Transactional Memory ter beschikking stelt. Sun Microsystems lijkt ook (gedeeltelijk) Transactional Memory te ondersteunen met de nieuwe Rock processor. Als alternatief wordt gekeken naar een Hybride model. Een

Hybride model laat zoveel mogelijk over aan de hardware. Als de hardware niet aan de vraag kan voldoen, handelt de software de transactie af. Het ontwikkelen van Transactional Memory is niet eenvoudig. Met name als het gaat om het ongedaan maken van een schrijfactie naar een externe bron zoals een harddisk.
In de Azul Java Virtual Machine wordt Transactional Memory gebruikt, zodat threads die door hetzelfde Java synchronized blok lopen, speculatief en parallel kunnen worden uitgevoerd met behoud van de semantiek. Deze techniek vergroot de schaalbaarheid zolang er geen conflicten tussen threads optreden. De ervaring leert dat deze techniek niet altijd een groot voordeel oplevert. Soms levert het zelfs helemaal geen voordeel op.

Alternatieven

Message passing is een programmeerstijl waarbij berichten worden gebruikt om delen van programma's te synchroniseren. Kilim is een message passing Java framework om parallel mee te programmeren. Het framework is net als Erlang gebaseerd op user-level threads (actors), maar is nog lang niet volwassen. Een actor gebaseerd model lijkt voor de programmeur eenvoudiger te begrijpen dan een mix van de objecten uit java.util.concurrent. GigaSpaces gebruikt een processing unit concept dat ook lijkt op een actor gebaseerd model. Naast actor gebaseerde modellen kijken onderzoekers ook naar speculeren op het niveau van threads. Daarbij worden uitkomsten voorspeld en kan vooruit worden gelopen op het normale verloop. Bij een correcte voorspelling wordt de berekening niet opnieuw uitgevoerd. Dit kan soms winst opleveren, maar kan ook zorgen voor een verlies in snelheid als de voorspellingen niet kloppen.

Patterson van de Berkley Universiteit ziet Transactional Memory niet als de juiste oplossing voor het behalen van meer snelheid. Er zijn maar weinig programma's die echt veel capaciteit nodig hebben en daarom moet volgens hem worden gefocust op bepaalde karakteristieken van programma's. Berkley heeft een overzicht gemaakt van dertien eigenschappen (dwarfs) waar echt veel rekencapaciteit voor nodig is. Deze aanpak richt zich meer op het optimaliseren van bestaande concepten die voldoen aan deze eigenschappen.
Autotuning met behulp van Machine Learning is één van de ideeën waar Berkley wel toekomst in ziet. Autotuning optimaliseert een softwareprogramma zoals een expert dat normaal gesproken doet. Daarnaast denkt Patterson dat het correct vergelijken van systemen belangrijker zal worden om processor en geheugen beter op elkaar af te stemmen. Een processor met een hoge piekcapaciteit hoeft niet altijd goed te presteren onder 'normale' omstandigheden. Om die reden is er een performancemodel ontwikkeld. Met dit model kunnen systemen op een laag niveau beter worden vergeleken. Helaas blijken benchmarks op applicatie softwareniveau zoals SpecJBB tot nu toe vaak erg onrealistisch. Hopelijk komen er in de toekomst manieren om ook op applicatieniveau beter te kunnen vergelijken.

SaaS

Er wordt nagedacht of Software as a Service (SaaS) en Cloud Computing een oplossing zouden kunnen zijn voor de grote hoeveelheid rekencapaciteit die nodig is voor de nieuwe generatie programma's. SaaS biedt een kostenmodel waarbij programma's die veel capaciteit nodig hebben, alleen betalen voor het verbruik. Het systeem met een hoge rekencapaciteit bevindt zich ergens anders in het netwerk. Op het moment dat een klant gebruik wil maken van de dienst, maakt hij verbinding met het krachtige systeem aan de andere kant van het netwerk.
Ook in de datacenters worden multi-core systemen gebruikt. En hoewel aan de server kant natuurlijkerwijs al meer parallelle applicaties worden ontwikkeld, zorgt dat niet altijd voor de beoogde snelheidswinst. "Cloud Computing en SaaS bieden geen technische oplossing voor de schaalbaarheid op multi-core systemen, meer een verschuiving van wiens probleem het is," zo vertellen Cliff Click (Chief JVM Architect)en Gil Tene (CTO) van Azul Systems. Snellere algoritmes worden nog belangrijker als blijkt dat de klant voor het processorverbruik gaat betalen. Een langzamer algoritme wordt op dat moment duurder voor de dienstverlener.

Conclusie

In dit artikel is niet gekeken naar het verschil tussen client en server. Maar aan beide kanten groeit het aantal multi-core systemen. Over het algemeen zal het werk van een ontwikkelaar lastiger worden, omdat een programma niet meer zonder inspanning kan profiteren van capaciteit in een systeem. Daarom zullen we onze denkwijze steeds meer op multi-core systemen aan moeten passen.

Voor het berekenen van de theoretische snelheidswinst kan de wet van Amdahl worden toegepast. Helaas staan garbage collection en voldoende heap geheugen in de praktijk eerst nog in de weg. Java.util.concurrent vereenvoudigt een gedeelte van het werk van de ontwikkelaar en zal ook in de toekomst waarschijnlijk worden uitgebreid. Ondanks java.util.concurrent is de softwareindustrie op zoek naar nieuwe concepten (waaronder Transactional Memory) om het parallel programmeren eenvoudiger te maken. Software as a Service is een veelbelovende ontwikkeling, maar biedt geen technische oplossing om te schalen op multi-core systemen. De parallelle revolutie is begonnen en de gratis performance lunch is voorbij. Welkom in het multi-core tijdperk! «

Referenties

http://kadijk.net/interviews/azulanswers.pdf
http://kadijk.net/interviews/gigaspacesanswers.pdf
http://www.gartner.com/it/page.jsp?id=643117
http://www.youtube.com/watch?v=A2H_SrpAPZU
http://www.gotw.ca/publications/concurrency-ddj.htm
http://www.innoveerjijmee.nl/includes/img.asp/id,141/JavaMemoryModel.pdf
http://www.ibm.com/developerworks/java/library/j-jtp11137.html
http://www.youtube.com/watch?v=1FX4zco0ziY
http://gee.cs.oswego.edu/dl/jsr166/dist/jsr166ydocs/
http://www.cs.wisc.edu/multifacet/papers/tr1593_amdahl_multicore.pdf
http://www.youtube.com/watch?v=KfgWmQpzD74
http://research.sun.com/projects/dashboard.php?id=29
http://www.youtube.com/watch?v=78DF0lpKdJg
http://today.java.net/pub/a/today/2008/10/14/introduction-to-servlet-3.html
http://www.youtube.com/watch?v=37NaHRE0Sqw
http://www.youtube.com/watch?v=IgKYhGa8O6k
http://www.javaworld.com/javaworld/jw-09-2007/jw-09-multicore-processing.html?page=1
Java Concurrency in Practice by Brian Goetz, Tim Peierls, Joshua Bloch, Joseph Bowbeer, David Holmes, Doug Lea
The Art of Multiprocessor Programming by Maurice Herlihy, Nir Shavit