10.1 Multiprocessor and multicore scheduling

Monday, April 27, 2020 7:21 PM

Dette kapittelet ser nærmere på prosess- og tråd-tidsstyring. Dette blir mye mer komplisert i et multiprosessor-system.

Et multiprosessor-system kan klassifiseres ved følgende:

- **Løst koblede eller distribuerte multiprosessorer, eller cluster:** Inneholder en samling av relativt autonome systemer, der hver prosessor har sitt eget hovedminne og I/O-kanaler.
- **Funksjonelt spesialiserte prosessorer:** Et eksempel på dette er en I/O-prosessor. I dette tilfelle er det en master, generelt-bruks (general-purpose) prosessor; spesialiserte prosessorer er kontrollert av master-prosessoren og tilbyr tjenester til den. (Kommer tilbake til dette i Kap 11).
- **Tett koblede multiprosessorer:** Inneholder et sett med prosessorer som deler et felles hovedminne og er under integrert kontroll av operativsystemet.

Sistnevnte system kan sikkert gjenkjennes fra tidligere kapitler, og er den vi ser nærmere på.

Granularitet (detaljnivå)

Detaljnivået eller frekvensen av synkronisering kan grupperes inn i:

Granularitet (størrelse)	Beskrivelse	Synkroniserings-intervall (instruksjoner)	Eksempel
Fine	Parallellitet innebygd i en enkelt instruksjonsstrøm.	< 20	Representerer en mer kompleks bruk av parallellitet enn ved bruk av tråder.
Medium	Parallell prosessering eller multitasking innenfor en enkelt applikasjon (*threading*).	20-200	For tråding - programmeren må eksplisitt spesifisere potensiell parallellitet i applikasjonen.
Grov	Multiprosessering av samtidige prosesser i et multiprogrammeringsmiljø.	200-2000	Programmer som håndteres enten ved multiprogrammering eller ved multiprosessering
Veldig grov	Distribuert prosessering på tvers av nettverksnoder for å danne et enkelt datamiljø.	2000-1M	^
Uavhengig	Flere urelaterte prosesser tilsvarer ingen synkronisering	Ingen	Banktransaksjoner. Operasjonene går raskere, men prosessene er uavhengige.

Designet av tidsstyring på en multiprosessor involverer tre sammenhengende utfordringer:

1. Tildeling av prosesser til prosessorer
2. Bruken av multiprogrammering på individuelle prosessorer
3. Dispatching av en prosess

Tildeling av prosesser til prosessorer
En forenklet visualisering er å se på prosessorer som en samlet ressurs, og så skal prosesser tildeles prosessorer ved etterspørsel. Spørsmålet da er om denne tildelingen skal skje statisk eller dynamisk. Hvis en prosess permanent tildeles en prosessor fra aktivering til den er fullført (statisk tildeling), så må det være en dedikert kortsiktig-kø for hver prosessor. Fordelen med dette er at det er lite overhead som går til tidsstyringsfunksjoner, ettersom tildelingen skjer én gang. I tillegg muliggjør det gruppe-tidsstyring (mer om dette senere). En ulempe er dog at en prosessor kan stå inaktiv mens en annen prosessor har flere prosesser i sin kø. Dette kan overkommes med en felles kø som alle prosessorene henter fra.
Det er vanlig å bruke både statisk og dynamisk tildeling med cache.

Bruken av multiprogrammering på individuelle prosessorer
For tradisjonelle multiprosessorer som håndterer synkronisering på grovere eller uavhengige detaljnivå burde hver prosessor kunne bytte mellom et antall prosesser for å oppnå størst utnyttelse. For applikasjoner med synkronisering på medium detaljnivå som kjører på multiprosessorer med mange prosessorer er det ikke like klart. Dersom flere prosessorer er tilgjengelige er det ikke lenger et overordnet mål om å holde hver prosessor så opptatt som mulig, og ser derfor istedenfor på hva som gir god ytelse i gjennomsnitt for applikasjonene. En applikasjon med et antall tråder kan for eksempel gjøre det dårligere dersom ikke alle trådene er tilgjenlig for å kjøres samtidig.

Dispatching av en prosess

Den siste utfordringen ligger i å velge hvilken prosess som skal kjøres. I motsetning til uniprosessor-systemer, så er det unødvendig komplekst for multiprosessorer å bruke prioritet og tidsstyring basert på brukshistorikk. For multiprosessorer brukes heller enklere metoder som gir mindre overhead. Dette forklares nærmere i seksjonen under.

Prosess-tidsstyring (for multiprosessor-systemer)

I tradisjonelle multiprosessor-systemer er ikke prosesser dedikert til prosessorer. Det er istedenfor en enkel kø for alle prosessorer, eller flere køer av ulike prioritetsgrader hvis prioritet benytetes. Uansett henter prosessorer sine prosesser fra samme hovedkilde. Det er forskning som viser at spesifikke tidsstyringsdisipliner (som vi så på i Kap 9) ikke er like effektive når det er flere prosessorer i spill, og viktighetsgraden synker ytterligere med flere prosessorer. Derfor kan en enkel FCFS-ordning eller FCFS med prioriteringsordning være tilstrekkelig for et multiprosessor-system. Dette er samme algoritme som brukes for uniprosessorer, bortsett fra at valget faller på en av de enklere variantene. *Dette viser at behovet til prosesser ikke avviker så mye mellom uniprosessor-systemer og multiprosessor-systemer.*

Tråd-tidsstyring (for multiprosessor-systemer)

For tråder er execution annerledes enn for prosesser. En applikasjon kan implementeres som et sett med tråder som samarbeider og executer samtidig i samme adresserom. Fordelen med tråder kommer tydeligere frem i et multiprosessor-system med reell parallellitet, noe som fører til at vi velger andre algoritmer enn for et uniprosessor-system når vi ønsker å utnytte den reelle parallelliteten.

Det er spesielt to viktige aspekter vi ønsker å utnytte ved bruk av tråder. Den første er at vi kan ønske å sikre at en tråd alltid kjører på den samme prosessoren – for å utnytte bruken av cache. Det andre er at trådene kan kjøre på hver sin prosessor for å sikre samkjøring mellom dem og dermed god kommunikasjon mellom trådene. Med flere prosesorer tilgjengelig kan man fokusere mer på slike hensyn enn med uniprosessering – der fokuset istedenfor ligger på å holde prosessoren aktiv til enhver tid.
Det er fire metoder for tidsstyring av tråder og tildeling av prosessorer:

1. **Lastdeling (Load share):** Dette omhandler selv-tildeling av prosessorer. Dette er den enkleste og kanskje mest brukte metoden. Prosesser tildeles ikke til en bestemt prosessor. En global kø med klare tråder opprettholdes, hvorav hver prosessor henter tråder når den er tilgjengelig.

   Fordeler
   - Lasten fordeles likt utover prosessorene, slik at prosessorene ikke går på tomgang
   - Trenger ikke en sentralisert tidsstyrer fordi prosessoren selv velger den neste tråden når den er klar
   - En global kø brukes med en av tidsstyringsdisiplinene nevnt i Kap 9

   Ulemper
   - Den sentraliserte køen opptar en region i minnet som må aksesseres med gjensidig utelukkelse. Dette kan medføre en flaskehals-effekt hvis flere prosessorer ser etter arbeid samtidig
   - Avbrutte tråder har liten sannsynlighet for å fortsette kjøring på samme prosessor, hvilket gir dårlig utnyttelse av cache
   - Hvis det er høy grad av kommunikasjon mellom trådene i programmet vil prosessbyttene gi svært dårlig ytelse

2. **Gruppe-tidsstyring:** En tidsstyringsalgoritme der relaterte tråder eller prosesser kjører samtidig på forskjellige prosessorer. Vanligvis gjelder dette tråder som hører til samme prosess. Siden hver tråd får en prosessor sikrer dette samkjøring av trådene, samt at de alltid er klare til å kommunisere seg imellom på samme tidspunkt. Dette medfører også færre trådskifter ved koordingering/kommunikasjon mellom trådene.
   Gruppe-tidsstyring er nyttig for applikasjoner som krever inter-kommunikasjon mellom tråder, eksempelvis ved de som har medium/fine størrelse av granularitet.

3. **Dedikert tildeling av prosessorer:** Dedikert prosessor-tildeling er en ekstrem variant av gruppe-tidsstyring og dedikerer en gruppe prosessorer til en applikasjon for utførelse. Hver av trådene i applikasjonen får sin egen prosessor til applikasjonen er fullført (statisk tildeling). Dette sikrer samkjøring av trådene, og gir ingen trådskifter – noe som medfører full hastighet på applikasjonen. Det er dog dårlig utnyttelse av prosessorer, ettersom de kan bli stående i tomgang ved for eksempel I/O-forespørsel. Både gruppe-tidsstyring og dedikert prosessor-tildeling er gode til å unngå problemer som thrashing og prosessor-fragmentering (hvor noen prosessorer står på tomgang mens andre står i arbeid).

4. **Dynamisk tidsstyring:** Noen applikasjoner tillater språk- og system-verktøy for å tillate at flere tårder i en prosess kan endres dynamisk. Dette er den mest effektive, men likevel minst brukte algoritmen. Antall tråder i en prosess kan endres underveis i execution, hvilket lar OS-et tilpasse lasten for å forbedre utnyttelsen av ressursene. Det vil si at antall prosessorer til en prosess også kan variere tilsvarende. I dynamisk tidsstyring bør programmet og systemet samhandle om tilordning av tråder til prosessorer. Strategien er som følger:

   Når en jobb etterspør en eller flere prosessorer...

1. Hvis det er noen inaktive prosessorer; bruk dem til å tilfredsstille forespørselen
2. Hvis det ikke er noen inaktive prosessorer (og jobben med etterspørselen er en ny ankomst); tildel jobben en enkel prosessor ved å fjerne en fra en jobb som for øyeblikket har mer enn én prosessor
3. Hvis ingen deler av forespørselen kan imøtekommes; la jobben være utestående inntil en prosessor blir tilgjenglig, eller forespørselen forfaller

   Ved frigjøring av en eller flere prosessorer...

4. Scan den nåværende køen for utilfredse forespørsler for prosessorer. Tildel en enkel prosessor til hver jobb i listen av jobber som ikke har noen prosessorer. Scan så listen igjen og tildel resten av prosessorer på FCFS-vis.

Tidsstyring ved multikjerne vs. multiprosessor

Mange multikjerne-systemer og multiprosessor-systemer benytter samme algoritmer for tidsstyring – nemlig lastdeling. Lastdeling sikrer utnyttelse av prosessorer, men ettersom trådene ikke executer på samme prosessor gir det ikke full utnyttelse av caching. I multiprosessor-systemer er det viktig å sikre høy utnyttelse av prosessorkraft, mens i multikjerne-systemer er det viktig å holde felles minneaksess innenfor chip-en. Ved behov for felles lageraksess kan man bruke nabokjerner som deler cache.

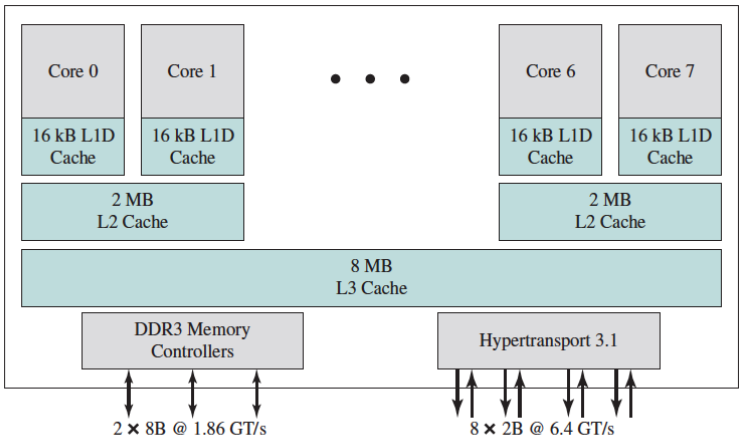

Figure 10.3 AMD Bulldozer Architecture

Legg merke til cache-nivået L2 i figuren som muliggjør fellesaksess av delt cache for to nabokjerner.