ELEKTRONIK TIDNINGEN

Tomas Evensen
Teknichef
Wind River

Multikärnor – utmaningarna och hur de kan mötas

Hypervisorer, SMP, AMP och exekvering direkt på hårdvaran utan mellanhänder – det är några teknologier du måste behärska om du önskar få full kapacitet ur multikärnorna.

Redaktör
Jan Tångring
jan@etn.se
0734-17 13 09

EMBEDDED EXPERT

© Elektroniktidningen och Wind River Systems, 12 januari 2010

www.etn.se/expert — kostnadsfria tekniska rapporter

Multikärnor – utmaningarna och hur de kan mötas

Hypervisorer, SMP, AMP och exekvering direkt på hårdvaran utan mellanhänder – det är några teknologier du måste behärska om du önskar få full kapacitet ur multikärnorna.

Av Tomas Evensen, Wind River Systems

Tomas Evensen är teknikchef på Wind River med ansvar för företagets kommande teknik och arkitektur. Tomas chefar även över Vx Works Engineering och alla lanseringar av VxWorks-plattformar. Han har sin grundutbildning på KTH och var under 15 år vd för svenska Diab Data.

Det sker just nu en parallell utveckling på ett antal fronter – teknologiska och kommersiella – som ändrar förutsättningarna för hur man utvecklar och konstruerar inbyggda system:

- klockfrekvenstaket
- multikärnor för avlastning av hårdvara och konsolidering av multiprocessorsystem
- hypervisorer
- ökade krav på prestanda och funktionalitet i samma produkt
- högre krav på säkerhet och sekretess, inklusive certfiering.

Den gemensamma nämnaren för alla dessa utvecklingslinjer är multikärnor och virtualisering.

Klockfrekvensen planar ut

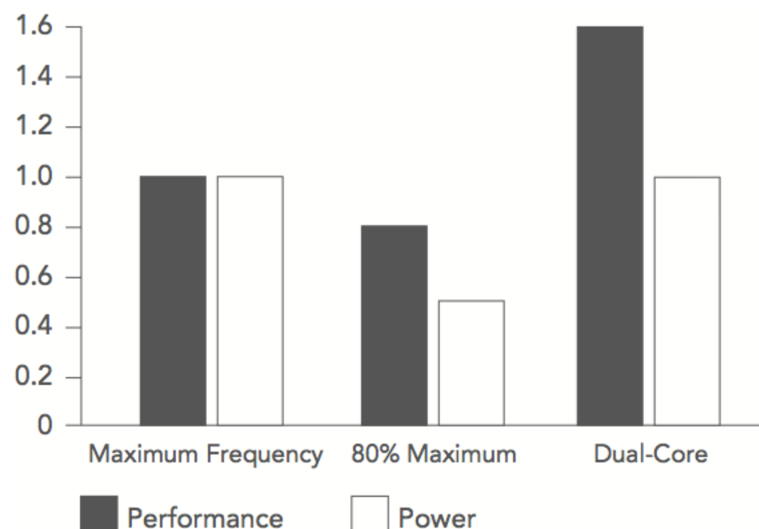
Om du tittar längst upp i högra hörnet på en godtyckligt vald produktplanering så ser du numera så gott som alltid en multikärna – en processor med flera cpu-kärnor.

Vadan detta plötsliga intresse för multikärnor?

Svaret är att det är fysikens lagar som tvingar fram det. Historiskt har processormakare länge kunnat höja prestanda genom ökad klockfrekvens. Om det inte varit för att processtorlekarna samtidigt krympt, skulle energiförbrukningen ha ökat med en ännu högre faktor.

För några år sedan började utvecklingen plana ut. Samtidigt som kravet på ökande prestanda låg kvar. Arkitekterna fick då börja ta till finurliga knep för att höja prestanda och fick offra allt mer kisel på samma beräkningar: **superrörledningar** (super pipelines, instruktioner delas upp i mindre steg som vart och ett kan exekveras snabbare), **superskaläritet** (flera instruktioner startas på samma klockcykel), **branch prediction** (gissa i förväg vilken kod som kommer att exekveras för att minimera den negativa effekten av långa rörledningar), **registeralias** (virtuella processorregister i rörledningen som låter dig parallellisera mer), med mera.

Idag har vi kommit till en punkt då investeringarna i högre frekvens och ansträngningarna att parallellisera en



Figur 1 Hur prestanda och effekt förhåller sig till klockfrekvens

12 januari 2010 © Wind River Systems och Elektroniktidningen Sverige AB

i grunden seriell kodsekvens har börjat ge alltför låg avkastning. Man skulle i princip fortfarande kunna höja prestanda lite till, men till priset av dramatiskt ökad energiförbrukning.

Tre faktorer gör att detta händer just nu. Den första är att strömförbrukningen är direkt proportionell mot frekvensen. Den andra att du måste köra på högre spänningsnivåer för att få upp frekvensen, och strömförbrukningen är proportionell mot kvadraten på spänningen. Den tredje är att ju snabbare du klockar processorn desto större blir skillnaden mellan processorhastighet och busshastighet. Det innebär att du måste betala ett pris i form ökad transistorvolym. Dels för att implementera nyss nämnda smarta knep och dels för att öka cachestorleken. Och fler transistorer betyder högre energiförbrukning.

Figur 1 visar vad som sker med strömförbrukningen vid ökande frekvens. Det första stapelparet visar prestanda och energiförbrukning för en processor som kör i full frekvens. Det andra paret visar att om du minskar klockfrekvensen med 20 procent, så i det närmaste halveras effektanvändningen medan du fortfarande får ut det mesta av prestandan. Det sista stapelparet visar vad som händer om du lägger till ytterligare en kärna i den lägre frekvensen: strömförbrukningen blir ungefär densamma som för den kärna som körs i full takt, men prestanda blir i detta fall 60 procent högre.

Detta är vad man teoretiskt kan förutspå. Och det låter ju fantastiskt bra! Så varför har inte vi gjort det för länge sedan?

Svaret är att vi faktiskt har gjort det. Multiprocessorteknik har redan flera decennier på nacken. Skillnaden var att det förr användes i första hand för att få ut mer prestanda. Dessutom byggde man förstås på den tiden systemen av separata processorchips och ibland multipla kretskort. Det är först nyligen som det blivit tekniskt möjligt och motiverat att sätta flera kärnor på samma chip.

Ur hårdvaruperspektiv är det här en superb lösning – fler MIPS på mindre ström. Problemet är att det är svårt att komma åt att utnyttja den ökade prestanda som teoretiskt ska finnas där. På den gamla goda tiden – innan vi slog i frekvenstaket – gick det normalt att återanvända programvara i nästa processorgeneration. Och den exekverade allt snabbare varje gång. Idag måste man addera processorkärnor att höja prestandan. För att kunna utnyttja dem krävs parallellt exekverande programtrådar.

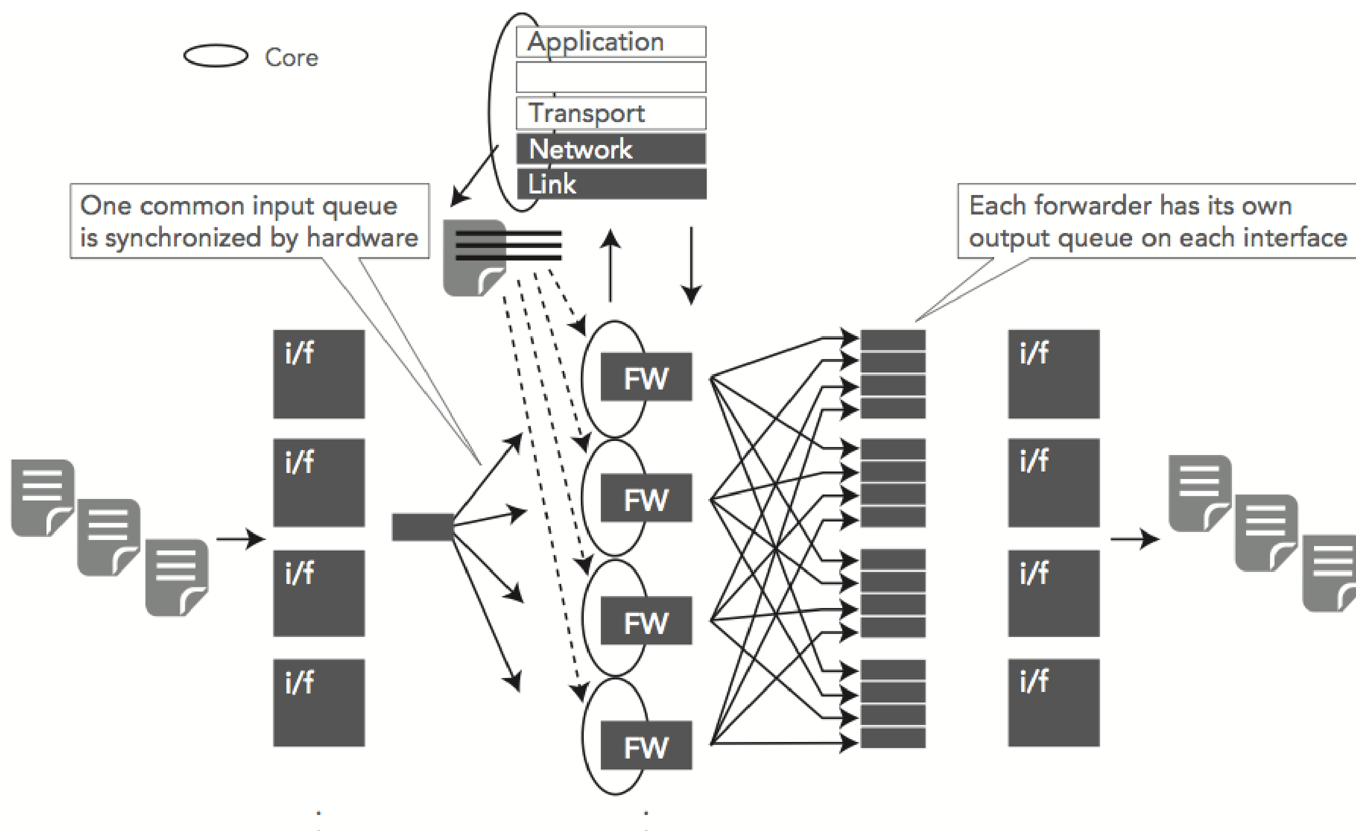
Problemet är att i många, kanske de flesta, inbyggda applikationer kör processorn en och samma tråd under merparten av tiden. Om du i det läget lägger till en processorkärna kommer den bara att sitta sysslolös medan tråden fortsätter att underhålla den första kärnan.

Utmaningen att få prestanda ur en multikärna ligger i att få tillämpningen att exekvera programkod parallellt. Generellt är det en svår utmaning, men det finns vissa väl förstådda specialfall.

Multicore avlastar hårdvara

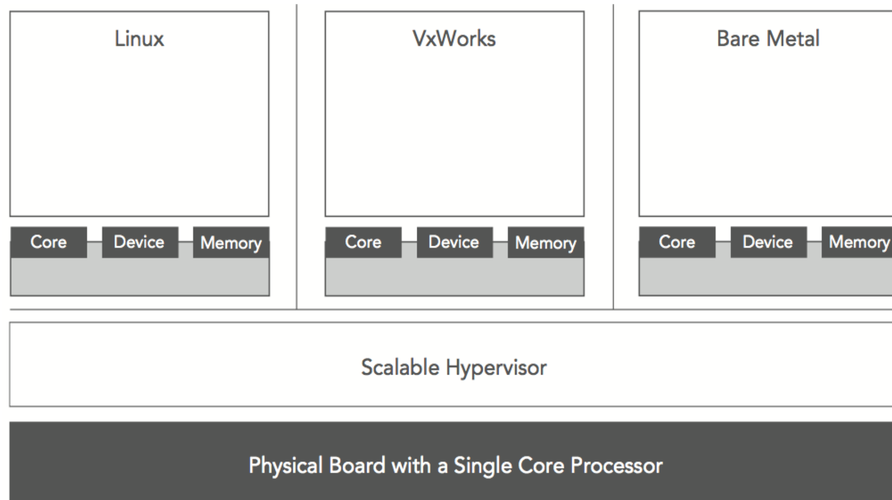
Ett beprövat och effektivt sätt att sysselsätta multikärnor är att ge dem arbetsuppgifter som tidigare utförts i särskild hårdvara som specialanpassade processorer, fältprogrammerbara grindmatriser (FPGA) eller applikationsspecifika integrerade kretsar (ASIC). Eftersom dessa uppgifter redan en gång implementerats i separata enheter, är många parallelliseringsfrågor redan lösta. Genom att använda flera stycken relativt långsamma kärnor, som samtidigt är uppbackade av lågnivåstöd i hårdvara för specifika uppgifter, kan du på detta sätt få ut god parallell prestanda. Det är också ett system som är relativt enkelt att konstruera och implementera jämfört med äldre metoder.

Ta en router som exempel. Den viktigaste uppgift den har är att undersöka inkommande datapaket och bestämma vilken port de ska vidarebefordras till. Här kan några parallellt exekverande kärnor ge hög kapacitet, om de får en smula assistans av dedikerad hårdvara



Figur 2 **Så här kan en multikärna avlasta hårdvara i en routertillämpning.**

12 januari 2010 © Wind River Systems och Elektroniktidningen Sverige AB



Figur 3 En hypervisor som låter virtuella kort dela processorkärna.

för bufferthantering, krypering och andra väldefinerade uppgifter (se **figur 2**).

Multicore konsoliderar multiprocessorsystem

Det finns många typer av utrustning som redan använder multiprocessning för att höja prestanda eller för att kombinera system med olika krav. Självklart vill man i sådana system idag använda billigare, mindre energikrävande lösningar i form av multikärnor.

Samtidigt är en multikärna mer än ett antal processorenheter ihoptryckta till ett chip. Det finns skillnader som gör migreringen från multiprocessor till multikärna icketrivial.

En av de viktigaste är att separationen mellan processor är mer distinkt i ett flerprocessorsystem. Normalt sitter en väldefinierad buss mellan processorerna. Och även om de har gemensamt externt minne, exekverar de oberoende av varandra.

Så är inte fallet i en multikärna. Beroende på arkitektur, delar kärnorna på flera saker, till exempel avbrottsenheter, enheter och cachar. Vid symmetrisk multiprocessning (när man kör ett gemensamt operativsystem) är detta en fördel. Men det är en nackdel när två separata operativsystem körs, till exempel ett realtidsoperativsystem (RTOS) som Wind Rivers VxWorks tillsammans med ett generellt operativsystem som Linux.

När du kör flera operativsystem på en multikärna, behöver du antingen operativsystem som vet mycket om varandra, som i ett master-slaveupplägg, eller så kan du förenkla livet genom att använda en **hypervisor** vilket är ett program som administrerar körningen av andra operativsystem, så kallade **gästoperativsystem.**

Hypervisorer för inbyggda system

En hypervisor (eller supervisor) är ett program som kör och administrerar en eller flera instanser av olika operativsystem. Tekniken är att skapa partitioner (eller virtuella processorkort) som körs ovanpå hypervisorn. Partitionerna skapas genom att man virtualiserar olika aspekter av systemet. Generellt kan tre beståndsdelar virtualiseras:

1. CPU: Flera virtuella processorer kan köras ovanpå en fysisk CPU eller kärna. De körs under tidsdelning eller under en prioritetsbaserad algoritm som ger varje virtuell processor en andel av de fysiska processorcyklerna.
2. Minne: Du kan dela upp det fysiska minnet så att olika partitioner använder varsin del av det fysiska minnet. Ett operativsystem som körs under en sådan partition kan dessutom använda virtuellt minne för att implementera processer. I det fallet får man en minneshierarki i tre nivåer: en för hypervisorn, en för OS som körs i det virtuella kortet, och en för program som körs i en process.
3. Enheter: När virtuella kort behöver dela på enheter som seriella portar, Ethernet-portar och grafik, måste också dessa virtualiseras. Detta sker vanligen via ett gränssnitt så att operativsystemet kan göra API-anrop istället för att fysiskt accessa enheten. Den faktiska koden som hanterar enheten kan husera på ett av två ställen, antingen i hypervisorn eller en av gästoperativsystem.

Ett mycket vanligt fall när du använder en hypervisor på en multikärna är att du inte lägger multipla partitioner på en ensam kärna utan tvärtom tilldelar en eller flera kärnor ett enda virtuellt processorkort. Denna typ av hypervisor kallas **supervisor**. Med en supervisor behöver du inte

Symmetrisk och assymmetrisk multiprocessning

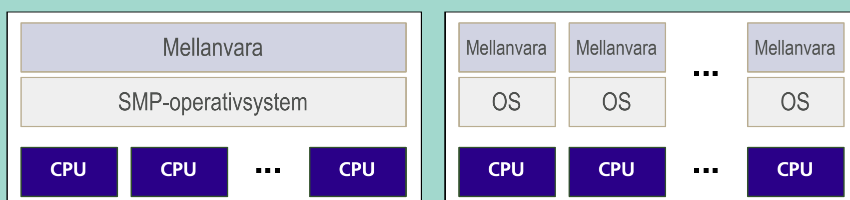
Det finns två grundläggande metoder för operativsystemet att administrera kärnorna i en multikärna. **Symmetrisk multiprocessning** (SMP) innebär att ett enda operativsystem styr flera kärnor. När en kärna blir ledig går den till kön och hämtar nästa programtråd som är redo att köras.

Vid **assymmetrisk multiprocessning** (AMP) använder man istället ett operativsystem per kärna. De kan vara två kopior av samma operativsystem eller två helt skilda operativsystem, som ett RTOS som VxWorks plus ett generellt OS som Linux.

Det finns för- och nackdelar med båda strategierna, och det bästa valet beror på tillämpningen. SMP har fördelen att man kan lastbalansera trådar över processorkärnorna så länge som det finns fler körklara programtrådar än processorkärnor. AMP är å andra sidan effektivare eftersom mindre synkronisering behövs mellan kärnorna.

För att få högre förutsägbarhet och prestanda i ett SMP-system kan man oftast använda så kallad **affinitet** – att låsa en specifik tråd till en specifik kärna. Det minimerar de negativa cacheeffekter som uppstår när en tråd ska migreras från en kärna till en annan. Å andra sidan förlorar du med affinitet möjligheten att lastbalansera trådar.

AMP används typiskt tillsammans med en supervisor eller hypervisor som hanterar gemensamma resurser och skyddar operativsystem från varandra för att förhindra att ett fel på den ena påverkar den andra.



Figur 4 Symmetrisk multiprocessning (SMP) ... och assymmetrisk (AMP)

12 januari 2010 © Wind River Systems och Elektroniktidningen Sverige AB

tidsdela kärnorna mellan partitionerna, och därmed kan motsvarande operativsystem exekveras mycket effektivare. Du bör dock fortfarande virtualisera de två andra aspekterna – minne och enheter – för att se till att du får fullt skydd mellan operativsystemen, så att de kan dela enheter.

Det finns två sorters hypervisorer. **Typ 1** är dedicerad, liten och körs direkt på hårdvaran. **Typ 2** körs typiskt ovanpå eller i kombination med ett fullskaligt värdoperativsystem och använder dess resurser.

En bra hypervisor för inbyggda system är av typ 1, medan serverhypervisorer typiskt är av typ 2.

Skillnaderna mellan en inbyggd hypervisor (som Wind River Hypervisor) och en serverhypervisor (som VMware eller Kernel-based Virtual Machine, KVM) beror på olika krav. I en inbyggd hypervisor är prestanda och isolering de två främsta drivkrafterna, medan bakåtkompatibilitet (att kunna köra ett gäst-OS oförändrat) är ett starkare krav på en serverhypervisor.

Kraven på inbyggda hypervisorer märks i hur de konstrueras:

- IO-enheter avbildas hela vägen in till gästoperativsystemet, för bästa prestanda och för separation.
- Hypervisorn är skalbar i meningen att det är optionellt vilka aspekter som virtualiseras, för att man ska kunna göra en avvägning mellan prestanda och isolering.
- Hypervisorn ska vara typiskt vara liten för att bli snabb och för att förenkla certifiering.
- Gästoperativsystem ska typiskt vara paravirtualiserade. Detta innebär att de modifieras för att höja prestanda.

Hur en hypervisor implementeras varierar med hur mycket stöd hårdvaran ger. Vissa moderna processorer implementerar trippelnivåminnessystem i hårdvara. Detta kan potentiellt minimera prestandaförlusterna som en hypervisor ger. Men en hypervisor kan också vara snabb på processorer utan specifikt hårdvarustöd, om paravirtualiseringen lagts på rätt nivå.

Utöver den grundläggande utmaningen att dela upp koden för parallell exekvering, finns många andra frågor som uppstår i projekt som använder flerkärniga chips. Några exempel:

- Körtidsstöd för OS-konfigurering, resursdelning och uppstart.
- Kommunikation mellan kärnorna (IPC, interprocesskommunication)
- Stöd i utvecklingsverktygen för konfigurering, prototyper, analys, diagnos och test.

Alla dessa problem måste lösas om du vill kunna sätta ditt ihop ett system på rimlig tid.

Val av operativsystem och konfiguration är viktigt. Ofta är ett OS redan bestämt, kanske genom ett arv från ett tidigare projekt. Eller på grund av särskilda krav. Om man behöver realtidsbeteende eller -prestanda är ett RTOS ett givet val. Om den enorma tillgången på mjukvara för generella operativsystem känns lockande, kan ett sådant OS vara lämpligt.

En svårare fråga är hur OS:et ska arbeta mot multikärnan. Det finns olika användningsfall:

- Ett eller flera operativsystem?
- SMP eller AMP? Eller båda? Det sistnämnda alternativet kan vara en fördel i asymmetriska system som använder icke-uniform minnesaccess (non uniform memory access) eller hårdvarumultitrådning.
- Kan en hypervisor vara en fördel?
- Vilket OS ska du använda för dedikerade kärnor?

En fördel med multikärnor och hypervisorer är att du kan välja en kombination av olika aspekter efter behov.

Snabba kommunikationsprimitiv mel-

Användningsfall 1:

SMP

Figur 4 på nästa sida visar en SMP-implementerad nätverksväxel. Parallelliteten i multikärnan utnyttjas för att langa så många paket som möjligt genom en gigabitväxel.

Operativsystemet kan antingen vara ett RTOS som VxWorks eller ett generellt OS som Linux. Om man utnyttjar affinitet (programtrådar låsta till kärnor) kan paketförmedlingsmotorn i dataplanet köras direkt i cacheminnet med understöd av avlastningsfunktioner som hanterar buffring, kryptering, med mera.

Användningsfall 2:

Övervakad AMP

Figur 5 (se nästa sida) visar samma samma nätverksväxel som figur 4, men implementerad med hjälp av övervakad AMP. I detta system är en av kärnorna avdelad att sköta kontrollplanet och kör där ett fullskaligt operativsystem. Om kontrollplanet skulle behöva ännu mer processorkraft skulle man kunna tilldela operativsystemet ytterligare ett eller ett par kärnor att användas i SMP-läge.

Övriga kärnor används för dedikerade uppgifter, som paketförmedling. Dessa kärnor har ett litet operativsystem, alternativt ett realtidskörsystem (”executive”), som ordnar så att paketförmedlaren kan jobba effektivt.

Under processorkärnorna körs en supervisor som hanterar delade enheter som till exempel avbrottshanteraren. I detta scenario är supervisorn bara aktiv när det behövs, vanligtvis under installationen. Därefter kan operativsystem och övervakare köra i full fart på sina respektive kärnor.

SMP-versionen av växeln har uppenbara fördelar, bland annat att det i allmänhet är lättare att hantera och utveckla för bara ett enda operativsystem.

Men också AMP-versionen i figur 4 har fördelar, som:

• *Felisolering*: Om en kärna går ner, kan övriga fortsätta att fungera.

• *GPL-isolering*: Om operativsystemet är Linux, är AMP ett bra sätt att isolera dess licensregler från ägd programvara i processorkärnorna i dataplanet.

• *Skalbarhet*: AMP går vanligen att skala upp till ett större antal kärnor.

Användningsfall 3:

Hypervisor

Ett delvis annorlunda användningsfall är när du behöver blanda olika typer av operativsystem.

Kanske vill du ha ett generellt operativsystem för dess rika ekosystem av programvara för grafik eller nätverk. Medan ditt andra operativsystem kan behöva realtidsprestanda. Eller så finns befintlig kod som är svår att portera till ett nytt operativsystem. En annan vanlig situation är att delar av det inbyggda systemet ska certifieras till högre nivåer än vad som låter sig göras under Linux.

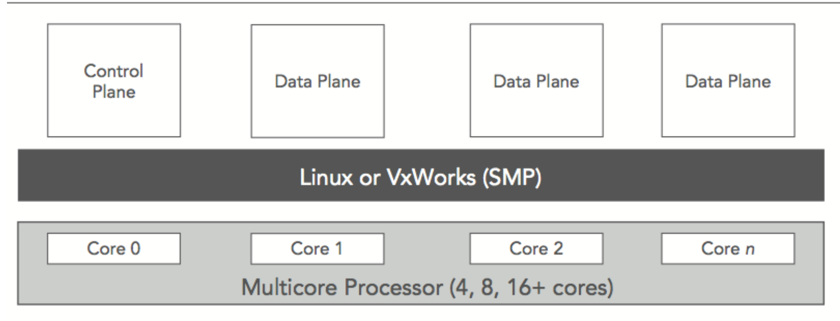
Oavsett orsaken till att man vill blanda OS, är en hypervisor (**figur 6**) ett effektiv sätt att hantera och separera partitioner, vare sig man kör på en trådschemalagd enkelkärna eller på en multikärna. En hypervisor ger ett system av ”jämlika” gästoperativsystem – de behöver inte veta av varandra i någon större utsträckning. Om exempevis ett av dem kraschar kan övriga köra vidare medan det omstartas.

Det finns mängder av sätt att konfigurera multikärnor och operativsystem. Här är tre vanliga användningsfall.

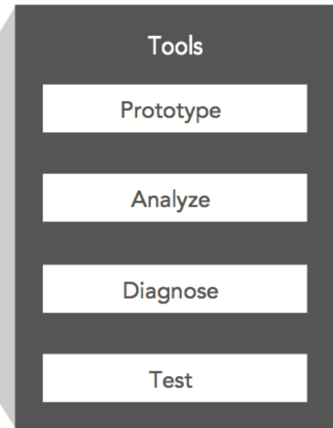
12 januari 2010 © Wind River Systems och Elektroniktidningen Sverige AB

Figur 5 SMP-lösning för prestanda

Characteristics
- Performance Focused
- Multicore of 4+ Cores
- System Partitioned
  - Control Plane
  - Data Plane (Using Affinity)

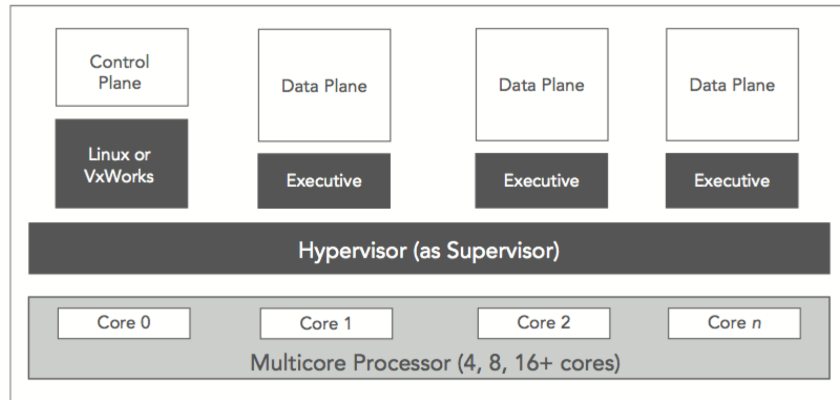


Network Equipment

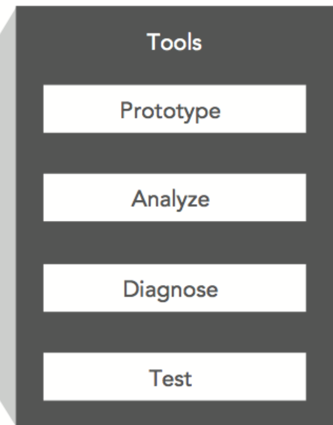


Figur 6 Övervakad AMP för prestanda och säkerhet

Characteristics
- Performance Focused
- Scalable Protection via Supervisor
- Multicore of 4+ Cores
- System Partitioned
  - Control Plane
  - Data Plane (Using Dedicated Executives)

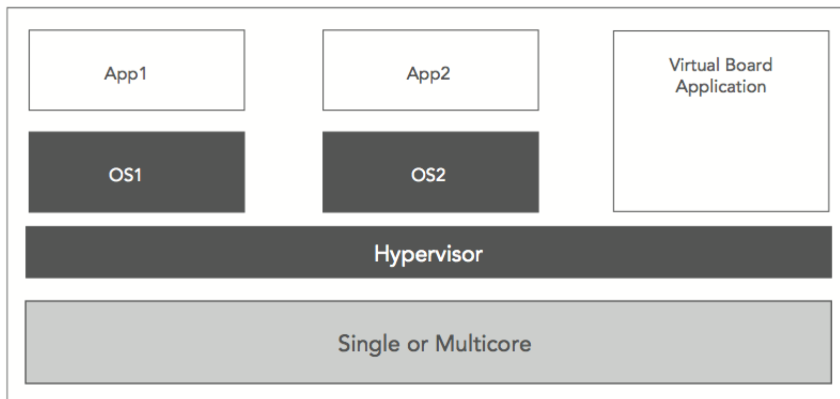


Network Equipment

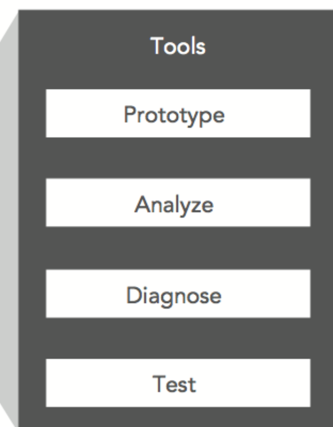


Figur 7 Hypervisor för konsolidering och säkerhet

Characteristics
- Multiple Different OSes
  - Real-Time (VxWorks), General (Linux), ...
- Full Protection via Hypervisor
- One or More Cores
- Certification Possible



Consolidation/ Migration



12 januari 2010 © Wind River Systems och Elektroniktidningen Sverige AB

lan kärnor och partitioner är inte bara viktiga för de data som tillämpningen utväxlar utan möjliggör också andra funktioner, som in- och utdata från externa källor, *printf()*, filsystem, nätverk, felsökning, och så vidare.

Om det finns fler kärnor än in- och utenheter, så måste de vara delbara. Ta en avlusare som exempel. Om du vill felsöka flera partitioner samtidigt kan du normalt inte ha en separat avlusningsenhet för varje kärna. Istället vill du ha en anslutning till ett av operativsystemen och sedan komma åt de andra partitionerna via en proxy.

Beroende på dina behov, kommer du att vilja kunna kommunicera på olika nivåer. Ibland är gemensamt minne och några få funktioner, som låsning, de enda primitiv du behöver. Andra gånger behöver du meddelandesystem som kan arbeta på en högre abstraktionsnivå. Håll ögonen öppna efter standardiserade API:er och protokoll för att få kod som är lätt att portera. Många IPC-mekanismer (interprocess communication) använder standardgränssnittet *socket*, vilket gör dina program mycket portabla från hårt kopplade system till större system distribuerade över TCP/IP.

Verktyg utgör ett än viktigare stöd för utvecklaren när man använder multikärnor. Komplexiteten i ett multikärnesystem gör det mycket svårt att fortsätta i gamla utvecklingsvanor. Cykeln redigera-kompilera-avlysa räcker inte längre. Nya utmaningar kring konfiguration, prototypning, diagnos, analys och test kräver en ny nivå av verktygsstöd.

Prototyper och simulering

Det finns många anledningar till att det blivit allt populärare att utveckla system på en simulering av den riktiga hårdvaran. Kanske hårdvaran ännu inte är tillgänglig eller är alltför dyr för kunna användas av alla utvecklare. Kanske du behöver felsöka ditt system på ett sätt som inte är möjligt med riktig hårdvara. Men ofta är skälet att det helt enkelt är smidigast för de allra flesta programvaruutvecklare att slippa bry sig om hårdvarufrågor och istället kunna fokusera på en faktor i taget. I och med att tidig kontinuerlig testning blir allt viktigare, fyller simulatorer en växande roll.

Det finns olika typer av simulatorer som fyller olika behov. Ibland kan en snabb funktionell simulator som härmar operativsystemets beteende vara tillräcklig. Dina tillämpningar kan köras snabbt på simulatorer som utnyttjar instruktionsuppsättningen på värddatorn.

I andra änden av skalan finns instruktionsuppsättningssimulatorer (ISS). De har två fördelar för multikärneavlusning: exakt exekvering och systemsimulering.

När en bra ISS simulerar ett system, kommer det att köra koden exakt likadant varje gång. Så fungerar det inte när man kör på riktig hårdvara, där små tidsskillnader leder till skilda beteenden i tillämpningen. I och med att timingproblem därmed uppstår på ett förutsägbart sätt blir det mycket lättare att felsöka eftersom problemet kommer att se likadant varje gång.

Systemsimulering innebär att man kan simulera hela systemet ihop och stoppa alla processorer och enheter samtidigt för att titta på vad som händer.

Kombinationen av dessa egenskaper är mycket användbar för felsökning i multikärnor.

Vissa simulatorer har ännu mer avancerade funktioner, som kodexekveringsspårning och bakåtstegning. Det senare är mycket användbart om det fungerar i varje kontext, inklusive under interaktion mellan enheter.

Diagnos och avlusning

En annan tuff uppgift i multikärnor är felsökning. Felsökning i samverkade programtrådar är en storleksordning svårare än felsökning i sekventiell kod. Detta eftersom man måste ta hänsyn till att andra programtrådar eller processorkärnor kan manipulera delade data vid varje given tidpunkt. Detta får varenda kodrad att plötsligt framstå i nytt ljus.

De två vanligaste multitrådbuggarna är dödlägen (deadlock) och kapplöpning (race condition). Ett dödläge innebär att två eller flera trådar har ställt varsin semafor men väntar på semaforer som någon annan tråd låst. En kapplöpning innebär att ett programs funktion beror av i vilken ordning parallella programtrådar råkar använda samma resurs. Det är lätt hänt att en kapplöpning uppstår när du avlusar ett dödläge, och vice versa.

Det svåraste med kapplöpningar är att eftersom de beror på dynamiskt bestämda tidsförlopp, så kommer de bara att manifesteras under bestämda omständigheter. Eftersom tidsförloppet vanligen skiljer sig från körning till körning, blir problemsymptomen vanligen olika för varje körning. När en skrivning är maskerad, som i **figur 9**, kan det ta miljoner cykler innan ett problem blir synligt och leder till korrupta data eller, i värsta fall, en systemkrasch.

Verktyg för körtidsanalys kan vara till stor hjälp i dessa situationer. De arbetar typiskt genom att logga information allteftersom händelser inträffar, så du kan analysera data i efterhand när ett problem dykt upp. Ofta kan du lägga till egna

Direkt på hårdvaran/realtidskörsystem

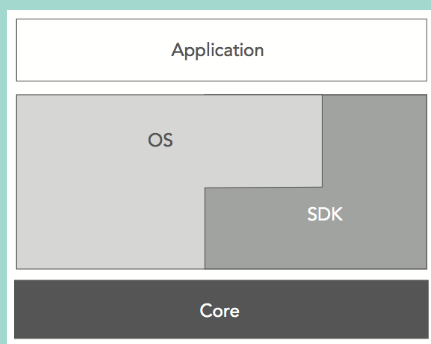
Dedikerade processorkärnor kan användas för specifika uppgifter.

Man kunde kanske tro att när programmet som ska exekveras i kärnan är enkelt, som en poll-slinga, så behöver man inget operativsystem. Och det stämmer. I alla fall inte ett fullskaligt operativsystem. Det kan vara fullt tillräckligt med ett litet **realtidskörsystem** (executive) med ett API för åtkomst till hårdvaran.

Ofta erhåller du ett sådant från kretstillverkaren. Men om du börjar addera egen kod till realtidskörsystemet, kanske du upptäcker att du behöver ännu mer stöd.

Till exempel – skulle det inte vara toppen om du kunde lägga in en *printf()*-sats i koden och omdirigera den till en seriell port som är mappad till din Linuxpartition? Och hur fungerar det med avlusning?

Om du på detta vis märker att du vill ha mer stöd än ett enkelt realtidskörsystem kan ge så kan en bättre lösning vara att använda ett operativsystem som kan skalas ner till att få ett litet dynamiskt fotavtryck (den aktiva koden) och därmed rymmas i cachen men ändå har anslutningsmöjligheter, avlusningsmöjligheter, analysverktyg, och så vidare. I den bästa av världar bifogar halvledarförsäljaren ett sådant litet hårt integrerat operativsystem med sin kod.



Figur 8 Arkitektur för ett realtidskörsystem (bare metal executive small execuive)

Figur 8 visar ett sådant litet operativsystem som är integrerat i kretsäljarens utvecklingsmiljö och dels tillhandahåller generiska funktioner och dels specifik access till hårdvaruresurser.

12 januari 2010 © Wind River Systems och Elektroniktidningen Sverige AB

användardefinerade händelser så att du kan se när vissa delar av tillämpningen exekeverades relativt andra händelser. Genom att stoppa in användardefinerade händelser i kod som manipulerar delade data, kan du upptäcka om det förekommer att uppgifter hämtas utan skydd från en mutex.

Det kan vara ett tungt jobb att infoga kod som söker efter tidsförloppsproblem, du måste kompilera och starta om programmet och senare komma ihåg att ta bort den. Det finns bra verktyg som låter dig infoga kod i din tillämpning dynamiskt i ett system under drift. Konceptet kan jämföras med brytpunkter.

Denna typ av felsökning, som gör att du kan se vad som händer utan att stoppa systemet, är mycket användbar i många sammanhang. Men ibland behöver du gå ner på djupet och se vad som verkligen händer nära hårdvaran. Övriga verktyg som beskrivs här har nackdelen att de oftast ändrar beteendet lite grand, antingen genom att stoppa in instrumenteringskod eller genom att simulera den verkliga hårdvaran.

De flesta processorer nu för tiden har stöd i hårdvara för avlusning. Så kallad on-chip debugging görs normalt tillgänglig via några stift (till exempel JTAG) på kortet. Via dem kan en on-chip debugger styra processorerna direkt utan programvaruagent. När det gäller multikärnor kan avlusaren styra samtliga processorer och enheter genom en enda kedja av kommandon kallad scankedja.

Moderna on-chip debuggers kan samtidigt styra (starta, stoppa, och så vidare) flera processorer på samma scankedja med mycket kort kryptid (tiden från det att den första kärnan reagerat tills alla kärnor reagerat). Detta är viktigt för att de andra kärnorna inte ska rusa iväg och ändra på tillstånd eller överfylla buffertar när en av dem stoppas. Se också till att du kan styra kärnorna från en enda integrerad utvecklingsmiljö (IDE) där du kan se och hantera processorkärnorna sida vid sida.

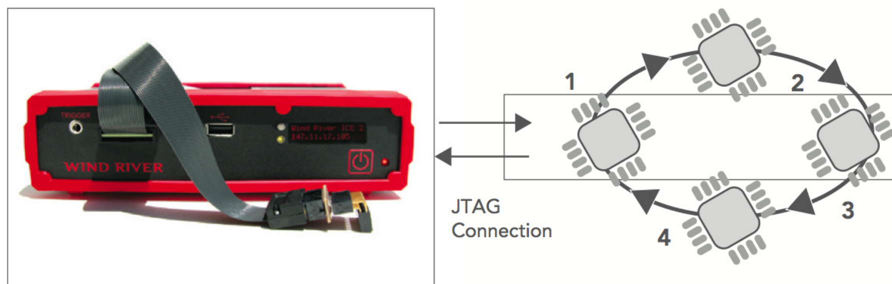
Möjligheten att stoppa alla kärnor samtidigt gör on-chip debuggern till en mycket kraftfull systemnivåavlusare eftersom helas systemet stoppas.

Förutom verktyg för analys av körtidsproblem och on-chipdebuggers, används även traditionella agentbaserade felsökare, framför allt för att hitta problem i sådan programkod som kan stoppas utan att störa resten av systemet.

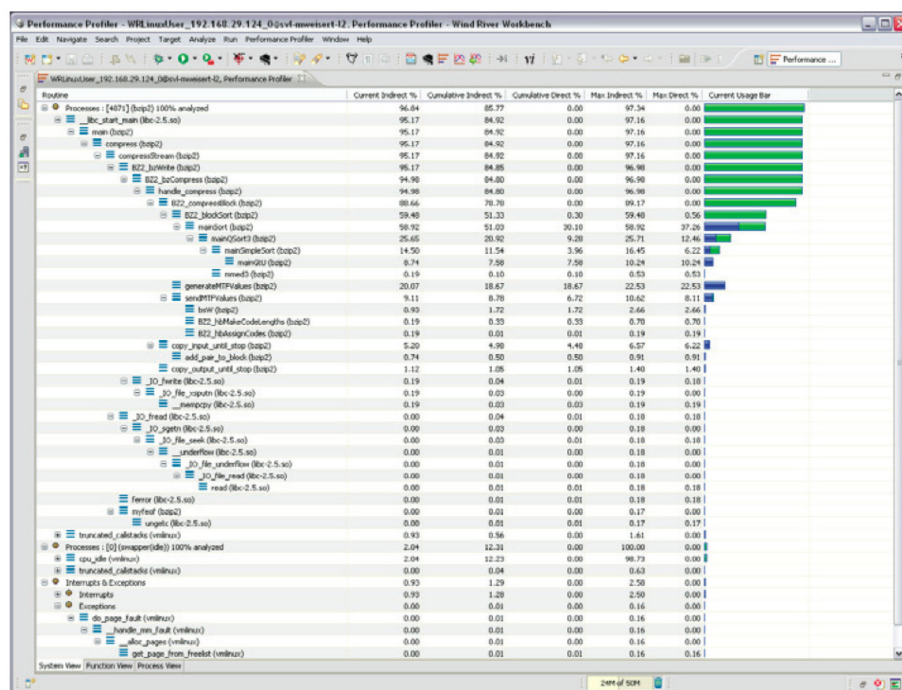
Analys av prestanda

Ett vanligt problem för utvecklare är att de inte får förväntade prestanda när de växlar till multicoresystem. Det är notoriskt svårt att förutse, analysera och förstå ett multikärnesystems beteende. Program vägrar levera önskad prestanda av en rad olika skäl, som synkroniseringsproblem (exempelvis väntan på semaforer), cacheproblem, minne- och IO-begränsade program, och så vidare. Utan hårda data är det mycket svårt att spekulera om vad som pågår. En profilerare är då ett mycket bra verktyg för identifiera tidstjuvar i koden. Se till att din profilerare hierarkiskt kan ange hur mycket tid som tillbringas i anropsträd, och inte bara platt hur mycket tid som tillbringas i enskilda funktioner.

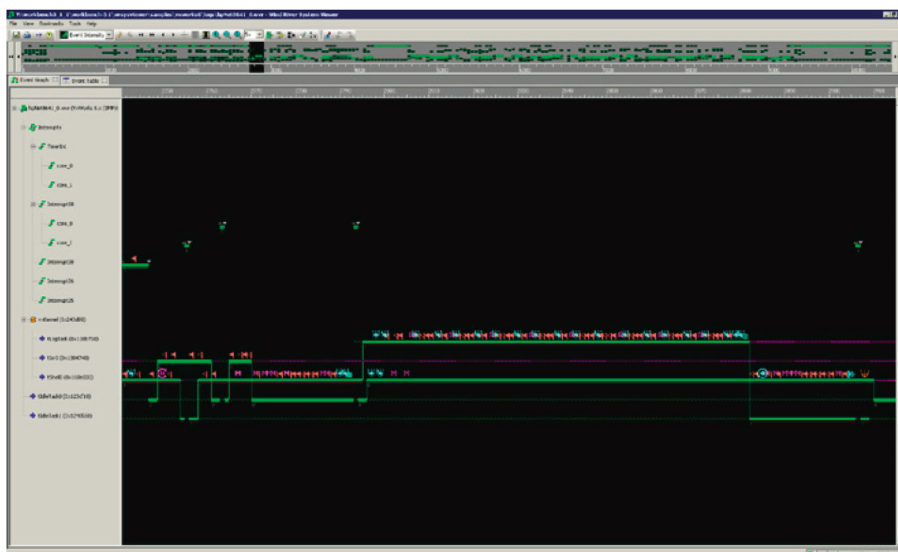
Rustad till tänderna med informationen från profileraren, kan du sedan fatta välgrundade beslut om var du ska lägga



Figur 9 Avlusare ansluten via JTAG.



Figur 10 Hierarkisk profilering i Wind River Workbench.



Figur 11 Systemvy i Wind River Workbench

12 januari 2010 © Wind River Systems och Elektroniktidningen Sverige AB

tid på optimering.

För att se vad som faktiskt pågår i ett multikärnesystem måste du också köra verktyg som analyserar körtidsbeteendet. Utdata kan se ut ungefär som i skärmbilden i **figur 11** som visar hur två parallellt exekverande processorer migrerar från en tråd till en annan. Markörer på tidslinjen motsvarar händelser som att ta och släppa semaforer. Du kan borra ner dig i händelserna för att se underliggande data.

Slutsatser

- Multikärnor har kommit för att stanna. Av rent fysiska skäl.
- Utveckling för multikärnor kan skilja sig dramatiskt från enkelkärnor.
- Det är effektivare att använda kommersiellt tillgängliga verktyg och plattformar än att försöka uppfinna hjulet på nytt.
- Ingen ensam strategi passar alla typer av användningsfall för multikärnor.
- Ingen silverkula transformerar godtycklig seriell kod till ett multikärneprogram.
- Du behöver ett urval körtidskonfigureringar och verktyg att välja mellan för att kunna köra din tillämpning effektivt.
- Dina verktyg och körmiljöer måste vara väl integrerade, inte bara med varandra utan också med din hårdvara. Endast då kan du dra full nytta av multikärnor.

12 januari 2010 © Wind River Systems och Elektroniktidningen Sverige AB