



Multicore – Möjliga scenarion för framtiden

Christoph Kessler, Linköpings universitet

Sedan 1967 förutsäger *Moore's Lag* pålitligt en exponentiell tillväxt av antalet transistorer per chip, ungefär med en faktor 2 vartannat år, och denna trend kommer av allt att döma också att fortsätta under det nästa årtiondet. På grund av energi- och värmeproblemet kan däremot taktfrekvensen inte längre ökas nämnvärt framöver; istället kan vi förvänta oss en exponentiell tillväxt av antalet kärnor per chip. Men idag är det långt ifrån klart hur framtidens many-core arkitekturer kommer att se ut för att uppnå en bra balans mellan prestanda, energiförbrukning och programmerbarhet. De följande avsnitten visar några möjliga scenarion för arkitektur och programmering av kommande many-core processorer.

Arkitekturkoncept

Idag ser vi, grovt förenklat, två olika ansatser att organisera multicore-arkitekturer: Homogena och heterogena multicores.

Homogena multicore-processorer innehåller flera kopior av ett existerande core-design, som oftast redan stöder flera hårdvarutrådar per kärna, och kärnorna delar L2-cacheminne på chipen. Några aktuella exempel är Intels quadcore-Xeon Core™ i7 eller Suns

UltraSPARC™ T2-Niagara med 8 kärnor och 64 hårdvarutrådar. Programmeringsmodellen är SMP (symmetric shared-memory multiprocessor) som är relativt komfortabelt för programmeraren, även om skalbarheten är begränsad genom bandbredden till off-chip minnet. Standard-operativsystem och programmeringsverktyg kan användas. Men utan parallellisering (t.ex. trådning) uppsnabbas enstaka program inte, och då antalet av oberoende program som vanligtvis kan köras samtidigt är mycket begränsat i praktiken (ungefär 4 på en vanlig laptop) har man knappast nytta av de tiotals kärnor och kanske hundratals hårdvarutrådar som vi snart kan förvänta oss – programvaruindustrin måste därför på sikt leverera parallelliserad mjukvara, annars skulle vi helt enkelt inte ha råd med ytterligare tekniska framsteg som kräver mer prestanda.

I *heterogena multicore-system* är de olika kärnorna specialiserade på olika uppgifter och har ofta olika instruktionssatser. En eller ett fåtal general-purpose kärnor som används framförallt för att köra sekventiella programdelar och operativsystemet kompletteras med mer eller mindre programmerbara acceleratorer för att

snabba upp beräkningar av en viss typ, oftast dataparallella beräkningar där samma operation körs på alla element av en array eller dataström. Detta koncept passar utmärkt för exempelvis vetenskapliga beräkningar, digital signalbehandling, bildbehandling, grafik, spel, datakompression, kryptering, mönstermatchning mm. Ett typiskt exempel är Cell Broadband Engine™ av Sony/Toshiba/IBM, som kombinerar en standard-PowerPC™ kärna med 8 slavprocessorer som är optimerade för dataparallella beräkningar på instruktionsnivå (SIMD). Beräkningar som kan skrivas om för att köras effektivt på slavprocessorerna istället för masterkärnan kan uppnå en 1 – 2 storleksordning högre prestanda med ungefär samma energiåtgång. Också GPU:ar, som stöder dataparallella beräkningar med hundratals hårdvarutrådar, kan räknas till heterogena system då de avlastar standardprocessorn i systemet. Tyvärr är effektiv programmering av heterogena multicoresystem oftast mera komplicerad än för homogena multicoresystem eftersom flera nivåer av parallellitet måste koordineras (t.ex. för Cell: trådning över slavprocessorerna, SIMD-

parallellisering, instruktionsparallellitet, och överlappning av beräkningar med minnesaccesser), oftast manuellt, för att komma bara i närheten av maximalprestandan.

Ett möjligt scenario för den närmaste framtiden är en viss konvergens av homogena och heterogena multi-/many-core system. Ett fåtal standard-processorkärnor som är optimerade för sekventiell kod behövs för att köra sekventiella programdelar, input-output etc; dessa kombineras med kanske hundratals enklare processorer för välparallelliserbara programdelar samt acceleratorer för specialuppgifter; hela systemet använder sig av samma basinstruktionssats. Intels Larrabee-processor är ett aktuellt exempel på en börjande konvergens av CPU- och GPU-funktionalitet.

Ingen exponentiell tillväxtkurva kan fortsätta för evigt. Enligt nuvarande prognoser kan ökningen av chip-integrationen och därmed Moores Lag med hjälp av olika tekniska förbättringar fortskrida i kanske 10 år till, sen nås grundläggande fysikaliska gränser (atomnivå), så att man måste byta till en helt ny teknologi. Ingen vet idag vad som kommer därefter som

kan säkerställa en fortsatt prestandatillväxt. Dynamiskt rekonfigurerbara element på chipen kan vara en möjlighet att förbättra befintlig teknik. Eller kanske är kvantdatortekniken då redo att ta över? Kan nya halvledarmaterialier eller organiska molekyler, nanotubes mm ersätta silicium? Ett mer konventionellt scenario är att man vid behov skapar större enheter med flera chips igen, såsom CC-NUMA och kluster – under antagandet att IT-världen har emellertiden klarat övergången till parallellprogrammering skulle det vara ett naturligt val. Mest sannolikt är kanske en allmän övergång till cloud-computing, dvs bara viss grundfunktionalitet (t.ex. en browser) behövs i våra framtida laptops, smartphones mm, medan själva beräkningskraften och data samlas i ett nätverk av servercentra.

Men med hundratals kärnor per chip om några år har vi nog viktigare problem att lösa än att hur man kan få ännu fler. Bandbredden till off-chip minnet är redan idag en flaskhals, och detta problem förvärras med ett växande antal kärnor och hårdvarutrådar. Dessutom ökar frekvensen för transienta hårdvarufel (dvs enstaka bits byter felaktigt

mellan 0 och 1) avsevärt; kanske måste en signifikant andel av de ytterligare hårdvaruresurserna avsättas bara för att säkerställa feltolerans.

Den främsta utmaningen är däremot programvaran – programspråk, kompilatorer, bibliotek, applikationer. Hur kan man programmera manycore-processorer och få prestanda, portabilitet, abstraktion och produktivitet samtidigt? Ett möjligt scenario lite längre fram är utvecklingen av hårdvarustöd för enklare och starkare parallella programmeringsmodeller såsom BSP (bulk-synchronous parallelism) ovanpå processorns nativa programmeringsmodell. Förutom förbättrad portabilitet och tidsförutsägbarhet har dessa modeller egenskaper såsom deterministisk parallell programexekvering och strikt minneskonsistens som känns naturligare för programmeraren. Vissa effektivitetsförluster i realiseringen av dessa modeller kan accepteras i utbyte mot högre programmerarproduktivitet.

Programmering

Utvecklingen av parallella program med hjälp av parallella algoritmer, datastrukturer och programspråk,

liksom parallellisering av befintlig sekventiell programvara är generellt svår och felbenägen, och kan bara till en viss del automatiseras med hjälp av verktyg. Överhuvudtaget kan man bara få en märkbar uppsnabbning om det finns tillräckligt mycket utnyttjbar parallellitet under den större delen av beräkningen – dvs om den sekventiella andelen är relativt liten (*Amdahls Lag*). Även parallella algoritmer måste anpassas, t.ex. genom att minimera antalet accesser till off-chip minne.

De mest prestandakritiska komponenterna kommer tills vidare att skrivas i nativa programspråk såsom C och C++, parallella delar givetvis på relativt låg abstraktionsnivå med trådbibliotek, SIMD-intrinsics mm, men så småningom förhoppningsvis också i nya programspråk med inbyggd explicit parallellitet, såsom X10 eller OpenMP för homogena och OpenCL för heterogena multicoresystem. Fler nya multicoreprocessorer såsom Sun Rock kommer också ha hårdvarustöd för så kallat *transaktionsminne* – spekulativ parallellisering av exekveringen av kritiska sektioner. Detta kan ersätta lås-baserad synkronisering och löser därmed flera problem såsom den inherenta

sekventialiseringen av lås-skyddade kritiska sektioner eller risken för deadlocks.

Parallellisering är ett kostsamt sätt att snabba upp och att framtidssäkra mjukvaran, men på lång sikt har vi väl inget annat val. Ändå är det viktigt att påpeka att det finns fortfarande mycket uppsnabbningspotential i valet av snabba (sekventiella) algoritmer, rätt kombination av kompilatoroptimeringar och överhuvudtaget ett mer prestandamedvetet sätt att programmera – speciellt för de icke-parallelliserbara delarna av programmet. Detta ställer nya krav på utbildningen av programmerare, men också på programspråk- och kompilatorutveckling och mjukvarudesign.

På många universitet världen över diskuteras fortfarande om och i vilken form man ska ta upp parallellprogrammering samt parallella algoritmer och datastrukturer i datatekniska utbildningsprogram – och därmed lägga till ännu en ny dimension av komplexitet i programmeringen. I bästa fallet, argumenterar somliga, räcker det med att ett fåtal expertprogrammerare utvecklar

optimerade biblioteksrutiner som kapslar parallelliteten, speciellt för heterogena multicoresystem och specifika tillämpningsområden med en hög grad av återanvändning. Å andra sidan är den sekventiella von-Neumann-beräkningsmodellen som vi använde i 60 år inte mera naturligt än motsvarande parallella modeller, heller tvärtom: Det går alltid att automatiskt sekventialisera ett parallellt program, medan den omvända riktningen kan vara svårt eller rentav omöjligt att automatisera. Att lära sig ett parallellt "tänkande" från grunden är en bra idé, och då är det bäst att börja med det snarast.

Kompilator teknik har en nyckelroll både vid och efter övergången till parallellprogrammering, t.ex. genom att utnyttja en större del av den kvarvarande optimeringspotentialen i sekventiell kod, stödja utvecklingen av nya parallella programspråk, skapa verktyg för analys, optimering och verifikation av ny parallell kod och likaså för analys, optimering och parallellisering av sekventiell legacy-kod. En klar trend går mot dynamiska och spekulativa optimeringar. Komplexiteten av moderna datorarkitekturer (såsom djupa minneshierarkier) gör det allt svårare att välja rätt kombination av

optimeringar för plattformen. Lösningen är *autotuning*, en ny teknik som låter t.ex. kompilatorn lära sig själv från observationer på exempelkod vilka transformationer som ska genomföras, och i vilken turordning, för att få bra prestanda vid framtida kompileringar. Ansatsen kommer ursprungligen från biblioteksgeneratorer som automatiskt optimerar specifika rutiner såsom matrisoperationer, FFT eller sortering, för en konkret exekveringsplattform.

Komponentteknologin är redan välbeprövad inom mjukvaruutvecklingen och kan erbjuda nya uppslag för ett effektivt utnyttjande av multicore-plattformar: I ett nytt forskningsprojekt undersöker vi hur komponentgränssnitt kan utökas med prestanda-metadata så att komponenter redovisar hur de använder systemresurser, t.ex. hur deras implementation skalar i antalet hårdvarutrådar. Det finns speciella punkter i koden, såsom anrop av komponentfunktioner, där de tillåter kontrollerade dynamiska adapteringar t.ex. i resursanvändandet eller genom att välja den förväntat snabbaste bland flera ekvivalenta komponentfunktioner, baserad på

aktuellt tillgängliga resurser och aktuella parametervärden. Med andra ord, komponenter blir mer grey-box, mer adaptiva och mer förutsägbara i sin prestanda.

I den "gamla" sekventiella tiden var ett programspråk såsom C/C++ en slags universell plattform med hög portabilitet, det räckte ofta att kompilera om för att få exekverbar kod för en ny processor eftersom både implementerade den sekventiella von-Neumann-programmeringsmodellen. Dagens och troligtvis också framtidens multi- och manycoreprocessorerna har ingen motsvarande universell programmeringsmodell – somliga använder delat minne, andra meddelandebaserad eller channel-baserad kommunikation mellan kärnorna osv. Kompilatorn kan inte förstå programmerarens intentioner för att abstrahera från de använda konstrukterna och automatiskt portera koden effektivt till en annorlunda parallell plattform. En konsekvens är att (även nya, parallella) programspråk har för låg abstraktionsnivå för att kunna garantera universell portabilitet över hela spektrumet av framtidens parallella system. En lösning på längre sikt kan vara *modelldriven utveckling*: Existerande modelleringskoncept för

sekventiella (t.ex. state machines) och jämlöpande (t.ex. Petri-nät) beräkningar kompletteras med nya modelleringsselement för explicit parallella beräkningar, dvs parallellitet redovisas explicit, men dess utnyttjande på en specifik manycore-processor specificeras separat med plattformspecifika mappnings- och optimeringsverktyg och källkodsgenerering. På det sättet kan exekverbara modeller bli den nya plattform-oberoende "källkoden" för framtidens parallella processorarkitekturer. Men här behövs fortfarande mycket forskning och utveckling för att skapa rätt modelleringspråk och verktyg för detta ändamål.



VI ÄR ALLA KONSULTER PÅ COMBITECH, ett av Sveriges största konsultföretag som kombinerar teknik, miljö och säkerhet. Vi är ca 800 medarbetare på 20 orter.