Kap. 9: Skalerbare, "Multithreaded" og dataflyt arkitekturer

- **Teknikker for "latency hiding"**
- **Delt virtuelt lager**
  - Eng. Shared Virtual Memory (SVM)
- **Scalable Coherent Interface (SCI)**
  - Kap. 9.1.4.
  - Supplert med:The Scalable Coherent Interface and Related Standards Projects, David B. Gustavson, IEEE Micro, feb 1992, side 10-22. (Blir pensum)
- **Litt mer om lager-konsistens**
  - Dekket grundig i høst
  - Kursorisk
- **"Multithreading"-prinsipper**
  - kontekst-bytte
- **"Fin-kornete multimaskiner"**
  - *et utvalg av*
    - » MIT J-machine, Caltech Mosaic C, Stanford DASH, KSR-1, Tera multiprosessor
- **Dataflyt og hybride arkitekturer**
  - Dataflyt-maskiner,
    - » prinsipper
    - » ETL / EM-4
    - » Manchester Dataflow Machine (egen artikkel)

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Teknikker for "latency hiding" ("skjuling av langsomme aksesser")

- **Tidsforsinkelse ved hovedlageraksess blir et stadig større probem**
  - økt bruk av fysisk distribuert delt lager i store og skalerbare maskiner
    - » forsinkelse ved aksess til fjerntliggende adresser
  - prosessorhastighet øker raskere enn lagersystemer og sammenkoplingsnettverk
- **Hovedteknikker for å redusere effekten av lang aksesstid til hovedlageret ("latency hiding")**
  - "prefetching"
    - » hente instruksjoner og data nærmere prosessoren før de blir brukt ("så nært vi har plass til")
  - duplisert lagring i konsistente hurtigbuffere
    - » generell teknikk, ikke nødvendigvis bare i hurtigbuffere
  - "friere" (Eng. relaxed) lager konsistens modeller
    - » tillater buffring og pipelining av lageraksesser
    - » "økt ytelse oppnås ved at programmerer tar over noe av ansvaret for å sikre riktig rekkefølge på lageraksesser"
  - kontekst-bytte
    - » ventetid ved lageraksess bør om mulig fylles med eksekvering av en annen prosess

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Stanford DASH multiprosessor

- **NUMA multiprosessor**
  - hierarkisk (2 nivåer)
    - » prosessor-klynger med felles lager
    - » sammenkoplingsnettverk
  - fysisk distribuert lager
  - konsistente hurtigbuffere
    - » strategi: ugyldiggjøring
    - » distribuert katalog over hurtigbuffer-kopier
  - 3-nivå hurtigbuffer
    - » 2-nivå hurtigbuffer i hver prosessor
    - » felles hurtigbuffer for hver prosessor-klynge
- **Fig. 9.1**
- **Tilbyr virtuelt delt lager**
  - alle programmer på maskinen ser ett stort virtuelt adresserom

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Delt virtuelt lager (Eng. shared virtual memory (SVM))

- **Ett stort logisk (virtuelt) adresserom for alt som skjer på maskinen**
  - Fysisk plassering av ulike deler kan variere dynamisk under kjøring
  - Tillater at program og data er større enn det det er plass til i *en* fysisk node
  - Gir økt programvare-portabilitet
  - Fig 9.2a
- **Teknisk løsning**
  - delt inn i (relativt store) sider
  - hver node må ha en lagerhåndterer som kan finne ut hvor den aktuelle blokken er lagret
    - » analogt med hurtigbuffer-problematikken der en node kan ha noen sider i lokalt lager (hurtigbuffer) og noen sider lagret andre steder (hovedlager)
    - » "read-only"-sider kan lagres i flere kopier
    - » ved skriving kan en sikre konsistent tilstand ved ugyldiggjøring
    - » "page-fault"
  - store variasjoner i realisering (Tabell 9.1)
    - » arbeidsdeling HW/SW
    - » "skriv-ugyldiggjør" vs. "skriv-oppdater"
- **et nytt og "hett" tema**
  - mye forskning pågår

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Teknikker for forhåndshenting (Eng. prefetching)

- **Prefetching ("forhåndshenting")**
  - benytter kunnskap om forventede aksess-bom for å flytte disse data nærmere prosessoren før de behøves
- **"Binding prefetching"**
  - verdien av en kommende lagerreferanse bindes (avgjøres) ved forhåndshenting
  - kan begrense hvor tidlig en kan utføre forhåndshenting
    - » (for å unngå å bruke en gal verdi)
    - » redusert ytelse
- **"Nonbinding prefetching"**
  - forhåndshentede data holdes oppdatert helt til de benyttes
- **Maskinvarebasert prefetching**
  - Eksempler:
    - » store blokker i hurtigbufferet
    - » instruksjons-prefetching
- **Programvarestyrt prefetching**
  - selektiv henting ved spesielle instruksjoner
    - » reduserer krav til båndbredde
    - » gjør det mulig å "hente i god tid"

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Fordeler ved forhåndshenting

Side 480

- **Hovedfordel: data er allerede "inne"**
  - har hentet inn på forhånd, sparer venting på lager-aksess
  - ofte ikke mulig å kjenne adressen før rett før den benyttes
- **Større datastrukturer**
  - antas lagret i flere etterhverandrefølgende blokker og at vi "henter inn hele på en gang"
  - antar at vi ikke kan hente inn på forhånd
  - vanlig forsinkelse ved første blokk
    - » oppstart-tid
  - forsinkelse ved henting av resterende blokker "skjules" ved at lager-aksess behandles i pipeline
- **"Prefetch with ownership"**
  - forhåndshenting kan med fordel kombineres med noen hurtigbuffer koherens protokoller
    - » hente en blokk med eierskap (for skriving) kan være lønnsomt framfor først å hente en delt (read-only) blokk og deretter hente eierskap for skriving
    - » (dvs. "slå to fluer i ett smekk")

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

"Latency-hiding" benchmark resultater

- **Utført på en simulert DASH multiprosessor**
- **Hastighetsøkning ved ulike strategier for forhåndshenting**
  - Fig. 9.3
- **Hastighetsøkning ved lagring av delte data i hurtigbuffer**
  - Fig. 9.4

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Scalable Coherent Interface (SCI)

- **Pensum;**
  - Kap. 9.1.4. (Trykkfeil i kapittel-overskrift)
  - *The Scalable Coherent Interface and Related Standards Projects*, David B. Gustavson, IEEE Micro, feb 1992, side 10-22. (Blir pensum)
- **Standardiserings-samarbeid mellom div. buss-konstruktører og system-arkitekter**
  - 1987, Superbus Study Group
  - 1988-1992, IEEE P1596 SCI
- **Kraftig Norsk deltakelse!**
  - UiO (www.uio.no)
  - Dolphin ICS
  - Sintef DELAB
  - FFI - NDRE
  - TF

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

SCI: Motivasjon

- **Buss-teknologi, fundamentale begrensninger**
  - bare en dataoverføring av gangen
  - begrenset signal hastighet
- **Utvidelse av Buss-teknologi**
  - økt bussbredde
    ulempe: overhead ved arbitrering og adressering er fast
  - lengre blokker
    fordel: reduserer effekt av denne overhead
    ulempe: størrelse på hurtigbuffer-rammer begrenser
  - kortere buss
    fordel: raskere signalering
    ulempe: mindre nyttig buss
  - lavere signalspenning
    ulempe: øker støyproblemer
  - flere busser
    fordel: mer enn en overføring av gangen
    ulempe: øker kompleksiteten

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

SCI: Målsetting

- **Scalable**
  - samme mekanisme i PC´er og massivt parallelle superdatamaskiner
    - » reduserte kostnader gjennom volum-produksjon
    - » reduserte kostnader gjennom redusert opplæring
- **Coherent**
  - støtte effektiv bruk av hurtigbuffer i multiprosessorer med distribuert delt lager
- **Interface**
  - en åpen kommunikasjons-standard, tillater ulike produkter fra ulike leverandører å samarbeide greitt i et total-system

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

SCI: Hovedprinsipper

- **Asynkron kommunikasjon**
  - Grensesnittet venter ikke på at et signal har kommet fram før en sender neste signal. Dette gjør hastigheten (mer) uavhengig av systemets størrelse
  - mer krevende protokoller
- **Meldingsoverføring**
  - Hver "enkelt-kommunikasjon" (transaksjon) skjer ved å sende en pakke (adresse, kommando, evt. data)
  - 'pipelining' av bit
- **Lys-hastigheten begrenser kommunikasjonshastigheten på en fysisk linje, men ...**
- **SCI benytter flere signal-veier (linjer)**
  - flere uavhengige overføringer kan pågå samtidig
  - kan ha separat linje for hver prosessor, lager-enhet og IO-enhet
- **Vi har da et parallelt kommunikasjonsmedium, hvor ytelsen kan skaleres opp v.hj.a. parallellitet**

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

SCI-Hovedprinsipper

- **Linjer**
  - differensielle signaler, enveis, punkt til punkt
- **De samme protokoller er brukt innenfor en multiprosessor som i ett lokalnett**
- **Standard Produkter**
  - **Standard SCI modul**
    - » definerer signaler, tilkobling ogstrøm-tilførsel
    - » 16 bit brede linjer
    - **» 1 GB/s**
  - **Standard kabel og konnektor definerer bruk av signaler i en LAN**
    - » koaksialkabel
    - » middels avstander
    - » billigere
    - **» 1 Gb/s**
  - **Standard fiber-optisk serielt grensesnitt**
    - » LAN (kilometer)
    - » optisk fiber
    - **» 1Gb/s**

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

SCI: "Fysiske valg"

- **En-veis linjer**
  - mer skalerbare
  - gir mindre støy
- **På linjene: differensielle signaler, lav spenning**
  - gir mindre støy
  - tåler mere støy
- **Raske og smale linjer**
  - antar at tilkoplings-pinner alltid vil være en begrensning
- **Initielt; 16 bits brede, ECL-kompatible signaler**

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

SCI-node (grensesnitt)

- **Minst 2 linjer, inn og ut**
- **Linjene benyttes kontinuerlig**
  - sender evt. "tomgangs-pakker" (Eng. idle-symbols)
  - disse bidrar til synkroniseringen
- **Synkron Node (unntatt resynk)**
- **'Go' bit**
- **Tilbyr pakke-baserte protokoller for den type tjenester vi forventer fra en buss.**
  - hovedforskjell: Svar adskilt fra forespørsel
  - likner en del på såkalt "split-response-bus"
  - uavhengige buffer
  - **Fig.B**
  - **Fig.1 (Fig.C-Pensum)**

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

SCI: Alternative systemløsninger (1/2)

- **SCI tillater pipelining, gir mange muligheter for høy ytelse**
- **Høy-kostnad-systemer:**
  - Leverandør-spesifikk høy-ytelse-svitsj
    - » Leverer pakke til gitt adresse
    - » Mange muligheter
    - » Ytelse / kost
    - » Implementasjon ikke spesifert i SCI
    - **» Fig. 9.5c**
- **Lav-kost-systemer, ring-kopling**
  - Krever at en node må kunne videre-sende en pakke
    - » buffer-plass
    - » adresse-logikk
  - begrenset antall noder
  - **Fig. 9.5b**
- **Kombinasjon av svitsj og ring**
  - kobling av ringer med kompleks eller enkel svitsj
  - e.g. Torus

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

SCI: Alternative systemløsninger (2/2)

- **Node**
  - enkel
    - » prosessor, minne og I/O enheter
  - kompleks
    - » klynge av noder
      - 2-nivå
      - fler-nivå
  - **Fig**
- **Bro**
  - sammenkobling av SCI med andre standard busser
    - » SCI / VME (P1596.1)
    - » SCI / Futurebus
    - **» Fig. 9.5a**

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

SCI: Løsning av hurtigbuffer-koherens problemet

- **Massivt parallelle systemer med hurtigbuffer-koherens krever (repetisjon)**
  - katalog over hurtigbuffer kopier
  - katalogen bør lagres og oppdateres/benyttes distribuert
- **SCI, distribuert, dobbelt-lenket liste**
  - protokoller basert på punkt-punkt transaksjoner,
    - » begynner med etterspørsel (f.eks. prosessor)
    - » fullført med svarer (f.eks. minne eller en annen prosessor
  - memory-kontroller
    - » hver hurtigbuffer blokk(64 bytes):
      - liste hode ID
      - status bit
  - cache-kontrollere
    - » pekere til liste naboer
    - » status bit
  - **Fig. 9.6**

SCI: hurtigbuffer-koherens

- **Delt-lenket-liste transaksjoner**
  - Forespørsler køes i FIFO rekkefølge
  - Hive ut blokk (Eng. cache-line)
    - » ber naboer peke til hverandre
  - Lese **Fig. 9.7**
  - Skrive
    - » 'purge list' (ikke pensum)
  - Optimaliseringer(del av standard)
    - » kombinering av forespørsler (request combining)
      - eliminerer hot spots
    - » DMA lokal aksess
- **SCI definerer et sett låse-primitiver**
  - Mekanismer som garanterer eksklusiv bruk av Hurtigbuffer-liste
    - » masked swap
    - » compare and swap
    - » fetch and add
  - Kan brukes for distribuert og samtidig oppdatering av en delt lenket liste
  - Opererer gjennom svitsj eller bro

SCI: hurtigbuffer-koherens - Fordeler

- **Hot -Spots: vedlikeholdstrafikk spredd i systemet reduserer 'hot-spot' problemer**
- **Parallelitet: hver enkelt liste kan oppdateres samtidig av flere prosessorer**
- **Varighet av moduler**
  - ytelse uavhengig av langsomste kontroller
    - » bare de moduler som har kopi av data deltar
- **Skalerbarhet**
  - kan derfor lett ta i bruk nye og raskere/bedre cache-kontrollere uten å måtte kaste eldre deler av systemet
  - alltid nok plass til katalog
- **Vranglås og "livelock"**
  - multiprosessorer "framdriftsproblemer"
  - (Eng. forward progress issues)
  - vranglås
    - » SCI protokollen eliminerer avhengigheter
    - » programvare avhengigheter kan eksistere
  - livelock ("utsulting")
    - » ofte lav sannsylighet, men alvorlige og frekvente når de først har oppstått
    - » SCI unngår dette med bruk av 'fair policy'

SCI: Diverse

- **Standarden er (hovedsaklig) spesifisert i C-kode**
  - det meste av dette er eksekverbart:
    - » reduserer antall ulike tolkninger (tvetydighet)
    - » enklere testing
    - » forenkler nøyaktig simulering
- **SCI vil "følge med teknologien"**
  - f.eks. nye standarder for fiber-optikk

SCI: relaterte standard prosjekter

- **P1596 - 1992 SCI Standard**
- **P1596.1: SCI-Svitsjer**
  - SCI / VME Bro
  - SCI / PowerPC
  - SCI / Motorola 68040
  - Tillegg:
    - » Ruting av kringkastings-transaksjoner
    - » 'Acknowledge' av kringkastings-transaksjoner
- **P1596.2: Kiloprosessor utvidelse**
  - Cache-optimalisering for stort antall SCI prosessorer (mer enn "noen hundre")
    - » ***tre-strukturert cache-katalog***
    - » kombinering av forespørsler
    - » hurtig distribuering av data
    - » fra *O(n)* til *O(log n)*
  - Tillegg:
    - » forslag på nye instruksjoner i kommende prosessorer for hurtig SCI implementasjon
    - » hubs: samling av SCI broer
    - » SCI I/Obro
    - » SCI cluster adapter

SCI: relaterte standard prosjekter

- **P1596.3: Lav spenning differensielle signaler**
  - aktuell - 17 bits differensielle ECL signal
- **P1596.4: Ramlink**
- **P1596.5: Data overførings formater**
  - utveksling av data mellom ulike byte-adresserbare prosessorer på SCI
  - distribuert maskinomgivelse av heterogene arbeidsstasjoner
  - godkjent standard (de andre er i mindre grad det!)
- **P1596.6: SCI for real time applikasjoner**
  - implementert på toppen av eksisterende protokoll
    - » paritets-beskyttete linjer
    - » retry-mulighet innenfor lokal ring ved feil
    - » veksle "clock-master"
    - » 'end-end' feil retry

SCI RAMlink

- **Et alvorlig problem ved dagens lagersystemer er at det er nødvendig å benytte et større antall lager-kretser i parallelle banker for å oppnå den båndbredde som dagens mikro-prosessorer krever**
  - Siden kapasitet pr. krets øker vil den minste lager-konfigurasjon med egnet båndbredde nå et punkt hvor den har uforholdsmessig stor kapasitet
- **P1596.4 (RAMlink), høy-båndbredde lagerkrets grensesnitt**
  - forsøker å utnytte SCI teknologi til å gi større båndbredde med færre kretser
    - » flere RAM kretser i en ring koplet til en 8-bits SCI kontroller via 8 bits brede linjer
    - » 1 kontroller per ring
      - tidsplanlegger ring aktiviteter
      - grensesnitt til SCI eller en annen
    - » multi-ring implementasjon mulig
    - » 500MB/s
  - delt-svar (Eng. split response) pakke protokoll
    - » hode beskriver siste frist for svar

SCI: Andre relaterte standard prosjekter

- **Arbeidsgruppe for høy-hastighets seriell buss med lav kostnad**
  - 10-20 Mbytes/s
  - konsument markedet
- **SCI Logical Protocol Simulation**

Mer om lager-konsistens

- **Sekvensiell og svak konsistens ble forelest grundig i.f.m. kap. 5**
- **Kap. 9.1.5. er kursorisk**
  - Hovedinnhold og erfaringer foreleses og er pensum
  - Detaljert virkemåte/forståelse av PC og RC er ikke pensum
    - » *datamaskinarkitektur-2*
- **Prosessor konsistens (PC)**
  - Løser opp krav til rekkefølge på skrive-operasjoner
    - » gir mulighet for buffring og pipelining av disse
- **Release konsistens (RC)**
  - En av de frieste lagermodeller som er foreslått
- **En friere lagermodell gir økte muligheter for ytelses-forbedring ved å redusere effekten av langsomme lager-aksesser ("latency hiding")**
  - Fig. 9.9
  - Gir vanskeligere programmering
  - Gir økt maskinvare-kompleksitet
    - » Ekstra maskinvare kan i noen tilfeller være den samme som behøves for "prefetching" og kontekst-bytte, dvs. **kombinasjon av teknikker kan være ønskelig, se Fig. 9.10**

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Prinsipper for "Multithreading"

- ***"Multithreading"***
  - Hurtig bytte mellom flere (lettvekts) prosesser (el. tråder) for å øke prosessor-utnyttelse ved langsomme lageraksesser
    - » kontekst-bytte
    - » fyller ventetiden ved tidkrevende lager-aksesser med beregninger for andre kontekster
  - "**Thread**" = tråd = logisk kontroll-flyt
    - » en prosess sin vei gjennom programmet
  - En *tråd* representeres av sin *kontekst*
    - » prosessor-tilstand
  - Fig. 9.13a viser prinsippet
  - Langsommere lageraksesser krever flere tråder for å oppnå god prosessor-utnyttelse
    - » MEN, økt antall tråder og kontekst-bytte gir normalt dårligere treffrate i hurtigbufferet
  - En lager-aksess sin "retur-pakke" må være merket med identifikasjon av riktig mottaker-tråd
    - » bør realiseres i HW for hurtighet
    - » kan alltid realiseres i SW
  - Kontekst-bytte kan i noen tilfeller (fig 9.14) også gi bedre utnyttelse av funksjonelle enheter som ellers ville forbli uutnyttet p.g.a. data- eller kontroll-avhengigheter

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Modell av "fler-kontekst" prosessor

- **Effektiv "multithreading" krever hurtig kontekst-bytte**
- **En kontekst består av (Fig. 9.14a)**
  - programteller
  - register-sett med verdier
  - status-ord
- **Prosessor-arkitektur for hurtig bytte mellom et fast antall tråder**
  - et antall "thread-slots"
    - » = logisk prosessor
    - » består av programteller, instruksjons-dekoding, instruksjons-kø
    - » disse deler funksjonelle enheter og instruksjons-hente-enhet
  - Fig. 9.14b

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Kontekst-bytte og prosessor-effektivitet

- **Strategier for kontekst-bytte**
  - (1) **Bytt ved bom**
    - » prosessor bytter kontekst bare når det er sikkert at kjørende kontekst må vente et vesentlig antall sykler
  - (2) **Bytt ved aksess**
    - » tillater bytte ved hver load-operasjon uavhengig av om den gir hurtigbuffer-bom eller ikke
      - dvs. kan bytte prosess oftere enn ved (1)
  - (3) **Bytt etter hver instruksjon**
    - » tillater bytte ved hver eneste instruksjon
    - » kan gi redusert treffrate i hurtigbuffer
  - (4) **Bytt etter hver blokk av instruksjoner**
    - » vil gi bedre treffrate i hurtigbuffer p.g.a. bedret lokalitet
- **Prosessor-effektivitet (utnyttelse)**
  - Maksimer den andel av tiden som prosessoren aktivt utfører beregninger
    - » *Efficiency = T(busy) / [T(busy) + T(switching) + T(idle)]*
    - » "Multithreading": reduser *T(idle)* mest mulig uten å øke *T(switching)* for mye
- **"Multithreading" øker prosessor-effektivitet men også nettverks-trafikk**
  - en avveining er nødvendig

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Prosessor utnyttelse

R tid mellom to eksterne referanser

L varighet av ekstern referanse

E_1 effektivitet for "single-threaded" maskin

$$E_1 = \frac{R}{R+L}$$

C varighet av kontekst bytte

N antall kontekster i "multithreaded" maskin

E_{sat} eff. "multithreaded" med nok kontekster

E_{lin} eff. "multithreaded" under metningspunkt

N_d metningspunkt

$$E_{sat} = \frac{R}{R+C} \quad \text{(Se fig. 9.15 a)}$$

$$E_{lin} = \frac{NR}{R+C+L} \quad \text{(Se fig. 9.15 b)}$$

$$N_d = \frac{L}{R+C} + 1 \quad \Leftarrow \quad (N-1)(R+C) \geq L$$

Fig. 9.15c skal "rettes opp", og oppsummerer da dette.

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Flerdimensjonale arkitekturer

- **Topologi for sammenkopling av prosessorer i MPP systemer har gjennomgått en utvikling fra 1D ringer, til 2D rutenett og videre til 3D rutenett (el. kuber) av noder**
  - dette reduserer "lengste vei" når antall noder øker
  - økt antall dimensjoner gjør (fysisk) konstruksjon av maskinen vanskeligere
  - Fig. 9.17

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Wisconsin Multicube

- **2D rutenett av busser**
- **Hver prosessor er koplet via hurtigbuffer til en kolonne-buss og en rad-buss**
- **Hver lagermodul er koplet til en kolonne-buss**
  - Dette er "hjemmelager" for prosessorer koplet til denne kolonnen
- **Rad-buss brukes for kommunikasjon mellom kolonner og cache-koherens kontroll**
- **Fig. 9.18a**
- **2-nivå hurtigbuffer**
  - 1) prosessor-cache, WT-strategi
    - » redusere aksess-tid
  - 2) snooping-cache
    - » stor cache for å redusere buss-trafikk
- **Arkitekturen kan generaliseres til en arkitektur der hver prosessor er koplet til *k* busser med *n* prosessorer koplet til hver buss, i en *n**k* multikube**
- **OMP-arkitekturen og ned side 504 utgår**

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

MIT J-Machine (Jellybean - machine)

- **Kap. 9.3.2**
  - basert på artikkel av Dally et. al. i IEEE Micro april 1992, den artikkelen er *ikke* pensum
  - Eks. 9.4 utgår
- **Forskningsprosjekt**
  - plattform for eksperimentering
    - » ulike modeller (paradigmer) for (fin-kornet) parallell programmering
      - f.eks. meldingsutveksling, dataflyt, m.fl.
    - » enkle (HW) mekanismer for kommunikasjon, synkronisering og adresse-oversetting
- **Arkitektur**
  - *k*-nær *n*-kube av MDP-noder (Fig. 2.20)
  - Første prototype: 8 x 8 x 16
    - » dvs. en redusert 16-nær 3-kube
    - » dvs. 3D rutenett
  - Maks. konfigurasjon: *64 k* noder (32 x 32 x 64)
    - » begrenset av adressering
  - Kubens sideflater kan brukes for tilkopling av IO-enheter
- **http://www.ai.mit.edu/projects/cva/cva_j_machine.html**

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Message-Driven Processor (MDP)

- **36 bits prosessor**
  - Spesialkonstruert VLSI-brikke, 1.1 mill. trans., fig. 9.19
  - funksjonalitet som vanlig prosessor men utvidet med funk. for komm., synk. og adresse-oversetting
  - 3 separate registersett for hurtig kontekst-bytte mellom
    - » bakgrunn, dvs. ingen melding har startet en beregning (Eng. task)
    - » håndtering av melding med prioritet-0
    - » håndtering av melding med prioritet-1
- **Meldingsdrevet**
  - [Hwa93]: gjør beregninger som respons på meldinger
    - » Ingen "receive"-instruksjon er nødvendig
    - » MDP starter en liten rutine ("task") for å håndtere en mottatt melding
    - » Meldingene driver beregninger
  - Objekter distribueres utover MDP-nodene
    - » operasjon på lokal node el. "remote"-node (f.eks. ved RPC (remote procedure call)
      - ytelsesoptimal distribuering av objekter?
  - I Message-Drive-C (MDC) [Maskit94]:
    - » et program er en samling av parallelle prosesser
    - » meldingsdrevet som forklart ovenfor
      - hver melding bringer med seg adresse for kode som skal utføres og argumenter

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

J-Machine: Ruting og kommunikasjon

- **MDP meldings-utveksling**
  - som en sekvens av SEND-instruksjoner
  - 2-veis kanaler
  - E-kube ruting
    - » En dimensjon av gangen, deterministisk, x->y->z
    - » Fig. 9.20
  - 1 linje i hver retning sammen med E-kube ruting gjør at vi ikke kan få *"ressurs-sykler"*
    - » får ikke vranglås
- **MDP router**
  - En ruter for hver dimensjon
    - » Pakke med relativ (*x,y,z*) adresse:
      - send videre langs *x*-dim. i riktig retning til relativ adr. = 0, gå da til neste dimensjon.
  - 2 prioritetsnivåer i hver ruter
    - » 2 virtuelle nettverk som deler samme fysiske kanal
    - » Prioritet-1 kan avbryte prioritet-0 om nødvendig
    - » Fig. 9.21
  - "Wormhole routing"
  - Blokkerende flyt-kontroll
    - » Fig. 9.22

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

J-Machine: Programvare

- **Effektive primitive mekanismer**
  - Realisert v.hj.a. HW
  - Utnytter parallellitet
  - Mekanismer for:
    - » **Kommunikasjon**
    - » **Synkronisering**
    - » **Dynamisk navngiving**
      - Binde navn til verdi/adresse
- **Disse mekanismene gir gode muligheter for å implementere en rekke programmeringsmodeller**
- **Eks. kommunikasjon**
  - en melding kan sendes til en annen node på under 2 mikro.sek.
  - passerer mellom-noder uten å forstyrre beregninger i noden
- **Concurrent Smalltalk (CST)**
- **Id**
  - funksjonelt programmeringsspråk opprinnelig konstruert for dataflytmaskiner
- **Message Driven C (MDC)**

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Caltech Mosaic C

- **Fin-kornet multimaskin**
  - Forskningsprototype
  - Etterfølger av "Cosmic Cube", tilsvarende nodeteknologi, men
    - » 60 x raskere
    - » 20 x lavere strøm-forbruk
    - » 100 x mindre
    - » 25 x billigere
  - Fokus på programvaren
  - Ønsket 3D rutenett men valgte 2D rutenett da det ga lettere fysisk konstruksjon
- **Node**
  - 11 MIPS prosessor
  - 64 Mbytes lager
- **Arkitektur**
  - Fig. 9.24
- **Hurtig meldingshåndtering**
  - Prosessor har 2 programtellere og 2 register-sett
    - » hurtig kontekstbytte mellom bruker-programmer og meldings-håndtering
  - Typisk 10 instruksjoner for å håndtere en melding
    - » i programvare p.g.a. ønsker om fleksibilitet

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Caltech Mosaic C

- **Programmeringsspråk**
  C+- (en utvidelse av C++)
- **"Teknologi-skalering av noder"**
  - Skalering med konstant node størrelse
    - » vil gi en-brikke-noder med økende lager-størrelse, prosesserings-kapasitet og kommunikasjons-båndbredde
  - Skalering med konstant node kompleksitet
    - » vil om ca. 10 år tillate at et Mosaic 8 x 8 prosessors kort realiseres på en enkelt brikke, med ca. 20 x høyere ytelse pr. prosessor

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Stanford DASH Multiprosessor

- **Pensum: kap. 9.4.1.**
- **John Hennessy m.fl.**
  - Stanford, siden 1988
- **DASH = Directory Architecture for Shared Memory**
  - Et addresserom, koherente hurtigbuffer og fysisk distribuert lager
    - » Logisk sett felles lager -> enklere bruk
    - » Meldingsutveksling -> realistisk/skalerbar maskin
- **Arkitektur (prototype)**
  - Fig. 9.1, side 477 (repetisjon)
  - 16 klynger av 4 prosessorer
  - Sammenkoplingsnettverk (Fig 9.25b)
    - » 2 parallelle 2D-rutenett
      - 16 bits kanaler
- **Realisering av klynge med 4 prosessorer**
  - Basert på arbeidsstasjon fra Silicon Graphics
    - » Begrenset adresserom i denne gjør at 16 klynger er maksimal konfigurasjon
    - » Bl.a. utvidet med nye kort for hurtigbuffer-katalog for klyngen og tilstandsmaskin for hurtibuffer koherens protokoll
  - Fig. 9.25a

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

DASH: Lager-hierarki og hurtigbuffer koherens

- **Lager-hierarki**
  1) prosessor hurtigbuffer
     - » T = 1 (klokkesyklus)
     - » buss-snusing (innen klynge)
  2) et annet prosessor hurtigbuffer innen samme klynge
     - » T = 30
  3) lagermodul i egen eller annen klynge
     - » Tilsvarer nivå 2 for egen klynge (dvs. T = 30)
     - » T = 100 (en annen klynge)
  4) prosessor-hurtigbuffer i en annen klynge
     - » T = 135

  ((Fig. 9.25c))

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

DASH: Katalogbasert protokoll for hurtigbufferkoherens

- **Hurtigbuffer-katalog for en klynge avlaster prosessorenes hurtigbuffer-kontrollere m.h.p. opprettholdelse av koherent tilstand**
- **Bruk av katalog gir skalerbarhet**
  - kan sende punkt-til-punkt oppdatering- eller ugyldiggjøringsmeldinger framfor å kringkaste disse
  - mengden av slik trafikk øker med økende deling av slike data, ikke med økende antall prosessorer i maskinen
- **Protokoll basert på skriv-ugyldiggjøring**
  - Tilstander
    - » "uncached"
    - » "shared"
    - » "dirty"
- **Protokollen er uavhengig av realisering av sammenkoplingsnettverket**
  - basert på å sende meldinger
  - "remote access cache" (Fig. 9.1) i en klynge holder rede på utestående forespørsler
  - logikken kan håndtere mange forespørsler samtidig
  - Fig. 9.26

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Kendall Square Research KSR-1

- **Om pensum, kap. 9.4.2.**
  - en del litt uklart/ufullstendig
  - delvis markedsførings-skryt
  - en del interessante/viktige prinsipper
  - "vurder selv hva som blir pensum!"
    - » forelesning og foiler indikerer godt
- **Første "kommersielle" COMA-multiprosessor**
  - COMA = Cache Only Memory Architecture
  - konkurs 1995/1996
- **Arkitektur**
  - Hierarki av ringer
    - » Ring-0: 8 mill. pakker/sek
    - » Ring-1: 8,16,32 el. 64 mill. pakker/sek
  - Enveis "slotted" ring, med 8-32 noder
    - » komm. mulig på hver "del-ring"
    - » dvs. samlet fysisk båndbredde skaleres lineært med # noder
      - men dette er vanskelig å utnytte
  - Maks 34 slike ringer
    - » maks 1088 noder
  - Fig. 9.27
  - Et vilkårlig antall nivåer er i prinsippet mulig

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

ALLCACHE; *innovasjon el. markedsførings-bløff?*

- **ALLCACHE**
  - Alt er lagret i cache, dvs. COMA
  - "*et helt nytt, revolusjonerede prinsipp! (?))*"
- **Node**
  - 64 bits superskalar spesial-prosessor
    - » 3.9 mill transistorer på 12 brikker
    - » både skalare og vektorielle beregninger
  - "primary cache" som brukes som et 32 Mbyte hovedlager.
  - "sub cache", 512 Kbyte
  - 75% av prosessor utgjør "søke-maskin"
    - » transport av data mellom noder
    - » opprettholde lagerkonsistens v.hj.a. distribuerte kataloger
    - » automatisk flytting og duplisering av data for økt ytelse
    - » data flyttes i "sub-pages" på 128byte
- **ALLCACHE** eliminerer lager-hierarkiet !?
  - Hva med "primary-cache" og "sub-cache" ?
  - ALLCACHE tilbyr ett-nivå-lager til programmereren
  - dette er nå ganske vanlig (jfr. SVM = Shared Virtual Memory)
  - Fig. 9.28

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Tera Multiprosessor (1/2)

- **Om pensum, kap. 9.4.3.**
  - mange interessante nye prinsipper
  - en del uklare/ufullstendige deler
- **Etterfølger av Denelcor HEP**
  - Smith, 1978
  - 16 prosessorer
    - » 10 MHz
    - » hver med 128 tråder (lettvekts-prosesser)
  - "feilet p.g.a. dårlig programvare/kompilatorer"
  - Tera, mye likt HEP, men
    - » ny teknologi, 400 MHz klokke, dvs. under 3ns klokkeperiode
- **Arkitektur**
  - delmengde av 3D rutenett
    - » Fig. 9.29

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Tera Multiprosessor (2/2)

- **Prosessorer**
  - Kan tidsmultiplekse opp til 128 tråder
  - Velger en "beregnings-klar" tråd ved hver klokke-sykel!
  - En del spesialiteter i instruksjons-settet
    - » F.eks. JUMP-OFTEN, JUMP-SELDOM
  - Spesialprosessor med stor grad av pipelining
    - » Fig. 9.30
    - » Tidsforsinkelse, (Eng. latency) er i snitt 70 klokke-sykler
    - » Trenger 70 tråder for å skjule dette
    - » 3 operasjoner pr. klokkesyklus er mulig
    - » 333MHz klokke gir 1Giga ops./sek.
    - » P.g.a. tidsmultipleksing vil hver tråd ikke oppleve mere enn ca. 100M ops./sek.
  - Håndtering av tråder
    - » Hastighet gjør at en må ha 128 kopier "av alt"
    - » Fig. 9.31
  - "Explicit-dependence lookahead"
    - » Hver instruksjon A inneholder et tre-bits felt som sier hvor mange etterfølgende instruksjoner som kan startes opp før vi møter en som *avhenger* av A.
    - » kan brukes for å starte opp flere instruksjoner fra en tråd på "full hastighet"

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Kornstørrelse i parallelle datamaskiner

- **Kap. 9.3.1 (m.m.)**
- *Kornstørrelse*
  - umoden terminologi
  - Eng. "grain size", "granularity"
  - [Hwa93] side 61: "Mål på mengden av beregninger involvert i en (SW) prosess"
  - [Fig. 2.5, side 62 (repetisjon)]
- **Gordon Bell**
  - Kilde: ULTRACOMPUTERS, A Teraflop Before Its Time, CACM, aug.92, side 27-47.
  - **Problem granularitet =**

    (# operasjoner på et koordinat-punkt)

    (nødv. datamengde innhentet fra nabo-punkter)
  - **Maskin granularitet =**

    (operasjonshastighet i node)

    (kommunikasjons-hastighet mellom nabo-noder)
- **Jo lavere maskin-granularitet jo mer finkornet kan vi parallellisere et problem**
  - Eks. CM5 : 102 operasjoner pr. ord overført, mens Paragon er bedre med 3 operasjoner pr. ord.

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Typiske parametre for "fin-, middels- og grovkornete maskiner"

"Klasse"	Finkornet	Middels-kornet	Grovkornet
Komm.-forsinkelse	Typisk < 1 millisek.	Typisk noen få millisekund.	Stor variasjon
Synkron-iseringstid	Mindre enn eller noen få mikrosek.	< 1 millisek.	Stor variasjon
Korn-størrelse	< 500 instruksjoner. Typisk 20	Typisk ca. 2000 instruksjoner	Jobb-trinn, el. del-program. Tusenvis av instruksjoner
Parallel-litetsgrad	1K - 64K	8-128	2-16
Kommentar	F.eks. MIT J-machine	F.eks. Intel iPSC/1	F.eks. CRAY

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Dataflytmaskiner og hybride arkitekturer

- **Dataflytmaskiner**
  - Prinsipper
  - Historisk utvikling
  - Statiske vs. dynamiske
  - **Eks. Manchester Dataflow Machine (MDM)**
  - Eks. ETL/EM-4 fra Japan
- **Krever nye maskinvareløsninger**
- **Gir muligheter for maskiner med stor grad av parallellitet i mange ledd**
  - egnet for VLSI !
- **Typisk arkitektur**
  - En ring med ulike enheter der pakker sirkulerer
    - » pipelining
- **Hybride arkitekturer**
  - Kombinasjon mellom dataflyt maskin og von Neumann-modellen
  - Eks. MIT/Motorola *T ***(Kap. 9.5.3 utgår)***

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Dataflytmaskiner: prinsipper

- **Pensum:**
  - Boka kap. 2.3 & 9.5
  - Artikkel om Manchester Dataflow Machine
  - Foiler fra forelesning
- **Dataflyt-graf** (Fig. 9.32a)
  - Node = operasjon
  - Pil = datapakke
  - En operasjon startes når alle inndata (operander) er tilgjengelige
    - » dvs. iboende, naturlig parallellitet
    - » data-drevet
    - » ingen program-teller
  - En node tilsv. en instruksjon
  - Alle "klare" operasjoner kan utføres samtidig
    - » hvis en har nok funksjonelle enheter
  - Operander kan ankomme "når som helst", og resultater (nye operander) kan produseres "når som helst"
    - » asynkron oppførsel
    - » buffring, pipelining er mulig

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Kontrollflyt og dataflyt

- **(Tradisjonell) kontroll-flyt**
  - Eksplisitt kontrollflyt (manipulering av programteller)
  - Data utveksles mellom instruksjoner via delte lagerlokasjoner
  - Data refereres via lager-adresse
  - En enkel kontroll-"tråd"
  - Flere parallelle kontroll-tråder
    - » *eksplisitt parallellitet*
- **Dataflyt maskiner**
  - Data-drevne beregninger, dvs. valg av instruksjoner for utførelse styres av data (operand)-tilgjengelighet
    - » *implisitt parallellitet*
  - Programmer er representert som rettede grafer som viser flyt av data mellom instruksjoner (noder)
  - Resultater sendes direkte som data-pakker mellom instruksjoner
  - Har ikke noe felles lager som flere instruksjoner referer til
    - » dvs. ikke side-effekter

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Dataflytmaskiner: historisk utvikling m.m.

- **Relativt gammelt tema**
- **Mye forskning**
  - Fundamentalt forskjellig =>> spennende
- **Sammenheng med funksjonelle språk, fornyet interesse?**
- **En del prototyper er bygget, men ingen med "revolusjonerende" ytelse**
  - Først i den senere tid har man klart å bygge maskiner med brukbar ytelse.
- **Fig. 9.32.b**

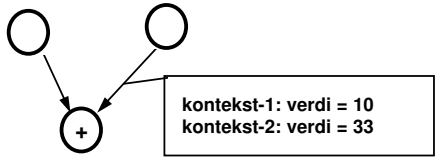
Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Statiske og dynamiske dataflytsystemer

- **Statiske systemer:**
  - tillater ikke samtidig reaktivering av kode
    - Dvs. en del av en dataflytgraf kan bare finnes i en "instans" av gangen
    - Maks. en datapakke kan eksistere på en linje i dataflyt-grafen av gangen
    - Datapakker kommuniseres direkte fra instruksjon til instruksjon
      - Kontroll-pakker benyttes som kvittering fra mottaker til avsender slik at avsender vet når den kan produsere en ny resultat-pakke
- **Dynamiske systemer:**
  - Tillater samtidig reaktivering dvs. samme kode kan utføres samtidig i ulike sammenhenger (Eng. context)
  - *Hva kan vi da ha ????* .....
    - "utbretting" av løkker (iter. nr.)
    - samtidige prosedyrekall
    - rekursive kall

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Dynamiske dataflyt-systemer realisering



Hvordan kan dette realiseres?

1) merke datapakke med kontekst-identifikator
- **antas heretter**
- **"tagged token" = merkete datapakker**

2) kopiering av kode

- **Må kunne ha flere enn en data-verdi på vei samtidig mellom to instruksjoner**
  - Har da ikke nok lagerplass i mottaker-instruksjonen
  - Egen enhet i maskinen der operand-pakker venter på "riktig makker"
  - Gir muliget for (logisk) buffring på hver linje

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Eksplisitt pakke-lagring (Eng. Explicit Token Store)

- **"Token-matching" er kostbart eller blir fort en flaskehals**
  - Parallell (assosiativ) søking (og evt. venting på) operand-"makker" til samme instruksjon med samme merkelapp-verdi
- **Alternativ:**
  - Lager for ventende token er eksplisitt adressert
  - Lagerlokasjonene har "full/empty"-bit

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

ETL/EM-4

- **Japansk dataflytmaskin**
  - Etterfølger av Sigma-1
  - Konstruert for 1024 prosessorer
  - Prototype med 80 prosessorer i 1990
  - Etterfølger:
    - » EM-4+ (med flyt-tall)
    - » EM-5 (?)
    - » EM-X
- **Arkitektur**
  - Omega nettverk for sammenkopling av noder
  - Distribuert programlager for eksplisitt pakke lagring
  - Fig. 9.33
- **Programvare**
  - C-kompilator med eksplisitt fork/join
    - » dvs. eksplisitt parallell kontrol-flyt

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Mere om dataflyt-maskiner

- **Temaer for forskning**
  - Språk og kompilatorer
    - » Id, VAL, LUCID, SISAL
  - Dynamisk lastfordeling, justering
    - » Justering av "Multiprosesseringsgrad"
      - jfr. "throttle unit" i MDM
      - mer generelt; OS el.l. kan velge ut et passende antall instruksjoner som er "klare" for utførelse
  - Hurtig kommunikasjon og ruting av pakker
  - Datastrukturer
    - » f.eks.effektiv håndtering av tabeller
    - » "I-structure"
  - Realisering av lager for instruksjoner

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Dataflytmaskiner, diskusjon

- **Framdeles stor uenighet om hvordan de bør bygges**
- **Fordeler**
  - Stor grad av parallellitet er mulig
  - Enkel oppgradering med flere prosessorer
  - Homogenitet og modularitet i maskinvare, dvs. velegnet for VLSI
  - Egnet for funksjonelle språk som vil kunne øke programmereres produktivitet
- **Ulemper**
  - en del av disse er løst/redusert nå:
    - » Programmerere må kanskje lære seg nye programmerings-språk
    - » Eksisterende programvare må kanskje forkastes
    - » Svært finkornet parallellitet (instruksjons-nivå)
      - gir stor overhead
    - » "single-assignment"-regelen o.a. fører til stort plass-forbruk
    - » lite effektivt ved håndtering av store tabeller og datastrukturer

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

Manchester Dataflow Machine (MDM)

- **Kilde: *The Manchester Prototype Dataflow Computer,* Gurd, Kirkham, Watson, CACM, jan 85, pp. 34-52**
- **Dataflytmaskin basert på dynamisk merking (Eng. tagging) av (små) datapakker (Eng. token)**
- **Ca. 1-2 MIPS i 1985**
- **Demontert i 1989**
- **C. Kirkham, januar 1993: *"I still believe in Dataflow"***

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

MDM: Dataflyt-programmer

- **Tre nivåer**
  - SISAL (Fig. 2)
  - Assembler (Fig. 3)
    - » variable fra SISAL
    - » operatorer fra dataflyt instruksjons-sett.
  - Maskin kode (Fig. 1)
- **Grafisk representasjon**
- **Virkemåte (Fig. 4)**

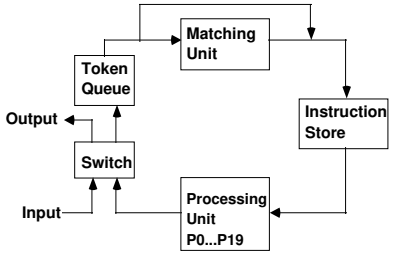
Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

SISAL

- **Streams and Iterations in a Single-Assignment Language**
- **Samarbeid mellom bl.a. DEC, Livermore Labs. (LLNL), Univ. i Manchester m.m.**
- **Finnes også på CRAY, VAX, Transputer m.fl.**
- **Det viktigste dataflytspråket**
- **PASCAL likt "single-assignment" språk**
  - Bare en tilordning til hver hver variabel i et program.
    - » lettere for kompilator å utnytte parallellitet i programmet
- **Eks. Fig. 2**
- **Videre lesing: Asgeir Langen´s diplom**

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

MDM: System struktur



- **Merkede datapakker = token**
- **Merkelapp (Eng. tag) =**
  - **iterasjonsnivå** (for løkker)
  - **aktiveringsnavn** (for samtidige prosedyrekall og rekursjon)
  - **indeks** (når samme kode behandler ulike deler av en datastruktur)
- **Tekniske løsninger**
  - Fig. 7 og 8

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

MDM: System evaluering

- **Testmetode**
  - Laster program, laster input-token inn i input-kø, starter klokke og slipper løs token fra køen. Stopper klokke ved første token mottatt hos vertsmaskin
- **Tidsforbruk er en funksjon av**
  - antall prosessorer / program / input
- **LÆRE OM :**
  - maskinvare utnyttelse og flaskehalser
  - parallellitet i programvaren
  - dataflyt-MIPS vs. "vanlig MIPS"
- **Reduser antall "variable"**
  - ulike kunstige situasjoner for å unngå å teste for mye av gangen
    - » Kunstige programmer
      - » F.eks. program med Pby = 1.00
    - » Simulator
- **Test-klasser**
  1) små programmer -> bruker ikke overflyts-enhet
  2a) moderat grad av overflyt
  2b) ekstrem grad av overflyt
- **Simulator av maskinen**
- **AvePara = T(1)/T(inf)**

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

"Bailey´s twelve ways to fool the masses..."

- **"Twelve ways to fool the masses when giving performance results on parallel computers"**
  1) Beskriv 32 bits ytelse som 64 bits ytelse
  2) Presenter indre-løkke ytelse som applikasjons-ytelse, neglisjer I/O
  3) Benytt assembler, mikro-kode og lavnivå-kode
  4) Skaler opp problemstørrelsen for å unngå synkende ytelse ved bruk av et stort antall prosessorer med overhead og kommunikasjons forsinkelser eller begrensninger
  5) Ekstrapoler ytelsesresultater basert på en (eller få) prosessorer til et system med full størrelse
  6) Sammenlikn med ikke-optimalisert, skalar CRAY kode.
  7) Sammenlikn direkte kjøretid med gammel kode på et foreldet system (f.eks. CRAY 1)
  8) Beskriv ekstra operasjoner som kreves når en bruker en parallell og gammeldags algoritme
  9) Beskriv prosessor utnyttelse, hastighetsøkning, eller Mflops/$ og ignorer ytelse
  10) Forvreng algoritmen slik at den passer arkitekturen men gir meningsløse resultater
  11) Sammenlikn resultater fra et belastet system med resultater fra et dedikert system
  12) Hvis alt annet mislykkes, vis bilder og videoer

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

MDM: Problemer

- **Lite effektiv håndtering av datastrukturer**
  - » spesial-HW
- **Arbitrering til/fra funk. enheter**
  - kan bli flaskehals
- **Eksperimenter:**
  - Prosessorene "sulter" når en stor andel av token ikke gir "match"
    - » større buffer på utgang av matching unit
  - Trenger bedre kompilatorer

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

MDM: Framover

- **SW og HW forbedringer**
- **SISAL kompilator**
  - Påvirke instruksjons-typer i de funksjonelle enheter (PE´er). Dvs. større "super-instruksjoner"
    - » (([Natvig ca. 88:] "*objektflytmaskiner*"))
    - » **Men**, hybrider med enklere operasjoner, RISC-inspirert [Hwang93]
- **For større programmer blir lagring av datastrukturer i Matching Unit en flaskehals**
  - Hierarki
    - » Side-inndeling, krever lokalitet i dataflyt-programmer
  - SSU (Struct Store Unit)
    - » Ekstra enhet som koples inn mellom de funksjonelle enhetene og IO-svitsj
- **"Intelligent" Token Queue**
  - Prioriterer blant buffrede token de som vil gi en bedre lastfordeling i systemet
- **Multiprosessor**
  - Hierarki av ringer
  - Distrib. matching
  - Last fordeling

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU

MDM: Et blikk bakover (1992)

- **Kilde; *Manchester Data-Flow: A Progress Report,* Gurd & Snelling, ICS'92.**
- **Historie**
  - Startet i 1976
  - MDM demontert i 1989
- **Inneholdt da bl.a.**
  - Egne Structure Store Units
  - "Throttle Unit"
    - » "Regulerings-spjeld, kveler"
    - » Store programmer kan generere for stor parallell aktivitet som drukner systemet i token som ikke kan prosesseres på lang tid
      - en slags "trashing"
    - » En enhet som koples inn i ringen og som tilordner unike "aktiverings-navn" (Dette er en del av "tag"-feltet, og brukes for å skille ulike instanser av samme kode)
      - ut i fra informasjon om belastning i systemet kan denne enheten "holde igjen" tildelingen av slike navn til belastningen synker.
- **Erfaringer**
  - mikrokodet instruksjonssett var et klokt valg
    - » forskning -> eksperimentering

Copyright © 1995 Lasse Natvig og Pauline Haddow, NTNU