Programmeertalen

Met de quad-core processoren die Intel en AMD de afgelopen maanden introduceerden, zijn we het many-core tijdperk ingegaan. Het extra prestatievermogen dat we met deze parallelle architecturen onder handbereik hebben, krijgen we echter niet cadeau. Intel ontwikkelt op dit moment een variant op C/C++ die programmeurs moet helpen de nieuwe processoren ook uit te nutten.

Nieuwe programmeertaal Ct moet multi-cores aan het werk houden

Intel ontwikkelt parallelle variant van C++

Hoewel menigeen al blij is met de quad-core processor in zijn nieuwe werkstation, staan we pas aan het begin van het multi-core tijdperk. Intel en AMD werken beide aan de volgende generatie processoren die het dubbele of zelfs het viervoudige aantal kernen moet bevatten.

Tachtig kernen

De Nehalem processor, de opvolger van Intel's huidige Penryn generatie, zal waarschijnlijk van acht kernen zijn voorzien. Elk daarvan kan weer twee threads executeren. Van AMD's Bulldozer worden varianten verwacht met acht en zestien kernen.

Hoewel beide processoren pas over een jaar over twee beschikbaar zullen komen, geeft Sun ons al een voorproefje van de richting die de technologie op dit moment op gaat. Hun UltraSparc T1 ondersteunt acht maal vier is 32 threads. De T2 verdubbelt dat aantal nog eens tot 64.

Als we Intel mogen geloven, zullen processoren in de toekomst honderden of zelfs duizenden threads kunnen verwerken. Hun Tera-scale onderzoeksprogramma is er op gericht om te onderzoeken wat de mogelijkheden van dergelijke architecturen zijn. De eerste echte processor is het afgelopen jaar gedemonstreerd. Deze Teraflops CPU bevat tachtig eenvoudige kernen, met elk twee floating-point units.

Multi-threading

Om de mogelijkheden van de nieuwe multi- en straks many-core processoren ook daadwerkelijk te kunnen gebruiken om problemen op te lossen, volstaan de huidige programmeermodellen niet meer. Ontwikkelaars van HPC-toepassingen (High-Performance Computing) kunnen uitstekend overweg met de bestaande floating-point en vector instructies (SIMD, Single Instruction, Multiple Data). Wie software schrijft voor clusters is ook goed bekend met de mogelijkheden van OpenMPI (Message Passing Interface) en PVM (Parallel Virtual Machine).

Het gebruik van multi-threaded programma's bleef tot nu echter beperkt tot SMP- (Symmetrical Multi-Processing) en NUMA-systemen. Deze technologie komt immers het best tot zijn recht op het shared memory model.

General-purpose-toepassingen

Hoewel threads ook het meest voor de hand liggende gereedschap lijken om de multi-cores te lijf te gaan, valt te betwijfelen of alle kernen van de nieuwe processoren daarmee ook daadwerkelijk aan het werk gehouden kunnen worden. Het is immers meestal geen enkel probleem om server-

A. Offerman,
adrian@offerman.net

applicaties met behulp van threads te parallelliseren. Men spreekt bij dit soort toepassingen ook wel van horizontale schaalbaarheid.

Om datzelfde te doen met general-purpose en desktoptoepassingen is echter veel lastiger. Behalve dat het parallellisme in de daar gebruikte algoritmen niet een-twee-drie voor het oprapen ligt, staan dergelijke applicaties vooral vaak op I/O te wachten. Dat betekent dat de versnelling die general-purpose toepassingen kunnen krijgen, sowieso beperkt zal blijven.

Tenslotte is de trend niet naar coderen dichter op de hardware maar juist andersom. Zakelijke toepassingen worden tegenwoordig in elkaar geklikt in een GUI builder. Productiviteit is belangrijker dan performance. Het is immers goedkoper om meer hardware aan te schaffen dan de extra ontwikkeluren te betalen.

Throughput computing

De oplossing die Intel heeft bedacht om programmeurs te helpen hun code te parallelliseren, heet Ct (C/C++ for Throughput Computing). Deze programmeertaal is onderdeel van het Tera-scale project. Ct moet het vooral gemakkelijker maken om parallellisme te specificeren, zowel van functies als van data.

Om met dat laatste beginnen: Ct biedt de programmeur allerlei nieuwe datastructuren. Naast de bestaande array's zijn er nu ook zogenaamde TVEC's, kort voor Throughput Vectoren, beschikbaar. Denk daarbij aan sparse matrices en associatieve array's. Of aan boomachtige structuren als [[1, 3], 6, [2, 4, 7]] (nested data parallel).

Op deze throughput vectoren zijn operaties gedefinieerd die impliciet parallel op de hele vector werken. Dat kan bijvoorbeeld een matrixvermenigvuldiging zijn of de optelling ervan. Andere operatoren kunnen het maximum van een vector opzoeken, de elementen roteren of ze optellen. Programmeurs kunnen de bestaande datastructuren en bijbehorende operatoren zelf uitbreiden (function overloading).

Futures

De operaties op de throughput vectoren (in het algemeen: collections) zijn op verschillende manieren door de compiler te parallelliseren. Afhankelijk van de structuur en onderliggende data-typen kan de code worden vertaald naar SSE vector-instructies en threads of de veel kleinere fibers.

Ook op functie-niveau biedt Ct de programmeur impliciet parallellisme. De zogenaamde futures zijn functies of kleinere stukjes code met een specificatie van de parameters die ze nodig hebben en hun output. Door hun al hun onderlinge afhankelijkheden te administreren en zo een soort boom op te bouwen, kunnen de futures in de vorm van threads worden geparallelliseerd.

Het grote voordeel van de collections en de futures is dat Ct beschikbaar parallellisme impliciet probeert te vangen in het programmeermodel. Daarmee wordt de programmeur ontlast die tot nu toe het meeste van de threads en hun onderlinge synchronisatie expliciet moest coderen.

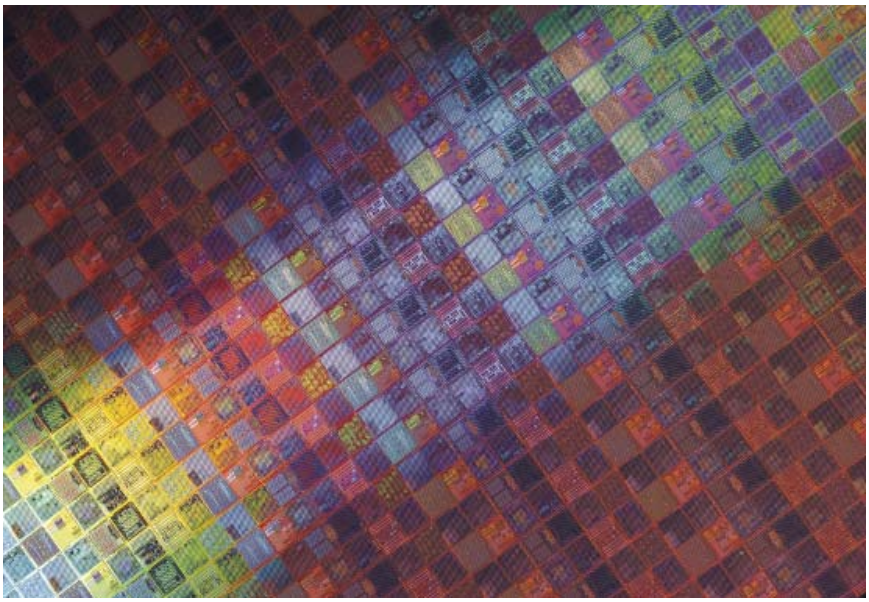
Processor-onafhankelijk

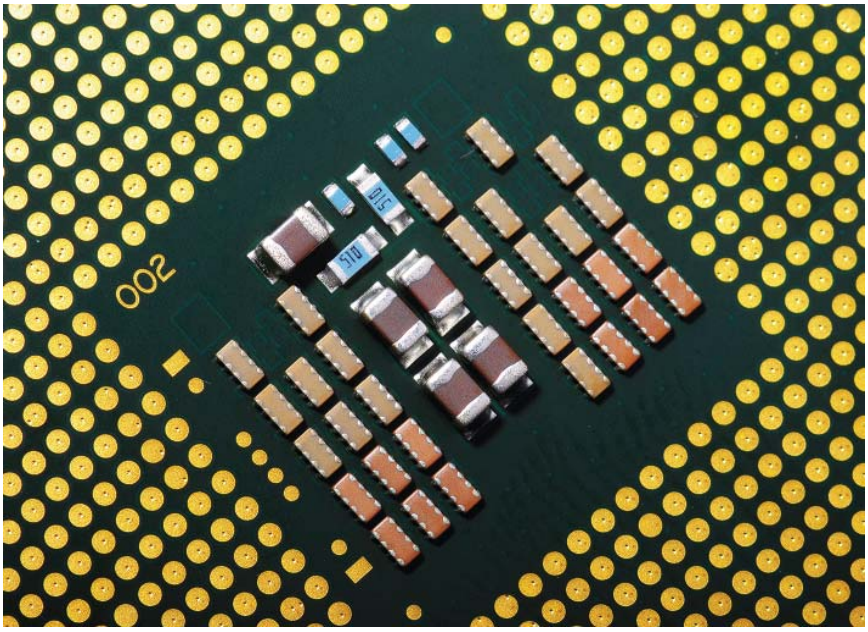
De Ct-compiler zelf bestaat uit de Ct API, een Application Library en een run-time engine. De libraries bevatten calls naar de Dynamic Engine en kunnen met een gewone C-compiler worden gecompileerd. In die eerste stappen wordt de Ct-code vertaald naar een tussenvorm die nog onafhankelijk is van de processor waarop de software uiteindelijk gaat draaien.

Het Virtual Intel Platform (VIP) is gebaseerd op de instructiesets voor X86 en Itanium-processoren. Belangrijkste verschil met echte machinecode voor een specifieke processor is dat de vectorinstructies zijn gedefinieerd in een vorm die onafhankelijk is van de precieze implementatie. Op die manier kunnen ook toekomstige uitbreidingen van de vectorinstructies worden ondersteund. Zo werkt Intel op dit moment aan de implementatie van SSE4.2 en is AMD bezig met SSE5.

De VIP-code wordt uiteindelijk geëxecuteerd door een JIT-compiler (Just-In-Time) en een Threaded Run-Time (TRT), tezamen met de Memory Manager geïntegreerd in de Dynamic

Intel CPU wafer





Een blik binnen de eerste duocore van Intel (foto Dré de Man)

Engine. De VIP Code Generator zorgt dat afhankelijk van het uiteindelijke platform de juiste SSE-libraries en functies worden geselecteerd.

Forward scaling

Minstens net zo belangrijk als de flexibiliteit in de vectorinstructies is de schaalbaarheid in het aantal threads dat door de hardware wordt ondersteund. Je kunt immers niet van programmeurs verwachten dat ze de code die ze nu voor een beperkt aantal threads schrijven straks aanpassen voor tientallen threads, later weer voor honderden threads, en nog weer later opnieuw voor duizenden threads.

Door de Ct-programmeertaal en de VIP-code onafhankelijk te maken van de precieze hardware eronder, kunnen programma's die nu in Ct worden geschreven en gecompileerd over twee jaar ook op een veel grotere architectuur worden gedraaid. Daarbij zou de code zich automatisch over de beschikbare hardware moeten verspreiden. Intel spreekt in dit verband van forward scaling.

Performance penalty

Het concept van Ct doet erg denken aan een parallelle variant van Java, maar dan voor C++. Interessant is dat Intel met zijn VIP-code dicht bij de X86 en Itanium ISA's (Instruction Set Architectures) is gebleven. Dat betekent dat Ct waarschijnlijk niet de performance penalty van Java zal hebben.
Bovendien heeft Intel deze kans aangegrepen om zijn Itanium-processoren weer wat dichter bij de X86 roadmaps te brengen.

Conclusie

Het concept van Ct doet erg denken aan een parallelle variant van Java, maar dan voor C++. Interessant is dat Intel met zijn VIP-code dicht bij de X86 en Itanium ISA's (Instruction Set Architectures) is gebleven. Omdat voor de executie daarvan minder voorwerk hoeft te worden verzet, zal Ct waarschijnlijk niet de performance penalty van Java hebben. Bovendien heeft Intel deze kans aangegrepen om zijn Itanium-processoren weer wat dichter bij de X86 roadmaps te brengen.

Of Ct ons daadwerkelijk gaat brengen wat nodig is om de many-core processoren ook voor general-purpose toepassingen flink bezig te kunnen houden, is nog maar de vraag. Als de programma's zelf maar weinig parallellisme bevatten, zijn het draaien van meerdere processen tegelijk en virtualisatie de enige manieren om een many-core systeem bezig te houden. Juist de onafhankelijkheid van de processen en virtuele machines die het zo makkelijk maakt om ze parallel te draaien, veroorzaakt echter een bottleneck naar de kritieke secties van het operating system en het geheugen. Meer hierover kunt u lezen in een volgend, uitgebreider artikel, dat zal verschijnen in SRM 3/2008. Ook vindt u meer informatie over dit onderwerp op de site van Computable: http://tinyurl.com/ywqdpa

Sun loopt voorop met multicore processoren. (foto: Sun)

Referentie

http://techresearch.intel.com/articles/Tera-Scale/1421.htm