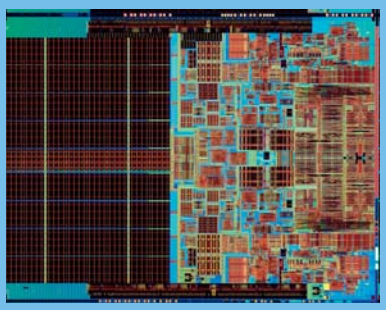
Multicore levert niets op



Het leek een briljant plan: processoren met meerdere cores. De praktijk blijkt echter toch iets grilliger. Door meerdere cores – of rekenkernen – op één chip te plaatsen kunnen we de verwerkingscapaciteit van die processor aanzienlijk verhogen. Immers, iedere core staat gelijk aan een zelfstandige processor. Een snelle rekensom leert dan dat een dualcore systeem dus twee keer zoveel verwerkingscapaciteit biedt als een enkelvoudig uitgevoerde processor. Houd rekening met wat overhead en we zitten bij twee cores op pakweg 190 procent van twee enkelvoudige processoren. Als zoete broodjes gaan ze over de toonbank. Jammer alleen dat de gemiddelde gebruiker daar niets mee opschiet. En dat is ook zo logisch, want in feite is multicore weinig anders dan compacte parallelle verwerking uit de jaren tachtig. Er is uiteraard wel een belangrijk verschil: de processoren die voorheen separaat op borden werden gemonteerd, zitten nu in een en dezelfde chip gebakken. Maar het blijft parallelle verwerking. En als er iets is waar we in de enterprise-wereld de afgelopen jaren weinig vooruitgang hebben geboekt, dan is het wel het programmeren voor parallelle platformen. Zelfs bij standaard applicaties zien we hooguit aarzelend eerste ministapjes richting goed gestructureerde parallelle verwerking ontstaan, maar buiten het technisch-wetenschappelijke wereldje wordt nog nauwelijks rekening gehouden met de mogelijkheid dat een applicatie in stukken kan worden gehakt, waarna iedere brok op een eigen core wordt gezet. In theorie hebben we dan een fikse performance-winst te pakken. Ware het niet dat bij gebrek aan goede methoden om vooraf te bepalen hoeveel tijd een deelverwerking nu precies nodig heeft en hoe we dus gaan voorkomen dat de ene core overuren maakt terwijl de andere stil staat, we nauwelijks van die parallelle verwerkingskracht gebruik maken. Zeker bij maatwerkapplicaties is dit een groot probleem. De synchronisatiemechanismen zijn wel voorhanden, maar het ontbreekt in veel gevallen aan de tools om de applicatiecode zodanig in te delen en te structureren dat wachttijden worden voorkomen. Gartner's advies? Laat die multicore-processoren voorlopig nog maar even zitten.

Ook Chrome wordt pluggable

Helaas zijn veel grote bedrijven niet de eerste die nieuwe technologie of nieuwe producten uitproberen. De kans is dus groot dat Internet Explorer nog altijd de standaard browser is. Zonde, want Firefox en hiervan afgeleide browsers zijn wat functionaliteit vaak al veel verder. Dat is met name te danken aan het feit dat Mozilla een structuur heeft gekozen waarbij individuele ontwikkelaars – en waarom zouden dat geen bedrijven kunnen zijn – zelf extra functionaliteit kunnen toevoegen.
Ook Google Chrome krijgt nu die mogelijkheid. Dat is extra interessant voor bedrijven, aangezien Chromium – de basis van Chrome – eerder een presentatielaag voor webapps kan worden genoemd dan een recht-toe-recht-aan browser. De browser is ontwikkeld met het oog op webapplicaties. De GUI is uiterst licht, terwijl daarnaast een container-achtige aanpak is gekozen waarbij webapps per tabblad in een afgeschermde ruimte 'draaien'. Met andere woorden: valt een webapp om wat voor reden dan ook 'om', dan blijft de browser zelf gewoon functioneren. Het ontwerpdocument staat hier: http://dev.chromium.org/developers/design-documents/extensions

De rubriek Unstructured signaleert nieuwe aan databases gelieerde concepten, veelal op het web. De rubriek staat onder redactie van Robbert Hoeffnagel. Tips, ideeën en commentaar kunt u sturen aan dbm@array.nl.

De database is het probleem

Google is wereldberoemd geworden met zijn aanpak waarbij niet mainframes of zware Unix-servers een hoofdrol spelen, maar simpel whitebox PC's waarop Linux is gezet. Menig IT-directeur zal in een onbewaakt moment wel eens gedacht hebben: waarom doen wij dat eigenlijk niet? Die klassieke mainframes zijn en blijven duur en als Google een architectuur weet te bedenken dat een paar zalen vol PC's goedkoper én sneller zijn dan een Z-series van IBM, moet een beetje bank of financiële instelling dat toch ook voor elkaar kunnen krijgen?

Dat blijkt in de praktijk toch een stuk lastiger dan gedacht. Neem de ervaring van een Amerikaanse verwerker van credit card-transacties. Hier werd de klassieke mainframe-architectuur met losse PC's nagebootst. De theoretisch berekende performance werd echter niet gehaald. Sterker nog: hoewel we dat vooraf wellicht anders zouden verwachten, schaalde de oplossing ook nog eens zeer slecht. Daarmee werd dit project vooral uit academisch oogpunt interessant. Want waarom schaalt het niet? Het probleem bleek de database. De mainframe-applicatie werkte als volgt: het ontving de transactie, schreef deze vervolgens weg naar de database, las de transactie weer uit de database, verwerkte de transactie en schreef het resultaat vervolgens weer naar die database. Met andere woorden: read, write, read, process, write. De architectuur van het mainframe kan dit wat snelheid betreft aan, zodat een acceptabel prestatieniveau bij het verwerken van een transactie wordt gehaald. Bij gebruik van PC-technologie bleek met name de beschikbare bandbreedte veel te gering om een vergelijkbare snelheid te halen. Dus koos men voor een andere aanpak; men ging terug naar Google's aanpak. Wat bleek? De zoekgigant heeft in feite het gehele internet 'in-memory' staan. Het werkgeheugen van al die losse PC's bij elkaar is niets minder dan een enorm gedistribueerde in-memory opslag. Geen lezen en schrijven naar disk maar gewoon alles in geheugen houden. Wordt 'de applicatie' – de zoekmachine – uitgebreid, dan wordt simpelweg extra hardware toegevoegd.

Met andere woorden: het werkgeheugen wordt verder uitgebreid. De rol van de database in deze aanpak? Die van archief.

De credit card-verwerker heeft die aanpak nagebootst. Dat leverde een veel simpeler verwerkingsproces op: read, process, write. Met andere woorden: een transactie wordt ontvangen, in-memory gehouden en direct verwerkt. Pas daarna wordt het resultaat voor archivering naar disk ofwel de database geschreven. Daarmee had deze financiële instelling voldoende aan twaalf standaard PC's om een Z-series mainframe er wat performance betreft uit te concurreren.

Het probleem van de database bleek uiteindelijk verder te gaan dan 'alleen maar' snelheid van lees- en schrijfoperaties. Relationele databases creëren allerlei afhankelijkheden tussen data. Afhankelijkheden en parallelle verwerking gaan echter niet goed samen, tenzij hier zeer goed over nagedacht is. Voor de huidige database-structuren en de huidige applicatiestructuren kunnen we echter niet verwachten dat hiermee rekening is gehouden. Wie naar een Google-achtige architectuur zou willen opstappen kan dus maar twee dingen doen: of alles in-memory brengen of overstappen naar een compleet nieuwe datastructuur.